University of Alberta

BUILDING PROBABILISTIC MOTION MODELS FOR SLAM

by

Artit Visatemongkolchai 

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Fall 2007

Your file Votre référence
ISBN: 978-0-494-33366-2
Our file Notre référence
ISBN: 978-0-494-33366-2

Abstract

This thesis describes the use of two sequential machine learning techniques for building a mobile robot's motion model, which is an essential component in SLAM algorithms. For building a static motion model, we apply the recursive least squares (RLS) algorithm to learn motion model parameters as soon as new data (the robot's pose from odometry readings and ground-truth poses) arrive. For building a dynamic motion model, our framework uses the bi-loop recursive least squares (BiRLS) algorithm to learn the parameters on the fly as the robot traverses the environment. These techniques for building motion models are integrated with FastSLAM 2.0 giving increased autonomy to the system by eliminating the human effort required to produce motion models. The performance of our newly acquired motion models are then evaluated objectively based on the robot's ground-truth poses in the context of FastSLAM 2.0.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Before robots can reach to the level of being common assistants and workers for their human owners as shown in the movie "I, Robot", by Ridley Lavine [1], they must be able to autonomously operate in unknown environments over an extended period of time.

Advances in the area of Simultaneous Localization and Mapping (SLAM) have come a long way towards bringing the prospect of truly autonomous mobile robots closer to reality [53], [22]. As the name implies, SLAM enables a mobile robot to simultaneously estimate a map of its environment and its pose relative to that map. Many solutions to the SLAM problem have been introduced over the past decade, such as Extended Kalman Filters (EKF) SLAM [49], FastSLAM [43], GraphSLAM [52], Sparse Extended Information Filters (SEIF) SLAM [51]. There are some who believe that we are now at the stage where it is hard to say whether SLAM is still an open field because many theoretical aspects have been mathematically solved. Having said that, there are still many open issues that could be further investigated to improve SLAM's accuracy and decrease complexity.

One such issue is to automatically learn a robot's motion model, which is an essential component in SLAM algorithms. The model is generally provided by a human based upon a combination of intuitions about the environment and experience with the robot. This could pose real difficulties for those who are new in this field or have never worked with particular robots before. It is generally sufficient to define a static model if we plan to deploy our robot in an environment for which we have good knowledge about its properties (e.g. ground surface), and if we do not expect these properties to change while the robot operates. However, a static model may not be sufficient for the robot to operate in the real world, which can be complex, unstructured, and dynamic. In this case, a static model is only a rough approximation. Therefore, it is worthwhile to develop a system that can learn the model as the robot operates.

There are two main contributions in this thesis. First, we describe the use of two sequential machine learning techniques to learn the robot's motion model. The recursive least squares (RLS) is used to build a static motion model. RLS can learn motion model parameters as soon as new data (odometric readings and ground-truth poses) arrive. Doing so allows the algorithm to detect when the model has actually converged and stop training. This is convenient because we can program

our robot to traverse the environment, collect data, and decide by itself when to stop. Moreover, we use the robot's poses reported from the FastSLAM 2.0 algorithm [44] as estimated ground-truth poses in order to avoid the need for external measurements and explicit requirements of ground-truth positions. In addition, the bi-loop recursive least squares algorithm (BiRLS) is applied to learn motion model parameters when the robot operates in a dynamic environment (i.e. environment with changing ground surfaces) because BiRLS keeps track of the changes in time-varying parameters better than RLS and is better suited for building a dynamic motion model. The use of sequential machine learning techniques and the use of applying the robot's poses from FastSLAM 2.0 as estimated ground-truth poses have the potential to significantly increase the autonomy of mobile robots by eliminating the human effort required to build the motion model. The model can be built using only on-board sensors. The performance of our newly acquired motion models are then evaluated objectively with the robot's ground-truth poses in the FastSLAM 2.0 framework.

Second, we study the sensitivity of the FastSLAM 2.0 algorithm to the motion model. As is known, the more accurate the motion model, which generates the proposal distribution, is, the more accurate the SLAM and localization algorithms. However, the FastSLAM 2.0's improved proposal distribution is not only generated from the motion model but also takes the most recent sensor measurement into consideration. Given accurate measurement, it is expected that the improved proposal distribution would be much more accurate (i.e. cover most of the areas representing the robot's true pose) compared to the proposal distribution generated merely from the motion model. Therefore, in our case, it is less obvious whether the accuracy of the motion model would still play an important role in the quality of the improved proposal distribution as accurate sensor measurements could compensate for any incorrect estimations from the motion model. This thesis seeks to answer this question.

This thesis is organized as follows. In next chapter we provide a brief summary of SLAM, motion models, and other related topics together with a brief survey of other work that have been done in automatic acquisition of motion models for mobile robots. We then discuss in Chapter 3 about specific algorithms, FastSLAM 2.0, the recursive least squares, and the bi-loop recursive least squares. In Chapter 4, our data gathering and parameter estimation processes are described. Chapter 5 is devoted to the discussion on our experimental results. Chapter 6 presents conclusions and discusses future directions to extend this thesis.

Chapter 2

Background

2.1 Introduction

This chapter presents some of the topics that form the background to this thesis. First we briefly discuss SLAM, which is a fundamental problem in mobile robotics and artificial intelligence (AI). We then discuss an odometry motion model and its role to SLAM, together with possible sources of odometry errors. Next, we explain the differences between traditional batch learning and sequential learning, which is the core idea of building motion models in this thesis. The later sections of this chapter review other work that has been done in automatic acquisition of a mobile robot's motion model and discuss the contributions reported in this thesis.

2.2 Simultaneous Localization and Mapping

Simultaneous Localization and Mapping (SLAM) is a fundamental problem and a highly active area in mobile robotics and artificial intelligence (AI) communities. The objective of SLAM is simple and direct but rather challenging to solve; how can a mobile robot operate in an unknown environment using only on-board sensors to simultaneously localize itself and build a map of its environment?

SLAM is considered a complex problem; a robot needs a consistent map to successfully localize itself to the environment and, at the same time, requires a good estimate of its pose to build that consistent map. This mutual dependency between the pose and the map estimates makes the SLAM problem hard [27],[9]. As a robot builds a map, the landmark location errors are dependent on the robot's pose error. As the robot localizes itself in this map, its pose estimate is dependent on the landmark error. These errors lie in the uncertainty inherent in the robot's interactions with the real world through noisy sensors and actuators [2].

Advances in SLAM are key components in providing mobile robots with the ability to operate with real autonomy. Before a robot can perform any useful applications, such as rescuing victims from collapsed buildings, it must first know where it is in the environment. At the current stage, SLAM has been formulated and solved as a theoretical problem in many different forms and has been employed in many practical autonomous vehicles, such as unmanned aerial vehicles, autonomous

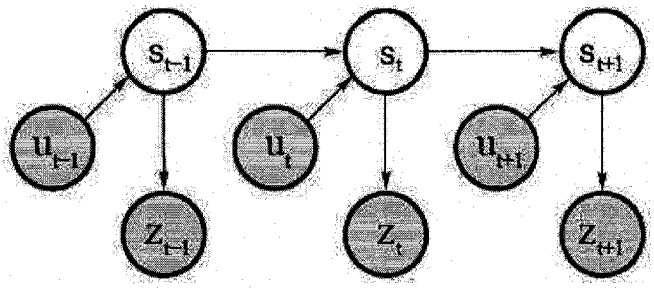

Figure 2.1: The Bayes network that characterizes the evolution of controls u, states s, and measurements z. Image modified from [53]

underwater vehicles, autonomous ground vehicles and household robots.

In brief, the SLAM problem is generally solved using a recursive Bayes filter. The belief distribution at time t represents the robot's pose and landmark positions, called the posterior distribution, is computed from the previous belief distribution at time $t-1$, along with the most recent control u_t and the most recent measurement z_t. The key assumption of SLAM is the Markov assumption that past and future data are independent if we know the current state of the robot.

The Bayes filter algorithm possesses two essential steps: the prediction and the update. The prediction step computes the state transition probability, called the proposal distribution, and is generated by a probabilistic motion model. This distribution specifies the probability that the robot arrives at the location s_t given that it started at the location s_{t-1} and performed action u_t. The update step computes the measurement probability, called the likelihood distribution, and is generated by a probabilistic sensor model. This distribution represents the probability of receiving a particular sensor reading given robot's current position s_t. The proposal and the likelihood distributions describe the dynamic stochastic system of the robot and its environment as shown in Figure 2.1. The state at time t is stochastically dependent on the state at time $t-1$ and the control u_t. The measurement z_t depends stochastically on the state at time t.

The robot's pose is estimated as a distribution because the robot environments are inherently unpredictable. While the degree of uncertainty is small in an empty small room, many environments, such as the outside world, residential homes, or even on Mars, are highly dynamic and unpredictable. Moreover, sensors are limited in what they can receive. Most common laser range finders provide small and constant errors in range and bearing [1] only up to 30 meters. Stereo cameras report accurate bearing information at the cost of poor range information, while sonar sensors generally provide accurate range but poor bearing information. Obviously, these sensor measurements are also subject to noise. Moreover, robot actuator consists of motors with odometers that are also unpredictable.

[1] A bearing information is the clockwise angle between a reference direction (i.e. the robot) and the direction to an object (i.e. the landmark)

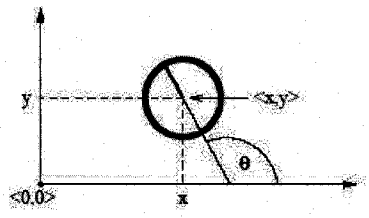

Figure 2.2: Robot pose shown in a global coordinate system. Image obtained from [53].

Uncertainty arises from many sources, such as control noises, the power of the batteries, and the inflation and wear of tires. It is important that a robot can handle these uncertainties so that it can operate with real autonomy in the environment.

2.3 Motion model

A probabilistic motion model plays an essential role in the prediction step of the Bayes filter as mentioned earlier. The main goal of the motion model is to capture the relationship between a control input to the robot and a change in the robot's configuration. There are typically two specific types of motion models: velocity motion model and odometry motion model. The first model assumes that the motion data u_t specifies the velocity command given to the robot's motor. The second model assumes that one has access to odometry information.

This thesis only focuses on the odometry motion model. Most commercial and educational robots are equipped with wheel encoders, which are suitable for odometry motion models. In practice, the odometry motion model tends to be more accurate than the velocity motion model because most commercial and educational robots do not execute velocity commands with the level of accuracy that can be obtained by measuring the revolution of the robot's wheels [53].

The odometry model uses the relative motion information, as measured by the robot's internal odometry. More specifically, in the time interval $(t-1, t]$, the robot advances from the pose s_{t-1} to the pose s_t. The pose of a mobile robot operating in a plane is shown in Figure 2.2. It comprises its two-dimensional planar coordinates relative to an external coordinate frame (x, y), along with its angular orientation θ. The odometry reports back to user a related advance from $s_{t-1} = (\bar{x}, \bar{y}, \bar{\theta})^T$ to $s_t = (\bar{x}', \bar{y}', \bar{\theta}')^T$, where the bar indicates that these are odometry measurements embedded in a robot internal coordinate whose relation to the global world coordinates is unknown . The odometry motion model then seeks to determine $p(s_t|s_{t-1}, u_t)$, where $u_t = (d, t)$ is given by translational movement and rotational movement respectively.

Without loss of generality, we assume that the true motion of the robot can be described by two independent normal distributions arising from translational and rotational movements. Generally, this assumption seems to be a good approximation for our robot's motion and is also very convenient

to sample and optimize. It is also commonly applied in the SLAM community [53] [22], [10], [41], [32]. Specifically, here is the motion model used in this thesis;

$$x' = x + Dcos(\theta + \frac{T}{2}) \tag{2.1}$$

$$y' = y + Dsin(\theta + \frac{T}{2}) \tag{2.2}$$

$$\theta' = \theta + T; \theta' \in (-\pi, \pi) \tag{2.3}$$

Given that $N(a, b^2)$ refers to a normal distribution with the mean a and the variance b^2, the true values of the translational movement D and the rotational movement T are defined as follows:

$$D \sim N(p_1 \cdot \bar{d} + p_2 \cdot \bar{t}, p_3 \cdot \bar{d}^2 + p_4 \cdot \bar{t}^2) \tag{2.4}$$

$$T \sim N(p_5 \cdot \bar{d} + p_6 \cdot \bar{t}, p_7 \cdot \bar{d}^2 + p_8 \cdot \bar{t}^2) \tag{2.5}$$

The translational and rotational movement according to the odometry are denoted as $\bar{d}$ and $\bar{t}$ respectively. p_i is a set of motion model parameters that need to be identified. As stated in (2.4) and (2.5), the true values of the translational movement D and the rotational movement T can each be represented by a normal distribution given the reported values, $\bar{d}$ and $\bar{t}$. The means scale linearly with both $\bar{d}$ and $\bar{t}$ while the variances scale with both $\bar{d}^2$ and $\bar{t}^2$. The variance terms indicate that the uncertainties in both D and T are related to the reported values $\bar{d}$ and $\bar{t}$. For example, a robot that is reported to have traveled in a straight line for 1 meter is expected to have a higher potential change in the facing angle compared with a robot with 0.1 meter of reported motion.

It is worth mentioning that the motion model used in this thesis is not the only motion model existing in the literature, and the choice of the motion model should be best representing the robot's true motion. For example, if the turn and drive commands were performed independently by our robot and the robot can travel in a straight-line, we would use a different motion model. Specifically, we would use the motion model proposed by [46]. On the other hand, the model used in this thesis accounts for simultaneous turning and driving [22] with constant velocity across the measured interval. When the robot turns and drives simultaneously, the distance traveled will actually be an arc, and not a straight line. Our motion model approximates [2] the arc by assuming that the robot sequentially performs half of its rotation, all of its translation, and then the other half of its rotation [10]. This motion model also assumes that the robot can only move in the heading direction it is facing (i.e. the robot cannot move sideways).

[2]The approximation error is expected to be small, especially if we break the robot's motion into small increments. The approximation error can then be absorbed as part of the noise.

Systematic errors	Non-Systematic errors
1) Unequal wheel diameters 2) Average of both wheel diameters differs from nominal diameter 3) Uncertainty about the effective wheelbase (due to non-point wheel contact with the floor) 4) Misalignment of wheels 5) Limited encoder resolution 6) Limited encoder sampling rate	1) Travel over uneven surfaces 2) Travel over unexpected objects on the surface 3) Wheel slippage: Slippery floors, over-acceleration, fast turning, external forces (human etc), internal forces (castor wheel etc)

Table 2.1: Odometry errors on differential-driven mobile robots

2.4 Odometry error

The robot's odometer reports the amount of translational and rotational movements at each time step. In a typical differential-driven mobile robot, these amounts are calculated from wheel encoders. Each motor is mounted onto each drive wheel to count the wheel revolutions. For example, when the robot traverses in a straight-line motion, the encoders reading should be identical for each wheel.

Although odometry information provides a system with the ability to calculate the robot's pose that is easily and cheaply implemented, it is still subject to systematic and non-systematic errors. Systematic errors are caused by imperfections in the design and mechanical implementation. They do not usually change during a normal run and tend to increase as time goes on. Non-systematic errors, on the other hand, are errors caused by interaction of a robot with unpredicted features of the environment, such as floor surface's irregularities, wheel-slippage or possibly batteries' power. We believe that it is impossible to determine all possible non-systematic errors in advance because they are entirely dependent upon the environment in which the robot operates. Table 2.1 shows the most common systematic and non-systematic errors for typical differential-drive robots [13].

As reflected in the literature, the two most notorious systematic errors with differential-driven mobile robots are unequal wheel diameters and uncertainty about the effective wheelbase [13]. Many mobile robots use rubber tires, which are difficult to manufacture to exactly the same diameter, to improve traction. Even if they are perfectly manufactured, they compress differently under asymmetric load distribution. These consequences can cause unequal wheel diameters. Moreover, the wheelbase is defined as the distance between the contact points of the two drive wheels of differential-drive robots; the distance must be known to compute an amount of rotation. Uncertainty in the effective wheelbase is caused by the fact that rubber tires contact the floor over an area, not a specific point. Unequal wheel diameters affect only straight-line motion while the uncertainty about the wheelbase affects only rotational movement of the robot [23].

There are a variety of techniques available to calibrate mobile robot odometry and model its error. UMB benchmark [12] is one of the popular techniques, which involves driving a robot clockwise

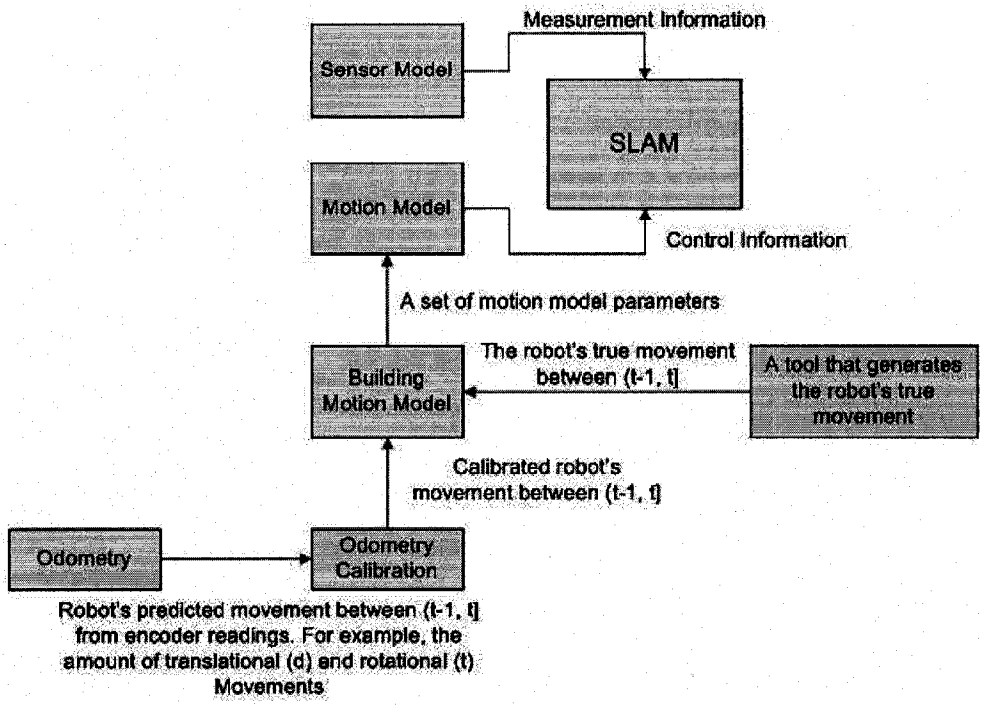

Figure 2.3: The roles of the odometry calibration process and the motion model acquisition process in the SLAM framework. The objective of the odometry calibration is to provide the best approximate in the robot's movement between (t-1,t]. The objective of the motion model acquisition is to estimate the model parameters based on the amount of movements reported from the odometry and the true amount of movements in order to build a probabilistic motion model for SLAM and localization algorithms

and counter-clockwise several times around a 4x4 meter closed path using a wall as a reference. This benchmark is mainly to calibrate the two most notorious systematic error sources: the unequal wheel diameter and the uncertainty about the wheel base. The authors claimed that their technique yields near optimal result. However, this method assumes a smooth surface for calibration and would be difficult for a robot with more than two drive wheels. Goel et al. [26] measured the actual velocities of the wheels and the velocity measurement from the encoders by putting the robot on the box, which allows the wheels to rotate freely in the air. By doing so, they found a relationship between the velocity returned by the encoders and the actual velocity. There are also many different techniques with different assumptions of modelling non-systematic errors expressed in terms of covariance matrices [33], [39], [40]. All these techniques calibrate and model odometry errors in constrained environments. If the underlying source of error changes, these techniques will need to be applied again.

These techniques of calibrating odometry are out of the scope of this thesis. We mainly focus on the work in automatically building a motion model for use in SLAM and localization algorithms. The difference between the odometry calibration and the motion model acquisition is illustrated

in Figure 2.3. The objective of the odometry calibration is to provide the best approximate in the robot's movement from $s_{t-1} = (\bar{x}, \bar{y}, \bar{\theta})$ to $s_t = (\bar{x}', \bar{y}', \bar{\theta}')$, and solutions to the odometry calibration generally depend on the robot's locomotion. For example, an efficient way to characterize the odometry error for differential-driven robots is obtained by modelling the error in each wheel separately [15], which may not be the case for that of synchronous mobile robots. On the other hand, the objective of the motion model acquisition process is to estimate the parameters, p_i in (2.4) and (2.5), based on the amount of movements reported from the odometry and the true amounts in order to build a probabilistic motion model for SLAM and localization algorithms. Estimating motion model parameters is essential for SLAM and localization algorithms regardless of the accuracy of the odometry, and solutions to the motion model acquisition process do not depend on the robot's locomotion.

We find that identifying motion model parameters (building a motion model) is an important issue but is infrequently discussed. Before we can practically run SLAM, we need to identify these parameters. Many motion models are hand tuned to generate proposal distributions that overestimate the posterior. Although overestimating the variance of the proposal distributions is generally a less serious problem than underestimating one [10], it may increase the error in terms of robot pose estimates and map estimates.

2.5 Static and dynamic motion models

Throughout this thesis, a static motion model refers to the motion model whose parameters are calibrated to best fit one particular environment and do not change over time. On the other hand, a dynamic motion model allows its parameters to adapt to changes as a robot operates in the environment. In order to determine parameters[3] of both motion models, accurate and reliable ground-truth poses are required. Parameters in the static model can be learnt both online and offline, while those in the dynamic motion model can only be learnt online.

The static motion model is sufficient if we expect to deploy our robot in the same, or similar, environment with low non-systematic errors. The model can generally be used with high confidence due to large amounts of useful training data. The dynamic motion model, on the other hand, is useful when we plan to deploy the robot in environments with different conditions in different areas, such as an environment with different ground surfaces. It is also useful in situations where we expect the robot's physical properties to change over time or in the environments with high non-systematic errors. In these scenarios, the static motion model is only a rough approximation.

[3] A motion model can be either parametric, which is the type we use in this thesis, or non-parametric.

Batch learning proceeds as follows:	Sequential learning proceeds as follows:
1) Initialize the weights	1) Initialize the weights
2) Process all the training data	2) Repeat the following steps: - Process one training case - Update the weights
3) Update the weights	

Table 2.2: Batch learning v.s. sequential learning

2.6 Learning characteristics

There are many ways to categorize learning methods. The distinctions are overlapping and can be confusing. For consistency, we will discuss the terminology used throughout this thesis. We first distinguish between batch and incremental learning. Batch learning applies when we process all the training data at once to learn the weights, while sequential learning applies when we process each training data case (not necessary one training data point) one by one as shown in Table 2.2.

2.7 Related work

Previous work in learning probabilistic motion model parameters for SLAM and localization algorithms has been fairly sparse. For building a static motion model, Eliazar and Parr [22] apply the weighted least squares technique to learn model parameters, which was a starting point to the techniques represented in this paper. Their technique is done offline by collecting the odometric readings together with the pose estimates given by their DP-SLAM 2.0 algorithm [21] as estimated ground-truth poses. They then use these data to learn their motion model parameters from multiple particles at each time step. The reported amount of odometric movements (translational and rotational) is identical for all particles at each time step, but each particle has its own amount of movements calculated from DP-SLAM 2.0. They also introduce a diagonal weight matrix with diagonal elements as the importance weight of each particle. Doing so allows their learning algorithm to determine how much each particle influences the final parameter estimates. They then showed the resulting map generated from DP-SLAM 2.0 with the use of their improved motion model to validate their approach. Kaboli, Bowling, and Musilek [32] present an offline calibration technique for both motion and sensor models for Monte Carlo Localization (MCL) from a Bayesian perspective. Specifically, given odometry and measurement data D and a prior distribution (initial parameter values) over model parameters $P(\Theta)$, the goal is to be able characterize and sample from the distribution $P(\Theta|D)$. As this posterior distribution does not have a simple closed form, they employ Markov chain Monte Carlo techniques (Gibbs sampling and Metropolis sampling) to sample from this posterior. Moreover, they described three different methods for incorporating the parameter samples drawn to the MCL. The first two methods select a single vector of model parameters using either the mean of the samples or the maximum posterior sample. The third novel method approxi-

mates the proposal distribution and the likelihood distribution with the integrations of the unknown model parameters as $P(s_t|s_{t-1}, u_t, D)$ and $P(z_t|s_t, D)$ respectively. They then demonstrated the effectiveness of their calibration technique compared to other motion and sensor models both in simulation and on a physical robot.

For building a dynamic motion model, Beeson, Murarka and Kuipers [10] use robot pose estimates from MCL as estimated ground-truth poses and learn motion model parameter with the weighted least squares as proposed by [22]. In order to evaluate the performance of different motion models, they use the global metric map to provide ground-truth poses of actual robot's poses at every sensor reading. Specifically, they took a trace and then ran their mapping algorithm offline to get a map. Therefore, they know the ground-truth poses at all locations in the trace, especially these ground-truth poses were used to create the map. They then took each of these ground-truth poses and calculated the next robot's pose according to the MCL algorithm. The resulting pose was compared with the next ground-truth pose to calculate the localization error.

Milstein and Wang [41] introduce a simple non-linear optimization technique to learn model parameters. Their technique is applied online. However, they do not learn motion model parameters based on the reported amount of movements from the odometry. They rather learn the model parameters using the reported amount of movements from the motion model and the estimated true amount of the robot's movements from MCL. Moreover, they evaluated the performance among different motion models by comparing the mean of the particles generated from each motion model to one generated from MCL (with the use of each particular motion model) at each time step. In other words, they did not compare the accuracy of MCL with the use of different motion models to a common robot's ground-truth path that is independent to their motion models. They simply assume MCL could successfully localize their robot regardless of the choice of their motion models. Both techniques of building a dynamic motion model mentioned here require a predetermined threshold to indicate when their motion model parameters shall be updated. For example, Beeson Murarka and Kuipers waited to learn the parameters until the robot's cumulative displacement is at least 1 meter and its cumulative rotation is at least 10 degrees from the last learning event.

It is worth noting that Beeson Murarka and Kuipers [10] concluded that the localization accuracy of the MCL with the use of one particular motion model outperformed the others using the standard paired t-test with 99% confidence. Similarly, Eliazar and Parr [22] simply showed the resulting map generated from the DP-SLAM 2.0 algorithm with the improved motion model and claimed that hand-tuned model would not generate a map with this level of accuracy. Both of these works did not give details as to how much of the improvement their motion model actually was compared to the others in term of the localization accuracy or the accuracy of landmark estimates. In other words, they did not provide evidence that there was really a need to build an accurate motion model for their SLAM and localization algorithms. We may not necessary need to do all the work, which could be computationally expensive, to identify the motion model parameters if the newly acquired motion

model can only slightly, for example less than 5%, improve the accuracy of SLAM or localization algorithms. In other words, we may not need to build a good motion model if it does not play an important role in our SLAM and localization algorithms.

Table 2.3 to Table 2.5 at the end of this chapter summarize previous works, together with what have been done in this thesis in the area of automatic acquisition of motion models for SLAM and localization algorithms.

2.8 Contributions

There are two main contributions in this thesis. First, we describe the use of two effective sequential learning techniques to learn motion model parameters. Experimental results indicate fair statistical evidence of our models in different environment settings. Second, we provide a comprehensive study on a role of the motion model in the FastSLAM 2.0 framework. A more detailed summary of the contributions reported in this thesis follows:

2.8.1 1. Effective sequential learning algorithms for acquiring static and dynamic motion models

Previous work by Eliazar and Parr [22] and by Kaboli, Bowling, and Musilek [32] acquired static motion models using batch learning. An important question arises in batch learning: what is the right amount of training data for our system? This issue is important because odometry is not always reliable [4] due to many sources of non-systematic errors. One extreme example is wheel slippage: If one or more wheels were to slip, then odometric readings at that time period could not be considered as useful data. Having many of these unreliable data may require a large amount of data to build a reliable motion model. In other words, we may need a large amount of useful data to compensate for unreliable data. As a result, we normally collect large amounts of training data that we hope are sufficient to build a reliable motion model. The motivation of applying a sequential learning technique is to be able to detect whether we have enough useful data to offset for unreliable data so that we do not need to collect more data, which could be difficult and time-consuming. In the case that the learning system can detect whether the motion model has converged (i.e. the difference between each pair of parameter obtained at time $t-1$ and at time t is smaller than a predetermined threshold), we can stop collecting more data.

In this thesis, the recursive least squares (RLS) is used to build a static motion model. Because RLS is a sequential learning algorithm, it can adjust motion model parameters as soon as new data (odometric readings and ground-truth poses) arrive. Therefore, RLS can, in some cases, detect when the model has converged and stop training. We also use the robot's poses reported from the FastSLAM 2.0 algorithm as estimated ground-truth poses in order to avoid the need for explicit requirements of ground-truth poses. This framework, as shown in Figure 2.4, is convenient because

[4] This is also a reason why a motion model is important.

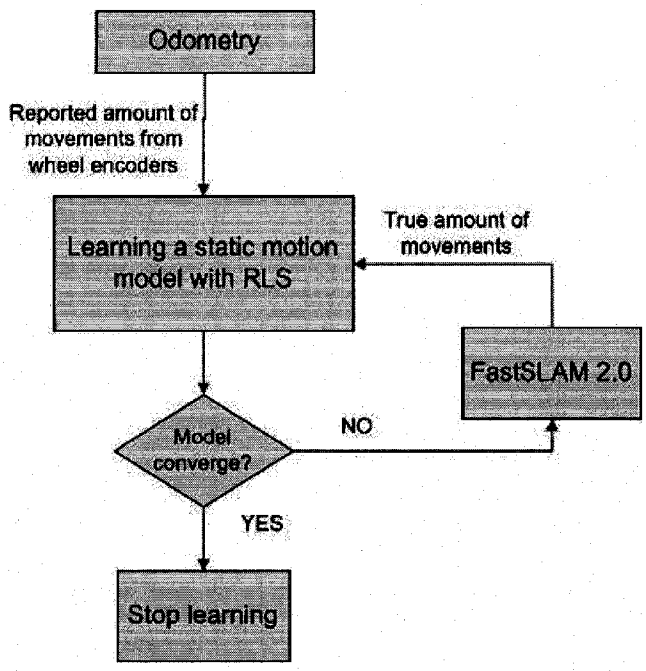

Figure 2.4: Our autonomous framework for building a static motion model with RLS. The learning process receive the reported amount of movements from the odometry and the estimated true amount of movements for FastSLAM 2.0 at each time step. The process then learns motion parameters and decide whether they all converge. If they are, then the process terminates, and we obtain the best static motion model for the particular environment.

we can program our robot to traverse the environment, collect data, and decide by itself when to stop. This framework can also eliminate the need of human intervention and can detect, in some cases, whether our motion model parameters have converged.

Moreover, the bi-loop recursive least squares algorithm (BiRLS) is applied to learn motion model parameters when the robot operates on a dynamically changing ground surface. BiRLS, also a sequential learning technique, can keep track of the changes in time-varying parameters better than RLS and is better suited for building a dynamic motion model. In contrast to the previous works by Milstein & Wang [41] and Beeson et al. [10] that required a predetermined threshold to indicate when the motion parameters shall be updated, BiRLS can adjust motion model parameters as soon as new data arrive. We believe that it can better reflect the changes in the environment in many scenarios. The performance of our newly acquired motion models are evaluated using data from a physical robot based on the robot's ground-truth poses from the ceiling tile tracking system in the context of FastSLAM 2.0.

2.8.2 A comprehensive study on a role of the motion model in the FastSLAM 2.0 framework

It is well-known that a particle filter SLAM algorithm is generally sensitive to the quality of the proposal distribution [9], [27], which is generated directly from the motion model. However, it is less obvious whether the accuracy of the motion model would still play an important role in the accuracy of FastSLAM 2.0 whose improved proposal distribution also takes the most recent sensor measurement into account. Given accurate measurements, it is expected that the FastSLAM 2.0's improved proposal distribution would be more accurate for representing the robot's true poses and would have much lower uncertainty compared to the proposal distribution generated merely from the motion model. In this thesis, we discuss and demonstrate that an accurate motion model significantly improves the localization accuracy of FastSLAM 2.0.

2.9 Summary

This chapter first briefly introduced SLAM and its relationship with the odometry motion model. Although odometry information provides a system to calculate the robot's pose that is easily and cheaply implemented, it is still subject to two types of errors: systematic and non-systematic errors, which were also discussed in detail. We then distinguished the difference between batch and sequential learning. Batch learning applies when we process all the training data at once to learn the weights while sequential learning, which is the fundamental concept of the learning algorithms presented in this thesis, applies when we process each training data case one by one. Lastly, we discussed and compared previous works in learning probabilistic motion model parameters for SLAM and localization algorithms. Although we believe that identifying motion model parameters is an important issue, it has been infrequently discussed in the SLAM literature.

Authors	SLAM/Localization	Static/Dynamic motion model	Ground-truth
Eliazar and Parr [22]	SLAM: DP-SLAM 2.0	Static motion model	DP-SLAM 2.0
Milstein and Wang [41]	Localization: MCL	Dynamic motion model	MCL
Beeson et al. [10]	Localization: MCL	Dynamic motion model	MCL
Kaboli et al. [32]	Localization: MCL	Static motion model	Overhead camera
This thesis	SLAM: FastSLAM 2.0	Both static and dynamic motion models	FastSLAM 2.0

Table 2.3: Summary of the related works and what have been done in this thesis in the area of automatic acquisition of motion models for SLAM and localization algorithms: Part 1

Authors	Proposal Distribution	Learning model parameters
Eliazar and Parr [22]	Directly generated from the motion model	Batch: Weighted recursive least squares
Milstein and Wang [41]	Directly generated from the motion model	Quasi-Sequential: Non-linear optimization (fminsearch: MATLAB). Require a predetermine threshold to indicate when the model shall be updated.
Beeson et al. [10]	Directly generated from the motion model	Quasi-Sequential: Weighted recursive least squares. Require a predetermine threshold to indicate when the model shall be updated.
Kaboli et al. [32]	Directly generated from the motion model	Batch: Bayesian calibration
This thesis	Generated from the motion model and the most recent measurement	Sequential: RLS and BiRLS to learn from the most recent data

Table 2.4: Summary of the related works and what have been done in this thesis in the area of automatic acquisition of motion models for SLAM and localization algorithms: Part 2

Authors	Evaluation of the newly acquired motion model's performance
Eliazar and Parr [22]	Showed resulting maps created by DP-SLAM 2.0 with the improved motion model
Milstein and Wang [41]	Compared the mean of the particles generated from each motion model to one generated from the MCL (with the particular motion model) at each time step.
Beeson et al. [10]	Used the standard paired t-test to conclude that one motion model was better than the other in terms of the localization accuracy based on the ground-truth poses according to the global metric map
Kaboli et al. [32]	Compared the localization accuracy of each motion model with the ground-truth poses obtained from an overhead camera.
This thesis	Compared the localization accuracy from each motion motion based on the ground-truth poses according to the ceiling tile tracking system (This system will be described in chapter 4)

Table 2.5: Summary of the related works and what have been done in this thesis in the area of automatic acquisition of motion models for SLAM and localization algorithms: Part 3

Chapter 3

SLAM, FastSLAM, and Least Squares Techniques

3.1 Introduction

In the previous chapter, we briefly discussed the Simultaneous Localization and Mapping (SLAM) problem and related topics. This chapter examines the structure of the SLAM problem in detail. Moreover, one of the popular solutions to the SLAM problem, the FastSLAM algorithm, is discussed. The FastSLAM algorithm utilizes Rao-Blackwellized particle filters. The algorithm uses a particle filter to estimate the posterior over the robot's path, and each particle possesses n Extended Kalman Filters (EKF) that estimate n landmark positions conditioned on the path estimate. FastSLAM 2.0, which is an improved version of the original FastSLAM algorithm, is used as a tool to evaluate the performance of our newly acquired motion models in this thesis and to provide the robot's estimated ground-truth poses for building our motion models. In addition, the sensitivity of the FastSLAM algorithm to the motion model, which is one of the main contributions in this thesis, is also discussed. The later sections of this chapter discuss the least squares algorithm and its approach for estimating motion model parameters. Several extensions of least squares including the recursive least squares (RLS) and the bi-loop recursive least square algorithms (BiRLS) are discussed. These two sequential machine learning techniques are applied to build a static and a dynamic motion model respectively.

The FastSLAM algorithm is discussed based on four assumptions. First, we assume a specific type of map, a landmark map. A map of the environment is a list of objects in the environment and their locations [53]. The landmark map provides a reference to a robot's pose by representing the environment with the Cartesian locations of parametric landmarks (such as points and lines) [8]. Formally, the landmark map Θ is a list of objects in the environment and can be described as:

$$\Theta = \{\Theta_1, \Theta_2, ..., \Theta_K\} \tag{3.1}$$

K is the total number of objects in the environment, and each Θ_k, with $1 \leq k \leq K$, specifies a

specific correspondence variable and the Cartesian location of the landmark.

Second, we assume a robot is equipped with a range and bearing sensor. These sensors can measure the range and the bearing of a landmark relative to the robot's local coordinate frame. Third, We assume that we can process one landmark at a time. Fourth, we assume known data association: landmarks can be uniquely identified. Therefore, each landmark has a specific correspondence variable.

The first two assumptions are merely for being consistent with our experiments in Chapter 5. The third assumption, which is a general assumption in the SLAM community [53], [44], is for simplicity in developing algorithms that implement probabilistic measurement models so that the robot's estimated pose can be incrementally updated for each observed landmark. The fourth assumption is important because one of the goals of this thesis is to investigate sensitivities of the FastSLAM 2.0 algorithm to different motion models. Thus, it is important to keep possible errors from other sources to a minimum. In localization and SLAM, correcting the robot's pose estimate relies on finding correct correspondence between a landmark observation and its associated map feature. Wrong data association results in an inconsistency where the robot's pose uncertainty decreases, but the estimate error actually increases. In the long run, the robot becomes lost [8]. If we were to assume unknown data association, it would be difficult to distinguish whether the FastSLAM's localization error along the robot's trajectory is from a motion model or from incorrect data association. Note that having a good motion model (or having an effective SLAM or localization algorithm) does help when having to do data association. We could have removed the assumption of known data association, but we do not want to introduce another source of error to our experiments, especially because many popular techniques to determine landmark correspondences are non-deterministic.

3.2 Simultaneous Localization and Mapping (SLAM)

The SLAM problem arises when a robot must simultaneously build a map of an unknown environment and localize itself within that map given only measurements z_t and controls u_t.

Generally, the solution to SLAM is an implementation of a recursive Bayes filter, which possesses two essential steps: the prediction and the update. This computation requires that a state transition model, also known as motion model, and an observation model, also known as sensor model, are defined to describe the effect of a control input and an observation respectively.

We use s_t to denote a state vector describing the robot's pose at time t. For robots operating in the plane, a pose is comprised of a robot's x, y coordinate in the plane and its heading direction, θ. A map contains K landmarks and shall be denoted as $\Theta = (\Theta_1, \Theta_2, ..., \Theta_K)$. The motion model estimates a proposal distribution on state transition:

$$p(s_t|u_t, s_{t-1}) \tag{3.2}$$

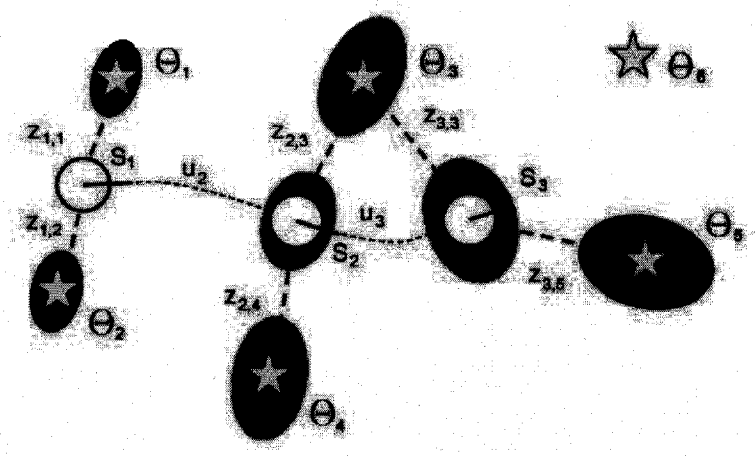

Figure 3.1: SLAM framework. Robot path error correlates errors in the map. The stars and the white circles are the true landmarks and robot poses at time t respectively. Their uncertainties are shown in dark ellipses, and $z_{k,l}$ refers to the k^{th} measurement to the l landmark. Image modified from [53].

Equation 3.2 is a function of the robot control u_t and the previous pose s_{t-1}. Recall that u_t is a control vector to move the robot from s_{t-1} to s_t.

The sensor model describes a likelihood distribution of observing z_t assuming that the robot's pose and the landmark positions are known:

$$p(z_t|s_t, \Theta, n_t) \tag{3.3}$$

where z_t refers to an observation to one specific landmark from the robot at time t, and $n_t \in \{1, 2, ..., K\}$ is the index of the landmark observed at time t. Recall that, we assume landmarks are uniquely identifiable throughout this thesis and the observation is in the form of the range and the bearing information of the landmark relative to the robot's local coordinate frame.

Probabilistically, SLAM is the problem of determining the location of all K landmarks and the robot's pose s_t from z_t and u_t:

$$p(s^t, \Theta|z^t, u^t, n^t) \tag{3.4}$$

We use the superscript ^{t} to refer to a set of variables from time 1 to time t. For example, $s^t = \{s_1, s_2, ..., s_t\}$ refers to the history of robot's pose. Figure 3.1 illustrates the SLAM framework. The robot must simultaneous estimate its pose and the landmark positions. Because the uncertainty in the robot's pose correlates in the uncertainty in landmark estimates, the overall uncertainty in the map estimate, represented by dark ellipses, generally increases with the distance travelled by the robot. The probability of the robot being at s_t depends on its previous pose s_{t-1} along with the most recent control u_t, and the most recent observation z_t.

The important insight in SLAM comes from the fact that the correlations between landmark estimates increase monotonically as more and more observations to previously-seen landmarks are made. In other words, the knowledge of the relative location of landmarks always improves and never diverges regardless to the robot motion [20]. This is because the uncertainty in earlier landmark observations is mostly caused by the robot pose. This statement is valid as the same measurement z_t is used to update all landmarks. As a result, relative locations among landmarks can be assumed to be known. When the previously-seen landmark Θ_n is observed, the robot pose estimate is updated. The other landmarks are also updated because the landmark Θ_n and the other landmarks are highly correlated. Therefore, observing previously seen landmarks not only improves the robot pose estimate but also reduces the uncertainty of other landmarks previously seen by the same robot in the map. In other words, when gaining information on the robot's pose, this information spreads to previously observed landmarks. This amazing effect here is that we do not have to model past poses explicitly [53]. They are correlated in the current map. However, these landmarks only form a so-called accurate relative map of the environment. They do not form a global map, which depends on the accuracy of the robot path estimate. More detail discussion of this topic can be found in [20].

If we consider, in Figure 3.1, the robot at pose s_2 observing two landmarks, Θ_3 and Θ_4, the relative location between these two landmarks is known with high accuracy. If the robot executes control u_3 to s_3, and observes Θ_3 again, the robot's pose is updated together with the position of Θ_3. By doing so, Θ_4's position is also updated although it is not seen from s_3.

3.2.1 The FastSLAM algorithm

The FastSLAM algorithm, introduced by [43], is based on an observation that the state of all landmarks are independent of each other conditioned on the robot's path. In other words, if we know the robot's path, we should be able to estimate locations of all landmarks independently of each other. This motivates the Rao-Blackwellized Particle filter scheme: the estimate of the robot's path is represented by a set of particles, and each particle has its own map through separate Extended Kalman Filters (EKF), one for each individual feature [18]. Based on this scheme, the SLAM posterior in (3.4) can be factored as follows:

$$p(s^t, \Theta | z^t, u^t, n^t) = p(s^t | z^t, u^t, n^t) \prod_{k=1}^{K} (\Theta_k | s^t, z^t, u^t, n^t) \tag{3.5}$$

This factorization states that the calculation of the posterior over the robot's paths and maps can be decomposed into K+1 posteriors. The first factor $p(s^t|z^t, u^t, n^t)$ represents the posterior over trajectories and is estimated nonparametrically using M particles. The second factor $(\Theta_k|s^t, z^t, u^t, n^t)$ represents the posterior over each landmark (hence the product over K) and is estimated by an EKF [36]. A mathematical derivation of this factored SLAM posterior can be found in [53].

FastSLAM was introduced as a more computational efficient alternative to EKF SLAM, which

Step	Descriptions
Prediction	Sample $s_t^{[m]}$ from the proposal distribution $p(s_t^{[m]}\|u_t, s_{t-1}^{[m]})$. We use the superscript notation $^{[m]}$ to refer to the m-th particle in the set.
Measurement update	Update the posterior over each landmark: $p(\Theta_k\|s^t, z^t, u^t, n^t)$ by updating the mean and the covariance of the corresponding EKF.
Importance weight	Calculate the importance weight $w^{[m]}$ for each particle.
Resampling	Draws M times from the particle set with a replacement according to the importance weight $w^{[m]}$. The weights are then reset uniformly.

Figure 3.2: Four steps of the FastSLAM algorithm

was first introduced by [49]. As the name implies, EKF SLAM uses an EKF to incrementally estimate the robot's pose along with the landmark positions. Although EKF SLAM has been applied with considerable success in many SLAM applications [53], its key limitation lies in the computational complexity required for updating the filter. The covariance matrix maintained by the EKF has $O(K^2)$ elements, all of which are updated even if just a single landmark is observed, due to the correlations in landmark estimates as discussed earlier. This quadratic update limits the algorithm to relatively sparse maps with a few hundred features . In practice, maps could contain more than 10^6 features [53], which are not suitable for the plain EKF SLAM. It is worth noting that this shortcoming has not remained unnoticed. Many researchers have proposed computationally efficient EKF SLAM algorithms by decomposing the map into submaps [29],[36],[16]. The update complexity is still quadratic, but these algorithms scale much better for problems with a large number of landmarks. FastSLAM, however, offers more computational efficiency to the SLAM problem. The update time can be as little as $O(MlogK)$ [43], where M is the number of particles. Moreover, FastSLAM can cope with non-linear motion models. This is important when robot's kinematics are highly non-linear or when the pose uncertainty is relatively high.

As with many particle filter based tracking algorithms, FastSLAM with M particles consists of 4 important steps as shown in Figure 3.2. The initial step involves the sampling of the robot's pose for time t using the motion model. The algorithm then updates EKF for all observed landmarks using the standard EKF update technique [53]. The motivation of using an EKF to update each observed landmark comes from the fact that measurements are rarely linear in practice. The next step deals with the calculation of an importance weight, one for each particle. The importance weight is then used to resample particles in the final step.

There are two versions of FastSLAM in the literature, FastSLAM 1.0 [43] and FastSLAM 2.0

[44]. We will discuss them in detail in the next section. With the improved proposal distribution that takes the most recent measurement information into account, FastSLAM 2.0 is an improvement of FastSLAM 1.0. The resulting advantage of FastSLAM 2.0 is that its proposal distribution is locally optimal [17]: conditioned upon the available s_{t-1}, z_t and u_t, it gives the smallest possible variance in the set of importance weights.

3.2.1.1 The FastSLAM 1.0 algorithm

As already mentioned, FastSLAM estimates the path posterior using the particle filter and maintains a separate EKF for each landmark. Therefore, the FastSLAM posterior over the robot's path and landmark positions is represented by the sample set:

$$\{s^{t,[m]}, \mu_1^{[m]}, \Sigma_1^{[m]}, \mu_2^{[m]}, \Sigma_2^{[m]}, ..., , \mu_K^{[m]}, \Sigma_K^{[m]}\} \tag{3.6}$$

Here $\mu_k^{[m]}$, and $\Sigma_k^{[m]}$ are the mean and the covariance of EKF representing the k-th landmark Θ_k attached to the m-th particle. We now discuss four important steps of FastSLAM in detail. For the mathematical derivation of the algorithm, refer to [53].

- **Prediction from a motion model**: FastSLAM 1.0 draws a sample of the new robot pose S_t from S_{t-1} by sampling from the motion model,

$$s_t^{[m]} \sim p(s_t|u_t, s_{t-1}^{[m]}) \tag{3.7}$$

The resulting sample $s_t^{[m]}$ is then added to a temporary set of particles.

- **Measurement update from a sensor model**: Next, FastSLAM 1.0 updates the posterior over each landmark: $(\Theta_k|s^t, z^t, u^t, n^t)$. It is important to consider that this posterior over each landmark is updated based on the assumption that the robot's pose was exactly known for all preceeding time steps [9]. Moreover, The update depends on whether or not $n_t = k$, that is, whether or not Θ_k is observed at time t. If it is not, then the most recent measurement z_t has no effect on the posterior, and there is no update required on the EKF's mean and the covariance of Θ_k as follows:

$$p(\Theta_k|s^t, z^t, n^t) = p(\Theta_k|s^{t-1}, z^{t-1}, n^{t-1}) \tag{3.8}$$

$$(\mu_{k,t}^{[m]}, \Sigma_{k,t}^{[m]}) = (\mu_{k,t-1}^{[m]}, \Sigma_{k,t-1}^{[m]}) \tag{3.9}$$

If Θ_k is observed at time t, the posterior over Θ_k can be obtained by applying Bayes' rule and the Markov assumption [43],

$$p(\Theta_k|s^t, z^t, n^t) = p(z_t|\Theta_k, s_t, n_t)p(\Theta_k|s^{t-1}, z^{t-1}, n^{t-1}) \tag{3.10}$$

The second factor of (3.10) is represented by a Gaussian distribution with the mean $\mu_k^{[m]}$ and the covariance $\Sigma_k^{[m]}$. For the new estimate at time t to be Gaussian, FastSLAM linearlizes likelihood model $p(z_t|\Theta_k, s_t, n_t)$ through Taylor expansion [53]. The new mean and covariance are obtained using the standard EKF measurement update as follows:

$$K_t^{[m]} = \Sigma_{k,t-1}^{[m]}(H_t^{[m]})^T(H_t^{[m]}\Sigma_{k,t-1}^{[m]}(H_t^{[m]})^T + Q_t)^{-1} \tag{3.11}$$

$$\mu_{k,t}^{[m]} = \mu_{k,t-1}^{[m]} + K_t^{[m]}(z_t - z'^{[m]}_t) \tag{3.12}$$

$$\Sigma_{k,t}^{[m]} = (I - K_t^{[m]}H_t^{[m]})\Sigma_{k,t-1}^{[m]} \tag{3.13}$$

K is the Kalman gain, which specifies the degree to which the new measurement is incorporated in the estimation. H is the Jacobian of the linearized measurement prediction $p(z_t|\Theta_k, s_t, n_t)$ with respect to the position of landmark k. $(z_t - z'^{[m]}_t)$ is the innovation, which is the difference between the actual measurement z_t and the expected measurement z'_t. The more certain the observation, the higher the Kalman gain, and the stronger the resulting location correction. Q is the covariance of the measurement probability. Refer to [16], [49] and the appendix section for a more information on EKF and to [53] for its mathematical derivation.

- **Importance weight**: An individual importance weight $w^{[m]}$ is assigned to each particle according to the quotient of the target and the proposal distribution:

$$\begin{aligned} w^{[m]} &= \frac{\text{target distribution}}{\text{actual proposal distribution}} \\ &= \frac{p(s_t^{[m]}|z^t, u^t, n^t)}{p(s^{t,[m]}|z^{t-1}, u^t, n^{t-1})} \end{aligned} \tag{3.14}$$

The actual proposal distribution in (3.14) is computed from the assumption that the set of temporary particles representing the robot's path in (3.7) is distributed according to $p(s^{t-1,[m]}|z^{t-1}, u^{t-1}, n^{t-1})$. On the other hand, the proposal distribution in (3.7) is generated with the assumption that the previous pose of the robot s_{t-1} is exactly known. In other words, the recursive equation of the Bayes filter at each time-step only requires the momentary pose estimate. Thus, the actual proposal distribution is calculated according to:

$$p(s^{t,[m]}|z^{t-1}, u^t, n^{t-1}) = p(s^{t-1,[m]}|z^{t-1}, u^{t-1}, n^{t-1})\; p(s_t^{[m]}|u_t, s_{t-1}^{[m]}) \tag{3.15}$$

Equation 3.14 can be approximated by the similar linear approximation used in the measurement update. In particular, the importance weight for each particle is defined as:

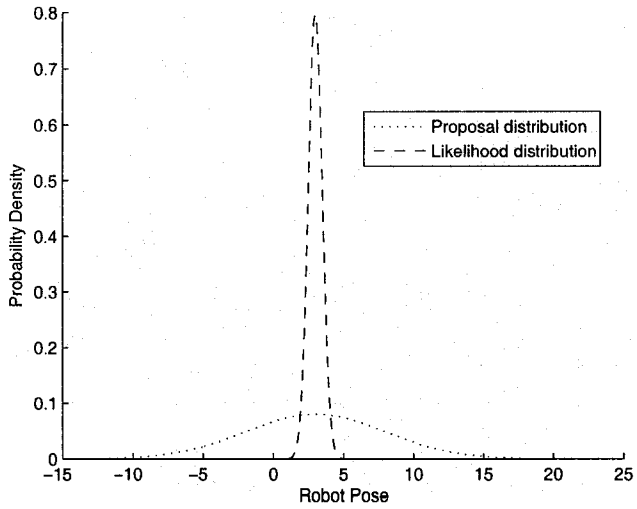

Figure 3.3: The scenario where the sensor information is more precise than the motion information

$$
\begin{aligned}
w_t^{[m]} &= \eta |2\pi Q_t^{[m]}|^{\frac{1}{2}} \exp\{-\frac{1}{2}(z_t - z'^{[m]}_t) Q_t^{[m]-1} (z_t - z'^{[m]}_t)\} \\
Q_t^{[m]} &= (H_t^{[m]})^T \Sigma_{k,t-1}^{[m]} H_t^{[m]} + Q_t
\end{aligned} \tag{3.16}
$$

Refer to [53] on page 448 for the derivation of calculating an importance weight.

- **Resampling**: Now that particles are extended according to the motion model and landmark positions are updated based on the sensor model, the particles are weighted according to the likelihood of the observation given the sampled poses. Then, particles are resampled according to their weights to allow them to concentrate in the areas with high likelihoods so particles with low importance weights are replaced by particles with high weights.

3.2.1.2 The FastSLAM 2.0 algorithm

FastSLAM 2.0 extends FastSLAM 1.0 with a single new idea: the proposal distribution takes the measurement z_t into account when sampling s_t. By doing so, the proposal distribution matches the true posterior $p(s^t, \Theta | z^t, u^t, n^t)$ more closely. Intuitively, the better the proposal distribution approximates the true posterior, the better the performance of the FastSLAM algorithm.

The proposal distribution in FastSLAM 1.0 could cause the algorithm to degenerate very quickly when the sensor information is significantly more accurate than the motion information. This scenario is likely to happen for robots equipped with laser range finders and odometers. Such a scenario is illustrated in Figure 3.3. The meaningful areas of the likelihood distribution are much more focused than the meaningful areas of the proposal distribution. Drawing samples from this proposal distribution could result in many particles having low or zero importance weights because only a

few particles are drawn from the area with high likelihoods. After resampling, only a few particles within that area survive. In the long run, the remaining particles become all or mostly identical, which could insufficiently encode uncertainty in the estimation. We will further discuss this issue in the next section.

- Prediction from a motion model and a sensor model

By taking the most recent measurement into consideration, the pose $s_t^{[m]}$ is drawn from the following improved proposal distribution:

$$s_t^{[m]} = p(s_t|s^{t-1,[m]}, u^t, z^t, n^t) \tag{3.17}$$

The proposal distribution in (3.17) can be reformulated with Bayes and Markov rules as follows:

$$p(s_t|s^{t-1,[m]}, u^t, z^t, n^t) = \eta^{[m]} \int p(z_t|\Theta_k, s_t, n_t) p(\Theta_k|s^{t-1,[m]}, z^{t-1}, n^{t-1}) d\Theta_{n_t} \\ p(s_t|s_{t-1}^{[m]}, u_t) \tag{3.18}$$

This improved proposal distribution is incrementally refined for each observation on each previously-seen landmarks $k = n_t$. η is a normalizer in Bayes rule, which is $p(z_t|s^{t-1,[m]}, u^t, z^{t-1}, n^t)$ in this particular case. Sampling from (3.18) is difficult since it does not possess a closed form solution. Fortunately, it can be approximated by a Gaussian [53], [44]:

$$\Sigma_{s_t}^{[m]} = [H_s^T (Q_t^{[m]})^{-1} H_s + R_t^{-1}]^{-1} \tag{3.19}$$

$$\mu_{s_t}^{[m]} = \Sigma_{s_t}^{[m]} H_s^T (Q_t^{[m]})^{-1} (z_t - z'^{[m]}_t) + s'^{[m]}_t \tag{3.20}$$

$$Q_t^{[m]} = Q_t + H_\Theta \Sigma_{k,t-1}^{[m]} H_\Theta^T \tag{3.21}$$

The Gaussian approximation of the improved proposal distribution is done in the same manner as the EKF [44]. The matrix R_t is the covariance matrix of the state transition probability $p(s_t|s_{t-1}^{[m]}, u_t)$. This covariance matrix represents the overall uncertainty in the robot's predicted pose from the motion model. The matrix $Q_t^{[m]}$ is the covariance matrix of the measurement probability $p(z_t|\Theta_k, s_t, n_t)$. This covariance matrix represents the overall uncertainty of the measurement. $s'^{[m]}_t$ is the robot's predicted pose sampled from $p(s_t|s_{t-1}^{[m]}, u_t)$, and $z'^{[m]}_t$ is the robot's predicted measurement to the landmark Θ_k. Notice at (3.20), $\Sigma_{s_t}^{[m]} H_s^T (Q_t^{[m]})^{-1}$ is essentially the Kalman gain, and $z_t - z'^{[m]}_t$ is the innovation in EKF. [1] Refer to the appendix section for a brief review on the EKF. H_s and H_Θ are the jacobians of the linearized measurement prediction $p(z_t|\Theta_k, s_t, n_t)$ with respect to the robot's pose s_t and the landmark position Θ_k respectively:

[1] The innovation vector is the difference between the observed measurement and the predicted one calculated based on the robot's predicted pose and its pose uncertainty. The Kalman gain additionally scales the innovation vector. The more certain the observation (compared to the prediction), the higher the Kalman gain, and hence the stronger the resulting pose correction.

$$H_s = \nabla_{s_t} h(\mu_{k,t-1}^{[m]}, s'^{[m]}_t) \tag{3.22}$$

$$H_\Theta = \nabla_{\Theta_t} h(\mu_{k,t-1}^{[m]}, s'^{[m]}_t) \tag{3.23}$$

The pose $s_t^{[m]}$ is then sampled from the Gaussian in (3.19) that approximates the posterior in (3.17). The full derivation of the FastSLAM 2.0's improved proposal distribution can be found in [53].

- **Measurement update from the sensor model**

The updating step remains conceptually the same as in FastSLAM 1.0 as follows::

$$K_t^{[m]} = \Sigma_{k,t-1}^{[m]} H_\Theta^T (Q_t^{[m]})^{-1} \tag{3.24}$$

$$\mu_{k,t}^{[m]} = \mu_{k,t-1}^{[m]} + K_t^{[m]} (z_t - z'^{[m]}_t) \tag{3.25}$$

$$\Sigma_{k,t}^{[m]} = (I - K_t^{[m]} H_\Theta) \Sigma_{k,t-1}^{[m]} \tag{3.26}$$

- **Importance weight**: As in FastSLAM 1.0, an individual importance weight $w^{[m]}$ is assigned to each particle according to the quotient of the target and the actual proposal distribution:

$$\begin{aligned} w^{[m]} &= \frac{\text{target distribution}}{\text{actual proposal distribution}} \\ &= \frac{p(s_t^{[m]} | z^t, u^t, n^t)}{p(s^{[m],t-1} | z^{t-1}, u^{t-1}, n^{t-1}) p(s_t^{[m]} | s^{[m],t-1}, z^t, u^t, n^t)} \end{aligned} \tag{3.27}$$

The first term of the actual proposal distribution in (3.27) arises from the assumption that the path in $s^{[m],t}$ has been generated according to the target distribution one step earlier. The second term of the actual proposal distribution is the pose sampling from the improved proposal distribution.

This expression can be approximated by the similar linear approximation used in the measurement update. In particular, the importance weight for each particle is defined as:

$$\begin{aligned} w_t^{[m]} &= |2\pi L_t^{[t]}|^{\frac{-1}{2}} \exp\{-\frac{1}{2}(z_t - z'^{[m]}_t) L_t^{[t]-1} (z_t - z'^{[m]}_t)\} \\ L_t^{[t]} &= {H_s}^T Q_t H_s + H_\Theta \Sigma_{k,t-1}^{[m]} H_\Theta^T + R_t \end{aligned} \tag{3.28}$$

3.2.2 Sensitivity of the FastSLAM algorithm to the motion model

The accuracy of the FastSLAM algorithm does not *directly* deteriorate with the accuracy of the proposal distribution, which is generated from the motion model. To a large degree, it is due to the loss of particle diversity in the particle sampling process, which is also known as the particle depletion problem [30], the particle impoverishment problem [27], the particle deprivation problem [53], and the particle degeneracy problem [11]. The loss of particle diversity often occurs when the

variance of the particle weights in the particle set is large. As a result, the resampling step selects many copies of a few highly weighted particles and discards ones with low weights. As time goes on, more and more particles are duplicated simply due to the random nature of the resampling step, which draws with replacement Y particles from the current particle set (the probability of drawing each particle is given by its importance weight). Note that resampling transform a particle set of Y particles into another particle set of the same size. Thus, if we run FastSLAM long enough, the algorithms will end up with one distinct particle that represents the robot's path and a map estimate regardless of the initial particle set size.

The loss of particle diversity may initially seem to be a desirable scenario because the filter actually converges to the maximum likelihood state. However, it is actually undesirable, especially in ambiguous situations as the filter may not be able to recover if the maximum likelihood state happens to be an incorrect state. Moreover, it can prevent the robot from closing the loop because the information acquired from previously-seen landmarks cannot be propagated through the entire map. We will discuss this issue later in this section. Fortunately, the more accurate the proposal distributions is, the slower the loss of diversity due to resampling in the FastSLAM algorithm, which results in a more accurate FastSLAM algorithm in the long run [9]. Therefore, we can conclude that the FastSLAM algorithm is sensitive to the motion model.

Each particle in FastSLAM can be viewed as a hypothesized path of the robot. Conditioned on each robot's path, the landmarks positions are independent. As a result, the correlations between the landmarks are conceptually represented in the collection of particles [42]. This is different from, for example, the EKF SLAM that maintains such correlations directly. Moreover, although each particle in FastSLAM estimates a map at time t with the assumption that the robot's pose was exactly known for all preceeding time steps, it is important to remember that past pose estimate errors were not forgotten; they are recored in map estimates [9]. Whenever resampling is performed, an entire pose history and a map hypothesis is lost for each particle not selected. This depletes the number of samples representing past poses and consequently erodes the information of the landmark position estimates conditioned on these past poses. In general, the more diverse the particle set, the more accurately FastSLAM can revise the robot's path (and thus the landmark positions) given an observation to a previously-seen landmark [2] [42].

The loss of particle diversity is best described with the loop closure example. In loop closure, a robot moves through unknown terrain and at some point encounters landmarks it has not seen for a long time. Since we assume known data association, the robot does not have any difficulties recognizing landmarks that have previously been observed. As a result, the robot should be able to correct its own pose estimate and reduce accumulated errors due to correlations in SLAM. Observing a

[2]The best way to think about this statement is to consider that if there are M particles, then we have M possible robot's paths. Some are more accurate than the others. When the robot observes a previously-seen landmark, each particle is assigned an importance weight score. Then M particles are resampled based on their scores. For each of the particle not selected, the path hypothesis is lost. If the algorithm works correctly, then lost path is expected to be the less accurate one. Therefore, we get a more accurate set of robot paths. The same idea applies for updating landmark positions.

previously-seen landmark improves the robot pose estimates and eliminates some of the uncertainty in landmark estimates previously seen by the same robot. Therefore, it is important to maintain the correlations between the robot's pose and the landmark positions so that the information acquired when closing a loop can be propagated through the entire map. Unlike EKF SLAM that maintains the correlations directly though its huge covariance matrix, FastSLAM maintains such correlations through the diversity in the particle set [53], [9]. Therefore, better diversity in particles results in better loop closing performance as new observations can affect the robot's pose of further back in the past so that it helps reducing the uncertainty around the robot's pose[3] and the landmark positions. Unfortunately, resampling throws away correlation information. New observations may not affect the landmark positions observed in the past if all particles share common history[4]. As a result, overall uncertainty in the FastSLAM algorithm tends to increase over time even in the loop closure scenario, where overall uncertainty should be reduced.

Although resampling could cause the loss of particle diversity, it is still important. It refocuses the particle set to areas in state space with high probability. Without resampling, many particles might be wasted in areas of low or zero probability representing the robot's pose. The choice of when to resample is still an open problem and requires practical experience. Resampling too often increases the risk of losing diversity. If we sample too infrequently, many samples might be wasted in regions of low probability, which could result in a divergence of the filter.

Accurate proposal distribution, which is generated from the motion model, could slow down the loss of particle diversity. If the samples were drawn from the target distribution, which is not available in general, their importance weights would be equal to each other (uniform weights) due to the importance sampling principle as shown in (3.14). On the other hand, the worse the approximation of the target distribution, the higher the variance of the importance weights, which could quickly lead to the loss of particle diversity as mentioned. Therefore, we want to keep particle weights as relatively uniform as possible so that fewer particles are eliminated in the resampling process. In other words, the more accurate the model motion is with respect to being an approximation of the target distribution, the slower the loss of particle diversity leads to the more accurate the FastSLAM algorithm. Therefore, FastSLAM is sensitive to the motion model.

Bailey [9] has shown that particle sample size does not affect the depletion rate. However, the rate of diversity (the diversity's speed) increases monotonically with respect to landmark density. Therefore, FastSLAM may be able to generate an accurate map in the short-term, but it tends to underestimate its own uncertainty in the long run, where we could end up with one distinct particle representing the robot's path. Lastly, it is worth nothing that the lack of long-term correlations in the FastSLAM representation is arguably its most important weakness compared to Gaussian-based SLAM techniques [53]. Although it is possible to slow down the loss of particle diversity, the problem cannot be completely avoided due to the nature of particle filters.

[3]The uncertainty around the robot's pose in FastSLAM is based on the variance of the importance weight in a particle set.
[4]This is the scenario where we only have one distinct particle representing the map

3.3 Least squares algorithms

In this section, we first discuss the least squares (LS) algorithm, which is a fundamental technique of learning motion model parameters in this thesis. The least squares approach to estimate motion model parameters is then described. Next, we discuss the recursive least-square (RLS) algorithm, which is used to build a static motion model in this thesis. As the name implies, RLS is the recursive version of LS. Computation of the LS algorithm can be arranged in such a way that the results obtained at time $t-1$ can be used to get the estimates at time t. The main objective of the RLS algorithm is to save computational time and space in adaptive systems where observations are obtained sequentially in real time.

Although the recursive version of the least square algorithm has been extensively utilized in many adaptive systems [7], its slow tracking capacity is a major disadvantage. Therefore, many modifications of this algorithm have received considerable attentions. The most notable one is the the recursive least squares algorithm with a forgetting factor (RLSFF), which is one of the extensions, is briefly discussed. The concept of the forgetting factor is to gradually discard older data in favor of more recent data. Lastly, we introduce the bi-loop recursive least squares algorithm (BiRLS), which is applied to build a dynamic motion model in this thesis.

3.3.1 The least squares (LS) algorithm

In the least squares estimation, model parameters of a linear model are chosen in such a way that the sum of the square of the difference between actual observations and predicted values is minimized. Because the differences are first squared, then summed, there are no cancellations between positive and negative errors.

We consider mathematical models that can be written in the form:

$$y(t) = \Phi(t)\Theta \tag{3.29}$$

$$= \phi_1(t)\theta_1 + \phi_2(t)\theta_2 + .. + \phi_m(t)\theta_m \tag{3.30}$$

where $y(t)$ and $\Phi(t)$ are a pair of observed variables at time t in which the row vector $\Phi(t) = [\phi_1(t)\ \phi_2(t)\ ...,\ \phi_M(t)]$ and $y(t)$ is a scalar. The column vector $\Theta = [\theta_1\ \theta_2\ ...\ \theta_M]^T$ contains a set of model parameters, where M is the number of model parameters.

In the least squares estimation, model parameters of a linear model are chosen to minimize the following loss function:

$$V(\Theta, t) = \frac{1}{2}\sum_{i=1}^{t}(y(i) - \Phi(i)\Theta)^2 \tag{3.31}$$

3.3.2 The least square approach to estimate motion model parameters

In this section, we describe the least square approach to estimate motion model parameters. Recall that the pose of a mobile robot operating in a plane comprises its two-dimensional planar coordinates relative to an external coordinate frame (x, y), along with its angular orientation θ. As discussed in the previous chapter, the motion model used in this thesis is defined as:

$$x' = x + Dcos(\theta + \frac{T}{2}) \tag{3.32}$$

$$y' = y + Dsin(\theta + \frac{T}{2}) \tag{3.33}$$

$$\theta' = \theta + T; \theta' \in (-\pi, \pi) \tag{3.34}$$

Given that $N(a, b^2)$ refers to a normal distribution with the mean a and the variance b^2, the true translational movement D and the true rotational movement T are defined as follows:

$$D \sim N(p_1 \cdot \bar{d} + p_2 \cdot \bar{t}, p_3 \cdot \bar{d}^2 + p_4 \cdot \bar{t}^2) \tag{3.35}$$

$$T \sim N(p_5 \cdot \bar{d} + p_6 \cdot \bar{t}, p_7 \cdot \bar{d}^2 + p_8 \cdot \bar{t}^2) \tag{3.36}$$

The translational and rotational movement according to the odometry are denoted as $\bar{d}$ and $\bar{t}$ respectively. p_i is a set of motion model parameters that will be grouped into four least squares systems and each system will be solved seperately. Specifically, we solve for p_i with least squares by formulating (3.35) and (3.36) as:

$$y_D(t) \sim N(\Phi_{mD}(t)\Theta_{mD}, \Phi_{vD}(t)\Theta_{vD}) \tag{3.37}$$

$$y_T(t) \sim N(\Phi_{mT}(t)\Theta_{mT}, \Phi_{vT}(t)\Theta_{vT}) \tag{3.38}$$

That is, we apply four least squares systems to estimate eight model parameters p_i. The first least squares system estimates the mean parameters of the D term in (3.35), denoted as $\Theta_{mD} = [p_1\ p_2]^T$, from $y'_D(t) = \Phi_{mD}(t)\Theta_{mD}$. The expected value of $y'_D(t)$ is the true amount of the translational movement D [5], and $\Phi_{mD}(t)$ refers to a row vector representing the reported odometry movements $[\bar{d}\ \bar{t}]$ at time t.

The second least squares system estimates the variance parameter of the D term in (3.35), denoted as $\Theta_{vD} = [p_3\ p_4]^T$, from $y''_D(t) = \Phi_{vD}(t)\Theta_{vD}$. The expected value of $y''_D(t) = (\Phi_{mD}\Theta_{mD} - D)^2$. Moreover, $\Phi_{vD}(t)$ refers to a row vector representing the squared reported odometry movements $[\bar{d}^2\ \bar{t}^2]$ at time t. The calculation is similar for the other two least squares systems that estimate

[5] The true amount of translational movement comes from another independent model (i.e. the ceiling tile tracking system or the FastSLAM 2.0 that will be discussed in section 4.5. Moreover, the true amount is not unique (i.e. it varies at each time step).

the mean parameters $\Theta_{mT} = [p_5\ p_6]^T$ and the variance parameters $\Theta_{vT} = [p_7\ p_8]^T$ of the rotational movement term (3.36). Note that we always have two model parameters in each least squares system.

The least squares approach to estimate the parameters of the mean and the parameters of the variance of a normal distribution is first introduced by [22]. It can be thought of as a joint optimization system that first estimates the parameters of the mean *independently* of the variance. The system then estimates the parameters of the variance given the estimated mean. This is logical as the variance of a probability distribution is the average of the squared differences between the data points and the mean. Therefore, the mean must first exist before we can calculate the variance (or the parameters of the variance).

3.3.3 The Recursive Least Squares (RLS) algorithm

Computation of the least squares algorithm can be arranged in such a way that the result (estimated model parameters) obtained at time $t-1$, $\Theta(t-1)$, can be used to get the estimate at time t, $\Theta(t)$. In other words, the objective of the RLS algorithm is to estimate $\Theta(t)$ given $\Theta(t-1)$ and a pair of observed variables $y(t)$ and $\Phi(t)$. This idea is most useful in the context of adaptive controllers, where measurements are obtained periodically in real-time. Moreover, it is desirable to make the calculation recursive to save computation time and space. The recursive form is given by:

$$K(t) = P(t-1)\Phi(t)^T(I + \Phi(t)P(t-1)\Phi(t)^T)^{-1} \tag{3.39}$$

$$\Theta(t) = \Theta(t-1) + K(t)(y(t) - \Phi(t)\Theta(t-1)) \tag{3.40}$$

$$P(t) = (I - K(t)\Phi(t))P(t-1) \tag{3.41}$$

This formula has a strong intuitive appeal [31], [7]: the parameter estimate $\Theta(t)$ is obtained by adding a correction to the previous estimate $\Theta(t-1)$. The correction is proportional to $y(t) - \Phi(t)\Theta(t-1)$, which refers to the difference between the true observed value $y(t)$ and the predicted observed value $\Phi(t)\Theta(t-1)$. The components of the matrix $K(t)$ are weighting factors that tell how much the correction and the previous estimate should be combined. The matrix $P(t)$ is the covariance matrix, which provides a measure of uncertainty in parameter values.

There are two variables in RLS involved in the recursions (those with time index $t-1$): $\Theta(t-1)$ and $P(t-1)$. They require initial values: $\Theta(0)$ and $P(0)$. RLS allows us to identify initial parameters with respect to our knowledge of the environment. Generally, if we have a good knowledge of the initial conditions of the environment, we can assign a small value to $P(0)$; otherwise, we can assign a high value to $P(0)$. A systematic way to determine initial values for the two variables can be found in [31], [14].

3.3.4 The Recursive Least Squares algorithm with a Forgetting Factor (RLSFF)

The recursive least squares algorithm has been widely used in adaptive systems due to its simplicity, good convergence properties, and small mean square error in stationary environments [50],[31]. However, RLS is not suitable for capturing the new values of the parameters when they are time-varying. In such scenarios, it is useful to give more emphasis to recent data than to older data, which results in the least squares loss function as follows [54]:

$$V(\Theta, t) = \frac{1}{2}\sum_{i=1}^{t} \lambda^{t-1}(y(i) - \Phi^T(i)\Theta)^2 \tag{3.42}$$

where λ is called a forgetting factor such that $0 < \lambda \leq 1$. The RLSFF can also be calculated recursively as follows:

$$K(t) = P(t-1)\Phi(t)^T(\lambda + \Phi(t)P(t-1)\Phi(t)^T)^{-1} \tag{3.43}$$

$$\Theta(t) = \Theta(t-1) + K(t)(y(t) - \Phi(t)\Theta(t-1)) \tag{3.44}$$

$$P(t) = (I - K(t)\Phi(t))P(t-1)\frac{1}{\lambda} \tag{3.45}$$

The main difference between the RLS and the RLSFF is how the covariance matrix $P(t)$ is updated. The covariance eventually decays to zero with time in the classical recursive least squares method. On the other hand, the covariance matrix in the RLSFF is divided by $\lambda \leq 1$ at each update as shown in (3.45). This slows down fading out of the covariance matrix [54]. As a result, parameter estimations converge more slowly.

As already discussed, λ gives a larger weight to more recent data in order to cope with changes in the environment. If $\lambda = 1$, all the data are weighted equally, which is the basic RLS. The choice of λ involves an interesting trade-off [48]. If λ is small, RLSFF can better adapt to the changes in the environment at the expense of sensitivity to system noise. It will also make the estimates uncertain (large $P(t)$ and hence $K(t)$ becomes large). If λ is large, it is difficult to track fast parameter variations. Therefore, the choice of λ depends on the system's environment. Typical choice of λ is in the range between 0.98 to 0.995 [35].

The RLSFF also only works well if the system has excitation. Otherwise, the forgetting factor leads to the covariance *windup* problem [24] because the covariance matrix is divided even if $P(t-1)\Phi(t)^T = 0$ (refer to (3.43)). This condition results in $K(t) = 0$ and implies that $y(t)$ does not contain any new information about the parameter Θ [7]. In this case, the covariance matrix $P(t)$ increases exponentially. As a result, the parameter estimates could deteriorate because any small change in future Φ will lead to large parameters changes.

3.3.5 The Bi-Loop Recursive Least Squares algorithm (BiRLS)

As discussed, choosing an inappropriate forgetting factor could have major consequences. If the forgetting factor is closer to 1, the algorithm is not suitable for tracking time-varying parameters. On the other hand, if the forgetting factor is far away from 1, the algorithm is very sensitive to system noise. Second, the RLSFF algorithm is not suitable for use in steady-state systems; old information is continually forgotten while there is very little new dynamic information coming in. In this scenario, the algorithm could lead to the covariance *windup* problem.

As a result, there are many extensions to the RLS algorithm to track fast time-varying systems and/or to avoid the covariance *windup* problem. One of the most popular schemes lies within the concept of a time-varying forgetting factor, which is also known as an adaptive forgetting factor, introduced by [24]. If the estimation error is small, the forgetting factor should be set as close to one as possible to allow the RLSFF to use most of the previous observations. On the other hand, if the estimation error is large, a smaller forgetting factor should be introduced to allow the RLSFF to use a more recent set of observations. Doing so should decrease the estimation error. This interesting idea has been applied in [34], [47], [56], [57],[50], [37].

Yu and Shih [55] recently introduce another solution to improve the RLS to track fast time-varying systems, called bi-loop recursive least squares algorithm with a forgetting factor (BiRLSFF). The main objective of BiRLSFF is to keep track of rapid, slow, or periodic changes in model parameters. As the name implies, BiRLSFF consists of two nested loops. The outer loop is identical to the RLSFF algorithm, and the purpose of the inner loop is to quickly update the system parameters using only the most recent data ($\Phi(t)$ and $y(t)$). We notice that the purposes of the inner loop and the forgetting factor are very similar in that they provide more emphasis to recent data than older data. Moreover, BiRLSFF may still encounter the covariance windup problem and requires an appropriate choice of the forgetting factor, which could be difficult to determine. As a result, we believe it is redundant and unnecessary to introduce the forgetting factor to the bi-loop least square algorithm.

Therefore, we made a slight modification to BiRLSFF to come up with the bi-loop recursive least squares (BiRLS) to learn motion model parameters of a dynamic motion model by setting the forgetting factor to one. The BiRLS algorithm is illustrated in Figure 3.4. The outer loop requires initial parameter values $\Theta(0)$ and the covariance matrix $P(0)$. The update step of the outer loop is identical to the RLS algorithm:

$$K(t) = P(t-1)\Phi(t)^T(I + \Phi(t)P(t-1)\Phi(t)^T)^{-1} \quad (3.46)$$

$$\Theta(t) = \Theta(t-1) + K(t)(y(t) - \Phi(t)\Theta(t-1)) \quad (3.47)$$

$$P(t) = (I - K(t)\Phi(t))P(t-1) \quad (3.48)$$

The inner loop requires initial parameters values $\Theta_{in}(0)$ and the covariance matrix $P_{in}(0)$, together with a pair of observed variables y_{in} and Φ_{in}. These values are initialized from the outer

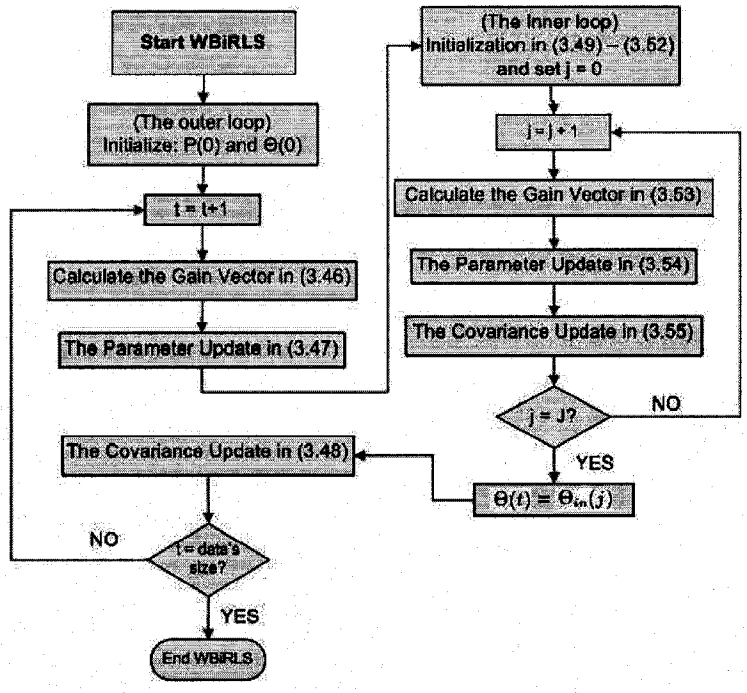

Figure 3.4: BiRLS's flowchart. BiRLS consists of two nested loop. The outer loop is identical to the RLS algorithm, and the purpose of the inner loop is to quickly update the system parameters using only the most recent data.

loop as:

$$\Theta_{in}(0) = \Theta(t) \tag{3.49}$$

$$P_{in}(0) = P(t) \tag{3.50}$$

$$y_{in} = y(t) \tag{3.51}$$

$$\Phi_{in} = \Phi(t) \tag{3.52}$$

The update step of the inner loop is similar to the outer loop with the use of initial values in (3.49) to (3.52) as follows:

$$K_{in}(j) = P_{in}(j-1)\Phi_{in}^T(I + \Phi_{in}P_{in}(j-1)\Phi_{in}^T)^{-1} \tag{3.53}$$

$$\Theta_{in}(j) = \Theta_{in}(j-1) + K_{in}(j)(y_{in} - \Phi_{in}\Theta_{in}(j-1)) \tag{3.54}$$

$$P_{in}(j) = (I - K_{in}(j)\Phi_{in})P_{in}(j-1) \tag{3.55}$$

The inner loop executes J times, where J denotes a predetermined integer. Generally, the larger J is, the more capability the system has to track changes in parameters at the costs of sensitivity to system noise and computational complexity. One can also set the inner loop to terminate when the

difference between each pair of $\theta_{in}(j)$ and $\theta_{in}(j-1)$ of all model parameters in Θ_{in} are less than a predetermined threshold. Once Θ_{in} has been estimated after J iterations in the inner loop, BiRLS returns to the outer loop and replaces $\Theta(t)$ with $\Theta_{in}(j)$. Notice that the covariance of the outer loop remains unchanged. Also notice that the purpose of the inner loop can be thought of as a part of the learning system that gives a more emphasis on recent data than older one.

From a practical point of view, it could be easier to set the value of J than to set a forgetting factor in RLSFF. We find it difficult to visualize how the forgetting factor would affect the weighting of our data. As is mentioned, a typical choice of λ is in the range between 0.98 to 0.995 [35], but we could not say that, for example, setting λ to 0.98 means that we would like to focus our learning mainly based on 98% of our data. On the other hand, it is more intuitive to set the value of J. The more emphasis we would like to give to the most recent data, the higher value for J we should set. Having said that, the choice of J still requires practical experience on the domain and some knowledge on the data (i.e. whether the noise in the data is high).

3.4 Summary

This chapter examined in detail the structure of the FastSLAM 2.0 algorithm, which is used as a tool to evaluate the performance of different motion models in Chapter 5 and to provide the robot's estimated ground-truth poses for building our motion models. Moreover, the sensitivity of the FastSLAM algorithm to the motion model was discussed to suggest that the motivation for acquiring a good motion model is strong. Later in the chapter, we discussed the least squares algorithm and its approach to estimate motion model parameters. We then described two sequential machine learning techniques based on the least squares algorithm to build our motion models. First, the recursive least squares (RLS) is used to build a static motion model as soon as new data arrive. Second, the bi-loop recursive least squares (BiRLS), a modification of [55], is applied to build a dynamic motion model because it can keep track of rapid, slow, or periodic changes in time-varying parameters better than RLS because BiRLS gives a more emphasis to the most recent data than older one. In the next chapter, we discuss the process of applying these two learning algorithms to learn motion model parameters as well as our data gathering process.

Chapter 4

Data Gathering and Parameter Estimation

4.1 Introduction

In this chapter, we first introduce a Magellan Pro robot, which is the robot used in this thesis and the ceiling tile tracking system that was implemented as a measurement tool to provide reliable ground-truth poses for evaluating the localization accuracy of the FastSLAM 2.0 algorithm using different motion models. We then discuss our data gathering method and the use of FastSLAM 2.0 to estimate the robot's true poses. We then provide graphical presentations of uncertainties in our data to show that these uncertainties are reasonable to be approximated by normal distributions. We also suggest that we can use poses according to FastSLAM 2.0 as estimated ground-truth poses to learn motion model parameters. Before closing this chapter, we discuss the process of applying RLS and BiRLS, discussed in the previous chapter, to build static and dynamic motion models respectively.

4.2 The robot

The Magellan Pro robot is a commercial indoor mobile robot made by the IRobot Corporation. It is a differential-driven robot with two drive wheels and a castor wheel as illustrated in Figure 4.1.

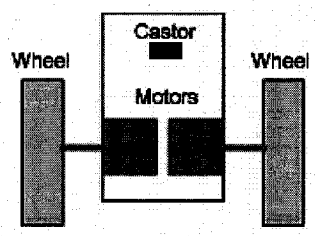

Figure 4.1: A typical differential-driven with two drive wheels and a castor

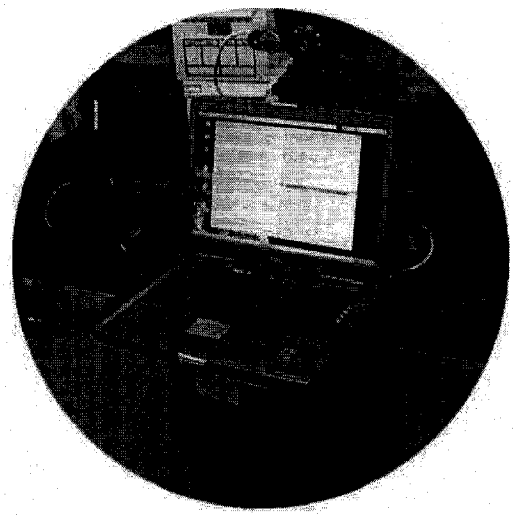

Figure 4.2: The Magellan Pro with a digital camera placed on the upper deck pointing upward

Each wheel is connected to its own motor, and the caster wheel is a non-driven wheel that provides balance to the robot. Straight-line motion is accomplished by spinning both wheels at the same velocity in the same direction. Rotation is done by spinning one of the wheels faster than the other, and in-place rotation (turning without moving) is done by spinning both wheels at the same rate in the opposite direction.

The robot, 40.6 centimeters in diameter and 25.4 centimeters tall, comes equipped with an on-board CPU running Red Hat Linux as shown in Figure 4.2. Its 16 sonar sensors provide long-range information to a distance of approximately fiver meters while the 16 infrared sensors are used to detect a presence of objects within 30 centimeters of the robot. The main problem we encountered with the Magellan Pro is that the robot could not move in a straight line, which we believe is due to the unequal wheel diameters. As a result, a sideways shift error is introduced to the robot. We will discuss this issue in detail in the latter part of this chapter.

4.3 The ceiling tile tracking system

The main objective of the ceiling tile tracking system, implemented by Jon Klippenstein [3], is to provide reliable ground-truth poses of a robot in order to evaluate the localization accuracy of various SLAM algorithms. A Dragonfly digital camera, manufactured by Point Grey Research Inc. [5], was placed on the upper deck of the Magellan Pro pointing upward as shown in Figure 4.2.

The communication between the robot and the camera is done via the Player software [25], which is a socket-based device server that allows control of a wide variety of the robot's sensors and actuators. It executes on a machine connected to the robot and offers a TCP socket interface to control clients. The control clients connect to the Player and communicate with the devices by

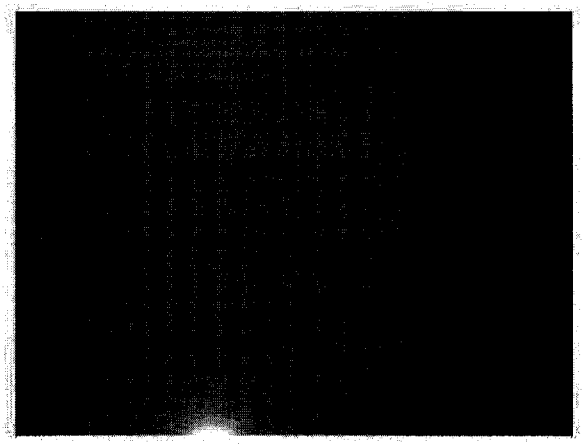

Figure 4.3: Ceiling tiles on the third floor in the Computing Science building. These tiles are divided by black support beams

exchanging messages over the TCP socket.

The system is built using feature detectors from OpenCV [4], which is an open-source computer vision library in C++ that includes implementations of many common vision algorithms. Ceiling tiles on the third floor in the Computing Science (CS) building are used for data collection. These large light-colored tiles are divided by black support beams as shown in Figure 4.3. When mounted on the robot, the camera is low enough to see nine ceiling tile corners, which are selected as initial corner points. The Dragonfly digital camera is programmed to take an image of the ceiling tiles as frequently as possible.

The captured image first has radial lens distortion removed so that lines formed by the ceiling tile support beams appear straight. The image is then used in the Canny Edge Detection process to find sharp contrasts in intensities that correspond to both sides of a support beam. This process significantly reduces the amount of data in the image while preserving the image's most important structural features. The Hough transform is then used to isolate lines in the image. Therefore, a set of points created by the intersection of each pair of lines can be clustered into possible ceiling tile corners. Each cluster is expected to consist of many points so the mean of all the points in the cluster is selected as a corner point. At least four corner points must exist in each image, and each corner point must be at least a certain distance away from all other points in the cluster in order for the system to continue. The system then uses these reference corner points to calculate the robot's ground-truth poses.

We assume that the ceiling tile tracking system provides reliable ground-truth poses of the robot. With this system, the robot could traverse for a long period around the building using ceiling corner points as references and come back to its starting position with an error of less than 5 centimeters, which is manually measured from the difference between the starting pose of the robot and its return pose.

4.4 Data discussion

The Magellan Pro robot was programmed to traverse on a tile surface of the third floor of the CS building autonomously using the ceiling corner points as reference points in order to simultaneously collect two data sets at each time step: the robot's true translational D and rotational movements T based on the ceiling tile tracking system and the robot's predicted translational $\bar{d}$ and rotational $\bar{t}$ movements from the odometry. Note that the ceiling tile tracking system actually reported the robot's current pose (x,y,θ) at each time step. We needed to calculate the amount of translational and rotational movements from each pair of the reported poses. Another alternative for the data gathering process would be to drive the robot manually using a joystick.

With these two data sets, we observed:

1) At each time step, the average amount of the translational movement reported from the odometry underestimated the average amount of the actual movement by the robot. Specifically, the average error between the two values was approximately a centimeter for the robot's movement of approximately 15 centimeters. On the other hand, the average amount of the rotational movement reported from the odometry overestimated the actual movement by the robot. Specifically, the average error between the two values was approximately 0.35 degrees for the robot's rotational movement of approximately 4 degrees.

2) The robot has a sideways shift to its right. If it is programmed to go straight, it always moved sideways to the right. The sideways shift is believed to be introduced from unequal wheel diameters of the Magellan Pro. As already mentioned, the robot's wheels are difficult to manufacture to exactly the same diameter. Even if they are, they compress differently under asymmetric load distribution. For example, the camera that was mounted on the robot could result in a different weight distribution on each wheel. When the robot was programmed to go straight, the command was sent to its two motors to spin at the same velocity. However, if the wheel diameters are unequal, spinning each wheel at the same velocity would result in the rotation of the robot. In our case, the left wheel diameter is believed to be larger than the right one. Therefore, when the robot traversed around using the ceiling corner points as reference points, it always turned briefly to its right and the ceiling tile tracking system needed to adjust the robot back with left turns. In other words, the robot did not move in a straight-line.

4.5 Using FastSLAM 2.0 to estimate the robot's true poses

We have quietly assumed throughout this thesis that we can use poses according FastSLAM 2.0 as the robot's estimated true poses. An important question arises: we need a motion model to run FastSLAM 2.0 at the first place, but how do we get that initial motion model? We suggest making an educated guess for an initial motion model. Then we can apply RLS and FastSLAM 2.0 to bootstrap our initial motion model to a more refined model as shown in Figure 4.4. We will discuss

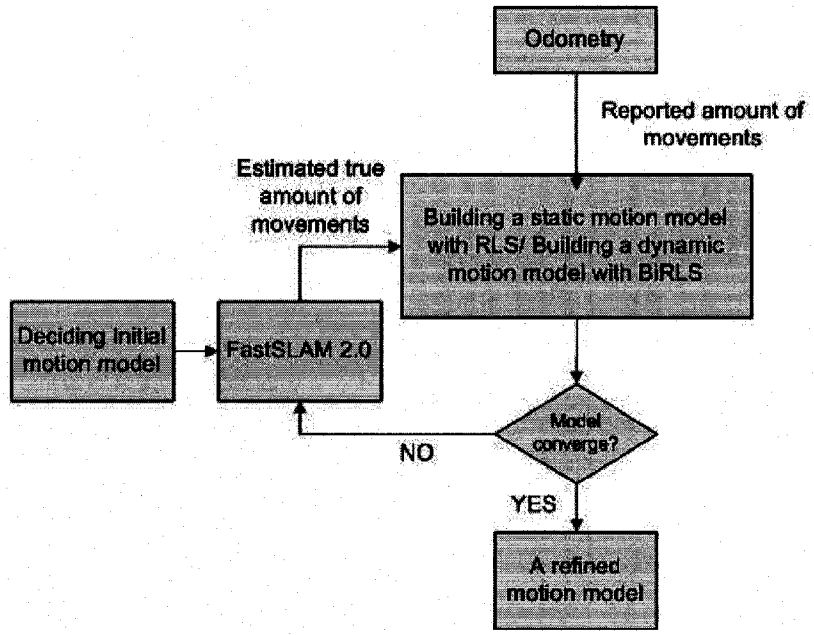

Figure 4.4: The procedure of using FastSLAM 2.0 to generate the robot's estimated true poses. We first need to determine an initial motion model that will be used by the FastSLAM 2.0. With the initial motion model, we can then apply our framework of building a motion model.

the choice of selecting an initial motion model and demonstrate that our framework can bootstrap a crude motion model to a more refined model in section 5.5.

Moreover, FastSLAM 2.0 does not report the translational movement (D) and the rotational (T) movement that we need to learn motion model parameters together with the reported translational movement (d) and the reported rotational movement (t) from odometry as discussed in section 3.3.2. FastSLAM 2.0 only reports, at time t, the estimated robot pose s_t, one for each particle. Therefore, we suggest using the robot's pose according to the mean of the particles in FastSLAM 2.0 as the robot's estimated true pose at each time step. From each pair of the robot's pose, we can calculate the amount of estimated true translational movement based on the Euclidean distance from (x_{t-1}, y_{t-1}) to (x_t, y_t). The amount of estimated true rotational movement can also be easily calculated from the different between the robot's heading at time t (Θ_t) and the robot's heading at time $t-1$ (Θ_{t-1}).

Our main motivation of using FastSLAM 2.0 to estimate the robot's true poses is to eliminate the need for explicit requirements of ground-truth poses. However, if we have a constrained environment (i.e. the environment with ceiling tiles in our case) that we can easily obtain the robot's true poses, we may not need to run FastSLAM 2.0 at all. The motion model can still be obtained sequentially from the robot's true poses generated from other sources (i.e. scan matching) using our framework. In other words, our framework is not limited only to the use of FastSLAM 2.0.

4.6 Uncertainties in the robot's movement

Recall that the motion model used throughout this thesis is defined as:

$$x' = x + Dcos(\theta + \frac{T}{2}) \tag{4.1}$$

$$y' = y + Dsin(\theta + \frac{T}{2}) \tag{4.2}$$

$$\theta' = \theta + T; \theta' \in (-\pi, \pi) \tag{4.3}$$

Given that $N(a, b^2)$ refers to a normal distribution with the mean a and the variance b^2, the true values of the translational movement D and the rotational movement T are defined as follows:

$$D \sim N(p_1 \cdot \bar{d} + p_2 \cdot \bar{t}, p_3 \cdot \bar{d}^2 + p_4 \cdot \bar{t}^2) \tag{4.4}$$

$$T \sim N(p_5 \cdot \bar{d} + p_6 \cdot \bar{t}, p_7 \cdot \bar{d}^2 + p_8 \cdot \bar{t}^2) \tag{4.5}$$

where $\bar{d}$ and $\bar{t}$ are the translational and rotational movements respectively according the odometry. The true translation and rotational movements are denoted as D and T respectively. p_i is a set the motion model parameters that need to be identified.

This motion model assumes that the true values of D and T (estimated from FastSLAM 2.0 with the use of an accurate motion model in our case) are distributed normally with respect to the reported values ($\bar{d}$ and $\bar{t}$). In other words, the differences between the true values and the reported values (the prediction errors) in our data are expected to follow normal distributions. Therefore, it is important to consider whether the prediction errors can reasonably be approximated by normal distributions. Figure 4.5 shows a histogram and the best fit Gaussian of the prediction error in the translational movement for the robot's movement of approximately every 15 centimeters. This prediction error is calculated from the estimated true translational movement from FastSLAM 2.0 and the reported translational movement from the odometry. Similarly, Figure 4.6 shows a histogram and the best fit Gaussian of the prediction error of the rotational movement for the robot's movement of approximately every 4 degrees. The Magellan Pro robot was programmed to automatically traverse approximately 300 meters on the third floor in the Computing Science building to collect these data. According to these graphs, our data do not largely deviate from the normal distribution and we believe the prediction errors can reasonably be approximated by normal distributions.

It is worth mentioning that there are many statistical tests for normality, such as the Geary's test, the D'Agostino-Pearson omnibus test, the Lilliefors test, or The Shapiro-Wilk normality test. Refer to [6] for a review on different normality tests. However, we believe that these statistical tests do not provide any meaningful information to us; minor deviations from normality may be flagged as statistically significant although these deviations would not dramatically affect our application. All these tests will probably reject the hypothesis that our data are normally distributed because more

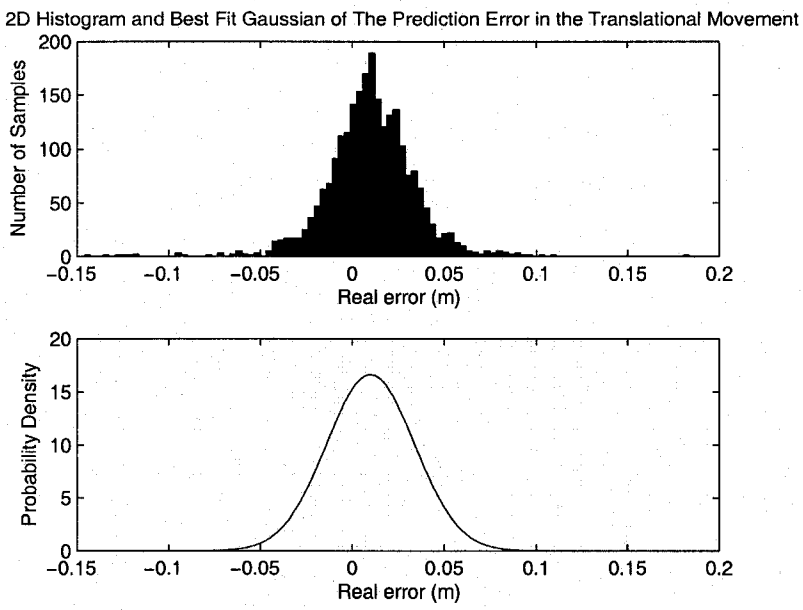

Figure 4.5: A histogram (Above) and the best fit Gaussian (Below) of the translational movement's prediction error

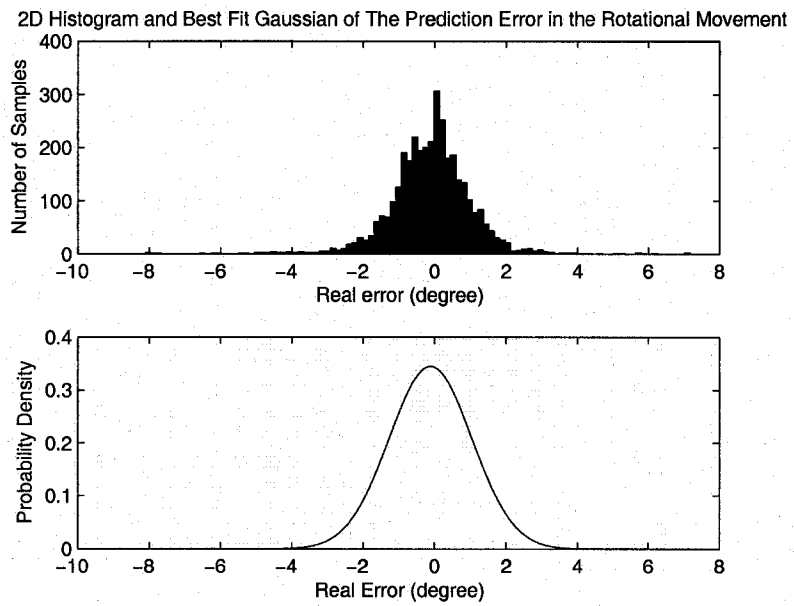

Figure 4.6: A histogram (Above) and the best fit Gaussian (Below) of the rotational movement's prediction error

than 5% of our data are two standard deviations away from the mean. In fact, many values are three or more standard deviations away from the mean. This is due to the nature of our application that odometry readings are not always reliable nor predictable.

4.7 Parameter estimation

We have discussed, from a mathematical point of view, the recursive least squares (RLS) and the bi-loop recursive least squares (BiRLS) in the previous chapter. In this section, we discuss in detail how these algorithms are used to learn motion model parameters. For building a static motion model with RLS, we stop learning the parameters as soon as they converge (i.e. when the difference between each pair of $\theta(t)$ and $\theta(t-1)$ in the vectors $\Theta(t)$ and $\Theta(t-1)$ is smaller than a predetermined threshold). On the other hand, because BiRLS is applied to build a dynamic motion model, it continues learning the parameters as long as the robot operates.

We assume that the robot reports at each time step the amount of translational ($\bar{d}$) and rotational ($\bar{t}$) movements from the odometry and the estimated true movements (D and T) from FastSLAM 2.0. With the given data above, our objective is to learn the motion model parameters p_i from:

$$D \sim N(p_1 \cdot \bar{d} + p_2 \cdot \bar{t}, p_3 \cdot \bar{d}^2 + p_4 \cdot \bar{t}^2) \tag{4.6}$$

$$T \sim N(p_5 \cdot \bar{d} + p_6 \cdot \bar{t}, p_7 \cdot \bar{d}^2 + p_8 \cdot \bar{t}^2) \tag{4.7}$$

where $N(a, b^2)$ refers to a normal distribution with the mean a and the variance b^2.

Here we review the least squares approach to estimate model parameters described in section 3.3.2. Equations (4.6) and (4.7) can be formulated to the least squares systems as follows:

$$y_D(t) \sim N(\Phi_{mD}(t)\Theta_{mD}, \Phi_{vD}(t)\Theta_{vD}) \tag{4.8}$$

$$y_T(t) \sim N(\Phi_{mT}(t)\Theta_{mT}, \Phi_{vT}(t)\Theta_{vT}) \tag{4.9}$$

Therefore, we apply four least squares systems to estimate eight model parameters p_i. The first least squares system estimates the mean parameters of the D term in (4.8), denoted as $\Theta_{mD} = [p_1 \ p_2]^T$, from $y'_D(t) = \Phi_{mD}(t)\Theta_{mD}$. The expected value of $y'_D(t)$ is the true amount of the translational movement D, and $\Phi_{mD}(t)$ refers to a row vector representing the reported odometry movements $[\bar{d} \ \bar{t}]$ at time t.

The second least squares system estimates the variance parameter of the D term in (4.8), denoted as $\Theta_{vD} = [p_3 \ p_4]^T$, from $y''_D(t) = \Phi_{vD}(t)\Theta_{vD}$. The expected value of $y''_D(t) = (\Phi_{mD}\Theta_{mD} - D)^2$. Moreover, $\Phi_{vD}(t)$ refers to a row vector representing the squared reported odometry movements $[\bar{d}^2 \ \bar{t}^2]$ at time t. The calculation is similar for the other two least squares systems that estimate the mean parameters $\Theta_{mT} = [p_5 \ p_6]^T$ and the variance parameters $\Theta_{vT} = [p_7 \ p_8]^T$ of the rotational movement term (4.8). Note that we always have two model parameters in all cases in this study.

4.7.1 Building a static motion model with the RLS algorithm

The motivation of applying the RLS algorithm is to allow motion model parameters to be updated as soon as new data arrive. Doing so allows RLS to detect when the model has actually converged and stop training. As already mentioned, it would also be more convenient if we could avoid the need for external measurements and explicit requirements of ground-truth poses. Therefore, the robot's ground-truth poses are estimated at each time step from the mean of the particles according to the FastSLAM 2.0 algorithm *after* resampling, instead of using the poses reported from the ceiling tile tracking system.

As stated in the previous chapter, the recursive form of the recursive least squares algorithm is given as follows:

$$K(t) = P(t-1)\Phi(t)^T(I + \Phi(t)P(t-1)\Phi(t)^T)^{-1} \tag{4.10}$$

$$\Theta(t) = \Theta(t-1) + K(t)(y(t) - \Phi(t)\Theta(t-1)) \tag{4.11}$$

$$P(t) = (I - K(t)\Phi(t))P(t-1) \tag{4.12}$$

where $y(t)$ and $\Phi(t)$ are a pair of observed variables at time t in which the row vector $\Phi(t) = [\phi_1(t)\ \phi_2(t)\ ...,\ \phi_M(t)]$ and $y(t)$ is a scalar. The column vector $\Theta(t) = [\theta_1(t)\ \theta_2(t)\ ...\ \theta_M(t)]^T$ contains a set of model parameters at time t, where M is the number of model parameters, which is always two in our case. $P(t)$ refers to the covariance matrix at time t.

We first need the initial values for $\Theta(0)$ and $P(0)$. Generally, if we believe $\Theta(0)$ is a very good first estimate, we assign a small value to $P(0)$; otherwise, $P(0)$ should be assigned a high value. Reference [14] defines $P(0)$ as:

$$P(0) = diag(1/K) \tag{4.13}$$

where $0 < K \leq 1$ is a scalar value. The higher the value of K is, the higher confidence we have in our initial value of $\Theta(0)$. Moreover, the size of $P(0)$ must be M x M.

4.7.1.1 Learning the mean parameters p_1 and p_2 of the D term

To estimate the mean parameters of the D term in (4.6), we set up the least squares system in (4.8) such that $y'_D(t) = \Phi_{mD}(t)\Theta_{mD}$ where $y'_D(t)$ refers to a scalar value of the estimated true translational movement D at time t calculated from the mean of the particles in the FastSLAM 2.0 algorithm. Moreover, $\Phi_{mD}(t)$ is a row vector representing the reported odometry movements $[\bar{d}\ \ \bar{t}]$ at time t. With these two input parameters $y'_D(t)$ and $\Phi_{mD}(t)$ together with $\Theta_{mD}(t-1)$ and $P_{mD}(t-1)$, we can recursively calculate the RLS solution of the mean parameters $\Theta_{mD}(t) = [p1\ p2]^T$ at each time step.

4.7.1.2 Learning the variance parameters p_3 and p_4 of the D term

The variance of the D term has a quadratic dependence on the reported odometry values ($\bar{d}$ and $\bar{t}$). To estimate the variance parameters of the D term in (4.6), we set up the other least squares system in (4.8) such that $y''_D(t) = \Phi_{vD}(t)\Theta_{vD}$. In this case, $y''_D(t)$ refers to $(\Phi_{mD}(t)\Theta_{mD}(t) - D)^2$, which is the square of the difference between the predicted translational movement at time t denoted as $\Phi_{mD}(t)\Theta_{mD}(t)$ (based on the mean parameters and the reported amount of movements reported from the odometry in section 4.7.1.1) and the estimated true translational movement at time t denoted as D, which is calculated from the mean of the particles in the FastSLAM 2.0 algorithm.

Moreover, $\Phi_{vD}(t)$ is a row vector representing the squared reported odometry movements $[\bar{d}^2\ \bar{t}^2]$ at time t. With these two input parameters $y''_D(t)$ and $\Phi_{vD}(t)$ together with $\Theta_{vD}(t-1)$ and $P_{vD}(t-1)$, we can recursively calculate the RLS solution of the variance parameters $\Theta_{vD} = [p3\ p4]^T$ at each time step.

4.7.1.3 Learning the mean parameters p_5 and p_6 of the T term

To estimate the mean parameters of the T term in (4.7), we set up the least squares system in (4.9) such that $y'_T(t) = \Phi_{mT}(t)\Theta_{mT}$ where $y'_T(t)$ refers to a scalar value of the estimated true rotational movement T at time t calculated from the mean of the particles in the FastSLAM 2.0 algorithm. Moreover, $\Phi_{mT}(t)$ is a row vector representing the reported odometry movements $[\bar{d}\ \ \bar{t}]$ at time t. With these two input parameters $y'_T(t)$ and $\Phi_{mT}(t)$ together with $\Theta_{mT}(t-1)$ and $P_{mT}(t-1)$, we can recursively calculate the RLS solution of the mean parameters $\Theta_{mT}(t) = [p5\ p6]^T$ at each time step.

4.7.1.4 Learning the variance parameters p_7 and p_8 of the T term

The variance of the T term has a quadratic dependence on the reported odometry values ($\bar{d}$ and $\bar{t}$). To estimate the variance parameters of the T term in (4.7), we set up the other least squares system in (4.9) such that $y''_T(t) = \Phi_{vT}(t)\Theta_{vT}$. In this case, $y''_T(t)$ refers to $(\Phi_{mT}(t)\Theta_{mT}(t) - T)^2$, which is the square of the difference between the predicted rotational movement at time t denoted as $\Phi_{mT}(t)\Theta_{mT}(t)$ (based on the mean parameters and the reported amount of movements reported from the odometry in section 4.7.1.3) and the estimated true rotational movement at time t denoted as T, which is calculated from the mean of the particles in the FastSLAM 2.0 algorithm.

Moreover, $\Phi_{vT}(t)$ is a row vector representing the squared reported odometry movements $[\bar{d}^2\ \bar{t}^2]$ at time t. With these two input parameters $y''_T(t)$ and $\Phi_{vT}(t)$ together with $\Theta_{vT}(t-1)$ and $P_{vT}(t-1)$, we can recursively calculate the RLS solution of the variance parameters $\Theta_{vT} = [p7\ p8]^T$ at each time step.

4.7.2 Building a dynamic motion model with the BiRLS algorithm

The BiRLS algorithm is applied to learn motion model parameters for a dynamic motion model because BiRLS can keep track of the changes in time-varying systems better than RLS at the cost of possibly slower convergence of parameter estimates. However, its slow convergence property does not play an important role in this context as long as the algorithm can keep adjusting the parameters to best reflect the current state of the robot and the environment (the surface it moves on, the level of the battery's power, or the inflation and wear of tires).

As stated in the previous chapter, BiRLS consists of two nested loops. The outer loop requires initial values of $\Theta(0)$ and the covariance matrix $P(0)$ and is defined as follows:

$$K(t) = P(t-1)\Phi(t)^T(I + \Phi(t)P(t-1)\Phi(t)^T)^{-1} \quad (4.14)$$

$$\Theta(t) = \Theta(t-1) + K(t)(y(t) - \Phi(t)\Theta(t-1)) \quad (4.15)$$

$$P(t) = (I - K(t)\Phi(t))P(t-1) \quad (4.16)$$

where $y(t)$ and $\Phi(t)$ are a pair of observed variables at time t in which the row vector $\Phi(t) = [\phi_1(t)\ \phi_2(t)\ ...,\ \phi_M(t)]$ and $y(t)$ is a scalar. The column vector $\Theta(t) = [\theta_1(t)\ \theta_2(t)\ ...\ \theta_M(t)]^T$ contains a set of model parameters at time t, where M is the number of model parameters.

The inner loop requires initial parameters values $\Theta_{in}(0)$ and the covariance matrix $P_{in}(0)$, together with a pair of observed variables y_{in} and Φ_{in}. These values are initialized from the outer loop as $\Theta_{in}(0) = \Theta(t)$, $P_{in}(0) = P(t)$, $y_{in} = y(t)$, and $\Phi_{in} = \Phi(t)$. The update step of the inner loop is as follows:

$$K_{in}(j) = P_{in}(j-1)\Phi_{in}^T(I + \Phi_{in}P_{in}(j-1)\Phi_{in}^T)^{-1} \quad (4.17)$$

$$\Theta_{in}(j) = \Theta_{in}(j-1) + K_{in}(j)(y_{in} - \Phi_{in}\Theta_{in}(j-1)) \quad (4.18)$$

$$P_{in}(j) = (I - K_{in}(j)\Phi_{in})P_{in}(j-1) \quad (4.19)$$

The inner loop executes J times, where J denotes a predetermined integer.

Because learning motion model parameters with RLS and BiRLS requires the same input parameters at each time step, the methods described below is identical to 4.7.1.1-4.7.1.4 and are listed only for completeness.

4.7.2.1 Learning the mean parameters p_1 and p_2

To estimate the mean parameters of the D term in (4.6), we set up the least squares system in (4.8) such that $y'_D(t) = \Phi_{mD}(t)\Theta_{mD}$ where $y'_D(t)$ refers to a scalar value of the estimated true translational movement D at time t calculated from the mean of the particles in the FastSLAM 2.0 algorithm. Moreover, $\Phi_{mD}(t)$ is a row vector representing the reported odometry movements

$[\bar{d} \;\; \bar{t}]$ at time t. With these two input parameters $y'_D(t)$ and $\Phi_{mD}(t)$ together with $\Theta_{mD}(t-1)$ and $P_{mD}(t-1)$, we can recursively calculate the RLS solution of the mean parameters $\Theta_{mD}(t) = [p1 \; p2]^T$ at each time step.

4.7.2.2 Learning the variance parameters p_3 and p_4

The variance of the D term has a quadratic dependence on the reported odometry values ($\bar{d}$ and $\bar{t}$). To estimate the variance parameters of the D term in (4.6), we set up the other least squares system in (4.8) such that $y''_D(t) = \Phi_{vD}(t)\Theta_{vD}$. In this case, $y''_D(t)$ refers to $(\Phi_{mD}(t)\Theta_{mD}(t) - D)^2$, which is the square of the difference between the predicted translational movement at time t denoted as $\Phi_{mD}(t)\Theta_{mD}(t)$ (based on the mean parameters and the reported amount of movements reported from the odometry in section 4.7.2.1) and the estimated true translational movement at time t denoted as D, which is calculated from the mean of the particles in the FastSLAM 2.0 algorithm.

Moreover, $\Phi_{vD}(t)$ is a row vector representing the squared reported odometry movements $[\bar{d}^2 \; \bar{t}^2]$ at time t. With these two input parameters $y''_D(t)$ and $\Phi_{vD}(t)$ together with $\Theta_{vD}(t-1)$ and $P_{vD}(t-1)$, we can recursively calculate the RLS solution of the variance parameters $\Theta_{vD} = [p3 \; p4]^T$ at each time step.

4.7.2.3 Learning the mean parameters p_5 and p_6 of the T term

To estimate the mean parameters of the T term in (4.7), we set up the least squares system in (4.9) such that $y'_T(t) = \Phi_{mT}(t)\Theta_{mT}$ where $y'_T(t)$ refers to a scalar value of the estimated true rotational movement T at time t calculated from the mean of the particles in the FastSLAM 2.0 algorithm. Moreover, $\Phi_{mT}(t)$ is a row vector representing the reported odometry movements $[\bar{d} \;\; \bar{t}]$ at time t. With these two input parameters $y'_T(t)$ and $\Phi_{mT}(t)$ together with $\Theta_{mT}(t-1)$ and $P_{mT}(t-1)$, we can recursively calculate the RLS solution of the mean parameters $\Theta_{mT}(t) = [p5 \; p6]^T$ at each time step.

4.7.2.4 Learning the variance parameters p_7 and p_8 of the T term

The variance of the T term has a quadratic dependence on the reported odometry values ($\bar{d}$ and $\bar{t}$). To estimate the variance parameters of the T term in (4.7), we set up the other least squares system in (4.9) such that $y''_T(t) = \Phi_{vT}(t)\Theta_{vT}$. In this case, $y''_T(t)$ refers to $(\Phi_{mT}(t)\Theta_{mT}(t) - T)^2$, which is the square of the difference between the predicted rotational movement at time t denoted as $\Phi_{mT}(t)\Theta_{mT}(t)$ (based on the mean parameters and the reported amount of movements reported from the odometry in section 4.7.2.3) and the estimated true rotational movement at time t denoted as T, which is calculated from the mean of the particles in the FastSLAM 2.0 algorithm.

Moreover, $\Phi_{vT}(t)$ is a row vector representing the squared reported odometry movements $[\bar{d}^2 \; \bar{t}^2]$ at time t. With these two input parameters $y''_T(t)$ and $\Phi_{vT}(t)$ together with $\Theta_{vT}(t-1)$ and $P_{vT}(t-1)$, we can recursively calculate the RLS solution of the variance parameters $\Theta_{vT} = [p7 \; p8]^T$ at each time step.

4.8 Summary

In this chapter, we first discussed the Magellan Pro robot and the ceiling tile tracking system. Moreover, the data gathering process was discussed. The robot was programmed to traverse automatically using the ceiling corner points as references to simultaneously obtain two data sets: the robot's true translational D and rotational movements T based on the ceiling tile tracking system and the robot's predicted translational $\bar{d}$ and rotational $\bar{t}$ movements from the odometry. We then suggested that poses according to FastSLAM 2.0 could be used to estimate ground-truth poses to learn motion model parameters. Therefore, the movements according to the ceiling tile tracking system (true movements) and the reported movements from the odometry (predicted movements) were used to generate movements according to the FastSLAM 2.0 algorithm. In addition, we discussed that the prediction errors, which were calculated from the ground-truth poses and the reported poses, could reasonably be approximated by normal distributions.

Next, we described the process of learning motion model parameters with RLS and BiRLS. RLS was used to build a static motion model using the robot's poses according to FastSLAM 2.0 as the estimated ground-truth poses. Doing so results in the system with certain level of autonomy because we can avoid the need for external measurements and explicit requirements of ground-truth poses. We could just let the robot traverse the environment running SLAM. The robot will determine by itself whether it has built a static motion model for the particular environment. Similarly, BiRLS was applied to learn motion model parameters using the robot's poses according to FastSLAM 2.0 as the estimated ground-truth pose on the fly the robot operates. In the next chapter, we will test our approaches in a simulation using control data collected from the Magellan Pro robot.

Chapter 5

Experiments

5.1 Introduction

In the previous chapter, we described the process of learning motion model parameters with RLS and BiRLS. In this chapter, we conduct two main experiments using control data collected from a robot to test our approaches. In the first experiment, our static motion model built with RLS is compared to other motion models that vary the means and the variances (of the D and T terms) of our motion model in the context of FastSLAM 2.0's localization accuracy. In the second experiment, we compare our dynamic motion model built with BiRLS to the static motion model for the robot operating on a dynamically changing ground surface. Later in this chapter, we discuss some of the interesting issues related to our experimental results, including a role of the motion model in generating the FastSLAM 2.0's improved proposal distribution and the benefits of well-calibrated mean and variance of the proposal distribution. Lastly, we demonstrate that our framework can start with a crude initial motion model and bootstrap itself towards a more refined motion model.

5.2 Experimental setup

Our approaches were tested in a simulation using control data collected from the Magellan Pro robot. Specifically, we first programmed the robot to traverse on a tile surface on the third floor of the Computing Science building to simultaneously collect two data sets: the robot's poses according to odometry readings and ground-truth poses according to the ceiling tile tracking system. These two data sets were used in the simulation. From each of the ground-truth poses, we simulated measurement readings of a 180 degree generic laser range finder. Each scan consists of range and bearing measurement to simulated landmarks within 30 meters with the range uncertainty of 10 centimeters and the bearing uncertainty of 1 degree. Range and bearing uncertainties can generally be assumed to be constant (independent of the measured range), assuming sufficient reflectivity on a landmark. Simulating the measurement readings prevents scenarios where sensors are unable to find good robot pose estimates (noisy measurement readings), which could influence the comparison among different motion models.

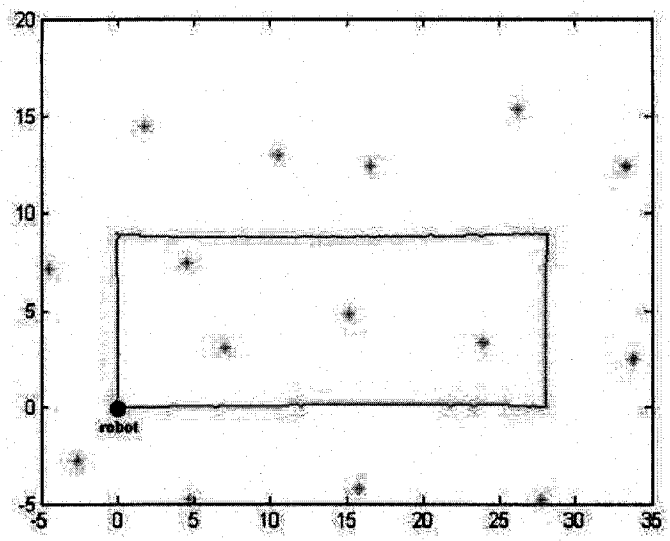

Figure 5.1: The environment used for our experiments. Both x and y axes are in meters. The dark circle denotes the robot's starting position and the stars refer to landmarks whose positions were predefined. The solid path indicates the true trajectory of the robot, which comes from the ceiling tile tracking system.

All experiments in this chapter were performed in the environment described below unless otherwise stated. There were 15 simulated landmarks whose positions were predefined as shown in Figure 5.1. Each case simulated the robot traversing one round on the third floor of the Computing Science building (approximately 80 meters) on tiles and was repeated for 50 trials to compute the RMS pose error along the entire trajectory by comparing the robot's pose according to the FastSLAM 2.0 algorithm to the pose reported from the ceiling tile tracking system (the ground-truth pose) at each time step. The error was recorded every time the robot had moved approximately 15 centimeters or had turned approximately 10 degrees. We assumed known data associations to prevent wrong landmark correspondences from influencing the comparison among motion models.

5.3 Experimental results

In this section, we first demonstrate the effectiveness of our static motion model built with RLS (using the robot's true poses generated from FastSLAM 2.0 with the use of an accurate motion model) compared with other six motion models that vary the mean and the variance of our model in the environment described in section 5.2. The goal here is to demonstrate the effectiveness of our model and show that having an inaccurate mean and/or inaccurate variance of the proposal distribution significantly affects the FastSLAM 2.0's localization accuracy. We then show the effectiveness of our dynamic motion model built with BiRLS when the robot operates on a dynamically changing ground surface in order to validate that BiRLS can adapt model parameters to reflect the current condition of the environment.

5.3.1 Effectiveness of our static motion model built with RLS

In this section, we show the effectiveness of our motion model built with RLS compared to six other motion models with respect to the number of particles from 10 to 100 particles (with an increment of 10).

Model 0-RLS refers to the motion model whose parameters in (2.4) and (2.5) were learned from the RLS algorithm as discussed in the previous chapter. The training data (FastSLAM 2.0's poses and the poses reported from the odometry) were obtained from the process discussed in section 4.4. It is worth noting that learning motion model parameters with ground-truth poses from FastSLAM 2.0 or from the ceiling tile tracking system does not really affect final parameter estimates because FastSLAM 2.0 can successfully localize the robot along the trajectory. We will discuss this issue in section 5.4.3. In addition, Model 1-OV and Model 1-OM refer to the motion models whose variances and means of the D and T terms in (2.4) and (2.5) were artificially increased by 50% respectively from those of the Model 0-RLS. Model 1-OMV refers to the motion model whose means and variances of the D and T terms in (2.4) and (2.5) were *both* artificially increased by 50 %.

Model 2-UV and Model 2-UM refer to the motion models whose variances and means of the D and T terms in (2.4) and (2.5) were artificially decreased by 50% respectively from those of the Model 0-RLS. Model 2-UMV refers to the motion model whose means and variances of the D and T terms in (2.4) and (2.5) were *both* artificially decreased by 50 %.

The experimental results of the FastSLAM 2.0's localization accuracy of each model set (Model O, Model 1,and Model 2) are shown in Figure 5.2 and Figure 5.3. Their corresponding data are listed in Table 5.1 and Table 5.2 respectively. As the result in Figure 5.2 indicates, the average RMS pose error of the resulting path from FastSLAM 2.0 with Model 1-OM and Model 1-OMV were 0.509 and 0.346 meters respectively, which were reduced by the use of Model 0-RLS to 0.118 meters (The improvements were approximately 330% and 193% respectively). However, it is statistically impossible to distinguish the accuracy of FastSLAM 2.0 with Model 0-RLS and that with the use of Model 1-OV. Moreover, the result in Figure 5.3 indicates that the average RMS pose error of the resulting path from FastSLAM 2.0 with Model 2-UV, Model 2-UM, and Model 2-UMV were 0.179, 0.736 and 1.817 meters respectively, which were also reduced by the use of Model 0-RLS to 0.118 (The improvements were approximately 52%, 535%, and 1440% respectively). These experimental results indicate that the use of motion model built with the recursive least squares algorithm significantly improves the FastSLAM 2.0's localization accuracy in most cases.

However, the level of improvements surprised us. We did not expect the motion model to play such an important role in the accuracy of FastSLAM 2.0 whose improved proposal distribution also takes the most recent measurement into account. With the improved proposal distribution, we expected that accurate measurements could always correct poor pose estimates from the motion model and that the improved proposal distribution would match the posterior more closely. Moreover, as

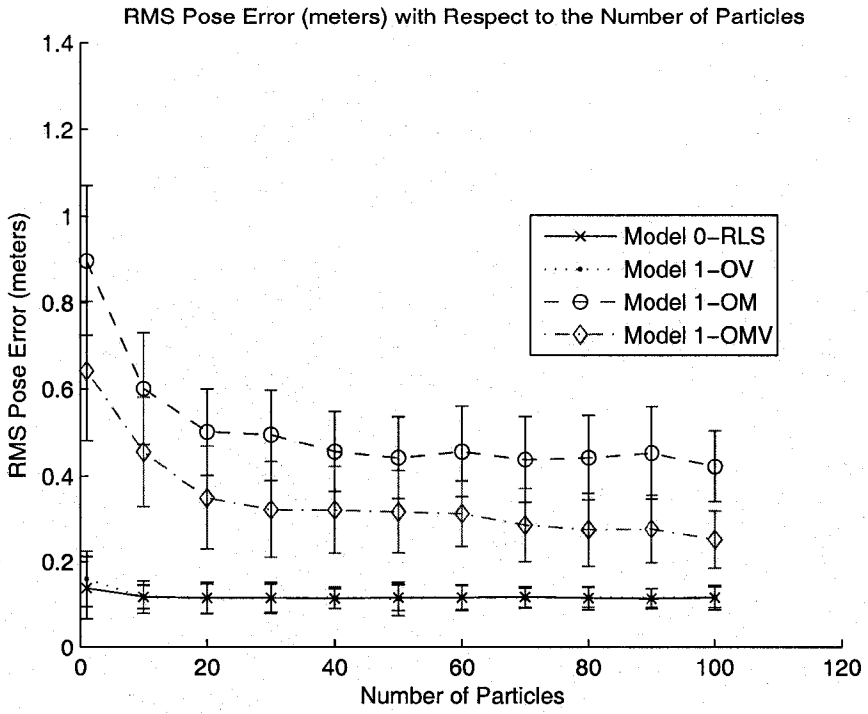

Figure 5.2: The FastSLAM 2.0's RMS pose error along the entire trajectory with the use of Model 0-RLS, Model 1-OV, Model 1-OM, and Model 1-OMV.

	RMS pose error in meter (standard deviation)			
# of Particles	**Model 0-RLS**	**Model 1-OV**	**Model 1-OM**	**Model 1-OMV**
1	0.139 (0.073)	0.146 (0.065)	0.897 (0.173)	0.642 (0.161)
10	0.118 (0.028)	0.117 (0.038)	0.601 (0.129)	0.455 (0.126)
20	0.115 (0.037)	0.114 (0.035)	0.501 (0.099)	0.349 (0.119)
30	0.115 (0.034)	0.116 (0.037)	0.494 (0.103)	0.322 (0.112)
40	0.114 (0.024)	0.116 (0.026)	0.456 (0.092)	0.321 (0.101)
50	0.116 (0.031)	0.113 (0.039)	0.442 (0.094)	0.317 (0.096)
60	0.116 (0.033)	0.117 (0.028)	0.457 (0.104)	0.313 (0.077)
70	0.118 (0.025)	0.115 (0.024)	0.438 (0.097)	0.286 (0.086)
80	0.115 (0.028)	0.117 (0.023)	0.442 (0.102)	0.275 (0.085)
90	0.114 (0.024)	0.116 (0.022)	0.453 (0.106)	0.277 (0.079)
100	0.116 (0.029)	0.118 (0.025)	0.442 (0.081)	0.252 (0.067)
	0.118 ♠	0.120 ♠	0.509 ♠	0.346 ♠

Table 5.1: Experimental results of the localization accuracy of FastSLAM 2.0 with Model 0-RLS, Model 1-OV, Model 1-OM, and Model 1-OMV respectively with respect to the number of particles. The table shows the RMS pose error(m) with the standard deviation of 50 independent trials in each case. Moreover, the ♠ refers to the average of the RMS pose error for each model calculated from the RMS pose error of each number-of-particles case.

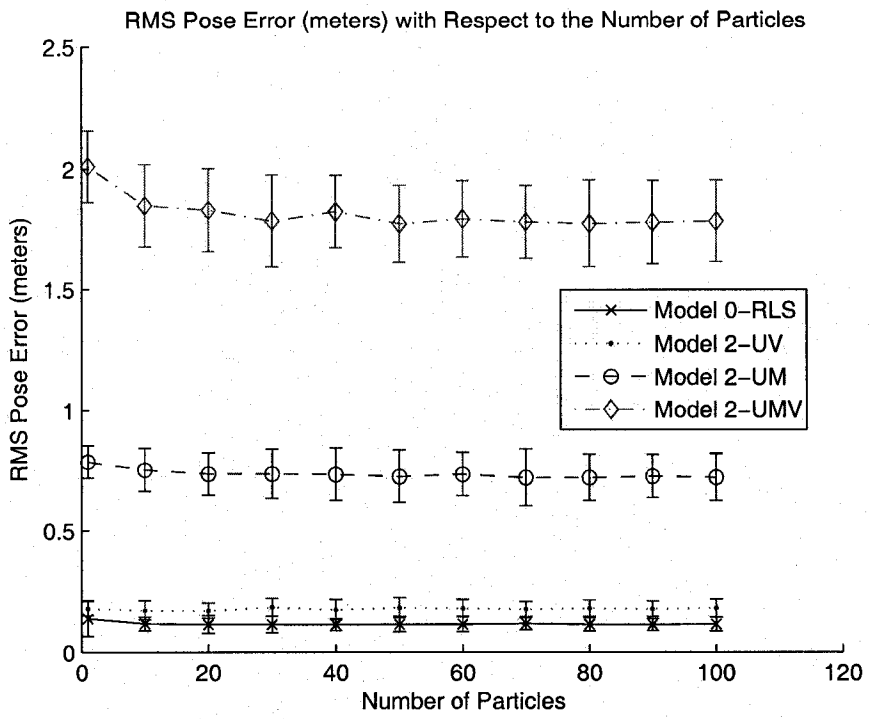

Figure 5.3: The FastSLAM 2.0's RMS pose error along the entire trajectory with the use of Model 0-RLS, Model 2-UV, Model 2-UM, and Model 2-UMV.

	RMS pose error in meter (standard deviation)			
# of Particles	**Model 0-RLS**	**Model 2-UV**	**Model 2-UM**	**Model 2-UMV**
1	0.139 (0.073)	0.161 (0.029)	0.786 (0.066)	2.011 (0.148)
10	0.118 (0.028)	0.173 (0.039)	0.753 (0.088)	1.849 (0.171)
20	0.115 (0.037)	0.172 (0.031)	0.736 (0.087)	1.830 (0.172)
30	0.115 (0.034)	0.186 (0.037)	0.738 (0.101)	1.785 (0.191)
40	0.114 (0.024)	0.176 (0.042)	0.434 (0.107)	1.824 (0.150)
50	0.116 (0.031)	0.184 (0.041)	0.726 (0.108)	1.774 (0.161)
60	0.116 (0.033)	0.181 (0.037)	0.735 (0.089)	1.794 (0.159)
70	0.118 (0.025)	0.178 (0.037)	0.721 (0.117)	1.781 (0.152)
80	0.115 (0.028)	0.181 (0.034)	0.724 (0.096)	1.774 (0.181)
90	0.114 (0.024)	0.179 (0.031)	0.726 (0.089)	1.779 (0.174)
100	0.116 (0.029)	0.182 (0.036)	0.722 (0.097)	1.784 (0.170)
	0.118 ♠	0.179 ♠	0.736 ♠	1.817 ♠

Table 5.2: Experimental results of the localization accuracy of FastSLAM 2.0 with Model 0-RLS, Model 2-UV, Model 2-UM, and Model 2-UMV respectively as with respect to the number of particles. The table shows the RMS pose error(m) with the standard deviation of 50 independent trials in each case. Moreover, the ♠ refers to the average of the RMS pose error for each model calculated from the RMS pose error of each number-of-particles case.

Motion model	RMS Pose Error in meter (std)
Static motion model (RLS)	0.127(0.021)
Dynamic motion model (BiRLS)	0.106(0.018)

Table 5.3: Summary of FastSLAM 2.0's RMS pose error(m) using our static motion model (built from RLS) and our dynamic motion model (built from BiRLS). This experiment was run for 50 independent trials and each with 10 particles.

the result in Figure 5.2 indicates, an increase in the variance of the proposal distribution did not strongly affect the localization accuracy of the FastSLAM 2.0 algorithm. Similarly, an increase in the number of particles did not significantly affect its localization accuracy in most cases (Figure 5.2 and Figure 5.3). We believe that these interesting issues are due to the nature of FastSLAM 2.0 and are not caused by the learning algorithm (RLS). These issues will be discuss in detail in a later section of this chapter.

5.3.2 Effectiveness of our dynamic motion model built with BiRLS

In this section, we compare the static RLS motion model (from the previous section) with the dynamic BiRLS motion model. Experiments were performed in the environment with 15 simulated landmarks whose positions were predefined. Each case simulated the robot traversing one round on the third floor of the Computing Science building (approximately 80 meters) in areas with approximately 60% tile and 40% carpet as shown in Figure 5.4 and was repeated for 50 trials to calculate the root mean square (RMS) error along the entire trajectory.

The RLS motion model was built from the data set collected on the tile surface previously discussed in section 4.4, and their parameters were not allowed to change throughout the robot's operation [1]. The BiRLS motion model, on the other hand, learned the motion model parameters on the fly as the robot operates using the most recent pose from FastSLAM 2.0 as the ground-truth pose at each time step. As the results in Table 5.3 indicate, the FastSLAM 2.0's RMS pose error along the entire trajectory from using of the BiRLS dynamic motion model was approximately 20% lower than that from using of the RLS static motion model. This experimental result indicates that the BiRLS dynamic motion model can keep track of changes in the environment when the robot operates on a dynamically changing ground surface.

[1] The purpose of this experiment is to compare a static motion model to a dynamic motion model. We are not interested in comparing RLS and BiRLS in term of their abilities to adapt. As a result, we first built a static motion model with RLS and used this motion model in FastSLAM 2.0 in this experiment.

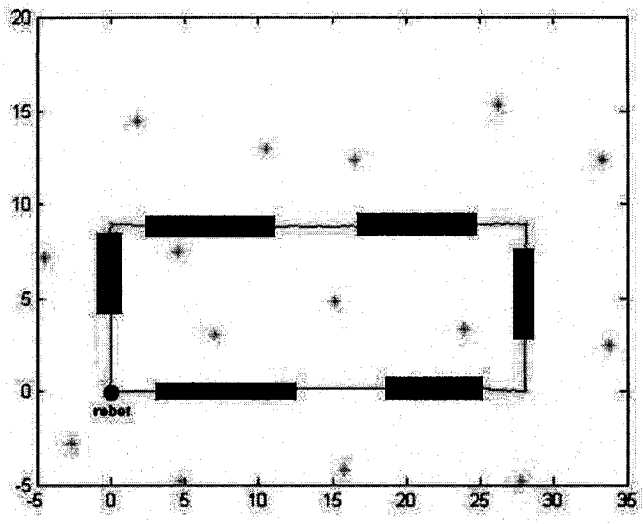

Figure 5.4: The environment used for our experiment to show the effectiveness of our dynamic motion model for the robot operating on a dynamically changing ground surface. Both x and y axes are in meters. The dark circle denotes the robot's starting position and the stars refer to landmarks whose positions were predefined. The solid path indicates the true trajectory of the robot, which comes from the ceiling tile tracking system. Dark squares along the trajectory indicates carpeted areas. The rest are tiled areas.

5.4 Discussions

In this section, we discuss interesting issues from this research. Note that all discussion here depends on the assumption that the likelihood distribution (i.e. a distribution that is generated from a sensor model) accurately represents the robot's true pose with low uncertainty. We believe this is a valid assumption in the case of many common laser range-finders. Moreover, we would not use FastSLAM 2.0 in the first place if we have poor sensors because we may end up integrating inaccurate information to our prediction (i.e. proposal distribution). However, if the likelihood distribution wrongly represents the robot pose (i.e. poor sensors), then the conclusions and observations presented here must be restated. First, we explain a role of the motion model in generating the improved proposal distribution of the FastSLAM 2.0 algorithm. With this understanding, we then discuss our experimental results from sections 5.3.1 and 5.3.2. Next, we set up experiments to demonstrate that we could use the poses generated from FastSLAM 2.0 as the robot's estimated true poses to learn motion model parameters. We then summarize the benefits of having an accurate mean and variance of the proposal distribution for the FastSLAM 2.0's algorithm. Lastly, we discuss Eliazar and Parr's motion model, which is proposed to handle sideways shifts, and demonstrate its accuracy.

5.4.1 A role of the motion model in FastSLAM 2.0

As mentioned in section 5.3.1, we did not expect the motion model to play such an important role in the accuracy of the FastSLAM 2.0 algorithm as indicated by our experimental results in Figure

5.2 and Figure 5.3. We expected that accurate measurements could correct poor pose estimates from the motion model. However, we did not take into consideration that the improved proposal distribution is not just the estimated likelihood distribution. As previously stated in chapter 3, the improved proposal distribution is generated from both the likelihood distribution and the proposal distributions:

$$s_t^{[m]} = p(s_t|s^{t-1,[m]}, u^t, z^t, n^t) \quad (5.1)$$

$$= \eta^{[m]} \int p(z_t|\Theta_k, s_t, n_t) p(\Theta_k|s^{t-1,[m]}, z^{t-1}, n^{t-1}) d\Theta_{n_t} \; p(s_t|s_{t-1}^{[m]}, u_t) \quad (5.2)$$

where $p(s_t|s_{t-1}^{[m]}, u_t)$ is the part that incorporates predicted poses from the motion model into the improved proposal distribution. FastSLAM 2.0 uses a Gaussian distribution to approximate (5.2). This Gaussian is computed for each particle in the same manner as EKF [44], [28], [42]. Refer to the appendix section for more detail.

FastSLAM 2.0 assumes that the uncertainties in landmark estimates can also be approximated by Gaussian distributions. Therefore, each of the N Gaussians representing the robot's pose (the improved proposal distribution) is then incrementally updated by the K Gaussians representing landmark estimates, where N and K are the number of particles and the number of observed landmarks respectively. Therefore, with the assumption that the robot observes at least one of the previously-seen landmarks at each time step, the calculation of the improved proposal distribution is similar to the calculation of the posterior distribution in EKF localization [49] that the robot pose estimate is incrementally updated for each observation on a previously-seen landmark. If the robot only observes new landmarks, the improved proposal distribution cannot be computed because there is no measurement information to correct the robot's predicted pose.

Figure 5.5 and Figure 5.6 illustrate the EKF localization for simplistic one-dimensional localization scenarios with two different motion models. In Figure 5.5, the robot predicts its current pose by the proposal distribution generated from a fairly accurate motion model and then acquires the likelihood distribution from its accurate sensor. These two distributions are assumed to be Gaussians. Combining the proposal and the likelihood distributions yields another Gaussian, known as the posterior distribution. This distribution represents the robot's pose according to EKF localization. Note that the uncertainty of the posterior distribution is smaller than both contributing Gaussians (the proposal and the likelihood), which is a general characteristic of information integration in Kalman filters [53].

Another important characteristic of the EKF lies in the relationship between the innovation vector and the Kalman gain. In brief, the innovation vector is the difference between the observed measurement and the predicted one calculated based on the robot's predicted pose and its pose uncertainty. The Kalman gain additionally scales the innovation vector. The more certain the observation (compared to the prediction), the higher the Kalman gain [53], and hence the stronger the

resulting pose correction. In other words, if the measurement noise is much smaller than that of the prediction, the robot pose estimate will be based much more on the observation than the prediction. See the appendix section for a more detailed discussion on this issue.

If we assume that the likelihood distribution accurately represents the robot's true pose and the EKF's posterior distribution is the FastSLAM 2.0's improved proposal distribution, drawing samples from the improved proposal distribution in Figure 5.5 allows most of the particles to focus in the areas with high likelihoods of representing the robot's true pose, which slows down the loss of particle diversity problem as discussed in section 3.2.2. On the other hand, if the means of the proposal and the likelihood are further away as shown in Figure 5.6, the resulting improved proposal distribution could barely intersect the peak of the likelihood distribution. Drawing samples from this improved proposal distribution possibly results in only a few particles having high importance weights because only a fraction of the drawn samples would cover the regions that have high likelihoods. Having only a few particles with high importance weights could speed up the loss up particle diversity and may prevent the algorithm from keeping track of the robot's true path. A similar scenario from Figure 5.6 occurs when the mean of the proposal distribution is completely displaced from the mean of the likelihood distribution. For example, wheel slippage could cause the motion model to create the proposal distribution that is further away from the likelihood distribution.

Because the FastSLAM 2.0's improved proposal distribution is generated in the same manner as the posterior distribution in EKF localization, the uncertainty in the improved proposal distribution is also smaller than the uncertainties in the original proposal and likelihood distributions. We will discuss this fact in the appendix section. According to [53], this fact is natural since integrating two independent estimates should make the robot more certain than each estimate in isolation. Related discussion on this topic can be found in [19]. Moreover, we believe that the more accurate the original proposal distribution[2] (generated from the motion model) is, the more accurate the FastSLAM 2.0's improved proposal distribution would be in representing the robot's true pose [3]. Note that this fact applies for EKF localization as well. The more accurate the proposal distribution is, the more accurate the EKF localization algorithm. Having the accurate improved proposal distribution results in most of the particles focusing in the areas representing the true pose, which slows down the loss of particle diversity problem and improves the accuracy of the FastSLAM 2.0 algorithm in the long run.

5.4.2 Discussion on our experimental results

In this section, we discuss some of the results from section 5.3.1 with the assumption that sensor information is precise. In other words, we assume that the likelihood distribution accurately represents

[2] An accurate proposal distribution refers to the proposal distribution whose mean is close to to the peak of the likelihood distribution, assuming that the sensor information is precise with low uncertainty.

[3] We are not saying that the more accurate motion model is, the more accurate the FastSLAM 2.0 algorithm. The motion model does more than just generating a proposal distribution. It is also used to compute the target distribution in importance sampling.

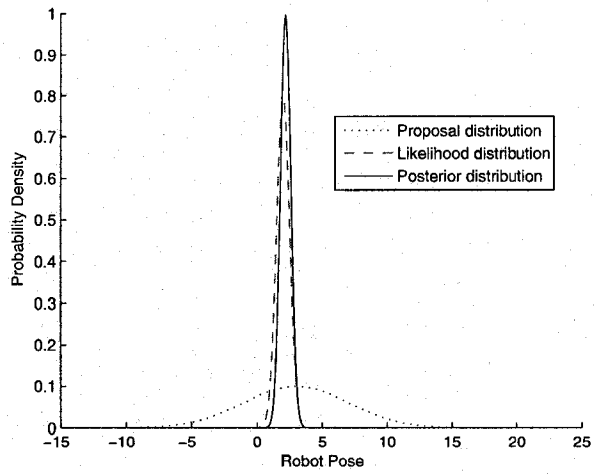

Figure 5.5: Illustration of the EKF localization. The proposal distribution accurately represents the robot's true pose. This proposal distribution together with an accurate likelihood distribution result in an accurate posterior distribution in EKF localization. Here we assume that the EKF's posterior distribution is the FastSLAM 2.0's proposal distribution.

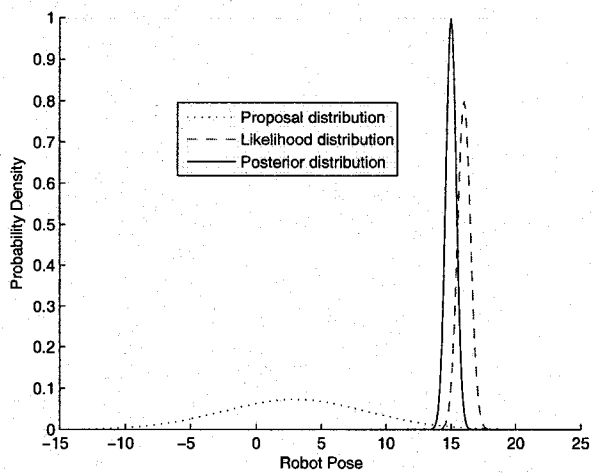

Figure 5.6: Illustration of the EKF localization. The mean of the proposal distribution further from the mean of the likelihood distribution. This proposal distribution together with an accurate likelihood distribution result in a posterior distribution in EKF localization that is slightly less accurate. Here we assume that the EKF's posterior distribution is the FastSLAM 2.0's proposal distribution.

the robot's true pose.

Figure 5.7 represents a possible scenario where the proposal distribution is generated from Model 0-RLS. Because the mean of the proposal distribution is not far away from the peak of the likelihood distribution, the improved proposal distribution can cover the areas representing the robot's true pose, which is again assumed to be the area represented by the likelihood distribution. Figure 5.8 represents a possible scenario when Model 1-OM, which generates the proposal distribution whose mean was artificially increased by 50%, is used. In this case, the means of the proposal and the likelihood are further away. Fortunately, the uncertainty in the proposal distribution is high compared to that of the likelihood distribution. Therefore, the improved proposal distribution is generated based on more of the observation than the prediction based on the relationship between the Kalman gain and the innovation vector as previously discussed. The resulting improved distribution can still cover some areas representing the robot's true pose. However, these areas are smaller than that from using Model 0-RLS. This could result in only a few particles having high importance weights and could speed up the loss of particle diversity, and the FastSLAM 2.0 could become inconsistent, which results in large RMS pose error along the entire trajectory.

Figure 5.9 demonstrates a possible scenario when Model 2-UMV, which generates the proposal distribution whose mean and variance were artificially increased by 50%, is used. In this scenario, the robot's pose according to the mean of the proposal distribution is highly inaccurate. Moreover the uncertainty of the proposal distribution is much smaller than other cases we have discussed so far. Thus, the improved proposal distribution is generated with less correction from the observation. As a result, the resulting improved proposal distribution can only represent the small areas representing the robot's true pose, which results in very large RMS pose error.

As previously mentioned, the localization accuracy of FastSLAM 2.0 is statistically indistinguishable whether using Model 0-RLS or Model 1-OV, which was surprising at first. However, an increase in the variance of the proposal distribution implies that we *trust* even more on the observation compared to the case from using Model 0-RLS. This fact could explain the result we obtained from the experiment. Lastly, section 5.3.2 shows that the dynamic motion model built with BiRLS could keep track of changes in time-varying parameters for the robot operated on a dynamically changing ground surface better than the static motion model built with RLS. In this scenario, the mean of the proposal distribution was adjusted to reflect the current ground-surface at each time step, and this resulted in a more accurate improved proposal distribution and a more accurate FastSLAM 2.0's localization accuracy.

5.4.3 A comparison between the ceiling tile tracking system and the FastSLAM 2.0 algorithm

This section discusses our choice of using the poses generated from FastSLAM 2.0 as the robot's estimated true poses to learn motion model parameters, instead of using the poses according to

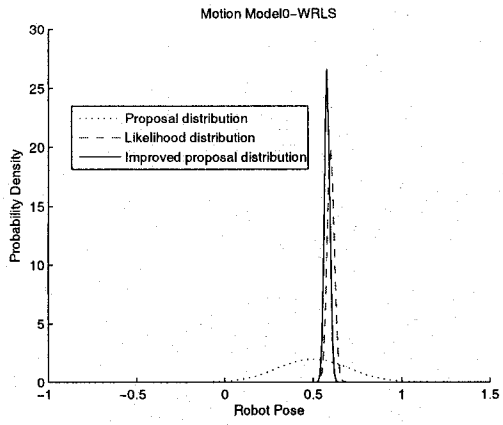

Figure 5.7: A possible scenario to illustrate the FastSLAM 2.0's improved proposal distribution generated from the model 0-RLS motion model and accurate laser measurement

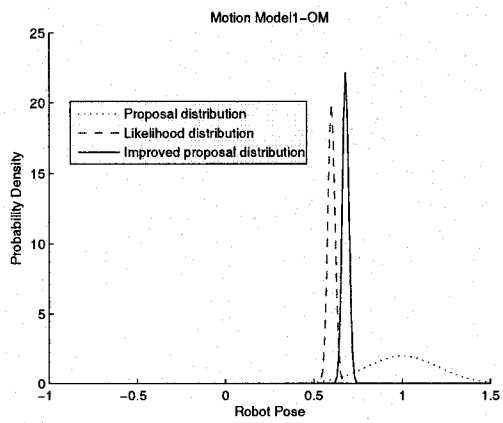

Figure 5.8: A possible scenario to illustrate the FastSLAM 2.0's improved proposal distribution generated from the model 1-OM motion model and accurate laser measurement

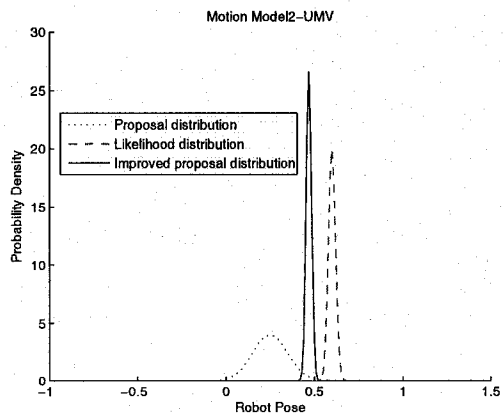

Figure 5.9: A possible scenario to illustrate the FastSLAM 2.0's improved proposal distribution generated from the model 2-UMV motion model and accurate laser measurement

the ceiling tile tracking system. Doing so raises two questions: How much are the differences in final parameter estimates? How much do these differences affect the localization accuracy of the FastSLAM 2.0 algorithm?

To answer the first question, we compared the parameters learned from the movements according to the odometry readings and FastSLAM 2.0 with 10 particles, denoted as p_i, and the parameters learned from the movements according to the odometry readings and the ceiling tile tracking system, denoted as p_i'. These two sets of parameters were learned offline. We discovered that they were almost identical, and none of the differences between each pair of the mean parameters was greater than 5%. We believe this result was due to the fact that the parameters were learned from the amount of translational and rotational movements reported at *each time step*. Although the FastSLAM 2.0's pose error accumulates over time, its reported amount of movements at time t is independent of the amounts reported at time $t-1$. In our case, we believe FastSLAM 2.0 can report an accurate amount of movements from s_{t-1} to s_t.

To answer the second question, we set up an experiment to run FastSLAM 2.0 with each of the two motion models, denoted as Model A and Model B. Each motion model was repeated for 50 trials in the environment described in section 5.1. The localization accuracy of FastSLAM 2.0 with Model A was statistically indistinguishable from that of Model B. In fact, their RMS pose errors and their standard deviations were almost identical.

Therefore, we conclude that learning motion model parameters with ground-truth poses from FastSLAM 2.0 or from the ceiling tile tracking system does not significantly affect the final parameter estimates.

5.4.4 Benefits of well-calibrated mean and variance of the proposal distribution

This section discusses the benefits of having a well calibrated mean and variance of the proposal distribution in the context of FastSLAM 2.0. In other Rao-Blackwellized particle filter SLAM algorithms, increasing the number of particles may be used as an alternative to generating an accurate proposal distribution [22]. In other words, a system with a poor proposal distribution may require a very large number of particles to successfully track the state of the robot. In the worse case, the system may never be able to track the robot regardless of the number of particles. Therefore, we believe that one of the benefits of well-calibrated means and variances of the proposal distributions for Rao-Blackwellized particle filter SLAM algorithms is that fewer particles are required to represent the robot's state.

However, an increase in the number of particles did not significantly affect the FastSLAM 2.0's localization accuracy as shown in Figure 5.2 and Figure 5.3. We believe that it is because the improved proposal distribution is generated in the same manner as the posterior distribution in the EKF localization. As a result, the uncertainty in the improved proposal distribution is smaller than

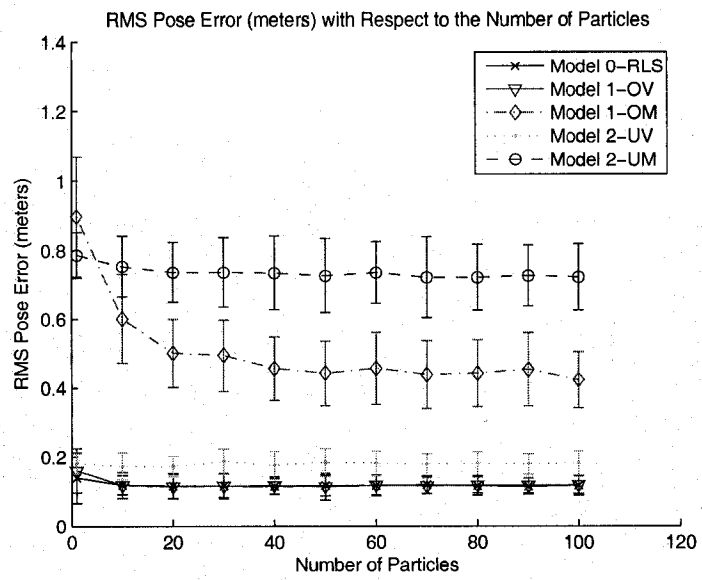

Figure 5.10: The effects of overestimating and underestimating the mean and the variance of the proposal distribution. The Model 1-OV and Model 1-OM refers to the motion models whose mean and variance were artificially increased by 50% from those of Model 0-RLS. The Model 2-UV and Model 2-UM refer to the motion models whose mean and variance were artificially decreased by 50% from those of Model 0-RLS.

that of the contributing distributions as discussed in section 5.4.1. If we assume that the sensor information is accurate with low uncertainty, the uncertainty in the improved proposal distribution is expected to be very small regardless of the quality of the proposal distribution. As a result, this allows us to sample possible robot poses in small areas that have high likelihoods. The number of samples, therefore, does not play an important role in this case because of small sampling areas. However, the accuracy of the proposal distribution still plays an important role in generating an accurate improved proposal distribution, and the more accurate the improved proposal is, the more accurate the FastSLAM 2.0 algorithm.

Figure 5.10 summarizes the effects of over/underestimating the mean and the variance of the proposal distribution in the FastSLAM 2.0's framework and suggests that it is much more important to have a well-calibrated mean than to have a well-calibrated variance that could possibly be compensated for by accurate sensor measurements.

Figure 5.10 also suggests another interesting result: the use of pessimistic motion model, which refers to a model that generates proposal distributions whose variances overestimate their true uncertainties, did not significantly affect the FastSLAM 2.0's localization accuracy. This result came from the motion model that overestimated the variance of our motion model by 50%. What would happen then if we increase artificial noise even more than 50%? As indicated by the results in Table 5.4, the localization accuracy of FastSLAM 2.0 with the Model 0-RLS was statistically indistinguish-

Inflated artificial noise	RMS Pose Error in meter (std)
0% (Model 0-RLS)	0.118(0.029)
50% (Model 1-UV)	0.120(0.032)
100%	0.122(0.036)
150%	0.122(0.043)
200%	0.132(0.042)
250%	0.135(0.047)

Table 5.4: The effects of overestimating the variance of the proposal distribution at different levels in the FastSLAM 2.0's localization accuracy. The table shows the RMS pose error (m) from FastSLAM 2.0 with 10 particles and the standard deviation of 50 independent trials in each case.

able from the accuracy of FastSLAM 2.0 with the use other motion models that generate proposal distributions whose variances were increased by 50% to 250% (with an increment of 50%).

In conclusion, an increase in the number of particles does not significantly affect the FastSLAM 2.0's localization accuracy, and having a well-calibrated mean of the proposal distribution reduces the significant of errors. Moreover, the use of pessimistic motion model does not significantly affect the FastSLAM 2.0's localization accuracy.

5.4.5 Motion model that accounts for sideways shifts

We have concluded in the previous section that having a well-calibrated mean of the proposal distribution reduces the significancy of errors. It is also known that the motion model used throughout this thesis is just an approximation of the robot's true motion. These findings motivate the need of having a motion model that can better approximate the robot's true movement (i.e. a model that generates a proposal distribution with a more accurate mean). Eliazar and Parr [22] proposed the following motion model:

$$x' = x + Dcos(\theta + \frac{T}{2}) + Ccos(\theta + \frac{T+\theta}{2}) \quad (5.3)$$

$$y' = y + Dsin(\theta + \frac{T}{2}) + Csin(\theta + \frac{T+\theta}{2}) \quad (5.4)$$

$$\theta' = \theta + T; \theta' \in (-\pi, \pi) \quad (5.5)$$

This motion model takes into account the ability of the robot to move in a direction that is not

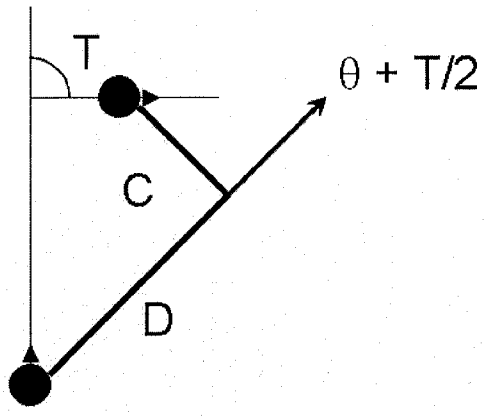

Figure 5.11: Eliazar and Parr's motion model. The movement in the D direction is referred to as the major axis, and the movement in the C direction is referred to as the minor axis, which represents sideways shifts in the orthogonal direction to the major axis. Image courtesy of Austin Eliazar.

solely determined by the beginning and the end facing angle of the robot so it is able to handle sideways shifts. Moreover, the model separates the robot's movement into two principle components. The movement in the D direction is referred to as the major axis, and the movement in the C direction is referred to as the minor axis that represents sideways shifts in the orthogonal direction to the major axis as shown in Figure 5.11.

The true values of D, C, and T are distributed normally with respect to the reported translational and rotational movements (d and t) as follows:

$$D \sim N(p_1 \cdot \bar{d} + p_2 \cdot \bar{t}, p_3 \cdot \bar{d}^2 + p_4 \cdot \bar{t}^2) \tag{5.6}$$

$$C \sim N(p_5 \cdot \bar{d} + p_6 \cdot \bar{t}, p_7 \cdot \bar{d}^2 + p_8 \cdot \bar{t}^2) \tag{5.7}$$

$$T \sim N(p_9 \cdot \bar{d} + p_{10} \cdot \bar{t}, p_{11} \cdot \bar{d}^2 + p_{12} \cdot \bar{t}^2) \tag{5.8}$$

In order to validate the effectiveness of Eliazar and Parr's motion model, we first learned 12 parameters in (5.6) to (5.8) with the RLS algorithm (similar to the way our Model 0-RLS was built in 5.3.1). We then compared the FastSLAM 2.0's localization accuracy with Eliazar and Parr's motion model to that with the use of Model 0-RLS in the environment described in section 5.2. Each model was used to run FastSLAM 2.0 for 50 trials to compute the RMS pose error along the entire trajectory. As the result in Table 5.5 indicates, the level of improvement using Eliazar and Parr's motion model was statistically indistinguishable from using of Model 0-RLS motion model.

Therefore, we conclude that although Eliazar and Parr's motion model could handle sideways shifts, the level of improvement from using this model with FastSLAM 2.0 on our data set was insignificant. This result could come from the fact that our robot was simulated to be able to observe landmarks at approximately every 15 centimeters of movement to correct its pose estimates from the motion model. With this small amount of movement, the amount of sideways shifts was small as

Motion model	RMS Pose Error in meter (std)
Model O-RLS	0.118(0.029)
Eliazar and Parr's model	0.115(0.027)

Table 5.5: Summary of FastSLAM 2.0's RMS pose error(m) with the use of Model 0-RLS and the Eliazar and Parr's motion models.

well. As a result, having the component to handle the shifts did not result in better estimation of the mean of the proposal distribution. On the other hand, the difference between the two motion models could be significant if we forced the robot to move a large distance (i.e. 1 meter) before observing landmarks or in scenarios where only poor measurement readings were available.

5.5 Ability of our framework to bootstrap the motion model

As discussed in section 4.5, one of the motivations of using our framework for building a motion model is to be able to start with a crude initial motion model and allow the model to bootstrap itself towards a more refined model in a similar environment. In this section, we demonstrate such ability of our framework (using RLS and FastSLAM 2.0). As already mentioned, we first need an initial motion model in order to learn motion model parameters using the robot's estimated true poses from FastSLAM 2.0. Based on the finding that the use of pessimistic motion model does not significantly affect the FastSLAM 2.0's localization accuracy, we set the model parameters of our initial model, denoted as Model I, as follows: $[p_1, p_3] = 1.0$, $[p_6, p_8] = 2.0$, and $[p_2, p_4, p_5, p_7] = 0$. Refer back to section 2.3 for our parameterized motion model. We believe that our initial motion model is realistic given that we do not have any prior knowledge on a robot we have.

We first ran FastSLAM 2.0 with the use of our initial motion model (Model I) in the environment described in section 5.2 and recorded its localization error. We then applied our framework to build a static motion model using RLS with the robot's estimated true poses from FastSLAM 2.0 with the use of the Model I. The resulting motion model is denoted as Model II. Next we ran FastSLAM 2.0 with the use of the Model II in the same environment described in section 5.2 and recorded its localization error. As our results in Table 5.6 indicate, the FastSLAM 2.0's RMS pose error along the entire trajectory from using the Model II was approximately 40% lower than that from using the Model I. An increase in the RMS pose error of the FastSLAM 2.0 with the use of Model I is probably due to the bias in the odometry (i.e. the error is not zero-mean), which could be identified by our framework when building the Model II. Therefore, we can conclude that when presented with a less accurate initial motion model, our proposed framework could bootstrap the initial model towards a more refined motion model.

Motion model	RMS Pose Error in meter (std)
Model I (Initial model)	0.187(0.054)
Model II (Bootstrapped model)	0.126(0.020)

Table 5.6: Summary of FastSLAM 2.0's RMS pose error(m) with the use of an initial motion model (Model I) and with the use of a more refined motion model (Model II) built from our framework. This experiment was run for 50 independent trials and each with 10 particles.

5.6 Summary

In this chapter, we tested our motion models in a set of experiments using control data collected from the Magellan Pro robot. The experimental results indicated that the use of static motion model built with RLS significantly improved the FastSLAM 2.0's localization accuracy compared to the use of other motion models, which generated proposal distributions whose means and variances were artificially increased or decreased. The results also demonstrated the effectiveness of our dynamic motion model built with BiRLS over the static motion model in the FastSLAM 2.0 framework. The FastSLAM 2.0's RMS pose error along the entire trajectory from using our dynamic model was approximately 20% lower than that from when we use the static model.

Moreover, we discussed the role of the motion model in the FastSLAM 2.0 algorithm. The experimental results first surprised us as we did not expect the motion model to play such an important role in generating the FastSLAM 2.0's improved proposal distribution. We expected that accurate measurements could correct poor pose estimates from the motion model. However, the improved proposal distribution is not just the estimated likelihood distribution but is generated from both the proposal distribution and the likelihood distribution in a similar manner as the posterior distribution of EKF localization. With this understanding, we discussed our experimental results in detail.

We then set up experiments to demonstrate that we could use the poses generated from FastSLAM 2.0 (instead of poses according to the ceiling tile tracking system) as the robot's estimated true poses to learn motion model parameters. As a result, we could eliminate the need for external measurements and explicit requirements of ground-truth poses. We also provided a brief summary on the benefits of having accurate mean and variance of the proposal distribution for the FastSLAM 2.0 algorithm. It turned out that having a well-calibrated mean of the proposal distribution reduced the significant of errors, and the use of pessimistic motion models did not significantly affect the FastSLAM 2.0's localization accuracy. We then discussed Eliazar and Parr's motion model, which was proposed to handle sideways shifts, and demonstrated its accuracy. Lastly, we demonstrated that our framework can start with a crude motion model and bootstrap itself towards a more refined model.

Chapter 6

Conclusion

6.1 Main Result

In this thesis, we described the use of two sequential machine learning algorithms for building a mobile robot's probabilistic motion model, which is an essential component in SLAM algorithms. For building a static motion model, we applied the recursive least squares (RLS) algorithm to learn motion model parameters as soon as new data arrived without the need for external measurements, explicit requirements of ground-truth poses, and human intervention. We could just let the robot traverse the environment running SLAM and the robot will determine by itself whether it has built the best static motion model for the particular environment. For building a dynamic motion model, we used the bi-loop recursive least squares (BiRLS) algorithm to learn the parameters on the fly as the robot operates. The main advantage of BiRLS is that it could keep track of rapid, slow, or periodic changes in model parameters. Although our methods were designed for building motion models for the FastSLAM algorithm (i.e. we used the robot's pose based on the mean of the particles to calculate the true amount of movements), they could easily be applied for the use in other SLAM or localization algorithms. They only require the true amount of movements and the predicted ones from the odometry. Our techniques were then tested in a simulation using control data collected from a real robot, and the experimental results demonstrated the effectiveness of our frameworks both in the static environment and in the environment with dynamically changing ground surface. Moreover, our experimental results also indicated that the motion model still plays an important role in the accuracy of the FastSLAM 2.0 algorithm.

To the best of our knowledge, we believe this is the first time the sequential machine learning algorithms have been applied to learn the model parameters. All previous works in building static motion models were done with batch learning while previous techniques of building dynamic motion models required predetermined thresholds to indicate when their parameters should be updated. This thesis also studied in detail the role of the motion model in the FastSLAM 2.0 algorithm whose improved proposal distribution is not only generated from the motion model but also takes the most recent measurement into consideration.

6.1.1 Justification: Our motion model

Instead of the motion model used in this thesis, many would suggest the use of the model with the *turn-travel-turn* approach proposed by [45] and used in [53], [10] as follows:

$$x' = x + \delta_{trans} cos(\theta + \delta_{rot1}) \tag{6.1}$$

$$y' = y + \delta_{trans} sin(\theta + \delta_{rot1}) \tag{6.2}$$

$$\theta' = \theta + \delta_{rot1} + \delta_{rot2}; \tag{6.3}$$

Given $s_{t-1} = (\bar{x}, \bar{y}, \bar{\theta})$ and $s_t = (\bar{x}', \bar{y}', \bar{\theta}')$, $\delta_{trans}, \delta_{rot1}$, and δ_{rot2} are calculated as:

$$\delta_{rot1} = \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta} \tag{6.4}$$

$$\delta_{trans} = \sqrt{(\bar{x}' - \bar{x})^2 + (\bar{y}' - \bar{y})^2} \tag{6.5}$$

$$\delta_{rot2} = \bar{\theta}' - \bar{\theta} - \delta_{rot1} \tag{6.6}$$

This motion model assumes that the true translational (δ_{trans}) and two rotational movements (δ_{rot1} and δ_{rot2}) are distributed normally with respect to the reported values. This means this model assumes that we get δ'_{trans}, δ'_{rot1}, and δ'_{rot2} from odometry readings.

Mathematically, this model better approximates the robot's true motion than the model used in this thesis. However, it introduces another rotational component δ_{rot2} in (6.6), which also assumes Gaussian noise. Essentially, we get $T = \delta_{rot1} + \delta_{rot2}$. In our system, the high uncertainty in the robot's rotational movement is due to the robot's physical property. Having to draw a sample from a normal distribution with high variance could result in the sample being far away from the true value (unlucky draw). From a practical point of view with our data set, we found, in most cases, that drawing samples from two Gaussians representing the robot's true rotation resulted in the amount of rotation being further away from the true amount than drawing from one Gaussian (our motion model). This justifies the choice of the motion model used in this thesis.

6.1.2 Justification: Learning a motion model from a single robot's pose

Instead of learning motion model parameters from the mean of the particles in this thesis, we could learn model parameters for the FastSLAM algorithm from multiple particles at each time step similar to [22], [10] (Refer to chapter 2 in the related work section for more detail). Doing so would allow learning algorithms to determine how much each particle influences the final parameter estimates based on its importance weight. We initially tried this approach and found that it did not result in a more accurate motion model (i.e. the model did not result in a more accurate FastSLAM 2.0's localization accuracy). This result was not surprising because of small sampling areas within the FastSLAM 2.0's improved proposal distribution that resulted in most of the particles focusing in

around the same areas. Most of them also had similar importance weight's scores. Therefore, learning model parameters from multiple particles at each time step may *not* result in a more accurate motion model compared with the model that was learned from a single robot's pose (i.e. the mean of the particles) for the FastSLAM 2.0 framework given an accurate sensor.

6.2 Significant of the work

The research carried out in this thesis is important because it is well-known that the FastSLAM algorithm is sensitive to the accuracy of the proposal distribution, which is generated from the motion model. Specifically, the more accurate the proposal distribution is, the slower the loss of particle diversity in the particle sampling process, and the more accurate the FastSLAM algorithm in the long run. In other words, the rapid loss of particle diversity prevents a consistent long-term estimate of the FastSLAM algorithm [9]. Thus, the motivation for acquiring a good motion model is quite strong.

It is worth mentioning that building an accurate motion model is not the only research direction to improve the performance of the FastSLAM algorithm by preventing the rapid loss of particle diversity. The other interesting direction is to develop new sampling strategies. The basic FastSLAM algorithm resamples particles according to their importance weights at every update step. Liu [38] has shown that if the importance weights of particles are relatively uniform, resampling only decreases effectiveness of the particle filter to keep track of the system. Therefore, many researchers have focused their research on how to and when to effectively resample [27], [11].

Generally, an accurate motion model allows us to draw particles in areas with high likelihoods. Doing so results in particle weights remaining relatively uniform, which should *indirectly* slow down the loss of particle diversity. On the other hand, developing a smarter way to resample *directly* slows down the loss of particle diversity.

6.3 Future research direction

We have thoughtfully investigated in the area of sequentially learning motion model parameters. One issue that we would like to point out is that it may not be beneficial to focus a research on developing a *better* and a more sophisticated learning algorithm to build a motion model, especially if we want to use the motion model for the FastSLAM 2.0 algorithm. In this case, even if the learning algorithm could improve the accuracy of the final parameter estimates by, for example less than 5%, it may not significantly affect the accuracy of the FastSLAM 2.0 algorithm [1]. We may want to allocate the available computational resource to the other FastSLAM 2.0's components that could play a more important role on its accuracy (i.e. sensor models). Moreover, Eliazar and Parr

[1] We are not saying that the model model is not an important component for FastSLAM 2.0. In fact, having an inaccurate motion model could significantly increase the FastSLAM 2.0's localization error as indicated by our experimental results. We just want to point out that a *small* improvement on the learning algorithm may not significantly affect the accuracy of the FastSLAM 2.0 algorithm.

suggested it would be interesting if we could relax the assumption that the true motion of the robot can be described by two independent normal distributions and allow the motion to be described by a single multivariate distribution. We agree that it would be interesting but doubt whether it would result in a more accurate proposal distribution.

The future research directions we are interested are as follows. First, we are interested in building a more effective dynamic motion model that can automatically determine when the model needs to be updated. Updating motion model parameters with BiRLS at every time step could be very sensitive to system noise although such situations rarely occur in our data sets. Second, we would like to study the relationship between the sensor and the motion models. According to our findings in this thesis, we strongly believe that there is a connection between the two in the FastSLAM 2.0 algorithm. For example, if the sensor model is noisy, we may want to artificially inflate the noise into the motion model. Lastly, we are interested in the framework, proposed by Grisetti, Stachniss and Burgard [28], which approximates the proposal distribution purely from the scan matching technique. Their framework did not require an odometry motion model because they believe that the purely laser-based proposal distribution is well-suited to predict the robot's motion in most cases. However, there are some situations in which knowledge from the odometry motion model is important to focus the proposal distribution. For example, an open free space without any landmarks within the laser range. In such a situation, the information from the odometry motion model provides the best estimate of the robot's pose. We believe it could be practical if we could come up with a framework that can automatically determine the best way to generate the proposal distribution.

Bibliography

[1] The official website of I,Robot, http://www.irobotmovie.com, 2004.

[2] The 2006 SLAM summer school, http://www.robots.ox.ac.uk/~sss06, 2006.

[3] http://www.cs.ualberta.ca/~jklipp, 2007.

[4] OpenCV: The open computer vision library, http://sourceforge.net/projects/opencvlibrary, 2007.

[5] The point gret research, http://www.ptgrey.com, 2007.

[6] Henderson A.R. Testing experimental data for univariate normality. *Clinica Chimica Acta*, (366), 2006.

[7] K.J. Astrom and B. Wittenmark, editors. *Adaptive Control*. Addison Wesley, second edition, 1994.

[8] T. Bailey. Mobile robot localisation and mapping in extensive outdoor environments. *PhD Thesis*, 2002.

[9] T. Bailey, J. Nieta, and E.M. Nebot. Consistency of the FastSLAM algorithm. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, 2006.

[10] P. Beeson, A. Murarka, and B. Kuipers. Adapting proposal distributions for accurate, efficient mobile robot localization. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, 2006.

[11] R.K. Beevers and H.H. Huang. Fixed-lag sampling strategies for particle filtering slam. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, 2007.

[12] J. Borenstein and L. Feng. Correction of systematic odometry errors in mobile robots. 1995.

[13] J. Borenstein and L. Feng. Measurement and correction of systematic odometrys in mobile robots. In *Proceedings of the IEEE Transaction on Robotics and Automation*, 1996.

[14] W.L. Brogan, editor. *Modern Control Theory*. Prentice Hall, 1991.

[15] K.S. Chong and L. Kleeman. Accurate odometry and error modelling for a mobile robot. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, 1997.

[16] G. Dissanayake, P. Newman, H.F. Durrant-Whyte, S. Clark, and M. Csorba. An experimental and theoretical investigation into simultaneous localisation and map building. In *Lecture Notes in Control and Information Sciences, Experimental Robotics VI.*

[17] A. Doucet. On sequential simulation-based methods for bayesian filtering. Technical report, Cambridge University Department of Engineering, 1998.

[18] A. Doucet, J.F.G. de Freitas, and N.J. Gordon, editors. *Sequential Monte Carlo Methods In Practice*. Springer, 2001.

[19] H.F. Durrant-Whyte. Consistent integration and propagation of disparate sensor information. *International Journal in Robotics Automation*, 1987.

[20] H.F. Durrant-Whyte and T. Bailey. Simultaneous localisation and mapping (SLAM): Part 1: The essential algorithms. *Robotics and Automation Magazine*, 2006.

[21] A. Eliazar and R. Parr. DP-SLAM 2.0. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, 2004.

[22] A. Eliazar and R. Parr. Learning probabilistic motion models for mobile robots. In *Proceedings of the Twenty First International Conference on Machine Learning (ICML)*, 2004.

[23] L. Feng, J. Borenstein, and B. Everett. Where am I?: Systems and methods for mobile robot positioning. Technical report, The University of Michigan, 1994.

[24] T.R. Fortescue, L.S. Kershenbaum, and B.E. Ydstie. Implementation of self-tuning regulators with variable forgetting factors. *Automatica*, (17), 1981.

[25] B. Gerkey, R.T. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th International Conference on Advanced Robotics (ICAR)*, 2003.

[26] P. Goel, I. Roumeliotis, and G.S. Sukhatme. Robust localization using relative and absolute position estimates. 1999.

[27] G. Grisetti, C. Stachniss, and W. Burgard. Improving grid-based SLAM with rao-blackwellized particle filters by adaptive proposals and selective resampling. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, 2005.

[28] G. Grisetti, C. Stachniss, and W Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 2006.

[29] J.E. Guivant and E.M. Nebot. Optimization of the simultaneous localization and map-building for real-time implementation. In *Proceedings of the IEEE Trans. Robotics and Automation*, 2001.

[30] D. Haehnal, D. Fox, W. Burgard, and S. Thrun. A highly efficient FastSLAM algorithm for generating cyclic maps of large-scale environments from raw laser range measurements. 2003.

[31] S. Haykin, editor. *Adaptive Filtering Theory*. Prentice Hall, 2002.

[32] A. Kaboli, M. Bowling, and P. Musilek. Bayesian calibration for Monte Carlo Localization. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI)*, 2006.

[33] A. Kelly. Fast and easy systematic and stochastic odometry calibration. 2004.

[34] S. Koike. Adaptive forgetting factor recursive least squares adaptive threshold nonlinear algorithm (RFF-RLS-ATNA) for identification of nonstationary systems. In *Proceedings of the IEEE Transactions on Acoustics, Speech, and Signal Processing (ICASSP)*, 2003.

[35] Ljung. L., editor. *System Identification: Theory for the User*. Prentice Hall Information and System Sciences Series, 1987.

[36] J.J. Leonard and H.J.S. Feder. A computational efficient method for large-scale concurrent mapping and localisation. *Robotics Research, The Ninth International Symposium (ISRR'99)*, 2002.

[37] S.H. Leung and C.F. So. Gradient-based variable forgetting factor RLS algorithm in time-varying environments. In *Proceedings of the IEEE Transactions on Signal Processing*, 2005.

[38] J.S. Liu. Metropolized independent sampling with comparisons to rejection sampling and importance sampling. In *Statistics and Computing*, 1996.

[39] A. Martinelli. The odometry error of a mobile robot with a synchronous drive system. In *Proceedings of the IEEE Transaction on Robotics and Automation*, 2002.

[40] A. Martinelli and R. Siegwart. Estimating the odometry error of a mobile robot during navigation. In *Proceedings of European Conference on Mobile Robots (ECMR)*, 2003.

[41] A. Milstein and T. Wang. Localization with dynamic motion models. In *Proceedings of the International Conference on Informatics in Control (ICINCO)*, 2006.

[42] M. Montemerlo. FastSLAM: A factored solution to the simultaneous localization and mapping problem with unknown data association. *PhD Thesis*, 2003.

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

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

[45] I. Rekleitis. Cooperative localization and multi-robot exploration. *Ph.D. dissertation, School of Computer Science, McGill University, Montreal, Quebec, Canada*, 2003.

[46] N. Roy and S. Thrun. Online self-calibration for mobile robots. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, 1999.

[47] I. Sanchez. Recursive estimation of dynamic models using cook's distance, with application to wind energy forecast. *Statistics and Econometrics Series*, (15), 2002.

[48] S.L. Shah and R.W. Cluett. Recursive least squares based estimation schemes for self-tuning control. *The Canada Journal of Chemical Engineering*, (69), 1991.

[49] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In *Proceedings of the 2nd Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, 1986.

[50] S. Song, J.S. Lim, S.J. Baek, and S. K.M. Gauss newton variable forgetting factor recursive least squares for time varying parameter tracking. *Electronics Letters*, (36), 2000.

[51] S. Thrun, Y. Liu, D. Koller, A.Y. Ng, A. Ghahramani, and H.F. Durrant-Whyte. Simultaneous localization and mapping with sparse extended information filters. *International Journal of Robotics Research*, 23, 2004.

[52] S. Thrun and M. Montemerlo. The GraphSLAM algorithm with applications to large-scale mapping of urban structures. *International Journal on Robotics Research*, 25, 2005.

[53] S. Thrun, B. Wolfram, and D. Fox, editors. *Probabilistic Robotics*. MIT Press, 2005.

[54] A. Vahida, A. Stefanopoulou, and H. Peng. Recursive least squares with forgetting for online estimation of vehicle mass and road grade: Theory and experiments. *Vehicle System Dynamics*, (43), 2005.

[55] G. Welch, , and G. Bishop. An introduction to the Kalman filter, http://www.cs.unc.edu/~welch, 2006.

[56] W.C. Yu and N.Y. Shih. Bi-loop recursive least squares algorithm with forgetting factors. *the IEEE Signal Processing Letters*, (13), 2006.

[57] J. Zeng and R.A. de Callafon. Recursive least squares generalized FIR filter estimation for active noise cancellation. *Workshop on Adaptation and Learning in Control and Signal Processing*, 2004.

[58] H.. Zhang. Two-dimensional optimal sensor placement. In *Proceedings of the EEE Transactions on Systems, Man and Cybernetics*, 1995.

[59] W. Zhuang. RLS algorithm with variable forgetting factor for decision feedback equalizer over time-variant fading channels. *Wireless Personal Communications*, (9), 1998.

Appendix A

Gaussian Filters

In this appendix, we first discuss the Kalman filter in the context of SLAM and localization problems with the main focus to discuss one of its property: the uncertainty of the posterior distribution, also known as the desired belief, is conceptually smaller [1] than those in the proposal distribution and the likelihood distribution. We then discuss the Extended Kalman Filter, which is used to approximate the FastSLAM 2.0's improved proposal distribution. We will not derive these filters. For their full derivations, refer to the chapter 3.2 of [53].

A.1 The Kalman Filter

The Kalman filter represents the belief at time t (i.e. the state of the robot s_t) by a Gaussian distribution with the mean μ_t and the covariance Σ_t. The control and the measurement vectors at time t are denoted by u_t and z_t respectively. Given the prediction probability $p(s_t|u_t, s_{t-1})$ that must be a linear function with $s_t = A_t s_{t-1} + B_t u_t + \epsilon_t$ [2], and the measurement probability $p(z_t|s_t)$ that must also be a linear function with $z_t = C_t s_t + \delta_t$ [3], the Kalman filter is stated in Figure A.1.

The prediction steps are in lines 2 and 3 where the predicted belief, represented by $\bar{\mu}_t$ and $\bar{\Sigma}_t$, is calculate from the belief one time step earlier and the most recent control u_t. As the prediction step cannot account for the added Gaussian noise, the uncertainty in the predicted belief will be higher than that from the belief one time step earlier. The measurement update steps are in lines 4 to 6 where the predicted belief is transformed into the desired belief by incorporating the measurement z_t. Because the filter calculates the desired belief using the new *observed* measurement, its covariance is also changed to reflect new information, which results in a reduced uncertainty (compared to the uncertainty in the prediction step).

Line 5 indicates two key components of the Kalman Filter: the innovation vector and the Kalman

[1] It is conceptual because we compare matrices.

[2] A_t and B_t are matrices. A_t is a square matrix of size n x n, where n is the dimension of the state vector s_t. B_t is a n x m matrix, where m is the dimension of the control vector u_t. The vector ϵ_t describes the prediction Gaussian noise whose size is of the same dimension as the state vector. Its mean is zero, and its covariance is denoted as R_t.

[3] C_t is a k x n matrix, where k is the dimension of the measurement vector z_t, and the vector δ_t describes the measurement Gaussian noise with zero mean and covariance Q_t.

gain. The innovation vector $z_t - C_t\bar{\mu}_t$ refers to the difference between the actual measurement z_t and the predicted measurement $C_t\bar{\mu}_t$ calculated with the consideration of the predicted mean and covariance in lines 2 and 3. The matrix K calculated in line 4 is called the Kalman gain, also known as the blending factor, which minimizes the desired posterior covariance in line 6 and is defined as follows:

$$\begin{aligned} K_t &= \bar{\Sigma}_t C_t{}^T (C_t \bar{\Sigma}_t C_t{}^T + Q_t)^{-1} \\ &= \frac{\bar{\Sigma}_t C_t{}^T}{C_t \bar{\Sigma}_t C_t{}^T + Q_t} \end{aligned} \tag{A.1}$$

The relationship between the Kalman gain and the innovation vector is that the innovation vector gives the offset between the predicted and actual measurement, and the Kalman gain scales the innovation vector according to the uncertainty in the measurement. The more certain the observation, the higher the Kalman gain, and the more of the correction made to the predicted belief. As we can easily see from (A.1), if the measurement covariance Q_t approach zero, the larger the Kalman gain. Specifically,

$$\lim_{R_t \to 0} K_t = C_t{}^{-1} \tag{A.2}$$

On the other hand, if the prediction covariance $\bar{\Sigma}_t$ approaches zero, the smaller the Kalman gain as:

$$\lim_{\bar{\Sigma}_t \to 0} K_t = 0 \tag{A.3}$$

In other words, the smaller the measurement noise covariance, the more *trust* the filter has on the actual measurement z_t. On the other hand, the larger the measurement noise covariance, the less *trust* the filter has on the actual measurement while the predicted measurement $C_t\bar{\mu}_t$ is *trusted* more [55].

One important property of the Kalman filter is that the uncertainty in the desired belief Σ_t is conceptually smaller than the uncertainties from both contributing Gaussians (R_t and Q_t). Specifically, the covariance matrix of the desired belief Σ is given as [19]:

$$\Sigma = (Q^{-1} + R^{-1})^{-1} \tag{A.4}$$

Equation (A.4) assumes that each model has a zero-mean additive noise that follows the Gaussian distribution, and all models are independent of each other [58].

1: TheKalmanFilter(μ_{t-1},Σ_{t-1},u_t,z_t)

2: $\bar{\mu}_t = A_t\mu_{t-1} + B_t u_t$
3: $\bar{\Sigma}_t = A_t\Sigma_{t-1}{A_t}^T + R_t$

4: $K_t = \bar{\Sigma}_t {C_t}^T (C_t\bar{\Sigma}_t {C_t}^T + Q_t)^{-1}$
5: $\mu_t = \bar{\mu}_t + K_t(z_t - C_t\bar{\mu}_t)$
6: $\Sigma_t = (I - K_t C_t)\bar{\Sigma}_t$
7: return μ_t, Σ_t

Figure A.1: The Kalman Filter algorithm

A.2 The Extended Kalman Filter

In brief, the Extended Kalman Filter (EKF) relaxes the KF's assumption that the prediction and the observation probabilities must be linear functions as they are rarely linear in practice. In EKF, the prediction and the observation probabilities are governed by nonlinear functions g and h respectively:

$$s_t = g(u_t, s_{t-1}) + \epsilon_t \tag{A.5}$$

$$z_t = h(s_t) + \delta_t \tag{A.6}$$

As we can easily notice from (A.5), the non-linear function g relates the state at the previous time step s_{t-1} to the state at the current time step s_t by including the most recent control vector u_t. Similarly, the non-linear function h in (A.6) relates the state s_t to the most recent measurement vector z_t.

Similar to KF, EKF represents the belief at time t (i.e. the robot's pose s_t) by the mean μ_t and the covariance Σ_t. However, because the EKF's belief is only an approximation, its objective is not for computing the exact belief as is the case for KF. It is rather to estimate one. In other words, the belief of the random variables would be no longer Gaussian after undergoing their respective nonlinear transformations (i.e. the robot's belief after execute control actions). EKF simply approximates the belief by linearlization.

Figure A.2 states the EKF algorithm, which is similar to the KF algorithm in many ways. However, the linear prediction and observation functions (line 2 and line 5) are replaced by their nonlinear generalizations. The question now arises: how does EKF transform linear functions to non-linear functions? Generally, linearization approximates the non-linear function f by a linear function that is tangent to f at the mean of the input Gaussian. Specifically, EKF utilizes the method called the Taylor expansion. Notice from Figure A.2, EKF uses Jacobian matrices G_t and H_t instead of the corresponding linear system matrices A_t, B_t, and C_t in KF. These Jacobians serve to correctly propagate only the relevant component of the information to the state. For example, $H_t\bar{\Sigma}_t H_t$ in line 3

1: TheExtendedKalmanFilter(μ_{t-1},Σ_{t-1},u_t,z_t)

2: $\bar{\mu}_t = g(u_t, \mu_{t-1})$
3: $\bar{\Sigma}_t = G_t \Sigma_{t-1} {G_t}^T + R_t$

4: $K_t = \bar{\Sigma}_t {H_t}^T (H_t \bar{\Sigma}_t {H_t}^T + Q_t)^{-1}$
5: $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
6: $\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$
7: return μ_t, Σ_t

Figure A.2: The Extended Kalman Filter algorithm

could represent the measurement uncertainty due to uncertainty in the robot's pose in the context of mobile robot localization.

A.2.1 The FastSLAM 2.0's improved proposal distribution

As already stated in the chapter 3 in this thesis, FastSLAM 2.0 approximates its improved proposal distribution for the m-th particle with a Gaussian approximation in the same manner as the EKF as follows:

$$s_t^{[m]} = p(s_t | s^{t-1,[m]}, u^t, z^t, n^t) \sim N(\mu_{s_t}^{[m]}, \Sigma_{s_t}^{[m]}) \tag{A.7}$$

where $\mu_{s_t}^{[m]}$ and $\Sigma_{s_t}^{[m]}$ are the mean and the covariance of the improved proposal distribution respectively. They are defined as follows:

$$\Sigma_{s_t}^{[m]} = [H_s^T (Q_t^{[m]})^{-1} H_s + R_t^{-1}]^{-1} \tag{A.8}$$

$$\mu_{s_t}^{[m]} = \Sigma_{s_t}^{[m]} H_s^T (Q_t^{[m]})^{-1} (z_t - {z'}_t^{[m]}) + {s'}_t^{[m]} \tag{A.9}$$

$$Q_t^{[m]} = Q_t + H_\Theta \Sigma_{k,t-1}^{[m]} H_\Theta^T \tag{A.10}$$

The matrix R_t is the covariance matrix of the state transition probability $p(s_t | s_{t-1}^{[m]}, u_t)$. This covariance matrix represents the overall uncertainty in the robot's predicted pose from the motion model. The matrix $Q_t^{[m]}$ is the covariance matrix of the measurement probability $p(z_t | \Theta_k, s_t, n_t)$. This covariance matrix represents the overall uncertainty of the measurement. ${s'}_t^{[m]}$ is the robot's predicted pose sampled from $p(s_t | s_{t-1}^{[m]}, u_t)$, and ${z'}_t^{[m]}$ is the robot's predicted measurement to the landmark Θ_k, using the robot's predicted pose and its uncertainty. Notice at (A.9), $\Sigma_{s_t}^{[m]} H_s^T (Q_t^{[m]})^{-1}$ is essentially the Kalman gain in EKF (line 4 in Figure A.2), and $z_t - {z'}_t^{[m]}$ is the innovation in EKF (line 5 in Figure A.2). H_s and H_Θ are the Jacobians of the linearized measurement prediction $p(z_t | \Theta_k, s_t, n_t)$ with respect to s_t and Θ_k respectively. Note that the improved proposal distribution is incrementally refined for each observation on a previously-seen landmark. Refer to [44] and page 452 of the [53] for a full derivation of the FastSLAM 2.0's improved proposal distribution.

Here we rewrite the Kalman gain of the improved proposal distribution in a more familiar format as follows:

$$\begin{aligned} K_t^{[m]} &= \Sigma_{s_t}^{[m]} H_s^T (Q_t^{[m]})^{-1} \\ &= \frac{\Sigma_{s_t}^{[m]} H_s^T}{Q_t + H_\Theta \Sigma_{k,t-1}^{[m]} H_\Theta^T} \end{aligned} \tag{A.11}$$

The denominator part of $K_t^{[m]}$ refers to the sum of the two uncertainties representing the overall measurement prediction uncertainty: Q_t and $H_\Theta \Sigma_{k,t-1}^{[m]} H_\Theta^T$. Q_t refers to the uncertainty due to the measurement noise, and $H_\Theta \Sigma_{n_t,t-1}^{[m]} H_\Theta^T$ ignores the measurement noise and projects the previous uncertainty in observing the landmark k through a linear approximation of the measurement function. Notice that $\Sigma_{k,t-1}^{[m]}$ is mapped into observation uncertainty by multiplication with H_Θ, which is the Jacobian of the measurement function with respect to the observed landmark Θ_k location. In short, $Q_t + H_\Theta \Sigma_{k,t-1}^{[m]} H_\Theta^T$ represents the overall measurement uncertainty of observing Θ_k.

The numerator part of $K_t^{[m]}$ is a little bit more complicated. From (A.8), $\Sigma_{s_t}^{[m]}$ can be interpreted as follows: the overall measurement prediction uncertainty (the denominator part of $K_t^{[m]}$ mentioned earlier) is mapped back into the robot's pose uncertainty by the multiplication with the Jacobian of the measurement function with respect to the robot's pose H_s. This uncertainty together with the adding of the overall motion uncertainty R_t results in the overall robot's pose uncertainty. Note that the matrix R_t takes into consideration both the previous uncertainty in the robot's pose s_{t-1} and the uncertainty in the robot motion.

Moreover, it is easily to prove from (A.8) that the uncertainty in the improved proposal distribution is conceptually smaller than the uncertainties in both sensor and motion measurements (Q_t and R_t). By assuming that H_s is an identity matrix we get $\Sigma_{s_t}^{[m]} = (Q_t^{[m]-1} + R_t^{-1})^{-1}$. According to (A.10), we know that $Q_t^{[m]} = Q_t + \Sigma_{k,t-1}^{[m]}$ in which Q_t is the uncertainty in the likelihood distribution. Moreover, R_t is the overall uncertainty of the robot's predicted pose without incorporating sensor information. In other words, R_t is the sum of two uncertainty components: the motion noise and the uncertainty due to the initial pose uncertainty. As a result, we can conclude that the uncertainty in the improved proposal is conceptually smaller than the uncertainties in both sensor and motion measurements.