Real-time Particle Filters

Cody Kwok† **Dieter Fox**† **Marina Meilă**‡

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

{ctkwok,fox}@cs.washington.edu, mmp@stat.washington.edu

Technical Report UW-CSE-02-07-01

Abstract

Particle filters estimate the state of dynamical systems from sensor information. In many real time applications of particle filters, however, sensor information arrives at a significantly higher rate than the update rate of the filter. The prevalent approach to dealing with such situations 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 real-time particle filters, which make use of *all* sensor information even when the filter update rate is below the update rate of the sensors. This is achieved by distributing samples among the different observations arriving during a filter update. Hence the approach represents posteriors by mixtures of sample sets. The weights of the mixture components are set so as to minimize the approximation error introduced by the mixture representation. Minimization is achieved by gradient descent using efficient Monte Carlo approximation of the gradients. Thereby, our approach focuses computational resources (samples) on valuable sensor information. Experiments using data collected with a mobile robot show that our approach yields strong improvements over other approaches.

1 Introduction

Due to their sample-based representation, particle filters are well suited to estimate the state of non-linear dynamic systems. Over the last years, particle filters have been applied with great success to a variety of state estimation problems including visual tracking [1], speech recognition [2], mobile robot localization [3, 4, 5], map building [6], people tracking [7, 8], and fault detection [9, 10]. A recent book provides an excellent overview of the state of the art [11]. The increased representational power of particle filters, however, comes at the cost of higher computational complexity.

The application of particle filters to online, real-time estimation raises new research questions. The key problem in this context is: *How can we deal with situations in which the rate of incoming sensor data is higher than the update rate of the particle filter?* To the best of our knowledge, this problem has not been addressed in the literature so far. The prevalent approach in real time applications is to update the filter as often as possible and to discard sensor information that arrives during the update process. Obviously, this approach is prone to loosing valuable sensor information. At first sight, the sample based representation of particle filters suggests an alternative approach similar to an any-time implementation: Whenever a new observation arrives, sampling is interrupted and the next observation is processed. Unfortunately, such an approach can result in too small sample sets, causing the filter to diverge [11, 12].

In this paper we introduce real-time particle filters (RTPF) to deal with constraints imposed by limited computational resources. Instead of discarding sensor readings, we distribute the samples among the different observations arriving during a filter update. Hence RTPF represents densities over the state space by *mixtures* of sample sets, thereby avoiding the problem of filter divergence due to an insufficient number of independent samples. The weights of the mixture components are computed so as to minimize the approximation error introduced by the mixture representation. The resuling approach naturally focuses computational resources (samples) on valuable sensor information.

The remainder of this paper is organized as follows: In the next section we outline the basics of particle filters in the context of real-time constraints. Then, in Section 3, we introduce our novel technique to real-time particle filters. Finally, we present experimental results followed by a discussion of the properties of RTPF.

2 Particle filters

Particle filters are a sample-based variant of Bayes filters, which recursively estimate posterior densities, or beliefs Bel, over the state x_t of a dynamical system (see [13] for details):

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

Here z_t is a sensor measurement and u_{t-1} is control information describing the dynamics of the system. Particle filters are Bayes filters which represent beliefs by sets S_t of weighted samples $\langle x_t^{(i)}, w_t^{(i)} \rangle$. 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):

1. *Resampling:* Draw with replacement a random state x from the set S_{t-1} according to the (discrete) distribution defined through the importance weights $w_{t-1}^{(i)}$.

2. *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.

3. *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 $\langle x', w' \rangle$ representing the posterior. After n iterations, the importance weights of the samples are normalized so that they sum up to one. Particle filters can be shown to converge to the true posterior even in non-Gaussian, non-linear dynamic systems [14].

A typical assumption underlying particle filters is that all samples can be updated whenever new sensor information arrives. Under realtime conditions, however, it is possible that the update cannot be completed before the next sensor measurement arrives. This can be the case for computationally complex sensor models or whenever the underlying posterior requires large sample sets [12]. The majority of filtering approaches deals with this problem by skipping sensor information that arrives

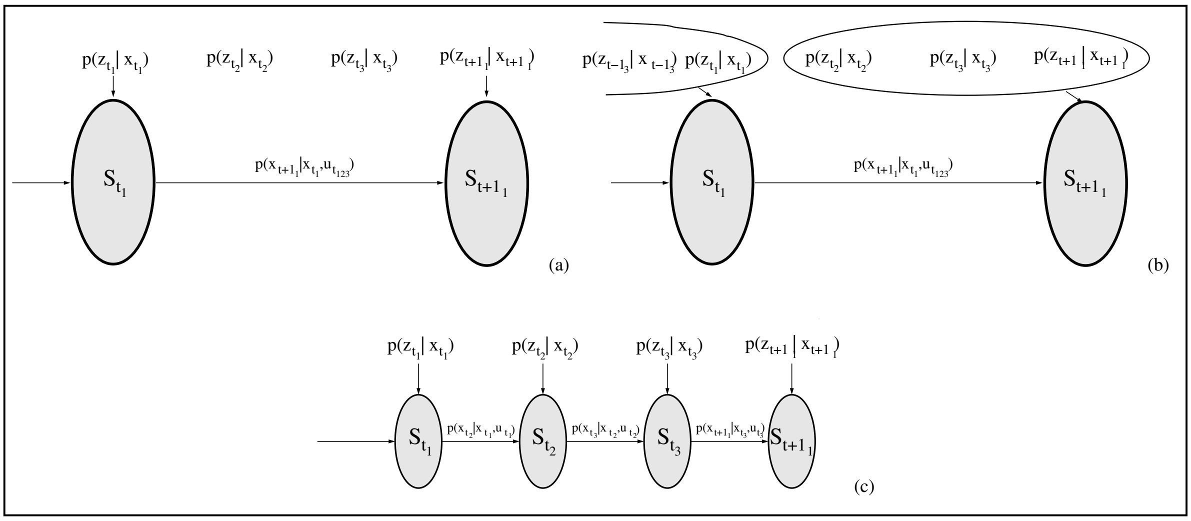


Figure 1: Different strategies for dealing with limited computational power. All approaches process the same number of samples per estimation interval (window size three). (a) Skip observations, *i.e.* integrate only every third observation. (b) Aggregate observations within a window and integrate them in one step. (c) Reduce sample set size so that each observation can be considered.

during the update of the filter. While this approach works reasonably well in many situations, it is prone to miss valuable sensor information.

Before we discuss different ways of dealing with such situations, let us introduce some notation. We assume that observations arrive at time intervals Δ, which we will call *observation intervals*. Let n be the number of samples required by the particle filter. Assume that the resulting update cycle of the particle filter takes $k\Delta$ and is called the *estimation interval* or *estimation window*. Accordingly, k observations arrive during one estimation interval. We call this number the *window size* of the filter, *i.e.* the number of observations obtained during a filter update. Estimation intervals (windows) are indexed by $\ldots t-1, t, t+1, \ldots$. Inside an estimation interval t, the k 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}.

For a window of size one, the filter can be updated on each observation. Fig. 1 illustrates different approaches to dealing with window sizes larger than one. The simplest and most common aproach is shown in Fig. 1(a). Here, observations arriving during the update of the sample set are discarded, which has the obvious disadvantage that valuable sensor information might get lost. The approach in Fig. 1(b) overcomes this problem by aggregating multiple observations into one. While this technique avoids the loss of information, it is not applicable to arbitrary dynamical systems. For example, it assumes that observations can be aggregated optimally, and that the integration of an aggregated observation can be performed as efficiently as the integration of individual observations, which is often not the case. The third approach, shown in Fig. 1(c), simply stops generating new samples whenever an observation is made (hence each sample set contains only n/k samples). While this approach takes advantage of the any-time capabilities of particle filters, it is susceptible to filter divergence due to an insufficent number of samples [12, 11].

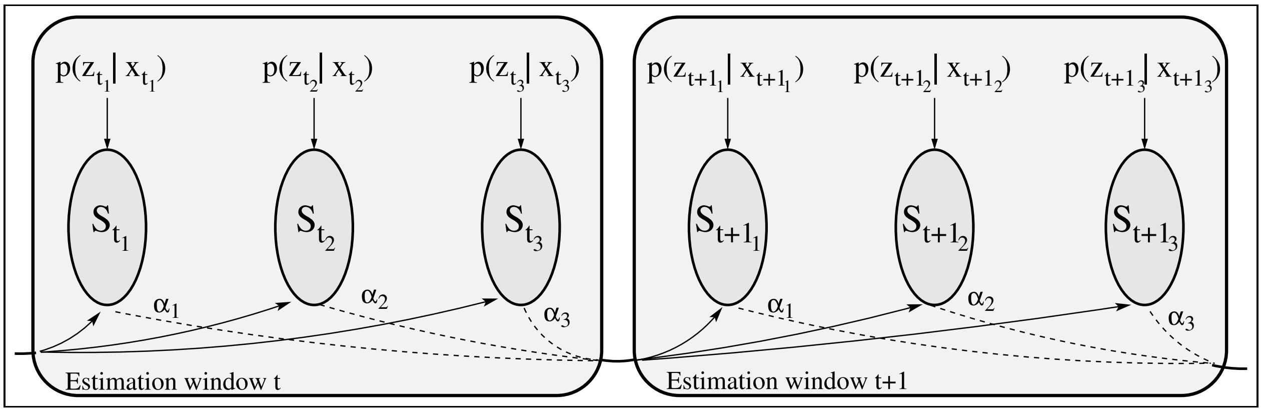


Figure 2: Real time particle filters. The n samples are distributed among the observations within one estimation interval (window of size three). The belief is a mixture of the individual sample sets. For each window the weights α_i of the mixture components are chosen so that the approximation error introduced by the mixture is minimal. Note that each arrow additionally contains the system dynamics $p(x_{t+1_i} \mid x_{t_j}, u_{t_{ij}})$ between time t_j and $t + 1_i$ ($1 \leq i, j \leq 3$).

3 Real time particle filters

In this paper we propose real time particle filters (RTPFs), a novel approach to dealing with limited computational resources. The key idea of RTPFs is to consider *all* sensor measurements by distributing the samples among the observations within an update window. Additionally, by weighting the different samples sets within a window, our approach focuses the computational resources (samples) on the most valuable observations. Fig. 2 illustrates the approach. As can be seen, 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 only technical problem is that the mixture components represent the state of the system at different points in time. However, the complete belief can be generated by keeping track of the dynamics between the individual mixture components.

Compared to the first approach discussed in the previous section, this method has the advantage of not skipping any observations. In contrast to the approach shown in Fig. 1(b), RTPFs do not make any assumptions about the nature of the sensor data, *i.e.* whether it can be aggregated or not. The difference to the third approach (Fig. 1(c)) is more subtle. In both approaches, each of the k sample sets can only contain n/k samples. The belief state that is propagated by RTPF to the next estimation interval is a mixture distribution where each mixture component is represented by one of the k sample sets, all generated independently from the previous window. Thus, the belief state propagation is simulated by $k \cdot \frac{n}{k}$ sample trajectories, that for computational convenience are represented at the points in time where the observations are integrated. In the approach (c) however, the belief propagation is simulated with only n/k independent samples.

We will now show how RTPF determines the weights of the mixture belief. The key idea is to choose the weights that minimize 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 within the update window.

3.1 Mixture representation

Let us restrict our attention to one estimation interval consisting of k observations. The optimal belief $Bel_{opt}(x_{t_k})$ at the end of an estimation window results from iterative application of the Bayes filter update [13]:

$$Bel_{opt}(x_{t_k}) \propto \int \ldots \int \prod_{i=1}^{k} p(z_{t_i} \mid x_{t_i}) \; p(x_{t_i} \mid x_{t_{i-1}}, u_{t_{i-1}}) \; Bel(x_{t_0}) \; dx_{t_0} \ldots dx_{t_{k-1}}. \tag{2}$$

Here $Bel(x_{t_0}$ denotes the belief generated in the previous estimation window. In essence, (2) computes the belief by integrating over all *trajectories* through the estimation interval, where the start position of the trajectories is drawn from the previous belief $Bel(x_{t_0})$. The probability of each trajectory is determined using the control information $u_{t_0}, u_{t_1}, \ldots, u_{t_{k-1}}$, and the likelihoods of the observations $z_{t_1}, \ldots, z_{t_k}$ along the trajectory. Now let $Bel_i(x_{t_k})$ denote the belief resulting from integrating only the $i-th$ observation within the estimation window. RTPF computes a mixture of k such beliefs, one for each observation. The mixture, denoted $Bel_{mix}(x_{t_k} \mid \alpha)$, is the weighted sum of the mixture components $Bel_i(x_{t_k})$, where α denotes the mixture weights:

$$Bel_{mix}(x_{t_k} \mid \alpha) \propto \sum_{i=1}^{k} \alpha_i \int \ldots \int p(z_{t_i} \mid x_{t_i}) \prod_{j=1}^{k} p(x_{t_j} \mid x_{t_{j-1}}, u_{t_{j-1}}) \; Bel(x_{t_0}) \; dx_{t_0} \ldots dx_{t_{k-1}} \tag{3}$$

where $\alpha_i \geq 0$ and $\sum_i \alpha_i = 1$. Here, too, we integrate over all trajectories. In contrast to (2), however, each trajectory selectively integrates only one of the k observations within the estimation interval[1].

3.2 Optimizing the mixture weights

We will now turn to the problem of finding the weights of the mixture. These weights reflect the "importance" of the respective observations for describing the optimal belief. The idea is to set them so as to minimize the approximation error introduced by the mixture distribution. More formally, we determine the mixing weights α^* by minimizing the KL-divergence [15] between Bel_{mix} and Bel_{opt}.

$$\begin{aligned} \alpha^* &= \underset{\alpha \in \mathcal{A}}{\operatorname{argmin}} \; KL(Bel_{mix}(\cdot \mid \alpha) || Bel_{opt}) & (4) \\ &= \underset{\alpha \in \mathcal{A}}{\operatorname{argmin}} \int Bel_{mix}(x_{t_k} \mid \alpha) \; log \; \frac{Bel_{mix}(x_{t_k} \mid \alpha)}{Bel_{opt}(x_{t_k})} \; dx_{t_k}. & (5) \end{aligned}$$

In the above we omitted time indices and denoted $\mathcal{A} = \{\alpha \mid \sum_{i=1}^{k} \alpha_i = 1, \; \alpha_i \geq 0\}$. Optimizing the weights of mixture approximations can be done using EM [16] or (constrained) gradient descent [17]. We perform a small number of gradient descent steps to find the mixture weights. Denote by $J(\alpha)$ the criterion to be minimized in (5). The gradient of $J(\alpha)$ is given by

$$\begin{aligned} \frac{\partial J}{\partial \alpha_i} &= \int \frac{\partial}{\partial \alpha_i} Bel_{mix}(x_{t_k}|\alpha) \log Bel_{mix}(x_{t_k}|\alpha) - \frac{\partial}{\partial \alpha_i} Bel_{mix}(x_{t_k}|\alpha) \log Bel_{opt}(x_{t_k}) \\ &= 1 + \int Bel_i(x_{t_k}) \log Bel_{mix}(x_{t_k}|\alpha) - Bel_i(x_{t_k}) \log Bel_{opt}(x_{t_k}) dx_{t_k} \\ &= 1 + \int Bel_i(x_{t_k}) \; \log \; \frac{Bel_{mix}(x_{t_k}|\alpha)}{Bel_{opt}(x_{t_k})} \; dx_{t_k}, \qquad i = 1, \ldots k. & (6) \end{aligned}$$

[1]Note that typically the individual predictions $p(x_{t_j} \mid x_{t_{j-1}}, u_{t_{j-1}})$ can be "concatenated" so that only two predictions for each trajectory have to be performed, one before and one after the corresponding observation.

The start point α_s for the gradient descent is chosen to be the center of the weight domain $\mathcal{A}$, that is $\alpha_s = [\frac{1}{k}\ \frac{1}{k}\ \ldots\ \frac{1}{k}]^T$.

3.3 Monte Carlo gradient estimation

The exact computation of the gradients in (6) requires the computation of the different beliefs, each in turn requiring several particle filter updates (see (2), (3)), and integration over all states x_{t_k}. This is clearly not feasible in our case. We solve this problem by Monte Carlo approximation.

First, we generate trajectories using the previous belief and the control information $u_{t-1_1}, \ldots, u_{t_{k-1}}$. Since the previous belief is a mixture of weighted sample sets from the previous estimation window, we draw a sample x_{t-1} from a sample set S_{t-1_j} with probability given by the mixture weights α_j of the previous window. This sample is moved forward in time by consecutively drawing samples x_{t_i} from the distributions $p(x_{t_i} \mid x_{t_{i-1}}, u_{t_{i-1}})$ at each time step $t_i,\ i = 1, \ldots k$. The resulting trajectories are drawn from the following proposal distribution f, which is the same as Bel_{opt}, Bel_i, and Bel_{mix}, ignoring observations:

$$f(x_{t_k}) \;=\; \int \!\ldots\! \int \prod_{i=1}^{k} p(x_{t_i} \mid x_{t_{i-1}}, u_{t_{i-1}})\ Bel(x_{t_0})\ dx_{t_0} \ldots dx_{t_{k-1}} \tag{7}$$

Using importance sampling, we obtain sample-based estimates of Bel_i and Bel_{opt} by simply weighting each trajectory with $p(z_{t_i} \mid x_{t_i})$ and $\prod_{j=1}^{k} p(z_{t_j} \mid x_{t_j})$, respectively (compare (2) and (3)). Bel_{mix} is generated with minimal computational overhead by averaging the weights computed for the individual Bel_i distributions. The use of the same trajectories for all distributions has the advantage that it is highly efficient and that it reduces the variance of the gradient estimate. This variance reduction is due to using the same random bits in evaluating the diverse scenarios of incorporating one or another of the observations [18].

Further variance reduction is achieved by using stratified sampling on trajectories. The trajectories are grouped by determining connected regions in a grid over the state space (at time t_1). Neighboring cells are considered connected if both contain samples. To compute the gradients by formula (6), we then perform summation and normalization over the grouped trajectories. Empirical evaluations showed that this grouping greatly reduces the number of trajectories needed to get smooth gradient estimates. An additional, very important benefit of grouping is the reduction of the bias due to different dynamics applied to the different sample sets in the estimation window. In our experiments the number of trajectories is less than 2% of the total number of samples, resulting in a computational overhead of about 1% of the total estimation time.

To summarize, the RTPF algorithm works as follows. The number n of independent samples needed to represent the belief, the update rate of incoming sensor data, and the available processing power determine the size k of the estimation window. Given a window size k, RTPF computes the optimal weights of the mixture distribution at the end of each estimation window. This is done by gradient descent using the Monte Carlo estimates of the gradients. The resulting weights are used to generate samples for the individual sample sets of the next estimation window. To do so, we keep track of the control information (dynamics) between the different sample sets of two consecutive windows. The algorithm is shown in Table 1.

The algorithm is extremely similar to the vanilla particle filter update [11, 19, 13]. However, there are two key differences to the original algorithm. First, instead of drawing samples from a single, previous sample set, RTPF generates samples using the mixture of sample sets from

1. **Inputs:**	
k	*// Number of sample sets in an estimation window*
l	*// Position within current estimation window*
n	*// Total number of samples in a window*
$S_{t-1_1}, \ldots, S_{t-1_k}$	*// Mixture sets of previous estimation window*
$\alpha_{t-1_1}, \ldots, \alpha_{t-1_k}$	*// Weights of the mixture distribution*
$u_{t-1_1}, \ldots, u_{t-1_k}, u_{t_1}, \ldots, u_{t_{l-1}}$	*// Control measurements*
z_{t_l}	*// Current observation*
2. $t := t_l$	*// For simplicity, time t_l is referred to by t*
3. $S_t := \emptyset; \;\; \nu = 0$	*// Current sample set and normalization factor*
4. **for** $i := 1, \ldots, k$ **do**	*// Draw $\frac{n}{k}$ samples from previous estimation window*
5. $\quad n_i = \alpha_{t-1_i} \frac{n}{k}$	
6. $\quad$ **for** $m := 1, \ldots, n_i$ **do**	*// Generate samples proportional to mixture weight*
$\qquad$ *// Resampling: Draw state from S_{t-1_i}*	
7. $\qquad$ Sample an index j from the discrete distribution given by the weights in S_{t-1_i}	
$\qquad$ *// Sampling: Predict next state*	
8. $\qquad$ Sample $x_t^{(m)}$ from $p(x_t \mid x_i, u_i, \ldots, u_{t-1})$ using $x_i^{(j)}$ and $u_{t-1_1}, \ldots, u_{t-1_k}, u_{t_1}, \ldots, u_{t_{l-1}}$	
9. $\qquad w_t^{(m)} := p(z_t \mid x_t^{(m)});$	*// Compute importance weight*
10. $\qquad \nu := \nu + w_t^{(m)}$	*// Update normalization factor including mixture weight*
11. $\qquad S_t := S_t \cup \{\langle x_t^{(m)}, w_t^{(m)} \rangle\}$	*// Insert sample into sample set*
12. $\qquad m := m + 1$	*// Update number of generated samples*
13. **for** $m := 1, \ldots, \frac{n}{k}$ **do**	*// Normalize importance weights*
14. $\quad w_t^{(m)} := w_t^{(m)} / \nu$	
15. $l := l + 1$	*// Increment time index within estimation window*
16. **if** $l = k$ **then**	*// Advance window?*
17. $\quad \alpha_{t_1}, \ldots, \alpha_{t_k} = $ OPT-MIX($S_{t_1}, \ldots, S_{t_k}$)	*// Compute mixture weights using Monte-Carlo gradients*
18. $\quad l := 0$	*// Set index to beginning of next window*
19. **return** S_t	

Table 1. The RTPF algorithm. Window size k is a parameter determined by the desired number n of samples and the processing power of the system. The sample sets $S_{t-1_1}, \ldots, S_{t-1_k}$ along with the weights $\alpha_{t-1_1}, \ldots, \alpha_{t-1_k}$ represent the mixture distribution of the previous estimation window $t-1$ (cf. Figure 2). The function OPT-MIX computes the optimal mixture weights using Monte-Carlo gradient estimation.

the previous estimation window. This mixture is mainly established in steps 4 and 5, where step 4 loops over all mixture sets of the previous window, and step 5 uses the mixture weights to determine the number of samples (n_i) to be drawn from each set. Steps 6–12 are identical to the plain SISR update of particle filters. The second key difference to other particle filters lies in steps 15–18, where the update of the estimation windows is handled. Whenever an estimation window is completed, step 17 of the RTPF algorithm computes the mixture weights using the algorithm described in Sections 3.2 and 3.3.

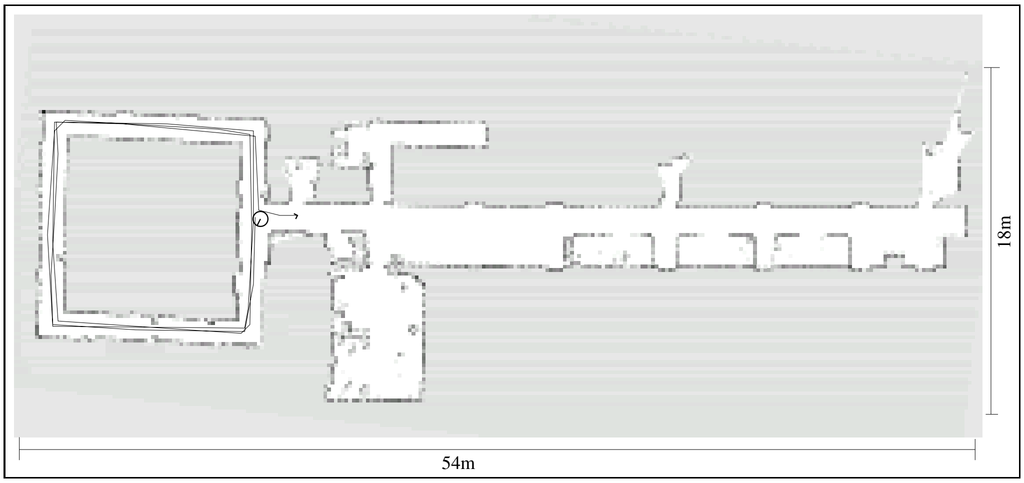


Fig. 3: Map of the environment used for the experiment. A robot was placed in the office floor and moved around the symmetric loop on the left. The task of the robot was to determine its position within the map, using data collected by two distance measuring devices, one pointing to its left, the other pointing to its right.

4 Experiments

In this section we evaluate the effectiveness of RTPF against the alternatives, using data collected from a mobile robot in a real-world environment. Figure 3 shows the setup of the experiment: The robot was placed in the office floor and moved around the loop on the left. The task of the robot was to determine its position within the map, using data collected by two laser-beams, one pointing to its left, the other pointing to its right. The two laser beams were extracted from a planar laser range-finder, allowing the robot only to determine the distance to the walls on its left and right. Between each observation the robot moved approximately 51cm (see [13] for details on robot localization and sensor models). Note that the loop in the environment is symmetric except for a few "landmarks" along the walls of the corridor. Localization performance was measured by the average distance between the samples and the reference robot positions, which were computed offline.

In the experiments, our real-time algorithm, RTPF, is compared to particle filters with skipping observations, called "Skip data" (Figure 1a), and particle filters with insufficient samples, called "Naive" (Figure 1c). Furthermore, to gauge the efficiency of our mixture weighting, we also obtained results for our real-time algorithm without weighting, and we denote this variant "Uniform".

The experiment is set up as follows. First, we fix the sample set size n which is sufficient for the robot to localize itself. In our experiment n is set empirically to 20,000 (the particle filters may fail at lower n, see also [12]). We then vary the computational resources, resulting in different window sizes k. Larger window size means lower computational power, and the number of samples generated for each observation (n/k) decreases. The approach "Skip data" will keep generating samples until it reaches n, skipping the $k-1$ observations in between, while "Naive" integrates all observations but only maintains a sample set of size n/k. In contrast, RTPF considers all observations and spreads the n samples over the observations within an estimation window.

Figure 4 shows the evolutions of average localization errors over time, using different window sizes. Each graph is obtained by averaging over 30 runs with different random seeds and start

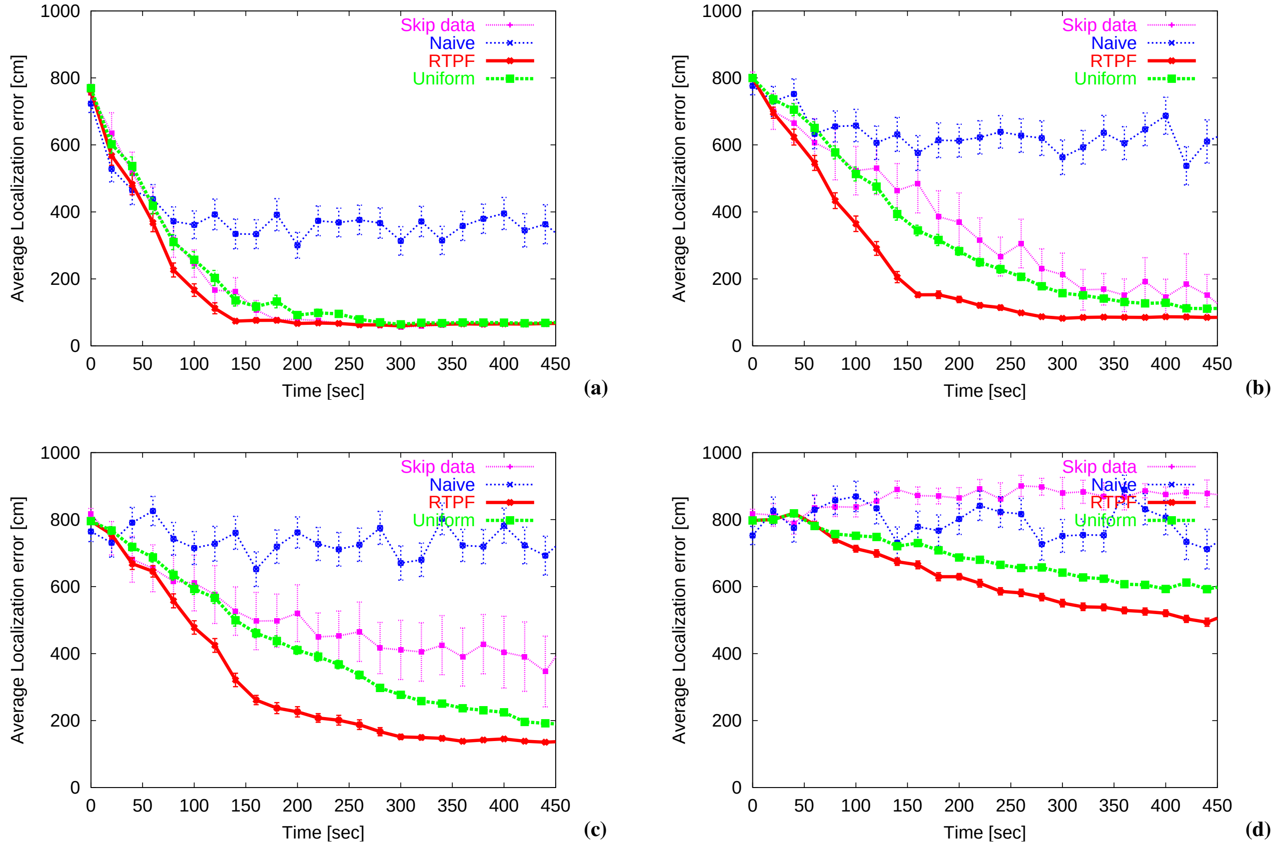
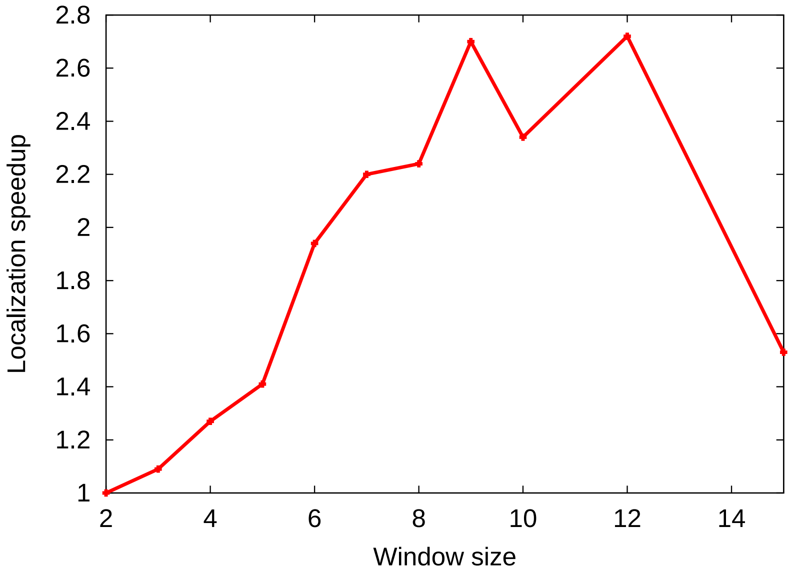


Fig. 4(a)-(d): Performance of the different algorithms for window sizes of 4, 8, 12, and 20, respectively. The x-axis represents time elapsed since the beginning of the localization experiment. The y-axis plots the localization error measured in average distance from the reference position. Each point is averaged over 30 runs, and error bars indicate 95% confidence intervals.

positions. The error bars indicate 95% confidence intervals. As the figures show, "Naive" gives the worst results, which is due to insufficient numbers of samples, resulting in divergence of the filter. While "Uniform" performs slightly better than "Skip data", RTPF is the most effective of all algorithms, localizing the robot in the least amount of time. Furthermore, RTPF shows the least degradation with limited computational power (larger window sizes). The key advantage of RTPF over "Uniform" lies in the mixture weighting, which allows our approach to focus computational resources on valuable sensor information, for example when the robot passes an informative feature in one of the hallways. For short window sizes (Fig. 4(a)), this advantage is not very strong since in this environment, most features can be detected in several consecutive sensor measurements.

Without mixture weighting of RTPF, we did not expect "Uniform" to outperform "Skip data" significantly. To see this, consider one estimation window of length k. Suppose one of the k observations detects a landmark, or very informative feature in the hallway. Assigning uniform weights to each of the k sample sets implies that n/k samples of the "Uniform" algorithm register the landmark. However, if "Skip data" is lucky and hits the landmark, all n samples register, resulting in very good localization performance. Since such a lucky event only occurs every k-th time the robot passes such a landmark, "Skip data" misses the landmark most of the time. When averaged over many different runs, the mean performance of "Uniform" and "Skip data" therefore is very similar. Note that since "Uniform" does not skip any sensor data, it detects all landmarks



(b)

Fig. 5: Localization speedup of RTPF over "Skip data" for various window sizes. The advantage of RTPF increases with the difficulty of the task, *i.e.* with increasing window size. Between window size 6 and 12, RTPF localizes at least twice as fast as "Skip data".

in all runs, resulting in significantly lower variance of the localization errors. In contrast to both approaches, RTPF detects all landmarks *and* generates more samples for the landmark detections, thereby gaining the best of both worlds, and Figures 4(a)–(d) show this is indeed the case.

In Figure 5 we compare the performance gain of RTPF over "Skip data" for different window sizes in terms of localization time. We considered the robot to be localized if the average localization error remains below 200 cm over a period of 10 seconds. If the run never reaches this level, the localization time is set to the length of the entire run, which is 574 seconds. The x-axis represents the window size and the y-axis the localization speedup. For each window size speedups were determined using t-tests on the localization times for the 30 pairs of data runs. All results are significant at the 95% level. Figure 5 shows that with increasing window size, the localization speedup also increases. At small window sizes the speedup is 20-50%, but it goes up to 2.7 times for larger windows, demonstrating the benefits of the RTPF approach over traditional particle filters. Ultimately, for very large window sizes, the speedup decreases again, which is due to the fact that none of the approaches is able to reduce the error below 200cm within the run time of an experiment (cf. Figure 4(d)).

5 Conclusions

In this paper we tackled the problem of particle filtering under the constraint of limited computing resources. Our approach makes near-optimal use of sensor information by dividing sample sets between all available observations and then representing the state as a mixture of sample sets. Next we optimize the mixing weights in order to be as close to the true posterior distribution as possible. Optimization is performed efficiently by gradient descent using a Monte Carlo approximation of the gradients.

We showed that RTPF produces significant performance improvements in a robot localization task. The results indicate that our approach yields significantly better results than all alternative methods for dealing with limited computation. Furthermore, RTPF localized the robot more than 2.7 times faster than the original particle filter approach, which skips sensor data. Based on these results, we expect our method to be highly valuable in a wide range of real-time applications of particle filters. RTPF yields maximal performance gain for data streams containing highly valuable sensor data occuring at unpredictable time points.

Currently, RTPF ignores the influence of the weighting scheme on the effective sample size. Imagine, for example, the extreme situation in which RTPF assigns zero weight to all but one sample set. In this case, only $1/k$-th of the samples are actually used in the next estimation window, which might result in an insufficient number of samples. In our experiments we did not find any evidence that RTPF requires more samples than the original (skip data) approach. This might be due to the fact that RTPF focuses the belief faster, thereby reducing the number of samples needed to approximate the belief. A thorough investigation of these connections is part of our future work.

So far RTPF uses fixed sample sizes and fixed window sizes. The next natural step is to adapt these two "structural parameters" to further speed up the computation. For example, by the method of [12] we can change the sample size on-the-fly, which in turn allows us to change the window size. Ongoing experiments suggest that this combination yields further performance improvements: When the state uncertainty is high, many samples are used and these samples are spread out over multiple observations. On the other hand, when the uncertainty is low, the number of samples is very small and RTPF becomes identical to the vanilla particle filter with one update (sample set) per observation.

Note also that our approach has a broader scope than particle filtering, applicable with minor modifications to Bayesian estimators in general. In this context, similarity with the method described in [20] becomes noticeable. Their work also uses mixtures to compute a tractable approximation to a belief state that is propagated through time. The approach optimizes 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 [20] and our mixture representation is not restricted to one observation interval, but rather spans an entire estimation interval.

References

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

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

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

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

[5] P. Jensfelt, O. Wijk, D. Austin, and M. Andersson. Feature based condensation for mobile robot localization. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, 2000.

[6] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 2002.

[7] D. Schulz, W. Burgard, D. Fox, and A. B Cremers. 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)*, 2001.

[8] M. Montemerlo, S. Thrun, and W. Whittaker. Conditional particle filters for simultaneous mobile robot localization and people-tracking. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, 2002.

[9] V. Verma, J. Langford, and R. Simmons. Non-parametric fault identification for space rovers. In *International Symposium on Artificial Intelligence and Robotics in Space (iSAIRAS)*, 2001.

[10] N. de Freitas. Rao-Blackwellised particle filtering for fault diagnosis. *IEEE Aerospace,*, 2002.

[11] A. Doucet, N. de Freitas, and N. Gordon, editors. *Sequential Monte Carlo in Practice*. Springer-Verlag, New York, 2001.

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

[13] D. Fox, S. Thrun, F. Dellaert, and W. Burgard. Particle filters for mobile robot localization. In Doucet et al. [11].

[14] P. Del Moral and L. Miclo. Branching and interacting particle systems approximations of feynam-kac formulae with applications to non linear filtering. In *Seminaire de Probabilites XXXIV*, number 1729 in Lecture Notes in Mathematics. Springer-Verlag, 2000.

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

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

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

[18] P. R. Cohen. *Empirical methods for artificial intelligence*. MIT Press, 1995.

[19] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for on-line non-linear / non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, XX, 2001.

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