2009

Bring Consciousness to Mobile Robot Being Localized

Jingxi Chen
University of Windsor

Bring Consciousness to Mobile Robot Being Localized

by

Jingxi Chen

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

Windsor, Ontario, Canada

2009

Bring Consciousness to Mobile Robot Being Localized

by

Jingxi Chen

APPROVED BY:

_____

Dr. Jonathan Wu
Department of Electrical & Computer Engineering

_____

Dr. Alioune Ngom
Department of Computer Science

_____

Dr. Dan Wu, Advisor
Department of Computer Science

_____

Dr. Jessica Chen, Chair of Defense
Department of Computer Science

22 May 2009

Author's Declaration of Originality

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's 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 and have included copies of such copyright clearances to my appendix.

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 one of the most important problems in robotics research. Localization is the process of a robot finding out its location given a map of its environment. Knowing its location is a necessary prerequisite for many other robotic tasks. A number of successful localization solutions have been proposed, among them, the well-known and popular Monte Carlo Localization (MCL) method. However, in all these methods, the robot itself does not carry a notion whether it has or has not been localized, and the success or failure of localization is judged by normally a human operator of the robot. In this paper, we put forth a novel method to bring consciousness to a mobile robot so that the robot can judge by itself whether it has been localized or not without any intervention from human operator. In addition, the robot is capable to notice the change between global localization and position tracking, hence, adjusting itself based on the status of localization. A mobile robot with consciousness being localized is obviously more autonomous and intelligent than one without.

Keywords: single-robot, localization, Monte Carlo, belief, cluster

Dedication

I dedicate this thesis to my parents. Without their patience, understanding, support, and most of all love, the completion of this work would not have been possible.

Acknowledgements

I would like to express my gratitude to all those who gave me the possibility to complete this thesis. I am deeply indebted to my supervisor, Dr. Dan Wu, who provided me with fundamental advices, stimulating suggestions and long lasting patience helping me in all the time of research and writing of this thesis. The completion of this work would not have been possible without his support, hard work and endless efforts.

I want to thank my external reader, Dr. Jonathan Wu, my internal reader, Dr. Alioune Ngom, and my thesis committee chair, Dr. Jessica Chen for spending their time on reviewing this thesis and for all their help, support, interest, and valuable hints.

My friends helped me solve many difficult problems during my research. I want to thank all of them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1
Introduction

In order to successfully navigate a mobile robot, the robot must know where it is and then decide where to move. Not only a pre-requisite to many navigation tasks, but also a fundamental problem to make a truly autonomous robot, the robot has to determine its location as accurately as possible. Thus, mobile robot localization has been acknowledged as one of the key issues in robotics [4]. Formally, the problem of mobile robot localization, also known as position estimation problem, is to estimate the pose of a robot given the map of the environment [26]. The pose of a mobile robot normally is described by a two-dimensional planar coordinates augmented with its angular orientation $(x, y, \theta)'$. There are two types of localization problems: local localization and global localization. Local localization [1, 2, 7], namely position tracking, calculates the current robot pose with the known initial position and heading direction. It is the most simple localization problem and only needs to compensate the error of dead-reckoning during movement. Local localization problem has been paid by far a plenty of attention in the literature due to the robot initial pose can be known as a prior. On the other hand, the global localization problem is a more challenging [6, 8, 9]. Most accurate and efficient approaches successfully employed by local localization cannot handle the global localization problem. Global localization needs to estimate the robot's pose without prior knowledge of its initial pose, but through sensors perceiving the outside physical world. These two types of localization problems are not absolutely isolated. Global localization and position tracking are also two different stages of localization which can be transformed from each other.

A lot of localization algorithms have been proposed to date. Typical examples include Kalman filter [11, 12, 13, 14], Grid localization [15, 16, 17], Monte Carlo localization [18, 19, 20] and some hybrid approaches [30, 39]. The Kalman filter

technique is commonly used in local localization. The robot estimates its pose continuously by counterbalancing the odometric error using the sensor data. Therefore, if the initial pose is accurate and sensor error is small, the Kalman filter can provide efficient, accurate, and continuous localization result. On the other hand, Grid localization is widely used for the global localization. A typical approach used by grid localization is to compute positional probabilities of all cells in the grid. Hence, Grid based localization requires a large amount of computation time, and the accuracy of localization depends on the cell size. Another popular global localization technique is Monte Carlo localization (MCL). It is less computational expensive than Grid localization because the probability computation is carried out only for the random samples, whose number if normally is much smaller than the number of cells in a gird. MCL often provides more accurate results than Grid localization because the samples can take any pose regardless of the cell size. However, the efficiency and resolution of MCL is lower than Kalman filter in local localization [26]. Some other hybrid methods take a combination of either Grid localization and a Kalman filter, or a Kalman filter and MCL. Utilizing merits of each method, these hybrid approaches improve the efficiency of localization [39].

1.1 Motivation

Among many localization techniques, MCL has become a popular and valuable tool in recent years. MCL takes a lot of obvious advantages than other localization techniques [59]. In contrast to Kalman filter-based techniques which only work well for unimodal distributions, MCL is able to represent multi-modal distributions and can globally localize a robot. MCL is more accurate than Grid localization with a fixed cell size. Moreover, it dramatically reduces amount of memory required compared to Grid localization and it can integrate measurements at a considerably higher frequency. Otherwise, easy to implement is also one of the bonus coming from MCL. The merits of MCL attract much attention in literature. Recently, many researchers studied how

to reduce the computational time of MCL, and how to increase the accuracy of pose estimation and deal with some inherent drawbacks of MCL, such as loss of diversity [20, 22, 38]. In many research papers presenting the experimental results of the MCL algorithms, the results are likely revealed in pictures in which the particles concentrate around the robot positions to show the algorithm succeeded [26, 36, 39, 40]. Using pictures to show the results takes certain advantage. According to the particle filter, the pose of one random particle does not make any sense. Only the particle set composed by a large number of particles approximates the correct posterior. It is hard to know the localization result from a data array that lists all the location and orientation of each particle. But showing every particle in a picture gives us an instant sense about the distribution of particle set. Through this way, human beings can quickly understand the localization is not successful when the particles are spread through the environment and that particles concentrate successfully means the robot is certain at a position. However, this information cannot be directly comprehended by the robot. This line of reasoning put forth an interesting question: How does the robot know whether it has been successfully localized? Within the framework of MCL, this amounts to ask if a robot knows whether the particles are concentrated. Imagine a robot equipped with a color camera is operating in a square room with four land marks at the corner, initialized with uniform distribution of particles. At the beginning, the mean of particles is placed at the center of the room, which is far from the robot true position. After the first marker detection, nearly all particles are drawn toward closing to the robot, so is the mean of particles. After several more detections, the particles are clustered around the true robot position. At this stage of localization, the robot can merely track its position indicated by the mean of particles. The key point of successful localization in this scenario is the robot can know the particles become clustered around the true robot position, and then start position tracking. Moreover, if we simply define a few different outcomes of running MCL algorithm, such as the robot is globally uncertain, the belief of robot is concentrated around several possible locations or the robot's belief is centered on correct pose, can a robot distinguish from

these different localization outcomes? If the robot can know the stages changing from the global localization to local localization, the robot is able to choose a localization technique which is more accurate and efficient but may can only be used in position tracking, such as Kalman filter techniques, instead of the previous one, such as MCL, used in global localization. Therefore, according to the stages of localization, the robot is able to adjust itself getting efficiency.

In recent years, most papers related to MCL focus on improving the accuracy and efficiency of the algorithm or extending MCL to different robot platforms, such as multi-robot localization. In the new probabilistic approach to collaborative multi-robot localization proposed by Fox and Thrun [60], after the two robots detected each other, the robot within highly uncertain can obtain location information from the internal beliefs of the other robot which is confident about its position. One measure of performance in their experiment is the average time that the robot takes to find out where it is. To determine the stop condition that robot has successful localized itself, it is assumed the termination is achieved if the localization error falls below 1.5 meters. The error is measured by averaging over the distance of all particles from a reference position. The reference positions are points at the robot's trajectory estimated by measuring the starting position of each run and performing position tracking off-line using MCL. However, computing the estimation error at the reference positions to tell the robot has successfully localized itself takes the assumption that the true locations of the robot are known during localization. In fact, the ground truth for these reference points is not available in real time. In their method, it is simulated by a particle filter with very large number of particles (far more than actually needed) performing position tracking. Another method improving the efficiency of MCL is adapting the sample size in particle filters through KLD-sampling proposed by Fox [32]. The efficiency of particle filters is increased by a statistical approach adapting the size of sample sets during the estimation process. The approximation error introduced by the sample-based representation of the particle filter is bound by the Kullback-Leibler

distance. The adaptation approach chooses a small number of particles if the density represented by particles is centered at a small part of the navigation space, and it chooses a large number of particles if the localization uncertainty is high. When the particles converge to the robot's current pose, the number of particles is reduced. At first glance, they adaptively changed the number of particles when the localization stage changed from global localization to position tracking. But the adaptive number of particles cannot offer exact information to explicitly distinguish different stages of localization and it is hard to say how many particles means the robot successfully localized itself. They focused on adjusting the number of particles in order to enhance the efficiency of particle filter. The distinction of different stages of localization and the terminating condition of particle filter are not given. In several other existing works [37, 38, 41], giving mobile robot consciousness about being localized in real time has not been paid too much attention as well. The problem of when the robot is considered by itself as successfully localized is not fully explored. For example, when the Monte Carlo Localization is applied on the robot mini-rover with low-cost IR sensors [40], correct localization is assumed when all the particles are contained inside the area covered by the robot. This approach can be only used in simulation since the true position of robot is also assumed to be known in real time. But knowing ground truth in global localization is impossible for a real robot.

1.2 Contributions

This thesis is only concerned on the problem of Monte Carlo localization in indoor environments, particularly in small-scale room with robot equipped with low-cost sensors. In this thesis, we propose a novel approach to notice the robot when the position of robot is successfully determined in global localization and help the robot distinguish from different stages during localization time. Our framework is based on Monte Carlo Localization, which maintains a set of samples to represent uncertainty, yet can not explicitly offer numeric probability density values itself. In

order to obtain the distributed information of particles from a macro view, particles are not only treated individually as a single point in our method, but also the concentrated particles are analyzed as a whole entity. Based on checking the relative location of particles, the robot can know whether it is in a localization stage that most particles are located around one point, or that the robot is still global uncertain. It offers a chance the robot can make a choice keeping on global localizing itself or starting position tracking and navigating to the target, or even adjusting itself to get better localization result. By analyzing how concentrated particles are, the robot can know the process of localization instead of that a human being stares at the screen to see whether the particles get together. In this way, the robot is more intelligent since it takes the work of human. Otherwise, our method provides an approach to express localization outcomes in a numerical way. The result of localization can be showed in pictures along with description of numeric values to explain or compare to other localization result. In addition, what we do is trying to bring consciousness to mobile robots. For human beings, a lot of everyday activity is automatically controlled, that is the detailed control of joints is unconscious. When we are walking under ordinary conditions, we don't notice the control of lifting or dropping down feet. But when the environment changed or under some conditions, such as dizzy, our control of our actions becomes very conscious and deliberate. If a hollow confronted in front of us on the road while we are walking, our brain will start to intervene the action to avoid falling into the hollow [46]. The same situation can be viewed in mobile robots. In regular, Monte Carlo localization automatically helps the robot localize itself. But when the effect of localization reaches to a predefined level or sensors fail to return correct data, our approach bring consciousness to the robot that can know the change. For example, in position tracking, the mean of particles can tell the robot where it is. But if the sensors are broken when the robot is navigating, the false sensor readings may result particles move dispersedly. In this case, the location information obtained from the mean of particle set becomes incorrect. To avoid the false believing of location, through using our method the robot will be aware about the exception

occurred.

1.3 Outline

This remainder of the thesis is organized as follows.

Chapter 2: Background knowledge. This chapter is focused on the materials that the proposed approach is based on. First, we will introduce the major stumbling block uncertainty in robotics and provide a comprehensive overview of probabilistic robotics. And then the description of basic probabilistic concepts, formal model of robot environment interaction and the recursive algorithm for state estimation Bayes filter will be given. As the Monte Carlo localization is the fundament of the proposed method, it is specifically emphasized later.

Chapter 3: The combined MCL-Clustering algorithm. The proposed method, combined MCL-Clustering, is presented in detail in this chapter. First are the statement of the problem and the general description of our method, and then followed by which clustering algorithm is chose and how to combine it with MCL.

Chapter 4: Implementation and experiment results. Two types of experiments in different environment are designed to verify the performance of the proposed approach. One is tracking without perception and the other is global localization. The detailed information of the implementation and the experimental results will be resolved. These experimental results will confirm how successful our proposed method is.

Chapter 5: Conclusion and future work. The conclusion of the thesis is brought in this final chapter, and a frame of future work is presented.

Chapter 2
Background Knowledge

This chapter provides the background knowledge which the proposed method is based on. First, we will articulate the basic idea of probabilistic robotics, followed by the problem of mobile robot localization. Then, Monte Carlo localization (MCL) algorithm is explained since it is one of the most important probabilistic algorithms for mobile robot localization and also the foundation of the proposed method.

2.1 Uncertainty in Robotics

Robotics is the science of developing techniques for robot perceiving information on environment through sensors and manipulate through physical devices [26]. Robotics systems have become an increasingly important part of human society. Commonly seen robotic systems include mobile robots for the Mar exploration, industrial robotic arms in assembly factory and robots used for search and rescue [47]. These systems have successfully provided a huge number of labor-saving devices and have at times released humans from doing boring and dangerous jobs such as painting cars or checking suspicious packages in public place. If the robot can be as intelligent as humans, the impact would be dramatically enormous. Imagine all the cars that are safely travelling by themselves on the road decreasing the numbers of traffic accidents, automatic mobile robot groups searching lost people in desert or checking radioactive materials under ocean, or service robots in hospital assisting patients. To accomplish these real world robotic applications, robots have been challenged to be capable to handle a variety of uncertainties which exist in physical world.

Several factors contribute to the robot's uncertainty [24]. Robot environments are inherently unpredictable, such as office and private home which are highly dynamic

and particularly the uncertainty is high for proximity of people. And sensors of robot are short of what they can perceive. Physical mechanism limits the resolution and range of a sensor. Robot actuation which involves motors is also unpredictable. Effects like control failure and mechanical noise bring uncertainty. Uncertainty may also come from the software controlling the robot. Models used in software are abstractions of the outside world. All models of the outside world are approximate. The environment can only be partially modeled. Uncertainty is further caused by approximations of robotic algorithms. Accuracy and response time are always the two sides that resist each other. Many popular robotic algorithms sacrificed accuracy in order to achieve speedy response. Researchers have developed a series of paradigms for robot design, but these frameworks are not robust enough when the robot faces sensor and model limitations [26]. As robots are entering into human life more closely, uncertainty in robots has become a major issue for designing capable real world robot systems.

2.2 Probabilistic Robotics

The probabilistic robotics is a relatively new approach to robotics which addresses the problem of uncertainty in robot perception and action. The core idea in probabilistic robotics is to use calculus of probability theory to represent uncertainty explicitly [24]. Unlike the previous approaches relying on a single best guess of what might be the case, probabilistic algorithms describe the robot and the environment using random variable. In particular, there are two basic models involved in probabilistic robotics: perception, the way sensor is processed, and action, the way robot behaviors. By doing so, probabilistic robotics provides a great way to accommodate the uncertainty that comes from most robot practice. As a result, they perform excellently in the face of uncertainty.

Programming robots probabilistically has a lot of benefits and has already reached

a great success in the field of robotics [24]. Using techniques in probabilistic robotics, a robot does better and is more robust than the one that does not carry a notion of its own uncertainty. Largely ignoring the problem of uncertainty and assuming a full and accurate model of the robot and the environment can be given does not work appropriately. In fact, certain probabilistic approaches are nowadays the only known working solutions to difficult robot estimation problems, such as kidnapped robot problem. Additionally, probabilistic algorithms do not have strong requirements on the accurate models of robot and environment than many classical planning algorithms. And finally, probabilistic algorithms are broadly applicable to nearly every robot problem in practice. For instance, the driverless car Stanley, a successful demonstration of probabilistic robotics, build by Stanford Racing Team competed in and won the 2005 Defense Advanced Research Project Agency (DARPA) Grand Challenge, which requires each team create a fully autonomous navigated vehicle able to pass through the road in a given map of desert with limited time [28].

2.2.1 State

In probabilistic robotics, we describe the robot and environments using the notion of state, which can be defined as a collection of all aspects of the robot and its environment that can impact the future [26]. The state variables that tend to change over time will be called dynamic state, such as walking people around the robot. And the state that is not changing will be called static state, such as location of walls in buildings. The state also involves variables related to robot itself, such as its pose, velocity and so on. In this thesis, state is denoted x; the state at time t is denoted x_t. Time defined here is discrete. That is, all states can be described at discrete time steps $t = 0, 1, 2\ldots$. The initial state of the robot will be denoted as time $t = 0$. For robot action, the state includes variables for the configuration of the robot's actuators. They might be the joint angles of revolute joints. The configuration of a robot is often referred to as kinematic state. And the robot velocity and velocities of its joints are

commonly referred to as dynamic state. The location and features of surrounding objects in the environment are also state variables. An object may be a chair, a box or a wall. Features of these objects may be their texture or color. The location of objects in the environment is static in this thesis. Many other variables that may impact a robot's operation can be state variables as well. The list of all possible state variables is endless. Typical state variable used in this thesis is the robot pose that includes robot's location and orientation relative to a global coordinate system. Strictly speaking, mobile robots have six such state variables, three for Cartesian coordinates, and three for angular orientation. But for robots defined in planar environments, the pose is usually given by three variables, two location coordinates in the plane and the heading direction.

A state is called complete if the current state is the best predictor of the future [26]. It means the knowledge of past states, measurements, or controls do not impact on the prediction of the future. Temporal processes which meet the conditions are commonly known as Markov chains. The environment that conforms to Markov chains assumes that past and future data are independent if the current state x_t is known. The notion of state completeness is mostly of theoretical importance. In practice, it is impossible to specify a complete state for any realistic robot system. However, complete state representations are often preferable to reduce the computational complexity. And in practice, this reduction has been found to be surprisingly robust enough.

2.2.2 Robot Environment Interaction

The environment, or world, is a dynamical system that has its own state. The robot can obtain information about its environment using its sensors. The robot can also influence its environment through its manipulating [50]. These are the two basic types of interactions between a robot and its environment, as shown in Figure 2.1. The

first one is called environment sensor measurements, by which the robot uses its sensors to acquire information about the state of its environment. For example, a robot might take a sonar scan or a laser scan to receive information about the location of people or desks. The result of this perceptual interaction is called a measurement, also called observation or perception. The second one is the control actions which actively assert forces on the robot's environment. Examples of control actions are robot motion and the manipulation of objects. Hypothetically, the record of all past sensor measurements and control actions kept by a robot is often referred as the data set. Consistent with the two types of environment interactions, there are two different data streams in a robot.

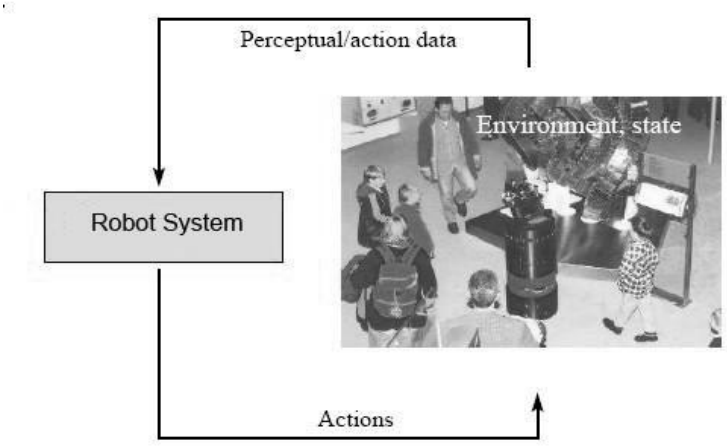

Figure 2.1: Robot interactions with environment. [26]

The first stream is environment measurement data which provides information about a temporary state of the environment. The measurement data at time *t* is denoted z_t. Usually, the robot is simply assumed taking exactly one measurement at a time. The notation $z_{t1}:z_{t2} = z_{t1}, z_{t1+1}, z_{t1+2}, \ldots, z_{t2}$ denotes the set of all measurements acquired from time *t1* to time *t2*, where $t1 < t2$. The second one is control data which provides information about the change of state in the environment. In mobile robotics, typical examples of control data are the velocity and odometers of a robot. Odometers are sensors that measure the revolution of a robot's wheels. Although odometers are sensors, they are usually treated as control data, as they measure the effect of a control

action. Control data is denoted u_t. And we will denote the collection of control data by $u_{t1:t2}$, for $t1 < t2$: $u_{t1:t2} = u_{t1}, u_{t1+1}, u_{t1+2}, \ldots, u_{t2}$. Both measurement data and control data play fundamental roles in a robotic system. They take place concurrently. Environment perception tends to increase the robot's knowledge, as it provides information about the environment's state. On the other hand, control tends to result a loss of knowledge due to the inherent noise in robot action.

2.2.3 Belief

In probabilistic robotics, the notion of belief is used to reflect the robot's internal knowledge about the state of the environment [26]. The state cannot be measured directly. Instead, it must be inferred from the measurement data and control data. Probabilistic robotics represents beliefs through conditional probability distributions. Belief is the posterior probability distribution over state variables conditioned on the perception data and control data. We denote belief over the state variable x_t by $bel(x_t)$, which is defined as $bel(x_t) = p(x_t|z_{1:t}, u_{1:t})$. This posterior is the probability distribution over the state x_t at time t, conditioned on all past measurements $z_{1:t}$ and controls $u_{1:t}$. In the literature, synonyms for belief are state of knowledge and information state.

2.2.4 Probabilistic Generative Laws

The calculation of state is governed by probabilistic laws in probabilistic robotics. The state x_t is generated from the state x_{t-1} by probability distribution $p(x_t|x_{0:t-1}, z_{1:t-1}, u_{1:t})$. The evolution of state x_t is conditioned on all past states, measurements, and controls. According to the assumed Markov chains we mentioned above, if the state x is complete then it is a sufficient summary of all that happened in previous time steps. In particular, x_{t-1} is a sufficient statistic of all previous controls and measurements up to this time, that is, $u_{1:t-1}$ and $z_{1:t-1}$. Namely, in all the variables in the expression above

only the control u_t matters if we know the state x_{t-1}. Thus, we can obtain the following equation: $p(x_t|x_{0:t-1}, z_{1:t-1}, u_{1:t}) = p(x_t|x_{t-1}, u_t)$. The property we used here is called conditional independence. It states that certain variables are independent of others if one knows the values of another group of variables, the conditioning variables. The same in calculating the probability of measurements, if x_t is complete, we have an important conditional independence: $p(z_t|x_{0:t}, z_{1:t-1}, u_{1:t}) = p(z_t|x_t)$. In other words, the state x_t is the only variable needed to predict the measurement z_t. Knowledge of any other variable, such as controls, measurements, or even past states, does not matter. The two probabilities here are the core models for state estimation. The probability $p(x_t|x_{t-1}, u_t)$ is the state transition probability or called motion model. It specifies how environment state evolves over time as a function of robot controls u_t and previous state x_{t-1}. The probability $p(z_t|x_t)$ is the measurement probability, also called measurement model. The measurement probability specifies the probabilistic law by which measurements z_t are generated from the environment state x_t. Figure 2.2 shows the evolution of states, measurements and controls, defined through these probabilities. The state x_t at time t is stochastically dependent on the state x_{t-1} at time $t-1$ and the control u_t. The measurement z_t depends on the state x_t at time t. The state transition probability $p(x_t|u_t, x_{t-1})$ and the measurement probability $p(z_t|x_t)$ together describe the dynamic system of the robot and its environment.

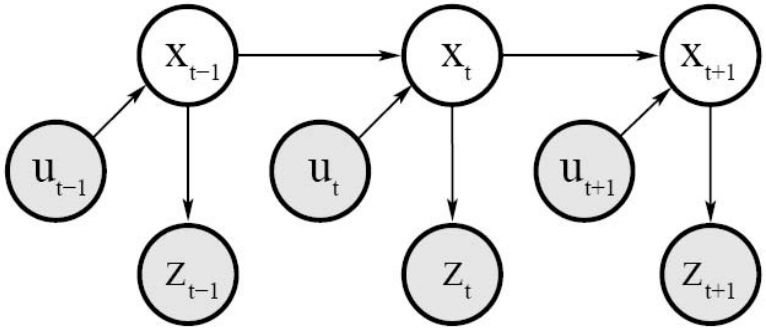

Figure 2.2: The evolution of states, measurements and controls. [26]

2.2.5 The Bayes Filter Algorithm

The most general algorithm for state estimation is Bayes filter [33, 50]. It

calculates the posterior probability $bel(x_t)$ according to the measurement and control data as follows:

$$bel(x_t) = p(x_t|z_{0:t}, u_{0:t}) \tag{2.1}$$

where x_t denotes the robot pose (x, y, θ) at time t, $z_{0:t} = \{z_0, z_1, \ldots, z_t\}$ denotes the sensor readings up to t, and $u_{0:t} = \{u_0, u_1, \ldots, u_t\}$ is the control data changing the state of the world. The input of Bayes filter is the belief $bel(x_{t-1})$ at time $t-1$, along with the most recent control u_t, and the most recent measurements z_t. The output is the belief $bel(x_t)$ at time t. The measurements of sensors and the control information are corrupted with noise. In order to deal with these uncertainties, Bayes filter is conducted in two phases: prediction phase and update phase. Each phase corresponds to a probability model to deal with the errors: motion model (action model) and measurement model (sensor model). In prediction phase, it processes the control u_t by calculating a predicted belief $\overline{bel}(x_t)$ over the state x_t based on the prior belief at state x_{t-1} and the control u_t (Equation 2.2). $\overline{bel}(x_t)$ is the prediction of the current state before incorporating the measurement at time t. In measurement update phase, the probability that the measurement z_t may have been observed is multiplied by the predicted belief $\overline{bel}(x_t)$ for each hypothetical posterior state x_t (Equation 2.3). The prediction and update phase can be represented by the following equations respectively:

$$\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1} \tag{2.2}$$

$$bel(x_t) = \eta p(z_t|x_t) \overline{bel}(x_t) \tag{2.3}$$

where η is the normalizing constant, $p(x_t|u_t, x_{t-1})$ is the motion model, and $p(z_t|x_t)$ is the measurement model. The two expressions only describe a single iteration of the Bayes filter algorithm to explain the update rule. This update rule is applied recursively, calculating the belief $bel(x_t)$ from the belief $bel(x_{t-1})$ calculated at the

former step. To calculate the posterior belief recursively, the algorithm needs an initial belief $bel(x_0)$ at time $t = 0$. In position tracking, $bel(x_0)$ is initialized with a point mass distribution which centers all probability mass on the correct value of x_0, and assigns zero probability anywhere else. In global localization, $bel(x_0)$ may be initialized using a uniform distribution over the domain of x_0. The general algorithm for Bayes filtering is depicted in Table 2.1.

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

Table 2.1: The Bayes Filter. [26]

There exists quite a variety of algorithms that are derived from the Bayes filter. Bayes filters are implemented in several different ways. Each one is distinct by different assumptions of the measurement probability, the state transition probability and the initial belief. No unique best approximation for computing beliefs can be applied for all robotics problems. Finding a suitable approximation is usually a challenging problem. When choosing an approximation, one has to trade off the computational efficiency, accuracy of the approximation and ease of implementation.

2.3 Mobile Robot Localization

Bayes filter offers a valuable tool for state estimation. One of the instances is the mobile robot localization. Mobile robot localization is the problem of determining the pose of a robot in a given map of the environment [57]. It is also called position estimation, which is a basic perceptual problem in robotics. Obviously, most robotics tasks require knowledge of the location of objects which are being manipulated

relative to the pose of the robot. Localization is also the process of establishing correspondence between the coordinate system of given map and the robot's local coordinate system. Knowing the pose of the robot determines this coordinate transformation. Thus, the robot can express the location of interested objects within its own coordinate frame. It is a necessary pre-requisite for robot navigation. Figure 2.3 depicts a graphical model for the single mobile robot localization problem. The difference between this figure and the Figure 2.2 we mentioned above in state evolution is the given map. The robot is given a map of its environment and its job is to find its position relative to the map based on the observation data and control data. The shaded notes indicate the values are known: the map *m*, the measurements *z*, and the control *u*. And the white notes mean the robot pose *x* should be inferred.

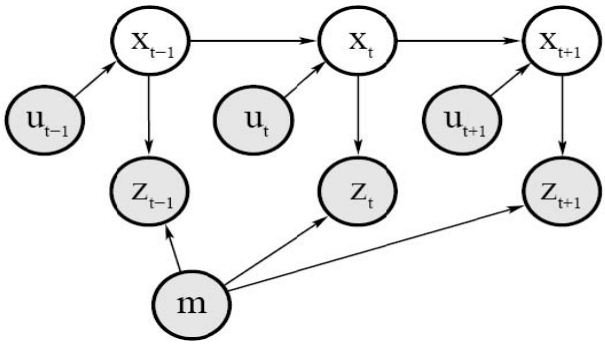

Figure 2.3: Graphical model of mobile robot localization. [26]

2.3.1 Categories of Localization Problems

Different mobile robot localization problem comes with different localizing difficulty. Not every localization problem is equally hard. Roughly, the localization problems can be divided into four classes, according to the property of the environment and the initial knowledge that a robot may have relative to the given map [26].

The first class is characterized by whether the knowledge of pose is available at the initial time to a robot. Here are two types of localization problem: local

localization (position tracking) and global localization. In local localization, it assumes the initial robot pose is known. The localization problem becomes to accommodate the noise in robot's moving. It is so called because the uncertainty is local and confined to the region near the robot's true position. Methods for local localization often assume the pose error is small. The pose uncertainty is often approximated by a unimodal distribution, such as Gaussian. In global localization, the initial pose of the robot is not given. The robot should determine its pose through scratch. It cannot assume boundary of the pose error. Global localization is more difficult than local localization.

The second class is characterized by the environment. Environments can be static or dynamic. In static environment, only the robot moves. That is, the only state cared about is the pose of the robot. All other objects in the environments stay at the same location forever. In dynamic environments, other than the robot, the location or configuration of objects changes over time, such as people, daylight, movable furniture, or dogs. Clearly, most real environments are dynamic, with state changes occurring in a different flavor. Localization in dynamic environments is more difficult than localization in static ones.

The third class that characterizes different localization problems is based on the fact whether or not the localization algorithm controls the motion of the robot: passive localization and active localization. In passive localization, the localization module only works as an observer on the robot. The control of the robot does not include facilitating localization. The robot might move randomly or do its own jobs. Active localization algorithms control the robot aimed to minimize the localization error or the costs that risk a poorly localized robot moving into dangerous place. Active approaches to localization problem usually have much better localization results than passive ones, such as coastal navigation.

The last class of the localization problem is characterized by the number of robots included: single-robot localization and multi-robot localization. Single-robot localization is the most commonly studied one. Dealing with a single robot only, all data only needs to be collected on a single robot platform, and no communication issue comes in this problem. The multi-robot localization problem is brought by a group of robots. The issue of multi-robot localization arises from the representation of beliefs and the nature of the communication between them.

The most important characteristics of the mobile robot localization problem are covered in these four categories. In this thesis, we deal with local and global localization of passive, single robot in static environments.

2.3.2 Map Representation

Mobile robot localization problem assumes that the robot was given a map in advance. In quite a few real world applications, maps are often available as a priori or can be constructed by hand. The map specifies the environment in which measurements are generated. Formally, a map m is a list of objects in the environment with their properties [26]:

$$M = \{m_1, m_2, \ldots, m_N\}, \tag{2.4}$$

where N is the total number of objects in the environment, and each m_n with $1 <= n <= N$ specifies a property. Maps are usually divided into two types, known as feature-based and location-based [26]. In feature-based maps, n is a feature index. The value of m_n specifies the location of the feature in the map coordinate system. In location-based maps, the index n contains a specific location. Location-based maps contain information for any location, no matter whether or not there is an object there. This is quite different in feature-based maps. Feature-based maps only specify the locations of the objects contained in the map. The two kind maps are showed in Figure

2.4. The left one is a location-based map. The black area indicates the location is occupied and the white area is free space. The right one is a feature-based map including line features, point features and arc features.

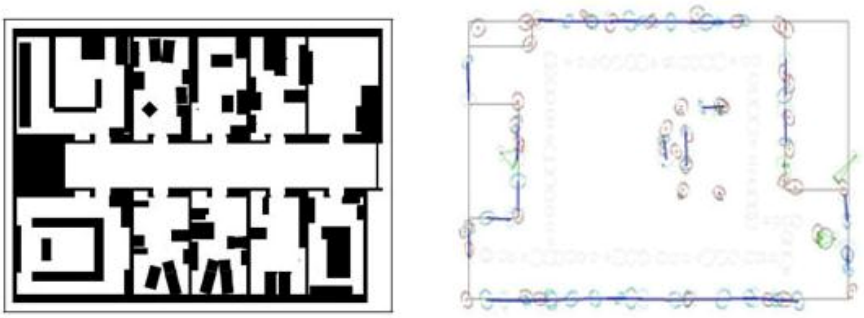

Figure 2.4: Example maps used for robot localization. (a) Location based map and (b) feature based map. [59, 77]

2.3.3 Related Works

To address the problem of mobile robot localization, researchers have developed a lot of techniques in the field of mobile robot. In probabilistic robotics, localization algorithms are variant implementations of the Bayes filter. The major algorithms are Kalman filter-based localization, Multi-hypothesis tracking filter, Grid localization and Monte Carlo Localization [26]. We will introduce the first three in this section, and Monte Carlo Localization will be focused at the next section.

The basic idea of Kalman filter is the beliefs of state are represented by multivariate Gaussian distributions. Namely, at time t, the probability over the state x is characterized by the mean u_t and the covariance Σ. Kalman filter has a crucial assumption that the state transition probability $p(x_t|u_t, x_{t-1})$, the measurement probability $p(z_t|x_t)$ must be linear functions in the arguments with Gaussian noise added and the initial belief $bel(x_0)$ must be normally distributed. To relax these linearity assumptions, extended Kalman filter localization and unscented Kalman filter localization were introduced [62]. In both of them, uncertainty in tracking is

represented by unimodal Gaussian. EKF uses Taylor series expansion to linearize the transformation of a Gaussian, but UKF uses a weighted statistical linear regression process. They are efficient and robust for position tracking problems, but not applicable to global localization problems. To overcome the difficulties of Kalman filter, Multi-hypothesis tracking filter represents a belief of location by multiple Gaussians, which is mixture of normal distributions [6, 63]. It can solve the global localization problem, at the cost of increased computational complexity.

The grid localization approximates the posterior using a histogram filter over a grid decomposition of the pose space. Histogram filters decompose the state space into many regions and represent the posterior for each region by a single probability value which is given by piecewise constant functions [65]. We need trade off accuracy and computational efficiency when decomposing a continuous state space into multidimensional grids. The decomposition of state space into a large number of grid cells means much smaller approximation errors. But it will suffer the expense of increased computational complexity. The grid localization can also solve the global localization problem and handle non Gaussian probability distribution.

2.3.4 Monte Carlo Localization

Monte Carlo Localization (MCL) is an implementation of Bayes filter as well. Other than representing the probability density function itself analytically, MCL represents beliefs of state by maintaining a set of samples that are stochastically sampled [59]. The filter used in MCL representing posteriors by finitely many samples is known as particle filter. It can represent a much broader space of distributions than Gaussians and can model nonlinear transformations of random variables [67]. Before the description of Monte Carlo Localization, we will go through the particle filter used in MCL algorithm at first.

The particle filter used in MCL represents the posterior distribution $p(x_t|z_{0:t}, u_{0:t})$ by a set of random samples drawn from this distribution. Each particle, which is a sample of the posterior distribution, represents a possible state to be estimated at time t. In other words, a particle is a hypothesis indicating what the true world state might be at time t. The number of particles is often a large number to reduce the approximation error in practice. The input of particle filter is the particle set x_{t-1}, along with the most recent control u_t and the most recent measurement z_t. Three steps constitute the core of particle filter: sampling, importance weighting and resampling [66]. In sampling, a hypothetical sample set X'_t is generated based on the probability $p(x_t|u_t, x_{t-1})$ and the previous sample set X_{t-1} distributed by $bel(X_{t-1})$. This step involves sampling from the state transition distribution $p(x_t|x_{t-1}, u_t)$. The obtained set of particles is the filter's representation of the predicted belief. Then, in importance weighting, the importance factor $\omega_t^{(i)}$ is calculated for each particle using the probability of the measurement z_t under the particle x_t:

$$\omega_t^{(i)} = \eta p(z_t|x_t^{(i)}) \tag{2.5}$$

where η is the normalizing constant and $p(z_t|x_t)$ is the measurement probability. The weights are used to incorporate the measurement z_t into the particle set. In resampling, the probability of drawing each particle is given by its importance weight. The particles which have a high likelihood associated with them are selected with higher probability, and in doing so the new sample set X_t is randomly chosen from X'_t according to the distribution defined by importance factor $\omega_t^{(i)}$:

$$X_t = \{x_t^{(i)} | i = 1 \dots N\} \sim \{x_t'^{(i)}, \omega_t^{(i)}\} \tag{2.6}$$

By incorporating the importance weights into the resampling process, the distribution of particles change from predicted belief to the posterior belief. More important are the particles with lower importance weights tend to be not contained in final particle set X_t. To initialize the filter, we start at time $t = 0$ with a uniform

distributed sample set and the importance factor $\omega_t^{(i)}$ is initialized to *1/N*. Through recursive computing the three steps, the particles converge to the pose with highest probability.

There are several advantages of using particle approximation [59]. In the original form of Kalman filter, it does not correctly cope with non Gaussian or non linear motion and measurement models, which is unable to recover from position tracking failures. And Kalman filter cannot handle multi-modal densities which are often encountered during global localization. Although these limitations can be accommodated using Extended Kalman filter, most of these difficulties derive from the restricted Gaussian density assumption inherent in the Kalman filter. On the other hand, particle approximation is not bound to certain limited parametric subset of distributions. It is able to represent multi-modal distributions and thus MCL can globally localize a robot. It dramatically reduces amount of memory required compared to grid localization, as particle filter only focus on the particles representing the possible poses of robot instead of keeping all the grids status in the pose space. In addition, it can integrate measurement at a considerably higher frequency which benefits the robot fast localization. Also, particle filter is easy to implement.

MCL is an algorithm that particle filter is applied to estimate posteriors over robot poses in the problem of mobile robot localization. The state transition probability $p(x_t|x_{t-1}, u_t)$ and probabilistic models of sensor measurements $p(z_t|x_t)$ in particle filter correspond to the motion model and measurement model in mobile robot localization respectively. Motion model describes the kinematic states that a robot assumes when executing the motion command u_t and x_{t-1}. Measurement model describe the formation process by which sensor measurements are generated. Table 2.2 is the basic MCL algorithm written in pseudocode, which can be obtained directly by substituting the appropriate probabilistic motion and measurement models into the algorithm particle filter.

1: **Algorithm MCL**($\mathcal{X}_{t-1}, u_t, z_t, m$)**:**
2: $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
3: for $m = 1$ to M do
4: $x_t^{[m]} = \text{sample\_motion\_model}(u_t, x_{t-1}^{[m]})$
5: $w_t^{[m]} = \text{measurement\_model}(z_t, x_t^{[m]}, m)$
6: $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
7: endfor
8: for $m = 1$ to M do
9: draw i with probability $\propto w_t^{[i]}$
10: add $x_t^{[i]}$ to $\mathcal{X}_t$
11: endfor
12: return $\mathcal{X}_t$

Table 2.2: Monte Carlo Localization. [59]

MCL takes the previous belief of state x_{t-1}, control measurement u_t, observation z_t and the given map m as input. Line 4 is the step that involves sampling from the motion model, using particles from present belief as beginning. The measurement model is then used in line 5 in order to calculate the importance weight of that particle. The weight for each particle is normalized from line 9 to line 11. Line 12 to line 15 is the process of resampling for MCL algorithm. Finally, it returns the updated particle set at time t. The initial belief $bel(x_0)$ is represented by the set of samples at the beginning. In global mobile robot localization, it is obtained by randomly drawing M such particles from the uniform distribution over the robot poses space, and assigning the equal importance weight M^{-1} to each particle. As we mentioned, MCL is a recursive algorithm. But there is no stop condition indicated in MCL itself. Usually, it exists during the robot's whole life.

Figure 2.5 shows an experiment example of MCL, in which a robot operating in an office environment of size 54m x 18m [26]. The robot is equipped with laser range finders and given a map of the environment. In Figure 2.5(a), after moving nearly 5m, the robot is still globally uncertain about its position and the particles are spread

uniformly through major parts of the free space. Even as the robot reaches the upper left corner of the map in Figure 2.5(b), its belief is still concentrated around four possible locations due to the symmetry of the environment. Finally, after moving nearly 55m, Figure 2.5(c) shows the ambiguity is resolved and the robot knows where it is. The most of samples are concentrated nearby the robot's true position.

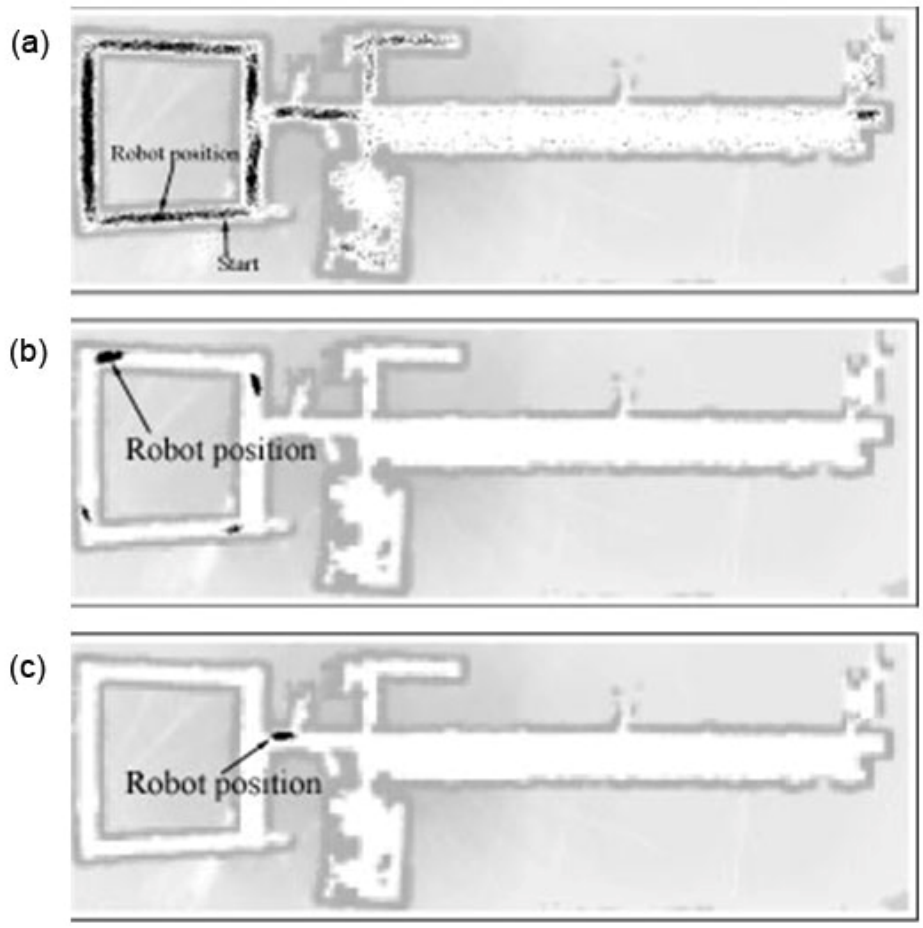

Figure 2.5: Example of Monte Carlo localization. (a) After the robot moves 5m. (b) The robot reaches the upper left corner. (c) After the robot moves approximately 55m. [26]

Chapter 3
The Combined MCL-Clustering Algorithm

3.1 Motivation of Our Method

Many important works on MCL were proposed in recent years [19, 20, 36, 38]. Most of them focus on improving the accuracy and efficiency of the localization algorithm. In 2003, in order to improve the efficiency of MCL, the method adapting the size of particle set during localization was proposed [32]. The key idea is to bind the approximation error of the particle filter measured by the Kullback-Leibler distance. It uses a small number of particles if the distribution of particles is unimodal, and uses a large number of particles if the samples represent a multimodal sample distribution. In 2004, another paper titled “Real-time particle filters” was published [41]. It addressed the problem that sensor information arrives at a higher rate than the update rate of the filter. It represents posteriors as mixtures of particle sets, where each mixture component integrates one observation of all sensors arriving during a filter update. What’s more, other topics related to MCL include many vision-based Monte Carlo localization methods or several updated Monte Carlo localization approaches that run faster and can offer more accurate results. At the same time, MCL was tested in various environments on different robot platforms. In most of these papers, the effectiveness of proposed methods are normally shown using pictures. In the picture, the particles concentrate around the robot position to demonstrate that the algorithm is successful. For example, Figure 3.1 shows the particle set of the robot Robin when it passes the corridor and another robot Marian is operating in the room adjacent to the corridor [60]. The tow pictures (in Figure 3.1) demonstrate the detection of Marian helped the robot Robin successfully localized itself, as the distribution of particle set representing belief of Robin changes from high uncertain to very confident in a certain position. However, it is important to note that the robot does not know whether it is

localized. Here comes an important question: How does the robot know it has successfully localized itself?

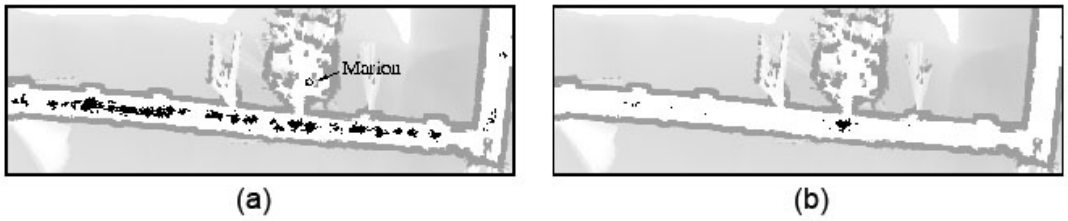

Figure 3.1: An example of two stages in MCL. (a) Global uncertain and (b) confident in a certain position [60]

3.2 The Proposed Method

3.2.1 Problem Statement

For a long time, humans want to make robots intelligent as humans. One of the most primitive mental activities of humans is clustering [34], which is used to handle the huge amount of information we receive every day. For robots, dealing with each individual particle as a specific piece of information would be not enough. We can do the same clustering thing within MCL framework. Through checking clustering information of particle set, the robot can know whether it is localized. Figure 3.2 shows two pictures of localization stages in which the robot true position is represented by the large red cycle. Human beings can easily understand the localization is not ready in Figure 3.2(a), and in Figure 3.2(b) it shows particles concentrated successfully around the true position of the robot. But this information cannot be directly perceived by robots. It should be inferred from the distribution of the particles. Through observing pictures in which different localization outcomes are shown, we plan to bring consciousness to the robot make it have the same understanding of localization outcomes as humans. The robot will notice whether it has localized inferred from the distribution of particles.

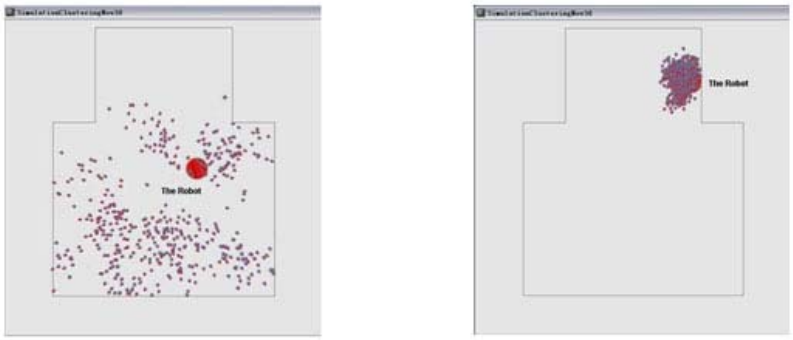

Figure 3.2: An example of two stages in MCL. (a) Global uncertain and (b) confident in a certain position.

3.2.2 Stages of Localization

Within MCL, the concept stages of localization in our method is referred as the different distributions of particle set which have significant characteristics that can be distinguished from each other. As we mentioned above, global localization and position tracking are two typical stages of localization. Usually, humans get the sense of localization stages from pictures in which the distribution of particle set are shown. Six pictures in Figure 3.3 illustrate the possible cases of particle distribution in MCL. In Figure 3.3(a), particles are full filled in the whole free space. The robot totally has no idea where it is. After the robot got its first sensor reading of wall in the environment, the particles are spread around the wall (Figure 3.3(b)). Then, with the localization kept on, the particles may be concentrated in one major part with a little noise (Figure 3.3(c)) or get into several large parts (Figure 3.3(d, e)). Along with several times the robot got detection, the ambiguity is decreased. Finally, particles are centered in one part (Figure 3.3(f)). In literature, no strict definition of localization stages was proposed and even no researchers paid attention to. We simply describe the stages of localization here to give a common sense.

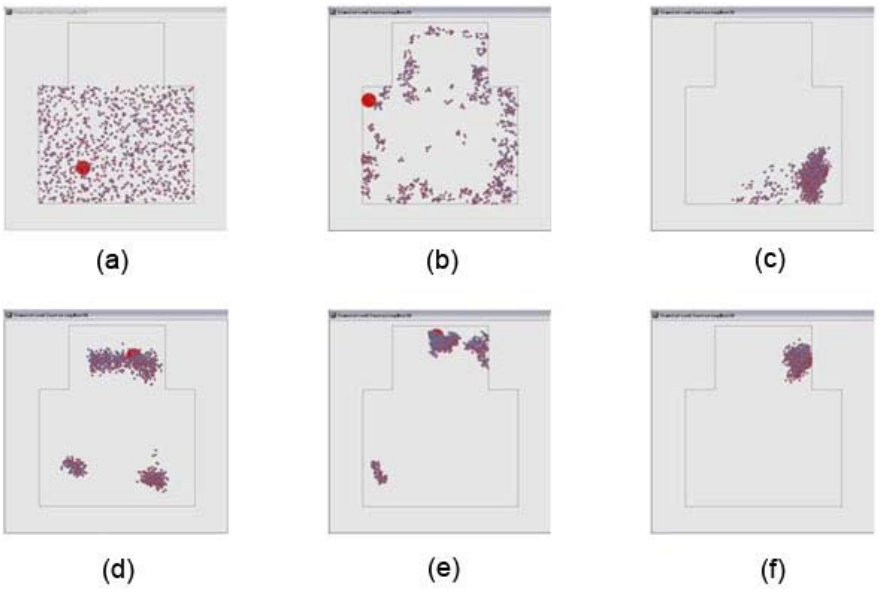

Figure 3.3: An example of six stages in MCL. (a) Uniform distribution of particles. (b) Particles are spread around the wall. (c) Particles are concentrated in one major part with a little noise. (d, e) Particle set splits into several large parts. (f) Particles are centered in one position.

3.2.3 Description of Proposed Method

Our framework is based on Monte Carlo Localization, which is combined with a clustering algorithm. Although MCL is a recursive robot localization algorithm, it does not have a terminate condition to stop the recursive process and there is no way to notify the robot stages of localization. To offset the shortcomings of MCL, in our proposed method, the distribution of the particle set is analyzed by sending to the clustering part. Then, the clustered particle set is further used to extract information. In order to describe the stages of localization, three characteristic variables are calculated respectively. They are

1. n_c , the number of clusters,
2. n_{max} , the number of particles in the cluster which has the maximum number of particles compared to other clusters, and
3. p_{max}, the percentage of n_{max} in the current whole set of particles.

The number of clusters n_c indicates how many clusters the particle set has. That is how many areas the particles spread at. Both of the n_{max} and p_{max} refer to the size of a cluster. But the third characteristic variable, the percentage p_{max} , is more general than n_{max} as the meaning of n_{max} depends on the size of particle set. Usually, p_{max} is used as a major indicator to distinct the stages of localization. For instance, when the p_{max} exceeds a predefined threshold, the robot will believe the particles are well congregated. The first one, current number of clusters n_c is used as a secondary indicator of the stages of localization. For example, if the clustered particle set has two large percentages *p1* and *p2*, corresponding to the size of cluster one and cluster two, one is 53% and the other one is 45%. And the number of clusters n_c is 3, indicating that 3 clusters exist. So, based on these analyses, the robot can infer it may be at two possible areas that the two large groups of particles represent and the rest of particles are very likely the noise.

3.3 Selection of Appropriate Cluster Algorithm

3.3.1 Introduction of Clustering

Clustering is a subfield of pattern recognition whose goal is the classification of objects into a number of categories or classes [34]. The objects can be images or hand writing characters or any type of measurements that need to be classified. In pattern recognition, one kind of clustering is known as unsupervised learning or learning without a teacher, as the class labeling of the patterns to be classified is not given as prior. Clustering allows us to uncover the similarities or differences among patterns, then to draw useful conclusions. The idea of clustering has been applied in many fields, such as life science, social science, medical science, earth science, and engineering. The definition of clustering is based on the definition of a single cluster. In literature, the clusters are described as “continuous regions of this space containing a relatively high density of points, separated from other high density regions by

regions of relatively low density of points." [68] Clusters described in this way are very close to the visual perception of the scenes in MCL. Particles may concentrate at a few different locations of the map after the robot moved a short time perceiving outside word in an indoor environment. Here, we give a simple idea of what clustering is. We define it as the partition of a data set

$$X = \{x_1, x_2, \ldots, x_n\}, \tag{3.1}$$

where X is a set of vectors constituted by n vectors x_i. The set X includes a group of vectors, into m small sets (clusters C_i),

$$C_1, \ldots, C_m,$$

if the following conditions are met:

$$C_i \neq \Phi,\ i=1, \ldots, m; \quad \cup_{i=1}^{m} C_i = X; \quad C_i \cap C_j = \Phi,\ I \neq j,\ i, j = 1, .., m \text{ [34]}$$

That is, at least there exists one cluster; all clusters constitute the whole data set and no vector belongs to two different clusters. Clearly, the vectors in the data set contained in the same cluster C_i are more "similar" to each other and less "similar" to the vectors of the other clusters. The definition of "similar" and "dissimilar" will be given in next section.

3.3.2 Proximity Measures and Representatives

This section talks about the definitions concerning measures between vectors and the measures between clusters. The first type measures are defined as proximity measures. Proximity measure quantifies how "similar" or "dissimilar" two vectors are. We choose Euclidean distance $d(x,y) = \sum\sqrt{(x_i - y_i)^2}$ between vectors as our proximity measures which is a dissimilarity measure d described as a function as follows:

$$d: X \text{ x } X \text{ -> } R \tag{3.2}$$

where R is the set of real numbers and X is the data set, such that

$$\exists d_0 \in R: -\infty < d_0 \leq d(x, y) < +\infty, \quad \forall x, y \in X \tag{3.3}$$

$$d(x, x) = d_0, \quad \forall x \in X \tag{3.4}$$

and

$$d(x, y) = d(y, x), \quad \forall x, y \in X \tag{3.5}$$

Equivalences (3.3) and (3.4) indicate that there is a minimum dissimilarity level value d_0 when the measure of dissimilarity is between the vectors itself. Equivalence (3.5) means the dissimilarity measure between vectors is independent of the order. The similarity measure on X is defined as nearly the same way, in which the only difference is $\exists s_0 \in R: -\infty < s(x, y) \leq s_0 < +\infty, \quad \forall x, y \in X$. No doubt Euclidean distance is a dissimilarity measure on X, with $d_0 = 0$; the distance of a vector from itself is equal to 0, and it is easy to say that $d(x, y) = d(y, x)$.

In many clustering algorithms, a vector x assigned to a cluster C takes into account the proximity measure between x and C, namely $g(x, C)$. Many ways can be used for the definition of $g(x, C)$. One of them is that C is equipped with a representative and the proximity between x and C is measured as the proximity between x and the representative of C. Several types of representatives have been used in the literature, such as point representatives (Figure 3.4(a)), hyperlane representatives (Figure 3.4(b)) and hyperspherical representatives (Figure 3.4(c)). Among them, point representatives are suitable for compact clusters (Figure 3.4(a)), which correspond to the scene that we always see from the show of the distribution of particles in MCL on screen.

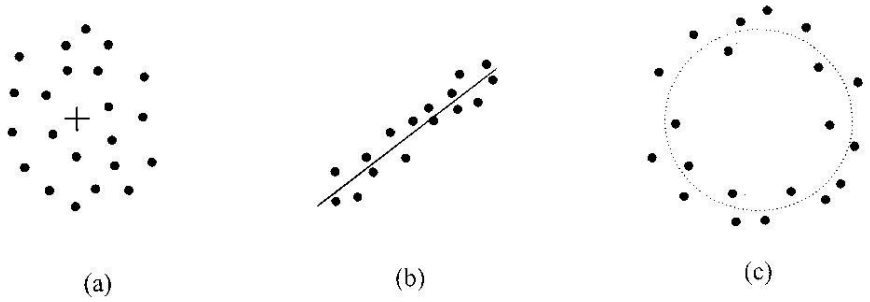

Figure 3.4: Several types of representatives for different clusters: (a) Point representative for compact cluster;(b) Hyperplane representatives for clusters of linear shape;(c) Hyperspherical representatives for clusters of hyperspherical shape. [34]

The mean point is used as the representative of a cluster in this thesis:

$$m_p = \frac{1}{n_p}\sum_{x \in C} X \tag{3.6}$$

where n_p is the cardinality of C. It is the most common choice when point representatives are employed and when dealing with data of a continuous space.

3.3.3 Categories of Clustering Algorithms

After adopting an appropriate proximity measure and criterion for clustering, a specific algorithmic scheme need be chosen to unravel the clustering structure of the data set. A clustering algorithm is a learning procedure that tries to identify the specific characteristics of the clusters underlying the data set. A lot of clustering algorithms have been used in many different fields. The major clustering algorithms include sequential algorithms, hierarchical clustering algorithms, clustering algorithms based on cost function optimization and others [72]. As we need calculate the clusters in real time and probabilistic robotics have been criticized by computation complexity, the speed of the clustering algorithm is the most important reason we consider.

The sequential algorithms are fast methods and quite straightforward. In most of them, all the feature vectors are inputted to the algorithm once. These methods usually produce compact and hyperspherically shaped clusters, depending on the proximity

measure used. One of the sequential algorithms is chosen to calculate the clusters in our proposed method because of its fast speed. We will focus more on this clustering algorithm later. Compared to the sequential algorithms, the hierarchical clustering algorithms produce a hierarchy of nested clustering instead of a single clustering. The total number of operations required by these clustering algorithms is proportional to $O(N^3)$, which is a huge computation burden. Even efficient computational schemes for hierarchical clustering are employed, this cannot become less than $O(N^2)$. And the clustering algorithms based on cost function optimization evaluate a clustering by a cost function. They terminate when an optimum of the cost function is achieved. This category of clustering is computationally very demanding and usually the number of clusters is kept fixed. It offends the rule our method needs the clusters is evolving along with localization time.

3.4 The Appropriate Clustering Algorithm: BSAS

Here, we focus on a basic sequential algorithmic scheme (BSAS) for clustering, which is employed in our proposed method. All clustering algorithms in this sequential category share the same attributes. For example, all the vectors are presented to the algorithm only once. The number of cluster is not known as a prior. In fact, new clusters are created as the algorithm evolves. The core idea of the BSAS is the following: As each new sample is considered, it is either assigned to an existing cluster or assigned to a newly created cluster, depending on its distance from the already formed ones. The user-defined parameter required by the algorithmic scheme is the threshold of dissimilarity θ. Let m be the number of clusters that the algorithm has created up to now, and then the algorithmic scheme can be stated as (Table 3.1):

The Basic Sequential Algorithmic Scheme (BSAS)	
1.	m = 1
2.	$C_m=\{x_1\}$
3.	For i=2 to N
4.	-Find C_k:$d(x_i,C_k)=\min_{1\leq j\leq m} d(x_i,C_j)$
5.	-If($d(x_i,C_k)> \theta$) then
6.	* m = m+1
7.	* $C_m = \{x_i\}$
8.	-Else
9.	* $C_k = C_k \cup \{x_i\}$
10.	*Where necessary, update representatives
11.	-End {if}
12.	End {For}

Table 3.1: The Basic Sequential Algorithmic Scheme (BSAS). [34]

BSAS takes the data set X that need to be clustered and the threshold of dissimilarity θ as input. Line 1 and 2 initialize the first cluster including the first vector x_1. Line 3 to line 12 is a large for loop going through the rest of the data. Line 4 calculates dissimilarity measures between current vector and every cluster to find a minimum one. From line 5 to line 11, if the minimum measure is larger than θ, a new cluster that contains current vector will be created. Otherwise, the considered vector will be assigned to the existing cluster which has a minimum dissimilarity measure to it. Other than dissimilarity measure, different choices of $d(x, C)$ may lead to different algorithms. And when C is represented by a single vector, $d(x, C)$ becomes $d(x, C) = d(x, m_c)$ where m_c is the representative of C. In the case in which the mean vector is used as a representative, the updating may take place in an iterative fashion, that is, $m_{c_k}^{new} = (n_{c_k}^{old} * m_{c_k}^{old} + x)/ n_{c_k}^{new}$, where $n_{c_k}^{new}$ is the cardinality of C_k after the assignment of x to it and $m_{c_k}^{new}$ is the representative of C_k after the assignment of x to it.

The BSAS scheme can be used with similarity instead of dissimilarity measures with appropriate modification; that is, the min operator is replaced by max. BSAS,

with point cluster representatives, favors compact clusters. Thus, it is not recommended if there is strong evidence that other types of clusters are present. No doubt, MCL, which always produces compact clusters, does not have this trouble. The BSAS algorithm performs a single pass on the entire data set, *X*. During each iteration, the distance of the vector currently considered from each of the clusters produced so far is computed. Because the final number of clusters *m* is expected to be much smaller than *N*, the time complexity of BSAS is *O(N)*. Such fast computation time is the major reason that BSAS is chosen aside from other clustering algorithms.

Tow unfavorable facts matter the result of BSAS [70]. It is easy to notice that the order in which the vectors are presented to the BSAS plays an important role in the clustering results. Different presentation ordering may lead to totally different clustering results. The number of clusters and the clusters themselves may differ. Another important factor affecting the result of the clustering algorithm is the choice of the threshold. This value directly affects the number of clusters formed by BSAS. If the threshold is too small, unnecessary clusters will be created. On the other hand, if the threshold is too large a smaller number than appropriate number of clusters will be created. In both cases, the number of clusters that best fits the data set is missed. The threshold depends on the interpretation the expert or the experiment experience gives.

To reduce the effects of the above two factors, many modifications of BSAS are proposed. One is that the vectors of *X* have to be presented twice to the algorithm [71]. The algorithmic scheme consists of two phases. The first phase involves the determination of clusters, assigning some of the vectors to them. During the second phase, the unassigned vectors are presented for a second time to the algorithm and assigned to the appropriate cluster. Another modified BSAS employs two thresholds, which defines a "gray" region [70]. If the dissimilarity level *d(x, C)* of a vector *x* from its closest cluster *C* is less than Θ_1, *x* is assigned to *C*. If *d(x, C)* > Θ_2, a new cluster is formed and *x* is placed in it. Otherwise, when the dissimilarity level *d(x, C)* drops in

this region [Θ_1, Θ_2], there exists uncertainty and the assignment of x to a cluster will take place at a later stage. However, all the modifications suffer increased computational complexity. Based on our experiments, the basic sequential algorithmic scheme is robust enough to get the solution we want. Therefore, only BSAS is adopted in the thesis.

3.5 Details of Proposed Method

After introducing specific clustering algorithm, proximity measures and representatives, now we present the proposed method, the combined MCL-Clustering algorithm. A pseudocode description of the MCL-Clustering is summarized in Table 3.2. It is clear that the input of the algorithm is almost the same as MCL, including the previous state particles X_{t-1}, the motion u_{t-1} and observation z_t. In fact, the first step of the algorithm is regular MCL to localize the robot. Other than the parameters relative to MCL, the total number of current particles n_{total} and an additional criteria θ for clustering are kept for clustering part. Criterion θ is used as the threshold in BSAS to determine whether a particle belongs to an existing cluster or is assigned to a newly created cluster. And the total number of current particles n_{total} is used to compute p_{max}. Starting from the second step, extra process of particle set is affiliated. At first, we take the resampled particle set X_t and the criterion θ as input of the basic sequential algorithmic scheme to obtain a clustered particle set C_t . Then, by analyzing the clustered particle set C_t , we calculate the number of clusters n_c , the number of particles in the cluster which has the maximum number of particles compared to other clusters n_{max}, and the percentage of n_{max} in the current whole set of particles p_{max}. The variable n_{max} is an intermediate variable used to calculate p_{max} at step 5 which is relative to the total number of current particles. If the particle size is not kept fixed, the value of n_{max} does not make sense. As we suggested before, p_{max} is used as a major indicator. At step 6 of the algorithm (Table 3.2), when p_{max} exceeds a predefined maximum allowable percentage q, the robot can start doing other jobs or stop to

indicate localization is successfully completed. The n_c, current number of clusters, is a secondary indicator. To explain the outcome of localization, it is suggested that all of the variables p_{max}, n_c and even θ are shown along with pictures of the distribution of the particle set. Choosing one of p_{max} and n_c or both of them to facilitate the robot relies on the specific environment and robot platform. The explanation of choice may be given by experts or a lot of previous experience.

The combined MCL-Clustering algorithm
1. $X_t = MCL(X_{t-1}, u_t, z_t)$
2. $C_t = BSAS(X_t, \theta)$ //clustered particle set
//number of clusters in clustered particle set
3. $n_c = numberOfClusters(C_t)$
// the number of particles in the cluster which has the maximum number of particles compared to other clusters
4. $n_{max} = maxParticleNumbers(C_t)$
// the percentage of n_{max} in the current whole set of particles, n_{total} is the number of current particles
5. $p_{max} = n_{max}/n_{total}$
6. if ($p_{max} > q$)
stop to indicate localization is successful completed or start doing other jobs
7. Return X_t, C_t, n_c, n_{max}, p_{max}

Table 3.2: The combined MCL-Clustering algorithm.

Chapter 4
Implementation and Experiment Results

The implementation of our proposed method and experiment results are described in this chapter. The implementation will be presented at first, covering hardware platform and its setup, programming environment and MCL algorithm implemented on iRobot Create. Then we will present the experimental results.

4.1 Implementation Details

4.1.1 Hardware Platform

The hardware platform we used to test the performance of our method is the robot called Create. Create is an autonomous mobile robot for educators and developers built by iRobot Corporation [80]. iRobot has made some of the world's most important robots. Today, iRobot has grown to a $307 million public company that employs more than 400 of the robot industry's top professionals, including mechanical, electrical and software engineers and related support staff. iRobot Create is a complete robot development kit which allows us to program the robot without having to worry about mechanical assembly and low-level code. Create is a low-cost robot available for research and education. The premium development package which includes Create programming robot, command module, advanced power battery system, virtual walls, standard remote and self charging home base with fast charger only costs $299. Create is derived from another iRobot cleaning robot called Roomba, but without the vacuum cleaning brushes. Nowadays, Roomba is widely used for cleaning floors in homes and businesses. The platform has been proved as a low cost robot available for research and education.

Although Create provides a convenient programming interface for controling the robot without knowing the details inside the Create, knowing how it actually works can help diagnose problems encountered in experiments. Create is organized in three sections:

Sensor front: Nearly all of the sensors, such as bump, wall, cliff sensors, and home base contacts, are at the front of the robot. The Create is designed to always travel forward, so all the most sensitive sensors are mounted on the movable front bumper. This movable bumper provides a mechanical way to measure contact and absorbs shock to minimize damage.

Motor middle: The main drive motors and battery are all in the center. This part gives the Create ability of moving and offers the power for long last motion. It is very close to the center of its body.

Cargo bay: An importance connector is located in the front middle of the cargo bay. It makes extension to attach other peripheral devices such as additional sensors, light, or motors to the Create easy. It can also be used to connect Create to a laptop. The free space of cargo bay can be used to add a payload to the back portion of the robot, changing the center of gravity of the robot.

A robot without any sensor is just a fancy toy. The Create has a wide variety of sensors for navigating, which mainly include mechanical bump sensors, infrared wall sensors. In particular, for detecting dangerous conditions, it also has infrared cliff detectors and wheel drop sensors. When Create detects being hanged in the air, the wheels will pause. Details of major sensors are described below.

Bump sensors: Create has two bump sensors on the front. The spring-based front bumper moves to trigger one or both of these sensors. Each is implemented as an

optical interrupter, which senses the absence of light and changes an electrical signal.

Infrared Sensors: All six infrared sensors are on the front bumper. Four of them are the cliff sensors and facing down, and another one to the right is the wall sensor. These five sensors have LED emitters inside, looking for the reflected light of the LED's. Cliff sensors are looking for light reflected from the floor and wall sensor is looking for light reflected from a wall. The last infrared sensor is the one for remote control, virtual wall and docking station. It looks like a small round plastic button at the 12 o'clock position on the head of Create.

Internal Sensors: The internal wheel encode sensors and wheel drop sensors are most commonly used. The wheel encode sensors will be discussed shortly. The wheel drop sensors are equivalent to cliff detection. They detect when the wheels have extended down, indicating that the Create is in some dire situation. Moreover, there are several internal sensors for power measurement, since power is so important in a robotic system. They are useful for estimating and presenting capacity of the battery.

The wheel encode sensors offer odometry data such as the distance and angle travelled which represent the controls in the odometry motion model. Details of odometry motion model used for the experiments will be described later. The distance is obtained from the optical interrupter sensor on the wheels. The value comes from counting the number of beam interruptions caused from the toothed interrupter disc. The angle value comes from an odometrical difference way. Create has a distance sensor on each wheel, and the angle value in the sensor data is the difference traveled by each wheel. This difference describes a rotation around the center point between the two wheels. The distance and angle together describe the path Create travels. Reading them periodically enables us to build up the complete path travelled by connecting path segments that each reading represents.

Another important device of Create that cannot be ignored is the BAM wireless accessory. BAM Wireless Accessory provides Bluetooth capabilities to Create. It allows us to connect and communicate with Creative using any Bluetooth enabled device, such as laptop. It's the most flexible method of communication, allowing a range up to 100 meters and connection quality over distance drops slowly.

4.1.2 Programming Environment

The Create Open Interface (OI) consists of an electronic interface and a software interface for controlling Create's behavior and reading its sensors. The electronic interface includes a 7 bin Mini-DIN connector and a DB-25 connector in the Cargo Bay for connecting hardware and electronic for sensors and actuators such as a robotic arm or light sensor to Create. The software interface lets us manipulate Create's behavior and read its sensors through a series of commands including mode commands, actuator commands, song commands, demo commands, and sensor commands that we send to Create's serial port via a PC or microcontroller that is connected to the Mini-DIN connector or Cargo Bay Connector.

The Cargo Bay Connector, located in the front middle of the cargo bay, contains 25 pins that we can use to attach electronics for peripheral devices such as additional sensors. The Cargo Bay Connector provides four digital inputs, an analog input, three digital outputs, three high-current low side driver outputs, a charging indicator, a power toggle input, serial Tx and Rx, a 5V reference, battery ground and battery voltage.

The Create OI has four operating modes: Off, Passive, Safe, and Full. After a battery change or when is first supplied, the OI is in "off" mode. Once it receives the Start command, we can enter into any one of the four operating modes by sending a mode command to the OI. We can also switch between operating modes at any time

by sending a command to the OI for the operating mode that we want to use.

Passive Mode: Upon sending the Start command, the OI enters into Passive mode. When the OI is in Passive mode, we can request and receive sensor data using any of the sensors command, but we cannot change the current command parameters for the actuators. To change how one of the actuators operates, we must switch from Passive mode to Full mode or Safe mode.

Safe Mode: When we send a Safe command to the OI, Create enters into Safe mode. Safe mode gives us full control of Create, with the exception of the following safety-related conditions: detection of a cliff and a wheel drop, or charger plugged in and powered. Create stops all motors and reverts to the Passive mode when one of the above safety-related conditions occur while the OI is in Safe mode.

Full Mode: When we send a Full command to the OI, Create enters into Full mode. Full mode gives us complete control over Create, all of its actuators, and all of the safety-related conditions that are restricted when the OI is in Safe mode, as Full mode shuts off the cliff, wheel-drop and internal charger safety features. To put the OI back into Safe mode, we must send the Safe command.

Figure 4.1 illustrates the relationships between each states and how to transfer from one to another.

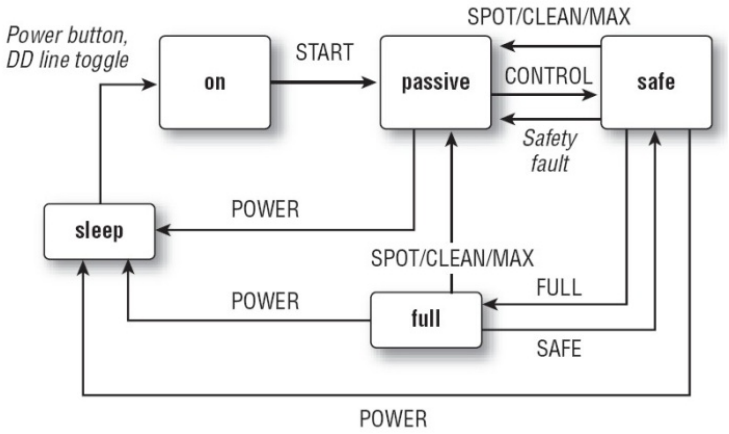

Figure 4.1: Create ROI state. [25]

An encapsulation of the Create OI binary commands into a more easy-to-use Java class is the RoombaComm API. Although it is designed for robot Roomba, it works very well with Create. When first experimenting with the Create OI, it's common to use some sort of a serial terminal program that can send binary sequences to try out various Create OI commands. However, this quickly gets tiresome. It would be a lot easier if there was a library to codify the exact recipe needed to make something work. The RoombaComm API is just such an encapsulation of the Create OI binary commands into a more easy-to-use Java class. The other main benefit is how the Create OI commands are then used as "primitives" to create more complex behaviors.

The programming language and environment called Processing we used to code and run our program is free and open source for people who want to write graphical programs quickly. Processing allows us to write simple 2D interface enables visualization of all the Create's local sensing. Processing supports RoombaComm API and all Java Class. Processing is implemented in Java. The Processing language is really no different from Java at all; it just removes the visible overhead and complexity from Java. So Processing is a Java IDE of sorts, albeit a simplified and specialized one. It enables us to write quick graphical code sketches that respond to user input. The process has been streamlined and made transparent to the user. Its sketching metaphor and helper functions enable us to try out ideas fast. Its direct

approach to painting on the screen is attractive. And it's a good tool for beginning programmers due to its simplified environment and language. Processing is a great environment for getting coding ideas up and running fast, especially if an animated graphics component is included. It's pretty easy to use the full Java class library or wrap up any other Java class into a Processing library.

4.1.3 Implementations of MCL

As we mentioned in chapter 2, in probabilistic robotics, the two basic models in mobile robot localization, which correspond to perception and action, are measurement model and motion model, respectively. We will describe the implementations of them and the resampling algorithm we used in detail below.

(a) Motion Model

Within the framework of MCL, motion model corresponds to the step sampling from the state transition distribution $p(x_t|u_t, x_{t-1})$, which generates a hypothetical state x_t based on the particle set x_{t-1} and the control u_t. The motion model plays an essential role in the prediction step of MCL. The robot motion of probabilistic robotics conforms to the fact that the outcome of a control is uncertain, because of the control noise or unmodeled effects. Thus, the outcome of a control will be represented by a posterior probability.

The robot motion, formally kinematics, is the calculating of the effect of control actions on the configuration of a robot [74]. The configuration of a mobile robot is commonly described by three variables in planar environments, referred as pose (x, y, θ). The pose without orientation is robot's location. The probabilistic kinematic model, or motion model, plays the role of the state transition model $p(x_t|u_t, x_{t-1})$ in mobile robotics. The x_t and x_{t-1} are both robot poses and u_t is a motion command. This model describes the posterior distribution over states of kinematic when then motion

command u_t is executed at x_{t-1}. Figure 4.2 shows two examples of kinematic model for a mobile robot controlled in a planar environment with the robot's pose initialed as x_{t-1}. The shaded area shows the distribution $p(x_t|u_t, x_{t-1})$: the darker a pose, the more likely it is. In Figure 4.2(a), the robot moves forward some distance, with the increased translational and rotational error as indicated. Figure 4.2(b) shows the outcome of a more complicated motion command, which results to a larger spread of uncertainty.

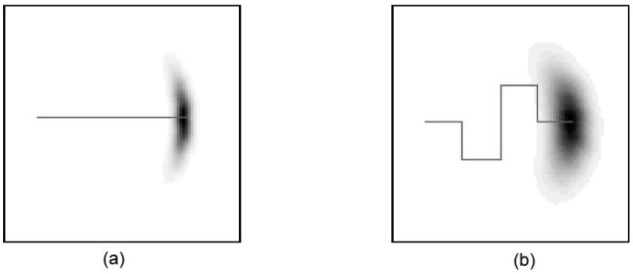

Figure 4.2: Probabilistic generation of robot kinematic. (a) A path of moving forward. (b) A path of more complicated motion command. The darker an area, the more likely robot is. [24]

There are two common probabilistic motion models for mobile robots: velocity motion model and odometry motion model [26]. Velocity motion model assumes we control a robot through two velocities, a rotational and a translational velocity and odometry motion model assumes we have access to odometry information, which is commonly obtained by integrating wheel encoder information. Velocity motion model calculates the probability $p(x_t|u_t, x_{t-1})$ of being at x_t after executing the control x_t at the state x_{t-1}. It assumes that the control is carried out for the fixed short time duration Δt. Odometry motion model is the one used in our proposed method.

The odometry information consists of the distance a robot passed and the angle a robot rotated. Most of the commercial robots provide odometry using kinematic information. Create has wheel encoders to obtain odomety information. Moreover, we can get the odometry reading of Create through the Open Interface. Strictly speaking, odometry information is sensor measurement, not control. However, in robotics, odometry is normally considered as controls. Practical experience suggests that

odometry is usually more accurate than velocity. In odometry model, the robot motion in the time interval [*t-1, t*] is approximated by a rotation, followed by a translation and a second rotation (Showed in Figure 4.3). Both the turns and translation suffer noisy such as drift and slippage. These three steps are calculated together with motion error.

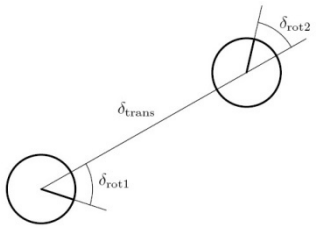

Figure 4.3: Odometry model. [26]

The details of the odometry motion model used in our experiments are showed in Table 4.1. The algorithm sample_motion_model_odometry accepts an initial pose x_{t-1} and an odometry reading u_t as input, and outputs the predicted pose x_t distributed according to the state transition probability $p(x_t|u_t, x_{t-1})$. Sample based means it uses particles to represent robot pose. Each particle is randomly guessed representing a likely pose of robot. And the particles move based on the odometry information. Unfortunately, in reality, the robot motion is not perfect and accurate. If the command dictates the robot go straight, it probably goes through a curve. Therefore noises need to be simulated. Noise is modeled by a normal distribution with zero-centered random variables with finite variance. We have four robot-specific error parameters. From line 5 to line 7, we calculate the motion that the control indicates at first, and then add noisy. The sample function return a random number based on the normal distribution. The $\alpha 1$ and $\alpha 2$ are rotation error which works on rotation. The $\alpha 3$ and $\alpha 4$ are translation error which works on translation [26].

1: **Algorithm sample_motion_model_odometry**(u_t, x_{t-1})**:**

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

3: $\delta_{\text{trans}} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^s}$

4: $\delta_{\text{rot2}} = \bar{\theta}' - \bar{\theta} - \delta_{\text{rot1}}$

5: $\hat{\delta}_{\text{rot1}} = \delta_{\text{rot1}} - \textbf{sample}(\alpha_1 \delta_{\text{rot1}} + \alpha_2 \delta_{\text{trans}})$

6: $\hat{\delta}_{\text{trans}} = \delta_{\text{trans}} - \textbf{sample}(\alpha_3\, \delta_{\text{trans}} + \alpha_4(\delta_{\text{rot1}} + \delta_{\text{rot2}}))$

7: $\hat{\delta}_{\text{rot2}} = \delta_{\text{rot2}} - \textbf{sample}(\alpha_1 \delta_{\text{rot2}} + \alpha_2 \delta_{\text{trans}})$

8: $x' = x + \hat{\delta}_{\text{trans}}\, \cos(\theta + \hat{\delta}_{\text{rot1}})$

9: $y' = y + \hat{\delta}_{\text{trans}}\, \sin(\theta + \hat{\delta}_{\text{rot1}})$

10: $\theta' = \theta + \hat{\delta}_{\text{rot1}} + \hat{\delta}_{\text{rot2}}$

11: *return* $x_t = (x', y', \theta')^T$

Table 4.1: Sample base odometry motion model algorithm. [26]

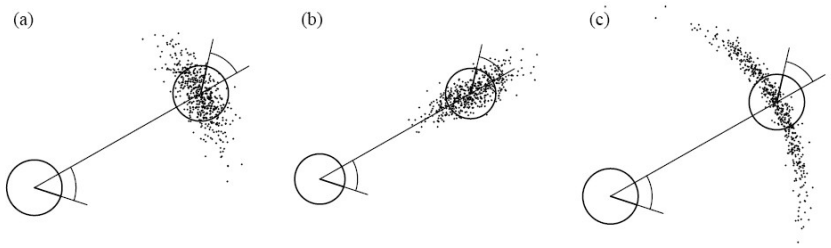

Figure 4.4: Sampling from the odometry motion model. (a) With moderate error parameters. (b) With small angular error but large translational error. (c) With large angular and small translational error. [26]

Here are two examples of odometry motion model implementation. In Figure 4.4, it uses 500 particles. Figure 4.4(a) is a normal one. Figure 4.4(b) is with large translation error but small rotation error and Figure 4.4(c) is with large rotation error but small translation error. The effects of different translation and rotation settings are showed. Another example (Figure 4.5) is a picture for a non-sensing robot [61]. The solid line displays the robot control information, and the particles represent the robot's belief at different location on time. The robot starts with the all particles concentrated. It illustrates how the uncertainty grows as the robot moves. If a robot cannot get information from outside environment, it's going to be lost quickly. The particles are spread across an increasingly large space.

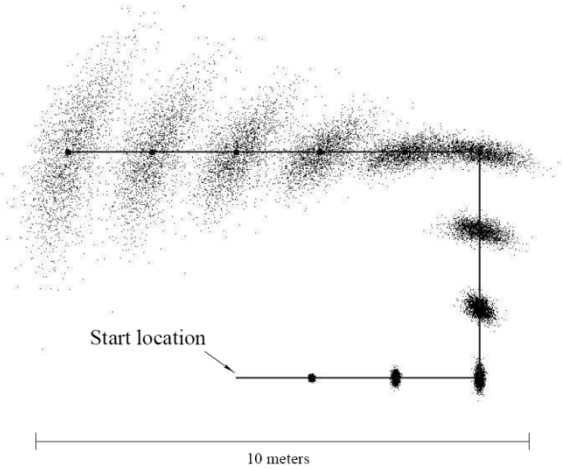

Figure 4.5: Sampling approximation of the position belief for a non-sensing robot. [61]

(b) Measurement Model

Another important model in probabilistic robotics is the measurement model. The probabilistic models of sensor measurements $p(z_t|x_t)$ are essential for the measurement update step in MCL. Measurement models describe the formation process by which sensor measurements are generated in physical world. Today's robots use a variety of different sensors, such as tactile sensors, range sensors, or cameras. Probabilistic robotics explicitly models the inherent uncertainty in sensor measurements. Create has bumper sensors and infrared sensors. The bumper sensors return feedbacks only when they detect a hard surface. The sensors equipped on Create for detecting the external environment are really limited. In our experiment, the positive return from bump sensors means that Create touches the wall. Then, high weight will be assigned to the particles which are around the wall, and low weight will be given to the rest of particles. The weighted particles are forward to the next step importance sampling.

(c) Resampling

Another important component of MCL is known as resampling or importance sampling. In our experiment, the algorithm low variance sampling is chose for resampling. The basic idea of low variance sampler includes a sequential stochastic process. Instead of choosing M random numbers and selecting those particles that

correspond to these random numbers, this algorithm computes a single random number and selects samples according to this number but still with a probability proportional to the sample weight [26]. The algorithm showed in table 4.2 selects particles by repeatedly adding the fixed amount M^{-1} to r and by choosing the particle that corresponds to the resulting number. The while loop at step 8 stops when i is the index of the particle such that the corresponding sum of weights exceeds U. Then the particle is selected. The low variance sampler is very efficient. It has a complexity of $O(N)$.

```
1:    Algorithm Low_variance_sampler(X_t, W_t):
2:        X̄_t = ∅
3:        r = rand(0; M^-1)
4:        c = w_t^[1]
5:        i = 1
6:        for m = 1 to M do
7:            u = r + (m − 1) · M^-1
8:            while u > c
9:                i = i + 1
10:               c = c + w_t^[i]
11:           endwhile
12:           add x_t^[i] to X̄_t
13:       endfor
14:       return X̄_t
```

Table: 4.2: Low variance sampler for particle filter. [26]

4.2 Experimental Results

The performance of our method is tested on both real and simulated robots. The goal of the experiments is to verify that: If the robot is lost or successfully localize itself according to the distribution of the particles showed on screen, the robot can understand the changing stages of localization indicated by the distribution of the particles nearly as well as people.

In both the real robots experiment and simulation, we track three important characteristic variables, namely, (1) the number of cluster n_c, (2) the number of particles in the cluster which has the maximum number of particles n_{max}, and (3) the percentage of n_{max} in the current whole set of particles p_{max}.

4.2.1 Experiments Using Real Robots

The experimental results performed on the real robot Create is presented in this section. To demonstrate the robot has consciousness about being lost or localized, we designed two types of experiments. In the first experiment we test the case of tracking the robot without receiving sensor readings, in which the robot will be aware of being lost. The second experiment is for global localization, in which the robot will be aware of being localized. The purpose of these experiments is that the robot can have consciousness about the changing stages of Monte Carlo Localization. As a result, we will see that in tracking without perception the robot is going to stop when a large number of dispersed particles are seen on screen. And in global localization, the robot will stop to show it is successfully localized when the particles are centered on the robot position and give the location and orientation according to the cluster that includes the maximum number of particles.

(a) Tracking without Sensor Readings

Tracking without sensor readings was tested by placing Create in a field of 152.5x152.5cm around with wall. During tracking, the robot is not allowed to get any sensor reading. It drives in a square whose sides have length 80cm. At time 19s, the robot stopped as it found itself with high uncertainty in Figure 4.6(f). In this test, the number of particles is initialized as 1000 and the criterion of clustering θ is set to 17cm which is the same as the radius of the robot. Criterion θ is a threshold used in BSAS to determine whether a particle belongs to an existing cluster or is assigned to a newly created cluster. Therefore, the cover area of particles assigned to the same cluster will be a circle which has the same size as the robot. Another parameter need to be set as priori is the threshold q of p_{max} that indicates the robot is lost. We set it to 20% of current number of total particles, which means if the number of particles in the largest cluster is lower than 20% of the total particle size, the robot will believe it has lost. Here, the “large” or “small” cluster is decided by the number of particles a cluster

contains. The criterion θ and the threshold q are set according to our previous experience of working with Create. There is no definite answer about which is the perfect setting for these parameters.

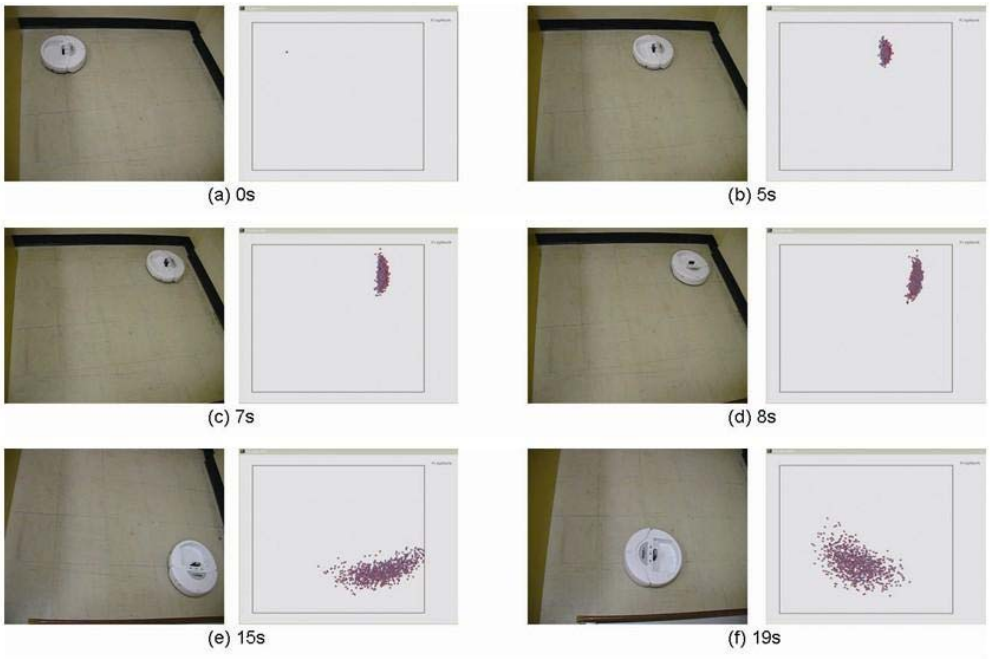

Figure 4.6: The robot true pose and distribution of particles during experiment tracking without sensor readings at time (a) 0s, (b) 5s, (c) 7s, (d) 8s, (e) 15s, (f) 19s.

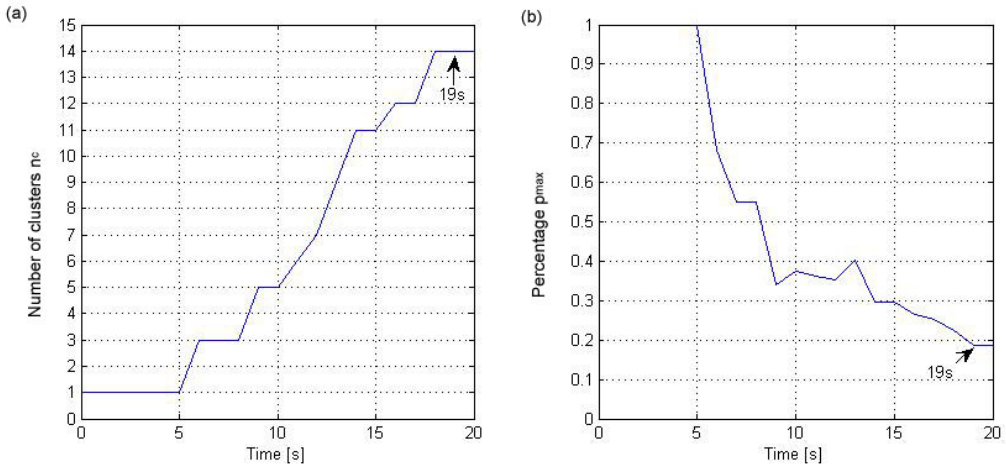

Figure 4.7: The plots of corresponding (a) number of clusters n_c and (b) the percentage of n_{max} in the current whole set of particles p_{max} .

The value of n_c, n_{max}, p_{max} at time 0s, 5s, 7s, 8s, 15s, 19s	0s	5s	7s	8s	15s	19s
Number of clusters: n_c	1	1	3	3	11	14
Number of particles in the cluster with the maximum number: n_{max}	1000	1000	551	551	296	184
Percentage of n_{max} in the current whole set of particles p_{max}	1.0	1.0	0.551	0.551	0.296	0.185

Table 4.3: The value of n_c, n_{max}, p_{max} at time 0s, 5s, 7s, 8s, 15s, 19s.

Figure 4.6 shows the robot true pose and the distribution of the particles at time 0s (a), 5s (b), 7s (c), 8s (d), 15s (e), 19s (f). In each subfigure, the left picture shows the robot true pose and the right picture is the distribution of the particles at the same time. The plots of two corresponding characteristic variables, number of clusters n_c and the percentage of n_{max} in the current whole set of particles p_{max} , are showed in Figure 4.7. n_{max} is the number of particles in the cluster which has the maximum number of particles. Table 4.3 lists the value of n_c, n_{max} and p_{max} at time 0s, 5s, 7s, 8s, 15s, 19s in each column. During time 0s to 5s, the particles are concentrated around the robot in one cluster. However, starting from time 5s with increasing uncertainty, the number of clusters n_c goes up monotonously, and the percentage p_{max} goes down very quickly. The Create started first rotation at time 7s and finished it at time 8s. The curve of p_{max} stops going down during this time. The same case occurred at time 15s. When the robot gets close to the left down corner, the particles are spread around a large area. At this moment, p_{max} goes down to 0.185 which is lower than the threshold 20%. Therefore, the robot stops as it has been lost. This experiment illustrates the robot can understand the case that particles are spread across an increasingly large area when uncertainty grows as the robot moves.

(b) Global Localization

Global localization is performed in a 152.5x122cm field. Figure 4.8 illustrates our combined MCL-Clustering method in the experiment. Shown there is a sequence of particle sets during global localization of the robot Create with only bump sensor used.

Sensor measurements correspond to the detection of the walls placed around the field. Even though, Create demonstrates its ability localizing itself from scratch without knowledge of its starting location. The algorithm described in Table 3.4 is used to determine whether the robot has successfully localized itself. The path of robot is set to turn right 120° if the left bump sensor give a positive response and turn left 120° if the right bump sensor give a positive response, and move forward otherwise. The robot path setting is based on our previous work with Create. Different path setting will not affect the result of our method. In this particular experiment, the algorithm is initialized by drawing 2000 particles from a uniform probability density. But the number of particles is not kept fixed from beginning to the end. Each particle represents a possible location and orientation of the center of Create. Considering the radius of Create, between the area that particles can survive and the wall, there exists blank space. If the particle moves outside the area, it will be removed. After each time of resampling, we add two times particles around the particles that have high weight than the particles added around the particles that have low weight. With newly added particles, the particles size gets back to almost 2000. The two user defined variables are set as follows. The first one, criterion of the clustering θ, is set as the same as the radius of Create. That is the cover area of particles included in the same cluster will be a circle as the same size as Create. Another predefined variable, the threshold q that decides the maximum allowable value of p_{max}, is set to 75% of current number of total particles, which means if a cluster includes particles more than 75% of current number of total particles, the robot will believe the particles are concentrated and stop to indicate localization is successfully completed.

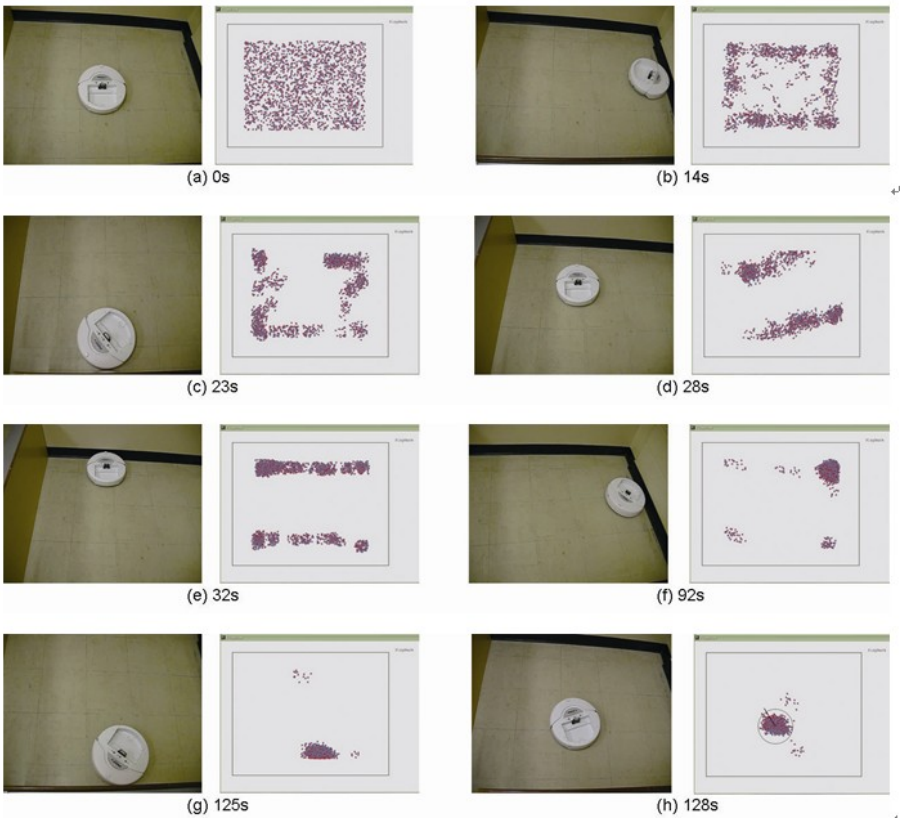

Figure 4.8: The robot true pose and distribution of particles during experiment global localization at time (a) 0s, (b) 14s, (c) 23s, (d) 28s, (e) 32s, (f) 92s, (g) 125s, (h) 128s.

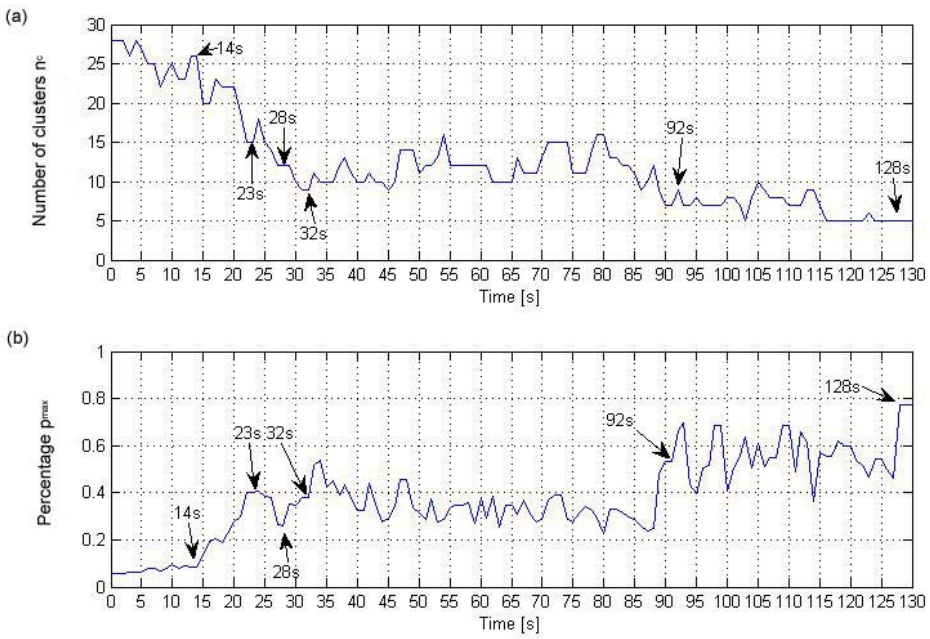

Figure 4.9: The plots of corresponding (a) number of clusters n_c and (b) the percentage of n_{max} in the current whole set of particles p_{max} .

The value of n_c, n_{max}, p_{max} at time 0s, 14s, 23s, 28s, 32s, 92s, 125s, 128s	0s	14s	23s	28s	32s	92s	125s	128s
Number of clusters: n_c	28	26	15	12	9	9	6	5
Number of particles in the cluster with the maximum number: n_{max}	111	23	87	349	34	1295	632	1109
Percentage of n_{max} in the current whole set of particles p_{max}	0.055	0.082	0.401	0.257	0.382	0.665	0.542	0.776

Table 4.4: The value of n_c, n_{max}, p_{max} at time 0s, 14s, 23s, 28s, 32s, 92s, 125s, 128s.

The distribution of particles at different times is analyzed below. Figure 4.8 shows the pose of robot and the distribution of particles at time 0s (a), 14s (b), 23s (c), 28s (d), 32s (e), 92s (f), 125s (g), and 128s (h). In each subfigure, the left picture shows the robot true pose and the right picture is the distribution of the particles at the same time. The plots of two corresponding characteristic variables, number of clusters n_c and the percentage of n_{max} in the current whole set of particles p_{max} , are showed in Figure 4.9. Table 4.4 lists the value of n_c, n_{max} and p_{max} at time 0s, 14s, 23s, 28s, 32s, 92s, 125s, 128s in each column. At time 14s in Figure 4.8(b), after the robot got its second detection of the wall, most particles are concentrated around the wall, but still a few noisy particles are at other area of the room. At this time, the percentage p_{max} starts going up and the cluster number n_c starts going down (Figure 4.9). The time 23s is the time robot got its third detection of the wall. Several regions with concentrated density of particles appeared (Figure 4.8(c)). The distribution of the particles corresponds to the local maxima of p_{max} at time 23s in Figure 4.9(b). Environment perception provides information for localization, so it tends to increase the robot's knowledge. On the other hand, the motion tends to reduce the knowledge due to the inherent noise in robot actuation. When the robot is moving before or after touching the wall, the motion uncertainty is increasing. After a short time moving without detection, at time 28s, a local minimum of p_{max} occurs. But when the robot had its fourth detection of the wall, the p_{max} goes up back (Figure 4.9(b)). After that, the robot goes through a period of fluctuating time. The detection of wall increases p_{max} and the motion introduces a loss of p_{max}. Until the time 92s, after the robot touched the wall at

upper right corner (Figure 4.8(f)), most particles are centered at the upper right corner which is very closed to the robot true location. The percentage p_{max} goes up to a new level. At time 125s, the robot got its last detection of wall (Figure 4.8(g)). Finally, at time 128s, p_{max} goes to 0.7755 which exceeds the predefined threshold 75%. So, the robot thought it had found itself and then stopped. The big circle in Figure 4.8(h) illustrates the location and orientation of the largest cluster in the particle set. It corresponds to the true robot pose very well. The Error distance is measured between the robot true position and the pose of representative of the largest cluster. The difference on coordinates x is 1.53cm, y is 0.65cm and on heading direction θ is 9.36°. In our reiterative running of the experiment, we found that MCL may fail a few times. It results our method indicate a false position to the robot. Details and handling of this problem will be discussed after the section of experimental results. The result of each time running our experiment are still robust to demonstrate the ability of our method telling robot whether the localization is successfully achieved.

4.2.2 Simulation Results

We have tested the performance of the implemented MCL-Clustering algorithm by simulation. Two cases are simulated. One is tracking without perception and the other is global localization. The simulated robot is a Roomba like robot which is low-cost robot with uncertainty of motion nearly 35% in distance and 25% in angle [35]. However, simulating a robot with such uncertainty of motion and using only tactile sensing of the robot we were able to implement and demonstrate successful global localization and the robot will stop after it thinks it localize well.

(a) Tracking without Sensor Readings

The simulation of tracking without sensor readings was performed by placing robot in a field of 624x528pixel. During tracking, the path of the simulated robot is not allowed to touch the wall. The robot drives in a rectangle which has length 424

pixels and width 180 pixels. At time 9s, the robot stopped as it found itself with high uncertainty in Figure 4.10(f). In this test, the number of particles is initialized as 300 and the threshold of the clustering θ is set to 60 pixels which is three times than the radius of the robot, 20 pixels. Another parameter need to be defined as priori is the threshold q of p_{max} that indicates the robot is lost. We set it to 20% of current total particles, which means if number of particles in the largest cluster is lower than 20% of the total particle size, the robot will believe it has lost.

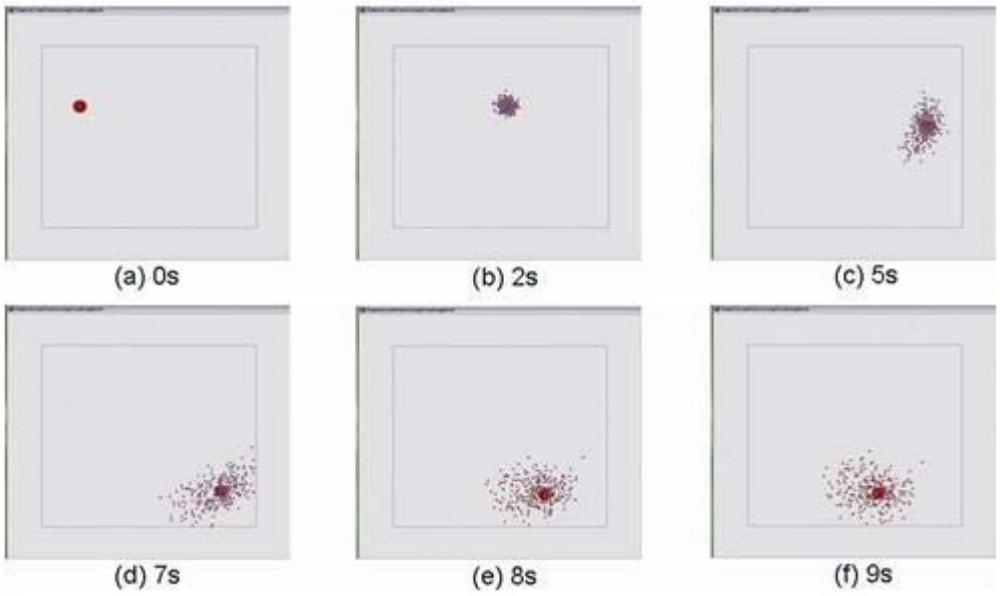

Figure 4.10: The simulated robot pose and distribution of particles during experiment tracking without sensor readings at time (a) 0s, (b) 2s, (c) 5s, (d) 7s, (e) 8s, (f) 9s.

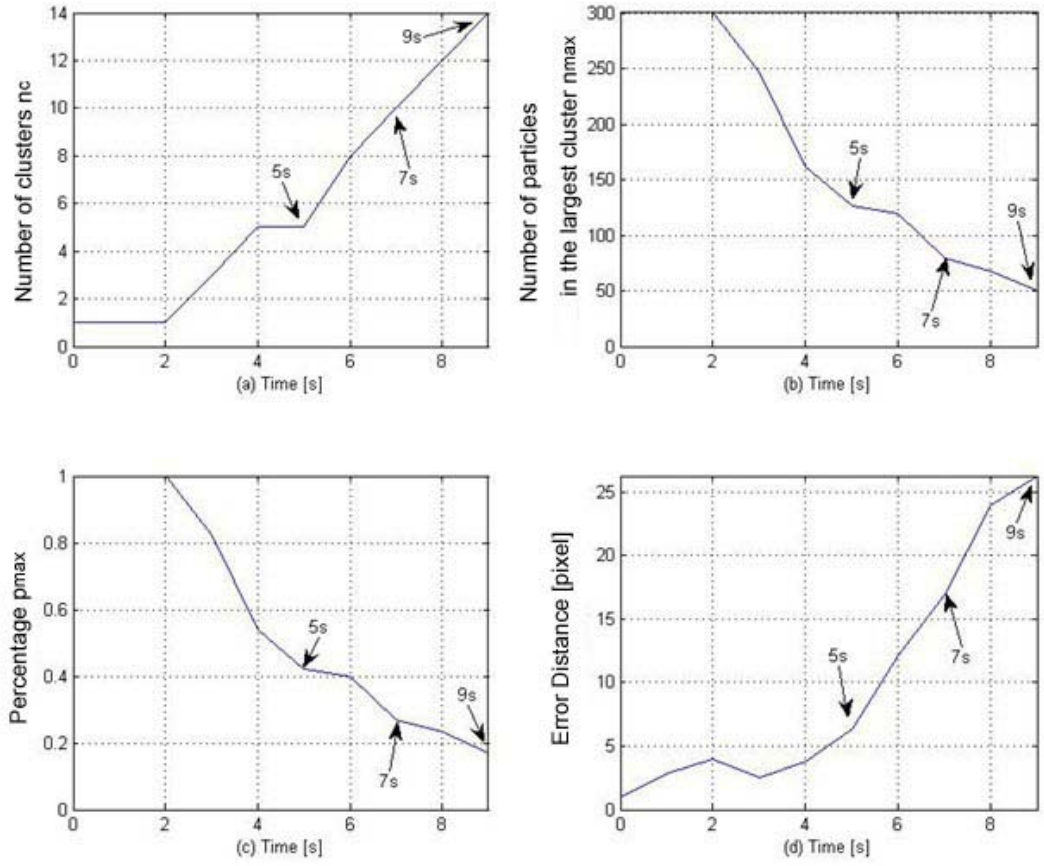

Figure 4.11: The plots of corresponding (a) number of clusters n_c , (b) the number of particles in the largest cluster n_{max}, (c) the percentage of n_{max} in the current whole set of particles p_{max}, and (d) the Error distance.

The value of n_c, n_{max}, p_{max} and error distance at time 0s, 2s, 5s, 7s, 8s, 9s	0s	2s	5s	7s	8s	9s
Number of clusters: n_c	1	1	5	10	12	14
Number of particles in the cluster with the maximum number: n_{max}	300	300	126	80	68	50
Percentage of n_{max} in the current whole set of particles p_{max}	1.0	1.0	0.42	0.268	0.233	0.171
Error distance [pixel]	1.0	2.88	6.3	16.9	23.9	26.3

Table 4.5: The value of n_c, n_{max}, p_{max} and error distance at time 0s, 2s, 5s, 7s, 8s, 9s.

Figure 4.10 shows the pose of robot and the distribution of particles at time (a) 0s, (b) 2s, (c) 5s, (d) 7s, (e) 8s, (f) 9s. In each subfigure, the large red cycle indicates the pose of the simulated robot. The plots of three corresponding characteristic variables, number of clusters n_c, the number of particles in the largest cluster n_{max}, the

percentage of n_{max} in the current whole set of particles p_{max} , and the error distance are showed in Figure 4.11. The Error distance is measured between the robot true position and the pose of representative of the largest cluster. As the simulated robot can offer the robot true position during localization, we can track the error distance in real time. Table 4.5 lists the value of n_c, n_{max}, p_{max} and error distance at time 0s, 2s, 5s, 7s, 8s, 9s in each column. At time 2s, the particles are aggregated around the robot in one cluster (Figure 4.10(b)). However, starting from time 2s with increased uncertainty, the number of clusters n_c and error distance goes up monotonously, but n_{max} and p_{max} goes down nearly linearly (Figure 4.11). Finally, p_{max} goes down to 0.171 which is lower than the threshold 20%. Therefore, the robot stops as it has been lost. During tracking, the increased error distance illustrates the uncertainty grows as the robot moves and the experimental result demonstrates the proposed method can offer a safety mechanism in case of the robot cannot get any sensor reading from outside world.

(b) Global Localization

Global localization is simulated in an area similar with the one used in [35] (Figure 4.12). It consists of an upper square 260x180pixel and a down square 420x330pixel. The robot moves with the initial position unknown and turns right 120° when it touches the wall. The same as above, the robot path setting is based on our previous work with Roomba. Different path setting will not affect the result of our method. In this test, the number of particles is initialized as 1000 and the criterion of the clustering θ is set as the same as the above trial, 60pixel. The threshold q is set to 80% of current total particles, which means if a cluster has particles over 80% of total, the robot will believe the particles are concentrated and stop to indicate localization is successfully completed.

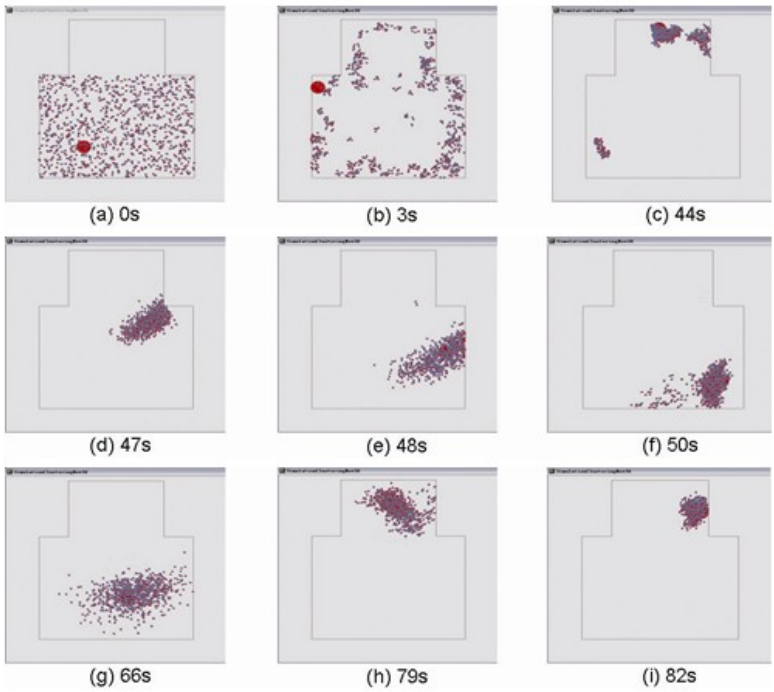

Figure 4.12: The simulated robot pose and distribution of particles during experiment global localization at time (a) 0s, (b) 3s, (c) 44s, (d) 47s, (e) 48s, (f) 50s, (g) 66s, (h) 79s, (i) 82s.

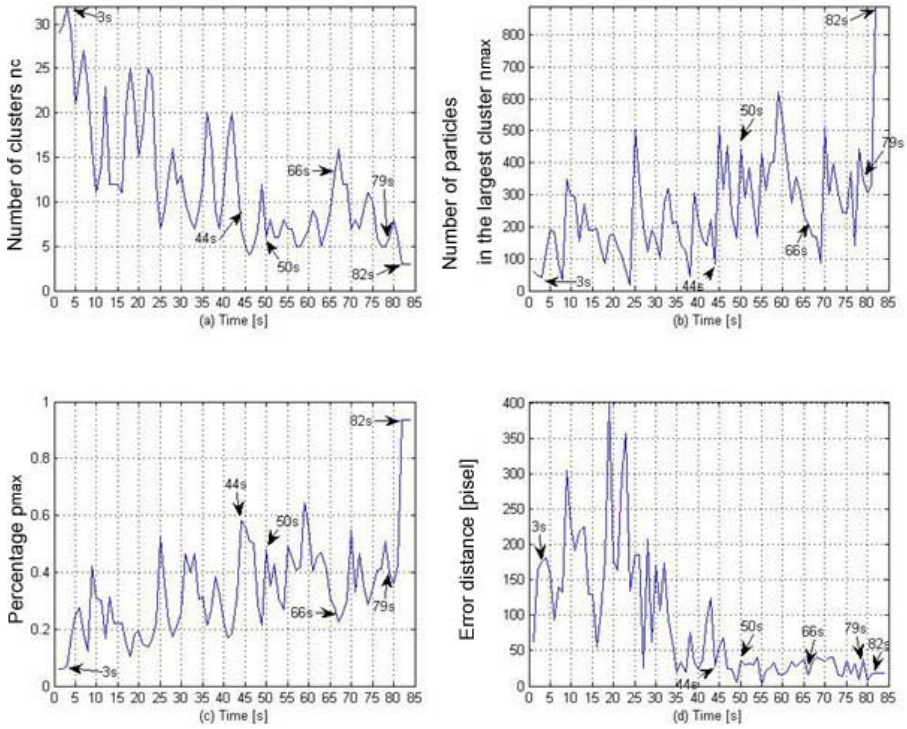

Figure 4.13: The plots of corresponding (a) number of clusters n_c , (b) the number of particles in the largest cluster n_{max}, (c) the percentage of n_{max} in the current whole set of particles p_{max}, and (d) the Error distance.

The value of n_c, n_{max}, p_{max} and error distance at time 0s, 3s, 44s, 47s, 48s, 50s, 66s, 79s, 82s.	0s	3s	44s	47s	48s	50s	66s	79s	82s
Number of clusters: n_c	29	32	7	5	7	6	13	6	3
Number of particles in the cluster with the maximum number: n_{max}	59	39	84	451	239	444	210	347	889
Percentage of n_{max} in the current whole set of particles p_{max}	0.059	0.072	0.583	0.502	0.288	0.478	0.277	0.401	0.937
Error distance [pixel]	61.97	175.3	28.62	23.9	24.52	35.31	14.8	38.11	17.63

Table 4.6: The value of n_c, n_{max}, p_{max} and error distance at time 0s, 3s, 44s, 47s, 48s, 50s, 66s, 79s, 82s.

Figure 4.12 shows the pose of robot and the distributions of particles at time (a) 0s, (b) 3s, (c) 44s, (d) 47s, (e) 48s, (f) 50s, (g) 66s, (h) 79s, (i) 82s. In each subfigure, the large red cycle indicates the pose of the simulated robot. The plots of three corresponding characteristic variables, number of clusters n_c, the number of particles in the largest cluster n_{max}, the percentage of n_{max} in the current whole set of particles p_{max} , and the error distance are showed in Figure 4.13. The Error distance is also measured between the robot true position and the pose of representative of the largest cluster. Table 4.6 lists the value of n_c, n_{max}, p_{max} and error distance at time 0s, 3s, 44s, 47s, 48s, 50s, 66s, 79s, 82s in each column. At 0s, particles are full filled the whole down area. The robot starts with the initial position unknown but assumed at down (Figure 4.12(a)). At 3s, the robot touches a wall and particles concentrate beside the wall too (Figure 4.12(b)). Because the robot only knows it is beside a wall, the particles are around all sides of the wall. This time, the number of clusters n_c is high and p_{max} is low. Starting from 44s, after the robot touches the upper wall, the most likely pose is very close to robot's position (Figure 4.12(c)). But, while the robot's moving, the motion uncertainty increases. Corresponding to Figure 4.13, during this time, n_c increases and p_{max} decreases. The trend of these two characteristic variables is the same after the robot touches a wall at 50s (Figure 4.12(f)). As the robot always gets a measurement when it touches a wall and then quickly lost while moving, the

plots of characteristic variables are like wave shown in Figure 4.13. At 66s (Figure 4.12(g)), the robot nearly lost itself again and p_{max} goes to a local minimum. Finally, at 82s, after the perceiving at the upper-right corner, the robot found where it is and stopped (Figure 4.12(i)). The final number of clusters n_c is 3 and p_{max} goes to 0.9368. The error distance at this moment is 17.63pixel on screen. It's very small compared to the environment and even smaller than the robot's radius 20pixel. The result of our simulation demonstrates the ability of our method telling robot whether the localization is successfully achieved.

4.3 Discussion of Orientation

In our proposed method, the proximate measure for clustering particles is calculated as the Euclidean distance between two particles. The distance includes tow dimension Cartesian coordinates (x, y). However, the robot pose is described by a vector of three dimensions (x, y, θ). The assumption that the particles in the same cluster will rarely have different orientation is made in our method. We will analyze the feasibility of this assumption in this section.

If we have two sets of particles at the same location, the orientation of one set corresponds to the direction of the robot and the orientation of other set points to another bearing. While the robot is moving forward, the two sets of particles will move according to its original heading. After the measure step of MCL, obviously the particle set with incorrect heading will be given low weight and gradually disappear. The resampling step is a probabilistic implementation of the Darwinian idea of survival of the fittest. Through many times selection, only the particles have the same orientation as the robot can survive. Our experiments support this observation as well.

In fact, if needed, the orientation can be the third dimension easily added to the current approach. All we need to do is changing the distance calculation expression.

Since the unit of the orientation and the location are different, the range of the orientation value should be adjusted to the same order as the location.

4.4 Limitation of MCL in the Experiment

An important question of MCL is how many particles should be used for a specific localization problem. Unfortunately, there is no perfect answer to this question. We just know the quality of the particle based representation increases with the number of particles. But, even with a large number of particles, it may happen that there are no particles in the vicinity of the robot correct position. This problem is known as the particles deprivation problem [26]. Due to this inherent fault, MCL may fail during localization. Especially in a symmetric environment, it is possible that the particles get together at the other side instead of the true position of the robot (Figure 4.14). If this case happens, it's dangerous for our proposed method as the robot will take the false position as its belief. It is the same as the robot is kidnapped. Our method cannot handle this inherent problem of MCL. Therefore, when using the combined MCL-Clustering to help robot know whether it is successfully localized, it is suggested to make sure the failure of MCL will not occur or the characteristic variables are only used as description of localization outcomes along with pictures in which the distribution of the particles are shown. In future work, we will try to check the measurement probability used in the algorithm Augmented_MCL[61] to verify whether the particles are concentrated around the robot true pose.

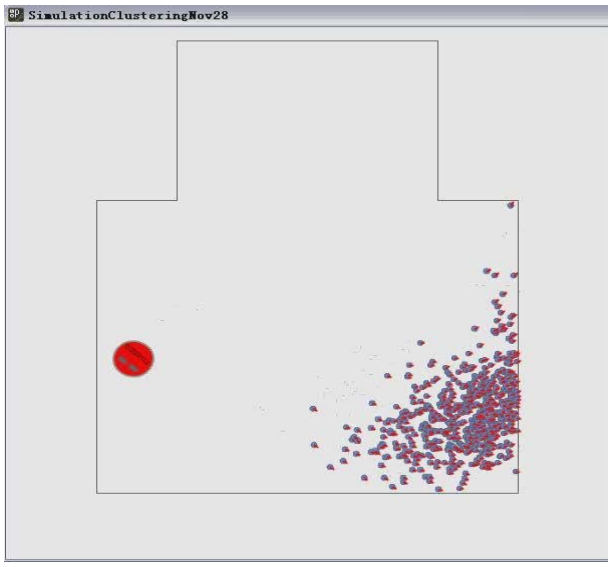

Figure 4.14: The case MCL fails.

Chapter 5
Conclusion and Future Work

5.1 Conclusion

Monte Carlo Localization estimates the robot pose by randomly drawing particles according to state transition probability and update state through measurement probability. Most existing approaches focus on the accuracy and efficiency of MCL. However, as a recursive state estimation algorithm, no stop condition is presented in the MCL algorithm when a robot has successfully localized itself. In this thesis, we develop a combined MCL-Clustering algorithm that brings consciousness to the robot being localized or lost. It notices the robot when the position of robot is successfully determined in global localization and helps the robot distinguish from different stages during localization time. By analyzing how many clusters the particle set has and how many particles are included in each cluster, the robot may know whether it is successfully localized or not instead of that a human being stares at the screen observing the distribution of the particles. After comparing many different clustering algorithms, we demonstrate the Basic Sequential Algorithm Schema is appropriate for clustering the particle set in real time. Otherwise, our method provides an approach to express localization outcomes in a numerical way. The outcome of localization can be explained in pictures along with the corresponding values of the characteristic variables of our method. Experimental results, performed in both real and simulated environments, show that our approach can notice the robot whether the uncertainty is increased over threshold during tracking, or tell the robot if the global localization is completed.

5.2 Future Work

In future work, we will help the robot verify the failure case as a kidnapped problem and recover from it. In addition, the topics about selection of proximity measure, selection of the clustering algorithm and active localization can be discussed from current approach as well.

Failure of MCL: One problem of our current method comes from the inherent limitation of MCL. It is known as the particle deprivation problem. Even with a large number of particles, it may happen that there are no particles around the correct state. Currently, our method cannot verify and make sure the final particles in the cluster are close to the robot true pose. If the particle deprivation occurs, the robot will believe it in an incorrect location. For future work, the robot may assume it at the final cluster place and then do more measurements to verify if the state showed by particles is correct.

Orientation: We have discussed including the orientation in Euclidean distance is not necessary in most cases. In our method, the orientation of the robot is not considered without loss correctness of the result while reducing the computation complexity. However, in a very strong symmetry environment (a square room), two particle set with different orientation can cross at one point, forming a cluster that has two headings. In this case, the orientation can be added as one dimension when computing the proximity measure or the robot can stay at this undetermined location and perceive the outside world in order to decide which direction is the heading.

Clustering algorithm: Considering the computing complexity, the Basic Sequential Algorithmic Scheme was chosen as the clustering algorithm. One limitation of BSAS is the clustering results may be affected by the order that the vectors are presented in. Although the impact of order is not prominent in our

experiment, the different presented ordering indeed leads to different clustering results. And as one of the clustering algorithms, the choice of threshold always depends on the platform of robot and the navigated environment. It only can be explained by experienced experts, no established uniform threshold for all cases.

Active localization: The localization described in our approach is totally passive. The information of clustered particles is merely exploited for telling the robot being localized or lost, not aimed at speeding up the robot localization. The robot is controlled through a preset movement method, and the robot's navigation does not facilitate the localization process. Therefore, one objective for future research is to control the robot so as to minimize the localization uncertainty by setting movements increasing the percentage p_{max} in our method.

Although limitation exists, our method does bring consciousness to the robot being localized or lost. By doing so, the robot can know the stages of localization, assisting the decision making of the robot.

Bibliography

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

[2] L. Feng, J. Borenstein and H.R. Everett. "Where am I?" Sensors and methods for autonomous mobile robot positioning. *Technical Report UM-MEAM-94-12, University of Michigan*, Ann Arbor, MI, 1994.

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

[4] J.S. Gutmann, W. Burgard, D. Fox and K. Konolige. An experimental comparison of localization methods. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*. 2002.

[5] R.G. Simmons and S. Koenig. Probabilistic robot navigation in partially observable environments. *In Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1995.

[6] P. Jensfelt and S. Kristensen. Active global localisation for a mobile robot using multiple hypothesis tracking. In *Proceedings of the IJCAI Workshop on Reasoning with Uncertainty in Robot Navigation*, pages 13–22, Stockholm, Sweden, 1999.

[7] P. Jensfelt and H.I. Christensen. Pose tracking using laser scanning and minimalistic environmental models. *IEEE Transactions on Robotics and Automation* 17:138-147, 2001.

[8] D. Caltabiano, G. Muscato and S. Sessa. A new global localization algorithm based feature extraction and particle filter. *Control and Automation, 2006. MED '06. 14th Mediterranean Conference on* 28-30, pp 1-6, June 2006.

[9] H. Fang, M. Yang and R. Yang. Ground Texture Matching based Global Localization for Intelligent Vehicles in Urban Environment. *Proc. of the 2007 IEEE Intelligent Vehicles Symposium*, Istanbul, Turkey, June 2007.

[10] J.J. Leonard and H.F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation* 7:376-382. 1991.

[11] S. Kwon, K.W. Yang and S. Park. An Effective Kalman Filter Localization Method for Mobile Robots. *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pp 1524-1529, Oct. 2006.

[12] T.H. Cong, Y.J. Kim and M. Lim. Hybrid Extended Kalman Filter-based localization with a highly accurate odometry model of a mobile robot. *Control, Automation and Systems, 2008. ICCAS 2008. International Conference on* 14-17 Oct. 2008 Page(s):738 – 743.

[13] F. Kong, Y. Chen, J. Xie, G. Zhang and Zude Zhou. Mobile Robot Localization Based on Extended Kalman Filter. *Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on* Volume 2, Page(s):9242 – 9246.

[14] C. Takenga, T. Peng and K. Kyamakya. Post-processing of Fingerprint Localization using Kalman Filter and Map-matching Techniques. *Advanced Communication Technology, The 9th International Conference on* Volume 3, 12-14 Feb. 2007 Page(s):2029 – 2034.

[15] A.C. Schultz and W. Adams. Continuous localization using evidence grids. Robotics and Automation, 1998. Proceedings. *1998 IEEE International Conference* on Volume 4, 16-20 May 1998 Page(s):2833 - 2839 vol.4.

[16] A Howard, M.J. Mataric and G.S. Sukhatme. Cooperative relative localization for mobile robot teams: An ego-centric approach. In *Proc. of The naval Reasearch Laboratory Workshop on Multi-Robot Systems*, Washington, D.C. 2003.

[17] W. Burgard, A.B. Cremers, D. Fox, D. Hahnel, G. Lakemeyer, D. Schulz, W. Steiner and S. Thrun. Experiences with an interactive museum tour-guide robot. *Artificial Intelligence* 114:3-55, 1999.

[18] T. Rofer and M. Jungel. Vision-Based Fast and Reactive Monte-Carlo Localization. *Proc. of the 2003 IEEE International Conference on Robotics & Automation,* Taipei, Taiwan, September, 2003.

[19] J. Liu, K. Yuan, W. Zou, and Q. Yang. Monte Carlo Multi-Robot Localization Based on Grid Cells and Characteristic particles. *Proceedings of the 2005 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, Monterey, California, USA, 24-28, July, 2005.

[20] X. Ma, X. Dai and W. Shang. Vision-based Extended Monte Carlo Localization for Mobile Robot. *Proc. of the IEEE International Conference on Mechatronics & Automation,* Niagara Falls, Canada, 2005.

[21] S. Lenser and M. Veloso. Resetting Localization for Poorly Modeled Mobile

Robots. *Proc. of the 2000 IEEE International Conference on Robotics & Automation*, San Francisco, CA, April 2000.

[22] R. Ueda, T. Fukase, Y. Kobayashi, T. Arai, H. Yuasa and J. Ota. Uniform Monte Carlo Localization - Fast and Robust Self-localization Method for Mobile Robots. *Proc. of the 2002 IEEE International Conference on Robotics & Automation* Washington, DC, May 2002.

[23] L. Armesto, J. Tornero and L. Domenech. Improving Self-Localization of Mobile Robots Based on Asynchronous Monte-Carlo Localization Method. *Emerging Technologies and Factory Automation, 2006. ETFA '06. IEEE Conference* on Volume , Issue , 20-22 Sept. 2006 Page(s):1028 – 1035.

[24] S. Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52-57, 2002.

[25] T. E. Kurt. 2006. Hacking Roomba. WILEY Press.

[26] S. Thrun, W. Burgard, and D. Fox. 2005. Probabilistic Robotics. MIT Press.

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

[28] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Ne-fian, and P. Mahoney. Stanley, the robot that won the DARPA Grand Challenge. *Journal of Field Robotics, Forthcoming*.

[29] B. Tribelhorn, and Z. Dodds. Envisioning the Roomba as AI Resource: A Classroom and Laboratory Evaluation. In *Proc. AAAI Spring Symposium*, 2007.

[30] A Gasparri, S. Panzieri, F. Pascucci and G. Ulivi, A Hybrid Active Global Localization Algorithm for Mobile Robots. *International Conference on Robotics and Automation*, Roma, Italy, 10-14 April 2007.

[31] D. Fox, W. Burgard, and S. Thrun. Active markov localization for mobile robots. *Robotics and Autonomous System*, vol. 25, pp. 195-207, 1998.

[32] D. Fox, Adapting the Sample Size in Particle Filters Through KLD-Sampling. *The International Journal of Robotics Research,* Vol. 22, No. 12, pp. 985-1003, 2003.

[33] Richard O. Duda, Peter E. Hart and David G. Stork, Pattern Classification. Wiley, 2001.

[34] S. Theodoridis and K. Koutroumbas, Pattern Recognition. Academic Press, 2006.

[35] B. Tribelhorn and Z. Dodds, Evaluating the Roomba: A low-cost, ubiquitous platform for robotics research and education. *IEEE International Conference on Robotics and Automation*, 2007.

[36] R. Barea, E. Lopez, L.M. Bergasa, S. Alvarez and M. Ocana. Detection Model in Collaborative Multi-Robot Monte Carlo Localization. *Proc. of the IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications (DIS' 06)*, 2006.

[37] P. Heinemann, J. Haase and A. Zell. A Combined Monte-Carlo Localization and Tracking Algorithm for RoboCup. *Proc. of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, China, 2006.

[38] G. Cen, H. Nakamoto, N. Matsuhira and I. Hagiwara. Effective Application of Monte Carlo Localization for Service Robot. *Proc. of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Diego, CA, USA, 2007.

[39] X. Zhang, X. Chen, J. Li and X. Li. Vision-based Monte Carlo – Kalman Localization in a Known Dynamic Environment. Control. *Automation, Robotics and Vision, 2006. ICARCV 06. 9th International Conference* on Volume, Issue, 5-8 Dec. 2006 Page(s):1 – 7.

[40] F. Abrate, B. Bona and M. Indri. Monte Carlo Localization of mini-rovers with low-cost IR sensors. *Advanced intelligent mechatronics, 2007 ieee/asme international conference* on Volume, Issue, 4-7 Sept. 2007 Page(s):1 – 6.

[41] C. Kwok, D. Fox and M. Meila. Real-time Particle Filters. *In Advances in Neural Information Processing Systems* 15, 2002.

[42] A R. Vahdat, N. N. Ashrafoddin and S. S. Ghidary. Mobile Robot Global Localization using Differential Evolution and Particle Swarm Optimization. *IEEE Congress on Evolutionary Computation 2007*: 1527-1534.

[43] T. Kwon, J. Yang, J. Song and W. Chung. Efficiency Improvement in Monte Carlo Localization through Topological Information. *Proc. of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, China, 2006.

[44] A Rauber, J. Paralic and E. Pampalk. Empirical Evaluation of Clustering Algorithms. *Journal of Information and Organizational Sciences*, vol. 24, pp. 2000, 2000.

[45] J. R. Searle. Minds, brains, and programs. *Behavioral and Brain Science* 3(3): 417-457, 1980.

[46] B. J. MacLennan. Consciousness in robots: the hard problem and some less hard problems. *Robot and Human Communication (ROMAN), IEEE International Workshop on, pp. 434- 439, Aug, 2005.*

[47] M. S. Lavine, D. Voss and R. Coontz. A Robotic Future. *Magazine Science*, Vol. 318, pp. 1083, 2007.

[48] F. L. Lewis, M. Fitzgerald and K. Liu. Robotics. *ACM Computing Surveys*, Vol. 28, No. 1, March 1996.

[49] E. Garcia, M.A. Jimenez, P.G. De Santos and M. Armada. The evolution of robotics research. *Robotics & Automation Magazine*, IEEE Volume 14, Issue 1, March 2007 Page(s):90 – 103.

[50] S. Russell and P. Norvig. Artificial Intelligence: A Modern Approach. Englewood Cliffs, NJ: Prentice Hall, 2002.

[51] M.H. Degroot. Probability and Statistics. Reading, MA: Addison-Wesley. 1975.

[52] K. Subrahmaniam. A Primer In Probability. New York, NY: M. Dekker. 1979.

[53] G.C. Casella and R.L. Berger. Statistic Inference. Pacific Grove, CA: Wadsworth& Brooks. 1990.

[54] M.A. Tanner. Tools for Statistical Inference. New York: Springer Verlag. 3rd edition. 1996.

[55] L. Devroye, L. Gyorfi and G. Lugosi. A Probabislistic Theory of Pattern Recognition. New York, NY: Springer-Verlag. 1996.

[56] R.O. Duda, P.E. Hart and D. Stock. Pattern classification and scene analysis (2nd edition). New York: John Wiley and Sons. 2000.

[57] W. Burgard, A. Derr, D. Fox, and A.B. Cremers. Integrating global position estimation and position tracking for mobile robots: The dynamic markov localization approach. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'98)*, 1998.

[58] W. Burgard, D. Fox, D. Hennig, and T. Schmidt. Estimating the absolute position of a mobile robot using position probability grids. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Menlo Park, August 1996. AAAI, AAAI Press/MIT Press.

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

[60] D. Fox, W. Burgard, H. Kruppa, and S. Thrun. A probabilistic approach to collaborative multi-robot localization. *Autonomous Robots on Heterogeneous Multi-Robot System*, vol. 8, no. 3, pp. 325–344, June 2000.

[61] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. Monte Carlo Localization: Efficient Position Estimation for Mobile Robots. *Proc. of the Sixteenth National Conference on Artificial Intelligence* (AAAI'99). 1999.

[62] R. van der Merwe. Sigma-Point Kalman Filters for Probabilistic Inference in Dynamic State-Space Models. PhD thesis, OGI School of Science & Engineering. 2004.

[63] I.J. Cox, and J.J Leonard. Modeling a dynamic environment using a Bayesian multiple hypothesis approach. *Artificial Intelligence* 66:311-344. 1994.

[64] A Nourbakhsh, R. Powers and S. Birchfield. DERVISH an office-navigating robot. *AI Magazine* 16. 1995.

[65] D.W. Scott. Multivariate density estimation: theory, practice, and visualization. John Wiley and Sons, Inc. 1992.

[66] A Doucet, J.F.G de Freitas and N.J. Gordon. Sequential Monte Carlo Methods In Practice. New York: Springer Verlag. 2001.

[67] R.Y. Rubinstein. Simulation and the Monte Carlo Method. John Wiley and Sons, Inc. 1981.

[68] B. Everitt, S. Landau and M. Leesse Cluster Analysis. Arnold, 2001.

[69] J.J Amador. Sequential clustering by statistical methodology. *Pattern Recognition Letters*, Vol. 26, pp. 2152-2163, 2005.

[70] P. Trahanias and E. Scordalakis. An efficient sequential clustering method. *Pattern Recognition*, Vol. 22(4), pp. 449-453, 1989.

[71] A Juan and E. Vidal. Comparison of four initialization techniques for the k-medians clustering algorithm. *Proc. of Joint IAPR International Workshops SSPR2000 and SPR2000, Lecture Notes in Computer Science,* Vol 1876, pp. 842-852, Springer Verlag, Alacant, Sept. 2000.

[72] P. Berkhin. Survey of clustering data mining techniques. *Technical Report*, Accrue Software Inc. 2002.

[73] R.L. Mantaras and J. Aguilar-Martin. Self-learning pattern classification using a sequential clustering technique. *Pattern Recognition*, Vol. 18(3/4), pp. 271-277, 1985.

[74] I.J. Cox and G.T. Wilfong. Autonomous Robot Vehicles. Springer Verlag. 1990.

[75] R. Murphy. Introduction to AI Robotics. Cambridge, MA: MIT Press. 2000.

[76] D. KortenKamp, R.P. Bonasso and R. Murphy. Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems. Cambridge, MA: MIT/AAAI Press. 1998.

[77] S. Lee, D. Cho, W. Chung, J.H. Lim and C.U. Kang. Feature Based Map Building Using Sparse Sonar Data. *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference* on Volume , Issue , 2-6 Aug. 2005 Page(s): 1648 – 1652

[78] URL: http://www.processing.org/

[79] URL: http://www.darpa.mil/grandchallenge05/

[80] URL: http://www.irobot.com/

VITA AUCTORIS

NAME	Jingxi Chen
PLACE OF BIRTH	Changsha, China
YEAR OF BIRTH	1984
EDUCATION	School of Software Engineering Huazhong University of Science and Technology Wuhan, China 2003 – 2007 B.Eng.
	School of Computer Science University of Windsor Windsor, Ontario, Canada 2007 – 2009 M. Sc.