Distributed Computation Particle Filters on GPU Architectures for Real-Time Control Applications

Mehdi Chitchian, Andrea Simonetto, Alexander S. van Amesfoort, and Tamás Keviczky

***Abstract*— We present the design, analysis, and real-time implementation of a distributed computation particle filter on a graphic processing unit (GPU) architecture that is especially suited for fast real-time control applications. The proposed filter architecture is composed of a number of local subfilters that can share limited information among each other via an arbitrarily chosen abstract connected communication topology. We develop a detailed implementation procedure for GPU architectures focusing on distributed resampling as a crucial step in our approach, and describe alternative methods in the literature. We analyze the role of the most important parameters such as the number of exchanged particles and the effect of the particle exchange topology. The significant speedup and increase in performance obtained with our framework with respect to both available GPU solutions and standard sequential CPU methods enable particle filter implementations in fast real-time feedback control systems. This is illustrated via experimental and simulation results using a real-time visual servoing problem of a robotic arm capable of running in closed loop with an update rate of 100 Hz, while performing particle filter calculations that involve over one million particles.**

***Index Terms*— Distributed algorithms, distributed computation particle filters, graphic processing unit (GPU) architectures, visual servoing.**

I. Introduction

LONG after their first appearance [1], particle filters still represent an active area of research. Because of their generality and simplicity, they have become a topic of constantly growing interest, development, and numerous applications. Particle filters, together with unscented Kalman filters [2] and moving horizon estimators [3], 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 out perform 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 this can seriously limit the applicability of particle filters.

Manuscript received April 19, 2012; revised November 2, 2012; accepted December 3, 2012. Manuscript received in final form December 13, 2012. Recommended by Associate Editor G. De Nicolao.

M. Chitchian and A. S. van Amesfoort are with the Parallel and Distributed Systems Group, Delft University of Technology, Delft 2628BX, The Netherlands (e-mail: mehdi.chitchian@gmail.com; a.s.vanamesfoort@tudelft.nl).

A. Simonetto and T. Keviczky are with the Delft Center for Systems and Control, Delft University of Technology, Delft 2628BX, The Netherlands (e-mail: a.simonetto@tudelft.nl; t.keviczky@tudelft.nl).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCST.2012.2234749

Recent developments aimed at overcoming this difficulty could lead to faster yet accurate algorithms. Most notably, the rise of general-purpose graphics processing unit hardware (GPU) is making massive parallelization of algorithms possible by distributing independent tasks over many computing units leading to significant improvements in execution time. This trend is reflected in reasonably priced desktop supercomputers hitting the market featuring a few thousand processing units. At the same time, clusters of these systems are becoming available to many researchers. Hence, the development of software frameworks that take advantage of massive parallelism is essential to enable novel applications in computational science and engineering.

In this paper, we focus on devising an algorithm for implementing particle filters on GPU architectures. The proposed method considers a particle filter as a network of smaller filters, where each of them exchanges data locally based on the network topology. The resulting algorithm relies on the local communication of the computing units (i.e., the smaller filters) and their distributed computations. In particular, we devise a distributed computation particle filter and demonstrate its performance through experimental and numerical tests. Besides its distributed nature, our method (in contrast to those available in the literature) provides the users with different tuning parameters to adapt the algorithm to their specific application. The mentioned characteristics enable us to implement on a robotic arm setup a real-time feedback controller running at 100 Hz based on the estimate of a distributed computation particle filter with over one million particles. To the best of our knowledge, this is one of the first real-time implementation of a feedback controller with these characteristics using particle filters, which supports the idea of fast yet accurate nonlinear estimation.

Our main contribution in this paper is the proposed distribution of the computations among the different computing units.[1] This proposed idea will be shown to outperform standard implementations based on parallel computing (instead of distributed ones). In particular, we will be able to increase the number of particles, the sampling frequency, and the state dimension often by orders of magnitude with respect to state-of-the-art GPU solutions. Furthermore, we will show that our scheme has comparable accuracy with centralized sequential particle filters (with the same number of total particles), which

[1] In this respect, even though we have implemented our solution on a GPU architecture, we could envision the same solution to be implemented in multicore systems or a combination of both, thus the idea is not hardware-specific.

require 10–100 times more computational time (when using a high number of particles) than our proposed distributed implementation.

The remainder of this paper is organized as follows. In Section II we introduce the concept of particle filters, and define the problem setup and the main research question, namely, the distribution of the computations among different computing units. Some background information on distributed techniques and the formal definition of distributed computation particle filters are given in Section III, whereas in Section IV we survey the available methods to solve the presented problem. We propose our algorithm in Section V, analyzing in detail its different parts, its capabilities, and the effect of the user-tunable parameters. Experimental and simulation results are described in Section VI to assess the validity of the approach and show that nonlinear estimation can be run efficiently and at a relatively high sampling frequency. Finally, in Section VII we draw our conclusions and recommendations for future research studies.

II. Problem Setup and Research Question

We consider the nonlinear time-invariant dynamical system

$$\mathbf{x}(k) = f(\mathbf{x}(k-1), \mathbf{w}(k-1)) \quad (1)$$

$$\mathbf{z}(k) = h(\mathbf{x}(k), \boldsymbol{\mu}(k)) \quad (2)$$

where f and h are nonlinear functions, $\mathbf{x}(k)$ is the state vector at discrete time k, the inputs $\mathbf{w}$ and $\boldsymbol{\mu}$ are the process and measurement noise terms, respectively, and $\mathbf{z}(k)$ is the measurement vector. We assume the initial condition $\mathbf{x}(0)$ to be known or estimated, while we also assume the noise terms to be independent random variables with probability density function (pdf) $\pi_{\mathbf{w}}(\mathbf{w})$ and $\pi_{\boldsymbol{\mu}}(\boldsymbol{\mu})$, respectively.

We are interested in estimating the state $\mathbf{x}(k)$ via the noisy measurements $\mathbf{z}(k)$. To solve this problem, we consider particle filter estimators. Particle filters estimate the state $\mathbf{x}(k)$ via the *a posteriori* pdf $p(\mathbf{x}(k)|\mathbf{z}(k))$. Since, in most cases, this *a posteriori* pdf cannot be evaluated because of the complexity of the underlying dynamical system (1), the basic idea is to draw m random samples, or particles, $\{\mathbf{x}(k)^j\}_{j=1,\ldots,m}$, from a given proposal distribution $q(\mathbf{x}(k)|\mathbf{z}(k))$ with the same support as $p(\mathbf{x}(k)|\mathbf{z}(k))$. Often, this proposal distribution is chosen to be the *a priori* distribution $p(\mathbf{x}(k)|\mathbf{x}(k-1))$, as is done in sample importance resample (SIR) filters. Adopting this choice, the random samples can be computed recursively as

$$\begin{aligned} \mathbf{x}(k)^j &= \mathbf{x}(0), \qquad \text{for } j = 1, \ldots, m \\ \mathbf{x}(k)^j &= f(\mathbf{x}(k-1)^j, \mathbf{w}(k-1)^j) \end{aligned} \quad (3)$$

where $\mathbf{x}(k-1)^j$ is the jth sample at the discrete time $k-1$ and $\mathbf{w}(k-1)^j$ is randomly drawn from the process noise pdf, i.e., $\mathbf{w}(k-1)^j \sim \pi_{\mathbf{w}}(\mathbf{w})$. These samples are then weighted recursively via [4]

$$\begin{aligned} w(0)^j &= \frac{1}{m}, \qquad \text{for } j = 1, \ldots, m \\ w(k)^j &= \frac{p(\mathbf{x}(k)^j|\mathbf{z}(k))}{q(\mathbf{x}(k)^j|\mathbf{z}(k))} \frac{p(\mathbf{x}(k)^j|\mathbf{z}(k))}{p(\mathbf{x}(k)^j|\mathbf{x}(k-1)^j)} \\ &= w(k-1)^j \, p(\mathbf{z}(k))|\mathbf{x}(k)^j) \end{aligned} \quad (4)$$

where $p(\mathbf{z}(k))|\mathbf{x}(k)^j)$ is the likelihood that the measure $\mathbf{z}(k)$ is observed given the sample $\mathbf{x}(k)^j$. If the measurement noise $\boldsymbol{\mu}$ is additive and its pdf $\pi_{\boldsymbol{\mu}}(\boldsymbol{\mu})$ is Gaussian with zero mean and Σ as covariance matrix, then (4) can be simplified (up to a normalization) into

$$w(k)^j = w(k-1)^j \exp\left(-||\mathbf{z}(k) - h(\mathbf{x}(k)^j)||^2_{\Sigma}\right) \quad (5)$$

where, for a vector v, the notation $||v||^2_A = v^\top A v$.

Given the couples $(\mathbf{x}(k)^j, w(k)^j)$, we can approximate the *a posteriori* pdf $p(\mathbf{x}(k)|\mathbf{z}(k))$ by the use of Dirac's deltas, δ, as

$$\begin{aligned} p\left(\mathbf{x}\left(k\right)|\mathbf{z}\left(k\right)\right) &\approx \hat{p}\left(\mathbf{x}\left(k\right)|\mathbf{z}\left(k\right)\right) \\ &= \frac{1}{\Omega\left(k\right)} \sum_{j=1}^{m} w\left(k\right)^j \delta\left(\mathbf{x}\left(k\right) - \mathbf{x}\left(k\right)^j\right) \end{aligned} \quad (6)$$

with $\Omega(k) = \sum_j w(k)^j$. Finally, the *a posteriori* pdf gives the means to estimate $\mathbf{x}(k)$ as $\hat{\mathbf{x}}(k)$, which was our objective. For example, we can choose $\hat{\mathbf{x}}(k)$ to be the particle with the highest weight.

In addition, after the weight determination, the particle population is typically resampled and the new weights are set to be identical. This 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 weights. However, it also introduces 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 contain many repeated points. Therefore the choice of the number of particles and of the resampling procedure fundamentally determines the properties of the particle filter.

We can summarize a prototypical SIR particle filter algorithm as follows [4]:

1) draw m samples $\mathbf{x}(k)^j$ from the *a priori* distribution $p(\mathbf{x}(k)|\mathbf{x}(k-1))$, using (3);
2) compute the importance weight $w(k)^j$ for each j, using (4);
3) compute the state estimate according to the approximated *a posteriori* pdf (6);
4) resample the set of particles according to their weight;
5) set $w(k)^j = 1/m$ for all particles j.

Typically, when the state dimension is larger than 5, a high number of particles is required to capture the nonlinear underlying dynamics and measurement equations (1) and (2) and to obtain accurate estimates. Often, this endangers the real-time capabilities of particle filters in usual single-core CPU systems, and it is a fundamental drawback when accurate yet fast algorithms are needed, for example, for high-frequency feedback control purposes. With the rise of reasonably priced GPU hardware, these disadvantages could be overcome by distributing the computation among the different computing

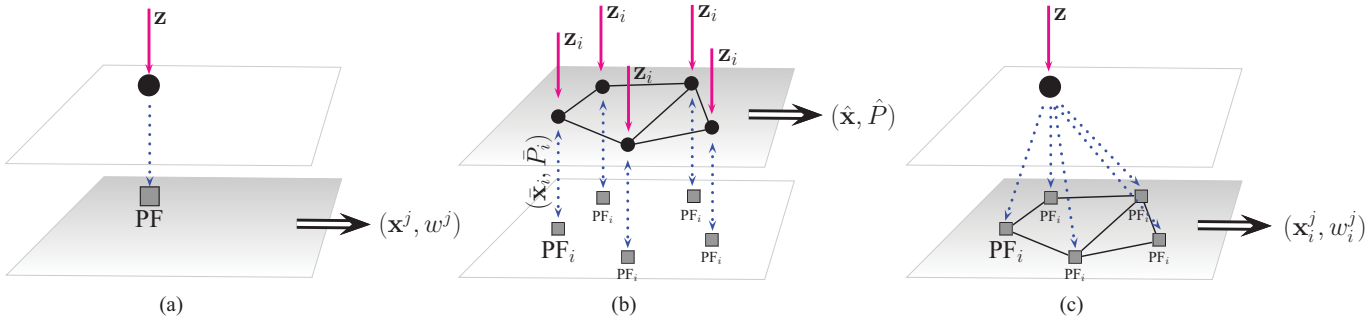

Fig. 1. Distributed particle filter classes along with the standard centralized approach. In the distributed sensing setting, the exchange of information and the distributed estimation process is done at the level of the sensors that share typically the local means and covariances of the state estimates $(\bar{x}_i, \bar{P}_i)$. Via some consensus mechanism [5], they agree on a common couple $(\hat{x}, \hat{P})$. In the distributed computation setting, the distribution and communication is done at the level of computing units with particle-weight couples. The final outcome is a number of different *a posteriori* distributions represented via $(\mathbf{x}_i^j, w_i^j)$. The fact that these distributions are different means that the different local particle filters do not necessarily have to agree on a common one. (a) Standard centralized approach. (b) Distributed sensing approach. (c) Distributed computation approach.

units, making accurate yet fast particle filters realizable. The main question we pose as a research problem is the following.

(Q) Can particle filters be run efficiently enough, for example, by the use of GPU architectures, and deliver accurate estimates to be implemented in high-sample-rate real-time feedback control applications?

III. Distributed Computation Particle Filters

Distributed algorithms for estimation have been receiving increased attention from the research community. The reason is twofold. First, 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. Second, computing the estimate using particle filters even with a few sensors can be computationally too intensive for a single-core CPU. Therefore the question of how to distribute this computation among different units becomes relevant.

In order to avoid a possible source of confusion given that the literature in this area can be less explicit and rather diverse, we believe it is important to distinguish the above two concepts in a clear way. We define two main classes of distributed particle filters as follows.

We call distributed sensing particle filter the class of algorithms that cope with a potentially large number of different sensors. Thus we assume that there is a particle filter running on each of these sensors and is using local measurements. The situation is depicted in Fig. 1(b), which shows a sensor network that processes the measurements and sends them to local particle filters. These filters cannot exchange information with each other and they can only send back to the sensors their local estimates with some meaningful parameterization of the *a posteriori* distribution (e.g., the mean and covariance of the estimate). Since, in general, the local particle filters have different particle population and measurements, they do not speak the same language. The exchange of information and the distributed estimation is done at the level of the sensor network, whose sensors do speak the same language (e.g., state estimates and their covariances). Examples for this class of approaches can be found in [6]–[11].

We refer to the second class of algorithms as distributed computation particle filters, which have access to all sensor measurements but use only a subset of particles in each computing unit. The different units where the distributed particle filters are running are depicted in Fig. 1(c). Since the different local filters have the same measurement vector, their language is compatible with each other and therefore they can exchange information, i.e., the particle-weight couples $(\mathbf{x}^j, w^j)$. The distributed estimation is done in this case at the level of the filters, whereas the sensors are only a means through which the measurements become available. This is the class of algorithms that we will study in this paper.

In the next section we will review the body of algorithms that can be considered a distributed computation particle filter. It is somewhat surprising that the exploitation (and design) of the communication network among the units, as sketched in Fig. 1(c), is a concept that is rather absent in the reviewed literature. This concept, which is extensively used in the sensor network community, is one of the main ingredients that will enable us to devise more efficient particle filters.

IV. Related Work

In the past decade, with the rise of the massive parallelization made possible by GPUs, many researchers have analyzed, studied, and designed versions of distributed computation particle filters. These algorithms differ in the number of particles they can handle, the specific parallelization, and the degree of communication between the computing units. In the next paragraphs we will examine a number of strategies to implement distributed and parallel particle filters.

To the best of our knowledge, the first work dealing with parallelism in particle filters is [12]. In this paper, the particle population is partitioned into several subsets, each assigned to a separate processor. Sampling, weight calculations, and resampling are performed independently and locally for each subset. The authors consider the weighted sum of all the particles as the estimate. This estimation is achieved by

calculating for each subset a local estimate and a local sum of weights, which are, subsequently, gathered centrally and combined into a global estimate. The authors show that local resampling is comparable with global resampling, in terms of estimation error.

In [13], three methods are proposed to implement distributed computation particle filters: 1) global distributed particle filter (GDPF); 2) local distributed particle filter (LDPF); and 3) compressed distributed particle filter (CDPF). With GDPF, only the sampling and weight calculation steps run in parallel on different processors, while resampling is performed centrally. All particle data is transferred to a central unit for the resampling step, and the new particles are sent back to each processor. The central unit calculates the global estimate from the particle data. With LDPF, resampling is also performed locally on each processor without any communication with other processors. Aggregated particle data is sent to a central unit in order to calculate the global estimate similar to the algorithm of [12]. In CDPF, similar to GDPF, resampling and the calculation of the global estimate are performed centrally, but only a small representative subset of the particles of each processor is sent to the central unit. This paper concludes from a number of simulations that LDPF provides both better estimation and performance.

Two distributed computation particle filter algorithms are proposed in [14]: 1) resampling with proportional allocation (RPA), and 2) resampling with nonproportional allocation (RNA). Both algorithms perform the sampling and weight calculation stages in parallel. In the RPA method, the resampling stage involves centralized communication, whereas in the RNA method it is performed completely locally. Different particle exchange mechanisms are discussed to improve the performance of this local resampling step, but it is rather unclear how these particles are selected. Furthermore, the number of particles that are exchanged among the cores is a significant ratio of the total population (at least 25% of all particles of each processing element). In both cases (RPA and RNA), the estimate is calculated as the weighted average of all particles from all processing elements. It is argued that RPA provides a better estimation, while RNA has a simpler design. In a later work [15], the authors compare a standard particle filter with a Gaussian particle filter on a field-programmable grid array (FPGA). The presented results indicate that the Gaussian particle filter, while being faster than a standard particle filter, is equally accurate for (near-) Gaussian problems.

A number of the previously presented algorithms (GDPF, RNA, RPA, Gaussian particle filter) are compared using a parallel implementation on a multicore CPU for a bearings-only tracking experiment in [16]. The comparison goes only until 10 000 particles. As expected, the Gaussian particle filter outperforms (in terms of accuracy over computational time) all other algorithms, since the estimation problem is Gaussian. The other particle filter algorithms (GDPF, RNA, RPA) exhibit similar estimation accuracy. In terms of runtime performance, both RNA and the Gaussian particle filter achieve near-linear speedup with respect to the number of cores for a large number of particles, while GDPF and RPA exhibit only sublinear speedup.

An interesting particle filter implementation is presented in [17], where the authors exploit GPU-specific hardware features. In this paper, first, a parallel approach for sampling and weight calculations is proposed, and then the resampling step is performed using a specific hardware feature of GPUs called the rasterizer. In practice, this step is close to the RNA algorithm of [14] but, since pseudorandom numbers are generated on the host CPU and successively transferred to the GPU, the performance of the filter is severely reduced. In fact, about 85% of the total runtime is spent on generating pseudorandom numbers and transferring them to the GPU, making this implementation unsuitable for real-time estimation in complex problems.

The GPU implementation described in [18] consists of parallel sampling, parallel weight calculations, and resampling performed locally on the different computing units. For the sampling step, the authors propose to use the finite-redraw importance-maximizing (FRIM) method, which checks the weight of the drawn particle and redraws until a particle with a reasonable weight is constructed. We note that the FRIM method is known to reduce the required total number of particles, but a fixed number for maximum number of redraws has to be imposed to limit the iterations. The generation of random numbers is performed on the host CPU, as in [17], and subsequently copied into the GPU. This makes their presented implementation rather limited. In fact, with the use of a low-performance laptop-GPU, they are able to run experiments only up to 4000 particles with execution times around 200 ms in the best case. It is unclear how the estimate is calculated from the weighted particle set and whether it is executed on the GPU.

A recent study [19] investigates a particle filter for localization and map matching for vehicle applications on a CPU using OpenMP and on a GPU using CUDA. The state dimension is only four, and the estimation does not benefit from more than 32 000 particles, but the application is nevertheless an interesting and well-explained case for particle filters. Experiments show that, with 128 000 particles, a CPU is 4.7 times faster on six cores than sequentially, while a GPU is another 16 times faster. The proposed algorithm runs parallel sampling and weight calculations but the resampling step is done on the host CPU in a centralized fashion. However, the resampling is performed only when the particle variance is above a threshold. This scheme seems to be faster on average than the other mentioned algorithms, but it suffers from high peaks of computation time when the resampling is performed, which is undesirable for real-time applications.

Another recent study [20] applies a GPU-accelerated particle filter for visual tracking. The particle filter is only partially executed on the GPU (and therefore this algorithm will not be used for comparison with other methods). The GPU is used for accelerating the image processing steps of the application. The sampling, estimation, and resampling stages are performed on the host CPU and only the weights are calculated on the GPU. They use a standard SIR particle filter implementation. The experiments go up to 4000 particles achieving around 40 f/s.

TABLE I
AVAILABLE METHODS TO IMPLEMENT DISTRIBUTED COMPUTATION PARTICLE FILTERS

Refs.		Sampling + Weight	Resampling	Estimation	Particles	State Dimension	Runtime [ms]
[12]		Local	Local	Central	32 000	3	1–100
[13]	GDPF	Local	Central	Central	5000	5	100
	LDPF	Local	Local	Central	5000	5	10
	CDPF	Local	Central§	Central§	5000	5	10
[14]	RPA	Local	Central§	Central	50 000	3	1
	RNA	Local	Local	Central	50 000	3	0.1–1*
[17]		Local	Central§	Central	1 000 000	2	1000
[18]		Local	Local	Unknown	4000	3	200
[19]		Local	Central	Central	130 000	4	10

§ only part of the particles are sent to a few specific coordinating computing units.

* these are theoretical limits based on the considered hardware rather than a measured performance.

Table I summarizes the surveyed methods and highlights the degree of centralization still presented in many of them. In particular, the resampling stage is performed either in a centralized fashion (where all or a significant part of the particles of each core are sent to some specific coordinating computing unit), or locally, without exchange of particles. This way of operating is typical from a parallel perspective.

We note, however, that these ways to resample the particle population can degrade the performance of the filter (in terms of computational time and accuracy) rather significantly. In order to address this problem, we will introduce in the next section the concept of distributed resampling, which will enable us to overcome these difficulties and achieve an improvement over the aforementioned methods. Furthermore, we will explain how the different (user-tunable) parameters can affect performance.

V. PROPOSED APPROACH

In this section we present our proposed approach to implement distributed computation particle filters. First we give a brief introduction to GPU architectures and we introduce the concept of topology. Then, we describe the distributed resampling techniques, which is the core of our method. Finally, after presenting our algorithm, we analyze the selection of its (user-tunable) parameters.

A. GPU Architecture and Topology

GPUs are programmed to exploit their inherent parallelism to execute a significant number of tasks at the same time. In this paper, we utilize CUDA [21] as a programming interface. A CUDA application is divided into a host side and a device side. The host refers to the CPU that is connected to one or more devices (i.e., the GPUs). The host manages device memory, initiates data transfers, and launches kernels on the device. Kernels are special functions executed on the device in parallel. Each kernel typically consists of numerous threads grouped into thread blocks. Limited fast-access shared memory is available to all threads from a single block for local communication, while slower access global device memory is available to all threads. The thread groups and the host can access the global device memory and this is typically the way the data is shared (although it is also the main cause of bottlenecks in standard implementations).

Fig. 2 depicts the terminology and the architecture. For our implementation, each thread corresponds to a single particle, while each thread block relates to a local particle filter.

In Fig. 2, we also illustrate the concept of topology, which will be important for our implementation. Since the access to the global device memory is usually a bottleneck, it is important to limit this operation and group the data that each thread block needs to read. Our idea is to map the global device memory into a specified topology that formally defines this grouping procedure. Using graph theory terminology [22], each thread block is a node, while if two nodes can access each other's data we say that there is an edge between them. The set of nodes $\mathcal{V}$ and the set of edges $\mathcal{E}$ define a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a specified topology. Since for a given graph $\mathcal{G}$ the topology is fixed, we often use the symbol $\mathcal{G}$ to refer to both interchangeably. In this context, each thread block has a set of neighbors, i.e., the thread blocks it can share the data with.

We consider each kernel to consist of N thread blocks (later local filters) labeled with the index $i = 1, \ldots, N$, whereas each thread block has m threads (later particles), labeled with the index $j = 1, \ldots, m$. The number of neighbors of each thread block is indicated with N_i.

Moreover, as a further abstraction of the hardware/software level, we refer to the thread blocks as computing units that are able to send and receive data from the neighboring computing units, via the graph $\mathcal{G}$. Since the access to global device memory is often a bottleneck and since the local shared memory is limited by the hardware, the communication among the computing units cannot grow arbitrarily.

B. Distributed Resampling

In typical implementations of distributed computation particle filters, the sampling and weight computation steps are done locally (in parallel) in a rather straightforward fashion. We also follow this standard strategy. The resampling step is however more delicate.

Resampling is a critical step in particle filters. On one side, it is necessary since it can provide the chance for good particles to spread themselves and it overcomes the degeneracy phenomenon, where after a few iterations, all but one particle will have negligible weights. However, on the other side, it introduces practical (and often application-specific) issues, notably the impossibility to run the code in parallel, since all the

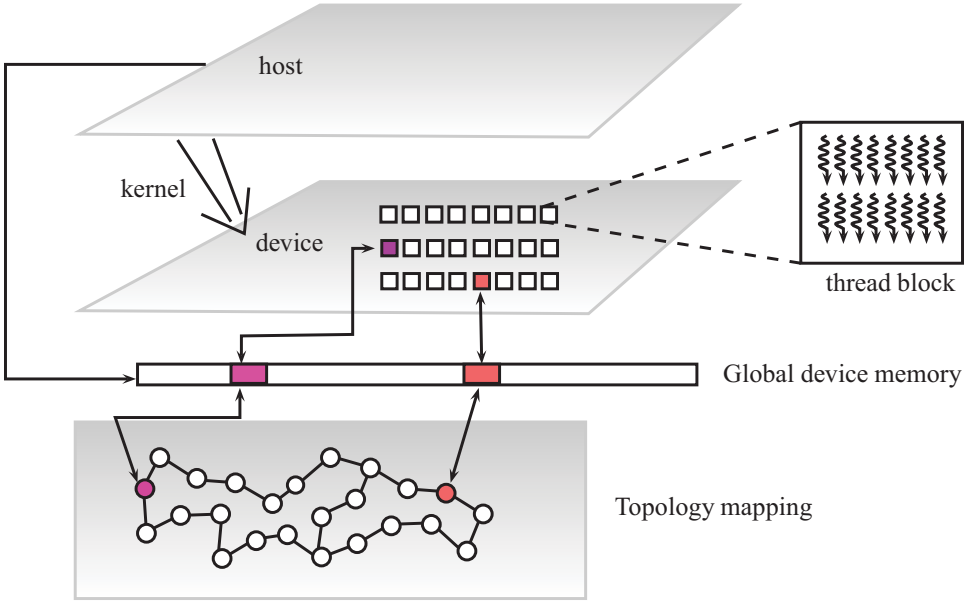

Fig. 2. Basic concept of GPU architectures as available in CUDA. The global device memory is mapped into a specific topology representing data exchange between thread blocks.

particles must be combined, and the loss of diversity among the particles as the resultant samples contain many repeated points. In this context, it is rather crucial to devise carefully the resampling stage and give to the user tunable parameters to overcome the mentioned application-specific issues.

As surveyed in Section IV, the resampling step is often performed with a degree of centralization. Typically, all the particles or (a significant) part of them are sent to a limited number (often one) of coordinating computing units (i.e., thread groups) that are in charge or performing all the calculations.

In contrast with these methods, we propose a distributed approach to the resampling stage. Our idea is rather simple yet extremely effective; it is based on the consideration that there is no need to have specific coordinating computing units that collect particles and perform calculations. In fact, each of the already present computing units (the local particle filters) can do the calculation themselves. Furthermore, there is no need to send all the particles to all the units; in fact, only a few are necessary to the resampling step. In our approach, each computing unit sends to its neighbors only t representative particles (the ones with locally the highest weights) and performs the resampling stage on its resulting $m+tN_i$ particles (we recall that N_i is the number of neighbors a computing unit has). We note that this simple idea is extremely powerful. In fact, we note the following.

(P1) The number of shared particles is a small part of the population, since typically $tN_i \ll m$. However, as we will see in the experimental and simulation tests of Section VI, the fact that the method is not completely local (meaning $t > 0$, in contrast with local methods where $t = 0$, for example [12], [14], [18]) increases significantly the accuracy of the filter.

(P2) There is no need for centralized data collection, making the resampling step fast and efficient.

(P3) Both the topology and the number of shared particles t are user parameters that can be adjusted to the application at hand. In some cases (for example, in high process noise setting), we will see that having an all-to-all topology (similar to the centralized collection unit in the CDPF algorithm of [13]) could lead to worse performance than a ring topology. As we will explain in Sections V-E and VI, this is due to the loss of diversity introduced by the resampling stage.

This idea of distributed resampling has been presented in [23]. A similar approach is also presented in [24], whose authors propose a modification of the distributed resampling idea of [23] on a ring topology, where the local computing units substitute their highest weight particles with the ones of the neighboring units. Both these works show in simple numerical simulations the similar performance of the distributed strategy with a standard single-core CPU implementation that uses the same number of particles Nm, in terms of estimation quality.

In contrast and in addition to the aforementioned works, in this paper we implement the distributed resampling scheme on a GPU architecture and test it on a real hardware platform. Furthermore, we analyze the effect of the parameters of algorithm, namely, the number of shared particles t and the topology $\mathcal{G}$ on the quality of the estimate.

C. Proposed Algorithm

Our proposed algorithm consists of the following high-level steps.

1) *Sampling and Weight Calculation:* this step is done locally on each computing unit.
2) *Distributed Resampling Step:* this step is done in a distributed computation fashion as explained in Section V-B.

Algorithm 1 Distributed Computation Particle Filter: High-Level Description on Each Computing Unit i

Input: $\{\mathbf{x}_i(k-1)^j\}_{j=1,\ldots,m}, \mathbf{z}(k)$

1. Local Filter:

for $j = 1$: m

1.1: sampling: $\mathbf{x}_i(k)^j \sim p(\mathbf{x}_i(k)|\mathbf{x}_i(k-1)^j)$

1.2: weight_calculation: $w_i(k)^j = p(\mathbf{z}(k)|\mathbf{x}_i(k)^j)$

end

2. Sorting: sort $\{\mathbf{x}_i(k)^j\}_{j=1,\ldots,m}$ according to $\{w_i(k)^j\}_{j=1,\ldots,m}$

3. Estimation: local_estimation: pick $\mathbf{x}_i(k)^j$ with maximal $w_i(k)^j$

4. Particle Exchange:

foreach neighbor **do**

send and receive t particle-weight couples to/from neighbors

end

5. Resampling: resample the $m + N_i t$ particles into m particles

6. Reset: set $w_i(k)^j = 1/m$ for all j

Output: $\{\mathbf{x}_i(k)^j\}_{j=1,\ldots,m}$

3) *Local Estimation:* this step is done locally on each computing unit, picking the local particle with maximum weight.
4) *Global Estimation:* in addition to local estimation, we provide to the user a global estimate which is selected as the best particle among the local estimates. This selection is also done in parallel via a parallel reduction on the winners from each block. In our experiments we have noted that the extra run-time spent in the global estimation kernel is extremely limited and completely justifiable. However, we remark that local estimation is already sufficient to obtain accurate results.

Algorithm 1 describes more in detail how these three high-level phases are translated into high-level commands, while high-level implementation details are presented next. For more hardware- and implementation-related details, the reader is referred to [25].

D. High-Level Implementation Details

The details of how Algorithm 1 has been implemented on real hardware platforms is fully reported in [25]. Nonetheless, the next paragraphs give an account of these implementation details from a high-level perspective.

First of all, as mentioned in Sections V-A and V-C, each particle is processed by a single GPU thread and each local particle filter by a thread block. The particle filter operations of Algorithm 1 along with the important, but often overlooked, pseudorandom number generation are mapped to the following CUDA kernels.

1) *Pseudo-Random-Number Generation:* Particle filters rely heavily on (pseudo-random) number generators (which are needed for both the sampling as well as resampling calculations). The generation is done on the GPU at the start of each filter iteration using a GPU-optimized variant of Mersenne–Twister [26], which is a highly popular pseudo-random-number generator for Monte Carlo simulations.
2) *Sampling and Importance Weight Calculation:* This kernel is rather standard, see, for example, [16].
3) *Local Sorting:* All particles within each local filter are also sorted according their weights, in order to facilitate particle exchange and local estimation. This procedure is implemented with a bitonic sorting network, for an efficient parallel execution on the GPU.
4) *Local/Global Estimate:* We select the particle with the highest weight for each block, which has already been calculated by the previous kernel. Moreover, we provide to the user a global estimate which is selected as the best particle among the local estimates.
5) Particle Exchange: The exchange topology ($\mathcal{G}$) determines whom each filter exchange particles with.
6) *Resampling:* The resampling step generates a new particle set by drawing randomly (with replacement) from the original set according to particle weights. This kernel is implemented in two stages. First, the cumulative (prefix) sum of all the weights is computed. Second, for each particle a previously drawn uniform random number from $\mathcal{U}(0, \sum_j w_i(k)^j)$ is matched to a new particle using a binary search over the cumulative sum of the weights.

E. Analysis of the Algorithm

In this section we provide some insights in the selection of the parameters t and $\mathcal{G}$ of Algorithm 1, while, in general, the selection of N and m are dictated by hardware limitations.

Let $\hat{p}_i(\mathbf{x}(k)|\mathbf{z}(k))$ be the local approximation of the *a posteriori* pdf, different for each local filter, which can be written as

$$\hat{p}_i\left(\mathbf{x}\left(k\right)|\mathbf{z}\left(k\right)\right) = \frac{1}{\Omega_i\left(k\right)} \sum_{j=1}^{m} w_i\left(k\right)^j \delta\left(\mathbf{x}\left(k\right) - \mathbf{x}_i\left(k\right)^j\right) \quad (7)$$

where the subscript i indicates that weights and particles are referred to the local filter i, and $\Omega_i(k) = \sum_{j=1}^{m} w_i(k)^j$. Let $\tilde{w}_i(k)^j$ be the weight of particle j of filter i after the communication step but before resampling. Let $\tilde{m}_i = m + N_i t$ be the total number of particles of filter i before resampling, and let $\tilde{\Omega}_i(k) = \sum_j \tilde{w}_i(k)^j$.

First of all, we support the intuitive claim that the most representative particles of the $\hat{p}_i(\mathbf{x}(k)|\mathbf{z}(k))$ in (7) are the ones with highest weights, which justifies the communication strategy in Algorithm 1. In order to show this, we utilize the Kullback–Leibler (KL) divergence [27] which measures the distance between two pdfs. In particular, the smaller the KL divergence, the closer the two pdfs. Consider the approximated *a posteriori* $\hat{p}_i^{(t)}(\mathbf{x}(k)|\mathbf{z}(k))$ computed using only $t < m$

particles, as

$$\hat{p}_i^{(t)}\left(\mathbf{x}\left(k\right)|\mathbf{z}\left(k\right)\right)=\frac{1}{\Omega_i^{(t)}\left(k\right)}\sum_{j=1}^{t}w_i\left(k\right)^j\,\delta\left(\mathbf{x}\left(k\right)-\mathbf{x}_i\left(k\right)^j\right) \tag{8}$$

with $\Omega_i^{(t)}(k)=\sum_{j=1}^{t}w_i(k)^j$; then the claim that the t particles with highest weights are the most representative for (7) is formally expressed as follows.

Proposition (From [24]): The KL divergence between (7) and its approximation (8), which employes only $t<m$ particles, i.e., $D(\hat{p}_i,\,\hat{p}_i^{(t)})$, can be written as

$$D(\hat{p}_i,\,\hat{p}_i^{(t)})=-\log\left(\sum_{j=1}^{t}\frac{w_i(k)^j}{\Omega_i(k)}\right)=-\log\left(\frac{\Omega_i^{(t)}(k)}{\Omega_i(k)}\right) \tag{9}$$

and it is minimal when we use for (8) the t particles with the highest weights.

We can distinguish two main aspects that affect the performance of Algorithm 1. First, we have to analyze how well each of the local *a posteriori* $\hat{p}_i(\mathbf{x}(k)|\mathbf{z}(k))$ represents the global $\hat{p}(\mathbf{x}(k)|\mathbf{z}(k))$. This aspect dictates the estimation quality of the distributed computation particle filter with respect to a classical sequential single-core CPU implementation. Second, it is important to study how distorted each of the local $\hat{p}_i(\mathbf{x}(k)|\mathbf{z}(k))$ becomes after the resampling step. This distortion is a measure of the distance between the resampled and the initial population. In particular, high values of distortion generally mean that the filter will be affected by the degeneracy/loss of diversity phenomenon, where only a few particles have nonzero weight.

In order to analyze the quality of each local *a posteriori* $\hat{p}_i(\mathbf{x}(k)|\mathbf{z}(k))$ with respect to the global $\hat{p}(\mathbf{x}(k)|\mathbf{z}(k))$, we make use of Proposition 1 applied to these distributions and we define the cumulative sum of the local KL divergences as

$$\begin{aligned}\sum_{i=1}^{N}D(\hat{p},\,\hat{p}_i)&=-\sum_{i=1}^{N}\log\left(\sum_{j=1}^{\tilde{m}_i}\frac{w_i(k)^j}{\Omega(k)}\right)\\&=-\sum_{i=1}^{N}\log\left(\frac{\Omega_i(k)}{\Omega(k)}+\sum_{j=m+1}^{m+N_it}\frac{w_i(k)^j}{\Omega(k)}\right)\end{aligned} \tag{10}$$

where we recall that $\Omega(k)=\sum_{j=1}^{Nm}w(k)^j$.

Let $W=\sum_{j=m+1}^{m+N_it}w_i(k)^j$. By the fact that we are sharing the t particles with highest weight, we could approximately consider $w_i(k)^j$ to be the same for each j, i.e., $w_i(k)^j\approx\bar{w_i}(k)$, and approximate W as

$$W\approx\bar{w_i}(k)N_it.$$

Consider the derivative of the cumulative sum (10) with respect to W (and therefore with respect to N_it since $\bar{w_i}(k)$ is constant) as

$$\frac{\partial}{\partial W}\sum_{i=1}^{N}D(\hat{p},\,\hat{p}_i)=-\frac{\Omega(k)}{\Omega_i(k)+W} \tag{11}$$

which is a minimum for $W=0$.

From (10) and (11), we can infer the following.

(S1) The higher the N_it, the closer the local and global *a posteriori* pdfs are. This follows from (10) with $W\to\infty$ (or in the approximated sense, with $N_it\to\infty$). Therefore, increasing the communication leads to an increase of estimation quality for a given time step k (note that the effect on the time step $k+1$ depends also on the resampling, which is analyzed next). In particular, if we choose all-to-all communication and $t=m$, then each local population is comprised at least of the m particles with highest weight, which are the most representative to describe $\hat{p}(\mathbf{x}(k)|\mathbf{z}(k))$.

(S2) The gain in increasing N_it, or for a given N_i, in increasing the number of shared particles t, is maximum when $W=0\approx N_it$. In other words, we can expect a more significant increase in the estimation quality passing from $t=0$ to $t=1$ than passing from $t=1$ to $t=2$.

Besides choosing t and N_i (and therefore the topology $\mathcal{G}$) to minimize the KL divergence between the local $\hat{p}_i(\mathbf{x}(k)|\mathbf{z}(k))$ and the global $\hat{p}(\mathbf{x}(k)|\mathbf{z}(k))$ while maintaining N_it as small as possible to limit the communication effort, the effect of the resampling stage is also an important aspect to consider. Assume that $N_it\ll m$ and thus consider the local particle population to be uncorrelated among the local filters i. Define the distortion [28] of $\hat{p}_i(\mathbf{x}(k)|\mathbf{z}(k))$ after the resampling step as its KL divergence with the global *a posteriori* $\hat{p}(\mathbf{x}(k)|\mathbf{z}(k))$. We note that this measure is different from the one in (10), since we use for $\hat{p}_i(\mathbf{x}(k)|\mathbf{z}(k))$ the resampled weights (before resetting them). The dependence of the distortion on the local weights before resampling $\tilde{w}_i(k)^j$ can be approximated as [28]

$$\mathcal{D}_{\tilde{w}}\approx\sum_{i=1}^{N}\Omega_i(k)\sum_{j=1}^{\tilde{m}_i}\left\lceil\frac{\tilde{w}_i(k)^j}{\Omega_i(k)}\right\rceil\log\left(\left\lceil\frac{\tilde{w}_i(k)^j}{\Omega_i(k)}\right\rceil\right). \tag{12}$$

In order to avoid distortion and, therefore, to maximize the estimation quality, $\mathcal{D}_{\tilde{w}}$ has to be minimized, which is achieved when the different $\tilde{w}_i(k)^j$ within the same local filter i have similar values. On the contrary, when only few particles have the highest weights, the distortion is close to its maximum and, therefore, degeneracy can be expected to occur. In particular, we can distinguish two extreme cases.

(S3) Relatively low process noise but high measurement noise. In this case, the particles have similar weights and, therefore, the resampling step does not cause high levels of distortion, even for $N_it\neq 0$.

(S4) Relatively high process noise but low measurement noise. In this case, few particles have the highest weight and, therefore, the resampling step causes high levels of distortion. In particular, given t, the higher the N_i, the higher the distortion.

In the next section we will illustrate in practice the insights on the selection of the parameters t and N_i, which we have presented in this section.

VI. Numerical and Experimental Results

A. Setup

In order to test, verify, and benchmark our distributed computation particle filter implementation, we use the realistic

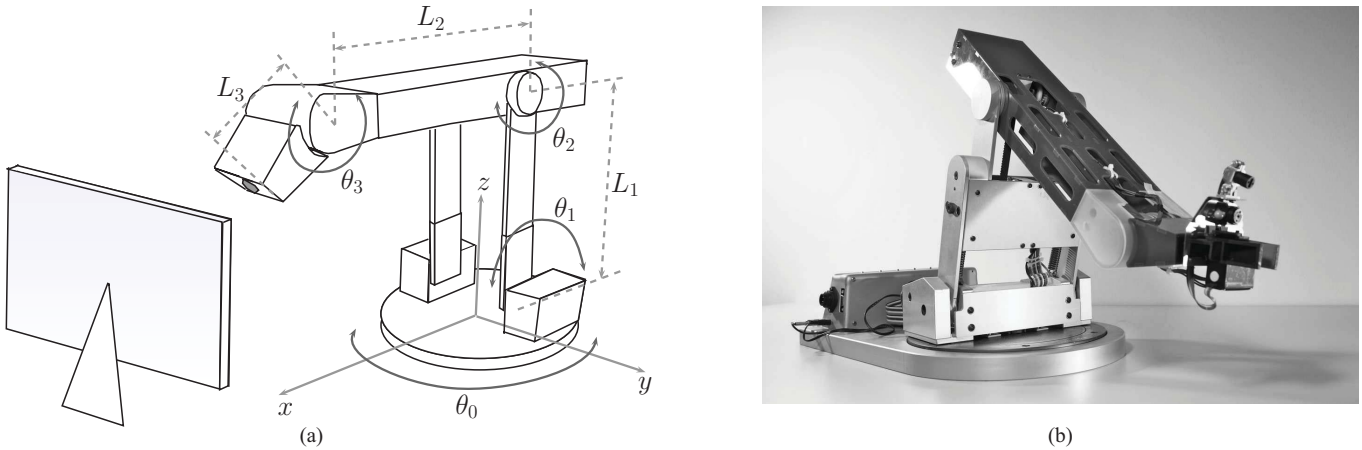

Fig. 3. Robotic arm used for the experimental test. (a) Real testbed. (b) Schematic representation showing the camera at the end tip, marked with a grey circle, and the monitor on which a moving object are displayed.

industrial application of a robotic arm. The main reason for such a choice is that the measurement equations of this application are highly nonlinear and extremely challenging for standard estimation techniques both in terms of accuracy and computational time.

The robotic arm in this experiment has a number of joints, $J = 3$, which can be controlled independently. It has one degree of freedom per joint plus the rotation of the base. Each joint has a sensor to measure its angle. There is a camera mounted at the end of the arm. This camera is used for tracking an object which is moving on a monitor on a fixed yz plane. The real robotic arm as well as a schematic representation is shown in Fig. 3.

Let $\theta_i(k)$ be the angle of the joint i at the discrete time k ($i = 0$ represents the rotational degree of freedom of the base). Let $\mathbf{p}_{\mathrm{w}}(k) = (x(k), y(k), z(k))^\top \in \mathbb{R}^3$ be the position of the object to be tracked at the discrete time step k in the fixed reference system of the robotic arm, as indicated in Fig. 3, while let $(v_x(k), v_y(k), v_z(k))^\top \in \mathbb{R}^3$ be its velocity. We consider $x(k)$ to be known *a priori* and $v_x(k)$ to be zero for all k. Denote by

$$\mathbf{x}(k) = (\theta_0(k), \ldots, \theta_J(k), y(k), z(k), v_y(k), v_z(k))^\top$$

the state of the arm and object dynamics. We model the angle dynamics as discrete-time single integrators and the object dynamics as a discrete-time double integrator,

$$\theta_i(k) = \theta_i(k-1) + \mathbf{w}_{\theta_i}(k-1), \quad i = 0, \ldots, J \tag{13a}$$
$$y(k) = y(k-1) + v_y(k-1)\Delta t + \mathbf{w}_y(k-1) \tag{13b}$$
$$z(k) = z(k-1) + v_z(k-1)\Delta t + \mathbf{w}_z(k-1) \tag{13c}$$
$$v_y(k) = v_y(k-1) + \mathbf{w}_{v_y}(k-1) \tag{13d}$$
$$v_z(k) = v_z(k-1) + \mathbf{w}_{v_z}(k-1) \tag{13e}$$

where the terms $\mathbf{w}$ model the process noise and Δt is the sampling time. The system of dynamical equations (13) will represent our *a priori* distribution $p(\mathbf{x}(k)|\mathbf{x}(k-1))$, as explained in Section II, (3).

The camera mounted at the end effector of the robotic arm detects the object displayed on the monitor in its own frame of reference. Let $\mathbf{p}_{\mathrm{s}}(k) = (x_{\mathrm{c}}(k), y_{\mathrm{c}}(k))^\top$ be the position of the object in the camera moving frame at the discrete time k. This position is measured in pixels. To relate $\mathbf{p}_{\mathrm{s}}(k)$ to the actual coordinates of the object in the robot fixed frame, first, we have to use a camera model that translates pixels into meters, and then perform a chain of translations and rotations to change the reference frame. The camera is modeled by the traditional pinhole projection with added radial lens distortion, see [29]–[31] for details. The model for the measured observations of the moving object is composed of three classes of maps: rigid body transformations, projections, and "distortion" maps. We emphasize the first two, since the lens distortion is known, i.e., the camera is calibrated *a priori*. Let $\mathbf{p}' = (x', y', z')^\top$ be a 3-D point described in generic coordinates. Let $\varphi : \mathsf{SE}(3) \times \mathbb{R}^3 \to \mathbb{R}^2$ be the standard rigid body transformation

$$\varphi(R, \mathbf{p}', \mathbf{q}) = R\mathbf{p}' + \mathbf{q}$$

where R and $\mathbf{q}$ are the traditional rotation matrix and translation vector, respectively. The camera pinhole projection model is realized by the projection map $\pi : \mathbb{R}^3 \to \mathbb{R}^2$

$$\pi(\mathbf{p}') = \frac{1}{z'}\begin{pmatrix} x' \\ y' \end{pmatrix}.$$

The composition of the maps is described graphically by the informal diagram

$$\underset{\text{world}}{\mathbf{p}_{\mathrm{w}}} \overset{\varphi}{\longleftrightarrow} \underset{\text{camera}}{\mathbf{p}_{\mathrm{c}}} \overset{\pi}{\longrightarrow} \underset{\text{plane}}{\mathbf{p}_{\mathrm{p}}} \overset{\psi}{\longleftrightarrow} \underset{\text{sensor}}{\mathbf{p}_{\mathrm{s}}}$$

where ψ describes the lens distortions. The full sensor model is described by

$$\mathbf{p}_{\mathrm{s}} = \psi \circ \pi \circ \varphi(R, \mathbf{p}_{\mathrm{w}}, \mathbf{q}) + \mu_{\mathrm{s}} \tag{14}$$

with μ_{s} an additive noise term. We note that the couple $(R, \mathbf{q})$ depends nonlinearly on the configuration of the robotic arm and, thus, on the angles θ_i and on the geometry, i.e., the length and the number J of the joints. Hence the model (14) can be translated into the compact measurement equation

$$\mathbf{p}_{\mathrm{s}}(k) = h_{\mathrm{s}}(\mathbf{x}(k)) + \mu_{\mathrm{s}}(k) \tag{15}$$

where h_s is the nonlinear function that represents the composition of the three maps in the sensor model (14). In order to complete our measurement model, we add the independent measurements of the angles

$$\tilde{\theta}_i(k) = \theta_i(k) + \mu_{\theta_i}(k), \qquad i = 0, \ldots, J \tag{16}$$

(with μ_{θ_i} sensor noise). Denote the measurement vector by $\mathbf{z} = (\mathbf{p}_s^\top, \tilde{\theta}_0, \ldots, \tilde{\theta}_J)^\top$ and the measurement noise vector with $\boldsymbol{\mu} = (\mu_s^\top, \mu_{\theta_0}, \ldots, \mu_{\theta_J})^\top$. We can write the complete measurement equation for the robotic arm setup as

$$\mathbf{z}(k) = h(\mathbf{x}(k)) + \boldsymbol{\mu}(k). \tag{17}$$

We note that (17) is a special case of (2) with additive measurement noise, from which we derive the *a posteriori* distribution $p(\mathbf{z}(k)|\mathbf{x}(k))$.

In the next subsections we will analyze several experimental and simulation results. The first aim of the experiments is to show the performance of the proposed Algorithm 1 in estimating the state $\mathbf{x}$ given the noisy observation $\mathbf{z}$. In particular, we will focus only on a part of the state, namely, the position of the object in the world coordinates, i.e., (y, z). We will describe the dependences of the estimation error on the different parameters of the algorithm, as well as its runtime performances. We will show its scalability with respect to the dimension of the state vector (arbitrarily varied in the simulation runs changing the number of joints J, therefore augmenting the number of θ_i to consider in (13a) and changing the geometry, i.e., the rigid body transformation φ in (14)).

The second aim of the experiments is to demonstrate one of the main contributions of this paper—fast real-time feedback control based on the proposed algorithm is possible and can achieve satisfactory results, in contrast with traditional single-core CPU implementations. The control objective is to track a moving object with the robotic arm while it traverses the screen, as described next.

For experimental and simulation purposes, we use the commercially available GTX 580 GPU and, unless differently stated, we choose a ring topology for the underlying communication graph, and we select $t = 1$ for the number of exchanged particles among the computing units.

B. Experimental Results

We use the robotic arm platform of Fig. 3 for the experiments throughout this section in order to examine the filter behavior under different conditions. The parameters of the platform and the particle filter are listed in Table II (Experiments column), where all noise terms are modeled as Gaussian (since this turns out to be a rather realistic model for the noise of the setup).

In the experiments, the robotic arm is directed to follow a moving object (a white dot) while it traverses the monitor. The robotic arm pose is controlled so that the camera keeps the object in view while staying at about 2 cm from the monitor itself (thus the camera does not have a view of the whole monitor), see Table II. The implemented controller is a discrete-time proportional-integral-differential (PID) controller based on the estimated position and the estimated joint angles

TABLE II
EXPERIMENT AND SIMULATION PARAMETERS

	Experiments	High-Noise Simulations
Process Noise*		
$\mathbf{w}_{\theta_i}$	0.015 rad	0.075 rad
$\mathbf{w}_y, \mathbf{w}_z$	0.001 m	0.005 m
$\mathbf{w}_{v_y}, \mathbf{w}_{v_z}$	0.05 m/s	0.25 m/s
Measurement Noise*		
μ_s	10 px	10 px
μ_{θ_i}	0.01 rad	0.01 rad
Other Parameters		
Δt	0.01s	0.01s
T_f**	200s	200s
Velocity of the Target	$\simeq 0.03$ m/s	$\simeq 0.03$ m/s
$\frac{\text{Camera View Area}}{\text{Total Area}}$	12%	–

* both process noise and measurement noise are chosen to be Gaussian with zero mean and indicated standard deviation.
** corresponding to 10 complete trajectory loops.

with the sampling rate of 100 Hz. This update rate is close to the hardware limit.

The performance metric we consider is the estimation error of the position of the target.[2] In particular we define the average error $\mathbf{e}$ as

$$\mathbf{e} = \frac{1}{T_f} \sum_{k=1}^{T_f} ||\hat{\mathbf{p}}_\mathrm{w}(k) - \mathbf{p}_\mathrm{w}(k)|| \tag{18}$$

where $\hat{\mathbf{p}}_\mathrm{w}(k)$ is the estimated position by the filter and $\mathbf{p}_\mathrm{w}(k)$ is the true position, while T_f is the final discrete time step of the experiment and $||\cdot||$ represents the 2-norm. We remark that, by definition, $\mathbf{e}$ is always positive, i.e., $\mathbf{e} > 0$.

Fig. 4 illustrates the estimation of the position (y, z) superimposed on the actual trajectory of the object in low particle and high particle settings. In Fig. 4(a) we have selected $N = 32$ and $m = 64$, and we note that the robotic arm loses track of the object and it cannot complete the trajectory. On the contrary, In Fig. 4(b), in the setting $N = 2048$ and $m = 512$, we can achieve better estimation which translates into the accomplishment of the object following the task. In Fig. 4(a), we have also indicated the dimensions of the camera view area (empty rectangle).

Fig. 5 shows the average estimation error $\mathbf{e}$ for different settings (the standard deviation is not displayed but is around 1 mm for all the considered settings). We note the general (expected) trend that a higher number of filters increases the accuracy. Moreover, we achieve an average error of 3 mm for the best setting (with 1M particles at 100 Hz), which is considered a remarkable result given the experimental setup.

[2]This metric has been chosen because it provides an intuitive and physical interpretation of the average error in terms of distances, which is our primary focus for control purposes. Other metrics that could offer valuable insights in the performance of the algorithm are, for example, the total error metric based on the whole state error, considered in [25], or a weighted metric based on an estimation of the covariance matrix of the *a posteriori* distribution.

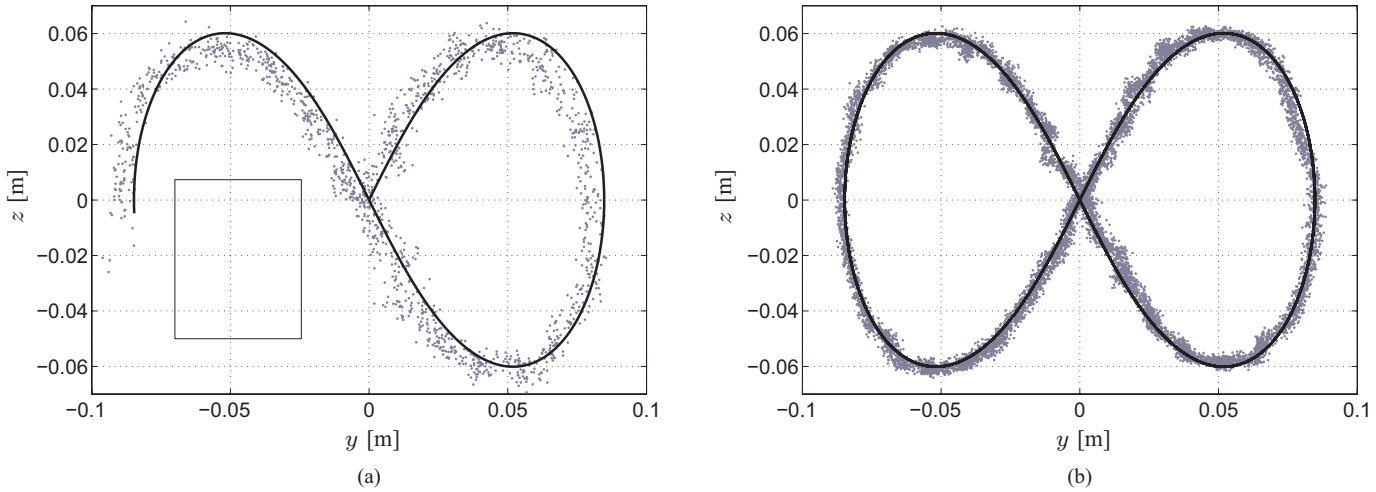

Fig. 4. Estimated position (y, z) (in grey dots) superimposed on the actual trajectory of the object. (a) Setting is $N = 32$ and $m = 64$, which does not allow the robotic arm to follow the object. (b) Setting is $N = 2048$ and $m = 512$, which allows the robotic arm to follow the object very well. Both experiments have been run at 100 Hz. The empty rectangle on the left figure represents the camera view area.

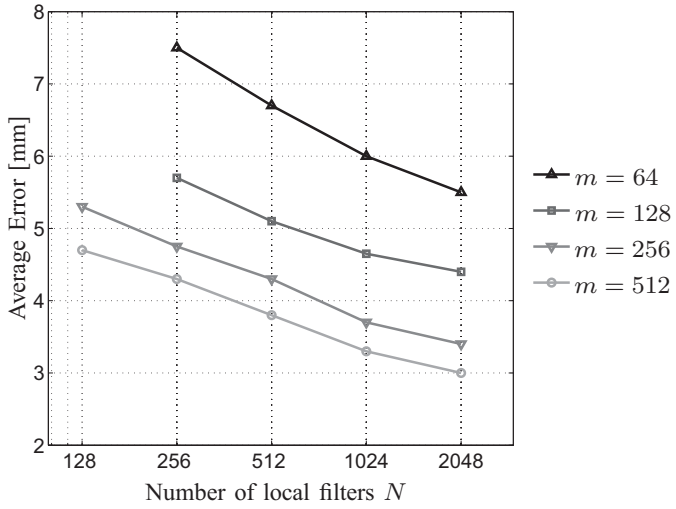

Fig. 5. Averaged estimation error $\mathbf{e}$ in a number of experiments with different (N, m) settings. The standard deviation (not shown) is around 1 mm.

C. High-Noise and Large-Scale Simulation Results

In order to further assess our implementation in different scenarios, we simulate the filtering problem under high process noise and in large-scale settings. Both cases serve to illustrate the performance of the algorithm in situations that might be encountered in real-life applications.

First of all, we increase the noise parameters as expressed in Table II and perform 100 simulation runs for each (N, m) setting (therefore the average error is also averaged on the 100 simulation runs). Figs. 6–7 show the results for different topology choices $\mathcal{G}$, and thus for different N_i, and for a different number of exchanged particles t. Although we have performed simulations with several different $\mathcal{G}$ and t, we report here only the most indicative ones, while the interested reader is referred to [25]. In Figs. 6 and 7, the standard deviation of the averaged error is not depicted for readability but it varies between 40% of the average value, for cases in which $\mathbf{e} \gg 10$ mm, and less than 10% of the average value when $\mathbf{e} < 10$ mm (this will be also the case in Fig. 10).

As we may note from Fig. 6 (where we use $t = 1$), in this high-noise setting, the ring topology performs in general better than the all-to-all topology. This is in contrast with the design choice of the available algorithms, e.g. [13], where only all-to-all communication is considered. We remark that this effect is due to the lack of diversity in the resampling stage, and expected from our statement **(S4)** in Section V-E.

Furthermore, from Fig. 7 (where only the ring topology is used) and Fig. 6(a) we see that even a small number of exchanged particles can make a significant difference compared to the no-communication choice ($t = 0$). Moreover, as expected from statement **(S2)** in Section V-E, we note that this difference is not so marked when passing from $t = 1$ to $t = 4$, meaning that the real improvement is in communicating itself and not in the number of exchanged particles.[3] This is in contrast with the design choice of some of the available methods, e.g. [12], [14], [18], where either no communication is chosen or 25% of the total particles are shared.

As a second variation on the experimental results, we increase the number of state dimensions augmenting[4] the number of joints J with the setting $N = 2048$, $m = 512$. This case illustrates the scalability of the algorithm, in terms of runtime, with respect to the state dimension. As we see from Fig. 8 the particle exchange step, as well as resampling and (global) estimate, requires a relatively limited runtime and they scale better than the sampling step. This was to be expected because, when the state dimension increases, the sampling is the most affected task.[5]

[3]Which is also good news for limiting the communication because, the lower the t, the less communication is required per local filter, for a given N_i.

[4]We remark that this augmentation of the state dimension is done by increasing the number of angles to consider in the state equation (13a), and by changing the rigid body transformation φ (14), to accommodate the modified geometry of the arm.

[5]As seen in Section V-D, the other kernels are pseudo-random-number generation, sorting, and local and global estimation.

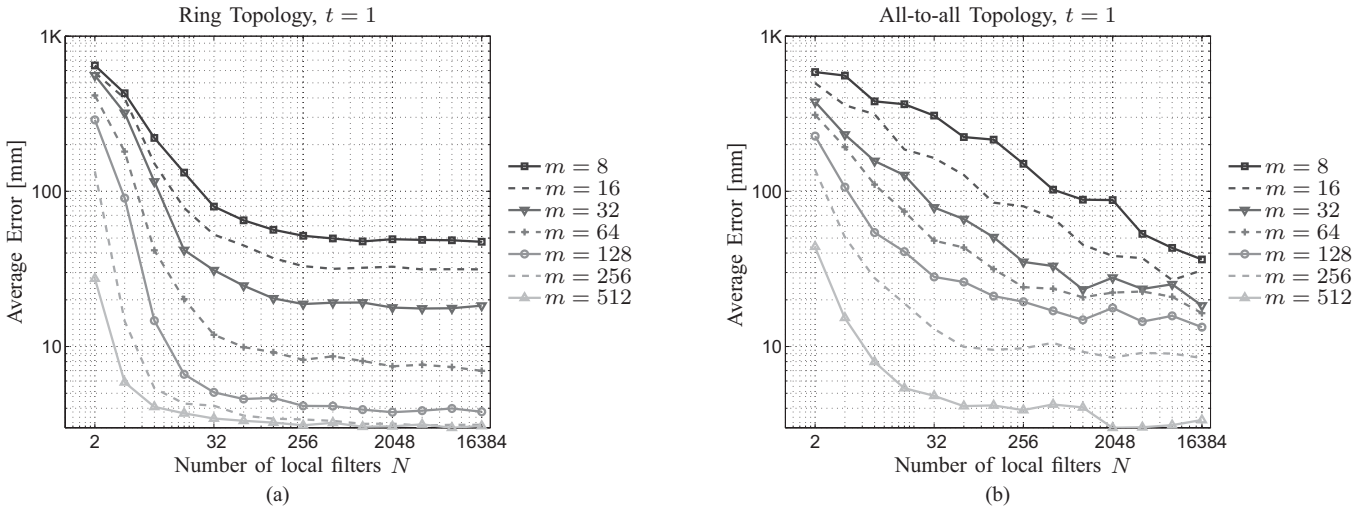

Fig. 6. Average error (**e** averaged over the 100 simulation runs) for different (N, m) settings varying the communication topology $\mathcal{G}$. (a) Setting: ring topology, $t = 1$. (b) Setting: all-to-all topology, $t = 1$.

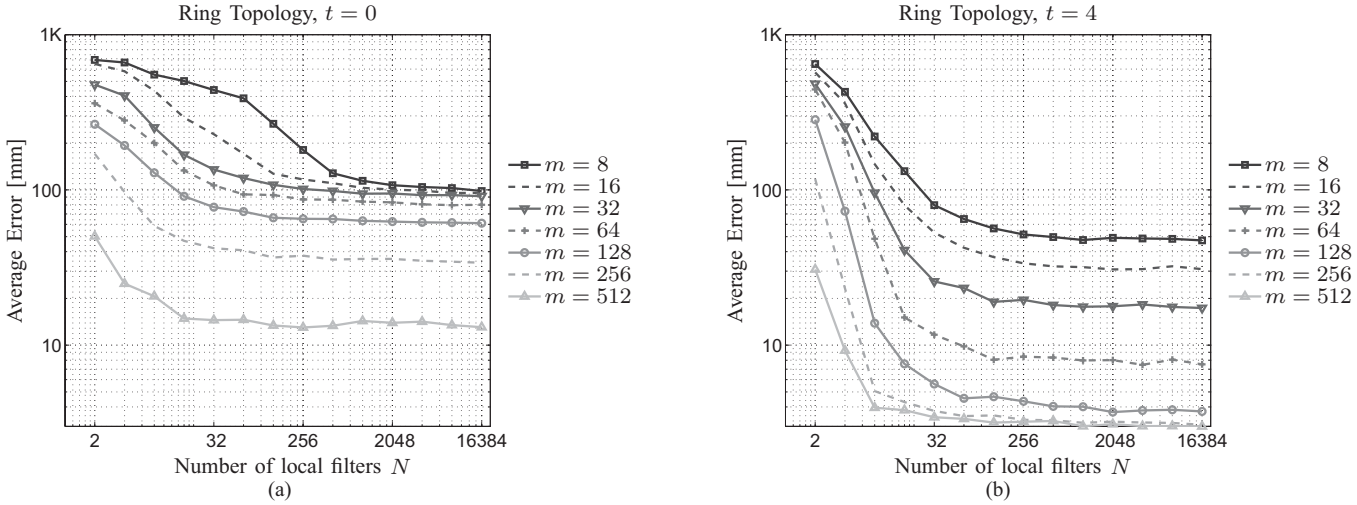

Fig. 7. Average error (**e** averaged over the 100 simulation runs) for different (N, m) settings varying the number of exchanged particles t. (a) Setting: ring topology, $t = 0$. (b) Setting: ring topology, $t = 4$.

D. Comparison With a Centralized Sequential Implementation

As a final set of simulation runs, we use the high-noise settings (Table II) and compare the proposed distributed particle filter implementation with a sequential centralized implementation (i.e., a standard particle filter that runs sequentially on a single-core architecture).

First of all, we consider in Fig. 9 the runtime [ms] for the distributed implementation and the centralized one (the centralized implementation has been implemented on a Intel Core i7-2820QM processor running at 2.3 GHz). We set $m = 512$ and we vary N. As we note from Fig. 9, the runtime of the centralized algorithm scales worse than the proposed distributed method (increasing N). Furthermore, for a high particle setting (Nm >16 K), the distributed algorithm is 10–>100 times faster than the sequential centralized implementation. The opposite, i.e., the distributed particle filter is slower than the centralized implementation, is instead reasonable with a low number of particles (Nm < 1K), since the single core is more powerful computation-wise than the local cores of the GPU architecture and there is no communication involved.

Fig. 9 can also be used for a comparison with the methods presented in the literature. With reference to Table I, we can report in Table III the performance of the proposed method.

As we see from Table III, the presented method outperforms in terms of the number of particles, state dimension, and/or runtime state-of-the-art methods employing GPU architectures. Furthermore, Algorithm 1 increases the achievable performances often by orders of magnitude.

In Fig. 10 we report the estimation error in both the distributed implementation and the centralized one. As we notice, the estimation error for cases in which $m \geq 128$ is comparable with the centralized setting. Furthermore, in the case of $m = 512$ and $N \geq 1024$, the distributed algorithm delivers better estimates than the centralized one. This has to

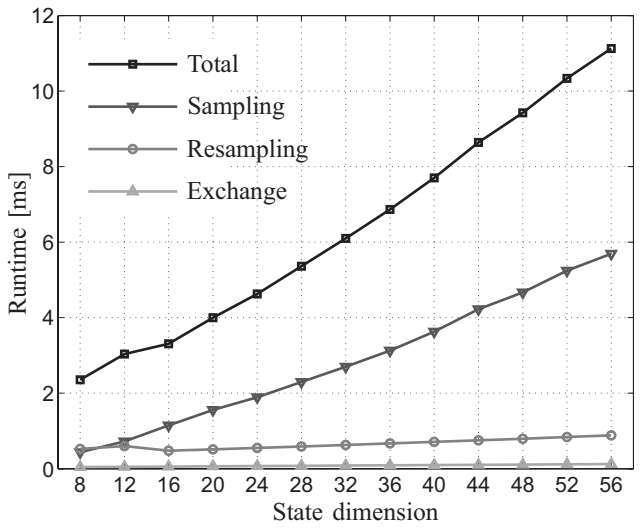

Fig. 8. Runtime of the some of the kernels of the distributed computation particle filter implementation varying the state dimension in the setting $N = 2048$ and $m = 512$.

TABLE III
PERFORMANCE OF THE PROPOSED APPROACH

Ref.	Sampling + Weight	Resampling	Estimation	Particles	State Dimension	Runtime [ms]
Algorithm 1	local	distributed	local	64 000	8	0.3
				1M	8	2.3
				4M	8	4.6

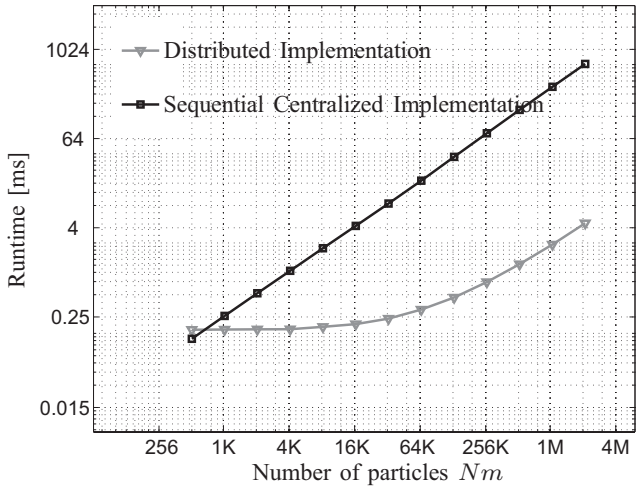

Fig. 9. Runtime comparison of the proposed distributed implementation and a sequential centralized implementation. In the distributed one, we fix $m = 512$ and we vary N.

do with the loss of diversity in the centralized implementation. Fig. 10 gives also extra insights on the selection of m and N given a total number of particles Nm. In fact:

(I1) for high values of Nm, it appears better, in terms of estimation error, to choose a high value for m. This configuration leads to a small number of accurate filters;
(I2) for low Nm settings, the opposite seems to be more recommendable. This leads to a high number of less accurate filters (but yet with more particle diversity).

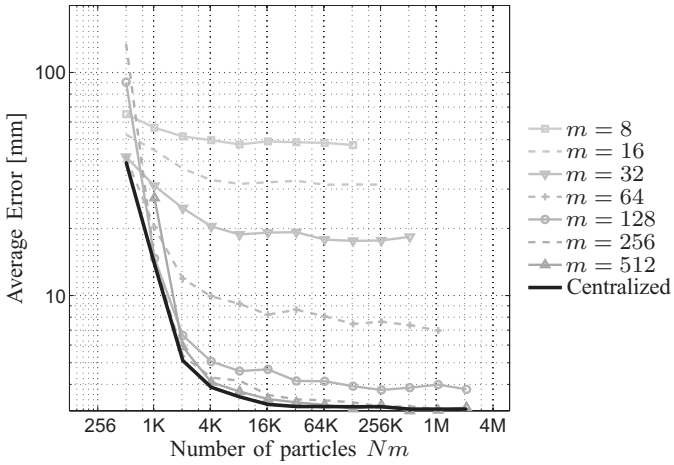

Fig. 10. Comparison of the estimation error between distributed and centralized sequential algorithms.

VII. CONCLUSION

A. Summary

In this paper, we showed that fast yet accurate nonlinear estimation is realizable and can be used in relatively high sample rate real-time feedback controllers. In particular, we designed, analyzed, and implemented a distributed computation particle filter that can handle over a million particles at 100 Hz with remarkable estimation accuracy. This was one of our goals as formulated in question **(Q)** in Section II, which has been therefore answered positively.

The results obtained were made possible by the use of a novel distributed resampling technique that is based on the exchange of particles via specified topologies that link local filters. This has several advantages, among which are **(P1)**, **(P2)**, and **(P3)**, as discussed in Section V-B. In particular, it is crucial in limiting the number of exchanged particles and still increasing the estimation accuracy (without the need for centralized data collection).

Furthermore, we were able to infer some theoretical properties of the proposed algorithm, which also demonstrated why our method performs better than available implementations. These properties, explicitly **(S1)**, **(S2)**, **(S3)**, and **(S4)** of Section V-E, indicate that the choice of exchange topology (e.g., the number of neighbors) and number of exchanged particles are critical for the accuracy of the filter and depend on the problem at hand. In general, a significant gain in accuracy has to be expected passing from no-exchange to exchange of particles among the local filters (even if the filters exchange just one particle), while for high process noise settings it is advisable not to choose topologies with a high number of neighbors, which will cause high levels of distortion.

Through simulations and experiments we showed that our proposed method outperforms other implementations that can be found in the literature. In particular, our implementation increased the number of particles, the state dimension, and/or the sampling frequency often by orders of magnitude with respect to state-of-the-art GPU solutions (typically based on parallel algorithms instead of our distributed ones). Moreover, we showed that the proposed scheme has compara-

ble accuracy with centralized sequential particle filters (with the same number of total particles), which require 10–100 times more computational time (when using a high number of particles) than our proposed distributed implementation.

Finally, we gained further insight on the selection of some user-tunable parameters, such as number of local filters N and number of particles in each of them m. In particular, we saw that the importance of choosing a higher m when the total number of particles, Nm, is high than when Nm is low, which has been formulated in **(I1)** and **(I2)** of Section VI-D.

B. Future Work

As future work directions, we plan to further analyze the proposed algorithm. We would like to use graph theoretical notions, such as algebraic connectivity and graph diameter, to propose metrics to decide whether a specific topology is better than another with respect to estimation accuracy, continuing the discussion of Section V-E. We have in mind to study the optimal selection of the number of local filters N and the number of local particles m to guarantee a certain level of accuracy and sampling rate, further investigating insights **(I1)** and **(I2)** of Section VI-D.

Finally, we envision the possibility of merging distributed sensing particle filters with distributed computation ones to obtain a fast and accurate nonlinear estimator that can also work well in distributed sensing scenarios.

REFERENCES

[1] N. Gordon, D. Salmond, and A. Smith, "Novel approach to nonlinear/non-Gaussian bayesian state estimation," *IEE-Proc.-F*, vol. 140, no. 2, pp. 107–113, Apr. 1993.

[2] S. J. Julier and J. K. Uhlmann, "Unscented filtering and nonlinear estimation," *Proc. IEEE*, vol. 93, no. 3, pp. 401–422, Mar. 2004.

[3] C. Rao, J. Rawlings, and D. Mayne, "Constrained state estimation for nonlinear discrete-time systems: Stability and moving horizon approximations," *IEEE Trans. Automat. Control*, vol. 48, no. 2, pp. 246–257, Feb. 2003.

[4] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-Gaussian bayesian tracking," *IEEE Trans. Signal Process.*, vol. 50, no. 2, pp. 174–188, Feb. 2002.

[5] J. Cortés, "Distributed algorithms for reaching consensus on general functions," *Automatica*, vol. 44, no. 3, pp. 726–737, 2008.

[6] M. Coates, "Distributed particle filters for sensor networks," in *Proc. 3rd Int. Symp. Inform. Process. Sensor Netw.*, Apr. 2004, pp. 99–107.

[7] X. Sheng, Y. Hu, and P. Ramanathan, "Distributed particle filter with gmm approximation for multiple targets localization and tracking in wireless sensor network," in *Proc. 4th Int. Symp. Inform. Process. Sensor Netw.*, Apr. 2005, pp. 181–188.

[8] D. Gu and H. Hu, "Target tracking by using particle filter in sensor networks," *Int. J. Robot. Automat.*, vol. 24, no. 3, pp. 171–182, 2009.

[9] S. Lee and M. West, "Markov chain distributed particle filters (MCDPF)," in *Proc. IEEE 48th Conf. Decis. Control*, Dec. 2009, pp. 5496–5501.

[10] S. Farahmand, S. I. Roumeliotis, and G. B. Giannakis, "Set-membership constrained particle filter: Distributed adaptation for sensor networks," *IEEE Trans. Signal Process.*, vol. 59, no. 9, pp. 4122–4138, Sep. 2011.

[11] A. Simonetto and T. Keviczky, "Distributed nonlinear estimation for diverse sensor devices," in *Distributed Decision Making and Control* (Lecture Notes in Control and Information Sciences). New York: Springer-Verlag, 2012, pp. 147–169.

[12] O. Brun, V. Teuliere, and J. Garcia, "Parallel particle filtering," *J. Parallel Distrib. Comput.*, vol. 62, no. 5, pp. 1186–1202, 2002.

[13] A. S. Bashi, V. P. Jilkov, X. R. Li, and H. Chen, "Distributed implementations of particle filters," in *Proc. IEEE Conf. Inform. Fusion*, Jul. 2003, pp. 1164–1171.

[14] M. Bolić, P. M. Djurić, and S. Hong, "Resampling algorithms and architectures for distributed particle filters," *IEEE Trans. Signal Process.*, vol. 53, no. 7, pp. 2442–2450, Jul. 2005.

[15] M. Bolić, A. Athalye, S. Hong, and P. Djurić, "Study of algorithmic and architectural characteristics of Gaussian particle filters," *J. Signal Process. Syst.*, vol. 61, no. 2, pp. 205–218, 2010.

[16] O. Rosén, A. Medvedev, and M. Ekman, "Speedup and Tracking Accuracy Evaluation of Parallel Particle Filter Algorithms Implemented on a Multicore Architecture," in *Proc. IEEE Int. Conf. Control Appl.*, Sep. 2010, pp. 440–445.

[17] G. Hendeby, R. Karlsson, and F. Gustafsson, "Particle filtering: The need for speed," *EURASIP J. Adv. Signal Process.*, vol. 1, no. 5, pp. 1–9, 2010.

[18] M.-A. Chao, C.-Y. Chu, C.-H. Chao, and A.-Y. Wu, "Efficient parallelized particle filter design on CUDA," in *Proc. IEEE Workshop Signal Process. Syst.*, Oct. 2010, pp. 299–304.

[19] K. Par and O. Tosun, "Parallelization of particle filter based localization and map matching algorithms on multicore/manycore architectures," in *Proc. IEEE Intell. Veh. Symp.*, Jun. 2011, pp. 820–826.

[20] J. Brown and D. Capson, "A framework for 3-D model-based visual tracking using a GPU-accelerated particle filter," *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 1, pp. 68–80, Jan. 2012.

[21] *CUDA C Programming Guide*. NVIDIA Corporation, Santa Clara, CA, USA, October 2010.

[22] C. Godsil and G. Royle, *Algebraic Graph Theory* (Graduate Text in Mathematics). New York: Springer-Verlag, 2001.

[23] A. Simonetto and T. Keviczky, "Recent developments in distributed particle filters: Towards fast and accurate algorithms," in *Proc. 1st IFAC Workshop Estimat. Control Netw. Syst.*, Sep. 2009, pp. 138–143.

[24] B. Balasingam, M. Bolić, P. M. Djuric, and J. Míguez, "Efficient distributed resampling for particle filters," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, May 2011, pp. 3772–3775.

[25] M. Chitchian, A. S. van Amesfoort, A. Simonetto, T. Keviczky, and H. J. Sips, "Particle filters on multi-core processors," Dept. Comput. Sci., Delft Univ. Technology, Delft, The Netherlands, Tech. Rep. PDS-2012-001, (Feb. 2012) [Online]. Available: http://www.pds.ewi.tudelft.nl/fileadmin/pds/reports/2012/PDS-2012-001.pdf. Code available at: https://github.com/alxames/esthera

[26] M. Saito and M. Matsumoto. (2010, Jun.). *A Variant of Mersenne Twister Suitable for Graphic Processors* [Online]. Available: http://arxiv.org/abs/1005.4973

[27] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. New York: Wiley, 1991.

[28] J. Míguez, "Analysis of parallelizable resampling algorithms for particle filtering," *Signal Process.*, vol. 87, no. 12, pp. 3155–3174, 2007.

[29] J.-Y. Bouguet. (2010). *Camera Calibration Toolbox for Matlab* [Online]. Available: http://www.vision.caltech.edu/bouguetj/calib_doc/

[30] S. Hutchinson, G. Hager, and P. Corke, "A tutorial on visual servo control," *IEEE Trans. Robot. Automat.*, vol. 12, no. 5, pp. 651–670, Oct. 1996.

[31] D. van der Lijn, G. Lopes, and R. Babuska, "Motion estimation based on predator/prey vision," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2010, pp. 3435–3440.

Mehdi Chitchian received the B.Sc. and M.Sc. degrees in computer science from the Delft University of Technology, Delft, The Netherlands, specializing in parallel and distributed systems.

He has been involved in multi-disciplinary research projects with the Delft Biorobotics Lab as well as the Delft Center for Systems and Control. His academic interests include research in the application of high-performance computing technologies in robotics and other control systems.

Andrea Simonetto received the M.Sc. degree in space engineering (cum laude) from both Politecnico di Milano and Politecnico di Torino, Torino, Italy, in 2008, and the Ph.D. degree in systems and control from the Delft University of Technology, Delft, The Netherlands, in 2012.

He is currently a Post-Doctoral Researcher with the Circuits and Systems Group, Electrical Engineering Department, Delft University of Technology. He was a Visiting Researcher with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, and with KTH, Royal Institute of Technology, Stockholm, Sweden. His current research interests include distributed estimation, control, and optimization with applications in sensor networks and mobile robotics.

Alexander S. van Amesfoort received the B.Sc. and M.Sc. degrees in computer science from VU University, Amsterdam, The Netherlands, with a specialization in parallel computing and visualization, and the Ph.D. degree in parallel computing from the Parallel and Distributed Systems Group, Delft University of Technology, Delft, The Netherlands.

He is currently finalizing his dissertation, while developing radio astronomy data processing software for the LOFAR telescope at ASTRON, The Netherlands Institute for Radio Astronomy, Dwingeloo, The Netherlands. He is particularly interested in cooperating with application experts to adapt their algorithms to modern parallel architectures. His current research interests include accelerator architectures, cluster and grid computing, compilers, and computer graphics.

Tamás Keviczky received the M.S. degree in electrical engineering from the Budapest University of Technology and Economics, Budapest, Hungary, in 2001, and the Ph.D. degree from the Control Science and Dynamical Systems Center, University of Minnesota, Minneapolis, in 2005.

He is currently an Assistant Professor with the Delft Center for Systems and Control, Delft University of Technology, Delft, The Netherlands. He has been a Research Intern with Honeywell Laboratories, Minneapolis, and a Post-Doctoral Scholar with the Control and Dynamical Systems, California Institute of Technology, Pasadena. His current research interests include distributed optimization and optimal control, model predictive control, and distributed control and estimation of large-scale systems with applications in aerospace, automotive, mobile robotics, industrial processes, and infrastructure systems.

Dr. Keviczky was a co-recipient of the AACC O. Hugo Schuck Best Paper Award for Practice in 2005.