A New Compensation Technique Based on Analysis of Resampling Process in FastSLAM

Nosan Kwak *
School of Electrical Engineering
Seoul National University
robot97@snu.ac.kr

Gon-Woo Kim †
Applied Robot Tech.
KITECH
kgw0510@kitec.re.kr

Beom-Hee Lee ‡
School of Electrical Engineering
Seoul National University
bhlee@snu.ac.kr

Abstract

The state-of-the-art FastSLAM algorithm has been shown to a cause particle depletion problem while performing simultaneous localization and mapping for mobile robots. As a result, it always produces over-confident estimates of uncertainty as time progresses. This particle depletion problem is mainly due to the resampling process in FastSLAM, which tends to eliminate particles with low weights. Therefore, the number of particles to conduct loop-closure decreases, which makes the performance of FastSLAM degenerate. The resampling process has not been thoroughly analyzed even though it is the main reason for the particle depletion problem. In this paper, standard resampling algorithms (systematic residual and partial resampling), a rank-based resampling adopting genetic algorithms are analyzed using computer simulations. Several performance measures such as the effective sample size, the number of distinct particles, estimate errors, and complexity are used for the thorough analysis of the resampling algorithms. Moreover, a new compensation technique is proposed instead of resampling to resolve the particle depletion problem in FastSLAM. In estimate errors, the compensation technique outperformed other resampling algorithms though its run-time was longer than those of others. The most appropriate

*Ph. D. Student
†Ph. D.
‡Professor, *Fellow, IEEE*

time to instigate compensation to reduce the run-time was also analyzed with diminishing the number of particles.

keywords: FastSLAM, Particle filter, Resampling, Particle depletion, Compensation, Mobile robot

1 Introduction

The simultaneous localization and mapping (SLAM) is a fundamental problem found in mobile robots to perform autonomous tasks, such as exploration and navigation in an unknown environment. SLAM can be applied in a wide range of tasks from an indoor environment[1] to an underwater environment.[2] The SLAM problem occurs when a robot knows neither a map of an environment nor its own pose in the environment. The problem is not so simple because a robot needs to obtain a map of its environment and at the same time localize itself to the map. Moreover, the robot has to make relative observations to its ego-motion and objects in its environment, which are both corrupted by noise.[3, 4] Therefore, environment modeling and localization should be considered as a dual problem.[5]

Many approaches have been centered on the stochastic formulation of the SLAM problem in which landmark estimates and a robot pose are obtained as probability distributions. In this point of view, the two key computational solutions to the SLAM problem are using the extended Kalman filter for SLAM (EKF-SLAM) and using the Rao-Blackwellized particle filters for SLAM (FastSLAM), respectively. EKF-SLAM has served as the main approach to the SLAM problem for the last fifteen years. However, EKF-SLAM is known to have two major well-known shortcomings: quadratic computational complexity and sensitivity to failures in data association. First, since observation-update step requires that all landmarks and the joint covariance matrix be updated at every observation,

computation of EKF-SLAM quadratically grows with the number of landmarks.[6] Second, it is especially difficult to deal with a loop-closure problem, which occurs when a robot returns to observe landmarks again after a large traverse. These shortcomings consequently make it difficult to apply EKF-SLAM to real and large environments. Recently, the FastSLAM algorithm[3] has been proposed as an alternative solution to the SLAM problem. It uses a particle filter instead of the Kalman filter to approximate the ideal recursive Bayesian filter. FastSLAM is an instance of the Rao-Blackwellized particle filter (RBPF)[7], which factors the full SLAM posteriors exactly into a product of a robot path posterior and landmark posteriors, conditioned on the robot path estimate.[8] FastSLAM has two significant advantages over EKF-SLAM. First, by factoring the full SLAM posteriors FastSLAM has linear time-complexity. Second, unlike EKF-SLAM, FastSLAM allows each particle to perform its own data association, which implements multi-hypothesis data association.[9] The ability to simultaneously pursue multiple data associations makes FastSLAM significantly more robust to data association problems than algorithms based on incremental maximum likelihood data association such as EKF-SLAM.[10]

FastSLAM, however, has some drawbacks as well. In the literatures,[3, 11] FastSLAM has been noted to degenerate over time. It is impossible to prevent this degeneracy, as proven by Kong-Liu-Wong theorem.[12] This degeneracy causes particles estimating the pose of a robot to lose their diversity, the so called particle depletion. For example, better diversity of particles results in better loop-closure performance because new observations can affect the pose of the robot.[10] The main reason for losing diversity of particles is due to the resampling process in FastSLAM. By throwing away improbable paths of the robot, resampling eventually causes all of the particles to share a common history at some point in the past, and then new observations cannot affect the locations

of landmarks observed prior to this point.[10] Thus, keeping diversity of particles in FastSLAM is very important for reliable loop-closure and consistent map building. When a particle set loses its diversity, it tends to underestimate its own uncertainty. As a result, it prevents from building a consistent map.[11]

Bailey *et al.*[11] studied about the consistency of FastSLAM, which is an ability of a filter to accurately estimate uncertainty. It showed that in the short-term, FastSLAM may produce consistent uncertainty estimates, but in the long-term, it will degenerate with time, regardless of the number of particles and the density of landmarks within an environment. It also mentioned that FastSLAM always produces optimistic estimates of uncertainty. Consequently, FastSLAM is unable to adequately explore state space to be a reasonable Bayesian estimator. However, it showed that the algorithm is practically capable of generating an accurate map in real outdoor environments.[13] They also researched how fast the diversity is lost, what the effect of loss of diversity has on the filter's uncertainty estimate, and how parameters such as the number of particles and landmark density affect estimation errors.[11] Grisetti *et al.* provided a compact map model, which can share large parts of the model of the environment.[14] However, they barely dealt with effects of resampling, which is the main reason for the particle depletion problem. To limit the particle depletion many algorithms have been proposed in the field of particle filters.[15–17] Robot paths, for example, are artificially perturbed after resampling, which resulted in enlarging the variance of importance weights of particles.[18] Grisetti *et al.* presented an approach to selectively carry out resampling to reduce particle depletion.[19] Higuchi proposed various heuristic procedures taken from the genetic algorithms (GA) to introduce such a diversity among particles.[20]

In this paper, two things are mainly researched to alleviate the particle depletion problem. The

first one is a thorough analysis of resampling algorithms, and the other is a new compensation technique to resolve the particle depletion problem.

For the analysis, standard resampling algorithms are selected among several algorithms, which are residual systematic resampling (RSR)[21] and partial resampling (PR).[22] In addition, a rank-based resampling (RBR)[23] which is a sampling algorithm in GA is adopted as a resampling algorithm especially for the particle depletion problem. These resampling algorithms are analyzed to clarify their effects on SLAM performance. Resampling algorithms have not been thoroughly analyzed even though they are the main reason for the particle depletion problem. In this work, the effective sample size, the number of distinct particles, estimate errors, and complexity are used for the thorough analysis.

More importantly, the second research is that a compensation technique instead of resampling is proposed to prevent particles from being depleted. It probabilistically compensates particles with low weights using particles with high weights, so no particle is rejected in a particle set. Also, for alleviating computational cost of the compensation algorithm, scheduling of compensation is studied to decide the time when it is most appropriate to instigate compensation using the effective sample size. Deciding the most appropriate time to instigate resampling is still a problem to be resolved.[24] To decide the time, several simulations for scheduling of compensation are performed with varying the number of particles.

This paper is organized as follows: In Section 2, FastSLAM 2.0, which is superior to FastSLAM 1.0 in nearly all aspects, is briefly illustrated with a graphical procedure. General resampling algorithms for FastSLAM are introduced in Section 3, and they are thoroughly analyzed with computer simulations, followed by discussions in Section 4. In Section 5, a compensation technique is proposed

to prevent particles from being depleted, followed by discussions about its performance. Finally, conclusion is presented in Section 6.

2 FastSLAM Algorithm

The FastSLAM's key mathematical insight pertains to the fact that the full SLAM posterior can be factored as follows when the correspondences $c_{1:t} = c_1, ..., c_t$ are known:[10]

$$p(x_{1:t}, \theta | z_{1:t}, u_{1:t}, c_{1:t}) = p(x_{1:t} | z_{1:t}, u_{1:t}, c_{1:t}) \prod_{n=1}^{N} p(\theta_n | x_{1:t}, z_{1:t}, c_{1:t}) \tag{1}$$

where $x_{1:t}$ is the robot path from start till time t, θ is the map, and $z_{1:t}$ and $u_{1:t}$ are the measurements and controls till time t, respectively. N is the number of features. FastSLAM uses a particle filter to compute the posterior over robot paths, denoted by $p(x_{1:t} | z_{1:t}, u_{1:t}, c_{1:t})$. For each feature in the map, FastSLAM uses a separate estimator over its location $p(\theta_n | x_{1:t}, z_{1:t}, c_{1:t})$, one for each feature. The feature estimators are conditioned on the robot path, which means there is a separate copy of each feature estimator, one for each particle. More precisely, map feature locations are estimated using EKFs. Due to the factorization, FastSLAM can maintain a separate EKF estimator for each feature, which makes the update more efficient than that in EKF-SLAM. By independently keeping the feature estimates FastSLAM avoids the quadratic cost of computing a joint map covariance matrix. However, the dependency on the robot path is the key weakness of FastSLAM, which means the implicit dimension of the state-space increases with time.[10]

A particle at time t, $Y_t^{[k]}$ in FastSLAM is denoted by

$$Y_t^{[k]} = \langle x_t^{[k]}, \mu_{1,t}^{[k]}, \Sigma_{1,t}^{[k]}, ..., \mu_{N,t}^{[k]}, \Sigma_{N,t}^{[k]} \rangle \tag{2}$$

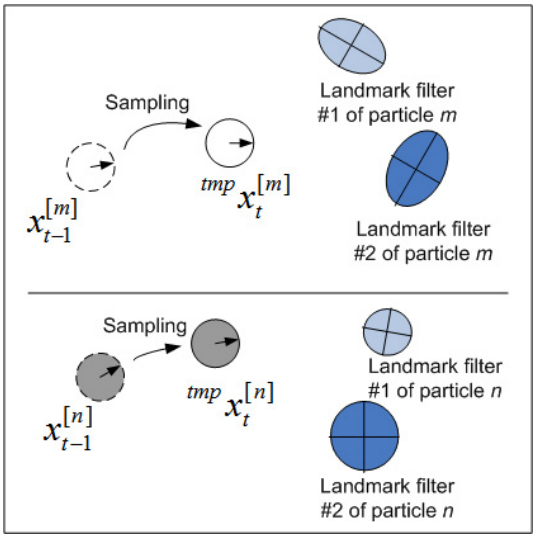
where the $[k]$ indicates the index of the particle, and $x_t^{[k]}$ is the robot pose estimate of the k-th

particle at time t. Only the most recent pose $x_t^{[k]}$ is used in the FastSLAM, so a particle keeps only the most recent pose. $\mu_{n,t}^{[k]}, \Sigma_{n,t}^{[k]}$ are mean and covariance of the Gaussian distribution, representing the n-th feature location relative to the k-th particle, respectively. Altogether, these elements form the k-th particle, $Y_t^{[k]}$, and there are totally M particles and N features in a particle set. The basic steps of the FastSLAM 2.0 algorithm are as follows:[10]

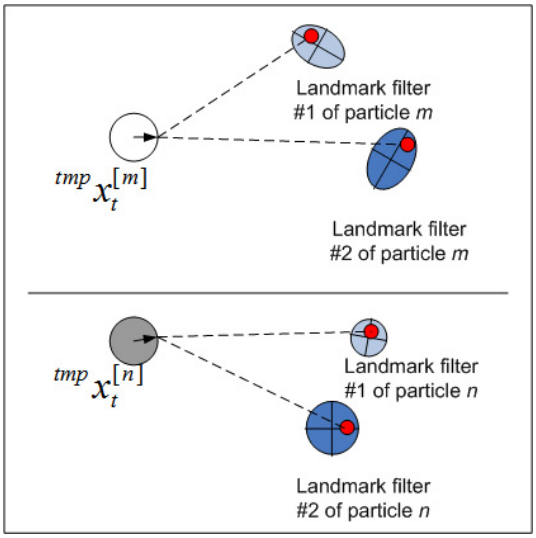
- Step1: Sampling. $x_t^{[k]} \sim p(x_t | x_{t-1}^{[k]}, z_{1:t}, u_{1:t}, c_{1:t})$
- Step2: Measurement update. For each observed feature identify the correspondence j for the measurement z_t^i, and incorporate the measurement z_t^i into the corresponding EKF by updating the mean $\mu_{j,t}^{[k]}$ and covariance $\Sigma_{j,t}^{[k]}$.
- Step3: Importance weight. Calculate the importance weight $w^{[k]}$ for k-th particle.
- Step4: Resampling. Sample M particles with replacement, where each particle is sampled with a probability proportional to $w^{[k]}$.

A simple graphical procedure of the FastSLAM 2.0 algorithm is described in Fig. 1. For convenience, only two particles among in a particle set are shown while performing FastSLAM. In Fig. 1(a), each particle in the particle set at time $(t-1)$ samples a pose using the proposal distribution which takes the measurement z_t into account. All of the sampled poses constitute a temporary particle set. Then, each particle updates the posterior over the feature estimates, based on the measurement z_t and the sampled pose ${}^{tmp}x_t^{[k]}$ as shown in Fig. 1(b). As a result, in this case, the covariances of feature estimates are reduced. The next step is to compute the importance weight for k-th particle using the following quotient:

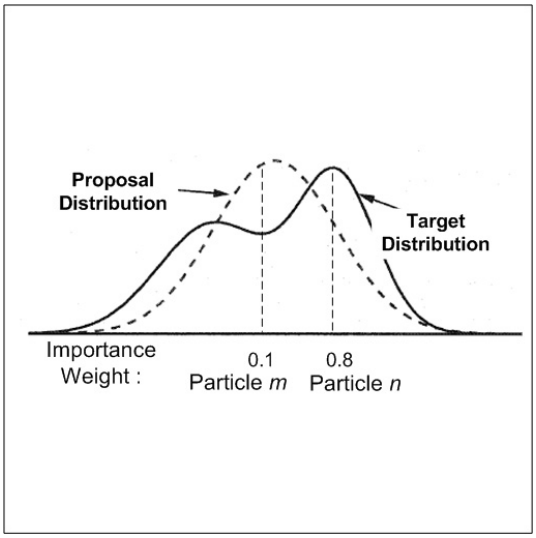
$$w_t^{[k]} = \frac{\text{target distribution}}{\text{proposal distribution}}$$



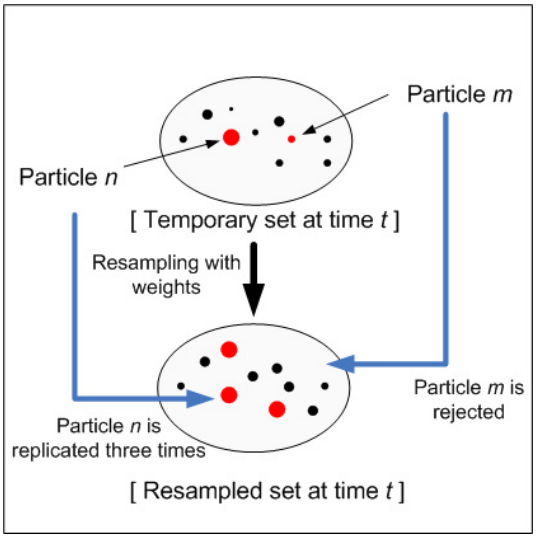
(a) Sampling



(b) Measurement Update



(c) Importance weight



(d) Resampling

Fig. 1 A graphical procedure of FastSLAM 2.0 Algorithm. (a) Pose sampling of two particles: ${}^{tmp}x_t^{[m]}$ and ${}^{tmp}x_t^{[n]}$ using control input and current measurements where the superscript tmp means a particle is included in a temporary particle set, (b) measurement update, which is performed per particle, (c) importance weight, and (d) resampling; particles are replicated or rejected in the set.

Since it is usually impossible to sample from the true posterior (target distribution), it is common to sample from an easy-to-implement distribution, the so-called proposal distribution. As shown in Fig. 1(c), in regions where the target distribution is larger than the proposal distribution, the particles receive high weights, and in regions where target distribution is smaller than the proposal distribution, the particles are given low weights. As a result, particle n received 0.8 whereas particle m received 0.1 as importance weights. The last process is resampling, which draws M particles with replacement from the temporary particle set. As shown in Fig. 1(d), the temporary particle with high importance weight, ${}^{tmp}x_t^{[n]}$ is replicated three times, whereas the one with low importance weight, ${}^{tmp}x_t^{[m]}$ is rejected or thrown away in the particle set by the resampling process. This means the robot path and feature locations estimated by the rejected particle are lost. Thus, the resampling is a crucial process in the aspect of diversity of particles.

3 General Resampling Process for FastSLAM

The basic idea of the resampling process is to eliminate the particles that have low importance weights and replicate particles with high importance weights. Dependency on the robot path is recorded in the weight of a particle and in its feature estimates. As time progresses, errors of feature estimates become larger unless particles are resampled. Thus, the necessity of the resampling process arises from the fact that the particles in a temporary set do not yet match the target distribution. Resampling can avoid the particle degeneracy by weighing particles, followed by replacing particles with the importance weights. As a result, the resulting particle set indeed approximates the target distribution. This has been shown to allow consistent recursive estimation with the fixed number of particles.[25]

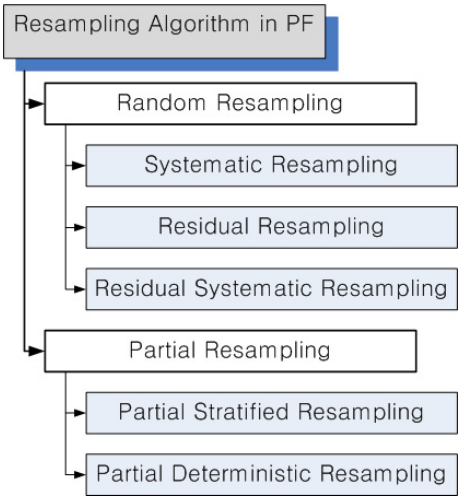


Fig. 2 Categorization of resampling algorithms in particle filters (PF).

The innate disadvantage of FastSLAM is that past pose estimate errors are not forgotten, which means they are recorded in the feature estimates. Whenever the resampling process is conducted, the entire robot paths and feature estimates of rejected particles are lost forever. Consequently, the number of particles representing past paths and feature estimates decreases. This is called the particle depletion problem.[16] In other words, the particle set loses its diversity, and it becomes over-confident as resampling is repeated over and over again. Bailey *et al.* studied consistency of a particle filter in FastSLAM, and confirmed that FastSLAM can not produce consistent estimates in the long-term.[11] However, they did not deal with various resampling algorithms, which are critical to the particle depletion problem.

In fact, the resampling algorithms have been researched in the field of particle filters (PF)[21], which are categorized in Fig. 2. Standard algorithms used for random resampling are different variants of stratified resampling such as residual resampling (RR) and systematic resampling (SR). The SR[22] is the most commonly used since it is a fast resampling algorithm for computer simulations. The

partial resampling (PR)[22] is to perform resampling only on particles with high weights and replace particles having negligible weights with them. In the PF, particles with moderate weights are not resampled. In this work, residual systematic resampling (RSR) and partial stratified resampling (PSR) among general resampling algorithms are selected for performance analysis because the RSR produces the identical resampling result as the SR with less operations and less memory access,[21] and the PSR can adjust the number of particles participating in resampling to alleviate particle depletion. In addition, a rank-based selection in genetic algorithms (GA) is newly proposed in this work as a resampling method to keep particle diversity as long as possible.

The resampling process is crucial in every implementation of particle filtering because without it the variance of the particles' weights quickly increases. Therefore, in this section, general resampling algorithms will be analyzed in the aspect of SLAM using several performance measures such as number of distinct particles, the effective sample size, complexity, and estimate errors. After that, their limitations will be shown in Section 4. Furthermore, a new compensation technique will be suggested for resolving the limitations in Section 5.

3.1 Residual Systematic Resampling (RSR)

Standard algorithms used for random resampling are different variants of stratified sampling such as residual resampling (RR) and systematic resampling (SR). The SR[22] is the most commonly used since it is the fastest resampling algorithm for computer simulations. The RSR[21] proposed by Bolić *el al.* produces the identical resampling result as the SR with less operations and less memory access. Thus, in this section, the SR is first introduced, followed by the RSR.

The SR performs resampling in the same way as a basic random resampling algorithm with one

Table I Systematic resampling algorithm.

$(nReplicated) = \mathbf{SR}(L, M, w)$

Generate a random number, $U \sim \mathbf{U}([\mathbf{0}, \frac{\mathbf{1}}{\mathbf{M}}])$

$s = 0$

for $k = 1$ to L

    $i = 0$

    $s = s + w^{[k]}$

    while$(s > U)$

        $i = i + 1$

        $U = U + \frac{1}{M}$

    end

    $nReplicated(k) = i$

end

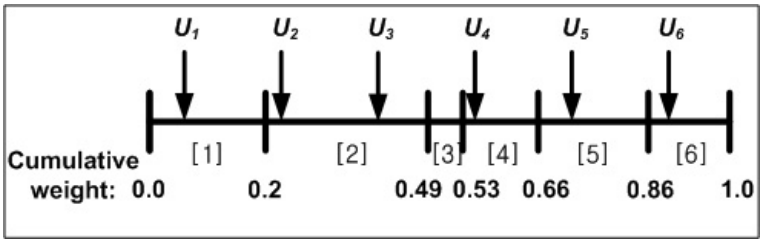


Fig. 3 Systematic resampling example. The 2nd particle, [2] is replicated twice, and the 3rd particle, [3] is thrown away, so the number of distinct particles is reduced from 6 to 5 after the resampling.

exception. Instead of independently drawing each uniform random number, U_k from $\mathbf{U}([\mathbf{0},\mathbf{1}])$ for $k = 1, ..., M$, where $\mathbf{U}([\mathbf{0},\mathbf{1}])$ denotes the uniform distribution on the interval $[0, 1]$, and M is the number of resampled particles, it draws U according to $U \sim \mathbf{U}([\mathbf{0}, \frac{\mathbf{1}}{\mathbf{M}}])$ and updates the uniform random number by $U_k = U + (k-1)/M$. The SR algorithm is shown in Table I. In the table, L is the input number of particles, and M is the number of particles generated after resampling, and w is an array of scaled weights of particles. The output *nReplicated* is an array of indices, which means how many times each particle is replicated. For example, assume that there are six particles and six new particles should be resampled with their scaled importance weights: $w^{[1]} = 0.2$, $w^{[2]} = 0.29$, $w^{[3]} = 0.04$, $w^{[4]} = 0.13$, $w^{[5]} = 0.2$, and $w^{[6]} = 0.14$. Their cumulative weights are shown in Fig. 3. According to the SR algorithm, the first uniform random number U_1 is drawn from the uniform distribution, $\mathbf{U}([\mathbf{0}, \frac{\mathbf{1}}{\mathbf{6}}])$, let say, $U_1 = 0.06$. Then, $U_2 = U_1 + \frac{1}{6} = 0.23, U_3 = 0.39, U_4 = 0.56, U_5 = 0.73$, and $U_6 = 0.89$. After the SR, the new particle set consists of {1,2,2,4,5,6}. The second particle is replicated twice and the third particle is rejected in the new particle set. The advantages of the SR are that first, particles are drawn from almost all intervals, and second, it can perform resampling relatively fast.

In the same way, the RSR algorithm draws the first uniform random number, $U_0 = \Delta U_0$ but

Table II Residual systematic resampling algorithm.

$(nReplicated) = \mathbf{RSR}(L, M, w)$

Generate a random number, $\Delta U_0 \sim \mathbf{U}([\mathbf{0}, \frac{\mathbf{1}}{\mathbf{M}}])$

for $k = 1$ to L

$nReplicated(k) = [(w^{[k]} - \Delta U_{k-1}) \cdot M] + 1$

$\Delta U_k = \Delta U_{k-1} + nReplicated(k)/M - w^{[k]}$

end

updates it by $\Delta U_k = \Delta U_{k-1} + nReplicated(k)/M - w^{[k]}$. The RSR algorithm is described in Table II. In the RSR algorithm, the updated uniform random number is formed in a different fashion, that is, it requires only one iteration loop. In addition, the RSR performs resampling in a fixed time whereas the SR does not. The reason is that the SR replicates random number of particles, which makes an unspecified number of operations.

3.2 Partial Resampling (PR)

The main objective of the PR[22] is to perform resampling only on particles with high weights and replace particles having negligible weights with them. Particles with moderate weights are not resampled. In the PR, the resampling process is preceded by grouping the particles based on their weights. The weight of each particle is compared with a high and a low thresholds, T_h and T_l, respectively. Particles with weights between those two thresholds are considered moderate and are not resampled. Let the number of particles with weights greater than T_h and less than T_l be denoted by M_h and M_l, respectively. A sum of weights, S_{hl} of the particles that are resampled is computed by $S_{hl} = \sum_{j=1}^{M_h+M_l} w^{[j]}$, where j is chosen so that the conditions $w^{[j]} > T_h$ or $w^{[j]} < T_l$ can be satisfied.

Then, one of stratified resampling algorithms such as the SR is conducted only on particles whose weights satisfy the threshold constraints.

The PR can perform resampling fast because it resamples much smaller number of particles. Also, it preserves moderate particles by not participating them in the resampling process. Moreover, the PR can control the thresholds either to keep particle diversity or reduce the degeneracy.

3.3 Rank-based Resampling (RBR)

The RBR is adopted as a resampling algorithm from the rank-based selection in GA.[23, 26] This algorithm is newly proposed in this work to keep particle diversity as long as possible. The RBR determines the rank of each particle by its weight and then, gives it a selection probability. With the selection probabilities, the RBR performs one of the stratified resampling algorithms. In this work, the following linear equation is used to give the selection probability, $p(k)$ of k-th particle based on its rank:

$$p(k) = \frac{1}{M}\left[\eta_{max} - (\eta_{max} - \eta_{min})\frac{(rank(k) - 1)}{M - 1}\right] \tag{3}$$

where η_{max}/M is the maximum selection probability of the highest weight, and η_{min}/M is the minimum selection probability of the lowest weight. The particle at the first rank gets the highest selection probability whereas the particle at the last rank gets the lowest. When the number of particles is fixed, $\eta_{min} = 2 - \eta_{max} \geq 0$ should be satisfied, and η_{max} usually has a value between 1 and 2. The resampling performance varies dependent on the η_{max}. Relation between selection probabilities and ranks of particles with changing η_{max} is plotted in Fig. 4 using the example in Section 3.1. The RBR is called an indirect resampling algorithm since it neglects the relative information among the weights and assigns selection probabilities based on their ranks.

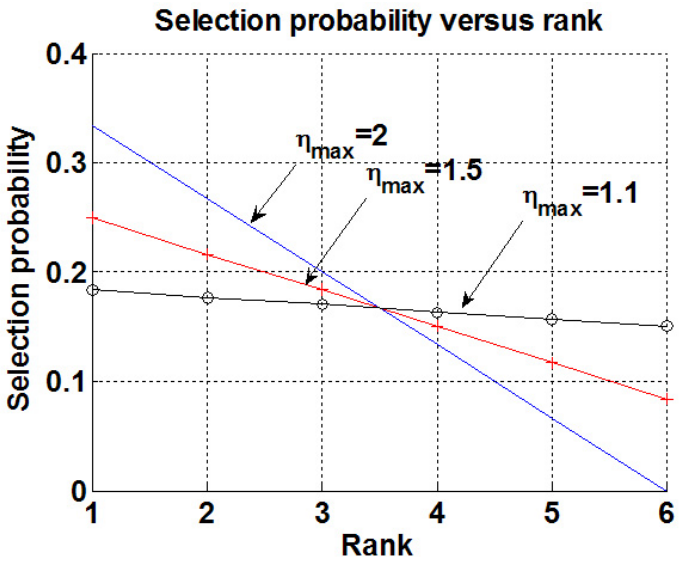


Fig. 4 Selection probabilities versus ranks of particles with changing η_{max}.

4 Analysis of Resampling Process

In this section, performance of each resampling algorithm in the previous section is analyzed using computer simulations. The effective sample size, the number of distinct particles, estimate errors, and complexity are used for the thorough analysis. Simulations were performed on the sparse indoor environment shown in Fig. 5. Simulations on a sparse environment are mainly analyzed because a dense map is not realistic; an environment rarely has a dense map in an indoor environment. Simulations were run with 100 particles because the effect of increasing the number of particles more than that is not significant.[11] In Fig. 5, the point **A** is where the feature error significantly increases, and the points **B** and **C** are inner and outer loop-closure points, respectively. The weights of all the particles are initialized with the same weight after resampling. The motion noise and the observation noise of the robot were set to $(0.3m/s, 3°/s)$ and $(0.1m, 1°)$, respectively. Control time and observation time were set to $25ms$ and $200ms$, respectively. Every result averaged 50

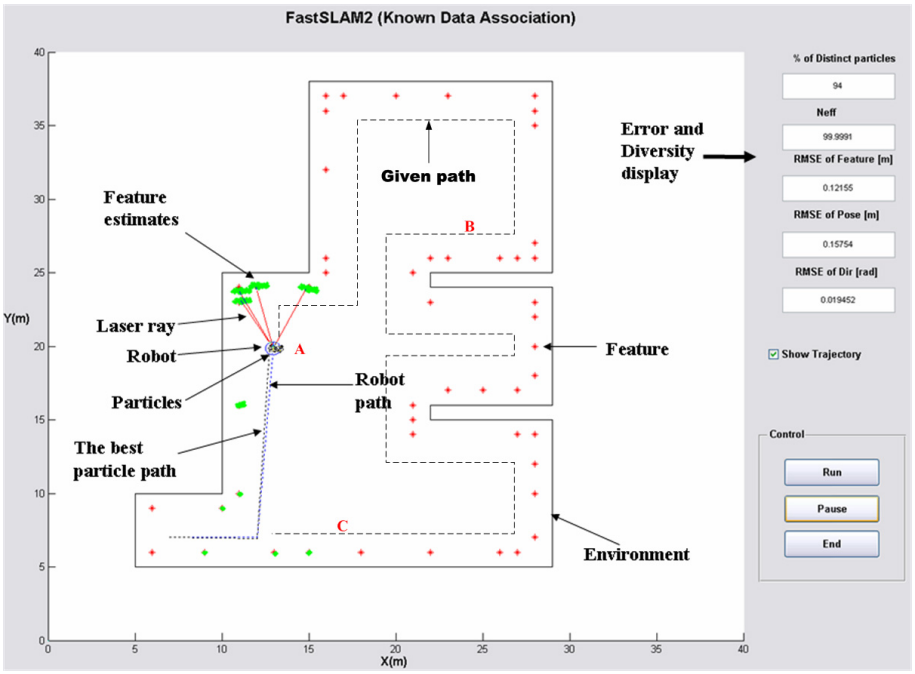


Fig. 5 The simulation environment: a sparse environment with 50 features. The robot path is given, but the robot cannot follow the path due to the motion noise. The feature error significantly increases at point **A**, and the inner loop-closure occurs at point **B** and the outer loop-closure at point **C**.

experiments in which a 3.0Ghz PC with 1GB RAM was used.

4.1 Number of Distinct Particles

A measure for the rate of loss of particle diversity is obtained by recording the number of distinct particles in the set representing a landmark. Once a landmark goes out of the robot's sight, resampling causes some particles to be rejected and others to be replicated. At first, all of the particles are distinct, which means they have different feature estimates about a landmark. As time passes, only particles with high weights survive, and particles with low weights disappear together with

their feature estimates. Thus, the number of distinct estimates of the landmark becomes smaller. The number of distinct particles is counted after every resampling process, and the results of the resampling algorithms are shown in Fig. 6 including no-resampling (NR) case.

In all of the resampling algorithms, the number of distinct particles exponentially decreases although there are a little differences among them. Especially the RSR shows the fastest convergence, and the RBR shows the slowest convergence in Fig. 6. Better particle diversity in the RBR than that in others is due to the characteristic of the RBR. Even though the variance of particles' weights is large, the RBR re-assigns the selection probability based on the rank of each particle. As the η_{max} in Eq. (3) varies, the number of distinct particles changes. In this work, the η_{max} is set to 1.5. However, after every resampling, the number of distinct particles significantly decreases. Soon after closing the inner loop, there is left only one particle having the estimate of the landmark in the RSR. The larger number of distinct particles is, the better it is to close a loop because new observations can affect the locations of the landmark. Thus, it is worthwhile to maintain the maximum particle diversity.

4.2 Effective Sample Size

To estimate how well the current set of M particles represents the true posterior, Liu[27] introduced the effective number of particles or the effective sample size, N_{eff}. One can not exactly evaluate N_{eff}, but an estimate of N_{eff} is given by

$$\widehat{N_{eff}} = \frac{1}{\sum_{k=1}^{M} (w^{[k]})^2} \tag{4}$$

The idea behind this measure is to determine the variance of particles' weights. As time progresses, estimation errors grow larger, variance of particles' weights increases, and statistical accuracy is

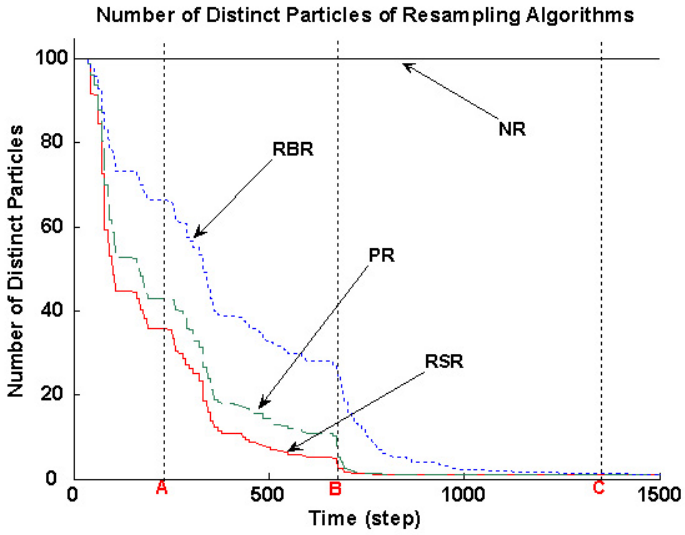
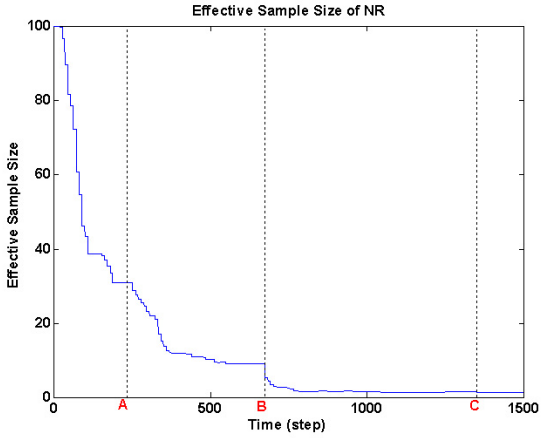
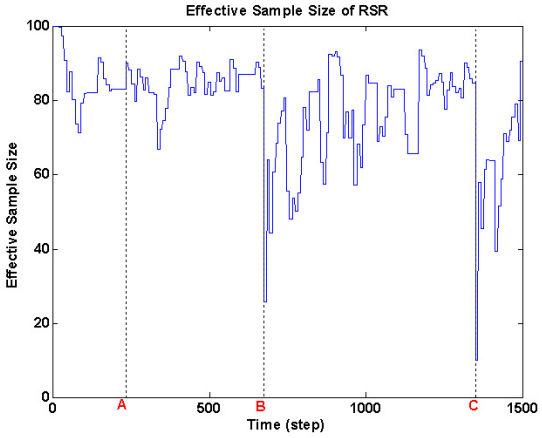


Fig. 6 Number of distinct particles of resampling algorithms including no-resampling (NR).

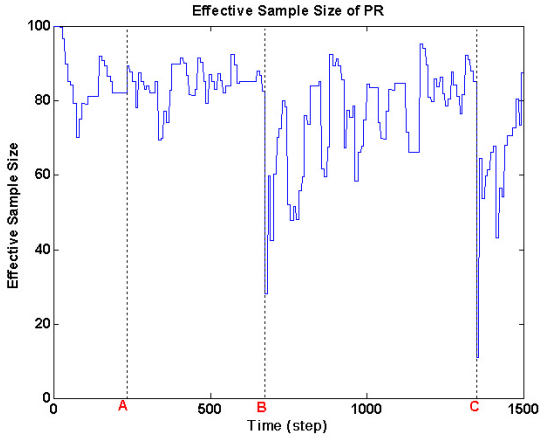
degraded. Thus, the larger $\widehat{N_{eff}}$ is, the better a particle filter estimates the target distribution. In fact, this $\widehat{N_{eff}}$ is commonly used as a measure for degeneracy of particle filters. Stachniss *et al.*[28] found that $\widehat{N_{eff}}$ stayed constant when the new information was not helpful in identifying the unlikely hypotheses or estimates represented by the individual particles. In addition, $\widehat{N_{eff}}$ decreased over time when the new information could be used to identify that some particles are less likely than others. The effective sample sizes of the resampling algorithms while performing FastSLAM are plotted, and the results are compared to that of NR in Fig. 7. Note that in all of the experiments, resampling was performed once the effective sample size fell below 75% of the total number of particles rather than after each observation. In Fig. 7, $\widehat{N_{eff}}$ of NR exponentially decreases while performing FastSLAM since particles degenerates over time without resampling. All of the resampling algorithms show almost the same pattern. At point **B**, the inner loop is closed, and at point **C**, the outer loop is closed. As a result, the effective sample size significantly



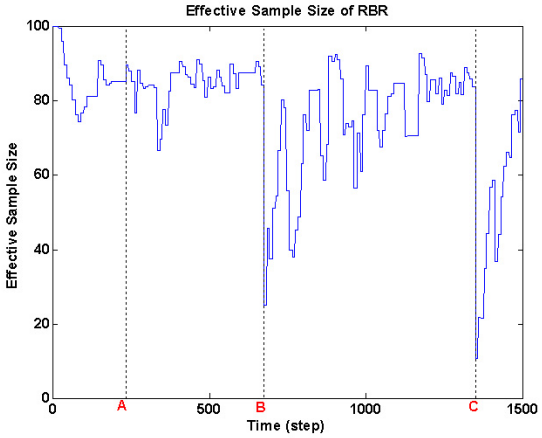
(a) NR



(b) RSR



(c) PR



(d) RBR

Fig. 7 $\widehat{N_{eff}}$ of NR, RSR, PR, and RBR while performing FastSLAM.

decreases at the two points. The decrease of the effective sample size at points **B** and **C** means that a few particles have much higher weights than the others. Consequently, particles with low weights disappeared in the particle set. However, the effective sample size is recovered soon after resampling. This is because new particles having the same weight were drawn from the proposal distribution of the survived particles. If particles were rejected in the set, survived particles would share a common history or a robot path since loss of particles means loss of paths including feature estimates dependent on those paths. To sum up, frequent loop-closing accelerates particle depletion although it reduces errors of feature estimates and the robot pose.

4.3 Estimate Errors

The objective of SLAM is to build an accurate map about an environment and exactly localize a robot based on the map. Therefore, the root mean square (RMS) errors of feature estimates, f_{rmse} and pose estimates, p_{rmse} are key performance measures in SLAM, which are defined by

$$f_{rmse} = \sqrt{\frac{1}{M \cdot N} \sum_{k=1}^{M} \sum_{i=1}^{N} (\theta_i - \mu_{i,t}^{[k]})^2} \tag{5}$$

$$p_{rmse} = \sqrt{\frac{1}{M} \sum_{k=1}^{M} (x_t^{true} - x_t^{[k]})^2} \tag{6}$$

where θ_i and x_t^{true} are true poses of a landmark i and the robot, respectively. Here, M is the number of particles, and N is the number of features which a particle has.

First, the RMS feature error of each resampling algorithm is shown in Fig. 8. For comparison, the feature error of the NR is also displayed. In the NR, the RMS feature error does not decrease at all since all of the particles survive and their estimation errors gradually increase. At point **A**, the RMS feature errors drastically increase in all the algorithms because many landmarks are seen, and

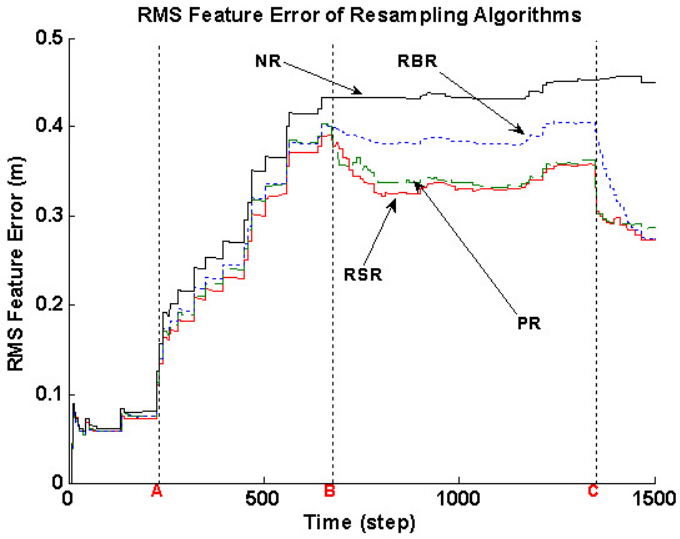


Fig. 8 The RMS feature errors of the resampling algorithms.

the robot rotates. Rotation usually causes a bigger odometry error than translation. At the inner loop-closure, point **B**, the RSR and the PR reduce the feature errors by more that $5cm$, but not the RBR. However, at the outer loop-closure, point **C**, all the algorithms rapidly reduce the feature errors significantly, even though the RBR reduces errors a little bit slowly.

Second, the RMS position error of each resampling algorithm is shown in Fig. 9. Note that the position error is computed using only x-y positions of particles. At the loop-closure points, all the algorithms including the NR reduce the RMS position errors. In the NR, unlike the feature error, the position error is reduced at the loop-closure points because particles are drawn taking into account the initial feature estimates that are more precise than the last. On the contrary, the feature error did not decrease because a new observation can not affect feature estimates in the past. In Fig. 9, error reduction performance of the RSR and the PR is better than that of the RBR as the case of the RMS feature error. The position errors converge to a very small value, about $0.05m$ after

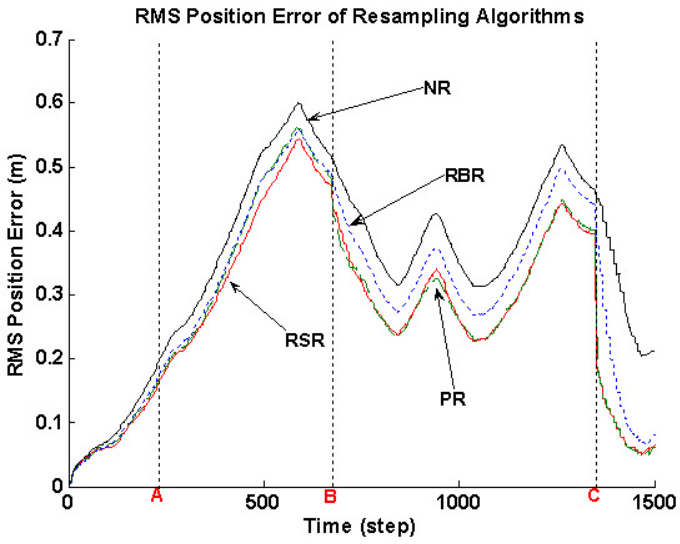


Fig. 9 The RMS position errors of the resampling algorithms.

closing the outer loop.

4.4 Computational Complexity

Computational complexity should be considered when a robot conducts practical tasks. The computational complexity of the FastSLAM algorithm requires $O(MN)$ where M is the number of particles, and N is the number of landmarks in the map.[3] The linear complexity in M is by processing M particles for every update. The linear complexity in N is due to the resampling process since particles in the set may be replicated several times. At this time, the length of the particles depends linearly on N, which makes the copying operation to also be linear in the size of the map. In fact, Montemerlo presented an implementation that has $O(Nlog_2M)$ time-complexity.[3] The complexity of the RSR, PR, RBR is $O(MN)$.

In this work, the processing time and the number of function calls of each resampling algorithm

Table III Comparison of the computational complexity of the resampling algorithms.

Algorithm	Complexity	# of func. calls	Run-time (s)
RSR	$O(MN)$	2296	8.23
PR	$O(MN)$	2301	11.62
RBR	$O(MN)$	2670	12.34

Table IV Summary of performance of the resampling algorithms.

Algorithm	Avr. num. of distinct particles	Avr. $\widehat{N_{eff}}$	RMS feature error (m)	Avr. Run-time (s)
RSR	13.7583	78.3545	0.2719	0.1646
PR	16.7826	78.2514	0.2781	0.2324
RBR	26.3222	74.8688	0.3004	0.2468

were obtained after the robot closed the outer loop. The number of function calls of each algorithm is varied with particle diversity, exactly the variance of particles' weights. The computational complexity of each resampling algorithm is summarized in Table III. The number of function calls and the run-time in Table III are the sums of 50 experiments. Resampling performance is summarized in Table IV, which shows that there is no particularly good resampling algorithm among them. In this case, scheduling of resampling is a more important factor for improving the SLAM performance. This will be dealt in the next section.

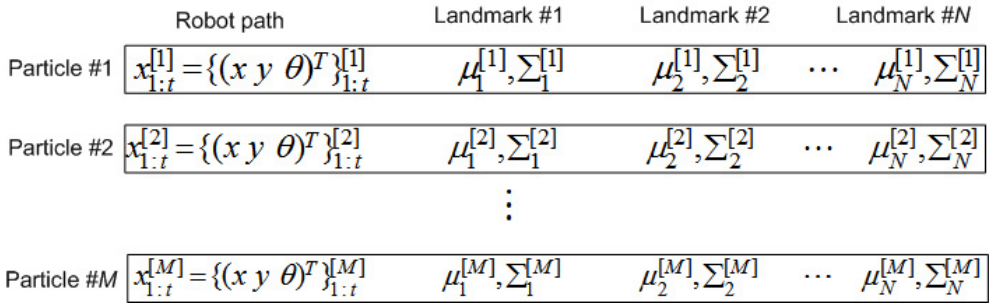


Fig. 10 A particle set in FastSLAM consists of a path estimate, $x_{1:t}^{[k]}$ and a set of estimates of individual landmark locations with associated covariances, $(\mu^{[k]}, \Sigma^{[k]})$.

5 Compensation Technique

The number of distinct particles exponentially decreases in every resampling algorithm while the RMS errors are reduced. However, in the NR, the RMS feature error did not decrease even though all of the particles survive. Therefore, we propose a new technique that particles with low weights are not rejected but compensated by particles with high weights. There is no rejected particle in this compensation technique. Actually, the compensation technique does not resample particles, but it has as the same effect as other resampling algorithms. To reduce computational cost scheduling of compensation and reducing the number of particles are also considered in this section.

5.1 The Compensation Algorithm

A particle consists of a robot path, mean and covariance of each landmark as shown in Fig. 10. In this figure, N and M are the number of landmarks and particles, respectively. Note that, when implementing the FastSLAM algorithm, each particle represents a robot path, $x_{1:t}^{[k]}$ but the recursive equations at each time-step require only the most recent pose estimate, $x_t^{[k]}$. Dependency on the

robot path is recorded in the weight of a particle and its landmark estimates.

Instead of resampling, particles with low weights are compensated by particles with high weights using the following equations:

$$x_t^{[l]} = w_t^{[l]} \times x_t^{[l]} + w_t^{[h]} \times x_t^{[h]} \tag{7}$$

$$\mu_i^{[l]} = w^{[l]} \times \mu_i^{[l]} + w^{[h]} \times \mu_i^{[h]}, \text{ where } i = 1, ..., N \tag{8}$$

Here, the superscripts $[l]$ and $[h]$ mean particles with low and high weights, respectively. For instance, $\mu_i^{[l]}$ represents the i-th landmark estimate of a particle with a low weight. In the compensation algorithm, the feature covariances of a particle is not compensated to reduce computational cost. Note that only particles with low weights are compensated in the process whereas all of the particles with high weights are preserved. Selection of a pair of particles for compensation can be easily done by using probabilities or by the best one and the worst one, the second best and the second worst, and so on. One can notice that the selection procedure has a similar mathematical structure as that of the selection methods in GA.[20]

To evaluate the performance of the compensation algorithm, various simulations were performed, and the results are compared with the best one among the resampling algorithms in the previous section. First, the effective sample size of the compensation algorithm is shown in Fig. 11. The effective sample size of the compensation algorithm is similar to those of other resampling algorithms, but its effective sample size is slightly larger at the inner and outer loop-closure.

Second, the RMS estimate errors of the compensation algorithm are shown in Fig. 12, compared with the result of the RSR. In the RMS errors, the compensation algorithm outperforms the RSR, which shows the best result in the previous section.

Third, the effect of compensation pair selection is summarized in Table V. Three selection meth-

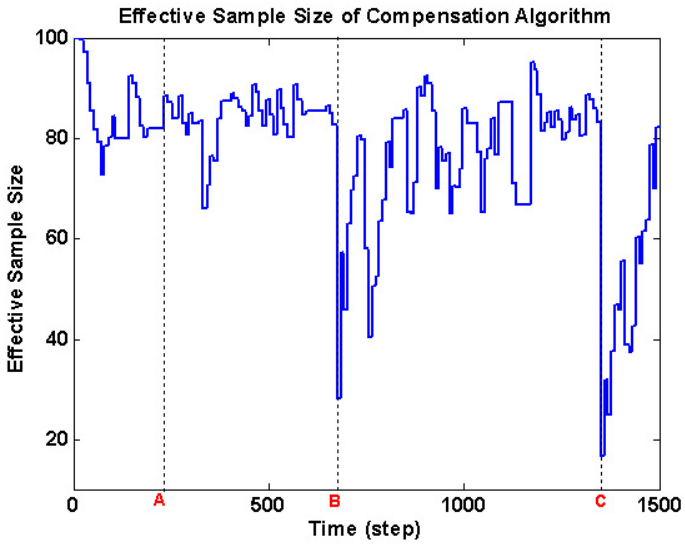
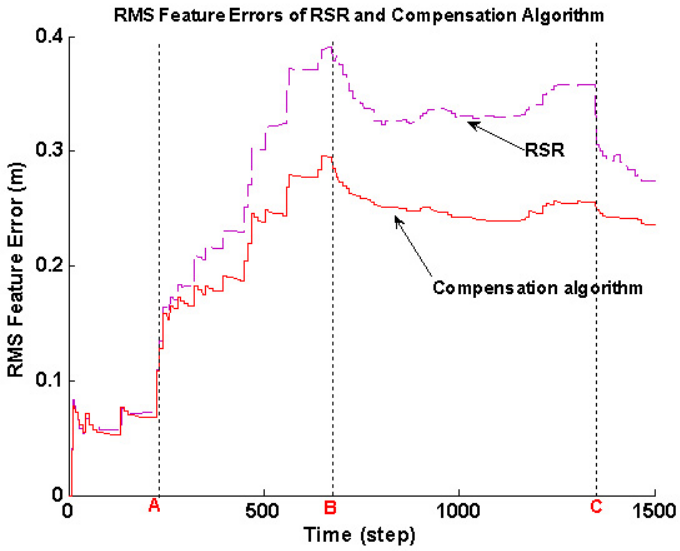


Fig. 11 The effective sample size of the compensation algorithm.

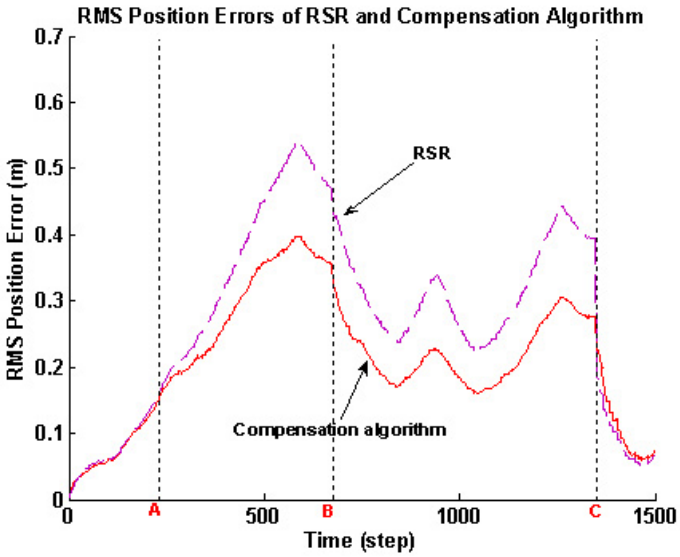
Table V Performance comparison according to selection methods.

Selection Method	Avr. $\widehat{N_{eff}}$	RMS feature error (m)	Avr. Run-time (s)
Random	76.187	0.2487	1.158
Roulette Wheel	76.189	0.2547	0.869
Rank-based	77.384	0.2109	1.108

ods were used, which are random, roulette wheel, and rank-based selection. The random method randomly selects one among particles with low weights and the other among particles with high weights. In the roulette wheel selection[29], each slice on a roulette wheel has a width corresponding to the particle's selection probability. The particle that takes a large width has a high chance to be selected. The rank-based method performs selection by pairing up the best one and the worst one, the second best and the second worst, and so on. In Table V, the rank-based selection produces the smallest feature error though its run-time is longer than the roulette wheel selection.



(a) Comparison of the RMS feature errors of RSR and the compensation algorithm



(b) Comparison of the RMS position errors of RSR and the compensation algorithm

Fig. 12 Comparison of the RMS errors of RSR and the compensation algorithm.

Table VI Performance comparison with varying the compensation ratio.

Compensation Ratio	Avr. $\widehat{N_{eff}}$	RMS feature error (m)	Avr. Run-time (s)
10%	73.009	0.3169	0.241
20%	74.908	0.2817	0.386
50%	76.618	0.2512	0.694
100%	77.384	0.2109	1.108

At last, the influence of the number of compensated particles is summarized in Table VI. In the table, the compensation ratio of 100% means all of the particles with low weights have been compensated. As the compensation ratio increases, the effective sample size increases and the RMS feature error decreases. The disadvantage of the compensation algorithm is its run-time, which increases as the compensation ratio increases. The compensation algorithm has many multiplications proportional to the number of particles and features. Because of this disadvantage, the compensation algorithm is suitable for a sparse environment with a small particle size. However, the disadvantage is not critical for implementation, and it can be resolved by scheduling of compensation.

5.2 Scheduling of Compensation

Scheduling of resampling is very important and is an open problem. As presented in Section 4, there is little difference in the performance of resampling algorithms. In this case, the best time to instigate resampling is the problem rather than which resampling algorithm is to be used. Likewise, the compensation algorithm should reduce computational cost by scheduling. Simulations were performed to decide when it is the best time to instigate compensation by varying the threshold for compensation, T_N using the effective sample size. More specifically, compensation is instigated after

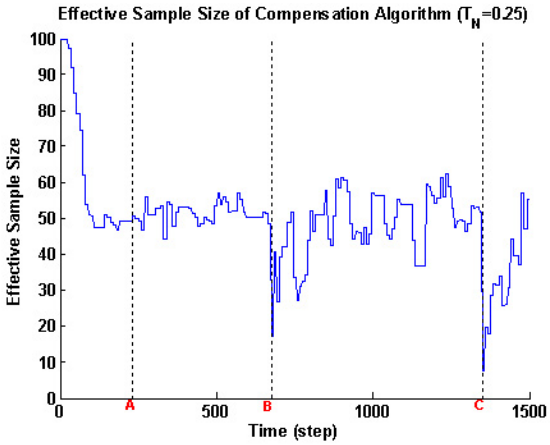
the ratio of the effective sample size falls below the T_N. For instance, when $T_N = 1.0$, compensation is conducted after each observation.

First, comparisons of the effective sample size and the RMS feature error of the compensation algorithm are shown in Fig. 13 and in Fig. 14 when the T_N changes from 0.25 to 1.0. In Fig. 13, the effective sample sizes when $T_N = 0.25$ and $T_N = 0.5$ are much smaller than those when $T_N = 0.75$ and $T_N = 1.0$. In Fig. 14, it seems that the performance when $T_N = 1.0$ is best. However, in that case the compensation algorithm does not reduce the RMS feature error at the outer loop-closure, point **C** because particles lost their diversity due to the frequent compensation. On the contrary, when $T_N = 0.75$, the RMS feature error is reduced at the outer loop-closure.

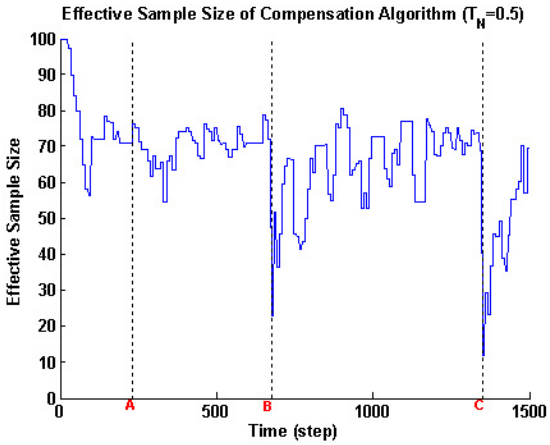
Second, a comparison of the number of the algorithm calls is shown in Fig. 15 for checking computational cost of the compensation algorithm. In this figure, the number of calls when $T_N = 1.0$ is more than twice of that when $T_N = 0.75$. Thus, $T_N = 1.0$ is not adequate for the algorithm as a threshold in the aspect of the computational cost. Based on the simulation results, the proper threshold is 0.75 for when to instigate compensation.

5.3 Relation between the number of particles and performance

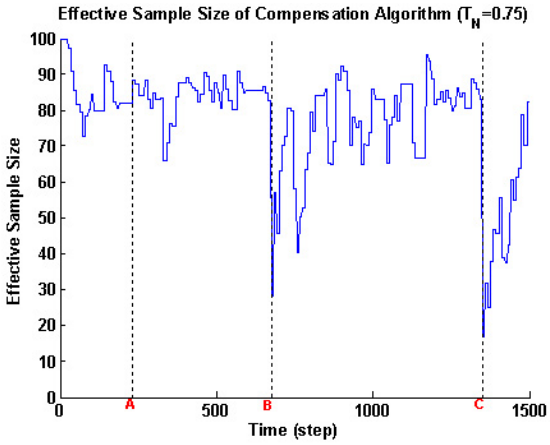
The compensation algorithm, compared with other resampling algorithms, takes more time to compensate particles with low weights though it produces much better performance in estimate errors. Simulations with varying the number of particles were performed to know that it is reasonable to reduce the number of particles. The simulation result as well as the result of the RSR, when the numbers of particles were 10, 50, and 100 is shown Fig. 16. The result using 50 particles is similar to that using 100 particles, and outperforms the result of the RSR using 100 particles. The com-



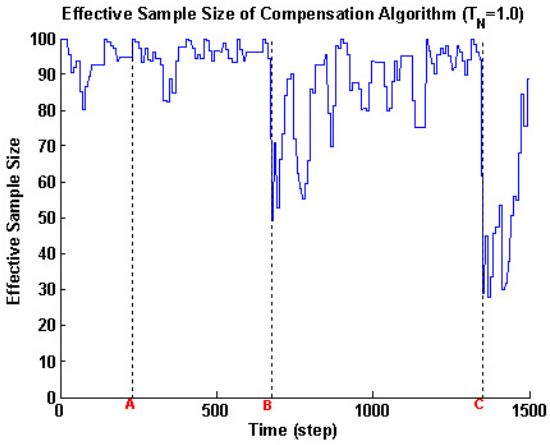
(a) $T_N = 0.25$



(b) $T_N = 0.5$



(c) $T_N = 0.75$



(d) $T_N = 1.0$

Fig. 13 The effective sample sizes with different thresholds for the compensation algorithm

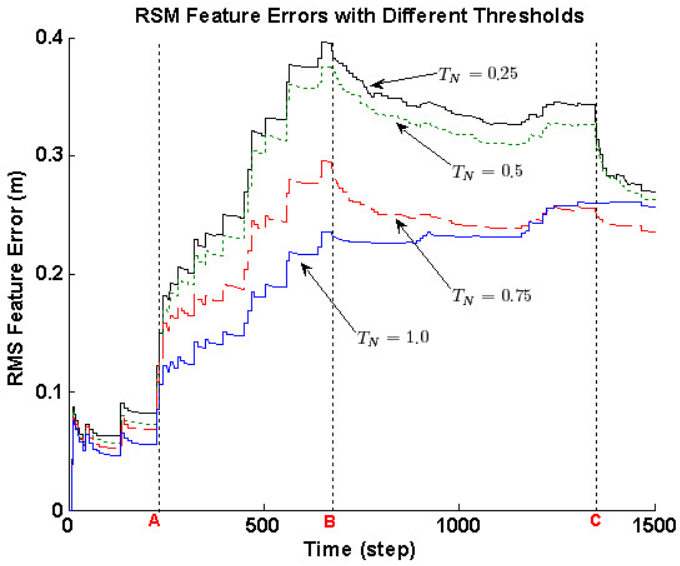


Fig. 14 Resampling performance of the compensation algorithm with different thresholds.

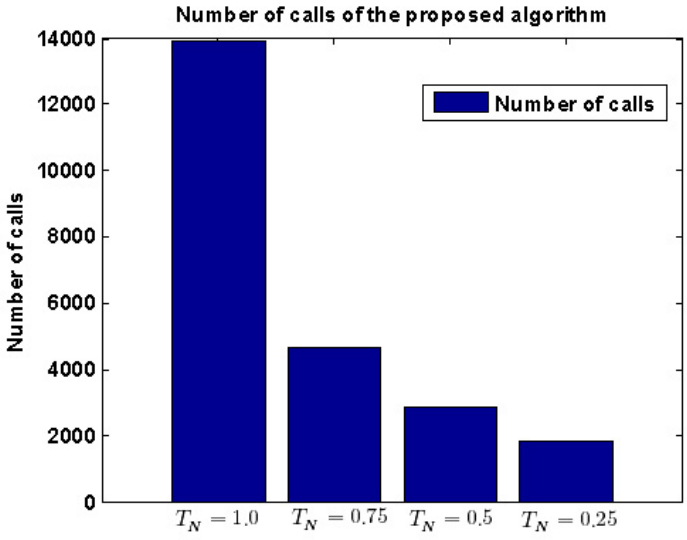


Fig. 15 Number of the algorithm calls with different thresholds.

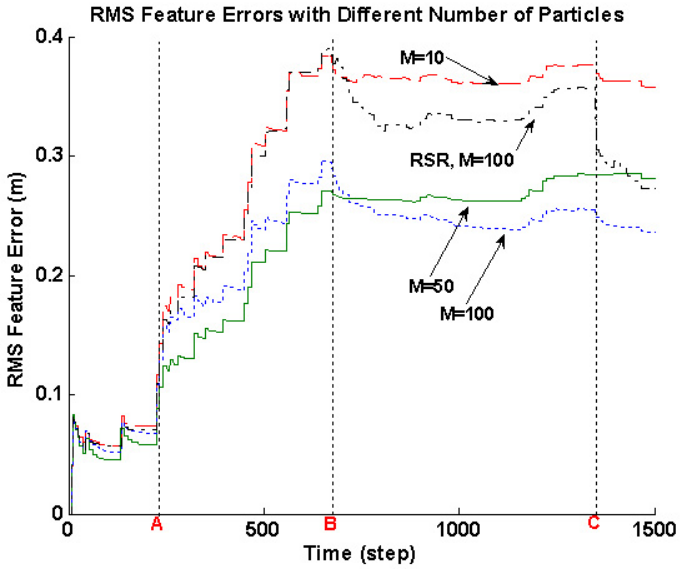


Fig. 16 Comparison of the RMS feature errors with different number of particles of the compensation algorithm and that of the RSR with 100 particles.

pensation algorithm does not show degeneracy of the algorithm even though the number of particles decreases by 50 particles. When 50 particles are used, compensation is conducted in the half time of when 100 particles are used. The run-time can be more reduced by diminishing the number of particles to be compensated.

6 Conclusion

FastSLAM has been shown to cause the particle depletion problem, so it always produces over-confident estimates of uncertainty as time progresses. FastSLAM degenerates over time because of the particle depletion. The particle depletion problem is mainly due to the resampling process in FastSLAM, which in turn eliminates reasonable particles with low weights.

In this paper, using the computer simulations we analyzed resampling algorithms: RSR, PR, and RBR which was adopted as a resampling algorithm to alleviate particle depletion. The performance measures for the thorough analysis are the effective sample size, the number of distinct particles, estimate errors, and complexity. The RSR takes the fastest time to perform resampling, and the RBR keeps distinct particles longest. The estimate errors, the paramount measure for SLAM are small in the RSR and the PR, but not in the RBR. However, all of the resampling algorithms could not resolve the particle depletion problem, which means the SLAM performance degenerates over time.

Thus, we proposed the compensation technique instead of resampling for resolving the depletion problem. It probabilistically compensates particles with low weights using particles with high weights. As a result, estimate errors are significantly reduced compared to general resampling algorithms though its run-time is longer than that of them. Since the particles in the compensation technique are distinct, many particles performed the inner and outer loop-closure, so they reduced the RMS errors.

Moreover, we decided the threshold for the time when it would be best to instigate compensation. We also reduced the its run-time without loss of performance by diminishing the number of particles. In the future, we will reduce the run-time of the compensation algorithm not by reducing the number of particles but by efficiently selecting a pair of particles to be compensated.

Acknowledgements

This work was supported in part by MIC & IITA through IT Leading R&D Support Project, the ASRI, the BK21 Information Technology at Seoul National University, and the Seoul R&BD

Program(10689M092991), Korea.

References

1. S. Ekvall, D. Kragic, and P. Jensfelt. "Object detection and mapping for service robot tasks." *Robotica*, 25(02), 175–187 (2007).

2. J. L. Leonard, R. N. Carpenter, and H. J. S. Feder. "Stochastic mapping using forward look sonar." *Robotica*, 19(05), 467–480 (2001).

3. M. Montemerlo. *FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem With Unknown Data Association*. Ph.D. thesis, Carnegie Mellon University (2003).

4. S. Fazli and L. Kleeman. "Simultaneous landmark classification, localization and map building for an advanced sonar ring." *Robotica*, 25(02), 1–14 (2006).

5. M. A. Salichs and L. Moreno. "Navigation of mobile robots: open questions." *Robotica*, 18(03), 227–234 (2000).

6. P. Newman, J. Leonard, J. D. Tardos, and J. Neira. "Explore and return: experimental validation of real-time concurrent mapping and localization." *IEEE International Conference on Robotics and Automation*, 2, 1802– 1809 (2002).

7. A. Doucet, N. de Freitas, K. Murphy, and S. Russell. "Rao-blackwellised particle filtering for dynamic bayesian networks." *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pp. 176–183 (2000).

8. K. Murphy. "Bayesian map learning in dynamic environments." *Advances in Neural Information Processing Systems (NIPS)*, 12, 1015–1021 (1999).

9. M. Montemerlo and S. Thrun. "Simultaneous localization and mapping with unknown data association using fastslam." *Proceedings of 2003 IEEE International Conference on Robotics and Automation*, 2 (2003).

10. S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, Cambridge (2005).

11. T. Bailey, J. Nieto, and E. Nebot. "Consistency of the fastslam algorithm." *IEEE Intl. Conf. on Robotics and Automation*, pp. 424–427 (2006).

12. A. Kong, J. S. Liu, and W. H. Wong. "Sequential imputations and bayesian missing data problems." *Journal of the American Statistical Association*, 89(425), 278–288 (1994).

13. M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. "Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges." *Proceedings of the seventeenth International Joint Conference on Artificial Intelligence (IJCAI-03)* (2003).

14. G. Grisetti, G. D. Tipaldi, C. Stachniss, W. Burgard, and D. Nardi. "Fast and accurate slam with rao-blackwelized particle filters." *Robotics and Autonomous Systems*, 55, 30–38 (2007).

15. A. Doucet and N. J. Gordon. "Simulation-based optimal filter for manoeuvring target tracking." *SPIE proceedings series*, pp. 241–255 (1999).

16. R. van der Merwe, N. de Freitas, A. Doucet, and E. Wan. "The unscented particle filter,

cambridge university engineering department, 2000." Technical report, CUED/F-INFENG/TR 380, Cambridge, England (2000).

17. J. S. Liu, R. Chen, and T. Logvinenko. "A theoretical framework for sequential importance sampling and resampling." *Sequential Monte Carlo Methods in Practice*, pp. 225–246 (2001).

18. N. J. Gordon, D. J. Salmond, and A. F. M. Smith. "Novel approach to nonlinear/non-gaussian bayesian state estimation." *Radar and Signal Processing, IEE Proceedings F*, 140(2), 107–113 (1993).

19. G. Grisetti, C. Stachniss, and W. Burgard. "Improved techniques for grid mapping with rao-blackwellized particle filters." *IEEE Transactions on Robotics*, 23(1), 34–46 (2007).

20. T. Higuchi. "Monte carlo filter using the genetic algorithm operators." *Journal of Statistical Computation and Simulation*, 59(1), 1–23 (1997).

21. M. Bolić, P. M. Djuric, and S. J. Hong. "Resampling algorithms for particle filters: A computational complexity perspective." *Eurasip Journal on Applied Signal Processing*, 2004(15), 2267–2277 (2004).

22. —. "New resampling algorithms for particle filters." *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'03)*, 2 (2003).

23. J. E. Baker. "Adaptive selection methods for genetic algorithms." *Proceedings of the 1st International Conference on Genetic Algorithms table of contents*, pp. 101–111 (1985).

24. H. Durrant-Whyte and T. Bailey. "Simultaneous localization and mapping: Part i." *IEEE Robotics and Automation Magazine*, 13(2), 99–108 (2006).

25. D. Crisan and A. Doucet. "A survey of convergence results on particle filtering methods for practitioners." *IEEE Transactions on Signal Processing*, 50(3), 736–746 (2002).

26. D. Whitley. "The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best." *Proceedings of the Third International Conference on Genetic Algorithms*, 1, 116–121 (1989).

27. J. S. Liu. "Metropolized independent sampling with comparisons to rejection sampling and importance sampling." *Statistics and Computing*, 6(2), 113–119 (1996).

28. C. Stachniss, G. Grisetti, and W. Burgard. "Recovering particle diversity in a rao-blackwellized particle filter for slam after actively closing loops." *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)* (2005).

29. T. Bäck, D. B. Fogel, and Z. Michalewicz. *Evolutionary Computation 1: Basic Algorithms and Operators*. Institute of Physics Publishing, Bristol (2000).