FastSLAM with Look-ahead RBPF

Steven Gao
Department of Computer Science
University of British Columbia
Vancouver, BC V6T 1Z4, Canada
sxgao@cs.ubc.ca

Reza Lotun
Department of Computer Science
University of British Columbia
Vancouver, BC V6T 1Z4, Canada
rlotun@cs.ubc.ca

Abstract

In this paper we present an implementation of the Rao-Blackwellised particle filtering (RBPF) with one step look-ahead and apply the algorithm within the domain of agent navigation. Specifically we tackle the simultaneous localization and mapping problem (SLAM), which describes how a agent must concurrently attempt to determine its location and generate a map of the surrounding landmarks. Our implementation is built on top of previous implementation of normal RBPF using a technique called fastSLAM. We compare the performance of normal RBPF and look-ahead RBPF in terms of computational time and accuracy of state estimation.

1 Introduction

We apply a variant of a Rao-Blackwellised particle filtering to the FastSLAM algorithm [1]. In this section we introduce various concepts used in our work, namely particle filtering, Rao-Blackwellisation, the FastSLAM algorithm, and look-ahead RBPF.

1.1 Simultaneous Localization and Mapping (SLAM)

The problem of simultaneously determining of map of the environment and one's location within it is widely regarded [2] as being one of the fundamental problems in robotics. At an abstract level the problem seems to mirror the classic 'chicken-and-egg' problem - "location" makes sense only with respect to a map or model of the environment, and determining such a map of course depends where one is within the environment.

In SLAM, an agent or robot's state within an environment is given by (x, y, ϕ) where x and y are its Cartesian (surface) coordinates within the environment and ϕ is its orientation angle. The moving robot makes noisy measurements of the environment usually with laser range finders, which give the range and bearing to various features or landmarks.

Formally, we can model the environment to be F uniquely distinguishable landmarks. A *map* of the environment Θ consists of the relative distance (component-wise) to each of the F landmarks, if they are visible. The robot's current state $z^t = (x, y, \phi)$ is where in Θ it currently is at time t. Its control signal u^t is a displacement which directs it where in the environment it should move to next. We denote by y^t its measurements at time t, given by a list of landmarks it sees and relative distances to them. The problem of SLAM then is to

determine

$$p(z^t, \Theta | y^t, u^t) \tag{1}$$

known as the SLAM posterior.

1.2 Monte Carlo Methods

Given some target probability density function $p(x)$ defined on a high-dimensional space, we seek to draw an i.i.d set of samples $\{x^{(i)}\}_{i=1}^N$. The simplest Monte Carlo simulation of such a distribution $p(x)$ would be

$$p_N(dx) = \frac{1}{N} \sum_{i=1}^{N} \delta_{x^{(i)}}(dx)$$

where $\delta_{x^{(i)}}$ denotes the Dirac-delta function at $x^{(i)}$. Such an approach would allow the approximation of intractable integrals (as the integral over the Dirac-delta function would "pick out" the function in the integral evaluated at the samples).

If the target distribution $p(x)$ is difficult to sample from, then another Monte Carlo variant called *importance sampling* can be used instead. The idea is to use a simpler to evaluate *proposal distribution* $q(x)$ to sample from instead. A weight is then calculated

$$w(x) = \frac{p(x)}{q(x)}$$

and thus approximated integrals involving $p(x)$ become

$$\int p(x)dx = \sum_{i=1}^{N} p(x^{(i)})w^{(i)}$$

For dynamic models, *particle filtering* is the generalization of Monte Carlo sampling. Dynamic models consist of three equations: the initial probability $p(x_0)$, the transition probability $p(x_t|x_{t-1}), t \leq 1$ and the observation model $p(y_t|x_t), t \leq 1$. The observations y_t are assumed to be conditionally independent given the process x_t and of the marginal distribution $p(y_t|x_t)$.

In dynamic settings, inference falls into three categories:

1. Filtering: $p(x_t|y_{1:t})$
2. Prediction: $p(x_{t+\tau}|y_{1:t})$
3. Smoothing: $p(x_{t-\tau}|y_{1:t})$

In the filtering problem we have a *prediction* and *filtering* step:

$$p(x_t|y_{1:t-1}) = \int p(x_t|x_{t-1})p(x_{t-1}|y_{1:t-1})dx_{t-1} \tag{2}$$

$$p(x_t|y_{1:t}) = \frac{p(y_t|x_t)p(x_{t-1}|y_{1:t-1})}{\int p(y_t|x_t)p(x_t|y_{1:t-1})dx_t} \tag{3}$$

For high dimensional state spaces, estimates typically exhibit high variance. The idea of *Rao-Blackwellised sampling* is to sample on some subspace of the large state space, and compute the expect value of the rest of the space analytically, all of which result in reduced variance in the estimate.

1.3 Approaches to SLAM

Approaches to SLAM include using an Extended Kalman Filter (EKF) [3] to represent the robot's internal map and pose estimate by a high-dimensional Gaussian over all features in the map and robot states. The limitations of this approach are computational in nature [4], as maintaining such a multivariate Gaussian requires time quadratic in the number of features in the map. Another approach using a Thin Junction Tree Filter (TJTF) [5] represents approximation of the belief state as a junction tree. For each filter update the junction trees is periodically thinned by efficient maximum likelihood projections. Such a representation has a linear-space belief state and linear-time filtering operation. Another approach using Rao-Blackwellised particle filtering in an algorithm called FastSLAM [1, 6, 4] is presented in the next section.

1.4 FastSLAM

As pointed out in [7], the SLAM posterior 1 can be factored as

$$p(z^t, \Theta | y^t, u^t) = p(z^t | y^t, u^t) \prod_{n=1}^{F} p(\Theta_n | z^t, y^t) \tag{4}$$

This says that calculation of the posterior over robot paths and internal maps can be decomposed into $F + 1$ recursive estimators, one over the robot state $p(z^t|y^t, u^t)$ and F separate estimators over feature locations $p(\Theta_n|z^t, y^t)$.

In FastSLAM, the posterior over robot paths is estimated using a particle filter, or more specifically a Rao-Blackwellised particle filter since it only considers a subspace of the possible space. The remaining F posterior of feature locations (which are conditional on the sampled robot pose z^t) are calculated by using extended Kalman filters (EKF). Since each EKF estimates a single landmark position (Θ_x^i, Θ_y^i), it is low-dimensional.

Each particle is of the form [4]

$$S_t^{[n]} = \langle z^{t,[n]}, \mu_{1,t}^{[n]}, \Sigma_{1,t}^{[n]}, \ldots, \mu_{F,t}^{[n]}, \Sigma_{F,t}^{[n]} \rangle \tag{5}$$

that is, at the t^{th} step in the robot's path, $S_t^{[n]}$ denotes the n^{th} particle out of all N particles, where $z^{t,[n]}$ is a position sample, $\mu_{i,t}^{[n]}$ is the mean of a landmark, and $\Sigma_{i,t}^{[n]}$ is its covariance matrix.

The filtering step $-$ generating a S_t from S_{t-1} $-$ involves using the control signal u_t and observation y_t in the following steps:

1. Sampling a new pose

$$z_t^{[n]} \sim p(z_t | z_{t-1}^{[n]}, u_t) \tag{6}$$

   where $p(z_t|z_t^{[n]}, u_t)$ is our "motion model".

2. Updating the observed landmark estimates. If a landmark is observed, which can be determined by y_t,

$$\mu_{t|t-1}^{(i)} = A(z_t^{(i)})\mu_{t-1}^{(i)} + F(z^{(i)})u_t \tag{7}$$

$$\Sigma_{t|t-1}^{(i)} = A(z_t^{(i)})\Sigma_{t-1}^{(i)}A(z_t^{(i)})^T + B(z_t^{(i)})B(z_t^{(i)})^T \tag{8}$$

$$y_{t|t-1}^{(i)} = C(z_t^{(i)})\mu_{t|t-1}^{(i)} + G(z_t^{(i)})u_t \tag{9}$$

$$ST^{(i)} = C(z_t^{(i)})\Sigma_{t|t-1}^{(i)}C(z_t^{(i)})^T + D(z_t^{(i)})D(z_t^{(i)})^T \tag{10}$$

for given A, B, C, D, F, and the Kalman update is then given by

$$\mu_t^{(i)} = \mu_{t|t-1}^{(i)} + \Sigma_{t|t-1}^{(i)} C(z_t^{(i)})^T S T^{-1(i)} (y_t - y_{t|t-1}^{(i)}) \quad (11)$$

$$\Sigma_t^{(i)} = \Sigma_{t|t-1}^{(i)} - \Sigma_{t|t-1}^{(i)} C(z_t^{(i)})^T S T^{-1(i)} C(z_t^{(i)}) \Sigma_{t|t-1}^{(i)} \quad (12)$$

3. Resampling is carried out according to the following weights

$$w_t^{[n]} = \frac{p(z^{t,[n]}|y^t, u^t)}{p(z^{t,[n]}|z^{t-1}, u^t)} \quad (13)$$

$$\propto p(y_t|y_{1:t-1}, z_{[n],1:t}) \quad (14)$$

which we take to be a Gaussian with mean $y_{t|t-1}^{(i)}$ and covariance ST. The derivation of this can be found in [4, 8].

1.5 Lookahead Rao Blackwellised Particle Filtering

In [8] a variant of Rao-Blackwellised particle filtering is presented with these differences:

1.

$$w_t^{[n]} \propto p(y_t|y_{1:t-1}, z_{[n],1:t})$$

$$\propto \sum_{z_t=1}^{n_z} p(y_t|y_{1:t-1}, z_{1:t-1}, z_t) p(z_t|z_{1:t-1}, y_{1:t-1}) \quad (15)$$

where n_z are all the possible states the robot could be in. Since this is a marginalization, it is an exact analytical solution.

2. Since the importance weights do not depend on z_t (because we are marginalizing over them), we can select particles before the sampling step, allowing the choice of the fittest particles at time $t-1$ using the information at time t.

2 Our Contribution

This section describes our contribution to the SLAM problem and discusses our implementation of look-ahead RBPF to this particular application domain.

2.1 Implementation Issues

Our code is built on top of a Matlab implementation of RBPF by van Loh Wenzel [9]. As a result we attempted to improve upon his code in terms of modularity, functionality and efficiency. We were able to in several ways. Firstly, we were able to vectorize many of the functions used heavily by the RBPF code. This reduced the running time of RBPF even for a large number of particles ($N = 200$).

Secondly, we altered the vision model to include a parameter (ϕ) for the robot's angle of vision. This implies that the robot can only detect features within its angle of vision. For example, if the robot is facing 45 degrees North of East (NE) and its vision angle is 90 degrees, then it can see any obstacles bound by North and East axis. The implementation in [9] uses a simpler vision model, which employs eight identical laser rays that leave the robot at fixed angles. So the robot would be able to detect features surrounding it at the specified fixed angles regardless of what angle it was facing. We feel that our model provides a more realistic model of how a robot (or human) would detect features in an environment. We discuss our vision model further in the next subsection.

Since RBPF requires a prior distribution to update the pose of the robot, the algorithm requires an initial set of particles that must be sampled without any "help". Depending on how one decides to sample the initial particles the robot's perform may vary considerably. If the initial set of particles are set close to the real pose of the robot, then the initial sampling will obviously represent the target distribution more accurately and our task is simplified. However, if the initial set of samples are selected uniformly at random, then SLAM will be much harder to perform. This is because the random samples could potentially be very far away from target distribution and so it will take more time steps for the particles to converge.

2.2 Robot Orientation

Previously, the discrete states z_t were modeled as coordinate positions $(x, y), 1 \leq x \leq W_x, 1 \leq y \leq W_y$. To make the state space more realistic, we added an orientation angle ϕ to each state (x, y, ϕ). Given state z_t - representing the position the robots believes itself to be in at step t - and a control signal u_t, $z'_t = (x', y', \phi')$ is calculated to be:

$$\begin{aligned} r &= \sqrt{u_{t_x}^2 + u_{t_y}^2} \\ \phi' &= \arctan(\frac{u_{ty}}{u_{tx}}) - \phi \\ x' &= r\sin(\phi + \phi') \\ y' &= r\cos(\phi + \phi') \end{aligned}$$

3 Testing and Results

To compare our look-ahead RBPF algorithm against normal RBPF we used a 10 x 10 grid world (small.data). The world contains a perimeter wall surrounding the outside edges. The world has exact 42 uniquely numbered features. The map is symmetric about the diagonal (from bottom left corner to top-right corner), which increases the difficulty of accurately mapping the features since the robot could see the same sequence of obstacles but actually be at one of two possible locations in the map. To compare the two algorithms we decided to calculate the average Euclidean distance between a given particle and the true position of the robot - we only considered the x and y coordinates. Thus, greater the distance between the proposed position of the particle the true position the greater the amount of error. Given that there were N particles we weighted each Euclidean distance value for one timestep and calculated the average over these weighted error measures. We also did the same thing for the mapped feature positions (i.e. calculate the Euclidean distance between the estimated position of a feature and that features true position). We averaged the values for both the localization error and map error over all timesteps to arrive at a single for each for a single run of the algorithm. We then plotted these two error measures versus the computational time required per timestep and also versus the number of particles used. We ran each configuration exactly three times and then averaged the results over those runs.

We also ran tests first using no noise in the scanned observations (i.e. simulating the fact that the robot's detection system is completely accurate) and then using a moderate amount of noise (i.e. weighting the covariance matrix high). We present the results of both no noise and moderate noise in the following subsections. Also, as noted above in our contributions section, our results reflect the fact that we do not use seeds for the random number generators (i.e. the Matlab functions rand and randn) and thus there is quite a bit variation between runs using the same number of particles and map.

Originally we had planned to run tests using larger sized maps (e.g. a 20 x 20 grid world),

containing more difficult configurations of features. However, we found that our look-ahead RBPF algorithm did not sale well to larger maps; although the increase in computational time per timestep was linear in the increase in size of the map it was still too slow to perform adequate trial runs.

3.1 Results using Zero Noise

In Figure 1 we see that using the same number of particles both RBPF and look-ahead RBPF perform roughly the same, although RBPF appears to do localization better. This is very discouraging considering that look-ahead RBPF is theoretically supposed to perform better than RBPF because it is using the optimal proposal distribution. It is most likely that our technique of marginalization over all discrete states in the map contains a logical bug. Note, that in our code we only marginalize over discrete states that are not features (i.e. open spaces). We assumed that the robot could be in four possible orientations (i.e. the angle it is facing). So for our given map we marginalize over $(100 - 42)x4 = 216$ states.

In Figure 2 we see a more discouraging result. For computation times per timestep RBPF flat out beats look-ahead RBPF in terms of accuracy. Our first thought was that perhaps because we were using zero noise in the scan readings that look-ahead, its advantage being that it can calculate weightings over all possible states before resampling, provided no advantage because the robot's observations at each timestep were completely accurate. However, we will see in the next subsection, when we used a moderate level of noise in the scan readings, RBPF still outperformed look-ahead RBPF.

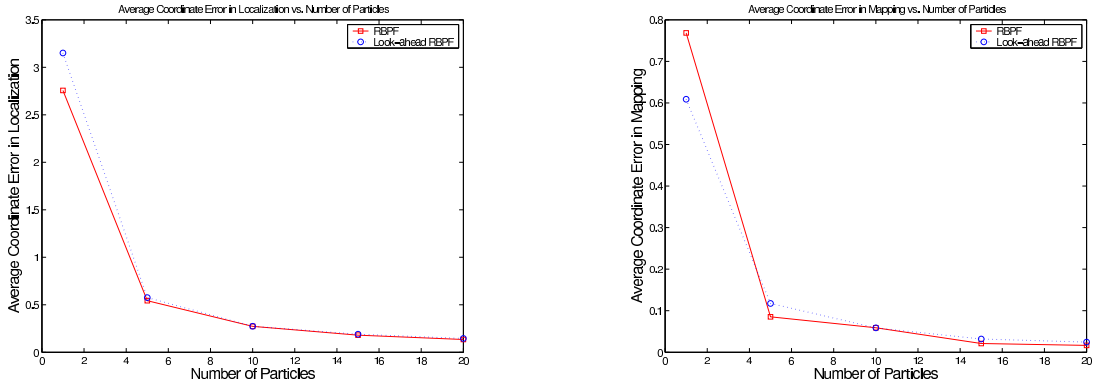

Figure 1: Small Map Localization Error in Low Noise

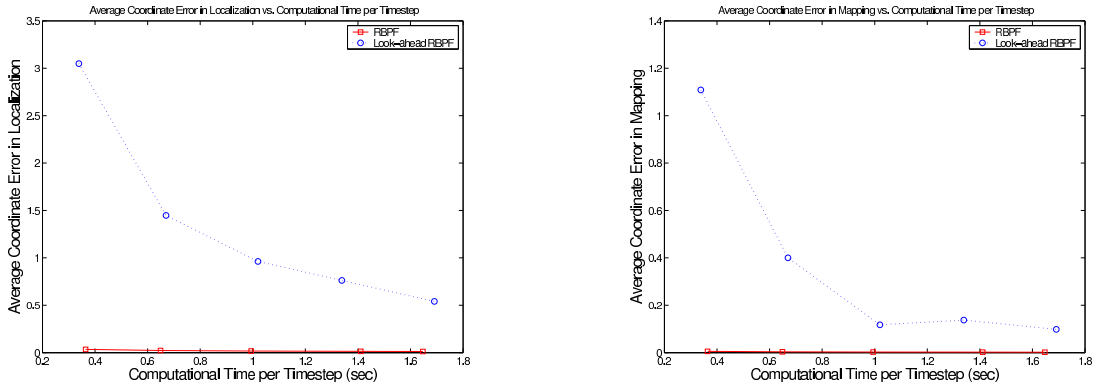

Figure 2: Small Map Localization Error vs. Time in Low Noise

3.2 Results using Moderate Noise

In Figures 3 and 4 we see again that RBPF performs better than look-ahead RBPF in almost every situation. The only exception is when both algorithms use only a single particle. In this particular case, look-ahead produces slightly more accurate estimation of the robot's position and much better map estimation of feature locations (i.e. look-ahead RBPF produces roughly half the amount of error as RBPF in this case). However, in terms of computational time per timestep, which is the key to an algorithm's success, it appears that even when there is noise in the scan readings RBPF is more reliable. This result appears questionable because in principle look-ahead RBPF is designed to outperform RBPF in this situation. Given that our observations maybe unreliable it is far better to be able to estimate using all possible discrete states.

At this point in our analysis we are quite sure that there is an error in our implementation and not simply because we are applying look-ahead to a new application domain. As was discussed in [8] when the number of discrete states is small (i.e. roughly speaking less than 1000 states) the weighting calculations can be computed exactly. Given that the total number of states is 216,the look-ahead RBPF algorithm should be tractable and produce very accurate results.

4 Conclusion

Based on our comparison of RBPF and look-ahead RBPF we can only conclude that there is an error in our implementation of look-ahead RBPF. Although analysis from [8] shows that

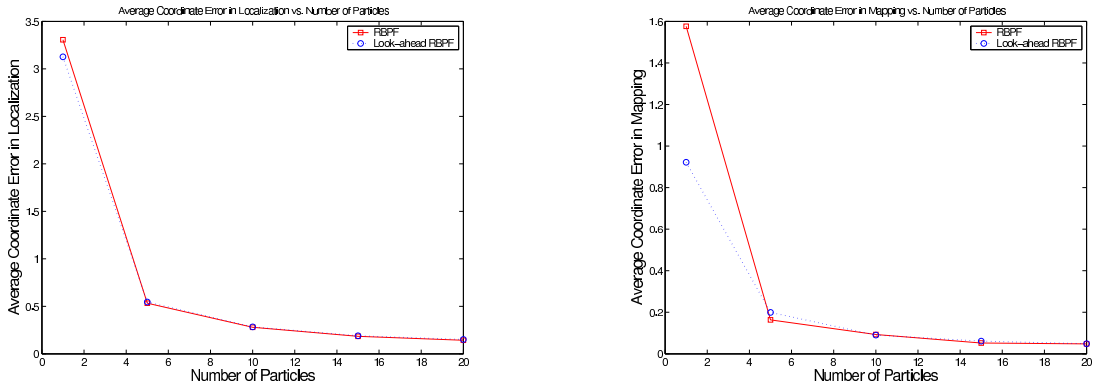

Figure 3: Small Map Localization Error in High Noise

the accuracy of RBPF and look-ahead RBPF eventually converge, given that the number of particles is increased high enough, look-ahead should outperform RBPF for few number of particles. Granted there will be differences in the performance results of both algorithms when they are applied to another application domain, namely SLAM, the dramatic evidence that RBPF outperforms look-ahead in every situation except when only a single particle is used should be a warning flag that indicates our implementation may contain a bug.

Aside from this, our results suggest that in the SLAM domain, for the experiments we performed, using more particles is better. Although our RBPF implementation used a simple proposal distribution, using a large number of particles made it fairly accurate. Although the computational burden of each particle in lookahead is large (for it grows with the map), the benefits gained should have been more substantial. More extensive tests are needed to determine the relationship between the two approaches.

5 Future Work

The immediate extensions to this algorithm involve dealing with some basic assumptions made, code optimizations and comparison with other methods:

- The body of work mentioned above assumes known map correspondences. This is an obvious simplification not applicable in reality. The application of the codebase to unknown data correspondence could be made using the method of maximum-likelihood discussed in [4].
- Currently, many parts of the FastSLAM algorithm are implemented naively. The

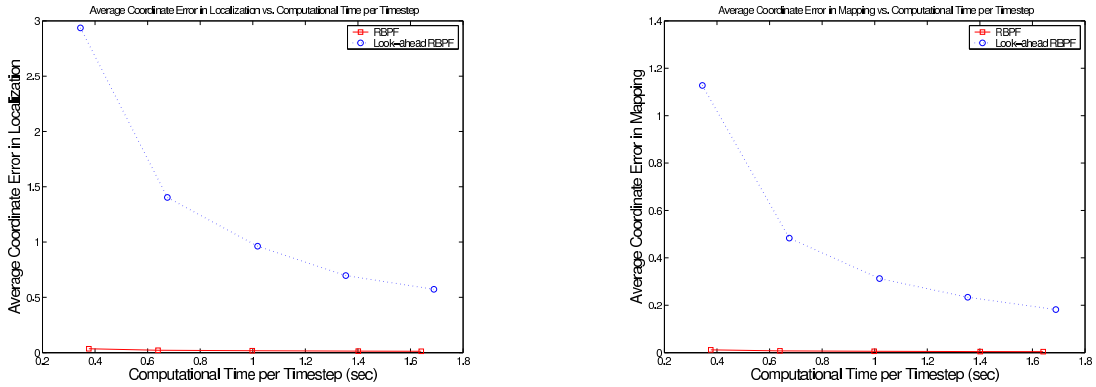

Figure 4: Small Map Localization Error vs. Time in High Noise

use of advanced datastructures [10] to update the map mean and covariances for all the particles, such as a binary tree [1], can offer some badly needed optimizations.

- An improved version of FastSLAM dubbed FastSLAM 2.0 [4, 6] takes into account the measurements y^t. An implementation of these ideas and how lookahead applies to them could be made. A more thorough comparison between FastSLAM 1.0, FastSLAM 2.0, with and without lookahead can be made against the Thin Junction Tree Filter method [5] to see in what regime each excels at.

A few more interesting extensions involve reducing the computational complexity of the lookahead method and applying it to a non-trivial problem:

- "Fast-methods" - those family of methods related to fast multi-pole methods, the fast Gauss transform, distance transforms, kd-trees, ball trees, dual trees, etc. - can be used to try to reduce the complexity of the marginalization at each step.

- The FastSLAM+Lookahead variant can be applied to the simple Wumpus World problem, which would require the use of some form of decision processes.

Acknowledgements

We wish to thank Kevin Murphy for fruitful discussions as well as James Cook, Iryna Shypnyk and Roland von loh Wenzel for their Matlab code which we based our work on.

References

[1] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. Fastslam: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the National Conference on Artificial Intelligence*, 2002.

[2] S. Thrun. Probabilistic algorithms in robotics. *AI Magazine*, 21(4):93–109, 2000.

[3] M. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localization and map building (slam) problem. *IEEE Transactions on Robotics and Automation*, 17(3):229–241, 2001.

[4] S. Thrun, M. Montemerlo, D. Koller, B. Wegbreit, J. Nieto, and E. Nebot. Fastslam: An efficient solution to the simultaneous localization and mapping problem with unknown data association. *Journal of Machine Learning Research*, 2004. To appear.

[5] Mark A. Paskin. Thin junction tree filters for simultaneous localization and mapping. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 1157–1164, San Francisco, CA, 2003. Morgan Kaufmann Publishers.

[6] M. Montemerlo and S. Thrun. Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 735–744, 2001.

[7] K. Murphy. Bayesian map learning in dynamic environments. In *Advances in Neural Information and Processing Systems (NIPS)*. MIT Press, 1999.

[8] N. de Freitas, R. Dearden, F. Hutter, R. MoralesMenendez, J. Mutch, and D. Poole. Diagnosis by a waiter and a mars explorer. In *Invited paper for Proceedings of the IEEE, special issue on sequential state estimation*, 2003.

[9] R. W. van Loh Wenzel. Giving a compass to a robot probabilistic techniques for simultaneous localization and map building (slam) in mobile robotics. 2004.

[10] U. Frese. Treemap: An $o(logn)$ algorithm for simultaneous localization and mapping. In C. Freksa, editor, *Spatial Cognition IV*. Springer Verlag, 2004.