Real-time Particle Filters Using Mixtures of Samples Sets

Cody Kwok[†] **Dieter Fox**[†] **Marina Meila**[‡]

[†]Dept. of Computer Science & Engineering, [‡]Dept. of Statistics
University of Washington
Seattle, WA 98195

Abstract

Recently, particle filters have been applied with great success to a variety of state estimation problems. In many real time applications, sensor information arrives significantly faster than the particle filter can process. The prevalent approach to this problem is to update the particle filter as often as possible and to discard sensor information that cannot be processed in time. In this paper we present an alternative approach. Our real time particle filter makes use of all sensor information by partitioning sample sets into mixtures of smaller sample sets. Each small sample set contains as many samples as can be processed between the arrival of two observations. These small sets are combined as soon as the total number of samples in the mixture is sufficiently high. Using efficient subsampling, we determine the mixture weights so as to minimize the KL-divergence between the mixture density and the true posterior. Thereby, our approach focuses computational resources on valuable sensor information. Experiments using data collected with a mobile robot show that our approach yields drastic improvements over alternative techniques.

Introduction

Estimating the state of a dynamic system from noisy sensor measurements is extremely important in areas as different as speech recognition, target tracking, mobile robot navigation, and computer vision. Particle filters have been applied with great success to a variety of state estimation problems (Doucet, de Freitas, & Gordon 2001; Isard & Blake 1998; Fox *et al.* 1999; Lenser & Veloso 2000; Schulz *et al.* 2001; Vermaak *et al.* 2002). The success of particle filters is mostly due to their efficiency and ability to represent a wide range of probability densities. The key idea of this filter technique is to represent probability densities by sets of samples, or particles. The efficiency of particle filters lies in the way they place computational resources. By sampling in proportion to likelihood, particle filters focus the computational resources on regions with high likelihood, where things really matter.

The application of particle filters under real-time constraints (Dean & Boddy 1988; Zilberstein & Russell 1995; Horvitz 2001) introduces several important research questions. The sample based representation of particle filters is well suited for an any-time implementation: Whenever an estimate of the system state is needed, sampling can be interrupted and an approximate estimate can be generated. Since the variance of the sample based approximation decreases with the number of samples, the quality of the solution/estimate increases with the computational resources. However, due to the dependencies between sample sets at subsequent time steps, interrupting the generation of samples too early can result in divergence of the filter. Statistical error bounds provide means for determining the number of samples necessary to achieve a desired approximation quality, and the minimum number of samples required strongly depends on the complexity of the underlying distribution (Fox 2001). This minimum imposes a lower bound on the computational complexity of the particle filter. However in many real-time applications, sensor information may arrive faster than this lower bound, and cannot be processed fully. Therefore, a key question is how to make optimal use of the available information given the limited time available.

A prevalent approach to this problem is to update the filter as often as possible, and to discard sensor information that arrives during the update process. In this paper we introduce an alternative approach. Instead of discarding sensor readings, we integrate all of them into proportionally smaller sample sets. These sets are then recombined into a mixture distribution that is propagated into the next estimation cycle, which eliminates the problem of filter divergence. Furthermore, in order to minimize approximation error, our approach places more computational resources on more valuable sensor information. This is achieved by weighting the sample sets inside the mixture.

The remainder of this paper is organized as follows: In the next section we outline the basics of Bayes filters and particle filters. Then we introduce our novel technique to real-time particle filters. An efficient approach to choosing the weights of the mixture components is described next, followed by experimental results.

Bayes filters and particle filters

In this section we briefly review the standard Bayes filter and particle filter (our approach can easily be extended to more complex techniques such as Rao-Blackwellised particle filters (Doucet *et al.* 2000)). Bayes filters address the problem of estimating the state x of a dynamical system from sen-

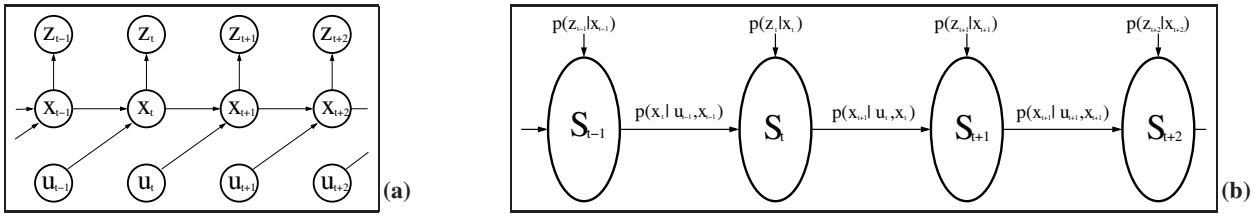

Figure 1: a) Recursive Bayes filter: State x_t only depends on the previous state x_{t-1} and the control information u_{t-1}. Observations z_t solely depend on the current state. b) Implementation as particle filter. The state is represented by sample sets S_t. The sample set at time t is generated by drawing samples from the previous set using the control information u_{t-1}. Observations are integrated by weighting each sample using the likelihood of the observation $p(z_t \mid x_t)$ as importance weights.

sor measurements. The key idea of Bayes filtering is to recursively estimate the posterior probability density over the state space conditioned on the data collected so far. The data consists of an alternating sequence of time indexed observations z_t and control measurements u_t. The posterior at time t is called the belief $Bel(x_t)$, defined by

$$Bel(x_t) \quad = \quad p(x_t \mid z_t, u_{t-1}, z_{t-1}, u_{t-2} \ldots, u_0, z_0)$$

Bayes filters make the assumption that the dynamic system is Markov, *i.e.* observations z_t and control measurements u_t are conditionally independent of past measurements and control readings given knowledge of the state x_t (cf. 1(a)). Under this assumption the posterior belief can be updated efficiently using the following two update rules: Whenever a new control measurement u_{t-1} is received, the state of the system is *predicted* according to

$$Bel^-(x_t) \quad \longleftarrow \quad \int p(x_t \mid x_{t-1}, u_{t-1})\; Bel(x_{t-1})\; dx_{t-1} \qquad (1)$$

and whenever an observation z_t is made, the state estimate is *corrected* according to

$$Bel(x_t) \quad \longleftarrow \quad \alpha\; p(z_t \mid x_t) Bel^-(x_t)\,. \qquad (2)$$

Here, α is a normalizing constant which ensures that the belief over the entire state space sums up to one. $p(z_t \mid x_t)$, the observation model, describes the likelihood of making observation z_t given that the current state is x_t. The dynamics of the system are modeled by the conditional probability $p(x_t \mid x_{t-1}, u_{t-1})$.

Implementations of Bayes filters mostly differ in the way they represent densities over the state x_t. For example, Kalman filters are Bayes filters which make the restrictive assumption that the posterior can be represented by Gaussian distributions. Multi hypothesis tracking allows to represent multi modal state densities (Bar-Shalom & Fortmann 1988). Discrete, piecewise constant representations are able to approximate a wide range of distributions (Koller & Lerner 2001; Burgard *et al.* 1996).

Particle filters

Particle filters are a variant of Bayes filters which represent the belief by sets S_t of weighted samples $\langle x_t^{(i)}, w_t^{(i)} \rangle$, distributed according to $Bel(x_t)$. Here each $x_t^{(i)}$ is a state, and the $w_t^{(i)}$ are non-negative numerical factors called *importance weights*, which sum up to one. The basic form of the particle filter realizes the recursive Bayes filter according to a sampling procedure, often referred to as sequential importance sampling with resampling (SISR) (Doucet, Godsill, & Andrieu 2000; Doucet, de Freitas, & Gordon 2001).

Resampling: Draw with replacement a random state x from the set S_{t-1} according to the (discrete) distribution defined through the importance weights $w_{t-1}^{(i)}$. This state can be seen as an instance of the belief $Bel(x_{t-1})$.

Sampling: Use x and the control information u_{t-1} to sample x' according to the distribution $p(x' \mid x, u_{t-1})$, which describes the dynamics of the system.

Importance sampling: Weight the sample x' by the observation likelihood $w' = p(z_t \mid x')$.

Each iteration of these three steps generates a sample $\langle x', w' \rangle$ drawn from the posterior belief $Bel(x_t)$. After n iterations, the importance weights of the samples are normalized so that they sum up to one. It can be shown that this procedure in fact approximates the Bayes filter, using a sample-based representation (Doucet, Godsill, & Andrieu 2000). The temporal evolution of particle filters is illustrated in Figure 1)b).

Real time particle filters

So far we assumed that Bayes filters can be updated whenever new sensor information arrives. In many applications however, the sensor model in (2) cannot be fully evaluated before the next sensor measurement arrives. More formally, we assume that observations arrive at intervals Δ called *observation interval*. For a fixed number of samples, the computing/estimation cycle of the particle filter takes $k\Delta$ and is called the *estimation interval*. The *window size* of the filter is the number of observations k that arrive during one update of the filter. An estimation interval is denoted by t. Inside the interval, observations arrive at times $t_i, i = 1, \ldots k$. The state at time t_i is denoted by x_{t_i} and the corresponding observation by z_{t_i}.

The vast majority of filtering approaches deals with the problem of limited computational power by simply skipping sensor information that arrives during the update of the filter (see *e.g.* (Fox *et al.* 1999)). While this approach may work well in many situations, it is prone to miss valuable sen-

sor information. An illustratiuon of this approach for window size three is given in Figure 2(a). Using this approach, two observations have to be skipped during the update of the sample set.

At first glance, particle filters appear to suggest a very elegant, any time solution to this problem (Dean & Boddy 1988). One could interrupt the generation of new samples whenever a sensor measurement arrives. This technique, depicted in Figure 2(b), adapts the number of samples to the available time (smaller sample sets are illustrated by the smaller ellipses). While this approach makes use of all observations, it is not feasible in general since the error of the sample based approximation increases as the number of samples decreases (Doucet, de Freitas, & Gordon 2001). If the number of samples is too small, the estimate of the particle filter diverges (Fox 2001).

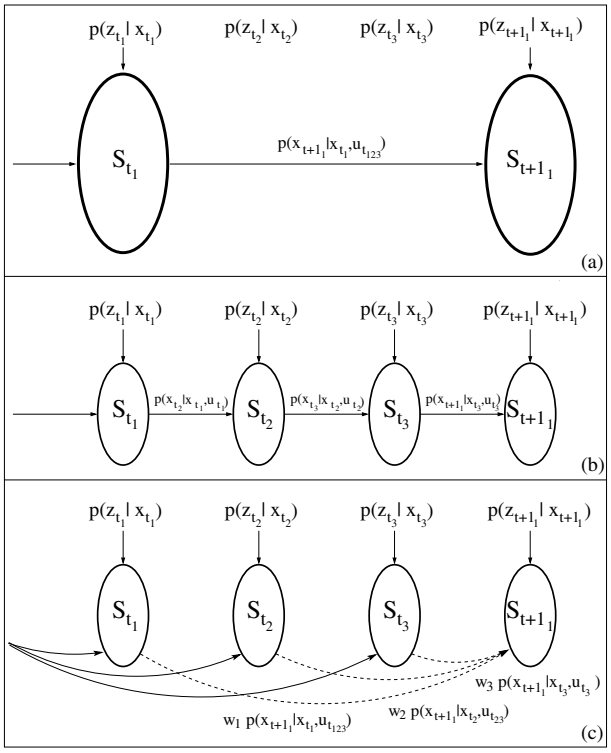

Figure 2: Different strategies for dealing with limited computational power. All approaches process the same number of samples per estimation interval (three observations). (a) Skip observations. All samples are used for one set and only every third observation is integrated. (b) Partition the samples into smaller sets and integrate each observation. (c) Same as in (b), but sample sets represent a mixture density. The w_i denote the weights of the mixture components.

We propose an alternative approach which makes use of the flexible, sample based representation of particle filters while avoiding the problem of divergence due to insufficient sample set sizes. The idea of our method is to partition the sample set into k smaller sample sets, each incorporating exactly one of the observations. So, instead of one sample set at time t, we maintain k smaller sample sets at $t_i, i = 1, \ldots k$. We treat such a "virtual sample set", or belief, as a mixture of the distributions represented in it. The approach is illustrated in Figure 2(c). Compared to the first approach, this method has the advantage of not skipping any observations. The difference to the second approach (Figure 2(b)) is more subtle. In approach (c), the estimation interval contains k observations, and k sample sets of size n/k. The belief state that is propagated to the next estimation interval is a mixture distribution where each mixture component is represented with one of the k sample sets. Thus, the belief state propagation is simulated by $n \cdot \frac{n}{k}$ sample trajectories, that for computational convenience are represented at the points in time where the observations are integrated. In approach (b) however, the estimation interval equals the observation interval, therefore the belief propagation is simulated with only n/k samples.

We will now show how to determine the weights of the mixture belief. The key idea is to choose the weights that minimizes the KL-divergence between the mixture belief and the optimal belief. The optimal belief is the belief we would get if there was enough time to compute the full posterior. For illustration purpose, we will first describe this approach by only considering observations. The derivation including the system dynamics will be given in the section after.

Observations only

Let us restrict our attention to one estimation interval consisting of k observations. The optimal belief $Bel_{opt}(x)$ for such an interval results from iterative application of the Bayes filter update rule (2)

$$Bel_{opt}(x) \quad = \quad \prod_{i=1}^{k} \alpha_i \; p(z_i \mid x) \; Bel^{-}(x). \qquad (3)$$

Here the time index t is omitted, and $Bel^{-}(x)$ denotes the belief at the beginnning of the estimation interval.

Let $Bel_i(x)$ denote the belief resulting from integrating only the $i-th$ observation within the estimation interval. $Bel_{mix}(x \mid w)$, the belief resulting from mixing the individual beliefs $Bel_i(x)$, can be computed by

$$\begin{aligned} Bel_{mix}(x \mid w) \quad &= \quad \sum_{i=1}^{k} w_i \; Bel_i(x) \\ &= \quad \sum_{i=1}^{k} w_i \; \alpha_i \; p(z_i \mid x) Bel^{-}(x) \quad (4) \end{aligned}$$

where $w_i \geq 0$ and $\sum_i w_i = 1$. The α_i are the normalization constants resulting from each observation integration. The mixing weights w_i reflect the "importance" of the respective observations for describing the optimal belief. The idea is to set these weights so as to minimize the approximation error introduced by the mixture distribution. More formally, we determine the mixing weights w^* by minimizing the KL-divergence (Cover & Thomas 1991) between Bel_{mix} and Bel_{opt}

$$w^* \; = \; \underset{w \in \mathcal{W}}{\operatorname{argmin}} \; KL(Bel_{mix}(\cdot \mid w) || Bel_{opt}(\cdot))$$

$$= \operatorname*{argmin}_{w \in \mathcal{W}} \int Bel_{mix}(x \mid w)\ log\ \frac{Bel_{mix}(x \mid w)}{Bel_{opt}(x)} dx \quad (5)$$

In the above we denoted $\mathcal{W} = \{w \mid \sum_{i=1}^{k} w_i = 1,\ w_i \geq 0\}$. Before we describe an efficient algorithm for optimizing these weights, we show how to consider the system dynamics.

Observations and dynamics

To compute the optimal belief under consideration of the system dynamics, one has to perform k belief updates using both (1) and (2):

$$\begin{aligned} Bel_{opt}(x_\tau) \quad = \quad & \int \ldots \int p(x_\tau \mid x_{t_k}, u_{t_k}) \cdot \\ & \prod_{i=1}^{k} \alpha_i\ p(z_{t_i} \mid x_{t_i})\ p(x_{t_i} \mid x_{t_{i-1}}, u_{t_{i-1}}) \cdot \\ & Bel(x_{t-1})\ dx_{t-1}, dx_{t_1}, \ldots, dx_{t_k} \end{aligned} \quad (6)$$

Here τ denotes a time in the future. The time index t_0 represents the previous estimation interval $t-1$. In essence, (6) computes the belief by integrating over all *trajectories* through the estimation interval. The probability of each trajectory is determined using the control information $u_{t-1}, u_{t_1}, \ldots, u_{t_k}$, and the likelihoods of the observations along the trajectory.

The mixture belief including dynamics is given by

$$\begin{aligned} Bel_{mix}(x_\tau) \quad = \quad & \sum_{i=1}^{k} w_i \int \ldots \int p(x_\tau \mid x_{t_k}, u_{t_k}) \cdot \\ & \alpha_i\ p(z_{t_i} \mid x_{t_i}) \prod_{j=1}^{k} p(x_{t_j} \mid x_{t_{j-1}}, u_{t_{j-1}}) \cdot \\ & Bel(x_{t-1})\ dx_{t-1} dx_{t_1} \ldots dx_{t_k} \end{aligned} \quad (7)$$

Here, too, we integrate over all trajectories. In contrast to (6), however, each trajectory selectively passes through only one of the k observations in the estimation interval. We will now turn to the problem of finding the weights of the mixture belief.

Optimizing the mixture representation

Minimizing the KL-divergence

Optimizing the weights of mixture approximations is often done using EM (Poland & Shachter 1993) or (constrained) descent using the gradient (Jaakkola & Jordan 1997). Unfortunately, running several iterations of EM or using any other iterative method is too expensive in our case. Therefore, we are going to use a simple heuristic, that requires at most k gradient computations. The heuristic exploits the convexity of the KL divergence which guarantees, under mild technical conditions, that our problem (5) has exactly one solution.

Denote by $J(w)$ the criterion to be minimized in (5). We will compute its gradient in a few selected points and then use a linear interpolation scheme to approximate the minimum of J in the domain $\mathcal{W}$.

The gradient of J is given by

$$\frac{\partial J}{\partial w_i} = 1 + \int Bel_i(x)\ \log\ \frac{Bel_{mix}(x|w)}{Bel_{opt}(x)}\ dx \quad (8)$$

where $i = 1, \ldots k$.

Let us denote by w^i, w^{mid} respectively the corners and the middle point of the domain $\mathcal{W}$

$$\begin{aligned} w^i \quad &= \quad [\,0 \quad 0 \quad \ldots \quad \underset{i}{1} \quad \ldots \quad 0], \quad i = 1, \ldots k \\ w^{mid} \quad &= \quad [\tfrac{1}{k} \quad \tfrac{1}{k} \quad \ldots \quad \ldots \quad \tfrac{1}{k}] \end{aligned} \quad (9)$$

We compute the gradients at these points, $g_i = \nabla J(w^i)$, $g_{mid} = \nabla J(w^{mid})$. Then we compute an approximate minimum along each line segment $\overline{w^i, w^{mid}}$ and finally we take the arithmetic mean of these minima to be our approximation to the global minimum of J in the domain $\mathcal{W}$.

MCMC gradient estimation

The computation of the gradients in (8) requires integration over all possible states x and the computation of the different beliefs for each of these states, each in turn requiring the computation of up to k integrals (see (4), (6), and (7)). This is clearly too expensive to compute in our case. We solve this problem by Monte Carlo approximation.

First, we generate trajectories using the previous belief (represented by a mixture of weighted sample sets) and the control information $u_{t-1}, u_{t_1}, \ldots, u_{t_k}$. To do so, we draw a sample x_{t-1} from a sample set S_{t-1_j} of the previous estimation window with probability given by the previous mixture weights w_j. This sample is moved forward in time by drawing a new sample $x_{t_{i+1}}$ from the distribution $p(x_{t_{i+1}} \mid x_{t_i}, u_{t_i})$ at each time step t_i, $i = 1, \ldots k-1$. To simulate forward movement past the current time horizon t_k (for which we have no control information) we "blur" the last sample set by convolving it with a kernel whose width depends on the most recent control values u_{t_i}.

Our next step is to obtain $Bel_i(x_\tau)$ and $Bel_{opt}(x_\tau)$. To estimate $Bel_i(x_\tau)$ we weight each trajectory by $p(z_{t_i} \mid x_{t_i})$ then perform summation and normalization on a grid over x_τ. The use of the grid speeds up computation by replacing the computation of a k-dimensional integral with just one dimension. $Bel_{opt}(x_\tau)$ is estimated by a summation on the same grid, after having incorporated all the k observations in each trajectory.

Finally, the gradient is estimated by formula (8) where the integral is replaced by summation on the grid. Note that $Bel_{mix}(x_\tau|w)$ is either one of $Bel_i(x_\tau)$ or their arithmetic mean so no extra integration is necessary to obtain its values.

The function GRADIENT in the next subsection encapsulates the whole approximation process.

In this scheme, the computation of the mixing weights w_i requires propagating a sample set through all the observations. This in fact implements a particle filter at a time scale k times smaller than the original one. Therefore, we cannot afford to use a large number of samples for this estimation. The number of trajectories generated, n_s, should be significantly smaller than the "regular" number of samples n/k. In our experiments we use an n_s that results in a time spent estimating the mixing weights of about 1% of the total estimation time.

Our gradient estimation method is necessarily noisy, so it is essential to provide sampling distributions that (1)

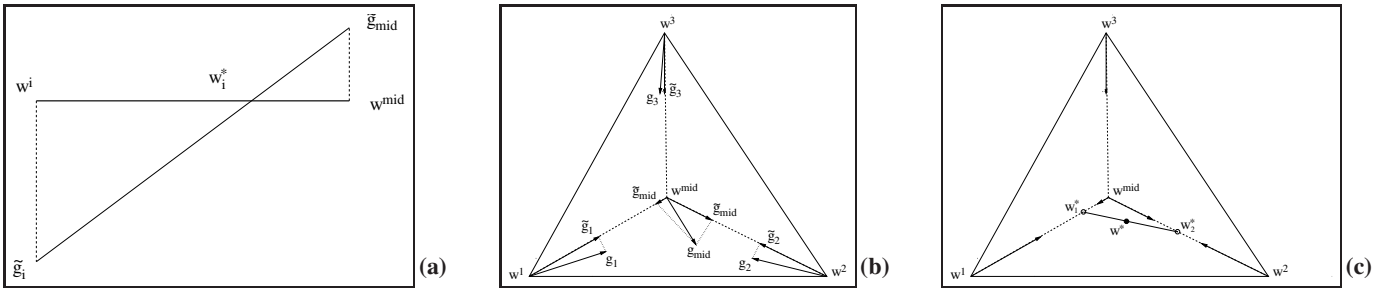

Figure 3: (a) Finding the minimum w_i^* along segment $\overline{w^i w^{mid}}$ by linear interpolation. (b,c) Illustration of the APPROXIMATE-MIN algorithm for $k = 3$. (b) The gradients in the corners and middle are projected onto the segments that connect the middle with the corners. (c) Along segments $\overline{w^1 w^{mid}}$, $\overline{w^2 w^{mid}}$ the minima w_1^*, w_2^* are computed. The approximation of the global minimum w^* is the middle of $\overline{w_1^* w_2^*}$. The projected gradient has no zero on the third segment $\overline{w^3 w^{mid}}$ so that no w_3^* is computed.

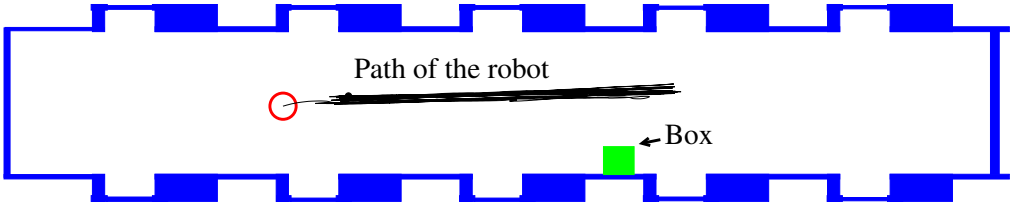

Figure 4: Map and path of the robot used for the experiments.

cover all the regions where the estimated distributions Bel_i, Bel_{opt} are non-zero; and (2) recycle randomness(), *i.e.*use the same random bits in evaluating the diverse scenarios of incorporating one or another of the observations. The simple trajectory generation algorithm described above satisfies this requirement, but is "wasteful of samples" since many trajectories will be weighted very low by the observations. We are considering replacing this method with a more efficient one in future work.

The mixture weights optimization algorithm

The algorithm can be summarized as follows:

Algorithm APPROXIMATE-MIN
Input GRADIENT(w) a function that approximates $\nabla J(w)$

compute $g_{mid} =$ GRADIENT(w^{mid})
for $i = 1, 2, \ldots k$
  compute $g_i =$ GRADIENT(w^i)
  project g_i, g_{mid} on the line segment $\overline{w^i, w^{mid}}$
  obtaining $\tilde{g}_i, \tilde{g}_{mid}$
  if $\tilde{g}_i < 0,\ \tilde{g}_{mid} > 0$
    there is a minimum w_i^* of J inside $\overline{w^i, w^{mid}}$
    we approximate it linearly as in figure 3a)
  else if $\tilde{g}_i < 0,\ \tilde{g}_{mid} < 0$
    the minimum of J is beyond the end of the segment, do nothing
  else if $\tilde{g}_i > 0,\ \tilde{g}_{mid} > 0$
    the minimum w_i^* of J along $\overline{w^i, w^{mid}}$ is at w^i
  else if $\tilde{g}_i > 0,\ \tilde{g}_{mid} < 0$
    impossible, J is convex
form the set C^* containing all the w_i^* computed in the cycle
compute $w^* = \frac{1}{|C^*|} \sum_{i \in C^*} w_i^*$

Output w^*

Figure 3 illustrates the algorithm. Note that if $\tilde{g}_{mid} < 0$ on some segment $\overline{w^i, w^{mid}}$, it means that there is no minimum of J along the segment, so that the gradient at the opposite end need not be computed. As the gradient g_{mid} will always have a negative projection on at least one segment, it means that in practice we need to compute at most $k-1$ corner gradients. We have omitted this from the above algorithm for the sake of clarity.

Another remark is that in our calculations, ∇J represents the "unconstrained" gradient, i.e the gradient of J without taking into account the constraints. This simplification is justified by the fact that we further project this gradient onto segments contained in the admissible domain $\mathcal{W}$ which has the effect of implicitly satisfying the constraints.

Experiments

In this section we evaluate the effectiveness of our algorithm against the alternatives, using data collected from a mobile robot in a real-world environment. Figure 4 shows the setup of the experiments: the robot was placed in the corridor and moved up and down with a constant velocity of 30 cm/sec. The corridor ($23 \times 4.5\, m^2$, all doors closed) is symmetric except for a single obstacle on its side, and this obstacle can only be detected by the robot's sonar sensors. The robot is only allowed to use the 4 sonar sensors installed on either side of its body. The robot is successfully localized if it is 80% sure that it is within $1m^2$ and 40 degrees of the true position, *i.e.* the sum of importance weights of the samples in this region is greater than 0.8.

In the following discussions, our real-time algorithm is denoted by RT. We compare it against particle filter with skipping observations PF_s(Figure 2a), and with insufficient samples PF_i(Figure 2b). Furthermore, to gauge the efficiency of our mixture weighting, we also obtained results

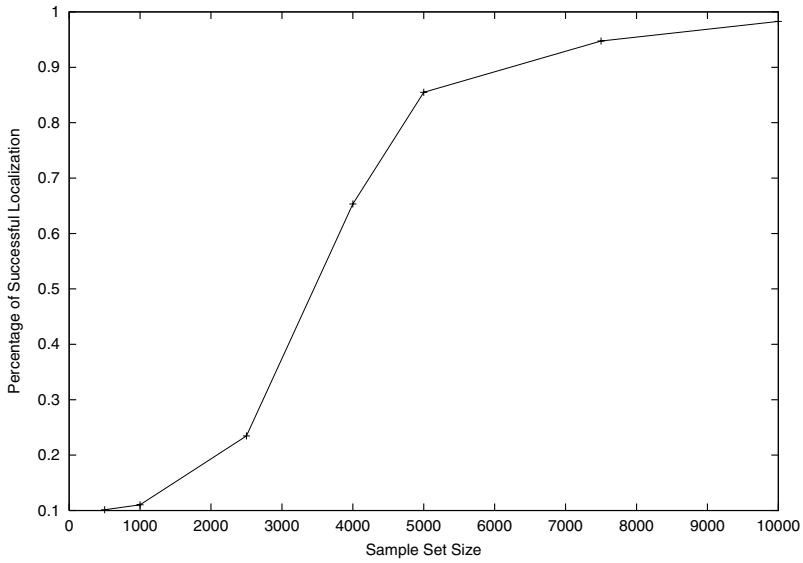

Figure 5: Localization success rates for different sample set sizes. The robot localizes very reliably in this environment given 5000 or more samples, but degrades quickly with less.

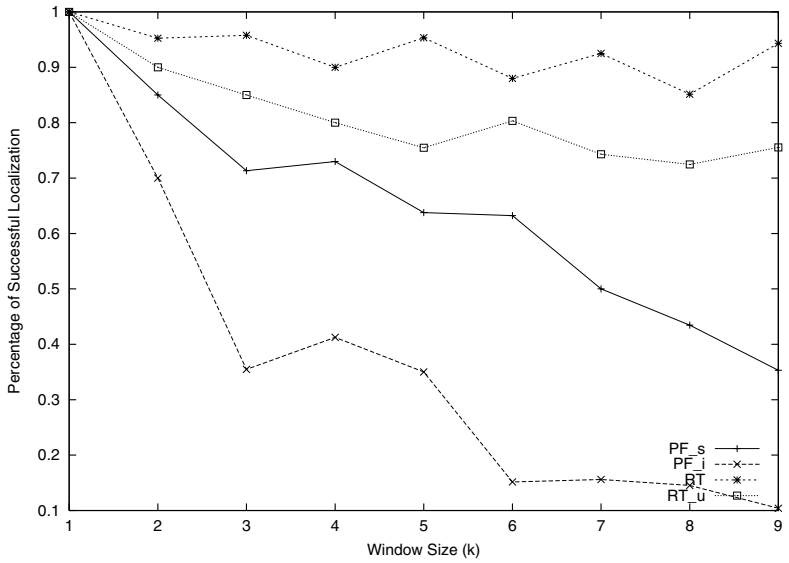

Figure 6: Localization success rates for different window sizes. Larger window size means less samples per observation interval. The real time particle filters outperform others, with the mixture weighted variant maintaining 90% success rate most of the time.

for our real-time algorithm without weighting, and we denote this variant RT_u.

The experiment is set up as follows. First, a fixed sample set size N is chosen. We then vary the number of sample sets k in a window, which implies changing the number of samples n_{obs} that can be generated per observation interval. If k increases, less samples can be generated for each observation interval, and RT, RT_u and PF_s increases their estimation intervals to maintain N. PF_s will skip $k-1$ observations. PF_i will have smaller sample set sizes which are equal to n_{obs}, but will still integrate all the observations.

It should be noted that N has to be chosen carefully because we want to make sure changes in sample set sizes affects the algorithms, which isn't always the case. Figure 5 shows the localization success rate of a particle filter with different sample set sizes in this map. Using more than 5000 samples, the particle filter has more than adequate samples and will almost always localizes successfully. With less than 5000 however, its performance degrades noticably. We therefore choose $N = 5000$.

In the first experiment we compare the localization success rate of the algorithms on different k's. Each data point consists of 40 runs, each run has different start positions in the path shown in Figure 4, and different random seeds. The success rate is simply the percentage of runs in which the robot successfully localizes, defined in the beginning of this section. The result of this experiment is shown in Figure 6. The superiority of RT and RT_u over PF_s and PF_i sug-

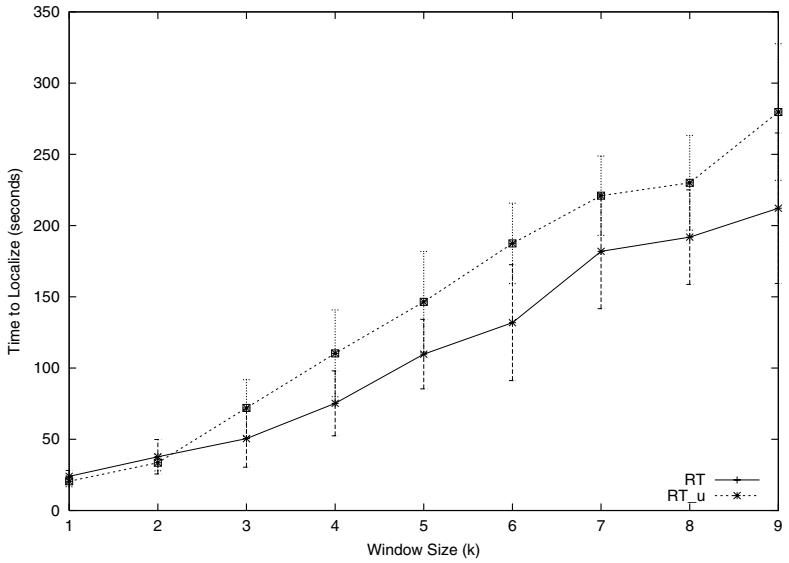

Figure 7: Time taken by real time particle filters to localize. Mixture weighting helps convergence, shown by the shorter time required by RT.

gests that, by spreading out samples into smaller sets, particle filters can maintain their performance given less computational resources. Among the original filters, PF_iś success rate degrades more drastically than PF_s due to the fact that it has less and less samples in a set with increasing k, leading to bigger estimation errors and eventually localization failures. PF_i also performs badly at higher k because it skips $k-1$ observations, which means it is likely to ignore important observations (the box in our map). RT_u maintains a consistent success rate and degrades slowly. Since the smaller sample sets in RT_u are unweighted, the important observations become "diluted" when there are more sets in a window. RT counters this effect by mixture-weighting the sample sets, and is capable of maintaining a success rate of over 90% most of the time.

In the second experiment we compare the time required by the real time particle filters to localize. For RT_u and RT we compute the time required to reach localization success with different k's. Each data point is again generated from 40 different runs. The results, shown in Figure 7, suggests that RT localizes faster than RT_u on average. This demonstrates that by putting more resources on important observations, mixture weighting enables faster convergence.

Conclusions

In this paper we tackled in the context of particle filters the problem of making near-optimal use of the information provided by the sensors, under the constraint of limited computing resources. Our solution is to divide our sample set between all available observations and then to represent the current state of belief as a mixture of the posterior beliefs that would result from taking into account each observation alone. Next we optimize the mixing weights in order to be as close to the true posterior distribution as possible. Our optimization method is a very efficient heuristic that produces significant improvements in a robot localization task.

So far we have used a fixed sample size n and a fixed window size k. The next natural step is to adapt these two "structural parameters" to further speed up the computatation. For example, by the method of (Fox 2001) we can adaptively reduce the sample size n after localization has taken place, which in turn would allow us to reduce the window size k. More complex strategies involving the confidence level of (Fox 2001) are also possible.

Note also that our approach has a broader scope than than particle filtering, applicable with minor modifications to Bayesian estimators in general. In this context, similarity with the method described in (Boyen & Koller 1998) becomes noticeable. Their work also uses mixtures to compute a tractable approximation to a belief state that is propagated through time. It optimize the mixture representation by projecting onto the true posterior. However, they only update the mixture weights, while our mixture components arise naturally from integrating the observation. We are motivated by real-time constraints that are not present in (Boyen & Koller 1998) and our mixture representation is not restricted to one observation interval, but rather spans an entire estimation interval.

References

Bar-Shalom, Y., and Fortmann, T. 1988. *Tracking and Data Association*. Mathematics in Science and Engineering. Academic Press.

Boyen, X., and Koller, D. 1998. Tractable inference for complex stochastic processes. In *Proc. of the Conference on Uncertainty in Artificial Intelligence (UAI)*.

Burgard, W., Fox, D., Hennig, D., and Schmidt, T. 1996. Estimating the absolute position of a mobile robot using position probability grids. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*.

Cover, T. M., and Thomas, J. A. 1991. *Elements of Information Theory*. Wiley Series in Telecommunications. New York: Wiley.

Dean, T. L., and Boddy, M. 1988. An analysis of time-dependent planning. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 49–54.

Doucet, A., de Freitas, J., Murphy, K., and Russell, S. 2000. Rao blackwellised particle filtering for dynamic bayesian networks. In *Proc. of the Conference on Uncertainty in Artificial Intelligence (UAI)*.

Doucet, A., de Freitas, N., and Gordon, N., eds. 2001. New York: Springer-Verlag.

Doucet, A., Godsill, S., and Andrieu, C. 2000. On sequential monte carlo sampling methods for Bayesian filtering. *Statistics and Computing* 10(3).

Fox, D., Burgard, W., Dellaert, F., and Thrun, S. 1999. Monte Carlo Localization: Efficient position estimation for mobile robots. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*.

Fox, D. 2001. KLD-sampling: Adaptive particle filters and mobile robot localization. In *Advances in Neural Information Processing Systems (NIPS)*.

Horvitz, E. 2001. Principles and applications of continual computation. *Artificial Intelligence* 126. Special issue on Tradeoffs under Bounded Resources.

Isard, M., and Blake, A. 1998. Condensation – conditional density propagation for visual tracking. *International Journal of Computer Vision* 29(1):5–28.

Jaakkola, T., and Jordan, M. 1997. Improving the mean field approximation via the use of mixture distributions. In *Learning in Graphical Models*. Kluwer.

Koller, D., and Lerner, U. 2001. Sampling in factored dynamic systems. In Doucet et al. (2001).

Lenser, S., and Veloso, M. 2000. Sensor resetting localization for poorly modelled mobile robots. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*.

Poland, W., and Shachter, R. 1993. Mixtures of gaussians and minimum relative entropy techniques for modeling continuous uncertainties. In *Proc. of the Conference on Uncertainty in Artificial Intelligence (UAI)*.

Schulz, D., Burgard, W., Fox, D., and Cremers, A. B. 2001. Tracking multiple moving targets with a mobile robot using particle filters and statistical data association. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*.

Vermaak, J., Andrieu, C., Doucet, A., and Godsill, S. 2002. Particle methods for bayesian modelling and enhancement of speech signals. *IEEE Transactions on Speech and Audio Processing*. Accepted for publication.

Zilberstein, S., and Russell, S. 1995. Approximate reasoning using anytime algorithms. In Natarajan, S., ed., *Imprecise and Approximate Computation*. Dordrecht: Kluwer Academic Publishers.