Self-adaptive Markov Localization for Single-Robot and Multi-Robot Systems

Lei Zhang

ACADÉMIE DE MONTPELLIER

UNIVERSITÉ MONTPELLIER II

- SCIENCES ET TECHNIQUES DU LANGUEDOC -

THÈSE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ MONTPELLIER II

Discipline : Génie Informatique, Automatique et Traitement du Signal
Formation Doctorale : Systèmes Automatiques et Microélectroniques
École Doctorale : Information, Structures et Systèmes

présentée et soutenue publiquement

par

Lei ZHANG

le 15 janvier 2010

Titre :

Self-Adaptive Markov Localization for Single-Robot and Multi-Robot Systems

---

JURY :

Mr.	Michel DEVY	Directeur de recherche CNRS au LAAS	Rapporteur
Mr.	Luc JAULIN	Professeur, ENSIETA Brest	Rapporteur
Mr.	René ZAPATA	Professeur, Université Montpellier II	Directeur de Thèse
Mr.	Bruno JOUVENCEL	Professeur, Université Montpellier II	Examinateur
Mr.	Nacim RAMDANI	Maître de conférences, Université Paris 12	Examinateur
Mr.	Lionel LAPIERRE	Maître de conférences, Université Montpellier II	Examinateur
Mr.	Erwann LAVAREC	PDG, WANY S.A.	Examinateur

Contents

List of Figures

List of Algorithms

List of Tables

CHAPTER 1

INTRODUCTION

Please make your mind to imagine such a scenario: when you have completed the day's work, you are ready to go back home. How do you go home? Perhaps you will answer: "Just walk back home.". However, the problem is not as simple as you think. Before and during you go home, you must solve three problems. Firstly, you must know where you are; then, you must know where your home is; finally, you should plan a route to go home. For human beings, the abilities to solve these problems are part of our instinct. Sometimes, it is so simple that we ignore the existence of this talent. However, if robots are to achieve human-like autonomy, they must possess these capabilities. It is not an easy task to make robots have such capabilities. Because of this, Leonard *et al.* summarize the problem of autonomous navigation into answering three questions [Leonard91].

- **Where am I?** This question is related to how to find out the whereabouts of a robot in a given environment, which is known as *localization*. Localization is defined as the problem of determining the pose of a robot given a map of the environment and sensors data [Burgard97c, Fox98a, Fox99b]. Usually, the mobile robot pose comprises its $x - y$ coordinates and its orientation.
- **Where am I going?** and **How should I get there?** These two questions point out how to specify a goal and how to plan a path to achieve this goal. In mobile robotics, they are often involved in *goal recognition*, *obstacle avoidance* [Cacitti01, Zapata04, Zapata05], *path planning and following* [Lapierre07a, Lapierre07b] and *motion planning* [Fort-Piat97, Laumond98], etc.

Thus, effective localization is a fundamental prerequisite for achieving autonomous mobile robot navigation. This thesis focuses on finding out a robust and reliable localization approach, which is also required to answer the remaining two questions.

1.1 LOCALIZATION PROBLEMS

According to the type of knowledge that is available initially and at run-time and the difficulty of finding a solution, localization problem can be divided into

three sub-problems: position tracking, global localization and the kidnapped robot problem [Cox90, Roumeliotis00, Thrun00a, Thrun05].

- Position tracking assumes that the robot knows its initial pose [Schiele94, Weiss94]. During its motions, the robot can keep track of its movement to maintain a precise estimate of its pose by accommodating the relatively small noise in a known environment.

- More challenging is the global localization problem [Thrun00a, Milstein02]. In this case, the robot does not know its initial pose, thus it has to determine its pose in the following process only with control data and sensors data. Once the robot determines its global position, the process continues as a position tracking problem. To solve the initial localization problem, Jaulin *et al.* propose a guaranteed Outlier Minimal Number Estimator (OMNE), which is based on set inversion via interval analysis [Jaulin02]. They apply this algorithm to the initial localization of an actual robot in a partially known 2D environment.

- The kidnapped robot problem appears when a well-localized robot is teleported to some other place without being told [Thrun00a, Thrun00b, Thrun05]. Robot kidnapping can be caused by many factors. Generally, we summarize the kidnapped robot problem into two categories: real kidnapping and localization failures.

  – The first one occurs when the robot is really kidnapped. For example, someone takes the robot to other place; or an accident causes the robot to drastically drift.

  – Localization failures can make the robot think itself to be kidnapped. For example, when the robot moves into a incomplete part of the map, unmodeled objects can cause the robot to think that it is kidnapped. It can also bring about kidnapping when the crowd passes next to the robot. There are many other reasons that can lead to localization failures, such as mechanical failures, sensor faults and wheel slip [Stéphant04, Stéphant07].

  In practice, real kidnapping is rare; however kidnapping is often used to test the ability of a localization algorithm to recover from global localization failures. This problem is the hardest of the three localization sub-problems. Difficulties come from two sources: one is how to determine the occurrence of kidnapping; the other is how to recover from kidnapping. To some extent, to recover from kidnapping can be considered as estimating globally the robot's pose once again if the robot finds the occurrence of kidnapping.

Another classification is based on the number of robots involved in localization. Localization problems can be classified into single-robot localization and multi-robot localization.

- Single-robot localization is the most fundamental localization problem, in which just a single robot is involved. Sometimes, several robots independent localization is also considered as single-robot localization, since there is no cooperation among them. The main advantage of single-robot localization lies in its simplicity as it need not consider the issues such as communication, data exchange and data fusion.

- Multi-Robot Localization emphasizes cooperative localization and allows robots to exchange and share information through communication. Exchanging position information in teams of robots can increase the robustness and efficiency of the localization algorithm. Sharing sensor information among multiple robots can lower the costs of the entire system. Sharing information among different sensor platforms among multiple robots can make robot teams be able to adapt to a more complex environment.

Localization problems suppose that the robot is given a pre-known map. In other words, localization problems always arise in known environments. However, the work space of a robot is not only limited to the known environment. The robot should be capable of working in an unacquainted environment, too. When the robot navigates in the strange place, it should have the ability to acquire a map of its environment while simultaneously localizing itself relative to this map [Choset01, Dissanayake01, Thrun05, Howard06]. This problem is known as the Simultaneous Localization And Mapping problem, abbreviated as SLAM. It is also called Concurrent Mapping and Localization (CML). This dissertation does not involve the SLAM problem, however to solve the SLAM problem efficiently will be considered as future works of this thesis.

1.2 LOCALIZATION APPROACHES

Since effective localization is the basis of achieving other tasks, considerable research effort has been directed to this problem and a variety of approaches have been devised to solve this problem.

Existing localization techniques for mobile robots can be roughly classified into two categories: relative localization, including dead-reckoning methods, and absolute localization, including active beacons, landmarks and map-based positioning [Borenstein96, Zhou07]. Most implementations of these approaches are based on deterministic forms; however a robot's world is filled with uncertainties. Uncertainties can be summarized as following aspects [Thrun05].

- The robot's environments are unpredictable.

- The robot's hardware is subject to noise.

- The robot's software is approximate.

To deal with so many uncertainties, probabilistic approaches are undoubtedly promising candidates to provide a comprehensive and real-time solution of the

mobile robot localization problem. Instead of calculating a single pose of the robot, probabilistic localization represents the robot's pose by probability distributions over a whole state space. Among probabilistic localization approaches, Markov localization based on the Bayes filter is the most popular one [Thrun05]. In the Markov localization framework, the localization problem is described as estimating a posterior belief of the robot's pose at present moment conditioned on the whole history of available data and a given map. It can be stated in equation as follows:

$$bel(s_t) = p\left(s_t \,|z_{0:t}, u_{1:t}, m\right) \tag{1.1}$$

where s_t is robot's pose at time t, which is composed by its two-dimensional planar coordinates and its orientation. The belief function $bel(s_t)$ represents the density of probability of the pose s_t. The term $z_{0:t}$ represents all the exteroceptive measurements from time $\tau = 0$ to $\tau = t$; $u_{1:t}$ represents control data from time $\tau = 1$ to $\tau = t$ and m denotes the given map.

Markov localization addresses the position tracking problem, the global localization problem and the kidnapped robot problem in static environments [Thrun05]. Extended Kalman Filter (EKF) localization, grid localization and Monte Carlo localization (MCL) are three most common Markov localization algorithms. EKF localization focuses on solving the position tracking problem [Kalman60, Grewal93, Gasparri06]. Grid localization and MCL can deal with both the position tracking problem and the global localization problem [Burgard96, Burgard97b, Dellaert99, Fox99a, Thrun00c, Thrun05]. In order to solve the kidnapped robot problem, some improved MCL algorithms are proposed, such as the Augmented_MCL algorithm [Thrun05] and the Mixture MCL algorithm [Thrun00d, Thrun00b, Thrun05].

Another interesting localization approach based on interval analysis is proposed in [Kieffer00, Jaulin01, Seignez06, Jaulin09], which is also robust to accommodate all sources of uncertainty listed above. Interval analysis has been applied to Bayesian estimation [Jaulin06], which may provide an inspiration to integrate the two theories for solving localization problems.

1.3 OBJECTIVE

The objective of this dissertation is to find out a general solution for the position tracking problem, the global localization problem and the kidnapped robot problem by using simple range finders. Moreover, this algorithm should be capable of being implemented to both single-robot and multi-robot systems. Thus, this thesis presents four contributions, which are summarized as follows.

- We devise a simulator that has ability to drive both the simulated robot and the real robot. EKF localization, grid localization and MCL are studied intuitively through simulations and extensive simulation results are given in this thesis. Through analyzing and comparing simulation results, we discuss advantages and disadvantages of each algorithm.

- In order to solve all three localization problems, we propose an improved

Monte Carlo localization algorithm with self-adaptive samples, named as Self-Adaptive Monte Carlo Localization (SAMCL). This algorithm employs a pre-caching technique to reduce the on-line computational burden. Thrun *et al.* [Thrun05] use this technique to reduce costs of computing for beam-based models in the ray casting operation. Our pre-caching technique decomposes the state space into two types of grids. The first one is a three-dimensional grid that includes the planar coordinates and the orientation of the robot, denoted as G_{3D}. It is used to reduce the on-line computational burden of MCL. The second grid is a two dimensional "energy" grid, denoted as G_E. It is used to calculate the Similar Energy Region (SER) which is a subset of G_E. Its elements are these grid cells whose energy is similar to robot's energy. SER provides potential information of robot's position; thus, sampling in SER is more efficient than sampling randomly in the whole map. Finally, SAMCL can solve position tracking, global localization and the kidnapped robot problem together thanks to self-adaptive samples. Self-adaptive samples in this thesis are different from the KLD-Sampling algorithm proposed in [Fox03a, Thrun05]. The KLD-Sampling algorithm employs the sample set that has an adaptive size to increase the efficiency of particle filters. Our self-adaptive sample set has a fixed size, thus it does not lead to the expansion of the particle set. This sample set can automatically divide itself into a global sample set and a local sample set according to different situations, such as when the robot is kidnapped or fails to localize globally. Local samples are used to track the robot's pose, while global samples are distributed in SER and used to find the new position of the robot.

- The SAMCL algorithm is extended to handle the multi-robot localization problem through a Position Mapping (PM) algorithm. This algorithm reduces furthest the communication delay and the computational complexity, since it only synchronizes one location and one belief instead of information of the whole particle set from every other robot. That allows one robot to cooperate with multiple robots at the same time rather than one robot.

- The SAMCL algorithm is tested on a Pioneer 3-DX mobile robot only equipped with sixteen ultrasonic range finders in a real office environment. The validity and the efficiency of the algorithm are demonstrated by experiments carried out with different intentions. Extensive experiment results and comparisons are also given in this thesis.

1.4 THESIS OUTLINE

Overall, the organization of this dissertation can be summarized into two parts. Part 1, including Chapter 2 and 3, presents foundations of Markov localization. Part 2, including Chapter 4 to 7, introduces the main contributions of this thesis.

Chapter 2 addresses motion models, perception models and maps, the three important known components for solving localization problems. We establish both

deterministic models and probabilistic models for robot motion and robot perception, respectively. Moreover, we introduce two representations of maps: geometric maps and topological maps. These models will be implemented to concrete localization algorithms.

Chapter 3 starts with a review of the basic theory of Bayes filters, and then we introduce the Markov localization algorithm based on them. Finally, we present respectively three concrete localization algorithms: EKF localization, grid localization and MCL.

In Chapter 4, we start with the description of the simulator. Then EKF localization, grid localization and MCL are tested by simulations. Tests are executed along with two directions: position tracking and global localization. Simulation results are analyzed and compared, in order to study advantages and disadvantages of each algorithm.

Chapter 5 presents the Self-Adaptive Monte Carlo Localization (SAMCL) algorithm. We first describe the theory of the SAMCL algorithm, and then it is evaluated by simulations. The SAMCL algorithm provides a general solution for the position tracking problem, the global localization problem and the kidnapped robot problem, thus simulations focus on testing its performance of solving the three problems. Moreover, computational efficiency of SAMCL and regular MCL are compared by simulations.

Chapter 6 discusses the multi-robot localization problem. This problem can be solved by the Position Mapping (PM) algorithm, which can be integrated into the SAMCL algorithm as an extension. The validity and the efficiency of the PM algorithm are tested by simulations carried out on two individual computers, which can communicate by wireless network (WiFi).

In Chapter 7, we address the implementation issues. In order to demonstrate the validity and the efficiency of the SAMCL algorithm, we test it with a Pioneer 3-DX mobile robot at the first floor of our laboratory. Moreover, we compare sampling in SER with sampling randomly in the capability of recovering from kidnapping.

Finally, we conclude with a summary of our work and suggest future extensions in Chapter 8.

Part I

Foundations of Markov Localization

CHAPTER 2

MOTION MODELS, PERCEPTION MODELS AND MAPS

Contents

2.1 INTRODUCTION

This chapter describes the three known components for solving the localization problem: motion models, perception models and maps. Motion models describe the robot's state transitions, which specify how a robot changes from one pose to another pose under motion controls. Our exposition focuses on Wheeled Mobile Robots (WMRs) kinematics for robots operating in a planar environment. Perception models are as important as motion models in the mobile robot localization problem. They describe the formation process by which sensor measurements are generated in the physical world [Thrun05]. In order to express the process of generating measurements, a specification of the environment is necessary. Maps provide a concrete or abstract description of the physical world. Both deterministic

models and probabilistic models are introduced for motion models and perception models. Generally, probabilistic models are generated by adding noise variables to deterministic forms. The probabilistic models are provided for implementing the filter algorithm that will be described in the following chapters.

2.2 MOTION MODELS

Robot kinematics is the central topic of this section. Here, we will introduce the deterministic form and the probabilistic form, respectively. The probabilistic form is derived from the deterministic form. Our works exclusively address in Wheeled Mobile Robots (WMRs). Other systems such as legged robots, are not covered in this section.

2.2.1 Deterministic motion models

In this section we introduce the deterministic form of motion models. For the deterministic form, we assume that all the motion controls are error-free.

2.2.1.1 Representing robot pose

A WMR is modeled as a planar rigid body that rides on an arbitrary number of wheels [Cox90, Siegwart04, Solutions07]. The total dimensionality of the WMR on the plane is summarized by three variables, referred to as a pose (or position). These three variables are the two-dimensional planar coordinates and the angular orientation relative to a global coordinate frame (see Figure 2.1). The former is denoted as x and y and the latter as θ. Thus, the pose of a robot is described by the following vector. Note the use of the subscript G to clarify this pose in the global reference frame.

$$s_G = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \tag{2.1}$$

Specially, pose without orientation will be called location [Thrun05]. The location of a robot is described by two-dimensional vectors. The concept of location will be used in the following chapter to distinguish from the concept of pose.

$$l_G = \begin{pmatrix} x \\ y \end{pmatrix} \tag{2.2}$$

Robot motion can be described as the robot's pose transition according to its control data (see Figure 2.2). This process is modeled as:

$$s_t = f(u_t, s_{t-1}) \tag{2.3}$$

where s_{t-1} and s_t denote the robot's pose at time $t-1$ and t, and u_t denotes the motion control executed on s_{t-1}. Control data are often obtained from velocity commands or extracted from odometry. Thus, we introduce robot motion in two

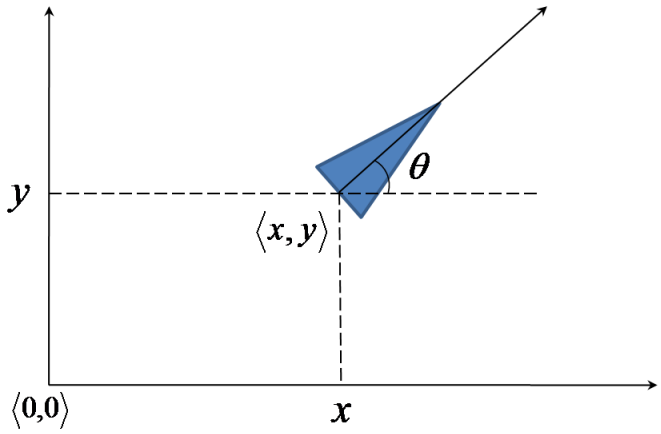


Figure 2.1: A robot's pose shown in the global reference frame.

motion models: velocity motion model and odometry motion model. In practice, odometry models tend to be more accurate than velocity models [Thrun05]. However, odometry is only available in retrospect, after the robot moved. It cannot be used for these planning algorithms that have to predict the effects of motion. Therefore, odometry models are usually applied for estimation, whereas velocity models are used for motion planning [Thrun05].

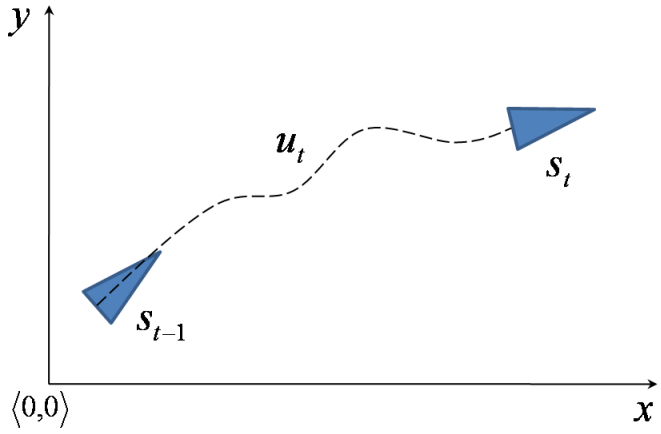


Figure 2.2: The robot's pose transition shown in the deterministic form.

2.2.1.2 Velocity motion model

The velocity motion model assumes that we can control a robot through two velocities, a translational and a rotational velocity [Thrun05]. The translational velocity at time t is denoted as v_t, and the rotational velocity is denoted as ω_t. Hence, the control vector u_t at time t is given by:

$$u_t = \begin{pmatrix} v_t \\ \omega_t \end{pmatrix} \tag{2.4}$$

Thrun *et al.* present a kinematic model of WMRs. The detail derivation of this expression is shown in [Thrun05].

$$\begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix} = \begin{pmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{pmatrix} + \begin{pmatrix} -\frac{v_t}{\omega_t} sin(\theta_{t-1}) + \frac{v_t}{\omega_t} sin(\theta_{t-1} + \omega_t \Delta t) \\ \frac{v_t}{\omega_t} cos(\theta_{t-1}) - \frac{v_t}{\omega_t} cos(\theta_{t-1} + \omega_t \Delta t) \\ \omega_t \Delta t \end{pmatrix} \tag{2.5}$$

where Δt is a fixed small enough sampling interval.

In [Laumond98], an alternative continuous kinematic model is introduced, which is small-time controllable from everywhere.

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} cos(\theta) \\ sin(\theta) \\ 0 \end{pmatrix} v_t + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \omega_t \tag{2.6}$$

where $(\dot{x}, \dot{y}, \dot{\theta})^T$ describes an instantaneous state of the robot. Thus a successor pose $s_t = (x_t, y_t, \theta_t)^T$ according to the predecessor pose $s_{t-1} = (x_{t-1}, y_{t-1}, \theta_{t-1})^T$ in a small enough sampling time $\Delta t \in (t-1, t]$ can be calculated as:

$$\begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix} = \begin{pmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{pmatrix} + \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} \Delta t \tag{2.7}$$

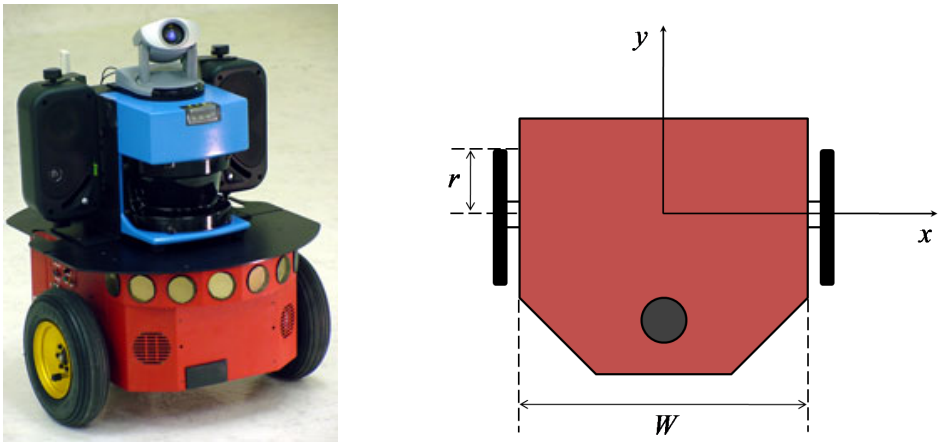
This equation is a first-order approximation of the continuous model (Equation 2.6).

In fact, not all WMR systems are able to impose control on the translational velocity and the rotational velocity directly, hence some control transforms are necessary. Let us consider two most common systems: the differentially driven WMR and the car-type WMR.

- **Differentially driven WMRs.** A differentially driven WMR consists of two driving wheels and one or two castor wheels. In a differentially driven WMR, the acceleration of each driving wheel is controlled by an independent motor. The stability of the platform is insured by castors [Laumond98]. The reference point of the robot is the midpoint of the axle of the two wheels. The motion of a differentially driven WMR is achieved by controlling the spinning speed of each wheel,ϖ_l and ϖ_r. If $\varpi_l = \varpi_r$, the robot moves in straight-line. The turning is achieved when $\varpi_l > \varpi_r$ or $\varpi_l < \varpi_r$. Figure 2.3 represents a general arrangement of a differentially driven WMR. W designates the distance between the driving wheels and r is the radius of the wheel. We calculate the translational velocity v_t and the rotational velocity ω_t as follows [Laumond98, Siegwart04]:

$$v_t = \frac{r}{2}(\varpi_{r,t} + \varpi_{l,t}) \tag{2.8}$$

$$\omega_t = \frac{r}{W}(\varpi_{r,t} - \varpi_{l,t}) \tag{2.9}$$



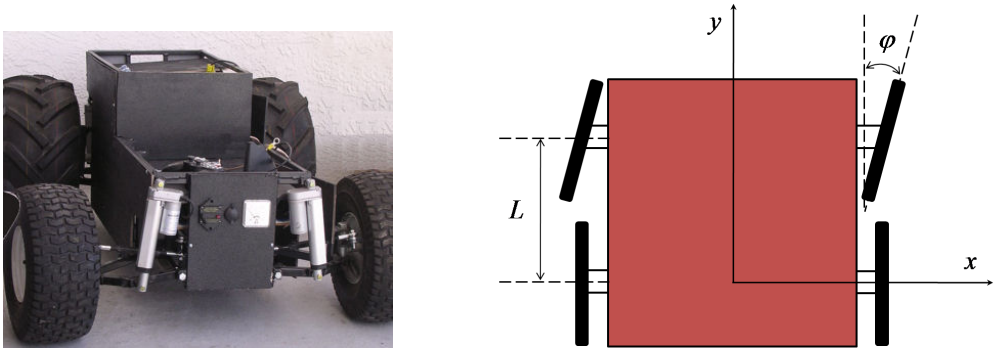
(a) (b)

Figure 2.3: Differentially driven WMR.

- **Car-like WMRs.** A Car-like WMR has two controls: the accelerator and the steering wheel [Laumond98, Stéphant04, Stéphant07, Solutions07]. The driving wheels are either front wheels or rear wheels. The reference point of the robot is the midpoint of the rear axle. The distance between both front and rear axles is L. We denote ς_t as the speed of the front wheels of the car and φ as the angle between the front wheels and the main direction of the car (see Figure 2.4). The transforms of control values are shown as follows:

$$v_t = \varsigma_t cos(\varphi) \tag{2.10}$$

$$\omega_t = \frac{\varsigma_t}{L} sin(\varphi) \tag{2.11}$$



(a) (b)

Figure 2.4: Car-like WMR.

2.2.1.3 Odometry motion model

The odometry motion model uses the odometry measurements as the basis for calculating the robot's motion over time [Thrun05]. Odometry is commonly obtained by integrating wheel encoder information. Suppose the robot moves from the pose s_{t-1} to the pose s_t in the time interval $(t-1, t]$. The odometry reports back a related motion from the pose $\bar{s}_{t-1}$ to the pose $\bar{s}_t$. Here the bar indicates that the motion information is obtained from odometry measurements. The motion control can be extracted from the relative difference between $\bar{s}_{t-1}$ and $\bar{s}_t$.

$$u_t = \begin{pmatrix} \bar{s}_{t-1} \\ \bar{s}_t \end{pmatrix} \tag{2.12}$$

We assume that the difference is virtually generated by three motions: a rotation, followed by a straight line motion (translation) and another rotation [Thrun05]. This assumption is not unique. We can even think that the difference of odometry is generated by: a translation along $x-$axis, another translation along $y-$axis and a rotation. As shown in Figure 2.5, the first rotation is denoted by $\bar{\beta}_1$, a translation by $\bar{\lambda}$ and the second rotation by $\bar{\beta}_2$. Thus, the motion control is given by:

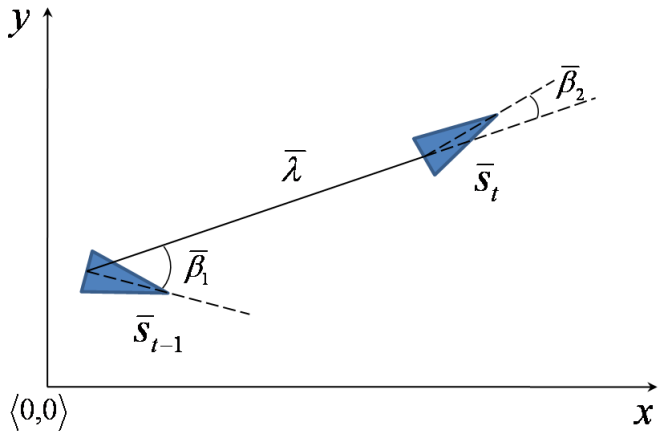


Figure 2.5: Odometry model: the robot motion is transformed into a rotation $\bar{\beta}_1$, a translation $\bar{\lambda}$ and a second rotation $\bar{\beta}_2$.

According to simple trigonometry, the values of the two rotations and the translation can be calculated from the odometry reading.

$$\bar{\beta}_1 = \text{atan2}(\bar{y}_t - \bar{y}_{t-1}, \bar{x}_t - \bar{x}_{t-1}) - \bar{\theta}_{t-1} \tag{2.13}$$

$$\bar{\lambda} = \sqrt{(\bar{x}_t - \bar{x}_{t-1})^2 + (\bar{y}_t - \bar{y}_{t-1})^2} \tag{2.14}$$

$$\bar{\beta}_2 = \bar{\theta}_t - \bar{\theta}_{t-1} - \bar{\beta}_1 \tag{2.15}$$

where atan2 is a variation of the arctangent function. Most programming languages provide an implementation of this function. In MATLAB, the range of the function is extended to the closed interval $[-\pi, \pi]$.

Under the assumption that the robot motion is noise-free, we consider $\bar{\beta}_1$, $\bar{\lambda}$ and $\bar{\beta}_2$ as the true values of the two rotation and the translation of the real robot. Thus the pose of the real robot is obtained by:

$$\begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix} = \begin{pmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{pmatrix} + \begin{pmatrix} \bar{\lambda} cos(\theta_{t-1} + \bar{\beta}_1) \\ \bar{\lambda} sin(\theta_{t-1} + \bar{\beta}_1) \\ \bar{\beta}_1 + \bar{\beta}_2 \end{pmatrix} \tag{2.16}$$

2.2.2 Probabilistic motion models

The motion models discussed so far are deterministic forms under the noise-free assumption. However, the real robot motion is inherently unpredictable. Uncertainty arises from effects like control noise, wear-and-tear, modeling errors, and mechanical failure. Hence, it is necessary to study the probabilistic motion model. This model is denoted as the conditional density [Thrun05]:

$$p(s_t \,|u_t, s_{t-1}) \tag{2.17}$$

where s_t and s_{t-1} are both robot poses, and u_t is a motion command. This model describes the posterior distribution over kinematic states that a robot assumes when executing the motion command u_t at s_{t-1} [Thrun05] (see Figure 2.6). Different from deterministic models that determine the sole pose, probabilistic models calculate the probabilities for all the possible poses in the robot's space.

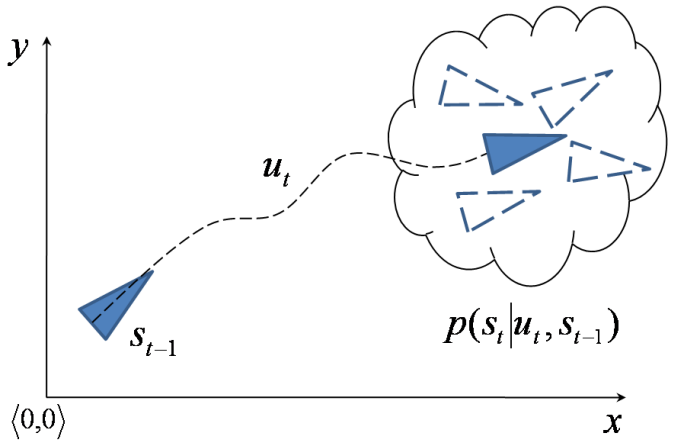


Figure 2.6: The robot's pose transition shown in the probabilistic form.

Corresponding to deterministic motion models mentioned above, we present probabilistic algorithms for computing $p(s_t \,|u_t, s_{t-1})$ based on the velocity motion model and odometry motion model, respectively.

2.2.2.1 Probabilistic velocity motion model

In [Thrun05] Thrun *et al.* propose the probabilistic algorithm based on their velocity motion model and give the complete mathematical derivation. But con-

sidering the computational complexity, we select the second velocity motion model to extend the probabilistic model.

The algorithm for computing the probability $p(s_t \,|u_t, s_{t-1})$ based on velocity information is shown in Algorithm 2.1. The inputs of this algorithm are an initial pose s_{t-1}, a successor pose s_t and a control u_t. The output is the probability $p(s_t \,|u_t, s_{t-1})$. We assume that the control is carried out in a small enough duration Δt. The actual velocities $\tilde{v}$ and $\tilde{\omega}$ are calculated in lines 2 and 3, which can be derived from Equation 2.7. The function $\mathbf{prob}(x, b^2)$ is introduced in [Thrun05], which computes the probability density of its argument x under a zero-centered distribution with variance b^2 (for example the Gaussian distribution or the triangular distribution). σ_1 and σ_2 are two error parameters relative to the control velocities v and ω (standard deviation).

Algorithm 2.1: Probabilistic algorithm of velocity motion model

1: **Input:** $s_{t-1} = (x_{t-1}, y_{t-1}, \theta_{t-1})^T, s_t = (x_t, y_t, \theta_t)^T, u_t = (v, \omega)^T$
2: $\tilde{v} = \frac{x_t - x_{t-1}}{cos(\theta_{t-1})\Delta t}$ or $\tilde{v} = \frac{y_t - y_{t-1}}{sin(\theta_{t-1})\Delta t}$
3: $\tilde{\omega} = \frac{\theta_t - \theta_{t-1}}{\Delta t}$
4: $p_1 = \mathbf{prob}(v - \tilde{v}, \sigma_1^2)$
5: $p_2 = \mathbf{prob}(\omega - \tilde{\omega}, \sigma_2^2)$
6: **Output:** $p = p_1 \cdot p_2$

The particle filter [Metropolis49] is a special case. Instead of computing the posterior $p(s_t \,|u_t, s_{t-1})$ directly, it generates random samples according to this conditional probability. Algorithm 2.2 gives a method to generate random samples from $p(s_t \,|u_t, s_{t-1})$. It inputs an initial pose s_{t-1} and a control u_t and it outputs a random pose s_t. Lines 2 and 3 calculate the noisy control parameters. Errors are generated by the function $\mathbf{sample}(0, b^2)$ [Thrun05]. This function generates a random sample from a zero-centered distribution with variance b^2. σ_1 and σ_2 are two error parameters relative to the control v and ω. Lines 4 to 6 show the procedure of generating a sample, which is identical to the robot motion model with noisy controls. Thus, 100 samples can be obtained by executing iteratively this algorithm 100 times.

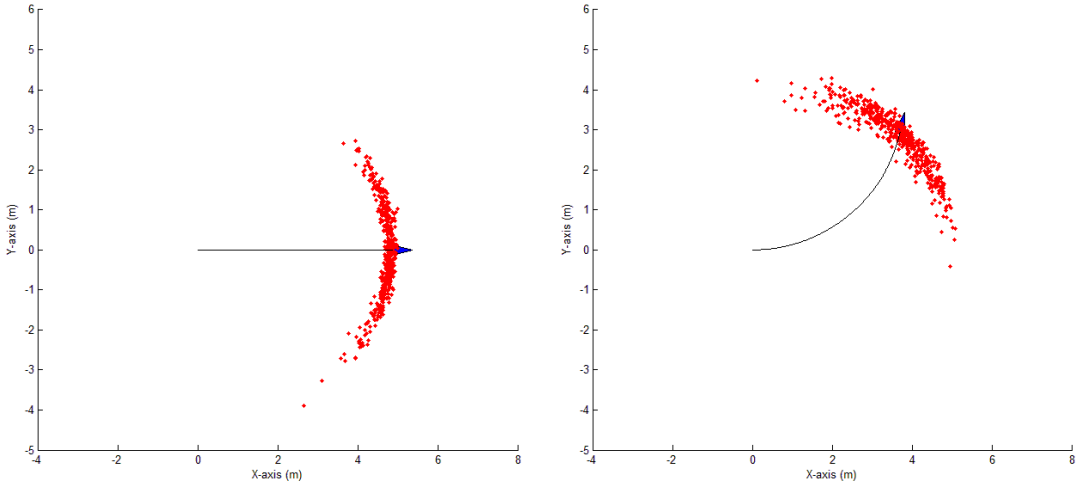
Figure 2.7 shows two examples that illustrate sample sets generated by Algorithm 2.2. Each diagram shows 500 samples, which are distributed according to $p(s_t \,|u_t, s_{t-1})$. The area owning the more samples, the more likely the robot is. The non-sensing robot advances in two types of motion. Figure 2.7(a) shows the robot moves in a linear motion and Figure 2.7(b) shows the robot moves in a circular motion.

2.2.2.2 Probabilistic odometry motion model

Odometric information can be technically considered as sensor measurements, however we can extract the control information from odometry (see Section 2.2.1.3).

Algorithm 2.2: Sampling algorithm of velocity motion model

1: **Input:** $s_{t-1} = (x_{t-1}, y_{t-1}, \theta_{t-1})^T, u_t = (v, \omega)^T$
2: $\tilde{v} = v + \textbf{sample}(0, \sigma_1^2)$
3: $\tilde{\omega} = \omega + \textbf{sample}(0, \sigma_2^2)$
4: $x_t = x_{t-1} + \tilde{v}\Delta t cos(\theta_{t-1})$
5: $y_t = y_{t-1} + \tilde{v}\Delta t sin(\theta_{t-1})$
6: $\theta_t = \theta_{t-1} + \tilde{\omega}\Delta t$
7: **Output:** $s_t = (x_t, y_t, \theta_t)^T$



(a) (b)

Figure 2.7: Sampling with the velocity motion model obtained by MATLAB. The robot advances in two types of motion: (a) a linear motion and (b) a circular motion.

The algorithm for computing $p(s_t \,|u_t, s_{t-1})$ based on odometry motion model is depicted in Algorithm 2.3. It inputs the initial pose s_{t-1} and the control $u_t = (\bar{s}_{t-1}, \bar{s}_t)^T$ obtained from odometry. It outputs the probability $p(s_t \,|u_t, s_{t-1})$. Lines 2 to 4 calculate virtual control parameters $\bar{\beta}_1$, $\bar{\lambda}$ and $\bar{\beta}_2$ based on odometry information. They are the inverse odometry motion model (Equation 2.16). Lines 5 to 7 calculate β_1, λ and β_2 for a pair of known poses (s_{t-1}, s_t). Lines 8 to 10 calculate the errors between odometry and the known poses for each virtual control parameter. These errors are represented by the distribution with zero mean and variance σ^2. As above, σ_1, σ_2 and σ_3 are three error parameters relative to the virtual control $\bar{\beta}_1$, $\bar{\lambda}$ and $\bar{\beta}_2$.

Algorithm 2.4 depicts the sampling algorithm of odometry motion model, which is devised specially for particle filters. It generates random samples according to $p(s_t \,|u_t, s_{t-1})$ instead of computing the posterior. The inputs of this algorithm are an initial pose s_{t-1} and a control u_t obtained from the odometry readings. The

Algorithm 2.3: Probabilistic algorithm of odometry motion model, adapted from [Thrun05]

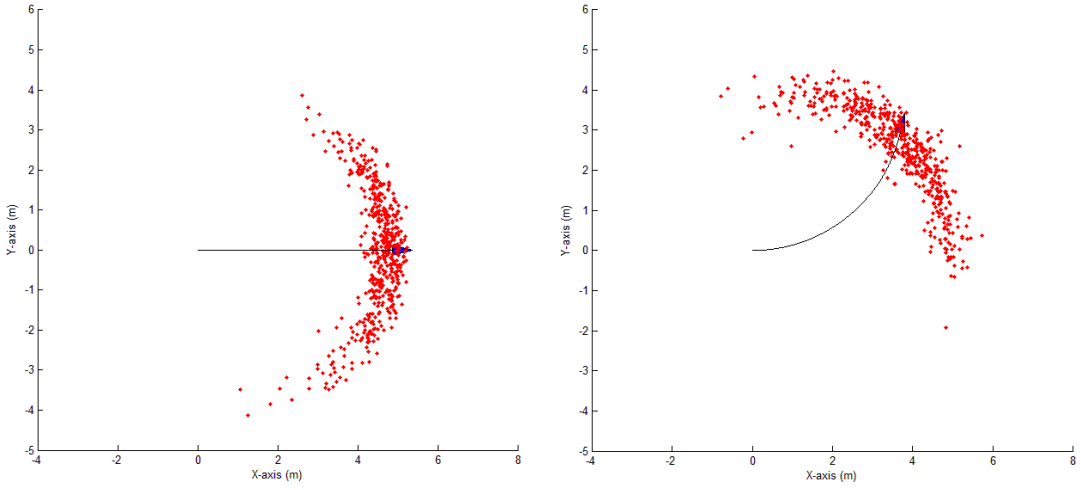
1: **Input:** $s_{t-1} = (x_{t-1}, y_{t-1}, \theta_{t-1})^T, s_t = (x_t, y_t, \theta_t)^T, u_t = (\bar{s}_{t-1}, \bar{s}_t)^T$
2: $\bar{\beta}_1 = \text{atan2}(\bar{y}_t - \bar{y}_{t-1}, \bar{x}_t - \bar{x}_{t-1}) - \bar{\theta}_{t-1}$
3: $\bar{\lambda} = \sqrt{(\bar{x}_t - \bar{x}_{t-1})^2 + (\bar{y}_t - \bar{y}_{t-1})^2}$
4: $\bar{\beta}_2 = \bar{\theta}_t - \bar{\theta}_{t-1} - \bar{\beta}_1$
5: $\beta_1 = \text{atan2}(y_t - y_{t-1}, x_t - x_{t-1}) - \theta_{t-1}$
6: $\lambda = \sqrt{(x_t - x_{t-1})^2 + (y_t - y_{t-1})^2}$
7: $\beta_2 = \theta_t - \theta_{t-1} - \beta_1$
8: $p_1 = \textbf{prob}(\bar{\beta}_1 - \beta_1, \sigma_1^2)$
9: $p_2 = \textbf{prob}(\bar{\lambda} - \lambda, \sigma_2^2)$
10: $p_3 = \textbf{prob}(\bar{\beta}_2 - \beta_2, \sigma_3^2)$
11: **Output:** $p = p_1 \cdot p_2 \cdot p_3$

output is a random pose s_t distributed according to $p(s_t \,|u_t, s_{t-1})$. Lines 2 to 4 calculate virtual control parameters $\bar{\beta}_1$, $\bar{\lambda}$ and $\bar{\beta}_2$ based on odometry information. Lines 5 to 7 sample the three virtual control parameters β_1, λ and β_2 from the virtual control parameters of odometry by the distribution with zero mean and variance σ^2. σ_1, σ_2 and σ_3 are relative to the virtual control $\bar{\beta}_1$, $\bar{\lambda}$ and $\bar{\beta}_2$. Lines 8 to 10 generate a random pose by employing the odometry motion model with the noisy controls.

Algorithm 2.4: Sampling algorithm of odometry motion model, adapted from [Thrun05]

1: **Input:** $s_{t-1} = (x_{t-1}, y_{t-1}, \theta_{t-1})^T, u_t = (\bar{s}_{t-1}, \bar{s}_t)^T$
2: $\bar{\beta}_1 = \text{atan2}(\bar{y}_t - \bar{y}_{t-1}, \bar{x}_t - \bar{x}_{t-1}) - \bar{\theta}_{t-1}$
3: $\bar{\lambda} = \sqrt{(\bar{x}_t - \bar{x}_{t-1})^2 + (\bar{y}_t - \bar{y}_{t-1})^2}$
4: $\bar{\beta}_2 = \bar{\theta}_t - \bar{\theta}_{t-1} - \bar{\beta}_1$
5: $\beta_1 = \bar{\beta}_1 + \textbf{sample}(0, \sigma_1^2)$
6: $\lambda = \bar{\lambda} + \textbf{sample}(0, \sigma_2^2)$
7: $\beta_2 = \bar{\beta}_2 + \textbf{sample}(0, \sigma_3^2)$
8: $x_t = x_{t-1} + \lambda cos(\theta_{t-1} + \bar{\beta}_1)$
9: $y_t = y_{t-1} + \lambda sin(\theta_{t-1} + \bar{\beta}_1)$
10: $\theta_t = \theta_{t-1} + \beta_1 + \beta_2$
11: **Output:** $s_t = (x_t, y_t, \theta_t)^T$

Figure 2.8 illustrates 500 samples generated by Algorithm 2.4. The non-sensing robot advances in two types of motion: a linear motion (see Figure 2.7(a)) and a circular motion (see Figure 2.7(b)). As shown here, samples are more spread out than Figure 2.7 due to more motion noise.



(a) (b)

Figure 2.8: Sampling with the odometry motion model obtained by MATLAB. The robot advances in two types of motion: (a) a linear motion and (b) a circular motion.

2.3 PERCEPTION MODELS

Next to motion, cognizing the surrounding environment is another foundational task for the autonomous robot localization. To describe the measurement process, we need to develop perception models (also called measurement models). Perception models describe the formation process that generates sensor measurements in the physical world [Thrun05].

Taking measurements is done by a variety of sensors in the physical world. Sensors can be classified roughly into two categories: proprioceptive sensors and exteroceptive sensors. Proprioceptive sensors measure values internal to the robot, such as motor speed, wheel load and battery voltage. Exteroceptive sensors acquire information from robot's environment, such as distance measurements, light intensity and sound amplitude [Siegwart04, Solutions07]. In this section, we first present briefly two kinds of exteroceptive sensors: range finders and vision-based sensors. Then we discuss probabilistic perception models for mobile robot localization. Probabilistic perception models are developed by Thrun *et al.* in [Thrun05]. We just employ it in our localization algorithm.

2.3.1 Range finders

Range finders are these devices that can measure distance from the observer to a target, which are the most popular sensors in robotics. To measure distance, there are basically three approaches [Borenstein96]:

- Time-of-flight measurement technique.
- Phase-shift measurement (or phase-detection) ranging technique.
- Frequency-modulated (FM) radar technique.

Here, we only discuss Time-of-flight range sensors and present two most common time-of-flight sensors: the ultrasonic sensor and the laser range finder.

Time-of-flight range sensors make use of the propagation speed of a sound or an electromagnetic wave to measure distance [Borenstein96, Siegwart04]. The travel distance of a sound or an electromagnetic wave is given by:

$$d = v \cdot t \tag{2.18}$$

where d = round-trip distance, v = speed of propagation and t = elapsed time. Note that the distance d is the round-trip distance and must be reduced by half to result in actual range to the target.

The advantages of time-of-flight systems arise from the direct nature of their straight-line active sensing. The absolute range to an observed point is directly available as output with no complicated analysis required, and the technique is not based on any assumptions concerning the planar properties or orientation of the target surface [Borenstein96].

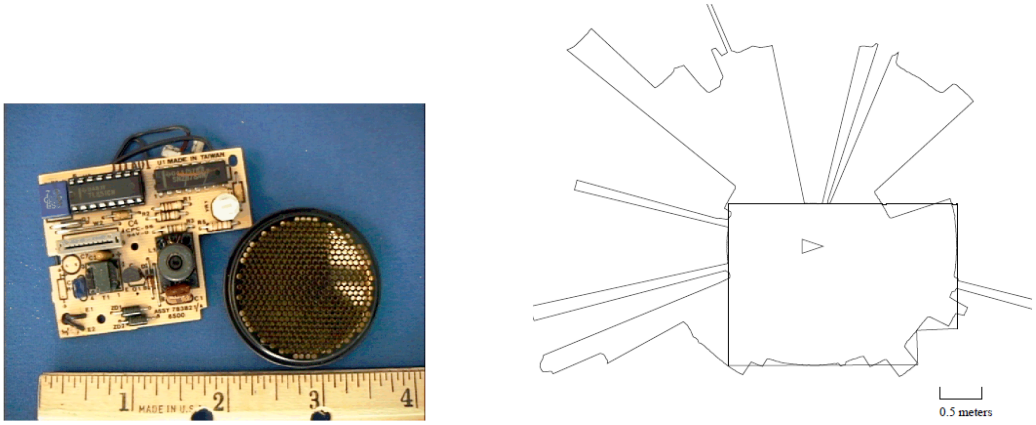
To characterize errors is necessary for whichever kind of sensors. Potential errors of time-of-flight systems are mainly from the following aspects [Borenstein96, Siegwart04]:

- Uncertainties in determining the exact time of arrival of the reflected signal.
- Inaccuracies in the measurement of the round-trip time of flight, particularly for laser range sensors.
- The dispersal of the transmitted beam cone with the measurement distance increase, mainly for ultrasonic range sensors.
- Interaction of the incident wave with the target surface, for example, surface absorption, specular reflections.
- Variations in the speed of propagation, particularly for ultrasonic range sensors.
- The speed of the mobile robot and the speed of target particularly in the case of a dynamic target.

2.3.1.1 Ultrasonic sensors

The ultrasonic sensor is today one of the most common device employed on indoor mobile robotics systems. It offers a low-cost solution to the localization problem for mobile robots. Due to the availability of its easy application, much research has been conducted investigating applicability in the area of robot navigation [Borenstein91a, Borenstein95, Kieffer00, Wu01, Moreno02, Burguera07, Hsu09].

Figure 2.9(a) shows two fundamental components of a Polaroid ranging module: the ultrasonic transducer and the ranging module electronics. Figure 2.9(b) shows a typical ultrasound scan superimposed on a hand-measured map of the room. From this figure, we can see that the ultrasonic sensor has large errors and even it is inability in some situations.



(a) (b)

Figure 2.9: (a) An OEM kit of the Polaroid ultrasonic sensor included the transducer and an electronics interface board. Image courtesy of Johann Borenstein, University of Michigan [Borenstein96]. (b) Typical scan of an ultrasonic system. Images courtesy of John J. Leonard, MIT [Leonard92].

For ultrasonic sensors, there are two typical errors: a misreading for smooth surfaces and crosstalk.

- **The misreading.** When an ultrasonic sensor measures smooth surfaces at an angle, the echo is reflected specularly and travels into a direction that is outside of the sensing envelope of the receiver, as illustrated in Figure 2.10(a). In this case, the sensor often reports overly large measurements when compared with the true distance.

- **Crosstalk.** This phenomenon can occur in both single-robot systems and multi-robot systems, as shown in Figure 2.10(b) and (c). For single-robot systems, it often occurs in cluttered environments, sound waves can reflect (multiple) from objects and can then be received by other sensors. For multi-robot systems, sound waves can be emitted by one robot and be received by another

robot. A method to detect and reject crosstalk is proposed by Borenstein *et al.* [Borenstein95].

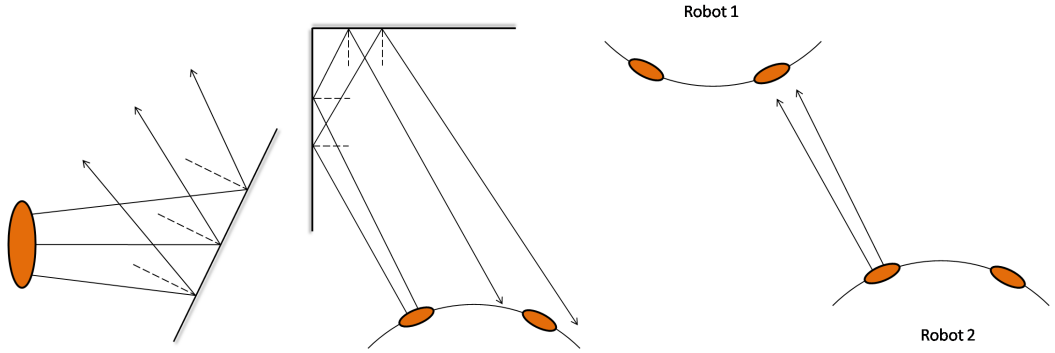


Figure 2.10: (a) A misreading of the ultrasonic sensor. (b) The crosstalk phenomenon in single-robot systems. (c) The crosstalk phenomenon in multi-robot systems.

2.3.1.2 Laser range finders

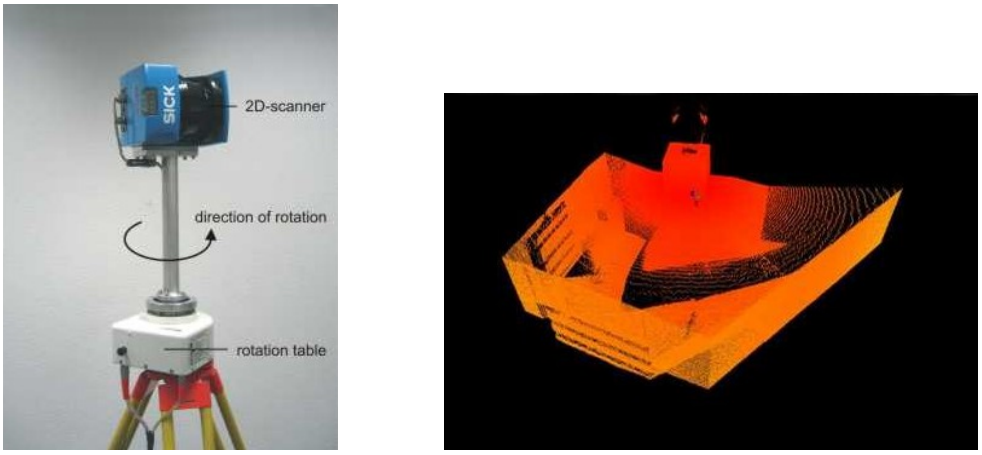
The Laser Range Finder, also referred as laser radar or lidar, is a laser-based time-of-flight ranging system, which has the similar principle to the ultrasonic sensor. To measure the distance, it also emits a signal and records its echo. But different to the ultrasonic sensor, it uses laser light instead of sound. Since the laser provides much more focused beams, it achieves significant improvements over the ultrasonic range sensor. Considerable research has been dedicated to mobile robot navigation by using laser range finders [Arsenio98, Zhang00, Schulz03, Madhavan04, Silva07, Harrison08]

Figure 2.11(a) shows a terrestrial 3D laser scanner ZLS07 developed by Zogg *et al.* [Zogg07] and (b) shows a typical scan result by ZLS07 in an underground utility cavern.

The laser range finder is more accurate than the ultrasonic sensor, but much more expensive. Like the ultrasonic sensor, the measured medium is an important aspect of impacting on measurement accuracy. For example, a highly polished surface will reflect the incident laser. Different to ultrasonic sensors, the laser range finder cannot detect the presence of optically transparent materials such as glass or light-absorbing objects.

2.3.2 Vision-based sensors

Compared with range finders, visual sensing can provide a tremendous amount of information about a robot's environment, and it is potentially the most powerful



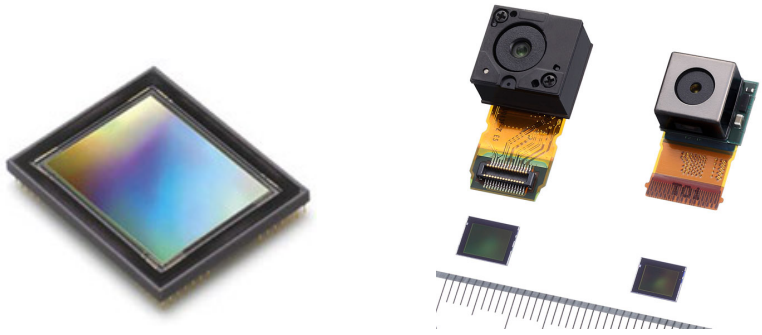
(a) (b)

Figure 2.11: (a) A terrestrial 3D laser scanner ZLS07 based on Sick LMS 200. (b) Point cloud of an underground utility cavern, acquired with ZLS07. Images courtesy of Hans-Martin Zogg, Swiss Federal Institute of Technology, ETH Zurich [Zogg07]

source of information among all the sensors used on robots to date [Siegwart04, Solutions07].

CCD (charge-coupled device) and CMOS (complementary metal oxide semiconductor) are the two current technologies for creating vision sensors. Both of two are semiconductor devices that can convert optical images into the digital signal. Digital signal can be stored and processed easily by computers, thus information hidden in pictures can be extracted according to the needs. In a CCD sensor (see Figure 2.12(a)), every pixel's charge is transferred through a very limited number of output nodes (often just one) to be converted to voltage, buffered, and sent off-chip as an analog signal. All of the pixel can be devoted to light capture, and the output's uniformity (a key factor in image quality) is high. In a CMOS sensor (see Figure 2.12(b)), each pixel has its own charge-to-voltage conversion, and the sensor often also includes amplifiers, noise-correction, and digitization circuits, so that the chip outputs digital bits. These other functions increase the design complexity and reduce the area available for light capture. With each pixel doing its own conversion, uniformity is lower. But the chip can be built to require less off-chip circuitry for basic operation [DAL].

The most promising sensor for the future of mobile robotics is likely vision [Siegwart04], therefore a great deal of effort has been directed at applying vision sensors to the field of mobile robots. To mimic the capabilities of the human vision system, many biologically-inspired robotic vision systems are proposed, too. Siagian *et al.* present a robot localization system using biologically-inspired vision [Siagian07]. Their system models two extensively studied human visual capabilities: one extracting the "gist" of a scene to produce a coarse localization hypothesis, and the other refining it by locating salient landmark regions in the scene. The



(a) (b)

Figure 2.12: (a) A commercially available CCD chip: DALSA $36 \times 48mm^2$ 48M-pixel CCD. (b) Commercially available SONY Exmor CMOS camera sensors.

gist and salient landmark features are then further processed using a Monte Carlo localization algorithm to allow the robot to generate its position. Similarly using Monte Carlo localization, Wolf *et al.* use a vision-based approach that combines image retrieval techniques with Monte Carlo localization to solve the mobile robot localization problem [Wolf02]. Their image retrieval system uses invariant features in order to find the most similar matches. Courbon *et al.* solve global robot localization by finding out the image which best fits the current image in a set of prerecorded images (visual memory) [Courbon08]. They propose a hierarchical process combining global descriptors computed onto cubic interpolation of triangular mesh and patches correlation around Harris corners.

2.3.3 Feature extraction

There are two strategies for using uncertainty sensors data to determine the robot's position. One is to use raw sensor measurements and the other is to extract features from sensor measurements. A feature extraction process can be defined as to extract information from one or more sensor readings first, generating a higher-level percept that can then be used to inform the robot's model and perhaps the robot's action directly [Siegwart04]. Mathematically, the feature extractor can be denoted as a function f and the features extracted from sensors data are given by $f(z_t)$.

Features are recognizable structures of elements in the environment [Siegwart04]. For range sensors, features may be lines, corners or local minima in range scans, which correspond to walls, corners or objects such as tree trunks. When using cameras to localization, popular features include edges, corners, distinct patterns and objects of distinct appearance [Thrun05].

In the physical world, these physical objects, which can be recognized by their distinct features, are often called landmarks. Typically, landmarks have a fixed and known position, relative to which a robot can localize itself [Borenstein96]. Moreover, landmarks should be easy to identify. Landmarks are commonly dis-

tinguished by two types: artificial landmarks and natural landmarks. Natural landmarks are those objects or features that are already in the environment and have a function other than robot navigation; artificial landmarks are specially designed objects or markers that need to be placed in the environment with the sole purpose of enabling robot navigation [Borenstein96].

2.3.3.1 Feature extraction from range finders

Most of features extracted from range finders are geometric primitives such as line segments or circles. Borges *et al.* present the Split-and-Merge Fuzzy (SMF) line extractor for line extraction in 2D range images provided by laser range finders, which uses a prototype-based fuzzy clustering algorithm in a split-and-merge framework [Borges00, Borges02]. They compare SMF with other two classic algorithm: Line Tracking (LT) and Iterative End-Point Fit (IEPF) using simulated and real data in [Borges02, Borges04]. Their experimental results show that SMF outperforms LT and IEPF but each algorithm has limitations. Sack *et al.* compare three different approaches for learning line maps from range data in [Sack04]. Their experimental results demonstrate that the incremental approach and the offline approach perform better than the Expectation-Maximization (EM) technique, furthermore the incremental approach generates more accurate results than the offline and the EM techniques. Another comparison of line extraction methods from range data are presented in [Nguyen07]. Nguyen *et al.* present an experimental evaluation of six popular line extraction algorithms applied to 2D laser scans for indoor environments. They compare and discuss the advantages and drawbacks of each algorithm in several aspects: speed, complexity, correctness and precision. Additionally, they test and compare the line extraction algorithms in the Orthogonal SLAM (OrthoSLAM) application. Their experimental results show that the two algorithms *Split-and-Merge* and *Incremental* have best performances because of their superior speed and correctness.

Several curvature-based feature extraction algorithms are developed in the literature. Madhavan *et al.* propose a natural landmark navigation algorithm for autonomous vehicles [Madhavan04]. They develop a multi-scale Curvature Scale Space (CSS) algorithm to identify, extract and localize landmarks characterized by points of maximum curvature in laser scans. Núñez *et al.* propose a geometrical feature detection system for laser data segmentation based on adaptive curvature estimation [Núñez08]. This algorithm can divide the laser scan into line segments and curve segments.

2.3.3.2 Feature extraction based on vision

To extract features using vision-based sensors covers the field of computer vision and image processing, which is far beyond the scope of this thesis. In the recent two decades, considerable research effort has been dedicated to these fields. The foundation of computer vision is introduced in [Haralick92, Ritter96]. Borges *et al.* [Borges02] present the methods of extraction linear features from 2D images.

More practical applications of computer vision to the mobile robot navigation are concerned in [Devy93, Devy95, Murrieta-Cid02, Sola08].

2.3.4 Probabilistic perception models

In reality, all types of sensors are imperfect devices. They perceive the world with both systematic and random errors. To model these uncertain measurements, we employ the probabilistic perception model. It is described as a conditional probability distribution:

$$p(z_t \,|s_t, m\,) \tag{2.19}$$

where s_t and z_t are the robot pose and measurements at time t, respectively. m is the map of the environment. Generally speaking, a map is a set of data that is necessary to describe the environment and localize the robot. More details about maps will be discussed in Section 2.4.

In some cases, many sensors generate more than one numerical measurement value, such as cameras generate entire arrays of values or range finders usually generate entire scans of ranges. Additionally, one robot may be equipped with several sensors, for example the Pioneer 3-DX mobile robot equipped with sixteen ultrasonic range finders. These sensors will generate a list raw scans. The number of such measurement value within a measurement z_t is denoted by I [Thrun05].

$$z_t = \left\{z_t^1, \cdots, z_t^I\right\} \tag{2.20}$$

We assume that these measurements have independent noise over time. The probability $p(z_t \,|s_t, m\,)$ can be obtained as the product of the individual measurement likelihoods [Thrun05].

$$p(z_t \,|s_t, m\,) = \prod_{i=1}^{I} p(z_t^i \,|s_t, m\,) \tag{2.21}$$

In this section we only intuitively review two probabilistic perception models, one is based on range finders and the other is based on features. These models are introduced in detail in [Thrun05].

2.3.4.1 Perception model based on range finders

Range finders only measure the range to the closest object that can be "seen". In [Thrun05], Thrun *et al.* introduce three range finder models: the beam-based model, the likelihood field model and the correlation-based measurement model (map matching). Each model has its own advantages and disadvantages. The beam-based model links closely to the geometry and physics of range finders, which is the preferable model of ultrasonic sensors and a good model of the laser range finders. However, the beam-based model is *lack of smoothness* and *computational involved* [Thrun05]. On the contrary, the likelihood field model overcomes these

disadvantages of the beam-based model, but it lacks a plausible physical explanation. It does not consider information involved in free-space and occlusions in the interpretation of range measurements. Similarly to the likelihood field model, map matching is also missing of the physical explanation. However, this model takes into account the free-space and it can be implemented efficiently. In this section, we only focus on the beam-based model because our experiments are based on ultrasonic sensors.

The beam-based model merges four types of measurement errors: measurement noise of measuring the known objects, errors due to unexpected objects, errors due to failures to detect objects and random unexplained noise. Therefore, the desired probability $p(z_t \,|s_t, m)$ is modeled as a mixture of four densities [Thrun05].

1. **Measuring the known objects with measurement noise.** If the sensor detects an object that has been modeled in the map, the result distribution is modeled by a Gaussian distribution with the mean at the distance to this object. This Gaussian is denoted by p_k, its mean (the true distance to the object) is denoted by d_{obj} and its standard deviation by σ_k. Usually, d_{obj} is obtained using *ray casting* [Thrun05]. In practice, the measurement values of the range sensor are limited to the interval $[0, z_{max}]$, where z_{max} denotes the maximum sensor range. Thus, the probability of such measurements is described as:

$$p_k(z_t^i \,|s_t, m) = \begin{cases} \eta \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(z_t^i - d_{obj})^2}{2\sigma_k^2}} & \text{if } 0 \leq z_t^i \leq z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{2.22}$$

   where η is the normalizer, which is calculated as

$$\eta = \left(\int_0^{z_{max}} \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(z_t^i - d_{obj})^2}{2\sigma_k^2}} dz_t^i \right)^{-1} \tag{2.23}$$

   Figure 2.13 depicts graphically this Gaussian distribution p_k with mean d_{obj} and standard deviation σ_k. The standard deviation σ_k is an intrinsic noise parameter of the measurement model, which determines the width of the Gaussian distribution. A low standard deviation indicates that the data points tend to be very close to the mean, while high standard deviation indicates that the data are spread out over a large range of values.

2. **Unexpected objects.** In Markov localization, the world model is generally assumed to be static and complete [Fox99b]. However, the real environments of mobile robots are often dynamic and incomplete, for example, some unmodeled obstacles and moving people. These objects that are not represented in the map can cause shorter ranges than distances presented in the map, because

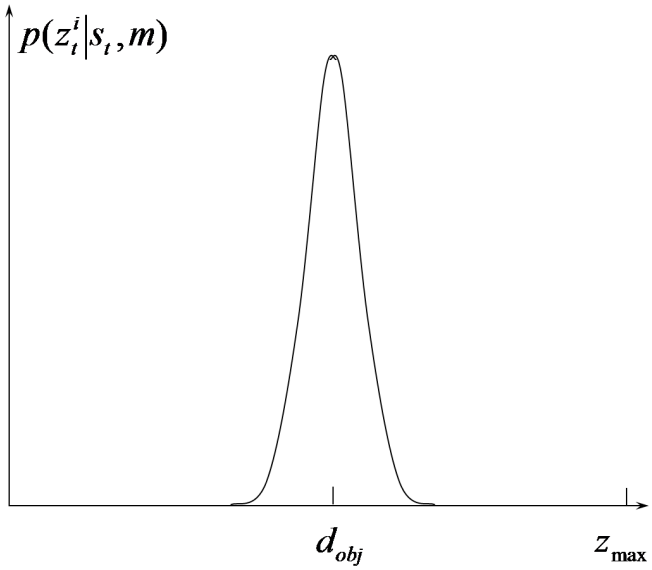


Figure 2.13: Measurements of the known objects are modeled as the Gaussian distribution p_k.

range finders only measure the range to the closest object. In this situation, the likelihood of measuring unexpected objects decreases with the difference of range. Figure 2.14 illustrates this situation. Ob_1 is a known obstacle and Ob_2 is an unknown obstacle. Ob_2 brings on the distance $d_2 < d_1$ and the probability $p_2 < p_1$.

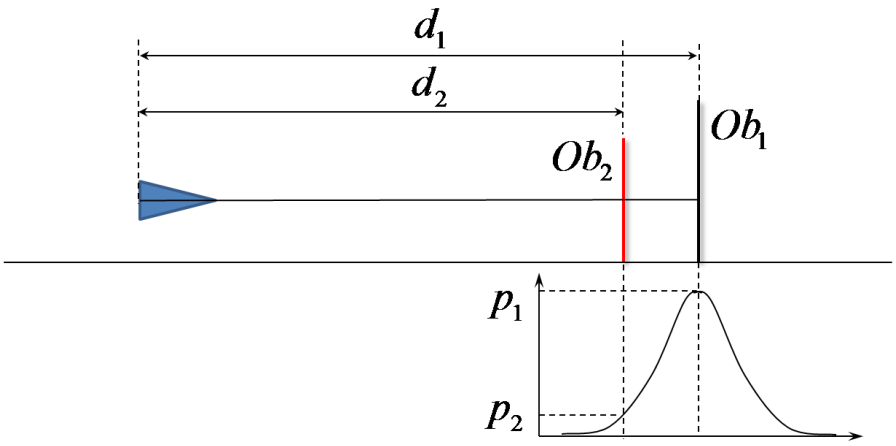


Figure 2.14: Measurement caused by the unexpected object. Ob_1 is a known obstacle and Ob_2 is an unknown obstacle.

In order to compensate the differences, the probability of these measurements caused by unexpected objects is modeled by an exponential distribution, denoted as p_{uk}.

$$p_{uk}(z_t^i \,|s_t, m) = \begin{cases} \eta \lambda_{uk} e^{-\lambda_{uk} z_t^i} & \text{if } 0 \leq z_t^i \leq d_{obj} \\ 0 & \text{otherwise} \end{cases} \tag{2.24}$$

The interval of this distribution is $[0, d_{obj}]$ because range measurements to unexpected objects are always shorter than true distances. λ_{uk} is an intrinsic parameter of this measurement model, which determines the slope of the exponential curve. The normalizer η evaluates to

$$\eta = \left(\int_0^{d_{obj}} \lambda_{uk} e^{-\lambda_{uk} z_t^i} dz_t^i \right)^{-1} = \frac{1}{1 - e^{-\lambda_{uk} d_{obj}}} \tag{2.25}$$

Figure 2.15 illustrates this exponential distribution p_{uk}. As shown in the figure, the probability compensation for unexpected objects decreases exponentially with the range z_t^i. In other words, the unexpected object is closer to the known object, its effect is smaller.

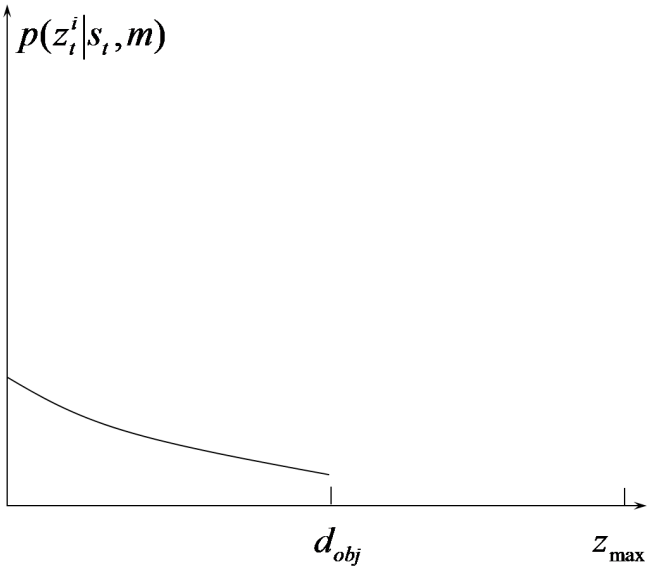


Figure 2.15: Measurements of the unknown objects are modeled as the exponential distribution p_{uk}, adapted from [Thrun05].

3. **Detect failures.** This situation occurs when sensors fail to detect obstacles. For instance, ultrasonic sensors are misreading when measuring smooth surfaces at an angle (see Section 2.3.1.1) and laser range finders detect optically transparent materials or light-absorbing objects (see Section 2.3.1.2). This kind of sensor failures often returns a maximum measurement. Mathematically, the probability of this case is modeled as a point-mass distribution p_f centered at z_{max}.

$$p_f(z_t^i \,|s_t, m) = \begin{cases} 1 & \text{if } z_t^i = z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{2.26}$$

As shown in Figure 2.16, to illustrate this point-mass distribution, p_f is described as a very narrow uniform distribution centered at z_{max}.

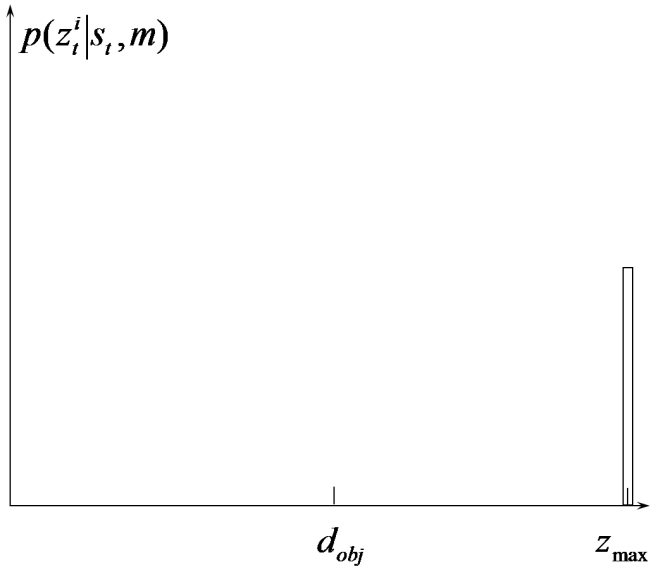


Figure 2.16: Measurement fails are modeled as the point-mass distribution p_f, adapted from [Thrun05].

4. **Random measurements.** Sometimes, range finders produce completely unexplainable measurements, such as the crosstalk phenomenon of ultrasonic sensors (see Section 2.3.1.1). The probability of such unexplainable measurements is modeled as a uniform distribution p_r spread over the entire measurement range $[0, z_{max})$.

$$p_r(z_t^i \,|s_t, m) = \begin{cases} \frac{1}{z_{max}} & \text{if } 0 \leq z_t^i < z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{2.27}$$

Such a uniform distribution p_r is depicted graphically in Figure 2.17.

To obtain the final probability distribution $p(z_t^i \,|s_t, m)$, these four different distributions are mixed by the weighted average.

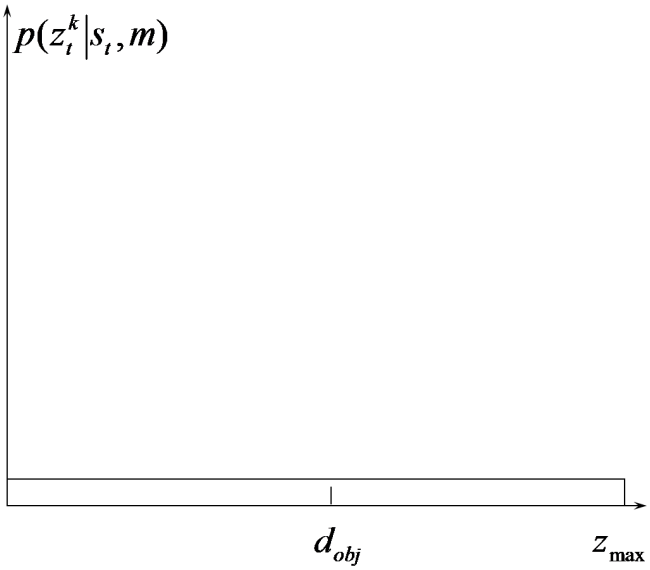


Figure 2.17: Unexplainable measurements are modeled as the uniform distribution p_r, adapted from [Thrun05].

$$p(z_t^i|s_t, m) = \begin{pmatrix} \alpha_k \\ \alpha_{uk} \\ \alpha_f \\ \alpha_r \end{pmatrix}^T \cdot \begin{pmatrix} p_k(z_t^i|s_t, m) \\ p_{uk}(z_t^i|s_t, m) \\ p_f(z_t^i|s_t, m) \\ p_r(z_t^i|s_t, m) \end{pmatrix} \tag{2.28}$$

where the parameters $\alpha_k, \alpha_{uk}, \alpha_f$ and α_r determine the existence proportion of these four distributions, thus

$$\alpha_k + \alpha_{uk} + \alpha_f + \alpha_r = 1 \tag{2.29}$$

Now, how to choose the various parameters of the beam-based perception model is a problem. These parameters include the mixing parameters $\alpha_k, \alpha_{uk}, \alpha_f$ and α_r, and also include σ_k and λ_{uk}. The simplest method is to estimate the values of these parameters according to experience. Most of the time, we can obtain the perfectly acceptable results. A more formal approach is presented in [Thrun05]. This approach uses actual data to learn these parameters.

Algorithm 2.5 presents an algorithm to implement the beam-based range finder model. It accepts the robot pose s_t, the complete range scan z_t and the map m as input. It outputs the mixture distribution $p(z_t^i|s_t, m)$. Line 2 initializes the desired probability P_{mix}. In line 6, individual measurement probabilities are multiplied according to Equation 2.21. Line 4 computes the true range to the object d_{obj} by using ray casting. Line 5 computes the mixed probability for each individual range measurement z_t^i based on Equation 2.28.

Figure 2.18 illustrates a typical density function of the mixture distribution $p(z_t^i|s_t, m)$ generated by Matlab. As shown in the figure, characteristics of all

Algorithm 2.5: Beam-based range finder model, adapted from [Thrun05]

1: **Input:** $s_t, z_t = (z_t^1, \cdots, z_t^I), m$
2: **Initialization:** $P_{mix} = 1$
3: **for** $i = 1$ to I **do**
4: compute d_{obj} for each measurement z_t^i using *ray casting*
5: $p = \alpha_k \cdot p_k(z_t^i \,|s_t, m) + \alpha_{uk} \cdot p_{uk}(z_t^i \,|s_t, m)$
$+\alpha_f \cdot p_f(z_t^i \,|s_t, m) + \alpha_r \cdot p_r(z_t^i \,|s_t, m)$
6: $P_{mix} = P_{mix} \cdot p$
7: **end for**
8: **Output:** P_{mix}

four basic models are still remained in the mixture distribution. In the program of generating this figure, these intrinsic parameters of the beam-based model are chosen as $\alpha_k = 0.4, \alpha_{uk} = 0.3, \alpha_f = 0.2, \alpha_r = 0.1, \sigma_k = 0.2$ and $\lambda_{uk} = 0.5$.

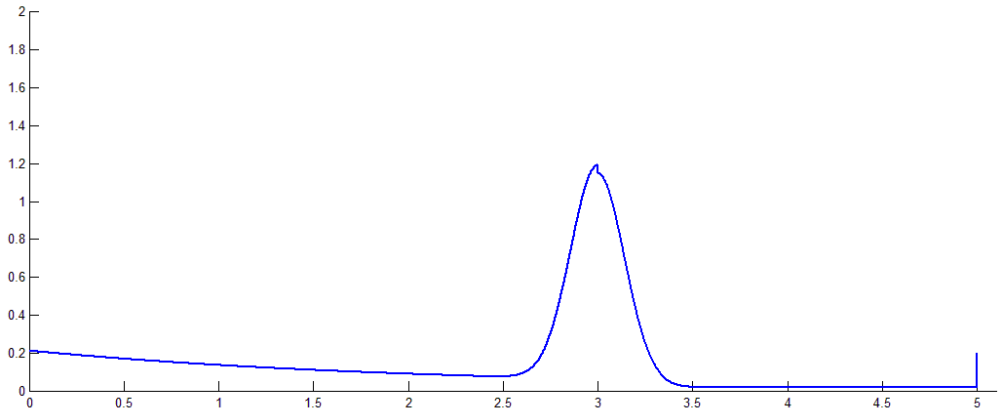


Figure 2.18: Density of the mixture distribution $p(z_t^i \,|s_t, m)$, generated by Matlab.

2.3.4.2 Perception model based on features

A feature extracted from the sensor measurement can be represented by the range r, the bearing ϕ and the signature τ [Thrun05]. Thrun *et al.* assume that a signature is a numerical value, which may equally be an integer that characterizes the type of the observed landmark, or a multi-dimensional vector characterizing a landmark [Thrun05].

$$f_t^1 = \begin{pmatrix} r_t^1 \\ \phi_t^1 \\ \tau_t^1 \end{pmatrix} \tag{2.30}$$

Thus, all features extracted from the sensor measurement are given by

$$\begin{aligned} f(z_t) &= \left\{ f_t^1, f_t^2, \cdots \right\} \\ &= \left\{ \begin{pmatrix} r_t^1 \\ \phi_t^1 \\ \tau_t^1 \end{pmatrix}, \begin{pmatrix} r_t^2 \\ \phi_t^2 \\ \tau_t^2 \end{pmatrix}, \cdots \right\} \end{aligned} \tag{2.31}$$

here, the number of features identified by the robot at each time step is variable.

Under the conditional independence between features assumption, we have

$$p(f(z_t) \,|s_t, m) = \prod_i p(r_t^i, \phi_t^i, \tau_t^i \,|s_t, m) \tag{2.32}$$

The feature-based sensor model can be described as follows, assuming that the i^{th} feature observed at time t corresponds to the j^{th} landmark in the map.

$$\begin{pmatrix} r_t^i \\ \phi_t^i \\ \tau_t^i \end{pmatrix} = \begin{pmatrix} \sqrt{(m_{j,x} - x)^2 + (m_{j,y} - y)^2} \\ \text{atan2}(m_{j,y} - y, m_{j,x} - x) \\ m_{j,\tau} \end{pmatrix} + \begin{pmatrix} \epsilon_{\sigma_r^2} \\ \epsilon_{\sigma_\phi^2} \\ \epsilon_{\sigma_\tau^2} \end{pmatrix} \tag{2.33}$$

where, the robot pose is given by $s_t = (x, y, \theta)^T$. $m_{j,x}$ and $m_{j,y}$ denote the coordinates of a feature in the global coordinate frame. $m_{j,\tau}$ denotes a signature in the map. $\epsilon_{\sigma_r^2}, \epsilon_{\sigma_\phi^2}$ and $\epsilon_{\sigma_\tau^2}$ are three independent Gaussian error variables with zero-mean and standard deviations σ_r, σ_ϕ and σ_τ, respectively.

The algorithm for calculating the probability of an observed feature with known correspondence is depicted in Algorithm 2.6. It accepts as input an observed feature f_t^i, the robot pose s_t, the feature identity c_t^i and the map m. It outputs the probability $p(f_t^i \,|c_t^i, s_t, m)$. The feature f_t^i is obtained from sensor data. The feature identity c_t^i is a correspondence variable between the feature f_t^i that is observed by the robot and the landmark m_j in the map [Thrun05]. Here, c_t^i is known and $c_t^i \leq N$, N is the number of landmarks in the map. Line 2 represents the i^{th} feature observed at time t corresponds to the j^{th} landmark in the map. Lines 3 to 5 calculate the range and the bearing to the landmark. Lines 6 to 8 calculate the errors of the range and the bearing between the sensor measurement and landmark, respectively. They are represented by the normal distribution with zero mean and variance σ^2. Since the correspondence c_t^i is assumed to be known, we can obtain the signature $\tau = m_{j,\tau}$ directly. The probability $\mathbf{prob}(\tau_t^i - \tilde{\tau}, \sigma_\tau^2)$ will not be involved in the algorithm. However, in the case of unknown correspondence, this probability should be considered.

2.4 MAPS

A pre-existing map is the third known component to solve the mobile robot localization problem. In [Thrun05], Thrun *et al.* give the definition of map. A map m is a list of objects in the environment along with their properties:

Algorithm 2.6: Feature-based sensor model with known correspondence, adapted from [Thrun05]

1: **Input:** $f_t^i = (r_t^i, \phi_t^i, \tau_t^i)^T, s_t = (x_t, y_t, \theta_t)^T, c_t^i, m$
2: $j = c_t^i$
3: $\tilde{r} = \sqrt{(m_{j,x} - x_t)^2 + (m_{j,y} - y_t)^2}$
4: $\tilde{\phi} = \text{atan2}(m_{j,y} - y_t, m_{j,x} - x_t)$
5: $p1 = \textbf{prob}(r_t^i - \tilde{r}, \sigma_r^2)$
6: $p2 = \textbf{prob}(\phi_t^i - \tilde{\phi}, \sigma_\phi^2)$
7: **Output:** $p = p1 \cdot p2$

$$m = \{m_1, m_2, \cdots, m_I\} \tag{2.34}$$

where I is the total number of objects in the environment, and each m_i with $1 \leq i \leq I$ specifies a property.

2.4.1 Map representation

In the mobile robotics literature, various map representations are proposed. To choose an appropriate map representation, Siegwart *et al.* think that three aspects should be considered [Siegwart04]:

1. The precision of the map must appropriately match the precision with which the robot needs to achieve its goals.

2. The precision of the map and the type of features represented must match the precision and data types returned by the robot's sensors.

3. The complexity of the map representation has direct impact on the computational complexity of reasoning about mapping, localization and navigation.

In general, there are two ways to represent maps: geometric maps and topological maps [Borenstein96].

2.4.1.1 Geometric maps

A geometric map represents objects according to their absolute geometric relationships [Borenstein96]. We introduce two most popular geometric representations: the occupancy grid map and the line map.

- **The occupancy grid map:** The occupancy grid technique is a popular and simple way to represent the geometric map. The occupancy grid representation is a fixed decomposition technique, which represent the environment using

regular-spaced grids. Each grid cell may be either filled or empty. In probabilistic robotics, the occupied cell is noted "1" and the free one are noted "0". The notation $p(m_i)$ refers to the probability that a grid cell is occupied. The occupancy grid technique is particularly suitable for robots equipped with range finders because the range values of each sensor, combined with the absolute position of the robot, can be used directly to update the filled or empty value of each grid cell [Siegwart04].

Moravec *et al.* are the first to introduce the occupancy grid map in conjunction with mobile robot [Moravec85]. Borenstein *et al.* firstly adopt the occupancy grid map for collision avoidance and they refined this method with histogram grid [Borenstein90, Borenstein91b, Borenstein91a].

Figure 2.19 shows an occupancy grid map. The dimension of the map is about $25m \times 10m$ and the resolution of the grid is $0.8m \times 0.8m$. This is a low resolution occupancy grid map. As shown, some line features have been lost in this resolution.

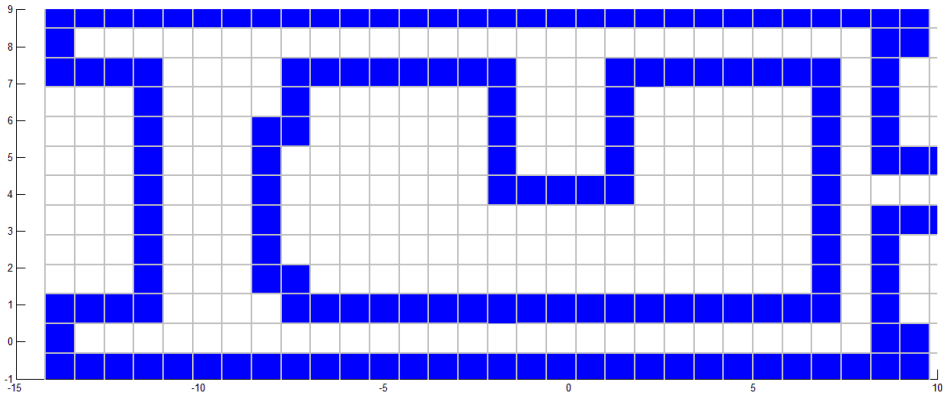


Figure 2.19: A occupancy grid map.

The advantages of the occupancy grid map are that it is easy to build and can easily be updated upon sensory input [Zhang00, Sack04]. Especially, the occupancy grid map is suitable for imprecise range sensors [Zhang00]. However, the disadvantages lie in the huge memory requirements and the limited accuracy due to the discretization [Sack04]. Its accuracy entirely depends on the resolution of the grid. The finer grained can get a more accurate representation, but at the expense of increased memory requirements.

- **The line map:** To overcome these limitations, many works have been focused on the line map [Zhang00, Sack04, Nguyen07]. A line map is composed of a finite number of line segments: $L = l_1, l_2, \cdots, l_N$. Line maps are widely used in the indoor environment and the structured outdoor environment. These envi-

ronments usually consist of planar structures such as walls, doors or cupboards, all of which can be described by linear characteristics.

To represent a structured environment, the line map only uses a small number of line segments whereas the occupancy grid map may require thousands of grid cells. Thus, the line map requires significantly less memory than the occupancy grid map [Sack04]. Figure 2.20 displays a line map, which represents the same environment to Figure 2.19. As shown in the figure, the line map is more accurate than the occupancy grid map since it provides the floating point resolution and does not exist the discretization problems [Sack04].

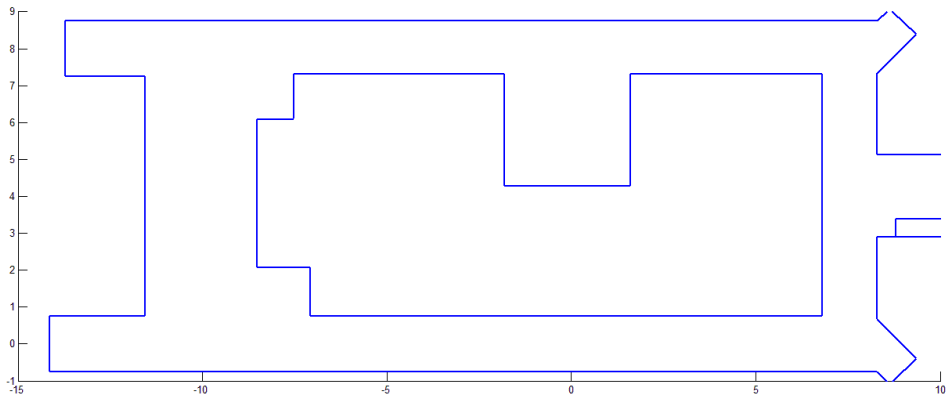


Figure 2.20: A line map.

2.4.1.2 Topological maps

An alternative map representation is the topological map. The topological map is a coarse graph-like representation based on recording the geometric relationships between the observed features [Borenstein96, Friedman07]. The nodes represent the observed features in the environment and the arcs represent paths between the features [Kortenkamp94, Borenstein96]. Figure 2.21 shows a topological map of an indoor office environment [Siegwart04].

A detailed comparison of the topological representation and the grid-based representation is introduced in [Thrun98a, Thrun98b]. Furthermore, authors present an integrated approach to mapping indoor robot environments, which combines topological maps and grid-based maps. Grid-based maps are learned using artificial neural networks and Bayes rule. Topological maps are generated on top of the grid-based maps by partitioning the latter into coherent region. This combination gains advantages from both two approaches. A further work is presented in [Friedman07], Friedman *et al.* propose a novel approach to build the topological map in the indoor environment, which is named Voronoi random fields (VRFs). VRFs apply discriminatively trained conditional random fields to label the points of Voronoi graphs extracted from occupancy grid maps.

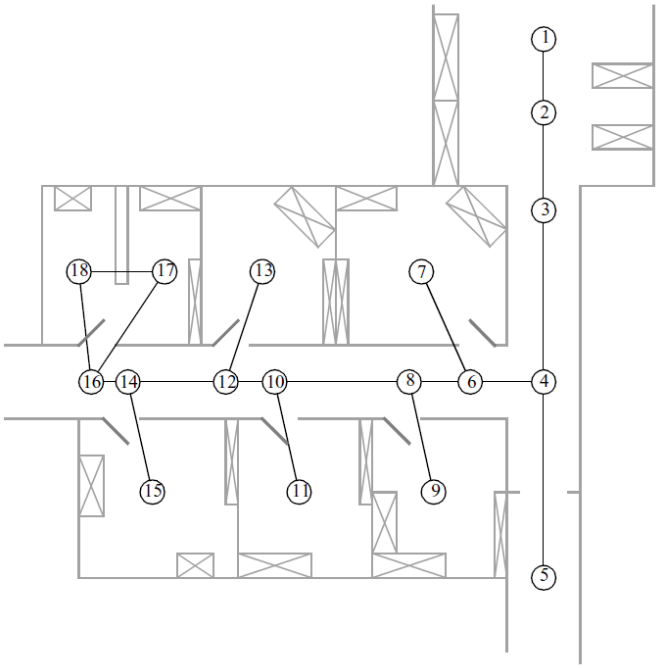


Figure 2.21: A topological map. Image courtesy of Roland Siegwart [Siegwart04]

2.5 SUMMARY

This chapter introduced motion models, perception models and maps. They are important known conditions for solving the robot localization problem.

- For motion models, we presented their deterministic forms and probabilistic forms, respectively. Probabilistic forms can be obtained by adding noise variables that characterize the types of uncertainty exiting in robotic actuation to deterministic models. Two deterministic models are introduced: the velocity motion model and the odometry motion model. The former represents controls by a translational and a rotational velocity. The latter uses odometric measurements to calculate the robot's motion. Odometric measurements can be decomposed into an initial rotation, followed by a translation, and a final rotation. Based on both motion models, we derived two probabilistic algorithms. One calculates the probability $p(s_t \,|u_t, s_{t-1})$ in a closed form. The other generates samples from the probability $p(s_t \,|u_t, s_{t-1})$, which is designed specially for particle filters.

- In the section of perception models, we introduced range finders and visual sensors. Based on these two types of sensors, we discussed feature extraction. We presented two probabilistic perception models. The first one is based on range finders, which characterizes the probability $p(z_t \,|s_t, m)$ with a mixture model that addressed four types of noise. An alternative one is based on features. The feature extracted from sensor measurements can be represented by the range, bearing and signature.

- We introduced two ways to represent maps: geometric maps and topological maps. The former is a concrete representation and the latter is an abstract graph-like representation. Two geometric maps are specified: the occupancy grid map and the line map. Since the line map requires less memory than the occupancy grid map in the implementation, we select it to use in simulations and experiments.

CHAPTER 3

MARKOV LOCALIZATION

Contents

3.1 INTRODUCTION

Probabilistic localization algorithms are variants of the Bayes filter. Markov localization is the straightforward application of Bayes filters to the localization problem [Thrun05]. Markov localization addresses the problem of state estimation by computing a probability distribution over all possible locations in the environment [Burgard97a, Fox98a, Fox98b, Gutmann98, Fox99b, Baltzakis02, Gutmann02, Baltzakis03, Thrun05]. In this chapter, we will present some classic Markov localization algorithms. The models discussed in previous chapter will be applied to concrete algorithms. Before introducing Markov localization, we briefly review the basic theory of the Bayes filter that is useful throughout the Markov localization. Some useful concepts of probability theory are presented in Appendix A.

3.1.1 Bayes filters

The Bayes filter technique provides a powerful statistical tool to understand and solve robot localization problems [Dellaert99, Roumeliotis00, Thrun00d, Fox03b, Fox03c, Bekkali08, Blanco08, Ko08]. It calculates recursively the belief distribution

$bel(*)$ from measurement and control data [Thrun05]. The Bayes filter makes a Markov assumption, that is, the past and future data are independent if one knows the current state.

Let $bel(s_t)$ denote the robot's subjective belief of being at position s_t at time t. Here, s_t is a three-dimensional variable $s_t = (x_t, y_t, \theta_t)^T$, comprising its $x - y$ coordinates in the Cartesian coordinate system and its orientation θ. The belief distribution is the posterior probability over the state s_t at time t, conditioned on all past measurements Z_t and all past controls U_t.

$$bel(s_t) = p(s_t \,|Z_t, U_t) \tag{3.1}$$

We define measurements Z_t and controls U_t as follows.

$$\begin{aligned} Z_t &= \{z_t, z_{t-1}, \cdots, z_0\}, \\ U_t &= \{u_t, u_{t-1}, \cdots, u_1\}. \end{aligned} \tag{3.2}$$

where controls U_t are often obtained from measurements of the proprioceptive sensor such as odometry.

We consider that the exteroceptive measurements and the odometry measurements are independent and we treat them separately.

$$bel(s_t) = p\,(s_t \,|Z_t) \cdot p\,(s_t \,|U_t) \tag{3.3}$$

The term $p\,(s_t \,|Z_t)$ is denoted as $bel_{sen}(s_t)$, which represents the posterior belief after integrating the perception data.

$$\begin{aligned} bel_{sen}(s_t) &= p\,(s_t \,|Z_t) \\ &\overset{Bayes\ rule}{=} \frac{p\,(z_t \,|s_t, Z_{t-1})\, p\,(s_t \,|Z_{t-1})}{p\,(z_t \,|Z_{t-1})} \\ &= \eta p\,(z_t \,|s_t, Z_{t-1})\, p\,(s_t \,|Z_{t-1}) \\ &\overset{Markov\ assum.}{=} \eta p\,(z_t \,|s_t)\, p\,(s_{t-1} \,|Z_{t-1}) \\ &= \eta p\,(z_t \,|s_t)\, bel_{sen}(s_{t-1}) \end{aligned} \tag{3.4}$$

where η is a normalization constant that ensures $bel_{sen}(s_t)$ to sum up to one.

The term $p\,(s_t \,|U_t)$ is denoted as $bel_{odo}(s_t)$, which represents the posterior belief after integrating the odometry data.

$$\begin{aligned} bel_{odo}(s_t) &= p\,(s_t \,|U_t) \\ &\overset{Total\ prob.}{=} \int p\,(s_t \,|s_{t-1}, U_t)\, p\,(s_{t-1} \,|U_t) ds_{t-1} \\ &\overset{Markov\ assum.}{=} \int p\,(s_t \,|s_{t-1}, u_t)\, p\,(s_{t-1} \,|U_{t-1}) ds_{t-1} \\ &= \int p\,(s_t \,|s_{t-1}, u_t)\, bel_{odo}(s_{t-1}) ds_{t-1} \end{aligned} \tag{3.5}$$

We multiply $bel_{sen}(s_t)$ by $bel_{odo}(s_t)$ to get the final localization formula:

$$\begin{aligned} bel(s_t) &= bel_{sen}(s_t) \cdot bel_{odo}(s_t) \\ &= \eta p\left(z_t \,|s_t\right) bel_{sen}(s_{t-1}) \cdot \int p\left(s_t \,|s_{t-1}, u_t\right) bel_{odo}(s_{t-1}) ds_{t-1} \\ &= \eta p\left(z_t \,|s_t\right) \int p\left(s_t \,|s_{t-1}, u_t\right) [bel_{sen}(s_{t-1}) \cdot bel_{odo}(s_{t-1})] ds_{t-1} \\ &= \eta p\left(z_t \,|s_t\right) \int p\left(s_t \,|s_{t-1}, u_t\right) bel(s_{t-1}) ds_{t-1} \end{aligned} \tag{3.6}$$

where the probability $p\left(s_t \,|s_{t-1}, u_t\right)$ is called the prediction model or the motion model, which denotes the transition of robot state. The probability $p\left(z_t \,|s_t\right)$ is the correction model or the sensor model, which incorporates sensor information to update robot state. Specific motion models and sensor models are introduced in Chapter 2.

In practice, the implementation of Equation 3.6 is divided into two stages: prediction and correction.

- **Prediction.** In this stage, a posterior, before incorporating the latest measurement z_t and just after executing the control u_t, is calculated. Such a posterior is denoted as follows:

$$\begin{aligned} \overline{bel}(s_t) &= p(s_t \,|Z_{t-1}, U_t) \\ &= \int p\left(s_t \,|s_{t-1}, u_t\right) bel(s_{t-1}) ds_{t-1} \end{aligned} \tag{3.7}$$

- **Correction.** In this stage, the latest measurement z_t is incorporated to calculate $bel(s_t)$ from $\overline{bel}(s_t)$.

$$\begin{aligned} bel(s_t) &= \eta p\left(z_t \,|s_t\right) \overline{bel}(s_t) \\ &= \eta p\left(z_t \,|s_t\right) \int p\left(s_t \,|s_{t-1}, u_t\right) bel(s_{t-1}) ds_{t-1} \end{aligned} \tag{3.8}$$

3.1.2 Markov localization algorithm

The straightforward application of Bayes filters to the localization problem is called Markov localization. It requires a map m as input and assumes this map is static and Markov (or complete) [Thrun05].

Algorithm 3.1 depicts the basic Markov localization algorithm. This algorithm is recursive, that is , the belief $bel(s_t)$ at time t is calculated from the belief $bel(s_{t-1})$ at time $t-1$. It inputs the belief $bel(s_{t-1})$ at time $t-1$, along with the most recent control u_t, the most recent measurement z_t and the map m. It outputs the belief

$bel(s_t)$ at time t. Line 3 implements the prediction stage. It calculates a belief over the pose s_t based on the prior belief over the pose s_{t-1}, the control u_t and the map m. Line 4 implements the correction stage. It updates the posterior by multiplying the belief $\overline{bel}(s_t)$ by the correction model $p(z_t \,|s_t, m)$ that the latest measurement z_t has been observed. The result is normalized by the normalization constant η.

Algorithm 3.1: Markov localization algorithm, adapted from [Thrun05]

1: **Input:** $bel(s_{t-1}), u_t, z_t, m$
2: **for** all s_t **do**
3: $\quad \overline{bel}(s_t) = \int p(s_t \,|s_{t-1}, u_t, m) bel(s_{t-1}) ds_{t-1}$ *% prediction*
4: $\quad bel(s_t) = \eta p(z_t \,|s_t, m)\, \overline{bel}(s_t)$ *% correction*
5: **end for**
6: **Output:** $bel(s_t)$

The localization problem can be divided into three sub-problems: position tracking, global localization and the kidnapped robot problem [Roumeliotis00, Thrun00a, Thrun05]. Markov localization is designed to solve the position tracking problem, the global localization problem and the kidnapped robot problem. Position tracking assumes that the robot knows its initial pose [Schiele94, Weiss94]. During its motions, the robot can keep track of movement to maintain a precise estimate of its pose by accommodating the relatively small noise in a known environment. More challenging is the global localization problem [Thrun00a, Milstein02]. In this case, the robot does not know its initial pose, thus it has to determine its pose in the following process only with control data and sensors data. Once the robot determines its global position, the process continues as a position tracking problem. The kidnapped robot problem considers that a well-localized robot is teleported to some other place without being told [Thrun00a, Thrun00b, Thrun05]. In practice, the robot is rarely kidnapped. However, kidnapping tests the ability of a localization algorithm to recover from global localization failures. This problem is more difficult than global localization. Difficulties come from two sources: one is how a robot knows it is kidnapped, the other is how to recover from kidnapping. The latter can be processed as a global localization problem.

Since the basic Markov localization algorithm is recursive, computing the posterior belief $bel(s_t)$ requires an initial belief $bel(s_0)$ at time $t = 0$ as boundary condition. Thus, the initial belief $bel(s_0)$ is set differently according to the initial knowledge of the robot's pose [Thrun05].

- **Known initial pose.** The position tracking problem assumes that the robot knows its initial pose, thus $bel(s_0)$ is initialized by a point-mass distribution. The known initial pose of the robot is denoted by $\tilde{s}_0$.

$$bel(s_0) = \begin{cases} 1 & \text{if } s_0 = \tilde{s}_0 \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

In practice, the initial pose of the robot is often known approximately, thus $bel(s_0)$ is usually initialized by a narrow Gaussian distribution centered around the known initial pose $\tilde{s}_0$ with the covariance Σ (Equation A.5).

$$bel(s_0) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\{-\frac{1}{2}(s_0 - \tilde{s}_0)^T \Sigma^{-1}(s_0 - \tilde{s}_0)\} \tag{3.10}$$

- **Unknown initial pose.** In the global localization problem, the robot is not told its initial pose, thus $bel(s_0)$ is initialized by a uniform distribution over the state space to reflect the global uncertainty of the robot.

$$bel(s_0) = \frac{1}{|S|} \tag{3.11}$$

where $|S|$ represents the volume (Lebesgue measure) of the state space of all poses in the map.

- **Partially known initial pose.** If the robot has the partial knowledge about its initial pose, such as a certain area, $bel(s_0)$ may be initialized by a uniform distribution over this area and zero anywhere else.

Extended Kalman Filter (EKF), Grid localization and Monte Carlo Localization (MCL) are three classic Markov localization algorithms. Among the existing position tracking algorithms, the Extended Kalman Filter (EKF) is one of the most popular approaches [Kalman60, Leonard91, Grewal93, Thrun05]. EKF assumes that the state transition and the measurements are Markov processes represented by nonlinear functions. The first step consists in linearizing these functions by Taylor expansion and the second step consists in a fusion of sensors and odometry data with Kalman Filter. However, plain EKF is inapplicable to the global localization problem, because of the restrictive nature of the unimodal belief representation. To overcome this limitation, the multi-hypothesis Kalman filter is proposed [Cox94, Reuter00, Roumeliotis00, Jensfelt01]. It represents beliefs using the mixture of Gaussian distributions, thus it can proceed with multiple and distinct hypotheses. However, this approach inherits the Gaussian noise assumption from Kalman filters. This assumption makes all practical implementations extract low-dimensional features from the sensor data, thereby ignoring much of the information acquired by the robot's sensors [Thrun00b].

Grid localization and MCL are two most common approaches to deal with the global localization problem. Grid localization approximates the posterior using a histogram filter over a grid decomposition of the pose space [Borenstein91b, Borenstein91a, Burgard96, Burgard97b, Thrun05]. MCL is based on a particle filter that

represents the posterior belief by a set of weighted samples (also called particles) distributed according to this posterior [Metropolis49,Dellaert99,Fox99a,Thrun99b, Fox00a, Thrun00c, Kwok04, Thrun05, Siagian07, Hester08, Prestes08]. The crucial disadvantage of these two approaches is that they bear heavy on-line computational burden. For grid localization, the resolution of the grid is a key variable. The precision and efficiency of implementation depend on it. The finer grained can get a more accurate result, but at the expense of increased computational costs. The implementation of MCL is more efficient than Grid localization, because it only calculates the posteriors of particles. However, to obtain a reliable localization result, a certain number of particles will be needed. The larger the environment is, the more particles are needed. Actually each particle can be seen as a pseudo-robot, which perceives the environment using a probabilistic measurement model. At each iteration, the virtual measurement takes large computational costs if there are hundreds of particles. Furthermore, the fact that MCL cannot recover from robot kidnapping is its another disadvantage. When the position of the robot is well determined, samples only survive near a single pose. If this pose happens to be incorrect, MCL is unable to recover from this global localization failure.

Thrun *et al.* [Thrun05] propose the Augmented_MCL algorithm to solve the kidnapped robot problem by adding random samples. However, adding random samples can cause the extension of the particle set if the algorithm cannot recover quickly from kidnapping. This algorithm draws particles either according to a uniform distribution over the pose space or according to the measurement distribution. The former is inefficient and the latter can only fit the landmark detection model (feature-based localization). Moreover, by augmenting the sample set through uniformly distributed samples is mathematically questionable. Thus, Thrun *et al.* [Thrun00d, Thrun00b, Thrun05] propose the Mixture MCL algorithm. This algorithm employs a mixture proposal distribution that combines regular MCL sampling with an inversed MCL's sampling process. They think that the key disadvantage of Mixture MCL is a requirement for a sensor model that permits fast sampling of poses. To overcome this difficulty, they use sufficient statistics and density trees to learn a sampling model from data.

Different localization approaches represent this posterior $bel(s_t)$ in different ways. The Kalman filter is a Gaussian filter, and represents the posterior belief $bel(s_t)$ by its mean μ_t and its covariance Σ_t

$$bel(s_t) = \det(2\pi\Sigma_t)^{-\frac{1}{2}}\exp\{-\frac{1}{2}(s_t-\mu_t)^T\Sigma_t^{-1}(s_t-\mu_t)\} \tag{3.12}$$

Grid localization uses a collection of discrete probability values $\{p_{k,t}\}$ to represent the posterior $bel(s_t)$ over a grid decomposition of the pose space:

$$bel(s_t) \sim \{p_{k,t}\}_{k=1,\cdots,K} \tag{3.13}$$

where each probability $p_{k,t}$ is defined over a grid cell.

MCL represents the posterior belief $bel(s_t)$ by a set of N weighted particles distributed according to this posterior:

$$bel(s_t) \sim \left\{ \left\langle s_t^{[n]}, \omega_t^{[n]} \right\rangle \right\}_{n=1,\cdots,N} \tag{3.14}$$

where $\omega_t^{[n]}$ is an importance factor.

3.2 EXTENDED KALMAN FILTER LOCALIZATION

The Kalman filter was invented in 1960 by Rudolph E. Kalman as a recursive solution to linear filtering and prediction problems [Kalman60]. But now, the Kalman filter has been extensively used to solve the mobile robot localization problems, particularly the position tracking problem [Leonard91, Arsenio98, Gutmann98, Roumeliotis00, Jensfelt01].

The Kalman filter represents the posterior belief $bel(s_t)$ by a Gaussian with mean μ_t and covariance Σ_t, if the following three conditions are fulfilled [Thrun05].

1. The next state must be a linear function of the previous state and the control with added Gaussian noise.

   $$s_t = A_t s_{t-1} + B_t u_t + w_t \tag{3.15}$$

   where s_t and s_{t-1} are state vectors, u_t is the control vector at time t. A_t is a square matrix of size $n \times n$ and B_t is a matrix of size $n \times r$. n is the dimension of the state vector s_t. If s_t represents a robot's pose, then $n = 3$. r is the dimension of the control vector u_t. The vector w_t describes the motion noise, which is a Gaussian random vector and has the same dimension as the state vector.

   $$p(w_t) \sim \mathcal{N}(w_t; 0, R_t) \tag{3.16}$$

   where $\mathcal{N}(w_t; 0, R_t)$ denotes the Gaussian distribution with mean zero and covariance R_t (an $n \times n$ matrix).

   By combining Equation 3.15 with the definition of the multivariate normal distribution (Equation A.5), the motion model of the Bayes filter can be described as follows

   $$\begin{aligned} p\left(s_t \,|s_{t-1}, u_t\right) = & \det(2\pi R_t)^{-\frac{1}{2}} \\ & \exp\{-\frac{1}{2}(s_t - A_t s_{t-1} - B_t u_t)^T R_t^{-1}(s_t - A_t s_{t-1} - B_t u_t)\} \end{aligned} \tag{3.17}$$

2. Observations must be linear functions of the state with added Gaussian noise.

   $$z_t = C_t s_t + v_t \tag{3.18}$$

where C_t is a matrix of size $k \times n$. k is the dimension of the measurement vector z_t. The vector v_t describes the measurement noise, which has the same dimension as the measurement vector. Its distribution is a multivariate Gaussian with zero mean and covariance Q_t (a $k \times k$ matrix).

$$p(v_t) \sim \mathcal{N}(v_t; 0, Q_t) \tag{3.19}$$

The perception model can be represented by the following multivariate Gaussian.

$$p\left(z_t \,|s_t\right) = \det(2\pi Q_t)^{-\frac{1}{2}} \exp\{-\frac{1}{2}(z_t - C_t s_t)^T Q_t^{-1}(z_t - C_t s_t)\} \tag{3.20}$$

3. The initial belief $bel(s_0)$ must be Gaussian. Its mean can be denoted by μ_0 and covariance by Σ_0.

$$bel(s_0) = \det(2\pi\Sigma_0)^{-\frac{1}{2}} \exp\{-\frac{1}{2}(s_0 - \mu_0)^T \Sigma_0^{-1}(s_0 - \mu_0)\} \tag{3.21}$$

If all above three assumptions are met, the posterior of the Bayes filter is a Gaussian for any point in time t.

$$bel(s_t) = \det(2\pi\Sigma_t)^{-\frac{1}{2}} \exp\{-\frac{1}{2}(s_t - \mu_t)^T \Sigma_t^{-1}(s_t - \mu_t)\} \tag{3.22}$$

However, motion models and perception models are nonlinear in practice. The plain Kalman filter is inapplicable in such situations. A Kalman filter that linearizes about the current mean and covariance is referred to as an extended Kalman filter or EKF [Maybeck79]. Now, the state transition and the measurement are governed by nonlinear functions g and h, respectively.

$$s_t = g(u_t, s_{t-1}) + w_t \tag{3.23}$$

$$z_t = h(s_t) + v_t \tag{3.24}$$

where the random variables w_t and v_t again represent the state transition noise and the measurement noise as in Equation 3.15 and Equation 3.18. The nonlinear function g relates the state transition from time $t-1$ to time t under the control u_t. The nonlinear function h relates the measurement z_t and the state s_t.

The EKF uses Taylor expansion to linearize nonlinear functions. The Taylor expansion is a representation of a function as an infinite sum of terms calculated from the values of its derivatives at a single point. For the nonlinear function g, the derivative is given by its partial derivative and the point is chose at the mean of the posterior μ_{t-1}.

$$g(u_t, s_{t-1}) \approx g(u_t, \mu_{t-1}) + \frac{g'(u_t, \mu_{t-1})}{1!}(s_{t-1} - \mu_{t-1}) \tag{3.25}$$

where

$$\begin{aligned} g'(u_t, \mu_{t-1}) &:= \left.\frac{\partial g(u_t, s_{t-1})}{\partial s_{t-1}}\right|_{s_{t-1}=\mu_{t-1}} \\ &= G_t \end{aligned} \tag{3.26}$$

where G_t is called Jacobian of the function g, which is a matrix of size $n \times n$. n is the dimension of the state s_t. In practical applications, we ignore high order terms in Equation 3.25.

$$g(u_t, s_{t-1}) \approx g(u_t, \mu_{t-1}) + G_t(s_{t-1} - \mu_{t-1}) \tag{3.27}$$

The motion model is thus written by the following multivariate normal distribution.

$$\begin{aligned} p\left(s_t \,|s_{t-1}, u_t\right) \approx \det(2\pi R_t)^{-\frac{1}{2}} \exp\{-\frac{1}{2}(s_t - g(u_t, \mu_{t-1}) - G_t(s_{t-1} - \mu_{t-1}))^T \\ R_t^{-1}(s_t - g(u_t, \mu_{t-1}) - G_t(s_{t-1} - \mu_{t-1}))\} \end{aligned} \tag{3.28}$$

For the measurement function h, the same linearization is implemented. The Taylor expansion is developed around $\bar{\mu}_t$, which is considered as the most likely state at time of linearization.

$$h(s_t) \approx h(\bar{\mu}_t) + h'(\bar{\mu}_t)(s_t - \bar{\mu}_t) \tag{3.29}$$

where

$$\begin{aligned} h'(\bar{\mu}_t) &:= \left.\frac{\partial h(s_t)}{\partial s_t}\right|_{s_t=\bar{\mu}_t} \\ &= H_t \end{aligned} \tag{3.30}$$

where H_t is the Jacobian matrix of the measurement function with the dimension of $k \times n$.

Thus, the perception model can be written by a Gaussian as follows

$$\begin{aligned} p\left(z_t \,|s_t\right) \approx \det(2\pi Q_t)^{-\frac{1}{2}} \exp\{-\frac{1}{2}(z_t - h(\bar{\mu}_t) - H_t(s_t - \bar{\mu}_t))^T \\ Q_t^{-1}(z_t - h(\bar{\mu}_t) - H_t(s_t - \bar{\mu}_t))\} \end{aligned} \tag{3.31}$$

The EKF algorithm is described in Algorithm 3.2. It accepts as input the belief represented by μ_{t-1} and Σ_{t-1} at time $t-1$, the control u_t and the measurement

z_t. It outputs the belief represented by μ_t and Σ_t at time t. This algorithm is implemented in two steps: prediction (or the control update) and correction (or the measurement update), which correspond to the Bayes filter (see Figure 3.1).

1. The prediction step, including lines 2 and line 3, calculates the predicted belief $\overline{bel}(s_t)$ represented by $\bar{\mu}_{t-1}$ and $\bar{\Sigma}_{t-1}$. This belief is obtained by incorporating the control u_t, before incorporating the measurement z_t.

2. The correction step, including lines 4 to 8, calculates the predicted belief $\overline{bel}(s_t)$ represented by $\bar{\mu}_{t-1}$ and $\bar{\Sigma}_{t-1}$. This belief is obtained by incorporating the control u_t, before incorporating the measurement z_t. The variable $\hat{d}_t$, computed in line 4 is called the measurement innovation (or residual). It reflects the discrepancy between the predicted measurement $h(\bar{\mu}_t)$ and the actual measurement z_t. Line 5 calculates the innovation (or residual) covariance D_t. Line 6 computes a variable K_t that is called Kalman gain. It specifies to what extent the innovation should be taken into account in the posterior state estimate. The new mean of the posterior belief is calculated in line 7, by adjusting it in proportion to the Kalman gain K_t and the innovation $\hat{d}_t$. Finally, the new covariance of the posterior belief is determined in line 8.

Algorithm 3.2: Extended Kalman filter algorithm

1: **Input:** $\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$
2: $\bar{\mu}_t = g(u_t, \mu_{t-1})$
3: $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$ *% prediction*
4: $\hat{d}_t = z_t - h(\bar{\mu}_t)$
5: $D_t = H_t \bar{\Sigma}_t H_t^T + Q_t$
6: $K_t = \bar{\Sigma}_t H_t^T D_t^{-1}$
7: $\mu_t = \bar{\mu}_t + K_t \hat{d}_t$
8: $\Sigma_t = (I - K_t H_t) \bar{\Sigma}_t$ *% correction*
9: **Output:** μ_t, Σ_t

If the EKF algorithm is implemented accurately and the initial values μ_0 and Σ_0 can reflect accurately the distribution of the initial state, there will be some properties [Wik].

$$\mathrm{E}(s_t - \mu_t) = \mathrm{E}(s_t - \bar{\mu}_t) = 0 \tag{3.32}$$

$$\mathrm{E}(\hat{d}_t) = 0 \tag{3.33}$$

Equation 3.32 and Equation 3.32 reflect that expected values (mean errors) of all estimates are zero.

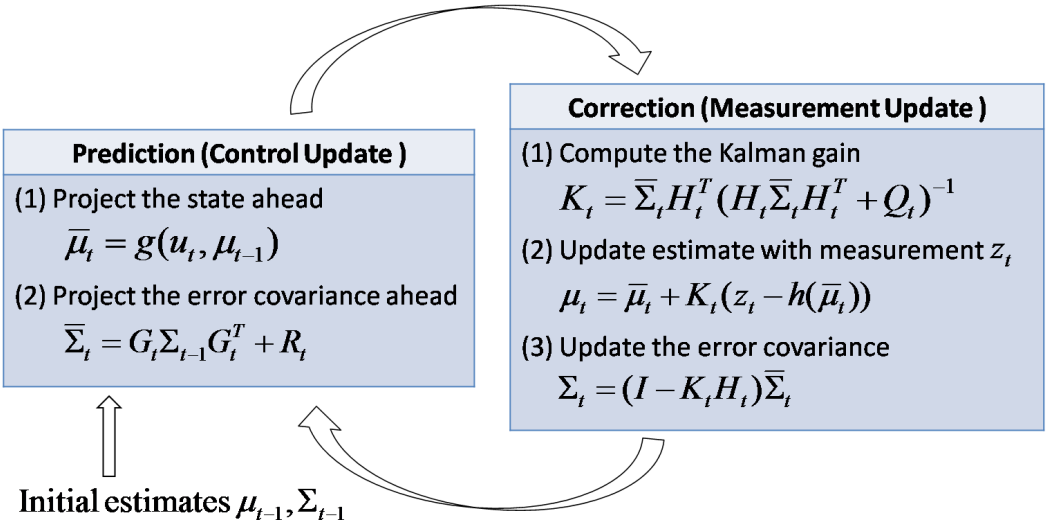


Figure 3.1: A complete diagram of the operation of the extended Kalman filter, adapted from [Welch95].

$$\Sigma_t = \mathrm{Cov}(s_t - \mu_t) \tag{3.34}$$

$$\bar{\Sigma}_t = \mathrm{Cov}(s_t - \bar{\mu}_t) \tag{3.35}$$

$$D_t = \mathrm{Cov}(\hat{d}_t) \tag{3.36}$$

Equation 3.34, 3.35 and Equation 3.36 show that covariance matrices accurately reflect the covariance of estimates.

The EKF discussed so far is a general situation. In order to extract a EKF localization algorithm, we need to select the proper motion model and perception model. We now discuss a concrete implementation of the EKF for the velocity motion model and the feature-based perception model (see Chapter 2). Here, we only briefly restate the definition. The velocity motion model is defined as

$$\begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix} = \begin{pmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{pmatrix} + \begin{pmatrix} \tilde{v}_t cos(\theta)\Delta t \\ \tilde{v}_t sin(\theta)\Delta t \\ \tilde{\omega}_t \Delta t \end{pmatrix} \tag{3.37}$$

where $\tilde{v}_t$ and $\tilde{\omega}_t$ denote the true translational velocity and rotational velocity, respectively. They are generated by motion control, $u_t = (v_t, \omega_t)^T$, with added Gaussian noise.

$$\begin{pmatrix} \tilde{v}_t \\ \tilde{\omega}_t \end{pmatrix} = \begin{pmatrix} v_t \\ \omega_t \end{pmatrix} + \begin{pmatrix} \epsilon_{\sigma_1^2} \\ \epsilon_{\sigma_2^2} \end{pmatrix} = \begin{pmatrix} v_t \\ \omega_t \end{pmatrix} + \mathcal{N}(0, M_t) \tag{3.38}$$

where $\epsilon_{\sigma_1^2}$ and $\epsilon_{\sigma_2^2}$ are two independent Gaussian error variables with zero-mean and standard deviations σ_1 and σ_2, respectively. σ_1 and σ_2 are two variables relative to the control velocities v and ω.

Thus, the motion model can be decomposed into a noise-free model with a random Gaussian noise.

$$\underbrace{\begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix}}_{s_t} = \underbrace{\begin{pmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{pmatrix} + \begin{pmatrix} v_t cos(\theta)\Delta t \\ v_t sin(\theta)\Delta t \\ \omega_t \Delta t \end{pmatrix}}_{g(u_t, s_{t-1})} + \mathcal{N}(0, R_t) \tag{3.39}$$

The same decomposition is applied to the feature-based perception model. The correspondence is assumed to be known via the correspondence variable c_t. Let $j = c_t^i$ be the identity of the i^{th} feature observed at time t corresponds to the j^{th} landmark in the map.

$$\underbrace{\begin{pmatrix} r_t^i \\ \phi_t^i \\ \tau_t^i \end{pmatrix}}_{z_t^i} = \underbrace{\begin{pmatrix} \sqrt{(m_{j,x} - x_t)^2 + (m_{j,y} - y_t)^2} \\ atan2(m_{j,y} - y_t, m_{j,x} - x_t) - \theta_t \\ m_{j,\tau} \end{pmatrix}}_{h(s_t, j, m)} + \mathcal{N}(0, Q_t) \tag{3.40}$$

where $m_{j,x}$ and $m_{j,y}$ denote the coordinates of the i^{th} landmark detected by the robot (that is identical to j^{th} landmark in the map); $m_{j,\tau}$ is its signature.

The EKF localization algorithm is depicted in Algorithm 3.3, which assumes knowledge of the landmark correspondences. This algorithm is derived from the EKF algorithm (Algorithm 3.2). The input is the posterior belief of the robot pose at time $t-1$, represented by the mean μ_{t-1} and the covariance Σ_{t-1}. To update these parameters, it requires the control u_t, a set of feature measurements z_t along with the correspondence variables c_t at time t, and the map m. The output is a new estimate of the robot pose at time t, represented by μ_t and Σ_t. Like the basic EKF algorithm, it also implements in two steps: prediction and correction.

1. Prediction includes lines 2 to 6. Lines 2 and 3 compute the necessary Jacobians for linearizing the motion model. Line 4 determines the motion noise covariance matrix in control space, where σ_1 and σ_2 are two variables relative to the control velocities. Lines 5 and 6 implement the motion update and the result is the predicted belief $\overline{bel}(s_t)$ represented by $\bar{\mu}_t$ and $\bar{\Sigma}_t$. The multiplication $V_t M_t V_t^T$ in line 6 translates the motion noise from control space to state space.

2. Correction includes lines 7 to 19. Line 7 computes the measurement noise covariance matrix. Lines 8 to 17 implement the measurement update in a loop through all features observed at time t. Line 9 represents the i^{th} feature observed by the robot is the j^{th} landmark in the map. Lines 10 to 12 calculate a predicted measurement $\tilde{z}_t^i$ and the Jacobian of the measurement model. Line 13 calculates the innovation covariance D_t^i, which depicts the uncertainty corresponding to the predicted measurement $\tilde{z}_t^i$. The Kalman gain K_t^i is calculated in line 14. Lines 15 and 17 update the mean $\bar{\mu}_t$ and the covariance $\bar{\Sigma}_t$. Finally, the new pose estimate is obtained in lines 18 and 19.

Algorithm 3.3: EKF localization algorithm with known correspondences

1: **Input:** $\mu_{t-1}, \Sigma_{t-1}, u_t, z_t, c_t, m$

2: calculate Jacobian $G_t = \frac{\partial g(u_t, s_{t-1})}{\partial_{s_{t-1}}}\Big|_{s_{t-1}=\mu_{t-1}}$

3: calculate Jacobian $V_t = \frac{\partial g(u_t, s_{t-1})}{\partial_{u_t}}\Big|_{s_{t-1}=\mu_{t-1}}$

4: $M_t = \begin{pmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{pmatrix}$

5: $\bar{\mu}_t = \mu_{t-1} + \begin{pmatrix} v_t \cos(\mu_{t-1,\theta})\Delta t \\ v_t \sin(\mu_{t-1,\theta})\Delta t \\ \omega_t \Delta t \end{pmatrix}$

6: $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + V_t M_t V_t^T$

7: $Q_t = \begin{pmatrix} \sigma_r^2 & 0 & 0 \\ 0 & \sigma_\phi^2 & 0 \\ 0 & 0 & \sigma_\tau^2 \end{pmatrix}$

8: **for** all observed features $z_t^i = (r_t^i, \phi_t^i, \tau_t^i)^T$ **do**

9: $j = c_t^i$

10: $\tilde{z}_t^i = \begin{pmatrix} \sqrt{(m_{j,x} - \bar{\mu}_{t,x})^2 + (m_{j,y} - \bar{\mu}_{t,y})^2} \\ atan2(m_{j,y} - \bar{\mu}_{t,y}, m_{j,x} - \bar{\mu}_{t,x}) - \bar{\mu}_{t,\theta} \\ m_{j,\tau} \end{pmatrix}$

11: calculate Jacobian $H_t^i = \frac{\partial h(s_{t,j,m})}{\partial s_t}\Big|_{s_t=\bar{\mu}_t}$

12: $D_t^i = H_t^i \bar{\Sigma}_t [H_t^i]^T + Q_t$

13: $K_t^i = \bar{\Sigma}_t [H_t^i]^T [D_t^i]^{-1}$

14: $\bar{\mu}_t = \bar{\mu}_t + K_t^i (z_t^i - \tilde{z}_t^i)$

15: $\bar{\Sigma}_t = (I - K_t^i H_t^i) \bar{\Sigma}_t$

16: **end for**

17: $\mu_t = \bar{\mu}_t$

18: $\Sigma_t = \bar{\Sigma}_t$

19: **Output:** μ_t, Σ_t

The EKF offers an elegant and efficient tool for state estimation in robotics. Its strength lies in its simplicity and its computational efficiency [Thrun05]. The computational efficiency arises from the fact that it represents the belief by a multivariate Gaussian distribution. However, a unimodal Gaussian is usually a good representation of uncertainty in the position tracking problem whereas it is not in the global localization problem. Furthermore, linearized Gaussian techniques tend to work well only if the position uncertainty is small. Some multihypothesis extensions make EKF accommodate in the global localization problem [Cox94, Reuter00, Roumeliotis00, Jensfelt01, Thrun05].

3.3 GRID LOCALIZATION

Grid localization uses a histogram filter to approximate the posterior over a grid decomposition of the state space [Thrun05]. The histogram filter decomposes a continuous state space into finitely many bins or regions.

$$\begin{aligned} \mathbf{dom}(S_t) &= \psi_{1,t} \cup \psi_{2,t} \cup \cdots \cup \psi_{K,t} \\ &= \bigcup_k \psi_{k,t} \end{aligned} \tag{3.41}$$

where S_t is the random variable describing the state of the robot at time t. The function $\mathbf{dom}(S_t)$ denotes the state space. Each $\psi_{k,t}$ represents a convex region. All these regions together form a partition of the state space. For each $i \neq k$ we have

$$\psi_{i,t} \cap \psi_{k,t} = \emptyset \tag{3.42}$$

For grid localization, each $\psi_{k,t}$ represents a grid cell.

Figure 3.2 illustrates the histogram representation of the normal distribution. This is an example how a histogram filter represents a random variable. Within each region ψ_k, we assign a uniform probability to each state s. Here, we distinguish probability P from the probability density p.

$$P(\psi_k) = p(s_t) \times |\psi_k| \tag{3.43}$$

where $|\psi_k|$ is the volume of the region ψ_k.

The value of the uniform probability is usually approximated by the density of the mean state in region ψ_k that is denoted as $\hat{s}_k$. Thus, geometrically the probability within region ψ_k is equal approximately to the acreage of the small rectangle region.

$$P(\psi_k) = p(\hat{s}_k) \times |\psi_k| \approx \int_{\psi_k} p(s) ds \tag{3.44}$$

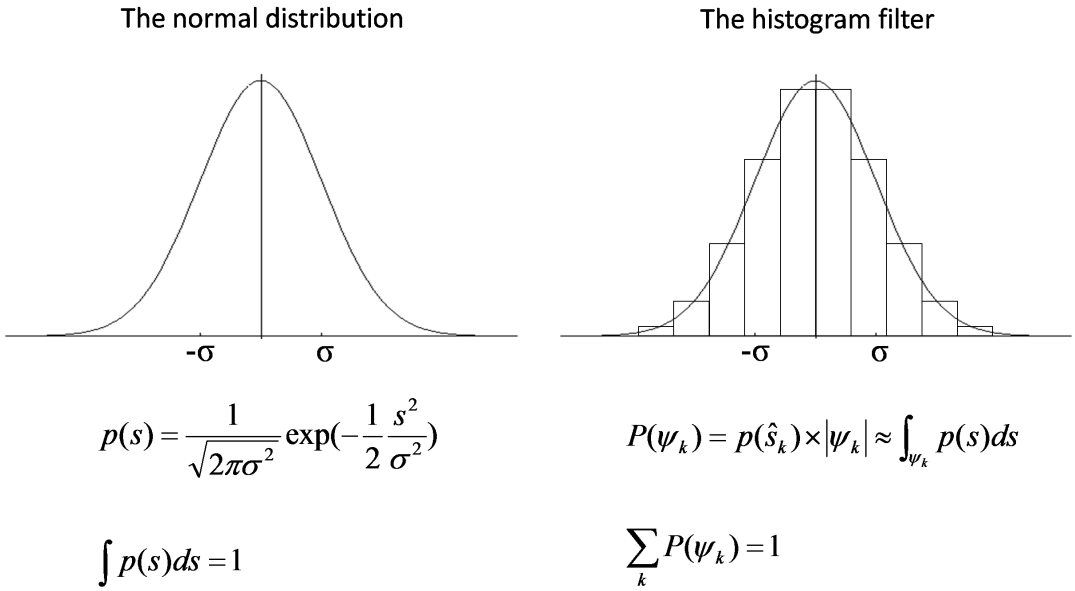
where $\hat{s}_k$ is the mean state in region ψ_k, which may be represented by

$$\hat{s}_k = |\psi_k|^{-1} \int_{\psi_k} s ds \tag{3.45}$$

In the Markov localization, two key probabilities should be calculated, which are the motion model and the perception model. Grid localization just computes the two probabilities as follows

$$p(\psi_{k,t} \,|u_t, \psi_{i,t-1}) \approx \eta \, |\psi_{k,t}| \, p(\hat{s}_{k,t} \,|u_t, \hat{s}_{i,t-1}) \tag{3.46}$$

where η is the normalizer to ensure that this approximation is a valid probability distribution.



(a) (b)

Figure 3.2: Histogram representation of the normal distribution.

$$p(z_t \,|\psi_{k,t}) \approx p(z_t \,|\hat{s}_{k,t}) \tag{3.47}$$

The grid localization algorithm is depicted in Algorithm 3.4. It is derived from the Markov localization algorithm (see Algorithm 3.1). The input of the algorithm is the discrete probability distribution $\{p_{k,t-1}\}$, along with the most recent control u_t, measurement z_t and the map m. It outputs the discrete probability values $p_{k,t}$ at time t. The variables ψ_k and ψ_i denote two individual grid cells, thus there are two inner loop iterating through all grid cells. Line 3 calculates the prediction based on a motion model (motion update). This prediction is then updated in line 4 based on a perception model (perception update). Specific motion models and perception models can be found in Chapter 2. The final probabilities $p_{k,t}$ are normalized in line 6.

Algorithm 3.4: Grid localization algorithm, adapted from [Thrun05]

1: **Input:** $\{p_{k,t-1}\}_{k=1,\cdots,K}, u_t, z_t, m$
2: **for** $k = 1$ to K **do**
3: $\quad \overline{p}_{k,t} = \sum_i p(S_t = \psi_k \,|S_{t-1} = \psi_i, u_t, m) p_{i,t-1}$ *% prediction*
4: $\quad p_{k,t} = p\,(z_t \,|S_t = \psi_k, m)\,\overline{p}_{k,t}$ *% correction*
5: **end for**
6: normalize $p_{k,t}$
7: **Output:** $\{p_{k,t}\}_{k=1,\cdots,K}$

Grid localization is a nonparametric filter, so it has the ability to represent multi-modal distributions. Furthermore, it always maintains a global multi-modal distributions over the whole environment. Thus, it can solve the position tracking problem, the global localization problem and the kidnapped robot problem. However, the key disadvantage of grid localization is the computational complexity. The localization accuracy and the computational efficiency are two contrary variables. To obtain a more accurate result, we need a finer decomposition. But such a decomposition can cause computational costs to be increased exponentially. Moreover, to represent the robot's pose, a three-dimensional grid is needed. In the algorithm 3.4, the motion update requires a convolution, which is a six-dimensional operation for the three-dimensional grid. Therefore, the basic grid localization algorithm cannot be executed in real-time. To reduce the computational complexity of grid localization, a number of techniques are introduced in [Thrun05], such as *pre-caching technique*, *sensor subsampling*, *delayed motion updates* and *selective updating*.

Our studies focus on the pre-caching technique. It decomposes the state space of the robot into grid and pre-computes measurements for each grid cell. Measurement results are cached in memory. When the algorithm is implemented, instead of the measurement operation, a much faster table lookup is used. We develop this technique into Monte Carlo localization that will be introduced in next section.

3.4 MONTE CARLO LOCALIZATION

Monte Carlo Localization (MCL) is based on a particle filter, which represents the posterior belief $bel(s_t)$ by a set of N weighted samples S_t distributed according to this posterior. As a consequence, the more intensive the region is populated by samples, the more likely the robot locates there.

$$S_t = \left\{ \left\langle s_t^{[n]}, \omega_t^{[n]} \right\rangle \right\}_{n=1,\cdots,N} \tag{3.48}$$

where S_t is a particle set. Each particle $s_t^{[n]}$ with $1 \leq n \leq N$ denotes a concrete instantiation of the robot's pose at time t. N is the number of particles in the particle set S_t. It may be a fixed value or changing with some quantities related to the belief $bel(s_t)$ [Fox01, Fox03a, Blanco08]. The $\omega_t^{[n]}$ is the non-negative numerical factor called importance factor. We interpret $\omega_t^{[n]}$ as the weight of a particle.

As MCL is a Bayes-based Markov localization algorithm, it calculates the particle set S_t recursively from the set S_{t-1}. The basic MCL algorithm is depicted in Algorithm 3.5. It represents the posterior belief $bel(s_t)$ by a set of particles S_t, therefore it accepts as input a particle set S_{t-1} along with the latest control u_t, measurement z_t and the map m. It outputs the particle set S_t at time t. $\bar{S}_t$ is a temporary particle set, which represents the belief $\overline{bel}(s_t)$. Before each iteration, we empty the temporary particle set $\bar{S}_t$ and the particle set S_t. This recursive algorithm is realized in three steps.

1. Line 4 generates a sample $s_t^{[n]}$ based on the sample $s_{t-1}^{[n]}$, the control u_t and

the map m. It is implemented according to sampling algorithms of motion models presented in Section 2.2.2. Obviously, the pair $\left(s_t^{[n]}, s_{t-1}^{[n]}\right)$ is distributed according to the product distribution.

$$p\left(s_t^{[n]} \left| s_{t-1}^{[n]}, u_t, m\right.\right) \times bel(s_{t-1}^{[n]}) \tag{3.49}$$

In accordance with the literature on the Sampling Importance Resampling (SIR) algorithm [Smith92, Doucet00], this distribution is called the proposal distribution. It corresponds to the Equation 3.7 of Bayes filters except for the absence of the integral sign.

2. Line 5 calculates the importance factor $\omega_t^{[n]}$ for each particle $s_t^{[n]}$. The important factor is used to correct the mismatch between the proposal distribution and the desired target distribution specified in Equation 3.8. It is restated here for the MCL algorithm.

$$\eta p\left(z_t \left| s_t^{[n]}\right.\right) p\left(s_t^{[n]} \left| s_{t-1}^{[n]}, u_t, m\right.\right) bel(s_{t-1}^{[n]}) \tag{3.50}$$

Thus, the importance factor $\omega_t^{[n]}$ is the probability of the measurement z_t under the a hypothetical state $s_t^{[n]}$, which incorporates the measurement z_t into the particle set.

$$\begin{aligned}
\omega_t^{[n]} &= \frac{\text{target distribution}}{\text{proposal distribution}} \\
&= \frac{\eta p\left(z_t \left| s_t^{[n]}\right.\right) p\left(s_t^{[n]} \left| s_{t-1}^{[n]}, u_t, m\right.\right) bel(s_{t-1}^{[n]})}{p\left(s_t^{[n]} \left| s_{t-1}^{[n]}, u_t, m\right.\right) bel(s_{t-1}^{[n]})} \\
&= \eta p\left(z_t \left| s_t^{[n]}\right.\right)
\end{aligned} \tag{3.51}$$

where the normalization η is a constant, which plays no role in the computation since the resampling takes place with probabilities proportional to the importance weights [Thrun05].

The process of calculating the importance factor is the measurement update, which can be implemented based on probabilistic algorithms of perception models introduced in Section 2.3.4. The importance factor $\omega_t^{[n]}$ can be seen as the weight of a particle $s_t^{[n]}$. Thus, the weighted particle set $\bar{S}_t$ can represent approximately the posterior belief $bel(s_t)$, but it does not distribute with this posterior yet.

3. To make the weighted particle set $\bar{S}_t$ distribute according to the posterior belief $bel(s_t)$, this algorithm involves resampling (or called importance sampling) [Thrun05]. It is implemented in lines 9 to 12. Resampling re-draws N

particles according to the posterior belief $bel(s_t)$ to replace the temporary particle set $\bar{S}_t$ (see Appendix B). It transforms the temporary particle set $\bar{S}_t$ into a new particle set of the same size. Before resampling, the particle set is distributed according to $\overline{bel}(s_t)$. After resampling, the particle set is distributed according to $bel(s_t)$.

Algorithm 3.5: Basic MCL algorithm, adapted from [Thrun05]

1: **Input:** S_{t-1}, u_t, z_t, m
2: $\bar{S}_t = S_t = \emptyset$
3: **for** $n = 1$ to N **do**
4: generate a particle $s_t^{[n]} \sim p\left(s_t \left| s_{t-1}^{[n]}, u_t, m\right.\right)$ *% prediction*
5: calculate an importance factor $\omega_t^{[n]} = p\left(z_t \left| s_t^{[n]}, m\right.\right)$ *% correction*
6: add $\left\langle s_t^{[n]}, \omega_t^{[n]} \right\rangle$ to $\bar{S}_t$
7: **end for**
8: normalize ω_t
9: **for** $n = 1$ to N **do**
10: draw $s_t^{[n]}$ with importance factors $\omega_t^{[n]}$ *% resampling*
11: add $s_t^{[n]}$ to S_t
12: **end for**
13: **Output:** S_t

MCL is a nonparametric filter, too. It inherits the nonparametric nature that can represent complex multi-modal probability distributions. Thus it is applicable to both the position tracking problem and the global localization problem. MCL is implemented more efficiently than grid localization, because it is a random sampling algorithm that only calculates the posterior beliefs of samples. In practice, to obtain the same accurate result, MCL needs less samples than grid cells of grid localization in the same environment. However, the accuracy is increased with the size of the sample set. It also trades off the accuracy of localization and the computational efficiency. To employ the pre-caching technique can allay this conflict.

MCL performs poorly when the noise level is too small. Since errors of sensor models are described by Gaussian distributions, accurate sensors bring on a strict "resampling rule". In other words, accurate sensors generate sharp Gaussian distributions, thus only little particles that are very close to the robot's pose can survive after resampling. To deal with this problem, a simple method is to add artificial noise to the sensor readings. The more sensible solution is that for a small fraction of all particles, the role of the motion model and the measurement model are reversed. Such an approach is named the mixture MCL algorithm [Thrun00d, Thrun00b, Thrun05].

The basic MCL algorithm cannot solve the kidnapped robot problem, because its particles only survive near the most likely pose when the robot's pose is determined. If the robot is kidnapped to other place, there will not be particles near the new pose, so it is unable to recover from this kidnapping. A simple solution to this problem is to add random particles to the particle set such as Augmented MCL in [Thrun05], but this method suffers from significant computational overhead. A better way to approach the degeneracy phenomenon is to modify the proposal distribution of MCL, such as the mixture MCL algorithm [Thrun00d, Thrun00b, Thrun05]. An alternative approach named Bacterial Colony Growth Framework (BCGF) is proposed by Gasparri *et al.* [Gasparri08a, Gasparri08b]. It is a new biology-inspired approach, which is composed of two different levels of execution: a background level and a foreground level. The first is based on models of species reproduction to maintain the multi-hypothesis, while the second selects the best hypotheses according to an exchangeable specialized strategy.

3.5 HYBRID APPROACHES

In many practical situations, the single approach will not work sufficiently well, since any approach has complementary strengths and weaknesses. Thus, some hybrid strategies are proposed.

Baltzakis *et al.* [Baltzakis02,Baltzakis03] propose a probabilistic framework for modeling the robot's state and sensory information, based on Switching State-Space Models. The proposed approach combines the advantages of both the Kalman filter Linear models and the Hidden Markov Models (HMM), relaxing at the same time inherent assumptions made individually in each of these existing models. This framework uses HMM models to handle the qualitative aspects of the problem, i.e., perform coarse localization, and Kalman filters to handle the metric aspects, that is, elaborate on the previous result and provide accurate localization.

Another similar idea is proposed by Gasparri *et al.* [Gasparri07]. The implementation of their algorithm relies on two steps. First a particle filter is used to find out the most likely hypotheses with the assumption of stillness of the robot. Thereafter safe trajectories are planned and executed to reduce the remaining ambiguities using an extended Kalman filter for each hypothesis when the robot is moving.

Prestes *et al.* [Prestes08] present a strategy that combines path planning based on boundary value problems (BVP) and MCL to solve the global localization problem in sparse environments. This approach distributes particles only in relevant parts of the environment using the information about the environment structure. Afterwards, it leads the robot along these regions using the numeric solution of a BVP involving Laplace Equation.

3.6 SUMMARY

In this chapter, we introduced Markov localization and three concrete localization algorithms.

- We started with a review of the basic theory and derivation of the Bayes filter and we obtained the final localization formula.
- Table 3.1 summarizes and compares three Markov localization approaches: EKF, grid localization and MCL.

Table 3.1: Comparison of three Markov localization approaches.

	EKF	Grid localization	MCL
Posterior representation	Gaussian (μ_t, Σ_t)	histogram	particles
Position Tracking	yes	yes	yes
Global Localization	no	yes	yes
Kidnapping	no	yes	no
Efficiency	fast	slow	medium

- The localization problem can be divided into three sub-problems: position tracking, global localization and the kidnapped robot problem. EKF localization is the most common technique for dealing with the position tracking problem. It applies the extended Kalman filter to the localization problem, which represents the posterior belief $bel(s_t)$ by a Gaussian with mean μ_t and covariance Σ_t. Its advantages are simplicity and computational efficiency. But, inapplicability to global localization is its main disadvantage.
- Grid localization represents the posterior belief $bel(s_t)$ by using a histogram filter. It is capable of dealing with the position tracking problem and the global localization problem. The main shortcoming of grid localization lies in its computational complexity. The resolution of the grid trades off accuracy and efficiency. That makes grid localization be difficult to implement in real time.
- MCL represents the posterior belief $bel(s_t)$ by a set of weighted particles. It also can solve the position tracking problem and the global localization problem. It is more efficient than grid localization, since it only calculates the posterior beliefs of samples. Usually, MCL requires less particles than grid cells of grid localization in the map of the same size. Nevertheless, it also has to face the computational efficiency problem. The basic MCL do not have the ability to solve the kidnapped robot problem. Both grid localization and MCL are non-parametric.

Part II

Contributions of Thesis

CHAPTER 4

SIMULATION

Contents

4.1 INTRODUCTION

In the previous chapters, we have established motion models, perception models and map models for solving the localization problem. Moreover, we have introduced several classic localization approaches. In this chapter, we present some concrete implementations that link individual models and algorithms by means of simulation. The simulation is employed since it allows us to systematically vary key parameters such as the noise level, thereby enabling us to test the algorithms in extreme situations. We not only show localization results of these algorithms, but also make the comparison between these algorithms. Exhaustive simulation results make us understand intuitively advantages and disadvantages of these algorithms. This chapter starts with the description of our simulator, and then simulation results and comparison results are shown.

4.2 DESCRIPTION OF SIMULATOR

The simulator is programmed in MATLAB. MATLAB is a high-level technical computing language and interactive environment for algorithm development, data

visualization, data analysis, and numeric computation. It can solve technical computing problems faster than with traditional programming languages, such as C, C++, and Fortran [Mat]. Our simulator is just named as "a simulator", but actually it controls both the simulated robot and the real robot. It drives the real robot through the C language interface. The GUI (Graphical User Interface) of the simulator is shown in Figure 4.1. It is composed of some function panels, which control four main functions: localization, obstacles avoidance, path planning and path following. We will introduce briefly the common panels and the panels relative to the localization problem.

- The **Control Button** panel comprises four buttons: **INIT**, **RUN**, **STOP** and **PAUSE**. **INIT** is used to reset the simulator, such as the initialization of states, parameters and drawing. **RUN** and **STOP** are used to start and terminate the simulator, respectively. **PAUSE** can suspend the execution of simulator.

- The **VISU** panel is designed to adjust the axis of the figure object in order that the simulation process can display completely in the window. Buttons **N**, **E**, **S** and **O** control the axis to move in the direction of the north, the east, the south and the west, respectively. Button **H** makes the axis return to the initial position. In order to display the robot's position at any time, button **G** can take the axis directly to the robot's position.

- In the panel **Map**, we can either use the button **OBSTACLES** to construct a map or choose a pre-built map in the listbox. We can also add obstacles to a pre-built map by using the button **OBSTACLES**.

- From the listbox of the panel **Method**, we can choose which method we want to use. The simulator integrates many approaches, such as EKF localization, grid localization, MCL localization and SAMCL (self-adaptive Monte Carlo localization introduced in Chapter 5).

- In the panel **Motion Control**, we can choose which motion control model will be used. For example, the differentially driven WMR is controlled by the spinning speed of two wheels and the car-like WMR needs a translational velocity and a rotational velocity as input. The **JoyStick** model can control the robot by using the slider at the bottom of the panel. The **But** model controls the robot by a target following algorithm. The **Path Following** model uses a path following algorithm [Lapierre07a, Lapierre07b] to control the robot.

- In the panel **Other**, the button **Pre-caching** is used to pre-cache a map by employing the pre-caching technique (see Section 5.2.1). The button **Record** is used to record the process of the simulation into a small video and the button **Save Trajectory** is used to save the trajectory passed by the robot.

- The **State** panel shows the state in four dimensions: the angular velocity of wheels, the rotational velocity and the translational velocity for the car-like WMR model and the simulation time.

- The **DVZ & Path** panel includes control components of obstacles avoidance and path planning.

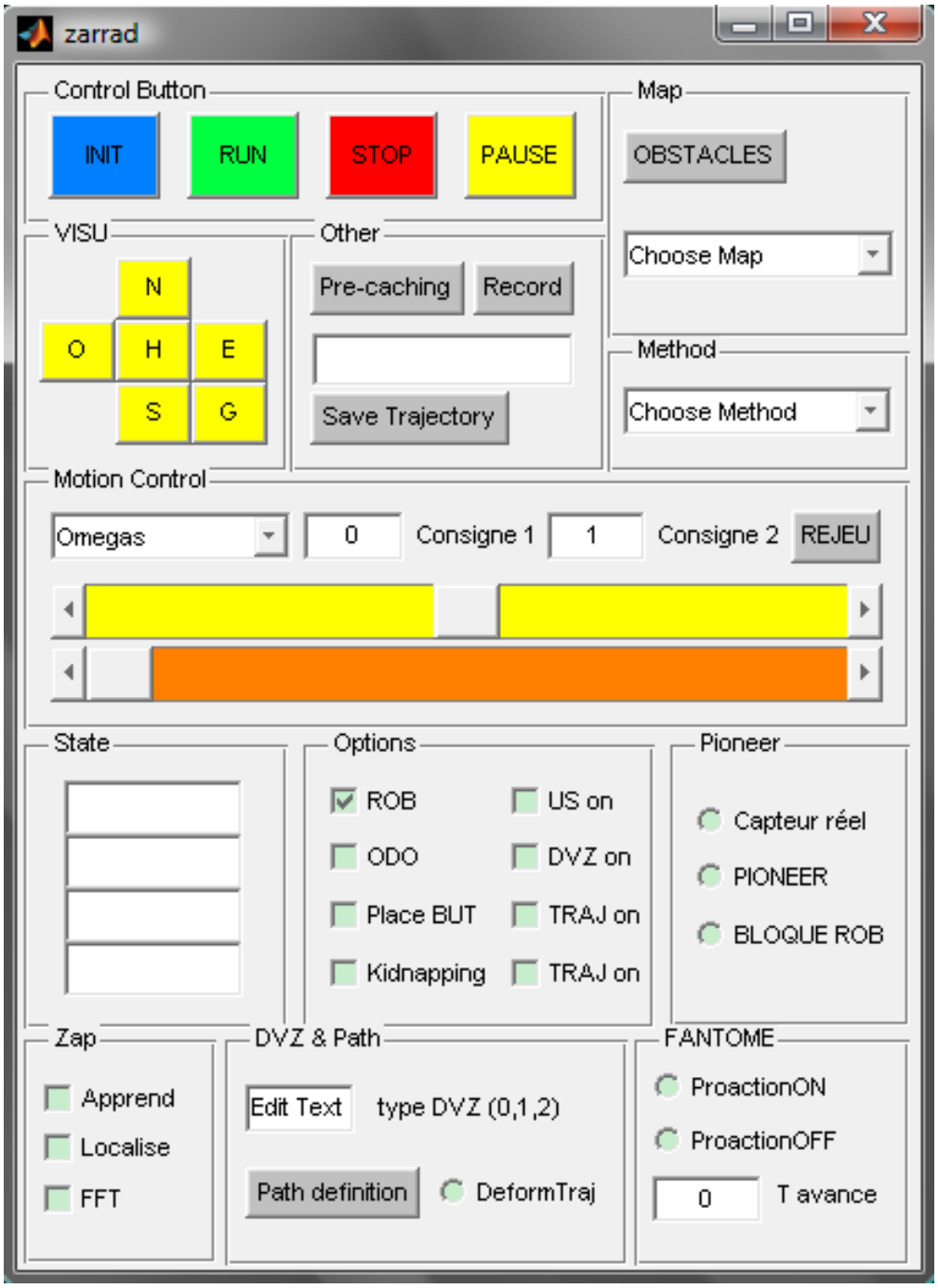
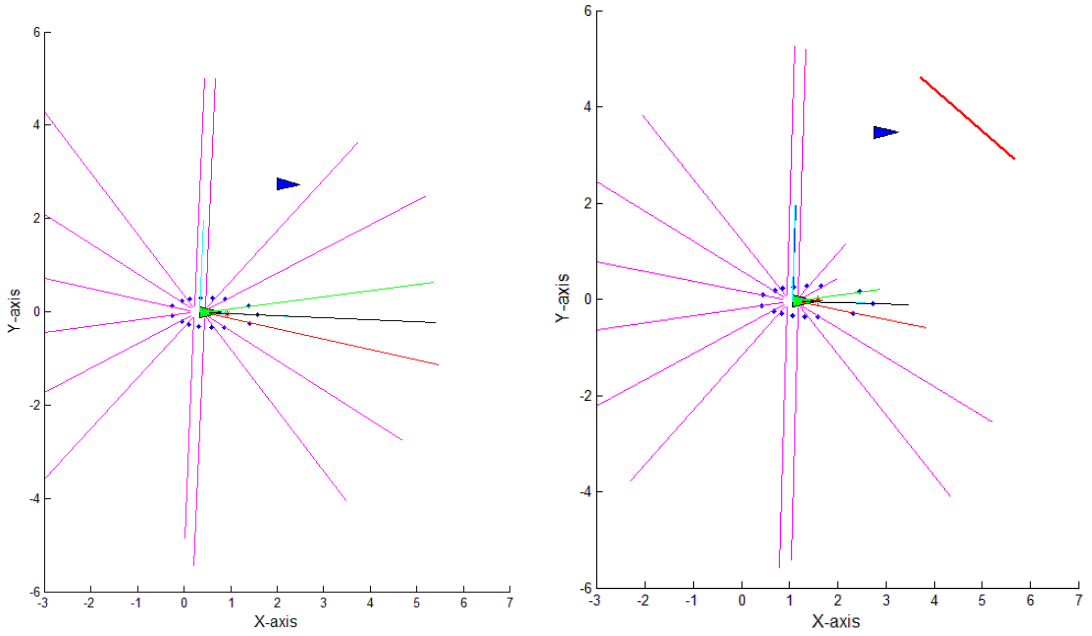


Figure 4.1: The GUI of the simulator programmed in MATLAB.

Figure 4.2 shows the operation window of the simulator. The simulator simulates a Pioneer 3-DX mobile robot equipped with 16 ultrasonic sensors. The blue (or deep gray in the grayscale image) triangle denotes the real robot and the

green (or light gray) triangle denotes the odometry robot. The red (or deep gray) segment denotes an obstacle (see Figure 4.2(b)). Segments distributed around the odometry robot denote range sensors, which surround the odometry robot but reflect measurements of the real robot (as shown in Figure 4.2(b)). In reality, we cannot know the pose of the real robot thus only odometry surrounded with sensors is displayed in the operation window. The ellipse consisted of blue points is Deformable Virtual Zone (DVZ) used to control the collision avoidance process [Cacitti01, Zapata04, Zapata05]. For the same reason, DVZ is distributed around odometry but reflects the detection of the real robot.



(a) (b)

Figure 4.2: The operation window of the simulator.

The simulated error is drawn from a normal distribution with mean zero and standard deviation σ ($\mathcal{N}(0, \sigma^2)$). In order to reflect the noise level (or error level), we define a scalar value Λ, which represent the noise by a percentage form.

$$\text{noise level } (\Lambda) = \frac{\text{standard deviation } (\sigma)}{\text{maximum range}} \times 100\% \tag{4.1}$$

4.3 SIMULATION SETUP

The implementation of simulations employs the following models and algorithms.

- **Motion model** uses the velocity motion model introduced in Section 2.2.2.1.
- **Perception model** employs the mixture perception model of range finders presented in Section 2.3.4.1. If the parameters $\alpha_{uk} = \alpha_f = \alpha_r = 0$ in Equation

2.28, this model is called the plain measurement model since only the Gaussian noise is considered.

- **Map** uses the line map introduced in Section 2.4.1.1, in which only walls are modeled.
- **EKF localization** employs Algorithm 3.2. This algorithm is originally designed for feature-based sensor model; however we simulate the imprecise ultrasonic range finder. Hence, we have to adapt this algorithm for range finders. Instead of the measurement of features $z_t^i = (r_t^i, \phi_t^i, \tau_t^i)^T$, the measurement of range finders only consist of the range r_t^i.
- **Grid localization** uses Algorithm 3.4.
- **MCL localization** employs Algorithm 3.5.

4.4 SIMULATION RESULTS

In this section, simulation results of EKF localization, grid localization and MCL localization are shown. Then, these approaches are compared in the aspects of efficiency and robustness. Simulation results are obtained by a low-end notebook PC.

4.4.1 Position tracking: EKF versus MCL

As discussed in the previous chapters, both the EKF algorithm and the MCL algorithm can solve the position tracking problem. This simulation focuses on testing and comparing the performances of the two algorithms when they deal with the position tracking problem. In this simulation, the robot will go around in an absolutely symmetrical closed corridor. The perception noise level and the motion noise level are about 4% and 3.53% for each wheel, respectively.

Figure 4.3 illustrates the position tracking results of a mobile robot using the EKF algorithm in an absolutely symmetrical closed corridor. As we already discussed, the EKF represents the posterior belief by a multivariate Gaussian distribution. This Gaussian is indicated by an uncertainty ellipse drawn in the reference frame of the robot. The center and semi-axes of the ellipse correspond to the mean and the covariance of the EKF, respectively. In the figure, the red ellipse is drawn in each interval of 20 program iterations. The size of the ellipse augments with the increased uncertainty. Three trajectories are shown: the robot's trajectory, the odometry's trajectory and the EKF's trajectory. They are denoted by the black solid line (line A), the green dash-dot line (line C) and the red dotted line (line B), respectively. Due to the motion errors, the odometry's trajectory (line C) is very different from the robot's trajectory (line A). But the EKF's trajectory (line B) almost overlaps the robot's trajectory (line A), that means a very satisfactory localization.

In the simulation, we can calculate easily the errors relative to the position of the robot. The localization error curves of the EKF and odometry are shown

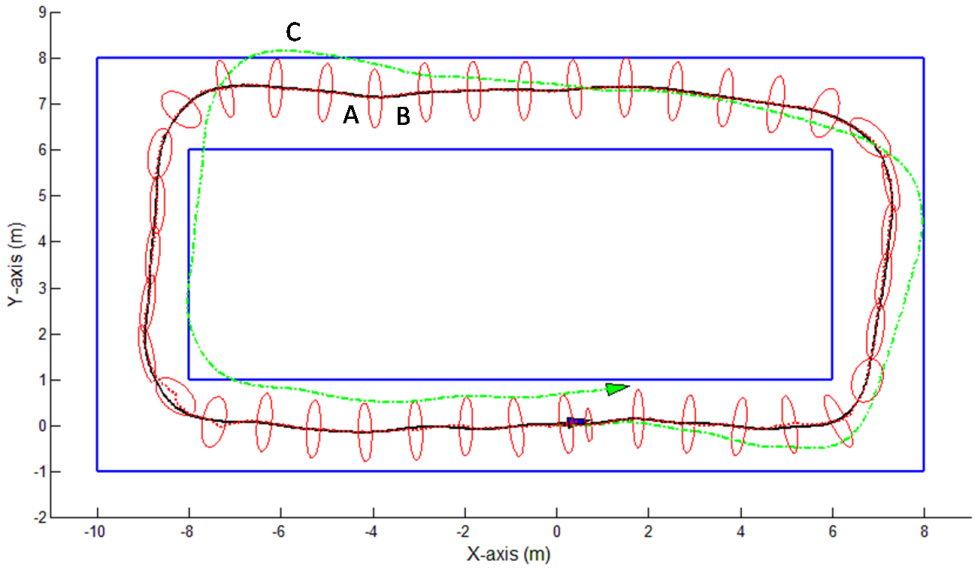


Figure 4.3: Position tracking using EKF in an absolutely symmetrical closed corridor. The trajectories of robot, odometry and EKF are displayed by the black solid line (line A), the green dash-dot line (line C) and the red dotted line (line B), respectively.

in Figure 4.4. Because the robot moves around in a closed corridor, odometry accumulates huge errors over time. However, the EKF algorithm estimates the robot's pose with constant and small errors compared with odometry.

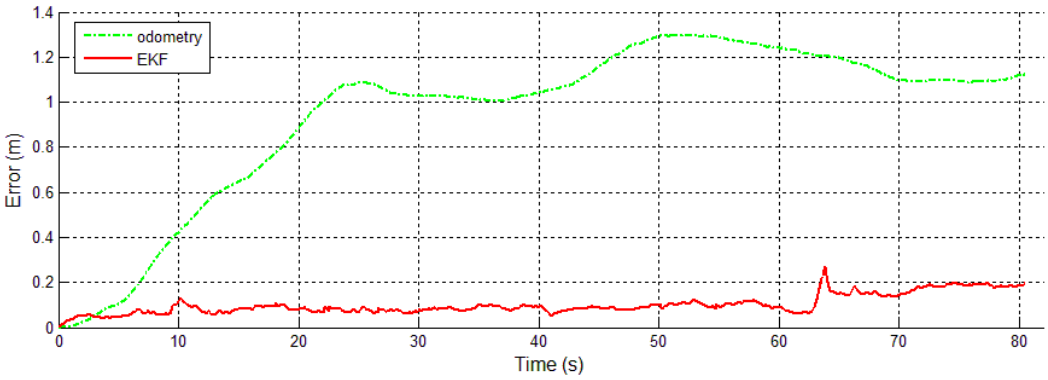


Figure 4.4: Localization errors of position tracking using EKF in an absolutely symmetrical closed corridor. EKF errors and odometry errors are plotted by the red solid line and the green dash-dot line, respectively.

The same simulation with the same parameters is executed by the MCL algorithm. We use 300 particles in the simulation. Figure 4.5 shows the trajectories of robot, odometry and particles. Since there are 300 particles' trajectories overlapping, so they look like a red belt.

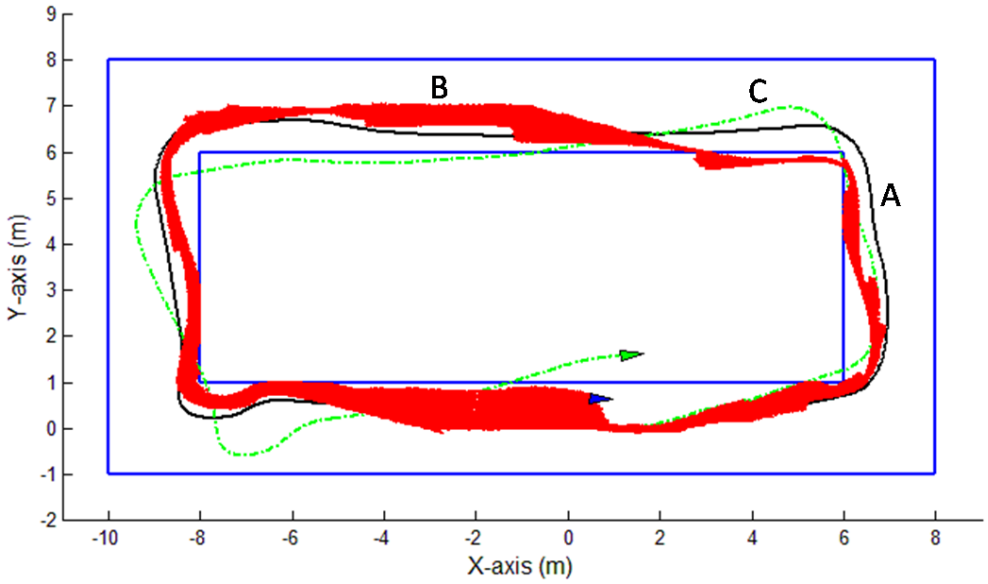


Figure 4.5: Position tracking using MCL in an absolutely symmetrical closed corridor. The trajectories of robot, odometry and particles are displayed by the black solid line (line A), the green dash-dot line (line C) and the red dotted line (line B), respectively.

In practice, we usually need to determine only one pose as the estimated pose of the robot at each iteration and we can control the robot based on this estimated pose. The pose of the particle owning the maximum probability or the arithmetic mean of poses of all particles may be simply chosen as this unique estimated pose. However, these choices partly or wholly ignore importance weights of particles. Here, we select a weighted mean of poses of all particles with respect to their importance weights. In particle filters, the importance weight of the particle is a probability. Therefore, we can understand this weighted mean as the expected value of particles. In equations, we have

$$\begin{aligned} \check{s}_t &= \sum_{n=1}^{N} s_t^{[n]} \omega_t^{[n]} \\ &= \mathrm{E}(s_t) \end{aligned} \tag{4.2}$$

Figure 4.6 shows the three trajectories. The MCL's trajectory is represented by the expected value of particles. The red dotted line (line B), the black solid line (line A) and the green dash-dot line (line C) depict the trajectories of MCL, the robot and odometry, respectively. As shown in the localization results, MCL has bigger errors than EKF in such an absolutely symmetrical closed corridor. The same result can be found in Figure 4.6. Compared with Figure 4.4, localization errors of MCL are lager than EKF. The MCL algorithm performs worse than the EKF algorithm for two main reasons.

1. EKF represents the posterior belief using a unimodal Gaussian, which has a better performance than MCL that represents the posterior belief by multi-hypothesis in the position tracking problem.

2. In an absolutely symmetrical corridor environment, there are not enough differences to update beliefs of particles. Most of the time, probabilities of particles tend to be equal, so particles converge slowly. That leads to bigger errors.

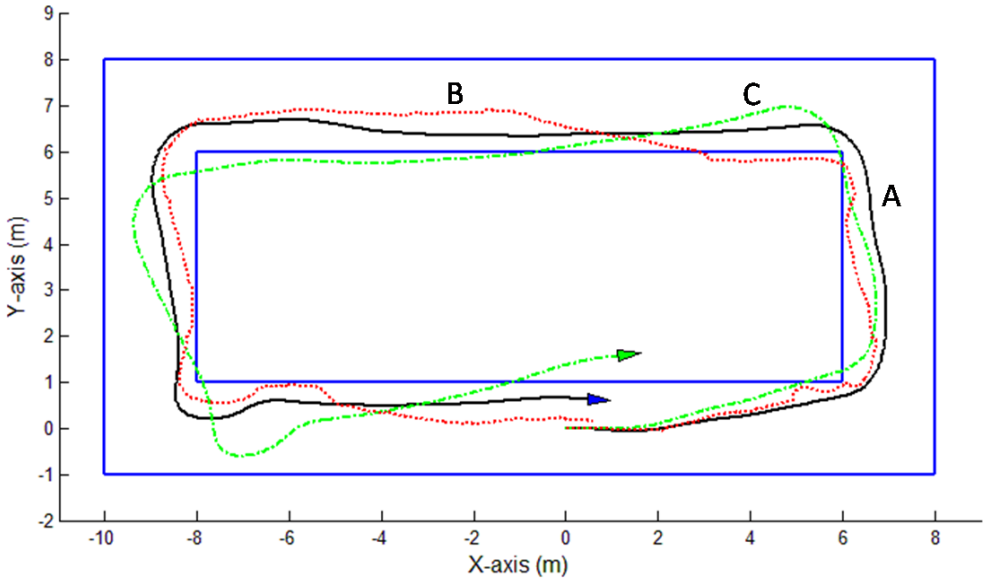


Figure 4.6: Position tracking using MCL in an absolutely symmetrical closed corridor. The localization result is represented by the expected value of particles.

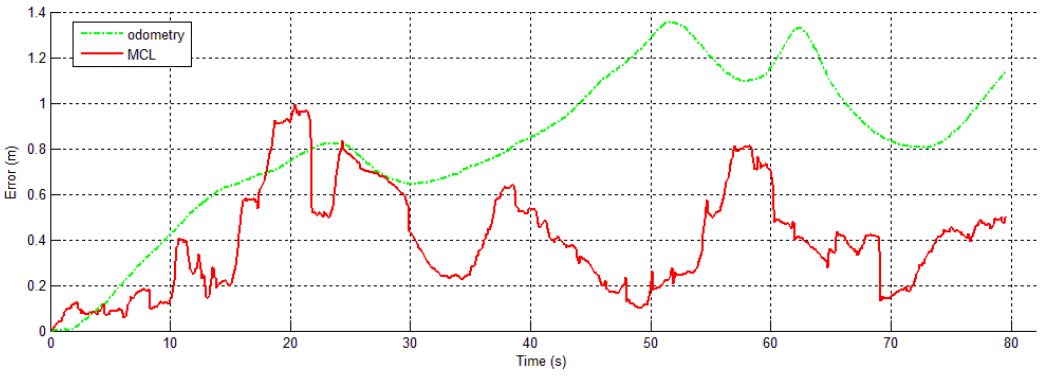


Figure 4.7: Localization errors of position tracking using MCL in an absolutely symmetrical closed corridor. MCL errors and odometry errors are plotted by the red solid line and the green dash-dot line, respectively.

Figure 4.8 shows average error curves of EKF (red dotted line) and MCL (black solid line) at different odometric noise level. This simulation tests and compares

the robustness of EKF and MCL in solving the position tracking problem. It is executed in the absolutely symmetrical corridor map. In this simulation, we use a fixed perception error. Considering the simulation simulating an ultrasonic perception system, the perception error level is fixed at 4%. We increase the motion error gradually, EKF is more accurate when the error is small, but it failed when the error mounted up to a certain level. The MCL algorithm always maintained at an acceptable error level. This result shows that the MCL algorithm has much greater fault tolerance than the EKF algorithm.

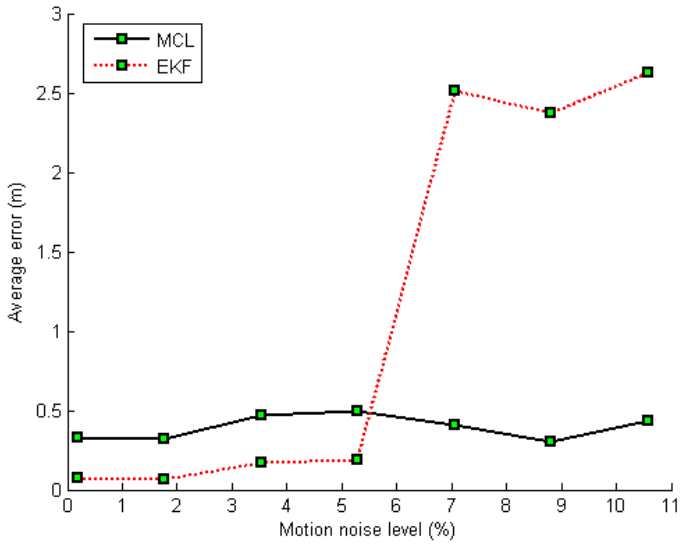


Figure 4.8: Average localization errors of EKF (red dotted line) and MCL (black solid line) as a function of motion noise.

Another simulation is executed in a quasi-symmetrical corridor map. Both the EKF algorithm and the MCL algorithm are tested for solving the position tracking problem in this map. Figure 4.9 depicts the localization trajectories obtained by using the EKF algorithm. The EKF's trajectory (line B) almost covers the robot's trajectory (line A). Figure 4.10 depicts the localization error curves, almost all of the localization errors are less than $0.5m$. Thus, the EKF algorithm is fully capable of estimating the robot's position in this map.

The MCL algorithm is tested with the same parameter settings in the quasi-symmetrical corridor. As shown in Figure 4.11 and Figure 4.12, the MCL algorithm performs as well as the EKF algorithm in this map. The non-symmetric parts of the map provide differences to update beliefs of particles for the MCL algorithm.

According to previous simulation results, we will make a brief summary of the performances of EKF and MCL in the position tracking problem.

- In the absolutely symmetrical corridor environment, EKF performs better than MCL.
- In the quasi-symmetrical corridor environment, MCL performs as well as EKF.

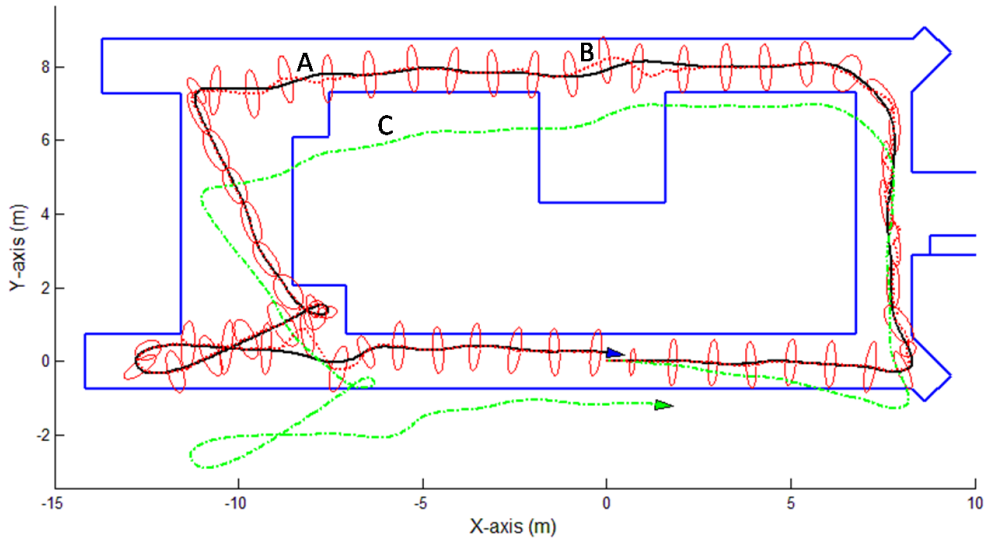


Figure 4.9: Position tracking using EKF in a quasi-symmetrical corridor. The trajectories of robot, odometry and EKF are displayed by the black solid line (line A), the green dash-dot line (line C) and the red dotted line (line B), respectively.

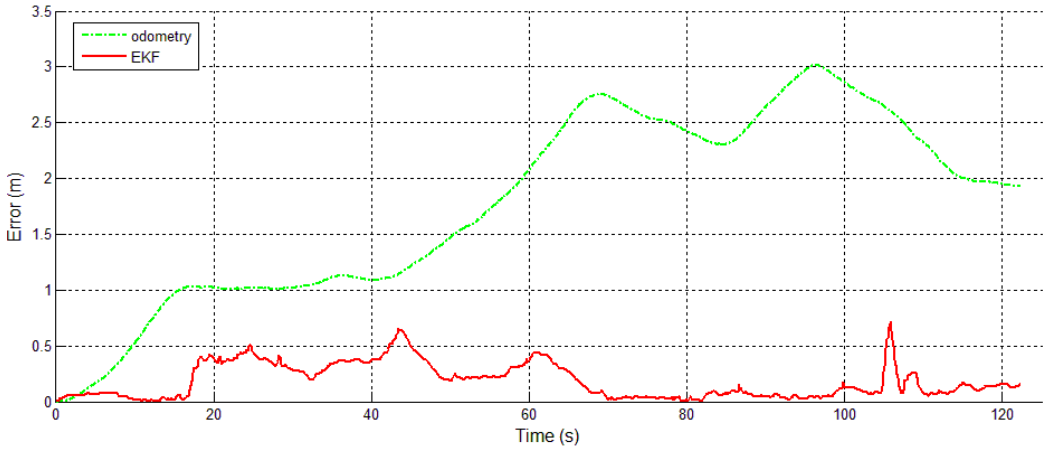


Figure 4.10: Localization errors of position tracking using EKF in the quasi-symmetrical corridor. EKF errors and odometry errors are plotted by the red solid line and the green dash-dot line, respectively.

- Generally, MCL has much greater fault tolerance than EKF.

4.4.2 Global localization using MCL

This simulation aims at testing the ability of global localization and the robustness of the MCL algorithm. The quasi-symmetrical corridor map (the same as using in position tracking) is used. In order to test the robustness of MCL, we add 6% perception noise and 8.82% motion noise to each wheel (larger than the test of

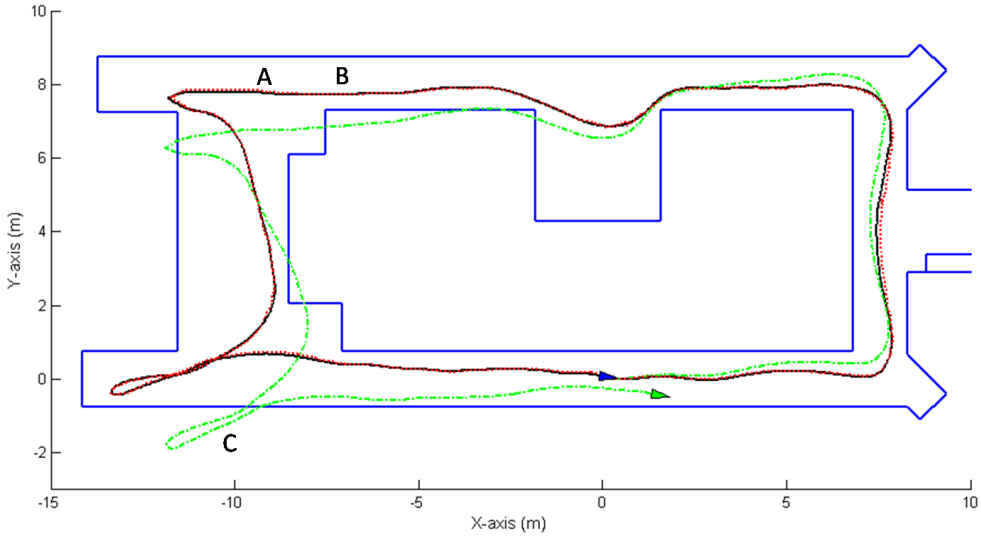


Figure 4.11: Position tracking using MCL in a quasi-symmetrical corridor. The trajectories of robot, odometry and MCL are displayed by the black solid line (line A), the green dash-dot line (line C) and the red dotted line (line B), respectively.

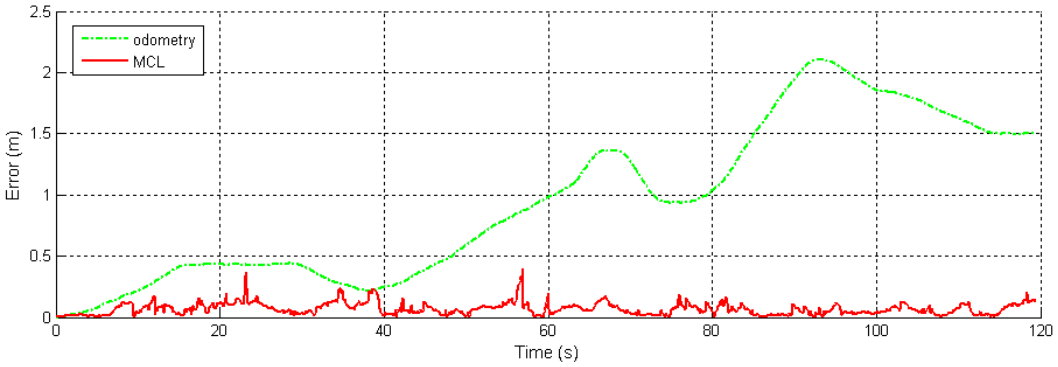


Figure 4.12: Localization errors of position tracking using MCL in the quasi-symmetrical corridor. MCL errors and odometry errors are plotted by the red solid line and the green dash-dot line, respectively.

position tracking). In the global localization problem, the robot does not know its initial pose. Thus, there are 300 particles distributed randomly in the whole map with uniform probabilities (as shown in Figure 4.13).

Figure 4.14 shows the localization trajectories. The red dotted line (line B) depicts the trajectory of MCL. This trajectory has bigger errors at the beginning, since particles are distributed randomly in the initialization phase. After particles converging, this trajectory covers well the robot's track (line A). However, the odometry's track (line C) is far away from the robot due to the motion error.

The same result can be learned in Figure 4.15. Since particles are initialized by

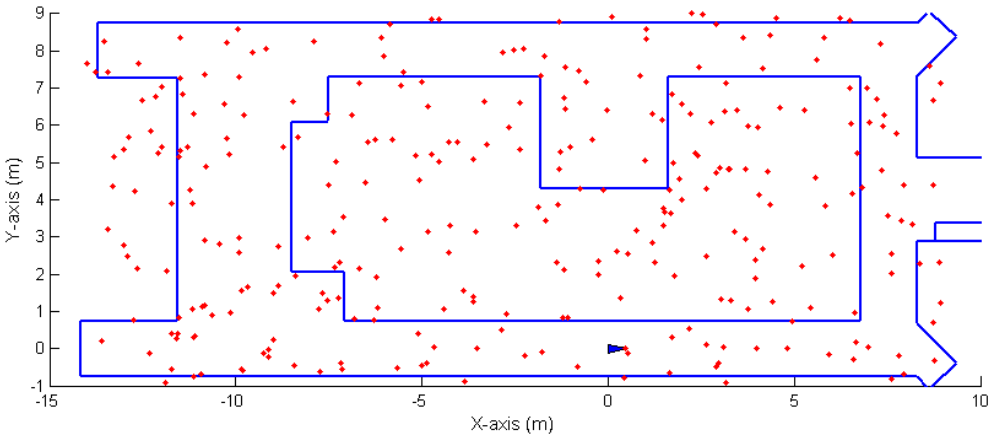


Figure 4.13: Initialization of MCL global localization in a quasi-symmetrical corridor.

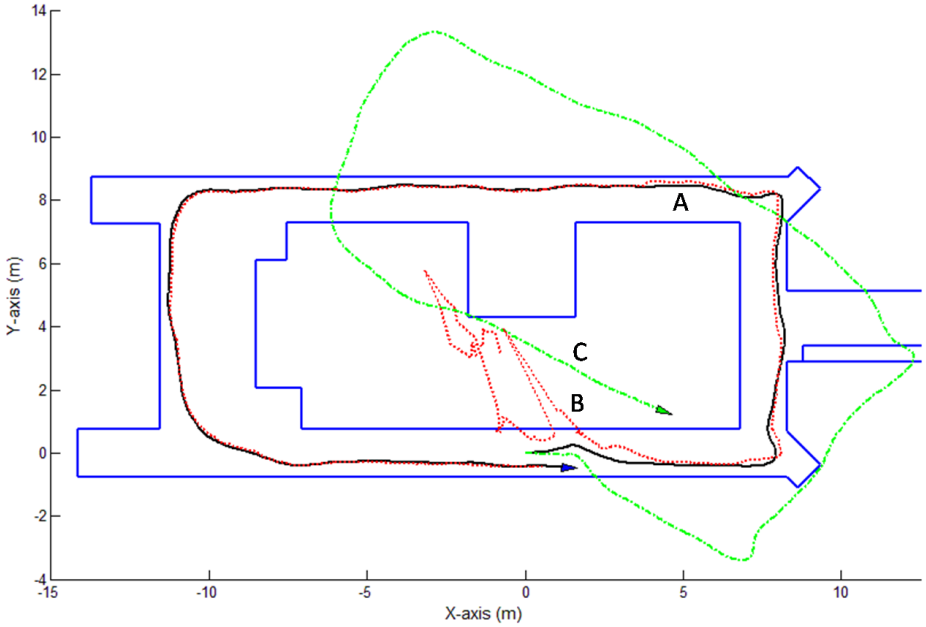


Figure 4.14: Global localization using MCL in a quasi-symmetrical corridor. The trajectories of robot, odometry and MCL are displayed by the black solid line (line A), the green dash-dot line (line C) and the red dotted line (line B), respectively.

a random distribution, the localization errors of MCL are large at the beginning. But the errors decrease quickly with particles converging. In contrast, odometry has small errors at the beginning, but its errors accumulate with time.

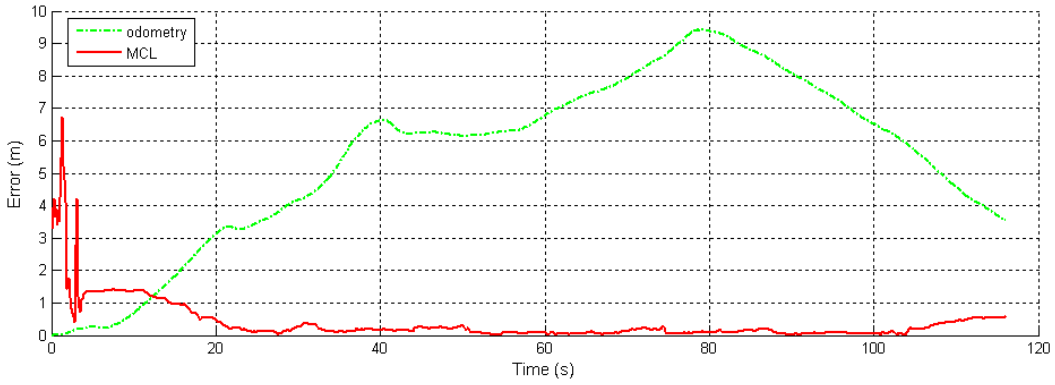


Figure 4.15: Localization errors of global localization using MCL in a quasi-symmetrical corridor. MCL errors and odometry errors are plotted by the red solid line and the green dash-dot line, respectively.

4.4.3 Grid localization

As discussed in Section 3.3, the primary disadvantage of grid localization lies in its computational complexity. For example, the quasi-symmetrical corridor map used in MCL has a dimension of $25m \times 10m$. Using the MCL algorithm, the localization can be achieved by using only 300 particles. However, even if grid localization employs a coarse resolution of $0.5m \times 0.5m$ (the localization errors of MCL are less than $0.5m$) to decompose this map, it will produce 1000 grid cells that are much more than the number of particles. Moreover, if a three-dimensional grid (included x, y and θ) is used, the motion update requires a convolution, which is a 6-D operation [Thrun05]. The measurement update of a full scan is a costly operation, too. The computational burden is so huge that this algorithm almost cannot be executed in real-time.

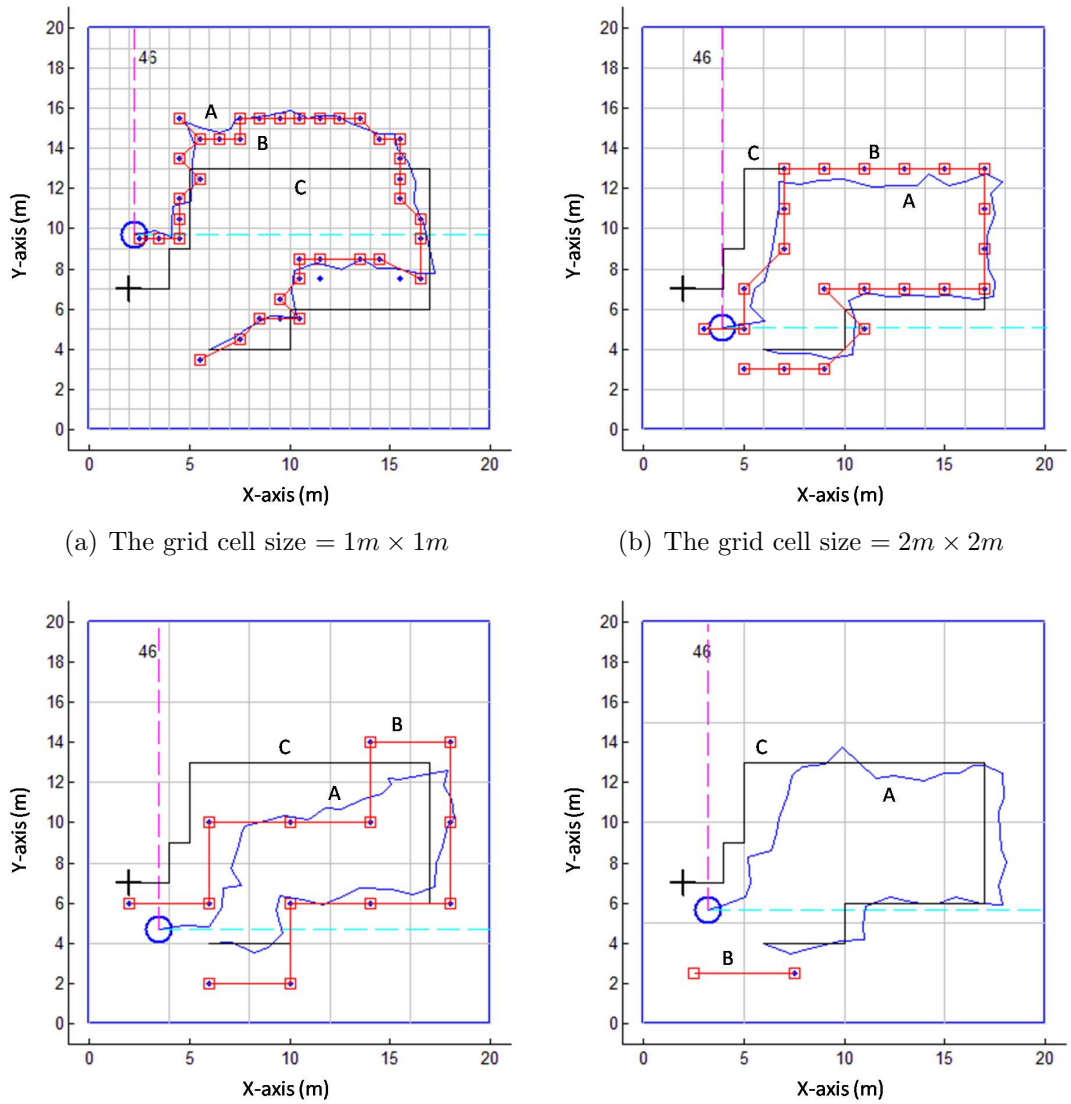
For the above reason, we design a simpler simulator. The simulated robot's pose only comprises $x - y$ coordinates without considering its orientation θ, thus a two dimensional grid is used corresponding to the robot's pose. The robot is equipped with two mutually perpendicular range finders, which are assumed to be capable of measuring anywhere in the map. The simulated environment is a square room of $20m \times 20m$.

Figure 4.16 shows the localization results of 2-D grid localization by using different resolution grids. The blue circle denotes the simulated robot and its trajectory is drawn by blue line (line A). The black cross denotes odometry robot and the black line (line C) is its trajectory. The red line (line B) links these grid cells owning the largest probability.

The grid cell with the biggest probability is annotated by the red square. The grid cell with bigger probability (higher than 0.1) is only annotated by the blue dot. As shown, some grid cells with bigger probability are passed by the robot's trajectory. This proves it is more reasonable that the probabilistic approach represents uncertainties using a probability distribution instead of relying on a single

"best position".

When a finer resolution is used, the red line (line B) is closer to the blue line (line A), namely, the localization is more accurate. In the simulation, resolutions shown in Figure 4.16(a), (b), (c) can be used to estimate the robot's pose. But when the grid cell size augments to $5m \times 5m$, the estimation of the robot's pose cannot be achieved yet (as shown in Figure 4.16(d)).



(a) The grid cell size = $1m \times 1m$

(b) The grid cell size = $2m \times 2m$

(c) The grid cell size = $4m \times 4m$

(d) The grid cell size = $5m \times 5m$

Figure 4.16: Localization results of 2-D grid localization with different resolutions.

Figure 4.17 plots the average localization error curve as a function of grid cell size. As to be expected, the localization error increases as the resolution decreases. When the grid resolution is equal to $5m$, the localization cannot be achieved.

Figure 4.18 plots the average localization time curve as a function of grid resolution. The total time necessary to localize a robot decreases as the grid

becomes coarser.

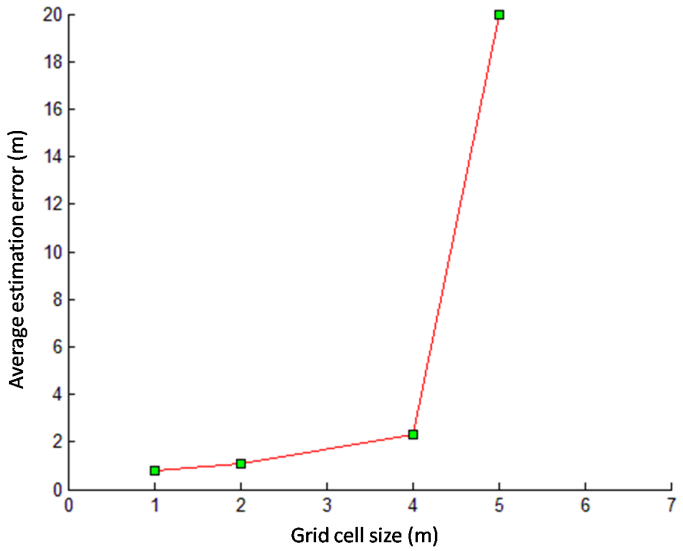


Figure 4.17: Average localization error as a function of grid cell size.

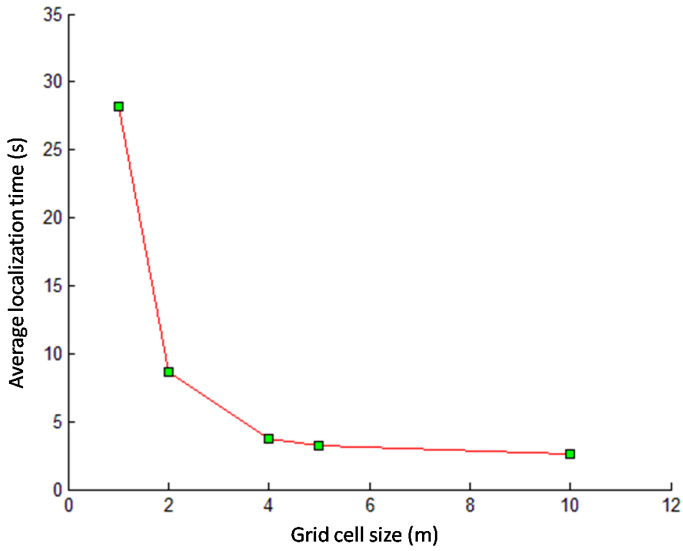


Figure 4.18: Average localization time needed for global localization as a function of grid resolution.

These simulation results demonstrate the fact discussed in Section 3.3, using the finer grid can get a more accurate result, but at the expense of increased computational costs.

4.4.4 Mixture perception model in the dynamic environment

In section 2.3.4.1, we have discussed a mixture perception model merging four types of measurement errors. This mixture perception model is tested and compared with the plain measurement model (only the Gaussian noise is considered) in a dynamic environment. The simulated robot (the blue or deep gray triangle) is equipped with five ultrasonic sensors around its head. The sensor range is assumed to be large enough to detect anywhere of the map. The simulated environment is a square room of $20m \times 20m$.

The mixture model is implemented with the MCL algorithm. At the initial step, 150 particles are distributed randomly in the whole map (global localization). Figure 4.19(a) depicts MCL with the plain measurement model and Figure 4.19(b) depicts MCL with the mixture perception model. Both two can achieve the localization at the 50^{th} step, since there are no unknown obstacles. The first two unknown obstacles (upper and left) are added in the robot environment at the 100^{th} step and the other two unknown obstacles (lower and right) are added at the 200^{th} step. MCL with the plain measurement model fails to localize the robot when unknown obstacles are added (see Figure 4.19(a)). However, MCL with the mixture perception model performs well during the whole positioning process (see Figure 4.19(b)).

4.5 SUMMARY

In this chapter, we tested EKF localization, grid localization and MCL by simulations. We also analyzed and compared these simulation results.

- We started with the description of the simulator. The GUI and the operation window are depicted.
- Table 4.1 summarizes and compares simulation results of EKF and MCL in position tracking.

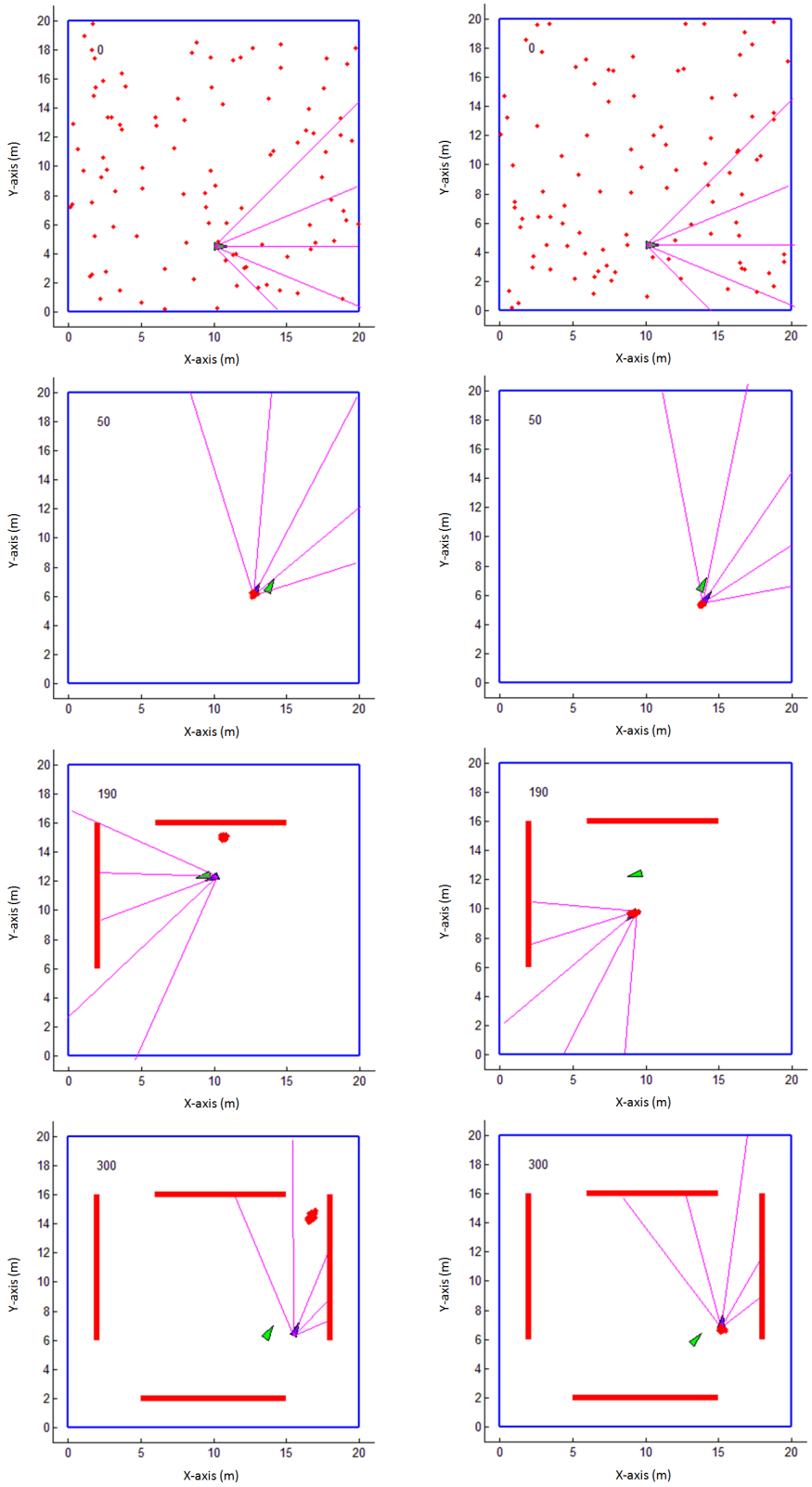
Table 4.1: Intuitive comparison of EKF and MCL by simulation.

	EKF	MCL
Position tracking in the absolutely symmetrical environment	good	medium
Position tracking in the quasi-symmetrical environment	good	good
Robustness with respect to errors	low	high

- The first simulation tested EKF and MCL for position tracking in an absolutely symmetrical closed corridor. Localization results show that EKF has a better performance than MCL. EKF represents the posterior using the unimodal Gaussian, which is more suitable to track the robot in such an environment. However, MCL maintains the multi-modal distribution and beliefs of

particles tend to be equal most of the time, thus particles converge slowly in the symmetrical environment.

- Another trial of position tracking was implemented in a quasi-symmetrical corridor. This time, both EKF and MCL can work well in this environment. MCL can update beliefs of particles when it feels differences of the environment.
- We compared EKF with MCL in the robustness. MCL can tolerate more errors than EKF.
- Global localization was tested with MCL in the quasi-symmetrical corridor. MCL performed well in global localization.
- Grid localization was tested by a simpler simulator due to considering its computational costs. It is evaluated by using different resolution grids. Simulation results demonstrate that the finer grid can get a more accurate result, but at the expense of increased computational costs. If we want to use the grid approach, we must find out the balance between the accuracy and the efficiency.
- We employed MCL to compare the plain measurement model with the mixture perception model in the dynamic environment.



(a) (b)

Figure 4.19: Comparison of (a) MCL with the plain measurement model and (b) MCL with the mixture perception model in the dynamic environment.

CHAPTER 5

THE SAMCL ALGORITHM

Contents

5.1 INTRODUCTION

In this chapter, we introduce the Self-Adaptive Monte Carlo Localization algorithm, abbreviate as SAMCL. As discussed thus far, the localization problem can be divided into three sub-problems: position tracking, global localization and the kidnapped robot problem. In the position tracking problem, the initial robot pose is known. But for global localization, the robot has no initial knowledge of its pose. The most difficult kidnapping problem can be described as a well-localized robot is teleported to some other place without being told. Due to the specific nature of these problems, it is difficult to find a general solution for all three.

The SAMCL algorithm, an improved Monte Carlo localization algorithm using self-adaptive samples, is devised to solve all the three sub-problems. As discussed in Section 3.4, MCL is applicable to position tracking and global localization, however it has to face the issue of computational efficiency if a large number of particles are used. Moreover, MCL is unable to recover from kidnapping (global

localization failures), since particles only survive near a single pose once the position of the robot is determined. To overcome the problems of the MCL algorithm, the SAMCL algorithm makes three contributions.

Firstly, it employs a pre-caching technique to reduce the on-line computational burden of MCL. Thrun, Burgard, and Fox [Thrun05] use this technique to reduce costs of computing for beam-based models in the ray casting operation. Our pre-caching technique decomposes the state space into two types of grids. The first one is a three-dimensional grid denoted as G_{3D} that includes the planar coordinates and the orientation of the robot. It is used to reduce the on-line computational burden of MCL.

The other grid is a two dimensional energy grid denoted as G_E. We define energy as the special information extracted from measurements. The energy grid is used to calculate the Similar Energy Region (SER) which is a subset of G_E [Zhang09a]. Its elements are these grid cells whose energy is similar to robot's energy. SER provides potential information of robot's position, thus, sampling in SER is more efficient than sampling randomly in the whole map. That is the second contribution.

Finally, SAMCL can solve position tracking, global localization and the kidnapped robot problem together thanks to self-adaptive samples. Self-adaptive samples are different from the KLD-Sampling algorithm proposed in [Fox03a,Thrun05]. Their sample set has an adaptive size, which can increase the efficiency of particle filters. Our self-adaptive sample set has a fixed size, thus it does not lead to the expansion of the particle set. In order to solve the kidnapping problem, a number of global samples are necessary. "When to generate global samples" and "where to distribute global samples" are two main problems. The self-adaptive sample set can automatically divide itself into a global sample set and a local sample set according to different situations. Local samples are used to track the robot's pose, while global samples are distributed in SER and used to find the new position of the robot.

5.2 ALGORITHM DESCRIPTION

The SAMCL algorithm is implemented in three steps [Zhang09b, Zhang09e], as illustrated in Figure 5.1.

- **Pre-caching the map.** The first step accepts the map m as input. It outputs a three-dimensional grid G_{3D} and a two-dimensional energy grid G_E. The grid G_{3D} stores measurement data of the whole map and the grid G_E stores energy information. This step is executed off line to reduce the on-line computational burden.

- **Calculating SER.** The inputs of the second step are the energy grid G_E obtained off-line in the pre-caching phase and the measurement data z_t of the robot at time t. The output is SER. This step is run on line.

- **Localization.** The last step accepts as input the particle set S_{t-1}, control data

u_t, measurement data z_t, the three-dimensional grid G_{3D} and SER. It outputs the particle set S_t. This step is also run on line.

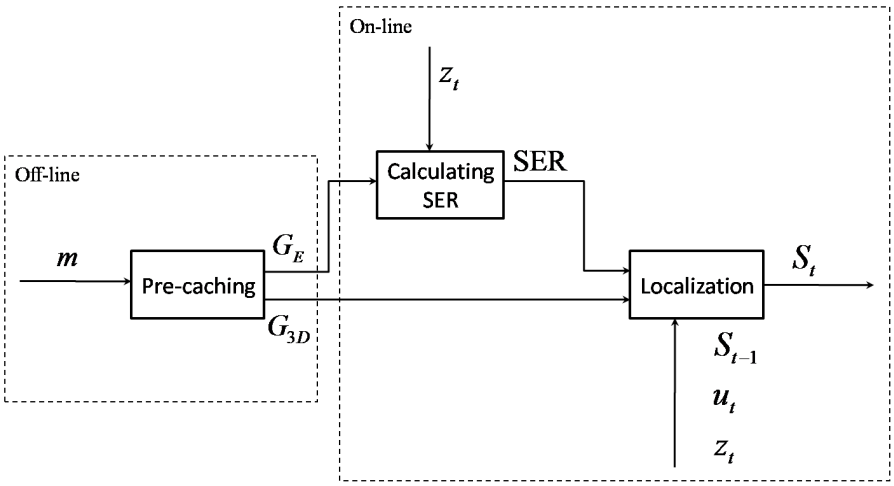


Figure 5.1: The process of the SAMCL algorithm.

5.2.1 Pre-caching the map

In the localization problem, the map is supposed to be pre-known by the robot and be static. Hence, a natural idea is to decompose the given map into grid and to pre-compute measurements for each grid cell. Our pre-caching technique decomposes the state space into two types of grids.

- **Three-dimensional grid (G_{3D}).** The map is decomposed into a three-dimensional grid that includes planar coordinates and the orientation. Each grid cell is seen as a pseudo-robot that perceives the environment at different poses and stores these measurements. When SAMCL is implemented, instead of computing measurements of the map for each particle on line, the particle is matched with the nearest grid cell and then simulated perceptions stored in this cell are assigned to the particle. Measurements are pre-cached off line, hence the pre-caching technique can reduce the on-line computational burden. Obviously, the precision of the map describing depends on the resolution of the grid.

- **Two-dimensional energy grid (G_E).** Each grid cell of the energy grid pre-computes and stores its energy. Energy is the special information extracted from measurements. For range sensors, the measurement data are distances, denoted as d for an individual measurement. We define i^{th} sensor's energy as $1 - d_i/d_{\max}$, d_i is the measurement of i^{th} sensor and $d_{\max}$ is the maximum distance that sensors are able to "see". Then we calculate the sum of energy of all the sensors. The advantage of using total energy of all the sensors is no need to consider the orientation of the robot, thus we can reduce one-dimensional

calculation. These grid cells nearby obstacles will have larger energy than those in the free space.

Please note that we can calculate the sum of energy to reduce one-dimensional calculation based on an assumption that the robot's sensors are distributed uniformly or quasi-uniformly around its circumference. The reason is simple. If a robot has non-uniformly distributed sensors, it will obtain different energy at the same location but different orientations. Figure 5.2 shows an example. A robot with non-uniformly distributed sensors measures in a long and narrow room at different orientations. Energy of case (a) can be computed as follows:

$$E_a = (1 - \frac{d_1}{d_{\max}}) + (1 - \frac{d_2}{d_{\max}}) + (1 - \frac{d_3}{d_{\max}}) \tag{5.1}$$

Energy of case (a) can be computed as follows:

$$E_b = (1 - \frac{e_1}{d_{\max}}) + (1 - \frac{e_2}{d_{\max}}) + (1 - \frac{e_3}{d_{\max}}) \tag{5.2}$$

Obviously, we have

$$E_a > E_b \tag{5.3}$$

Hence, we must take the orientation into account when these robots are used.

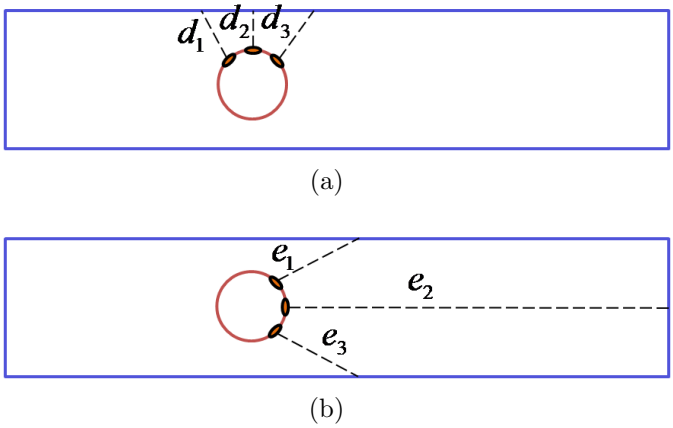


Figure 5.2: A robot with non-uniformly distributed sensors measuring in a long and narrow room at different orientations. Energy of case (a) and case (b) is different, even if the robot is at the same location.

The process of calculating energy for grid cells is shown in Algorithm 5.1. It inputs the map m and outputs the two-dimensional energy grid G_E. In line 4, each sensor of one grid cell measures the map using ray casting and gives the distance $d_i^{[k]}$. Line 5 computes energy $\tilde{a}_i^{[k]}$ of the i^{th} sensor of the k^{th} grid cell. Line 6 computes total energy $\tilde{E}(k)$ of the I sensors of the k^{th} grid cell. In line 7, we normalize total energy $\tilde{E}(k)$. Hence, energy $\tilde{a}_i^{[k]}$ and total energy $\tilde{E}(k)$ has the same value interval $[0, 1]$ as probability density. This energy grid is used to calculate SER and will be presented in Section 5.2.2.

Algorithm 5.1: Calculating energy for each grid cell

1: **Input:** m
2: **for** all the grid cell $k \in \{1, \cdots, K\}$ **do**
3: **for** all the range sensors $i \in \{1, \cdots, I\}$ **do**
4: measure and get distance $\tilde{d}_i^{[k]} < d_{\max}$
5: $\tilde{a}_i^{[k]} = 1 - \tilde{d}_i^{[k]}/d_{\max}$
6: $\tilde{E}(k) = \sum_{i=1}^{I} \tilde{a}_i^{[k]}$
7: **end for**
8: normalize $\tilde{E}(k) = \frac{1}{I}\tilde{E}(k)$
9: **end for**
10: **Output:** G_E

5.2.2 Calculating SER

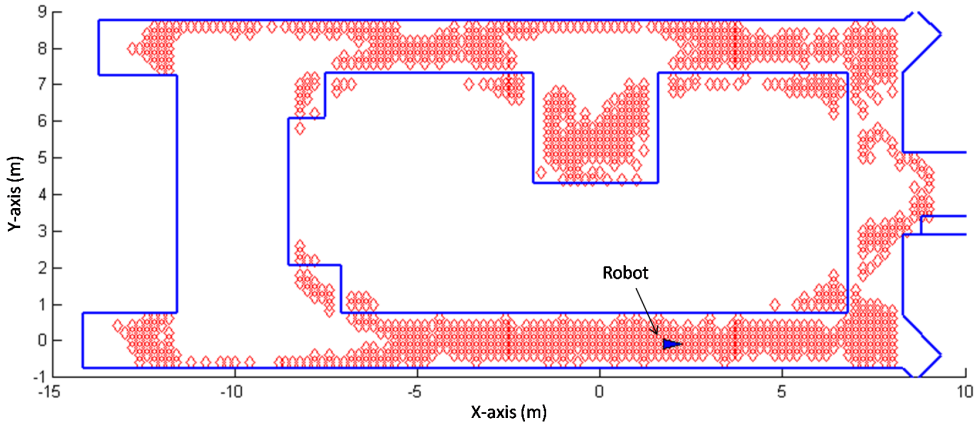
Similar energy region (SER) is defined as a subset of G_E. Grid cells in SER have similar energy with the robot. SER may be seen as the candidate region for sampling, in which particles have higher probability. Information provided by SER is used to match the position of the robot, such as the robot is in the corridor or in the corner, is nearby obstacles or in the free space. Figure 5.3 shows SER when the real robot is located in a corridor (a) and in a corner (b). To distribute global samples, SER provides an a priori choice. Sampling in SER solves the problem of where to distribute global samples. Obviously, sampling in SER is more efficient than sampling stochastically in the entire map. Especially, if the robot is in a distinct region such as Figure 5.3(b), the advantage of sampling in SER is more significant.

An algorithm to calculate SER is shown in Algorithm 5.2. It accepts as input the energy grid G_E obtained off-line in the pre-caching phase and the range measurements d_t of the robot at time t. It outputs SER. Lines 2 to 6 compute total energy of the I sensors for the real robot. Lines 7 to 9 compares total sensor energy of the real robot with total sensor energy of each grid cell. If the difference is smaller than a given threshold δ, we define this grid cell as a SER cell.

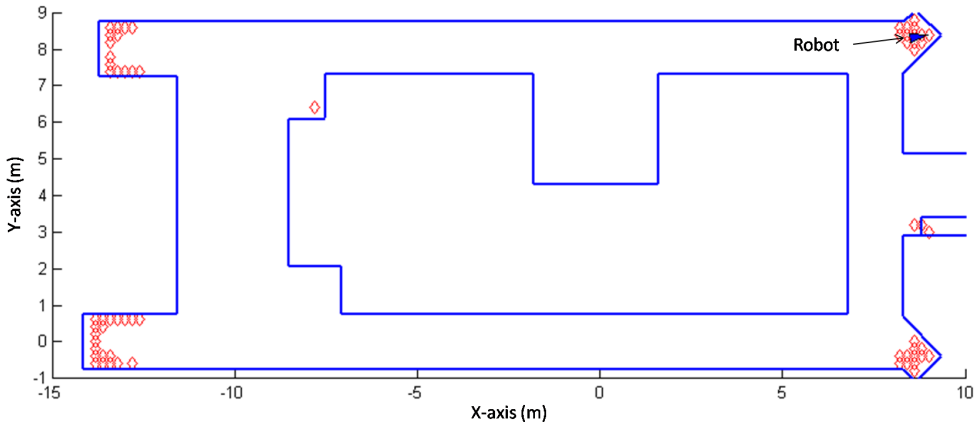
5.2.3 Localization

The SAMCL algorithm uses self-adaptive samples to solve the position tracking, global localization and the kidnapped robot problems together. Self-adaptive samples can automatically divide themselves into a local sample set and a global sample set and transform between them according to different situations. SAMCL maintains local samples by regular MCL and distributes global samples in SER.

When the robot is well localized, SAMCL only maintains local samples around the robot. Once the robot is kidnapped, part of samples migrate from local samples



(a)



(b)

Figure 5.3: SER when the robot is (a) in the corridor and (b) in the corner.

to global samples. After the robot re-localizes itself, global samples are converted as one part of local samples. Global samples are able to help the robot recover from kidnapping. But they may also induce a wrong reaction, for instance, in symmetrical environments, all the particles in symmetrical regions may have high probability and the pose of robot could be ambiguous. Hence, the idea is that global samples only appear when the robot is "really" kidnapped. The main question is to know when the robot is kidnapped. We value whether the robot is kidnapped by measuring the probabilities of particles. If the maximum of probabilities of particles is less than a given threshold, the robot will deduce that it has been kidnapped.

The SAMCL algorithm is summarized in Algorithm 5.3. It inputs the particle

Algorithm 5.2: Calculating SER algorithm

1: **Input:** G_E, d_t
2: **for** all the range sensors of the real robot $i \in \{1, \cdots, I\}$ **do**
3: $a_i = 1 - d_i/d_{\max}$
4: $E = \sum_{i=1}^{I} a_i$
5: **end for**
6: normalize $E = \frac{1}{I}E$
7: **for** all the grid cell $k \in \{1, \cdots, K\}$ **do**
8: defining the grid cell k as a SER cell, if $\left|E - \tilde{E}(k)\right| < \delta$
9: **end for**
10: **Output: SER**

set S_{t-1} at time $t-1$, motion control u_t, measurements d_t of the range sensors, the three-dimensional grid G_{3D} and SER. It outputs the particle set S_t. Here, N_T denotes the total number of particles used in this algorithm, N_G is the number of global samples distributed in SER, and N_L denotes the number of local samples used for tracking the robot. We explain this algorithm in five parts.

Part1: sampling total particles. Line 2 generates a particle $s_t^{[n]}$ for time t based on the particle $s_{t-1}^{[n]}$ and the control u_t. Line 3 determines the importance weight of that particle. Particularly, measurements of the particle are searched in G_{3D}.

Part2: determining the size of global sample set and local sample set. This part distributes the number of global samples and local samples according to the maximum of importance factors ω_t. If ω_t^{max} is less than the threshold ξ, we assume the robot is kidnapped, part of particles N_G are divided as global samples. If not, all the particles are local samples. The parameter α determines the ratio of global samples and local samples. Here, the problem of when to generate global samples is solved. The reason why we do not use all the particles as global samples is that the robot may mistakenly believe that it is kidnapped. This more often occurs in incomplete maps. Keeping part of local samples can reduce this mistake. ξ is a sensitive coefficient, which determines the sensitivity of SAMCL. The greater ξ may make robot more sensitive to kidnapping, but on the other hand the robot mistakes more frequently.

Part3: resampling local samples. The operation to resample local samples is identical to regular MCL. At the beginning, importance factors ω_t are normalized. Local samples are drawn by incorporating the importance weights.

Part4: drawing global samples. A real trick of the SAMCL algorithm is in part 4, global samples are distributed in SER with a uniform distribution. The advantage of sampling in SER is more efficient. This part is only executed when the robot considers itself to be kidnapped.

Part5: combining two particle sets. At last, local sample set S_t^L and global

sample set S_t^G are combined. The new sample set S_t will be used in the next iteration.

Algorithm 5.3: SAMCL algorithm

Input: S_{t-1}, u_t, d_t, G_{3D}, SER

Sampling total particles

for $n = 1$ to N_T **do**
  generate a particle $s_t^{[n]} \sim p\left(s_t \left| s_{t-1}^{[n]}, u_t \right.\right)$ % motion model
  calculate importance factor $\omega_t^{[n]} = p\left(z_t \left| s_t^{[n]}, G_{3D} \right.\right)$ % perception model
end for

Determining the size of global sample set and local sample set

if $\omega_t^{max} < \xi$ **then**
  $N_L = \alpha \cdot N_T$
else
  $N_L = N_T$
end if
$N_G = N_T - N_L$

Resampling local samples

normalize ω_t
for $n = 1$ to N_L **do**
  draw $s_t^{[n],L}$ with distribution $\omega_t^{[n]}$
  add $s_t^{[n],L}$ to S_t^L
end for

Drawing global samples

for $n = 1$ to N_G **do**
  draw $s_t^{[n],G}$ with the uniform distribution in SER
  add $s_t^{[n],G}$ to S_t^G
end for

Combining two particle sets

$S_t = S_t^L \cup S_t^G$
Output: S_t

5.3 SIMULATION RESULTS

The SAMCL algorithm inherits all the advantages of MCL, hence it has the ability to solve the position tracking problem and the global localization problem. Moreover, it improves in several aspects compared with the regular MCL.

- It is more efficient than the plain MCL algorithm, since it employs an off-line pre-caching technique.
- Similar Energy Region (SER) provides potential information of the robot's pose. Hence, sampling in SER is more efficient than sampling randomly in the entire environment.
- It can settle the kidnapped robot problem by using self-adaptive samples.

Thus, simulations focus on comparing SAMCL with MCL in computational efficiency and evaluating the performance of the SAMCL algorithm to solve position tracking, global localization and the kidnapped robot problem.

5.3.1 Position tracking

The purpose of this simulation is to evaluate the ability of the SAMCL algorithm to track the robot's position. We use the same quasi-symmetrical corridor map and the same number of particles (300 particles) as testing in MCL (see Figure 4.11). 6% perception noise and 8.82% motion noise are added in the sensor model and the motion model, respectively. Figure 5.4 and Figure 5.5 depict localization results in different ways. The former shows the trajectories of robot, odometry and SAMCL and the latter presents the localization error curves of SAMCL and odometry. From the two figures, it is easy to find that SAMCL performs as well as MCL (see Figure 4.11 and Figure 4.12) in position tracking.

5.3.2 Global localization

This simulation aims at testing the global localization ability of the SAMCL algorithm. The quasi-symmetrical corridor map and 300 particles are used (see Figure 4.13). In order to test the robustness of SAMCL, we add 6% perception noise and 8.82% motion noise to each wheel. The parameter settings and the experimental environment are the same as testing MCL (see Section 4.4.2). Figure 5.6 shows the trajectories of robot, odometry and SAMCL and Figure 5.7 shows the localization error curves of SAMCL and odometry. Since particles are initialized by a random distribution in the global localization problem, the localization errors are bigger at the beginning. But errors decrease with particles converging. Simulation results show that the performance of SAMCL is as well as MCL (see Figure 4.14 and Figure 4.15) in global localization.

5.3.3 Comparison of computational efficiency

As discussed thus far, SAMCL is more efficient than regular MCL due to employing the off-line pre-caching technique. Figure 5.8 plots execution time curves of MCL without the pre-caching technique and SAMCL as a function of the number of particles. The execution time is the robot online implementation time of the first 20 steps. As to be expected, the execution time increases with the number of

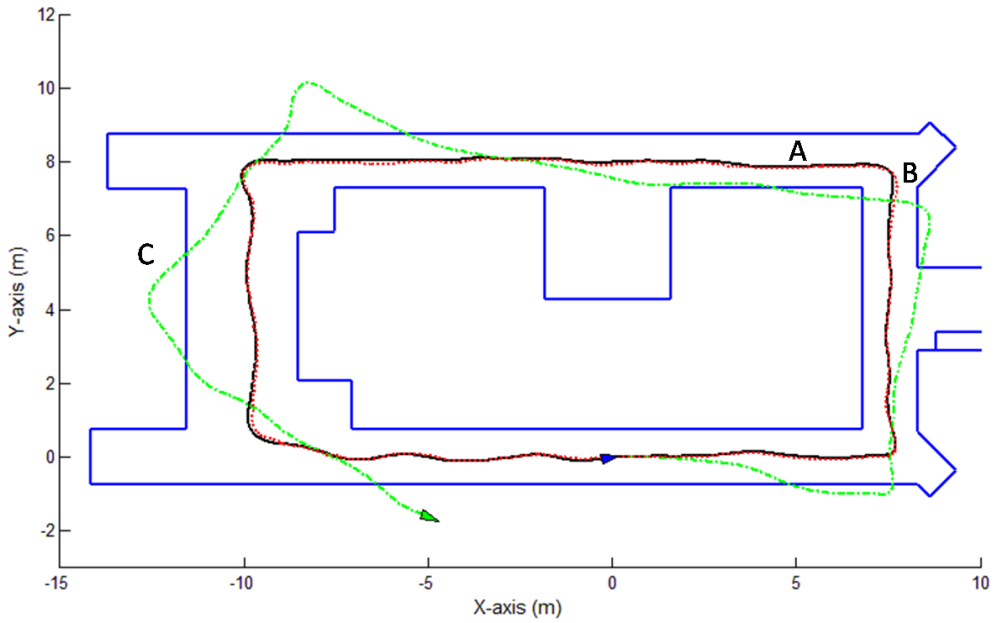


Figure 5.4: Position tracking using SAMCL in a quasi-symmetrical corridor. The trajectories of robot, odometry and SAMCL are displayed by the black solid line (line A), the green dash-dot line (line C) and the red dotted line (line B), respectively.

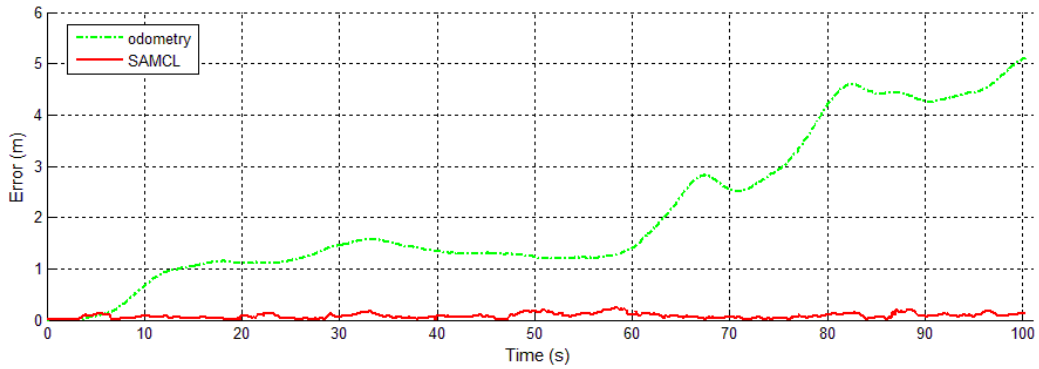


Figure 5.5: Localization errors of position tracking using SAMCL in a quasi-symmetrical corridor. SAMCL errors and odometry errors are plotted by the red solid line and the green dash-dot line, respectively.

particles, both for regular MCL (red dotted line) and for SAMCL (black solid line). However, the augmentation of the execution time of regular MCL is enormous. Particles from 1 to 1000, the execution time of regular MCL increases about 395 seconds, but for SAMCL, the execution time only increases about 4 seconds.

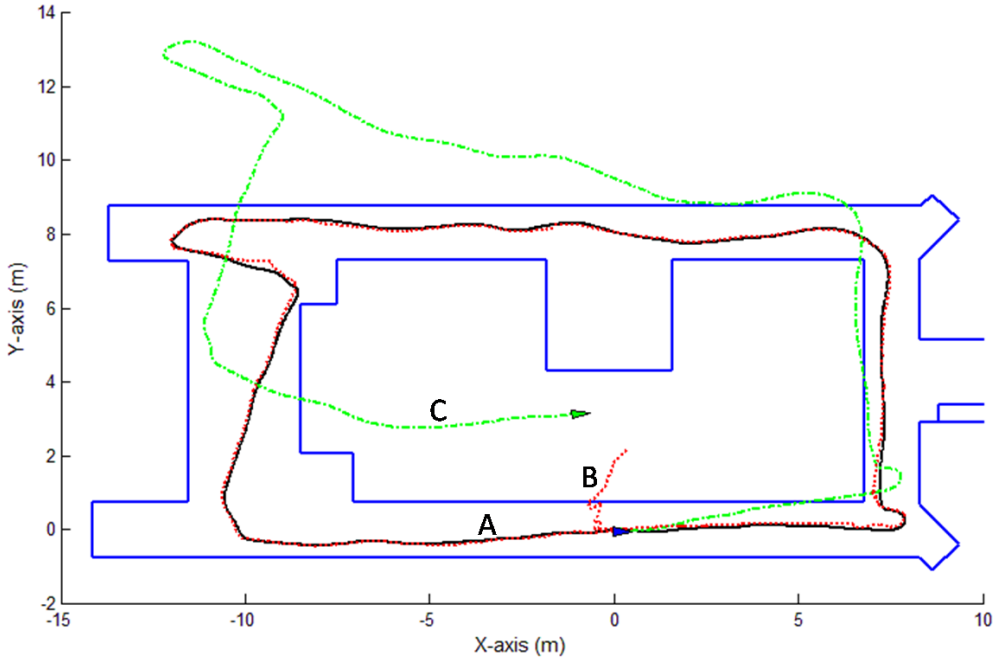


Figure 5.6: Global localization using SAMCL in a quasi-symmetrical corridor. The trajectories of robot, odometry and SAMCL are displayed by the black solid line (line A), the green dash-dot line (line C) and the red dotted line (line B), respectively.

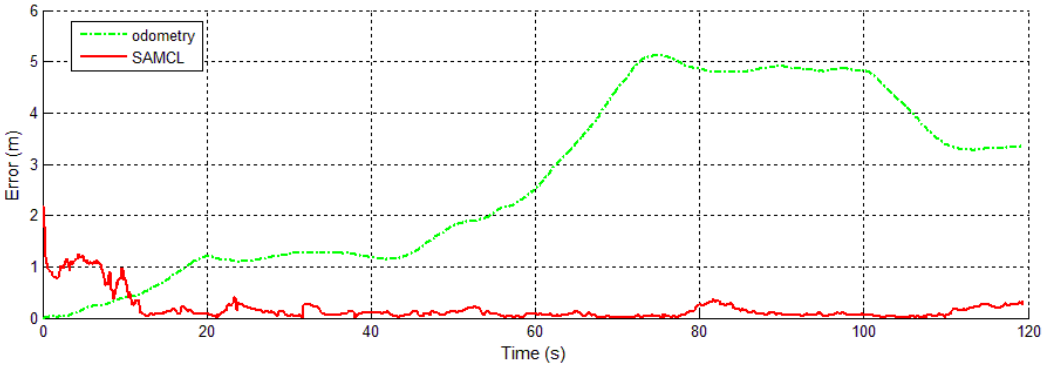


Figure 5.7: Localization errors of global localization using SAMCL in a quasi-symmetrical corridor. SAMCL errors and odometry errors are plotted by the red solid line and the green dash-dot line, respectively.

5.3.4 Kidnapping

Kidnapping is the most difficult problem in three sub-problems of localization. Thus, we design three trials to evaluate the ability of SAMCL to recover from kidnapping. These trials are based on global localization. particles are initialized to distribute randomly in the map with uniform probabilities.

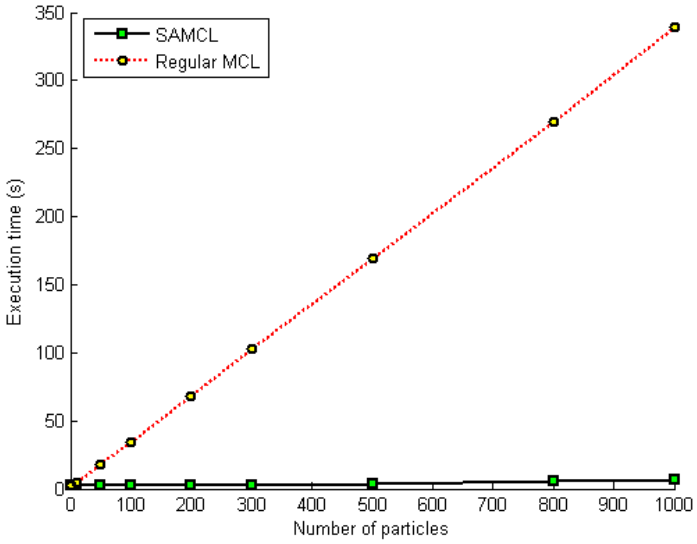


Figure 5.8: Execution time of regular MCL and the SAMCL algorithm as a function of the number of particles.

5.3.4.1 Kidnapping in different environments with known heading direction

In the first simulation, the robot is kidnapped from the corridor to the room located in the middle of the map. To reduce the difficulty, the heading direction of the robot is supposed to be known after kidnapping. We add 6% noise to sensors and 8.82% noise to each wheel. 300 particles are used to estimate the robot's pose.

As discussed in section 3.4, the basic MCL algorithm can solve the global localization problem but cannot recover from robot kidnapping. Since all particles only survive near the most likely pose once the robot's pose is determined, there will be no particle near the new pose. In other words, the plain MCL algorithm does not have ability to re-distribute global samples. That is quite obvious from the results in Figure 5.9. The robot is kidnapped from the corridor (position 1) to the room (position 2) after particles converging. Both particles and odometry fail to track the robot.

The same simulation is executed by the SAMCL algorithm. As shown in Figure 5.10, the robot's trajectory (line A) shows that the robot is kidnapped from the corridor (position 1) to the room (position 2). The odometry's trajectory (line C) shows that odometry has totally lost. However, the trajectory of SAMCL (line B) re-tracks the robot's trajectory (line A) with only little delay.

In order to depict kidnapping more clearly, trajectories are decomposed into X-axis and Y-axis as shown in Figure 5.11. It can be found easily that kidnapping happens both in the X-axis direction and the Y-axis direction at $t = 8.5s$. Actually, the robot is kidnapped from the coordinate $(3.94, 0.28)$ to the coordinate $(-5.80, 5.52)$. In this trial, SAMCL finds and recovers from kidnapping very soon, since the robot is kidnapped between two different environments (from the corridor

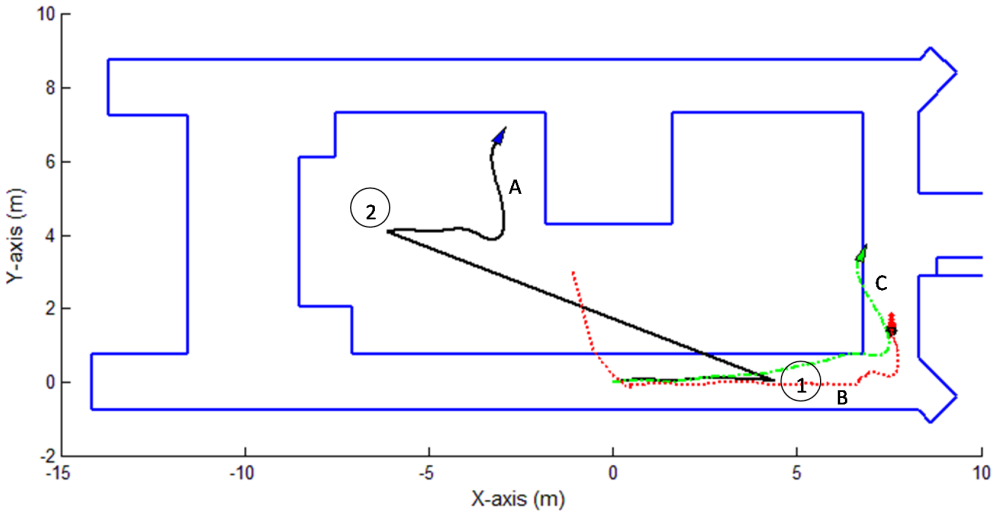


Figure 5.9: MCL for robot kidnapping. The robot is kidnapped from the corridor to the room with known heading direction. The trajectories of robot, odometry and MCL are displayed by the black solid line (line A), the green dash-dot line (line C) and the red dotted line (line B), respectively.

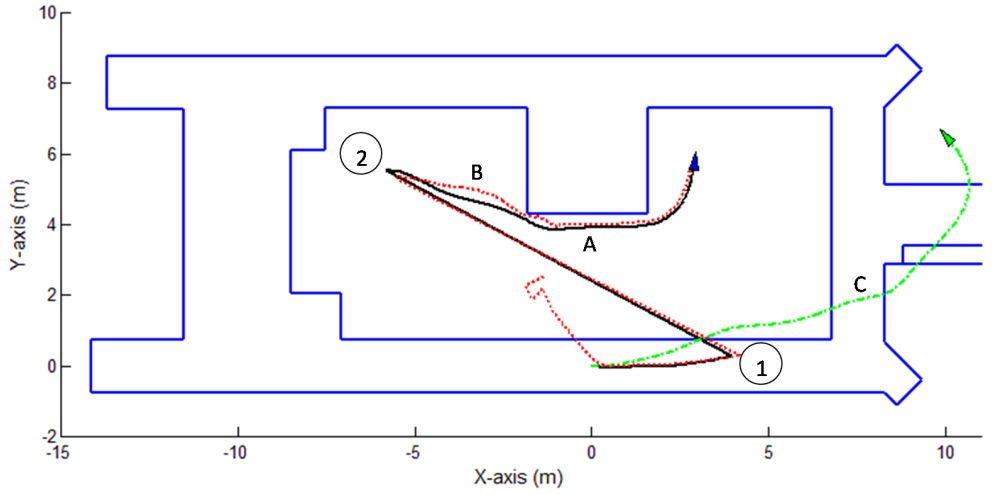
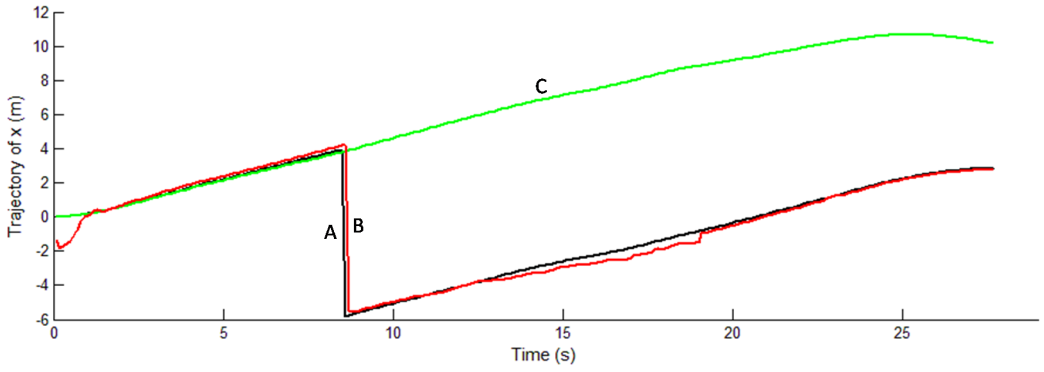


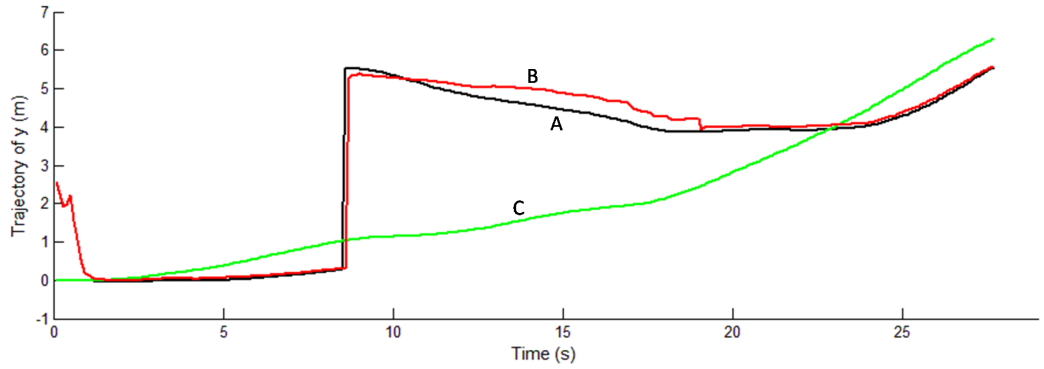
Figure 5.10: SAMCL for robot kidnapping. The robot is kidnapped from the corridor to the room with known heading direction. The trajectories of robot, odometry and SAMCL are displayed by the black solid line (line A), the green dash-dot line (line C) and the red dotted line (line B), respectively.

to the room).

Figure 5.12 plots the localization error curves of SAMCL and odometry. Both SAMCL errors and odometry errors increase suddenly when the robot is kidnapped, however SAMCL recovers in a flash.



(a)



(b)

Figure 5.11: Trajectories are decomposed to (a) X-axis and (b) Y-axis. Line A, line B and line C depict the trajectories of robot, SAMCL and odometry, respectively.

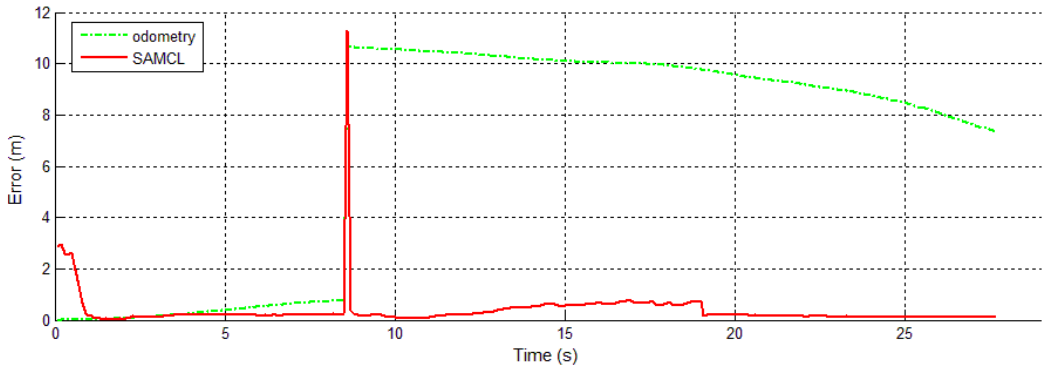


Figure 5.12: Localization errors of SAMCL and odometry. The robot is kidnapped from the corridor to the room with known heading direction. SAMCL errors and odometry errors are plotted by the red solid line and the green dash-dot line, respectively.

5.3.4.2 Kidnapping in the same environment with known heading direction

The second trial is more challenging, since the robot is kidnapped in the same corridor. The SAMCL algorithm cannot find kidnapping until the robot moves

to the lower right corner. The parameter settings and the map used here are the same as the first one. The heading direction of the robot is also supposed to be known after kidnapping.

Figure 5.13 depicts the trajectories of robot, odometry and SAMCL. The robot is kidnapped from position 1 to position 2 in the same corridor. After kidnapping happens, odometry still naively believes that the robot is on the track. However, SAMCL can find and recover from kidnapping. In practice, SAMCL does not perceive kidnapping immediately since kidnapping occur in the same corridor (no environment changes). This is clearly depicted in Figure 5.14.

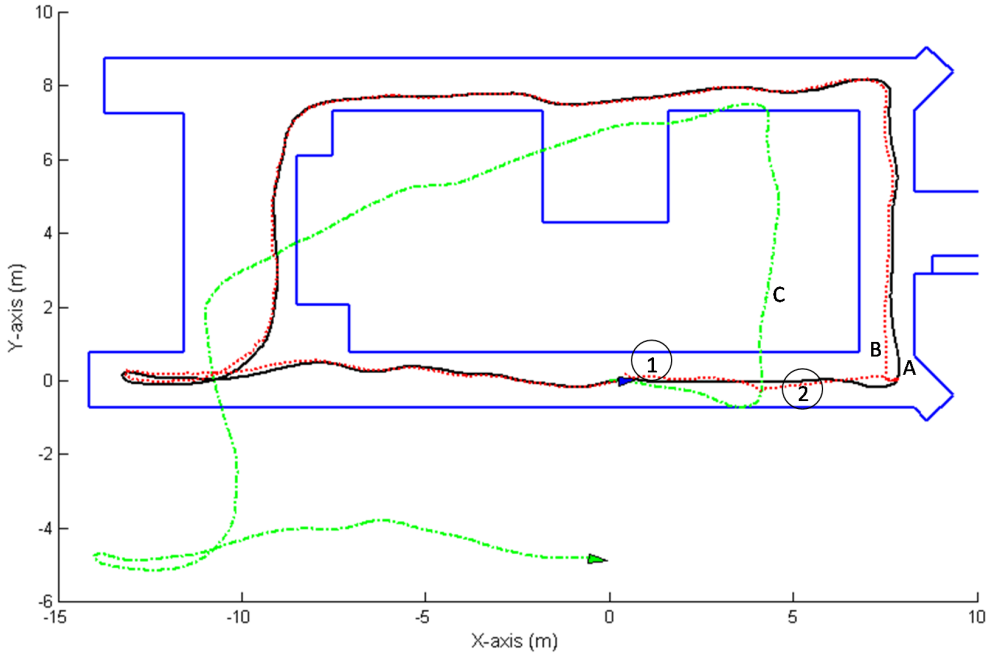
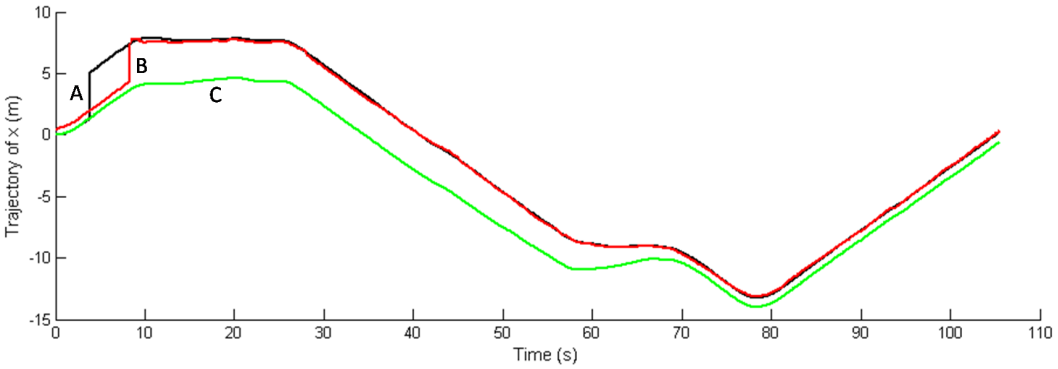


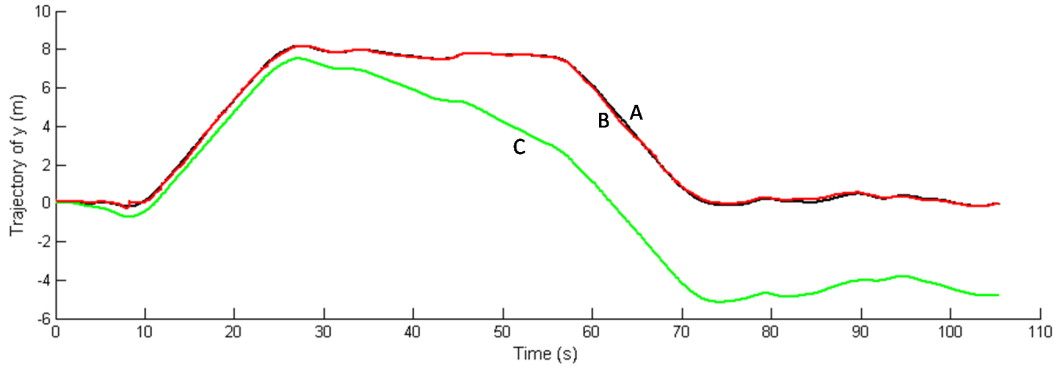
Figure 5.13: SAMCL for robot kidnapping. The robot is kidnapped in the same corridor with known heading direction. The trajectories of robot, odometry and SAMCL are displayed by the black solid line (line A), the green dash-dot line (line C) and the red dotted line (line B), respectively.

In the Figure 5.14, trajectories of robot, SAMCL and odometry are decomposed into X-axis and Y-axis, respectively. From Figure 5.14(a), we can find that the robot is kidnapped about at $t = 3.7s$ and it is abducted about $3.8m$ far away in the X-axis direction. The SAMCL's trajectory shows that SAMCL does not realize kidnapping immediately until the environment changes. Thus, to recover from this global localization failure, SAMCL uses about $4.5s$. In the Y-axis direction, there is no visibly kidnapping occurred (see Figure 5.14(b)).

The localization error curves of SAMCL and odometry are shown in Figure 5.15. Sudden changes of error curves denote that kidnapping has happened. The SAMCL algorithm recovers from kidnapping with some delay but odometry loses itself totally.



(a)



(b)

Figure 5.14: Trajectories are decomposed to (a) X-axis and (b) Y-axis. Line A, line B and line C depict the trajectories of robot, SAMCL and odometry, respectively.

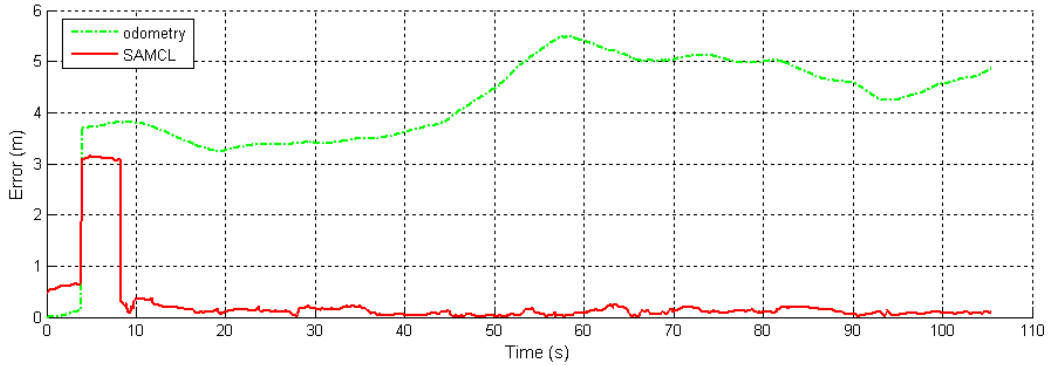


Figure 5.15: Localization errors of SAMCL and odometry. Kidnapping occurs in the same corridor with known heading direction. SAMCL errors and odometry errors are plotted by the red solid line and the green dash-dot line, respectively.

5.3.4.3 Kidnapping in different environments with unknown heading direction

The most difficult one for the robot is the third trial. In the previous two simulations, the heading direction of the robot is supposed to be known after kidnapping.

However, there is no knowledge about the heading direction of the robot in this simulation. That means neither the $x - y$ coordinates nor the orientation are known after the robot is kidnapped. The robot is completely lost. To recover from kidnapping, we have to use more particles. Three times more than the previous two simulations (900 particles) are employed in this simulation.

Figure 5.16 illustrates the trajectories of robot, odometry and SAMCL. Line A shows that the robot is kidnapped from the corridor (position 1) to the room (position 2). Line B depicts that SAMCL can find kidnapping quickly but it does not recover immediately. Since the lack of the heading direction, SAMCL needs some time to converge its particles. Line C shows that odometry is not aware of kidnapping.

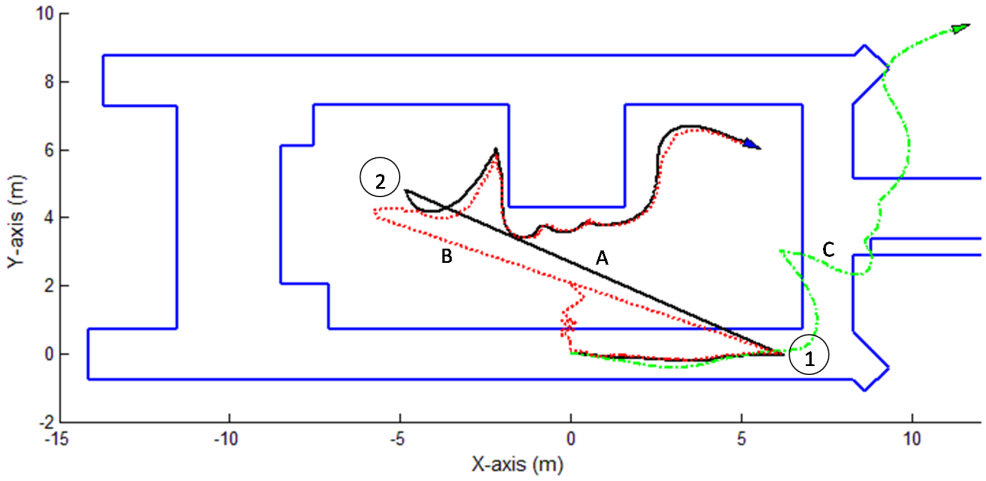


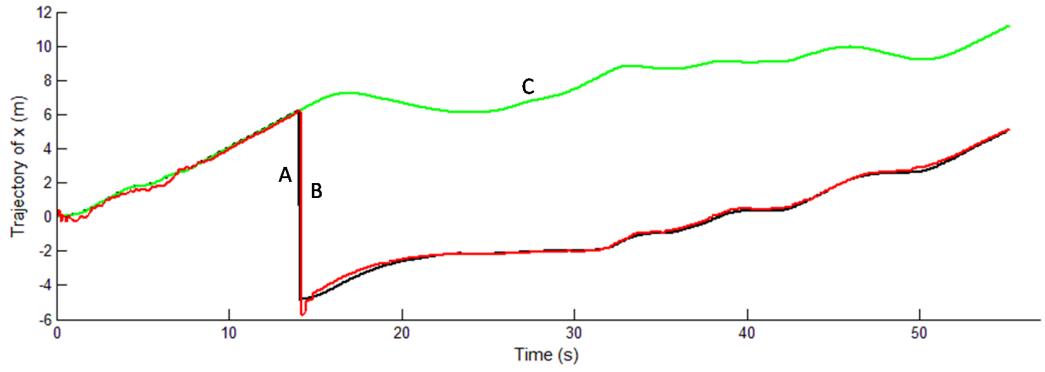
Figure 5.16: SAMCL for robot kidnapping. The robot is kidnapped from the corridor to the room with unknown heading direction. The trajectories of robot, odometry and SAMCL are displayed by the black solid line (line A), the green dash-dot line (line C) and the red dotted line (line B), respectively.

After trajectories are decomposed into X-axis and Y-axis (as shown in Figure 5.17), we can find that the robot is kidnapped from the coordinate $(6.24, -0.02)$ to the coordinate $(-4.85, 4.84)$ at $t = 14s$. SAMCL finds and recovers from kidnapping within $1s$.

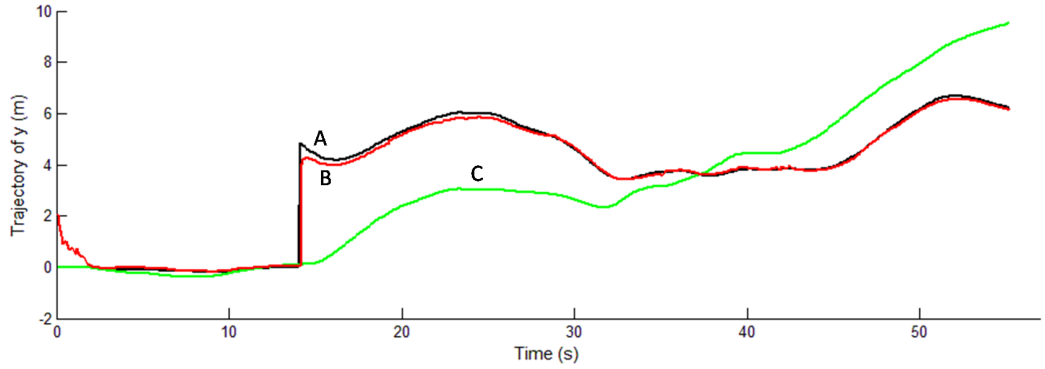
Figure 5.18 plots the localization error curves of SAMCL and odometry. The same as previous two trials, SAMCL can recover from kidnapping quickly and then localize the robot accurately.

5.3.4.4 Kidnapping in the same environment with unknown heading direction

The case of kidnapping occurred in the same environment with unknown heading direction is similar to the previous simulations. However, there are more SERs when the robot lies in the corridor than it lies in a distinct region (as shown in



(a)



(b)

Figure 5.17: Trajectories are decomposed to (a) X-axis and (b) Y-axis. Line A, line B and line C depict the trajectories of robot, SAMCL and odometry, respectively.

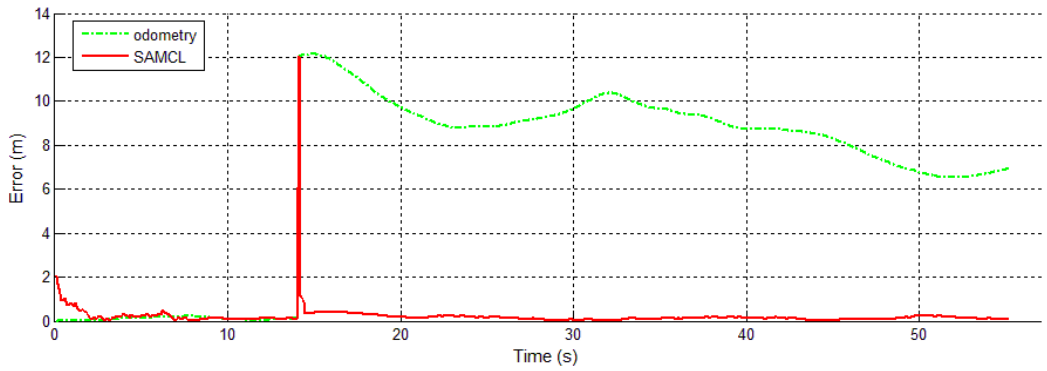


Figure 5.18: Localization errors of SAMCL and odometry. The robot is kidnapped from the corridor to the room with unknown heading direction. SAMCL errors and odometry errors are plotted by the red solid line and the green dash-dot line, respectively.

Figure 5.3). Hence, to recover from kidnapping, the algorithm needs more samples (even more than Kidnapping in Section 5.3.4.3). This leads to an augmentation

of computation and it is difficult to implement the algorithm in real-time.

5.4 SUMMARY

In this chapter, we presented an improved Monte Carlo localization with self-adaptive samples (SAMCL) to solve the localization problem. Comparisons of SAMCL and other three plain Markov localization algorithms (EKF, grid localization and MCL) are summarized in Table 5.1, which is the complement of Table 3.1.

Table 5.1: Comparison of SAMCL, EKF, grid localization and MCL.

	EKF	Grid localization	MCL	SAMCL
Posterior representation	Gaussian (μ_t, Σ_t)	histogram	particles	particles
Position Tracking	yes	yes	yes	yes
Global Localization	no	yes	yes	yes
Kidnapping	no	yes	no	yes
Efficiency	fast	slow	medium	fast

- The SAMCL algorithm inherits all the advantages of MCL, moreover it improves in several aspects. SAMCL employs an off-line pre-caching technique to reduce the expensive on-line computational costs of regular MCL. We defined Similar Energy Region (SER), which provides potential information of the robot's pose. Hence sampling in SER is more efficient than sampling randomly in the entire environment. By using self-adaptive samples, SAMCL can deal with the kidnapped robot problem as well as position tracking and global localization.

- We tested respectively the abilities of SAMCL to solve position tracking, global localization and the kidnapped robot problem by simulations. Position tracking and global localization were tested by using the same simulation settings as MCL. Results show that SAMCL performs as well as MCL both in position tracking and global localization.

- We compared SAMCL with regular MCL in computational efficiency. Due to employing the pre-caching technique, SAMCL is much more efficient than regular MCL without the pre-caching technique.

- Kidnapping was tested by three simulations with different difficulties. In the first one, the robot is kidnapped from the corridor to the room and its heading direction after kidnapping is supposed to be known. In the second one, the

robot is kidnapped in the same corridor. It is more difficult because SAMCL cannot feel kidnapping in the same environment. Kidnapping is recovered until the robot moves into a different terrain. In this simulation, the robot also knows its heading direction after kidnapping. The third one is the most challenging, since there are no knowledge about the heading direction of the robot after it is kidnapped from the corridor to the room. SAMCL performed well in all the three simulations.

CHAPTER 6

MULTI-ROBOT LOCALIZATION

Contents

6.1 INTRODUCTION

Cooperative multi-robot localization is different from single-robot localization or multiple robots independent localization. It emphasizes cooperative localization and allows robots to exchange and share information through communication. Thus cooperative multi-robot localization has following advantages [Cao95,Fox00a, Fox00b, Roumeliotis02, Burgard05, Liu05].

- Firstly, multiple robots can exchange their position information, which might increase the robustness and efficiency of the localization algorithm. If one robot of a robot team has been well localized, the confidence of its own position would affect any other robot who can communicate with it. All these robots can upgrade their positions based on their internal relationships with the well localized robot.

- Secondly, multiple robots can share their sensor information, which might lower the costs of the entire system. For example, several robots of a robot team are equipped with expensive and accurate sensors such as laser range finders, whereas others are only equipped with low cost sensors such as ultrasonic range finders. By sharing sensor information, this robot team can almost get the

same performance as the team where each robot is equipped with expensive and accurate sensors.

- Thirdly, multiple robots can share information among different sensor platforms, such as some robots are equipped with vision sensors and others are equipped with range sensors, which make this robot team be able to adapt to a more complex environment.

In order to solve the cooperative multi-robot localization problem, many variations of MCL have been proposed. Fox *et al.* propose a sample-based version of Markov localization [Fox00a, Fox00b]. They use probabilistic detection models to model the robots' abilities to recognize each other. When one robot detects another, these detection models are used to synchronize the individual robots' beliefs. They employed density trees [Omohundro90] to integrate information from other robot's belief.

Liu *et al.* propose a MCL approach based on grid cells and characteristic particles [Liu05]. They use grid cells to partition the whole particle set into several areas and a changeable grid cells method to get the characteristic particles. The characteristic particles are used to represent the whole particle set. Thus, the robot just synchronizes beliefs of the characteristic particle set got from another. This approach can reduce the communication delay and the computational complexity to a certain extent.

Gasparri *et al.* propose an alternative approach that does not rely on Bayesian frameworks. It is named the Bacterial Colony Growth Framework (BCGF) [Gasparri08a, Gasparri08b]. This approach is based on the models of species reproduction and it provides a framework for carrying on the multi-hypothesis. The authors devised an algorithm extension (CBCG) to exploit information derived by collaboration among robots.

We devise the Position Mapping (PM) algorithm to integrate information derived from cooperation among robots [Zhang09c, Zhang09d]. The PM algorithm is integrated into the SAMCL algorithm as an extension. Compared with other multi-robot localization algorithms, the PM algorithm has two advantages. Firstly, it allows one robot to cooperate with K robots rather than one robot at the same time. Secondly, it synchronizes only one position and one belief instead of information of the whole particle set. This reduces furthest the communication delay and the computational complexity.

6.2 THE POSITION MAPPING APPROACH

For cooperative multi-robot localization, when robots are within their range of visibility, three tasks should be completed by each robot: detection, data exchange and location [1] update.

[1]Concepts of a position (or pose) and a location have been distinguished in Section 2.2.1.1. A position comprises the $x - y$ coordinates and the heading direction θ, but a location only comprises the $x - y$ coordinates. We emphasize the location update, since no additional direction

- **Detection.** In cooperative multi-robot localization, perception data are distinguished into two types: environment measurements and robot detections.

$$Z_t = \left\{ Z_t^e, Z_t^d \right\} \tag{6.1}$$

where environment measurements are denoted as Z_t^e, which are identical with Equation 3.2 in Section 3.1.1.

$$Z_t^e = \left\{ z_t^e, z_{t-1}^e, \cdots, z_0^e \right\} \tag{6.2}$$

Robot detections are denoted as Z_t^d, which provide the information of presence or absence of other robots and their relative relationships. If one robot R_i detects K other robots, we have

$$Z_t^d = \left\{ z_t^{R_{i,1}}, z_t^{R_{i,2}}, \cdots, z_t^{R_{i,K}} \right\} \tag{6.3}$$

where the term $z_t^{R_{i,j}}$ represents measurements (such as geometric relationships, colors, and shapes) obtained by the robot R_i when the robot R_i detects the robot R_j.

Now suppose environment measurements Z_t^e and robot detections Z_t^d are independent. The probability $p\left(s_t \left| Z_t \right.\right)$ can be calculated as follows:

$$\begin{aligned} p\left(s_t \left| Z_t \right.\right) &= p\left(s_t \left| Z_t^e, Z_t^d \right.\right) \\ &= p\left(s_t \left| Z_t^e \right.\right) \cdot p\left(s_t \left| Z_t^d \right.\right) \end{aligned} \tag{6.4}$$

where the calculation of the term $p\left(s_t \left| Z_t^e \right.\right)$ is identical with Equation 3.4. The probability $p\left(s_t \left| Z_t^d \right.\right)$ is not calculated directly. It will be merged into $p\left(s_t \left| Z_t \right.\right)$ by the PM algorithm.

- **Data exchange.** If one robot detects K other robots, it sends data to K other robots, including its own estimated location, the confidence of this location and detections. At the same time, it receives the same data from K other robots.
  - The estimated location is denoted as l_t, which can be the mean μ_t of EKF, grid cells of grid localization or particles of MCL.
  - The confidence represents how much the robot trusts in the estimation of its location, which is denoted as $\hat{\omega}_t$.
  - Detection is denoted as Z_t^d as above. It represents relative relationships between the robot and other robots.

  An example of data exchange among three robots is shown in Figure 6.1.

constraints are used, otherwise it is called position update. However, this approach is named position mapping since it can also handle the case when considering direction information.

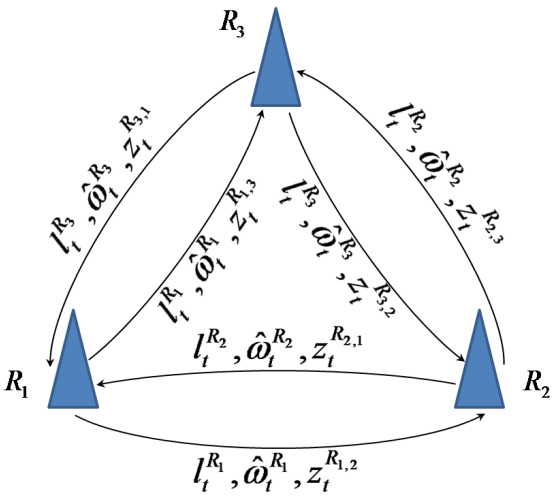


Figure 6.1: Data exchange among three robots.

- **Location update.** Based on the estimated locations received from K other robots and relationships extracted from detections, the robot calculates K possible locations. In other words, if one robot detects K other robots, K possible locations will be mapped. Thus, the position (location) mapping is a bijective mapping (bijection), which can be stated in equation as follows:

$$l_t^p = f(l_t^e) \tag{6.5}$$

where, l_t^p represents a possible location and l_t^e represents an estimated location received from another robot. The function $f(*)$ is a one-to-one correspondence.

These possible locations replace the locations of particles owning the minimum importance weight. The importance weights of these particles are replaced by the confidence received from K other robots. Then the estimated location of the robot is updated actually after resampling.

6.3 ALGORITHM DESCRIPTION

This section introduces the concrete implementation of the PM algorithm. As mentioned above, a robot using the PM algorithm has the ability to cooperate with K other robots at the same time, which is realized by projecting K possible locations. However, to present conveniently, multi-robot cooperation is simplified to bi-robot cooperation. For example, the cooperation of three robots shown in Figure 6.1 can be decomposed to three couples of bi-robot cooperation. We define generally two robot R_i and R_j, where $i \in \{1, \cdots, K+1\}$, $j \in \{1, \cdots, K+1\}$ and $i \neq j$. The PM algorithm will be discussed in two cases: one is that the robot R_i detects the robot R_j but the robot R_j does not detect the robot R_i; the other is that the robot R_i and the robot R_j detect each other.

6.3.1 Only one robot detects the other

In this case, only the robot R_i detects the robot R_j. This may occur when the sensors of the robot R_j are incapable of detecting other robots; or when the robot R_i is out of the measurement range of the robot R_j. We will discuss how both two robot update their location using only one detection.

- **Detection.** We assume that the sensors of the robot R_i have the ability to measure geometric relationships to the robot R_j, including the distance $d_{i,j}$ and the angle $\alpha_{i,j}$ between locations of two robots. Figure 6.2 shows the geometric relationships when the robot R_i detects the robot R_j. In practice, the PM algorithm use the angle $\vartheta_{i,j}$ to calculate the possible location. Based on the obvious geometric laws, we have

$$\vartheta_{i,j} = \theta_i - \alpha_{i,j} \tag{6.6}$$

where the subscript $_{i,j}$ denotes the measurements provided by the robot R_i when it detects the robot R_j and the subscript $_i$ denotes the parameters belonging to the robot R_i. θ_i is the heading direction of the robot R_i, which can be obtained by odometry, localization algorithms or direction sensors. $\alpha_{i,j}$ is the relative angle between two robot, which is extracted from sensors of the robot R_i directly.

Thus, the detection $z_t^{R_{i,j}}$ provided by the robot R_i can be represented as:

$$z_t^{R_{i,j}} = \begin{pmatrix} d_{i,j} \\ \vartheta_{i,j} \end{pmatrix} \tag{6.7}$$

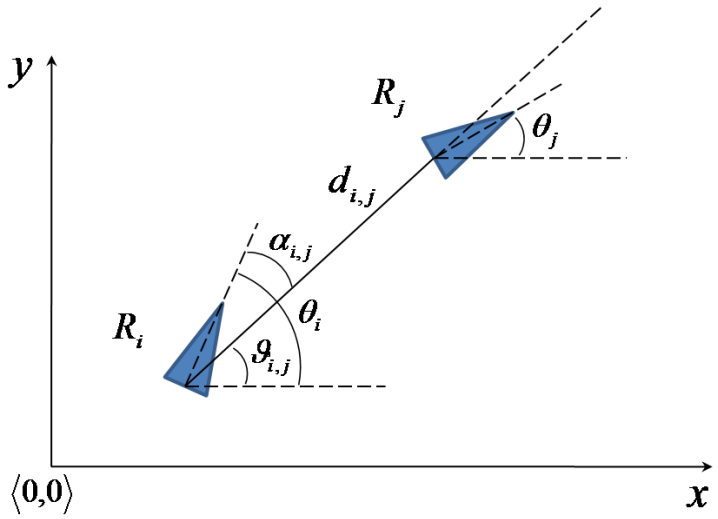


Figure 6.2: Geometric relationships when the robot R_i detects the robot R_j.

- **Data exchange.** When the robot R_i detects the robot R_j, they will exchange estimated locations and the confidence between each other. Moreover, they will share the detection $z_t^{R_{i,j}}$. Here, the estimated location is defined as the expected value (or called weighted mean) of particles (see Equation 4.2). The confidence is defined as the maximum of importance weights. In equations, we have

$$l_t^{R_i} = \mathrm{E}(L_t^{R_i}) \tag{6.8}$$

$$\hat{\omega}_t^{R_i} = \max(\omega_t^{R_i}) \tag{6.9}$$

$$R_i \xrightarrow{l_t^{R_i}, \hat{\omega}_t^{R_i}, z_t^{R_{i,j}}} R_j \tag{6.10}$$

where $l_t^{R_i}$ is the expected value of $L_t^{R_i}$ and $L_t^{R_i}$ denotes the location set of particles of the robot R_i. $\hat{\omega}_t^{R_i}$ represents the confidence of the location $l_t^{R_i}$. $z_t^{R_{i,j}}$ denotes the detection generated by the robot R_i. The arrow represents the flow direction of data.

For the robot R_j, we have the same calculations. Since the robot R_j does not detect the robot R_i, it only sends its estimated location $l_t^{R_j}$ and the confidence $\hat{\omega}_t^{R_j}$ to the robot R_i

$$l_t^{R_j} = \mathrm{E}(L_t^{R_j}) \tag{6.11}$$

$$\hat{\omega}_t^{R_j} = \max(\omega_t^{R_j}) \tag{6.12}$$

$$R_j \xrightarrow{l_t^{R_j}, \hat{\omega}_t^{R_j}} R_i \tag{6.13}$$

- **Location update.** The real "trick" lies in this step. Based on the estimated location of the robot R_j and the detection, the robot R_i caculates one possible location. The particles of the robot R_i owning the minimum importance weight are replaced by the possible location. Its importance weight are replaced by the confidence received from the robot R_j. This process can be described in equation as follows.

$$\underbrace{\begin{pmatrix} x_t^{Ri,min} \\ y_t^{Ri,min} \end{pmatrix}}_{l_t^{Ri,min}} = \underbrace{\begin{pmatrix} x_t^{R_j} \\ y_t^{R_j} \end{pmatrix}}_{l_t^{R_j}} + d_{i,j} \cdot \begin{pmatrix} cos(\vartheta_{i,j}) \\ sin(\vartheta_{i,j}) \end{pmatrix} \tag{6.14}$$

$$\omega_t^{R_i,min} = \hat{\omega}_t^{R_j} \tag{6.15}$$

where $l_t^{Ri,min}$ represents the location of the particle with the minimum importance weight of the robot R_i and $l_t^{R_j}$ is the estimated location of the robot R_j.

The location of the robot R_i will be updated after resampling. We discuss three possibilities for the location update.

- $\omega_t^{R_i,max} << \hat{\omega}_t^{R_j}$**.** In this case, the robot R_j is more confident of its position than the robot R_i. It often occurs when the robot R_j is localized better than the robot R_i. Thus, the robot R_i updates its location.
- $\omega_t^{R_i,max} >> \hat{\omega}_t^{R_j}$**.** On the contrary, the robot R_i is more confident of its position in this case. Thus the confidence received from R_j is helpless to update the location of the robot R_i.
- $\omega_t^{R_i,max} \approx \hat{\omega}_t^{R_j}$**.** In this case, the robot R_i and the robot R_j have the same confidence in their position. Whether both two robots trust their locations or not, the update is valueless.

By sharing the detection obtained from the robot R_i, the robot R_j can update its location through the same process.

6.3.2 Two robots detect each other

In this case, both two robots have the ability to detect each other, and that they generate detections to each other.

- **Detection.** For the robot R_i, it generates the detection $z_t^{R_{i,j}}$ to the robot R_j.

$$z_t^{R_{i,j}} = \begin{pmatrix} d_{i,j} \\ \vartheta_{i,j} \end{pmatrix} \tag{6.16}$$

At the same time, the robot R_j detects the robot R_i.

$$z_t^{R_{j,i}} = \begin{pmatrix} d_{j,i} \\ \vartheta_{j,i} \end{pmatrix} \tag{6.17}$$

- **Data exchange.** In this step, two robots exchange the estimated location, the confidence of this location and the detection between each other.

$$R_i \xrightarrow{l_t^{R_i}, \hat{\omega}_t^{R_i}, z_t^{R_{i,j}}} R_j \tag{6.18}$$

$$R_j \xrightarrow{l_t^{R_j}, \hat{\omega}_t^{R_j}, z_t^{R_{j,i}}} R_i \tag{6.19}$$

- **Location update.** Difficulties arise in this step, since both two robots have two detections. For example, the robot R_i has its own detection $z_t^{R_{i,j}}$ and the detection $z_t^{R_{j,i}}$ received from the robot R_j. The question is which one will be used to calculate the possible location. To solve this problem, we establish two rules:

  1. If the detections are obtained from different types of sensors, robots trust always the most precise one.
  2. If the detections are obtained from the same type of sensors, each robot trusts its own one.

  The first rule ensures that robots in the team can share the precise sensors. The second rule avoids introducing measurement errors to the robot team.

The PM algorithm as the extension is inserted into the SAMCL algorithm. The extended SAMCL algorithm for multi-robot localization is stated in Algorithm 6.1. We only introduce the part of the PM algorithm, the other parts have been presented in Section 5.2.3.

The PM algorithm. If the robot detects other robots, the PM algorithm will be activated. Line 3 computes the minimum importance factor. In line 4, a possible location is mapped from the location of R_k and the geometric relationship between them. This possible location replaces the location of the minimum weight particle. Generalized notations d and ϑ are the distance and the orientation in the global reference frame between two robots, which can be extracted and computed from sensor measurements. The minimum importance weight (ω_t^{min}) is replaced by the confidence received from R_k ($\hat{\omega}_t^{R_k}$).

6.4 SIMULATION RESULTS

This simulation tests the ability of cooperative multi-robot localization. The central question driving our simulation is: to what extent cooperative multi-robot localization improves the localization quality compared with single-robot localization?

In this scenario, we implemented the extended SAMCL algorithm on two individual computers, which simulated two robots (the robot R_1 and the robot R_2).

Algorithm 6.1: Extended SAMCL algorithm for multi-robot localization

1: **Input:** S_{t-1}, u_t, d_t, G_{3D}, SER

Sampling total particles

1: **for** $n = 1$ to N_T **do**
2: generate a particle $s_t^{[n]} \sim p\left(s_t \left| s_{t-1}^{[n]}, u_t \right.\right)$ % motion model
3: calculate importance factor $\omega_t^{[n]} = p\left(z_t \left| s_t^{[n]}, G_{3D} \right.\right)$ % perception model
4: **end for**

The PM algorithm

1: **if** the robot detects K other robots $\{R_1, \cdots, R_K\}$ **then**
2: **for** $k = 1$ to K **do**
3: $\omega_t^{min} = min(\omega_t)$
4: $\begin{pmatrix} x_t^{min} \\ y_t^{min} \end{pmatrix} = \begin{pmatrix} x_t^{R_k} \\ y_t^{R_k} \end{pmatrix} + d \cdot \begin{pmatrix} cos(\vartheta) \\ sin(\vartheta) \end{pmatrix}$
5: replace ω_t^{min} by $\hat{\omega}_t^{R_k}$
6: **end for**
7: **end if**

Determining the size of global sample set and local sample set

1: **if** $\omega_t^{max} < \xi$ **then**
2: $N_L = \alpha \cdot N_T$
3: **else**
4: $N_L = N_T$
5: **end if**
6: $N_G = N_T - N_L$

Resampling local samples

1: normalize ω_t
2: **for** $n = 1$ to N_L **do**
3: draw $s_t^{[n],L}$ with distribution $\omega_t^{[n]}$
4: add $s_t^{[n],L}$ to S_t^L
5: **end for**

Drawing global samples

1: **for** $n = 1$ to N_G **do**
2: draw $s_t^{[n],G}$ with the uniform distribution in SER
3: add $s_t^{[n],G}$ to S_t^G
4: **end for**

Combining two particle sets

1: $S_t = S_t^L \cup S_t^G$
2: **Output:** S_t

Both can communicate by wireless network (WiFi). The effective communication distance is set to $5m$. The maximum communication delay was limited to $0.5s$. If the communication time exceeds this limitation, the robot will discard these data. As mentioned previously, our algorithm only needs to exchange very little data, so the $0.5s$ of delay is sufficient in our trial.

In order to evaluate the benefits of multi-robot localization, we gave the sensitive coefficient ξ of the robot R_1 a very small value, so the robot R_1 is insensitive to kidnapping. The robot R_1 was kidnaped at the beginning of the trial. The initial location of R_1 was at the coordinate $(0, 8)$ in the map, but we told it that its initial location was at $(0, 0)$. Because of the quasi-symmetrical environment and low sensitivity to kidnapping, the robot R_1 did not find that it localized mistakenly until it met the robot R_2, which was always localized well. When the robot R_2 encountered the robot R_1, it sent its location and the confidence of this location to the robot R_1. Then the robot R_1 amended its location through the PM algorithm.

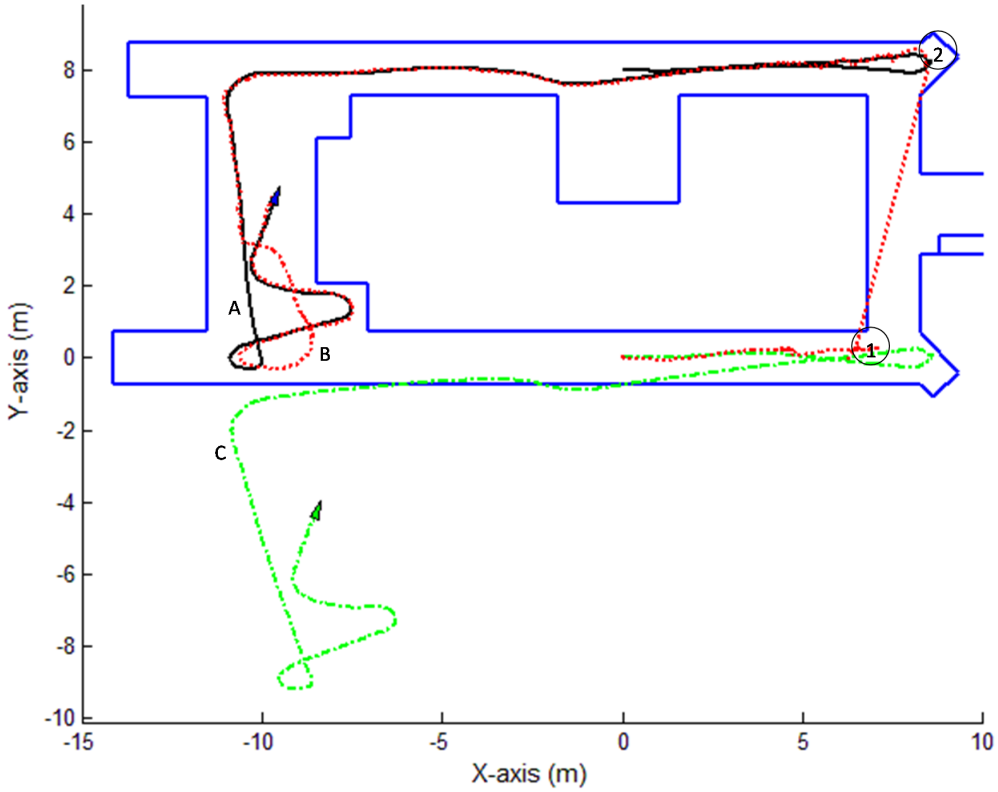
The simulation result is shown in Figure 6.3. Two robots encountered at $58.5s$, and their location were at $(8.5, 8.1)$ and $(7.2, 3.1)$ at that time. As shown in Figure 6.3(a), the localization trajectory (line B) shifts from position 1 to position 2, since at that position the robot R_1 corrects its trajectory.

Figure 6.4 depicts the sample distribution of the robot R_1 (a) and the robot R_2 (b) during the process of cooperative localization. The blue (or deep gray in the grayscale image) triangle denotes the real robot and the green (or light gray) triangle denotes the odometry robot. At $30s$, the robot R_1 and its samples lay in different sides of the corridor due to kidnapping, respectively. But the robot R_2 was localized correctly at that time. At $58.5s$, the robot R_1 was detected by the robot R_2, and then it started to update its location by sharing the detection of the robot R_2. At $59.4s$, the robot R_1 accomplished the location update and it recovered from kidnapping. During the whole process, the robot R_2 was always localized well. After updating their location, both two robots continued to explore in the map with the correct position estimation.

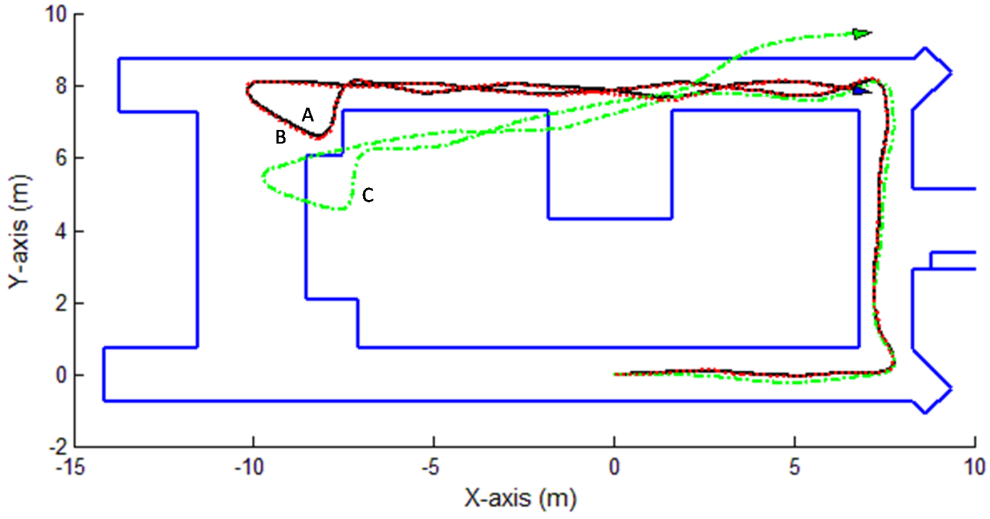
Because the "real" robot is simulated, we can track the position of the "real" robot at all times. Thus, we can calculate the errors relative to the position of the "real" robot. These error curves of the robot R_1 and the robot R_2 are plotted in Figure 6.5(a) and (b). Localization errors and odometry errors are delineated by the red solid line and the green dash-dot line, respectively. As shown in Figure 6.5(a), R_1 was kidnapped at the beginning. Owing to the symmetrical environment, it did not find that it was kidnapped until it met the robot R_2, and then it corrected its position. In this simulation, if the robot R_1 executes alone, it would spend more time on finding and recovering from kidnapping. This reflects the advantage of multi-robot localization compared with single-robot localization.

6.5 SUMMARY

In this chapter, SAMCL was extended to handle multi-robot localization through a position mapping (PM) algorithm.



(a)



(b)

Figure 6.3: Multi-robot localization using the PM algorithm. Trajectories of the robot R_1 and the robot R_2 are shown in (a) and (b). The trajectories of robot, odometry and SAMCL are displayed by the black solid line (line A), the green dash-dot line (line C) and the red dotted line (line B), respectively.

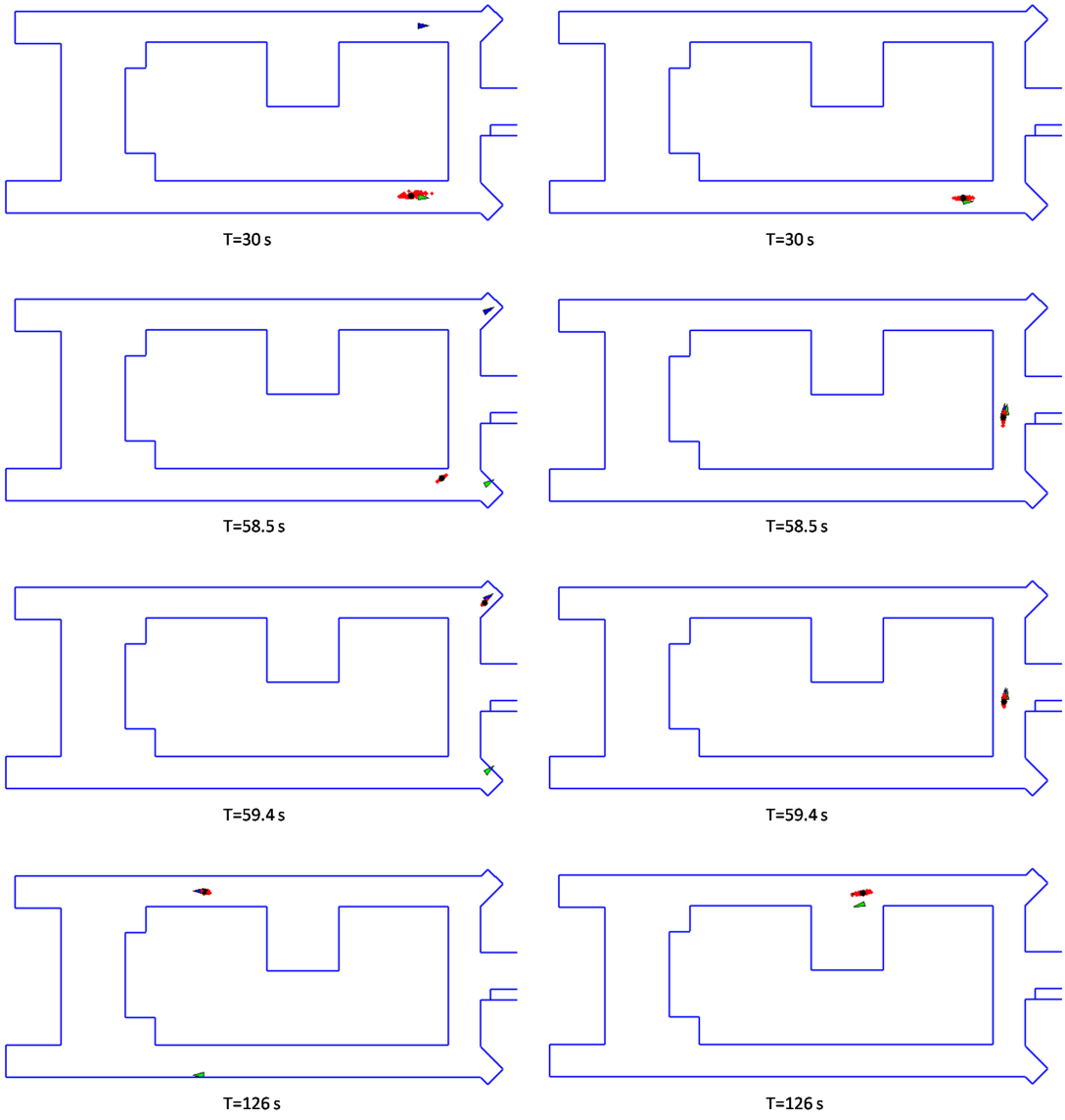
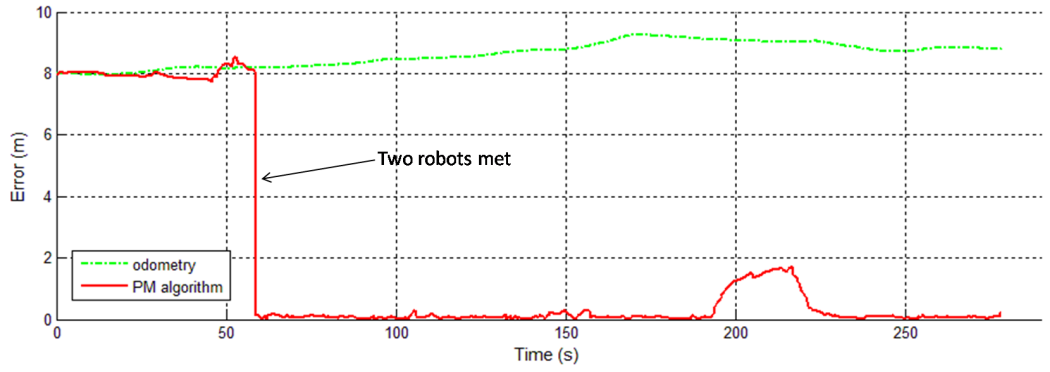


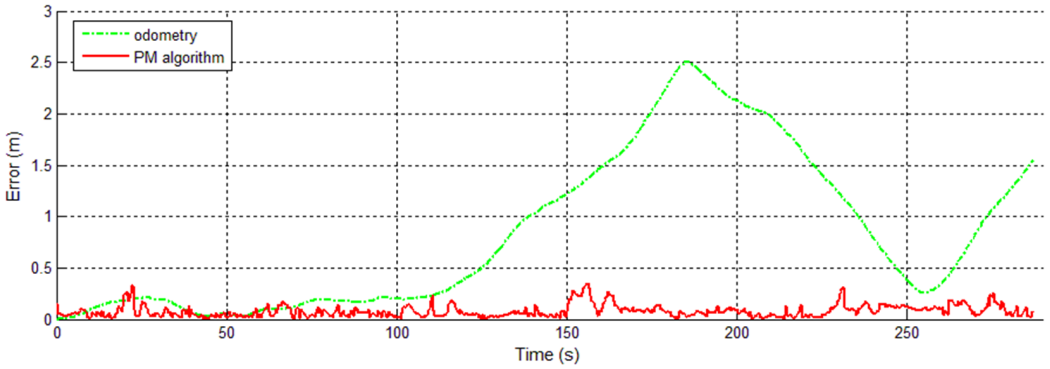
Figure 6.4: Sample distribution of (a) the robot R_1 and (b) the robot R_2 during the process of cooperative localization.

- The PM algorithm integrates information derived from cooperation among robots. In the robot team, one robot can calculate its possible locations from other robots' locations and relationships among them. The relation between the set of possible locations and the set of other robots is a bijection. In other words, one robot is detected, one possible location is mapped. These possible locations replace the locations of the particles owning the minimum importance weight .

- The extended SAMCL algorithm was tested by the simulation. In the simu-



(a)



(b)

Figure 6.5: Localization errors of the PM algorithm and odometry for (a) the robot R_1 and (b) the robot R_2.

lation, two robots are executed in a quasi-symmetrical environment. At the beginning, one robot was kidnapped to the other part of the symmetrical environment and the other robot was localized well. Due to the symmetrical environment, the kidnapped robot cannot be recovered until meeting the other one. The simulation result demonstrates cooperative multi-robot localization is more efficient than single-robot localization.

CHAPTER 7

IMPLEMENTATION

Contents

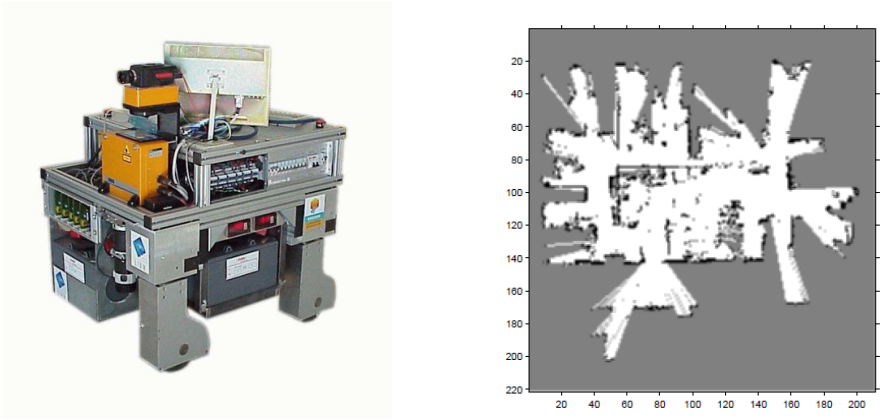
7.1 INTRODUCTION

Over the past decade, thousands of researchers have successfully implemented their algorithms and theories on a variety of platforms. The experiments that are carried out by real robots in real environments can help to narrow the gap between research and application.

Borges *et al.* achieve localization and mapping of the mobile robot in an office environment [Borges02]. Their experimental robot Omni (see Figure 7.1(a)) is equipped with a black and white video camera, a laser range finder and a laser goniometer. The experimental environment is the first floor of the LIRMM laboratory (see Figure 7.1(b)), which is the same as ours.

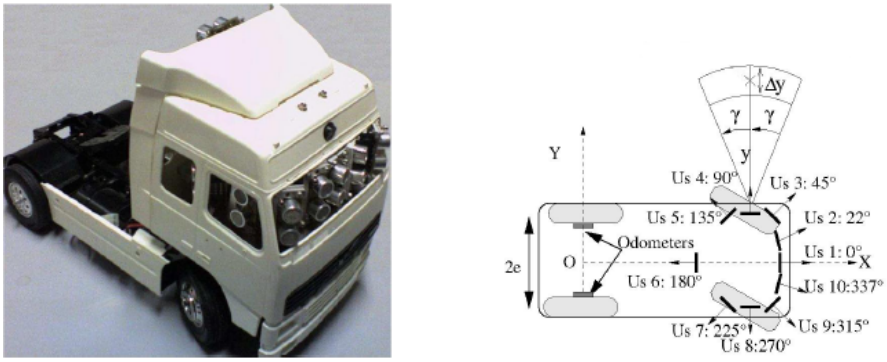
The works of Seignez *et al.* [Seignez06] concern studies and comparisons of four localization methods (including EKF, grid localization, MCL and the localization method based on interval analysis) in an office environment. To test these localization algorithms, they develop the Minitruck platform equipped with 10 ultrasonic range finders (see Figure 7.2).

Burgard *et al.* [Burgard98,Burgard99] develop an autonomous tour-guide robot named "Rhino" (see Figure 7.3(a)). It was deployed for a total of six days in May 1997 in the Deutsches Museum Bonn (see Figure 7.3(b)). A similar robot



(a) (b)

Figure 7.1: (a) Omni robot. (b) Occupancy grid map of the experimental environment. Images courtesy of Geovany A. Borges [Borges02].



(a) (b)

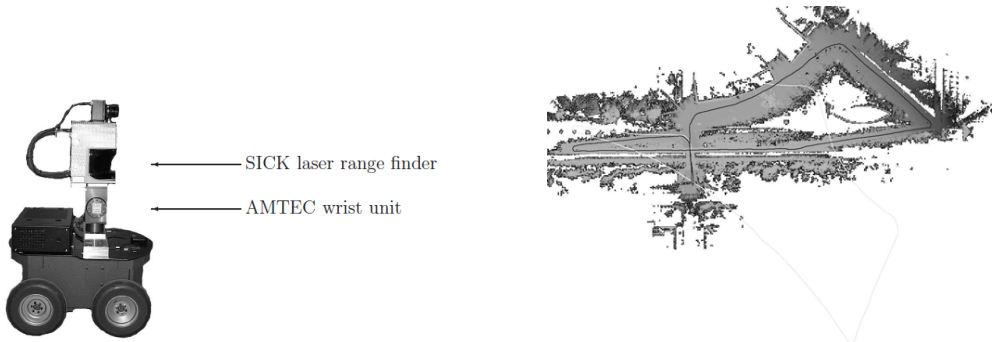
Figure 7.2: (a) The Minitruck platform. (b) Sonar locations. Images courtesy of Emmanuel Seignez [Seignez06].

is developed by Thrun *et al.* [Thrun00a], which is named "Minerva" (see Figure 7.3(c)). It operated for a period of 14 days in the Smithsonian's National Museum of American History (NMAH), during August and September of 1998 (see Figure 7.3(d)). Tasks of the two robots involved approaching people, interacting with them by replaying pre-recorded messages and displaying texts and images on on-board displays as well as safe and reliable navigation in unmodified and populated environments [Thrun99a].

Kümmerle *et al.* propose a novel combination of techniques for robust Monte-Carlo localization of a mobile robot in outdoor-terrains represented by multilevel surface (MLS) maps [Kümmerle08]. The approach is implemented and tested on a Pioneer II AT robot equipped with a SICK LMS laser range scanner and an AMTEC wrist unit (see Figure 7.4(a)) in a urban environment with a non-flat structure and multiple levels (see Figure 7.4(b)).

(a) (b) (c) (d)

Figure 7.3: (a) Rhino robot and its sensors. (b) Rhino gives a tour. (c) Minerva robot. (d) Minerva gives a tour. Images (a) and (b) courtesy of [Burgard98] and images (c) and (d) courtesy of [Thrun00a].



(a) (b)

Figure 7.4: (a) Herbert robot. (b) MLS map. Images courtesy of [Kümmerle08].

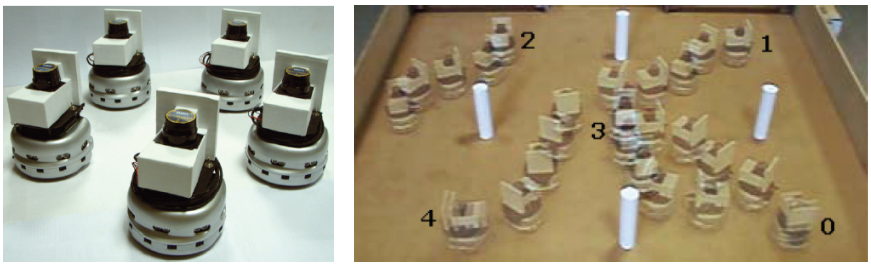
More recently, Sakai *et al.* propose an Augmented Unscented Kalman Filter (AUKF) to solve 6 degrees of freedom (6DOF) localization for planetary rovers [Sakai09]. The algorithm is implemented on the Micro6 robot equipped with a stereo camera, an Inertial Measurement Unit (IMU), and wheel encoders on the 6 wheels (see Figure 7.5(a)) on the outdoor rough terrain (see Figure 7.5(b)).

(a) (b)

Figure 7.5: (a) Micro6: the planetary rover testbed. (b) Experimental fields. Images courtesy of [Sakai09].

The works of Franchi *et al.* [Franchi09] address the mutual localization prob-

lem for a multi-robot system by using an innovative algorithm (the MultiReg algorithm) that computes on-line all the possible relative pose hypotheses, whose output is processed by a data associator and a multiple EKF to isolate and refine the best estimates. The algorithm is tested by a team of 5 Khepera III robots equipped with the Hokuyo URG-04LX laser range finder (see Figure 7.6).



(a) (b)

Figure 7.6: (a) 5 Khepera III robots. (b) Stroboscopic motion in the early steps of the experiment. Images courtesy of [Franchi09].

7.2 EXPERIMENT SETUP

The SAMCL algorithm described in this thesis has been tested with a Pioneer 3-DX mobile robot in a real office environment (see Figure 7.7).

Figure 7.7: Pioneer robot moving in the corridor.

The Pioneer robot is a wheeled mobile robot with two driving wheels and a caster wheel. It is equipped with sixteen ultrasonic range finders distributed around its circumference: two on each side, six forward at 15° intervals and six rear at 15° intervals (see Figure 7.8). As we have already discussed in Section 2.3.1.1,

ultrasonic sensors provide an imprecise and coarse perception of the environment. In the experiments, only ultrasonic range finders (no additional sensors) are used. The maximum ranges of sensors are limited to $5m$. The maximum speed of the Pioneer robot is limited to $0.367m/s$. The Pioneer robot is equipped with an onboard laptop with 1.06GHz Intel Core 2 Solo U2100 CPU and 1024M of RAM, and the SAMCL algorithm is implemented in MATLAB.

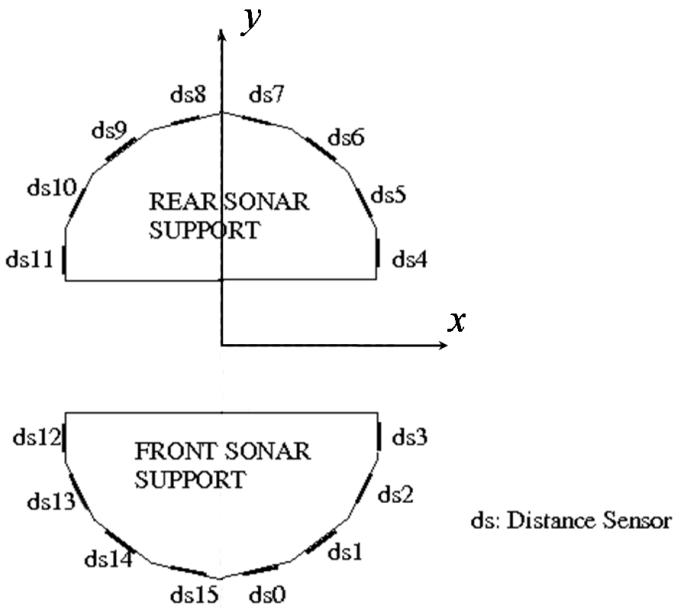


Figure 7.8: Sonar locations on the Pioneer 3-DX robot, adapted from [Cyb].

The experimental environment is the first floor of our laboratory. Its size is about $25m \times 10m$. Figure 7.9 shows the ground plan and the expected trajectory. The robot should follow this trajectory and go around in the corridor. The real environment of this corridor is shown in pictures of Figure 7.9. There are several unmodeled obstacles in the corridor, such as cabinets and tables (see pictures A and B). We use this incomplete map to test the robustness of our algorithm. The SAMCL algorithm inherits the advantages of the MCL algorithm and it employs the mixture perception model (see Section 2.3.4.1), so it can treat these unmodeled obstacles as sensors noise. Because our map is quasi-symmetrical, to recover from kidnapping in such maps is more difficult. The resolution of the three-dimensional grid G_{3D} is $0.2m \times 0.2m \times pi/32$ and the resolution of the energy grid G_E is $0.2m \times 0.2m$ in the experiments.

7.3 EXPERIMENTS

Three experiments were performed, each of them examining the SAMCL algorithm in different situations. The first one aims at testing the ability of global localization by using wheel encoder reading of the Pioneer robot as odometry. The second one focuses on testing the robustness of our method by adding artificial errors to wheel encoder reading. The last one tests the ability of recovering from kidnapping. In

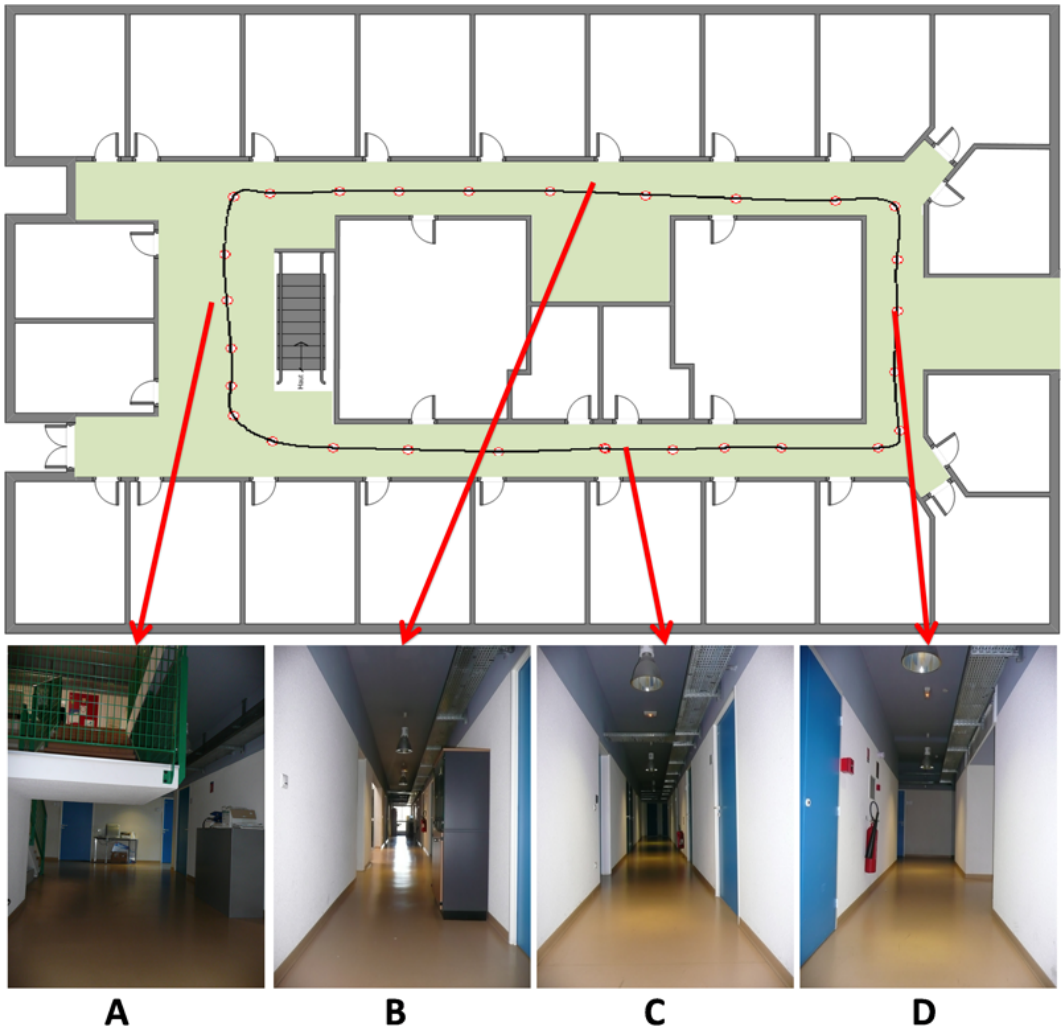


Figure 7.9: The ground plan including the expected trajectory. Pictures show the real environment (with unmodeled obstacles).

order to get reliable statistical results, each experiment is repeated 20 times. Since we did not employ any additional means to track the real robot, the final pose of the real robot is measured by hands at each experiment.

7.3.1 Global localization

The first experiment is designed to test the global localization ability of the SAMCL algorithm. Odometry was obtained from wheel encoder reading of the Pioneer robot. The initial pose of the robot was set differently to the initialization of odometry (a pose $(0, 0, 0)^T$). The initial orientation of the robot was given about $\theta \approx pi/4$ and its initial position was about $1m$ far from the origin of coordinates. The Pioneer robot moved around in the corridor and localized itself. Because of testing the ability of localization, the sensitive coefficient ξ was given a low sensitive value.

Figure 7.10 shows localization results. As usual, the localization result is represented by the expected value (or called the weighed mean) of particles at each iteration, which can be calculated by Equation 4.2 (see Section 4.4.1). Line A denotes the SAMCL's trajectory and line B denotes the trajectory given by odometry. The SAMCL's trajectory is drawn after particles converging. Obviously, it is more similar to the expected trajectory (see Figure 7.9) than the odometry's trajectory. The trajectory given by odometry has a about $pi/4$ slope because of the initial orientation error.

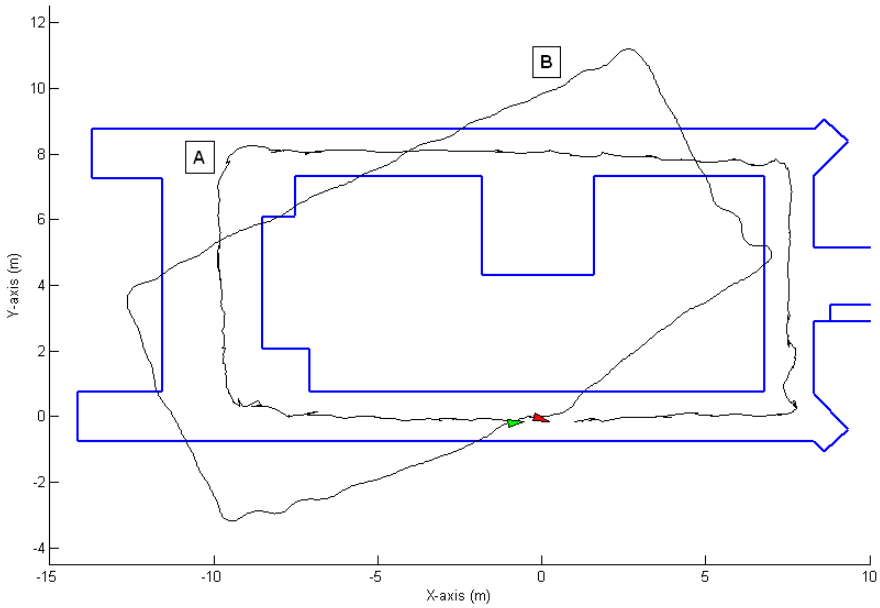


Figure 7.10: Global localization using SAMCL. Line A and line B denote the trajectories of SAMCL and odometry, respectively.

Table 7.1 shows average errors of the final poses given by the SAMCL algorithm and odometry. As shown in Table 7.1, Pioneer has a relatively precise odometry but the SAMCL algorithm provides more accurate localization results.

Table 7.1: Average errors of the final poses in global localization

	x	y	θ
Localization	$0.157m$	$0.092m$	$6.5°$
Odometry	$0.739m$	$0.215m$	$33.7°$

7.3.2 Global localization with artificial errors

The second experiment further tests the robustness of the SAMCL algorithm. In this experiment the Pioneer robot would localize itself with unfaithful odometry. In practice, these enormous errors of odometry are often caused by wheels sliding

on the smooth ground or by the robot passing the concave-convex road. In order to simulate coarse odometry, we added about 27% artificial errors to each wheel. In this experiment, ξ was given a low sensitive value as the first experiment.

The localization results are illustrated in Figure 7.11, line A and line B represent the trajectories of SAMCL and odometry, respectively. As we can see, odometry has totally lost because of gradually accumulated errors. On the contrary, SAMCL still gives good localization results.

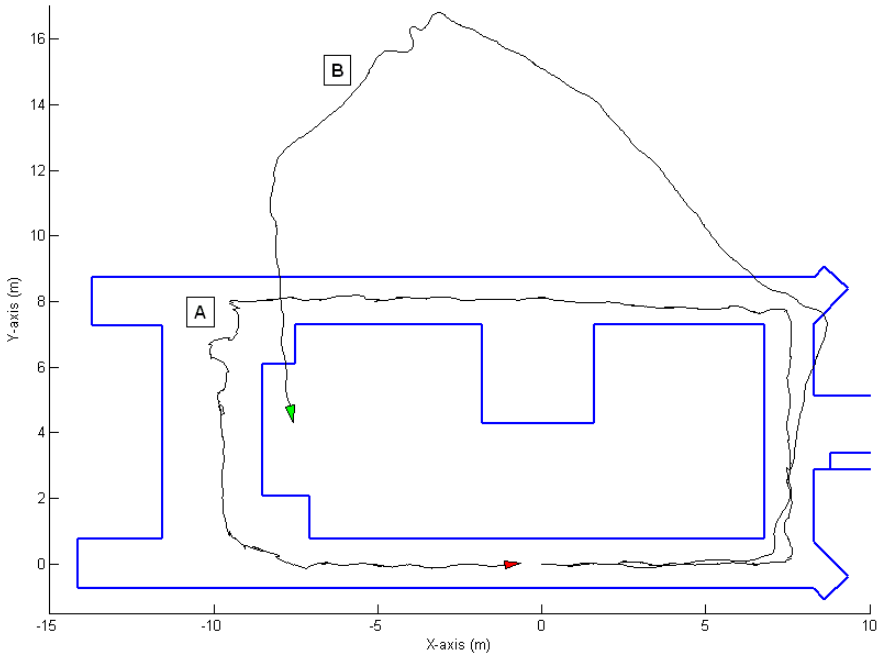


Figure 7.11: Global localization using SAMCL with artificial errors. Line A and line B present the trajectories of SAMCL and odometry, respectively.

Average errors of the final poses of the SAMCL algorithm and odometry are shown in Table 7.2. Odomery's errors are huge as a result of adding artificial errors, however the SAMCL algorithm still presents elegant localization results.

Table 7.2: Average errors of the final poses in global localization with artificial errors

	x	y	θ
Localization	$0.469m$	$0.031m$	$17.1°$
Odometry	$6.353m$	$7.301m$	$72.5°$

7.3.3 Kidnapping

The third experiment demonstrates the ability of the SAMCL algorithm to recover from kidnapping, which is the most difficult issue. We kidnapped the Pioneer robot

at the beginning of the trajectory after particles converging. Put differently, after the robot was well localized, we took it about $7m$ far away in its moving direction. Moreover, we added about 27% artificial errors to each wheel. In order to make the robot find kidnapping more quickly, the sensitive coefficient ξ was given a medium sensitive value.

Figure 7.12 illustrates the distribution of the self-adaptive sample set during the process of recovering from kidnapping. In the beginning, the robot is well localized as shown in Figure 7.12(a). Then the robot is kidnapped from position A to position B (position B is about $7m$ far away from position A in the robot's moving direction). Next, kidnapping brings on probabilities of particles reducing. When the maximum of probabilities is less than ξ, global samples are divided from the sample set and distributed in SER, as shown in Figure 7.12(b). The robot moves forward and perceives the environment. Because of the quasi-symmetry of environment, SAMCL generates three probable poses of the robot after resampling, depicted in Figure 7.12(c). The robot continues to move and perceive, SAMCL finally discards two probable poses and confirms the correct pose of robot, shown in Figure 7.12(d).

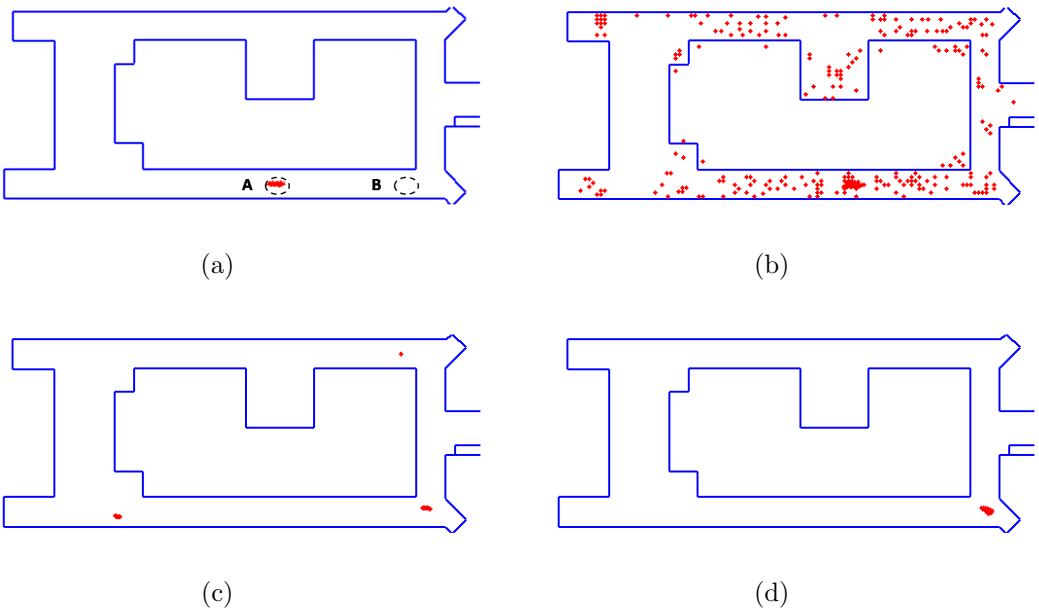
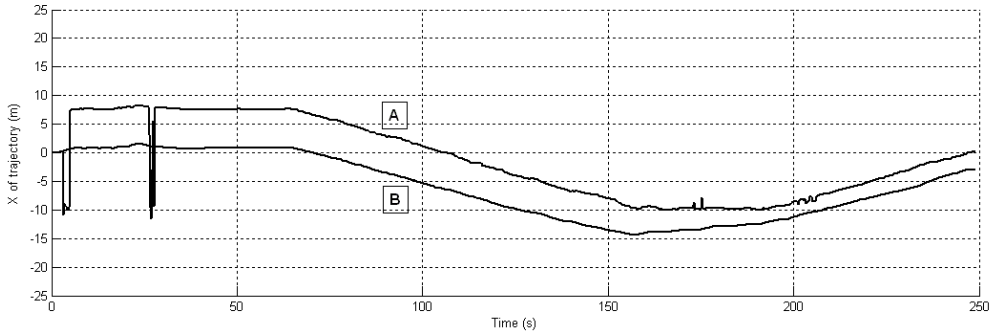


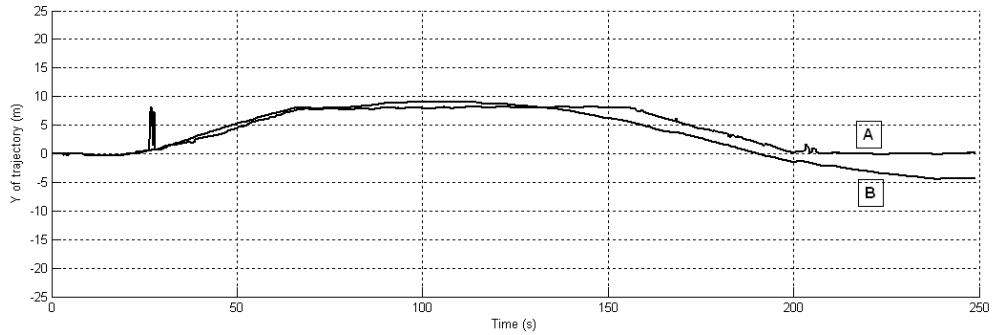
Figure 7.12: Distribution of the self-adaptive sample set during the process of recovering from kidnapping.

In this experiment, the final pose of the Pioneer robot is measured, that is $x = 0.79, y = 0.02$ in the Cartesian coordinate. For the convenience of analysis, trajectories given by the SAMCL algorithm (line A) and odometry (line B) are decomposed to X-axis and Y-axis. As shown in Figure 7.13, the final pose of localization is $x = 0.43, y = 0.09$, but the final pose of odometry is $x = -2.96, y = -4.35$. Obviously, the localization results are much better than odometry. From the figure, we can also find that the robot perceives kidnapping at $3^{rd}s$ and recovers

at $6^{th}s$. In the later process, it mistakes once, but it re-localizes in less than $2s$ interval. Average errors of the final poses of the SAMCL algorithm and odometry are shown in Table 7.3.



(a)



(b)

Figure 7.13: SAMCL for robot kidnapping. Trajectories are decomposed to (a) X-axis and (b) Y-axis. Line A and line B depict the trajectories of SAMCL and odometry, respectively.

Table 7.3: Average errors of the final poses in kidnapping

	x	y	θ
Localization	$0.605m$	$0.076m$	$13.2°$
Odometry	$5.6728m$	$5.017m$	$45.3°$

7.3.4 The success rate of recovering from kidnapping

In Chapter 5, we presented that sampling in SER is more efficient and more effective than sampling randomly from the theoretical view. Here, this conclusion was demonstrated by the simulation based on the experiment. Figure 7.14 shows the success rate of recovering from kidnapping as a function of the number of particles. The success rate is defined as:

$$success\ rate = \frac{the\ number\ of\ successful\ recoveries}{the\ total\ number\ of\ tests} \tag{7.1}$$

The success rate increases with the number of particles, both for sampling in SER and for sampling randomly. However, with the same size particle set, the success rate of sampling in SER is much higher than sampling randomly. For example, when using 300 particles, the success rate of sampling in SER may achieve to 33%, while this rate of sampling randomly is only 11%. To reach the same success rate, sampling randomly has to use 900 particles, while using 900 particles, the success rate of sampling in SER has achieved to 91%.

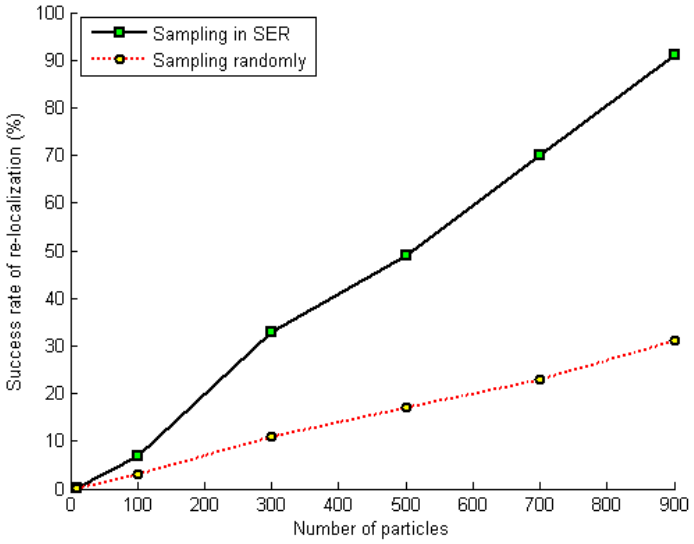


Figure 7.14: The success rate of recovering from kidnapping as a function of the number of particles.

7.4 SUMMARY

In this chapter, we designed three experiments to verify the validity of the SAMCL algorithm. The experimental robot is only equipped with imprecise ultrasonic sensors.

- The first one proved the ability of the SAMCL algorithm in global localization. The second one achieved global localization by adding artificial errors to wheel encoder reading. The third one confirmed the ability of recovering from kidnapping.

- By comparing sampling in SER with sampling randomly, we find that the success rate of recovering from kidnapping by sampling in SER is much higher than by sampling randomly when using the same size of particle set.

CHAPTER 8

CONCLUSION

8.1 SUMMARY OF THESIS

Localization has been considered as the fundamental problem to achieve other tasks. This thesis presented the self-adaptive Markov localization approach to solve single-robot and multi-robot localization problems. In order to find out an effective and efficient solution, we started with studies and comparisons of three regular Markov localization methods, including EKF localization, grid localization and MCL.

As discussed thus far, the localization problem can be classified into position tracking, global localization and the kidnapped robot problem according to localization difficulties. Most localization approaches focus on solving one of three sub-problems. In order to find out a general solution for all three, we proposed an improved Monte Carlo localization algorithm with self-adaptive samples (the SAMCL algorithm). This algorithm inherits all the advantages of MCL, moreover it improves in several aspects. SAMCL employs an off-line pre-caching technique to solve the expensive on-line computational cost problem of regular MCL. Further, we define Similar Energy Region (SER), which provides a priori information of the robot's pose. Hence sampling in SER is more efficient than sampling randomly in the entire environment. Because of using self-adaptive samples, SAMCL can deal with the kidnapped robot problem as well as position tracking and global localization.

Multi-robot localization is more robust and efficient than single-robot localization. In this thesis, we devised the Position Mapping (PM) algorithm to solve the multi-robot localization problem. The PM algorithm can be integrated into the SAMCL algorithm as its extension. This algorithm achieves multi-robot localization through completing three tasks: detection, data exchange and location update. Since it only synchronizes one location and one belief when the robot detected others, it reduces furthest the communication delay and the computational complexity. Moreover, the PM algorithm allows one robot to cooperate with multiple robots rather than one robot at the same time.

Experimental results, carried out in real and simulated environments, demonstrated that the SAMCL algorithm as well as its extended algorithm (the PM

algorithm) is capable for both single-robot localization and multi-robot localization. Simulations tested the performance of the SAMCL algorithm in position tracking, global localization and the kidnapped robot problem, respectively. The extended SAMCL algorithm was implemented on two individual computers that can communicate with wireless network (WiFi) to evaluate its ability in cooperative localization. Experiments were carried out on the Pioneer 3-DX mobile robot in a real office environment. Three experiments were designed to verify the validity of the SAMCL algorithm. The first one proved the ability of global localization. The second one achieved global localization by adding artificial errors to wheel encoder reading. The third one confirmed the ability of recovering from kidnapping. Moreover, extensive comparisons between the SAMCL algorithm and other algorithms were also given in this thesis.

8.2 FUTURE WORK

This dissertation raises several interesting topics, which can be summarized into two main extensions: one focus on improving the performances of the SAMCL algorithm both in theoretical and practical ways; the other aims at solving the Simultaneous Localization And Mapping (SLAM) problem (including multi-robot SLAM) on the basis of existing theories.

8.2.1 The SAMCL algorithm

Although the SAMCL algorithm has the ability to solve single-robot and multi-robot localization efficiently, it would be improved in two aspects.

Firstly, the SAMCL algorithm can only address localization problems in static environments due to the off-line pre-caching technique. Although the SAMCL algorithm can treat unmodeled obstacles as sensors noise, since it employs the mixture perception model (see Section 2.3.4.1), it is incapable when the environment has structural changes or robots work in the environment populated by people. We should develop an on-line map update technique that enables the robot to update and modify the pre-caching map on line when the robot feels changes of the environment.

Secondly, the SAMCL algorithm uses the fixed size grid to decompose the map. That is inefficient in the sparse environment. Decomposition techniques with adaptive size grid have been addressed in the recent literature. Thus, we should develop and apply a suitable decomposition technique to the SAMCL algorithm.

8.2.2 SLAM

Another research direction of future works is to solve the SLAM problem. The solutions may be based on the SAMCL algorithm.

We have studied EKF SLAM and FastSLAM [Thrun05]. The EKF SLAM algorithm applies the EKF to the online SLAM problem. It represents both the robot pose and the feature location by the EKF. The FastSLAM algorithm employs

particle filters for estimating the robot path and EKFs for estimating map feature locations. Differently to EKF SLAM, it uses a separate low-dimensional EKF for each individual feature [Thrun05].

In practice, both two SLAM algorithms have been applied with some success. However, they do not take global localization failures into account. As in localization problems, SLAM algorithms may also fail to localize the robot. How to recover from localization failures is an important issue for the SLAM problem, too. As discussed thus far, the SAMCL algorithm has the ability to recover from localization failures. Thus, an immediate idea is to apply the SAMCL algorithm to the SLAM problem. Unfortunately, a straightforward implementation of the SAMCL algorithm for the SLAM problem is inapplicable. The first difficulty arises from the off-line pre-caching technique that needs a pre-known map; however there is no pre-known map in the SLAM problem. Since the pre-caching technique decomposes the robot space into grid, we intend to develop an on-line occupancy grid mapping technique to generate the map and the energy map. More concrete implementations will be discussed in the future.

APPENDIX A

BASIC CONCEPTS IN PROBABILITY THEORY

A probability of an event A is represented by a real number in the range from 0 to 1 and denoted as $P(A)$.

For a discrete random variable X, the function $p(X = x)$ mapping a point in the sample space to the probability value is called a Probability Mass Function abbreviated as PMF. The sum of $p(x)$ over all values x in the entire sample space is equal to 1.

$$\sum_x p(X = x) = 1 \tag{A.1}$$

For a continuous random variable X, the function $F(x) = P(X \leq x)$ is called Cumulative Distribution Function abbreviated as CDF. Its derivative is called Probability Density Function abbreviated as PDF.

$$p(x) = \frac{d}{dx}F(x) \tag{A.2}$$

Just like discrete probability distribution, the PDF always integrates to 1 in the entire sample space.

$$\int p(x)dx = 1 \tag{A.3}$$

A frequently used PDF is that of the normal distribution (or Gaussian distribution). Its one-dimensional form is defined as

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{A.4}$$

where μ is the mean, σ^2 is the variance. The square root of the variance is called standard deviation.

The PDF of the multivariate normal distribution is described as

$$p(x) = \det(2\pi\Sigma)^{-\frac{1}{2}}\exp\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\} \tag{A.5}$$

where μ is the mean vector. Σ is a positive semi-definite and symmetric matrix called the covariance matrix.

The expected value (or mathematical expectation) of a random variable is the integral of the random variable with respect to its probability measure. If X is a discrete random variable with probability mass function $p(x)$, then the expected value is

$$\mathrm{E}(X) = \sum_x xp(x) \tag{A.6}$$

If the probability distribution of a continuous random variable X admits a probability density function $p(x)$, then the expected value can be computed as

$$\mathrm{E}(X) = \int xp(x)dx \tag{A.7}$$

If a random variable X has expected value $E(X) = \mu$, then the variance $\mathrm{Var}(X)$ of X is given by

$$\mathrm{Var}(X) = \mathrm{E}\left[(X-\mu)^2\right] \tag{A.8}$$

This definition encompasses random variables that are discrete and continuous.

The standard deviation of X is the square root of the variance.

$$\sigma = \sqrt{\mathrm{Var}(X)} = \sqrt{E\left[(X-\mu)^2\right]} \tag{A.9}$$

The variance is a special case of the covariance when the two variables are identical. The covariance between two real-valued random variables X and Y, with expected values $\mathrm{E}(X) = \mu$ and $\mathrm{E}(Y) = \nu$, is defined as

$$\mathrm{Cov}(X,Y) = \mathrm{E}\left((X-\mu)(Y-\nu)\right) \tag{A.10}$$

The joint probability is the probability of both events together. For a pair of continuous random variables, their joint cumulative distribution function is defined as

$$F(x,y) = P(X \leq x \text{ and } Y \leq y) \tag{A.11}$$

For discrete random variables X and Y, the joint probability mass function is written as

$$p(x,y) = p(X = x \text{ and } Y = y) \tag{A.12}$$

If X and Y are independent, we have

$$p(x,y) = p(x)p(y) \tag{A.13}$$

The conditional probability is the probability of some events X, given the occurrence of some other events Y. Such a probability is written as

$$p(x\,|y\,) = p(X = x\,|Y = y\,) \tag{A.14}$$

If $p(y) > 0$, the conditional probability is defined as

$$p(x\,|y\,) = \frac{p(x, y)}{p(y)} \tag{A.15}$$

If X and Y are independent, we have

$$p(x\,|y\,) = \frac{p(x)p(y)}{p(y)} = p(x) \tag{A.16}$$

The law of total probability is defined as "the prior probability of X is equal to the prior expected value of the posterior probability of X". That is, for any random variable Y,

$$p(x) = E\,[p(X\,|Y\,)] \tag{A.17}$$

In the discrete case, that is

$$p(x) = \sum_{y} p(x\,|y\,)p(y) \tag{A.18}$$

In the continuous case, that is

$$p(x) = \int p(x\,|y\,)p(y)dy \tag{A.19}$$

Another important theorem is Bayes rule, which shows how one conditional probability $p(x\,|y\,)$ depends on its inverse $p(y\,|x\,)$. Bayes rule relates the conditional and marginal probabilities of events X and Y, where Y has a non-vanishing probability $p(y) > 0$.

$$p(x\,|y\,) = \frac{p(y\,|x\,)p(x)}{p(y)} \tag{A.20}$$

where

- $p(x)$ is the prior probability or marginal probability of X. It is "prior" in the sense that it does not take into account any information about Y.
- $p(x\,|y\,)$ is the conditional probability of X given Y. It is also called the posterior probability because it is derived from or depends upon the specified value of Y.
- $p(y\,|x\,)$ is the conditional probability of Y given X.
- $p(y)$ is the prior or marginal probability of Y.

Intuitively, Bayes' theorem in this form describes the way in which one's beliefs about observing X are updated by having observed Y.

Since the denominator of Bayes rule $p(y)$ does not depend on X, the factor $p(y)^{-1}$ in the Equation A.20 will be the same for any value X in the posterior $p(x\,|y\,)$. Thus, the factor $p(y)^{-1}$ is often written as a normalizer in Bayes rule variable, and generically denoted η.

$$p(x\,|y\,) = \eta p(y\,|x\,)p(x) \tag{A.21}$$

APPENDIX B

RESAMPLING APPROACH

B.1 APPROACH DESCRIPTION

Given a probability density $p(x)$, the resampling process is to generate N samples that are distributed with the desired probability density $p(x)$. The transformation method consists of three steps [Hom].

1. Compute $y = P(x) = \int_a^x p(x')dx'$.
2. Sample y from an equi-distribution in the interval $(0, 1)$.
3. Compute $x = P^{-1}(y)$.

Then the variable x has the desired probability density $p(x)$.

B.2 SIMULATION RESULT

This simulation shows how to generate 10000 samples according to the Gaussian distribution with mean 0 and standard deviation 1 as follows:

$$p(x) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{x^2}{2}\right) \tag{B.1}$$

Figure B.1 illustrates the process of resampling. Figure B.1(a) shows the initial discrete Gaussian distribution $p(x)$ with 10000 samples. Figure B.1(b) plots the curve of the cumulative sum $y = P(x)$. Figure B.1(c) shows the inverse function of the cumulative sum $x = P^{-1}(y)$. Here, we sample the variable y in the interval $(0, 1)$ and compute the variable x. Two different lines denote the error between the sampling value and the actual value of the first sampling. Figure B.1(d) illustrates the new probability density curve of samples after resampling, which is still a Gaussian with mean 0 and standard deviation 1. Samples are distributed with the Gaussian, which is shown in Figure B.1(E).

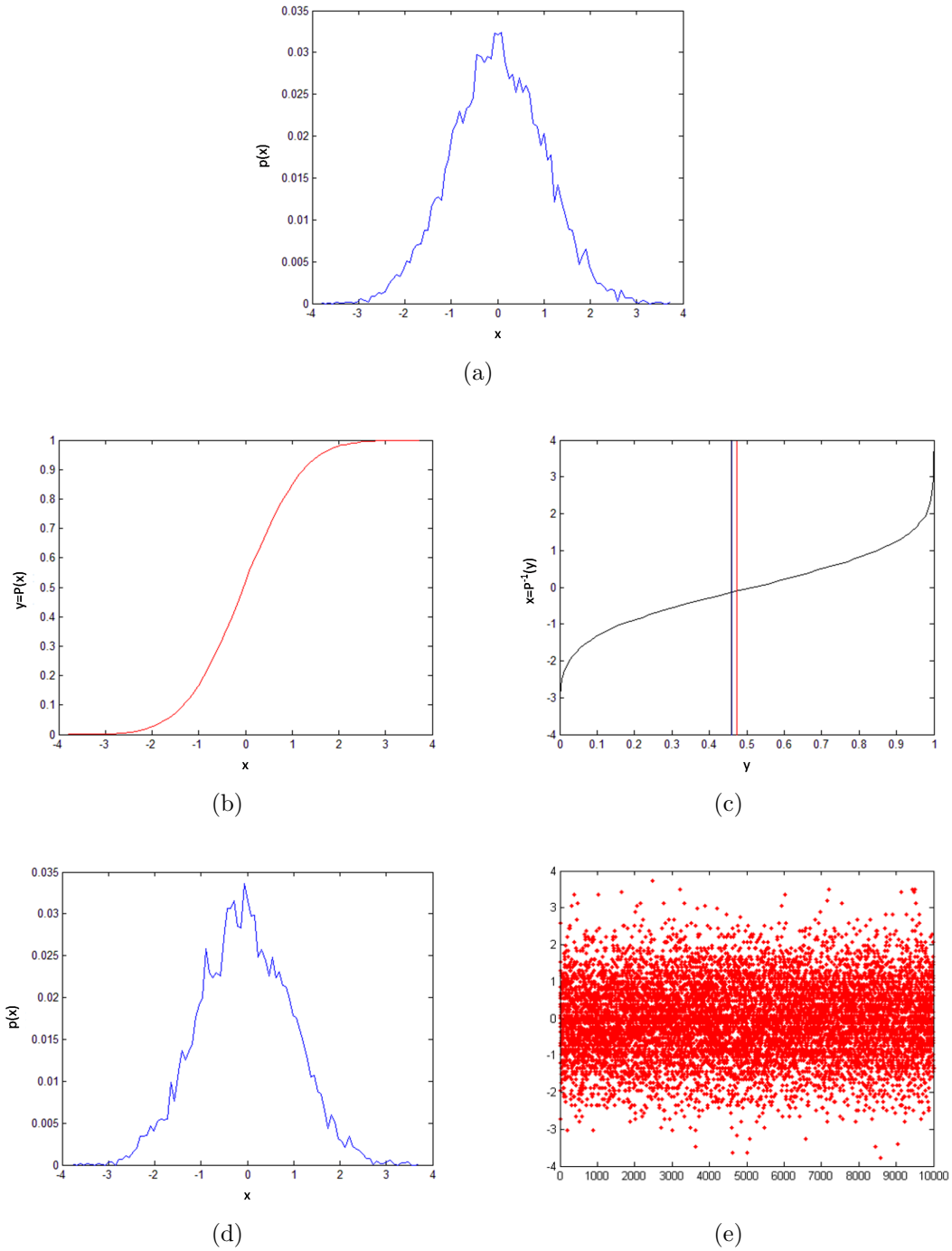


Figure B.1: Illustration of resampling.

APPENDIX C

RÉSUMÉ

INTRODUCTION

La navigation autonome d'un robot mobile se résume à la résolution de 3 problèmes fondamentaux, réponses aux 3 questions suivantes que "se pose le robot":

- **Où suis-je?** Répondre à cette question, c'est résoudre le problème de la localisation, c'est-à-dire déterminer la position et l'orientation du robot tant donnés une carte et un ensemble de mesures fournies par les capteurs.
- **Où vais-je?**
- **Comment y aller?**

La localisation est donc un prérequis fondamental pour qu'un robot mobile puisse de déplacer de manière autonome. Cette thèse se focalise sur la recherche de méthodes robustes de localisation.

Selon le type de connaissances dont dispose le robot, le problème de la localisation peut se diviser en 3 sous-problèmes:

- Le suivi de trajectoire, qui suppose que le robot connaît sa pose initiale et qu'il cherche à garder trace de son mouvement afin d'estimer sa pose malgré les erreurs de mesures.
- La localisation globale, qui suppose que le robot ne connaît pas sa pose initiale.
- Le robot kidnappé, le plus difficile des trois, qui suppose que le robot est bien localisé mais soudainement transporté. Ce problème a 2 intérêts: il permet la prise en compte d'erreurs importantes lors du processus de localisation globale et ensuite il sert de benchmark comparatif entre les approches de localisation globale trouves dans la littérature.

Une autre classification possible est basée sur le nombre de robots utilisés: on parlera alors de localisation mono-robot ou de localisation multi-robots.

Les différentes approches de localisation peuvent être regroupées en 2 grandes familles: la localisation déterministe et la localisation probabiliste. Cette dernière famille est très intéressante pour 3 raisons:

- L'environnement du robot est toujours imprédictible.
- Le hardware du robot est soumis au bruit.
- Le software du robot est approximatif.

Parmi les approches probabilistes, l'approche Markovienne, basée sur le filtre de Bayes est la plus populaire. Dans ce contexte, la pose du robot est remplacée par la probabilité conditionnelle:

$$bel(s_t) = p\left(s_t \,|z_{0:t}, u_{1:t}, m\right) \tag{C.1}$$

où s_t est la pose du robot à l'instant t. Cette fonction de "croyance" $bel(s_t)$ représente la densité de probabilité de la pose s_t, conditionnée par l'ensemble des mesures proprioceptives $z_{0:t}$ du temps $\tau = 0$ au temps $\tau = t$, par l'ensemble des données de contrôle $u_{1:t}$ et par la carte m.

La localisation Markovienne adresse le problème de suivi de trajectoires, celui de la localisation globale et celui du robot kidnappé. Trois grandes sous-familles se distinguent: le filtre de Kalman étendu (EKF), les méthodes de grille et la localisation de Monte Carlo par filtre particullaire (MCL).

Notre objectif est de trouver une solution générale pour les problèmes de suivi de trajectoires, de la localisation globale et du robot kidnappé.

- Nous avons développé un algorithme afin de résoudre à la fois le problème de suivi de trajectoires, celui de la localisation globale et celui du robot kidnappé un simulateur capable de tester ces méthodes.
- Nous avons étendu cet algorithme pour la localisation multi-robots.
- Nous avons testé notre approche sur un robot réel Pioneer 3-DX équipé de seulement 16 capteurs ultrasonores.

LES MODÈLES

Pour cela, il nous faut décrire les modèles d'action et de perception pour un robot de type unicycle évoluant dans un environnement plan.

La pose d'un robot mobile (Figure C.1) peut être représentée par:

$$s_G = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \tag{C.2}$$

Le mouvement du robot est représenté par:

$$s_t = f(u_t, s_{t-1}) \tag{C.3}$$

où s_{t-1} et s_t sont les poses à $t-1$ et t, et u_t est le vecteur de contrôle.

Par exemple, u_t peut être représenté par les vitesses de translation et de rotation:

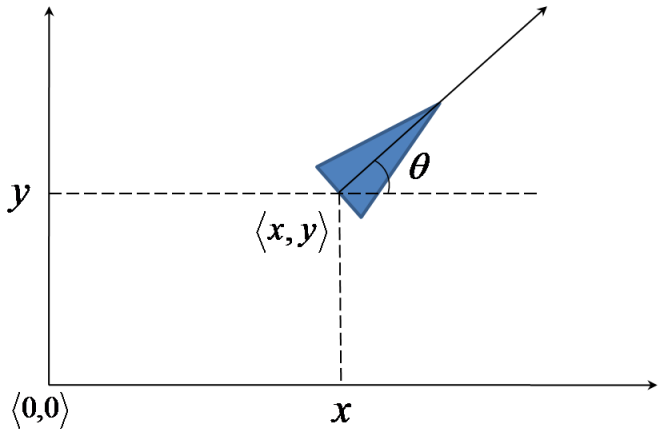


Figure C.1: La pose d'un robot.

$$u_t = \begin{pmatrix} v_t \\ \omega_t \end{pmatrix} \tag{C.4}$$

Le modèle probabiliste correspondant s'écrit comme la probabilité conditionnelle:

$$p(s_t \,|u_t, s_{t-1}) \tag{C.5}$$

Le modèle de perception quant à lui s'écrit:

$$p(z_t \,|s_t, m) \tag{C.6}$$

où

$$z_t = \left\{z_t^1, \cdots, z_t^I\right\} \tag{C.7}$$

est l'ensemble des mesures. Avec une distribution gaussienne on aurait:

$$p_k(z_t^i \,|s_t, m) = \begin{cases} \eta \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(z_t^i - d_{obj})^2}{2\sigma_k^2}} & \text{if } 0 \leq z_t^i \leq z_{max} \\ 0 & \text{sinon} \end{cases} \tag{C.8}$$

Enfin, la carte m (connue au départ de l'expérience) peut être vue comme une liste d'amers dans l'environnement:

$$m = \{m_1, m_2, \cdots, m_I\} \tag{C.9}$$

ou encore comme une grille d'occupation (Figure C.2); ou encore comme une liste d'objets géométriques (murs par exemple).

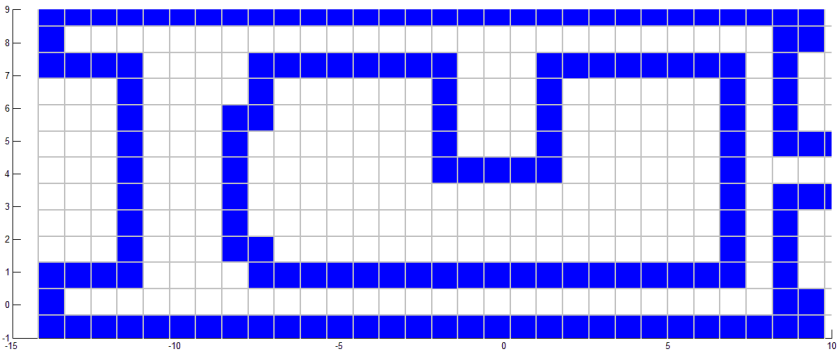


Figure C.2: Une carte grille d'occupation.

L'APPROCHE MARKOVIENNE

L'approche Markovienne est basée sur le filtre de Bayes dont l'algorithme récursif est donné par l'Algorithme C.1:

Algorithm C.1: Algorithme Markovien basée sur le filtre de Bayes

1: **Input:** $bel(s_{t-1}), u_t, z_t, m$
2: **for** all s_t **do**
3: $\quad \overline{bel}(s_t) = \int p\left(s_t \,|s_{t-1}, u_t, m\right) bel(s_{t-1}) ds_{t-1}$ *% prediction*
4: $\quad bel(s_t) = \eta p\left(z_t \,|s_t, m\right) \overline{bel}(s_t)$ *% correction*
5: **end for**
6: **Output:** $bel(s_t)$

Cette croyance peut revêtir 3 formes principales:

- Le filtre de Kalman récursif:

$$bel(s_t) = \det(2\pi\Sigma_t)^{-\frac{1}{2}} \exp\{-\frac{1}{2}(s_t - \mu_t)^T \Sigma_t^{-1}(s_t - \mu_t)\} \tag{C.10}$$

- L'approche par grille de probabilités:

$$bel(s_t) \sim \{p_{k,t}\}_{k=1,\cdots,K} \tag{C.11}$$

- Le filtre particulaire:

$$bel(s_t) \sim \left\{\left\langle s_t^{[n]}, \omega_t^{[n]} \right\rangle\right\}_{n=1,\cdots,N} \tag{C.12}$$

où $\omega_t^{[n]}$ est un facteur d'importance.

Notre méthode est basée sur le filtre particulaire donné par l'algorithme suivant (Algorithm C.2):

Algorithm C.2: Filtre MCL

Input: S_{t-1}, u_t, z_t, m
$\bar{S}_t = S_t = \emptyset$
for $n = 1$ to N **do**
  générer une particule $s_t^{[n]} \sim p\left(s_t \left| s_{t-1}^{[n]}, u_t, m\right.\right)$ *% prediction*
  calculer un facteur d'importance $\omega_t^{[n]} = p\left(z_t \left| s_t^{[n]}, m\right.\right)$ *% correction*
  ajouter $\left\langle s_t^{[n]}, \omega_t^{[n]} \right\rangle$ à $\bar{S}_t$
end for
normaliser ω_t
for $n = 1$ to N **do**
  choisir au hasard $s_t^{[n]}$ selon $\omega_t^{[n]}$ *% ré-échantillonnage*
  ajouter $s_t^{[n]}$ to S_t
end for
Output: S_t

Plusieurs simulations ont permis de comparer les différentes approches probabilistes. Par exemple, on peut constater que le filtre particulaire permet de localiser le robot (trajectoire rouge qui se superpose à la trajectoire noire) alors que l'odométrie (trajectoire verte) ne localise pas le robot (Figure C.3).

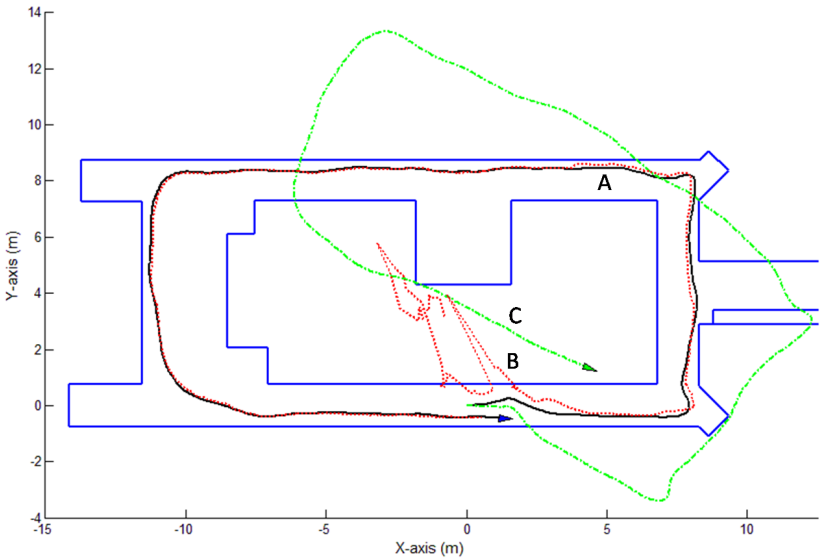


Figure C.3: La localisation globale utilisant MCL dans un couloir quasi-symétrique.

La figure suivante (Figure C.4) compare les erreurs odométriques (en vert) avec les erreurs du filtre de Monte Carlo dans le cas de la localisation globale.

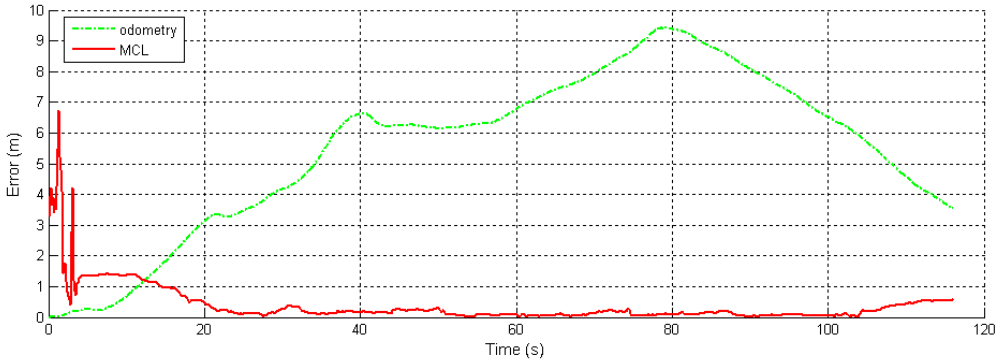


Figure C.4: Les erreurs de la localisation globale utilisant MCL dans un couloir quasi-symétrique.

L'ALGORITHME SAMCL

L'algorithme SAMCL (Algorithm C.3) que nous avons développé est implémenté en 3 étapes (Figure C.5):

- **Pre-Mise en cache de la carte.** La carte m est l'entrée de cet algorithme qui s'effectue hors-ligne. Ses sorties sont une grille tridimensionnelle G_{3D} qui stocke les données issus des capteurs ultrasonores, et une grille bidimensionnelle G_E qui stocke l'énergie-capteurs. Chaque cellule de G_{3D} peut-être vue comme un robot virtuel placé dans un lieu particulier de l'environnement et qui effectue une série de mesures sur cet environnement. Chaque cellule k de G_E représente une position (sans orientation) du même robot virtuel. Son contenu est défini par:

$$\tilde{E}(k) = \sum_{i=1}^{I} \tilde{a}_i^{[k]} \tag{C.13}$$

avec

$$\tilde{a}_i^{[k]} = 1 - \tilde{d}_i^{[k]}/d_{\max} \tag{C.14}$$

qui est l'énergie du capteur i qui mesure une distance $d_i^{[k]}$.

- **Calcul des zones d' énergie similaire (SER).** L'entrée est la grille G_E obtenue hors-ligne. Sa sortie est l'ensemble des positions potentielles du robot définie par:

$$\left| E - \tilde{E}(k) \right| < \delta \tag{C.15}$$

Cette étape est calculée en-ligne.

- **Localisation.** L'entrée de la dernière partie est l'ensemble S_{t-1} des particules à l'instant $t-1$, le vecteur de contrôlel u_t, les mesures z_t, la grille G_{3D} et SER. Sa sortie est S_t, ensemble des particules à l'instant t. Cette étape est calculée en-ligne.

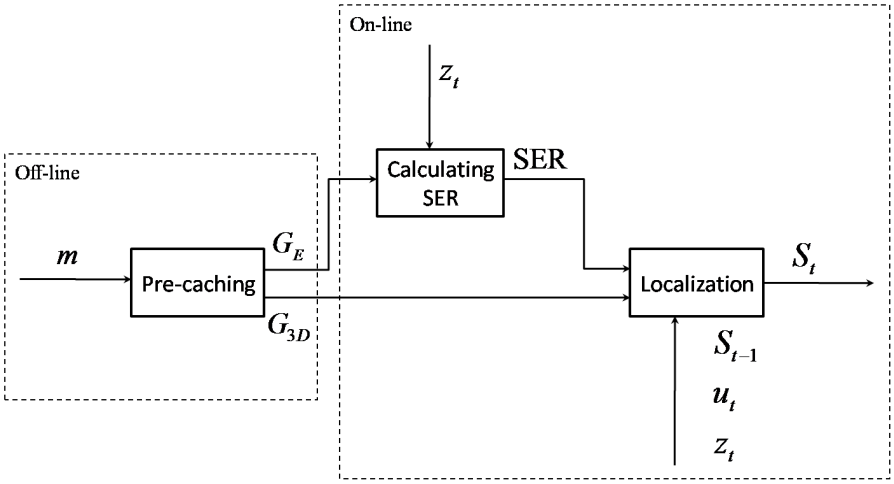


Figure C.5: Schèma SAMCL.

La Figure C.6 et la Figure C.7 sont une illustration des erreurs mesurées et comparées dans le cas de la localisation odométrique et de la localisation par SAMCL en suivi de trajectoire.

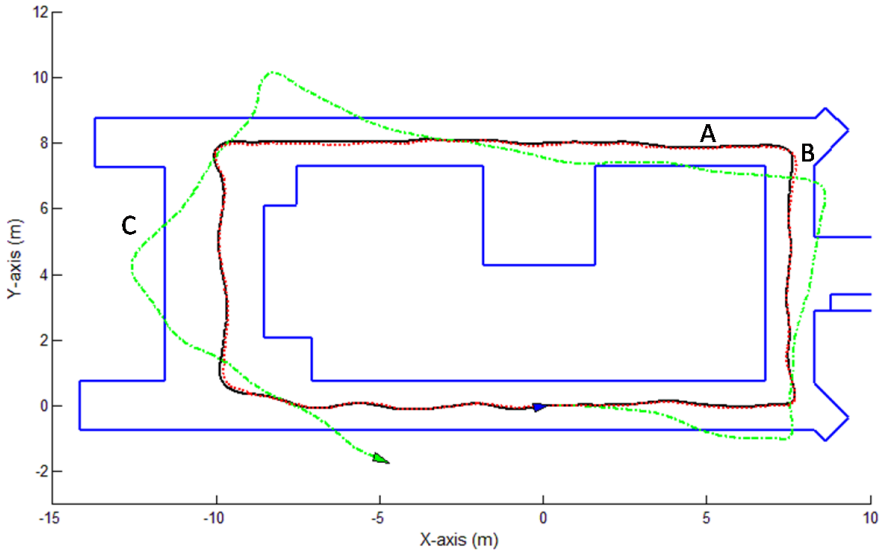


Figure C.6: Le suivi de trajectoire utilisant SAMCL dans un couloir quasi-symétrique.

La Figure C.8 et la Figure C.9 sont une illustration des erreurs mesurées et comparées dans le cas de la localisation odométrique et de la localisation par SAMCL en localisation globale.

Algorithm C.3: Algorithme SAMCL

1: **Input:** S_{t-1}, u_t, d_t, G_{3D}, SER

Sampling total particles

1: **for** $n = 1$ to N_T **do**
2: generate a particle $s_t^{[n]} \sim p\left(s_t \left| s_{t-1}^{[n]}, u_t \right.\right)$ % motion model
3: calculate importance factor $\omega_t^{[n]} = p\left(z_t \left| s_t^{[n]}, G_{3D} \right.\right)$ % perception model
4: **end for**

Determining the size of global sample set and local sample set

1: **if** $\omega_t^{max} < \xi$ **then**
2: $N_L = \alpha \cdot N_T$
3: **else**
4: $N_L = N_T$
5: **end if**
6: $N_G = N_T - N_L$

Resampling local samples

1: normalize ω_t
2: **for** $n = 1$ to N_L **do**
3: draw $s_t^{[n],L}$ with distribution $\omega_t^{[n]}$
4: add $s_t^{[n],L}$ to S_t^L
5: **end for**

Drawing global samples

1: **for** $n = 1$ to N_G **do**
2: draw $s_t^{[n],G}$ with the uniform distribution in SER
3: add $s_t^{[n],G}$ to S_t^G
4: **end for**

Combining two particle sets

1: $S_t = S_t^L \cup S_t^G$
2: **Output:** S_t

La Figure C.10 est une illustration des erreurs mesurées et comparées dans le cas de la localisation odométrique et de la localisation par SAMCL dans le cas du kidnapping.

Le tableau suivant (Table C.1) résume les différentes propriétés des approches classiques de localisation Markovienne et l'approche SAMCL.

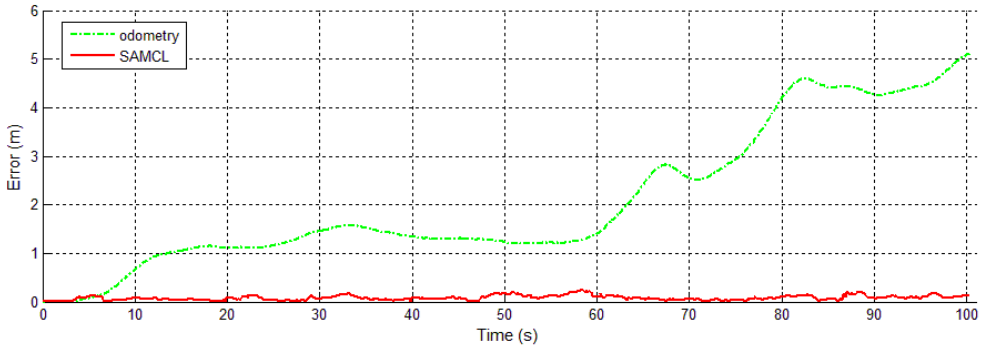


Figure C.7: Les erreurs du suivi de trajectoire utilisant SAMCL dans un couloir quasi-symétrique.

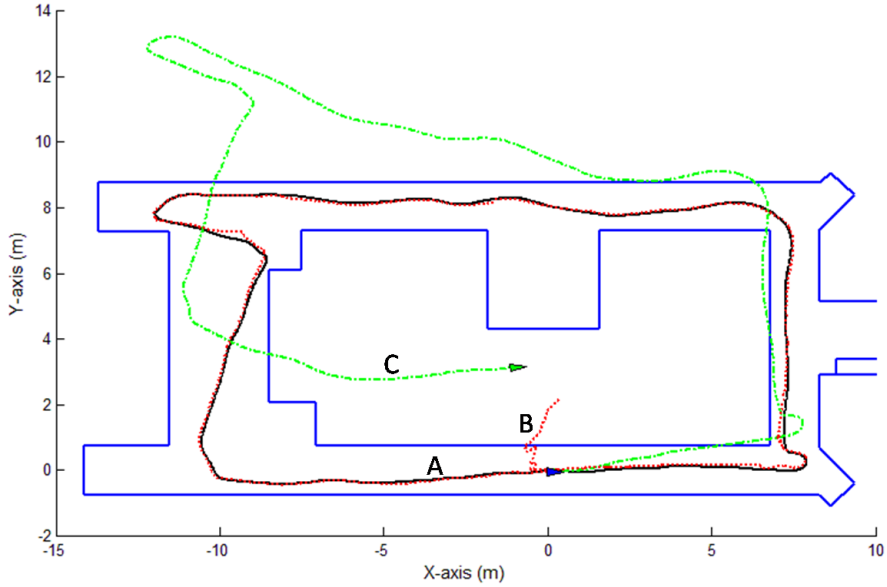


Figure C.8: La localisation globale utilisant SAMCL dans un couloir quasi-symétrique.

LOCALISATION MULTI-ROBOT

L'algorithme SAMCL peut être étendu au cas des systèmes multi-robots. La localisation multi-robots procède en 3 étapes:

- **La détection.** Les données sont cette fois divisées en 2 sous-ensembles: celles qui concernent la perception de l'environnement et celles qui concernent la détection des autres robots.

$$Z_t = \left\{ Z_t^e, Z_t^d \right\} \tag{C.16}$$

avec

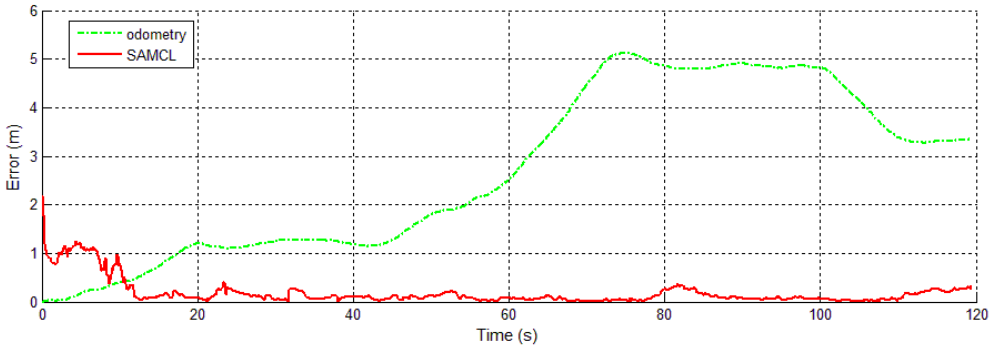
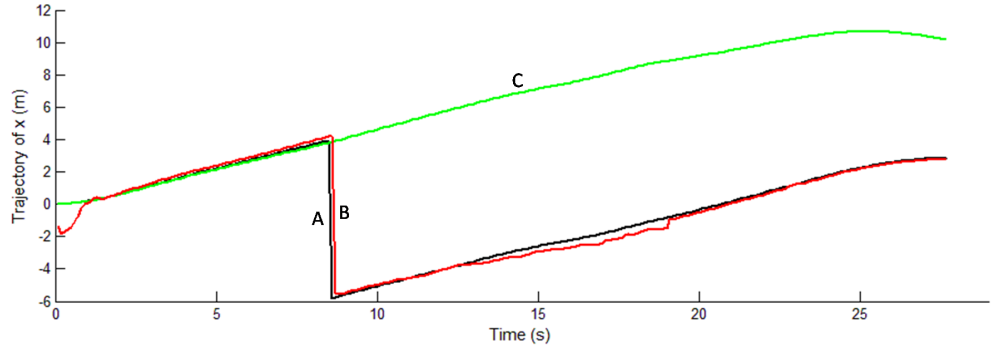
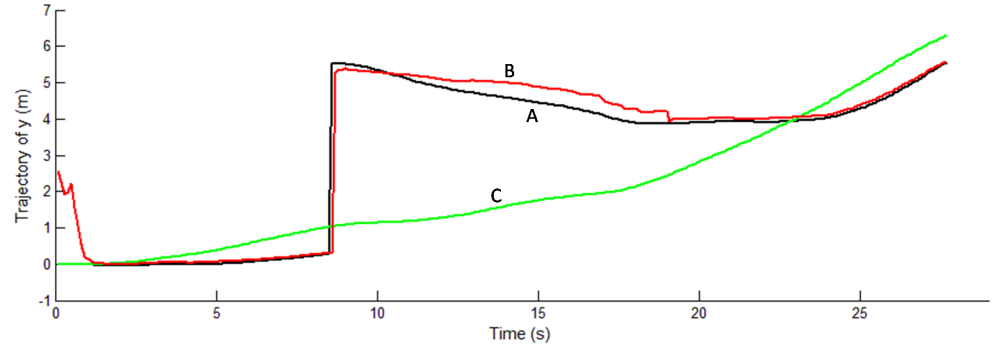


Figure C.9: Les erreurs de la localisation globale utilisant SAMCL dans un couloir quasi-symétrique.



(a)



(b)

Figure C.10: Les trajectoires sont décomposés selon (a) l'axe X et (b) l'axe Y.

$$Z_t^e = \left\{ z_t^e, z_{t-1}^e, \cdots, z_0^e \right\} \tag{C.17}$$

et

$$Z_t^d = \left\{ z_t^{R_{i,1}}, z_t^{R_{i,2}}, \cdots, z_t^{R_{i,K}} \right\} \tag{C.18}$$

Table C.1: Comparaison de SAMCL, EKF, l'approche par grille et MCL.

	EKF	Grid localization	MCL	SAMCL
Posterior representation	Gaussian (μ_t, Σ_t)	histogram	particles	particles
Position Tracking	yes	yes	yes	yes
Global Localization	no	yes	yes	yes
Kidnapping	no	yes	no	yes
Efficiency	fast	slow	medium	fast

où $z_t^{R_{i,j}}$ représente les relations géométriques obtenues par le robot R_i quand il détecte le robot R_j.

- **Echange de données.** Si un robot détecte K autres robots, il envoie des données à ces K autres robots, dont sa propre position estimée et la confiance qu'il met dans cette estimation (Figure C.11).

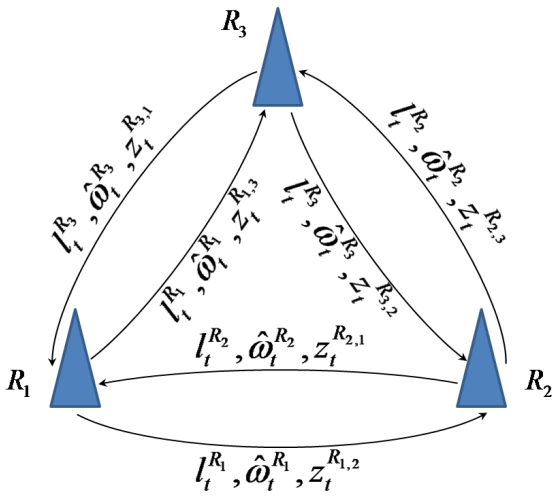


Figure C.11: L'échange de données entre les trois robots.

- **Mise à jour des positions.** Chaque robot calcule alors K positions.

$$l_t^p = f(l_t^e) \tag{C.19}$$

Ces positions possibles remplace les positions des particules ayant le poids le plus faible. Les poids de ces particules sont remplacées par la confiance qu'à le robot dans sa propre position.

L'algorithme global est donné par l'Algorithme C.4:

Algorithm C.4: L'algorithm SAMCL étendu pour la localization multi-robot

1: **Input:** S_{t-1}, u_t, d_t, G_{3D}, SER

Sampling total particles

1: **for** $n = 1$ to N_T **do**
2: generate a particle $s_t^{[n]} \sim p\left(s_t \left| s_{t-1}^{[n]}, u_t \right.\right)$ % motion model
3: calculate importance factor $\omega_t^{[n]} = p\left(z_t \left| s_t^{[n]}, G_{3D} \right.\right)$ % perception model
4: **end for**

The PM algorithm

1: **if** the robot detects K other robots $\{R_1, \cdots, R_K\}$ **then**
2: **for** $k = 1$ to K **do**
3: $\omega_t^{min} = min(\omega_t)$
4: $\begin{pmatrix} x_t^{min} \\ y_t^{min} \end{pmatrix} = \begin{pmatrix} x_t^{R_k} \\ y_t^{R_k} \end{pmatrix} + d \cdot \begin{pmatrix} cos(\vartheta) \\ sin(\vartheta) \end{pmatrix}$
5: replace ω_t^{min} by $\hat{\omega}_t^{R_k}$
6: **end for**
7: **end if**

Determining the size of global sample set and local sample set

1: **if** $\omega_t^{max} < \xi$ **then**
2: $N_L = \alpha \cdot N_T$
3: **else**
4: $N_L = N_T$
5: **end if**
6: $N_G = N_T - N_L$

Resampling local samples

1: normalize ω_t
2: **for** $n = 1$ to N_L **do**
3: draw $s_t^{[n],L}$ with distribution $\omega_t^{[n]}$
4: add $s_t^{[n],L}$ to S_t^L
5: **end for**

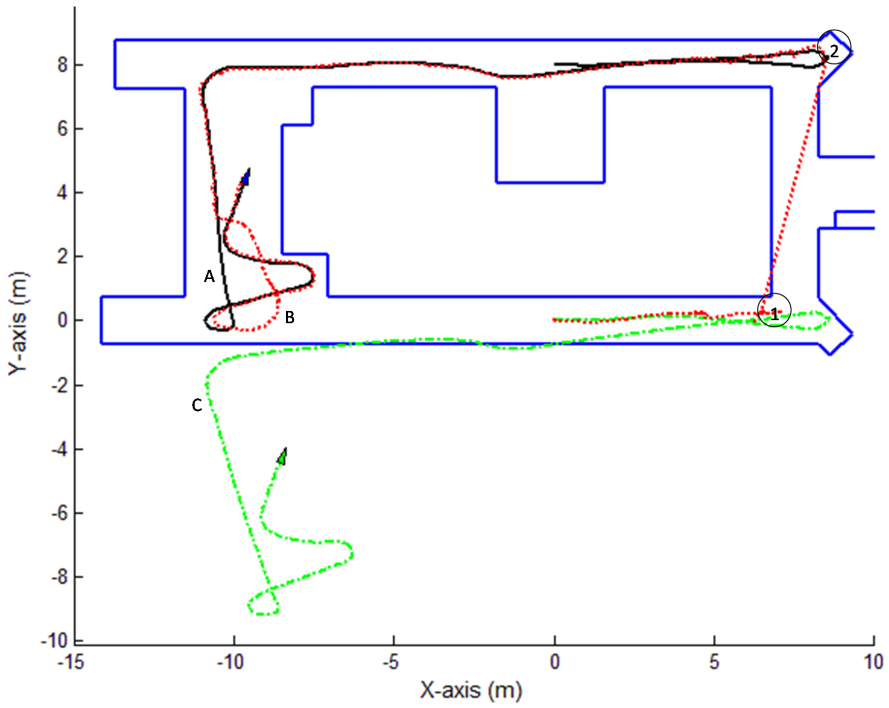
Drawing global samples

1: **for** $n = 1$ to N_G **do**
2: draw $s_t^{[n],G}$ with the uniform distribution in SER
3: add $s_t^{[n],G}$ to S_t^G
4: **end for**

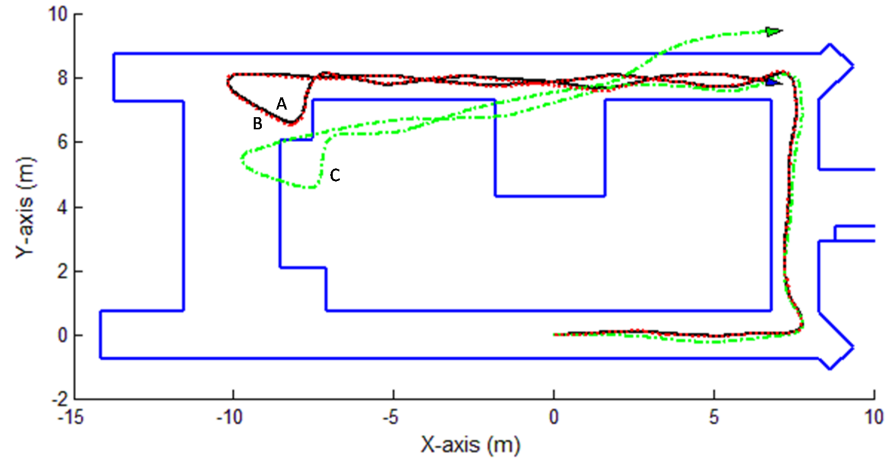
Combining two particle sets

1: $S_t = S_t^L \cup S_t^G$
2: **Output:** S_t

Dans cet exemple, le deuxième robot est bien localisé, alors que le premier a été kidnappé au début de l'expérience (Figure C.12).



(a)



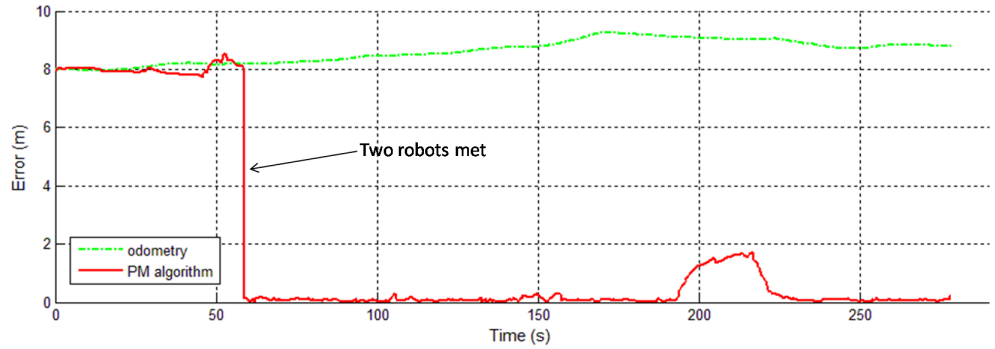
(b)

Figure C.12: La localisation multi-robot utilisant l'algorithme de PM.

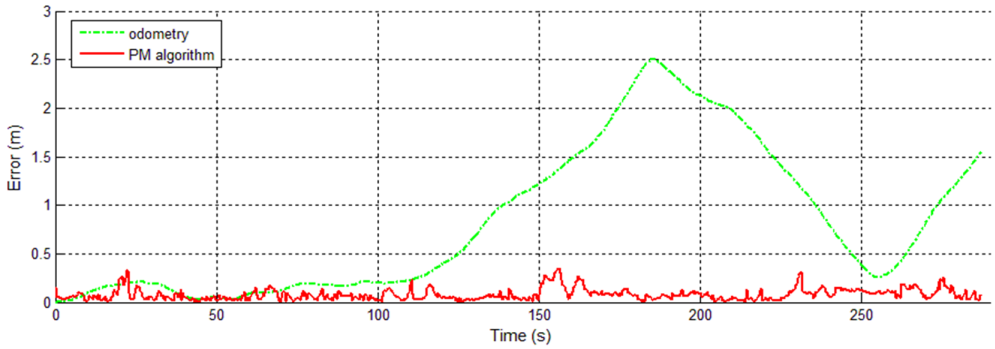
Les erreurs (odométriques et SAMCL) sont représentatives dans la Figure C.13.

IMPLÉMENTATION

Les algorithmes précédents ont été implémentés sur un robot réel de laboratoire (Figure C.14 (a)) au premier étage de notre laboratoire (Figure C.14 (b)).



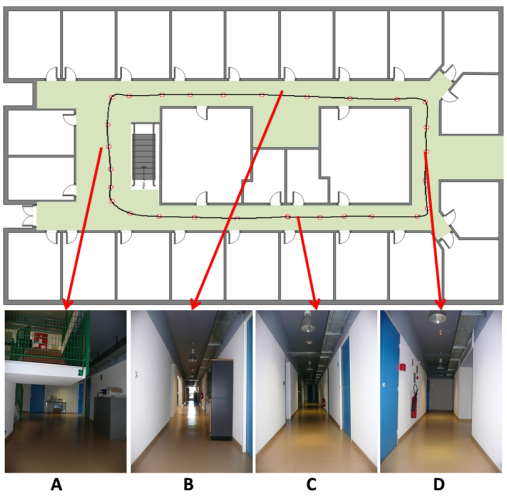
(a)



(b)

Figure C.13: Les erreurs de la localisation multi-robot.

(a)



(b)

Figure C.14: (a) Le robot Pioneer. (b) L'environnement expérimental.

La Figure C.15 montre la trajectoire odométrique (B) ainsi que la trajectoire obtenue par SAMCL (A) lorsque le robot effectue un tour de couloir avec une

erreur odométrique initiale.

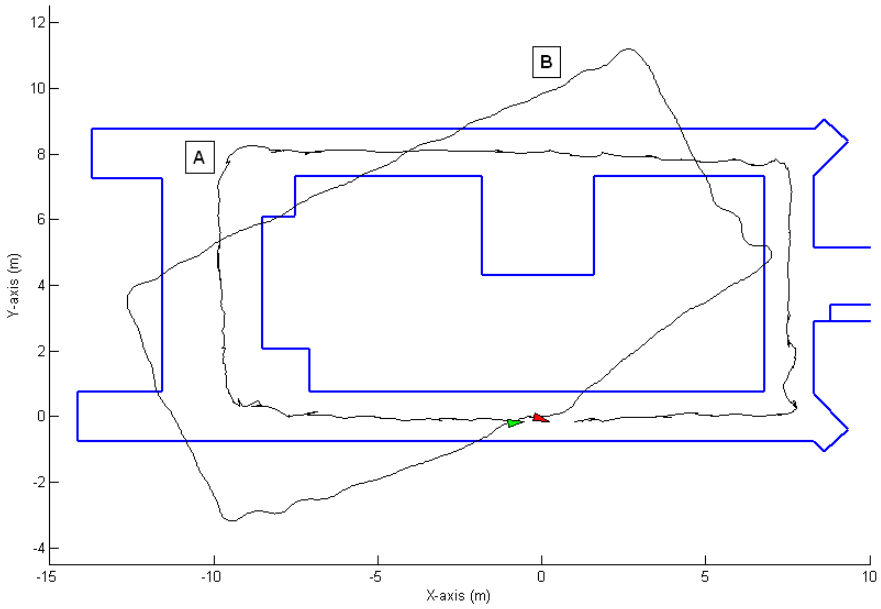


Figure C.15: La localisation globale avec une erreur odométrique initiale.

Table C.2: Erreurs moyennes pour la localisation globale.

	x	y	θ
Localisation	$0.157m$	$0.092m$	$6.5°$
Odométrie	$0.739m$	$0.215m$	$33.7°$

La figure suivante (Figure C.16) montre la trajectoire odométrique (B) ainsi que la trajectoire obtenue par SAMCL (A) lorsque le robot effectue un tour de couloir après ajout d'une erreur odométrique importante.

Table C.3: Erreurs moyennes pour la localisation globale avec ajout d'erreurs.

	x	y	θ
Localisation	$0.605m$	$0.076m$	$13.2°$
Odom	$5.6728m$	$5.017m$	$45.3°$

La figure suivante (Figure C.17) compare les taux de succès entre la méthode classique de Monte Carlo et la méthode SAMCL avec échantillonnage dans la zone d'énergie similaire, en fonction du nombre de particules.

Figure C.16: La localisation globale avec ajout d'erreurs artificielles.

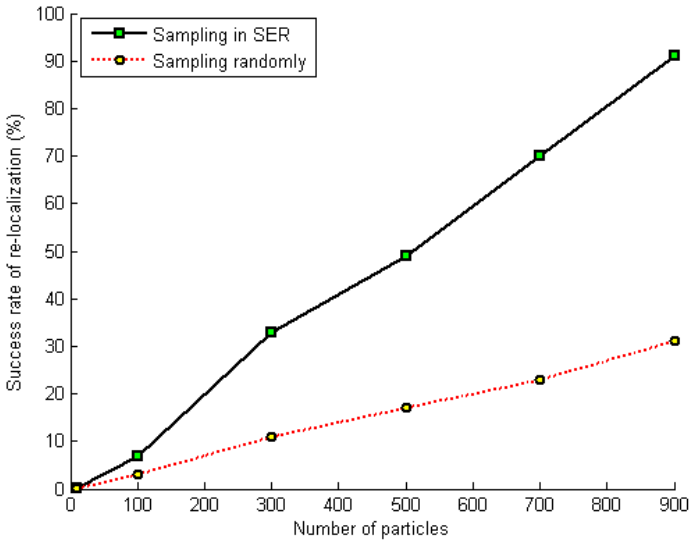


Figure C.17: Le taux de succès en fonction du nombre de particules.

Bibliography

[Arsenio98] A. Arsenio & M. I. Ribeiro. *Absolute localization of mobile robots using natural landmarks.* In Proceedings of IEEE International Conference on Electronics, Circuits and Systems, volume 2, pages 483–486 vol.2, 1998.

[Baltzakis02] H. Baltzakis & P. Trahanias. *Hybrid mobile robot localization using switching state-space models.* In Proceedings of IEEE International Conference on Robotics and Automation ICRA '02, volume 1, pages 366–373 vol.1, 2002.

[Baltzakis03] H. Baltzakis & P. E. Trahanias. *A Hybrid Framework for Mobile Robot Localization: Formulation Using Switching State-Space Models.* Autonomous Robots, vol. 15, no. 2, pages 169–191, 2003.

[Bekkali08] A. Bekkali & M. Matsumoto. *Bayesian sensor model for indoor localization in Ubiquitous Sensor Network.* In Proceedings of First ITU-T Kaleidoscope Academic Conference Innovations in NGN: Future Network and Services K-INGN 2008, pages 285–292, 2008.

[Blanco08] J. L. Blanco, J. Gonzalez & J. A. Fernandez-Madrigal. *An optimal filtering algorithm for non-parametric observation models in robot localization.* In Proceedings of IEEE International Conference on Robotics and Automation ICRA 2008, pages 461–466, 2008.

[Borenstein90] J. Borenstein & Y. Koren. *Real-time obstacle avoidance for fast mobile robots in cluttered environments.* In Proceedings of IEEE International Conference on Robotics and Automation, pages 572–577 vol.1, 1990.

[Borenstein91a] J. Borenstein & Y. Koren. *Histogramic in-motion mapping for mobile robot obstacle avoidance.* IEEE Transactions on Robotics and Automation, vol. 7, no. 4, pages 535–539, 1991.

[Borenstein91b] J. Borenstein & Y. Koren. *The vector field histogram-fast obstacle avoidance for mobile robots.* IEEE Transactions on Robotics and Automation, vol. 7, no. 3, pages 278–288, 1991.

[Borenstein95] J. Borenstein & Y. Koren. *Error Eliminating Rapid Ultrasonic Firing for Mobile Robot Obstacle Avoidance*. IEEE Transactions on Robotics and Automation, vol. 11, no. 1, pages 132–138, 1995.

[Borenstein96] J. Borenstein, H. R. Everett & L. Feng. Navigating mobile robots: Systems and techniques. A. K. Peters, Ltd., Wellesley, MA, 1996.

[Borges00] G. A. Borges & M.-J. Aldon. *A split-and-merge segmentation algorithm for line extraction in 2D range images*. In Proceedings of 15th International Conference on Pattern Recognition, volume 1, pages 441–444, 2000.

[Borges02] G. A. Borges. *Cartographie de l'environnement et localisation robuste pour la navigation de robots mobiles*. PhD thesis, Montpellier University II, 2002.

[Borges04] G. A. Borges & M.-J. Aldon. *Line Extraction in 2D Range Images for Mobile Robotics*. Journal of Intelligent and Robotic Systems, vol. 40, no. 3, pages 267–297, 2004.

[Burgard96] W. Burgard, D. Fox, D. Hennig & 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, pages 896–901. AAAI Press/MIT Press, 1996.

[Burgard97a] W. Burgard, D. Fox & S. Thrun. *Active mobile robot localization by entropy minimization*. In Proceedings of Second EUROMICRO workshop on Advanced Mobile Robots, pages 155–162, 1997.

[Burgard97b] W. Burgard, D. Fox & D. Hennig. *Fast Grid-based Position Tracking for Mobile Robots*. In Proceedings of the 21th German Conference on Artificial Intelligence, pages 289–300. Springer Verlag, 1997.

[Burgard97c] W. Burgard, D. Fox & S. Thrun. *Active mobile robot localization*. In Proceedings of IJCAI-97. Morgan Kaufmann, 1997.

[Burgard98] W. Burgard, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner & S. Thrun. *The interactive museum tour-guide robot*. In AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence, pages 11–18, 1998.

[Burgard99] W. Burgard, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner & S. Thrun. *Experiences with an interactive museum tour-guide robot*. Artificial Intelligence, vol. 114, no. 1-2, pages 3–55, 1999.

[Burgard05] W. Burgard, M. Moors, C. Stachniss & F. E. Schneider. *Coordinated multi-robot exploration.* IEEE Transactions on Robotics, vol. 21, no. 3, pages 376–386, 2005.

[Burguera07] A. Burguera, Y. Gonzalez & G. Oliver. *Probabilistic Sonar Scan Matching for Robust Localization.* In Proceedings of IEEE International Conference on Robotics and Automation, pages 3154–3160, 2007.

[Cacitti01] A. Cacitti & R. Zapata. *Reactive Behaviours of Mobile Manipulator Based on the DVZ Method.* In Proceedings of the 2001 IEEE International Conference on Robotics and Automation (ICRA 2001), pages 680–685, 2001.

[Cao95] Y. U. Cao, A. S. Fukunaga, A. B. Kahng & F. Meng. *Cooperative mobile robotics: antecedents and directions.* In Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots', volume 1, pages 226–234, 1995.

[Choset01] H. Choset & K. Nagatani. *Topological simultaneous localization and mapping (SLAM): Toward exact localization without explicit localization.* IEEE Transactions on Robotics and Automation, vol. 17, pages 125–137, 2001.

[Courbon08] J. Courbon, Y. Mezouar, L. Eck & P. Martinet. *Efficient hierarchical localization method in an omnidirectional images memory.* In Proceedings of IEEE International Conference on Robotics and Automation ICRA 2008, pages 13–18, 2008.

[Cox90] I. J. Cox & G. T. Wilfong, editors. Autonomous robot vehicles. Springer-Verlag New York, Inc., 1990.

[Cox94] I. J. Cox & J. J. Leonard. *Modeling a dynamic environment using a Bayesian multiple hypothesis approach.* Artificial Intelligence, vol. 66, no. 2, pages 311–344, 1994.

[Cyb] Cyberbotics Ltd. *http://www.cyberbotics.com/cdrom/common/doc/webots/guide/section7.4.html.*

[DAL] DALSA Corporation. *http://www.dalsa.com/corp/markets/CCD_vs_CMOS.aspx.*

[Dellaert99] F. Dellaert, D. Fox, W. Burgard & S. Thrun. *Monte Carlo localization for mobile robots.* In Proceedings of IEEE International Conference on Robotics and Automation, volume 2, pages 1322–1328, 1999.

[Devy93] M. Devy, J.-J. Orteu, J. L. Fuentes-Cantillana, J. C. Catalina, A. Rodriguez, D. Dumahu & P. V. d. Janti. *Mining robotics: Application of computer vision to the automation of a roadheader*. Robotics and Autonomous Systems, vol. 11, no. 2, pages 65–74, 1993.

[Devy95] M. Devy, R. Chatila, P. Fillatreau, S. Lacroix & F. Nashashibi. *On autonomous navigation in a natural environment*. Robotics and Autonomous Systems, vol. 16, no. 1, pages 5–16, 1995.

[Dissanayake01] M. W. M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte & M. Csorba. *A solution to the simultaneous localization and map building (SLAM) problem*. IEEE Transactions on Robotics and Automation, vol. 17, no. 3, pages 229–241, 2001.

[Doucet00] A. Doucet, S. Godsill & C. Andrieu. *On sequential Monte Carlo sampling methods for Bayesian filtering*. STATISTICS AND COMPUTING, vol. 10, no. 3, pages 197–208, 2000.

[Fort-Piat97] N. L. Fort-Piat, I. Collin & D. Meizel. *Planning robust displacement missions by means of robot-tasks and local maps*. Robotics and Autonomous Systems, vol. 20, no. 1, pages 99–114, 1997.

[Fox98a] D. Fox, W. Burgard & S. Thrun. *Active Markov Localization for Mobile Robots*. Robotics and Autonomous Systems, vol. 25, pages 195–207, 1998.

[Fox98b] D. Fox, W. Burgard, S. Thrun & A. B. Cremers. *Position estimation for mobile robots in dynamic environments*. In AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence, pages 983–988, 1998.

[Fox99a] D. Fox, W. Burgard, F. Dellaert & S. Thrun. *Monte Carlo Localization: Efficient Position Estimation for Mobile Robots*. In Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI'99), pages 343–349, July 1999.

[Fox99b] D. Fox, W. Burgard & S. Thrun. *Markov localization for mobile robots in dynamic environments*. Journal of Artificial Intelligence Research, vol. 11, pages 391–427, 1999.

[Fox00a] D. Fox, W. Burgard, H. Kruppa & S. Thrun. *Efficient multi-robot localization based on Monte Carlo approximation*. In J. Hollerbach & D. Koditschek, editors, Robotics Research: the Ninth International Symposium. Springer-Verlag, London, 2000.

[Fox00b] D. Fox, W. Burgard, H. Kruppa & S. Thrun. *A Probabilistic Approach to Collaborative Multi-Robot Localization*. Autonomous Robots, vol. 8, no. 3, pages 325–344, Juin 2000.

[Fox01] D. Fox. *KLD-Sampling: Adaptive Particle Filters.* In Advances in Neural Information Processing Systems 14, pages 713–720. MIT Press, 2001.

[Fox03a] D. Fox. *Adapting the Sample Size in Particle Filters Through KLD-Sampling.* International Journal of Robotics Research, vol. 22, no. 12, pages 985–1003, 2003.

[Fox03b] D. Fox, J. Hightower, H. Kauz, L. Liao & D. Patterson. *Bayesian techniques for location estimation.* In Proceedings of The 2003 Workshop on Location-Aware Computing, pages 16–18, 2003.

[Fox03c] V. Fox, J. Hightower, L. Liao, D. Schulz & G. Borriello. *Bayesian filtering for location estimation.* IEEE Pervasive Computing, vol. 2, no. 3, pages 24–33, 2003.

[Franchi09] A. Franchi, G. Oriolo & P. Stegagno. *Mutual Localization in a Multi-Robot System with Anonymous Relative Position Measures.* In IROS 2009: The 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 3974–3980, St. Louis, USA, 2009.

[Friedman07] S. Friedman, H. Pasula & D. Fox. *Voronoi Random Fields: Extracting Topological Structure of Indoor Environments via Place Labeling.* In Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI), pages 2109–2114, 2007.

[Gasparri06] A. Gasparri, S. Panzieri, F. Pascucci & G. Ulivi. *Genetic approach for a localisation problem based upon particle filters.* In Proceedings of 8th Int. Symp. on Robot Control (SYROCO 2006), Bologna, Italy, September 2006.

[Gasparri07] A. Gasparri, S. Panzieri, F. Pascucci & G. Ulivi. *A Hybrid Active Global Localisation Algorithm for Mobile Robots.* In Proceedings of IEEE International Conference on Robotics and Automation, pages 3148–3153, 2007.

[Gasparri08a] A. Gasparri & M. Prosperi. *A bacterial colony growth framework for collaborative multi-robot localization.* In Proceedings of IEEE International Conference on Robotics and Automation ICRA 2008, pages 2806–2811, 2008.

[Gasparri08b] A. Gasparri & M. C. F. Prosperi. *A bacterial colony growth algorithm for mobile robot localisation.* Autonomous Robots, vol. 24, no. 4, pages 349–364, 2008.

[Grewal93] M. S. Grewal & A. P. Andrews. Kalman filtering: theory and practice. Prentice-Hall, Inc., 1993.

[Gutmann98] J. S. Gutmann, W. Burgard, D. Fox & K. Konolige. *An experimental comparison of localization methods.* In Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, volume 2, pages 736–743 vol.2, 1998.

[Gutmann02] J. S. Gutmann & D. Fox. *An experimental comparison of localization methods continued.* In Proceedings of IEEE/RSJ International Conference on Intelligent Robots and System, volume 1, pages 454–459 vol.1, 2002.

[Haralick92] R. M. Haralick & L. G. Shapiro. Computer and robot vision. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1992.

[Harrison08] A. Harrison & P. Newman. *High quality 3D laser ranging under general vehicle motion.* In Proceedings of IEEE International Conference on Robotics and Automation ICRA 2008, pages 7–12, 2008.

[Hester08] T. Hester & P. Stone. *Negative information and line observations for Monte Carlo localization.* In Proceedings of IEEE International Conference on Robotics and Automation ICRA 2008, pages 2764–2769, 2008.

[Hom] Homepage of Franz J. Vesely. *http://homepage.univie.ac.at/∼veselyf2/cp_tut/nol2h/new/c3st_s2od.html.*

[Howard06] A. Howard. *Multi-robot Simultaneous Localization and Mapping using Particle Filters.* International Journal of Robotics Research, vol. 25, no. 12, pages 1243–1256, 2006.

[Hsu09] C.-C. Hsu, H.-C. Chen & C.-Y. Lai. *An Improved Ultrasonic-Based Localization Using Reflection Method.* In Proc. International Asia Conference on Informatics in Control, Automation and Robotics CAR '09, pages 437–440, 1–2 Feb. 2009.

[Jaulin01] L. Jaulin, M. Kieffer, O. Didrit & E. Walter. Applied Interval Analysis. Springer, 2001.

[Jaulin02] L. Jaulin, M. Kieffer, E. Walter & D. Meizel. *Guaranteed robust nonlinear estimation with application to robot localization.* IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, vol. 32, no. 4, pages 374–381, 2002.

[Jaulin06] L. Jaulin. *Computing minimal-volume credible sets using interval analysis; Application to Bayesian estimation.* IEEE Trans. on Signal Processing, vol. 54, no. 9, pages 3632–3636, 2006.

[Jaulin09] L. Jaulin. *A Nonlinear Set-membership Approach for the Localization and Map Building of an Underwater Robot using Interval Constraint Propagation.* IEEE Transaction on Robotics, vol. 25, no. 1, pages 88–98, Feb 2009.

[Jensfelt01] P. Jensfelt & S. Kristensen. *Active global localization for a mobile robot using multiple hypothesis tracking.* IEEE Transactions on Robotics and Automation, vol. 17, no. 5, pages 748–760, 2001.

[Kalman60] R. Kalman. *A New Approach to Linear Filtering and Prediction Problems.* Transactions of the ASME–Journal of Basic Engineering, vol. 82, no. Series D, pages 35–45, 1960.

[Kieffer00] M. Kieffer, L. Jaulin, E. Walter & D. Meizel. *Robust Autonomous Robot Localization Using Interval Analysis.* Reliable Computing, vol. 6, no. 3, pages 337–362, 2000.

[Ko08] J. Ko & D. Fox. *GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models.* In Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems IROS 2008, pages 3471–3476, 2008.

[Kortenkamp94] D. Kortenkamp & T. Weymouth. *Topological mapping for mobile robots using a combination of sonar and vision sensing.* In AAAI'94: Proceedings of the twelfth national conference on Artificial intelligence, volume 2, pages 979–984. AAAI, AAAI Press/MIT Press, 1994.

[Kümmerle08] R. Kümmerle, R. Triebel, P. Pfaff & W. Burgard. *Monte Carlo localization in outdoor terrains using multilevel surface maps.* Journal of Field Robotics, vol. 25, no. 6-7, pages 346–359, 2008.

[Kwok04] C. Kwok, D. Fox & M. Meila. *Real-time particle filters.* Proceedings of the IEEE, vol. 92, no. 3, pages 469–484, 2004.

[Lapierre07a] L. Lapierre, R. Zapata & P. Lépinay. *Combined Path-following and Obstacle Avoidance Control of a Wheeled Robot.* International Journal of Robotics Research, vol. 26, no. 4, pages 361–375, 2007.

[Lapierre07b] L. Lapierre, R. Zapata & P. Lépinay. *Simulatneous Path Following and Obstacle Avoidance Control of a Unicycle-type Robot.* In ICRA, pages 2617–2622. IEEE, 2007.

[Laumond98] J.-P. Laumond, editor. Robot motion planning and control. Springer-Verlag New York, Inc., 1998.

[Leonard91] J. J. Leonard & H. F. Durrant-Whyte. *Mobile robot localization by tracking geometric beacons.* IEEE Transactions on Robotics and Automation, vol. 7, no. 3, pages 376–382, 1991.

[Leonard92] J. J. Leonard & H. F. Durrant-Whyte. Directed sonar sensing for mobile robot navigation. Kluwer Academic Publishers, 1992.

[Liu05] J. Liu, K. Yuan, W. Zou & Q. Yang. *Monte Carlo multi-robot localization based on grid cells and characteristic particles.* In Proceedings of IEEE/ASME International Conference on Advanced Intelligent Mechatronics, pages 510–515, 2005.

[Madhavan04] R. Madhavan & H. F. Durrant-Whyte. *Natural landmark-based autonomous vehicle navigation.* Robotics and Autonomous Systems, vol. 46, no. 2, pages 79–95, 2004.

[Mat] The MathWorks, Inc. *http://www.mathworks.com/products/matlab/.*

[Maybeck79] P. S. Maybeck. Stochastic models, estimation and control. Academic Press, Inc., 1979.

[Metropolis49] N. Metropolis & S. Ulam. *The Monte Carlo Method.* Journal of the American Statistical Association, vol. 44, no. 247, pages 335–341, 1949.

[Milstein02] A. Milstein, J. N. Sánchez & E. T. Williamson. *Robust global localization using clustered particle filtering.* In AAAI-02, pages 581–586, 2002.

[Moravec85] H. Moravec & A. Elfes. *High resolution maps from wide angle sonar.* In Proceedings of IEEE International Conference on Robotics and Automation, volume 2, pages 116–121, 1985.

[Moreno02] L. Moreno, J. M. Armingol, S. Garrido, A. De La Escalera & M. A. Salichs. *A Genetic Algorithm for Mobile Robot Localization Using Ultrasonic Sensors.* Journal of Intelligent and Robotic Systems, vol. 34, no. 2, pages 135–154, 2002.

[Murrieta-Cid02] R. Murrieta-Cid, C. Parra & M. Devy. *Visual Navigation in Natural Environments: From Range and Color Data to a Landmark-Based Model.* Autonomous Robots, vol. 13, no. 2, pages 143–168, 2002.

[Nguyen07] V. Nguyen, S. Gächter, A. Martinelli, N. Tomatis & R. Siegwart. *A comparison of line extraction algorithms using 2D range data for indoor mobile robotics.* Autonomous Robots, vol. 23, no. 2, pages 97–111, 2007.

[Núñez08] P. Núñez, R. V. Martín, J. C. d. Toro Lasanta, A. Bandera & F. S. Hernández. *Natural landmark extraction for mobile robot navigation based on an adaptive curvature estimation.* Robotics and Autonomous Systems, vol. 56, no. 3, pages 247–264, 2008.

[Omohundro90] S. M. Omohundro. *Bumptrees for efficient function, constraint, and classification learning.* In NIPS-3: Proceedings of the 1990 conference on Advances in neural information processing systems 3, pages 693–699. Morgan Kaufmann Publishers Inc., 1990.

[Prestes08] E. Prestes, M. Ritt & G. Fuhr. *Improving Monte Carlo Localization in sparse environments using structural environment information.* In Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems IROS 2008, pages 3465–3470, 2008.

[Reuter00] J. Reuter. *Mobile robot self-localization using PDAB.* In Proceedings of IEEE International Conference on Robotics and Automation ICRA '00, volume 4, pages 3512–3518 vol.4, 2000.

[Ritter96] G. X. Ritter & J. N. Wilson. Handbook of computer vision algorithms in image algebra. CRC Press, Inc., Boca Raton, FL, USA, 1996.

[Roumeliotis00] S. I. Roumeliotis & G. A. Bekey. *Bayesian estimation and Kalman filtering: a unified framework for mobile robot localization.* In Proceedings of IEEE International Conference on Robotics and Automation (ICRA '00), volume 3, pages 2985–2992, 2000.

[Roumeliotis02] S. I. Roumeliotis & G. A. Bekey. *Distributed multi-robot localization.* IEEE Transactions on Robotics and Automation, vol. 18, no. 5, pages 781–795, 2002.

[Sack04] D. Sack & W. Burgard. *A comparison of methods for line extraction from range data.* In Proceedings of the 5th IFAC Symposium on Intelligent Autonomous Vehicles (IAV), 2004.

[Sakai09] A. Sakai, Y. Tamura & Y. Kuroda. *An Efficient Solution to 6DOF Localization Using Unscented Kalman Filter for Planetary Rovers.* In IROS 2009: The 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 4154–4159, St. Louis, USA, 2009.

[Schiele94] B. Schiele & J. L. Crowley. *A comparison of position estimation techniques using occupancy.* In Proceedings of IEEE International Conference on Robotics and Automation, volume 2, pages 1628–1634, 1994.

[Schulz03] D. Schulz, W. Burgard, D. Fox & A. B. Cremers. *People Tracking with Mobile Robots Using Sample-based Joint Probabilistic Data Association Filters.* International Journal of Robotics Research, vol. 22, no. 2, pages 99–116, 2003.

[Seignez06] E. Seignez. *Étude et comparaison expérimentale de méthodes de localisation multicapteurs.* PhD thesis, Université Paris-Sud 11, 2006.

[Siagian07] C. Siagian & L. Itti. *Biologically-inspired robotics vision Monte-Carlo localization in the outdoor environment.* In Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems IROS 2007, pages 1723–1730, 2007.

[Siegwart04] R. Siegwart & I. R. Nourbakhsh. Introduction to autonomous mobile robots. A Bradford Book, The MIT Press, 2004.

[Silva07] P. R. A. Silva & M. G. S. Bruno. *A Density-Assisted Particle Filter for Mobile Robot Localization with Uncertain Environment MAP.* In Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing ICASSP 2007, volume 3, pages III–1205–III–1208, 2007.

[Smith92] A. F. M. Smith & A. E. Gelfand. *Bayesian Statistics without Tears: A Sampling-Resampling Perspective.* The American Statistician, vol. 46, no. 2, pages 84–88, 1992.

[Sola08] J. Sola, A. Monin, M. Devy & T. Vidal-Calleja. *Fusing Monocular Information in Multicamera SLAM.* IEEE Transactions on Robotics, vol. 24, no. 5, pages 958–968, 2008.

[Solutions07] A. K. Solutions. Robotics. Infinity Science Press LLC, 2007.

[Stéphant04] J. Stéphant, A. Charara & D. Meizel. *Virtual sensor, application to vehicle sideslip angle and transversal forces.* IEEE Transactions on Industrial Electronics, vol. 51, no. 2, pages 278–289, 2004.

[Stéphant07] J. Stéphant, A. Charara & D. Meizel. *Evaluation of a sliding mode observer for vehicle sideslip angle.* Control Engineering Practice, vol. 15, pages 803–812, 2007.

[Thrun98a] S. Thrun. *Learning Metric-Topological Maps for Indoor Mobile Robot Navigation.* Artificial Intelligence, vol. 99, no. 1, pages 21–71, 1998.

[Thrun98b] S. Thrun, J. s. Gutmann, D. Fox, W. Burgard & B. J. Kuipers. *Integrating topological and metric maps for mobile robot navigation: A statistical approach.* In Proceedings of AAAI-98, pages 989–995. AAAI Press/MIT Press, 1998.

[Thrun99a] S. Thrun, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Hähnel, G. Lakemeyer, C. Rosenberg, N. Roy, J. Schulte, D. Schulz & W. Steiner. *Experiences with two deployed interactive*

tour-guide robots. In Proceedings of the International Conference on Field and Service Robotics, 1999.

[Thrun99b] S. Thrun, J. C. Langford & D. Fox. *Monte Carlo Hidden Markov Models: Learning Non-Parametric Models of Partially Observable Stochastic Processes.* In ICML '99: Proceedings of the Sixteenth International Conference on Machine Learning, pages 415–424. Morgan Kaufmann Publishers Inc., 1999.

[Thrun00a] S. Thrun, M. Beetz, M. Bennewitz, W. Burgard, A. Cremers, F. Dellaert, D. Fox, D. Hähnel, C. Rosenberg, N. Roy, J. Schulte & D. Schulz. *Probabilistic Algorithms and the Interactive Museum Tour-Guide Robot Minerva.* International Journal of Robotics Research, vol. 19, pages 972–999, 2000.

[Thrun00b] S. Thrun, D. Fox, W. Burgard & F. Dellaert. *Robust Monte Carlo Localization for Mobile Robots.* Artificial Intelligence, vol. 128, no. 1-2, pages 99–141, 2000.

[Thrun00c] S. Thrun. *Monte Carlo POMDPs.* In Advances in Neural Information Processing 12, pages 1064–1070, 2000.

[Thrun00d] S. Thrun, D. Fox & W. Burgard. *Monte Carlo localization with mixture proposal distribution.* In Proceedings of the AAAI National Conference on Artificial Intelligence, pages 859–865, 2000.

[Thrun05] S. Thrun, W. Burgard & D. Fox. Probabilistic robotics. The MIT Press, September 2005.

[Weiss94] G. Weiss, C. Wetzler & E. v. Puttkamer. *Keeping track of position and orientation of moving indoor systems by correlation of range-finder scans.* In Proceedings of the International Conference on Intelligent Robots and Systems, volume 1, pages 595–601, 1994.

[Welch95] G. Welch & G. Bishop. *An Introduction to the Kalman Filter.* Technique report, University of North Carolina at Chapel Hill, Department of Computer Science, 1995.

[Wik] Wikimedia Foundation, Inc. *http://en.wikipedia.org/wiki/Kalman_filter.*

[Wolf02] J. Wolf, W. Burgard & H. Burkhardt. *Robust vision-based localization for mobile robots using an image retrieval system based on invariant features.* In Proceedings of IEEE International Conference on Robotics and Automation (ICRA), 2002.

[Wu01] C.-J. Wu & C.-C. Tsai. *Localization of an Autonomous Mobile Robot Based on Ultrasonic Sensory Information.* Journal of Intelligent and Robotic Systems, vol. 30, no. 3, pages 267–277, 2001.

[Zapata04] R. Zapata, A. Cacitti & P. Lépinay. *DVZ-based collision avoidance control of non-holonomic mobile manipulators.* Journal européen des systmes automatisés, vol. 38, no. 5, pages 559–588, 2004.

[Zapata05] R. Zapata, P. Lépinay, B. Jacquot & D. R. Ocampo. *Co-design of fast biologically-plausible vision-based systems for controlling the reactive behaviors of mobile robots.* Journal of Robotic Systems, vol. 22, no. 7, pages 341–357, 2005.

[Zhang00] L. Zhang & B. K. Ghosh. *Line segment based map building and localization using 2D laser rangefinder.* In Proceedings of IEEE International Conference on Robotics and Automation ICRA '00, volume 3, pages 2538–2543 vol.3, 2000.

[Zhang09a] L. Zhang & R. Zapata. *Probabilistic Localization Methods of a Mobile Robot Using Ultrasonic Perception System.* In Proceedings of IEEE International Conference on Information and Automation (ICIA 2009), pages 1062–1067, 2009.

[Zhang09b] L. Zhang & R. Zapata. *A Three-step Localization Method for Mobile Robots.* In Proceedings of International Conference on Automation, Robotics and Control Systems (ARCS 2009), pages 50–56, 2009.

[Zhang09c] L. Zhang, R. Zapata & P. Lépinay. *Self-Adaptive Monte Carlo for Single-Robot and Multi-Robot Localization.* In Proceedings of IEEE International Conference on Automation and Logistics (ICAL 2009), pages 1927–1933, 2009.

[Zhang09d] L. Zhang, R. Zapata & P. Lépinay. *Self-Adaptive Monte Carlo Localization for Cooperative Multi-Robot Systems.* In 10th Towards Autonomous Robotic Systems (TAROS 2009), pages 259–266, 2009.

[Zhang09e] L. Zhang, R. Zapata & P. Lépinay. *Self-adaptive Monte Carlo localization for mobile robots using range sensors.* In Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2009), pages 1541–1546, 2009.

[Zhou07] Y. Zhou, W. Liu & P. Huang. *Laser-activated RFID-based Indoor Localization System for Mobile Robots.* In Proceedings of IEEE International Conference on Robotics and Automation, pages 4600–4605, 2007.

[Zogg07] H.-M. Zogg & H. Ingensand. *Terrestrial 3D-laser scanner ZLS07 developed at ETH Zurich: an overview of its configuration, performance and application.* In Optical 3-D Measurement Techniques VIII, 2007.

TITRE en français

Localisation Markovienne de Systèmes Mono-robot et Multi-robots Utilisant des Echantillons Auto-adaptatifs

RESUME en français

Afin de parvenir à l'autonomie des robots mobiles, la localisation efficace est une condition préalable nécessaire. Le suivi de position, la localisation globale et le problème du robot kidnappé sont les trois sous-problèmes que nous étudions. Dans cette thèse, nous comparons en simulation trois algorithmes de localisation Markovienne. Nous proposons ensuite une amélioration de l'algorithme de localisation de Monte Carlo par filtre particulaire. Cet algorithme (nommé SAMCL) utilise des particules auto-adaptatives. En employant une technique de pré-mise en cache pour réduire le temps de calcul en ligne, l'algorithme SAMCL est plus efficace que la méthode de Monte Carlo usuelle. En outre, nous définissons la notion de région d'énergie similaire (SER), qui est un ensemble de poses (cellules de la grille) dont l'énergie-capteur est similaire avec celle du robot dans l'espace réel. En distribuant les échantillons globaux dans SER lieu de les distribuer au hasard dans la carte, SAMCL obtient une meilleure performance dans la localisation et résout ces trois sous-problèmes.

La localisation coopérative de plusieurs robots est également étudiée. Nous avons développé un algorithme (nommé PM) pour intégrer l'information de localisation échangée par les robots lors d'une rencontre au cours d'une mission commune. Cet algorithme apparaît comme une extension à l'algorithme de SAMCL et a été validé en simulation.

La validité et l'efficacité de notre approche sont démontrées par des expériences sur un robot réel évoluant dans un environnement connu et préalablement cartographié.

TITRE en anglais

Self-adaptive Markov Localization for Single-Robot and Multi-Robot Systems

RESUME en anglais

In order to achieve the autonomy of mobile robots, effective localization is a necessary prerequisite. In this thesis, we first study and compare three regular Markov localization algorithms by simulations. Then we propose an improved Monte Carlo localization algorithm using self-adaptive samples, abbreviated as SAMCL. By employing a pre-caching technique to reduce the on-line computational burden, SAMCL is more efficient than regular MCL. Further, we define the concept of similar energy region (SER), which is a set of poses (grid cells) having similar energy with the robot in the robot space. By distributing global samples in SER instead of distributing randomly in the map, SAMCL obtains a better performance in localization. Position tracking, global localization and the kidnapped robot problem are the three sub-problems of the localization problem. Most localization approaches focus on solving one of these sub-problems. However, SAMCL solves all the three sub-problems together thanks to self-adaptive samples that can automatically separate themselves into a global sample set and a local sample set according to needs.

Cooperative localization among multiple robots is carried out by exchanging localization information derived from cooperation. We devise the Position Mapping (PM) algorithm to integrate this information, which can merge into the SAMCL algorithm as an extension.

The validity and the efficiency of our algorithm are demonstrated by experiments carried out with a real robot in a structured and known environment. Extensive experiment results and comparisons are also given in this thesis.

DISCIPLINE

Génie Informatique, Automatique et Traitement du Signal

MOTS-CLES

Robotique, localisation, multi-robot, filtres particulaires, Monte Carlo, méthodes stochastiques

INTITULE ET ADRESSE DE L'U.F.R. OU DU LABORATOIRE :

Le Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM)
UMR 5506 - CC 477
161 rue Ada
34392 Montpellier Cedex 5 - France