Recent Developments in Distributed Particle Filtering: Towards Fast and Accurate Algorithms

Andrea Simonetto, * Tamás Keviczky *

* Delft Center for Systems and Control, Delft University of Technology, The Netherlands
e-mail: { a.simonetto, t.keviczky } @ tudelft.nl.

Abstract: Particle filters have been widely used for the solution of optimal estimation problems in nonlinear non-Gaussian environments. One of their drawbacks is that these methods are computationally expensive. In the past few years, new developments have been made in trying to distribute the particle filter algorithm among different computing agents in order to make the underlying computations tractable. This period also witnessed the rise of general purpose GPU devices, which are making massive code parallelization possible. These developments have the potential to make the particle filter a viable alternative for real-time implementations in the near future, even when the number of required particles is high.

Keywords: Particle filters, distributed estimation algorithms, distributed particle filters, consensus filters, distributed computation, mobile robot tracking, general purpose GPU.

1. INTRODUCTION

After more than fifteen years since their first appearance (Gordon et al., 1993), Particle Filters still represent an active area of research. Due to their generality and simplicity, they have become a topic of constantly growing interest, development and numerous applications. Particle Filters, together with Unscented Kalman Filters (Julier and Uhlmann, 2004) and Moving Horizon Estimators (Rao et al., 2003), can provide a general framework for state estimation in nonlinear and non-Gaussian dynamical systems. Moreover, provided that the number of particles is high enough, they can easily outperform other estimation methods. Their main drawback is the required computational load. The necessary computational time for an accurate result is, in most cases, prohibitive for real-time applications and can seriously limit the applicability of Particle Filters.

Recent developments that are aimed at overcoming this difficulty could lead to faster yet accurate algorithms. On one hand, research on distributed estimation investigates how to distribute Particle Filters among different sensors. These studies suggest that there is a natural fit for these methods in the framework of consensus theory, which has received increased attention and developments in the past ten years (Ren and Beard, 2008). On the other hand, the rise of general purpose GPU applications is making massive parallelization of algorithms possible by distributing computing tasks on multiple cores leading to significant improvements in computational time. These trends are reflected in reasonably priced supercomputers hitting the market in these days featuring thousands of GPU cores. The reader is referred to Lozano and Otsuka (2008) and Hendeby et al. (2007) for standard Particle Filter applications on GPUs. These two new directions for improvement could merge in the near future, leading to the implementation of faster and more accurate Particle Filters, which can be a viable alternative for real-time applications, even when the number of particles is very high. It is not far-fetched to imagine thousands of particles running on each one of the thousand cores.

In this paper, the most important steps that need to be taken towards this goal are described. First, a survey on recent developments in Particle Filtering for distributed estimation is presented, focusing on their consistency and representation in the consensus theory framework. Then we shift our attention to methods that distribute the computation among different cores, and propose a simple yet effective algorithm. A simulation example representing a localization problem of unicycle robots via noisy range measurements is used to compare the distributed computation scheme with a standard one. The comparison parameters are chosen to be accuracy, computational time and required communication.

The paper is organized as follows: Section 2 provides a background on the general theory behind Particle Filters leading to the formulation of the Sampling Importance Resampling (SIR) filter algorithm. Section 3 introduces distributed versions of Particle Filters for two separate classes: Particle Filters for multiple and distributed sensors and Particle Filters for multiple and distributed computation cores with access to all the sensor measurement data. This section is a survey on recent developments in the field and provides a general way of representing these algorithms in the framework of consensus protocols. We also propose an algorithm within this framework that allows distributed computation among different computing cores. A numerical example is described in Section 4, while Section 5 discusses open issues for future research.
2. BACKGROUND AND GENERAL PROBLEM FORMULATION

We consider the following nonlinear time-invariant dynamical system

\begin{align*}
x(k) &= f(x(k-1), u(k-1), w(k-1)) \quad (1) \\
z(k) &= h(x(k), v(k)) \quad (2) \\
g(x(k)) &\leq 0 \quad (3)
\end{align*}

\( k = 0, 1, \ldots \)

where \( f \) and \( h \) are nonlinear functions, \( u(k-1) \) and \( x(k-1) \) are the control input and state vector at discrete time \( k-1 \), respectively. The inputs \( w \) and \( v \) are noise terms and \( z(k) \) is the measurement vector. Furthermore, \( g \) is a nonlinear function which imposes a set of constraints on the state. We will pose the estimation problem in the general framework of probabilistic estimators (Thrun et al., 2005), where given a set of measurements \( \mathbf{z}_k := \{ z(1), \ldots, z(k) \} \) and an initial condition \( x(0) \), we estimate the state \( \hat{x}(k) \) via the a posteriori PDF \( p(\mathbf{x}_k|\mathbf{z}_k) \), where \( \mathbf{x}_k := \{ x(0), \ldots, x(k) \} \).

Under the assumptions of linearity, Gaussian noise, Markovian process and absence of constraints, no algorithm can do better for this problem than a Kalman Filter (Arulampalam et al., 2002). However, when the underlying dynamics is highly nonlinear, or when the noise terms cannot be modeled as Gaussian, or when the state is constrained, the Kalman filter and its extensions fail. In order to overcome these drawbacks, typically, Monte Carlo techniques have been exploited with useful results leading to Particle Filters. Since, in most cases, the a posteriori PDF, \( p(\mathbf{x}_k|\mathbf{z}_k) \), cannot be evaluated because of the complexity of the underlying dynamical system, the basic idea of Particle Filters is to draw \( j = 1, \ldots, m \) random samples \( \{ x_k^j \} \) from some proposal distribution \( q(\mathbf{x}_k|\mathbf{z}_k) \) with the same support as \( p(\mathbf{x}_k|\mathbf{z}_k) \). By associating a weight

\[
\omega_k^j = \frac{p(x_k^j|\mathbf{z}_k)}{q(x_k^j|\mathbf{z}_k)}
\]

to the sample \( x_k^j \), we can approximate the quantity of interest by

\[
p(\mathbf{x}_k|\mathbf{z}_k) \approx \frac{1}{W_k} \sum_{j=1}^m \omega_k^j \delta(\mathbf{x}_k - \mathbf{x}_k^j)
\]

with \( W_k = \sum_j \omega_k^j \). Suppose \( \{ \{ \mathbf{x}_{k-1}^j, \omega_{k-1}^j \} \} \) are available at time \( k-1 \). Since \( p(\mathbf{x}_k|\mathbf{z}_k) \) is in most cases a probability density from which it is difficult to draw samples, they can be drawn from a factorized importance density \( q(\mathbf{x}_k|\mathbf{z}_k) \) given by

\[
q(\mathbf{x}_k|\mathbf{z}_k) = q(x(k)|\mathbf{x}_{k-1}, \mathbf{z}_k)q(x_{k-1}|\mathbf{z}_{k-1})
\]

with associated weights

\[
\omega_k^j \propto \frac{p(x_k^j|\mathbf{z}_k)}{q(x_k^j|\mathbf{z}_k)}
\]

\[
\omega_{k-1}^j \propto \frac{p(z(k)|\mathbf{x}_{k-1})p(x(k)|\mathbf{x}_{k-1}^j)}{q(x(k)|\mathbf{z}_k)}
\]

where the Bayes’ rule and the Markovian assumption have been exploited, and where the proportionality stands for a normalization (Guo et al., 2005). Given the recursive relations (6)-(7) and a proper initialization, a generic particle filter algorithm consists of the following steps:

1) Draw \( m \) samples \( x(k)^j \) from some proposal distribution \( q(x(k)|\mathbf{x}_{k-1}, \mathbf{z}_k) \).
2) Discard the particles which do not satisfy the constraints (3), see Lang et al. (2007).
3) Compute the importance weight \( \omega_{k-1}^j \) for each \( j \).
4) Compute the state estimate according to the a posteriori PDF.
5) Resample the set of particles according to their weight if necessary.

The algorithm is initialized by drawing a set of samples \( \{ x(1)^j \} \) from \( p(x_0) \), which corresponds to the prior distribution of \( x_0 \).

The resampling step is a crucial component of particle filter algorithms. Resampling is necessary since it can provide the chance for good particles to amplify themselves and produce better and more accurate results. Moreover, it overcomes the degeneracy phenomenon, where after a few iterations, all but one particle will have negligible weight. However, it introduces also other practical issues that need careful attention. First, it limits the opportunity to parallelize since all the particles must be combined. Second, the particles that have high weights are statistically selected many times. This leads to a loss of diversity among the particles as the resultant samples will contain many repeated points. Therefore the choice of the number of particles and of the resampling procedure will fundamentally determine the properties of the particle filter.

The class of algorithms called Sampling Importance Resampling, or SIR, filters (Arulampalam et al., 2002) is the simplest class of Particle Filters. Since they are easy to program and highly flexible, they are also the most widely used in real applications.

SIR Filters choose the proposal distribution to be the state equation (1), therefore:

\[
q(x(k)|\mathbf{x}_{k-1}^j, \mathbf{z}_k) = p(x(k)|x(k-1)^j)
\]

with associated weights

\[
\omega_k^j = \omega_{k-1}^j p(z(k)|x(k)^j)
\]

In Arulampalam et al. (2002) a summary of this approach can be found. Note that since the resampling is performed in every time step, the weights do not depend on \( \omega_{k-1}^j \), because \( \omega_{k-1}^j = 1/m, \forall j \).

3. DISTRIBUTED PARTICLE FILTERS

Distributed algorithms for estimation have been receiving increased attention from the research community. The reason is twofold: on one hand, a large amount of data that comes from several sensors has to be processed. This could easily turn out to be very time consuming, for instance if a central processor unit has to deal with several cameras sending high bandwidth videos simultaneously. In this respect, the aim would be to have different processor units that cope with the sensors in a distributed way. On the other hand, computing the estimate using Particle Filters even with a few sensors can be too computationally intensive for a single CPU. Therefore the question of how to distribute this computation among different cores can become relevant.
In this context, we will define two main classes of Distributed Particle Filters. We will call Distributed Sensing Particle Filter, or DSPF, the class of algorithms that copes with a potentially large number of different sensors. Thus we will assume that there is a Particle Filter running on each of these sensors and is using local measurements. The situation is depicted in Figure 1.b. Note that block $D$ stands for an information exchange process, where some sort of distributed computation takes place among the sensors, which will be described later. We will refer to the second class of algorithms as Distributed Computation Particle Filters, or DC PF, which have access to all sensor measurements but use only a subset of particles in each computing core. The different cores where the distributed Particle Filters are running are depicted in Figure 1.c along with a distributed computation block, $D$. The number of cores is in general different from the number of sensors.

DSPF are nowadays receiving increased attention within the area of distributed estimation. Since the field is rather new, the terminology is not always coherent. We will refer to methods that use central units to collect data from the sensors and compute the state estimate as Quasi-DSPF algorithms. This class has been studied in the literature starting from the work of Bashi et al. (2003), where three Quasi-DSPF algorithms have been presented, including more recent papers on Hierarchical DSPF (Huang et al., 2008). We will refer to Particle Filters that do not require centralized data collection as the standard class of DSPF.

Although references on DSPF date back to the pioneering works of Rosencrantz et al. (2003) and Coates (2004), detailed analysis and evaluation studies on their properties are very scarce. The main reasons are different. First, Particle Filters cannot easily share quantities based on their measurements: they cannot send particles among themselves because these are weighted with different measurements and thus they are non-compatible; moreover they cannot send raw measurement data since the communication cost would be too high (Coates, 2004). Second, Particle filters cannot easily be implemented in parallel: in general, the resampling step needs the whole particle population knowledge. Furthermore, it is crucial that the distributed algorithms are consistent with the centralized counterparts. For SIR filters this is naturally ensured by the relation

$$p(z(k)|x(k)) = \prod_{i=1}^{N} p(z_i(k)|x(k)),$$

which states that each sensor can agree on the same $p(z(k)|x(k))$ provided they have full information. Although this statement is true, we must make sure that the sensors speak the same language, i.e., either they are sharing raw measurements, or they have exactly the same particle population, or they have the same representation of $p(z(k)|x(k))$ as a function of $x(k)$.

The first situation above is exploited in Rosencrantz et al. (2003), where, in order to decrease the communication cost, a query-answer protocol is used. The basic idea is that each sensor keeps in memory the entire time-evolution of its particles and all the measurements, and it sends some of the particles to the neighbors. The neighbors decide if some measurements, at some time instant, are valuable for them and they reply with the data. This protocol is valuable when the sensors have enough data to run accurate Particle Filters on their own, and they need extra information only in some special cases. A typical application is localization in a building: when a sensor has a clear view of the object to be localized, it can run its own Particle Filter with no extra information. On the contrary, when the object is hidden behind a wall, it needs some data from other sensors which can see the object.

A completely different approach is studied in Coates (2004) where the sensors are supposed to share the very same particle population. An interesting algorithm is developed in the framework of parametric modeling for this situation.
purpose. However, this paper imposes strict limitations on the distributed algorithms, for instance, in some cases the sensors have to be synchronized, and in general, the communication graph among them has to have a specified structure: either a chain, a ring or a tree.

Another way to implement a consistent version of DSPF is to guarantee that all the sensors have the same representation of $p(z(k)|x(k))$. Since this is not easy to ensure, usually, this is relaxed requiring that the representation of the proposal distribution is the same. There are several reasons behind this choice. First, the sensors have different set of measurements and the particles are not compatible among the sensors, thus they cannot be shared. Second, even if the sensors agree upon a common, continuous, $p(z(k)|x(k))$, in the resampling stage they have to increase the number of particles in order to capture the important features of it. This last computation is something that should be avoided to limit the computational load.

For these reasons, recent research on DSPF is focused on guaranteeing that the PDF from which we draw samples is the same among the sensors. This idea is exploited in the papers of Sheng et al. (2005), Sheng and Hu (2005), Gu (2007) and Gu et al. (2008) where the authors use different models. In particular, the first three papers focus on a Gaussian Mixture Model (GMM). This representation has the drawbacks that, first, the sensors have to agree upon several variables if number of mixture is high, second, the local representation is built via an iterative optimization scheme, which requires time and may lead to local minima (see Sheng et al. (2005) for further details). On the contrary, as pointed out in the paper of Gu et al. (2008), the use of a mono-modal Gaussian distribution generated via an Unscented Transformation can have accurate results keeping the algorithm as simple as possible.

In fact, even if different representations are involved, these approaches can both fit naturally in the framework of consensus theory and they can be generalized in the context of Gaussian proposal distributions (Guo et al., 2007). The first step is to approximate the proposal distribution, for each particle $j$ using a Gaussian function as

$$q(x(k)|x_{k-1}, z_k) = q(x(k)|x(k-1), z(k)) = \mathcal{N}(\mu(k), \Sigma(k))$$

This proposal can also be a multi-modal Gaussian as in the GMM, and the following arguments will remain the same, but with more computations.

Since having different Gaussians for different particles is infeasible due to the fact that the sensors have different set of particles, as a second step we take $\mu(k)$ and $\Sigma(k)$ to be the same among the particles. The third step is representing the local mean $\mu_i(k)$ and the local covariance matrix $\Sigma_i(k)$ at each sensor $i$ as:

$$\mu_i(k) = \sum_{j=1}^{m} \omega_{i,j} x_i(k)^j$$

$$\Sigma_i(k) = \sum_{j=1}^{m} \omega_{i,j} (x_i(k)^j - \mu_i(k))(x_i(k)^j - \mu_i(k))^T$$

Then the problem of finding the couple $(\mu(k), \Sigma(k))$, given the different triples $(\mu_i(k-1), \Sigma_i(k-1), z_i(k))$ at each sensor, constraining the communication among them, can be formulated as a distributed estimation problem. This means that essentially well-known consensus protocols can be used for its solution, such as in Ren and Beard (2008) and Olfati-Saber (2007).

Algorithm 1 shows a high-level description of the method.

**Algorithm 1 DSPF, Gaussian proposal distribution**

```plaintext
1: if $k = 0$ then
2: $x(0) = x(0), \mu_i(0) = x(0), \Sigma_i(0) = \Sigma_0$
3: else
4: while new data exists do
5: Run a distributed estimation algorithm to agree upon $(\mu(k), \Sigma(k))$
6: Draw samples for each particle $j$:
7: $x(k)^j \sim \mathcal{N}(\mu(k), \Sigma(k))$
8: Discard the particles which do not satisfy the constraints (3).
9: Calculate the local weights:
10: $\omega_i^j = p(z(k)|x(k)^j)$
11: Calculate the local state estimate.
12: Resample using the local weights.
13: Calculate the new couple $(\mu_i(k), \Sigma_i(k))$.
end while
end if
```

We note that, as expressed in (Gu et al., 2008), if the sensors have the same number of particles $m$, each of the Particle Filters will have an accuracy comparable with a Particle Filter using global knowledge of all sensor data, $m$ particles, and the same type of proposal distribution as well. We conjecture that the accuracy of this centralized Particle Filter with $m$ particles represents a bound for the performance of the DSPF on each sensor.

### 3.2 Distributed Computation Particle Filters

The DCPFs aim to achieve a task that is different from the DSPF discussed so far. Given the set of all measurements, the DCPF problem asks the question of how to compute in a distributed way the best estimate. In other words, can we allocate a subset of particles to different computing cores and obtain the same accuracy as we would get if all the particles were used together, but without extensive communication among the cores?

Recent developments in general-purpose multi-core CPU architectures makes the study of this question very relevant and interesting. However, as it has been pointed out in the previous sections, distributing particle filter computations is not a trivial task, because usually the resampling step needs the knowledge of all the particles. For this reason the literature on this topic is not very extensive; the interested reader is referred to Brun et al. (2002) and Bolić et al. (2005) for some insights and algorithms which still involve some degree of centralized information.

We propose in the following a rather simple yet effective idea, which leads to distributed particle filter computations. Since all the cores have the same set of data, the
particles are compatible from core to core and can be shared. Of course, there is no point in sharing all the subsets of particles among the different cores, since this would simply lead to the non-distributed version. However, inspired by some ideas in Rosencrantz et al. (2003), it makes much more sense to share only a few representative particles. Let $N$ be the number of cores for instance. If each core is running $m$ particles and it shares only those $n < m$ with higher weights, then the communication is reduced and the accuracy will still approach that of a Particle Filter with $mN$ particles. A simulation example is shown in the next section to support this observation, while Algorithm 3 describes the main steps of the method.

**Algorithm 2** DCPF

1: if $k = 0$ then
2: $x(0)^j = x(0)$
3: else
4: while new data exists do
5: Draw samples for each $j$:
6: $x(k)^j \sim p(x(k)|x(k−1)^j)$
7: Discard the particles which do not satisfy the constraints (3).
8: Calculate the weights:
9: $\omega_k^j = p(z(k)|x(k)^j)$
10: Share $n$ particles with high weights with the neighbors.
11: Calculate the local state estimate.
12: Resample $m$ particles.
13: end while

4. A LOCALIZATION PROBLEM WITH UNICYCLE ROBOTS USING DCPF

The techniques presented in the previous section can be applied in various estimation problems. In this section, we consider a localization problem using noisy range measurements for mobile robot tracking, which is representative of a real testbed currently under construction by the authors.

Let $x_p(k) = (\xi, \eta)$ be the position vector of a robot in a 2D plane at a given time $k$. Let $\{s_1, …, s_N\}$ be the collection of the coordinates of the fixed sensor nodes. Let the range measurements for each sensor $z_i(k)$ be:

$$z_i(k) = ||x_p(k) - s_i|| + n_i(k) \quad i = 1, …, N$$

where $n_i(k)$ is a noise term. The localization problem can be expressed as finding an estimate of $x_p(k)$ given the measurements $z_i(k)$ and a set of constraints that define the operating area. The main issues to take into account are accuracy of the estimate, required sampling frequency and therefore computational effort of the algorithm, robustness against data loss and failures in the measurement process. The state of the unicycle robot is chosen as $x = \text{col}(x_p, \theta) = \text{col}(\xi, \eta, \theta)$, while the proposal distribution based on the state equation is:

$$\begin{align*}
\xi(k) &= \xi(k−1) + \tilde{v} \left(\sin(\theta(k−1) + r\Delta t) - \sin(\theta(k−1))\right) \\
\eta(k) &= \eta(k−1) - \tilde{v} \left(\cos(\theta(k−1) + r\Delta t) - \cos(\theta(k−1))\right) \\
\theta(k) &= \theta(k−1) + \tilde{r}\Delta t + \gamma\Delta t \\
\tilde{v} &= v + n_v \\
\tilde{r} &= r + n_r
\end{align*}$$

where $v$ and $r$ are the velocity and the angular velocity control inputs, respectively, and $n_v$, $n_r$, and $\gamma$ are noise terms. The weights can be written using a Gaussian PDF as

$$\omega_k^j = p(z(k)|x(k)^j) = \frac{1}{(2\pi)^{N/2}\det(\sigma^{-2}N)} \times \exp\left((d(k)^j - z(k))^T\sigma^{-2}(d(k)^j - z(k))\right)$$

where $\sigma$ is an $N \times N$ diagonal matrix collecting the expected standard deviations of the range measurement errors. Furthermore,

$$d_i(k)^j = \begin{bmatrix} \xi_j(k)^j \\ \eta_j(k)^j \\ 0 \end{bmatrix} - s_i$$

$$z(k) = \text{col}(z_1(k), …, z_N(k))$$

We assume to have 10 sensors sparsely distributed in the environment, and 16 cores to compute the estimate. Each core is assumed to have access to all the sensor measurements and can communicate with its neighbors. Figure 2 shows the computing core configuration, while Figure 3 depicts typical snapshots of a simulation with a Centralized Particle Filter. The particles are represented by dots, and the four distinct clouds refer to four different time instants. The robot trajectory is depicted in grey and it starts from the circle at the top of the figure. This figure is meant only to illustrate the constraint handling mechanism and the experiment environment, and is not representative of the performance of the actual algorithm. The particle clouds were made large artificially by avoiding the resampling step.

**Fig. 2.** Core configuration: the cores are represented by the squares, while the lines indicate possible communication paths.

The actual simulation parameters were $v = 30$ cm/s, $\sigma = 1$ cm, std($n_v$) = 50 cm/s, std($n_r$) = 0.1 rad/s and std($\gamma$) = 0.01 rad, where std is the standard deviation. Four different cases are considered:

- Case 1: $m = 5$, $n = 1$
- Case 2: $m = 10$, $n = 1$
- Case 3: $m = 20$, $n = 1$
Case 4: \( m = 80, \ n = 1; \)

where \( m \) is the number of particles in each core and \( n \) is the number of particles that each core sends to its neighbors.

The results are shown in Figure 4, where the left bar on each case (from 1 to 4) represents the average error of a Centralized PF with \( m \) particles. The middle bar and the right one represent the greatest and lowest average error of the DCPF on all cores. The last bar on the right represents the average error for a Centralized PF with \( m = 500 \) particles and it illustrates the performance that could be obtained with a high number of particles indicating an approximate lower bound. The vertical interval lines represent the standard deviation.

As this numerical example illustrates, the DCPF outperforms a standard Centralized PF. This can be seen more easily when the number of particles is low, since both approaches hit the same accuracy limit when using many particles. It can be seen that in case 1, the DCPF has an accuracy comparable with a centralized filter using \( mN = 80 \) particles. Moreover, this improvement seems not to depend on \( n \). This is encouraging since using the DCPF decreases computation time by a factor of at least \( (1 + 4n/m)/N; \) This speedup estimate follows from the fact that each core is at most dealing with \( m + 4n \) particles, where 4 is the maximum number of connections with neighboring cores. This result comes at the price of a small amount of communication among the cores. However, since the interconnection among them is rather sparse, it scales linearly with the number of cores, and it is not affected by data loss or network reconfiguration.

As a final remark, we note that the slight decrease in accuracy that is visible when the number of particles increases beyond a certain threshold, is probably due to the resampling strategy, and thus is inherent to the SIR Filter class and to the problem of loosing diversity in the particle population (Arulampalam et al., 2002).

5. CONCLUSIONS AND FUTURE WORK

In this paper we have presented several recent developments in Distributed Particle Filtering from a unified perspective. On one side, the distributed estimation problem was described, which can be rephrased as a consensus process on the proposal distribution. This leads to several open research problems, such as the use of different algorithms to estimate \( (\theta(k), \Sigma(k)) \). Moreover, a rigorous characterization of performance is still lacking, as we indicated by our conjecture in Section 3.1. On the other side, Particle Filters with distributed computation have created a rather new field of research. We have proposed a simple yet effective algorithm, which seems to perform very well in a numerical simulation, but more analysis needs to be done. We expect that a significant improvement in performance could be achieved especially for dynamical systems where the state vector dimension is high and therefore the number of particles should be high as well. We believe that there are ample opportunities for new algorithms based on a mixture of the presented techniques, giving birth to several open research questions in the near future.

REFERENCES


