2010

Two improved methods for mobile robot localization

Yuefeng Wang
University of Windsor

TWO IMPROVED METHODS FOR MOBILE ROBOT LOCALIZATION

by
Yuefeng Wang

A Thesis
Submitted to the Faculty of Graduate Studies
through School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada
2010

Your file Votre référence
ISBN: 978-0-494-62723-5
Our file Notre référence
ISBN: 978-0-494-62723-5

Dedication of Co-Authorship / Previous Publication

I. Co-Authorship Declaration

I hereby declare that this thesis incorporates material that is result of joint research, as follows:

This thesis also incorporates the outcome of a joint research undertaken in collaboration with Jingxi Chen and Sepideh Seifzadeh under the supervision of Dr. Dan Wu. The collaboration is covered in Chapter 3 of the thesis. In all cases, the key ideas, primary contributions, experimental designs, data analysis and interpretation, were performed by the author, and the contribution of co-authors was primarily through the provision of constructive comments.

I am aware of the University of Windsor Senate Policy on Authorship and I certify that I have properly acknowledged the contribution of other researchers to my thesis, and have obtained written permission from each of the co-author(s) to include the above material(s) in my thesis.

I certify that, with the above qualification, this thesis, and the research to which it refers, is the product of my own work.

II. Declaration of Previous Publication

This thesis includes two original papers that have been previously published/submitted for publication in peer reviewed journals, as follows:

Thesis Chapter	Publication title/full citation	Publication status
Chapter 3	A moving grid cell based MCL algorithm for mobile robot localization, The 2009 IEEE International Conference on Robotics and Biomimetics (ROBIO 2009)	published
Chapter 3, 4	A dynamic MCL algorithm based on clustering for mobile robot localization, The 2010 International IEEE/RSJ Conference on Intelligent Robots and Systems (IROS 2010)	submitted

I certify that I have obtained a written permission from the copyright owner(s) to include the above published material(s) in my thesis. I certify that the above material describes work completed during my registration as graduate student at the University of Windsor.

I declare that, to the best of my knowledge, my thesis does not infringe upon anyones copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

Abstract

Mobile robot localization is the problem of determining the robot's pose given the map of its environment, based on the sensor reading and its movement. It is a fundamental and very important problem in the research of mobile robotics.

Grid localization and Monte Carlo localization (MCL) are two of the most widely used approaches for localization, especially the MCL. However each of these two popular methods has its own problems. How to reduce the computation cost and better the accuracy is our main concern.

In order to improve the performance of localization, we propose two improved localization algorithms. The first algorithm is called moving grid cell based MCL, which takes advantages of both grid localization and MCL and overcomes their respective shortcomings. The second algorithm is dynamic MCL based on clustering, which uses a cluster analysis component to reduce the computation cost.

Dedication

This thesis is dedicated to my parents for their endless support.

Also, it is dedicated to the people I care about and the people who care about me.

Acknowledgements

My thanks and appreciation to my supervisor Dr. Dan Wu who helps me a lot during my whole master's study. His encouragement, guidance and support enable me to complete my research and write this thesis.

I am grateful as well to my external reader, Dr. Jonathan Wu, my internal reader, Dr. Yung Tsin and my thesis committee chair, Dr. Subir Bandyopadhyay for spending their previous time reviewing this thesis and giving their valuable comments and suggestions.

Also, I want to thank my friends. They help me solve many difficult problems during my research.

Lastly, I would like to express my deep and sincere gratitude to my parents, and they always give me great confidence and support whenever I meet any difficulty.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

For a mobile robot, it is very important to know its position as most robot's tasks need the positional information. This is the most fundamental problem in mobile robotics and known as mobile robot localization problem[1][2][3]. Informally, mobile robot localization problem is the problem of determining the robot's pose given the map of the environment and the sensor readings.

There are three kinds of mobile robot localization problems which are characterized by the type of initial knowledge of its pose: position tracking, global localization, and kidnapped robot problem[10]. In position tracking the initial pose is known, and the localization is achieved by compensating incremental noise in the movement. In global localization, the initial pose is unknown, and it is much more difficult and challenging since the robot has to determine its pose from scratch. The kidnapped robot problem occurs when the robot is taken from its current position to somewhere else without being notified the replacement during the localization process. It is a variant of the global localization which is even more difficult. In this thesis we mainly focus on global localization.

During the past two decades, many algorithms using probabilistic approaches for localization have been proposed, including grid localization[7], Monte Carlo localization(MCL)[1], and many hybrid approaches. These algorithms represent the uncertainty of a robot's pose by using probability distributions over the whole space of robot's possible poses instead of relying on a single best guess[10]. The probabilistic localization algorithms are part of probabilistic robotics, a research area that represents information using the calculus of probability theory. Building on the filed of mathematical statistics, probabilistic robotics endows robots with a new level of robustness in real-world situations[10].

Among all the probabilistic localization algorithms, grid localization and Monte Carlo localization(MCL) are most widely used, especially Monte Carlo localization. Grid localization approximates robot's pose in a metric model of environment[7]. The map of the environment is divided into grid cells, and each grid cell stores the probability that the robot is in this cell. MCL represents the pose of robot by maintaining a set of particle samples, which are randomly drawn according to the probability distributions of the robot's pose[1].

1.1 Motivation and Contribution

Grid localization and MCL are two of the most widely used approaches for localization, especially the MCL. Each of these two popular methods has its own problems. How to reduce the computation cost and better the accuracy is our main concern. In order to improve the performance of localization, we propose two improved localization algorithms both of which are extension of MCL. One is called *Moving Grid Cell Based MCL* which combines grid localization and MCL, and the other is *dynamic MCL based on clustering* which employs a clustering component to reduce the computational cost in the localization process.

1.2 Outline

The rest of the thesis is structured as follows.

Chapter 2: Background knowledge. This chapter provides the background knowledge of our proposed methods. First, we will explain the idea of probabilistic robotics, then the mobile localization problem is discussed. What's more, two main algorithms for mobile robot localization , grid localization and Monte Carlo localization, are presented.

Chapter 3: Proposed methods. In this chapter, two proposed methods are presented separately. Details of each method are discussed, also the illustrations of how each method works are shown.

Chapter 4: Experiment results. In this chapter, experiment results are demonstrated which show the advantage of both proposed methods compared with traditional MCL. Both experiments in the real environment of the physical world and simulated environment on PC are implemented.

Chapter 5: Conclusion and future works. The conclusion of the thesis is given in this chapter, and the future work is also presented.

Chapter 2

BACKGROUND KNOWLEDGE

This chapter provides the background knowledge of our proposed methods. We first review the basic ideas of probabilistic robotics. Then the problem of mobile robot localization is introduced. After that we explain the related knowledge about localization. Finally the most widely used two algorithms for mobile robot localization , grid localization and Monte Carlo localization, are discussed.

2.1 Uncertainty

By definition, robotics is the science of sensing and acting on the physical world by using computer-controlled devices[10]. Robotics systems have been widely used in the world around us and playing an increasing important role. For a robot , it usually consists of the four main components. (1) a physical body, so it can exist in the real world. (2) sensor, so it can sense the environment. (3) effector and actuators, so it can act. (4) a controller, so it can be autonomous[18].

To do tasks in the real world, robot has to accommodate many uncertainties[10], which are caused by a number of factors. First, the environments of the robot are usually unpredictable especially in the highly dynamic environments such as highways and offices. Second, the sensors always have their limitations. The range and resolution of a sensor relies on its physical limitations and the noises. Third, the motor used for the robot actuation is unpredictable. Control noise and mechanical failure always cause uncertainty. Fourth, the software of the robot may also cause uncertainty as all internal models of the physical world are approximate. The real world cannot be fully extracted into models. Finally some uncertainty arises from algorithmic approximations. In a real-time system, accuracy sometime has to be sacrificed in order to achieve timely response.

As the robot is more and more widely used, uncertainty is becoming a major issue for the design of robot systems. How to cope with uncertainty is the main concern for researchers.

2.2 Probabilistic Robotics

Probabilistic robotics is relatively new in the area of robotics which addresses the problem of uncertainty. The key idea in probabilistic robotics is to represent uncertainty using probability theory. Instead of a single best guess, probabilistic robotics represents information by using probability distributions over all possible guesses.[10]

Compared with traditional methods, probabilistic methods have a weaker requirement on the accuracy of the robot's model, so it prevents the programmer from the heavy workload of building accurate models. What's more, probabilistic methods have lower requirements on the accuracy of robotic sensors. Building on the filed of mathematical statistics, probabilistic robotics endows robots with a new level of robustness in real-world situations, such as localization[19], mapping[25], simultaneous localization and mapping

(SLAM)[22], planning[10] and control[10].

2.2.1 State

In probabilistic robotics, the *environment* is a dynamical system that possesses internal state. Robots can get information about the environment through sensors and maintain an internal belief about the environment.

Environments are characterized by *state*[10]. It is the collection of the information about the robot and its environment. State that changes over time such as moving people around the robot is called *dynamic state*, while others that remain static such as the location of a wall are called *static state*. The state also includes variables about robot itself such as pose, velocity and so on.

Typical state variables used in robotics are: (1) robot's pose which consists of location and orientation in a global coordinate. (2) in robot manipulation the state includes variables for the configuration of the robot's actuators which is often referred to as kinematic state. (3) robot's velocity and the velocities of its joints, which are usually referred to as dynamic state. (4) location and features of surrounding object in the environment. An object may be a wall or a desk, and features may be the visual appearance such as color or texture. (5) locations and velocities of moving objects and people may be state variables too.[10]

A state is called *complete* if it is the best predicator of the future. But in practice it is not possible to get a complete state for a robot system. A complete state not only includes all aspects of the environment that may affect the future but also the robot itself. Some of these aspects are very hard to get.

In this thesis, we use x_t to denote the state at time t and time is discrete, which means all event will take place at discreet time step t = 0, 1, 2, 3 ....

2.2.2 Environment Interaction

Between the robot and the environment there are two fundamental interactions: environment sensor measurement and control actions[4][10]. The robot can obtain information about the state of the environment through its sensors, and affect the environment through its actuators. Examples of the first type of interaction include the camera image or a range scan. The result of a perceptional interaction is called a measurement. Usually, sensor measurements arrive have some delay, so they provide information about the state of certain moments ago. Examples of the second type of interaction include the motion of robot or manipulation of an object. We assume that the robot always takes control actions even it does not perform any action itself. In practice, the robot continuously takes control actions and gets measurements at the same time.

The robot keeps a record of all past sensor measurements and control actions, which is referred to as the *data*. Through the two types of interactions , the robot receives two different data streams, measurement data and control data.

Measurement data gives a robot the information regarding of the momentary state of the environment. We assume that the robot gets one measurement at one time. The measurement data from time t_1 to time t_2 is denoted as $z_{t_1:t_2}$ and the measurement data at time t is denoted as z_t. Control data sometimes is also referred to as movement data or motion data in the context of mobile robot localization problem. We also assume that there is only one control data at one time, even the robot does not do anything. Control data provides information about the changes of the state. We use $u_{t_1:t_2}$ to denote the movement data from time t_1 to time t_2, and u_t to denote the movement data at time t.

Both measurement data and control data play very important roles. On one hand, measurement data provides information about the environment which helps to increase the robot's knowledge. Control data, on the other hand, brings a loss of knowledge because of the uncertainly in the real world. One thing needs to be emphasized on is that the sensor measurement and control actions take place at the same time.

Probabilistic approaches for robotics have two different components to process these two kinds of data[10]. One is measurement model, and the other is motion model. Measurement model, denoted as $p(z_t|x_t)$, is the conditional probability of z_t given the state x_t. Motion model is the state transition probability $p(x_t|u_t,x_{t-1})$. It is the posterior distribution of x_t after incorporating the control data u_t at x_{t-1}. This two models are very important for estimating robot's state.

2.2.3 Probabilistic Generative Laws

The evolution of state is controlled by probabilistic laws. The state x_t is conditioned on all past states, measurements and controls, which can be presented in the following form: $p(x_t|x_{0:t-1},z_{1:t-1},u_{1:t})$[10]. Here we assume that robot first takes a control action u_1, then gets a measurement z_1. If the state is complete then it is a sufficient summary of all past events. Particularly, x_{t-1} is a sufficient statistic for all previous controls $(u_{1:t-1})$ and measurements $(z_{1:t-1})$ up to time $t-1$. So state x_t could be expressed as $p(x_t|x_{0:t-1},z_{1:t-1},u_{1:t}) = p(x_t|x_{t-1},u_t)$. The equation is an instance of conditional independence, which means if we know the values of the conditioning variables, such as x_{t-1},u_t, then certain variables, such as x_t, are independent of other variables, such as $z_{1:t-1}$ and $u_{1:t-1}$. Also, if x_t is complete, we will get another important conditional independence: $p(z_t|x_{0:t},z_{1:t-1},u_{1:t}) = p(z_t|x_t)$, which means the state x_t is sufficient to predict the measurement data z_t, in other words, other variables such as past control data, measurement data and past states are not relevant.

The probability $p(x_t|x_{t-1},u_t)$ is called state transition probability, which shows how state x_t evolves based on the control data u_t and the previous state x_{t-1}. The probability $p(z_t|x_t)$ is the measurement probability which specifies how measurement data z_t is generated according to the state x_t. The state transition probability and the measurement probability present the dynamical stochastic system where the robot exists. Figure 2.1 illustrates the evolution of state and measurements. State x_t is stochastically dependent on the previous state x_{t-1} and the control data u_t, and the measurement z_t depends stochastically on the state x_t. The model in Figure 2.1 is well known as hidden Markov model or dynamic Bayes network[23][24][27].

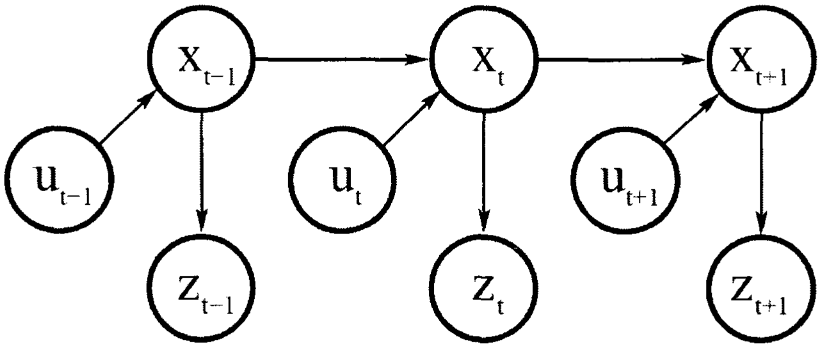

Figure 2.1: The dynamic Bayes network that characterizes the evolution of control, states and measurements.[10]

2.2.4 Belief

In this part we will introduce an important concept called *belief*[10]. Belief is robot's internal knowledge with respect to the state. The state usually cannot be measured directly, so the robot has to infer its belief from the data collected. In probabilistic robotics, belief is represented by conditional probability distributions.

A belief distribution assigns a probability to each possible state hypothesis with regards to the true state[10]. The belief at time t is denoted as $bel(x_t) = p(x_t|z_{1:t}, u_{1:t})$. It is a posterior probability over all possible states conditional on all past control data and all past measurement data collected so far. Sometimes it is often important to calculate a posterior after taking the control action u_t but before incorporating measurement data z_t, which is denoted as $\overline{bel}(x_t) = p(x_t|z_{1:t-1}, u_{1:t})$. This posterior is usually referred as prediction, and it reflects that $\overline{bel}(x_t)$ predicts state x_t based on previous state x_{t-1} without incorporating the measurement data z_t. Then we also need to calculate $bel(x_t)$ from $\overline{bel}(x_t)$ by incorporating z_t, which is called measurement update.

2.2.5 Bayes Filter

Bayes filter is a recursive algorithm that estimates the state of a dynamical system based on both measurement data and control data and it is the most general algorithm for calculating beliefs[1][10]. Table 2.1 shows a single iteration of the Bayes Filter algorithm. As shown in Table 2.1, the belief $bel(x_t)$ is calculated from the previous belief $bel(x_{t-1})$. The inputs are the $bel(x_{t-1})$, control data u_t and measurement data z_t. The output is the belief $bel(x_t)$. Bayes filter algorithm is performed in two essential phases, prediction phase (line 2) and update phase (line 3).

In the first phase, it calculates the belief over state x_t by incorporating the control data u_t based on the previous state x_{t-1}. Particularly, we can see that $\overline{bel}(x_t)$ is calculated by the integral of two probability distributions, the prior assigned to state x_{t-1} and the probability that u_t causes a transition from x_{t-1} to x_t[10]. This phase is called control update or prediction phase. In the second phase, it processes the probability that the measurement data z_t may be observed at state x_t and incorporates this probability $p(z_t|x_t)$ into $\overline{bel}(x_t)$.

Table 2.1: Bayes Filter [10]

```
Algorithm Bayes_filter ($bel(x_{t-1}, u_t, z_t)$)
1:  for all $x_t$ do
2:      $\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1}$
3:      $bel(x_t) = \eta p(z_t|x_t)\overline{bel}(x_t)$
4:  endfor
5:  return $bel(x_t)$
```

However $bel(x_t)$ may not integrate to 1, so it uses the normalization constant η to normalize the results. The second step is called measurement update or update phase.

Bayes filter needs an initial belief $bel(x_0)$ at time $t = 0$ as an input in order to recursively calculate the new belief. If the values of x_0 is known, then $bel(x_0)$ should be initialized with a point mass distribution which centers all probability mass on the value of x_0, and all the others are assigned a probability of zero. If the initial value x_0 is totally unknown, $bel(x_0)$ should be initialized using a uniform distribution over all possible values of x_0. If the initial value x_0 is partially known, then $bel(x_0)$ can be initialized by non-uniform distributions.

There is one important assumption called *Markov Assumption* which is adopted by Bayes filter. Markov assumption plays a fundamental role in this whole thesis. It assumes that past and future data are independent if the current state x_t is known. It tells that the current belief $bel(x_t)$ is sufficient to represent the past history of robot. In robotics, Markov assumption is only an approximation.

2.3 Mobile Robot Localization

Bayes filter is an important algorithm for state estimation problems, and it has many applications one of which is *mobile robot localization* problem. Mobile robot localization is the problem of determining a robot's pose given the map of the environment and the sensor readings[10][19]. It is one of the most important problems in mobile robotics as most robot's tasks need the positional information. In practice, the pose of robot cannot be sensed directly, so the pose has to be inferred from measurement data and control data. Also, a single measurement data is usually not enough to determine the pose, so the robot has to integrate data over time. Figure 2.2 illustrates a graphical model for localization. The goal of the robot is to determine its position based on the measurements and movements given the map of the environment. In Figure 2.2, the values of shaded nodes are known including the map *m*, the measurement *z* and the control *u*. The goal of localization is to calculate the robot's pose *x*.

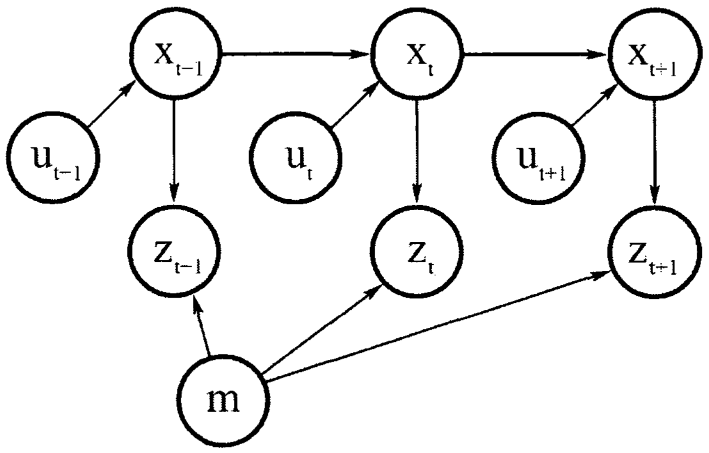

Figure 2.2: Graphical model of mobile robot localization.[10]

2.3.1 Category of Localization

From different aspects, localization can be divided into many different categories. According to the nature of environment and the initial knowledge of the robot, here we discuss four important types of localization problems[10].

Firstly, localization problems can be characterized by the type of knowledge whether is known at the beginning or at run-time. Under this category, there are three kinds of localization problems with an increasing difficulty. Position tracking(or local localization) is the simplest one. The initial pose of robot is known and the localization is done by accommodating the noise in the robot's movement. The uncertainty of the pose is usually approximated by a unimodal distribution such as a Gaussian. It is a local problem as the uncertainty is local and restricted to places near the robot's true pose. In global location, the initial pose is not known and the robot is placed somewhere in the environment. Global localization is more difficult than position tracking since it has to determine its pose from scratch. The third problem is called kidnapped robot problem. It is a variant of the global localization but more difficult. The robot is kidnapped and taken to somewhere else without being notified. Kidnapped robot problem becomes important because even the most state-of-the-art localization approaches can fail sometimes. The ability to recover from failures is especially important for truly autonomous robots.

Secondly, the environment has a substantial impact on the difficulty of localization[19]. Environments can be static or dynamic. In static environment the only variable quantity is the robot's pose. All other objects in the environment remain at the same place all the time. However in a dynamic environment, objects may change its position or configuration from time to time. Example of changes are like people, movable furnitures and so on. Most real environments in the physical world are dynamic. It is clear that localization in dynamic environments is more difficult than that in static ones.

Thirdly, according to whether or not the localization algorithm controls the motion, the localization can be divided into passive localization and active localization[6][17]. In passive localization, the localization algorithm only observes on the robot's operating, and has nothing to do with the control of robot. The motion of robot is not designed to facilitate localization so the robot may move randomly. In active localization, the algorithm controls the robot in order to minimize the error or cost during the localization. Active localization algorithms usually produce better results than passive ones.

Lastly, with respect to the number of robots involved the localization can be divided into single-robot localization and multi-robot localization[8][20]. Single robot localization is the most studied approach. It handles a single robot only and there is no communication problems since all the data is collected to a single robot platform. In multi-robot localization, the robots have to detect each other. The issues that arises usually include representation of beliefs and the communication between different robots.

In this thesis, we focus on the global passive localization for a single robot in a static environment.

2.3.2 Markov Localization

Localization algorithms are variants of the Bayes filter. In the context of localization, Bayes filter is also known as Markov localization[10][19]. Table 2.2 depicts the basic algorithm. Comparing with Table 2.1, we can see that the difference is that Markov localization needs the map m of the environment as one input. The map m is very important in the measurement model $p(z_t|x_t,m)$ (line 3), and is also needed in the motion model $p(x_t|u_t,x_{t-1},m)$ (line 4). The same as Bayes filter, Markov localization calculates the probabilistic belief

Table 2.2: Markov localization [10]

```
Algorithm Markov_localization ($bel(x_{t-1}, u_t, z_t, m)$)
1:  for all $x_t$ do
2:      $\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1}, m)bel(x_{t-1})dx_{x-1}$
3:      $bel(x_t) = \eta p(z_t|x_t, m)\overline{bel}(x_t)$
4:  endfor
5:  return $bel(x_t)$
```

$bel(x_t)$ at time t from time $t-1$ recursively.

Markov localization is able to handle the position tracking problem, the global localization problem and the kidnapped robot problem in static environment. In position tracking the initial pose is known, so $bel(x_0)$ is initialized by a point-mass distribution. However in practice the initial pose is often known in approximation, so $bel(x_0)$ is usually initialized by a Gaussian distribution centered around x_0. In global localization, the initial pose is unknown, so $bel(x_0)$ is initialized by a uniform distribution over all possible spaces in the map.

Markov localization is independent of the representation of the state space and it can be implemented by using different state representation methods, for example, histogram filter and particle filter.

2.3.3 Representation of State Space

In this part, we will discuss two state representation methods[21], histogram filter[7] and particle filter[26]. They approximate posterior over continuous spaces with finite values. Histogram filter decomposes the state space into finite regions and represents the cumulative posterior for each region by a histogram which assigns a single probability value to each region. Particle filter approximates the posterior by a finite number of samples which populate the state space, and the samples are drawn randomly from the posterior. As these two methods are well-suited for representing multi-modal beliefs, they are widely used when a robot has to deal with global uncertainty, such as global localization problem[10].

2.4 Localization Algorithms

Since mobile robot localization is one of the most important and fundamental problems in the field of mobile robot, so there are a number of probabilistic algorithms proposed for mobile robot localization. Many of them only address the position tracking problem, such as Extended Kalman Filter(EKF)[10]. They all employ Kalman filter which is based on the assumption that the uncertainty of the robot's pose can be represented by a unimodal Gaussian distribution. What's more, they adopt other assumptions such as Gaussian distributed noise and Gaussian distributed initial uncertainty. Under these assumptions Kalman filter performs very well for position tracking problem. In global localization problem the uncertainty of robot needs to be represented by multi-modal distributions, but Karman filter cannot, so it is not useful when dealing with global localization problem. In order to overcome this limitation of Kalman filter, Multi-hypothesis tracking(MHT) algorithm [5]represents the belief of pose by multiple Gaussians, which is mixture of normal distributions. It can handle the global localization problem, but the computational cost is very high. However grid localization and Monte Carlo localization(MCL) could handle multi-modal distribution at a reasonable computational cost which makes them suitable for global localization problem. In the following, we will discuss these two important global localization algorithms.

2.4.1 Grid Localization

In grid localization[7], it uses a histogram filter to represent the posterior belief over a grid decomposition of the pose space. Figure 2.3 demonstrates an example of grid decomposition. The map of the environment is divided into many grid cells. Each grid cell represents a robot's possible pose in the environment. Each layer represents a different orientation of the robot, and in this example only three orientations are shown.

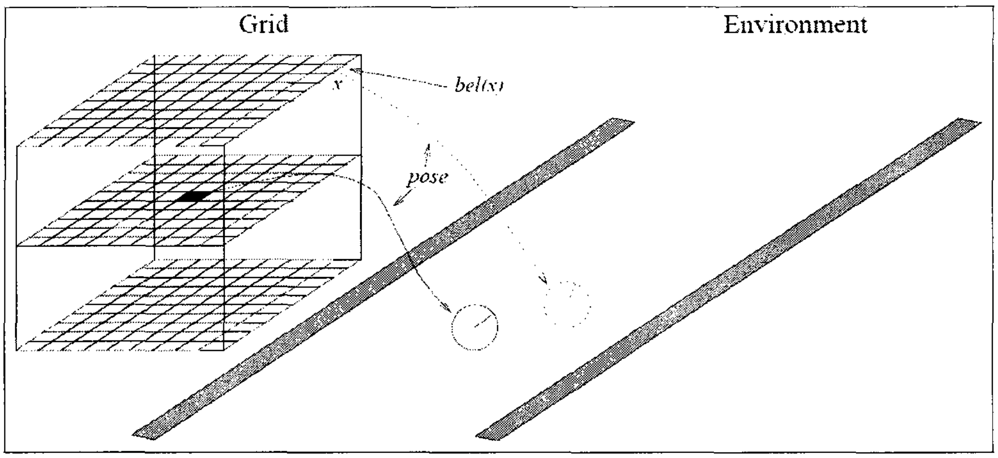

Figure 2.3: Example of grid decomposition over the robot pose.[10]

The algorithm of grid localization is depicted in Table 2.3. Grid cell is denoted as x_k, and each grid cell is attached with a probability $bel(x_t) = \{p_{k,t}\}$, which stands for the possibility that the robot is in this grid cell. The notion $mean(x_k)$ stands for the center-of-mass of the grid cell x_k. Grid localization is also a recursive algorithm, and in Table 2.3 it shows a single iteration. It needs the previous value $\{p_{k,t-1}\}$, the most recent measurement data z_t, control data u_t, and the map m. It goes through all the grid cells each time and

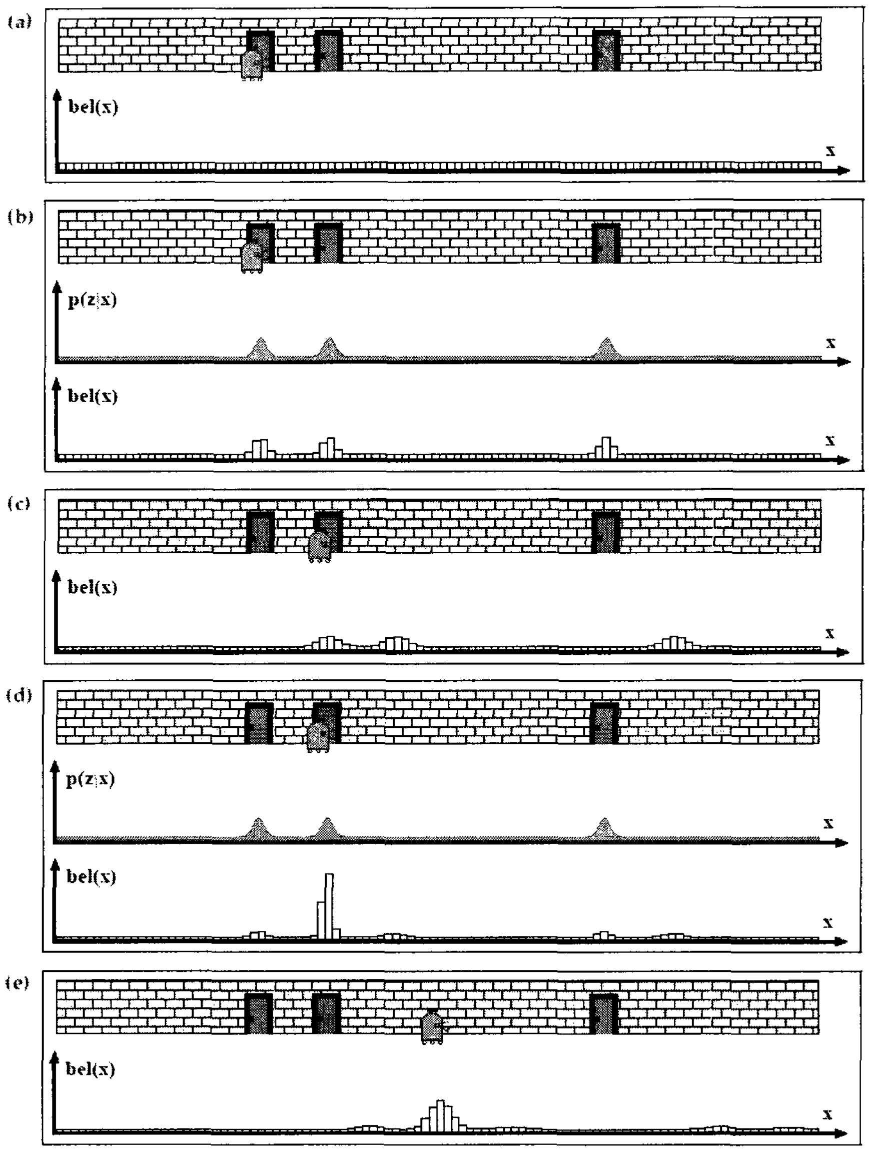

Figure 2.4: Example of grid localization in one-dimensional hallway.[10]

Table 2.3: Grid localization algorithm [10]

```
Algorithm Grid_Localization($\{p_{k,t-1}\}, u_t, z_t, m$)
1: for all $k$ do
2:     $\bar{p}_{k,t} = \sum_i p_{i,k-1}$ motion_model $(mean(x_k), u_t, mean(x_i))$
3:     $p_{k,t} = \eta$ measurement_model $(z_t, mean(x_k), m)$
4: endfor
5: return $\{p_{k,t}\}$
```

updates the probability for each grid cell. Line 2 incorporates the control data and line 3 incorporates the measurement data.

Figure 2.4 shows an example of grid localization in a one-dimensional hallway. In Figure 2.4 The robot starts without knowing its pose, so the belief is represented by a uniform histogram. Then in the following pictures, as it moves and senses, some grid cells' probability values are increasing while some are decreasing.

There are two issues in grid localization. One is the trade-off between the resolution of grid cells and the accuracy of result, and the state transition problem. The result of the grid localization depends on the resolution of grid cells. A finer resolution produces a better result, but also requires greater computational cost. While with a coarse resolution of grid cells, though the computational cost is reduced, the result may not be accurate. Another issue is in the motion model when dealing with a high-resolution measurement model and a coarse-resolution motion model. As only using the center of a grid cell to represent the grid cell, which in the Table 2.3 is denoted as $mean(x_k)$, may lead to a poor result. For instance, if the robot moves 1 cm/s, and the motion model updates every second, while the size of the grid cell is 15 cm. The robot may stay in the same grid cell even after the robot moves several steps. It will not cause a state transition. A common solution to this issue is to modify both the motion model and the measurement model by inflating the amount of noise. However this solution will reduce the information extracted from the measurement. Similarly we can also inflate the motion model so that the robot can be guaranteed to move from one grid cell to another even the motion between each update is smaller than the size of the grid cell. However this may make the robot move faster than commanded, which will cause more uncertainty in the process of localization.

2.4.2 Monte Carlo Localization

In this part, we will introduce Monte Carlo localization (MCL)[1], one of most popular localization algorithms. It is easy to implement and works both for position tracking and global localization problems [10] .

The filter used in MCL which represents posteriors by finitely many samples is known as particle filter[26], which we introduced in the previous part. MCL represents the belief $bel(x_t)$ by a set $\chi_t = \{x_t^{[1]}, x_t^{[2]}, \dots, x_t^{[M]}\}$ of M particles over the entire state space, and each particle denotes a possible pose of the robot. MCL is also a version of sampling/importance re-sampling (SIR)[11].

The MCL algorithm is depicted in Figure 2.4, and it shows a single iteration. The initial belief $bel(x_0)$ is represented by M particles that are uniformly and randomly distributed in the whole state space of the environment and each particle is assigned with a weight of M^{-1}, which is called importance factor. In each iteration, MCL algorithm takes as inputs the previous belief $bel(x_{t-1})$, movement data u_t, measurement data z_t and the map m of the environment .

MCL includes the following three steps:

Table 2.4: MCL algorithm [10]

```
Algorithm MCL($\chi_{t-1}, u_t, z_t, m$)
1:  $\bar{\chi}_t = \chi_t = \phi$
2:  for $m = 1$ to $M$ do
3:      $x_t^{[m]}$ = sample_motion_model($u_t, x_{t-1}^{[m]}$)
4:      $w_t^{[m]}$ = measurement_model($z_t, x_t^{[m]}, m$)
5:      $\bar{\chi}_t = \bar{\chi}_t + < x_t^{[m]}, w_t^{[m]} >$
6:  endfor
7:  for $m = 1$ to $M$ do
8:      draw $i$ with probability $\alpha$ $w_t^{[i]}$
9:      add $x_t^{[i]}$ to $\chi_t$
10: endfor
12: return $\chi_t$
```

(1) Robot motion. Motion model for sample particles is applied in line 3. After each step of movement, MCL incorporates the movement data u_t, and from the previous particle set it generates M new particles that approximate the robot's pose.

(2) Robot measurement. Measurement model is applied in line 4. In this step, sensor readings are incorporated by reweighting the sample set, during which the weight of each particle will change.

(3) Important resampling. This phase(line 7-10) is often referred as sequential importance sampling with resampling. New unweigted particles are drawn from the current sample set. The probability of drawing a new particle is related to it weight (importance factor).

When the MCL finishes successfully, most particles will converge to a certain area which represents the position of robot. In Figure 2.5, an example of MCL is shown. In

Figure 2.5 (a) The robot is globally uncertain about its position and the particles spread all possible spaces. In Figure 2.5 (b), when the robot reaches the upper left corner of the map, its belief is still concentrated around four possible locations. In Figure 2.5 (c), finally after several movements, the robot localizes itself and all particle converge to a small area.

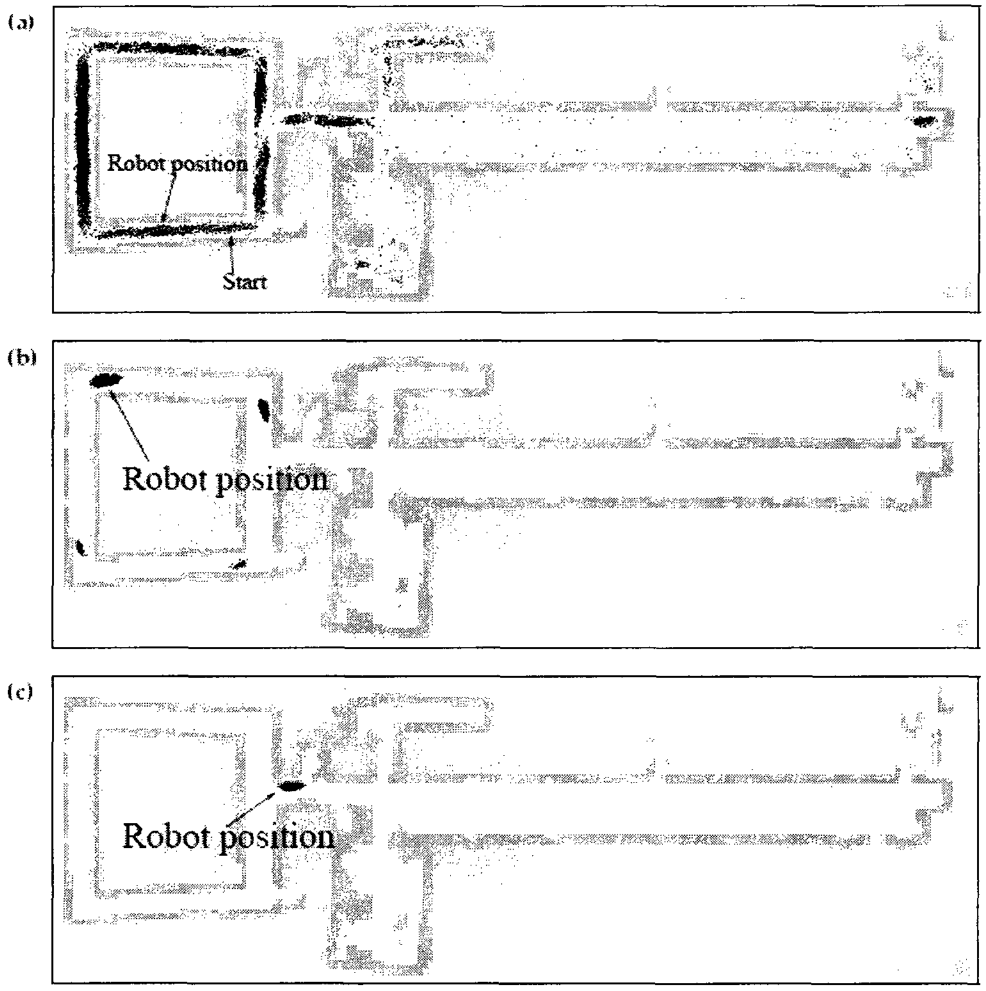

Figure 2.5: Illustration of Monte Carlo localization.[10]

Chapter 3

PROPOSED METHODS

[1]

3.1 Motivation

As discussed in the background knowledge, the grid localization has two problems: (1) the trade-off between the resolution of grid cell and accuracy of result and (2) the state transition problem. For the MCL algorithm, although it is one of the most efficient algorithms for mobile robot localization, yet it is still able to be improved with respect to efficiency and computational cost. For that purpose, how to reduce the number of particles and reduce the computational cost is the key concern. During the past years many algorithms that extend MCL have been proposed in order to further improve the performance and reduce the computational cost, such as adaptive samples based MCL approach[12], mixture MCL approach[14], coevolution based adaptive MCL[13], reverse MCL Approach[9] and so on.

[1]This chapter also incorporates the outcome of a joint research undertaken in collaboration with Jingxi Chen and Sepideh Seifzadeh under the supervision of Dr. Dan Wu. In all cases, the key ideas, primary contributions, experimental designs, data analysis and interpretation, were performed by the author, and the contribution of co-authors was primarily through the provision of constructive comments.

However combining grid localization and MCL is not well exploited, and few efforts have been made in incorporating clustering approach into MCL, so in this thesis we propose two different novel extensions of MCL. One is Moving Grid Cell Based MCL, which combines grid localization and MCL, and the other is dynamic MCL based on clustering, which employs a clustering component in the localization process.

3.2 Moving Grid Cell Based MCL

The first proposed method is called *Moving Grid Cell Based MCL*. It is based on grid localization and MCL, and could solve the problems existing in the traditional grid localization and reduce the computational cost of the whole localization process.

There are three parts of this method: (1) The first part is called moving grid cell localization part, and in this part the grid cells are used the same way as how particles are used in traditional MCL. The size of grid cells here is bigger than the size of those used in traditional grid localization, which makes the the number of grid cells much smaller. We consider each grid cell as a particle and apply MCL algorithm on these grid cells. Hence, these grid cells are moveable instead of stationary. So we eliminate the state transition problem of the traditional grid localization. Since the size of grid cells is big, so we can only get a coarse pose of robot once MCL algorithm is finished. (2) The second part is called verification part. we add the verification part in order to verify the result of moving grid cell localization part. This will help to improve the accuracy of localization. The verification is achieved through comparing the expected measurement data based on the previous result with the real measurement data the robot gets. If the difference of these two data is out of a certain range, then it suggests the accuracy does not meet our requirement, and the algorithm will go through the moving grid cell localization part again. (3) The third

Table 3.1: Moving Grid Cell Based MCL

```
Algorithm Moving_Grid_Cell_Based_MCL(Map)
1: Moving_Grid_Cell_Localization
2: Verify_Grid_Cell_Localization
3: if Verify_flag==true
4:     then go to line 6
5: else go to line 1
6: Initialise_MCL
7: MCL
```

part is traditional MCL part. We apply traditional MCL algorithm to obtain the final pose of the robot. Instead of in the whole environment, the particles are only initialized in the restricted area covered by those grid cells produced by the first part. Both the large size of grid cell in the first part and the restricted area where the particles are initialized in the third part let us use a smaller number of both grid cells and particles. This helps to reduce the computational cost. We outline the proposed *Moving Grid Cell Based MCL* algorithm in Table 3.1.

As shown in Table 3.1, the algorithm needs the map of the environment as an input. As mentioned there are three parts of our proposed algorithm, including (1) moving grid cell localization part (line 1), (2) verification part (line 2-5), and (3) the MCL part (line 6-7). Line 1 applies the moving grid cell localization algorithm which is shown in Table 3.2. Line 2 applies the verification grid cell localization algorithm which is shown in Table 3.3 . Line 7 applies the traditional MCL algorithm which is shown in Table 3.4. In the following, we will give the detailed descriptions of each part.

Table 3.2: Moving Grid Cell Localization

```
Algorithm Moving_Grid_Cell_localization(Map)
1: Intialise_Grid_Cell
2: For all grid cells
3:     Grid_Cell_Predict
4:     Grid_Cell_Update
5: End for
6: if Grid_Cell_Number < Grid_Cell_Threshold
7:     then Return Grid_Cell_Result
8: else go to line 2
```

3.2.1 Moving Grid Cell Localization Part

In the traditional grid localization, the number of grid cells is usually very large in order to get an accurate result. Two factors will affect the number of grid cells. One factor is the size of the grid cell, and the other is the resolution of orientation. A smaller cell size and finer resolution of orientation will lead to a more accurate result, however they will greatly increase the computational cost. In this part, we use a larger size of grid cell and only a small number of orientation, which makes the number of total grid cells much smaller.

As shown in Table 3.2, we first initialize all grid cells with equal probability that sum up to 1. We use 3-D representation of the map which includes x-dimension, y-dimensions and the orientation θ. The way the grid cell used in this part is the same as the particle used in the traditional MCL algorithm, so the grid cell can be regarded as a big particle. During the localization process the grid cell is moving like a particle. In line 3 it incorporates the movement data. Instead of stationary, the position of all grid cells will change in accordance with the movement of robot. If the grid is out of the environment, its probability will be set to zero. Line 4 will incorporate the measurement data and update the probability for grid cells. Then if the grid cell has a probability of zero, it will be removed from the grid cell set. As the number of grid cells reduces, when it reaches a certain predefined threshold according to the map, the moving grid cell localization will stop and return the grid cells left to the next part, otherwise it will continue.

We use a less number of grid cells and treat them as particles to get a coarse position in this part, and then in the third part we will get a more accurate position. As the initial number of grid cells is smaller, the computational cost is reduced. Moreover, during the process as the number of the grid cells is reducing as the probability of many grid cells are becoming zero which makes them discarded, so the computational cost is reduced further. Since we only use a small number of orientations for each grid cell in the part, in the third part we will compensate for this loss of accuracy in orientation, and we will explain this in more details in the third part. Because the grid cell is now moving in our proposed algorithm, the state transition problem existing in the traditional grid localization is avoided. Therefore, we don't have to worry about the motion model, the robot can move at any speed. After this first part, only several grid cells are left, the probability that these grid cells contain the true pose of robot are very high.

3.2.2 Verification Part

The Verify Grid Cell Localization algorithm is shown in Table 3.3. The input of this algorithm is the grid cells returned in the first part. For each grid cell returned from the moving grid cell localization part, it will first calculate how long it takes for the robot to reach the next landmark according to the current pose suggested by the grid cells, which is referred

Table 3.3: Verify Grid Cell Localization

```
Algorithm Verify_Grid_Cell_Localization(Grid_Cell_Result)
1: For all Grid_Cell_Result
2:     Grid_Cell_Predict
3:     Get_Measurement
4: End for
5: Compare Measurement with the Expectation
6: if difference within a Threshold
7:     then Verify_flag=true
8: else Verify_flag=false
```

to as *Expectation* in line 5, then it will let the robot move, and record the time the robot takes to reach the next landmark in the real environment.

If the difference between these two recorded times for each grid cell is within a certain predefined range, which means the results returned from last part are reliable, then the verification result will be true, and it will move on and pass the verified grid cell results to the MCL part. If the difference is out of the predefined range, which means the results from the first part are not reliable, and the verification result will be false, so it needs to go back to the first part and go through it again.

The verification part helps to improve the accuracy of localization. When the result accuracy of the first part does not meet the our requirement, the difference between the calculated time (*Expectation*) and the time robot takes in the real environment will be big, so the algorithm in Table 3.3 will find this out and go back to the moving grid cell localization part again.

Table 3.4: MCL algorithm [10]

```
Algorithm MCL($\chi_{t-1}, u_t, z_t, m$)
1:  $\bar{\chi}_t = \chi_t = \phi$
2:  for $m = 1$ to $M$ do
3:      $x_t^{[m]}$ = sample_motion_model($u_t, x_{t-1}^{[m]}$)
4:      $w_t^{[m]}$ = measurement_model($z_t, x_t^{[m]}, m$)
5:      $\bar{\chi}_t = \bar{\chi}_t + < x_t^{[m]}, w_t^{[m]} >$
6:  endfor
7:  for $m = 1$ to $M$ do
8:      draw $i$ with probability $\alpha\ w_t^{[i]}$
9:      add $x_t^{[i]}$ to $\chi_t$
10: endfor
12: return $\chi_t$
```

What's more, the verification part can be adjusted according to different situations. If a high accuracy is required, we can make a more complex verification in this part, which means not only to test the the next landmark, also the second next landmark and so on.

3.2.3 The MCL Part

The third part is the regular MCL part. It is the same as the traditional MCL as shown in Table 3.4 except how the particles are initialized. From the previous parts we have obtained a coarse pose of robot , so we only need to initialize the particles in the restricted areas instead of in the whole environment.

In line 6 of Table 3.4, the particles are generated within the grid cells which are returned from the moving grid cell localization part and verified in the second part. The x-dimension and y-dimension is randomly and uniformly generated inside the grid cells, but the orientation θ is generated according to the grid cell's orientation where the particle is in. Because in the moving grid cell localization part, the orientation of the grid cell is discrete and not all possible orientations are covered by grid cells, so if the orientation of the robot doesn't fall into the discrete orientation we choose, the accuracy might be questionable. So in the MCL part, we initialize the orientation of particles according to the grid cell's orientation, for example, if the orientation of the grid cell is θ, then the orientation of particles in this grid cell may be between $\theta - 15$ and $\theta + 15$. This will help to compensate the possible inaccuracy of the orientation in the moving grid cell localization part.

During the process of the MCL algorithm, the probabilities of particles are updated based on the motion model and measurement model. The MCL goes on until the localization is finished. The number of the particles used in this part is not fixed, we can change the number according to different situations based on the requirement of accuracy.

It is noted that since we already obtain a coarse pose of the robot in the moving grid cell localization part, then we generate particles only in a restricted area of the environment. We do not need as many particles as those used in the traditional MCL in which particles have to be populated in the whole possible spaces.

3.2.4 Illustration of the Proposed Method One

Figure 3.1 shows the progress when executing the proposed method one in a simulated environment. The big blue circle denotes the robot and the black line denotes the boundary of the environment. In Figure 3.1 (A) and (B), the colored squares denote the grid cells, and in Figure 3.1 (C) and (D) the small red circle denotes the particle.

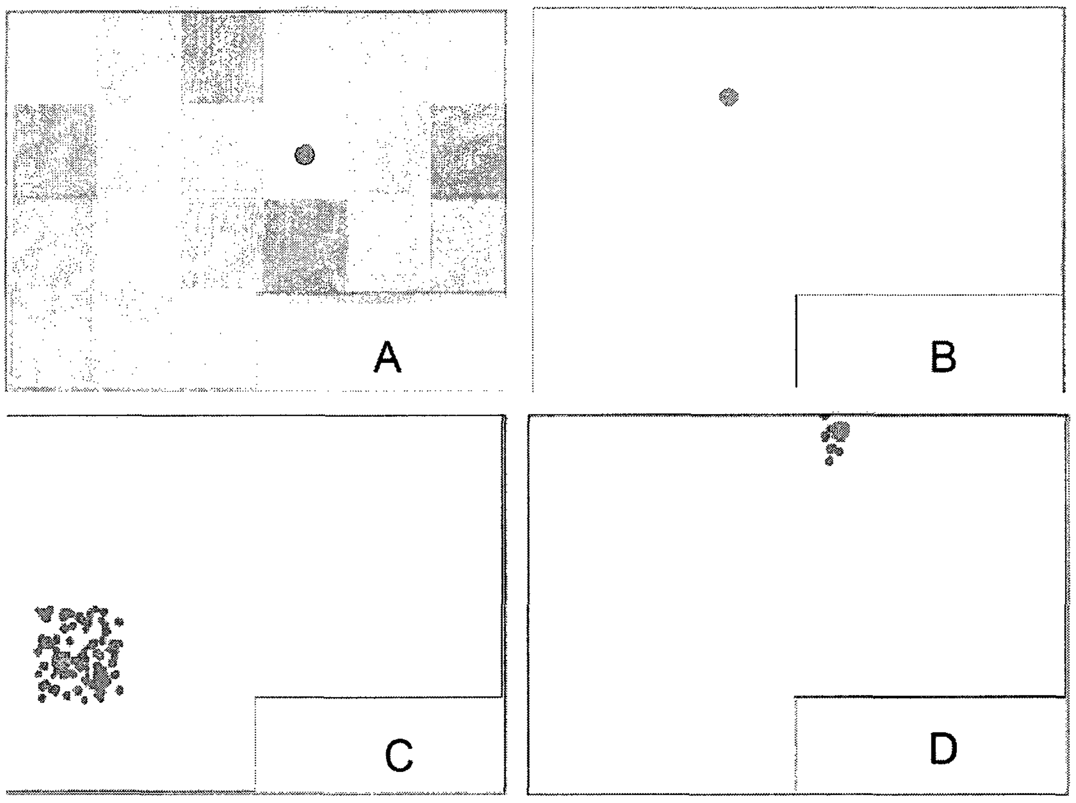

Figure 3.1: Illustration of the Proposed Method One. (A) Initialization of grid cells, (B) The moving grid cell localization finished, (C) Initialization of particles after verification, (D) Final result.

Figure 3.1 (A) shows the grid cells initialized in the the first part. Each colored square denotes a moving grid cell, and in order to make it easy to distinguish each grid cell, the color of a grid cell is different from its neighbors. Figure 3.1 (B) shows the position of the grid cell left after the moving grid cell localization part is done. Figure 3.1 (C) shows the particles initialized after the verification part, and Figure 3.1 (D) shows the result after the MCL part. Figure 3.1 (D) shows the final position of robot after the whole algorithm is finished.

3.3 Dynamic MCL Based on Clustering

The second proposed method is *dynamic MCL based on clustering*. In [16] a novel method based on clustering is proposed to help robot to be aware of its progress of localization. Inspired by that, we propose a dynamic MCL which significantly reduces the number of particles during the execution of localization by employing a clustering component. The overall structure of the proposal method is shown in Table 3.5.

As shown in Table 3.5, the second proposed method consists of three parts: (1) MCL+BSAS part (line 1-6), (2) Reducing part (line 7), (3) MCL part (line 8). The four inputs of the method are the map *Map* of the environment, the initial particle set χ which populates the whole environment, threshold θ for distance similarity used in the BSAS algorithm, and threshold η used for termination of the first part. Before we give the detailed descriptions of each part, first we will introduce the background knowledge of clustering.

Table 3.5: Dynamic MCL

```
Algorithm Dynamic MCL ($Map, \chi, \theta, \eta, n$)
1: Do {
2:     $\chi_t = MCL(\chi_{t-1}, u_t, z_t)$
3:     $C_t = BSAS(\chi_t, \theta)$
4:     $m = Max(C_t)$
5:     $p = m/N_{total}$
6:     } While ($p < \eta$)
7: $\chi'_t = Reduce(\chi_t, n)$
8:     MCL ($Map, \chi'_t$)
```

3.3.1 Clustering and BSAS Algorithm

By definition, a cluster is "an aggregate of points in the test space such that the distance between any two points in the cluster is less than the distance between any point in the cluster and any point not in it" [28]. Cluster analysis or clustering is the assignment of a set of points into clusters.

An important part in all clustering algorithms is to select a proximity measure or distance measure, which determines how the similarity of two data points is calculated[29]. The proximity measure affects the shape of the clusters, as some elements may be close to one another according to one distance and far away according to another. In the context of MCL localization, the pose of a robot consists of x and y coordinates and the accuracy of localization result has strong relation with Euclidean distance, so it is effective and reasonable that we choose the Euclidean distance $d(P_i, P_j) = \sqrt[2]{(x_i - x_j)^2 + (y_i - y_j)^2}$ as our proximity measure for two points P_i and P_j when clustering particles.

During clustering in order to calculate the distance $d(P_i, C_k)$ between a particle P_i and a cluster C_k which usually already contains a lot of particles in it, we need a representa-

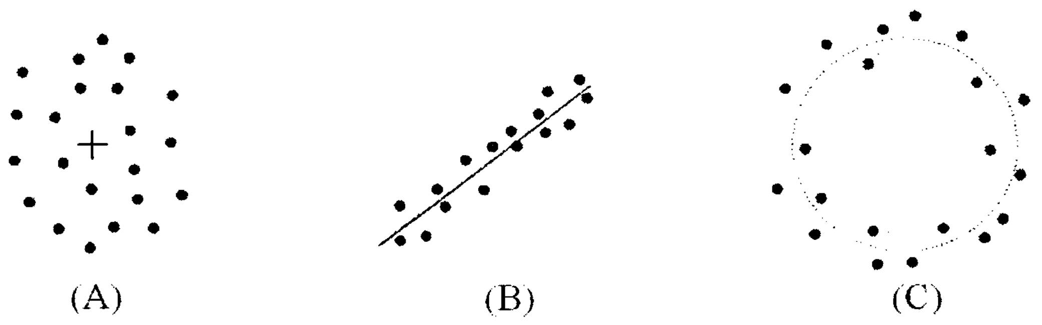

Figure 3.2: Cluster Representatives. (A) Point representative for compact clusters, (B) Hyperplane representatives for clusters of linear shape, (C) Hyperspherical representatives for clusters of hyperspherical shape.[15]

tive of the cluster C_k. As shown in Figure 3.2, there are three common options for representing the cluster, point representatives, hyperplane representatives and hyperspherical representatives[15]. In these three methods, the point representative is most suitable for compact clusters that usually appear in MCL. Therefore, for a cluster containing N particles, we use the mean point $P_{mean} = \frac{1}{N}\Sigma(P_i)$ as the representative of the cluster which is a very common choice.

Many types of algorithms have been proposed in the field of clustering, such as hierarchical clustering, partitional clustering, kernel-based clustering, sequential data clustering and so on[29][31]. Since in the localization we need to process the particles in real time, so the efficiency of clustering algorithm is very important and crucial for real time performance. In our proposed method, we have chosen the sequential algorithm *Basic Sequential Algorithmic Scheme* (BSAS)[29][30] due to its simplicity, efficiency, and easy implementation.

In BSAS, the number of clusters is not required to be known initially. During the clustering process, new clusters are created. Also each particle is presented to the algorithm only once during clustering.

Table 3.6: BSAS Algorithm [29]

```
Algorithm BSAS ($\chi(x_i...x_N),\theta$)
1: $m = 1, C_m = \{x_1\}$
2: for $i = 2$ to $N$ do
3:     find $C_K : d(x_i, C_k) = min_{1\leq j\leq m} d(x_i, C_j)$
4: if $d(x_i, C_k) > \theta$ then
5:     $m = m + 1, C_m = \{x_i\}$
6: else
7:     $C_k = C_k \bigcup \{x_i\}$, update
8:         its representative if necessary
9:     end if
10: end for
```

The BSAS algorithm is shown in Table 3.6, $\chi(x_i...x_N)$ is the input particle set to be clustered. For each particle, BSAS either assigns it to an existing cluster or a newly created cluster, depending on the distance from already formed clusters. The parameter θ is the threshold of dissimilarity, which determines how particles are clustered. Line 1 initializes the first cluster with the first point. Line 2 to line 10 loop through all the data left. Line 3 calculates dissimilarity measures between the current point and every existing clusters to find a minimum one. From line 5 to line 9, if the minimum measure is larger than θ, a new cluster will be created, otherwise the current point will be assigned to the existing cluster which has a minimum dissimilarity measure to it.

In the following part, detailed description of our second proposed method will be presented.

3.3.2 MCL+BSAS Part

The first part of this method is MCL+BSAS part. In this part, we employ the idea in [16]. This part is iterative and for each iteration, after MCL in line 2, we apply the clustering algorithm BSAS to the particle set in line 3 so that the BSAS algorithm can provide valuable information about the distribution of the particles.

As shown in Table 3.5, χ_t obtained in line 2 is the new particle set after one iteration of MCL. In line 3, C_t is the cluster set which we get after applying the BSAS algorithm to the whole particle set χ_t. Variable θ is used as the threshold in BSAS to decide whether a particles belongs to an existing cluster or be assigned to a newly created cluster. In line 4, after clustering we could find the cluster with the largest number of particles, and return the number of particles in this cluster as m. In line 5, the variable p is calculated, and p is defined as the percentage of m out of the total number of particles (N_{total}). p is used to evaluate the progress of localization by the MCL algorithm in line 2 and help us keep track of the convergence degree of particles.

When the value of p exceeds a predefined threshold η, the algorithm will assume the particles have concentrated to a certain degree such that the true robot position is more likely to be in this cluster which has the largest number of particles. With this newly obtained knowledge, we do not need to use as many as N_{total} particles for localization and we are ready to reduce the number of particles for the rest of the localization process. Then the algorithm will go to part two, the reducing part.

3.3.3 Reducing Part

In this part, we will reduce the number of particles and generate a new set of small number of particles based on the previous particles set. As shown in line 7 in Table 3.5, the inputs are the particle set χ obtained from the previous part and $n(n < 1)$ which indicates how many particles should be reduced. $\chi_t^{'}$ is the new particle set after reducing.

The reason that we try to reduce the number of particles is as follows: as the MCL goes on, the particles gradually converge to certain areas in the environment. These "certain areas" in our approach is the clusters obtained via the BSAS algorithm. In the cluster with the largest number of particles, the density of particles is very high. If this density exceeds a predefined threshold, then it is possible that we can proportionally reduce the number of particles in each cluster without jeopardizing the localization progress. For instance, if we only have three clusters, each having 300, 85, and 20 particles, respectively. If we proportionally reduce the number of particles in each cluster in half to 150, 42, and 10 particles, respectively, and if we continue the localization using the MCL algorithm, we may still succeed. So when the proposed algorithm finds the concentration of particles exceed a predefined threshold which means the algorithm no longer needs that number of particles, then we start to reduce the number of particles.

The method we use here for reducing the number of particles is to randomly pick a certain percent particles from the previous particles set in each cluster, and the percentage value is defined by the variable n. The reason of this is due to its simplicity in implementation as in mobile robot localization we mainly focus on the efficiency. After this part, the number of particles used in our algorithm is $n * N_{total}$, which is smaller than the original number, and hence this certainly reduces the computational cost.

3.3.4 MCL Part

The third part is the regular MCL part in line 8. It is shown in Table 3.7 and it uses the traditional MCL algorithm except the initial particle set used here, denoted as χ_t', is obtained

Table 3.7: MCL algorithm [10]

```
Algorithm MCL($\chi_{t-1}, u_t, z_t, m$)
1:  $\bar{\chi}_t = \chi_t = \phi$
2:  for $m = 1$ to $M$ do
3:      $x_t^{[m]}$ = sample_motion_model($u_t, x_{t-1}^{[m]}$)
4:      $w_t^{[m]}$ = measurement_model($z_t, x_t^{[m]}, m$)
5:      $\bar{\chi}_t = \bar{\chi}_t + < x_t^{[m]}, w_t^{[m]} >$
6:  endfor
7:  for $m = 1$ to $M$ do
8:      draw $i$ with probability $\alpha\ w_t^{[i]}$
9:      add $x_t^{[i]}$ to $\chi_t$
10: endfor
12: return $\chi_t$
```

from line 7. It is a much smaller subset of χ_t. The number of particles in χ_t' used in this part is much smaller than the particle set χ_t in line 7 when the algorithm exits the loop. The MCL algorithm in this part also takes the map *Map* of the environment as another input. The MCL goes on until the localization is finished.

3.3.5 Illustration of the Proposed Method Two

In this part, we illustrate how our proposed algorithm works in a simulated environment for the purpose of understanding the algorithm. More detailed experimental results will be presented in next chapter. Figure 3.3 shows the progress when implementing our second proposed algorithm in simulation. The small red circle denotes the particle, the big blue circle denotes the robot and the black line denotes the boundary of the environment.

Figure 3.3 (A) shows the initialization of particles and the particles populate the whole state space. Figure 3.3 (B) shows that the algorithm has found the number of particles in the largest cluster is larger than the predefined threshold θ, and the first part is finished. We can see from Figure 3.3 (B), there are two big concentrations of particles and this means the uncertainty of the robot's pose has been reduced a lot, compared with Figure 3.3 (A). Then in Figure 3.3 (C) it shows the algorithm has reduced the number of particles to one third based on the particle set in Figure 3.3 (B). Figure 3.3 (D) shows the localization is finished successfully using only one third of the original particles.

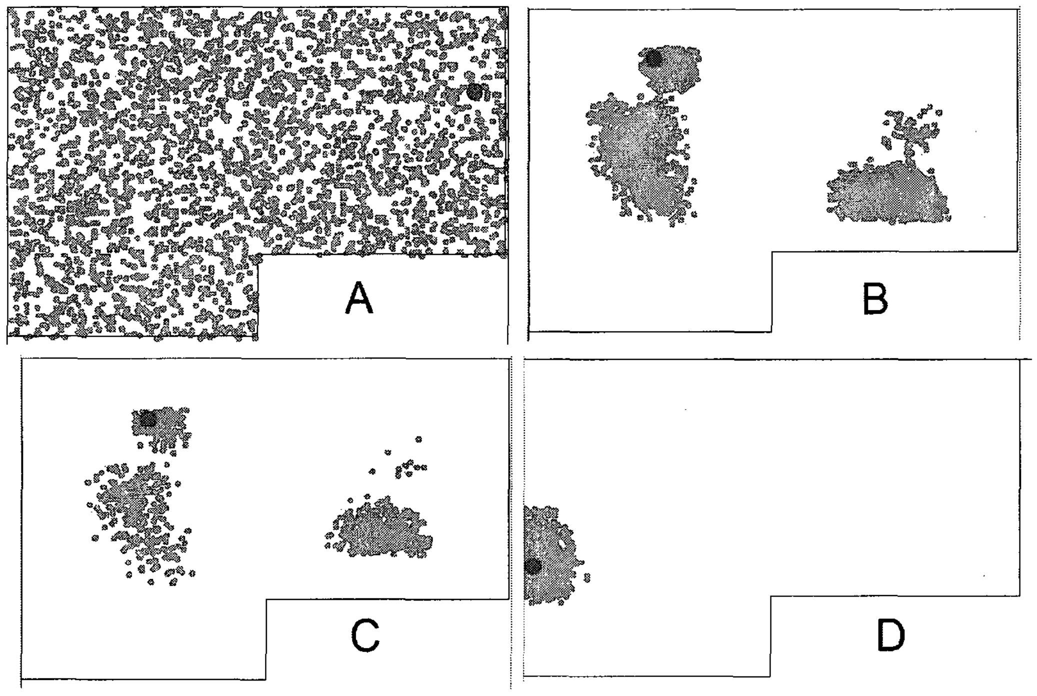

Figure 3.3: Illustration of the Proposed Method Two. (A) Initialization of particles, (B) Part 1 is finished, (C) Part 2 is finished, (D) Localization is finished.

Chapter 4

EXPERIMENT RESULTS

In this chapter, we will first present the implementation details of our experiments, including the hardware platform and software platform, then demonstrate the experimental results of the two proposed methods in both real and simulated environments.

4.1 Implementation Details

4.1.1 Hardware Platform

The hardware platform used to test our proposed methods is the LOGO MINDSTORMS NXT Robot, a programmable robotics kit released by LEGO GROUP in 2006[33][34]. It comes with the NXT-G programming software, and it supports many unofficial programming languages such NXC, NBC, LeJOS NXJ and RobotC[34].

Figure 4.1 shows the main components of the kit including the NXT intelligent brick, three motors, one touch sensor, one sound sensor, one ultrasonic sensor and one light sensor. The NXT kit also includes Lego Technic pieces such as gears, axles, and beams, which help

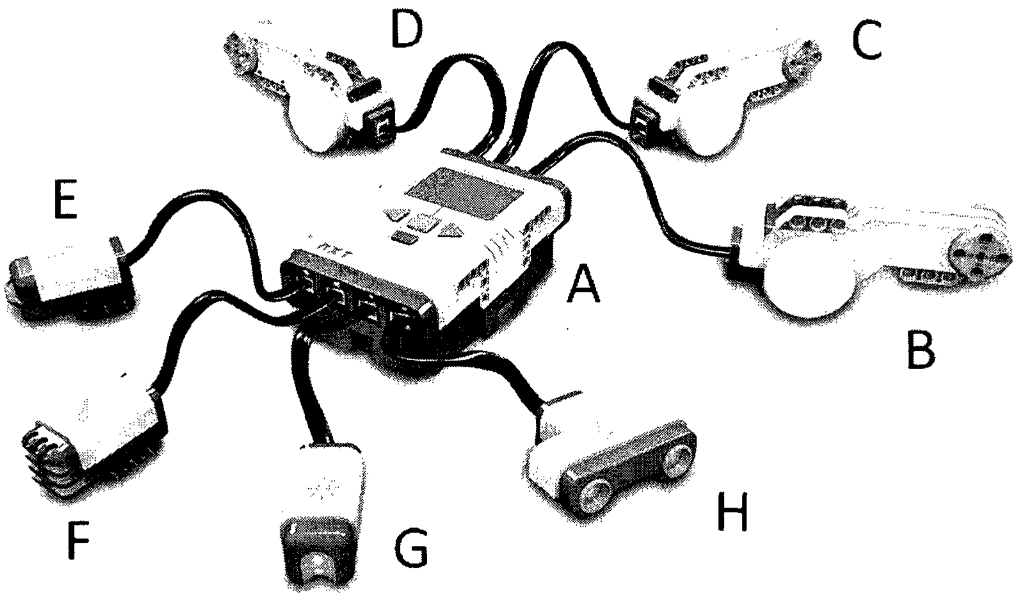

Figure 4.1: NXT main components. (A) Intelligent brick, (B)(C)(D) Motor, (E) Touch sensor, (F) Sound sensor, (G) Light sensor, (H) Ultrasonic sensor.[33]

to build the robot.[34] Figure 4.2 shows a robot built with the NXT kit.

The NXT brick is the most important part in this kit. It has three ports for connecting with the motor and four ports for connecting the sensor. The brick has a 100 x 64 pixel monochrome LCD display and four buttons that can be used to operate the menu. It also has a speaker which can play sound files at sampling rates up to 8 kHz. Power can be supplied by 6 AA (1.5 V each) batteries or by a Li-Ion rechargeable battery. NXT brick contains an Atmel 32-bit ARM processor running at 48 MHZ, and this processor has direct access to 64 KB of RAM.[32]

NXT supports both USB and bluetooth connection. The code and data can be upload to the NXT using these two methods, also the firmware can be upgraded by USB connection. The USB port can transmit data at 12 Mbits per second, and bluetooth transmits data at 460.8 Kbits per second. Bluetooth gives us a solution if the program is more than 256

KB, and allows the program access the memory of PC instead of relying on the built-in memory of NXT, so we can put the reflex actions on the NXT brick and the brains on the PC. Bluetooth also allows NXT interact with resources a computer interacts with, such as a webcam, database, network, printer and so on.[32]

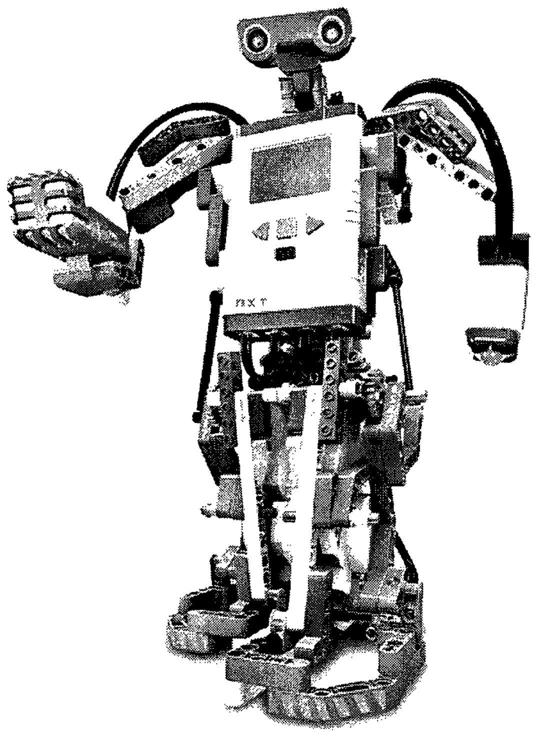

Figure 4.2: NXT Robot.[33]

The motor has built-in reduction gear assemblies with internal optical rotary encoders that sense their rotations within one degree of accuracy. The touch sensor could detect whether it is currently pressed, has been bumped, or released. The light sensor detects the light level in one direction, and also includes an LED for illuminating an object. The light sensor can sense reflected light values (using the built-in red LED), or ambient light. If calibrated, the sensor can also be used as a distance sensor. The sound sensor measures volume level on a scale of 0 to 100, 100 being very loud and 0 being completely silent.[32]

The ultrasonic sensor can measure the distance from the sensor to something that it is facing, and detect movement. It can show the distance in both centimeters and inches. The maximum distance it can measure is 255 cm with a precision of 3 centimeters. The ultrasonic sensor works by sending out ultrasonic sound waves that bounce off an object ahead of it and then back. It produces a sonar cone, which means it detects object in front of it within a con share. This cone opens at an angle of about 30 degrees.[32]

NXT also supports many third part sensors, which greatly increase the abilities to sense environmental conditions. HiTechnic is a company that make sensors for LEGO, such as compass sensor, tilt sensor and so on.[32]

4.1.2 Programming Platform

In this thesis the program is built using Java. In order to program with Java, we use two java packages, one is LeJOS NXJ and the other is iCommand[32][35].

LeJOS is a firmware replacement for NXT brick. It includes a Java virtual machine, which allows NXT be programmed in the Java programming language. As LeJOS is a firmware replacement, the new LeJOS NXJ firmware must be flashed onto the NXT brick , and replace the standard LEGO MINDSTORMS firmware(NXT-G). LeJOS includes a linker for linking user Java classes with classes.jar to form a binary file that can be uploaded and run on the NXT brick, and a PC API for writing PC programs that communicate with LeJOS NXJ programs using Java streams over Bluetooth or USB. The iCommand package is a sister-project of LeJOS NXJ. It mirrors LeJOS NXJ as closely as possible. The main difference is that the LeJOS NXJ runs on the NXT brick while iCommand runs on PC. The iCommand controls the NXT by sending individual commands through bluetooth connection, and gets the information from the NXT sensors. Using LeJOS and iCommand packages allow us not worry about the size of the code limited by the memory size of NXT brick, and access all the memory resource of PC.[35]

4.2 Experiment Design

The idea behind the experiment design is as follows. First we apply the traditional MCL to the environment, then we run our two proposed algorithms in the same environment, then we compare the performance of localization with traditional MCL.

In this part, we will first present the performance of traditional MCL in both real environment and simulated environment, then in the following parts, we will discuss the experiment of each proposed method.

4.2.1 Traditional MCL in Real Environment

Figure 4.3 shows the NXT used in the real experiment, and the environment is shown in Figure 4.4. The environment is asymmetric and the black line is the boundary of the region. The reason of testing our algorithm in this environment is due to its simplicity. But also the simpler the environment is, the more difficult the localization is. If there are too many unique landmarks which give a lot of information to robot, it will make localization much more easier. We use the light sensor of the NXT robot, which helps to detect the boundary of the region.

We first apply the traditional MCL in the environment to see how many particles are needed so that the localization successful rate is satisfactory. For each value of particle number, we run traditional MCL 20 times.

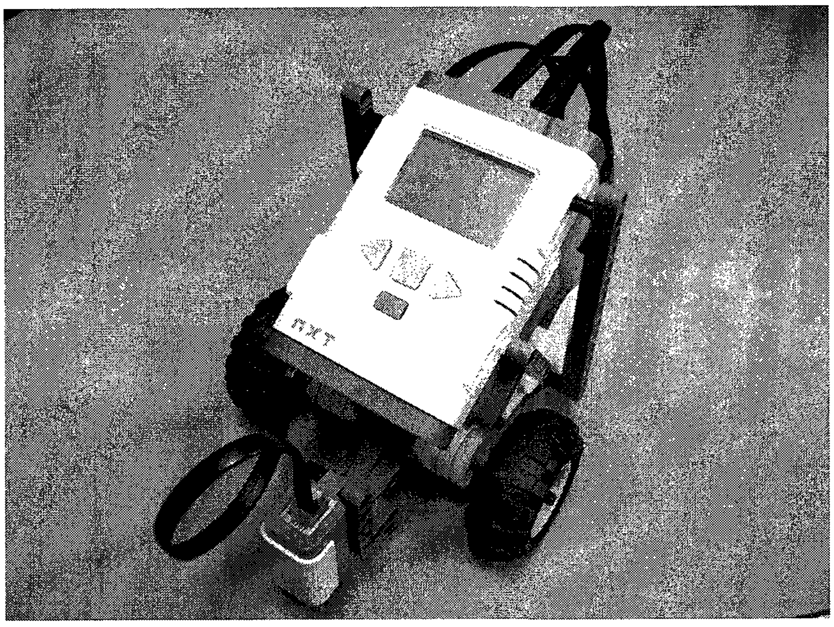

Figure 4.3: LEGO MINDSTORMS NXT in our experiment.

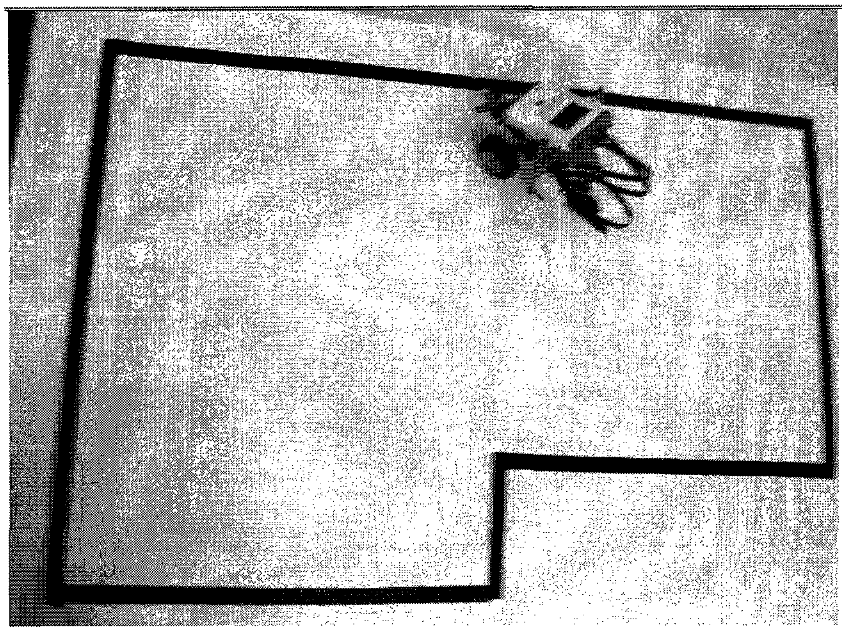

Figure 4.4: Environment for the real experiment.

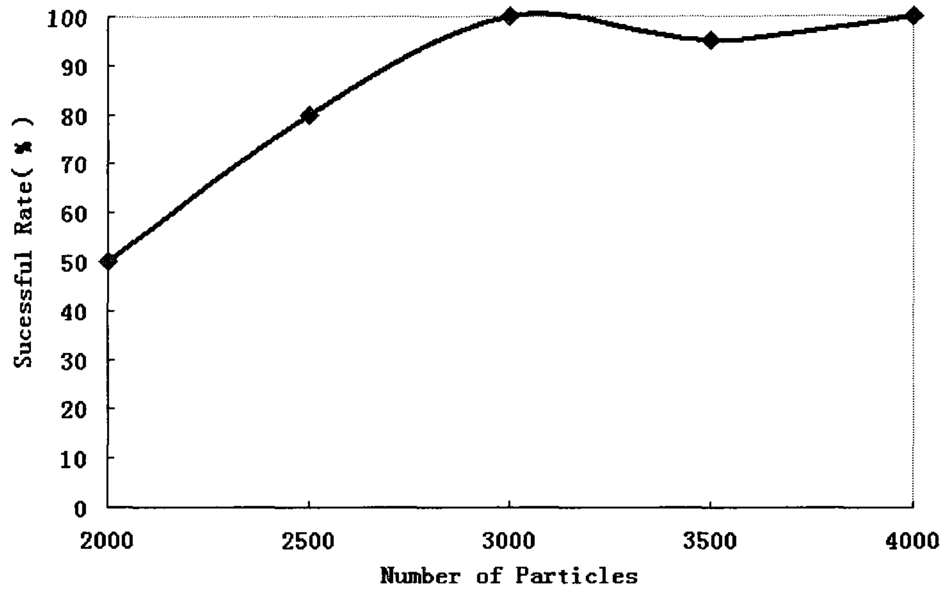

Figure 4.5: Performance of traditional MCL in the real environment.

As shown in Figure 4.5, the localization successful rate is a bit lower when the number of particle is below 3000. For 2000 particles, the successful rate is 50% and for 2500 particles, the successful rate is 80%. When the number of particle reaches 3000, the traditional MCL gets a successful rate of nearly 100%. For 3000 particles the successful rate is 100%, for 3500 particles the successful rate is 95%, and for 4000 particles, the rate is 100%.

4.2.2 Traditional MCL in Simulated Environment

Traditional MCL is also implemented on PC in an area which is proportionate to the one used in real environment. For each value of particle number, we also run the algorithm 20 times.

From Figure 4.6, we can see that the localization successful rate is also lower when the number of particle is below 3000. For 2000 particles, the successful rate is 40% and for

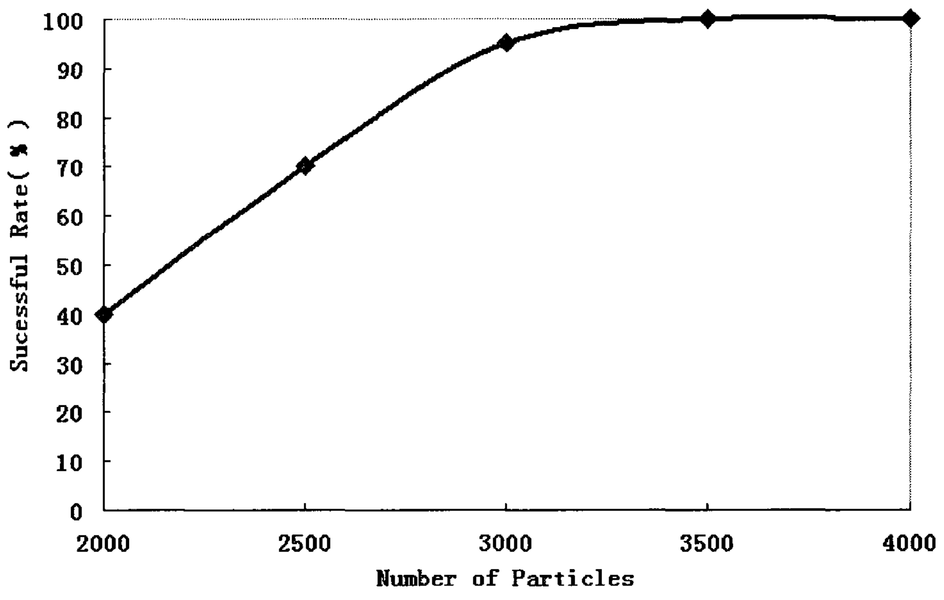

Figure 4.6: Performance of traditional MCL in the simulated environment.

2500 particles, the successful rate is 70%. For 3000 particles, the successful rate is 95%, and for 3500 and 4000 particles, both rates are 100%.

4.3 Experiment Result for Proposed Method One

4.3.1 Parameters Setting in Algorithm

In the proposed method one, we can see that the computational cost is determined by the number of grid cells and the number of particles. As the particles are only generated in the restricted area so this number is quite small. As discussed before, the number of orientations is a important factor that will affect the number of grid cells, so in our experiment we try different numbers of orientations, and see the performance of our proposed method.

4.3.2 Experiment in Real Environment

First the method is implemented in the same environment shown in Figure 4.4. The number of orientations of the grid cells is denoted as N. In the environment for one dimension there are 21 grid cells, so the total number of grid cells is $N * 21$. The number of particle used is 100, so the total number of grid cells and particles is $N * 21 + 100$, and it is denoted as N_{total}. For each N , we repeat the proposed methods 20 times.

Table 4.1: Proposed Method One Successful Rate (Real environment)

Number of Orientation (N)	12	24	36	48	60	72	96
N_{total}	352	604	856	1108	1360	1612	2116
Successful Rate(%)	25	40	55	75	85	85	90

As shown in the Table 4.1, the successful rate is quite low when N_{total} is below 1000, and when N_{total} is above 1000, the successful rates are apparently higher. In the Figure 4.5, we know that in the same environment the successful rate of traditional MCL is quite low when the number of particles is below 2500. After comparing the result of our proposed method one and the traditional MCL, we can find that our first proposed method can achieve higher successful rate of localization with lower computational cost.

4.3.3 Experiment in Simulated Environment

The proposed method one is also implemented on PC in the same simulated environment. Same as the last part, the number of orientation is denoted as N, and in the simulated environment for one dimension there are 21 grid cells so the total number of grid cells is $N * 21$. The number of particle used is also 100, so the total number of grid cells and particles is also $N * 21 + 100$, and it is denoted as N_{total}. For each N, we also repeat the proposed method 20 times.

Table 4.2: Proposed Method One Successful Rate (Simulated environment)

Number of Orientation (N)	12	24	36	48	60	72	96
N_{total}	352	604	856	1108	1360	1612	2116
Successful Rate(%)	20	30	50	75	80	80	90

As shown in the Table 4.2, the successful rate is also quite low when N_{total} is below 1000, and when N_{total} is above 1000, the successful rates become apparently higher. From Figure 4.6, we know that in the same simulated environment the successful rate of traditional MCL is quite low when the number of particles is below 2500. The experiment in the simulated environment also shows that our first proposed method can achieve higher successful rate of localization with lower computational cost.

4.4 Experiment Result for Proposed Method Two

4.4.1 Parameters Setting in Algorithm

It can be seen that in the proposed method two, there are three important parameters. They are the threshold θ used for clustering, threshold η used for terminating the first part, and the n used in the second part for determining how many particles should be reduced.

As mentioned in the previous section, θ determines the spreadness of the particles in a cluster, and if θ is too big the accuracy of localization will be greatly affected. η decides when to reduce the number of particles, if it is too big it will take more time to succeed, and then the reducing part will be delayed, which means the computational cost is not greatly reduced since most of the time we use the same number of particles as we started. n will also affect the accuracy of the localization because if we reduce too many particles that the remaining particles might not represent the uncertainty left in the localization process properly.

In our experiments, we set θ and η to fixed values which are appropriate for our experimental environment. The reason of doing this is that our main concern in MCL is the number of particles which is determined by the value of n. So we only use different values of n to test the proposed algorithm.

4.4.2 Experiment in Real Environment

From Figure 4.5, we know that the required number of particles for the robot to successfully localize itself in this environment is 3000.

After figuring out that 3000 particles are enough for this environment, we start to apply our proposed algorithm to see whether it performs well when the number of particles is reduced during the localization process.

For our second proposed algorithm, we set threshold θ which determines the spreadness of the cluster to be 9 cm, and set threshold η used for terminating the first part to be 25%, which means after the number of particles in biggest cluster reaches 25% of the total particles, it will start to reduce the number of particles in each cluster. Then for parameter n which determines how many particles are reduced, we choose three different values to see the localization successful rate. For each value we repeat the experiment 20 times.

Table 4.3: Proposed Method Two Successful Rate (Real environment $\theta = 9cm, \eta = 25\%, n = 1/2, 1/3, 1/4$)

Number of Particles	1/2	1/3	1/4
3000	100	95	55
3500	100	100	100
4000	100	100	100

For reducing part, we try three different n, i.e., 1/2, 1/3 and 1/4. As we can see from

Table 4.3, when we use 3000 particles, if we reduce the particles to half or one third, the successful rate is still high, however if we reduce to one fourth, the rate is much lower. The result is encouraging since reducing the number of particles to half of what is required by the traditional MCL still maintains a very high success rate. For 3500 or 4000 particles, reducing to half, one third or one fourth, all still produce very good results. The real environment experiments demonstrate that our proposed method two can produce very good results when we reduce the number of particles compared with using the traditional MCL.

4.4.3 Experiment in Simulated Environment

From Figure 4.6, we know that the required number of particles for the robot to successfully localize itself in this environment is also 3000.

Then we start to apply our second proposed algorithm to the simulated environment. We set threshold θ to be 60 pixels, and set threshold η used for terminating the first part to be 25%. Then for parameter n we choose the same three different values 1/2, 1/3 and 1/4. For each n, we repeat the experiment 20 times.

Table 4.4: Proposed Method Two Successful Rate (Simulated environment $\theta = 60pixel, \eta = 25\%, n = 1/2, 1/3, 1/4$)

Number of Particles	1/2	1/3	1/4
3000	100	95	60
3500	95	95	100
4000	100	100	100

For reducing part, we try three different n, 1/2, 1/3 and 1/4. As we can see from Table 4.4, when we use 3000 particles, if we reduce the particles to half or one third, the successful rate is still high, however if we reduce to one fourth, the rate becomes lower. For 3500 or 4000 particles, reducing to half, one third or one fourth, all still produce very good results.

The simulated experiments also show that our proposed method two performs very well when we reduce the number of particles significantly.

Chapter 5

CONCLUSION AND FUTURE WORKS

5.1 Conclusion

Mobile robot localization is a very important and fundamental problem in robotics. During the past decades, many algorithms for mobile robot localization have been proposed. Among these algorithms, Monte Carlo localization(MCL) is one of the most popular and efficient due to its better performance and less computational cost. However, MCL is still able to be further improved, so in this thesis we present two extensions of MCL both could improve the performance and reduce the computational cost of MCL. One is called moving grid cell based MCL algorithm which is a hybrid of grid localization and MCL, and the other is a dynamic MCL algorithm based on clustering.

Experiment results performed in both real and simulated environments demonstrate the effectiveness and low computational cost of each proposed method compared with traditional MCL.

5.2 Future Work

Our proposed methods can be further improved in the following aspects.

Active localization: The methods proposed in this thesis are passive, and the robot is controlled by a predefined movement pattern and the robot's navigation does not facilitate the localization progress. So one objective of this thesis is to incorporate active approaches to control the movement of the robot, which means actively selecting the most efficient motion direction and sensor direction.

Multi-robot localization: The methods proposed only deal with single robot localization problem. We want to apply our methods to multi-robot localization problem which is more difficult than single robot localization problem. In multi-robot localization, we not only have to consider the movement and measurement of the robot, also the detecting problem between different robots. The issues that arises usually include representation of beliefs and the communication between different robots.

Kidnapped robot problem: Our proposed methods focus on global localization. We also want to improve our methods to solve kidnapped robot problem, where the robot is kidnapped and taken to somewhere else without being notified. Kidnapped robot problem becomes important because localization approaches can fail sometimes. The ability to recover from failures is especially important for truly autonomous robots.

Bibliography

[1] D.Fox, W.Burgard, F.Dellaert and S.Thrun, Monte Carlo Localization:Efficient Position Estimation for Mobile Robots. Proc. AAAI-99, Orlando, USA.(1999).

[2] F.Dellaert, D.Fox, W.Burgard and S.Thrun. Monte Carlo Localization for Mobile Robots. IEEE International Conference on Robotics and Automation, ICRA.(1999)

[3] S.Thrun, D.Fox, W.burgard, and F.Dellaert. Robust monte carlo localization for mobile robots. Artificial Intelligence.(2001)

[4] S.Thrun, J.Schulte and C.Rosenberg. Interaction with mobile robots in public places. IEEE Intelligence Systems, Page(s):7-11.(2000)

[5] P.Jensfelt and S.Kristensen,Active global localization for a mobile robot using multiplehypothesis tracking. IEEE Trans.Robotics Automation.Vol.17,No.5,Page(s):748 - 760.(2001)

[6] N.Trawny and T.Barfoot. Optimized Motion Strategies for Cooperative Localization of Mobile Robots. Proceedings of the 2004 IEEE International Conference on Robotics and Automation.(2004)

[7] W.Burgard, D.Fox, D.Hennig, and T.Schmidt, Estimation the absolute position of a mobile robot using position probability grids. Proc. AAAI-96,Oregon, USA.(1996)

[8] D.Fox, W.Burgard, H.Kruppa and S.Thrun.A Probabilistic Approach to Collaborative Multi-Robot Localization. Autonomous Robots. Volume 8, Number 3, Page(s):325-344.(2000)

[9] H.Höse and H.L.Akın. The Reverse Monte Carlo localization algorithm. Robotics and Autonomous Systems ,55 , Pages(s):480-489.(2007)

[10] S.Thrun, W.Burgard and D.Fox. Probabilistic Robotics. The MIT Press.(2005)

[11] D.Rubin. Using the SIR algorithm to simulate posterior distributions. Bayesian Statistics 3. Oxford University Press.(1988)

[12] D.Fox. Adapting the Sample Size in Particle Filters Through KLD-Sampling. International Journal of Robotics Research.(2003)

[13] R.Luo and B.Hong. Coevolution Based Adaptive Monte Carlo Localizaton(CEAMCL). International Journal of Advanced Robotic Systems, Volume 1,Number 3, Page(s):183-190.(2004)

[14] S. Thrun, D. Fox, and W. Burgard. Monte carlo localization with mixture proposal distribution. Proceedings of the AAAI National Conference on Artificial Intelligence, Austin, TX.(2000)

[15] S. Theodoridis and K. Koutroumbas, Pattern Recognition. Academic Press.(2006)

[16] D.Wu, J.Chen and Y.Wang. Bring Consciousness to Mobile Robot Being Localized. Proceedings of the 2009 IEEE International Conference on Robotics and Biomimetics, Page(s):741-746.(2009)

[17] D.Fox, W.Burgard and S.Thrun. Active markov localization for mobile robots. Robotics and Autonomous Systems.(1998)

[18] Maja J Matarić. The robotics primer. The MIT Press.(2007)

[19] D.Fox, W.Burgard and S.Thrun. Markov Localization for mobile robots in dynamic environments. Journal of Artificial Intellgence Research, Page(s):391-427.(1999)

[20] S.I.Roumeliotis and G.A.Bekey. Distributed multi-robot localization. IEEE Trans. Robotics and Autumation, Page(s):781-795.(2000)

[21] R.Cheeseman and P.Smith. On the representation and estimation of spatial uncertainty. International Journal of Robotics, Page(s):56-58.(1986)

[22] M.Csorba. Simultaneous Localisation and Mapping Building. PhD.thesis. University of OXford.(1997)

[23] Z.Ghahramani. Learning Dynamic Bayesian Networks. Lecture Notes in Computer Science, Vol.1387, Page(s):168-197.(1997)

[24] L.P.Kaelbling, A.R.Cassandra and J.A.Kurien. Acting under uncertainty: Discrete bayesian models for mobile-robot navigation. Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems.(1996)

[25] S.Thrun. Robotic mapping: A survey. Exploring Artificial Intelligence in the New Millenium. Morgan Kaufmann.(2002)

[26] S.Thrun. Particle filters in robotics. Proceedings of the 17th Annual Conference on Unvertainty in AI(UAI).(2002)

[27] S.Thrun. Bayesian Landmark Learning for Mobile Robot Localization. Machine Learning. Springer.(1998)

[28] R.Xu and D.Wunsch. Clustering. John Wiley & Sons.(2009).

[29] B.Everitt, S.Landau and M.Leese. Cluster Analysis. Arnold Publisher.(2001)

[30] P.Trahanias and E.Scordalakis. An efficient sequential clustering method. Pattern Recognition, Vol.22(4), Page(s):449-453.(1989)

[31] P.Berkhin. Survey of clustering data mining techniques. Technical Report, Accrue Software Inc.(2002)

[32] B.Bagnall. Maximum Lego NXT:Building Robots with Java Brains. Variant Press.(2007)

[33] The LEGO Group.http://mindstorms.lego.com.

[34] Wikipedia website.http://en.wikipedia.org/wiki/Lego_Mindstorms_NXT.

[35] Lejos. http://lejos.sourceforge.net.

Vita Auctoris

NAME:	Yuefeng Wang
PLACE OF BIRTH:	Shandong, China
YEAR OF BIRTH:	1986
EDUCATION:	Shandong University, Shandong, China
	2004-2008 B.Eng.
	University of Windsor, Windsor, Ontario
	2008-2010 M.Sc.