Self-adaptive Monte Carlo Localization for Mobile Robots Using Range Finders

René Zapata, Lei Zhang, Pascal Lepinay

Self-adaptive Monte Carlo Localization for Mobile Robots Using Range Finders

Lei Zhang, René Zapata and Pascal Lépinay
Laboratoire d'Informatique, de Robotique et de
Microélectronique de Montpellier (LIRMM)
Université Montpellier II
161 rue Ada, 34392 Montpellier Cedex 5, France
{lei.zhang,rene.zapata,pascal.lepinay}@lirmm.fr

***Abstract*—In order to achieve the autonomy of mobile robots, effective localization is a necessary prerequisite. In this paper, 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 these 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. The validity and the efficiency of the SAMCL algorithm are demonstrated by both simulations and experiments carried out with different intentions. Extensive experiment results and comparisons are also given in this paper.**

***Index Terms*—Localization; Probabilistic approach; Self-adaptive; Monte Carlo; Mobile robot; Kidnapping;**

I. Introduction

Effective localization is a fundamental prerequisite for achieving autonomous mobile robot navigation. Localization is defined as the problem of determining the pose (or position) of a robot given a map of the environment and sensors data [1], [2], [3]. Usually, the mobile robot pose comprises its $x-y$ coordinates and its orientation.

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 [4], [5], [6], [7].

Position tracking assumes that the robot knows its initial pose [8], [9]. 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 [6], [10]. 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 [11]. 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 [6], [12], [7]. 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 [13], [14].

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.

Among the existing position tracking algorithms, the Extended Kalman Filter (EKF) is one of the most popular approaches [15], [16], [17], [7]. 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 [18], [19], [5], [20]. 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 [12].

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 [21], [22], [23], [24], [7]. 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 [25], [26], [27], [28], [29], [30], [31], [7], [32], [33], [34]. 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.* [7] 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.* [35], [12], [7] 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.

In this paper, we propose the Self-Adaptive Monte Carlo Localization algorithm (abbreviated as SAMCL) to solve the localization problem. This algorithm is derived from the MCL algorithm; however it is improved in three aspects. Firstly, it employs a pre-caching technique to reduce the on-line computational burden of MCL. Thrun *et al.* [7] 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) that 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. 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 in this paper are different from the KLD-Sampling algorithm proposed in [36], [7]. 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 extension 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 recover from kidnapping.

The rest of this paper is organized as follows. In section II and III, we briefly review Bayes filters and Monte Carlo localization. In section IV, we introduce the SAMCL algorithm. Simulation and experiment results are presented in section V and VI. Finally some conclusions are given in section VII.

II. Bayes Filter

MCL is a Bayes-based Markov localization algorithm. The Bayes filter technique provides a powerful statistical tool to understand and solve robot localization problems [26], [5], [35], [37], [38], [39], [40], [41]. It calculates recursively the belief distribution $bel(*)$ from measurement data and control data [7]. 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.

$$bel(s_t) = p(s_t \,|Z_t, U_t\,) \tag{1}$$

We define measurements Z_t and controls U_t as follows:

$$\begin{aligned} Z_t &= \{z_t, z_{t-1}, \cdots, z_0\}\,, \\ U_t &= \{u_t, u_{t-1}, \cdots, u_1\} \end{aligned} \tag{2}$$

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

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

$$bel(s_t) = p\left(s_t \left| Z_t \right.\right) \cdot p\left(s_t \left| U_t \right.\right) \tag{3}$$

The term $p\left(s_t \left| Z_t \right.\right)$ is denoted as $bel_m(s_t)$, which represents the posterior belief after integrating the perception data.

$$\begin{aligned} bel_m(s_t) &= p\left(s_t \left| Z_t \right.\right) \\ &\overset{Bayes\ rule}{=} \frac{p\left(z_t \left| s_t, Z_{t-1} \right.\right) p\left(s_t \left| Z_{t-1} \right.\right)}{p\left(z_t \left| Z_{t-1} \right.\right)} \\ &= \eta p\left(z_t \left| s_t, Z_{t-1} \right.\right) p\left(s_t \left| Z_{t-1} \right.\right) \\ &\overset{Markov\ assum.}{=} \eta p\left(z_t \left| s_t \right.\right) p\left(s_{t-1} \left| Z_{t-1} \right.\right) \\ &= \eta p\left(z_t \left| s_t \right.\right) bel_m(s_{t-1}) \end{aligned} \tag{4}$$

where η is a normalization constant that ensures $bel_m(s_t)$ to sum up to one.

The term $p\left(s_t \left| U_t \right.\right)$ is denoted as $bel_c(s_t)$, which represents the posterior belief after integrating the control data.

$$\begin{aligned} bel_c(s_t) &= p\left(s_t \left| U_t \right.\right) \\ &\overset{Total\ prob.}{=} \int p\left(s_t \left| s_{t-1}, U_t \right.\right) p\left(s_{t-1} \left| U_t \right.\right) ds_{t-1} \\ &\overset{Markov\ assum.}{=} \int p\left(s_t \left| s_{t-1}, u_t \right.\right) p\left(s_{t-1} \left| U_{t-1} \right.\right) ds_{t-1} \\ &= \int p\left(s_t \left| s_{t-1}, u_t \right.\right) bel_c(s_{t-1}) ds_{t-1} \end{aligned} \tag{5}$$

We multiply $bel_m(s_t)$ by $bel_c(s_t)$ to get the final localization formula:

$$\begin{aligned} bel(s_t) &= bel_m(s_t) \cdot bel_c(s_t) \\ &= \eta p\left(z_t \left| s_t \right.\right) bel_m(s_{t-1}) \\ &\quad \int p\left(s_t \left| s_{t-1}, u_t \right.\right) bel_c(s_{t-1}) ds_{t-1} \\ &= \eta p\left(z_t \left| s_t \right.\right) \\ &\quad \int p\left(s_t \left| s_{t-1}, u_t \right.\right) \left[bel_m(s_{t-1}) \cdot bel_c(s_{t-1})\right] ds_{t-1} \\ &= \eta p\left(z_t \left| s_t \right.\right) \int p\left(s_t \left| s_{t-1}, u_t \right.\right) bel(s_{t-1}) ds_{t-1} \end{aligned} \tag{6}$$

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

In practice, the implementation of Equation 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:

$$\begin{aligned} \overline{bel}(s_t) &= p(s_t \left| Z_{t-1}, U_t \right.) \\ &= \int p\left(s_t \left| s_{t-1}, u_t \right.\right) bel(s_{t-1}) ds_{t-1} \end{aligned} \tag{7}$$

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

$$\begin{aligned} bel(s_t) &= \eta p\left(z_t \left| s_t \right.\right) \overline{bel}(s_t) \\ &= \eta p\left(z_t \left| s_t \right.\right) \int p\left(s_t \left| s_{t-1}, u_t \right.\right) bel(s_{t-1}) ds_{t-1} \end{aligned} \tag{8}$$

III. Monte Carlo Localization

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

$$S_t = \left\{ \left\langle s_t^{[n]}, \omega_t^{[n]} \right\rangle \right\}_{n=1,\cdots,N} \tag{9}$$

Each particle $s_t^{[n]}$ with $1 \leq n \leq N$ denotes a concrete instantiation of the robot's pose at time t. The number of particles N may be a fixed value or changing with some quantities related to the belief $bel(s_t)$ [42], [36], [40]. The $\omega_t^{[n]}$ is the non-negative numerical factor called importance factor. We interpret $\omega_t^{[n]}$ as the weight of a particle.

The basic MCL algorithm is depicted in Algorithm 1, which calculates the particle set S_t recursively from the set S_{t-1}. 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. Obviously, the pair $\left(s_t^{[n]}, s_{t-1}^{[n]}\right)$ is distributed according to the product distribution.

$$p\left(s_t^{[n]} \left| s_{t-1}^{[n]}, u_t, m \right.\right) \times bel(s_{t-1}^{[n]}) \tag{10}$$

In accordance with the literature on the Sampling Importance Resampling (SIR) algorithm [43], [44], this distribution is called the proposal distribution. It corresponds to the Equation 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 8. It is restated here for the MCL algorithm.

$$\eta p\left(z_t \left| s_t^{[n]} \right.\right) p\left(s_t^{[n]} \left| s_{t-1}^{[n]}, u_t, m \right.\right) bel(s_{t-1}^{[n]}) \tag{11}$$

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.

$$\begin{aligned}\omega_t^{[n]} &= \frac{\text{target distribution}}{\text{proposal distribution}} \\ &= \frac{\eta p\left(z_t \left| s_t^{[n]}\right.\right) p\left(s_t^{[n]} \left| s_{t-1}^{[n]}, u_t, m\right.\right) bel(s_{t-1}^{[n]})}{p\left(s_t^{[n]} \left| s_{t-1}^{[n]}, u_t, m\right.\right) bel(s_{t-1}^{[n]})} \\ &= \eta p\left(z_t \left| s_t^{[n]}\right.\right) \end{aligned} \tag{12}$$

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 [7].

The process of calculating the importance factor is the measurement update. 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) [7]. 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 $\bar{S}_t$. 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 $bel(s_t)$.

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 $s_t^{[n]} \sim p\left(s_t \left| s_{t-1}^{[n]}, u_t, m\right.\right)$
5: calculate an importance factor $\omega_t^{[n]} = p\left(z_t \left| s_t^{[n]}, m\right.\right)$
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 $\omega_t^{[n]}$
11: add $s_t^{[n]}$ to S_t
12: **end for**
13: **Output:** S_t

Algorithm 1: Basic MCL algorithm, adapted from [7]

IV. The SAMCL Algorithm

The SAMCL algorithm is implemented in three steps, as illustrated in Figure 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.

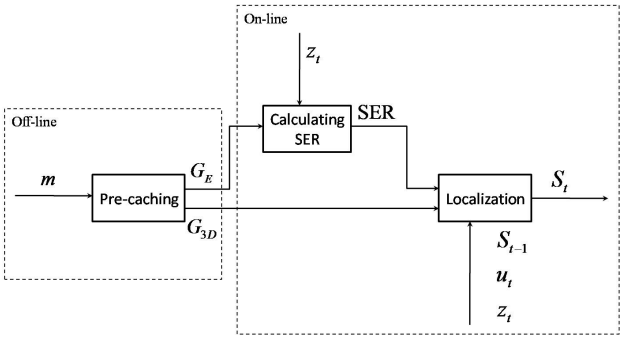


Fig. 1. The process of the SAMCL algorithm.

A. 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:

$$a_i = 1 - d_i/d_{\max} \tag{13}$$

where, d_i is the measurement of i^{th} sensor and $d_{\max}$ is the maximum distance that sensors are able to "see".

Then we define energy of a robot (or a grid cell) as the sum of energy of all the sensors.

$$E = \sum_{i=1}^{I} a_i \tag{14}$$

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 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:

$$E_a = (1 - \frac{d_1}{d_{\max}}) + (1 - \frac{d_2}{d_{\max}}) + (1 - \frac{d_3}{d_{\max}}) \tag{15}$$

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

$$E_b = (1 - \frac{e_1}{e_{\max}}) + (1 - \frac{e_2}{e_{\max}}) + (1 - \frac{e_3}{e_{\max}}) \tag{16}$$

Obviously, we have

$$E_a > E_b \tag{17}$$

If such robots are used, we provide two simple solutions:

1) Let the robot turn 360° at each position and take the measurements simultaneously. Like this, the robot can obtain the same measurement results as the robot equipped with the sensors that are distributed uniformly around its circumference.
2) Using orientation sensors, such as the compass.

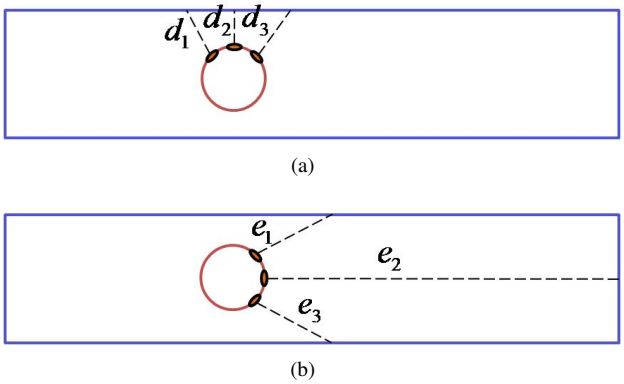


Fig. 2. A robot with non-uniformly distributed sensors measuring in a long and narrow room at different orientations. Energy of case (a) and case (b) is different, even if the robot is at the same location.

The process of calculating energy for grid cells is shown in Algorithm 2. 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 IV-B.

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

Algorithm 2: Calculating energy for each grid cell

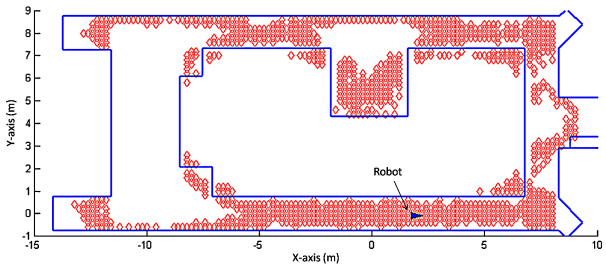
B. Calculating SER

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 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 3(b), the advantage of sampling in SER is more significant.

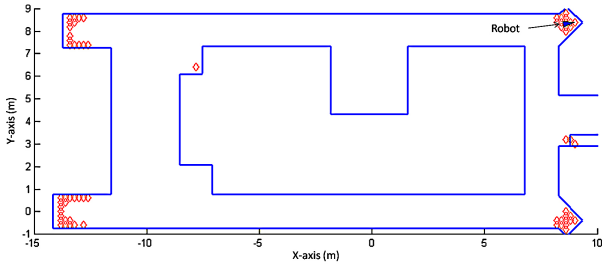
An algorithm to calculate SER is shown in Algorithm 3. It accepts as input the energy grid G_E obtained off-line in the pre-caching phase and the range measurements d_t of the robot at time t. It outputs SER. Lines 2 to 6 compute total energy of the I sensors for the real robot. Lines 7 to 11 compares total sensors energy of the real robot with total sensors energy of each grid cell. If the difference is smaller than a given threshold δ, we define this grid cell as a SER cell.

C. 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.



(a)



(b)

Fig. 3. SER when the robot is (a) in the corridor and (b) in the corner.

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

Algorithm 3: Calculating SER algorithm

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 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 4. It inputs the particle 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 particles 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.

V. 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.

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

Sampling total particles

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

Algorithm 4: SAMCL algorithm

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.

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.

$$\text{noise level } (\Lambda) = \frac{\text{standard deviation } (\sigma)}{\text{maximum range}} \times 100\% \qquad (18)$$

A. Position tracking

The purpose of this simulation is to evaluate the ability of the SAMCL algorithm to track the robot's position. We use a quasi-symmetrical corridor map and 300 particles. 6% perception noise and 8.82% motion noise are added in the sensor model and the motion model, respectively. Figure 4 and Figure 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 observe that SAMCL performs very well in position tracking.

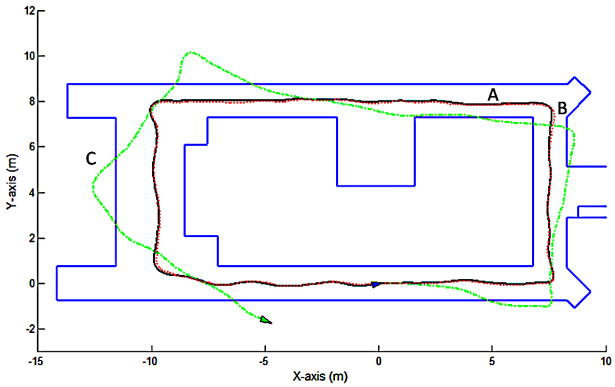


Fig. 4. Position tracking using SAMCL in a quasi-symmetrical corridor. The trajectories of robot, odometry and SAMCL are displayed by the black solid line (line A), the green dash-dot line (line C) and the red dotted line (line B), respectively.

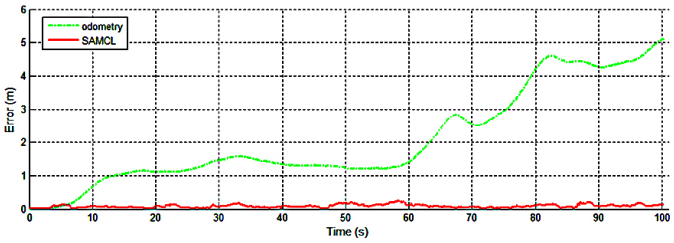


Fig. 5. Localization errors of position tracking using SAMCL in a quasi-symmetrical corridor. SAMCL errors and odometry errors are plotted by the red solid line and the green dash-dot line, respectively.

B. 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. In order to test the robustness of SAMCL, we add 6% perception noise and 8.82% motion noise to each wheel. Figure 6 shows the trajectories of robot, odometry and SAMCL and Figure 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 SAMCL has a good performance in global localization.

C. 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 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 particles, both

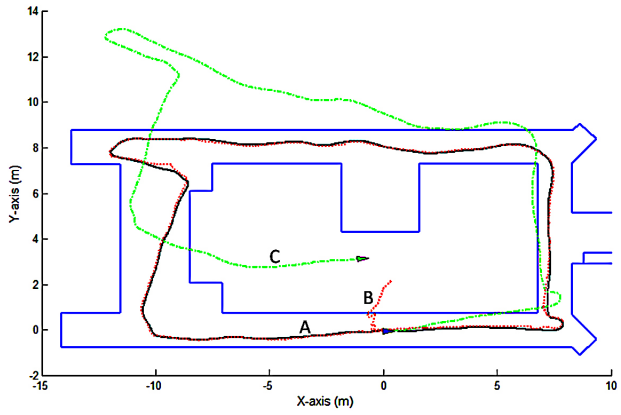


Fig. 6. Global localization using SAMCL in a quasi-symmetrical corridor. The trajectories of robot, odometry and SAMCL are displayed by the black solid line (line A), the green dash-dot line (line C) and the red dotted line (line B), respectively.

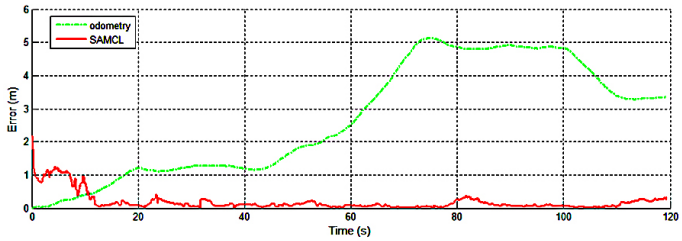


Fig. 7. Localization errors of global localization using SAMCL in a quasi-symmetrical corridor. SAMCL errors and odometry errors are plotted by the red solid line and the green dash-dot line, respectively.

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.

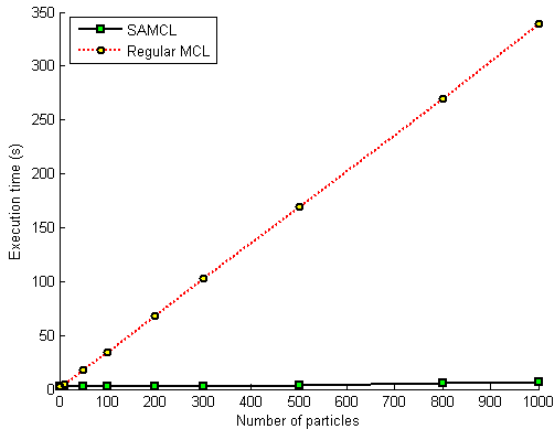


Fig. 8. Execution time of regular MCL and the SAMCL algorithm as a function of the number of particles.

D. 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.

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.

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 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.

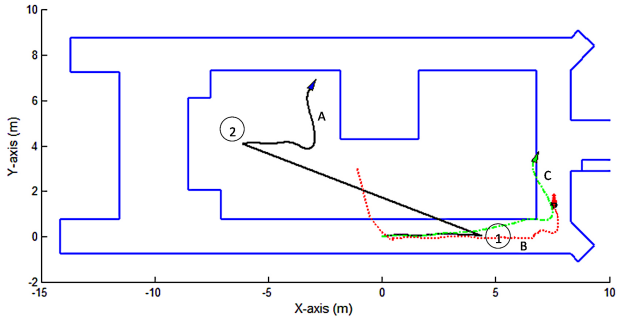


Fig. 9. MCL for robot kidnapping. The robot is kidnapped from the corridor to the room with known heading direction. The trajectories of robot, odometry and MCL are displayed by the black solid line (line A), the green dash-dot line (line C) and the red dotted line (line B), respectively.

The same simulation is executed by the SAMCL algorithm. As shown in Figure 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 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 to the room).

Figure 12 plots the localization error curves of SAMCL and odometry. Both SAMCL errors and odometry errors increase

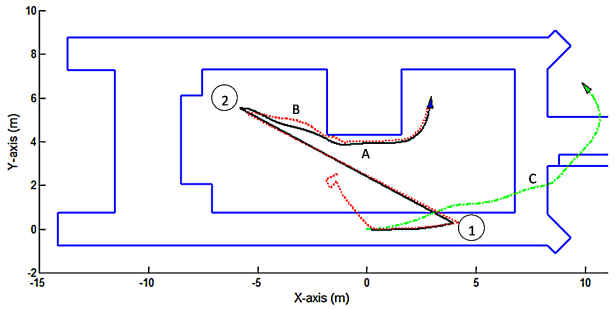
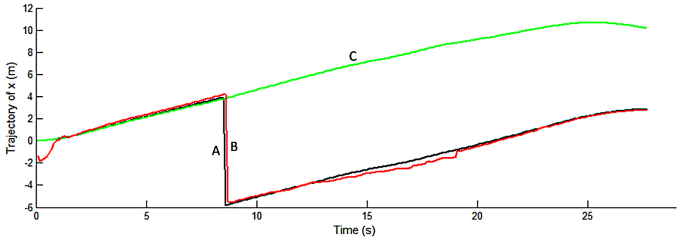
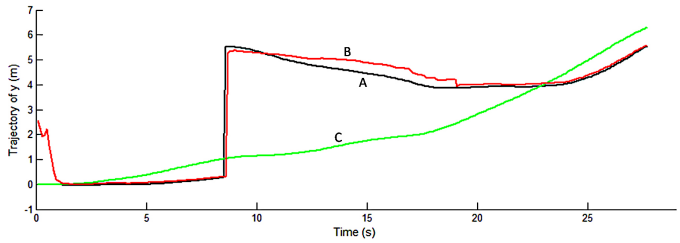


Fig. 10. SAMCL for robot kidnapping. The robot is kidnapped from the corridor to the room with known heading direction. The trajectories of robot, odometry and SAMCL are displayed by the black solid line (line A), the green dash-dot line (line C) and the red dotted line (line B), respectively.



(a)



(b)

Fig. 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.

suddenly when the robot is kidnapped, however SAMCL recovers in a flash.

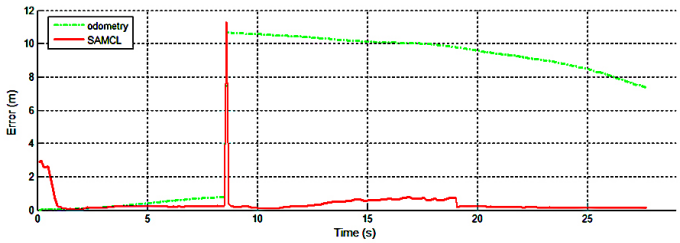


Fig. 12. Localization errors of SAMCL and odometry. The robot is kidnapped from the corridor to the room with known heading direction. SAMCL errors and odometry errors are plotted by the red solid line and the green dash-dot line, respectively.

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 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 14.

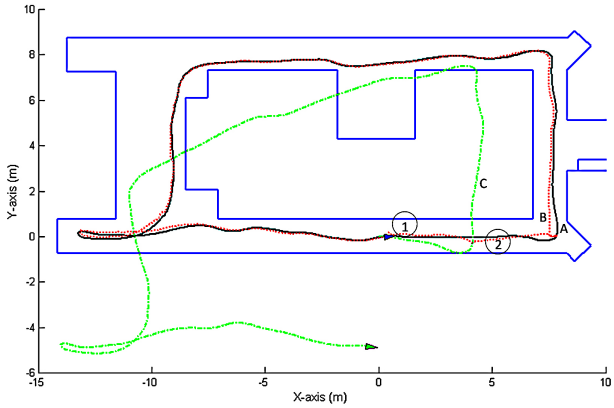


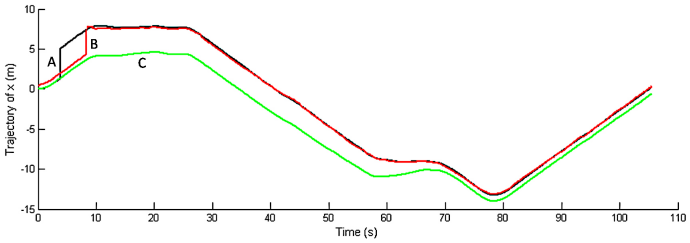
Fig. 13. SAMCL for robot kidnapping. The robot is kidnapped in the same corridor with known heading direction. The trajectories of robot, odometry and SAMCL are displayed by the black solid line (line A), the green dash-dot line (line C) and the red dotted line (line B), respectively.

In the Figure 14, trajectories of robot, SAMCL and odometry are decomposed into X-axis and Y-axis, respectively. From Figure 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 14(b)).

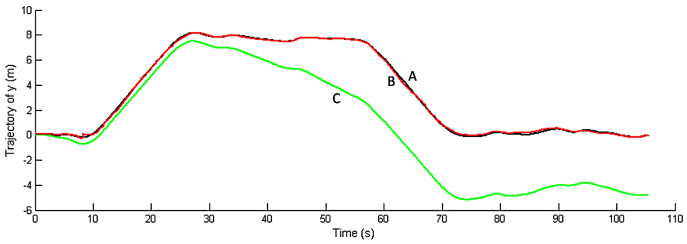
The localization error curves of SAMCL and odometry are shown in Figure 15. Sudden changes of error curves denote that kidnapping has happened. The SAMCL algorithm recovers from kidnapping with some delay but odometry loses itself totally.

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 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



(a)



(b)

Fig. 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.

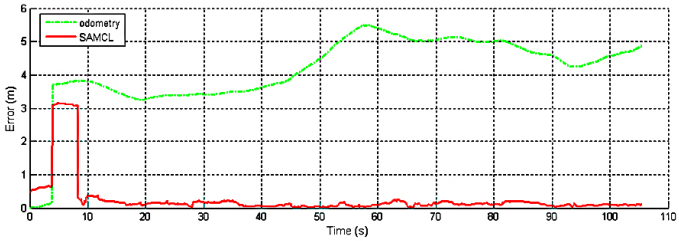


Fig. 15. Localization errors of SAMCL and odometry. Kidnapping occurs in the same corridor with known heading direction. SAMCL errors and odometry errors are plotted by the red solid line and the green dash-dot line, respectively.

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.

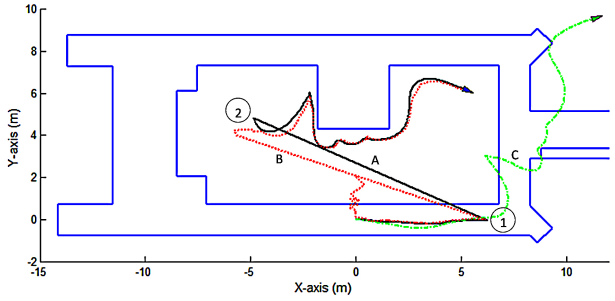
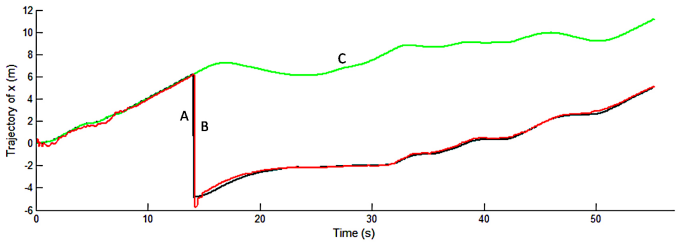
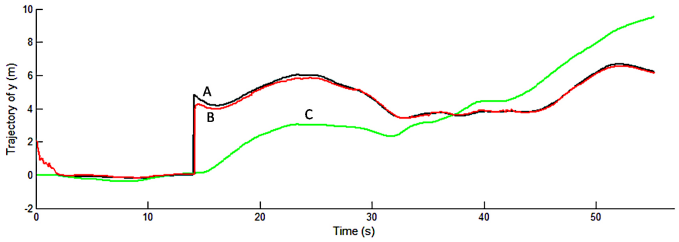


Fig. 16. SAMCL for robot kidnapping. The robot is kidnapped from the corridor to the room with unknown heading direction. The trajectories of robot, odometry and SAMCL are displayed by the black solid line (line A), the green dash-dot line (line C) and the red dotted line (line B), respectively.

After trajectories are decomposed into X-axis and Y-axis (as shown in Figure 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 $1s$.



(a)



(b)

Fig. 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 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.

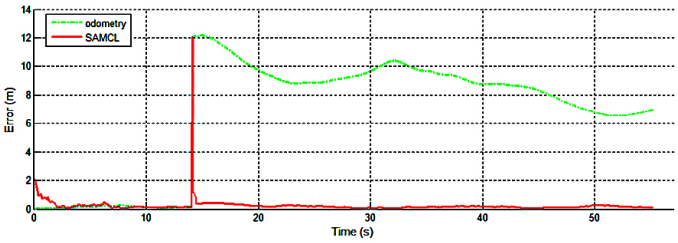


Fig. 18. Localization errors of SAMCL and odometry. The robot is kidnapped from the corridor to the room with unknown heading direction. SAMCL errors and odometry errors are plotted by the red solid line and the green dash-dot line, respectively.

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 Figure 3). Hence, to recover from kidnapping, the algorithm needs more samples (even more than Kidnapping in Section V-D3). This leads to an augmentation of computation and it is difficult to implement the algorithm in real-time.

VI. Experiments

The SAMCL algorithm described in this thesis has been tested with a Pioneer 3-DX mobile robot in a real office environment (see Figure 19).

Fig. 19. 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 20). 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.

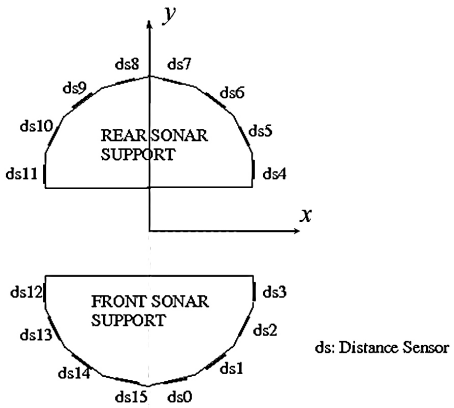


Fig. 20. Sonar locations on the Pioneer 3-DX robot, adapted from [45].

The experimental environment is the first floor of our laboratory. Its size is about $25m \times 10m$. Figure 21 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 21. 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 [7], 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.

Three experiments were performed, each of them examining

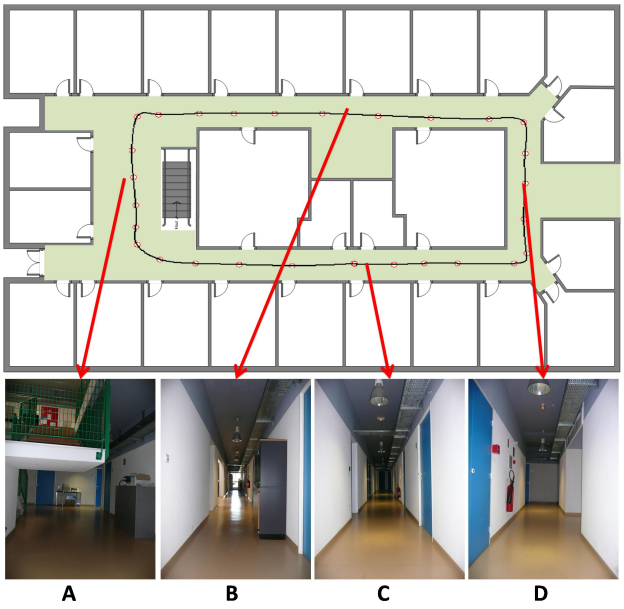


Fig. 21. The ground plan including the expected trajectory. Pictures show the real environment (with unmodeled obstacles).

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 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.

A. 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 22 shows localization results. As usual, the localization result is represented by the expected value (or called the weighed mean) of particles at each iteration. 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 21) than the odometry's trajectory. The trajectory given by odometry has a about $pi/4$ slope because of the initial orientation error.

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

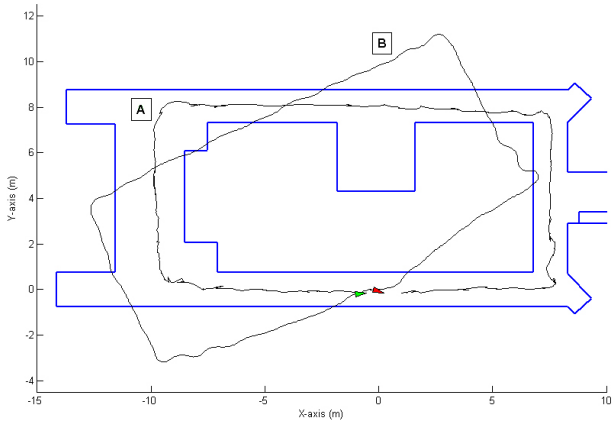


Fig. 22. Global localization using SAMCL. Line A and line B denote the trajectories of SAMCL and odometry, respectively.

TABLE I
AVERAGE ERRORS OF THE FINAL POSES IN GLOBAL LOCALIZATION

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

B. 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 23, 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.

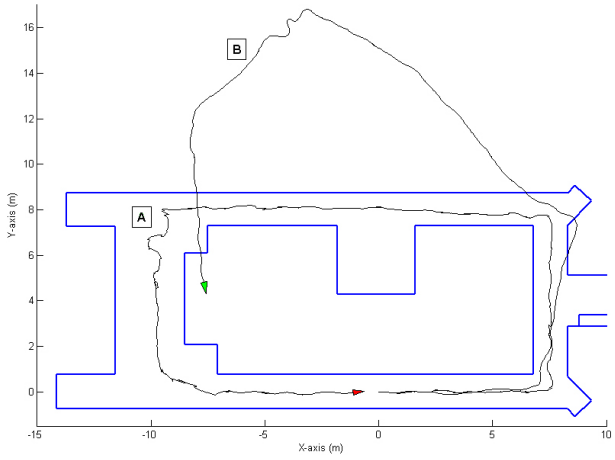


Fig. 23. Global localization using SAMCL with artificial errors. Line A and line B present the trajectories of SAMCL and odometry, respectively.

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

TABLE II
AVERAGE ERRORS OF THE FINAL POSES IN GLOBAL LOCALIZATION WITH ARTIFICIAL ERRORS

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

C. Kidnapping

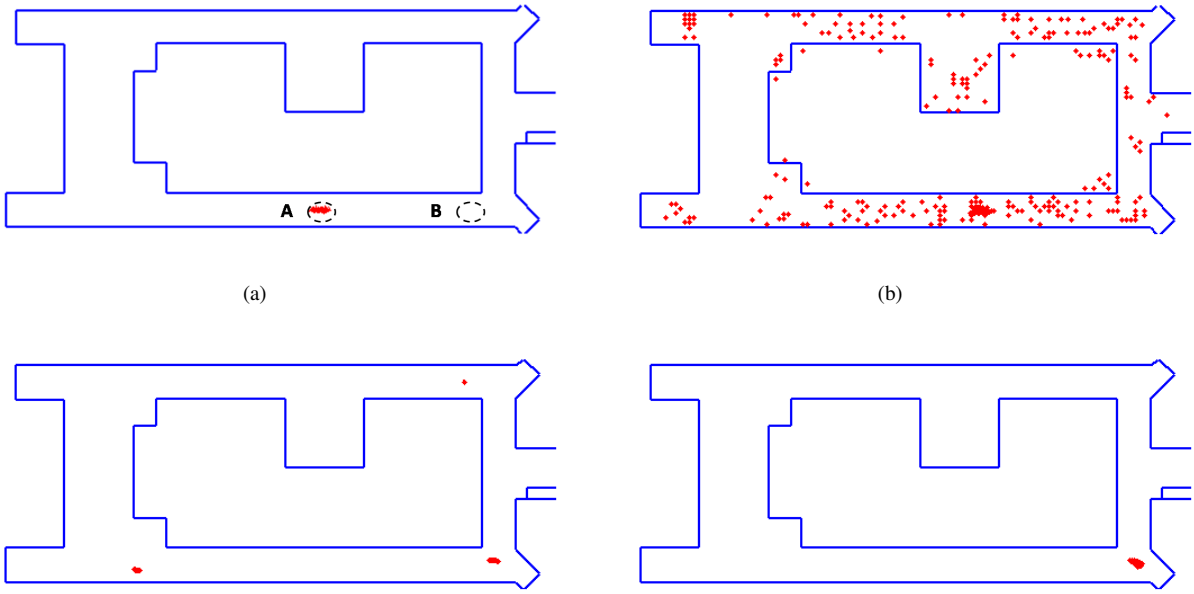
The third experiment demonstrates the ability of the SAMCL algorithm to recover from kidnapping, which is the most difficult issue. We kidnapped the Pioneer robot at the beginning of the trajectory after particles converging. Put differently, after the robot was well localized, we took it about $7m$ 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 24 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 24(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 24(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 24(c). The robot continues to move and perceive, SAMCL finally discards two probable poses and confirms the correct pose of robot, shown in Figure 24(d).

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 25, 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 III.

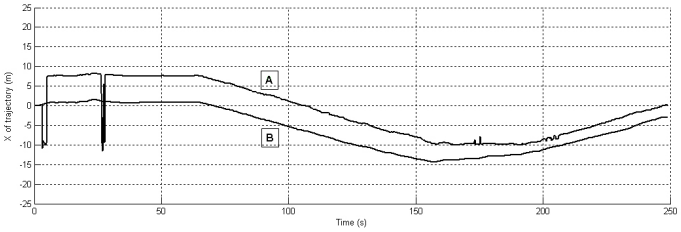
D. The success rate of recovering from kidnapping

We have 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

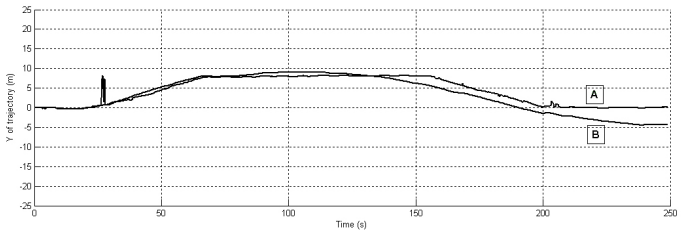


(a) (b) (c) (d)

Fig. 24. Distribution of the self-adaptive sample set during the process of recovering from kidnapping.



(a)



(b)

Fig. 25. 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 III
AVERAGE ERRORS OF THE FINAL POSES IN KIDNAPPING

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

simulation based on the experiment. Figure 26 shows the success rate of recovering from kidnapping as a function of the number of particles. The success rate is defined as:

$$success\ rate = \frac{the\ number\ of\ successful\ recoveries}{the\ total\ number\ of\ tests} \quad (19)$$

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%.

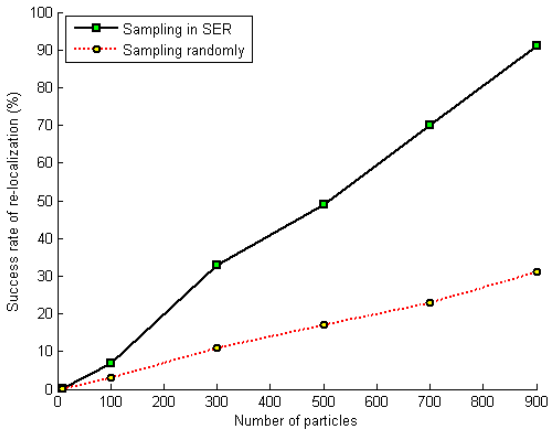


Fig. 26. The success rate of recovering from kidnapping as a function of the number of particles.

VII. Conclusions

In this paper, we proposed 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 IV.

TABLE IV
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.

The future work would address to the issue of applying the SAMCL algorithm in the multi-robot localization problem.

References

[1] W. Burgard, D. Fox, and S. Thrun, "Active mobile robot localization," in *Proceedings of IJCAI-97*. Morgan Kaufmann, 1997.

[2] D. Fox, W. Burgard, and S. Thrun, "Active markov localization for mobile robots," *Robotics and Autonomous Systems*, vol. 25, pp. 195–207, 1998.

[3] ——, "Markov localization for mobile robots in dynamic environments," *Journal of Artificial Intelligence Research*, vol. 11, pp. 391–427, 1999.

[4] I. J. Cox and G. T. Wilfong, Eds., *Autonomous robot vehicles*. Springer-Verlag New York, Inc., 1990.

[5] S. I. Roumeliotis and 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)*, vol. 3, 2000, pp. 2985–2992.

[6] S. Thrun, M. Beetz, M. Bennewitz, W. Burgard, A. Cremers, F. Dellaert, D. Fox, D. Hähnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz, "Probabilistic algorithms and the interactive museum tour-guide robot minerva," *International Journal of Robotics Research*, vol. 19, pp. 972–999, 2000.

[7] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. The MIT Press, September 2005.

[8] B. Schiele and J. L. Crowley, "A comparison of position estimation techniques using occupancy," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 2, 1994, pp. 1628–1634.

[9] G. Weiss, C. Wetzler, and E. von 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*, vol. 1, 1994, pp. 595–601.

[10] A. Milstein, J. N. Sánchez, and E. T. Williamson, "Robust global localization using clustered particle filtering," in *AAAI-02*, 2002, pp. 581–586.

[11] L. Jaulin, M. Kieffer, E. Walter, and 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, pp. 374–381, 2002.

[12] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust Monte Carlo localization for mobile robots," *Artificial Intelligence*, vol. 128, no. 1-2, pp. 99–141, 2000.

[13] J. Stéphant, A. Charara, and D. Meizel, "Virtual sensor, application to vehicle sideslip angle and transversal forces," *IEEE Transactions on Industrial Electronics*, vol. 51, no. 2, pp. 278–289, 2004.

[14] ——, "Evaluation of a sliding mode observer for vehicle sideslip angle," *Control Engineering Practice*, vol. 15, pp. 803–812, 2007.

[15] R. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[16] J. J. Leonard and H. F. Durrant-Whyte, "Mobile robot localization by tracking geometric beacons," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 376–382, 1991.

[17] M. S. Grewal and A. P. Andrews, *Kalman filtering: theory and practice*. Prentice-Hall, Inc., 1993.

[18] I. J. Cox and J. J. Leonard, "Modeling a dynamic environment using a bayesian multiple hypothesis approach," *Artificial Intelligence*, vol. 66, no. 2, pp. 311–344, 1994.

[19] J. Reuter, "Mobile robot self-localization using pdab," in *Proceedings of IEEE International Conference on Robotics and Automation ICRA '00*, vol. 4, 2000, pp. 3512–3518 vol.4.

[20] P. Jensfelt and S. Kristensen, "Active global localization for a mobile robot using multiple hypothesis tracking," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 5, pp. 748–760, 2001.

[21] J. Borenstein and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 278–288, 1991.

[22] ——, "Histogramic in-motion mapping for mobile robot obstacle avoidance," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 4, pp. 535–539, 1991.

[23] W. Burgard, D. Fox, D. Hennig, and T. Schmidt, "Estimating the absolute position of a mobile robot using position probability grids," in *Proceedings of the Thirteenth National Conference on Artificial Intelligence, Menlo Park*. AAAI Press/MIT Press, 1996, pp. 896–901.

[24] W. Burgard, D. Fox, and D. Hennig, "Fast grid-based position tracking for mobile robots," in *Proceedings of the 21th German Conference on Artificial Intelligence*. Springer Verlag, 1997, pp. 289–300.

[25] N. Metropolis and S. Ulam, "The Monte Carlo method," *Journal of the American Statistical Association*, vol. 44, no. 247, pp. 335–341, 1949.

[26] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte Carlo localization for mobile robots," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 2, 1999, pp. 1322–1328.

[27] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte Carlo localization: Efficient position estimation for mobile robots," in *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI'99)*, July 1999, pp. 343–349.

[28] S. Thrun, J. C. Langford, and 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*. Morgan Kaufmann Publishers Inc., 1999, pp. 415–424.

[29] D. Fox, W. Burgard, H. Kruppa, and S. Thrun, "Efficient multi-robot localization based on Monte Carlo approximation," in *Robotics Research: the Ninth International Symposium*, J. Hollerbach and D. Koditschek, Eds. London: Springer-Verlag, 2000.

[30] S. Thrun, "Monte Carlo POMDPs," in *Advances in Neural Information Processing 12*, 2000, pp. 1064–1070.

[31] C. Kwok, D. Fox, and M. Meila, "Real-time particle filters," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 469–484, 2004.

[32] C. Siagian and 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*, 2007, pp. 1723–1730.

[33] T. Hester and P. Stone, "Negative information and line observations for Monte Carlo localization," in *Proceedings of IEEE International Conference on Robotics and Automation ICRA 2008*, 2008, pp. 2764–2769.

[34] E. Prestes, M. Ritt, and 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*, 2008, pp. 3465–3470.

[35] S. Thrun, D. Fox, and W. Burgard, "Monte Carlo localization with mixture proposal distribution," in *Proceedings of the AAAI National Conference on Artificial Intelligence*, 2000, pp. 859–865.

[36] D. Fox, "Adapting the sample size in particle filters through kld-sampling," *International Journal of Robotics Research*, vol. 22, no. 12, pp. 985–1003, 2003.

[37] D. Fox, J. Hightower, H. Kauz, L. Liao, and D. Patterson, "Bayesian techniques for location estimation," in *Proceedings of The 2003 Workshop on Location-Aware Computing*, 2003, pp. 16–18.

[38] V. Fox, J. Hightower, L. Liao, D. Schulz, and G. Borriello, "Bayesian filtering for location estimation," *IEEE Pervasive Computing*, vol. 2, no. 3, pp. 24–33, 2003.

[39] A. Bekkali and 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*, 2008, pp. 285–292.

[40] J. L. Blanco, J. Gonzalez, and 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*, 2008, pp. 461–466.

[41] J. Ko and 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*, 2008, pp. 3471–3476.

[42] D. Fox, "Kld-sampling: Adaptive particle filters," in *Advances in Neural Information Processing Systems 14*. MIT Press, 2001, pp. 713–720.

[43] A. F. M. Smith and A. E. Gelfand, "Bayesian statistics without tears: A sampling-resampling perspective," *The American Statistician*, vol. 46, no. 2, pp. 84–88, 1992.

[44] A. Doucet, S. Godsill, and C. Andrieu, "On sequential Monte Carlo sampling methods for bayesian filtering," *STATISTICS AND COMPUTING*, vol. 10, no. 3, pp. 197–208, 2000.

[45] *http://www.cyberbotics.com/cdrom/common/doc/webots/guide/section7.4.html*, Cyberbotics Ltd.