10.2478/msr-2013-0047

The Parallel Bayesian Toolbox for High-performance Bayesian Filtering in Metrology

E. Garcia, T. Hausotte

Institute Manufacturing Metrology, University Erlangen-Nuremberg, 91052 Erlangen, Germany, garcia@fmt.fau.de

The Bayesian theorem is the most used instrument for stochastic inferencing in nonlinear dynamic systems and also the fundament of measurement uncertainty evaluation in the GUM. Many powerful algorithms have been derived and applied to numerous problems. The most widely used algorithms are the broad family of Kalman filters (KFs), the grid-based filters and the more recent particle filters (PFs). Over the last 15 years, especially PFs are increasingly the subject of researches and engineering applications such as dynamic coordinate measurements, estimating signals from noisy measurements and measurement uncertainty evaluation. This is rooted in their ability to handle arbitrary nonlinear and/or non-Gaussian systems as well as in their easy coding. They are sampling-based sequential Monte-Carlo methods, which generate a set of samples to compute an approximation of the Bayesian posterior probability density function. Thus, the PF faces the problem of high computational burden, since it converges to the true posterior when number of particles $N_P \rightarrow \infty$. In order to solve these computational problems a highly parallelized C++ library, called Parallel Bayesian Toolbox (PBT), for implementing Bayes filters (BFs) was developed and released as open-source software, for the first time.

In this paper the PBT is presented, analyzed and verified with respect to efficiency and performance applied to dynamic coordinate measurements of a photogrammetric coordinate measuring machine (CMM) and their online measurement uncertainty evaluation.

Keywords: Particle filter, Kalman filter, Bayesian estimation, parallel high-performance computing, CUDA, open source.

1. INTRODUCTION

TRADITIONAL coordinate measuring machines (CMM) were increasingly completed or superseded by mobile, dynamic coordinate measuring machines (DMM), such as laser tracker or photogrammetry systems (i.e. camera-based systems). They allow non-contact, in-situ and online coordinate measurements of complex moving objects over large measurement volumes along with excellent precision and high measurement rate directly in the shop-floor [1, 2].

Typical applications are dynamic motion measurements (traveled paths, positions, displacements, velocities, accelerations, jerks, orientations and vibrations), guidance and calibration of machines, robot trajectory measurements, deformation analysis as well as assembly and inspection of parts and structures.

Often, DMM are portable optical measurement systems, which perform online measurements, mostly in unstable environments. For these dynamic measurements, i.e. tracking the temporal evolution of an object's position, it is in general not possible to model, quantify or compensate significant error sources, such as optical aberrations (e.g. reflection errors, changing refractive index), imaging errors or dynamic errors (e.g. motion artifacts), during the measurement process. As a result the obtained measurements exhibit a lower signal-to-noise ratio and reduced measurement accuracy. Furthermore it is not possible to evaluate the measurement uncertainty according to the "Guide to the expression of uncertainty in measurement" (GUM) due to the model imperfection and dynamic characteristic of the measurement process.

A solution to improve the measurement accuracy and estimate the uncertainty of such dynamic measurements is given by the Bayesian inference. Based on a model of the process under investigation, it allows the online computation of the probability density functions (PDFs) of the output quantities and thus deriving the optimal estimate for them. In the first part of this paper the principals of measurement uncertainty evaluation according to the GUM and the fundamentals of Bayesian estimation theory will be briefly restated. The second part of the paper is devoted to introduce and demonstrate the Parallel Bayesian Toolbox (PBT) and its features in order to implement Bayesian estimation techniques in a straightforward and efficient way. In order to determine the required computation time for such post-processing, several benchmarks for synthetic computation tasks problems as well as dynamic coordinate measurement tasks will be presented.

2. BASIC PRINCIPLES OF MEASUREMENT UNCERTAINTY EVALUATION

A measurement result is generally expressed as a single measured quantity value and a measurement uncertainty [5, definition 2.9]. The de facto international standards to evaluate, calculate and express uncertainty in all kinds of measurements are the GUM and its extension the GS1 [3, 4]. These documents provide guidance on the uncertainty evaluation as a two-stage process: formulation and calculation. The formulation stage involves developing a measurement model relating output (measurand) y to input quantities $x_1,\ldots,x_N$, incorporating corrections and other effects as necessary and finally assigning probability distributions to the input quantities on the basis of available knowledge. The calculation stage consists of propagating the probability distributions for the input quantities through the measurement model $y=f(x_1,\ldots,x_N)$ to obtain the probability distribution for the output quantity. From this probability density function the expected value and the standard deviation of the output quantity as well as the coverage interval are calculated. In order to not go beyond the scope of this paper, it is referred to GUM and GS1 [3, 4] for further details of the propagation of distribution.

Finally, the measurement uncertainty is stated as uncertainty budget, which should include the measurement model, estimates, and measurement uncertainties associated with the quantities in the measurement model, covariances, type of applied probability density functions, degrees of freedom, type of evaluation of measurement uncertainty and the coverage interval. Here, the optimal solution would be an analytical calculation of the output PDF. But as this is not possible in general, especially for nonlinear dynamic processes, sequential Monte Carlo methods are utilized to achieve a Bayesian estimation of the output density, as described in the next section.

3. BAYESIAN ESTIMATION

A measurement process generates disperse (uncertain) observations of a true (but unknown) measurand. This is an erroneous transformation due to ubiquitous errors. Thus, a complete, exact or unique reconstruction of the true measurand is always impossible. But using Bayes' theorem it is possible to estimate the true measurand from observations, since the functional relation between both is preserved.

This estimation problem can be solved as a sequential probabilistic inference problem for nonlinear dynamic systems. The unknown measurand (e.g. coordinates) is modeled as a state of a dynamic system (e.g. moving object) that evolves over time and is observed with a particular measurement process. This allows the estimation of measurable states (e.g. position) as well as derived or not directly measurable states (e.g. velocities, acceleration and orientations) of arbitrary dynamic systems in a statistically sound way given uncertain, noisy or incomplete observations.

The nonlinear dynamic system is described by the general discrete-time stochastic state space model

$$\begin{aligned} \boldsymbol{x}_k &= \boldsymbol{f}_k(\boldsymbol{x}_{k-1}, \boldsymbol{v}_{k-1}), \\ \boldsymbol{y}_k &= \boldsymbol{h}_k(\boldsymbol{x}_k, \boldsymbol{w}_k), \end{aligned} \tag{1}$$

where $\boldsymbol{x}_k$ is the hidden system state vector evolving over time $k \in \mathbb{N}$ (discrete time index) according to the possibly nonlinear state transition function $\boldsymbol{f}_k$, the last state $\boldsymbol{x}_{k-1}$ and the process noise $\boldsymbol{v}_{k-1}$. The observation $\boldsymbol{y}_k$ related to the state vector via the nonlinear observation function $\boldsymbol{h}_k$ and the measurement noise $\boldsymbol{w}_k$, which is corrupting the measurement of the state.

This state space model corresponds to a first order hidden Markov model [6, 7] with an initial probability density, $p(\boldsymbol{x}_0)$ the state transition probability $p(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1})$ and the observation probability density $p(\boldsymbol{y}_k \mid \boldsymbol{x}_k)$. Both the state transition density and the observation likelihood are fully specified by $\boldsymbol{f}_k$ and the process noise probability $p(\boldsymbol{v}_k)$ and by $\boldsymbol{h}_k$ and the observation noise density $p(\boldsymbol{w}_k)$, respectively. The stochastic state-space model, together with the known statistics of the noise random variables as well as the prior distributions of the system states, defines a probabilistic model of how the systems evolves over time and how we inaccurately observe the evolution of this hidden state.

The objective is to estimate the hidden system states $\boldsymbol{x}_k$ in a recursive fashion (i.e. sequential update of previous estimate) as measurement $\boldsymbol{y}_k$ becomes available. This is the central issue of the sequential probabilistic inference theory, which is also referred to as online, sequential or iterative filtering. From a Bayesian perspective, the complete solution to this problem is given by the conditional posterior density $p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k})$ of the state $\boldsymbol{x}_k$, taking all observations $\boldsymbol{y}_{1:k} = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_k\}$ into account (Fig.1.).

The computation of the Bayesian posterior density $p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k})$ requires the incorporation of all measurements $\boldsymbol{y}_{1:k}$ in one step (batch processing). A method to recursively update the posterior density as new observations arrive is given by the recursive Bayesian estimation algorithm, which is faster and allows an online processing of data with lower storage costs and a quick adaption to changing data characteristics.

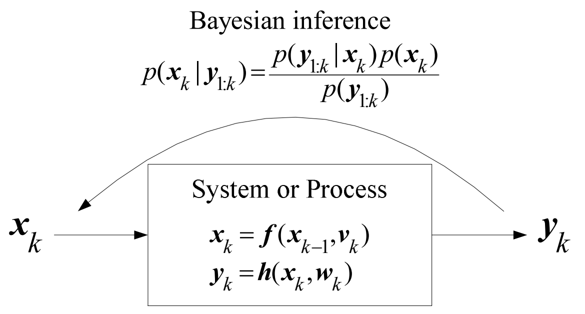
Bayesian inference
$p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k}) = \frac{p(\boldsymbol{y}_{1:k} \mid \boldsymbol{x}_k)p(\boldsymbol{x}_k)}{p(\boldsymbol{y}_{1:k})}$
System or Process
$\boldsymbol{x}_k = \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{v}_k)$
$\boldsymbol{y}_k = \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{w}_k)$
$\boldsymbol{x}_k$
$\boldsymbol{y}_k$

Fig.1. Recursive Bayesian estimation for dynamic systems.

Using the Bayes theorem and the discrete-time stochastic state space model (1), the posterior density can be derived and factored into the following recursive update form

$$p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k}) = \frac{p(\boldsymbol{y}_{1:k} \mid \boldsymbol{x}_k)p(\boldsymbol{x}_k)}{p(\boldsymbol{y}_{1:k})} = \frac{p(\boldsymbol{y}_k \mid \boldsymbol{x}_k)p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k-1})}{p(\boldsymbol{y}_k \mid \boldsymbol{y}_{1:k-1})}. \tag{2}$$

The computation of the Bayesian recursion essentially consists of two steps: the prediction step $p(\boldsymbol{x}_{k-1} \mid \boldsymbol{y}_{1:k-1}) \rightarrow p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k-1})$ and the update step $p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k-1}, \boldsymbol{y}_k) \rightarrow p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k})$. The prediction step involves using the process function and the previous posterior density $p(\boldsymbol{x}_{k-1} \mid \boldsymbol{y}_{1:k-1})$ to calculate the prior density of the state $\boldsymbol{x}_k$ at time k via the Chapman-Kolmogorov equation

$$p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k-1}) = \int p(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1})p(\boldsymbol{x}_{k-1} \mid \boldsymbol{y}_{1:k-1})d\boldsymbol{x}_{k-1}, \tag{3}$$

where the state transition PDF is given by[1]

$$p(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}) = \int \delta(\boldsymbol{x}_k - \boldsymbol{f}_k(\boldsymbol{x}_{k-1}, \boldsymbol{v}_k)) p(\boldsymbol{v}_k) d\boldsymbol{v}_k. \tag{4}$$

The prior of the state $\boldsymbol{x}_k$ at time k does not incorporate the latest measurement $\boldsymbol{y}_k$, i.e. the probability for $\boldsymbol{x}_k$ is only based on previous observations. When the new uncertain measurement $\boldsymbol{y}_k$ becomes available the update step is carried out and the posterior PDF of the current state $\boldsymbol{x}_k$ is computed from the predicted prior (3) and the new measurement via the Bayes theorem (2), where the observation likelihood PDF and the normalizing constant in the denominator are given by:

$$p(\boldsymbol{y}_k \mid \boldsymbol{x}_k) = \int \delta(\boldsymbol{y}_k - \boldsymbol{h}_k(\boldsymbol{x}_k, \boldsymbol{w}_k)) p(\boldsymbol{w}_k) d\boldsymbol{w}_k \tag{5}$$

$$p(\boldsymbol{y}_k \mid \boldsymbol{y}_{1:k-1}) = \int p(\boldsymbol{y}_k \mid \boldsymbol{x}_k) p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k-1}) d\boldsymbol{x}_k. \tag{6}$$

The equations (2-6) formulate the Bayesian solution to recursively estimate the unknown system states (measurands) of a nonlinear dynamic system from uncertain, noisy or incomplete measurements. But this is only a conceptual solution in the sense that in general the integrals cannot be determined analytically. Furthermore, the algorithmic implementation of this solution requires the storage of the entire (non-Gaussian) posterior PDF as an infinite dimensional matrix, because generally it cannot be completely described by a sufficient statistic of finite dimension. Only in some restricted cases a closed-form recursive solution is possible. For example with restriction to linear, Gaussian systems a closed-form recursive solution is given by the famous Kalman filter [8]. In most situations, either both the dynamic and the measurement process or only one of them is nonlinear. Thus the multi-dimensional integrals are not tractable and approximative solutions such as sequential Monte Carlo methods have to be used [8].

3.1. Sequential Monte Carlo methods

Sequential Monte Carlo (SMC) methods are stochastic sampling-based approaches, whereby the Monte Carlo integration with sequential importance sampling (SIS) is used to solve the high dimensional integrals of the Bayesian recursion. They make no explicit assumption about the form of the posterior density and approximate the Bayesian integrals with finite sums. These methods have the advantage of not being subject to the curse of dimensionality as well as not being constrained to linear or Gaussian models. The basic idea in SMC methods is to represent the Bayesian posterior density $p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k})$ by a set of random samples with associated weights w^i_k (also referred to as particles) and to recursively compute the posterior based on these weighted samples, that is

$$p(\mathbf{x}_k \mid \mathbf{y}_{1:k}) \approx \sum_{i=1}^{N_s} w^i_k \delta(\mathbf{x}_k - \mathbf{x}^i_k), \tag{7}$$

$$w^i_k = w^i_{k-1} \frac{p(\mathbf{y}_k \mid \mathbf{x}^i_k) p(\mathbf{x}^i_k \mid \mathbf{x}^i_{k-1})}{q(\mathbf{x}^i_k \mid \mathbf{x}^i_{k-1}, \mathbf{y}_k)}, \tag{8}$$

where $p(\mathbf{x}_k \mid \mathbf{y}_{1:k})$ is the true posterior, $q(\cdot)$ is the importance sampling density, $\mathbf{x}^i_k$ and w^i_k with $i = 1, \dots, N_P$ are the random samples and associated weights, whish normalize to $\sum_i w^i_k = 1$. In every time step the samples are drawn from the importance density, since it is not possible to sample from the posterior density directly. For this reason the choice of $q(\cdot)$ is a crucial design parameter in SIS methods. The objective is to draw samples in the region where the target state lies in (region of "importance") to achieve good state estimations and high computational efficiency. With an increasing number of particles $N_P \rightarrow \infty$ the Monte Carlo Approximation becomes an equivalent representation of the true posterior density and the SMC methods converges to the true Bayesian estimate.

An intrinsic problem with SIS is the circumstance, that after a few iterations, only some particles have significant weights, and the others are almost zero, this is called weight degeneracy or sample impoverishment problem. In order to improve the sample efficiency, usually some kind of resampling (selection) scheme is introduced, that avoids the problem of degenerate particles. The most common resampling strategies are multinomial, systematic, stratified or residual resampling [9]. The resampling duplicates the particles with high weights with several children, and assigns them equal weights. This eliminates particles with insignificant weights and chooses more particles in more probable regions. In general, the total number of particles is kept constant. This method of sampling is termed sampling importance resampling (SIR). The resulting algorithmic implementation consisting of predicting, updating and resampling the particles (weighted samples) is called particle filter. The PF is very general and very easy to code but faces the aforementioned problem of high computational burden, since it only converges to the true posterior when the number of particles goes to infinity. Thus, there is a direct dependency between the number of particles and the achievable estimation accuracy. The more particles were used the better is the approximation of the true Bayesian posterior density.

4. Parallel Bayesian Toolbox

The PF faces the problem of high computational burden, since it converges to the true posterior when number of particles $N_P \rightarrow \infty$. For typical low dimensional estimation problems the PF requires 2 to 6 orders of magnitude more computational throughput than the extended KF (EKF), to achieve the same accuracy [10].

In order to solve these computational problems and assist users in efficiently implementing Bayes filters (BFs), a highly parallelized C++ library, called Parallel Bayesian

[1] $\delta(\cdot)$ is the Dirac-delta function.

Toolbox (PBT) was developed and released as open-source software, for the first time [11]. The PBT is a very flexible, high-performance and platform independent[2] C++ programming library for implementing the most common Bayes filter - that is PFs, KFs and combinations of both - in an easily understandable, high level language, following the MATLAB/Octave programming language, for filtering, smoothing and predicting applications. This is achieved by using the Armadillo library [12] for easy coding, optimized linear algebra libraries[3] for numerical computations on central processing units (CPUs) and Nvidia's Compute Unified Device Architecture (CUDA) framework for performing computations on graphics processing units (GPUs). The source code of the PBT can be distributed and/or modified under the terms of the Berkeley Software Distribution (BSD) license or the GNU General Public License (GPL). This dual license allows the usage of the PBT in open source, but also in proprietary applications.

The PBT was designed and implemented that the following requirements are met.

- *Free and open source*: In order to analyze, modify and reuse software, its source code has to be available under an appropriate license.
- *Independent and flexible*: Software should be independent of hard- or software (i.e. independent of CPUs, GPUs, operating systems, software libraries etc.) and independent of any particular application (such as tracking, robotics, signal/image processing, econometrics etc.).
- *Performance*: For high computational throughput, all available CPUs and GPUs should be used optimally.
- *Modularity*: Software should consist of interchangeable, modular functional units in order to improve customization and reusability.
- *Simple to code and easy to understand*: Source code must be easy to understand, simple to read and easy to maintain.
- *Matlab/GNU Octave Interface*: Software should offer an interface to Matlab/GNU Octave, because they are de facto standards for technical computing, data analysis and algorithm development.

Although there is free and open source software available for implementing Kalman filters (see [18-19] and references herein) and/or particle filters (see [20-23] and references herein), none fulfils all requirements stated above[4]. Especially the last four criteria are not addressed adequately. In particular, no software exhibits an easy to use linear algebra library with bindings to optimized numerical libraries in order to transparently and optimally uses modern multi-core CPU/GPU architectures. Also a proper modularization to easily customize noise sources, resampling or state estimation methods and a Matlab/GNU Octave interface are missing. Implementing an extra MEX/OCT interface, which converts data types between Matlab/GNU Octave and the software library, is an additional tedious and error-prone work for practitioners and decreases the overall computational throughput.

The PBT itself uses CMake [13] as cross-platform build system and consists of five major modules. First, a BF module that provides all data management and interfaces. This module receives all measured data and distributes it to the other parts of the framework. The other modules are the state space model (SSM), describing the process and measurement equation, the resampling module implementing all common resampling strategies, the noise sources module providing sampling and evaluating functions for all distributions defined in [3, 4] and the state estimation module for computing a state estimate from the Bayesian posterior density in every time step. Additionally the toolbox features interfaces to the numerical computation systems MATLAB and GNU Octave as well as to the C++ development environment Automotive Data and Time Triggered Framework (ADTF). The overall architecture of the PBT is depicted in Fig.2. Example C++ code for implementing EKF and PF using PBT is given in Fig.3. and 4 respectively. This demonstrates the easy and straightforward usage of PBT. In the following sections, the main components and their subsystems will be explained in more detail.

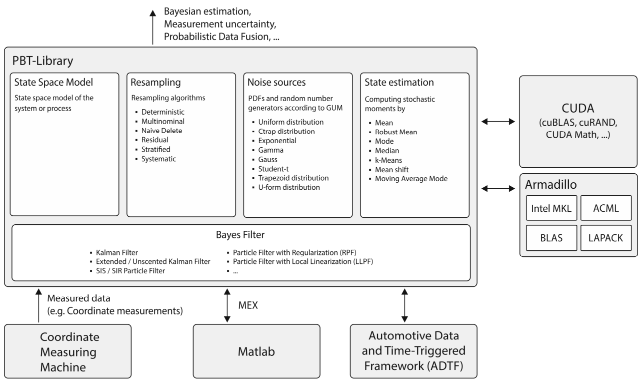

Fig.2. Architecture of the Parallel Bayesian Toolbox.

A. *Bayes Filter module*

This module has the role of a coordinator. It provides the filter algorithm and it organizes the data transfer between all other modules, as shown in Fig.3. The Bayes filter module calls the process and measurement function of the SSM in the prediction and update step, respectively. It is also used for the computation of the particle weights.

B. *State space model*

This module contains the dynamic model of the processor system under investigation and defines its process function ffun() and measurement function hfun(). The PBT offers predefined models for simple dynamics like steady state, linear, circular or kinematic trajectories [14] to assist the user. Dynamic coordinate measurements of all standard geometrical elements can be described with these included models. More complex models can be implemented by the user. In the case of KFs the user has to provide system, covariance and Jacobian matrices of the process and measurement equations. In the case of PFs the user has to specify ffun()and hfun() with all noise sources.

[2] Currently supported are Microsoft Windows and Linux on 32-bit or 64-bit processors.

[3] Supported libraries are Basic Linear Algebra Subprograms (BLAS), Linear Algebra PACKage (LAPACK), Automatically Tuned Linear Algebra Software (ATLAS), AMD Core Math Library and Intel Math Kernel Library (MKL).

[4] A very good overview of KF software is given in [17]. Some libraries for PF are compared in chapter 7 of the design document of The Bayesian Filtering Library available under [20].

C. Resampling module

The resampling module implements multinomial, systematic, stratified and residual. The resampling step is only applied to PFs. It is the only bottleneck in the whole toolbox, because some calculations cannot be parallelized, such as computing the cumulative sum.

```
void BFilterEKF::predict()
{
    particles.samples = ...
        process->ffun(&particles.samples);

    // calculate covariance matrix using the ...
        Jacobian F and process noise
    P = process->F*P*process->F.t() + process->W;
}

void BFilterEKF::update(fvec measurement)
{
    fvec vk;
    fmat Sk;
    fmat K;
    fmat I = eye(particles.samples.n_rows, ...
        particles.samples.n_rows);

    // calculating Kalman gain
    Sk = process->H * P * process->H.t() + ...
        process->W;
    K = P * process->H.t() * Sk.i();

    // difference between real and predicted ...
        measurement
    vk = measurement - ...
        process->hfun(&particles.samples);

    // calculating updated mean and covariance
    particles.samples = particles.samples + K*vk;
    P = (I - K * process->H) * P;
}
```

Fig.3. C++ implementation of EKF using the PBT.

```
#include "pbt.h"

filter = BFilterSIR();
arma::fvec measurement;

ResamplingMultinomial *resampler = new ...
    ResamplingMultinomial;
EstimationMean *estimator = new EstimationMean;
ModelSinus *model = new ModelSinus;

filter->setResampling(resampler);
filter->setEstimation(estimator);
filter->setModel(model);

filter.initialize();
filter.setParticles(samples, weights);
filter.setThresholdByFactor(threshold);

while (true)
{
    filter.predict();

    // fetch next measurement

    filter.update(measurement); // includes ...
        resampling step
}
```

Fig.4. C++ implementation of PF using the PBT.

D. Noise sources module

This module is most useful in the case of PFs. It consists of random number generators and probability density functions for numerous distributions. It is intended to help the user to model arbitrary noise sources in the SSM. For modeling noise sources/uncertainties according to GUM and GS1 [3, 4], the PBT includes functions for all PDFs that a specified in the GUM.

E. State estimation module

In every time step, the state estimation model is used to compute a state estimate from the particle approximation of the Bayesian posterior density. The standard estimator in every filter configuration is the weighted mean estimation. Other available estimators are median, robust mean, k-means, mean-shift and best particle estimator. The latter are especially useful in applications with multi-modal likelihood functions, e.g. in localization.

5. Performance benchmarks

Three different performance benchmarks were performed, in order to test the computation throughput of linear algebra calculations in PBT. First, the multiplication of non-square matrices was analyzed, since this is the most often used mathematical operation in BFs. Furthermore, in literature is only the case of square matrices considered. On numerous systems the matrix-matrix multiplication was performed with the dimensions d = (3, 15)(100, 500, 2000, 10000). A representative realized runtime graph is shown in figure 5 (a) and (b). In most configurations CPU and GPU show relatively equal performance. This is because both are optimized for these operations. However, the NVIDIA Tesla GPU generation (or newer generations), specialized for scientific computations, outperforms all other hardware configurations. Regarding this, the matrix-matrix multiplication on GPUs is only useful in combination with new GPU generations and/or with other operations that are more computationally intensive, e.g. coordinates transformations presented next.

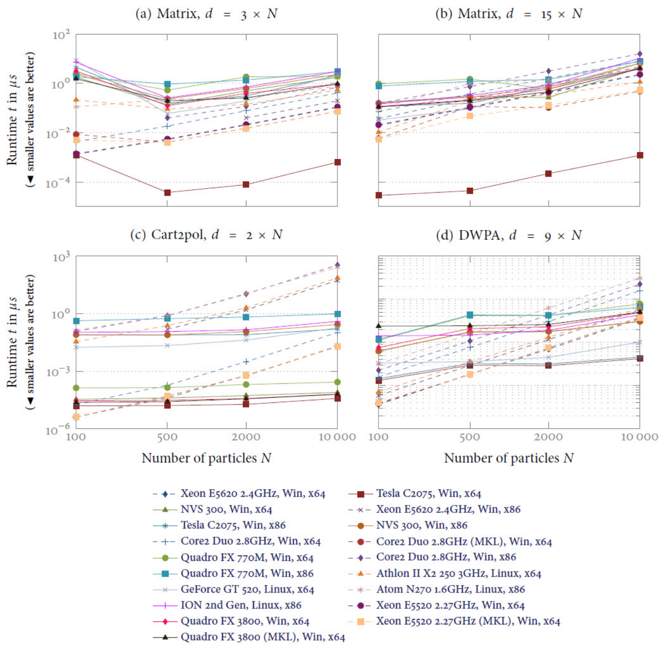

Fig.5. Performance comparison of (a)-(b) matrix-matrix multiplication of numerous CPUs, (c) the transformation of Cartesian to polar coordinates and (d) of Wiener process acceleration model (DWPA) of numerous CPUs (dashed lines) and GPUs (solid lines).

Second, the performance of the nonlinear transformation between Cartesian and polar coordinates was analyzed. This is one of the most often nonlinear transformations in filtering and localization applications. The results are given in Fig.4.(c). It can be seen that with increasing number of vector elements (random numbers) the computation time of CPUs gets higher, whereas the computation time of GPUs remains nearly constant. The reasons for this result are the special function units of GPUs for computing sine and cosine as well as the high parallelization of modern GPUs. While CPUs can compute 8 operations in parallel, unspecialized mainstream GPUs can compute 1536 operations at once (assuming 48 CUDA cores each with 32 threads). This complies with the results in [15].

Third, the achievable performance of filtering the motion of a point mass with SIR PF using the discrete Wiener process acceleration model (DWPA) as kinematic model [14] was examined. The benchmark results, for this very often used model, can be found in Fig.5.(d). For small matrices the runtime per one operation is relatively equal for GPU and CPU. But beginning with approximately 1000 particles the GPUs are faster than the CPU implementations. The lower rise of GPU runtime leads to the conclusion that GPUs should be used for PF configurations with more than 1000 particles, at least. All benchmark results revealed that using GPUs for computations can yield a significant performance improvement, especially for modern GPU architectures.

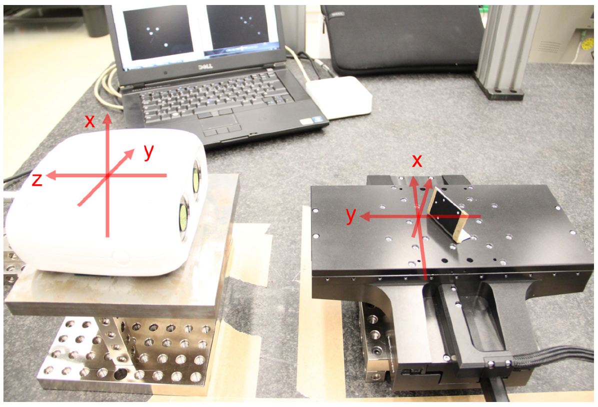

Fig.6. Measurement setup for dynamic coordinate measurements.

After these systhetic benchmarks the dynamic measurement task of tracking a freely moving target (marker) with a mobile, photogrammetric stereo-camera system is examined. The tracking target was moved by a high accuracy XY linear stage as depicted in Fig.6. along a given trajetory. 1000 data points were recorded and afterwards filtered with a SIR PF using the former DWPA model. This post-processing was performed offline. In order to analyze the runtime for such data processing for later online computations, a SIR PF was implemented with PBT, Bayes++ [23] and the Bayesian Filtering Library (BFL) [21]. These are the only available open source libraries that are under active development and feature interfaces and modules to implement particle filters in a comparable way to PBT. The benchmark results of a single filtering step for all three libraries are given in Fig.7. Shown are the averaged values over 10 runs. The following software was used for compiling the libraries: Microsoft Windows 7 (64-bit), Microsoft Visual C++ 2008 Compiler, Armadillo 3.2.2, Intel MKL 10.3 Update 9 (64-bit), CUDA Toolkit v4.2 and Boost 1.45. Because Bayes++ and BFL do not offer bindings for optimized linear algebra libraries[5], they are outperformed by PBT running on CPU as well as on GPU in case of using more than 500 particles. In the case of few particles the CPU-based computation outperforms GPU computing, due to memory data management overhead and memory transfer time between host and GPU device. But in the long run, GPU tremendously outperform CPU computing, because the initial small transfer time occurs only once.

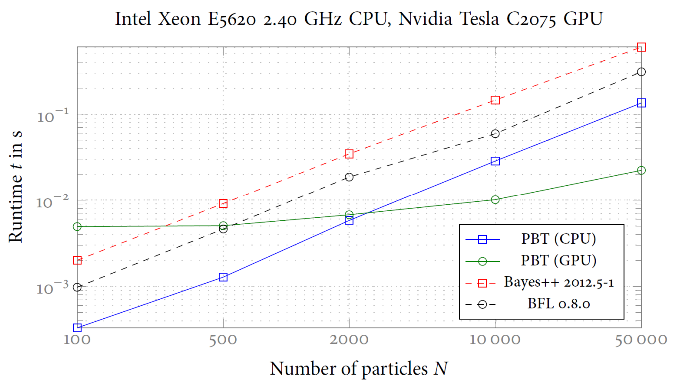

Fig.7. Comparison of runtime of PBT, Bayes++ and BFL.

As mentioned in the beginning of section 4, the PBT features interfaces to MATLAB and GNU Octave. The runtime performance of the PBT using the MATLAB MEX interface was further analyzed in a next experiment. Here, the same filter task was used to examine the performance improvement of PBT compared to a native MATLAB v7.14.0.739 (64-bit) implementation. The tests were run on Intel Xeon E5620 2.40 GHz CPU with Nvidia Tesla C2075 GPU. The benchmark results in Fig.8. show that PBT-MEX realizes throughout better computation times than the native MATLAB implementation. As in figure 7 it can be seen that in case of more than 2000 particles PBT-MEX on GPU is superior to PBT-MEX on CPU.

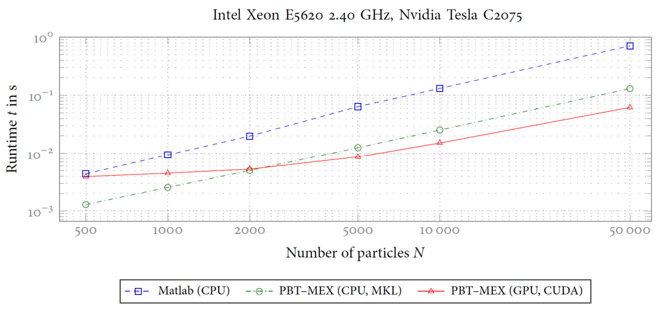

Fig.8. Comparison of runtime of MATLAB and PBT-MEX.

[5] Both use the Boost libraries for linear algebra and statistics, whereby BFL could also use LTI matrix library and Newmat library as external matrix libraries, see [21].

6. CONCLUSIONS

In this paper the open source Parallel Bayesian Toolbox (PBT) for implementing Bayes filter was presented. The toolbox is a very flexible, high-performance and platform independent C++ programming library for implementing KFs, PFs and combinations of both filter types. The presented benchmark results have shown that PBT is the fastest available open-source library for implementing Bayesian filtering, currently. Due to the high computational burden of such filters, the runtime performance is crucial for realizing an online processing of measured data e.g. dynamic coordinate measurements as in [16, 17].

For the first time, an open-source library is available that features a simple MATLAB-like high level language and efficiently utilize standard CPU/GPU architectures for implementing high-performance Bayesian filtering without the need of special computing hardware.

ACKLOWLEDGMENT

The research was supported by the German Research Foundation (Deutsche Forschungsgemeinschaft) under the DFG-Project Number WE 918/34-1.

REFERENCES

[1] Schwenke, H., Neuschaefer-Rube, U., Pfeifer, T., Kunzmann, H. (2002). Optical methods for dimensional metrology in production engineering. *CIRP Annals - Manufacturing Technology*, 51 (2), 685-699.

[2] Estler, W.T., Edmundson, K.L., Peggs, G.N., Parker, D.H. (2002). Large-scale metrology – an update. *CIRP Annals - Manufacturing Technology*, 51 (2), 587-609.

[3] BIPM, IEC, IFCC, ILAC, ISO, IUPAC, IUPAP and OIML. (2008). *Evaluation of measurement data - Guide to the expression of uncertainty in measurement (GUM 1995 with minor corrections)*. JCGM 100:2008.

[4] BIPM, IEC, IFCC, ILAC, ISO, IUPAC, IUPAP and OIML. (2008). *Evaluation of measurement data - Supplement 1 to the 'Guide to the expression of uncertainty in measurement' - Propagation of distributions using a Monte Carlo method*. JCGM 101:2008.

[5] BIPM, IEC, IFCC, ILAC, ISO, IUPAC, IUPAP and OIML. (2008). *International vocabulary of metrology — Basic and general concepts and associated terms (VIM)*. JCGM 200:2008.

[6] Cappé, O., Moulines, E., Ryden, T. (2005). *Inference in Hidden Markov Models*. Springer.

[7] Fraser, A.M. (2008). *Hidden Markov Models and Dynamical Systems* (1st ed.). Society for Industrial and Applied Mathematics.

[8] Doucet, A., de Freitas, N., Gordon, N. (2001). *Sequential Monte Carlo Methods in Practice*. Springer.

[9] Douc, R., Cappé, O., Moulines, E. (2005). Comparison of resampling schemes for particle filtering. In *Image and Signal Processing and Analysis : 4th International Symposium (ISPA 2005)*, 15-17 September 2005. IEEE, 64-69.

[10] Daum, F., Huang, J. (2003). Curse of dimensionality and particle filters. In *Aerospace Conference*, 8-15 March 2003. IEEE, Vol. 4, 4-1979-4-1993.

[11] Google Project Hosting. *Parallel Bayesian Toolbox*. http://code.google.com/p/parallel-bayesian-toolbox/.

[12] Sanderson, C. (2010). *Armadillo: An open source C++ linear algebra library for fast prototyping and computationally intensive experiments*. Technical Report. NICTA.

[13] *CMake home page*. http://www.cmake.org/.

[14] Bar-Shalom, Y., Li, X.-R., Kirubarajan, T. (2001). *Estimation with Applications to Tracking and Navigation : Theory Algorithms and Software*. Wiley – Blackwell.

[15] Rosenband, D.L., Rosenband, T. (2009). A design case study: CPU vs. GPGPU vs. FPGA. In *Formal Methods and Models for Co-Design : 7th IEEE/ACM International Conference (MEMOCODE'09)*, 13-15 July 2009. IEEE, 69-72.

[16] Garcia, E., Hausotte, T., Amthor, A. (2013). Bayes filter for dynamic coordinate measurements – Accuracy improvment, data fusion and measurement uncertainty evaluation. *Measurement*, 46 (9), 3737-3744.

[17] Garcia, E., Zschiegner, N., Hausotte, T. (2013). Parallel high-performance computing of Bayes estimation for signal processing and metrology. In *Computing, Management and Telecommunications (ComManTel)*, 21-24 January 2013. IEEE, 212-218.

[18] Welch, G., Bishop, G. *The Kalman Filter homepage*. http://www.cs.unc.edu/~welch/kalman.

[19] Identification and Decision Making Research Group, University of West Bohemia. *Nonlinear Estimation Framework homepage*. http://nft.kky.zcu.cz/nef.

[20] Cambridge University. *Sequential Monte Carlo methods (Particle filtering) homepage*. http://www-sigproc.eng.cam.ac.uk/smc/software.html.

[21] The Orocos Project. *The Bayesian Filtering Library*. http://www.orocos.org/bfl.

[22] BiiPS Project. *BiiPS (Bayesian inference with interacting Particle Systems) homepage*. http://alea.bordeaux.inria.fr/biips.

[23] Michael Stevens. *Bayes++. Open source Bayesian Filtering classes*. http://bayesclasses.sourceforge.net.

Received August 1, 2013.
Accepted December 12, 2013.