Robotica (2009) volume 27, pp. 853–859. © Cambridge University Press 2009
doi:10.1017/S0263574708005250

Improved particle fusing geometric relation between particles in FastSLAM

Inkyu Kim*, Nosan Kwak, Heoncheol Lee and Beomhee Lee

School of Electrical Engineering and Computer Science, Seoul National University, Korea.

(Received in Final Form: November 28, 2008. First published online: January 6, 2009)

SUMMARY

FastSLAM is a framework for simultaneous localization and mapping using a Rao-Blackwellized particle filter (RBPF). But, FastSLAM is known to degenerate over time due to the loss of particle diversity, mainly caused by the particle depletion problem in resampling phase. In this work, improved particle filter using geometric relation between particles is proposed to restrain particle depletion and to reduce estimation errors and error variances. It uses a KD tree (*k*-dimensional tree) to derive geometric relation among particles and filters particles with importance weight conditions for resampling. Compared to the original particle filter used in FastSLAM, this technique showed less estimation error with lower error standard deviation in computer simulations.

KEYWORDS: Particle filter; FastSLAM; Geometric relation.

1. Introduction

The simultaneous localization and mapping (SLAM) is a fundamental problem of robots to perform autonomous tasks, such as exploration and navigation in unknown environments. The two key computational solutions to the SLAM problem use the extended Kalman filter (EKF-SLAM) and the Rao-Blackwellized particle filter (FastSLAM). EKF-SLAM has served as the main approach to solve the SLAM problem for the last 15 years. However, EKF-SLAM is known to have two major well-known shortcomings: quadratic computational complexity and sensitivity to failures in data association.[1] Recently, the FastSLAM algorithm has been proposed as an alternative solution to the SLAM problem.[2] It uses the 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),[3] which factors the full SLAM posteriors into the product of two terms: a robot path posterior and landmark posteriors conditioned on the robot path estimate.[4] 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.[5] The ability to pursue multiple data associations simultaneously makes FastSLAM significantly more robust to data association problems than algorithms based on incremental maximum likelihood data association, such as EKF-SLAM.[1]

However, FastSLAM has some drawbacks. In the literatures,[6,7] FastSLAM has been noted to degenerate over time. This degeneracy happens when a particle set estimating the pose of the robot loses its diversity. In general, more diversity in a particle set results in better loop-closing performance. There are two main reasons for losing particle diversity in FastSLAM. First, sample impoverishment, caused by mismatch between target distribution and proposal distribution, produces improbable particles. Especially, if the robot's motion is very noisy while the robot's sensor is very accurate, this mismatch frequently occurs. Second, during the resampling process in FastSLAM, the improbable particles are thrown away, and only particles with high weights survive. Particle with important information may get low weight and dumped out. Lost information stored within the dumped particle cannot be recovered and the particle depletion problem happens.

The sample impoverishment problem and the particle depletion problem happen in all particle filters. The reason for the two problems is in sampling and resampling phases. On the view of sampling, the loss of particle diversity is due to poor samples generated in the sampling process. The choice of proposal distribution is the most critical design issue in particle filters to solve the sampling problem.[8] In this aspect, many schemes using mixture proposal distribution have been studied.[9,10] Another critical issue is the number of particles estimating the pose of the robot. If there were enough number of particles to represent the posterior distribution, sample impoverishment would not happen. There is, however, a trade-off between the number of particles and computational cost. Gordon *et al*. proposed a simple implement strategy so-called prior boosting or prior editing.[11] There is, however, no specific scheme to boost the number of particles. We proposed an adaptive prior boosting technique to control the number of particles using a neural network training method.[12] Boosting the number of particles showed improved performance, but also increased the computational cost.

Many algorithms[8,13,14] have been proposed to suppress particle depletion, but they are not good enough to fully prevent particle depletion. On the view of computational improvement, Grisetti *et al.*[15] proposed computation

* Corresponding author. E-mail: gimming9@snu.ac.kr

reduction by re-using proposal distribution for each particle. A different approach was proposed by Lee and Lee[16] using landmark geometric constraints. In their work, a landmark group is formed based on the geometric relationship between landmarks and particles, sharing observation data using the landmark group.

Many approaches to improve FastSLAM have been researched, but no method used the particle geometry information. In this paper, a novel resampling technique using geometric relation between particles is proposed to solve the particle depletion problem. Geometrically related particles satisfying the conditions are selected in the resampling process in FastSLAM to reduce sudden estimation failure which causes the situations of good particles being thrown away. This paper is organized as follows: In Section 2, the FastSLAM algorithm and its particle depletion problem are briefly described. An improved particle filter using particles' geometric relation is proposed in Section 3, and its simulation results are discussed in Section 4. Finally, the conclusion is presented in Section 5.

2. Particle Depletion Problem in FastSLAM

2.1. 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, $c_{1:t} = c_1, \ldots, c_t$ are known[1]:

$$\begin{aligned} & p(x_{1:t}, M | z_{1:t}, u_{1:t}, c_{1:t}) \\ & = p(x_{1:t} | z_{1:t}, u_{1:t}, c_{1:t}) \\ & \quad \times \prod_{n=1}^{N_f} p(m_n | x_{1:t}, z_{1:t}, u_{1:t}, c_{1:t}) \end{aligned} \tag{1}$$

where $x_{1:t}$ is the robot path from the start up to time t, M is the map, and $z_{1:t}$ and $u_{1:t}$ are the measurements and controls up to time t, respectively. 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 $p(m_n | x_{1:t}, z_{1:t}, c_{1:t})$, one for each $n = 1, \ldots, N_f$ where N_f is the number of features. The feature estimators are conditioned on the robot path, which means there are separate copies of each feature estimator, one for each particle. More precisely, feature locations are estimated using EKFs. Due to the factorization, FastSLAM can maintain a separate EKF for each feature, which makes the update more efficient than that in EKF-SLAM. By keeping the feature estimates independently, 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.

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

$$Y_t^{[k]} = \left\langle x_t^{[k]}, \mu_{1,t}^{[k]}, \Sigma_{1,t}^{[k]}, \ldots, \mu_{N_f,t}^{[k]}, \Sigma_{N_f,t}^{[k]} \right\rangle, \tag{2}$$

where $[k]$ indicates the index of the particle, and $x_t^{[k]}$ is the pose estimate of the robot at time t. Only the most recent pose $x_t^{[k]}$ is used in 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, representing the nth feature location relative to the kth particle, respectively. Altogether, these elements form the kth particle $Y_t^{[k]}$, and there are total N_p particles and N_f feature estimates in a particle set.

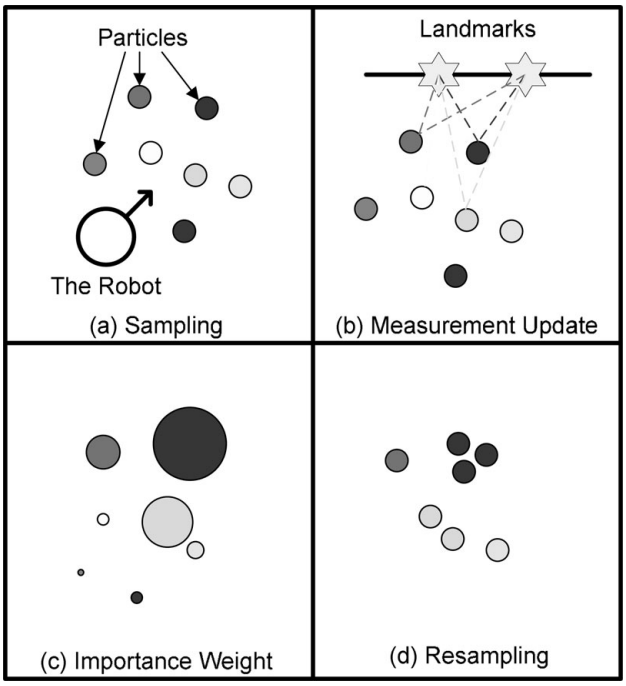


Fig. 1. The FastSLAM algorithm. (a) Sample particles using control input and current measurement, (b) measurement update, (c) importance weight, and (d) resampling.

The simple graphical procedure of the FastSLAM 2.0 algorithm is illustrated in Fig. 1. In Fig. 1(a), each particle samples a robot 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 as shown in Fig. 1(b). The next step is to compute the importance weight of kth particle using the following weighting function:

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

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. The particles receive higher weights where the target distribution is larger than the proposal distribution, and the particles are given lower weights where target distribution is smaller than the proposal distribution. The last process is resampling, which draws N_p particles from the temporary particle set. As shown in Fig. 1(d), the temporary particle with high importance weight is replicated three times whereas the one with low importance weight is thrown away by the resampling process. This means the robot path and feature estimates of the rejected particles are lost.

https://doi.org/10.1017/S0263574708005250 Published online by Cambridge University Press

2.2. Particle depletion problem

The particle filter suffers from the so called particle depletion problem, that is, the number of distinct particles is getting smaller as time passes.[6,17] At first, all of the particles are distinct, which means that they have different feature estimates about a landmark. As time passes, however, only particles with high weights survive in resampling step, and particles with low weights disappear together with their feature estimates.

The problem happens when a particle receives a low weight though it is very close to the real pose of the robot. This is so-called sudden estimation failure situation. So resampling only with weights might cause the particle depletion. Thus, a novel technique is proposed to resample with geometric relation between particles, together with weights.

3. Improved Particle Filter using Particle Geometry Information

3.1. The KD tree

Particles are sampled using control input and current measurement from previous particles and each particle has its own estimation of the real robot pose. As particles are predicted from the different particles of previous time step, particles are independent of one another. These independent particles form a geometric relation: how far particles are located, how particle are distributed, what direction particles are heading, etc. Among many geometric relations, particle distribution is considered as the geometric relation between particles in this paper.

If most of the highly weighted particles are gathered in a certain region, the real robot will be located in that region with a high probability. This is similar to the concept of the Particle Swarm Optimization (PSO). This concept was originated from the swarm behavior of insects like ants. In PSO, particles find the global best estimation by sharing information with the nearest particle. Then each particle adjusts its position according to the global best. This is like a swarm of ants looking for the food source. As the food source may exist near where many ants with food are gathered, ants without food adjust their goal toward the point where the ants with food came from. Using the concept of PSO in particle filter, particle receives confidence over its estimation if other particles with high weights are adjacent. Therefore, highly weighted adjacent particles, so called neighboring particles, should receive a credit as they estimate the real robot pose more precisely.

In order to search a group of highly weighted particles, neighboring particles from a reference particle should be selected. In an intuitive view, neighboring particles are located close to the reference particle. There are two problems when choosing neighboring particles with respect to the Euclidean distance. It is hard to define threshold to be chosen and the computation increases as Euclidean distance from the reference particle to every particle must be calculated. So a KD tree (k-dimensional tree)[18,19] is constructed using particle poses as the tool to select neighboring particles. The KD tree is a space-partitioning data structure for organizing points in a k-dimensional space.

Table I. Algorithm for building KD tree.

Algorithm BUILDKDTREE(P, depth)
Input: A set of points P and the current depth
Output: The root of a KD tree storing P.

1: *if* P contains only one point
2: *then return* a leaf storing this point
3: *else if depth* is even
4: *then* Split P into two subsets with a vertical line l through the median x-coordinate of the points in P. Let P_1 be the set of points to the left of l or on l, and let P_2 be the set of points to the right of l.
5: *else* Split P into two subsets with a horizontal line l through the median y-coordinate of the points in P. Let P_1 be the set of points below l or on l, and let P_2 be the set of points above l.
6: $v_{left} \leftarrow$BUILDKDTREE(P_1, $depth + 1$)
7: $v_{right} \leftarrow$BUILDKDTREE(P_2, $depth + 1$)
8: Create a node v storing l, make v_{left} the left child of v, and make v_{right} the right child of v.
9: *return* v

A KD tree is built according to the following algorithm on Table I.

The KD tree can search a node with single query covering rectangular range with $O(\sqrt{n})$ time, while searching neighboring particles using Euclidean distance requires $O(n^2)$ time. So the KD tree is a useful structure for range searching which makes the KD tree suitable for the structure to search neighboring particles. Figure 2 shows an example of building the KD tree. In Fig. 2, neighboring particles are easily selected by searching a subtree. For example, P_1, P_2, and P_3 are in the subtree l_4, meaning those three points are neighboring particles. To find the neighboring particles of P_1, they can easily be achieved by selecting a subtree where P_1 belongs then picking up the particles belonging to the subtree. There is no need to search all the particles whether they are neighboring particles or not.

3.2. Resampling with the KD tree structure

In FastSLAM, usual resampling technique is residual stratified resampling (RSR) technique which uses weight strata to decide how many copies of each particle should be made. The RSR technique shows quite nice performance, except for a sudden estimation failure which causes the loss of particle diversity. This sudden estimation failure happens when particle receives high weights which are not correctly pointing the real robot pose. Highly weighted particles with wrong estimation survive and samples for the next FastSLAM step are predicted based on those corrupted particles. If most of the particles are sampled based on those corrupted particles, particles may not be diversely distributed to let at least some particles to estimate the real robot pose correctly, resulting in the loss of particle diversity. This error accumulates as time passes and the error diverges.

In order to restrain losing diversity, keeping particles sufficiently near the real pose of the robot is very critical issue in FastSLAM. The proposed approach, geometric relation resampling (GRR) takes account of not only particle weights but also geometric relation between particles for resampling.

https://doi.org/10.1017/S0263574708005250 Published online by Cambridge University Press

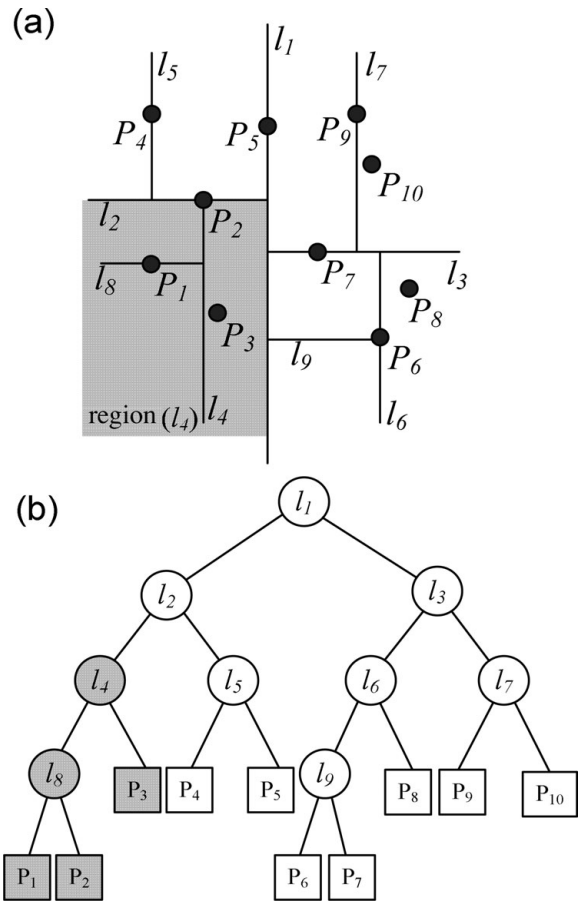


Fig. 2. A KD -tree: (a) the way the plane is subdivided; (b) the corresponding tree. The region (l_4) contains three points, P_1, P_2, and P_3 in subtree of node l_4. Particles with geometric relation can be obtained by searching a node and its subtree.

Using the GRR algorithm, the problem of sudden estimation failure can be reduced and diverging particles from the real pose can be suppressed.

The GRR algorithm selects particles for resampling according to the conditions of particle geometry and particle weights. First condition is the geometric distribution. As mentioned, neighboring particles should be selected first, in order to check whether a group of neighboring particles has relatively high weights or not. A group of particles is selected by using a KD tree then searching particles belonging to a subtree. It is important to decide from which depth of subtree particle should be drawn. If particles are drawn from the subtree of depth 0, every particle is selected and grouping the neighboring particles would be meaningless. In the RSR simulations, average ratio of particle rejection during the resampling step was 23.94%. It means if the number of particles is fixed, approximately one out of five particles was rejected and one out of five was copied twice. If those multi-copied particles are all neighboring particles, the group would have a higher possibility of correctly estimating the real robot pose than other groups. So in our algorithm, 20% of total particles are selected as one group. For example, if there are 50 particles, subtree of depth 2 is examined as a group, then particles selected for one group are 8–16 (16–32%).

After extracting the neighboring particles, the second condition, particle weights, should be analyzed. As mentioned above, every particle in a group should exceed certain weight to receive extra credit. The weight threshold is set to be the average of weights as only particles above average weight survives in the resampling step. Then the weight distribution should be analyzed. To certify the particle weight distribution to give an extra credit, there should be a tool to evaluate the particle weight distribution. So the estimation error potential, the possibility of estimation error depending on the real pose of the robot, is defined as follows:

$$\begin{aligned}&\text{Estimation error potential (EEP)}\\&= p_1 \cdot d_{1,\text{err}} \cdot N_1 + p_2 \cdot d_{2,\text{err}} \cdot N_2 + \cdots + p_n \cdot d_{n,\text{err}} \cdot N_n,\\&= w_1 \cdot d_{1,\text{err}} \cdot N_1 + w_2 \cdot d_{2,\text{err}} \cdot N_2 + \cdots + w_n \cdot d_{n,\text{err}} \cdot N_n\end{aligned} \tag{3}$$

where p_n stands for probability of a particle n to be selected, which also can be represented as particle n's weight w_n. $d_{n,err}$ stands for the estimation error, distance between the real pose of the robot and the particle n. N_n is the number of particle copies proportion to w_n.

A simplified example is shown in Fig. 3. Cases 1 and 2 in Fig. 3, which have same geometric particle distribution, are compared to evaluate which weight distribution is more robust depending on the real robot pose. In Fig. 3, case 1 with difference real robot position (a), (b), and (c), the EEP is 4, 10, and 16, respectively. And for case 2, no matter where the real robot is, the EEP is 9. Here we can check out whether the weighting function of importance weighting step in FastSLAM worked well or not. The best case is case 1 with real robot position of (a). P_1 is located closer to the real robot pose and P_1 receives higher weight, meaning that the weighting function works well. But in case 1 in Fig. 3(b) and (c), the weighting function fails to correctly give weights and as a result biased weight distribution increases the EEP. In case of evenly distributed weight, (a), (b), and (c) of case 2, the EEP is stable which means the estimation error is robust to the weight function failure. Though particle distributions of cases 1 and 2 are the same, biased weights may cause the mean and the variance of the EEP to grow. That is, resampling with a group of evenly weighted particles can reduce both the mean and the variance of the EEP. The evenness of weights in a group of neighboring particle is decided when the maximum weight difference is less than 30%, for the average of weight standard deviation over weight mean ratio in the RSR simulation was 28.41%.

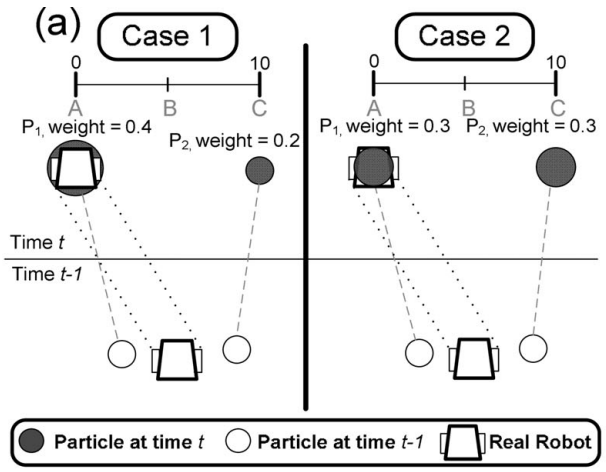
During the resampling phase, a group of particles is selected from the KD tree and particles' weights and weight distribution are verified whether a group estimates the real robot pose well or not. If particles satisfy the conditions, they are resampled additionally. By resampling with weights and geometric relationship, the GRR algorithm can reduce the estimate error and error variance and increase the FastSLAM consistency.

4. Simulation Results

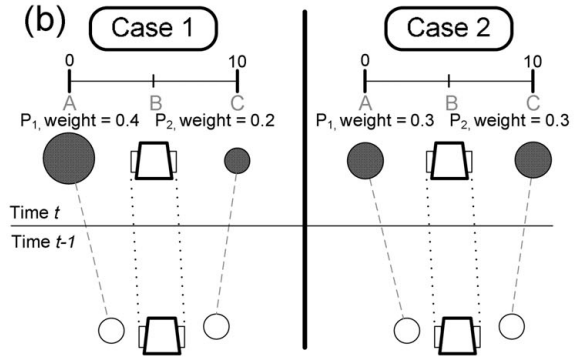
4.1. Simulation setup

Simulations were performed on a 25 m × 37 m simulated indoor environment with 50 artificial features, shown in Fig. 4.

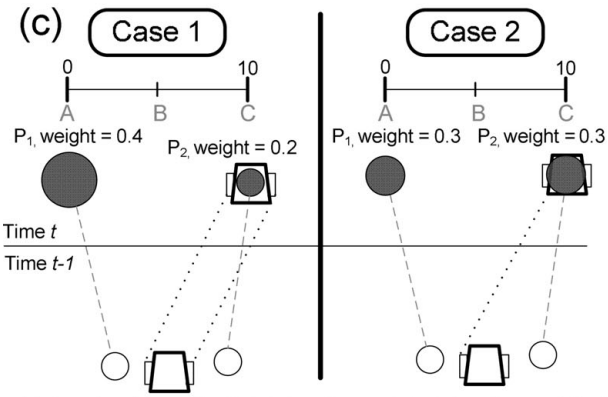
https://doi.org/10.1017/S0263574708005250 Published online by Cambridge University Press



(a) Real robot is in A. EEP of case 1 is 4, of case 2 is 9.



(b) Real robot is in B. EEP of case 1 is 10, of case 2 is 9



(c) Real robot is in C. EEP of case 1 is 16, of case 2 is 9

Fig. 3. Two examples of the particles with different weights, each with three possible real robot poses. Each case of the real robot poses, estimation error potential changes. In case 1 in (a), (b), and (c), estimation error potentials are fluctuant depending on the real pose of the robot. In case 2 in (a), (b), and (c), estimation error potentials are balanced, which means robustness to the weighting function failure.

Table II. Error probability results of two cases 1 and 2 in Fig. 3.

	Error probability				
	(a)	(b)	(c)	Mean of EPP	Variance of EPP
Case 1	4	10	16	10	36
Case 2	9	9	9	9	0

The robot path was given to tour around the indoor environment while obtaining the measurement data from the features. The FastSLAM algorithm estimates the position of robot as well as the position of the features simultaneously. During a simulation run, the estimation error of features decreases when robot closes the inner loop near point B and the outer loop near point C, as robot senses the feature observed before. The robot runs one lap around the environment in one simulation, and in total 50 simulations were run. Separate simulations were performed to evaluate the performance of RSR and GRR algorithm. Average pose and feature error were calculated with a 3.0-GHz PC with 1-GB RAM. Simulations were performed using FastSLAM 2.0 with known data association.

4.2. Simulation results

On average, 6.2 out of 50 particles were selected from the KD tree and resampled using the GRR algorithm. A single time step example of particle distribution is shown in Fig. 5. Particles are diagonally distributed from the real robot pose, and it's not clear which particles are correctly estimating the real robot pose by only looking at the particles' weights as they do not differ much no matter where the particles are located. But since particles selected by the GRR algorithm are the nearest particles to the real robot pose, resampling this group of particles would help improving the performance. Throughout the simulation, the average distance between the real pose of the robot and particles was 0.0331 while that from the KD tree was 0.0109. This result shows that neighboring particles selected by GRR algorithm successfully keep track of the real pose of the robot.

In Fig. 6, the comparison of the RMS errors of pose and error variance of the RSR and the GRR are shown. In the result of the RSR, which is the original FastSLAM resampling technique, sudden estimation failure occurs frequently. As mentioned before, sudden estimation failure happens when inappropriate particles are resampled and the particles lose diversity. For example, the RMS pose error was 0.6742 m when sudden estimation failure happened, while the average RMS pose error was 0.3454 m. As this estimation failure happens frequently, the error variance becomes large as shown in Fig. 6 with the range on the graph. Even though the original FastSLAM shows quiet nice performance, these failures cause the problem of consistency; the algorithm cannot always be trusted. Compared to the RSR, the GRR performed 32.7% less pose error during the process, along with 39.1% less error standard deviation. This result shows that GRR alleviates estimation failure problem, and the performance became more consistent with lower error variance. The improvement over feature estimation error can also be observed in Fig. 7. GRR showed 35.3% less feature error and 53.5% less error standard deviation than RSR.

As the GRR algorithm needs time for building a KD tree and selecting particles satisfying the conditions, the computation time increases. However, as a KD tree is a simple and fast structure, so average run-time increased only by 4.2%. The performance results are summarized in Table III.

https://doi.org/10.1017/S0263574708005250 Published online by Cambridge University Press

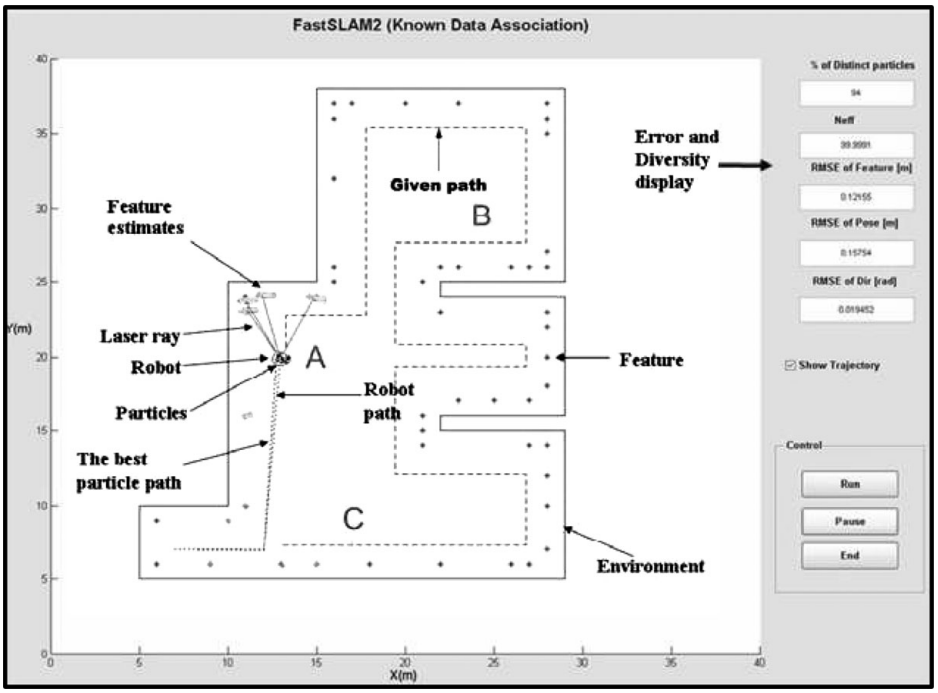


Fig. 4. Simulation environment: 25 m × 37 m indoor map with 50 features. The robot runs one lap around the environment in one simulation and 50 simulations were run using RSR and GRR algorithm separately.

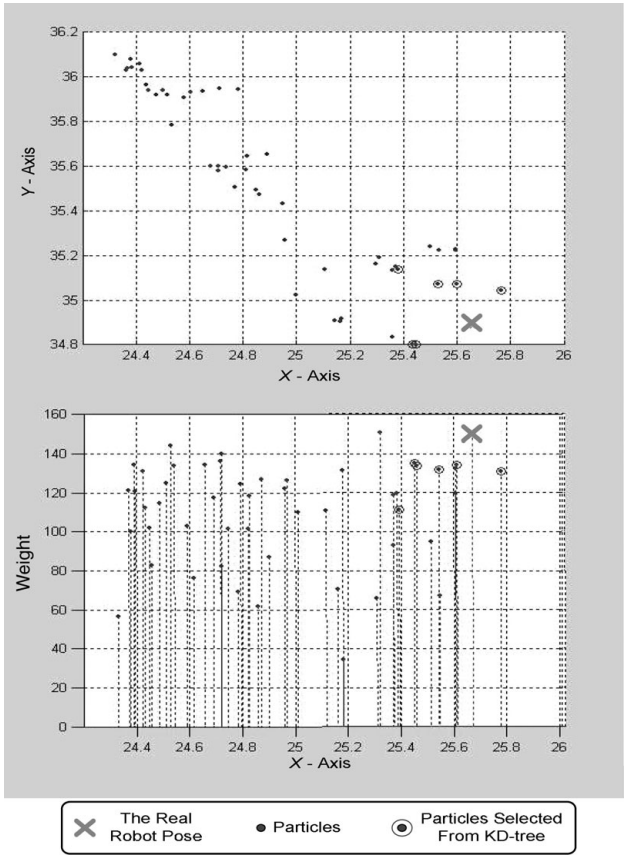


Fig. 5. Example of particle distribution at a single time step. Dots with circles are particles selected from the KD tree satisfying condition of the GRR. Selected particles are estimating the real robot pose more correctly than other particles.

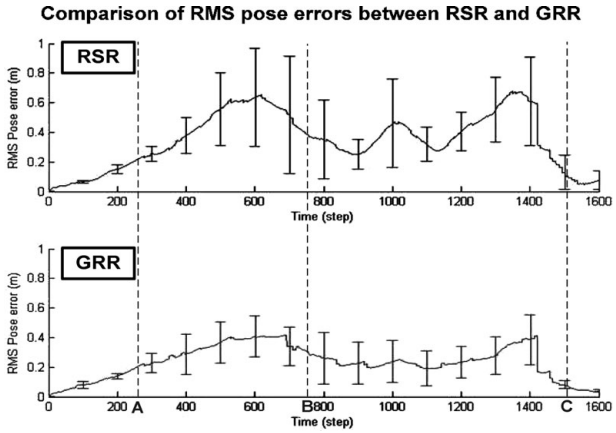


Fig. 6. Comparisons of RMS pose error and error standard deviation between RSR and GRR.

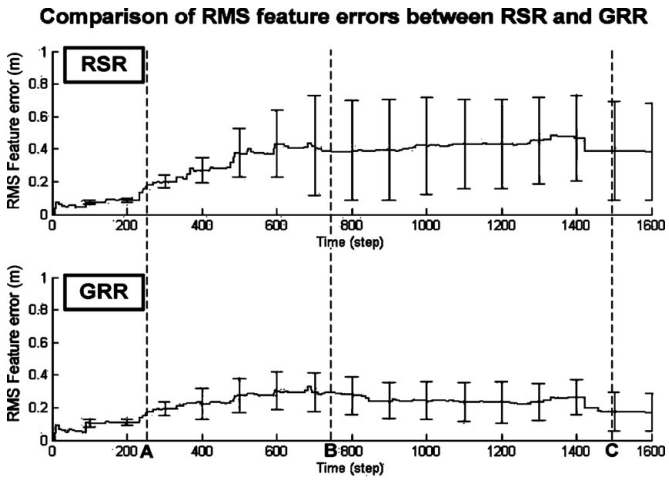


Fig. 7. Comparisons of RMS feature error and error standard deviation between RSR and GRR.

https://doi.org/10.1017/S0263574708005250 Published online by Cambridge University Press

Table III. Performance results of RSR and GRR.

	GRR	RSR	Remarks
Avg. RMS pose error	0.2326 m	0.3454 m	−32.7%
Avg. RMS pose error standard deviation	0.1010 m	0.1658 m	−39.1%
Avg. RMS feature error	0.2175 m	0.3361 m	−35.3%
Avg. RMS feature error standard deviation	0.0927 m	0.1998 m	−53.5%
Avg. run-time	174.49 s	167.51 s	+4.2%

5. Conclusions

The FastSLAM has been shown to inconsistent due to the sudden estimation failure by losing particle diversity. Particle depletion problem, one of the major reasons of the loss of particle diversity, happens in resampling by throwing away particles, resulting in the lack of the number of particles estimating the pose of the robot and the environment correctly. Resampling only according to the weights of particle causes errors and estimation error variances to grow when the weight function fails. Thus, in this work, GRR technique was proposed to reduce the estimation error and the error standard deviation to reduce estimation failure problem and increase performance consistency. The GRR technique uses the KD tree to build up the tree containing particle geometric relation. The KD tree is used in the resampling process to restrain particles to go awry from the real pose of the robot, which reduces chance of estimation failure. The GRR technique efficiently constrained estimation error variances as well as estimation error, compared to the previous FastSLAM using the RSR technique.

As a future work, implementation of the GRR technique will be conducted on a real mobile robot. In addition, research on use of particle vector information, along with other geometric tools like density tree will be studied.

Acknowledgments

This work was supported in part by MIC & IITA through IT Leading R&D Support Project, the Seoul R&BD Program (10689M92991), the MOCIE Industrial Technology Development Program, the ASRI, and the BK21 Information Technology at Seoul National University.

References

1. S. Thrun, W. Burgard and D. Fox, *Probabilistic Robotics* (MIT Press, Cambridge, MA, 2005).
2. M. Montemerlo, FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem With Unknown Data Association *Ph.D. Thesis* (Carnegie Mellon University, Pittsburgh, PA, 2003).
3. A. Doucet, N. de Freitas, K. Murphy and S. Russell, "Rao-Blackwellized Particle Filtering for Dynamic Bayesian Networks," *Proceedings of the 2000 Conference on Uncertainty in Artificial Intelligence* (Stanford University, Stanford, CA, 2000).
4. K. Murphy, "Bayesian Map Learning in Dynamic Environments," *Proceedings of the Advances in Neural Information Processing Systems* (MIT Press, Cambridge, MA, 1999).
5. M. Montemerlo and S. Thrun, "Simultaneous Localization and Mapping with Unknown Data Association using FastSLAM," *Proceedings of the IEEE International Conference on Robotics and Automation* (Taipei, 2003) pp. 1985–1991.
6. T. Bailey, J. Nieto and E. Nebot, "Consistency of the FastSLAM Algorithm," *Proceedings of the IEEE International Conference on Robotics and Automation* (Orlando, FL, 2006) pp. 424–429.
7. M. Bolic, P. M. Djuric and S. Hong, "Resampling algorithms for particle filters: A computational complexity perspective," *Eurasip J. Appl. Signal Process* **15**, 2267–2277 (2004).
8. R. Merwe, A. Doucet, N. de Freitas and E. Wan, "The Unscented Particle Filter," *Technical Report CUED/F INFENG/TR*, 380 (Cambridge University Engineering Department, 2000).
9. P. Elinas, R. Sim and J. J. Little, "SLAM: Stereo Vision SLAM Using the Rao-Blackwellised Particle Filter and a Novel Mixture Proposal Distribution," *Proceedings of the IEEE International Conference on Robotics and Automation* (Orlando, FL, 2006) pp. 1564–1570.
10. S. Thrun, D. Fox and W. Burgard, "Monte Carlo Localization with Mixture Proposal Distribution," *Proceedings of the American Association for Artificial Intelligence* (MIT Press, Cambridge, MA, 2000) pp. 859–865.
11. 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* vol. 140(2) (1993), pp. 107–113.
12. N. Kwak, I. K. Kim, H. C. Lee and B. H. Lee, "Adaptive Prior Boosting Technique for the Efficeint Sample Size in FastSLAM," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (San Diego, CA, 2007) pp. 630–635.
13. A. Doucet and N. J. Gordon, "Simulation-based optimal filter for maneuvering target tracking," *SPIE Proc.* **3809**, 241–255 (1999).
14. J. S. Liu, R. Chen and T. Logvinenko, "A Theoretical Framework for Sequential Importance Sampling with Resampling," *Proceedings of the Sequential Monte Carlo Methods in Practice* (Springer, New York, NY, 2001) pp. 225–246.
15. G. Grisetti, G. D. Tipaldi, C. Stachniss, W. Burgard and D. Nardi, "Fast and Accurate SLAM with Rao-Blackwellized Particle Filters," *Rob. Autonom. Syst.* **55**, 30–38 (2007).
16. S. Lee and S. Lee, "Recursive Particle Filter with Geometric Constraints for SLAM," *Proceedings of the IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems* (Heidelberg, 2006) pp. 395–401.
17. N. Kwak, G. W. Kim and B. H. Lee, "A new compensation technique based on analysis of resampling process in FastSLAM," *Robotica* vol. 26(2) (2008) pp. 205–217.
18. M. de Berg, M. van Krevel and M. Overmars, *Computational Geometry: Algorithms and Applications* (Springer, New York, NY, 1997).
19. S. Omohundro, "Bumptrees for Efficient Function, Constraint, and Classification Learning," *Proceedings of the Advances in Neural Information Processing Systems* vol. 3 (Denver, CO, 1990) pp. 693–699.