View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by University of Waterloo's Institutional Repository

Improved Particle Filter Based Localization and Mapping Techniques

by

Adam Milstein

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2008

©Adam Milstein 2008

AUTHOR'S DECLARATION

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

One of the most fundamental problems in mobile robotics is localization. The solution to most problems requires that the robot first determine its location in the environment. Even if the absolute position is not necessary, the robot must know where it is in relation to other objects. Virtually all activities require this preliminary knowledge. Another part of the localization problem is mapping, the robot's position depends on its representation of the environment. An object's position cannot be known in isolation, but must be determined in relation to the other objects. A map gives the robot's understanding of the world around it, allowing localization to provide a position within that representation. The quality of localization thus depends directly on the quality of mapping. When a robot is moving in an unknown environment these problems must be solved simultaneously in a problem called SLAM (Simultaneous Localization and Mapping). Some of the best current techniques for localization and SLAM are based on particle filters which approximate the belief state. Monte Carlo Localization (MCL) is a solution to basic localization, while FastSLAM is used to solve the SLAM problem. Although these techniques are powerful, certain assumptions reduce their effectiveness. In particular, both techniques assume an underlying static environment, as well as certain basic sensor models. Also, MCL applies to the case where the map is entirely known while FastSLAM solves an entirely unknown map. In the case of partial knowledge, MCL cannot succeed while FastSLAM must discard the additional information. My research provides improvements to particle based localization and mapping which overcome some of the problems with these techniques, without reducing the original capabilities of the algorithms. I also extend their application to additional situations and make them more robust to several types of error. The improved solutions allow more accurate localization to be performed, so that robots can be used in additional situations.

Acknowledgements

This thesis would not have been possible without the help of many people. I would especially like to thank my supervisor Dale Schuurmans for all his help and advice and my family for their continuous support. Finally, for all those who have aided me during the course of my work whom I have not mentioned specifically, (especially Beaufort and Beauregard, for helping me to keep things in perspective) thank you.

Table of Contents

AUTHOR'S DECLARATION ii
Abstract iii
Acknowledgements iv
Table of Contents v
List of Figures viii
List of Tables ix
Chapter 1 Introduction 1
1.1 Thesis Contributions 2
1.1.1 Cluster MCL 2
1.1.2 Dynamic Motion Models in MCL 3
1.1.3 Dynamic Maps in MCL 3
1.1.4 Skeletal FastSLAM 3
Chapter 2 Background 5
2.1 Occupancy Grid Mapping 5
2.1.1 Artificial Beacons 5
2.1.2 Occupancy Grid Maps 6
2.1.3 Mapping Technique 6
2.2 MCL 7
2.2.1 Recursive Bayes Filter 9
2.2.2 Particle Approximation 10
2.2.3 Resampling 11
2.2.4 Bias 11
2.2.5 Algorithm 13
2.2.6 Sensor Model 14
2.2.7 Motion Model 16
2.2.8 Importance Factor 18
2.2.9 Raytracing 19
2.2.10 Additional Techniques 19
2.3 FastSLAM 24
2.3.1 SLAM Problem 24
2.3.2 FastSLAM Derivation 25

2.3.3 Feature Based FastSLAM 26
2.3.4 Occupancy Grid FastSLAM 28
Chapter 3 Cluster MCL 31
3.1 Introduction 31
3.2 Cluster-MCL 32
3.2.1 Clustered Particle Filtering 32
3.2.2 Algorithm 33
3.2.3 Example 35
3.3 Experimental Evaluation 35
3.3.1 Simulated Data 36
3.3.2 Real Data 37
3.4 Conclusion 39
Chapter 4 Dynamic Motion Models in MCL 41
4.1 Introduction 41
4.2 Dynamic Motion 42
4.2.1 Motion Model Error 43
4.2.2 Variance Parameters 43
4.2.3 Expanded Motion Model 44
4.2.4 Specialized Motion Model 46
4.2.5 Updated MCL 48
4.2.6 Algorithm 48
4.2.7 Nonlinear Optimization 50
4.3 Experimental Evaluation 52
4.3.1 Static Motion Models 53
4.3.2 Standard Motion Model 55
4.3.3 Expanded Motion Model 57
4.3.4 Comparison of Models 58
4.3.5 Random Restart Optimization 60
4.3.6 Hardware Specialized Model 61
4.4 Conclusion 63
Chapter 5 Dynamic Maps in MCL 65
5.1 Introduction 65

5.2 Dynamic Maps 66
5.2.1 Factoring 67
5.2.2 Binary Object Bayes Filtering 68
5.2.3 Cell Correlations 69
5.3 Algorithm 70
5.4 Experimental Evaluation 72
5.4.1 FastSLAM Comparison 75
5.5 Conclusion 75
Chapter 6 Skeletal FastSLAM 77
6.1 Introduction 77
6.2 Skeletal FastSLAM 77
6.2.1 Creating the skeleton map 78
6.2.2 Monte Carlo Localization with Paths 78
6.2.3 Derivation of FastSLAM with Skeleton 78
6.2.4 Defining the skeleton model 79
6.2.5 Algorithm 80
6.2.6 Skeleton map parameters 81
6.3 Experimental Evaluation 81
6.3.1 Normal FastSLAM Implementation 82
6.3.2 Simulated data 82
6.3.3 Real data 83
6.4 Conclusion 85
Chapter 7 Conclusion 87
7.1 Future Work 88
Bibliography 91
Appendix A Standard Mathematical Definitions 95

List of Figures

Figure 2-1: Graphical model of localization problem as a POMDP. 8
Figure 2-2: Graphical representation of the behaviour of bias for a simple case................................ 13
Figure 2-3: Implementation of a standard laser rangefinder sensor model... 14
Figure 2-4: Probability values for a specific expected distance from Figure 2-3 16
Figure 2-5: Standard types of motion error... 16
Figure 2-6: Occupancy Grid Map Raytracing.. 18
Figure 3-1: Global localization using ordinary MCL 36
Figure 3-2: Global localization using Cluster-MCL. 36
Figure 3-3: Results of MCL and Cluster-MCL on Wean Hall dataset 3... 38
Figure 3-4: Results of MCL and Cluster-MCL on Gates data. 39
Figure 4-1: The geometry of the specialized motion model. 46
Figure 4-2: First environment used for experiments.. 52
Figure 4-3: Graph of distance travelled vs. range error for standard technique (Experiment A). 53
Figure 4-4: Graph of distance traveled vs. range error for optimal static technique (Experiment B).. 54
Figure 4-5: Graph of distance traveled vs. range error for global dynamic technique (Experiment C). 55
Figure 4-6: Graph of distance traveled vs. range error for regional dynamic technique (Experiment D). 56
Figure 4-7: Graph of distance traveled vs. range error for regional dynamic technique with expanded motion model (Experiment F).. 57
Figure 4-8: Surface of the error for the range parameters... 59
Figure 4-9: The global minimum in the error surface for the range parameters................................. 60
Figure 5-1: Before and after 2 passes through the environment 73
Figure 5-2: Before and after 5 passes through the environment using a schematic map.................... 74
Figure 6-1: Skeleton map probability model. 80
Figure 6-2: Error in position over time for normal FastSLAM vs. skeletal in a simple loop. 83
Figure 6-3: Two real environments with skeleton maps 84

List of Tables

Table 2-1: MCL Algorithm 13
Table 2-2: MCL with KLD Sampling 21
Table 2-3: Feature Based FastSLAM Algorithm 26
Table 2-4: Occupancy Grid FastSLAM algorithm 29
Table 4-1 Experimental results of all motion model algorithms. 62
Table 5-1: Dynamic MCL algorithm 71
Table 6-1: Experimental comparison of skeletal FastSLAM algorithm vs. original. 85
Table A-1: Standard Mathematical Definitions 95

Chapter 1
Introduction

In order for a mobile robot to accomplish virtually any useful task in an environment, it must know its own location relative to other locations in that environment. Especially when a task requires performing actions at specific locations, it is vital that the robot be able to travel reliably between those positions. The basic knowledge required for travel is the knowledge of position. You must know where you are in order to determine how to get to a destination, or even to determine if you are already at a destination.

Localization is the name given to the problem of determining a mobile robot's position according to a map of its environment. Solving the localization problem is a prerequisite for solving most other mobile robot problems. Robots are equipped with various sensors that provide certain information about the environment, and localization requires that these readings be somehow converted into an estimate of the robot's pose in the environment, according to a map. The problem, however, is that robots do not commonly have sensors that are able to provide a direct estimate of pose. Typical sensors might be cameras, range sensors and odometers and somehow these sensors must be used to provide the location estimate.

The objective of localization is to use the known information of the sensors, controls and map to determine the unknown location. Since localization is usually an ongoing, dynamic process, it usually also uses the knowledge of the robot's position prior to the current timestep. However, a companion problem to localization is global localization [Weiss et al. 1994; Borenstein et al. 1996], where the mobile robot's prior position is unknown. This is a much harder problem which also involves the same features as regular localization. A further increase in complexity occurs if the map is unknown. In order to properly estimate its position in this situation, a robot must simultaneously determine both the map and its position in that map. These tasks, which humans perform so easily, are fundamental to most meaningful robot problems.

Because it is such a fundamental problem, there have been many proposed solutions to localization. These techniques use a variety of methods from heuristic algorithms to complex mathematical derivations and have different sets of benefits and drawbacks. Some algorithms are more efficient while some have greater theoretical foundations and proofs of convergence. Also, many solutions assume prior knowledge of the map while others simultaneously generate the map and the robot's location. Other methods rely on offline processing after the robot has stopped to determine where it has been. However, one common feature is the assumption that the environment is unchanging.

There has always been a gulf between localization with a known map and localization that simultaneously generates the map and finds the robot's position. Localization is the name given to the problem of finding the robot's position in an existing map while SLAM (Simultaneous Localization and Mapping) involves determining the robot's location and also generating the map without initial knowledge of the configuration of the environment. However, there is currently no solution to problems between these two extremes. Localization assumes a completely known and static map while SLAM assumes an initially completely unknown one, which is also static. Although localization algorithms are designed to handle some inaccuracies in the map there is no provision for

correcting these errors. Similarly, SLAM solutions are designed to start with complete uncertainty and have no method for taking advantage of some minimal initial information, unless that information is encoded in the same form as the map generated by SLAM. Both kinds of solutions also have the additional problem that they cannot handle environments which change over time, except by being robust to gradually increasing errors in the map.

My research involves making improvements to certain powerful localization techniques, primarily Monte Carlo Localization (MCL) [Thrun 2000], so that it can gradually adapt to the environment. I have developed various techniques to reduce the dependence on the static map assumption. Without requiring a more detailed map originally, my methods allow a robot to improve its representation of the environment based on its own sensor observations. Similarly, I have worked on allowing some initial information to be incorporated into a SLAM solution, providing a more successful technique which makes use of some additional information. Although there are undoubtedly situations where localization with a static map is sufficient and other circumstances where no initial information is available to aid SLAM, many environments have features somewhere between these two extremes. Most real areas experience some changes in structure over time. Similarly, a region where a robot is to be deployed, even if it has not been accurately mapped beforehand, often has some known features. Allowing a robot not just to ignore these factors, but to actually make use of them, can provide a significant benefit in many common situations. My experiments have been performed using a two wheeled, differential drive near-holonomic robot operating in an indoor environment. Since the robot can only travel on flat ground I use a two dimensional representation of the environment.

1.1 Thesis Contributions

1.1.1 Cluster MCL

MCL often has problems localizing from complete uncertainty in environments with a significant amount of symmetry. When sensor readings correspond to multiple locations the localization algorithm eventually converges to an area around a single one of them, often at random. I have created a technique to use a higher level organization on the particle filter of MCL which allows it to consider the possibility of multiple locations for an indefinite number of timesteps. Even once MCL chooses one of these locations as the actual robot's position it still considers the less likely choices. Eventually, if more information is acquired to resolve the symmetry, the proper location can be identified. The cluster MCL algorithm works by forcing a certain number of hypotheses to exist independently, regardless of each one's overall probability.

Cluster MCL also provides a solution to the kidnapped robot problem. This is a problem in robotics where a robot that was properly localized is suddenly moved to a different location. The problem occurs because handling a kidnapping requires global localization, but this automatically invalidates the current localization result. If the robot has not been kidnapped, by far the most likely situation, this causes a localization failure while the robot is relocated. Cluster MCL allows the kidnapping detection to be handled without affecting the current localization, unless a new location is detected that is more probable. In that case, the new location is reported as the robot's position while the old location is still maintained in case no kidnapping actually occurred. By adding a second level

organization to MCL, cluster MCL is able to handle certain situations that ordinary MCL has trouble with.

1.1.2 Dynamic Motion Models in MCL

The success of MCL depends on its two models, the sensor model which gives the probability of the actual sensor readings and the motion model, which produces probable locations for the robot based on its starting position and reported motion. The motion model depends on certain parameters which must usually be determined using exhaustive testing and experimentation. Because these parameters are so difficult to discover, general values are often used which do not represent any specific situation particularly well. It is often easier to use these general parameters rather than perform the experimentation necessary to determine optimal values for a particular case. Also, simple motion models are used which depend on very few parameters that can be easily estimated. However, my research has determined that the optimal parameters for the motion model can be determined automatically without requiring anything more than these basic parameters. Thus, over time, the motion model improves to more accurate parameters. Since the parameters are found automatically there is no additional experimentation required, but there is a significant improvement in the results. Also, because the parameters gradually improve, it is possible to use much more complex motion models without suffering the penalty of increased preliminary work by a skilled user. The final benefit is that, by optimizing the parameters for different regions in the environment, the single motion model can adapt to changing local conditions, allowing a more accurate proposal for the robot's location. Using dynamic motion models allows MCL to improve its effectiveness over time, adapting to changing or local conditions and leaving greater tolerance for unanticipated sources of error.

1.1.3 Dynamic Maps in MCL

One of the primary limitations of MCL is the static map assumption. The algorithm assumes that the features of the environment do not change. Although certain implementations can handle moving objects, such as people, the underlying map is unchanging. However, this does not correspond to most real environments. An area that is actually being used will have many dynamic features. Doors will open and close, furniture will move, and other objects can be added or removed. Over time, the map becomes less accurate. In Chapter 5, I describe a system whereby the map can be adapted automatically to the changes in the environment. The result is that, over time, the robot's map becomes more accurate, rather than losing accuracy. With a more accurate map, MCL is more robust to other sources of error. The dynamic map algorithm allows MCL to correct its own representation of the environment instead of requiring the map to be periodically rescanned by a skilled user. Thus, a robot can be deployed for a much longer time without needing new data.

1.1.4 Skeletal FastSLAM

If the map of the environment is unknown, localization requires the generation of a map in a problem called simultaneous localization and mapping (SLAM). One of the most powerful solutions to SLAM is FastSLAM, which is based on particle filters. However, FastSLAM, and indeed all current

SLAM solutions, generate the map and localize with no prior information at all. But, in many cases some information about the environment is available, even if it is not enough for MCL to work. In Chapter 6, I describe a system to make use of very basic information about an environment to improve the behaviour of FastSLAM. By using a skeleton map of the major corridors in the environment the algorithm can converge to the correct map much more easily. A skeleton map can be easily generated if the topology of the environment is known, as is the case with most indoor environments, but the information is very helpful to SLAM. Although requiring more initial information seems like a reduction in the power of FastSLAM, in many common situations a skeleton map is readily available and so there is no drawback to making use of it. The ability to use some additional information if it is available adds to the versatility of the algorithm.

All of my research described in Chapter 3 through Chapter 6 extends the power of particle filter based localization and mapping algorithms to better handle more situations. Particle filtering techniques are very effective in robotics, but, because of the assumptions made by the current techniques of MCL and FastSLAM, there are some environments that are not well handled. I have created new techniques to augment these algorithms to improve localization and mapping in various situations, while not affecting their original properties. In most cases my techniques can be used without severely affecting the runtime of the underlying particle filter method. Thus, my research improves the adaptability of MCL and FastSLAM by providing better behaviour in many common environments, without compromising the desirable properties that these solutions provide.

Chapter 2
Background

My research primarily involves ways of improving localization and mapping for indoor environments. Currently, some of the best techniques for solving these problems are particle filter based approaches. For localization, Monte Carlo Localization (MCL), is one of the most effective solutions. MCL can also be used for the more complex problem of global localization. In order to simultaneously generate a map, the particle filtering approach is called FastSLAM. Both of these techniques can be used with occupancy grid maps, which are a common method for representing indoor environments. My research has improved on these basic techniques to provide improved solutions in many situations.

2.1 Occupancy Grid Mapping

In order to perform localization and mapping it is necessary to define some representation of the environment. Many probabilistic techniques for localization depend on the map being defined as a finite sized set of landmarks which the robot's sensors observe, giving their relative displacement from the robot. However, physical sensors do not usually detect landmarks unambiguously. Instead, they report the distance to the nearest obstacle, or return an image of the environment. In order to use a landmark based algorithm, the sensor readings must be pre-processed in a separate step to convert the raw sensor data into a set of detected landmarks, such as in [Leonard and Durrant-Whyte 1991]. The additional step introduces more error into any algorithm, as well as discarding much of the sensor information which does not detect any landmark.

One of the primary drawbacks of landmark based maps is the data association problem. Because raw sensor data is not labelled with the correct landmark detected, the sensor processing must somehow determine exactly which landmark was observed. If mistakes are made, the localization and mapping algorithms which depend on the sensor data will fail. In order to compensate for the data association problem, many localization and SLAM algorithms include a method for determining the associations between the sensor data and the landmarks. However, these techniques add significantly to the implementation complexity of the solutions. Also, they do not solve the problem of actually finding landmarks in the raw sensor readings. Two examples of these algorithms are GraphSLAM [Folkesson and Christensen 2004] and Sparse Extended Information Filters (SEIF) [Thrun et al. 2004], both of which can be implemented to handle data associations in a probabilistic way as described in [Thrun et al. 2005] (Sections 11.5, 12.8, 12.9). Even with these integrated solutions, the data association problem requires additional processing and adds another source of error, even though the algorithms are effective in some problems.

2.1.1 Artificial Beacons

One effective technique to reduce the data association problem is to place artificial beacons which can easily be detected by the sensors throughout the environment. For example, brightly coloured posts might be placed in an area and detected using vision sensors. By choosing unusual colour patterns, it

can become trivial to uniquely identify the beacons. Other robots have used special bar codes or reflectors. The sensor data provided to localization in these cases is not the raw information, but instead a list of the angles and directions to the specific beacons. Although this technique can be very effective, it relies on being able to extensively modify and control the robot's operating environment, which may not be possible.

2.1.2 Occupancy Grid Maps

One common technique for map representation that does not suffer from data associations, is to use occupancy grid maps to approximate the environment. An occupancy grid map represents the environment as a block of cells, each one either occupied, so that the robot senses an object at that location, or unoccupied, so that the robot senses no object. Unless your environment is composed entirely of cubes, occupancy grid maps cannot be absolutely accurate, but by choosing a small enough cell size they can provide all the necessary data for the robot to successfully perform its task. Any sensor will report the status of a set of grid cells that can be checked without reference to the rest of the map. An early implementation of occupancy grid maps was used in [Moravec 1988] to automatically generate a map of the environment. Sensor readings are compared to the map, altering the probability that observed cells are occupied. For example, a sonar sensor returns the closest object within a cone, so the cells in the volume of the cone closer than the reading are probably unoccupied. Moravec represents each cell as a probability of being unoccupied and initializes them to an unknown value. He describes a probabilistic technique to update cells for various types of sensors and gives a technique to allow the map to be updated as the robot moves. Unfortunately, this technique is not actually localization and does not help the robot know its own position. The map is maintained relative to the robot, rather than in a global frame of reference. In other words, the robot is assumed to be at a fixed location, while the map moves around it. As the robot moves, the map is blurred according to the motion. The robot's sensors can correct the map in its immediate area, but unobserved portions of the map must blur into uselessness. There is also no way to discover the robot's location in reference to previously visited locations.

Although the technique is problematic as a localization algorithm, it provides a very powerful way to represent the environment. Using an occupancy grid map allows the raw sensor data to be used without trying to detect and identify landmarks. Also, since raw data is used, no information is discarded because it does not correspond to a landmark. The only problem is that there are a huge number of map features, one for each grid cell. Algorithms which consider the relationship of the robot to a set of distinct landmarks cannot be applied when the number of features is so large. Thus, using occupancy grid maps limits the type of localization techniques that can be used.

2.1.3 Mapping Technique

To create an occupancy grid map, it is necessary to determine the occupancy probability of each cell. In order to do this efficiently, the assumption is often made that map cells are independent and that sensor readings do not introduce any dependencies. Although this is not strictly accurate, especially when considering adjacent cells representing the same physical object, it greatly simplifies the mapping algorithm without introducing any insoluble problems. As a result, the probability of a

particular map, represented by the variable m, can be factored into the product of the individual probabilities of its cells conditioned by the variable x_t representing the robot's state at time t and the variable z_t representing the sensor data at time t. The notation a^t means the entire history of the variable a, $a^t = \{a_0, ..., a_t\}$.

$$p(m \mid x^t, z^t) = \prod_n p(m_n \mid x^t, z^t) \qquad \textbf{2-1}$$

The probability of a particular cell is easy to determine given the robot's position and sensor readings, since it is determined by whether the robot observes the cell as unoccupied or occupied. Since the probability is determined by the robot's entire history all these sensor readings must be taken into account. The mapping algorithm usually builds the cell probabilities up iteratively, considering each pair $\{x_t, z_t\}$ from time $t = 0$ to the most recent reading. Although these readings could be considered in any order, the iterative processing makes the most sense, allowing additional readings to be added and leading eventually to simultaneous mapping and localization (SLAM) solutions such as described in section 2.3.

With occupancy grid maps, the mapping step must determine the probability of each cell, as represented by equation (**2-1**). Proceeding iteratively, the map cells are updated according to the position and sensor readings. Of course, it would require significant processing to update the entire map on each step, but this is unnecessary. Only the cells which are actually observed need to be updated. Each cell that is perceived by the sensor, given the robot's position, is updated depending on whether the sensor indicates it is occupied or unoccupied. Although the map is defined by the occupancy probability, for simplicity the actual values for each cell, given by $p(m_n \mid x^t, z^t)$, are calculated in log odds form.

$$l_{t,n} = \log \frac{p(m_n \mid x^t, z^t)}{1 - p(m_n \mid x^t, z^t)} \qquad p(m_n \mid x^t, z^t) = 1 - \frac{1}{1 + e^{l_{t,n}}} \qquad \textbf{2-2}$$

$$l_{t,n} = l_{t-1,n} + \log \frac{p(m_n \mid x_t, z_t)}{1 - p(m_n \mid x_t, z_t)} - \log \frac{p(m_n)}{1 - p(m_n)} \qquad \textbf{2-3}$$

$p(m_n)$ is a constant prior occupancy probability which is included in the final term of equation (**2-3**) to convert the observations into changes from the prior. The only part of equation (**2-3**) which is not already known is $p(m_n \mid x_t, z_t)$, which is called the inverse sensor model. Although a highly accurate inverse sensor model is difficult to determine, a simplified implementation that returns a high value if the sensors report an object in the cell, and a low value otherwise, is often acceptable. Occupancy grid mapping updates a map according to a sensor reading at a location so that, as evidence accumulates, the map becomes correct. Of course, the success of the mapping algorithm depends on the location x_t being correct, just as the success of localization depends on an accurate map.

2.2 MCL

Monte Carlo Localization uses models of various sensors, together with a recursive Bayes filter, to generate the belief state of a robot. In fact, MCL is a specific instance of a POMDP (Partially

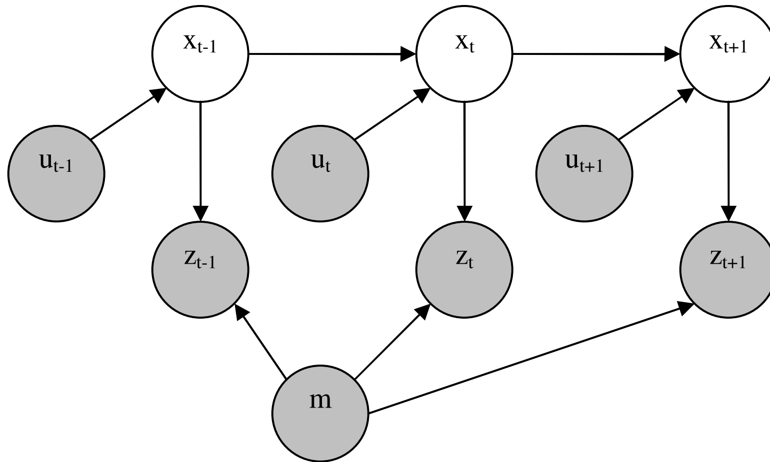


Figure 2-1: Graphical model of localization problem as a POMDP.

Unshaded x nodes represent unknown robot pose, shaded nodes represent known values. u are the controls given to the robot and z are the sensor readings received. All readings are taken relative to time t, except the map, m, which is considered to be static.

Observable Markov Decision Process). A standard form of MCL uses a motion model to predict the robot's motion, together with a sensor model to evaluate the probability of a sensor reading in a particular location. The sensor model necessarily includes a static map of the environment. The algorithm can be applied to virtually any robot with any sensor system, as long as these two models can be created. One common implementation where MCL is very successful is on a wheeled robot using a range sensor ,such as a laser rangefinder. A benefit of this combination is that the map and location used by the algorithm are in a human readable format. Although I give the general algorithm in the following sections, which should be applicable to other robots, where application specific details are required, I assume the type of robot as described.

Other localization algorithms than MCL exist, but they are currently more limited than MCL in terms of the environments in which they are effective, requiring specific environmental features in order to be effective. Most other localization algorithms require that the map be composed of discrete landmarks and often they increase in runtime with the size of the map. Extended Kalman Filter (EKF) localization [Leonard and Durrant-Whyte 1991] is an alternative technique that has both of these problems, which are exacerbated when landmarks cannot be identified exactly. Even with various optimizations to reduce execution time, such as using the unscented transform (UKF localization) [Julier and Uhlmann 1997] or multi hypothesis tracking (MHT), the algorithm still applies primarily to feature based maps [Thrun et al. 2005] (Section 7). Since a large, indoor environment is unlikely to have discrete, unambiguous features, these techniques are ineffective for the type of problem we are considering. In order to apply them it is usually necessary to preprocess the map as in [Leonard and Durrant-Whyte 1991] to create an artificial landmark based map. The map processing step introduces additional error and discards much of the information provided by the

original map. Another serious problem is that these alternative techniques cannot handle multiple hypotheses of the robot's location. Each one maintains only a single Gaussian representation of position. MCL, in contrast, can maintain multiple separate locations. Markov localization using probabilities over a grid is also possible, however it increases in runtime with the size of the state space, unlike MCL which only increases in runtime with the dimensionality of the state space. Since robots often operate in a large, real, dynamic environment, true Markov grid based localization requires such a large grid that it is usually impossible. Because of these drawbacks, MCL is currently the most effective localization technique based on the number of environments it applies to and the most commonly used, especially for real robots operating in real environments.

2.2.1 Recursive Bayes Filter

MCL is an implementation of a recursive Bayes filter. The posterior distribution of robot poses as conditioned by the sensor data is estimated as the robot's belief state. A key detail of the algorithm is the Markovian assumption that the past and future are conditionally independent given the present. For a robot this means that if its current location is known, the future locations do not depend on where the robot has been.

To produce a recursive Bayes filter, we represent the belief state of the robot as the probability of the robot's location conditioned by the sensor data, where sensors include odometry.

$$Bel(x_t) = p(x_t \mid z_t, z_{t-1} \ldots, z_0, u_t, u_{t-1}, \ldots, u_0) \quad \textbf{2-4}$$

x_t represents the robot's position at time t, z_t the robot's sensor readings at time t and u_t is the motion data at time t. To simplify the subsequent equations we use the notation $a^t = a_t, \ldots, a_0$.

$$Bel(x_t) = p(x_t \mid z^t, u^t) \quad \textbf{2-5}$$

While this equation is a good representation of the problem, it is not much use since it cannot be calculated as is. By applying a series of probabilistic rules, together with the Markovian assumption, equation (**2-4**) is converted into a more usable form. The first step is to use Bayes rule on equation (**2-5**) to provide a factorization.

$$Bel(x_t) = p(x_t \mid z_t, z^{t-1}, u^t) \quad \textbf{2-6}$$

$$= \frac{p(z_t \mid x_t, u^t, z^{t-1})\, p(x_t \mid u^t, z^{t-1})}{p(z_t \mid u^t, z^{t-1})} \quad \textbf{2-7}$$

Because the denominator is a normalizer, constant relative to the variable x_t, we can write equation (**2-7**) as

$$Bel(x_t) = \eta p(z_t \mid x_t, u^t, z^{t-1})\, p(x_t \mid u^t, z^{t-1}) \quad \textbf{2-8}$$

where $\eta = p(z_t \mid u^t, z^{t-1})^{-1}$ **2-9**

Once again, we make use of the Markovian assumption. In the case of the first probability term in equation (**2-8**) this means that since the robot's current location, x_t, is given, the current sensor

readings are independent of both the previous sensor readings, z^{t-1}, and the actions taken to get to location x_t, u^t. The result is that equation (**2-8**) simplifies to equation (**2-10**)

$$Bel(x_t) = \eta p(z_t \mid x_t) p(x_t \mid u^t, z^{t-1}) \quad \textbf{2-10}$$

Next, we expand the rightmost term by integrating over the state at time t-1.

$$Bel(x_t) = \eta p(z_t \mid x_t) \int_{x_{t-1}} p(x_t \mid u^t, x_{t-1}, z^{t-1}) p(x_{t-1} \mid u^t, z^{t-1}) dx_{t-1} \quad \textbf{2-11}$$

Once again, note that in the first term of the integral, the Markovian assumption makes x_t independent of the past, u^{t-1} and z^{t-1}, given the pose x_{t-1}. Also, we observe that the robot state at time t-1, x_{t-1}, is not affected by the next motion u_t. Although the motion u_t may in fact be affected by the location x_{t-1}, since the robot will probably only move in certain ways, we make the independence assumption for simplicity.

$$Bel(x_t) = \eta p(z_t \mid x_t) \int_{x_{t-1}} p(x_t \mid u_t, x_{t-1}) p(x_{t-1} \mid u^{t-1}, z^{t-1}) dx_{t-1} \quad \textbf{2-12}$$

Obviously, $p(x_{t-1} \mid z^{t-1}, u^{t-1})$ is $Bel(x_{t-1})$ giving us the recursive equation necessary for a recursive Bayes filter. η is a normalization constant that can be calculated by normalizing over the state space. $p(z_t \mid x_t)$ is the sensor model, representing the probability of receiving a particular sensor reading given a robot's location. Finally, $p(x_t \mid x_{t-1}, u_t)$ is the motion model. It is the probability that the robot arrives at location x_t given that it started at location x_{t-1} and performed action u_t. The sensor and motion model are representations of the physical components of the robot and must be determined experimentally for each robot and sensor device. The Bayes filter is examined in more detail in [Thrun et al. 2005] (Section 2.4).

2.2.2 Particle Approximation

It would appear that, given the two models, equation (**2-12**) is all that is necessary to perform localization with MCL. Unfortunately, a problem occurs with the integral. The equation requires integrating over the entire state space. Although we can evaluate the models at any point in the space, there is no closed form to the integral. Further, even a simple robot moves in a continuous, three dimensional state space with an x and y location together with an angle of rotation. Calculating the integral over this space is impossible, especially for a real time algorithm. In order to solve this problem, we approximate the continuous space with a finite number of weighted samples.

$$Bel(x_t) \approx \{x_t^i, w_t^i\}_{i=1,\ldots,N} \quad \textbf{2-13}$$

The integral over the space becomes a sum over the finite number of particles.

$$Bel(x_t) = \eta p(z_t \mid x_t) \sum_N p(x_t \mid u_t, x_{t-1}) p(x_{t-1} \mid u^{t-1}, z^{t-1}) \quad \textbf{2-14}$$

Of course, approximating the space results in a certain amount of error when low probability locations are not represented. If the robot is really at one of these locations it can never be localized. However, if the number of particles is well chosen, MCL works properly in most situations.

2.2.3 Resampling

One problem with using a finite set of particles to represent an infinite space is that the weight of particles representing a low probability location will quickly decrease. Assuming that these particles do not represent the location of the robot, their probability is unlikely to significantly increase. Since in MCL each particle is used to track the robot, a particle which almost certainly does not represent the robot's location will probably never be the correct location in the future. Similarly, if there are too few particles representing a high probability location, they will disperse and eventually lose the robot's position. What is needed is a method for relocating low probability particles to high probability locations and recalculating their probability. The method used in MCL is resampling. After the particles are weighted by the sensor model, they are resampled to represent the high probability locations. N particles are chosen randomly from the list of N weighted particles, with probability according to their weight. These particles are chosen with replacement, so that after a particle is chosen it remains in the original list and has the same probability of being sampled again. A high probability particle might be selected several times and so multiple copies might occur in the new list, while a low probability particle might never be chosen at all and its location would die out. The resampled list will thus have multiple particles in high probability locations and none in low probability ones. Another effect of resampling is to set all the sample weights to $\frac{1}{N}$. Instead of having individual weights representing the probability of a location, the number of particles indicates the probability. A high probability location will have many particles and thus, if the robot is present, it is likely to be tracked as it moves. Of course, low probability locations will die out and be unrepresented, so localization will fail if the robot is truly at one of these positions.

2.2.4 Bias

Representing an infinite space with a finite number of samples necessarily introduces some error. In order to accurately represent high probability locations, the particle filter discards lower probability regions as their low likelihood particles are not selected during the resampling process. Bias is the name given to the problem that MCL tends to consider only the highest probability locations, letting others be removed. The effect is that MCL is biased towards areas that have a large number of particles, tending to converge, over time, to a single cluster in the highest probability location. In most situations the high probability location contains the robot and so the convergence provides the correct result.

As an example, consider a situation with four particles evenly distributed between two separate locations A_1 and A_2. Assume that all sensor information is identical between the locations so that all particles always have an identical possibility of being selected during resampling. Also, in this example assume no motion occurs. We know that since two of the four particles are in each region, each particle x_1 through x_4 has a probability of 0.5 of being resampled in a particular location. Each of the four particles has two possible assignments (A_1 or A_2), leading to 16 possible configurations overall. Six of these configurations are the same as the original with two particles in each region. Two of them are total failures with all four particles in a single region. These are failures because there is no possible way for the correct, symmetrical configuration to be recovered from them. The

remaining eight configurations have three particles in one region with only one particle in the other. As a result, the expected number of resamples until the two regions are not equally represented is:

$$E(\overline{Same}) = \sum_{i=1}^{\infty} i \times \frac{10}{16} \times \left(\frac{6}{16}\right)^{i-1} = 1.6 \quad \textbf{2-15}$$

80% of the time this unbalance will be caused by three particles being in a single region while the remaining 20% of the cases are a total failure where all four particles have migrated to a single region, making the probability of sampling the other region zero. The expected number of resamples before total failure thus becomes:

$$E(Fail) = \frac{2}{10} E(\overline{Same}) + \frac{8}{10}\left(E(\overline{Same}) + E(Fail_2)\right) \quad \textbf{2-16}$$

Where $E(Fail_2)$ is the expected number of resamples to cause an unbalanced setup to fail completely.

From an unbalanced situation the same configurations are possible as before, however, the probability of selecting a particle from the region with one particle is only 0.25, while the other region has a probability of 0.75.

$$p(fail_2) = \left(\frac{3}{4}\right)^4 + \left(\frac{1}{4}\right)^4 = \frac{41}{128} = 0.3203125 \quad \text{probability of complete failure} \quad \textbf{2-17}$$

$$p(same_2) = 4\frac{1}{4}\left(\frac{3}{4}\right)^3 + 4\frac{3}{4}\left(\frac{1}{4}\right)^3 = \frac{60}{128} = 0.46875 \quad \text{probability of no change} \quad \textbf{2-18}$$

$$p(reset_2) = 6\left(\frac{1}{4}\right)^2\left(\frac{3}{4}\right)^2 = \frac{27}{128} = 0.2109375 \quad \text{probability of correction to first case} \quad \textbf{2-19}$$

The expected number of resamples until the state changes to either failure or the equal distribution case becomes:

$$E(Changes) = \sum_{i=1}^{\infty} i \times \frac{68}{128} \times \left(\frac{60}{128}\right)^{i-1} = \frac{32}{17} \quad \textbf{2-20}$$

The expected time to failure for the unbalanced case can then be written in terms of the expected time to failure for the first case.

$$E(Fail_2) = \frac{41}{68} E(Changes) + \frac{27}{68} E(Fail) \quad \textbf{2-21}$$

Subbing equation **2-21** back into **2-16** results in a solution to the overall number of resamples until failure from a balanced state.

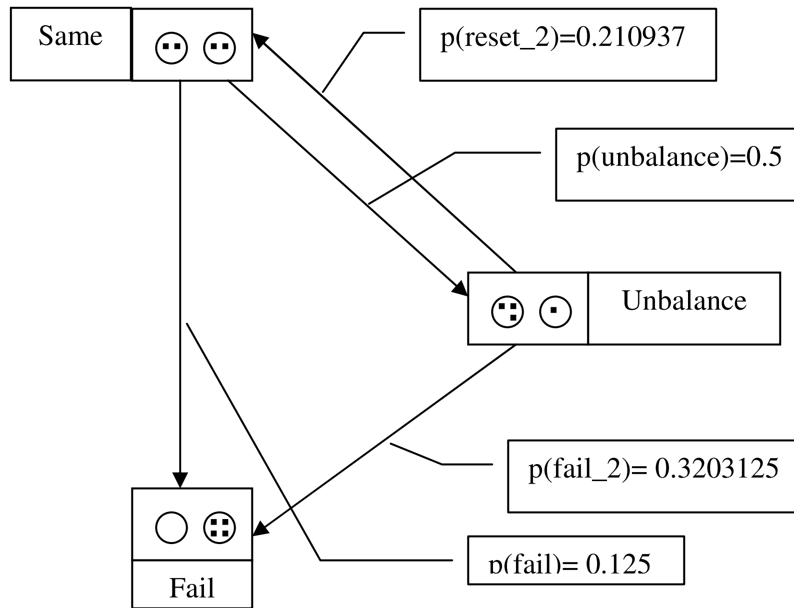


Figure 2-2: Graphical representation of the behaviour of bias for a simple case.

$$E(Fail) = \frac{2}{10} E(\overline{Same}) + \frac{8}{10}\left(E(\overline{Same}) + \frac{41}{68} E(Changes) + \frac{27}{68} E(Fail) \right) \quad \textbf{2-22}$$

$$E(Fail) = \frac{1812}{493} = 3.675456 \quad \textbf{2-23}$$

From this equation we can see that on average, it will take less than four resamples before all of the particles are in a single region, even though the distribution should actually still be evenly split between them. Because of the bias caused by the resampling of MCL's particle filter, the particles will always converge to a single location, even though this may not be the correct representation.

2.2.5 Algorithm

As the robot moves, it reports its odometry and sensor data to the MCL algorithm. After each reported move, every particle's new position is estimated according to the random motion model, based on the motion actually reported by the robot's dead reckoning. The particles are then updated with a weight determined by the sensor model for the particle's location. Finally, the particles are resampled by repeatedly choosing samples randomly, with replacement, from the current set, according to the weights assigned by the sensor model.

Table 2-1: MCL Algorithm

1:	Create a set $\{x_t^{[n]}, w_t^{[n]}\}$ from X_{t-1} by repeating N times:
1.1:	Choose a particle $x_{t-1}^{[n]}$ from X_{t-1}. Because of the resampling step this particle may be selected either iteratively or randomly.

1.2:	Next, draw a particle $x_t^{[n]} \sim p(x_t \mid u_t, x_{t-1}^{[n]})$. This particle is the result of a random motion according to the motion model.
1.3:	Set the weight of the particle using the sensor model: $w_t^{[n]} = p(z_t \mid x_t^{[n]})$.
2:	Resample randomly according to weight from $\{x_t^{[n]}, w_t^{[n]}\}$ into X_t, which causes the particle weights to become uniform.

The effect of resampling is to replace the weight of the individual particles with the number of particles at that location. On the robot's next move, the particles at a high probability location will spread out as they are moved randomly according to the motion model, with at least one probably landing in the robot's new location. Then, the resampling will cause more particles to appear at the correct location, while incorrect locations die out. Assuming that the models and map are accurate, MCL will correctly track the robot's changing location. Various parameters can be tuned manually to adjust the rate at which particles collect around a single hypothesis and the behaviour of the models. Once the belief over the robot's location is generated, a single location for the robot can be approximated as the mean of the particles.

2.2.6 Sensor Model

Corrections to the robot's location as determined by dead reckoning are made according to the robot's

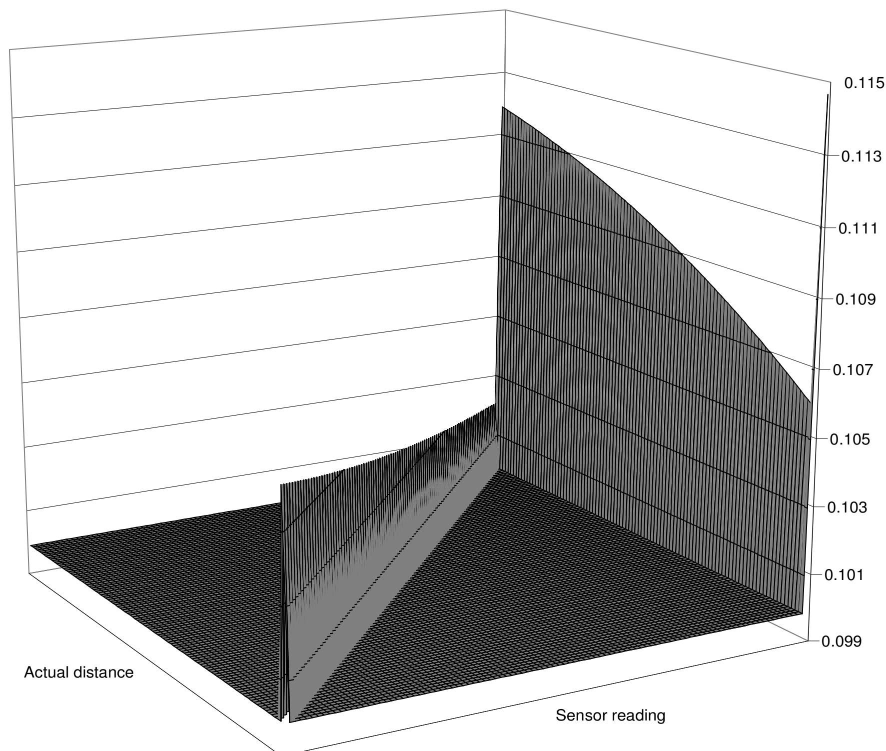


Figure 2-3: Implementation of a standard laser rangefinder sensor model.

other sensors. The sensors, usually some type of rangefinder device, determine the weight of each particle. The weight is calculated according to $p(z_t \mid x_t)$ which represents the sensor model, the probability of getting a particular sensor reading given a suggested robot location. The sensor model depends heavily on the exact physical sensors installed on the robot, so there can be no general equation. Since the model is not sampled, as is the motion model, it is often implemented as a large precalculated table, where any particular sensor probability can be quickly looked up. A table implementation allows a more complex function to be used than could be calculated in real time. One function that is sometimes used for a laser rangefinder device gives the probability of each possible returned range value, given each possible actual distance to a wall. Such a function can be composed of a Gaussian distribution centered on the actual wall distance, since that distance is the most probable return value, together with other functions depending on the features of the physical device. Common additions are an exponential function multiplied by a linear one representing false negative values and another exponential function representing false positives. Figure 2-3 gives a specific implementation of a sensor model.

The sample represented in Figure 2-3 is an implementation of this common model for a standard type of laser rangefinder, the SICK LMS200. For any specific laser reading and robot position the map is used to determine the expected distance to the wall. Given the distance returned by the laser and the actual distance determined by the map, the table that is graphed in Figure 2-3 returns the likelihood of getting that reading given the distance. Although the product of many such values is not a true probability, $p(z_t \mid x_t)$ can be determined by normalizing the likelihoods over all particles. It may look as if the function used in the figure is simply a standard Gaussian, however, the differences are visible on a closer examination as in Figure 2-4. The probabilities for a specific actual distance must sum to 1.0, which accounts for the peak at the maximum sensor reading value. This value is the probability that the laser does not observe any wall. Even with a good quality laser, this may occur frequently, depending on the environment. The close examination of a specific expected value in Figure 2-4 also reveals the effect of the false positive function, since the probability actually decreases, leading up to the peak around the expected distance. That function compensates for the possibility of a moving object, such as a person, passing between the robot and the wall. Although the effects of the additional function which are added to the model seem minor, they are sufficient to allow the particles to quickly congregate around the correct robot location in most situations, without requiring very many particles. Of course, this particular implementation assumes that individual range values are independent and it also violates the cell independence assumption as defined in section 2.1.3. However, that assumption was made specifically for the occupancy grid mapping algorithm and in order to provide a good sensor model we make different assumptions.

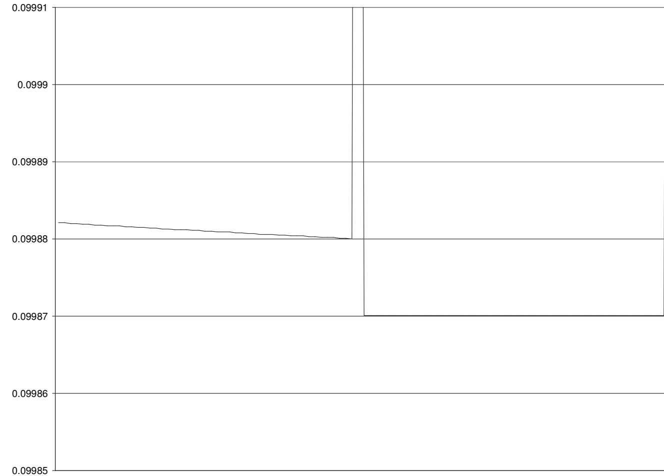


Figure 2-4: Probability values for a specific expected distance from Figure 2-3

The exact composition of the range probability function is so heavily dependent on the type and even the specific brand of sensor used, that no generally applicable functions or parameters can be given. Even another rangefinder device, a sonar, must have major modifications to the laser range probabilities in order to take into account multi-reflections, a common problem with sonars. The overall probability of a laser reading, which is composed of multiple range values in different directions, is the product of the probability of each range value. Given a robot position, the distance to the wall along each sensor ray can be determined from the map and the probability of the range value returned given that distance can be retrieved from the table. These probabilities are then multiplied together to get $p(z_t \mid x_t)$.

2.2.7 Motion Model

The motion model $p(x_t \mid x_{t-1}, u_t)$ is a critical part of MCL. As it is defined, MCL uses the motion

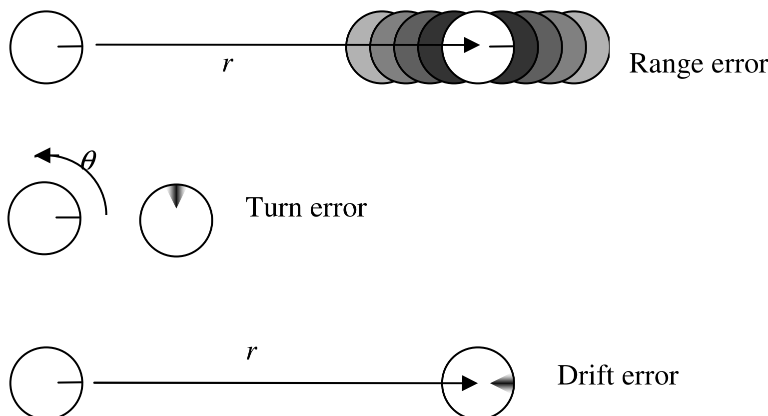


Figure 2-5: Standard types of motion error

model as the proposal distribution which predicts the location of each particle after a move. Although other proposal distributions are possible, as with SRL (section 2.2.10.4), the basic implementation is to determine the proposed location from the motion model. Unlike the sensor model, which gives the probability of getting a specific sensor reading at a particular location, it is necessary to sample from the motion model to create the proposal. Given a starting location and a reported motion (x_{t-1} and u_t), MCL requires that we be able to choose a final location randomly according to the motion model. Furthermore, this real-time requirement precludes us from using any motion model that is very complex. In fact, most motion models are a combination of simple Gaussian distributions. For a near-holonomic wheeled robot the most common representation is with two kinds of motion leading to three kinds of error. Each movement of the robot is represented as a linear movement followed by a stationary turn. Although a particular robot probably does not follow these exact motions, if we break the robot's motion into small increments we can use them as an approximation.

Each translation of the robot is approximated by a Gaussian where the mean is the reported distance and the variance is the reported distance multiplied by a parameter. This representation reflects the fact that the range error increases the further the robot travels. Rotation is also represented by a Gaussian. The mean is again the reported angle, but the variance is a parameter multiplied by the angle turned, added to another parameter times the distance moved. The variance takes into account both turn error, which increases as the robot turns, and drift error. Drift error is defined as the robot turning when it tries to go straight. Obviously, it increases the further the robot has travelled. Although it would seem that drift error should be minor, if it occurs at all, this is not in fact the case. Many near holonomic wheeled robots use a system where the difference in motion between the drive wheels is used to turn the robot. In such robots, moving forward is accomplished by turning both wheels the same amount, while turning is done by moving the wheels different amounts. It is very likely that, while moving forward, the wheels turn at slightly different rates, causing the robot to rotate. Figure 2-5 shows a graphical depiction of the three standard types of error. The three parameters involved in the model are often given as k_r for range error, k_θ for turn error, and k_d for drift error. The actual values, r and θ are determined according to the estimated values from the encoders, $\underline{r}$ and $\underline{\theta}$.

$$r \sim N(\underline{r}, k_r \bullet \underline{r}) \qquad \textbf{2-24}$$

$$\theta \sim N(\underline{\theta}, k_\theta \bullet \underline{\theta} + k_d \bullet \underline{r}) \qquad \textbf{2-25}$$

These two Normal distributions together represent the motion model for many common robots. However, the algorithms described in this section should work for any model, provided it is possible to sample from it. In general, some collection of Gaussians works well, since they are often good approximations to a physical system while at the same time being easy to sample from and optimize.

For robot motion models, it is usually assumed that the mean of a distribution from a reported motion will be the reported motion itself. Since the motion models are developed to represent the robot's motion, this would seem to be a reasonable assumption. If it was known that the mean of the robot's motion was different than the odometry, it would be necessary to change the odometry, since it is supposed to report motion as accurately as possible given the physical sensors used. There is no point in introducing additional parameters to alter the mean, because only one simple model is used to represent a robot's motion.

2.2.8 Importance Factor

In MCL, the weight of a particle is almost universally calculated based entirely on the sensor model. This weight, known as the importance factor of the sample, attempts to represent the difference between the proposal distribution generated by the motion model, and the target distribution $Bel(x_t)$. By using the ratio of distributions, we can derive the importance factor from first principles, allowing the possibility of using a different measure.

$$w_t^{[n]} = \frac{target\ distribution}{proposal\ distribution} \quad \textbf{2-26}$$

The target distribution is $p(x_t \mid z^t, u^t)$. The proposal distribution is given by $p(x_t|u_t,x_{t-1})p(x_{t-1}|z^{t-1},u^{t-1})$. This is the distribution of the samples $x_t^{[n]}$ before the re-sampling step. Using Bayes' rule the target distribution is expanded and the entire equation is simplified as follows.

$$w_t^{[n]} = \frac{p(x_t^{[n]} \mid z^t, u^t)}{p(x_t^{[n]} \mid u_t, x_{t-1})\, p(x_{t-1} \mid z^{t-1}, u^{t-1})} \quad \textbf{2-27}$$

$$w_t^{[n]} = \frac{\eta p(z^t \mid x_t^{[n]})\, p(x_t^{[n]} \mid u_t, x_{t-1})\, p(x_{t-1} \mid z^{t-1}, u^{t-1})}{p(x_t^{[n]} \mid u_t, x_{t-1})\, p(x_{t-1} \mid z^{t-1}, u^{t-1})} \quad \textbf{2-28}$$

$$w_t^{[n]} = \eta p(z^t \mid x_t^{[n]}) \quad \textbf{2-29}$$

The constant η can easily be ignored, since the importance weights are normalized in the re-sampling step. This leaves the term $p(z_t \mid x_t^{[n]})$, the sensor model, which is the importance factor usually used in MCL.

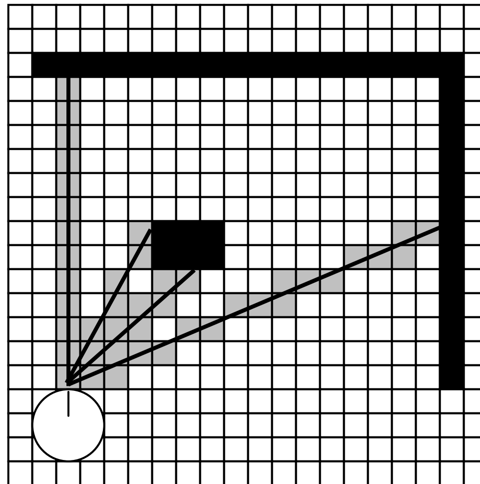


Figure 2-6: Occupancy Grid Map Raytracing

2.2.9 Raytracing

Calculation of the sensor model $p(z_t| x_t)$ involves determining the probability of receiving a particular sensor reading given the location in the environment. For a laser rangefinder, the readings are distance measurements. A common map implementation for MCL is an occupancy grid map with each cell holding the probability that it is occupied. Given a robot's possible location in the map, the expected distance to the wall is usually determined by raytracing from the robot to the nearest wall. Figure 2-6 illustrates the raytrace of four separate laser scans in an occupancy grid map. The shaded cells indicate all possible locations where the presence of an object might be detected by one of the scans. Each line length is the expected value of a laser reading taken by a robot at the marked starting location in a specific direction. The robot's physical sensors determine its actual distance to the wall and, once the distance from the map and the distance from the world are known, the probability can be calculated either mathematically or by a table lookup.

2.2.10 Additional Techniques

The basic MCL algorithm is very effective for localization, however there are several modifications that can provide an effective improvement in various circumstances.

2.2.10.1 Random Particles

Once MCL is modified to handle global localization properly, there is actually a simple modification that can handle the kidnapped robot problem. After the prediction step where the particles are moved according to the motion model, a small number of uniform random particles are added. Formally, the motion model is altered to have a probability of producing a location from a uniform distribution, instead of always producing if from the model in section 2.2.7. If the robot is accurately localized, these samples should have a low likelihood and will not be selected during resampling. Even if some of them survive, they will be removed on a subsequent update. However, if localization has failed, the new particles are just as likely to be high probability as any of the old ones. In that case, some of the random particles will survive, gradually spreading the distribution over the map. Since the particles are not clustered at a single high probability location, adaptive sampling will increase the number of particles while they are randomly scattered over the map. Eventually, some samples will be taken from the neighbourhood of the correct location and the distribution will converge, causing the number of samples to decrease again to what is necessary for position tracking.

Of course, the downside of random particles is that using them reduces the accuracy of ordinary MCL. In the overwhelmingly more common situation that the robot has not been kidnapped, the random particles add additional, unnecessary uncertainty to the distribution. It is even possible that the uncertainty caused by the random particles causes localization to fail when it would ordinarily succeed. Implementing this technique involves a balance between the overall number of particles and the number of random particles added. More random particles allow recovery from a kidnapping more quickly, while fewer such particles make the normal convergence more exact. There is no exact answer to this tradeoff, since in different situations, even in the same environment, MCL might require either kidnapping recovery or greater convergence. The designer must decide how much

accuracy can be traded for recovery from localization failures, based on the probability of kidnapping or localization failure.

2.2.10.2 Adaptive Sample Set Size

At first glance, it seems impossible for MCL to have a good set size. For normal localization, a small number of particles is necessary in order to improve execution speed, while global localization requires a large number. Similarly, a large number of samples helps with the kidnapped robot problem. Although global localization and localization failures are usually rare, it would still be useful if MCL could solve these problems. One possible solution is described in a paper by [Fox et al. 1999]. They propose a technique of adapting the sample size to the current conditions in localization. Thus, MCL can start with many particles in a uniform distribution over the state space, but as the distribution converges to a single area the number of particles decreases to what is necessary to maintain regular localization. If the robot becomes confused, for example because of traveling over rough terrain, the number of samples increases to cover the expanded belief. [Fox et al. 1999] use the simple technique of examining the unnormalized sample weights in the MCL algorithm. If the robot's position is well represented by a compact cloud of particles, the weights, from the sensor model $p(z_t \mid x_t)$, will be relatively large, while if the particles do not represent the correct location so well, their individual weights will be smaller. When the set is resampled, particles are chosen until the sum of their unnormalized weights exceeds a constant, γ. Thus, the sample set size automatically increases with an increase in uncertainty and decreases as the robot becomes more confident of its position. Adaptive sample set sizes handle the global localization problem well by decreasing the number of particles from the initial large number as the distribution converges. It can also handle disturbances that temporarily decrease the localization confidence. However, if the robot is kidnapped to a different location, adapting the sample set size will not help, since no samples will be drawn in the robot's new location. Using a technique as in section 2.2.10.1 may help with kidnapping when added to adaptive set size, but it might also cause larger sample sets to be created unnecessarily because of the low probability of the random particles.

2.2.10.3 KLD Sampling

We have seen that adapting the size of the sample set can improve the results of MCL, but the method used in [Fox et al. 1999] was only a heuristic technique. They used the sum of the unnormalized importance factors to determine how many samples were needed. The technique was effective, but in certain circumstances, for example with multiple equally likely hypotheses, it will not behave properly. A more formally defined method would be preferable, especially if it provided some bounds on error. One effective idea is called KLD-sampling and adapts the sample set size to effectively represent the current distribution. As the possible locations for the robot increase, the size increases and when the distribution collapses to fewer locations, KLD-sampling reduces the number of samples. The algorithm determines the number of samples that, with probability $1 - \delta$, make the error between the true posterior and the approximation less than ε. [Fox 2003]. The heart of KLD sampling is a measure between the true distribution and the sample based one that is called Kullback-Leibner divergence (KL distance). KL distance is a technique for measuring the distance between

two probability distributions which is combined with MCL to dynamically change the sample set size. The measure is calculated between two distributions $p(x)$ and $q(x)$ using the formula:

$$K(p,q) = \sum_x p(x) \log\left(\frac{p(x)}{q(x)}\right) \quad \textbf{2-30}$$

If we assume that samples are drawn from a set of k bins, then we can assume the size of the bins is randomly distributed according to a multinomial distribution. Fox [Fox 2003] showed with a set of standard derivations that this representation leads to the approximation for the number of particles necessary, n, in equation **2-31**, based on the Wilson-Hilferty transformation. In this equation, $z_{1-\delta}$ is an unusual term which refers to the upper 1 - δ quantile of the standard normal distribution. It cannot easily be calculated, but can be found for typical values of δ in statistical tables. Since 1 - δ is a constant giving the probability of the KL error, it is easy to choose a standard value that provides the accuracy desired. k is the number of bins with support, that is, the number of bins containing at least one sample. Although particles are drawn from a continuous space, rather than from bins, it is easy to define a theoretical set of bins over the state space and identify which bin any particular sample is from. Creating these bins should require minimal overhead, since very little new information is required.

$$n \cong \frac{k-1}{2\varepsilon}\left\{1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}}\, z_{1-\delta}\right\}^3 \quad \textbf{2-31}$$

Now that the number of particles necessary to represent the belief has been defined, it has to be incorporated into the MCL algorithm. The parameter needed is k, the number of bins that contain at least one particle. However, k is defined from the distribution, which consists of the n particles. Fortunately, the number of bins with support can be checked during sampling, and the number of particles needed can be continually updated after each sample, until the size of the set reaches the required value. The formula for n will usually increase with k, since the cubed term becomes unimportant as k increases. The set size is determined dynamically during sampling, increasing until it adequately represents the entire distribution.

Table 2-2: MCL with KLD Sampling

1:	$M = 0$, $M_x = \infty$, $k = 0$, $\alpha = 0$
2:	Create a set $\{x_t^{[n]}, w_t^{[n]}\}$ from $\{x_{t-1}^{[n]}, w_{t-1}^{[n]}\}$
2.1:	Choose a particle $x_{t-1}^{[n]}$ from $\{x_{t-1}^{[n]}, w_{t-1}^{[n]}\}$ randomly according to weight.
2.2:	Next, draw a particle $x_t^{[n]} \sim p(x_t \mid u_t, x_{t-1}^{[n]})$.
2.3:	Set the weight of the particle using the sensor model: $w_t^{[n]} = p(z_t \mid x_t)$.
2.4:	Add $<x_t^{[n]}, w_t^{[n]}>$ to the current set
2.5:	if $x_t^{[n]}$ falls into empty bin b
2.5.1:	$k = k + 1$
2.5.2:	b = non-empty
2.5.3:	if $k > 1$

2.5.3.1:	$M_x = \frac{k-1}{2\varepsilon}\left\{1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}} z_{1-\delta}\right\}^3$
2.6:	$M = M + 1$
3:	Repeat while $M < M_x$ (The number of particles created is less than the number needed)

It is easy to see that there is very little overhead necessary to augment MCL with KLD-sampling. Tracking the bins should be simple and the calculation of M_x is a constant time operation. The implementation uses the optimization of combining the resampling step of iteration t-1 with the particle selection of time t. This optimization reduces the processing required for each step while still producing the same results. However, at no particular stage does a set of unweighted samples exist. The weighted set can be used instead to determine the characteristics of the distribution, such as the mean, since the effects of the most recent motion are minimal. The KLD algorithm has the effect of providing a moving target for the sample set size, increasing the number of samples based on the area that they have to represent. Eventually, all new samples will occur in non-empty bins and the desired size of the set will stop increasing, while the number of samples continues to rise. The experiments in [Fox et al. 1999] showed that adaptive set sizes were an improvement over static sizes and the results of [Fox 2003] reinforce that result. KLD-sampling is very effective in global localization and when the localization error changes during execution, possibly because of temporary sensor failure or increased odometer noise. If the sensors fail temporarily, the previous technique will not work, since sample weights cannot be calculated without the sensors. The KLD-sampling technique provides a bound on the error which is introduced by the particle approximation, although various assumptions used mean the bound is not guaranteed for a real robot. However, KLD-sampling provides an improvement in MCL both in efficiency and localization performance, especially in situations where the variance of the distribution changes during execution.

2.2.10.4 SRL

Adding random samples to the set of particles can be helpful for solving the kidnapped robot problem [Fox et al. 1999], however, in a large space it is difficult to randomly select the correct location according to a uniform distribution. Also, the number of random samples must be kept small to avoid confusing ordinary position tracking, which is by far the most common circumstance. Adding only a small number of samples, it may take many cycles until the correct location is represented and localization begins to recover. In many environments, the robot is in danger if its localization fails. If the robot does not know where it is, it may be unable to avoid some hazards which it cannot detect. One serious problem is with descending staircases, which often cannot be sensed reliably. Further, in a time critical application such as robot soccer, the longer it takes to recover from kidnapping, the greater an advantage is given to the opposing team. A more effective technique than uniform random samples can be used in certain environments. Sensor Resetting Localization (SRL) is described by [Lenser and Veloso 2000] as a method for performing MCL with a step that draws some samples from the sensor distribution itself, instead of from the motion model applied to the previous state. SRL is more powerful than uniform random particles, since each new sample is guaranteed to be in a

high probability location. The MCL distribution is not harmed since low likelihood particles are not added.

SRL is implemented by adding additional steps into the MCL algorithm. After the new particle set has been created and weighted, but before resampling, the total of the importance factors is examined. The sum is the same as is used in [Fox et al. 1999] to dynamically determine the size of the sample set. If the sum of the weights is above a certain threshold, nothing more is done. However, if the sum is too low, additional samples are added from the sensor distribution $p(z_t \mid x_t)$ based on the weight sum. The number of new samples to be introduced is calculated by dividing the average particle weight by the threshold and multiplying by the current number of samples. The effect is to generate a percentage of samples based on the confidence of the current distribution. Of course, the SRL algorithm depends on having a sensor model from which samples can be taken, instead of merely determining the probability of individual particles. It is extremely difficult to sample from the distribution of a range sensor, since the distribution is likely to be both multimodal and discontinuous. However, if the localization is based on landmarks, it is easy to draw location samples from the sensor model. [Lenser and Veloso 2000] designed their algorithm for a robotic dog operating in a rectangular soccer field marked with special visual markers. Since the localization is based on vision of landmarks, it is easy to produce particles from the sensor model. Another reason to use SRL in this environment is because the legged motion of the robotic dogs has far greater error than wheeled motion. The error is especially pronounced when the robot collides with another robot or the boundaries of the arena. It is also possible, based on the rules of RoboCup, that a human may move the robot without warning. In this situation, it is critical that localization failures and kidnapping be resolved as quickly as possible. The results in the paper demonstrate that SRL is much more effective than uniform random particles in these scenarios.

Since SRL is very effective and also has a low overhead, it would be useful to incorporate it into MCL in many circumstances. Unfortunately, there is a serious drawback to using a technique which samples from the sensor model. If we must sample from the model, then the complexity of that model must be severely limited. In the case of localization, the term sensor model also includes the map of the environment. Using SRL thus requires that the sensor model, and by implication the map, be limited to a very low complexity. In [Lenser and Veloso 2000] the entire model consists of six landmarks and a probability distribution describing the accuracy of camera observations of these landmarks. Such a model easily lends itself to reversing the normal MCL process and drawing samples of state from the model itself, instead of just generating importance factors. However, SRL using a rangefinder and occupancy grid map is so complex that it is almost impossible to do in real time without additional approximations and complex algorithms [Thrun et al. 2001]. Although SRL is more effective than random particles, it can only be used in circumstances where the sensor model is very simple. In effect, it can only be used in landmark based maps. However, in those situations it can be very effective.

2.3 FastSLAM

2.3.1 SLAM Problem

The problem of determining the map and robot position at the same time is called Simultaneous Localization and Mapping, (SLAM). It involves finding the distribution over a state space which includes both robot position and the complete map. The given data is the sensor and odometry information from the start until the current time. Even the definition of SLAM results in two different problems. Determining the map and location during operation of the robot requires finding the current location x_t as well as the static map m. This results in the problem of online SLAM, which is intended to localize the robot during operation while also creating the map. Online SLAM is concerned with determining $p(x_t, m \mid z^t, u^t)$, which is the state at the current time. Using new information to update old estimates is not part of online SLAM, even though new information could update the map so that past localization could be corrected. Correcting past localization is often too computationally expensive for a real time solution. The other SLAM problem is called the full SLAM problem, and it involves finding the complete pose history of the robot and the map. The probabilistic formula is $p(x^t, m \mid z^t, u^t)$, since we want all of the robot's states, instead of just the current one. The difference is that full SLAM uses current data to correct past estimates. Online SLAM is used to localize the robot dynamically while full SLAM is often an offline algorithm concerned with finding out what the robot has already done. If the robot needs to make decisions based on its location, then it is necessary to use online SLAM. If the objective is to determine where a robot controlled by some other method, for example a human driver, has been, then full SLAM is more powerful. Both the problems have an application and the various solutions are similar, although not identical. In fact, online SLAM is the result of removing the past poses from the full problem using integration.

$$p(x_t, m \mid z^t, u^t) = \int_{x_{t-1}} \int_{x_{t-2}} \cdots \int_{x_1} p(x^t, m \mid z^t, u^t) dx_1 dx_2 \ldots dx_{t-1} \qquad \textbf{2-32}$$

Although SLAM is technically the definition of a particular problem, it is also the name given to the current set of solutions to the problem. These solutions all have several common elements which are shared by all effective solutions to both the online and full problems. One of the most important factors of most of the SLAM solutions is correspondences. Since SLAM considers the map as well as the robot pose, there must be some definition of a correct map. In SLAM, maps are defined as sets of objects and a correct map is one that has each object in the correct location. Of course, sensors do not report the location of specific objects, so it is necessary to find which object each sensor reading corresponds to. Unfortunately, it may be difficult to determine exactly which object is being observed. As we have seen, heuristic methods can be used to filter the raw sensor data into object locations, but any such technique will have a certain percentage of errors. Some SLAM algorithms explicitly take correspondence probabilities into account, adding yet another term to the posteriors. If we define c_t to be the set of correspondences between sensor readings and objects at time t, the online SLAM problem becomes $p(x_t, m, c_t \mid z^t, u^t)$, while the full problem is $p(x^t, m, c^t \mid z^t, u^t)$. In these two definitions we see that online SLAM only determines the current x_t and c_t at the current timestep, while full SLAM determines $x^t = \{x_0, ..., x_t\}$ and $c^t = \{c_0, ..., c_t\}$, the entire history, at

each timestep. Of course, increasing the size of the state space significantly increases the complexity of the problem and thus the run time of the solution. Many SLAM algorithms can be proved to eventually converge to the correct map, but only if the objects can be identified correctly [Montemerlo et al. 2002; Folkesson and Christensen 2004; Thrun et al. 2005] (Thrun: Sections 10-13). If identification is not certain, the guarantee of convergence is lost. The problem of determining which object is detected by a sensor reading is called the data association problem and it is the central drawback to many SLAM algorithms. In effect, SLAM usually creates a landmark based map, rather than a pure occupancy grid map. As we have seen, correctly identifying landmarks in localization is a difficult problem, which can be overcome by using raw sensor readings in the MCL algorithm. The data association problem in SLAM can be solved using a corresponding solution, however, this suffers from additional problems.

2.3.2 FastSLAM Derivation

Simultaneous Localization and Mapping is divided into two slightly different domains. The first, called online SLAM, is the problem of finding the robot's current pose x_t and the map m, given the sensor readings z^t and odometry u^t. The more complex problem is to find the robot's path $x^t = \{x_1, ..., x_t\}$ given the same data. Finding the complete path is called the full SLAM problem. Obviously, full SLAM is the more complete problem but online SLAM can be derived from full by integrating out the past poses, as shown in equation (**2-32**).

$$\text{Full SLAM}: \; p(x^t, m \mid z^t, u^t) \qquad \textbf{2-33}$$

$$\text{Online SLAM}: \; p(x_t, m \mid z^t, u^t) = \int_{x_{t-1}} \int_{x_{t-2}} \cdots \int_{x_1} p(x^t, m \mid z^t, u^t) dx_1 dx_2 \ldots dx_{t-1} \qquad \textbf{2-34}$$

The reduced problem is called online SLAM because it is simplified enough to be solved in real time, whereas most full SLAM solutions require offline processing.

One of the benefits of FastSLAM [Montemerlo et al. 2002] is that it simultaneously solves both the online and full SLAM problems. Of course, any solution to the full SLAM problem also produces a solution to online SLAM, but FastSLAM, although it works in real time, solves for the entire path of the robot. The key to the FastSLAM derivation is the assumption that if the robot's location is known, the locations of objects in the environment are independent. Thus, the SLAM problem can be factored into localization and mapping problems.

$$p(x^t, m \mid z^t, u^t) = p(x^t \mid z^t, u^t) \prod_n p(m_n \mid x^t, z^t) \qquad \textbf{2-35}$$

The first term is obviously a localization problem which requires finding the robot's path x^t given its odometry and sensor data. The second term is the mapping problem which finds a particular feature's position, m_n, given the robot's path and sensor readings. Equation (**2-35**) leads to an iterative algorithm for FastSLAM where the robot's position is calculated, and then the map is updated based on that position. Unfortunately, it is unlikely that at every step a single location for the robot can be derived. Nor can the algorithm rely on a probabilistic position, since the factoring is only valid given a fixed x^t. These requirements lend themselves to a technique that represents the robot's location as a

finite collection of exact states. Fortunately, such a technique, Monte Carlo Localization (MCL), exists.

2.3.3 Feature Based FastSLAM

Since particle filters represent the robot's belief as a set of discrete samples, each one corresponding to an exact location, the map features can be located in relation to the individual particles. The FastSLAM algorithm uses particle filters and Gaussian distributions in a technique called Rao-Blackwellized particle filters. Each particle contains the robot pose and the Kalman filter over each map feature. The correspondence decisions for the case of unknown correspondences are also made on a per particle basis. The benefit is that for any particular particle, the robot's location is known, so it should be easy to determine which landmark is observed by each sensor reading. In order to estimate the correspondences, the data stored by each particle includes the number of landmarks observed as well as the probability of existence for each feature. At each timestep, the algorithm updates each particle, first updating the robot state according to the reported motion, and then updating the observed features as determined by the estimated data associations. Finally, a new set of particles is resampled to get the belief at the current timestep.

Table 2-3: Feature Based FastSLAM Algorithm

[Thrun et al. 2005] (Section 13.7)

FastSLAM_1.0(z_t, u_t, Y_{t-1})

1: for $k = 1$ to M do // loop over all particles

2: $retrieve \left\langle x_{t-1}^{[k]}, N_{t-1}^{[k]}, \left\langle \mu_{1,t-1}^{[k]}, \Sigma_{1,t-1}^{[k]}, i_1^{[k]} \right\rangle, \ldots, \left\langle \mu_{N_{t-1}^{[k]},t-1}^{[k]}, \Sigma_{N_{t-1}^{[k]},t-1}^{[k]}, i_{N_{t-1}^{[k]},t-1}^{[k]} \right\rangle \right\rangle from\ Y_{t-1}$

3: $x_t^{[k]} \sim p(x_t \mid x_{t-1}^{[k]}, u_t)$

4: for $j = 1$ to $N_{t-1}^{[k]}$ do // measurement likelihoods

5: $\hat{z}_j = g(\mu_{j,t-1}^{[k]}, x_t^{[k]})$

6: $G_j = g'(\mu_{j,t-1}^{[k]}, x_t^{[k]})$

7: $Q_j = G_j^T \Sigma_{j,t-1}^{[k]} G_j + R_t$

8: $w_j = \left|2\pi Q_j\right|^{-\frac{1}{2}} \exp\{-\frac{1}{2}(z_t - \hat{z}_j)^T Q_j^{-1}(z_t - \hat{z}_j)\}$

9: endfor

10: $w_{1+N_{t-1}^{[k]}} = p_0$

11: $w^{[k]} = \max w_j$

12: $\hat{c} = \arg\max w_j$

13: $N_t^{[k]} = \max\{N_{t-1}^{[k]}, \hat{c}\}$

14: for $j = 1$ to $N_t^{[k]}$ do // update Kalman filters

15: if $j = \hat{c} = 1 + N_{t-1}^{[k]}$ then // is new feature?

16: $\mu_{j,t}^{[k]} = g^{-1}(z_t, x_t^{[k]})$

17: $\Sigma_{j,t}^{[k]} = G_j^{-1} R_t (G_j^{-1})^T$

18: $i_{j,t}^{[k]} = 1$

19: else if $j = \hat{c} \leq N_{t-1}^{[k]}$ then // is observed feature?

20: $K = \Sigma_{j,t-1}^{[k]} G_j Q_{\hat{c}}^{-1}$

21: $\mu_{j,t}^{[k]} = \mu_{j,t-1}^{[k]} + K(z_t - \hat{z}_{\hat{c}})^T$

22: $\Sigma_{j,t}^{[k]} = (I - KG_j^T)\Sigma_{j,t-1}^{[k]}$

23: $i_{j,t}^{[k]} = i_{j,t-1}^{[k]} + 1$

24: else // all other features

25: $\mu_{j,t}^{[k]} = \mu_{j,t-1}^{[k]}$

26: $\Sigma_{j,t}^{[k]} = \Sigma_{j,t-1}^{[k]}$

27: if $\mu_{j,t-1}^{[k]}$ outside perceptual range of $x_t^{[k]}$ then

28: // should feature have been seen?

29: $i_{j,t}^{[k]} = i_{j,t-1}^{[k]}$ // do not change

30: else

31: $i_{j,t}^{[k]} = i_{j,t-1}^{[k]} - 1$ // decrement counter

32: if $i_{j,t-1}^{[k]} < 0$ then

33: discard feature j

34: endif

35: endif

36: endif

37: endfor

38: $add \left\langle x_t^{[k]}, N_t^{[k]}, \left\langle \mu_{1,t}^{[k]}, \Sigma_{1,t}^{[k]}, i_1^{[k]} \right\rangle, \ldots, \left\langle \mu_{N_t^{[k]},t}^{[k]}, \Sigma_{N_t^{[k]},t}^{[k]}, i_{N_t^{[k]}}^{[k]} \right\rangle \right\rangle$ $to\ Y_{aux}$

39: endfor

40: $Y_t = \{\}$ // construct new particle set

41: do M times // resample

42: draw random index k with probability $\propto w^{[k]}$

43: $add \left\langle x_t^{[k]}, N_t^{[k]}, \left\langle \mu_{1,t}^{[k]}, \Sigma_{1,t}^{[k]}, i_1^{[k]} \right\rangle, \ldots, \left\langle \mu_{N_t^{[k]},t}^{[k]}, \Sigma_{N_t^{[k]},t}^{[k]}, i_{N_t^{[k]}}^{[k]} \right\rangle \right\rangle$ $to\ Y_t$

44: enddo

45: return Y_t

The basic algorithm can, as usual, be improved to be more efficient. The main improvement involves the representation of the map. Instead of copying the map into each sample, the samples contain only the changes from their parent particle. By using a tree representation, the efficiency of the FastSLAM algorithm can be O(M log N), where M is the number of particles and N is the size of the map. The optimization of only searching local features for correspondences can also be used to improve the unknown associations case. A further improvement, known as FastSLAM 2.0, uses the

sensor readings to predict a better particle during the particle filter step. When the motion of the robot does not correspond well to its model, or has a very high error, FastSLAM 2.0 can provide significant improvements in efficiency and accuracy. In fact, the algorithm can succeed with very few particles, and even with only one in some situations [Montemerlo et al. 2003]. Using sensor readings to improve the prediction is only possible because the definition of the map as a set of independent Gaussian landmarks allows sampling from the sensor model. In an occupancy grid map, that is usually not possible. These optimizations make FastSLAM one of the most efficient solutions to the SLAM problem. Experiments performed by [Montemerlo et al. 2002] successfully localized and created a map in a large environment using a small, fixed number of particles. Since FastSLAM produces the entire path as well as the current position it is also unique in solving both online and full SLAM in real time.

2.3.4 Occupancy Grid FastSLAM

Unlike most SLAM algorithms, it is possible to use FastSLAM on an occupancy grid map directly, without first processing it for landmarks. Since the algorithm updates the map with reference to a given robot position, the specific representation of the map as Gaussian landmarks is not required. Instead, an occupancy grid can be used and updated according to the standard techniques. As in [Moravec 1988], observed cells are updated according to whether the sensors observed them as occupied or unoccupied. However, since each particle represents an exact robot path, there is no need to blur past readings. The grid cell implementation even overcomes the data association problem, since landmarks can no longer move, the cell being observed is exactly determined by the robot's location. One specific implementation of FastSLAM with occupancy grid maps, called DP-SLAM, [Eliazar and Parr 2004] was able to successfully localize and map a real environment including a large loop.

The only serious problem with FastSLAM occurs with the difficult situation of loop closure. In other algorithms when the robot re-enters known territory it becomes necessary to search a much larger set of landmarks for correspondences, possibly the entire set. However, FastSLAM represents all possible robot positions in a finite set of samples. When it closes a loop, it can only be successful if some particle has followed an approximately correct path. The longer the loop, the greater the uncertainty of the robot's position. As uncertainty increases, the number of particles necessary to represent the belief also increases. Eventually, there will not be enough particles to represent the distribution and the correct location may be lost. FastSLAM alone suffers the problem, since all other SLAM solutions use the correlations to determine the position. The problem with particle filters is that they only represent the highest probability region of a distribution, whereas the Gaussian distributions used by other techniques represent the entire distribution. Of course, particles can represent much more complex and nonlinear distributions than is possible with Gaussians. The drawback to using particle filters in FastSLAM is that the number of particles maintains the diversity of the robot's position, and as soon as the uncertainty goes beyond the number of particles, the algorithm may fail. Thus, the size of the particle set must be tuned to the environment, based on the size of the longest loop, and increasing the number of particles to this extent may make the algorithm inefficient.

Grid based FastSLAM is performed by combining the MCL particle filtering with the occupancy grid mapping algorithm using Rao-Blackwellized particle filters [Montemerlo et al. 2002; Thrun et al. 2005] (Thrun: Section 13.10). Each particle consists of both the robot's state and an occupancy grid map. Of course, particle filtering could be used over the entire state space. However, this would require a number of particles exponential in the number of state variables, in this case the number of cells in the map. Instead, the factorization in equation (**2-35**) is used to separate the robot state from the map. The particle filter is only used for the robot's state, often x, y and orientation (θ) for a terrestrial indoor robot. The map for each particle is updated according to the occupancy grid mapping algorithm, with the position fixed at the position of the particle. This separation allows the occupancy grid algorithm to work with a guaranteed position, while still allowing for uncertainty in the robot's pose. By looking at the highest probability location we can determine the current best guess of the robot's position and the map. At each step, the set of N particles, X_{t-1}, is updated to X_t according to the algorithm shown in Table 2-4

Table 2-4: Occupancy Grid FastSLAM algorithm

1: for k = 1 to N
2: $\quad x_t^{[k]} \sim p(x_t \mid x_{t-1}^{[k]}, u_t, m)$
3: $\quad w_t^{[k]} = p(z_t \mid x_t^{[k]}, m)$
4: $\quad \forall n.\; l_{t,n}^{[k]} = l_{t-1,n}^{[k]} + \log \frac{p(m_n^{[k]} \mid x_t^{[k]}, z_t)}{1 - p(m_n^{[k]} \mid x_t^{[k]}, z_t)} - \log \frac{p(m_n)}{1 - p(m_n)}$
5: $\quad X_t' = X_t' \cup \langle x_t^{[k]}, m_t^{[k]}, w_t^{[k]} \rangle$
6: endfor
7: for k = 1 to N
8: $\quad$ draw i from X_t' with probability $\alpha\ w_t^{[i]}$
9: $\quad X_t = X_t \cup \langle x_t^{[i]}, m_t^{[i]} \rangle$
10: endfor

In Table 2-4, line 2 updates the set of particles by randomly choosing a new location for each one based on the previous location of the particle and the motion model. The sensor model is used to determine a weight for the new location based on the sensor readings in line 3. The primary difference from MCL occurs in line 4 where the occupancy probability of the map attached to the particle is updated according to the sensor readings used at the particle's new location. Of course, these maps are maintained via the log odds ratio as in section 2.1.3. Finally, in the loop of lines 7 - 10, the updated particles are resampled. N new particles are chosen randomly according to the weights, with replacement, to make the new particle set. Resampling has the effect of replacing the particle weight with the number of samples at a location. Thus, low probability locations die out while high probability locations gather enough particles that, on the next update, the correct location will be selected by the motion model in line 2.

Chapter 3
Cluster MCL

3.1 Introduction

Global localization is the problem of localizing a robot from a uniform posterior over the environment. Although the robot has a pre-existing map, it has no idea as to its actual position. With a reasonably accurate map of the environment, MCL has been shown to be effective in many situations. However, MCL suffers an important limitation: when samples are generated according to their posterior probability (as is the case in MCL), they often too quickly converge to a single, high-likelihood pose. This might be undesirable in symmetric environments, where multiple distinct hypotheses have to be tracked for extended periods of time. MCL often converges to one single location too quickly, ignoring the possibility that the robot might be somewhere else. This problem leads to suboptimal behaviour if there are two or more equally likely poses. In symmetric environments, it is desirable to maintain a higher diversity of the samples, despite the fact that likelihood-weighted sampling will favour a single robot pose.

The approach we present in this chapter introduces the idea of clusters of particles and modifies the proposal distribution to take into account the probability of a cluster of similar poses. Each cluster is considered to be a hypothesis of where the robot might be located and is independently developed using the MCL algorithm. The update of the probability of each cluster is done using the same Bayesian formulation used in MCL, thus effectively leading to a particle filter that works at two levels, the particle level and the cluster level. While each cluster possesses a probability that represents the belief of the robot being at that location, the cluster with the highest probability would be used to determine the robot's location at that instant in time.

The main benefit of using clusters is that it maintains diversity in the particle set while not reducing the rate at which a single location is found. The enforced diversity allows MCL to handle additional situations, such as symmetric environments, without compromising its behaviour in ordinary circumstances. The clusters also provide a mechanism to recover from kidnapping without the random samples affecting the localization quality.

Multiple belief states can also be represented using multi-hypothesis Kalman filters. This, however, inherits Kalman filters' limitations in that it requires noise to be Gaussian. Particle filters, on the other hand, can represent arbitrary distributions. Since all objects must have Gaussian noise for Kalman filters to work it is common to require that maps consist of a number of discrete landmarks [Austin and Jensfelt 2000; Jensfelt and Kristensen 2001]. MCL provides the great benefit of using raw sensor data directly. Other improvements to MCL, like dual-MCL, SRL and Mixture MCL (a technique for combining multiple proposal distributions) [Lenser and Veloso 2000; Thrun et al. 2000; Thrun et al. 2001], attempt to improve the proposal distribution, however they require that the distributions have certain specific features that are not often present. None of these other techniques function in the general case that we consider here.

Experiments have been conducted with both simulated data as well as data obtained from a robot, using laser range finder data collected at multiple sites. The environments are highly symmetric and the corresponding datasets possess only a very small number of distinguishing features that allow for global localization. Thus, they are good testbeds for our proposed algorithm. Results show that the Cluster-MCL algorithm is able to successfully determine the position of the robot in these datasets, while ordinary MCL often fails.

3.2 Cluster-MCL

3.2.1 Clustered Particle Filtering

In MCL, $p(z_t \mid x_t^{[n]})$ is calculated as the importance factor $w_t^{[n]}$. However, the analysis in section 2.2.8 suggests that a much broader range of functions may be used as proposal distributions. In particular, let $f_t(x_t)$ be a positive function over the state space. Then the following particle filter algorithm generates samples from a distribution $f_t(x_t)\mathrm{p}(x_t \mid z^t, u^t)$. Initially, samples are drawn from $f_0(x_0)$. New sample sets are then calculated via the following procedure:

First, draw a random particle $x_{t-1}^{[n]}$ from X_{t-1}. By assumption, this particle is distributed according to $f_{t-1}(x_{t-1})p(x_{t-1} \mid z^{t-1}, u^{t-1})$ with the distributions being equal as N approaches infinity.

Next, draw a state $x_t^{[n]} \sim p(x_t \mid u_t, x_{t-1}^{[n]})$.

In this case the resulting importance factor is easily computed as:

$$w_t^{[n]} = \frac{\textit{target distribution}}{\textit{proposal distribution}}$$

$$w_t^{[n]} = \frac{f_t(x_t^{[n]})p(x_t^{[n]} \mid z^t, u^t)}{f_{t-1}(x_{t-1}^{[n]})p(x_t^{[n]} \mid u_t, x_{t-1})p(x_{t-1} \mid z^{t-1}, u^{t-1})}$$

$$w_t^{[n]} = \frac{f_t(x_t^{[n]})\eta p(z^t \mid x_t^{[n]})p(x_t^{[n]} \mid u_t, x_{t-1})p(x_{t-1} \mid z^{t-1}, u^{t-1})}{f_{t-1}(x_{t-1}^{[n]})p(x_t^{[n]} \mid u_t, x_{t-1})p(x_{t-1} \mid z^{t-1}, u^{t-1})}$$

$$w_t^{[n]} \propto p(z^t \mid x_t^{[n]})\frac{f_t(x_t^{[n]})}{f_{t-1}(x_{t-1}^{[n]})} \qquad \textbf{3-1}$$

The proposed clustering particle filter employs such a modified proposal distribution. In particular, each particle is associated with one out of K clusters. We will use the function $c(x_t)$ to denote the cluster number. The function f_t assigns to each particle in the same cluster the same value; but this value may differ among different clusters. Moreover, f_t is such that the cumulative weight over all the particles in each cluster is the same for each cluster.

$$\sum_{x_t^{[n]} \in X_t : c(x_t^{[n]}) = k} f(x_t^{[n]})p(x_t^{[n]} \mid z^t, u^t) = \sum_{x_t^{[n]} \in X_t : c(x_t^{[n]}) = k'} f(x_t^{[n]})p(x_t^{[n]} \mid z^t, u^t) \qquad \textbf{3-2}$$

for $k \neq k'$.

Since the clusters are considered separately, and, more particularly, have their particle weights normalized independently of the other clusters, we see that equation **3-2** must be valid. However, we need to define $f(x_t^{[n]})$. Since these are equal for all x such that $c(x) = k$, it is sufficient to define $f\left(x_t^{[n]}\right) = B_{k,t}$ where $k = c(x_t^{[n]})$ and $B_{k,t}$ is the probability, at time t, that cluster k contains the actual robot position. We can estimate the $B_{k,t}$ values using standard Bayes filters. Here, we use k to represent a random variable over the clusters:

$$B_{k,t} = p(k_t \mid z^t, u^t)$$

$$B_{k,t} = \eta \int_{k_{t-1}} p(z_t \mid k_t) p(k_t \mid u_t, k_{t-1}) p(k_{t-1} \mid z^{t-1}, u^{t-1}) dk_{t-1} \qquad \textbf{3-3}$$

Since we use a finite number of samples to approximate the distribution, this becomes:

$$B_{k,t} = \eta \frac{\sum_k p(z_t \mid k_t) p(k_t \mid u_t, k_{t-1}) p(k_{t-1} \mid z^{t-1}, u^{t-1})}{n} \qquad \textbf{3-4}$$

Now we note that, although the robot can move from one point to another, particles cannot change clusters. That is, each particle starts in one cluster and remains in that cluster. This being the case,

$$p(k_t \mid u_t, k_{t-1}) = \begin{cases} 0 & \text{if } k_t \neq k_{t-1} \\ 1 & \text{if } k_t = k_{t-1} \end{cases} \qquad \textbf{3-5}$$

Therefore, $B_{k,t} \propto p(z_t \mid k_t) p(k_{t-1} \mid z^{t-1}, u^{t-1})$ **3-6**

We also note that a cluster is composed of a set of points. Therefore, $p(z_t \mid k_t)$ is related to $p(z_t \mid x_t)$. In fact, the distribution of sensor readings for a cluster must be the sum of the distributions of sensor readings for all points in the cluster. That is:

$$B_{k,t} \propto p(k_{t-1} \mid z^{t-1}, u^{t-1}) \sum_{x_t^{[n]} \in X_t : c(x_t^{[n]}) = k} p(z_t \mid x_t^{[n]}) \qquad \textbf{3-7}$$

Given equations (**3-3**) and (**3-7**) we can write

$$B_{k,t} = \gamma B_{k,t-1} \sum_{x_t^{[n]} \in X_t : c(x_t^{[n]}) = k} p(z_t \mid x_t^{[n]}) \qquad \textbf{3-8}$$

where γ is a normalization factor.

Having defined $f(x_t^{[n]}) = B_{k,t}$, we maintain the condition stated in equation (**3-2**) by normalizing after each iteration. Therefore, our modified proposal distribution is satisfies the requirements to be used as the proposal distribution in MCL.

3.2.2 Algorithm

Based on the mathematical derivation above, we have implemented an extension to MCL, called Cluster-MCL. Cluster-MCL tracks multiple hypotheses organized in clusters. The first task is to identify probable clusters. By iterating several steps through ordinary MCL, with an initial uniform

distribution of a large number of points, clusters develop in several locations. We then use a simple clustering algorithm to separate the points into different clusters. We match each point with a cluster based on the distance, in all three dimensions, between that point and the source point of the cluster. If a particle does not match any cluster it becomes the source point for a new one. The initial probability of a cluster is based on the number of points it contains. There are more robust clustering algorithms, based on the Expectation Maximization (EM) algorithm. However, these methods rely on an a priori knowledge of the number of clusters. Our method generates an arbitrary number of clusters determined by the number of clusters in the environment, with each cluster having an arbitrary size. The drawback is that several clusters may be created in almost the same location. We solve this problem by occasionally checking for overlapping clusters and combining them. Once clusters are generated, we select the most probable ones and discard the others.

Each cluster is then independently evolved using ordinary MCL. Thus, points selected for a particular cluster can only be drawn from that cluster. The probability of each cluster is tracked by multiplying the prior probability of the cluster by the average of the likelihood of the points in that cluster. These probabilities are kept normalized and correspond to the $B_{k,t}$ values as defined above.

There is the problem that, if there is an error in the map in the initial location, there may be no cluster generated at the correct location. We solve this problem, and also the kidnapped robot problem, by taking advantage of the independence of our clusters. The kidnapped robot problem is where the robot is moved by an outside force after being localized. Since clusters do not interfere with each other, we can add a cluster in a new location without affecting our existing clusters. After a predetermined number of steps, we restart a new instance of MCL with the particles uniformly distributed, with the purpose of finding the most likely cluster based on the current sensor data. Once the new cluster has converged to a single location, we check whether this new location overlaps an existing cluster within some threshold. If it does not, it covers a currently unrepresented point so we initialize it to have a small probability and begin tracking it. Otherwise, the location is already represented and we discard the new cluster and repeat the process. By doing this, we remain open to consideration of a completely new location for the robot based on the current sensor data.

To limit the number of clusters from growing out of bounds and to remain computationally efficient, we limit the number of clusters to a maximum pre-defined value. Additionally, by keeping the number of clusters fixed at all points in time, we prevent a cluster from gaining a high probability by competing with only few other clusters, which would tend to prevent that cluster from being overtaken when there are many other clusters. When adding a new cluster, the least probable cluster is removed, in order to keep the size fixed.

The robot's estimate of its own location is based on the most likely cluster, and can be obtained using the mean or highest weight particles in that cluster as with ordinary MCL. Since each cluster represents a distinct location, averaging across multiple clusters would produce a location between the clusters, in a region that where there is very little likelihood the robot is present. Each cluster is based around a high probability area, but the regions between clusters are low probability.

3.2.3 Example

If we refer back to the example given in section 2.2.4, we can see that resampling only within a cluster alters the transition probabilities between the states. In particular, p(unbalance) and p(fail) are both zero, because it is impossible to change the total number of particles in an individual cluster. Then, the expected value from equation **2-15** is calculated as:

$$E(\overline{Same}) = \lim_{j\to 0}\sum_{i=1}^{\infty} i \times j \times (1-j)^{i-1} = \lim_{j\to 0}\frac{1}{j} = \infty \qquad \textbf{3-9}$$

Which causes equation **2-16** to also be infinity, regardless of the characteristics of the unbalanced state, which is never reached.

$$E(Fail) = \frac{2}{10}E(\overline{Same}) + \frac{8}{10}\left(E(\overline{Same}) + E(Fail_2)\right) = \frac{2}{10}\infty + \frac{8}{10}\left(\infty + E(Fail_2)\right) = \infty \qquad \textbf{3-10}$$

Thus, no matter how many resampling steps are performed, the clusters retain equal particle mass. The additional value $B_{k,t}$ that must be tracked is also updated on each step according to equation **3-8**. However, because the sensor readings are the same for all particles in either region, according to the assumption in section 2.2.4:

$$p(z_t \mid x_t^{[n]}) = p(z_t \mid x_t^{[m]}) \forall c(x_t^{[n]}) = 1, c(x_t^{[m]}) = 2 \qquad \textbf{3-11}$$

And so:

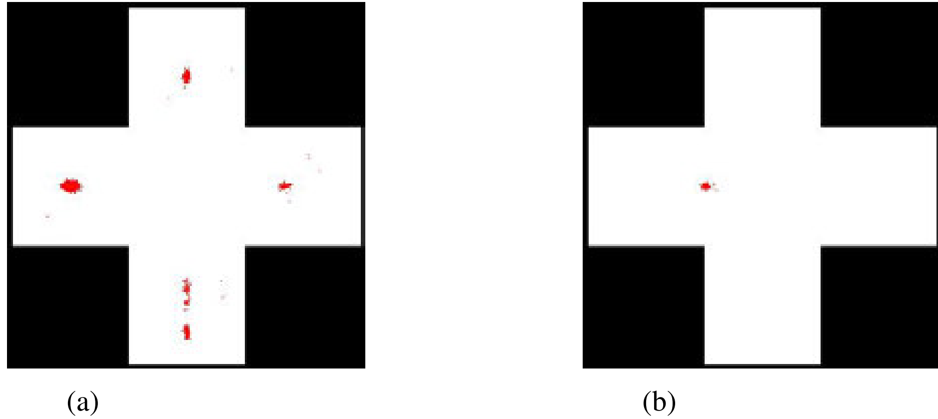
$$\sum_{x_t^{[n]} \in X_t : c(x_t^{[n]}) = 1} p(z_t \mid x_t^{[n]}) = \sum_{x_t^{[n]} \in X_t : c(x_t^{[n]}) = 2} p(z_t \mid x_t^{[n]}) \qquad \textbf{3-12}$$

With the result that $B_{1,t} = B_{2,t}$, given that $B_{1,0} = B_{2,0}$. which was another assumption. Since cluster probabilities must be normalized to eliminate the unknown constant and there are only two clusters we can determine that $B_{k,t} = 0.5$ for all clusters. The clustering algorithm completely solves the problem of bias in this example, maintaining two equally weighted clusters indefinitely.

3.3 Experimental Evaluation

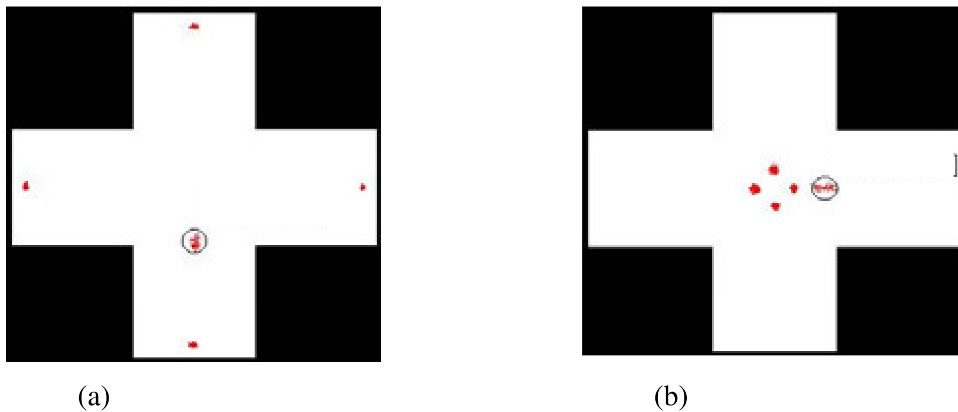
The Cluster-MCL algorithm was implemented and tested in both simulated and real environments. In these tests, we compare the performance of our Cluster-MCL algorithm with that of ordinary MCL. In all cases, we found that Cluster-MCL performed as well as ordinary MCL, and in several cases where ordinary MCL failed, Cluster-MCL succeeded.

Cluster-MCL is designed to eliminate the problems caused by bias in symmetrical environments. In order to demonstrate the improvements we found several environments with symmetrical areas and performed global localization on them. In completely symmetrical environments, the desired behaviour is for multiple locations to be represented when execution is complete, since there is no data available to resolve which of the symmetrical locations is correct. Because there is no available information to distinguish the locations, localization should report the problem, rather than erroneously reporting a single position. It is also common for environments to exhibit symmetry between certain regions with other unique areas. In such situations, finding the correct location



(a) (b)

Figure 3-1: Global localization using ordinary MCL



(a) (b)

Figure 3-2: Global localization using Cluster-MCL.

during global localization requires maintaining multiple hypotheses within the symmetrical areas and then resolving them to the single, correct, location once the robot enters an asymmetrical area. The data sets selected for the experiments demonstrate that Cluster-MCL correctly solves these situations, which are difficult for normal MCL. Our algorithm correctly maintains multiple clusters in symmetrical areas and can still resolve the correct position when unique areas are observed.

3.3.1 Simulated Data

For simulated environments, we generated two highly symmetrical maps to test on. Testing MCL and Cluster-MCL using these maps, we observed that Cluster-MCL correctly maintains all equally probable clusters, while ordinary MCL incorrectly and prematurely identifies a cluster around a single location. In Figure 3-2, we display the results of Cluster-MCL using one of the maps, and we can clearly see that there are multiple distinct clusters. Notice that Cluster-MCL maintains a posterior belief comprised of four distinctive poses, in contrast to conventional MCL, whose outcome is shown in Figure 3-1. Moreover, the clusters in Figure 3-2 are all just about equally probable, as demonstrated by our observation of the constant trading off of which cluster is most probable. We obtained similar results on the second map, which was a simple rectangle. These perfectly

symmetrical maps demonstrate that our algorithm correctly solves the bias problem of section 2.2.4 in symmetrical environments, preventing MCL from localizing to only a single location.

In Figure 3-2, the extra cluster (circled) is a randomly drawn cluster, used to make Cluster-MCL robust to the kidnapped robot problem.

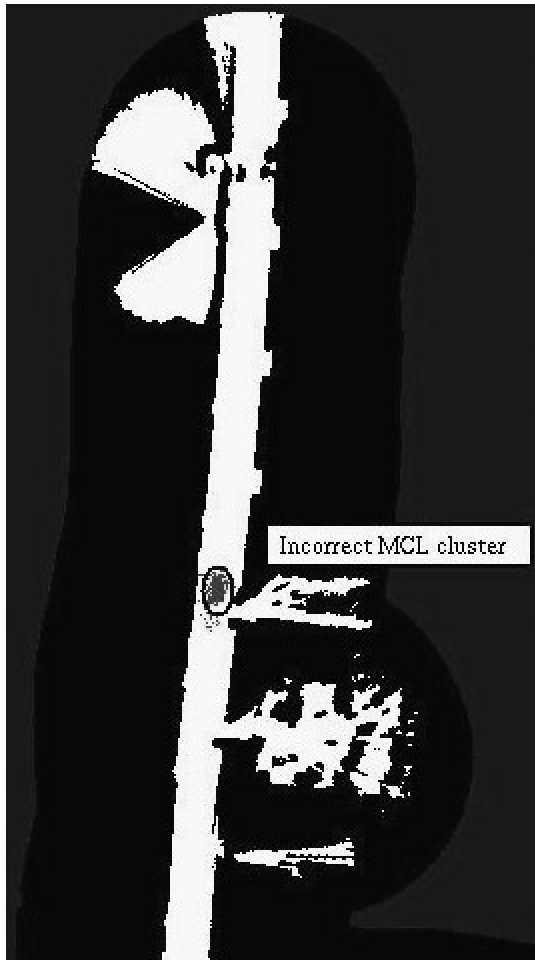
3.3.2 Real Data

To demonstrate the workings of our algorithm in practice, additional tests were performed using data collected from two real world environments. Our first environment consisted of a long corridor in Wean Hall at Carnegie Mellon University, with equally spaced doors and few distinguishing features, thus providing an environment with some symmetry. Our second environment consists of a room in the Gates Building at Stanford University, with two entrances opposite each other, two benches symmetrically placed and a file cabinet in each corner of the room. The datasets in both locations were collected using a robot equipped with a laser range finder.

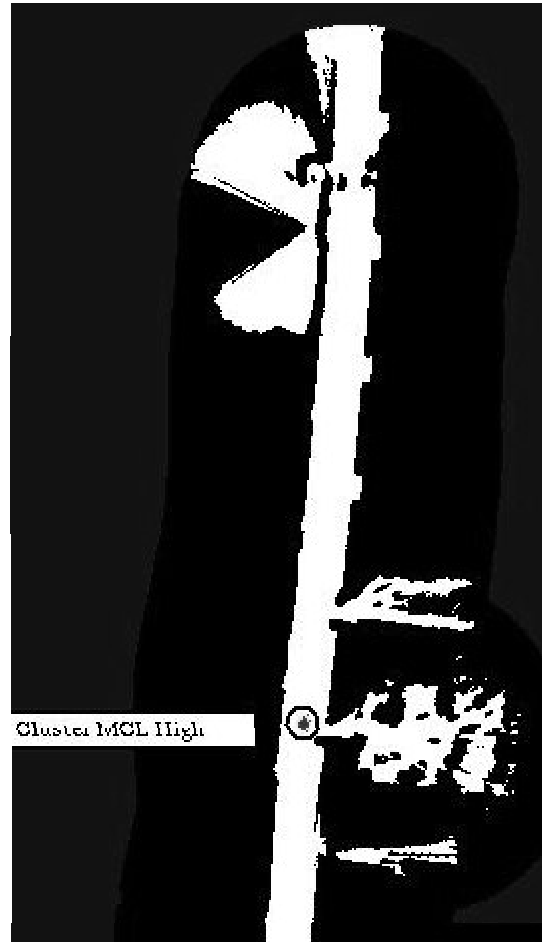
From these environments we collected nine datasets. From Wean Hall, we collected four datasets. In each dataset, the robot was given a different path with different features of the environment observed. Of the four cases, MCL was only able to correctly localize in three of them, while Cluster-MCL correctly identified the robot's position in all cases. In Figure 3-3, a comparison is given between MCL and Cluster-MCL on a particular dataset, number 3, from Wean Hall. On multiple executions over that particular dataset, ordinary MCL failed 100% of the time while Cluster-MCL had a 100% success rate. We show that ordinary MCL converges to the wrong location, while Cluster-MCL correctly identifies the robot's position.

In the Gates Building environment, five datasets on two different maps were collected. In all cases, Cluster-MCL performs at least as well as ordinary MCL. In four of the datasets, MCL and Cluster-MCL both correctly identify the robot's location. However, in the final dataset, MCL failed to consistently identify the correct location of the robot, while Cluster-MCL was able to localize to the correct position. The difference between the Wean Hall and Gates datasets is in the level of symmetry. To demonstrate the benefits of Cluster-MCL, we chose a more highly symmetrical environment in Gates and collected datasets which had two possible localizations until the final segment of them. We ran MCL and Cluster-MCL several times on those datasets and the results show that MCL had 50% accuracy in determining the correct position, while Cluster-MCL had 100% accuracy.

These data sets demonstrate that Cluster-MCL can correctly converge to the single correct location even as it maintains several high probability locations. Although these environments are symmetrical around the initial position of the robot, it eventually travels to a unique area. If the global localization algorithm converges to a single location within the symmetrical area there is a good chance that the location is incorrect. However, Cluster-MCL can maintain multiple hypotheses across the symmetries and eventually apply data from the unique areas to determine the correct cluster. These experiments demonstrate that the algorithm can determine a single correct location even as it maintains multiple clusters to handle symmetry.



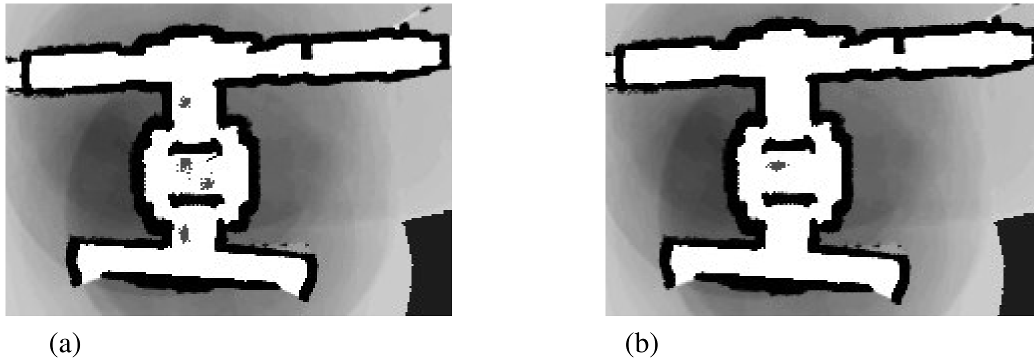
(a)



(b)

Figure 3-3: Results of MCL and Cluster-MCL on Wean Hall dataset 3.

MCL converges to an incorrect cluster in (a), while Cluster-MCL converges to the correct location in (b).



(a) (b)

Figure 3-4: Results of MCL and Cluster-MCL on Gates data.

Cluster-MCL tracks multiple possible clusters in (a) while ordinary MCL converges to a single, incorrect, cluster in (b).

3.4 Conclusion

In this chapter we introduced a cluster-based extension to MCL localization. Ordinary MCL can fail if the map is symmetrical, however, we proposed a method that retains multiple hypotheses for the robot's location, consistent with sensor data. Our method involves clustering the points, and then tracking the clusters independently, so as to avoid discarding other possible locations in favour of the most probable cluster at the current time step. By considering the probability of multiple clusters over a longer time, we are able to get a more accurate estimate of their likelihood. We have shown that this method satisfies the derivation of MCL and that the additional information we take into account allows us to eliminate some of the problems in MCL, as described in section 2.2.4, and better approximate the true posterior. Our experiments show that Cluster-MCL performs at least as well as ordinary MCL on several real datasets, and in cases where MCL fails, Cluster-MCL still finds the correct location. Finally, we have shown that Cluster-MCL maintains all of the correct possible locations in symmetrical environments, while MCL converges to a single, often incorrect, cluster.

Cluster-MCL might also provide benefits for planning in dangerous environments, since the planner could consider a set of high probability locations and avoid any actions that would be dangerous if the true location were any member of that set. With ordinary MCL only a single location exists and the planner might not be aware of possible problems if the location is incorrect. The robot might also be able to decide to search for distinct features in an attempt to resolve the uncertainty between multiple clusters. Allowing a planner to consider multiple possible locations could be beneficial in many situations.

The primary benefit of Cluster-MCL is that it reduces the bias problem in MCL caused by the particle filter of the prior distribution not representing some possibilities. By using multiple, independent clusters, the algorithm continues to consider lower probability locations, instead of relocating all the particles to the most likely region. Although it requires more particles than standard

MCL, Cluster-MCL allows particle filter based localization to be successful in many situations where standard MCL is infeasible, allowing robots to work in more areas. Maintaining the lower probability clusters, while still providing a single high probability location, is a powerful addition to MCL.

Chapter 4
Dynamic Motion Models in MCL

4.1 Introduction

Most of the time, Monte Carlo Localization works properly, finding the correct localization for the robot. However, it is possible to correct various errors in the models to allow MCL to converge to a correct solution even more accurately. Although improvements may be unnecessary when the algorithm is already working, by making corrections, the errors should not build up, and future situations may be easier to solve correctly. Since minor errors in MCL can combine to produce problems, reducing minor errors when they have little impact prevents those same errors from combining with other errors to cause localization failures.

One situation where reducing minor errors is critical is in the case of global localization. In this case, the entire space must be searched and minor errors can easily cause global localization to fail. If some of these minor errors can be removed during ordinary execution, then global localization in the future may be easier.

In this chapter, I demonstrate that it is possible to update the parameters of the motion model during execution of MCL to provide a more accurate idea of how the robot moves through the environment. In general, a single, simplified, motion model is created that reflects some idea about how a robot moves. This model is necessarily a generalization because the robot's motion is affected by various changing situations, such as the surface it moves on, and possibly the power of the batteries or the inflation and wear on the tires. While all of these situations could be monitored and manually compensated for, it would require an enormous amount of work to create a motion model that reflected all of these different states. It is also impossible to predict all possible circumstances, so such a complex model would be invalidated by any unanticipated change in conditions. By automatically updating the motion model according to the observations, it is possible to optimize the model to any situation, even if that situation has not been predicted in advance. As the model is updated, errors in MCL due to the motion model are reduced, leaving greater tolerance for errors caused by other factors.

Once the motion model is updated automatically, it becomes possible to easily use a different model. Ordinarily, a more complex model is impractical because every parameter must be manually adapted to each environment. To simplify the work required to adapt MCL to a new situation, the simplest possible motion model is used with the minimum number of parameters to tune. However, if we can update the parameters dynamically, a more complex model can be used. Since the parameters do not have to be manually tuned, the additional complexity does not require additional user work. Increasing the complexity of the motion model allows it to more accurately represent various conditions which would be inefficient to model manually. It also becomes possible to use a hardware specialized model. Even though some implementations use a complex or specialized manually tuned model, such a model is too difficult to generate in most situations. It is especially useful to increase the number of parameters in the motion model when they are determined separately for each region.

The regional algorithm benefits from being able to more closely adapt the representation to reality in each location.

Similar work on automatically determining motion models has been done by [Eliazar and Parr 2004; Kaboli et al. 2006], however, these techniques are not developed as real time dynamic algorithms. They also do not consider determining separate parameters for different regions. [Kaboli et al. 2006] use training data from a robot to calculate better parameters for MCL in an offline algorithm. [Eliazar and Parr 2004] apply an Expectation Maximization algorithm to data from FastSLAM in order to update the model. Although they consider using it during the execution of FastSLAM, their suggested technique is to detect changes in conditions and then use the EM algorithm on a small set of data, causing a temporary reduction in the performance of FastSLAM. My own algorithm, although it may not produce as accurate results as these algorithms, is unique in providing a real-time solution which can also be used to determine separate parameters for various regions.

4.2 Dynamic Motion

The MCL algorithm, as described in section 2.2, depends on certain static parameters that must be manually tuned for each implementation. In particular, the sensor model relies on a static map of the environment, while the motion model, $p(x_t \mid x_{t-1}, u_t)$, requires parameters that reflect the specific robot's motion in the particular environment. Since most interesting problems occur in dynamic environments, or environments with different conditions in different areas, these static parameters are only a broad approximation. Fortunately, MCL is robust to errors in the map and motion model and will successfully localize a robot as long as these parameters are a reasonable representation. However, the more error there is in the static parameters, the less tolerance the algorithm has for errors from other sources. For example, if the environment changes so that the map becomes less accurate, perhaps because of furniture being moved, then an error in the motion model might put the robot in the wrong location. If the changes in the map make an incorrect location look correct to MCL, then there is far less tolerance for the motion model to predict incorrect locations. Either of these errors might be recoverable on their own, but both together could cause a localization failure. If the motion model is correct, then the robot's next location will be predicted correctly, and the fact that there is a similar location somewhere else won't matter. Similarly, if the map is accurate, then an incorrect prediction from the motion model will be low probability and will die out in favour of the correct location. Over time, the errors caused by each static parameter add to the overall error, reducing MCL's tolerance towards additional errors is reduced until it becomes necessary to manually correct the parameters.

In the following subsections I give a description of the error in the motion model and define the necessary parameters for representing it. I also describe some additional motion models that are effective with the dynamic motion model algorithm. Finally, I explain how the MCL algorithm can be updated to implement dynamic motion models. The last subsection details the methods for verifying the optimization algorithm being used.

4.2.1 Motion Model Error

On each step of execution, MCL uses the motion model to predict a new location for the robot, and then uses the sensor model to correct that location. Before the resampling step, the mean of the particles represents the location determined by the motion model. After resampling, the mean represents the location of the robot according to the algorithm. This means that a side effect of executing MCL is a list of errors in the motion model. By recording these values, we can dynamically generate a set of errors that can be processed to correct the model. Since each correction comes attached to a particular location, we can even record in what part of the environment the error occurred.

Given a set of errors, it would be quite easy to determine the variance of a Gaussian distribution. However, with the Gaussian motion model we are using it is not quite so simple. The key realization is that we are not trying to calculate the variance, we are trying to find a parameter of the variance. Remember that the motion model for a differential drive robot depends on three parameters, range error, turn error, and drift error, represented as (k_r, k_θ, k_d). If we let r be the distance travelled and θ be the distance turned, while $\underline{r}$ and $\underline{\theta}$ are the estimations of these values returned by the motion model, then the distributions become:

$$r = N(\underline{r}, k_r \times \underline{r}), \quad \theta = N(\underline{\theta}, k_\theta \times \underline{\theta} + k_d \times \underline{r}) \tag{4-1}$$

which are both single valued Gaussians. During the execution of MCL proposed values $\{\underline{r}, \underline{\theta}\}$ are produced by the motion model. Then, the update and resampling steps produces a corrected location from which we determine the actual motion $\{r, \theta\}$. Thus, from MCL we are given a set of $\{r, \theta, \underline{r}, \underline{\theta}\}$ values and we wish to optimise the models in the parameters $\{k_r, k_\theta, k_d\}$.

4.2.2 Variance Parameters

Because we wish to determine parameters to the variance, instead of the variance itself, no standard technique for estimating Normal distributions will work. We cannot simply determine the mean and variance of the set of error values and use a single Gaussian distribution with those parameters. In fact, the problem is no longer a single distribution, but rather a set of Normal distributions, one for each value of ($\underline{r}$, $\underline{\theta}$). Fortunately, the problem can be solved if we treat it as a general equation, instead of specifically as a probabilistic distribution. The Gaussian equation for r is:

$$p(r) = \frac{1}{k_r \underline{r}\sqrt{2\pi}} e^{-(r-\underline{r})^2 /(2(k_r \underline{r})^2)} \tag{4-2}$$

Since we want to have an accurate model, we want the value of k_r that maximizes the probability. Given the set of data produced by MCL, we would like to maximize the probability obtained over that entire sample space.

$$\prod_{(r,\underline{r})} \frac{1}{k_r \underline{r}\sqrt{2\pi}} e^{-(r-\underline{r})^2 /(2(k_r \underline{r})^2)} \tag{4-3}$$

Of course, (**4-3**) is a little unwieldy to calculate, but a standard trick is to notice that if we maximize log($p(r)$) we also maximize $p(r)$. Thus we are left with:

$$\sum_{(r,\underline{r})} -\log(k_r \underline{r}\sqrt{2\pi}) - \frac{(r-\underline{r})^2}{2(k_r \underline{r})^2} \qquad \textbf{4-4}$$

which is quite straightforward to maximize. A similar process for θ gives us a slightly more complicated equation which is just as easy to solve.

$$\sum_{(r,\theta,\underline{r},\underline{\theta})} -\log((k_\theta \underline{\theta} + k_d \underline{r})\sqrt{2\pi}) - \frac{(\theta-\underline{\theta})^2}{2(k_\theta \underline{\theta} + k_d \underline{r})^2} \qquad \textbf{4-5}$$

Using an efficient nonlinear optimisation algorithm, we can maximize these equations over the parameters k_r, k_θ, k_d for sets of data obtained by MCL in real time. Although the functions are not concave in these parameters, we have good starting parameters available, since MCL is already using a motion model. The current parameters make a good starting point for the optimization. The new parameters can be used immediately, while the data is still collected to further refine them.

4.2.3 Expanded Motion Model

Since the motion model will be optimized automatically, certain simplifications that were used to make finding the parameters easier are no longer necessary. In particular, the assumption that the mean of the error in motion is zero can be discarded. Although the robot's odometry is designed so that the mean of the motion is the reported motion, in any particular situation there may be a difference. Certain conditions affect the mean as much as they do the variance. For example, different surfaces will cause the wheels to slip so that the robot moves only a percentage of its reported motion. Similarly, traveling downhill might cause the robot to travel further than the odometry. Although the odometry still reports mean values without systematic error when considering the entire history of the robot, in any subset of the robot's motion there may be some systematic error. Expanding the motion model to represent a consistent error results in two new parameters, l_r, representing the percentage of the distance traveled and l_θ, representing the percentage of the angle turned. Revising the Gaussian distributions with these parameters gives us:

$$r = N(l_r \times \underline{r}, k_r \times \underline{r}), \quad \theta = N(l_\theta \times \underline{\theta}, k_\theta \times \underline{\theta} + k_d \times \underline{r}) \qquad \textbf{4-6}$$

Of course, there are now two parameters to optimize for the range distribution, and three for the angular. However, although added parameters make the derivation more difficult, the actual computation during MCL doesn't change. Using the same procedure as for the original motion model, we want to maximize the probability $p(r)$ over the error samples.

$$p(r) = \frac{1}{k_r \underline{r}\sqrt{2\pi}} e^{-(r-l_r \underline{r})^2/(2(k_r \underline{r})^2)} \qquad \textbf{4-7}$$

In order to perform the maximization we take the product over the data set. As usual, the maximization is performed over the logarithm in order to simplify the computations.

$$\prod_{(r,\underline{r})} \frac{1}{k_r \underline{r}\sqrt{2\pi}} e^{-(r-l_r \underline{r})^2/(2(k_r \underline{r})^2)} \qquad \textbf{4-8}$$

$$\sum_{(r,\underline{r})} -\log(k_r \underline{r}\sqrt{2\pi}) - \frac{(r-l_r \underline{r})^2}{2(k_r \underline{r})^2} \qquad \textbf{4-9}$$

The equations for the angular probability are similar, resulting in a maximization function little changed from the previous motion model.

$$\sum_{(r,\theta,\underline{r},\underline{\theta})} -\log((k_\theta \underline{\theta} + k_d \underline{r})\sqrt{2\pi}) - \frac{(\theta - l_\theta \underline{\theta})^2}{2(k_\theta \underline{\theta} + k_d \underline{r})^2} \qquad \textbf{4-10}$$

These equations are almost identical to those resulting from the original motion model, except for a single added parameter. However, each distribution requires maximization over an additional parameter, bringing the angular error to three parameters and the overall motion model to five. With this increased number of parameters to the optimization, it may be useful to use an optimization technique which takes into account the gradient and possibly even the Hessian of the optimization function. Of course, there is no reason why such a technique could not have been used for the original problem as well. Fortunately, these derivatives are easy to calculate in symbolic form and the resulting equations are also easy to instantiate for any particular values. Of course, the size of the Hessian matrix increases with the square of the number of terms. However, because of the duplication and the independence between range and angle, it is still manageable in this case.

For the range distribution, there are two first order derivatives and three second order ones. Since we are maximizing equation (**4-9**), that is the one we differentiate.

$$\frac{\partial}{\partial k_r} = \sum_{(r,\underline{r})} -\frac{1}{k_r} + \frac{(r - l_r \underline{r})^2}{\underline{r}^2 k_r^3} \qquad \textbf{4-11}$$

$$\frac{\partial}{\partial l_r} = \sum_{(r,\underline{r})} \frac{r}{\underline{r} k_r^2} - \frac{l_r}{k_r^2} \qquad \textbf{4-12}$$

$$\frac{\partial^2}{\partial k_r^2} = \sum_{(r,\underline{r})} \frac{1}{k_r^2} - \frac{3(r-l_r \underline{r})^2}{\underline{r}^2 k_r^4} \qquad \textbf{4-13}$$

$$\frac{\partial^2}{\partial l_r^2} = \sum_{(r,\underline{r})} -\frac{1}{k_r^2} \qquad \textbf{4-14}$$

$$\frac{\partial^2}{\partial k_r \partial l_r} = \sum_{(r,\underline{r})} \frac{-2(r - l_r \underline{r})}{k_r^{\,3} \underline{r}} \qquad \textbf{4-15}$$

The derivatives for the angular distribution are derived in the same way, except there are three first derivatives and six second derivatives. All the derivatives for both distributions are simple to calculate numerically for a set of error values and allow any gradient or Hessian based nonlinear optimization technique to be used.

4.2.4 Specialized Motion Model

So far, we have been using motion models that apply to virtually any near-holonomic ground based robot. General motion models are common because they can be adapted easily to any type of robot. Changing the physical device only involves altering the parameters of the model. Since a static model usually has general parameters there is minimal work involved in adapting it to different hardware. However, using a more specific motion model that represents the specific platform involved requires completely redefining the model for each new robot. A new static motion model also needs extensive experimentation in order to determine the parameters, since there is unlikely to be data on the parameters used for a similar robot. With a general model the parameters used for other hardware can suggest approximate values for a new system. However, if each model is specific to the underlying robot then there are no similar implementations to check. Each instance requires independent experimentation to generate working parameters for the specialized motion model. Because of this added complexity, more general models are often used which can represent entire classes of robots. Figure 4-1 shows the geometry of the motion model based on the physical configuration of the robot. s_L and s_R represent the displacements of the left and right wheels. Given the robot wheelbase b the two displacements produce the x and θ values necessary to calculate the robot's motion.

However, a motion model which reflects the physical construction of the robot may offer some benefits which, even if they do not outweigh the ease of using a general model, are still useful. By representing the hardware directly we can create a specialized motion model which uses fewer parameters to provide the same abilities as a more complex, general model. Our expanded model uses five parameters, $\{k_r, k_\theta, k_d, l_r, l_\theta\}$ to represent our two wheeled, differential drive, near-

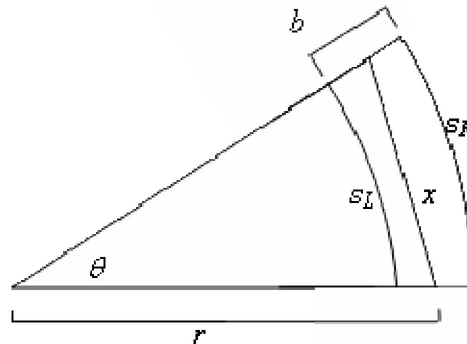


Figure 4-1: The geometry of the specialized motion model.

holonomic robot. However, with only two independent, motorized wheels a specialized model represents the same information with fewer parameters. Each wheel can be modelled with a parameter on variance and another on the mean, resulting in four parameters overall. Labelling these parameters with L and R for the left and right wheels we have $\{k_R, k_L, l_R, l_L\}$. A basic model which assumes no systematic error would only require two parameters, $\{k_R, k_L\}$. Examining Figure 4-1 we see that if we have the wheel displacement of the left and right wheels (s_L and s_R) we can determine the relative angle of motion θ, the displacement x, and the radius of the sector r using the equations:

$$s_L = (r - \frac{b}{2})\theta \qquad s_R = (r + \frac{b}{2})\theta \qquad r = \frac{x}{2\sin\frac{\theta}{2}} \qquad \textbf{4-16}$$

Where b is the size of the robot's wheelbase. Rearranging these equations to solve for x, r and θ we get:

$$\theta = \frac{s_R - s_L}{b} \qquad r = \frac{b \cdot s_L}{s_R - s_L} + \frac{b}{2} \qquad x = 2(\sin\frac{s_R - s_L}{2b})(\frac{b \cdot s_L}{s_R - s_L} + \frac{b}{2}) \qquad \textbf{4-17}$$

With these equations we can apply a motion model to the wheel displacements s_R and s_L and from the results obtain the actual motion of the robot in Cartesian space. The motion model is defined by two independent Gaussian distributions of the same form. We do not have the same problem as with the general model where the variance of one variable depends on the estimated value for the other. In the models described previously, the error in the angular distance moved depends on the value for range, but in this model, the two wheels are completely independent.

$$s_R = N(l_R\hat{s}_R, k_R\hat{s}_R) \qquad s_L = N(l_L\hat{s}_L, k_L\hat{s}_L) \qquad \textbf{4-18}$$

With these equations, the specialized motion model that more accurately reflects the characteristics of the robot can be used in the dynamic motion model algorithm without causing any important changes. The underlying MCL and dynamic motion systems do not change in any way. When the robot moves, the s_R and s_L values are received from the wheels and are sampled from the motion model as defined in equation (**4-18**). Then the relative angle change and displacement are calculated using (**4-17**). Finally, the MCL update step is performed and the corrected s_R and s_L values are determined from the robot position using (**4-16**). The behaviour of MCL is unchanged and the dynamic motion algorithm merely has to optimize the proper equations. Also, no exhaustive experimentation is necessary to determine the parameters for the new model. Instead, high variance parameters can be selected which work for long enough that the dynamic motion model algorithm can correct them. Eventually, the model will be optimized further than any amount of testing for static parameters could accomplish, since it will be able to take into account transient conditions that static parameters could not represent. For any particular physical robot, once a specialized motion model is defined, the dynamic algorithm can optimize the parameters, allowing the same algorithm to be used for multiple, different, highly specific implementations.

4.2.5 Updated MCL

Once we have new motion model parameters it is necessary to use them in MCL. However, if dynamically modifying the parameters violates the definition of MCL then it will be necessary to rederive it in order to prove that it is possible to use the parameters. Remember that MCL was defined in equation (**2-12**) as:

$$Bel(x_t) = \eta p(z_t \mid x_t) \int_{x_{t-1}} p(x_t \mid x_{t-1}, u_t) p(x_{t-1} \mid z^{t-1}, u^{t-1}) d_{x_{t-1}} \quad \textbf{4-19}$$

Altering the parameters or even redefining the motion model affects only the predictive part of the equation, which is $p(x_t \mid x_{t-1}, u_t)$. Technically, using past data to update the motion model violates the Markovian assumption, but since the update does not occur inside equation (**4-19**) we continue to assume it. Notice that any change to the motion model, even replacing it with entirely new equations, or replacing it with an infinite set of equations that change over time according to the results of MCL itself, does not alter the formal definition. As long as the resulting location of a motion is determined by the motion u_t and the prior location x_{t-1}, equation (**4-19**) is unaffected. MCL only needs to ask the motion model for a possible location, given a starting location and odometry, the actual mechanism by which the model fulfils that request is unimportant. Thus, despite all the underlying functionality determining the parameters of the motion model dynamically, the derivation of MCL, including the motion model, remains constant. It is only when MCL is implemented and the model must actually be calculated that the differences occur. Since none of the parts of the MCL equation are changed by dynamic motion models, the theoretical foundation of MCL remains unchanged and all of its features apply equally to the dynamic algorithm.

4.2.6 Algorithm

Now that we have a method to update the motion model dynamically, we need to integrate it with MCL, hopefully without significantly affecting the runtime. One of the benefits of MCL is that it is a fairly low cost algorithm, computationally, and it is important that we do not make changes that significantly increase the amount of time it takes to run. Since MCL must run in real time, whatever processing is necessary to update the motion model must not delay localization. With these requirements in mind, our dynamic motion model MCL algorithm provides a minor alteration that allows the parameters of the motion model to be recalculated and used.

At each MCL update step, a $\{r, \theta, \underline{r}, \underline{\theta}\}$ data point is recorded. Because the estimated points $\underline{r}$ and $\underline{\theta}$ are determined exclusively by the motion model, the only source of error between them and the true values must be caused by the motion model. The true values r and θ are calculated by the MCL algorithm and, as long as the robot remains localized, they must be correct. Thus, the error values used are caused only by the motion model and not by any other possible source of error. When enough new data points are recorded to make it worthwhile to calculate new parameters, the equation is maximized in the background, using whatever processing power is available when localization is finished. The actual number of data points needed for a successful update varies based on the complexity of the model and the optimization function used, but in practice MCL updates occur so frequently that there is no problem collecting the requisite number. For the experiments described here, between 50 and 100 data points were sufficient to produce good values. When the

maximization is complete, the new parameters are reported to the MCL algorithm. In fact, MCL itself is unaware of the changing parameters, since it just runs normally.

The logarithms of the equations for the systems of Gaussians, as defined in sections 4.2.2 and 4.2.3, are simply ordinary nonlinear functions that are maximized by finding the parameters which result in the greatest values. Of course, it would be impossible to try to find these parameters by trying every possible combination of values, especially since they are real valued. However, nonlinear optimization is a well studied area and many robust techniques exist for performing this operation. Any relatively efficient nonlinear optimizer should be effective and, although these algorithms can be complex, they are so commonly used that often an existing implementation can be found. The major problem with nonlinear optimization is caused by local minima in the function, which can sometimes cause suboptimal values to be returned. However, the best solution to this problem is to give the optimization algorithm initial values which are relatively close to the optimal ones. In the case of the motion model, the existing parameters have already been effective in localization and are thus probably fairly close to the optimal ones. Thus, by using these parameters as a starting point, most nonlinear optimizers should solve the maximization problem very quickly. The technique we used was the simplex search method of [Lagarias et al. 1998], as implemented in the Matlab program, but any similar algorithm should succeed.

In order to reduce the complexity of the calculation, only the most recent set of errors are used. When a predetermined number of corrections are recorded, each subsequent observation causes the oldest observation to be removed. This creates an upper bound for the maximization routine and also allows the dynamic model to update to changing conditions. For example, if the robot's tires deflate, or water is spilled on the floor, the motion of the robot would change. In that case, after a certain number of updates, all of the old data would be removed and the model would be calculated entirely based on the changed conditions.

In order to accommodate different conditions in different areas, data points are not stored globally but are instead recorded by region. Each region of the map has its own collection of data. If there are insufficient points to calculate the parameters, then the previous parameters are used. However, once the robot traverses an area enough that it can update the motion model, it calculates the parameters and stores them with the area. When it subsequently enters the same region, it can load the specific parameters. Any reasonable algorithm for defining regions can be used, smaller regions will be more accurate but will take longer to receive enough data, while larger regions will update sooner, but may represent multiple conditions. We used a predefined set of regions determined by the peaks in a potential field over the map. These regions tend to cover specific rooms or corridors where there is little change in the motion of the robot. Although this is not guaranteed, it has been the case in the environments used for the experiments. Since the motion data is recorded by region, the regions are kept distinct, with the robot traveling in only a single region at any time. If the regional algorithm fails to adequately determine regions for a particular environment, a more accurate detection of regions should be used, or the dynamic motion model algorithm limited to the global implementation.

The results of this dynamic motion model algorithm are a map annotated with the motion model parameters for different regions. Aside from changing the motion model during execution, the map can also be used to provide additional data for planning or analysis. For example, if a region causes a

high variance, then it might be better for the robot to avoid that region when path planning. Also, a significant change in variance might indicate some kind of anomaly that should be dealt with. A robot might also use the different parameters to identify different surfaces in the environment for another machine, perhaps planning a route that avoids certain kinds of surface. If the expanded motion model is used, the additional parameters on the mean can be even more informative. The l_r value indicates how efficiently the robot can move on the surface while the l_θ parameter gives a good indication of the robot's manoeuvrability.

Although creating dynamic motion models uses successful localization to correct errors in the model, it does not preclude using the same data to correct other errors. Since the MCL algorithm compensates for many different sources of error to provide accurate results, reducing those errors does not invalidate other functions which depend on correct localization. The various errors do not make localization less accurate. Instead, they build up until localization fails completely. If localization became less accurate, that would mean that the robot's most probable location was being reported by MCL as offset from the correct position. However, when this happens, particles will collect around the incorrect location. Eventually, the bias problem from section 2.2.4 means that the correct location will be unrepresented. At that point, the same error that identified the incorrect location would continue to offset the prediction, causing it to move at random. Thus, if MCL is working, it provides an accurate location, regardless of the specific errors in its various models. Correcting some of these errors allows MCL to be more tolerant of other sources of error. In particular, it is possible to dynamically update the map of the environment as in Chapter 5, while simultaneously dynamically updating the motion model. There is no reason why other parameters could not also be dynamically updated at the same time. Another problem is to determine whether MCL is actually working. This can be determined easily by observation if the robot's actual path is known, but for autonomous execution a simple heuristic can provide an estimate. One technique, as described in [Thrun et al. 2005] (Section 8.3.5), uses the sum of the individual particle weights. If these weights fall below some threshold then localization is considered to have failed.

4.2.7 Nonlinear Optimization

The primary calculations of the dynamic motion model algorithm are performed by the optimization step where the new parameters are determined. Although nonlinear optimization is usually a straightforward operation, there can sometimes be problems in its execution. These occur primarily when there are many local minima in the function being optimized. As described in Section 4.2.6, we optimize using a simplex search algorithm [Lagarias et al. 1998] implemented by the Matlab `fminsearch` algorithm. However, this algorithm updates the parameters to be closer to the nearest local minimum to the starting point and does not use any techniques for finding the overall global minimum. If the dynamic motion model equations have local minima that are different from the global minimum, then simplex search may not find the optimal values for the parameters. In order to avoid the problem, we need to use starting values which have the global minimum as the closest local minimum. If that is not the case, then some form of optimization which searches beyond local minima is necessary. These algorithms, however, are more complex and time consuming than basic local search.

The primary solution to local minima is to choose a starting value that is very close to the optimal value. Unfortunately, the definition of close depends on the function being optimized. Also, it is sometimes nontrivial to estimate good starting values that will result in the optimal parameters. Fortunately, in the case of dynamic motion models a good estimate is available without requiring any additional computation. The prior motion model parameters have been effective in tracking the robot. Although they are probably not optimal, they should be a good approximation to the optimal parameters. Local search should be able to quickly converge when the prior values are an effective approximation of the optimal values. However, without knowing the actual function that relates the parameters to the error we cannot say for certain that there are no local minima in any interval, regardless of how small it is. Without knowing the characteristics of the error function there is no way to determine the maximum allowable difference between the starting and optimal parameters. Further, although the priors provide good starting parameters, there is no clear method for improving the estimate if it turns out to be insufficiently accurate. Although there is no way of knowing the error function in a closed form, we used the data from a run of the MCL algorithm to plot a surface of the range error parameters versus the error in the estimate. In the experimental evaluation section (4.3) Figure 4-8 shows the entire surface over the possible parameter values while Figure 4-9 shows just the area around the global minimum. From these graphs we can see that, at least in this case, the error versus parameters function is relatively smooth and there are no local minima to disrupt the optimization function. Figure 4-8 also shows that the function is relatively flat for much of the k_r dimension and it is possible that some sets of data could cause a variation here that would create a local minima, but on average these should be smoothed out over time, given the overall shape of the surface.

In order to ensure that temporary conditions during the run of the dynamic motion model algorithm are not causing local minima which are not apparent in Figure 4-8 and Figure 4-9, we tried the algorithm with another optimization technique that is less affected by any local minima. Instead of using the prior parameters as initial values we performed the optimization multiple times with random initial values. The resulting parameters with the lowest error were used in the dynamic motion model algorithm. Of course, randomly restarting the optimization requires significantly more processing, resulting in less frequent updates, but it should eliminate errors caused by local minima. These results, shown by experiments J and K in Table 4-1 (page 62), indicate that the loss in processing time introduces more error than any gains based on local minima. Figure 4-8 seems to indicate that there are no such local minima in the functions, while these experiments indicate that there are no practical effects. Even if temporary local minima are being introduced by the reduced data sets being used for optimization, there is no evidence that these have any effect on the accuracy of the dynamic motion model algorithm. The basic simplex search technique using the prior parameters as the starting value should be sufficient to optimize the motion model. This is fortunate because using multiple random starting values for nonlinear optimization requires significant additional processing. The number of restarts required to find the local minimum increases exponentially with each additional parameter, so that only the simplest motion models can be used for this technique. As the number of parameters increases it becomes impossible to use the slower method, so it is fortunate that our experiments indicate that only a single search is necessary.

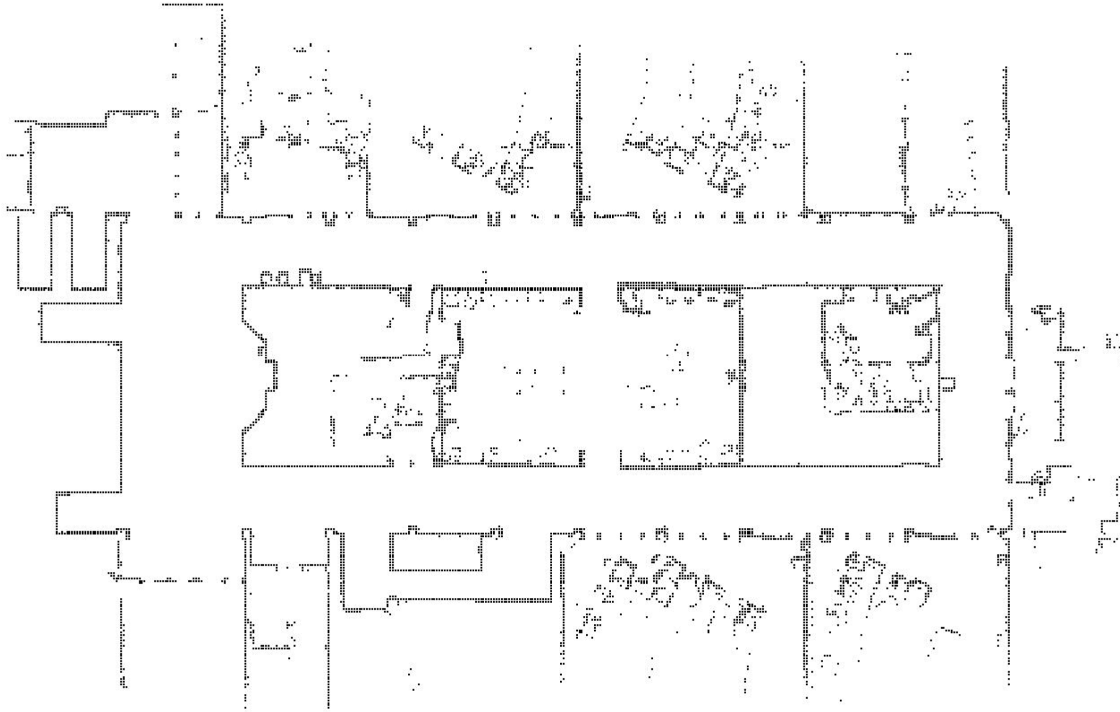


Figure 4-2: First environment used for experiments.

The occupancy grid map of one of the environments used for the experiments. The environment had concrete floors with no carpeting throughout.

4.3 Experimental Evaluation

The dynamic motion model algorithm was tested using a 2 wheeled Pioneer 3-DX differential drive near-holonomic robot equipped with a 180 degree SICK laser rangefinder. Data gathered by the robot over a traversal of the environment shown in Figure 4-2 was processed by both the normal MCL algorithm and various implementations of the dynamic motion model MCL algorithm. Since the environment is an actual building, various sources of localization error such as moving people, altered furniture, and opened or closed doors occur throughout. The parameters of the motion model were calculated by maximizing the equations as described using Matlab's fminsearch function. The results of the experiments, as given by the average percent errors in the robot's motion, show a marked improvement using dynamic motion models.

The primary question is whether dynamically updating the motion model parameters as described in Section 4.2 provides a benefit to the accuracy of the prediction phase of MCL. The experiments were selected to demonstrate the difference in error between the standard implementation of MCL and various dynamic techniques. By demonstrating a decrease in the error of the prediction phase using dynamic parameters, I show the benefit of my algorithm in various implementations.

The original MCL algorithm as described in section 2.2, with general parameters for the basic motion model, was used as a baseline for the tests, with results as described in experiment A (see Table 4-1). A modification of the standard technique with parameters optimized offline was performed in experiment B. The subsequent experiments C through F used the dynamic techniques, with experiments C and D using the basic motion model as described in Section 4.2.1. Experiments E and F used the expanded model described in Section 4.2.3. Finally, the motion model parameters were updated globally for experiments C and E and were updated separately for each region of the map in experiments D and F.

In addition to the basic experiments some additional data was gathered to test other techniques. Experiments J and K test an optimization technique involving restarting the simplex search at multiple random points. This was attempted using both the global (J) and regional (K) algorithms. Also, a motion model based on the particular hardware in the robot was tested in experiments G, H, and I. G shows the results of static parameters while H and I are the global and regional algorithms, respectively.

4.3.1 Static Motion Models

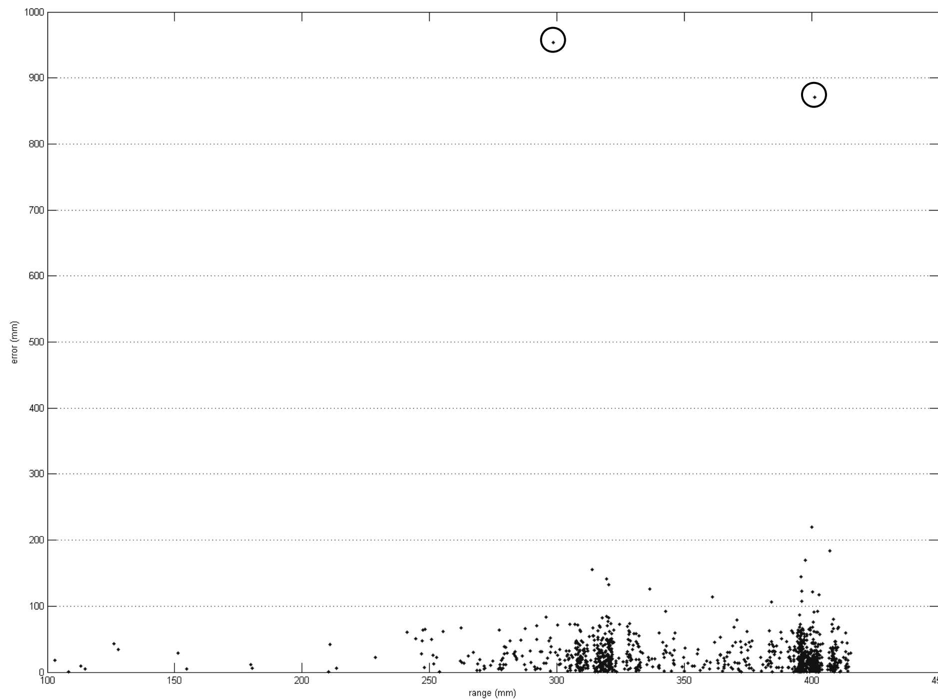


Figure 4-3: Graph of distance travelled vs. range error for standard technique (Experiment A).

Graph of the range error compared to the distance travelled for ordinary MCL. Note the two extreme outliers which are particularly hazardous for localization.

At first, the original, standard MCL algorithm was used with some default parameters for this class of robot (Experiment A). The default parameters and the standard, fixed motion model are the ordinary implementation of MCL. Of course, the parameters are specified for the particular class of robot, but they have not been adapted to the environment or the details of the individual robot. Although these parameters work, they are general, high variance parameters that give fair results for any similar device. With these parameters the average error was 1.7% for range and 4.5% for angle. Figure 4-3 shows the error in range versus distance moved for standard MCL. As well as the average error, we can see some outliers that are particularly dangerous for localization. A single motion with very high error is more likely to mislocalize the robot than several motions with more moderate error, since the location is corrected at each step. Of course, any errors may combine with errors from other sources to cause a localization failure. Because it is impossible to separate the angular error caused by turning from the angular error caused by range, any graph of angular error is not useful. The purpose of this experiment was to provide a baseline for comparison with the dynamic algorithms. Obviously, dynamic motion models must show some improvement over standard MCL to be viable.

Next, the dynamic motion model algorithm was used to calculate parameters based on the entire

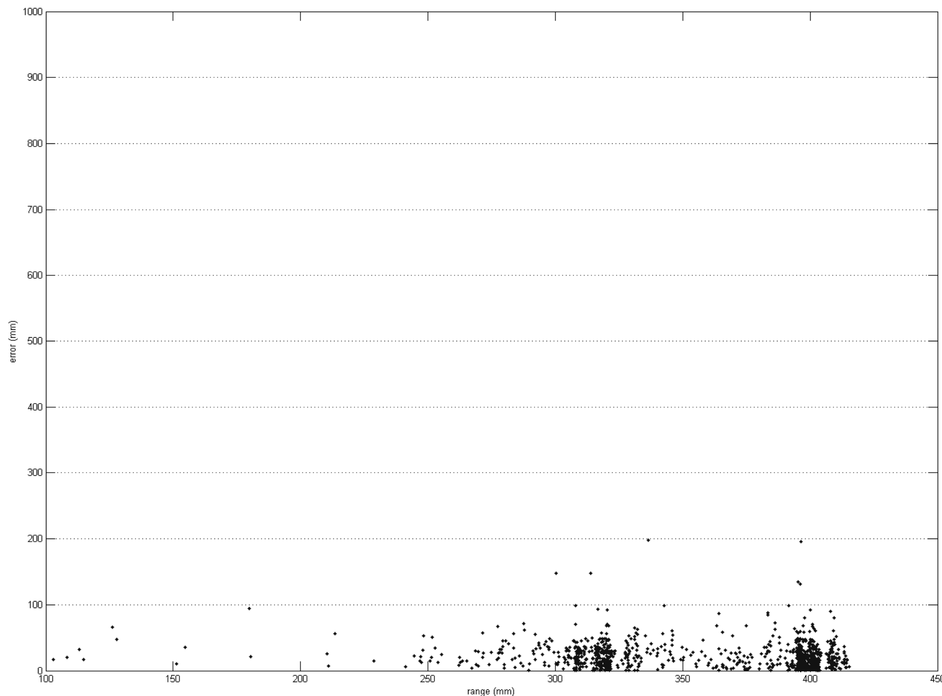


Figure 4-4: Graph of distance traveled vs. range error for optimal static technique (Experiment B).

Graph of range error for the default technique using the optimal parameters. Although some outliers have been removed, there is not much other improvement.

data set and MCL was run with these motion model parameters (Experiment B). This method is called optimal because it uses the best static parameters that can be calculated. The resulting error was 1.4% for range and 2.9% for angle. Of course, in practice, this method is impossible, because it involves knowing the observations that will be made before they are actually recorded. This method can be approximated by using a previous data set on the same environment to calculate the parameters, but it can never truly be implemented in real time. Figure 4-4 shows these motion parameters in action. Experiment B demonstrates the behaviour of the optimal parameters for a single, static motion model. Dynamically modifying the parameters needs to show an improvement over this static technique in order to be a useful solution. Otherwise, it would be simpler to determine the optimal parameters offline and never modify them. An improvement over the results of this experiment proves that dynamic improvement of the parameters provides a benefit that cannot be reproduced with standard MCL. Such an improvement would be based on compensating for various transient situations individually.

4.3.2 Standard Motion Model

The third test (Experiment C) involved dynamic motion models with global data. The parameters were updated during execution according to the preceding localization corrections. Motion error data

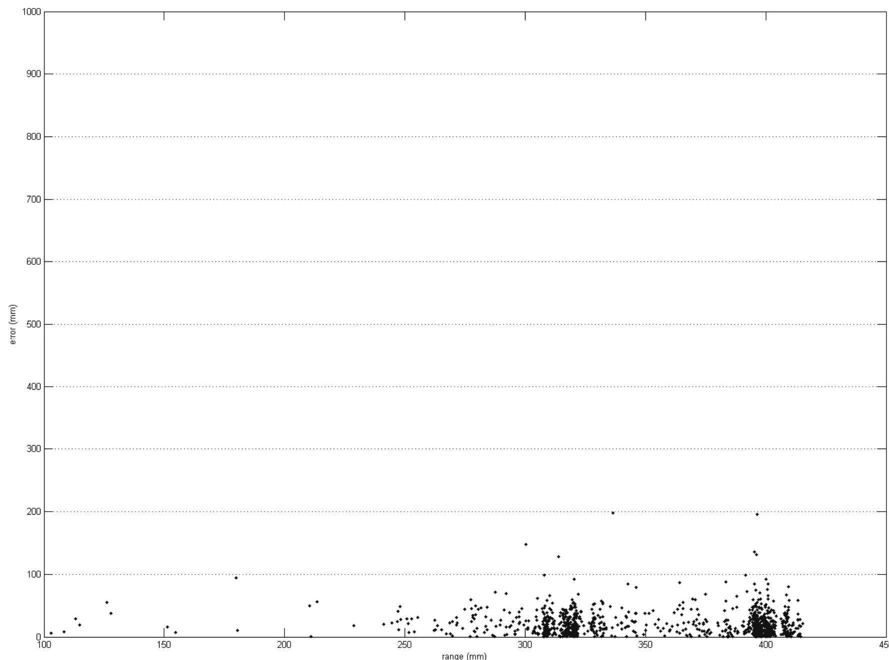


Figure 4-5: Graph of distance traveled vs. range error for global dynamic technique (Experiment C).

Graph of range error for the global dynamic technique. The distribution becomes more compressed as the error decreases.

was collected globally, not separated by region, for this test. With this method, 1.2% range error and 2.6% angle error were recorded with characteristics as shown in Figure 4-5. The improvement over experiments A and B demonstrates that a dynamic motion model provides a benefit over the standard technique, regardless of how good the static parameters are. The ability to compensate for changing conditions is a successful addition to MCL.

The full dynamic motion model algorithm was used next (Experiment D). Each region of the map was updated with its own data and produced its own corrections. This technique produced an error of 1.4% for range and 2.8% for angle with characteristics as shown in Figure 4-6. Although this experiment still provided an improvement over standard MCL, it did not improve over the results of altering the parameters globally. The basic motion model was unable to represent regional conditions well enough to compensate for the slower rate of updates. In the next section we can see that a more complex model allows the regional algorithm to provide an improvement, even in an environment without significant regional differences.

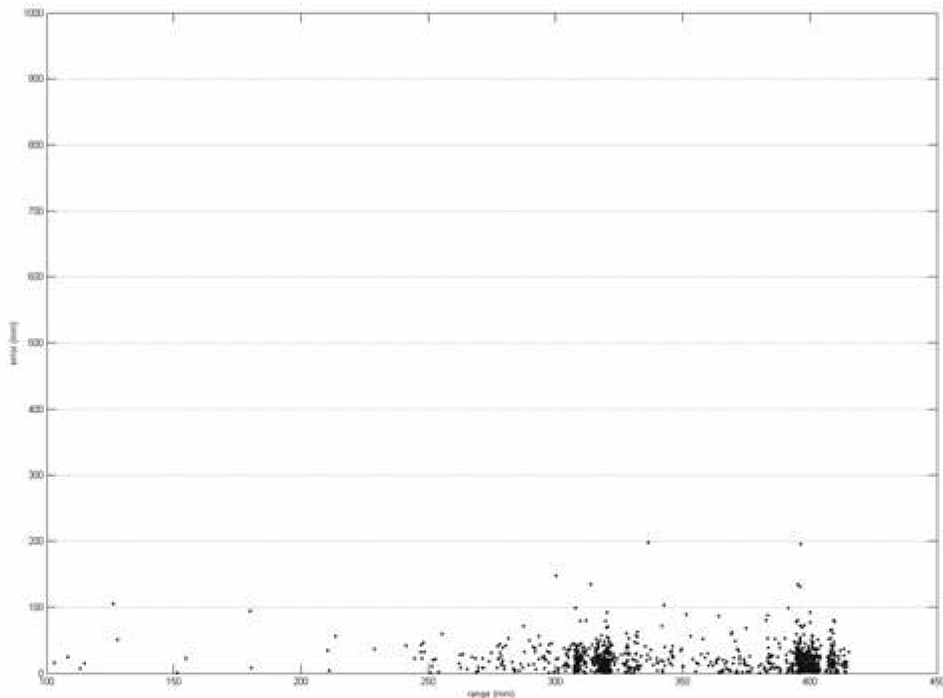


Figure 4-6: Graph of distance traveled vs. range error for regional dynamic technique (Experiment D).

Graph of range error for the regional dynamic technique. Again a small improvement is apparent compared to previous techniques.

4.3.3 Expanded Motion Model

All of these techniques were performed with the basic MCL motion model. The final tests were made with the expanded model that included parameters on the mean. These tests were only performed with the dynamic algorithms, since the static techniques would use parameters of 1.0, resulting in no change. Interestingly, the global dynamic technique (Experiment E) performed worse with the expanded model using regions (Experiment F) with a range error of 1.4% and an angular error of 4.0%. The reason for this is that the additional parameters in the model allow the algorithm to adapt better to highly local conditions, for example dirt or polish on the floor, or even the amount of wear caused by variable amounts of traffic. Globally, these parameters will almost always be 1.0. However, if the model is being updated according to the preceding error samples, the mean parameters will never reflect the current area, instead adapting the model to the previous regions. Although the variance might remain constant globally, the systematic error is highly dependent on

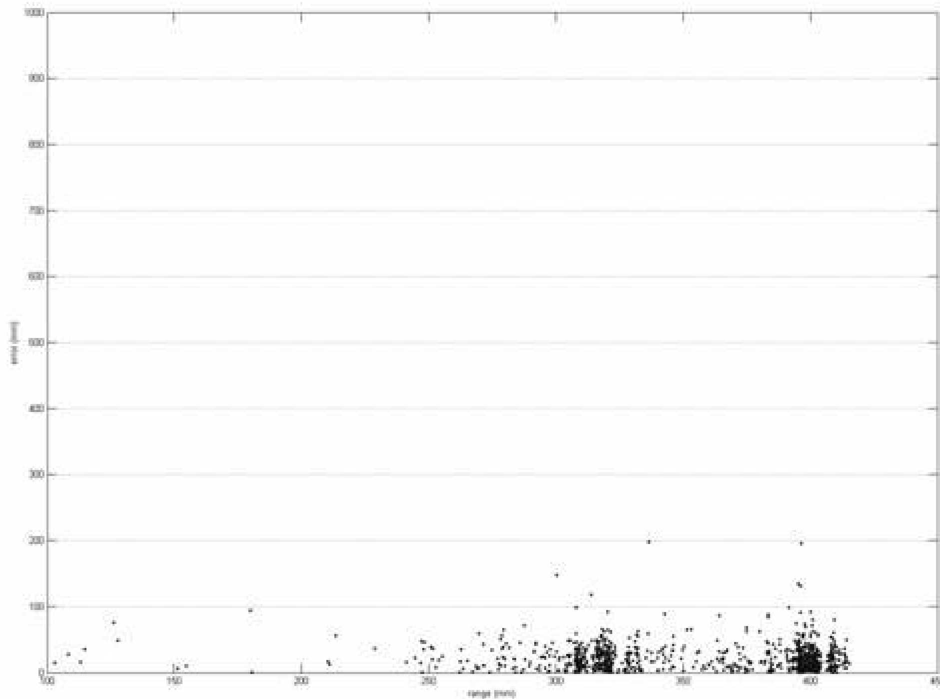


Figure 4-7: Graph of distance traveled vs. range error for regional dynamic technique with expanded motion model (Experiment F).

Range error for the expanded motion model with the regional dynamic technique. Not much improvement apparent, since the gains are mostly in angular error, which cannot be easily graphed.

region and adapting to a previous region will cause problems.

However, the expanded model was very effective with the full, regional algorithm, improving its results beyond the global technique using the basic model. The range error of 1.3% and angular error of 2.5% were slightly better than the standard technique (A), even though the environment had little variation in regional conditions and was more suited to the non-regional method. The angular error, which is more important since it causes increasing error, is also improved from the optimal technique (B). Figure 4-7 demonstrates this algorithm. Comparing experiments E and F demonstrates that there are circumstances where the regional algorithm provides an improvement, even though experiments C and D show that sometimes the global algorithm is better. The choice of algorithm depends on the specific model being used, as well as the composition of the environment.

4.3.4 Comparison of Models

As these results show, dynamic motion models are better able to represent the robot's motion, and the prediction phase of localization becomes more accurate. Table 4-1 shows a comparison of the various methods in the first environment section. All of the dynamic methods give similar results and they all produce superior motion predictions to the static motion model method that is the base case (experiment A). The particular method that is optimal in any given situation depends on the environment, although over the long run, the regional dynamic technique should produce the minimum error, especially if the expanded model is used. However, this convergence may require a large number of traversals in order to get the necessary number of data points for each region. Until execution reaches this point, the other techniques have a temporary advantage, since they require less data.

Observing Figure 4-3 through Figure 4-7, we can compare a graphical representation of the error that in some ways is more useful than the error score. Although all the graphs look similar, the outliers are reduced for the dynamic techniques. It is especially obvious when comparing the default technique to the various dynamic techniques. In localization, it is the outliers that are particularly dangerous since, regardless of the average error, if the predicted location is far enough from the actual location, localization may fail. MCL suffers from bias caused by representing an infinite space with a finite sample set. If a motion has enough error, there may be no particles at the correct location. This would cause a localization failure. The graphs show how the improved motion models reduce the greatest errors to improve localization more than is apparent from the percentage error.

The technique of calculating the global optimum parameters provides very good results, especially in an environment like this with little change in surface. Range error especially benefits from this technique, since it is relatively constant. However, generating this model requires manual collection and processing of data before execution, which somewhat defeats the purpose of a dynamic algorithm. The benefit is that offline processing can handle a larger number of data points, resulting in more accurate parameters. Of course, any changes, such as tire pressure, will invalidate the model. Although this technique uses part of the dynamic algorithm, it is not truly dynamic nor is it usually a practical method.

The choice between the two dynamic techniques depends on the circumstances. If the environment has different surfaces then having the parameters change with the region provides a benefit. If, on the

other hand, the surfaces are constant but the robot changes conditions, a globally dynamic technique will update more quickly, since the data points are all processed into the same model. A situation where this is useful might be when the robot changes its behaviour as its battery drains. The global technique could adapt faster to changing robot conditions, but it cannot recognize different surfaces. Note that the regional algorithm will eventually adapt to global conditions, but it will require more data since each region must be updated. The choice depends strongly on the environment, although the regional method is more adaptable. The regional technique also provides the opportunity to improve the motion model to more closely represent the actual conditions. Increasing the complexity of the model can allow the variations between smaller regions to be represented, even though it takes more data to optimize the increased number of parameters.

These results demonstrate that adding dynamic motion models to MCL provides a benefit to localization. Although slightly different dynamic techniques provide different advantages, they are all superior to the static technique. Aside from the tests described above, several other data sets in different environments were examined, with similar results. One such test involved a similar robot in a different building where the floor was carpeted instead of concrete. The map of the environment incorporated a serious error that caused localization to fail for most techniques. One corridor was actually much shorter than it appears in the map, causing reported motion along part of that corridor to have a large systematic error. Because of this, the range error in the dynamic techniques actually increases, as they increase the variance to handle the systematic error. Only the regionally dynamic

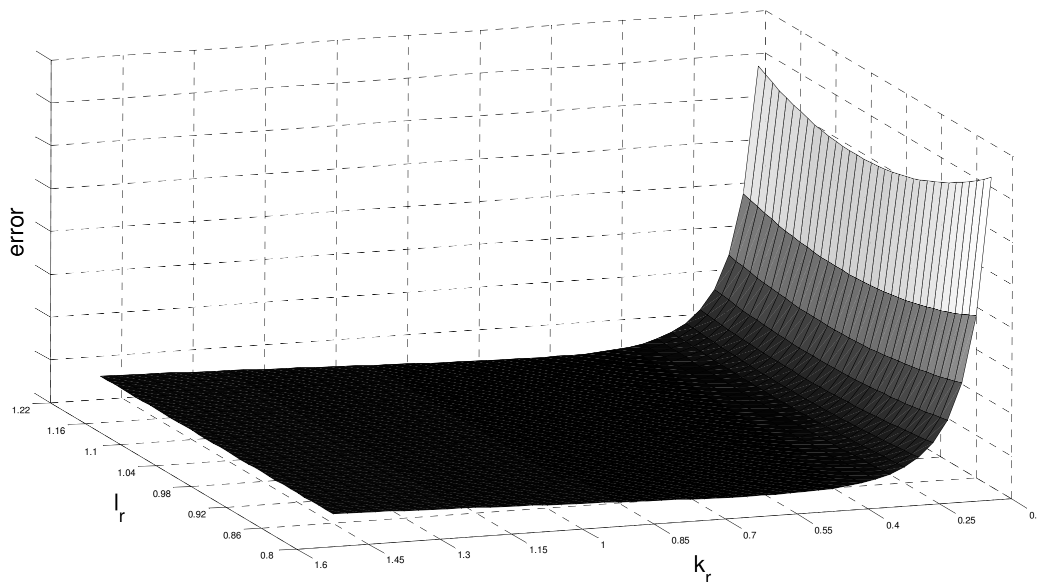


Figure 4-8: Surface of the error for the range parameters.

The surface shows how range error varies with the two range parameters for the data set collected by the dynamic motion algorithm

technique was able to successfully localize in this environment. Adding the expanded model caused a further improvement, decreasing angular error while increasing the range error even more. Adapting the parameters to give higher probability to motions which suffered from the mistake in the map caused motions not affected by the problem to have a higher error. The variance in the model had to take into account both the normal motion of the robot and the discontinuous motion. The result is an increase in the reported error in the prediction, since every motion had to be predicted in the correct location for both situations. Naturally, the prediction that is incorrect increases the reported error, even though the robot is successfully tracked. These results can be seen in the second environment section of Table 4-1.

4.3.5 Random Restart Optimization

Certain additional experiments were performed to test the behaviour of the dynamic motion model algorithm under other circumstances that do not necessarily compare to the results of A through F.

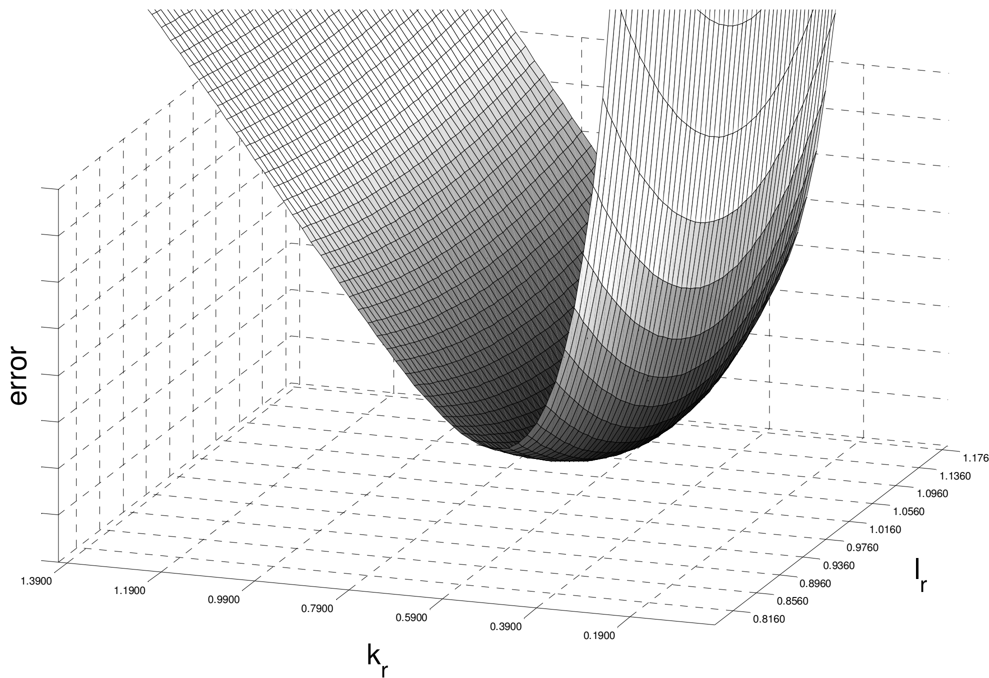


Figure 4-9: The global minimum in the error surface for the range parameters.

A zoomed in view for the error axis of the global minimum from Figure 4-8. Although there is not much change in error versus k_r for much of the range this surface shows that a global minimum does exist.

Throughout these experiments we have used a standard local search technique for optimization, however this method may not be accurate. As discussed in Section 4.2.7, various circumstances might require a global optimization technique. Although the surface in Figure 4-8 indicates that there is only a single minima in our function, various local conditions might cause temporary local minima. Experiments J and K test the dynamic algorithm with multiple, random starting points for optimization which should eliminate the problems of local minima. Of course, this type of optimization requires significantly more processing than local search, but it might be worth it if doing so solves a local minima problem. However, we see from the results that instead using multiple starting points causes a significant reduction in the performance of the dynamic motion model algorithm. The decrease in performance is caused because the increase in processing time for the optimization makes it impossible to have up to date parameters. Much of the data must be discarded, or is so old that its relevance has passed. This is especially obvious with the global technique (J) since that requires constant updates. For the regional technique (K) there is less of an effect because the calculations can be performed for one region before the robot re-enters it. However, some data is still lost because there is no longer time to develop parameters for all regions. Updating the parameters dynamically according to circumstances that have changed a significant time previously actually harms the performance of the algorithm, making it worse than the default case (A). Using multiple starting points results in 1.7% error for range and 14.7% error for angle using the global algorithm (J) and 1.3% range error with 5.3% error for the regional algorithm (K). These experiments demonstrate that global optimization methods reduce the performance enough to offset any possible gains from avoiding local minima for the dynamic motion model algorithm and that local methods, such as the fminsearch function being used are sufficient.

4.3.6 Hardware Specialized Model

Finally, the basic dynamic motion model algorithm was tested with a model representing the specific hardware configuration of the robot. Experiment G shows the results of MCL using static parameters for this model. Because no experiments were performed to optimize the parameters, the error is relatively large with 1.6% range and 22.5% angular error. However, the global dynamic algorithm (H) produced a large improvement resulting in 2.0% range and 6.7% angular error. Finally, the regional technique resulted in 2.3% range error and 4.1% angular error. This model is notable because it requires fewer parameters than the comparable general model to represent the same types of error. Although we used the standard types of motion error to demonstrate this specialized model, the results are difficult to compare to the other algorithms because, for the specialized model, range and angular error are correlated. The original models generate the two kinds of error separately but, by modelling the wheels, all types of position error are correlated. Although the error is worse than the original model the dynamic algorithm produces significant improvements over the specialized model with default parameters. It cannot produce improvements beyond the ability of the model to represent the robot. We also notice that the range error actually increases with the dynamic algorithms. However, the overall error is decreasing since the improvement in angular error is much greater than the increase in range error. It is hard to say what the relative importance of range error versus angular error is, but because of the potential for path divergence based on angular error we know that the weight of range error is lower. Because the specialized model represents the physical

robot, the slight differences between the model and the robot cause a certain minimum error. The important point is that the dynamic algorithm can produce an improvement using a specialized motion model that was previously impractical in many situations. The ordinary model can be applied to many different robots and corresponds to what is actually observed during motion. This means it is possible to adapt an implementation of the standard model to almost any robot, while the specialized model must be developed separately, with uniquely determined parameters, for each separate robot. Thus, even though the specialized model may be simpler, there is no pre-existing source for determining the parameters. The large angular error in the static experiment (G) is the result of not performing exhaustive experiments to optimize the model. However, with dynamic models the parameters can quickly be optimized to provide an acceptable amount of error. These experiments indicate that regardless of the actual motion model being used, the dynamic motion model algorithm can update the parameters from reasonable starting values. This result allows the algorithm from this chapter to be applied to localization on almost any robot, removing the need for determining the parameters by exhaustive experimentation.

Table 4-1 Experimental results of all motion model algorithms.

	First environment				Second environment			
	% range error		% angle error		% range error		% angle error	
		Standard deviation		Standard deviation		Standard deviation		Standard deviation
Standard (A)	1.6592	.0116	4.5311	.2614	7.2343	.0362	5.9671	.1810
Global Static (B)	1.3586	.0042	2.9509	.0340	8.6733	.0559	1.3428	.0351
Global Dynamic (C)	1.2418	.0063	2.6320	.0330	9.5313	.0512	1.4835	.0435
Regional Dynamic (D)	1.3882	.0067	2.7878	.0342	10.9178	.0473	1.7166	.0393
Expanded Global (E)	1.4043	.0045	4.0654	.1126	11.9911	.0443	3.4840	.0980
Expanded Regional (F)	1.2988	.0075	2.5213	.2333	11.2261	.0445	1.3250	.0439
Specialized Static (G)	1.6325	.0183	22.4845	.0427	4.0386	.0860	75.6111	.0575
Specialized Global Dynamic (H)	2.0350	.0085	6.7398	.0262	4.8643	.0351	63.2888	.0799
Specialized Regional Dynamic (I)	2.3196	.0132	4.1357	.0717	2.5378	.0498	64.2121	.0598
Random Restart Global Dynamic (J)	1.6555	.0112	14.6604	.5088	12.5248	.0443	12.1703	.0980
Random Restart Regional Dynamic (K)	1.2975	.0040	5.3045	.1040	11.7484	.0445	2.1998	.0439

4.4 Conclusion

This chapter shows the derivation and implementation of a technique for dynamically calculating the parameters of the Monte Carlo Localization motion model during ordinary execution of the algorithm. The technique requires very little overhead and provides a strong benefit over the ordinary technique of using a static model determined experimentally from a similar robot. In fact, the most common current technique is to estimate the model and modify it using trial and error until localization is successful. The problem is that performing experiments to determine the parameters is a difficult and time consuming process. Since the parameters of a real environment change over time, it is usually not worthwhile to develop an accurate model when an approximate one will still allow MCL to function. My dynamic motion model technique provides a viable alternative to both these methods, allowing an accurate model to be created and maintained without requiring skilled user input. Since the frequency and size of the updates can be modified to suit the platform, there are many situations that might benefit from using a dynamic model. Because MCL is running properly when the dynamic algorithm is active, there is no urgency in processing the error data into new parameters. Thus, the additional run time required can be limited to what is available on the particular platform. In fact, very good results can be obtained by using offline processing to determine a new model whenever conditions change. Although the offline method does not provide all the benefits of the dynamic algorithm, it provides a great improvement over the default method.

Another benefit of having dynamic motion models is that they can be used to automatically optimize a robot to different conditions in the environment. This may be an important feature for a robot that runs autonomously between different areas. It is impractical to perform laborious experiments to determine an optimal model for different regions, but a general model can be automatically refined into specific models for many different conditions. Increasing the complexity of the motion model also helps with determining specializations for different regions, since the added complexity allows it to represent the true conditions more closely.

By reducing the error due to the motion model in MCL, our technique provides localization with greater resilience to errors from other causes. The more accurate the various models are, the more tolerance MCL has towards random events that might otherwise cause it to fail. In some circumstances this may be a major benefit, but even if ordinary MCL is successful in an environment, a more accurate model cannot harm its execution.

Dynamically optimizing the motion model parameters also makes it possible to use other motion models that might not be feasible normally. Using a different model ordinarily requires so much work that, unless there is some clear benefit, it is not considered. However, dynamic motion models allow virtually any model to be tested without additional experimentation. Thus, models specialized to the specific hardware can be used as easily as a generally applicable motion model. A localization implementation can thus apply easily to multiple robots while still being specifically adapted to each one. Also, the underlying hardware can be changed without needing elaborate experimentation to alter the program, even with a hardware specific motion model.

Since dynamic motion model MCL provides an annotated map which includes motion model parameters, it may be possible to use those parameters in order to determine information about the environment. For example, by discovering the parameters caused by various types of surface, the

robot might be able to identify those same surfaces if it encountered them again. Also, the motion models might be taken into account in path planning in order to give the robot a preference for stable surfaces. Finally, a robot might detect a change in its parameters and use them to identify a malfunction, such as deflated tires. These uses for dynamic motion models would provide additional benefits to the algorithm, above the improvements it makes to localization.

The primary benefit of dynamic motion models is that it causes some sources of error to decrease over time instead of increasing. One of the primary assumptions of MCL is that the underlying map is correct and does not change. Changing the parameters of the motion model in response to new information essentially changes, or adds to, the map, allowing it to more accurately represent the environment. Ordinarily, the error from the static map and parameters remains constant if the environment is static, or increases over time if the environment is dynamic. Thus, over the long term, localization gathers more and more error. Dynamically changing the motion model allows some of the error to be reduced over time, making MCL practical for longer periods. This chapter also demonstrates that it is possible to violate the static map assumption while still retaining the power of Monte Carlo Localization. Considering the evidence of dynamic motion models, it should be possible to alter other normally static parameters in order to further reduce sources of error that usually increase over time. In Chapter 5 I describe a technique for dynamically modifying the actual map cells in this way.

Chapter 5
Dynamic Maps in MCL

5.1 Introduction

One drawback to localization with MCL is that it requires a static map of the environment. Sensor readings are compared with the expected values from the map and the comparison generates the probability of the robot's location. Errors in the map can be partially compensated for by increasing the error that is assumed for the sensors. By increasing the error in the sensors we allow MCL to consider map errors to be caused by incorrect sensor observations. Since the number of correct sensor readings will probably overrule incorrect ones, on average, observations are correct and are able to compensate for the occasional incorrect reading. However, because MCL combines sensor error and map error, as map error increases, the allowable sensor error decreases until finally the algorithm fails and the map must be rescanned. Each error in the map is usually a minor matter for a localized robot, but the combination of minor errors can cause problems.

A correctly localized robot rarely fails localization due to map errors, but this is not true of global localization, where the robot's initial location is unknown. Especially in symmetric environments, global localization can easily fail due to minor map errors that would be ignored by a localized robot.

The approach described in this chapter is based on the idea that if a robot is localized it may reasonably expect its sensor data to reflect the environment. If that is the case, then it should be possible to update the map according to the sensor data. If a known error in the map is fixed, then the robot will have a greater ability to deal with any subsequent errors. Since global localization may depend heavily on minor features, having an updated map can be a great benefit.

Violating the static map assumption and detecting changes allows localization to be more accurate and more robust to error. It also provides additional information that may be useful in planning the robot's activities. Detecting opening doors and moving objects makes path planning more reliable, because it will be based on a more accurate representation. Further, when a new opening into an unexplored area is detected, the robot can add the new region to the map. The dynamic map algorithm described here makes it far easier for a robot to be deployed long term in an environment where other agents, including humans, are present and making changes.

Dynamic maps for MCL can be implemented by identifying binary objects, such as doors, and tracking their status using probabilistic methods [Avots et al. 2002]. There are several benefits of having explicit objects. Since an object consists of multiple cells that have the same probability, each scan provides more information about the object, allowing its state to be altered more quickly. Also, since most of the map is not dynamic, the probability of objects can be changed much more rapidly. Changes in the objects probably will not be able to change the map to make an invalid location match the sensors. However, explicit objects need to be manually defined before execution, adding to the work of defining maps. Since objects are binary, either present or absent, a moving object must be represented explicitly by creating a binary object at each possible location. With the dynamic maps described here, an object can appear anywhere without external assistance. Finally, the method in

[Avots et al. 2002] involves an importance factor, which increases the runtime logarithmically in the number of objects, making it unsuitable for having each map cell dynamic.

Algorithms for simultaneous localization and mapping (SLAM) have the ability to localize the robot and generate the map simultaneously in real time [Montemerlo et al. 2002]. These algorithms are meant to dynamically alter the map in the same way as my dynamic map MCL. Many of these methods use an algorithm which is guaranteed to converge to a correct solution. However, they suffer from the data association problem. On every sensor scan it must be possible to uniquely identify which feature of the map is responsible for each sensor reading. If this is impossible, then the guarantee of correctness does not hold. SLAM does not discover and use cell correlations, so the rate of update is slower if the map changes, since each cell must be considered independently. Further, SLAM involves significantly more processing than MCL, using up computing power that may not be necessary, especially after the map is generated. Dynamic map MCL was created specifically to provide an accurately changing map without incurring any significant overhead. Since it is a constant time addition to MCL, the map can be updated without requiring any more computing power than ordinary localization. Of course, the map cannot be generated from nothing as it can with SLAM, but once the map exists it can be kept up to date almost without cost. SLAM also, in common with ordinary MCL, makes the assumption that the map is static. Over time, the algorithm becomes more certain of the map and any changes will take longer to appear. Dynamic MCL explicitly makes the assumption that the map will change.

Algorithms that consider dynamic environments typically assume a static map with dynamic elements, such as people, which must be eliminated from consideration. In effect, these algorithms assume a static map but allow an additional form of sensor noise in the form of moving people. [Hahnel et al. 2003] describes a method for creating a map, using standard EM SLAM techniques, which can discover the static map of the environment despite dynamic elements. Similarly, [Fox et al. 1999] gives an algorithm for using MCL in an environment with many moving objects. Although both these papers give a method for handling a dynamic environment, they both assume an underlying static map. The benefit of dynamic MCL is that the static map assumption is no longer necessary. As the algorithm runs, it changes the map to correspond to the environment. Since dynamic MCL is implemented as an augmentation to ordinary MCL, there is no reason that other augmentations could not be used if warranted by the problem. For example, the algorithm described in [Fox et al. 1999] to discard readings relating to dynamic objects during MCL can coexist with my algorithm for modifying the map in accordance with changes in the environment. Dynamic MCL allows fundamental changes to be accounted for, as opposed to merely ephemeral objects that are only observed once.

5.2 Dynamic Maps

In order to alter the map, it needs to be added to the MCL formula. Consider each cell of the map to be an independent object, which can be either present or absent. Although independence is usually not entirely valid, it is an assumption that is often made. Consider $y_t = \{y_{1,t},\ldots,y_{K,t}\}$ the set of individual cells in the map. Since we are considering these cells to be independent, if the location is known, then $p(y_t \mid x_t, z_t) = \prod p(y_{k,t} \mid x_t, z_t)$.

With this background, the new state equation is $p(y_t, x_t \mid \underline{z_t}, u_t)$. Unfortunately, it turns out that this equation cannot be factored, since the map state is not fully determined with only the current location. However, notice that each sample in MCL represents not only a current location, but also the history of locations that lead to that location. Since each particle is only moved according to the motion model, they may be considered as x^t instead of x_t with no change to the algorithm. If we use the equation $p(y_t, x^t \mid z^t, u^t)$ instead of $p(y_t, x_t \mid z^t, u^t)$, we are left with an equivalent factorization and we can thus use the MCL algorithm without significant changes. The factorization used is similar to the one in [Avots et al. 2002], which was used to add the state of doors into the MCL algorithm.

5.2.1 Factoring

The size of the state space of (y_t, x^t) is exponential in the size of y_t, so we need some way of factoring the posterior in order to reduce the state space.

First, Bayes rule and the Markovian property give us:

$$p(y_t, x^t \mid z^t, u^t) = \eta p(z_t \mid y_t, x_t) p(y_t \mid x^t, z^{t-1}, u^t) p(x^t \mid z^{t-1}, u^t) \qquad \textbf{5-1}$$

Now, consider the three parts of equation (**5-1**).

Without any data we assume that all states are equally likely, and also that the probability of a random sensor scan is a constant. Therefore:

$$p(z_t \mid x_t, y_t) = \frac{p(x_t, y_t \mid z_t) p(z_t)}{p(y_t, x_t)} = \eta' p(x_t \mid z_t) \prod_k p(y_{k,t} \mid x_t, z_t) \qquad \textbf{5-2}$$

Remembering that cells in the map change status independently in the model, and again using the Markovian assumption, we get:

$$p(y_t \mid x^t, z^{t-1}, u^t) = \prod_k \sum_{y_{k,t-1}} p(y_{k,t} \mid y_{k,t-1}) p(y_{k,t-1} \mid x^{t-1}, z^{t-1}, u^{t-1}) \qquad \textbf{5-3}$$

Finally:

$$p(x^t \mid z^{t-1}, u^t) = p(x_t \mid x_{t-1}, u_t) p(x^{t-1} \mid z^{t-1}, u^{t-1}) \qquad \textbf{5-4}$$

Recombining these three equations (**5-2**, **5-3**, and **5-4**) we can rewrite equation (**5-1**) as:

$$p(y_t, x^t \mid z^t, u^t) = \eta'' p(x_t \mid z_t) p(x_t \mid x_{t-1}, u_t) p(x^{t-1} \mid z^{t-1}, u^{t-1}) \prod_k p(y_{k,t} \mid x_t, z_t) \sum_{yk,t-1} p(y_{k,t} \mid y_{k,t-1}) p(y_{k,t-1} \mid x^{t-1}, z^{t-1}, u^{t-1}) \qquad \textbf{5-5}$$

With the parts simplifying as:

$$p(x_t \mid z_t) p(x_t \mid x_{t-1}, u_t) p(x^{t-1} \mid z^{t-1}, u^{t-1}) = p(x^t \mid z^t, u^t) \qquad \textbf{5-6}$$

$$\prod_k \left(p(y_{k,t} \mid x_t, z_t) \sum_{y_{k,t-1}} p(y_{k,t} \mid y_{k,t-1}) p(y_{k,t-1} \mid x^{t-1}, z^{t-1}, u^{t-1}) \right) = \prod_k p(y_{k,t} \mid x^t, z^t, u^t) \qquad \textbf{5-7}$$

The equality in equation **5-7** can be demonstrated using some basic probabilistic rules and assumptions about the correlations between the various random variables.

$$p(y_{k,t} \mid x^t, z^t, u^t) = \varphi p(x_t \mid y_{k,t}, x^{t-1}, z^t, u^t) p(y_{k,t} \mid x^{t-1}, z^t, u^t)$$

$$p(y_{k,t} \mid x^t, z^t, u^t) = \varphi p(x_t \mid y_{k,t}, z_t) \sum_{y_{k,t-1}} p(y_{k,t}, y_{k,t-1} \mid x^{t-1}, z^{t-1}, u^{t-1})$$

$$p(y_{k,t} \mid x^t, z^t, u^t) = \varphi p(x_t \mid y_{k,t}, z_t) \sum_{y_{k,t-1}} p(y_{k,t} \mid y_{k,t-1}) p(y_{k,t-1} \mid x^{t-1}, z^{t-1}, u^{t-1})$$

$$p(y_{k,t} \mid x^t, z^t, u^t) = \varphi' p(y_{k,t} \mid x_t, z_t) p(x_t) \sum_{y_{k,t-1}} p(y_{k,t} \mid y_{k,t-1}) p(y_{k,t-1} \mid x^{t-1}, z^{t-1}, u^{t-1})$$

$$p(y_{k,t} \mid x^t, z^t, u^t) = \varphi'' p(y_{k,t} \mid x_t, z_t) \sum_{y_{k,t-1}} p(y_{k,t} \mid y_{k,t-1}) p(y_{k,t-1} \mid x^{t-1}, z^{t-1}, u^{t-1})$$

Thus, the original equation is:

$$p(y_t, x^t \mid z^t, u^t) = p(x^t \mid z^t, u^t) \prod_k p(y_{k,t} \mid x^t, z^t, u^t) \qquad \textbf{5-8}$$

Which contains the original MCL posterior and a new probability for the cells in the map. See [Avots et al. 2002] for more details about the factorization.

5.2.2 Binary Object Bayes Filtering

Since the method for calculating $p(x^t \mid z^t, u^t)$ is already known for a given map in the MCL algorithm, the only new method needed is to calculate the probability of each cell in the map. These cells are binary objects since they are either present or absent. Each $y_{k,t}$ can be either 0 or 1 with the probability of each summing to 1. Thus, the method for calculating the probabilities is the same as in [Avots et al. 2002]. Let $\pi_{k,t} = p(y_{k,t} = 1 \mid x^t, z^t, u^t)$. Then

$$\pi_{k,t} = \frac{p(y_{k,t}=1 \mid x_t, z_t)\, p(z_t \mid x^t)}{p(y_{k,t}=1)\, p(z_t \mid x^t, z^{t-1}, u^t)} \pi^+_{k,t} \qquad \textbf{5-9}$$

where

$$\pi^+_{k,t} = p(y_{k,t} = 1 \mid y_{k,t-1} = 1)\pi_{k,t-1} + p(y_{k,t} = 1 \mid y_{k,t-1} = 0)(1 - \pi_{k,t-1}) \qquad \textbf{5-10}$$

In equation (**5-9**) the only unknown probability is $p(z_t \mid x^t, z^{t-1}, u^t)$ in the denominator. Rather than trying to calculate it, we exploit the fact that $y_{k,t}$ is binary so $(1 - \pi_{k,t})$ can be calculated in the same way as $\pi_{k,t}$ using $y_{k,t} = 0$ instead of $y_{k,t} = 1$. The two equations are then divided to cancel the unknown quantities.

$$\frac{\pi_{k,t}}{(1-\pi_{k,t})} = \frac{p(y_{k,t}=1 \mid x_t, z_t)}{1-p(y_{k,t}=1 \mid x_t, z_t)} \frac{1-p(y_{k,t}=1)}{p(y_{k,t}=1)} \frac{\pi^+_{k,t}}{\pi^-_{k,t}} \qquad \textbf{5-11}$$

The result, equation (**5-11**), consists entirely of known quantities. $p(y_{k,t} = 1)$ is the prior probability that a cell is occupied. The various $p(y_{k,t} \mid y_{k,t-1})$ values are the transition probabilities for a cell, $\pi_{k,t-1}$ are, of course, the prior occupancy probabilities and finally, $p(y_{k,t=1} \mid x_t, z_t)$ is the probability of occupancy given robot location and sensor data. To get a useful value from the odds ratio, we use the equality $\pi_{k,t} = 1 - (1 + \pi_{k,t}/(1 - \pi_{k,t}))^{-1}$.

The representation of $\pi_{k,t}$ is actually in closed form, so it requires only a constant time operation to calculate. Since $p(y_{k,t} = 1 \mid x_t, z_t)$ involves sensor values and raytraces which are already used for MCL, little additional processing should be required. It is possible to modify the importance factor, as in [Avots et al. 2002], to take into account the new map data, where each cell is not merely present or absent but has a probability of presence. Using this data results in a runtime increase at least logarithmic in the number of binary objects. The probability of a location becomes the sum of the probabilities of that location for both states of all visible objects, multiplied by the probability of the object states. While that is acceptable if there are only a small number of objects, such as doors, if the objects are the cells of a map, the number becomes unmanageable. However, most map data used for MCL is actually represented as probabilities in an occupancy grid map, but is thresholded to be either present or absent. I decided to use the same simplification for my algorithm and consider each cell as either present or absent depending on a threshold value on its probability. The processing time therefore remains unchanged, since the importance factor is calculated in the same way.

5.2.3 Cell Correlations

In order to perform the factorization, it is necessary to assume that map cells change independently of each other. However, this assumption is not entirely accurate. In fact, groups of adjacent cells that represent the same objects are likely to be completely dependent. To some extent ordinary MCL also assumes cells are independent, but it only becomes relevant when the cell probabilities are changed in dynamic MCL. It is easy to model correlations by annotating the map with correlation probabilities between adjacent cells, however, using this information is more difficult. Methods such as loopy belief propagation or variational methods [Jordan et al. 1999] can propagate belief through a connected graph, but they are time consuming and sometimes do not converge. Since dynamic MCL must run in real time without being much slower than ordinary MCL, these techniques are not sufficient. However, it should be noticed that the cell correlations in a map are of restricted types. Small groups of adjacent cells are highly correlated, while being uncorrelated with their neighbours. Because of the limited correlation, it is possible to use a modified variational technique in order to implement cell correlations. When a cell is updated, the update is propagated to adjacent cells along the links, but the propagation is not permitted to flow back to a cell that has already been modified. Also, the flow stops when the accumulated correlation probability falls below a threshold. It is necessary to bound the correlation probability to prevent too many updates, but in practice, only a few steps occur. These few steps are enough to achieve a significant improvement in the results.

The key to using cell correlations is to perform operations using two different and conflicting sets of assumptions. Each set of assumptions reduces one part of the problem to a solvable operation but makes the other part intractable. We have already seen that, by assuming cells to be independent, we can factor the belief as:

$$p(y_t, x^t \mid z^t, u^t) = p(x^t \mid z^t, u^t) \prod_k p(y_{k,t} \mid x^t, z^t, u^t) \quad \textbf{5-12}$$

This factorization is used to update the individual cells according to the robot's sensors. However, once the update is performed we discard both the assumption and the resulting factorization. Instead, we assume that each cell depends on its neighbours and is independent of the robot's sensors and position. According to this set of assumptions:

$$\begin{aligned} p(y_t, x^t \mid z^t, u^t) &= p(x^t \mid z^t, u^t) p(y_t \mid x^t, z^t, u^t) \\ &= p(x^t \mid z^t, u^t) p(y_t) \\ &= p(x^t \mid z^t, u^t) \prod_k p(y_{k,t} \mid y_{k-up,t}, y_{k-down,t} y_{k-left,t} y_{k-right,t}) \end{aligned} \quad \textbf{5-13}$$

The determination of the robot's position is unchanged, but the map cells now depend on their neighbours and not on the robot. By making this assumption any changes made to the map can be propagated to the adjacent cells and the weight of the cell correlations adjusted. Separating the algorithm into two phases with different assumptions allows the algorithm to consider additional dependencies without having to deal with the intractable problems caused by the interaction of the new dependencies with the old. In effect, during the first phase of the algorithm, as represented by equation (**5-12**), we assume that the probability of a cell being occupied depends only on what the robot senses directly, with additional effects coming from some unknown source. During the second phase, shown by equation (**5-13**), we assume that the probability of a cell being observed depends only on its neighbours, with other changes caused by external, unconsidered, forces. Of course, two sets of contradictory assumptions cannot possibly be a reflection of reality, however, each assumption is a reasonable simplification and using both sets iteratively results in less simplification than either set exclusively.

In dynamic MCL, it is necessary to modify the cell correlation probabilities dynamically on each cycle. However, given the nature of the sensors used, it is unlikely that adjacent map cells will be observed on a single scan. The solution to the problem is to cache observed changes to each cell until an adjacent cell has also been observed. At that point, the difference in the changes of the cells can be used to adjust the correlation between them.

Adding cell correlations significantly improves the dynamic MCL algorithm since a correlated group of cells can change together whenever any member of the group is observed. The result is that although the update of individual cells must be slow to allow localization to work, if a group of cells change they will update very quickly, since each observation will correlate them, and as they become more correlated every observation of a member of the group will update the entire group. Thus an object can appear or vanish more quickly than any single cell.

5.3 Algorithm

The preceding formulae can be used to augment an implementation of MCL in order to modify the map dynamically during processing. The MCL algorithm must raytrace along all sensor paths to calculate the probability of a particle. However, if the robot's position is known with high

probability, then any differences between the sensor reading and the raytrace are more likely to be errors in the map than in the sensors. In that case, the logical action is to correct the map.

The method I used is to consider each cell of the map to be present with probability $\pi_{k,t}$. On each step of the MCL algorithm an augmented raytracer is used for the robot's most likely location. The augmented raytracer follows a ray normally, passing through each map cell along the ray. However, at each cell along the path, the probability of that cell is altered according to equation (**5-11**). Although the augmented raytracer could be run on all samples, it is more productive to determine the most likely location and use the augmented raytracer only on it. Some metric must be used to determine the probability that the most likely location is correct. One such method is to compare the sum of the particle weights to some threshold as in [Thrun et al. 2005] (Section 8.3.5). If the weights are below the threshold, the robot is assumed to be lost and the new raytracer is not used. Another detail is that only the probabilities of observed cells are updated by the new raytracer. Although the robot's knowledge of unobserved cells gradually decreases over time, we make the assumption that the environment does not change except when it is observed. This assumption prevents the map from decaying to a uniform field in locations the robot does not see.

For calculating the sensor probability of each cell, the simplifying assumption that either that cell or the existing wall is correct is used. The assumption is necessary because the normalizer for the sensor probabilities is not known, so some method must be used to normalize the values. In practice, when a new cell becomes occupied, it exceeds the threshold before any other cell, and then the assumption becomes valid again. The short period during which it is invalid for some cells does not affect the operation of the algorithm.

In order to find the robot's most likely location, the sample with the highest importance factor is used. Other locations are possible, including the weighted average of all samples. The algorithm cannot run if the robot's location is unknown. As described in section 4.2.6 for dynamic motion models, MCL usually either produces a correct location or fails completely. As long as the overall error does not exceed MCL's ability to compensate, the high probability location will probably be correct.

These implementation details do not change the fundamental algorithm, which is an implementation of MCL together with the binary object formulae as described above. The only simplification to equation (**5-11**) is in the calculation of $p(y_{k,t} = 1 \mid x_t, z_t)$, a value which is at best a numerical approximation to the error in a physical sensor device.

The following pseudocode summarizes the algorithm for dynamic MCL.

Table 5-1: Dynamic MCL algorithm

```
1:  Repeat N times
2:          Draw a random particle
3:          Move particle according to the motion model
4:          Annotate particle with a weight from the sensor model
5:  Resample a new set of particles from the annotated set
6:  Find the most probable location (mean of particles)
```

```
7:    For each sensor reading
8:          Raytrace to the nearest occupied cell
9:          For each cell on the path
10:               Alter the occupancy probability of the cell
11:               Alter the occupancy probability of neighbouring cells according to influence
12:               Mark cell as observed
13:               If neighbouring cell marked observed
14:                     Adjust influence between cells
15:                     Unmark cells as observed
```

5.4 Experimental Evaluation

The dynamic map algorithm was implemented and tested using real data collected in our building. The data was created using a Pioneer 2Dxe robot equipped with a laser rangefinder. The objective of the tests was to show that the map could be updated correctly without introducing errors or causing localization to fail. Since the algorithm has an almost constant runtime there is no tradeoff necessary between the time required to update the map and the benefit obtained by doing so.

Dynamic map MCL is designed to gradually update the map of the environment used for localization. Ordinarily, MCL uses a static map which, in a dynamic environment, gradually becomes less accurate as the environment continues to change from the time the map was produced. The experiments were selected to validate the dynamic algorithm by demonstrating that, over time, the map becomes a more accurate representation of the environment. Accuracy was defined by comparing the additions to the map with the actual environment. Localization and global localization will perform better on a more accurate map. The experiments demonstrate that the map is updated correctly, the benefit obtained from this update depends on the specific problem.

Figure 5-1: Before and after 2 passes through the environment

Figure 5-1 shows the map of the environment used to generate the test data. Changes were made to the environment after the map was scanned by opening and closing doors and by placing boxes in the corridors. After 1 pass through the changed environment the robot has mostly added the new features to the map and has correlated the changed objects, allowing them to be completed very quickly.

After two passes, all changes have been completely added to the map. The rate of update is slower than in [Avots et al. 2002] because each cell must be observed several times, instead of each object. However, without correlations it takes at least five passes to completely adapt the map. Allowing cells to become correlated permits much faster updating without compromising localization. In [Avots et al. 2002] the dynamic objects can be updated in a single pass because they are manually defined ahead of time and are known to be completely correlated. Since dynamic MCL has no predefined objects or correlations, it is necessarily slower, but because it can discover the correlations it can still update very quickly.

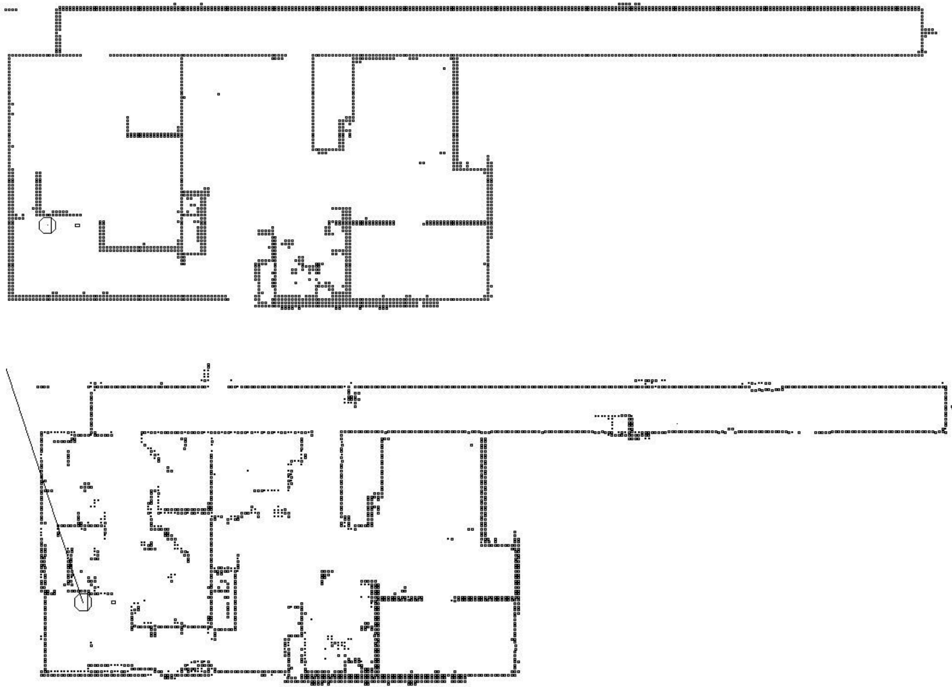


Figure 5-2: Before and after 5 passes through the environment using a schematic map.

Another test, shown in Figure 5-2, was to use the same data but starting with a map consisting of the minimum possible information. From a schematic map consisting of only the walls and partitions, the algorithm was able to adapt it with all the features that were missing. Those portions of the map that were observed were corrected properly. The benefit of being able to start with a limited map is that it may not be necessary to scan a map manually with a robot. Instead, the map could be entered using blueprints of the environment and, as the robot passed through, it could correct the map until it was accurate. Usually, MCL uses the most accurate map possible, since it will lose accuracy over time, but with a dynamic map the accuracy of the map increases as the robot traverses the environment. Of course, portions of the environment that were insufficiently observed were not completely added to the map, so the result is not identical to the environment. However, observed areas have become more accurate and the map will only become a better reflection of the environment as the robot traverses it over time.

Another feature noticeable in Figure 5-2 is that some of the objects in the corridor are somewhat more diffuse than they appeared in Figure 5-1. Because of the lack of features in the map, new objects cannot be added as accurately with reference to existing features. As the map is corrected and more distinct objects are added, the location of the new objects becomes clearer. After five passes, the objects are almost completely defined in the map, but some of them obviously require several more passes to fully correct them. The benefit of dynamic MCL is that the robot can operate independently of this process. As it performs its task, the map becomes more accurate. All other data files tested exhibited similar behaviour, with the observed portions of objects being added to the map and no new errors introduced.

5.4.1 FastSLAM Comparison

The dynamic MCL algorithm is very similar to FastSLAM, with the major difference being that FastSLAM keeps the map state separately for each particle, while dynamic MCL maintains a single global map. The single map results in two significant changes in the behaviour of the algorithm. First, the run time over ordinary MCL is only increased by a constant in terms of the number of particles. Regardless of how many particles are necessary for localization, dynamic map MCL requires the same amount of additional processing. FastSLAM requires additional processing that is at least linear in the number of particles, disregarding the work necessary to continuously copy the maps. Dynamic MCL thus requires significantly less processing power than FastSLAM.

The per particle map is what allows FastSLAM to determine a map from nothing while localizing, since it can maintain multiple hypotheses until the robot observes a distinguishing feature. However, these map hypotheses necessarily include some unlikely maps. Even if FastSLAM was initialized to the starting map, it still predicts multiple different maps, and some of these would be unlikely. When the environment is mostly known these borderline maps are unnecessary. The basis of dynamic MCL is that the map is mostly known. In this case, the robot's position can be determined and the map can be altered based on the single correct position, instead of updating based on multiple hypothesized positions. In the case with a pre-existing map, FastSLAM's ability to update the map is provided by dynamic MCL without the drawback of having to consider maps based on multiple conflicting paths. Of course, if the map is unknown considering multiple paths is necessary for success, so dynamic MCL is in no way a replacement for FastSLAM, it merely uses similar ideas to apply to a situation that FastSLAM does not handle well. If the map of the environment is mostly known in advance, dynamic MCL provides an efficient solution to handling dynamic elements and previously unobserved areas, without causing additional uncertainty.

To discover if dynamic MCL provides appreciable efficiency gains over FastSLAM when the appropriate map is available, the FastSLAM algorithm was run on the same data set as in Figure 5-1. FastSLAM was able to generate a map, but it took 428 seconds and, of course, did not include the areas that were not visited. In contrast, dynamic MCL completed the 2 passes in 68 seconds, an 84% improvement. When only minor features need to be updated in a mostly complete map, it is unnecessary to incur the cost of FastSLAM, since in these cases dynamic MCL is far more efficient while providing the same result. Dynamic MCL also allows previously visited areas to remain in the map, even if the robot has not observed them.

5.5 Conclusion

This chapter describes an augmentation to MCL which allows the map to be updated according to the sensor measurements of a localized robot without a serious increase in running time. By considering each cell of the map to be an independent binary object and by making some simplifying assumptions, the static map required by MCL can be modified dynamically without requiring any human intervention. Instead of becoming less accurate over time, the map becomes more accurate as the robot traverses the environment. Experiments with real datasets show that the map can be updated properly without introducing errors. A change in the environment can be reflected in the map after very few passes by the robot. Since the map is not updated incorrectly and the running time

is minimal, dynamic map techniques provide a useful addition to ordinary MCL in many situations. The result of the algorithm, having an accurate map, will always benefit the accuracy of MCL.

Dynamically correcting the map causes the largest source of error in MCL to decrease over time. Ordinarily, the best possible situation is for this error to remain constant, however in environments with dynamic elements, especially people, it is more likely that gradual changes occur. As the physical environment changes, errors build up in MCL, reducing its ability to handle any additional error. With dynamic updates the error is instead reduced over time, making localization more robust to other problems. Also, recognizing changes in the map might allow certain circumstances to be detected and considered in planning. For example, doors could be detected when they open and the robot could be sent to explore the new area. Also, new routes could be discovered as objects are moved. Removing the static map assumption greatly increases the power of MCL to handle real situations with dynamic elements. Furthermore, recognizing changes in the environment allows further improvements to be made at higher levels of control.

Chapter 6
Skeletal FastSLAM

6.1 Introduction

While FastSLAM is a good solution to localization and mapping, it suffers from some problems, notably the loop closure problem. As the robot travels around a loop in the environment, it has no way to incrementally correct its position. Only once the robot arrives at the end of the loop can it realize that the correct path is the one that arrives in the right place so that the map joins up. Because particle filtering only represents the highest probability locations, over a long loop the correct path may be lost. Overcoming this problem requires a large number of particles relative to the size of loops in the map, which means the algorithm increases in runtime and memory with the size of the map.

SLAM is normally defined as generating a map from total uncertainty about the environment. However, there is often some information available about the map, especially in an indoor environment. Unless your robot is a bulldozer, it is constrained to follow certain paths indoors, corresponding to the building's corridors. These corridors are relatively easy to describe based on a simple floor plan, or even by observing the environment. In this chapter, I demonstrate how minimal information about the skeleton of the environment can be used to improve FastSLAM and reduce the loop closure problem, requiring only enough particles for the local areas of uncertainty. Skeletal FastSLAM provides an intermediate step between pure localization with a static map and pure SLAM with total uncertainty about the environment. Many problems with some preexisting knowledge might benefit from this approach.

The primary contribution of skeletal FastSLAM is to allow some simple initial information to be used for SLAM that is much less than the total knowledge required by MCL and is easier to produce than a partial map of the same form as the calculated map that might be used as the initial state for SLAM. Although these two algorithms are powerful, there are many situations that are somewhere between the two conditions. For these problems, the choices are to accept the error caused by the uncertainty in MCL or to discard the initial information in SLAM. The algorithm described in this chapter allows FastSLAM to take advantage of some initial information. Similarly to the dynamic map MCL algorithm in Chapter 5, skeletal FastSLAM applies to problems with partial knowledge of the environment. The difference is that skeletal FastSLAM requires much less information than dynamic MCL and maintains a separate map for each particle. This allows it to successfully map unknown regions, where dynamic MCL requires enough map data to successfully localize at all times. The ability to use partial knowledge increases the usefulness of FastSLAM to situations that would ordinarily be much more difficult.

6.2 Skeletal FastSLAM

The key to using a skeleton map of the environment in FastSLAM is to realize that, especially in an indoor environment, the robot must follow certain paths. Obviously, a particle whose path corresponds to one of these corridors in the environment is more likely than one traveling at a tangent

to the corridor. Of course, this only applies if the particle is close enough to the corridor, but when one of the corridors affects the robot's path, it can act as a very useful indication of the correct path.

6.2.1 Creating the skeleton map

Of course, the first step in implementing this technique is to define what exactly is meant by a skeleton map. The skeleton is defined by a series of line segments marked by their endpoints. Each line segment marks a corridor that constrains the robot's direction of travel. It is not necessary for every possible path to be represented, only those composing major loops in the environment. The skeleton gives the direction and length of each corridor, including the structure of their intersections. Such a map is very easy to construct, especially in an indoor environment where a schematic diagram is often available. Also, buildings are usually constructed with corridors at right angles, making it easy to determine the intersections of the skeletal map. In such an environment, any user could easily define the necessary skeleton with minimal understanding of the underlying algorithm.

6.2.2 Monte Carlo Localization with Paths

Although MCL is originally defined to solve for only the robot's current position, as in section 2.2 and [Dellaert et al. 1999], it is trivial, by recording the past poses of each particle, to alter it to track the robot's entire path. The derivation is similarly easy to alter, again using the Markovian assumption, producing models identical to ordinary MCL.

$$p(x^t \mid z^t, u^t) = \eta p(z_t \mid x^t) \int_{x_{t-1}} p(x^t \mid u^t, x^{t-1}) p(x^{t-1} \mid z^{t-1}, u^{t-1}) dx_{t-1} \qquad \textbf{6-1}$$

$$p(x^t \mid u^t, x^{t-1}) = p(x_t \mid x^{t-1}, u^t) p(x^{t-1} \mid x^{t-1}, u^t)$$

$$p(x_t \mid x^{t-1}, u^t) = p(x_t \mid x_{t-1}, u^t) \qquad \text{Markovian assumption}$$

$$p(x_t \mid x^{t-1}, u^t) = p(x_t \mid x_{t-1}, u_t)$$

$$p(x^{t-1} \mid x^{t-1}, u^t) = 1$$

$$p(x^t \mid z^t, u^t) = \eta p(z_t \mid x_t) \int_{x_{t-1}} p(x_t \mid u_t, x_{t-1}) p(x^{t-1} \mid z^{t-1}, u^{t-1}) dx_{t-1} \qquad \textbf{6-2}$$

6.2.3 Derivation of FastSLAM with Skeleton

In order to consider a topological map in FastSLAM, we need to add it to the equations in a form that can be easily calculated. Let S be the skeletal map, then the FastSLAM factorization becomes:

$$p(x^t, m \mid z^t, u^t, S) = p(x^t \mid z^t, u^t, S) \prod_n p(m_n \mid x^t, z^t) \qquad \textbf{6-3}$$

We assume the map is independent of the skeleton given the robot's path. Thus, the occupancy grid mapping portion of FastSLAM is unchanged. Only the localization needs to take S into account.

$$p(x^t \mid z^t, u^t, S) = \eta p(z_t \mid x_t) \int_{x_{t-1}} p(x^t \mid u^t, x^{t-1}, S)\, p(x^{t-1} \mid z^{t-1}, u^{t-1}, S)\, dx_{t-1} \qquad \textbf{6-4}$$

Note that, because the map is independent of the skeleton, the skeleton does not affect the sensor readings z_t, which depend only on the robot's state, including the map.

The new motion model for localization is $p(x_t \mid u_t, x_{t-1}, S)$. However, it is not obvious how to sample from this model as required by MCL. Fortunately, given that the distance between x_t and x_{t-1} is small, we can factor the model into our original model and an additional term representing the motion probability of the motion given the skeleton map.

$$p(x^t \mid u^t, x^{t-1}, S) = p(S \mid x^t, u^t, x^{t-1})\, p(x^t \mid u^t, x^{t-1}) / p(S \mid u_t, x^{t-1}) \qquad \textbf{6-5}$$

Equation (**6-5**) can be greatly simplified using the Markovian assumption again. Also, notice that the same operation as in equation (**6-2**) produces the ordinary motion model in the second term, while still leaving the entire path for use with the skeleton map.

$$p(x^t \mid u^t, x^{t-1}, S) = \eta p(S \mid x^t)\, p(x_t \mid u_t, x_{t-1}) \qquad \textbf{6-6}$$

Having factored the motion model from the skeleton, all that remains is to convert $p(S \mid x^t)$ into a form that can be calculated.

$$p(x^t \mid u^t, x^{t-1}, S) = \eta p(x^t \mid S)\, p(S)\, p(x_t \mid u_t, x_{t-1}) / p(x^t) \qquad \textbf{6-7}$$

$$p(x^t \mid u^t, x^{t-1}, S) = \gamma\, p(x^t \mid S)\, p(x_t \mid u_t, x_{t-1}) \qquad \textbf{6-8}$$

This simplification also uses the assumption that all paths are equally likely and thus $p(x^t)$ is a constant. Putting equation (**6-8**) back into the localization formula of (**6-4**) results in localization which takes into account the skeleton map.

$$p(x^t \mid z^t, u^t, S) = \eta p(z_t \mid x_t)\, p(x^t \mid S) \int_{x_{t-1}} p(x_t \mid u_t, x_{t-1})\, p(x^{t-1} \mid z^{t-1}, u^{t-1}, S)\, dx_{t-1} \qquad \textbf{6-9}$$

The final equation indicates that the modification to the motion model can be considered to alter the weight of each particle. Thus, the localization step continues as normal while the probability of the particle's motion based on the skeleton map is multiplied with the sensor probability to determine the likelihood of the sample. The result will be to make particles which travel according to the skeleton more likely to be resampled than those which conflict.

6.2.4 Defining the skeleton model

In order to actually implement skeletal FastSLAM as defined in equation (**6-9**), we need to create a method of calculating the model $p(x_t \mid S)$. Fortunately, this model does not need to be sampled from as does the motion model, allowing more flexibility in its creation. Since the objective is to more highly weight paths the closer they are to the corridor, some type of probability based on the difference in angle between the path and the skeleton is the obvious choice. The model used is a Gaussian distribution centered at zero degrees mixed with a uniform distribution according to a gain value. The smaller the difference between the angle of the robot's path and the angle of the line

segment of the map, the more probable the particle. Of course, this only applies as the particle travels along the particular corridor. If the robot turns away from the corridor it is probably exploring some area not represented by the skeleton. In that case, the probability is a uniform distribution. Also, the current segment of the skeleton map that the robot is following is determined by which segment is closest. However, if the distance between the robot and the line is too great, then the probability is once again uniform, since a particle must be within a corridor to be affected by it. The result is a model that increases the probability of particles traveling along the skeleton and decreases the probability of those traveling at a tangent to it, while leaving those that are not following the skeleton unchanged. The model for particles within range of a skeleton line segment is illustrated in Figure 6-1. However, note that the actual values depend on the parameters selected for gain, variance, and threshold angle.

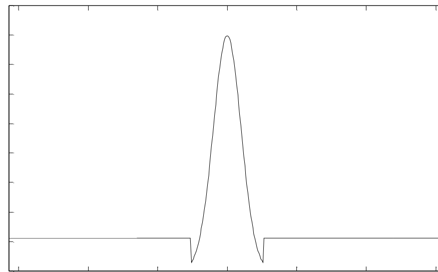


Figure 6-1: Skeleton map probability model.

6.2.5 Algorithm

Given the derivation in section 6.2.3 and the model from 6.2.4, the actual implementation of skeletal FastSLAM is relatively straightforward. In order to reduce errors caused by minor corrections in the robot's heading while it follows a corridor, linear regression is used to track the line which best fits the robot's path. The regression is restarted every time the robot changes its current closest line segment. Then, the difference between the robot's course and the direction of the skeletal line segment is simply the difference between the angle of two lines, a straightforward algebraic computation. With that angle, the skeleton map model can be evaluated and the only change in the algorithm in Table 2-4 occurs on line 3, which becomes $w_t^{[k]} = p(z_t \mid x_t^{[k]}, m) * p(x_t^{[k]} \mid S)$.

It is, of course, necessary to provide various parameters, notably the variance and gain of the skeleton model as well as the threshold distance for the robot to be within a corridor and the threshold angle for the model. However, most of these parameters depend on the physical features of the environment and good values can be determined by examining its structure. The threshold distance depends on the corridor width, while the threshold angle depends on the relative corridor angles. Finally, the gain depends on how well the environment is represented by the skeleton map. These values probably do not need to change between different environments or robots, unless there are radical differences in the map. Even then, producing the correct map will probably only require more particles. Of course, if the skeleton is completely incorrect, skeletal FastSLAM can produce no

improvement, and will probably need significantly more particles to converge than ordinary FastSLAM. The algorithm depends on having a reasonably accurate skeleton map.

Compared to ordinary FastSLAM, using a skeletal map adds runtime that is linear in the number of line segments in the skeleton. Since the topology of an indoor environment is usually fairly simple the increase in processing required is small. If skeletal FastSLAM can reduce the number of particles required for determining the correct map in an environment then the gains in processing time will more than offset the increase required to implement the algorithm.

6.2.6 Skeleton map parameters

With a little more work the skeleton map can be defined using parameters. By using the paths of all the particles, these parameters can be optimized to fit the skeleton to the data. After a long enough sequence of robot motions, a background optimization algorithm can be used to minimize the error between the paths and the skeleton line segments. Because the particle filter represents only the high probability paths, if the skeleton map lies along these paths, then it probably represents the data accurately. Using parameters for x and y offset, rotation and horizontal and vertical scale requires as many particles as ordinary FastSLAM at first, but once the skeleton is fitted, the set could be reduced.

Optimization is performed using a local simplex search algorithm [Lagarias et al. 1998], which is restarted at various parameter values to compensate for symmetry in the skeleton. Equation (**6-10**) shows the actual error measure used, which gives a diminishing weight as distance increases, so that data from where the robot is not following the skeleton does not unduly distort the results. The sum is over the distances between all past poses and the associated line segment.

$$error = \sum_{x} x^2 / (x+1)^2 \qquad \textbf{6-10}$$

6.3 Experimental Evaluation

In order to validate skeletal FastSLAM with occupancy grid maps it was tested against data sets gathered using a real robot in an indoor environment, as well as with various simulated data sets. The simulated data demonstrates the benefits of the algorithm in various situations, while the physical data sets show that it really does generate improvement in a real environment.

Loop closure is one of the major problems with FastSLAM and the skeletal algorithm is designed to reduce the processing required for loops in certain environments. The experiments were chosen to show the actual reduction provided by the skeleton in specific environments. From the results presented here we can determine that skeletal FastSLAM will provide a benefit in a wide range of circumstances where the fundamental assumption of fixed corridors applies. Since we cannot specifically test an algorithm's loop closure ability, we rely on tests of the minimum processing required to develop a map with the correct structure as determined by a human observer. Although this criteria is somewhat vague, there was no problem in making the decisions since the maps tended to either converge correctly or diverge to random nonsense. The minimum number of particles necessary to generate a correct solution was used to determine the minimum run time for each

algorithm. Since skeletal FastSLAM and ordinary FastSLAM require approximately the same amount of processing, the skeletal algorithm must converge on fewer particles to provide a benefit. Comparing the minimum run times for convergence proves the benefits of using the skeleton do not outweigh the extra processing required to compare particles to the skeleton map.

6.3.1 Normal FastSLAM Implementation

In order to evaluate the skeletal algorithm it was necessary to compare it to the behaviour of ordinary FastSLAM. To avoid differences caused by varying implementations of FastSLAM the implementation used was identical to the skeletal algorithm, except the $p(x_t^{[k]} \mid S)$ values are never calculated. Instead, a value of 1 is used which effectively reverses the change to line 3 of Table 2-4, re-establishing the original FastSLAM algorithm for occupancy grid maps. The implementation is based on the algorithm described in section 2.3.4.

At the beginning, a set of N particles is created, each one with a location of 0 for all location parameters (x, y, θ). Each particle also contains a map consisting entirely of unoccupied cells. As the robot moves it generates odometry readings u_t and sensor readings z_t. When the distance moved has passed a threshold, these values are processed into a new set of particles. A new position is drawn for each particle according to the motion model $p(x_t \mid x_{t-1}, u_t)$ using the odometry readings and the particle's current location. Then, the weight of each particle is determined from the sensor model $p(z_t \mid x_t)$ using that particle's own map and the current sensor readings. After calculating the weight, the sensor readings are then used to update the map separately for each particle according to the algorithm in section 2.1.3. Finally, the particles are resampled randomly, with replacement, according to their weights to generate a new uniformly weighted set of particles representing the current belief. This basic FastSLAM algorithm is used to provide comparison values for the skeletal algorithm. It is also the underlying implementation for skeletal FastSLAM, with the modifications as described in section 6.2.5.

6.3.2 Simulated data

Data sets generated from simulated environments test the basic behaviours of the skeletal algorithm in standard situations. One of the most basic environments is a wide, straight corridor. A 40 meter long corridor is typically a very difficult situation for FastSLAM because there is no indication as to the correct direction. A straight corridor gives almost exactly the same readings as one that curves slightly. Since the robot does not turn as it traverses the corridor there is no way for FastSLAM to correct the readings. Because of this it required a minimum of 210 particles for ordinary FastSLAM to produce the correct map using the data set. Compared to that, a single corridor is a very easy environment for skeletal FastSLAM, which required only 100 particles to converge. The run time for convergence of skeletal FastSLAM was 156 seconds for this data set, an improvement of 45% over regular FastSLAM's 283 seconds. Skeletal FastSLAM provides a serious advantage when an environment provides little information about global orientation.

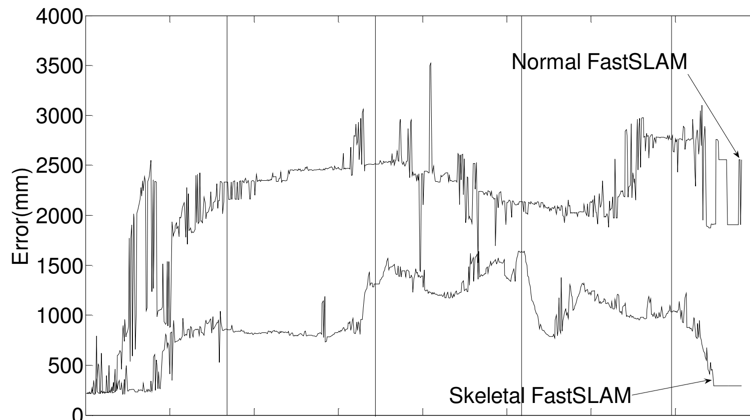


Figure 6-2: Error in position over time for normal FastSLAM vs. skeletal in a simple loop.

The next data set was a large loop around a 40 meter square. Because the turns allow the robot to see back along its course, this environment was easier for FastSLAM to handle. Ordinary FastSLAM required 100 particles to converge with the data, while the skeletal algorithm was successful with 30. The time for convergence of 181 seconds for skeletal FastSLAM was a 67% improvement over ordinary FastSLAM's 558 seconds. In Figure 6-2 we can see the results of this test. The vertical lines represent the corners in the environment and the robot reenters an area it has already traversed after the fourth vertical line. For normal FastSLAM the error drifts, generally increasing over time, until the final section where the loop is closed. At that point, the error drops abruptly. In contrast, skeletal FastSLAM tends to retain a relatively constant error, changing only at the corners of the map, marked by the vertical lines, where the skeleton algorithm does not apply. The error is so much smaller when the loop is closed that it quickly decreases back to almost 0. Normal FastSLAM only managed to converge by shifting the entire map, thus retaining a larger error. By reducing the error increase in the corridors, skeletal FastSLAM is able to correct the position much more quickly when the loop is finally closed. The simulated data indicates that the algorithm provides a major benefit in the situations where it applies and leaves more leeway for handling the remaining situations, such as the corners.

6.3.3 Real data

Figure 6-3: Two real environments with skeleton maps

Data from a 180 degree laser scanner mounted on a Pioneer 3Dxe differential drive near-holonomic robot was collected from two different real environments. Since there was no way to get the ground truth of the robot's position, I instead observed the minimum number of particles necessary for the map to converge to a representation which corresponded to the correct map. The primary observation about a correct map is that all of the loops are closed, connecting to the appropriate corridors. In practice, it was easy to determine if the map was correct, since an incorrect map diverged radically from the correct representation, becoming reduced to nonsense.

In the first environment in Figure 6-3, regular FastSLAM required at least 160 particles to succeed, while using the skeleton map only required 100. The 60% larger set of particles for ordinary FastSLAM is necessary because the environment contains many long loops. Without the skeleton, more particles are necessary to allow these loops to close properly. The runtime of skeletal FastSLAM was a 58% improvement over the ordinary algorithm, converging in only 466 seconds compared to 737.

The second environment only has a single corridor and the robot travels through two rooms. Although the path into the rooms is marked by the skeleton, the path between them is not. The greater area that is not represented by the skeleton, coupled with fewer loops, results in less of an improvement. Skeletal FastSLAM needed 110 particles to converge in this environment, while

ordinary FastSLAM needed 140, an increase of 27%. There was also an improvement of 23% in the runtime, with skeletal FastSLAM reducing the necessary time from 511 seconds to 391.

The improvements demonstrate that skeletal FastSLAM provides a significant improvement over ordinary FastSLAM, correctly converging with fewer particles, and thus less computation, using real data sets. Coupled with the simulated data the results show that using a skeleton map is an effective addition to FastSLAM.

Table 6-1: Experimental comparison of skeletal FastSLAM algorithm vs. original.

	Original		Skeletal		% improvement	
	particles	runtime	particles	runtime	particles	runtime
Simulated corridor	210	283.672s	100	156.329s	52.4%	44.9%
Simulated loop	100	558.078s	30	181.078s	60%	67.6%
Real environment 1	160	737.016s	100	466.938s	37.5%	36.6%
Real environment 2	140	511.047s	110	391.031s	21.4%	23.5%

6.4 Conclusion

FastSLAM is an effective solution to both the online and full simultaneous localization and mapping problem in indoor environments where individual features are hard to determine. However, it suffers from problems in loop closure which require progressively more particles as the size of loops in the environment increase. By adding an easily created skeletal map into the algorithm, it is possible to significantly obviate this problem, allowing the FastSLAM algorithm to solve local uncertainties while aiding it in closing loops. A skeleton map indicates the direction that the robot must be taking so that, instead of wasting particles on multiple divergent trajectories, the algorithm can concentrate them around the correct path, significantly reducing the need for additional particles. As the corridors increase in length, ordinary FastSLAM requires an increasing number of particles, while skeletal FastSLAM continues to require only enough for the local uncertainties, becoming independent of the overall size of the map.

The experiments I have performed indicate that the skeleton map is highly effective in reducing the number of particles necessary to achieve convergence. Using simulated data sets, we can observe the skeleton algorithm's behaviour in various fundamental situations. These results demonstrate how skeletal FastSLAM improves on ordinary FastSLAM in simple corridors and loops. By observing the results of real data sets, we can also see that the skeleton map provides a significant improvement in practice. Using a skeletal map is a low cost improvement to FastSLAM that is very useful in indoor environments whose overall configuration is known, even though the exact map may not be.

Skeletal FastSLAM, like dynamic map MCL, allows FastSLAM to handle situations with partial knowledge of the environment. Since initial knowledge no longer needs to be discarded the behaviour of the algorithm is improved. By allowing additional information to be applied in the FastSLAM algorithm I have created a technique that can be very effective in specific situations where

ordinary FastSLAM would require much more work. These methods lead to localization and mapping techniques that can generate a map and path from any type of starting information.

Chapter 7
Conclusion

One of the most important problems in robotics is localization, identifying a robot's location in the environment. It is necessary to solve localization before almost any useful task can be accomplished. There are many different localization techniques, from heuristic methods to algorithms based on formal mathematical derivations. These methods involve various tradeoffs between provable limits, processing required, accuracy, and the type of environment, robot, and sensors for which they will work. Pure localization algorithms require a pre-existing map of the area, while solutions to the simultaneous localization and mapping problem (SLAM) can generate a map simultaneously with the position data. One powerful technique that can be adapted for both localization and SLAM is called particle filtering. Monte Carlo localization (MCL) is an implementation of particle filters for localization while FastSLAM is an implementation for SLAM. Both of these techniques are straightforward to implement and provide an efficient solution in many situations. However, there are still drawbacks to using these techniques. Both of them rely on the environment having an unchanging map, whether it is known in advance or not. Also, the two techniques address the extremes of a completely known or completely unknown map, without being able to make use of partial information about the environment.

My research involves using the information provided by MCL when the robot is properly localized to determine additional information about the environment. Any additional information makes the map more accurate, allowing the behaviour of localization to improve. By reversing the trend for the map to become less representational over time, I hope to provide a localization algorithm suitable for long term deployment. Instead of requiring periodic modification by a skilled user, the robot itself will update its own information, eventually maintaining a representation of the environment more accurate than any human could define. This map will gradually alter itself as the environment changes so that even dynamic environments, where objects are periodically moved, will not require either special techniques or periodic map redefinitions. Making it possible for a robot to adapt to dynamic elements allows it to perform tasks with much less external input, permitting it to operate without requiring a teacher with special skills to manually modify its representation. Dynamic mapping is an important step in making robots more useful in real environments and for long term tasks.

A further benefit of dynamically improving a map is that localization can succeed even if some information about the environment is unknown. Since the map improves over time it eventually comes to be a fully accurate representation. All that is necessary is enough initial data so that localization can succeed at first. The self correcting map will converge to the environment, permitting localization to handle more error. This allows robots to be deployed in regions that are partially unknown, without requiring a SLAM solution to generate the map from uncertainty.

The map is not the only information that can be updated over time based on observations. It is also possible to update the parameters of the motion model to better correspond to the behaviour of the robot. These parameters can be stored separately for different regions of the map, further conforming the map to the environment. The better the convergence between the information in MCL and the

state of the world, the more leeway there is for handling errors. Using dynamic motion models updates the map with additional data that makes it a better representation. Usually, the map only represents the area in a form that is easy for humans to understand, but by automatically determining motion model parameters the map becomes a closer match to the world in terms of the robot's interaction with it. Improving the quality of the representation by adding new dimensions to the map allows MCL to operate with a better convergence even than using an optimal occupancy grid only map.

Symmetrical maps are a serious problem in global localization with MCL because of the problem of bias. Cluster MCL overcomes many of the problems of bias by forcing less probable locations to continue to exist, without compromising the highest probability location. It also provides a solution to the kidnapped robot problem by allowing random particles to be added that do not interfere with localization until a new position is found. The hierarchical technique of cluster MCL provides an improvement to MCL which allows it to better handle many problematic situations while still maintaining the underlying properties of the algorithm.

Another area of my research is allowing FastSLAM to make use of additional information. In many situations some data is available about the environment before localization begins. If the structure of the map is available, my algorithms allow FastSLAM to make use of it, converging much more easily than normal. Although MCL already handles the case where the environment is completely known, and FastSLAM handles the case where it is completely unknown, most situations are actually somewhere between the two. Skeletal FastSLAM takes advantage of the partial knowledge that is often available to offset one of the main disadvantages of regular FastSLAM, the loop closure problem. If skeletal information is available there is no reason not to use it, especially since there is little overhead to the algorithm. The ability to use additional information about the environment makes FastSLAM more powerful in many common situations.

My research has primarily been based on extending the effectiveness of particle filtering techniques for localization and SLAM in common situations. Although many of these situations can be handled by the ordinary techniques of MCL and FastSLAM, there are additional features involved that can significantly improve the results. The major insight in my research is that, although MCL requires the entire static map to be defined and FastSLAM uses no initial information at all, most localization problems provide something between these two extremes. I have created two techniques that work with partial knowledge of the map, depending on whether it is mostly known or mostly unknown. Also, cluster MCL handles cases with high symmetry while dynamic motion models allows more optimal parameters to be determined. All of these techniques provide significant improvements to the basic localization and mapping algorithms. By accommodating additional starting information and changing conditions, as well as allowing previously unknown features to be discovered, I hope to provide localization and mapping that is more effective in real environments.

7.1 Future Work

I intend to continue developing improvements to MCL and FastSLAM so as to handle additional situations with varying amounts of initial data. Also, I will perform research to improve the capability to dynamically alter parameters of the algorithms during operation, especially the map.

Eventually, I intend to produce a localization and mapping system that can operate efficiently with arbitrary amounts of starting data, from a full map to complete uncertainty. By taking dynamic elements into account this system will be able to handle any changes in the environment without losing any confidence. Allowing for different levels of starting data, this consolidated algorithm will be able to handle different levels of uncertainty in different areas, and so will be able to seamlessly travel between known regions and unexplored areas. Merging localization with mapping and detecting dynamic elements will allow a robot to be deployed to an area long term, without requiring human intervention. I hope to use this system to create an autonomous vending machine robot that will use dynamic elements of the map to detect and explore new regions while optimizing its distribution path. Such a robot would be an effective validation of my research, proving its utility for further development.

Bibliography

Austin, D. J. and P. Jensfelt (2000). Using multiple Gaussian hypotheses to represent probabilitydistributions for mobile robot localization. IEEE International Conference on Robotics and Automation (ICRA).

Avots, D., E. Lim, R. Thibaux and S. Thrun (2002). A probabilistic technique for simultaneous localization and door state estimation with mobile robots in dynamic environments. IEEE/RSJ International Conference on Intelligent Robots and System, 2002. .

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

Cheesman, P. and P. Smith (1986). "On the representation and estimation of spatial uncertainty." International Journal of Robotics **5**: 56-68.

Cox, I. J. (1991). "Blanche-an experiment in guidance and navigation of an autonomous robot vehicle." IEEE Transactions on Robotics and Automation **7**: 193-204.

Dellaert, F., D. Fox, W. Burgard and S. Thrun (1999). Monte Carlo localization for mobile robots. IEEE International Conference on Robotics and Automation.

Doucet, A. (1998). On Sequential Simulation-Based Methods for Bayesian Filtering. CUED/F-INFENG/TR. Cambridge, Cambridge University, Department of Engineering.

Elfes, A. (1987). "Sonar-based real-world mapping and navigation." IEEE Transactions on Robotics and Automation: 249-265.

Eliazar, A. and R. Parr (2004). DP_SLAM 2.0. ICRA. New Orleans, USA.

Eliazar, A. I. and R. Parr (2004). "Learning probabilistic motion models for mobile robots." ACM International Conference Proceeding Series.

Folkesson, J. and H. I. Christensen (2004). Graphical SLAM: A self-correcting map. ICRA.

Fox, D. (2003). "Adapting the sample size in particle filters through KLD-sampling." International Journal of Robotics Research **22**: 985-1003.

Fox, D., W. Burgard, F. Dellaert and S. Thrun (1999). Monte Carlo localization: Efficient position estimation for mobile robots. AAAI. Orlando, FL.

Fox, D., W. Burgard and S. Thrun (1999). "Markov localization for mobile robots in dynamic environments." Journal of Artificial Intelligence Research **11**(3): 391-427.

Hahnel, D., R. Triebel, W. Burgard and S. Thrun (2003). Map building with mobile robots in dynamic environments. IEEE International Conference on Robotics and Automation (ICRA).

Isard, M. and A. Blake (1998). "CONDENSATION: conditional density propagation for visual tracking." International Journal of Computer Vision **29**: 5-28.

Jensfelt, P. and S. Kristensen (2001). "Active global localization for a mobile robot using multiplehypothesis tracking." IEEE Transactions on Robotics and Automation **17**(5): 748-760.

Jordan, M. I., Z. Ghahramani, T. S. Jaakkola and L. K. Saul (1999). "An Introduction to Variational Methods for Graphical Models." Machine Learning **37**(2): 183-233.

Julier, S. and J. Uhlmann (1997). A new extension of the Kalman filter to nonlinear systems. International Symposium on Aerospace/Defense Sensing, Simulate and Controls. Orlando, FL.

Kaboli, A., M. Bowling and P. Musilek (2006). "Bayesian Calibration for Monte Carlo Localization." Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI): 964–969.

Lagarias, J. C., J. A. Reeds, M. H. Wright and P. E. Wright (1998). "Convergence properties of the Nelder-Mead simplex algorithm in low dimensions." SIAM Journal on Optimization **9**(1): 112-147.

Lenser, S. and M. Veloso (2000). Sensor resetting localization for poorly modelled mobile robots. ICRA.

Leonard, J. J. and H. F. Durrant-Whyte (1991). "Mobile robot localization by tracking geometric beacons." IEEE Transactions on Robotics and Automation **7**(3): 376-382.

Liu, J. S. and R. Chen (1998). "Sequential Monte Carlo Methods for Dynamic Systems." Journal of the American Statistical Association **93**(443): 1032-1044.

Lu, F. and E. Milios (1997). "Globally Consistent Range Scan Alignment for Environment Mapping." Autonomous Robots **4**(4): 333-349.

Milstein, A. (2005). Dynamic Maps in Monte Carlo Localization. The Eighteenth Canadian Conference on Artificial Intelligence.

Milstein, A., J. N. Sanchez and E. T. Williamson (2002). Robust global localization using clustered particle filtering. Proceedings of AAAI/IAAI**:** 581–586.

Milstein, A. and T. Wang (2006). Localization with Dynamic Motion Models. International Conference on Informatics in Control, Automation, and Robotics (ICINCO).

Montemerlo, M., S. Thrun, D. Koller and B. Wegbreit (2002). FastSLAM: A factored solution to the simultaneous localization and mapping problem. Proceedings of the AAAI National Conference on Artificial Intelligence**:** 593–598.

Montemerlo, M., S. Thrun, D. Koller and B. Wegbreit (2003). FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI).

Montemerlo, M., S. Thrun and W. Whittaker (2002). Conditional particle filters for simultaneous mobile robot localization and people-tracking. IEEE International Conference on Robotics and Automation (ICRA).

Moravec, H. P. (1988). "Sensor Fusion in Certainty Grids for Mobile Robots." AI Magazine **9**(2): 61-74.

Schulz, D., W. Burgard, D. Fox and A. B. Cremers (2001). Tracking multiple moving targets with a mobile robot using particle filters and statistical data association. ICRA.

Schulz, D. and D. Fox (2004). Bayesian color estimation for adaptive vision-based robot localization. Intelligent Robots and Systems, (IROS).

Simmons, R., J. L. Fernandez, R. Goodwin, S. Koenig and J. O'Sullivan (2000). "Lessons learned from Xavier." IEEE Robotics & Automation Magazine **7**(2): 33-39.

Simmons, R. and S. Koenig (1995). Probabilistic robot navigation in partially observable environments. International Joint Conference on Artificial Intelligence**:** 1080–1087.

Smith, R., M. Self and P. Cheeseman (1990). "Estimating uncertain spatial relationships in robotics." Autonomous Robot Vehicles **1**: 167-193.

Smith, R. C. and P. Cheeseman (1986). "On the Representation and Estimation of Spatial Uncertainty." The International Journal of Robotics Research **5**(4): 56.

Thrun, S. (1998). "Bayesian Landmark Learning for Mobile Robot Localization." Machine Learning **33**(1): 41-76.

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

Thrun, S. (2000). "Monte Carlo POMDPs." Advances in Neural Information Processing Systems **12**: 1064–1070.

Thrun, S. (2000). "Probabilistic Algorithms in Robotics." AI Magazine **21**(4): 93-109.

Thrun, S. (2002). Particle filters in robotics. 17th Annual Conference on Uncertainty in AI (UAI).

Thrun, S. (2002). Robotic Mapping: A Survey, School of Computer Science, Carnegie Mellon University.
Thrun, S., W. Burgard and D. Fox (2005). Probabilistic robotics, MIT Press.
Thrun, S., D. Fox and W. Burgard (2000). Monte carlo localization with mixture proposal distribution. AAAI National Conference on Artificial Intelligence**:** 859–865.
Thrun, S., D. Fox, W. Burgard and F. Dellaert (2001). "Robust Monte Carlo localization for mobile robots." Artificial Intelligence **128**(1-2): 99-141.
Thrun, S., J. Langford and V. Verma (2002). "Risk Sensitive Particle Filters." Advances in Neural Information Processing Systems **14**.
Thrun, S., Y. Liu, D. Koller, A. Y. Ng, Z. Ghahramani and H. Durrant-Whyte (2004). "Simultaneous Localization and Mapping with Sparse Extended Information Filters." The International Journal of Robotics Research **23**(7-8): 693.
Weiss, G., C. Wetzler and E. von Puttkamer (1994). Keeping track of position and orientation of moving indoor systemsby correlation of range-finder scans. Intelligent Robots and Systems (IROS).

Appendix A
Standard Mathematical Definitions

These are the standard definitions used throughout this document.

Table A-1: Standard Mathematical Definitions

Symbol	Definition
x	Robot location
(x, y, θ)	A position given by two offsets and an orientation.
u	Robot odometry data
z	Robot sensor data at
a_t	Value of a taken at time = t
a^t	$\{a_0, a_1, \dots, a_{t-1}, a_t\}$ All values of a from time = 0 to time = t
M, y	Map data
m_n	Map cell as identified by index n
$y_{k,t}$	Map cell identified by index k at time = t
$Bel(x_t)$	Belief state of the variable x at time = t
η	A normalization constant
x_t^i	A specific particle representing a hypothesis for the variable x at time t.
w_t^i	The importance weight of particle i
N	The total number of particles
$[n]$	The index of a random particle
θ	The robot's orientation
r	The distance travelled
$\underline{a}$	An estimate of the value of a.
k_r, k_θ, k_d	Parameters of the standard motion model
Σ	Covariance matrix
X_t	A set of particles representing $Bel(x_t)$