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

***Abstract*— Particle filters estimate the state of dynamic 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 representing posteriors as mixtures of sample sets, where each mixture component integrates one observation arriving during a filter update. The weights of the mixture components are set so as to minimize the approximation error introduced by the mixture representation. Thereby, our approach focuses computational resources on valuable sensor information. Experiments using data collected with a mobile robot show that our approach yields strong improvements over other approaches.**

I. Introduction

Over the last years, particle filters have been applied with great success to a variety of state estimation problems including visual tracking [31], speech recognition [58], mobile robot localization [21], [41], [33], map building [45], people tracking [52], [46], and fault detection [57], [12]. A recent book provides an excellent overview of the state of the art [16]. The key idea of this technique is to represent probability densities by sets of samples. Due to its sample-based representation, particle filters combine efficiency with the ability to represent a wide range of probability densities, allowing online, real-time estimation of nonlinear, non-Gaussian dynamic systems [13]. 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 question 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?* 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 losing 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 [16], [20].

In this article 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 resulting approach naturally focuses computational resources (samples) on valuable sensor information. Furthermore, we enhance real-time particle filters by *adapting* the size of the mixture using KLD-sampling [19], [20], a technique that determines the number of samples based on statistical bounds on the sample-based approximation quality. We empirically demonstrate that this approach is superior to alternative algorithms using particle filters for robot localization.

The remainder of this article 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 III, we introduce our novel technique to real-time particle filters. In Section IV, we review KLD-sampling and present adaptive real-time particle filters. Finally, we demonstrate the effectiveness of our approach using the task of mobile robot localization, followed by conclusions.

II. Bayesian Filtering and Particle filters

In this section we review the basics of Bayes filters and alternative representations for the probability densities, or beliefs, underlying Bayes filters. Then we introduce particle filters as a sample-based implementation of Bayes filters, followed by a discussion of their application to mobile robot localization (major parts of Section II-A through Section II-C are based on [20]). Finally, we discuss particle filters under real-time constraints.

A. Bayes filters

Bayes filters address the problem of estimating the state x of a dynamical system from sensor measurements. The key idea of Bayesian filtering is to recursively estimate the posterior probability density over the state space conditioned on the data collected so far. Without loss of generality, we assume that the data consists of an alternating sequence of time indexed observations z_t and control measurements u_t, which describe the dynamics of the system. 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. Under this assumption the posterior can be determined 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

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

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

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

Here, α is a normalizing constant which ensures that the belief over the entire state space sums up to one. The term $p(x_t \mid x_{t-1}, u_{t-1})$ describes the *system dynamics*, *i.e.* how the state of the system changes given control information u_{t-1}. $p(z_t \mid x_t)$, the *perceptual model*, describes the likelihood of making observation z_t given that the current state is x_t. Note that the random variables x, z and u can be high-dimensional vectors. The belief immediately after the prediction and before the observation is called the *predictive* belief $Bel^-(x_t)$, as given in (1). At the beginning, the belief $Bel(x_0)$ is initialized with the prior knowledge about the state of the system.

Bayes filters are an abstract concept in that they only provide a probabilistic framework for recursive state estimation. To implement Bayes filters, one has to specify the perceptual model $p(z_t|x_t)$, the dynamics $p(x_t \mid x_{t-1}, u_{t-1})$, and the representation of the belief $Bel(x_t)$. The properties of the different implementations of Bayes filters strongly differ in the way they represent densities over the state x_t. We will now briefly discuss different representations used in the context of mobile robot localization.

- Kalman filters are the most widely used variant of Bayes filters [24], [55], [61]. They approximate beliefs by their first and second moments, *i.e.* mean and covariance, and are optimal for linear systems. Non-linearities are typically approximated by linearization at the current state, resulting in the extended Kalman filter (EKF) and more recently, the unscented Kalman filter (UF) [34], [60]. These approximations, though non-optimal, have been shown to be highly effective even for non-linear systems [42], [44], [1], [29]. However, the unimodal posterior assumption limits its application in some estimation problems such as global robot localization [28].
- Topological approaches [35], [53], [39], [30], [8], [40] are based on symbolic, graph structured representations of the environment. The advantage of these approaches lies in their efficiency and in the fact that they can represent arbitrary distributions over the discrete state space. However, the coarseness of the representation provides only rough estimates, and requires the presence of many distinct features and the associated symbolic information in the environment.
- Grid-based, metric approaches rely on discrete, piecewise constant representations of the belief [7], [6], [38], [49]. In contrast to topological approaches, the metric approximations provide accurate estimates in combination with high robustness to sensor noise [5], [56], [22]. However, their computational complexity grows exponentially with the number of dimensions, rendering them impractical for most high-dimensional state spaces.
- Sample-based approaches represent beliefs by sets of samples, or particles [21], [14], [23], [41], [33], [27]. The key advantage of particle filters is their ability to represent arbitrary probability densities. These filters have been shown to converge to the true posterior even in non-Gaussian, non-linear dynamic systems [13]. Compared to grid-based approaches, particle filters are very efficient since they focus their resources (particles) on regions in state space with high likelihood. Since the efficiency of particle filters strongly depends on the number of samples used for the filtering process, several attempts have been made to make more efficient use of the available samples [25], [50], [26]. Since the worst-case complexity of these methods grows exponentially in the dimensions of the state space, it is not clear how particle filters can be applied to arbitrary, high-dimensional estimation problems. If, however, the posterior focuses on small, lower-dimensional regions of the state space, particle filters can focus the samples on such regions, making them applicable to even high-dimensional state spaces. We will discuss the details of particle filters in the next section.
- Mixed approaches use factorizations of the state space and apply different representations for the individual parts of the state space model. For example, Rao-Blackwellised particle filters sample only the discrete and non-linear parts of a state estimation problem. Conditioning on these particles allows one to analytically solve the remaining parts of the problem using Kalman filters, one for each particle [15], [12], [27], [48], [45], [52]. The well-known multi-hypothesis tracking technique [4], [11] can be seen as a special case of such Rao-Blackwellised particle filters.

B. Particle filters

Particle filters are a sample-based variant of Bayes filters, which represent the belief $Bel(x_t)$ by a set S_t of N_p weighted samples distributed according to $Bel(x_t)$:

$$S_t = \{\langle x_t^{(i)}, w_t^{(i)} \rangle \mid i = 1, \ldots, N_p\}$$

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, see also [43], [17], [16]). A time update of the basic particle filter algorithm is outlined in Table I.

1. **Inputs:**
   $S_{t-1} = \{\langle x_{t-1}^{(i)}, w_{t-1}^{(i)} \rangle \mid i = 1, \ldots, N_p\}$ representing belief $Bel(x_{t-1})$,
   control measurement u_{t-1},
   observation y_t
2. $S_t := \emptyset$, $\alpha := 0$ — *// Initialize*
3. **for** $i := 1, \ldots, N_p$ **do** — *// Generate N_p samples*

   // Resampling: Draw state from previous belief
4. Sample an index j from the discrete distribution given by the weights in S_{t-1}

   // Sampling: Predict next state
5. Sample $x_t^{(i)}$ from $p(x_t \mid x_{t-1}, u_{t-1})$ conditioned on $x_{t-1}^{(j)}$ and u_{t-1}
6. $w_t^{(i)} := p(y_t \mid x_t^{(i)})$; — *// Compute importance weight*
7. $\alpha := \alpha + w_t^{(i)}$ — *// Update normalization factor*
8. $S_t := S_t \cup \{\langle x_t^{(i)}, w_t^{(i)} \rangle\}$ — *// Insert sample into sample set*
9. **end do**
10. **for** $i := 1, \ldots, N_p$ **do** — *// Normalize importance weights*
11. $w_t^{(i)} := w_t^{(i)} / \alpha$
12. **end do**
13. **return** S_t

Table I: The basic particle filter algorithm.

At each iteration, the algorithm receives a sample set S_{t-1} representing the previous belief of the robot, a control measurement u_{t-1}, and an observation y_t. Steps 3–9 generate N_p samples representing the posterior belief. Step 4 determines which sample to draw from the previous set. In this resampling step, a sample index is drawn with probability proportional to the sample weight [1]. Once a sample index is drawn, the corresponding sample and the control information u_{t-1} are used to predict the next state $x_t^{(i)}$. This is done by sampling from the density $p(x_t \mid x_{t-1}, u_{t-1})$, which represents the system dynamics. Each $x_t^{(i)}$ corresponds to a sample drawn from the predictive belief $Bel^-(x_t)$ in (1). In order to generate samples according to the posterior belief $Bel(x_t)$, importance sampling is applied, with $Bel(x_t)$ as target distribution and $Bel^-(x_t)$ as proposal distribution [54], [23]. By dividing these two distributions, we get $p(y_t \mid x_t^{(i)})$ as the importance weight for the i-th sample. Step 7 keeps track of the normalization factor, and Step 8 inserts the new sample into the sample set. After generating N_p samples, the weights are normalized so that they sum up to one (Steps 10-12). It can be shown that this procedure in fact implements the Bayes filter, using an approximate, sample-based representation [17], [16]. Furthermore, the sample-based posterior converges to the true posterior as the number N_p of samples goes to infinity [13]. We discuss an application of particle filters to the robot localization problem in the following section.

C. Particle filters for mobile robot localization

In typical robot localization problems, the position of the robot is represented in the two-dimensional Cartesian space along with the robot's heading direction. Measurements z_t may include range measurements and camera images, and control information u_t usually consists of the robot's odometry readings. The next state probability $p(x_t \mid x_{t-1}, u_{t-1})$ describes how the position of the robot changes based on information collected by the robot's wheel encoders. This conditional probability is typically a model of robot kinematics annotated with white noise [21], [23]. The perceptual model $p(z_t \mid x_t)$ describes the likelihood of making the observation z_t given that the robot is at location x_t. This model is extracted from a map of the environment along with a model of the sensor noise. Particle filters have been applied to a variety of robot platforms and sensors such as vision [14], [41], [18], [62], [59] and proximity sensors [21], [23], [33].

Figure 1 illustrates the application of particle filters to mobile robot localization using sonar sensors. Shown there is a map of a hallway environment along with a sequence of sample sets during global localization. In global localization, the start location of the robot is completely unknown and the particles are initialized with a uniform distribution. The pictures demonstrate the ability of particle filters to represent a wide variety of distributions, ranging from uniform to highly focused. Especially in symmetric environments, the ability to represent ambiguous situations is of utmost importance for the success of global localization. In this example, all sample sets contain 100,000 samples. While such a high number of samples might be necessary to accurately represent the belief during early stages of localization (cf. 1(a)), it is obvious that only a small fraction of this number suffices to track the position of the robot once it knows where it is (cf. 1(c)). A statistical approach to adapt the number of samples during estimation was introduced in [20], and section IV provides a brief review.

[1]Resampling with minimal variance can be implemented efficiently (constant time per sample) using a procedure known under the name deterministic selection [36], [2] or stochastic universal sampling [3].

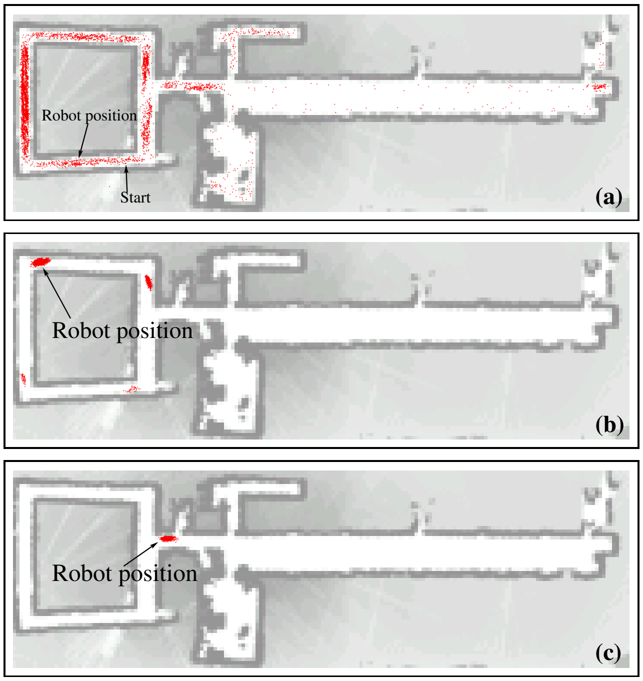


Fig. 1. Map of an indoor environment along with a series of sample sets representing the robot's belief during global localization using sonar sensors (samples are projected into 2D). The size of the environment is 54m × 18m. a) After moving 5m, the robot is still highly uncertain about its position and the samples are spread through major parts of the free-space. b) Even as the robot reaches the upper left corner of the map, its belief is still concentrated around four possible locations. c) Finally, after moving approximately 55m, the ambiguity is resolved and the robot knows where it is.

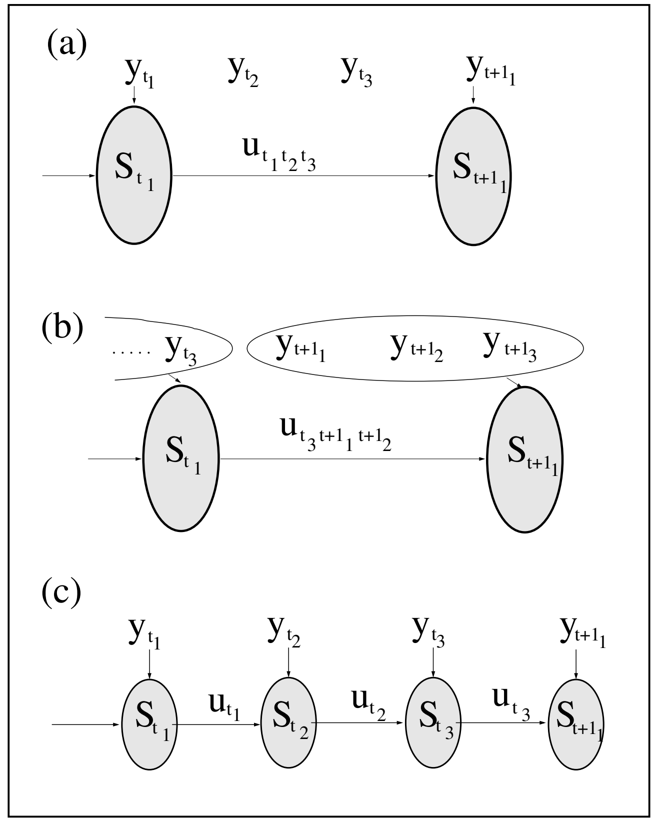


Fig. 2. Different strategies for dealing with limited computational power. All approaches process the same number of samples per estimation interval (window size three). a) Skip observations, *i.e.* integrate only every third observation. b) Aggregate observations and integrate them in one step. c) Reduce sample set size so that every observation can be integrated.

Robot localization has to be performed online, *i.e.* during the operation of a mobile robot. Hence, the consideration of real-time constraints and limited computational resources is extremely important for the successful application of particle filters to the robot localization problem.

D. Real-time considerations

A typical assumption underlying particle filters is that all samples can be updated whenever new sensor information arrives. Under real-time 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 [20]. Before we discuss ways of dealing with such situations, let us introduce some notation. We assume that observations arrive at time intervals Δ and that the particle filter requires N_p samples to represent the belief appropriately. Assume that the resulting update cycle of the particle filter takes $k\Delta$, which we call the *estimation interval* or *estimation window*. Accordingly, k observations arrive during one estimation interval. We denote this number the *window size* of the filter, *i.e.* the number of observations obtained during a filter update. The i-th observation and state within window t are denoted y_{t_i} and x_{t_i}, respectively.

Fig. 2 illustrates different approaches to dealing with window sizes larger than one. The simplest and most common approach is shown in Fig. 2(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. 2(b) overcomes this problem by aggregating multiple observations into one. While this technique avoids the loss of information, it is not applicable to arbitrary dynamic 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. 2(c), simply stops generating new samples whenever an observation is made (hence each sample set contains only N_p/k samples). While this approach takes advantage of the any-time capabilities of particle filters, it is susceptible to filter divergence due to an insufficient number of samples [20], [16].

III. Real time particle filters

We will now present 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 sample sets within a window, our approach focuses the computational resources (samples) on the most valuable observations. Fig. 3 illustrates the approach. As can be seen, instead of one sample set at time t, we maintain k smaller

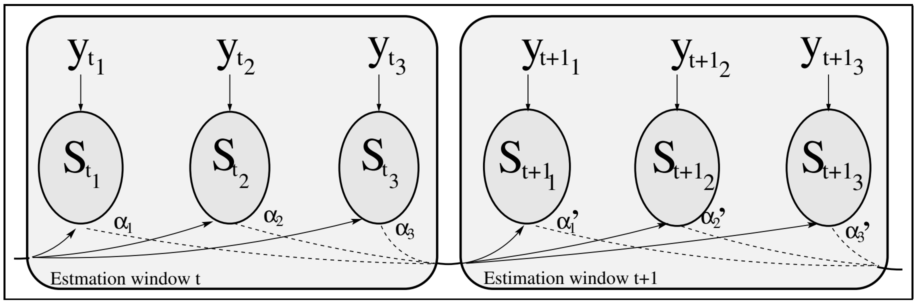


Fig. 3. Real time particle filters. The N_p samples are distributed among the observations within one estimation interval (window size three). The resulting belief is a mixture of the individual sample sets. For each window the weights α_i of the mixture components are chosen so that the approximation error introduced by the mixture is minimal.

sample sets at $t_1, \ldots t_k$. We treat such a "virtual sample set", or belief, as a mixture of the distributions represented in it. The mixture components represent the state of the system at different points in time. If needed, however, the complete belief can be generated by considering the control information 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. 2(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. 2(c)) is more subtle. In both approaches, each of the k sample sets can only contain N_p/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 $k \cdot \frac{N_p}{k}$ sample trajectories, that for computational convenience are represented at the points in time where the observations are integrated. In the approach shown in Fig. 2(c), however, the belief propagation is simulated with only N_p/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.

A. 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 on each observation:

$$Bel_{opt}(x_k) \propto \int \ldots \int \prod_{i=1}^{k} p(y_{t_i} \mid x_{t_i})\; p(x_{t_i} \mid x_{t_i-1}, u_{t_i-1}) \cdot Bel(x_{t_0}) dx_{t_0} \ldots dx_{t_{k-1}}. \tag{3}$$

Here $Bel(x_{t_0})$ denotes the belief generated in the previous estimation window. In essence, (3) 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 $y_{t_1}, \ldots, y_{t_k}$ 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. $Bel_i(x_{t_k})$ is given by the following rule:

$$Bel_i(x_{t_k}) \propto \int \ldots \int p(y_{t_i}|x_{t_i}) \cdot \prod_{j=1}^{k} p(x_{t_j}|x_{t_{j-1}}, u_{t_{j-1}}) Bel(x_{t_0}) dx_{t_0} ... dx_{t_{k-1}} \tag{4}$$

Note that here the product only contains observation-free trajectories since the only observation y_{t_i} moved outside the product. 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:

$$Bel_{mix}(x_{t_k} \mid \alpha) \quad \propto \quad \sum_{i=1}^{k} \alpha_i \; Bel_i(x_{t_k}) \tag{5}$$

where $\alpha_i \geq 0$ and $\sum_i \alpha_i = 1$. Here, too, we integrate over all trajectories. In contrast to (3), however, each trajectory selectively integrates only one of the k observations within the estimation interval[2].

B. 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 [10] between Bel_{mix} and Bel_{opt}.

$$\begin{aligned} \alpha^* &= \underset{\alpha \in \mathcal{A}}{\operatorname{argmin}} \; KL(Bel_{mix}(\cdot \mid \alpha) \mid\mid Bel_{opt}) \\ &= \underset{\alpha \in \mathcal{A}}{\operatorname{argmin}} \int Bel_{mix}(x_{t_k} \mid \alpha) \; \log \; \frac{Bel_{mix}(x_{t_k} \mid \alpha)}{Bel_{opt}(x_{t_k})} \; dx_{t_k}. \end{aligned} \tag{6}$$

In the above $\mathcal{A} = \{\alpha \mid \sum_{i=1}^{k} \alpha_i = 1, \; \alpha_i \geq 0\}$. Here we assume that Bel_{opt} is known. In practice this is not true and the next section shows how to estimate it. Optimizing the weights of mixture approximations can be done using EM [51]

[2]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.

or (constrained) gradient descent [32]. In our approach, we perform a small number of gradient descent steps to find the mixture weights. Denote by $J(\alpha)$ the criterion to be minimized in (6). The gradient of $J(\alpha)$ is given by

$$\begin{aligned}\frac{\partial J}{\partial \alpha_i} &= \int \frac{\partial}{\partial \alpha_i} Bel_{mix}(x_{t_k}|\alpha) \log Bel_{mix}(x_{t_k}|\alpha) - \\ &\qquad \frac{\partial}{\partial \alpha_i} Bel_{mix}(x_{t_k}|\alpha) \log Bel_{opt}(x_{t_k}) \\ &= 1 + \int Bel_i(x_{t_k}) \log Bel_{mix}(x_{t_k}|\alpha) - \\ &\qquad Bel_i(x_{t_k}) \log Bel_{opt}(x_{t_k}) dx_{t_k} \\ &= 1 + \int Bel_i(x_{t_k}) \ \log \ \frac{Bel_{mix}(x_{t_k}|\alpha)}{Bel_{opt}(x_{t_k})} \ dx_{t_k}, \\ &\qquad i = 1, \dots k. \end{aligned} \tag{7}$$

This derivation is based on the fact that the gradient of the mixture belief with respect to the i-th weight is given by the i-th mixture component. The start point α_s for the gradient descent is chosen to be the center of the weight domain $\mathcal{A}$, that is $\alpha_s = [\frac{1}{k} \ \frac{1}{k} \ \dots \ \frac{1}{k}]^T$.

The belief Bel_{opt} is the one that integrates all k observations in the same way as the basic particle filter. This is precisely what we want to avoid computing in the real time situation that is the object of this paper. Our idea is to obtain an estimate of Bel_{opt} from a number of samples $N' << N_p$, that will allow us to perform the gradient optimization but will not be computationally prohibitive.

C. Monte Carlo gradient estimation

This section shows how we obtain the gradients in (7) by Monte Carlo approximation. The approach is based on the observation that the beliefs in (7) share the same trajectories through space and differ only in the observations they integrate. Therefore, we first generate sample trajectories through the estimation window *without* considering the observations, and then re-weigh them according to the different beliefs needed for the gradient estimation.

Trajectory generation is done as follows: we draw a sample x_{t-1} from a sample set of the previous mixture belief, where the probability of choosing a set S_{t-1_j} is given by the mixture weight α_j. This sample is then 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, $i = 1, \dots k$. The resulting trajectories are drawn from the following proposal distribution f:

$$\begin{aligned} f(x_{t_k}) &= \int \!\! \dots \!\! \int \prod_{i=1}^{k} p(x_{t_i} \mid x_{t_{i-1}}, u_{t_{i-1}}) \cdot \\ &\qquad Bel(x_{t_0}) \, dx_{t_0} \dots dx_{t_{k-1}} \end{aligned} \tag{8}$$

In our implementation, the initial set of trajectory samples (which corresponds to the start of the trajectories) are drawn from the first sample set in the current estimation window [3]. The resulting set of trajectories contains trajectories $\langle x^{(m)}_{t_1:t_k}, w^{(m)}_{t_1:t_k} \rangle$. Next, we obtain sample-based estimates of Bel_i and Bel_{opt} by weighting each trajectory with $p(y_{t_i} \mid x_{t_i})$ or $\prod_{j=1}^{k} p(y_{t_j} \mid x_{t_j})$, respectively. These weights are the importance weights resulting from dividing (3) and (5) by (8), respectively. 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 [9].

[3]These samples are drawn uniformly, *i.e.* we ignore the weights that are based on the likelihood of the first observation (see Table IV).

D. Bias and clustering

We have implemented the above method and compared it with RTPF where the mixture weights α were uniform. Unfortunately, the weight optimization gave less benefits than we expected. Upon investigation we found that the problem was caused by a *bias* in our estimation, and its effect is particularly pronounced for robot localization tasks.

The key problem is that the sample trajectories within an estimation window tend to diverge over time. This divergence is caused by increased uncertainty due to applying larger and larger controls within an estimation window. An example is shown in Figure 4. A robot is placed in the environment shown and it moves along the corridor. Due to the similarity between the two hallways, the robot's location is not yet uniquely determined and the samples are spread out over the two clusters shown in the Figure. Suppose the distribution Bel_{t_1} in Figure 4(a) is the first set in an estimation window, and Figure 4(b)–(c) show the beliefs Bel_{t_2} and Bel_{t_3}, respectively. Obviously, the further the robot moves, the higher the uncertainty becomes as the distribution becomes more spread out. This is because at each predictive update, a sample is propagated forward in space by drawing it from the previous sample set using the distribution $p(x_{t_i} \mid x_{t_{i-1}}, u_{t_{i-1}})$. In this example, imagine the robot can only detect the distance to the walls on its left and right side. Due to the similarity of the hallways, all observations received in this window are virtually identical. Ideally, RTPF would give all sample sets the same weights, since they all contribute the same information if there were no real-time constraints. However, because the distributions are different, the gradients w.r.t. the different α_i's are also different. Empirical data suggest that most weight is usually given to the last sample set Bel_{t_k}. The reason for this is illustrated in Figure 4(d)–(f). These sample sets correspond to the resampled distributions after integrating the observations into the sample sets shown in Figure 4(a)–(c), respectively. As shown, the distribution in (c) was more spread out than (a), but the corresponding resampled distribution in (f) does not differ as much from (d). Hence it appears that the observation in (c) is more effective in reducing the KL-divergence than the observation in (a). For this reason, the bias tends to be stronger for larger controls. Such a bias toward sample sets that occur later within an estimation windows is clearly undesirable.

To overcome this problem, we settle for optimizing the weights only in order to *disambiguate* between groups of very

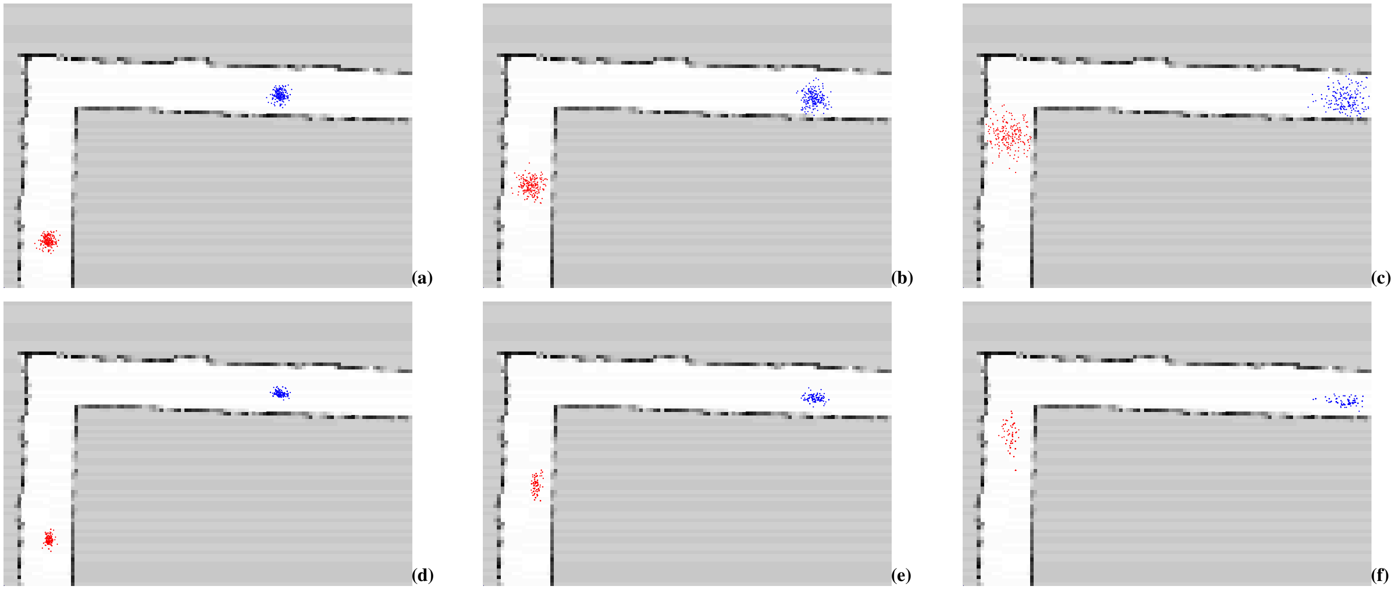


Figure 4. Bias at different time steps in an estimation window. (a)–(c) Distributions without considering observations. (a) Initial belief Bel_{t_1}. (b) Uncertainty increases as the robot moves. (c) At time t_3, Bel_{t_3} is spread out even more. (d)–(f) The resampled distributions after integration of an observation. The distribution for Bel_{t_3} changed more than Bel_{t_1} through resampling, even though the observations for both are the same.

distinct robot positions. This is done by determining if the particles representing the hypothetical robot states are grouped into disjoint regions (clusters). If multiple clusters are detected, then we optimize the mixture weights according to (7), where the beliefs Bel_{mix} and Bel_{opt} are now aggregated probability distributions over the different clusters. If no clusters are detected, the sample sets in the current estimation window receive equal weights.

Particle trajectories are clustered by determining connected regions in a grid over the state space at time t_1. Neighboring grid cells are considered connected if both contain samples. If more than one cluster is found, the belief associated with a cluster C_q is computed by summing up the weights of the trajectories in C_q. Suppose we have N_T trajectories, let C be a partition over the set of numbers $\{1, \ldots, N_T\}$. Then $C_q \in C$ is a set of indices labeling the trajectories. The clustered beliefs are thus

$$Bel(C_q) = \sum_{m \in C_q} Bel(x_{t_k}^{(m)}) \tag{9}$$

These beliefs form probability distributions over the clusters. We apply formula (7) to these distributions in order to compute the gradients of the mixture weights (with summation replacing the integral in (7)).

In our experiments we found that maximizing the mixture weights using clustered trajectories eliminates the bias over different sample sets within estimation windows. The result of this approach becomes clear in the situation shown in Figure 4. Suppose C_1 and C_2 are the two clusters in this example, corresponding to the two groups of samples. At t_1 (Figure 4(a)), $Bel_{t_1}(C_1) = 0.5$ and $Bel_{t_1}(C_2) = 0.5$. At t_2 and t_3, even as the distribution spreads out, $Bel_{t_2}(C_i)$ and $Bel_{t_3}(C_i)$ are still the same as $Bel_{t_1}(C_i)$. This is due to the fact that the observations only provide information that reduces uncertainty within clusters, but do not help to disambiguate between the different clusters. As a consequence, since our algorithm maximizes the mixture weights using clustered trajectories, all three sample sets will obtain the same weights.

An additional benefit of clustering is the reduction in the number of trajectories needed to get smooth gradient estimates. 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. For gradient descent, since our gradients are coarse approximations, we do not try to converge to the true minimum. Rather, we descend a small number of steps and use the value we stop at. The number of steps is determined empirically to be 5 in the experiments, but the results were not very sensitive to this parameter.

E. *Real-time particle filter algorithm*

To summarize, the number N_p 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 used by our RTPF algorithm. Given the 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, RTPF keeps track of the control information (dynamics) between the different sample sets of two consecutive windows. The RTPF algorithm is summarized in Table II.

The algorithm has several key differences to the original particle filter given in Table I. Instead of drawing samples from a single, previous sample set, RTPF generates samples from the mixture of sample sets from the previous estimation

1. **Inputs:**
   - k — *// Number of sample sets in an estimation window*
   - l — *// Position within current estimation window*
   - N_y — *// Number of samples that can be generated per observation*
   - $S_{t-1_1}, \ldots, S_{t-1_k}$ — *// 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*
   - y_{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 a total of N_y samples from previous window*
5. $\quad n_i = \alpha_{t-1_i} N_y$ *// Number of samples to be drawn from i-th set*
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 weights in S_{t-1_i}

   $\qquad$ *// Sampling: Predict next state*
8. $\qquad$ Sample $x_t^{(m)}$ from $p(x_t \mid x_{t-1_i}, u_{t-1_i}, \ldots, u_{t_{l-1}})$ using $x_i^{(j)}$ and $u_{t-1_i}, \ldots, u_{t-1_k}, u_{t_1}, \ldots, u_{t_{l-1}}$
9. $\qquad w_t^{(m)} := p(y_t \mid x_t^{(m)});$ *// Compute importance weight*
10. $\qquad \nu := \nu + w_t^{(m)}$ *// Update normalization factor*
11. $\qquad S_t := S_t \cup \{\langle x_t^{(m)}, w_t^{(m)} \rangle\}$ *// Insert sample into sample set*
12. $\quad$ **end do**
13. **end do**
14. **for** $m := 1, \ldots, N_y$ **do** *// Normalize importance weights*
15. $\quad w_t^{(m)} := w_t^{(m)} / \nu$
16. **end do**
17. **if** $l = 0$ **then** *// Compute the size of this window*
18. $\quad n_\chi :=$ REQUIRED-SAMPLES(S_t) *// Compute the required number of samples*
19. $\quad k := \lceil \frac{n_\chi}{N_y} \rceil$ *// Update the new window size*
20. **end if**
21. $l := l + 1$ *// Increment time index within this window*
22. $T :=$ EXTEND-TRAJECTORIES($T, S_t, u_{t_{l-1}}$) *// Update trajectories*
23. **if** $l = k$ **then** *// Advance window?*
24. $\quad \alpha_{t_1}, \ldots, \alpha_{t_k} :=$ OPT-MIX(T, k) *// Monte-Carlo gradients*
25. $\quad l := 0$ *// Set index to beginning of next window*
26. **end if**
27. **return** S_t

Table II. The RTPF algorithm. Window size k is a parameter determined by the desired number n_χ 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 3). The function REQUIRED-SAMPLES determines the number of samples needed to represent the belief. EXTEND-TRAJECTORIES extends the path of trajectories using the current control measurement. OPT-MIX determines the mixture weights at the end of an estimation window.

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 and 14–16 generate samples and normalize their weights virtually identical to the plain SISR update of particle filters. After generating the samples for the first sample set of an estimation window, the size of the estimation window is determined in steps 17–20. This size depends on the number of samples required to represent the belief. The approach used in function REQUIRED-SAMPLES will be described in Section IV. Step 22 adds the most recent time step to the set T, which represents the trajectories of particles through the current estimation window. These trajectories are used in steps 23–26 to determine the mixture weights of the sample sets once the last set in the window has been generated.

The function OPT-MIX is described in Table III. In step 2, the trajectories are clustered by determining the connected components of a grid laid over the state space at the first time step in the window. Steps 3–8 compute the statistics needed in the gradient steps. After normalizing these statistics, the mixture weights are initialized equally in steps 10–12. Steps 14–19 compute the gradients w.r.t. the individual mixture weights followed by a small step along these gradients (step 18). These gradient steps are performed N_I times (loop in step 13).

In the next section we will show how to adapt the number of particles and the size of the estimation windows during the estimation process. The key component of this section will be the function REQUIRED-SAMPLES used in step 18 of the RTPF algorithm.

Procedure OPT-MIX

1. **Inputs:**
   T, // *Set of trajectory samples*
   k, // *Size of window*
   N_I // *Number of gradient iterations (free parameter)*
2. $C :=$ CLUSTER-SAMPLES(T) // *C is a partition of the set $\{1, \ldots, N_T\}$*
3. **foreach** $C_j \in C$ **do** // *Compute the Bel_i's and Bel_{opt} using the clusters*
4.   **for** $i := 1, \ldots, k$ **do**
5. $Bel_i(C_j) := \sum_{m \in C_j} w_{t_i}^{(m)}$ // *Bel_i's are weighted by a single observation*
6. $Bel_{opt}(C_j) := \sum_{m \in C_j} \prod_{i=t_1}^{t_k} w_{t_i}^{(m)}$ // *Bel_{opt} is weighted by all the observations*
7.   **end do**
8. **end do**
9. Renormalize Bel_i's and Bel_{opt}
10. **for** $i := 1, \ldots, k$ **do** // *The α_i's are initialized to be uniform*
11. $\alpha_i := 1/|k|$
12. **end do**
13. **for** $s := 1, \ldots, N_I$ **do** // *Perform N_I gradient descents*
14.   **for** $i := 1, \ldots, k$ **do** // *Compute gradients for each α_i*
15. $\frac{\partial J}{\partial \alpha_i} := 1 + \sum_{C_j \in C} Bel_i(C_j) \log \frac{\sum_{m=1}^{k} \alpha_m Bel_m(C_j)}{Bel_{opt}(C_j)}$
16.   **end do**
17.   **for** $i := 1, \ldots, k$ **do** // *Change the α_i's based on the gradients*
18. $\alpha_i :=$ GRADIENT-DESCENT($\alpha_i, \frac{\partial J}{\partial \alpha_i}$)
19.   **end do**
20. **end do**
21. **return** $\alpha_1, \ldots, \alpha_k$

Table III. The OPT-MIX function for computing mixture weights. The start samples of each trajectory are used to form clusters for trajectories, using the procedure described in Section III-D. Then, Monte-Carlo gradients are performed over the clusters and a small number of gradient descents are performed to approximate the mixture weights. Note that Bel_{mix} is the numerator of line 15.

IV. ADAPTIVE REAL-TIME PARTICLE FILTERS

Since the complexity of particle filters depends on the number of samples used for estimation, several attempts have been made to make more efficient use of the available samples [25], [50], [26]. More recently, Fox [20] showed that adapting the size of sample sets during the estimation process can greatly increase the efficiency of particle filters. To see, consider the application of particle filters to the problem of mobile robot localization, as illustrated in Figure 1. Shown there is a sequence of sample sets during global localization. While a high number of samples might be necessary to accurately represent the belief shown in Fig. 1(a), it is obvious that only a small fraction of this number suffices to track the position of the robot once it knows where it is (cf. 1(c)). Clearly, the efficiency of particle filters can be greatly increased by adapting the number of samples during the localization process. Moreover, in the context of real-time particle filters, reducing sample size allows us to shrink estimation windows so that they become shorter. If the number of samples is high, these samples have to be spread out over multiple observations. On the other hand, when the uncertainty is low, the number of samples required is very small and RTPF becomes identical to the vanilla particle filter with one update (sample set) per observation. Besides higher convergence rate due to less samples, smaller window size also has the advantage of reducing the divergence due to large controls (Section III-C).

The number of required samples can be determined using results from previous work, *KLD-sampling* [20]. We first provide a brief description of this algorithm, followed by its integration into RTPF.

A. KLD-sampling

At each iteration of the particle filter, KLD-sampling determines the number of samples such that, with probability $1-\delta$, the error between the current posterior estimate and the sample-based approximation is less than ε.

To see, suppose that n samples are drawn from a discrete distribution with b different bins. Let the vector $\underline{X} = (X_1, \ldots, X_b)$ denote the number of samples drawn from each bin. $\underline{X}$ is distributed according to a multinomial distribution, *i.e.* $\underline{X} \sim \text{Multinomial}_b(n, \underline{p})$, where $\underline{p} = p_1 \ldots p_b$ specifies the probability of each bin. After observing the counts $\underline{X}$, the maximum likelihood estimate of $\underline{p}$ is given by $\widehat{\underline{p}} = n^{-1}\underline{X}$.

If we draw N_p samples from a distribution $\underline{p}$ with b bins, where

$$N_p \doteq \frac{b-1}{2\epsilon}\left\{1 - \frac{2}{9(b-1)} + \sqrt{\frac{2}{9(b-1)}}y_{1-\delta}\right\}^3, \quad (10)$$

then it can be shown that the KL-distance between the sample based maximum likelihood estimate and the true distribution $\underline{p}$ is less than ϵ (with probability $1-\delta$) [20]. In (10), $y_{1-\delta}$ is the upper $1-\delta$ quantile of the standard normal $\mathcal{N}(0,1)$

distribution. From (10) we see that the required number of samples is proportional to the inverse of the ε bound, and to the first order linear in the number of bins b.

This result can be applied to particle filters as follows. The error bounds ϵ and δ are fixed before the estimation process. Given ϵ and δ, it suffices to estimate the number of bins b in order to determine the number of necessary samples N_p at each iteration of the particle filter (see (10)). To determine b, KLD-sampling converts sample sets into multinomials using a discrete, piece-wise constant approximation such as a discrete density tree or a multi-dimensional histogram [37], [47]. The number of supported bins b is then estimated by the number of cells that contain at least one sample (KLD-sampling treats the particle filter belief as if it was the true belief). In the KLD-sampling algorithm, b is estimated *during sampling*. To be more specific, b is estimated for the proposal distribution $p(x_t \mid x_{t-1}, u_{t-1})Bel(x_{t-1})$ resulting from the first two steps of the particle filter update. The determination of b can be done efficiently by checking for each generated sample whether it falls into an empty bin or not. Sampling is stopped as soon as the number of generated samples exceeds the threshold specified through (10).

To summarize, KLD-sampling adapts the number of samples based on the approximation error introduced by the sample-based representation. It uses the predictive belief state as an estimate of the underlying posterior. Therefore, it is not guaranteed that this approach does not diverge from the true (unknown) belief. In practice, however, divergence only occurs when the error bounds are too loose.

B. KLD-sampling and real-time particle filters

In this section we show how to integrate KLD-sampling into RTPF, thereby enabling the approach to change the window size over time. As mentioned in Section III, RTPF represents the posterior by a mixture of sample sets. Given fixed computational resources and a fixed rate of incoming sensor information, the window size solely depends on the number N_p of samples. Therefore, reducing the number of samples during the localization process also allows us to reduce the window size, which results in better estimation performance.

Unfortunately, it is not immediately obvious how one can apply KLD-sampling to RTPF. Recall that KLD-sampling determines the required number of samples by counting the non-empty cells of a grid representing the underlying posterior belief. Let b denote the number of non-empty cells. In the case of RTPF, the posterior is a *mixture* of samples sets, with each set representing the belief at a different point in time. However, the union of these sets cannot be modeled by a single grid since the robot moves between the individual sample sets. One approach to determine b might be to synchronize the sample sets by "shifting" them to the same point in time. Technically this could be done by applying the appropriate control to the samples in the different sets. b could then be computed at the synchronization time, just like in KLD-sampling. However, such an approach is prohibitively expensive in real-time settings.

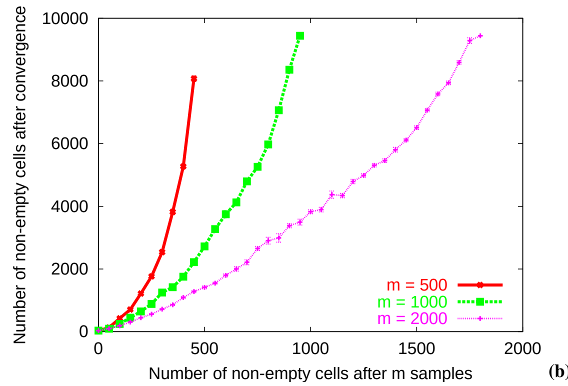


Fig. 5: The function G for different m. The $x-$axis represents $\tilde{b}$, the number of bins filled after m samples have been generated. The $y-$axis plots b, the number of cells filled after convergence. Error bars indicate 95% confidence intervals for the b values, based on the training data. On many data points the intervals are too small to be visible.

Our solution to this problem is based on the observation that it is possible to predict the total number of filled bins b after only a rather small number of samples have been generated. More specifically, we estimate b by a function $G(m, \tilde{b})$, where m is the number of samples generated so far and $\tilde{b}$ is the number of bins filled by these samples. In RTPF, we can use this function G after the generation of the *first* set in the estimation window. The number of bins $\tilde{b}$ filled by the m samples in this set is then used to predict the total number of bins b that would be filled if there was enough time to generate samples until convergence to b (often, this is not possible since the next observation arrived and the second set is generated).

The function G can be learned beforehand from data processed in a non real-time setting. In the case of robot localization, this data is obtained by performing multiple global localization runs using the adaptive particle filter on data sets collected in the environment. As samples are generated, we record tuples of the form $\langle m, \tilde{b}, b\rangle$, where m is the number of samples generated so far, $\tilde{b}$ is the number of cells filled by these m samples, and b is the final number of bins filled after the adaptive particle filter converged. The function $G(m, \tilde{b})$ can thus be defined by these tuples. Figure 5 shows the function learned for the environment described in the next section for some values of m. The three graphs are $G(500, \tilde{b}), G(1000, \tilde{b})$ and $G(2000, \tilde{b})$, respectively. The small confidence intervals show that, for example, 500 samples suffice to accurately predict b. Since G is learned from estimation runs in a given environment, one might wonder how specific this function is, or whether the function is applicable in a broader context. Therefore we learned G for two other indoor environments. The results suggest that G is rather insensitive to the changes in the environment and remains very similar to the one shown in Figure 5.

Armed with G, it is possible to estimate the number of required samples for adaptive real-time particle filters. As we generate samples for the first sample set within a window, we update the number of non-empty cells $\tilde{b}$ based on the number m of samples generated until the next observation arrives. $G(m, \tilde{b})$ gives us b, the number of non-empty cells

Procedure REQUIRED-SAMPLES

Inputs:
    S_t, — *// Current sample set*
    ϵ, δ — *// Adaptive sampling error bounds*
foreach $x_t \in S_t$ **do** — *// Put generated samples into bins*
    if $(x_t$ falls into empty bin $r)$ **then**
        $\tilde{b} := \tilde{b} + 1$ — *// Update number of bins with support*
        $r :=$ non-empty — *// Mark bin*
    end if
end do
$b = G(m, \tilde{b})$ — *// Adjust the full bin count using G*
$n_\chi := \frac{b-1}{2\varepsilon}\left\{1 - \frac{2}{9(b-1)} + \sqrt{\frac{2}{9(b-1)}}y_{1-\delta}\right\}^3$ — *// Update number of desired samples*
return n_χ

Table IV. Brief description of the REQUIRED-SAMPLES function for the adaptive RTPF algorithm. G is the learned mapping for estimating the number of occupied cells when only a fraction of the samples are available.

after convergence. Using b, Equation (10) gives us the number N_p of samples required to accurately represent the current belief. Whenever a new observation arrives, we generate a new sample set within the estimation window until the total number of samples reaches N_p. Hence, small numbers of samples result in short estimation windows. Once enough independent samples are generated, the estimation window is finished, the approach determines the weights for the different sample sets within the window, and starts a new estimation window. The computational overhead of this procedure is negligible.

For the adaptive RTPF algorithm, the function REQUIRED-SAMPLES (step 18 in Table II) is implemented using KLD-sampling combined with the function G, as illustrated in Table IV (see [20] for a more detailed description of the KLD-sampling part). The function takes as input the samples generated by the first set in the estimation window. It first counts the number of bins $\tilde{b}$ filled by the m samples in this set. The function G is used to predict the number of bins which is then used in the KLD-sampling equation.

As noted above, this function is called within the RTPF algorithm described in Table II. The number of samples returned by REQUIRED-SAMPLES is divided by the number of samples that can be generated per observation to determine the window size of RTPF.

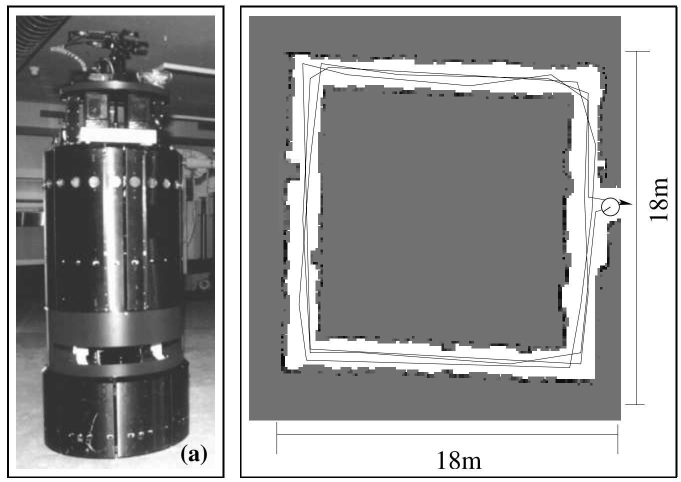


Fig. 6. Fig. 6: (a) The robot used in our experiment, an RWI B21. (b) Map of the environment used for the experiment. The robot was moved around the symmetric loop. The task of the robot was to determine its position using data collected by two distance measuring devices, one pointing to its left, the other pointing to its right.

V. EXPERIMENTS

In this section we evaluate the effectiveness of our real-time algorithms against the alternatives, using data collected with an RWI B21 mobile robot (Figure 6(a)) in a real-world environment. The data consists of a sequence of laser range-finder scans along with odometry measurements annotated with time-stamps to allow systematic real-time evaluations. We used a beam-based sensor model to compute the likelihood of the sensor scans [22], [23]. Figure 6(b) shows the setup of the experiment: The robot was placed in the office floor and moved around the loop. 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 (see [23] 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.

A. Real-time particle filters (RTPF)

In this experiment, our real-time algorithm, RTPF, is compared to particle filters with skipping observations, called "Skip data" (Figure 2a), and particle filters with insufficient samples, called "Naive" (Figure 2c). Furthermore, to gauge the efficiency of our mixture weighting, we also obtained results for our real-time algorithm without weighting, *i.e.* we used mixture distributions and fixed the weights to $1/k$. We denote this variant "Uniform". Finally, we also include as reference the Baseline approach, which is allowed to generate

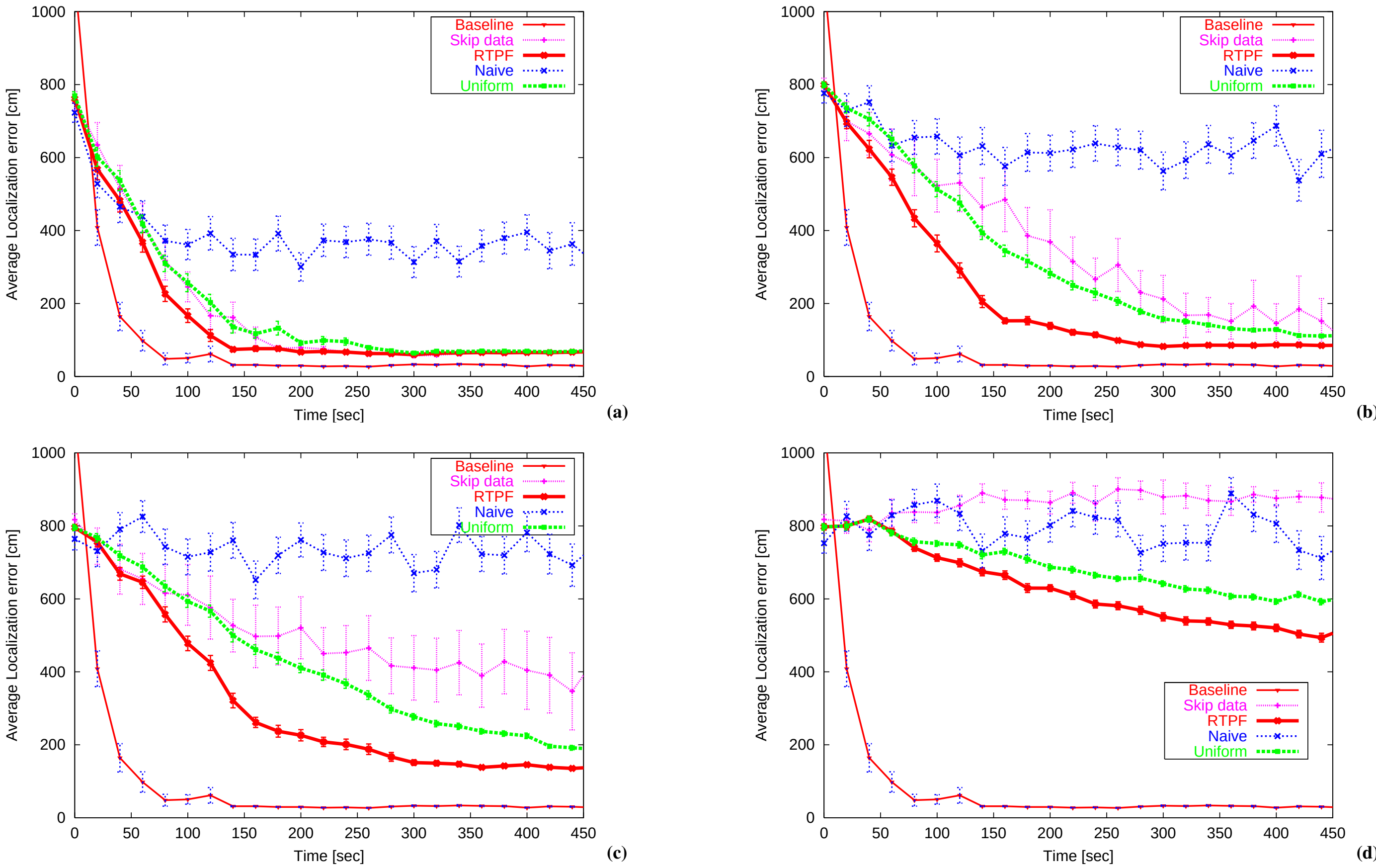


Fig. 7(a)-(d): Performance of the different algorithms for (a) 25%, (b) 12.5%, (c) 8% and (d) 5% processing powers, respectively. The x-axis represents time elapsed since the beginning of the localization experiment. The y-axis plots the localization error measured in average distance from the reference position. Each figure includes the performance achieved with unlimited computational power as the Baseline graph. Each point is averaged over 30 runs, and error bars indicate 95% confidence intervals.

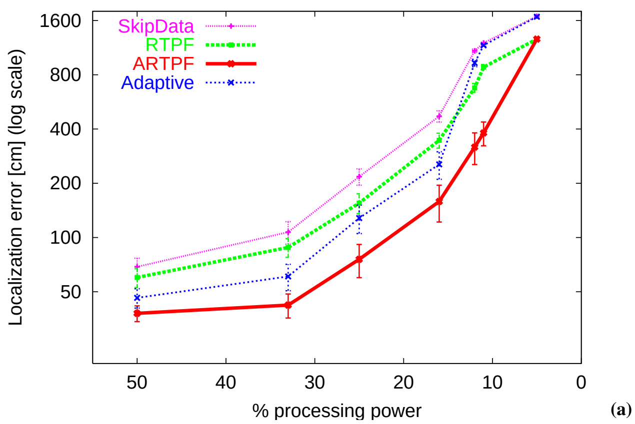
N_p samples for each observation, thereby not considering real-time constraints. [4] In this set of experiments, the robot moved approximately 50cm between each observation.

The comparison is performed as follows. First, the sample set size is set empirically to $N_p = 20{,}000$ which is sufficient for the robot to globally localize itself in this environment. We then vary the *processing power*, where each approach is given the same amount of processing power. For example, a processing power of 100% refers to the ability to update all 20,000 samples between two observations, and a processing power of 25% refers to the ability to update only a quarter of the required samples. For RTPF, lower processing power means larger estimation windows. For “Skip data”, on the other hand, 25% processing power results in skipping 3 observations during each integration.

Figure 7 shows the evolutions of average localization errors over time, using different processing powers. Each graph is obtained by averaging over 30 runs with different random seeds and start 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 more processing power (Fig. 7(a)), this advantage is not very strong since in this environment, most features can be detected in several consecutive sensor measurements. Note that because the Baseline approach was allowed to integrate all observations with all of the 20,000 samples, it converges to a lower error level than all the other approaches.

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 only one of the k observations detects a landmark, or very informative feature in the hallway. In such a situation, “Uniform” considers this landmark every time the robot passes it. However, it only assigns N_p/k samples to this landmark detection. “Skip data” on the other hand, detects the landmark only every k-th time, but assigns *all* N_p samples to it. Therefore, averaged over many different runs, the mean performances of “Uniform” and “Skip data” are similar. However, the variance of the error

[4] Sensor aggregation as shown in Fig. 2(b) is not applicable to our sensor model.



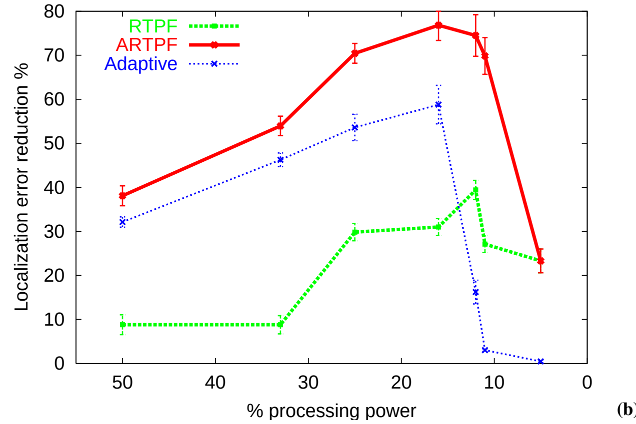


Fig. 9: (a) Average localization errors for different processing powers in log scale. Processing powers range from 50% to 5%. (b) Error reduction over "Skip data" for the same range of processing powers.

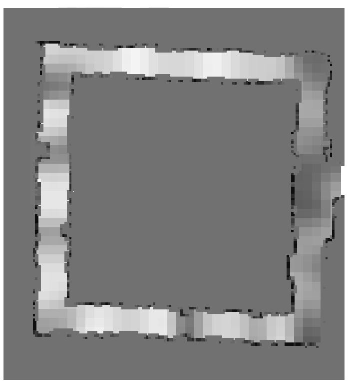


Fig. 8. Map of the environment along with average weights assigned to observations during one run. Darker regions represent higher weights.

is significantly lower for "Uniform" since it considers the detection in every run. 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 7(a)–(d) show this is indeed the case.

To shed light on the advantage of weighting the components of the mixture beliefs, we analyzed one experimental run. Figure 8 shows the average weights given to sample sets at different robot locations (higher weights are darker). It shows that higher weights are given to sample sets that integrate distinctive features in the environment. For instance, the protrusions along the corridors and the corners are assigned more weights than the corridors themselves. This demonstrates that our Monte Carlo gradient method is capable of determining the relative importance of observations and thereby focusing the processing power (samples) on informative observations.

B. *Adaptive real-time particle filters (ARTPF)*

We also evaluated the effectiveness of adaptive real-time particle filters, ARTPF, with a similar experimental methodology as in the previous section. In these experiments, ARTPF is compared to "Skip data", to adaptive particle filters with skipping observations (labeled Adaptive, [20]), and RTPF with fixed number of samples. The insufficient performance of "Naive" has already been demonstrated in the previous section. Essentially, Adaptive corresponds to "Skip data" with varying number of skipped observations, and ARTPF is the same as RTPF with varying window sizes. In this set of experiments, the robot moved approximately 70cm between each observation.

Figure 9 summarizes the performance of the approaches for different processing powers. In Fig. 9(a), the y-axis is the average localization error in log scale. ARTPF significantly outperforms the other approaches at all levels of processing power. At 5%, the adaptive approaches have no advantage over their fixed counterparts any more. This is due to the fact that the uncertainty never drops enough to allow significant decrease in sample set sizes. Fig. 9(b) provides another interpretation of the performance. It plots the *error reduction* achieved by each approach, compared to the vanilla particle filter ("Skip data"). Error reduction is measured by averaging the relative differences in localization errors. For example, at 15% processing power, ARTPF reduces the error by 77%, which means on average its localization error is less than one quarter of "Skip data". The results show that ARTPF is the best among the algorithms, followed by Adaptive in the 15-50% processing power range. Below this level, however, RTPF outperforms Adaptive due to its better ability to handle insufficient computational resources. Adaptive finally degenerates into "Skip data" since it skips too many observations at the beginning of localization and thus cannot reduce the number of samples during the localization run.

Figure 10 provides a more detailed evaluation for four levels of processing power. It shows the evolutions of average localization errors over time using processing powers of 25% down to 8%. Figure 10(a) and (b) show that at 25% and 20% processing power, ARTPF already localizes the robot significantly faster than the other methods. Furthermore, the two adaptive approaches ARTPF and Adaptive converge to the same error level as the Baseline approach, which is given 100% processing power. This is due to the fact that, as soon as the robot is localized, the number of samples drops enough to allow both approaches to integrate all observations, *i.e.* window size one for ARTPF. "Skip data" and RTPF, on the

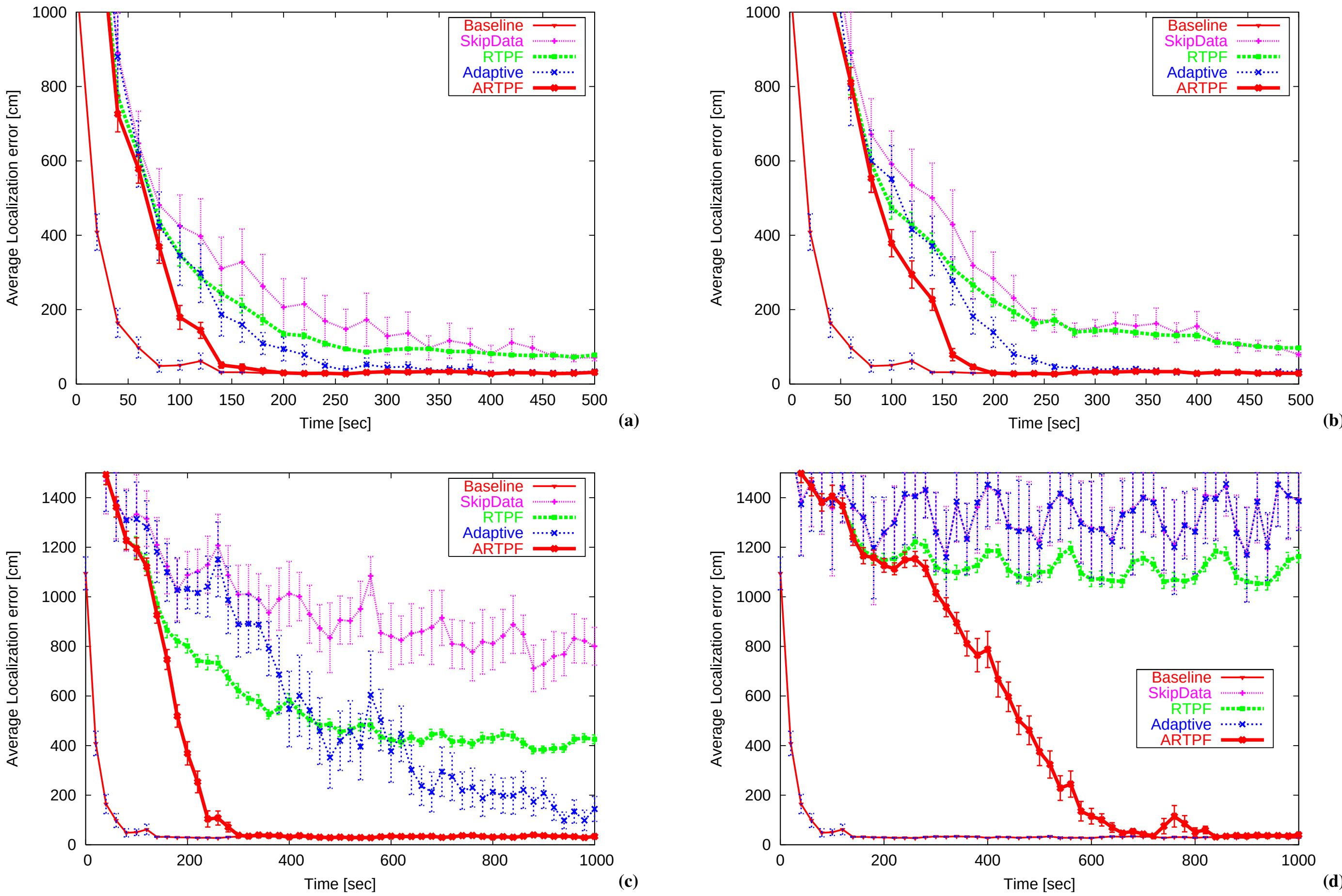


Fig. 10(a)-(d): Performance of the different algorithms for (a) 25%, (b) 20%, (c) 12.5% and (d) 8% processing powers. The x-axis represents time elapsed since the beginning of the localization experiment. The y-axis plots the localization error measured in average distance from the reference position. Each point is averaged over 20 runs, and error bars indicate 95% confidence intervals. All figures include the performance achieved with 100% processing power as the Baseline graph. Note that the scales in (a) and (b) are different from those in (c) and (d).

other hand, cannot reduce the number of samples, thereby skipping 3 (“Skip data”) or mixing 4 (RTPF) observations even when the robot is localized.

At processing power 12.5% (Figure 10 (c)), the advantage of ARTPF becomes more prominent. “Skip data” has to skip 7 observations and thus fails to localize the robot. RTPF is clearly superior to “Skip data”, but still fails to reach an acceptable error level. Adaptive skips too many observations in the beginning and thus converges very slowly. After 450 seconds of localization, however, Adaptive outperforms RTPF since Adaptive can decrease the sample set size, thereby achieving higher update rates. ARTPF clearly outperforms all other approaches by combining the advantages of real-time particle filters with adaptive sample sets. In the beginning, when the high uncertainty requires large sample sets, the error drops at the same rate as RTPF. After 150 seconds, however, the window size starts to decrease, thereby giving ARTPF a clear advantage over the other approaches. This advantage becomes even more prominent with processing power of 8% (Figure 10 (d)).

VI. Conclusions

In this article we addressed the problem of particle filtering under the constraint of limited computing resources. A key problem in this context arises if the rate of incoming sensor data is higher than the update rate of the particle filter. We introduce real-time particle filters, which make efficient use of sensor information by dividing sample sets between all available observations and then representing the state as a mixture of sample sets. The number of mixture components in each estimation window is given by the number of observations arriving during the time it takes to process enough samples so as to avoid divergence of the particle filter. The weights of the mixture components are optimized in order to be as close to the optimal posterior distribution as possible. Optimization is performed efficiently by gradient descent using a Monte Carlo approximation of the gradients. As a result of this weighting scheme, real-time particle filters automatically focus computing resources (samples) on the most informative observations.

We additionally presented adaptive real-time particle filters, an algorithm that combines the advantages of adaptive sample set sizes with real-time particle filters. By combining these techniques, the resulting approach can dynamically change the number of samples and hence the number of mixture components within an estimation window.

Empirical evaluations show that our techniques produce significant performance improvements in a robot localization task under computational limitations. The results indicate that real-time particle filters outperform all alternative particle filter methods for dealing with limited computation. Adaptive real-time particle filters yield even more drastic performance gains. In some experiments, this technique reduced the average error of the original particle filter by up to 77% (the original filter skips observations arriving during the update of a sample set).

A key advantage of our technique is that it makes no assumptions about the properties of sensor data or the process to be estimated. Hence, we expect our method to be highly valuable in a wide range of real-time applications of particle filters. Real-time particle filters yield maximal performance gain for data streams containing highly valuable sensor data occurring at unpredictable time points.

The application of particle filters under real-time constraints raises several questions for future research. Even real-time particle filters can diverge if there is not enough processing power to keep track of the dynamic system. In such situations, one could consider to control the dynamic system in order to avoid filter divergence. In mobile robotics, for example, the travel speed of the robot could be adapted to the certainty of the filtering process. The robot should move slowly if it is highly uncertain about its location, since the update rate of the particle filter is rather low. If it is tracking its location, however, it can move at high speeds while still being able to achieve good localization performance. Another important research question is how to adapt the representation of the Bayes filter over time. Instead of using particle filters or Kalman filters during the complete estimation process, one could switch between different representations, depending on the uncertainty and the available computing resources.

VII. Acknowledgments

This research is sponsored in part by the National Science Foundation (CAREER grant number 0093406).

References

[1] K. Arras and S. Vestli, "Hybrid, high-precision localization for the mail distributing mobile robot system MOPS," in *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, 1998.

[2] 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*, vol. XX, 2001.

[3] J. Baker, "Reducing bias and inefficiency in the selection algorithm," in *Proc. of the Second International Conference on Genetic Algorithms.*, 1987.

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

[5] W. Burgard, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun, "Experiences with an interactive museum tour-guide robot," *Artificial Intelligence*, vol. 114, no. 1-2, pp. 3–55, 1999.

[6] W. Burgard, A. Derr, D. Fox, and A. B. Cremers, "Integrating global position estimation and position tracking for mobile robots: the Dynamic Markov Localization approach," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1998.

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

[8] H. Choset and K. Nagatani, "Topological simultaneous localization and mapping (SLAM): toward exact localization without explicit localization," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 2, 2001.

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

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

[11] I. J. Cox, "A review of statistical data association techniques for motion correspondence," *International Journal of Computer Vision*, vol. 10, no. 1, pp. 53–66, 1993.

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

[13] 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*, ser. Lecture Notes in Mathematics. Springer-Verlag, 2000, no. 1729.

[14] F. Dellaert, W. Burgard, D. Fox, and S. Thrun, "Using the condensation algorithm for robust, vision-based mobile robot localization," in *Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 1999.

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

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

[17] A. Doucet, S. Godsill, and C. Andrieu, "On sequential Monte Carlo sampling methods for Bayesian filtering," *Statistics and Computing*, vol. 10, no. 3, 2000.

[18] S. Enderle, M. Ritter, D. Fox, S. Sablatng, G. Kraetzschmar, and G. Palm, "Soccer-robot locatization using sporadic visual features," in *Proc. of the International Conference on Intelligent Autonomous Systems (IAS)*, 2000.

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

[20] ——, "Adapting the sample size in particle filters through KLD-sampling," *International Journal of Robotics Research (IJRR)*, vol. 22, no. 12, 2003.

[21] 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.

[22] D. Fox, W. Burgard, and S. Thrun, "Markov localization for mobile robots in dynamic environments," *Journal of Artificial Intelligence Research (JAIR)*, vol. 11, pp. 391–427, 1999.

[23] D. Fox, S. Thrun, F. Dellaert, and W. Burgard, "Particle filters for mobile robot localization," in *Sequential Monte Carlo in Practice*, A. Doucet, N. de Freitas, and N. Gordon, Eds. New York: Springer-Verlag, 2001.

[24] A. Gelb, *Applied Optimal Estimation*. MIT Press, 1974.

[25] W. Gilks and C. Berzuini, "Following a moving target - Monte Carlo inference for dynamic Bayesian models," *Journal of the Royal Statistical Society, Series B*, vol. 61, no. 1, 2001.

[26] S. Godsill and T. Clapp, "Improvement strategies for Monte Carlo particle filters," in *Sequential Monte Carlo in Practice*, A. Doucet, N. de Freitas, and N. Gordon, Eds. New York: Springer-Verlag, 2001.

[27] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, and P.-J. Nordlund, "Particle filters for positioning, navigation and tracking," *IEEE Transactions on Signal Processing*, 2002.

[28] J. Gutmann, W. Burgard, D. Fox, and K. Konolige, "An experimental comparison of localization methods," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1998.

[29] J. Gutmann, T. Weigel, and B. Nebel, "Fast, accurate, and robust self-localization in polygonal environments," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1999.

[30] J. Hertzberg and F. Kirchner, "Landmark-based autonomous navigation in sewerage pipes," in *Proc. of the First Euromicro Workshop on Advanced Mobile Robots*. IEEE Computer Society Press, 1996.

[31] M. Isard and A. Blake, "Condensation – conditional density propagation for visual tracking," *International Journal of Computer Vision*, vol. 29, no. 1, pp. 5–28, 1998.

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

[33] 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.

[34] S. Julier and J. Uhlmann, "A new extension of the kalman filter to nonlinear systems," in *Proc. of AeroSense: The 11th International Symposium on Aerospace/Defense Sensing, Simulation and Controls*, 1997.

[35] L. P. Kaelbling, A. R. Cassandra, and J. A. Kurien, "Acting under uncertainty: Discrete Bayesian models for mobile-robot navigation," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1996.

[36] G. Kitagawa, "Monte Carlo filter and smoother for non-Gaussian nonlinear state space models," *Journal of Computational and Graphical Statistics*, vol. 5, no. 1, 1996.

[37] D. Koller and R. Fratkina, "Using learning for approximation in stochastic processes," in *Proc. of the International Conference on Machine Learning (ICML)*, 1998.

[38] K. Konolige and K. Chou, "Markov localization using correlation," in *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1999.

[39] B. Kuipers, "The spatial semantic hierarchy," *Artificial Intelligence*, vol. 119, pp. 191–233, 2000.

[40] B. Kuipers and P. Beeson, "Bootstrap learning for place recognition," in *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 2002.

[41] 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.

[42] J. J. Leonard and H. F. Durrant-Whyte, "Mobile robot localization by tracking geometric beacons," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 376–382, 1991.

[43] J. Liu and R. Chen, "Sequential Monte Carlo methods for dynamic systems," *Journal of the American Statistical Association*, vol. 93, no. 443, 1998.

[44] F. Lu and E. Milios, "Robot pose estimation in unknown environments by matching 2D range scans," *Journal of Intelligent and Robotic Systems*, vol. 18, 1997.

[45] 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.

[46] 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.

[47] A. W. Moore, J. Schneider, and K. Deng, "Efficient locally weighted polynomial regression predictions," in *Proc. of the International Conference on Machine Learning (ICML)*, 1997.

[48] K. Murphy and S. Russell, "Rao-Blackkwellised particle filtering for dynamic Bayesian networks," in *Sequential Monte Carlo in Practice*, A. Doucet, N. de Freitas, and N. Gordon, Eds. New York: Springer-Verlag, 2001.

[49] C. Olson, "Probabilistic self-localization for mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 16, no. 1, 2000.

[50] M. K. Pitt and N. Shephard, "Filtering via simulation: auxiliary particle filters," *Journal of the American Statistical Association*, vol. 94, no. 446, 1999.

[51] 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.

[52] D. Schulz, D. Fox, and J. Hightower, "People tracking with anonymous and id-sensors using Rao-Blackwellised particle filters," in *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[53] R. Simmons and S. Koenig, "Probabilistic robot navigation in partially observable environments," in *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1995.

[54] A. F. M. Smith and A. E. Gelfand, "Bayesian statistics without tears: A sampling-resampling perspective," *American Statistician*, vol. 46, no. 2, pp. 84–88, 1992.

[55] R. Smith, M. Self, and P. Cheeseman, "Estimating uncertain spatial relationships in robotics," in *Autonomous Robot Vehicles*, I. Cox and G. Wilfong, Eds. Springer Verlag, 1990.

[56] S. Thrun, M. Beetz, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Haehnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz, "Probabilistic algorithms and the interactive museum tour-guide robot minerva." *International Journal of Robotics Research (IJRR)*, vol. 19, no. 11, 2000.

[57] 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.

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

[59] N. Vlassis, B. Terwijn, and B. Kröse, "Auxiliary particle filter robot localization from high-dimensional sensor observations." in *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, 2002.

[60] E. Wan and R. van der Merwe, "The unscented kalman filter for nonlinear estimation," in *Proc. of Symposium 2000 on Adaptive Systems for Signal Processing, Communications, and Control*, 2000.

[61] G. Welch and G. Bishop, "An introduction to the kalman filter," Notes of ACM SIGGRAPH tutorial on the Kalman filter, 2001.

[62] J. Wolf, W. Burgard, and H. Burkhardt, "Robust vision-based localization for mobile robots using an image retrieval system based on invariant features," in *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, 2002.