

Target Tracking Performance Evaluation — A General Software Environment for Filtering

Gustaf Hendeby, Rickard Karlsson

Division of Automatic Control

E-mail: hendeby@isy.liu.se, rickard@isy.liu.se

24th January 2007

Report no.: [LiTH-ISY-R-2767](#)

Accepted for publication in IEEE Aerospace Conference, Big Sky, Montana, 2007

Address:

Department of Electrical Engineering

Linköpings universitet

SE-581 83 Linköping, Sweden

WWW: <http://www.control.isy.liu.se>

AUTOMATIC CONTROL
REGLERTEKNIK
LINKÖPINGS UNIVERSITET



Abstract

In this paper, several recursive Bayesian filtering methods for target tracking are discussed. Performance for target tracking problems is usually measured using the second-order moment. For nonlinear or non-Gaussian applications, this measure is not always sufficient. The *Kullback divergence* is proposed as an alternative to mean square error analysis, and it is extensively used to compare estimated posterior distributions for various applications. The important issue of efficient software development, for nonlinear and non-Gaussian estimation, is also addressed. A new framework in C++ is detailed. Utilizing modern design techniques an object oriented filtering and simulation framework is provided to allow for easy and efficient comparisons of different estimators. The software environment is extensively used in several applications and examples.

Keywords: tracking, filtering, particle filter

1. INTRODUCTION

Filtering and Performance Evaluation

Performance for target tracking problems is usually measured using the second-order moment, for instance using the *mean square error* (MSE). For non-Gaussian PDFs, for instance due to nonlinear applications, this measure is not always sufficient. Many classical estimation methods rely on linearization to handle nonlinear models, or consider just the first- and second-order moments of the underlying PDF. The Gaussian noise assumption is common, too. For instance, the *extended Kalman filter* (EKF), [1], [2], [3] has difficulties when higher order moments, or the full PDF is needed. *Multiple model filters* (MMF) [2][4] and more generally the *particle filter* (PF), [5], [6], [7], are methods that can be used to overcome this problem.

In the paper, the *Kullback divergence* (KD) is extensively used to compare estimated posterior distributions for various target tracking applications. The *Cramér-Rao lower bound* (CRLB) and the *root mean square error* (RMSE) are also used and compared to the KD. The motivation to use the KD, instead of the more common RMSE, is that in many applications the full PDF is needed, for instance in hypothesis calculations, probability calculations, for instance percentiles *etc.*, the performance of a good estimate of the PDF is essential. The KD provides a general method to compare the performance, without specifying the exact application. When the RMSE is used to evaluate performance in Monte Carlo simulations, the true trajectory is compared to the estimated ones, and the difference is used to calculate the RMSE. For the KD the true PDF is compared to the estimated one. For low dimensions the true PDF can be obtained by gridding the state space and the filtering problem can be solved numerically. In practice, for higher dimensions, the PF will be used as an estimate of the true PDF, and other sub-optimal estimators can be evaluated against it.

Several estimation techniques are compared, and methods with ability to express non-Gaussian posterior distributions are shown to give superior performance over classical second-order moment based methods. For instance in evaluating target presence or in hypothesis testing and detection applications, based on [8] and further extensions.

A General Software Environment for Filtering

The important issue of efficient software development, for nonlinear and non-Gaussian estimation, is also addressed. A new framework in C++, [9], is detailed. Utilizing modern design techniques an object oriented filtering and simulation framework is provided to allow for easy and efficient comparisons of different estimators and algorithms.

Motivating Applications

The C++ software environment developed for general filtering, have been inspired by several different implementations, both in MATLAB and C/C++ over the last years. Here, some of these applications will briefly be described in order to motivate the vast variety of different applications which can benefit from a common and efficient software development environment.

- *Surface/underwater map-aided positioning*: Combining information from a digital sea chart with measured radar returns is used as a *global positioning system* (GPS) free method for positioning of ships. Depth information from a database is used together with sonar depth measurements to position an underwater vessel, [10].
- *Positioning of an industrial robot*: A sensor fusion application, where motor angle and angular velocity measurements are fused with accelerometer information, utilizes modern filtering techniques to improve the positioning of the manipulator, [11].
- *Positioning of an unmanned autonomous vehicle* (UAV): Positioning and control of a UAV, using GPS information and *inertial measurement unit* (IMU) data, [12].
- *Automotive target tracking*: A collision mitigation system based on late braking where measurements are available from a forward looking radar was developed and tested in hardware, [13].
- *Bearings-only target tracking*: When passive sensors are used, only direction to the unknown target can be measured. However, by appropriate maneuvering, or by using multiple-sensors, the range and range rate can be estimated [14], [15].
- *Track before detect*: A track before detect system for extended target estimation was developed in [16].

Common for the applications is that they often (but not always) rely on a simple discrete-time dynamic system, which often is linear. The measurement relation is a nonlinear function, sometimes only given by a database. In Monte Carlo simulation studies or in field tests various estimators are compared. Often real-time performance is needed for the final application to be successful. Hence, a unifying software development environment where only the basic models are needed from the user, and that will handle the filtering, data logging, performance analyzing part, *etc.* is highly motivated.

Organization

The paper is divided in three parts; first the filtering and performance part, and second the software environment description. In Section 2 recursive Bayesian estimation is described, mainly utilizing the particle filter, but also sub-optimal methods are presented. In Section 3 basic statistic properties such as the Kullback information and the Cramér-Rao lower bound are defined. In Section 4

the software development kit for general nonlinear filtering in C++ is presented, with source code examples. In the third part, Section 5 provides several simulation studies that connects the first part of the paper with the second. Finally, in Section 6 concluding remarks are given.

2. RECURSIVE BAYESIAN ESTIMATION

Consider the discrete state-space model

$$x_{t+1} = f(x_t, u_t, w_t), \quad (1a)$$

$$y_t = h(x_t, e_t), \quad (1b)$$

with state variables $x_t \in \mathbb{R}^n$, input signal u_t and measurements $\mathbb{Y}_t = \{y_i\}_{i=1}^t$, with known PDFs for the process noise, $p_w(w)$, and measurement noise, $p_e(e)$. The nonlinear prediction density $p(x_{t+1}|\mathbb{Y}_t)$ and filtering density $p(x_t|\mathbb{Y}_t)$ for the Bayesian inference, [1], are given by

$$p(x_{t+1}|\mathbb{Y}_t) = \int_{\mathbb{R}^n} p(x_{t+1}|x_t)p(x_t|\mathbb{Y}_t) dx_t, \quad (2a)$$

$$p(x_t|\mathbb{Y}_t) = \frac{p(y_t|x_t)p(x_t|\mathbb{Y}_{t-1})}{p(y_t|\mathbb{Y}_{t-1})}. \quad (2b)$$

For the important special case of linear-Gaussian dynamics and linear-Gaussian observations the *Kalman filter* (KF), [17], gives a finite dimensional recursive solution. For nonlinear and/or non-Gaussian systems, the PDF cannot in general be expressed with a finite number of parameters. Instead approximate methods must be used. Usually this is done in one of two ways; either by approximating the system or by approximating the posterior PDF. See for instance, [18], [19].

The Extended Kalman Filter

For the special case of linear dynamics, linear measurements and additive Gaussian noise, the Bayesian recursions have a finite dimensional recursive solution, given by the KF. For many nonlinear problems the noise assumptions and the nonlinearity are such that a linearized solution will be a good approximation. This is the idea behind the *extended Kalman filter* (EKF), [2], [3], [20], where the model is linearized around the previous estimate. Here the time update and measurement update for the EKF are presented,

$$\begin{cases} \hat{x}_{t+1|t} = f(\hat{x}_{t|t}, u_t, \mathbf{E} w_t), \\ P_{t+1|t} = F_t P_{t|t} F_t^T + G_t Q_t G_t^T, \end{cases} \quad (3a)$$

$$\begin{cases} \hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t (y_t - h(\hat{x}_{t|t-1}, \mathbf{E} e_t)), \\ P_{t|t} = P_{t|t-1} - K_t H_t P_{t|t-1}, \\ K_t = P_{t|t-1} H_t^T (H_t P_{t|t-1} H_t^T + R_t)^{-1}, \end{cases} \quad (3b)$$

where the linearized matrices are given as

$$F_t = \nabla_x f|_{x_t=\hat{x}_{t|t}}, G_t = \nabla_w f|_{x_t=\hat{x}_{t|t}}, H_t = \nabla_x h|_{x_t=\hat{x}_{t|t-1}},$$

and noise covariances are given by

$$Q_t = \text{cov } w_t \text{ and } R_t = \text{cov } e_t.$$

Grid-Based Approximation

Another way to approximately solve (2) is with a grid approximation of the integral. The grid-based approximation of a general integral in \mathbb{R}^n is

$$\int_{\mathbb{R}^n} g(x_t) dx_t \approx \sum_{i=1}^N g(x_t^{(i)}) \Delta^n, \quad (4)$$

using a regular grid, where Δ^n represents the volume and where $\{x_t^{(i)}\}_{i=1}^N$ represents the value in a specific grid. The approximation error depends on the gridding, where Δ is the length of a side in the grid. In [21], the Bayesian approach is investigated for a discrete-time nonlinear system using this approximation. The Bayesian time update and measurement update are solved using an approximate numerical method, where the density functions are piecewise constant on regular regions in the state space. Applying this grid-based integration to the general Bayesian estimation problem, using the model with additive noise yields the following approximation

$$p(x_{t+1}^{(i)}|\mathbb{Y}_t) = \sum_{j=1}^N p_{w_t}(x_{t+1}^{(i)}|x_t^{(j)})p(x_t^{(j)}|\mathbb{Y}_t)\Delta^n, \quad (5a)$$

$$p(x_t^{(i)}|\mathbb{Y}_t) = \gamma_t^{-1} p_{e_t}(y_t|x_t^{(i)})p(x_t^{(i)}|\mathbb{Y}_{t-1}), \quad (5b)$$

where γ_t is a normalization factor used to make sure the total probability adds up to one. The *minimum variance estimate* (MVE) is approximated with

$$\hat{x}_{t|t} = \mathbf{E} x_t|\mathbb{Y}_t \approx \sum_{i=1}^N x_t^{(i)} p(x_t^{(i)}|\mathbb{Y}_t)\Delta^n. \quad (6)$$

In [22], the nonlinear and non-Gaussian problem is analyzed using grid-based integration as well as Monte Carlo integration, described next, for prediction and smoothing. Several nonlinear systems are compared using different techniques. Also [23], discusses the grid-based or *point-mass filter* (PMF) approach.

The Particle Filter

This section presents the *particle filter* (PF) theory according to [23], [6], [5], [7]. The PF achieves an approximate solution to the discrete time Bayesian estimation problem formulated in (2) by updating an approximate description of the posterior filtering density. The PF approximates the density $p(x_t|\mathbb{Y}_t)$ by a large set of N samples (particles), $\{x_t^{(i)}\}_{i=1}^N$, where each particle has an assigned relative weight, $\gamma_t^{(i)}$, chosen such that all weights sum to unity. The location and weight of each particle reflect the value of the density in the region of the state

space. The PF updates the particle location and the corresponding weights recursively with each new observed measurement. For the common special case of additive measurement noise the un-normalized weights are given by

$$\gamma_t^{(i)} = \gamma_{t-1}^{(i)} p_e(y_t - h(x_t^{(i)})), \quad i = 1, \dots, N. \quad (7)$$

The most straight forward way to propagate the particles in time is to just apply the time update function with independent process noise, but it is also possible to use other proposal densities than the one implicitly used in this simple case. Hence, proposing particles in the update step from

$$x_{t+1}^{(i)} \sim q(x_{t+1}^{(i)} | x_t^{(i)}), \quad i = 1, \dots, N. \quad (8)$$

Hence, the weights are calculated recursively as

$$\gamma_t^{(i)} = \gamma_{t-1}^{(i)} \frac{p_e(y_t - h(x_t^{(i)})) p(x_{t+1}^{(i)} | x_t^{(i)})}{q(x_{t+1}^{(i)} | x_t^{(i)})}, \quad i = 1, \dots, N, \quad (9)$$

where it is assumed that the weights sum to unity.

Using the samples (particles) and the corresponding weights the Bayesian equations can be approximately solved. To avoid divergence a resampling step is introduced, [5], which is referred to as *sampling importance resampling* (SIR). The PF method is given in Algorithm 1.

Algorithm 1 Generic particle filter (PF)

- 1: Generate N samples $\{x_0^{(i)}\}_{i=1}^N$ from $p(x_0)$.
 - 2: Compute $\gamma_t^{(i)} = \gamma_{t-1}^{(i)} p_e(y_t - h(x_t^{(i)}))$ and normalize, i.e., $\bar{\gamma}_t^{(i)} = \gamma_t^{(i)} / \sum_{j=1}^N \gamma_t^{(j)}$, $i = 1, \dots, N$.
 - 3: Generate a new set $\{x_t^{(i^*)}\}_{i=1}^N$ by resampling with replacement N times from $\{x_t^{(i)}\}_{i=1}^N$, with probability $\Pr\{x_t^{(i^*)} = x_t^{(i)}\} = \bar{\gamma}_t^{(i)}$. Let $\gamma_t^{(i)} = 1/N$.
 - 4: $x_{t+1}^{(i)} = f(x_t^{(i^*)}, u_t, w_t^{(i)})$, $i = 1, \dots, N$ using different noise realizations, $w_t^{(i)}$.
 - 5: Increase t and continue from step 2.
-

The estimate for each time, t , is often chosen as the MVE, i.e.,

$$\hat{x}_t = \mathbb{E} x_t = \int_{\mathbb{R}^n} x_t p(x_t | \mathbb{Y}_t) dx_t \approx \sum_{i=1}^N \gamma_t^{(i)} x_t^{(i)}. \quad (10)$$

The PF approximates the posterior PDF, $p(x_t | \mathbb{Y}_t)$, by a finite number of particles. There exist theoretical limits [6], that show that the approximated PDF converges to the true one as the number of particles tends to infinity.

There also exist several extensions and modifications to the basic PF algorithm, for instance utilizing structure in the problem studied. The *Rao-Blackwellized* or *marginalized particle filter* (MPF), [24], [25], [26], exploits linear Gaussian substructures, where the optimal solution is a clever combination of the PF and the KF. The *auxiliary particle filter* (APF), uses a proposal density based on the next measurement in order to more efficiently handle for instance heavy-tailed distributions, [27]. The *cost-reference particle filter* (CRPF), [28], uses a user-defined cost function instead of a probability assumption on the dynamic model to evaluate the likelihood. For maneuvering targets, it might be efficient to have a different rate in the state update compared to the measurements, this is described in the *variable rate particle filter* (VRPF), [29]. Another way to try to reduce the number of particles needed is to assigning a Gaussian distribution around each particle as in the *Gaussian sum particle filter* (GSPF), [30].

Multiple Model Filtering

By combining *multiple models* (MM) in a filter it is possible to obtain a better approximation of the underlying PDF than with a single linear Gaussian model. The general multiple model idea is often based on the *Gaussian sum* (GS) approximation, described in [4], [2]. The GS method approximates the PDF with a sum of Gaussian densities,

$$p(x_t | \mathbb{Y}_t) = \sum_{i=1}^N \gamma_t^{(i)} \mathcal{N}(\hat{x}_t^{(i)}, P_t^{(i)}), \quad (11)$$

where $x_t^{(i)}$ and $P_t^{(i)}$ represent mean and covariance of one hypothesis, and $\gamma_t^{(i)}$ represents the trust in the hypothesis.

The different hypotheses are then handled separately, usually using a KF to update $x_t^{(i)}$ and $P_t^{(i)}$. The probabilities of the hypotheses $\gamma_t^{(i)}$, that always must add to unity, is updated based both on the probability of the system switching between the different modes (a model feature) and the likelihood of the obtained measurements. Consider hypotheses \mathcal{H}_j and \mathcal{H}_i , then

$$\gamma_{t+1}^{(i)} \propto p(y_t | x_{t+1}^{(i)}) \sum_j \gamma_t^{(j)} \cdot \Pr(\text{go to } \mathcal{H}_j \text{ from } \mathcal{H}_i). \quad (12)$$

The MVE of the state can then be obtained as

$$\hat{x}_t = \sum_{i=1}^N \gamma_t^{(i)} \hat{x}_t^{(i)} \quad (13a)$$

$$P_t = \sum_{i=1}^N \gamma_t^{(i)} (P_t^{(i)} + (\hat{x}_t^{(i)} - \hat{x}_t)(\hat{x}_t^{(i)} - \hat{x}_t)^T). \quad (13b)$$

That is, the estimate is taken as a weighted average of the estimates under the different hypotheses.

In order to avoid exponential growth of hypotheses, pruning or merging techniques must be applied. Two common merging methods are the *generalized pseudo Bayesian* (GPB) method [31], [20] and the *interacting multiple model* (IMM) method, [32], [31], [33]. Both methods are filter algorithms for linear discrete-time filters with Markovian switching coefficients, and they differ only in when merging is performed.

3. STATISTICAL PROPERTIES

Cramér-Rao Lower Bound (CRLB)

The *Cramér-Rao lower bound* (CRLB), [34], [35], [36], offers a fundamental performance bound for unbiased estimators. For instance, the CRLB can be used for feasibility tests or to measure filter efficiency in combination with RMSE.

The CRLB along a given state trajectory can in principle be found as

$$\text{cov}(x_t - \hat{x}_{t|t}) \succeq P_{t|t}, \quad (14)$$

where $P_{t|t}$ is the CRLB, given by the EKF, around the true state, x_t . Here, $A \succeq B$ denotes that $A - B$ is positive semidefinite. More information about the CRLB and its extensions to dynamic systems can be found in [36], [23], [6].

Kullback Divergence (KD)

The *Kullback-Leibler information* (KLI) [37], [38] quantifies the difference between two distributions. The KLI is not symmetric in its arguments, and hence not a measure. If symmetry is needed the *Kullback divergence* (KD), constructed as a symmetric sum of two KLI [38], [39], can be used as an alternative.

The KLI is defined, for two proper PDFs p and q , as the expected value of the log likelihood ratio between p and q :

$$\mathcal{I}^{\text{KL}}(p, q) = \mathbb{E}_p \log \frac{p(x)}{q(x)} = \int p(x) \log \frac{p(x)}{q(x)} dx, \quad (15a)$$

when $p(x) \neq 0 \Rightarrow q(x) \neq 0$, otherwise $\mathcal{I}^{\text{KL}}(p, q) = +\infty$. That is, if x is obtainable from p but not q the existence of this x makes it possible to with certainty determine which distribution is true, and therefore the infinite KLI value. It can be shown that $\mathcal{I}^{\text{KL}}(p, q) \geq 0$ for all proper PDFs p and q and that $\mathcal{I}^{\text{KL}}(p, q) = 0 \Leftrightarrow p = q$. A small $\mathcal{I}^{\text{KL}}(p, q)$ indicates that the distribution p is similar to q .

The KLI is additive in independent variables, [38]. That is, for $p(x_1, x_2) = p_1(x_1)p_2(x_2)$ and $q(x_1, x_2) = q_1(x_1)q_2(x_2)$,

$$\mathcal{I}^{\text{KL}}(p, q) = \mathcal{I}^{\text{KL}}(p_1, q_1) + \mathcal{I}^{\text{KL}}(p_2, q_2).$$

New independent observations just add to total information available.

The symmetric KD is defined as

$$\mathcal{J}^{\text{K}}(p, q) = \mathcal{I}^{\text{KL}}(p, q) + \mathcal{I}^{\text{KL}}(q, p), \quad (15b)$$

and the KLI properties above hold.

The KLI is closely related to other statistical measures, *e.g.*, *Shannon's information* and *Akaike's information criterion* [39]. A connection to *Fisher information* can also be found [38], and KLI can be used to derive the EM-algorithm [40]. In [41], the use of information bounds for dynamic systems is discussed in quite general terms, and the KLI is shown to be a special case of the more general *Rényi's G-divergence*.

The KD evaluates any estimated PDF against, for instance, the true posterior PDF. The truth can in simulations be provided by a grid-based approach, see Section 2. This way a measure of the quality of an estimator can be obtained. Since the true PDF is not available in most applications it will in practice be used to compare suboptimal estimators against the PF. If the difference is small, the estimated PDFs are hard to tell apart.

Examples

In this section the previously described theory is applied to several examples.

Comparing Gaussian distributions—Assume $p_i = \mathcal{N}(\mu_i, \Sigma_i)$ (μ_i and Σ_i scalar for simplicity), then the KLI is:

$$\mathcal{I}^{\text{KL}}(p_1, p_2) = \mathbb{E}_{p_1} \log \frac{p_1}{p_2} = \frac{1}{2} \log \frac{\Sigma_2}{\Sigma_1} - \frac{1}{2} + \frac{\Sigma_1 + (\mu_1 - \mu_2)^2}{2\Sigma_2}. \quad (16)$$

For a difference in mean only ($\Sigma_1 = \Sigma_2 =: \Sigma$) this yields

$$\mathcal{I}^{\text{KL}}(p_1, p_2) = 0 - \frac{1}{2} + \frac{\Sigma + (\mu_1 - \mu_2)^2}{2\Sigma} = \frac{(\mu_1 - \mu_2)^2}{2\Sigma}, \quad (17)$$

where the only contributing component is the normalized squared difference in mean. For equal mean, $\mu_1 = \mu_2$, but different variance

$$\mathcal{I}^{\text{KL}}(p_1, p_2) = \frac{1}{2} \log \frac{\Sigma_2}{\Sigma_1} - \frac{1}{2} + \frac{\Sigma_1}{2\Sigma_2} = \frac{1}{2} \left(\frac{\Sigma_1}{\Sigma_2} - 1 - \log \frac{\Sigma_1}{\Sigma_2} \right). \quad (18)$$

Here, only the relative difference in variance, Σ_1/Σ_2 , is significant. \diamond

Generalized Gaussian Distribution—The *generalized Gaussian* PDF is given by

$$p(e; \nu, \sigma) = \frac{\nu \eta(\nu, \sigma)}{2\Gamma(\nu^{-1})} e^{-(\eta(\nu, \sigma)|x|)^\nu}, \quad (19)$$

with

$$\eta(\nu, \sigma) = \sigma^{-1} \sqrt{\Gamma(3\nu^{-1})/\Gamma(\nu^{-1})},$$

and Γ the Gamma function [42]. The parameter ν acts as a shape parameter, and σ^2 is the variance of the distribution. For $\nu \rightarrow 0^+$, the generalized Gaussian is a *Dirac distribution* and for $\nu \rightarrow +\infty$ a uniform distribution, both with variance σ^2 . This is illustrated in Figure 1. Two important special cases are; $\nu = 1$ and $\nu = 2$,

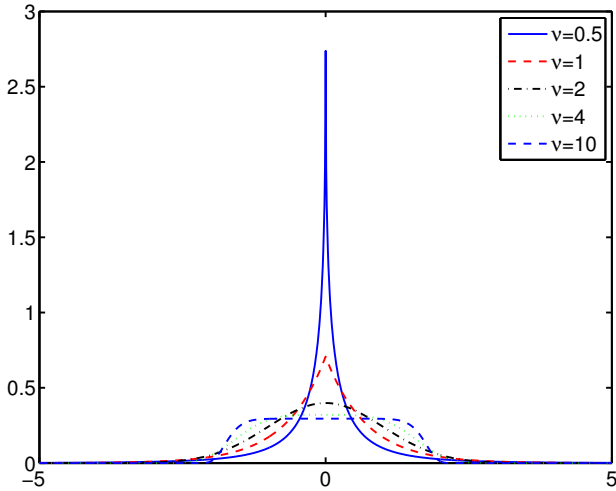


Figure 1. PDF of the generalized Gaussian distribution as a function of the shape parameter ν .

yielding the Laplacian and the Gaussian distribution, respectively.

The generalized Gaussian PDF provides the mean to study how small posterior deviations from Gaussianity affect performance. Figure 2 shows how much the generalized Gaussian distribution differs, in terms of KD (numerically evaluated), from the Gaussian distribution with the same variance as the shape parameter ν varies.

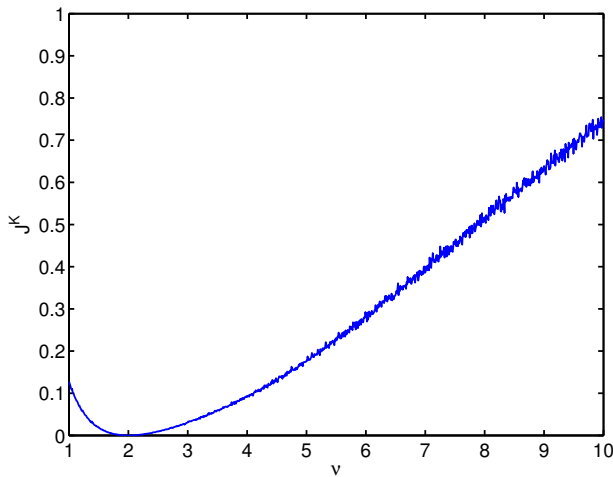


Figure 2. KD of a generalized Gaussian distribution (19), parametrized in ν . For $\nu = 2$ the generalized Gaussian coincides with the Gaussian distribution.

4. A C++ FILTERING ENVIRONMENT (F++)

A useful framework for development and evaluation of estimation algorithms must not only be intuitive and easy to use, but also numerically efficient. MATLAB has been very successful when it comes to the former objec-

tive, being extremely easy to use especially with the right toolboxes. The latter objective is better served by compiled programming languages, *e.g.*, C/C++ or Fortran, however without additional support these lack sufficient ability to express high-level mathematics, thus drawing attention away from algorithm development and evaluation. This paper presents a framework for efficient, still easy to use, algorithm development in C++. As will be seen in this section, this has been made possible through extensive use of the high-level programming concepts available in C++.

Tools for Scientific Computations

When it comes to doing quick evaluations and prototyping the predominant software today is MATLAB¹ [43] and different clones, *e.g.*, the GNU Octave² and Scilab³, both free software. The strength of these tools lies in the interactive environment they provide, where many functions and algorithms are already implemented making it easy to realize mathematical concepts. Furthermore, many researchers provide extensions in form of toolboxes to solve problems they find interesting. The price paid for this is computational speed — in most cases MATLAB, and similar tools, cannot be compared to more specialized software. This may not always be a problem, but large Monte Carlo simulations is one example where efficient and fast code is crucial.

For describing and working with model descriptions, the Modelica [44] language is another way to go. Modelica handles models described by *differential algebraic equations* (DAE), an extension to state-space models, where constraints are incorporated naturally. The DAE description comes especially handy when trying to model large systems. Smaller submodels can then be taken care of separately and be put together afterwards to form the complete system, which is then often in the form of a DAE. The Modelica language is implemented in commercial tools, *e.g.*, Dymola⁴, but there are also free alternatives, *e.g.*, OpenModelica⁵.

This paper presents an alternative to MATLAB for development, especially in cases where extensive Monte Carlo simulations are needed or where the computational burden is high. By providing an environment designed for signal processing in C++, where elements commonly needed are provided, development and testing can be moved to a faster environment with little extra effort.

¹MATLAB: <http://www.mathworks.com>

²GNU Octave: <http://www.gnu.org/software/octave>

³Scilab: <http://www.scilab.org>

⁴Dymola: <http://www.dynasim.com>

⁵OpenModelica:

<http://www.ida.liu.se/projects/OpenModelica>

Framework Design

The implemented framework for filtering applications is designed to be fast, but at the same time flexible enough to handle most problems directly and to be easy to use. Fulfilling these objectives are made possible by the use of the high-level abstractions and software constructions made available in C++. This section discusses the choices made at a high-level, specific implementation details are covered elsewhere.

Design Patterns—Modern *design patterns* have been used in the development of the software. Design patterns are general, programming language independent, conceptual high level solutions to common problems, [45]. Using well known and studied design patterns allow for faster development and robust design at a low cost. The used patterns include:

- *Smart pointers* — reference counted smart pointers are used to handle most dynamically allocated objects. Using the reference counted pointers minimizes the risk of memory leaks and segmentation faults due to invalid pointers. At the same time, they make the life easier for the user that does not have to care about explicitly destructing allocated resources. All this at a low cost in terms of increased computational complexity.
- *Object factories* — mechanisms with the sole purpose of providing an easy interface to create other objects, are used to eliminate the use of the two step object creation process otherwise needed. This is especially useful when loading objects from file, where it would be necessary for the user to check for object type and then have full knowledge of how to read every different kind of object from the file in order to create it without the help of an object factory.
- *Singletons* — object that by design may only exist as one instance are used both in object factories and for random number generation. The object factories uses singletons to hold a call back database for all different objects it is supposed to handle. Another singleton is used to provide a reliable random number generator.

Class Design— *Object oriented programming* (OOP) is a technique used in programming to separate large projects into separate units that interact only through well defined interfaces, and that can express relations between concepts through inheritance. This way it is easy to develop, test, and change different parts of a large system independently of the other parts. This is ideal for an estimation framework where it is often interesting to try different models, algorithms, *etc.* In this case the separation into logical components, *classes*, is based on the authors many years of experience of filtering applications, and on the analysis in for instance [46].

The simulation framework has tree major components that interacts: noise, models, and estimators. The idea

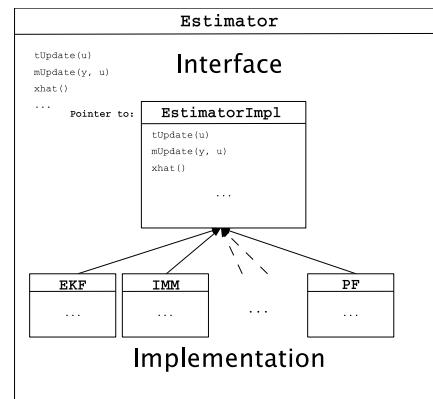


Figure 3. Illustration of the Estimator interface and its derived classes.

behind this setup is that by defining what constitutes a noise, a model, and an estimator, and then enforce it, it is possible to interchangeably use any implementation of these. This makes it possible to test new estimators or models in legacy code without hardly any changes at all.

Numerics— Concrete classes are used to represent and work with matrices and vectors. There are two reasons for this. One being that it should be easy to work with standard mathematical objects and expressions. The other reason is to ensure good numerical properties of all computations. The latter is obtained using standard numerical software; The Linear algebra package⁶ (LAPACK, [47]), which is also the foundation of MATLAB, and the GNU Scientific Library⁷ (GSL, [48]) in combination with GSL--⁸ for a more intuitive interface. Both software packages are known for their good numerical properties. GSL also has easy to use random numbers, that can be generated using various well-known methods.

For convenience of use, the framework also provides a simple singleton holding for random number generation. This to assure the availability of independent random numbers at all time.

External Interaction— The estimation framework is designed to be able to interact with MATLAB using the MATLAB file format. This way it is possible to make big simulations, save the result in a file and then just open it in MATLAB to visualize and/or further analyze the data. MATLAB files can also be used to store objects. The format used for this is hierarchical and uses structures to organize the information. As an example, a noise with Gaussian distribution, zero mean and unit variance is represented using the following structure (as written in MATLAB):

```
>> N = struct('ID', 'Noise::GaussNoise', ...  
            'mu', 0, ...
```

⁶LAPACK: <http://www.netlib.org/lapack>

⁷GNU GLS: <http://www.gnu.org/software/gsl>

⁸GSL--: <http://cholm.home.cern.ch/cholm/misc/gslmm>

```

        'Sigma', 1)
N =
    ID: 'Noise::GaussNoise'
    mu: 0
    Sigma: 1

```

This way to store objects has at least two advantages. First, it is easy to deal with and can be read almost as plain text. Second, it is easy to create and modify models using MATLAB. To utilize this storage method MATLAB must be present. If MATLAB is unavailable, this will be detected during the installation process and MATLAB based parts of the framework will be disabled. Generic programming techniques, where it at compile time is possible to decide what code should be used depending on the situation, are used for this purpose. This design choice, in combination with the choice of object representation, makes it easy to implement new file formats. An *extensible markup language* (XML) file format is planned for the future to allow for as close to full functionality without the need for MATLAB as possible.

A General C++ Filtering Environment: F++

In this section the software environment is presented highlighting some of its base components. To further illustrate the code, examples are given with code segments important to a user.

Estimators—The `Estimator` class is the interface to all estimators in the framework. Two important functions it provides are `tUpdate` and `mUpdate`, implementing the time update phase and measurement update phase of the filter. Another function is `xhat`, which provides the present point estimate. The `Estimator` class holds a reference counted pointer to a `EstimatorImpl` object, which is the mother of all estimators. Any new estimators must inherit `EstimatorImpl` and provide some basic estimator functionality demanded but the implementation of `Estimator`. Once that is done, the new estimator can be used anywhere an estimator is used.

Three estimators have been implemented so far, an EKF (which also acts as a KF), an IMM, and a PF. (See Figure 3.) Since an estimator is always referred to in terms of an `Estimator` object, when a new estimator has been implemented it can be used in legacy code without hardly any changes at all. At the same time, the inheritance from the base class `EstimatorImpl` and the access only through `Estimator` provides much of the standard behavior so that it does not have to be duplicated.

Below follows an example of how to process a set of data:

```

for (size_t i=0; i<Tmax; ++i) {
    filter.tUpdate(u[i]);
    filter.mUpdate(y[i], u[i]);
    Xhat[i] = filter.xhat();
}

```

Note that `filter` could be any estimator, for instance EKF, IMM, or PF, in a given implementation. But the code still remains the same.

Creation of `Estimator` objects are implemented using a system of factory objects providing an easy interface to both direct creation of estimators and to reading an estimator from disk without exactly knowing its type.

Models—All models used in the simulation framework are organized in a way similar to that of estimators. The interface class for models is `Model` which holds a reference counted pointer to a `ModelImpl` object. Based on `ModelImpl` it is easy to derive new specialized models for all needs. A few examples on how to do this is provided within the framework, including a linear model, `LinModel`, a template based model able to handle any type of model, and a model handling multiple models, as well as a some standard tracking models. (See Figure 4.) The code provided below shows what specializations must be done to implement the model needed for the simulation in Example I, Section 5:

```

Model::vector_t
Model::f(const vector_t& x,
         const vector_t& w,
         const vector_t& u) const
{
    double t = x.time;
    vector_type X = x.datum;
    vector_type W = w.datum;
    vector_type nextX(1);
    nextX[0] = 0.5*X[0] + 25.0*X[0]/(1+X[0]*X[0])
              + 8.0*cos(1.2*t) + W[0];
    return vector_t(nextX, t+T);
}

Model::vector_type
Model::h(const vector_t& x,
         const vector_t& e,
         const vector_t& u) const
{
    vector_type X = x.datum;
    vector_type E = e.datum;
    vector_type Y(1);
    Y[0] = X[0]*X[0]/20.0 + E[0];
    return vector_t(Y, x.time);
}

```

To use the EKF a few derivatives must also be written in a way similar to the functions listed above. Linear models are easily available by just supplying the appropriate matrices and noise:

```
model = createLinModel(F, G, H, W, E);
```

where `F`, `G`, and `H` are matrices, and where `W` and `E` are noise objects.

Noise—The last piece in the framework is a representation of noise, which is given by the interface class `Noise`, holding a reference counted `NoiseImpl` object representing the actual noise. The interface includes functions to generate random numbers from the distribution

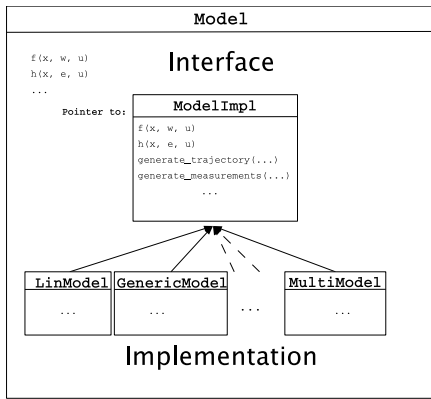


Figure 4. Illustration of the `Model` interface and its derived classes.

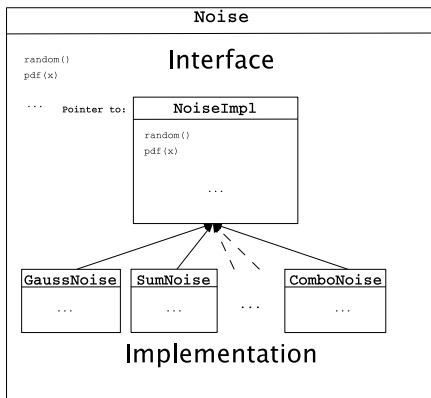


Figure 5. Illustration of the `Noise` interface and its derived classes.

represented by the class, and also a function to compute the likelihood of a certain value. Implementations of Gaussian noise, Gaussian sum noise, a noise similar to the Gaussian sums but with any noise as components, and stacked independent noises are currently available in the framework, and other distributions are easily added. (See Figure 5.) The following code shows how to generate samples from a Gaussian distribution with mean μ and (co)variance Σ :

```

N = createGaussNoise(Sigma, mu, "GaussDist");
x = N.random();

```

Simulations—To conduct simulations it is important to be able to easily generate trajectories and measurements for Monte Carlo simulations. Trajectories and measurements can be obtained from the model description with just a few lines of code, and then repeating this gives a Monte Carlo simulation study analyzed using RMSE and the result is saved to file. This is shown below:

```

for (long i=0; i<N_MC; ++i) {
  X = generate_trajectory(model, x0, U);
  Y = generate_measurement(model, X, U);

  // Xhat = ....; // Filtering loop
  Xs[i] = X;

```

```

  Xhats[i] = Xhat;
}
RMSE = rmse(Xs, Xhats);
write(outfile, "RMSE", RMSE);

```

It is just as easy to read data from a file using the environment, as writing it to file. It is hence not difficult to work with experimental data or data from simulations provided from elsewhere, *e.g.*, benchmark problems as those in [49].

Extensions—The simulation environment provides some basic noise classes, models, and estimators. However, in the future these will be extended as more applications implements new classes. The software is being developed in Linux, taking full advantage of all the powerful tools available there: *gcc*, *automake/autoconf*, *gdb*, *gprof*, *etc.* When compiled it provides the user with a library which makes it easy to interact with other user specific code. It is the authors ambition to provide both source code and a precompiled library, hence simplifying the usage. If possible, precompiled libraries will be provided for Microsoft Windows, at least for use in the Unix emulator Cygwin⁹.

Code Homepage, Download

The development framework F++ discussed in the above section is available as open source from the F++ homepage¹⁰. This homepage also offers a more complete documentation of the code and usage, as well as more advanced examples of the code being used.

5. SIMULATIONS

In the simulation study three different examples are studied to highlight the performance gain obtained from estimating the full PDF instead of just first and second order moments (Example I and II), and to exemplify the simulation framework on a realistic positioning example (Example III). Both the PF and the EKF are extensively used.

In the Monte Carlo simulation studies, the MSE is compared to the parametric CRLB. Furthermore, the KD, between the true state distribution (from the fine gridded PMF) and the distributions provided by the filters, are compared to capture more of the differences not seen in the second-order moment.

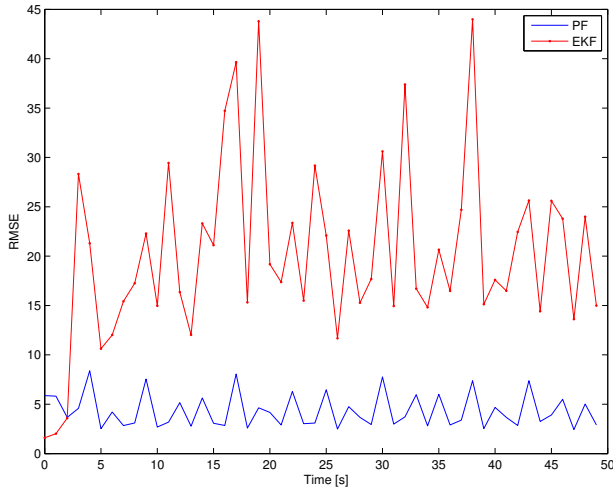


Figure 6. Example I. The RMSE for the one dimensional nonlinear model using 400 Monte Carlo simulations.

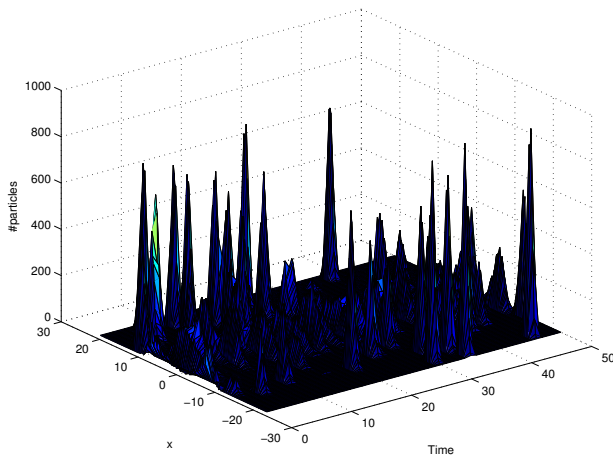


Figure 7. Example I. The PDF as a function of time for one realization.

Example I — One Dimensional Nonlinear System

Consider the following nonlinear model from [50], [5]

$$x_{t+1} = 0.5x_t + \frac{25x_t}{1+x_t^2} + 8\cos(1.2t) + w_t, \quad (20a)$$

$$y_t = \frac{x_t^2}{20} + e_t, \quad (20b)$$

with $w_t \sim \mathcal{N}(0, 10)$ and $e_t \sim (0, 1)$. In Figure 6 the RMSE for the EKF and the PF are depicted. As seen the PF outperforms the EKF. This can be explained studying Figure 7, where the PDF shows a quite multi-modal behavior for several time instances.

In Figure 8 the KD comparing the PF and the EKF solution to the estimation problem is shown, for one realization. To obtain the measure, the particle cloud from the

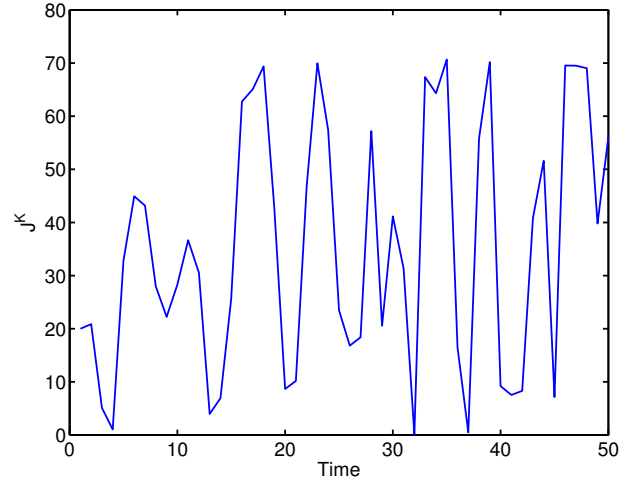


Figure 8. Example I. KD for the one dimensional nonlinear model using one realization.

particle filter was smoothed with a small Gaussian kernel to get a continuous PDF estimate before the KD was computed using a sum approximation. When the KD is small the EKF yields a PDF similar to the PF one, whereas when the KD is large the difference is more significant.

Example II — Range-Only Measurement

In a range-only measurement application based on [8], two range sensors are used. The measurements are illustrated in Figure 9(a). They provide the relative range to a single target, with measurement uncertainty, hence, producing a natural bimodal posterior distribution. The model used is:

$$x_{t+1} = x_t + w_t \quad (21a)$$

$$y_t = \begin{pmatrix} \|x_t - x_1^{\text{sensor}}\| \\ \|x_t - x_2^{\text{sensor}}\| \end{pmatrix} + e_t, \quad (21b)$$

with $w_t \sim \mathcal{N}(0, 0.1I_2)$, $e_t \sim \mathcal{N}(0, 0.1I_2)$, and initial position knowledge $x_0 \sim \mathcal{N}(0, 3I_2)$. The model assumes that the target is moving around according to a random walk.

A typical state distribution is given in Figure 9(b). Note the distinct bimodal characteristics of the distribution, as well as how poorly the EKF approximation describes the situation.

The MSE and KD from 100 Monte Carlo simulations are given in Figure 10 for an EKF, a PF ($N = 20\,000$ particles¹¹), and a PMF (regarded as the truth). Here, the MSE performance of the PF is slightly better than for the EKF, but not as good as the PMF. (Two poor PF estimates have large impact on the MSE.) However, the KD gives a clear indication that the PDF estimated with the

⁹CYGIN: <http://www.cygwin.com>

¹⁰F++: <http://www.control.isy.liu.se/resources/f++>

¹¹The number of particles has been chosen excessively large to get an accurate PDF estimate, with further tuning could be reduced significantly.

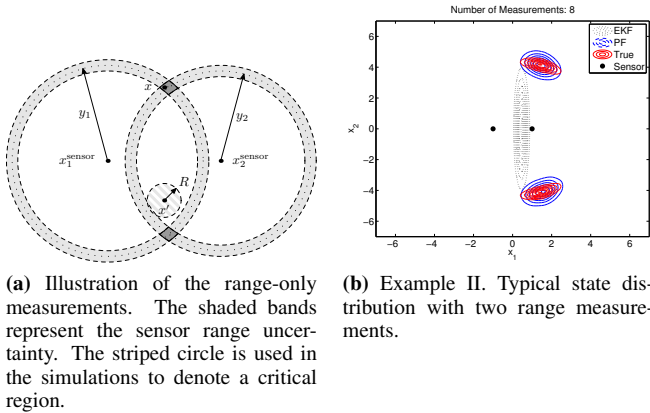


Figure 9. Example II. Scenario and typical PDF.

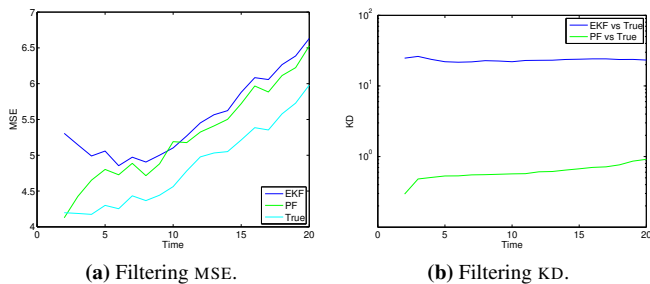


Figure 10. Example II. Simulated MSE and KD for the range-only system.

PF is better than the one from the EKF, as was to be expected due to the true bimodal PDF. This difference may be important, as will be shown next.

Using the estimated PDFs, it is also possible to detect if the tracked target is in the neighborhood of the point x' (not affecting the measurements), as illustrated in Figure 9(a). Assume that it is of interest to detect if $\|x - x'\|_2 < R$, where $x' = (0, 1)$ and $R = 0.5$. The probability for the target in the critical region, given the estimates from the different filters is given in Figure 11. Note that the EKF throughout the whole simulation indicates a much higher probability than is true. Furthermore, the PF reflects the actual situation well. The lack of descriptive power of the EKF results in an unnecessarily high degree of detections, which could be costly.

Example III — Inertial Measurement Unit (IMU)

In this section a positioning simulation study is presented using an *inertial measurement unit* (IMU) in combination with a GPS sensor. This is a rather common setup in many tracking and positioning applications. Principally, the IMU measures the acceleration using an accelerometer and angular rates from a gyro, from which the position and orientation can be derived by integra-

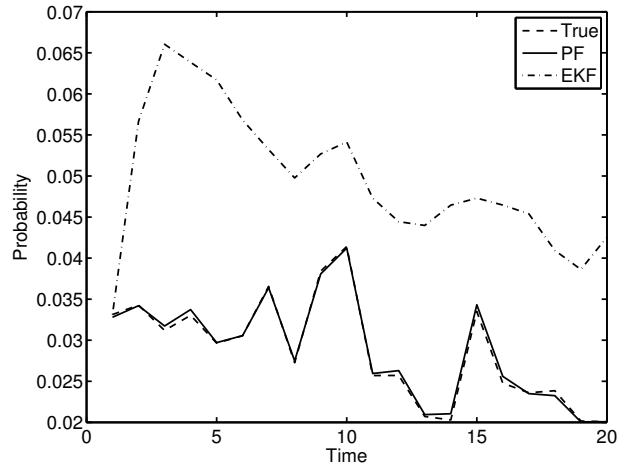


Figure 11. Example II. Probability of the target belonging to the critical region given by $\|x - (0, 1)\|_2 < 0.5$, see Figure 9(a).

tion. In this paper a point model for the object to positioning is used. This is a simplification, since the main objective here is to present the ability for the F++ framework to handle many states at a high data rate. In reality, for aircraft positioning, a full aircraft model coupling the kinematic part with the orientation part should be used. Here, using quaternions, $\mathbf{q} = (q_0, q_1, q_2, q_3)^T$, instead of Euler angles to represent the orientation, the continuous time dynamics for the point target is

$$\dot{x} = \begin{pmatrix} \dot{\mathbf{p}} \\ \dot{\mathbf{v}} \\ \dot{\mathbf{a}} \\ \dot{\mathbf{q}} \\ \dot{\boldsymbol{\omega}} \end{pmatrix} = \begin{pmatrix} \mathbf{v} \\ \mathbf{a} + \mathbf{u} \\ 0 \\ \frac{1}{2}S(\boldsymbol{\omega})\mathbf{q} \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ w_1 \\ 0 \\ w_2 \end{pmatrix}, \quad (22a)$$

$$y_t = h(x_t) + e_t, \quad (22b)$$

where

$$h(x_t) = \begin{cases} \begin{pmatrix} \mathbf{a}_m \\ \boldsymbol{\omega}_m \end{pmatrix} = \begin{pmatrix} \mathbf{q}(\mathbf{a} + \mathbf{g})\mathbf{q}^* \\ \mathbf{q}\boldsymbol{\omega}\mathbf{q}^* \end{pmatrix}, & \text{if IMU} \\ \begin{pmatrix} \mathbf{p} \\ \mathbf{v} \end{pmatrix}, & \text{if GPS} \end{cases}$$

where the state vector, $x \in \mathbb{R}^{16}$, consists of position ($\mathbf{p} \in \mathbb{R}^3$), velocity ($\mathbf{v} \in \mathbb{R}^3$), acceleration ($\mathbf{a} \in \mathbb{R}^3$), quaternions ($\mathbf{q} \in \mathbb{R}^4$ and $\|\mathbf{q}\| = 1$), and angular velocity from the rate gyro ($\boldsymbol{\omega} \in \mathbb{R}^3$), and where the gravitational force vector is denoted $\mathbf{g} = (0, 0, 9.82)^T$. The input signal \mathbf{u} is here fully known acceleration components.

$$x = (\mathbf{p} \ \mathbf{v} \ \mathbf{a} \ \mathbf{q} \ \boldsymbol{\omega})^T,$$

and where the acceleration and angular velocity are measured (\mathbf{a}_m and $\boldsymbol{\omega}_m$) by the IMU every 0.05 s, and the posi-

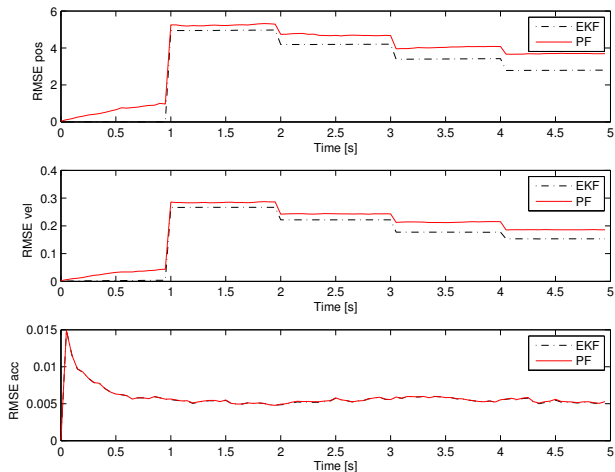


Figure 12. Example III. RMSE from 50 Monte Carlo simulation.

tion (p) and velocity (v) by the GPS every 1 s, and where

$$S(\omega) = \begin{pmatrix} 0 & -\omega_1 & -\omega_2 & -\omega_3 \\ \omega_1 & 0 & \omega_3 & -\omega_2 \\ \omega_2 & -\omega_3 & 0 & \omega_1 \\ \omega_3 & \omega_2 & -\omega_1 & 0 \end{pmatrix}.$$

Note that this is the dynamics for a point object, so the rotation and position dynamics are completely decoupled. In reality, in for instance aircraft positioning, the point model should be replaced by flight dynamics. In the example the IMU is running at 20 Hz, whereas GPS measurements are available at 1 Hz. The quaternion complex conjugate is denoted q^* . Also note that in order to represent the orientation a unit quaternion is assumed, which is achieved by normalizing q in the filter, to remove possible numerical discrepancies.

In the C++ framework the above continuous time model is discretized using a simple one-step ahead Euler method. For the EKF, it is first linearized and then discretized in order to obtain the system matrix, F . The continuous process noise is discretized assuming $Q^d = TQ^c$. All this is done analytically using the Symbolic Toolbox in MATLAB, and then automatically exported to C-code, which is readily compiled into the simulation environment.

In Figure 12 the RMSE for the EKF and the PF from 50 Monte Carlo simulations are depicted. The filters were initialized around the true value with a small uncertainty regions. As input, a deterministic, fully known sinusoidal acceleration input u was used. The simulation parameters are summarized in Table 1. Basically, the methods give the same RMSE. Due to the high dimension of the example, and that the current PF implementations uses prior sampling or the transitional density for particle proposal, this is too inefficient to achieve better RMSE values. In order to improve its performance, better pro-

Table 1. Example III. Parameters for IMU/GPS simulation.

Parameter	Value
MC simulations	50
Sample time IMU (T)	0.05 s
Sample time GPS	1 s
Number of particles	100000
<i>Measurement Noise:</i>	
cov p	diag(9, 9, 25)
cov v	diag(0.04, 0.04, 0.04)
cov a_m	diag(0.0001, 0.0001, 0.0001)
cov ω_m	diag($9 \cdot 10^{-8}$, $9 \cdot 10^{-8}$, $9 \cdot 10^{-8}$)
<i>Continuous Process Noise:</i>	
cov w_1	diag(0.0004, 0.0004, 0.0004)
cov w_2	diag($1 \cdot 10^{-8}$, $1 \cdot 10^{-8}$, $1 \cdot 10^{-8}$)

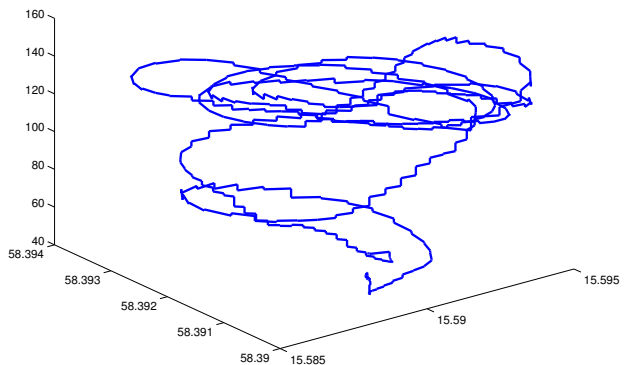


Figure 13. Example III. GPS from a flight-test with the u-blox GPS. The height coordinate plotted as a function of longitude and latitude from a small airfield in Linköping, Sweden.

posal densities could be one alternative. Probably, the EKF or a UKF can be used as a more efficient proposal. Also marginalization or Rao-Blackwellization could be applied, marginalizing the position and velocity states.

As mentioned, the simple point object dynamic model should be replaced by flight dynamics. In [12], this is described for a UAV project, where the EKF uses IMU and GPS measurements. In Figure 13 the u-blox¹² GPS position measurements for the small model aircraft used in the project, are depicted.

In realistic applications, the model can also be extended with bias and drift terms.

6. SUMMARY

In extensive Monte Carlo simulations the *Kullback divergence* (KD) and the *mean square error* (MSE) are evaluated for different estimators, such as the *extended Kalman filter* (EKF) and the *particle filter* (PF) for different tracking applications. It is shown that the KD is more relevant than MSE as a performance measure, for

¹²u-blox: <http://www.u-blox.com>

instance in determining target presence. The simulations have been conducted in a newly developed C++ simulation environment, designed to be suitable for nonlinear non-Gaussian problems. The environment has been presented in some detail, and the experience is a well functioning and fast development tool. Using modern design techniques the object oriented framework provides means for easy and efficient comparisons of different estimators, which is extensively used in several applications and examples.

ACKNOWLEDGMENT

This work has been funded by the Swedish Research Council. Also thanks to Mikael Borell at PowerMinds, for several fruitful discussions and joint work in experimental studies relating to inertial navigation and GPS positioning.

REFERENCES

- [1] A. H. Jazwinski, *Stochastic Processes and Filtering Theory*, vol. 64 of *Mathematics in Science and Engineering*, Academic Press, Inc, 1970.
- [2] B. D. O. Anderson and J. B. Moore, *Optimal Filtering*, Prentice-Hall, Inc, Englewood Cliffs, NJ, 1979.
- [3] T. Kailath, A. H. Sayed, and B. Hassibi, *Linear Estimation*, Prentice-Hall, Inc, 2000.
- [4] H. W. Sorenson and D. L. Alspach, "Recursive Bayesian estimation using Gaussian sums," *Automatica*, vol. 7, no. 4, pp. 465–479, July 1971.
- [5] N. J. Gordon, D. J. Salmond, and A. F. M. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," *IEE Proc.-F*, vol. 140, no. 2, pp. 107–113, Apr. 1993.
- [6] A. Doucet, N. de Freitas, and N. Gordon, Eds., *Sequential Monte Carlo Methods in Practice*, Statistics for Engineering and Information Science. Springer-Verlag, New York, 2001.
- [7] B. Ristic, S. Arulampalam, and N. Gordon, *Beyond the Kalman Filter: Particle Filters for Tracking Applications*, Artech House, Inc, 2004.
- [8] G. Hendeby, R. Karlsson, F. Gustafsson, and N. Gordon, "Performance issues in non-Gaussian filtering problems," in *Proc. Nonlinear Statistical Signal Processing Workshop*, Cambridge, UK, Sept. 2006.
- [9] B. Stroustrup, *The C++ Programming Language*, Addison-Wesley, 3 edition, 1997.
- [10] R. Karlsson and F. Gustafsson, "Bayesian surface and underwater navigation," *IEEE Trans. Signal Processing*, vol. 54, no. 11, pp. 4204–4213, Nov. 2006.
- [11] R. Karlsson and M. Norrlöf, "Position estimation and modeling of a flexible industrial robot," in *Proc. 16th Triennial IFAC World Congress*, Prague, Czech Republic, July 2005.
- [12] R. Karlsson, D. Törnqvist, A. Hansson, and S. Gunnarsson, "Automatic control project course: A positioning and control application for an unmanned aerial vehicle," *World Trans. Eng. and Tech. Edu.*, vol. 5, no. 2, pp. 291–294, 2006.
- [13] R. Karlsson, J. Jansson, and F. Gustafsson, "Model-based statistical tracking and decision making for collision avoidance application," in *Proc. American Contr. Conf*, Boston, MA, USA, July 2004, pp. 3435–3440.
- [14] R. Karlsson and F. Gustafsson, "Recursive Bayesian estimation: bearings-only applications," *IEE Proc.-Radar Sonar Navig.*, vol. 152, no. 5, pp. 305–313, Oct. 2005.
- [15] G. Hendeby, R. Karlsson, F. Gustafsson, and N. Gordon, "Recursive triangulation using bearings-only sensors," in *Proc. IEE TARGET*, Birmingham, UK, Mar. 2006.
- [16] Y. Boers, H. Driessen, J. Torstensson, M. Trieb, R. Karlsson, and F. Gustafsson, "Track-before-detect algorithm for tracking extended targets," *IEE Proc.-Radar Sonar Navig.*, vol. 153, no. 4, pp. 345–351, Aug. 2006.
- [17] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Trans. ASME*, vol. 82, no. Series D, pp. 35–45, Mar. 1960.
- [18] H. W. Sorenson, "Recursive estimation for nonlinear dynamic systems," in *Bayesian Analysis of Time Series and Dynamic Models*, J. C. Spall, Ed., pp. 126–165. Marcel Dekker, Inc, New York, NY, USA, 1988.
- [19] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking," *IEEE Trans. Signal Processing*, vol. 50, no. 2, pp. 174–188, Feb. 2002.
- [20] F. Gustafsson, *Adaptive Filtering and Change Detection*, John Wiley & Sons, Ltd, Chichester, West Sussex, England, 2000.
- [21] S. C. Kramer and H. W. Sorenson, "Bayesian parameter estimation," *IEEE Trans. Automat. Contr.*, vol. 33, no. 2, pp. 217–222, Feb. 1988.
- [22] H. Tanizaki, "Nonlinear and nonnormal filters using Monte Carlo methods," *Comput. Stat. & Data Analysis*, vol. 25, pp. 417–439, 1997.
- [23] N. Bergman, *Recursive Bayesian Estimation: Navigation and Tracking Applications*, Dissertations no 579, Linköping Studies in Science and Technology, SE-581 83 Linköping, Sweden, May 1999.
- [24] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, and P.-J. Nordlund, "Particle filters for positioning, navigation, and tracking," *IEEE Trans. Signal Processing*, vol. 50, no. 2, pp. 425–437, Feb. 2002.
- [25] A. Doucet, S. Godsill, and C. Andrieu, "On sequential Monte Carlo sampling methods for Bayesian filtering," *Statistics and Computing*, vol. 10, pp. 197–208, 2000.
- [26] T. Schön, F. Gustafsson, and N. Per-Johan, "Marginalized particle filters for mixed linear / nonlinear state-space models," *IEEE Trans. Signal Processing*, vol. 53, no. 7, pp. 2279–2289, July 2005.
- [27] M. K. Pitt and N. Shephard, "Filtering via simulation: Auxiliary particle filters," *JASA*, vol. 94, no. 446, pp. 590–599, June 1999.
- [28] J. Míguez, M. F. Bugallo, and P. M. Djurić, "A new class of particle filters for random dynamic systems with unknown statistics," *J. Appl. Signal Proc.*, vol. 15, pp. 2278–2294, 2004.
- [29] S. Godsill and J. Vermaak, "Models and algorithms for tracking using trans-dimensional sequence sequential Monte Carlo," in *Proc. IEEE Int. Conf. on Acoust., Speech, Signal Processing*, Montreal, Canada, May 2004, pp. 976–979.
- [30] J. H. Kotecha and P. M. Djurić, "Gaussian sum particle filtering," *IEEE Trans. Signal Processing*, vol. 51, no. 10, pp. 2602–2612, Oct. 2003.
- [31] H. A. P. Blom and Y. Bar-Shalom, "The interacting multiple model algorithm for systems with Markovian switching coefficients," *IEEE Trans. Automat. Contr.*, vol. 33, no. 8, pp. 780–783, Aug. 1988.
- [32] H. A. P. Blom, "An efficient filter for abruptly changing systems," in *Proc. 23rd IEEE Conf. Decis. and Contr*, Las Vegas, NV, USA, Dec. 1984, pp. 656–658.
- [33] Y. Bar-Shalom and X. R. Li, *Estimation and Tracking: Principles, Techniques, and Software*, Artech House, Inc, 1993.


- [34] H. Cramér, *Mathematical Methods of Statistics*, Princeton University Press, Princeton, NJ, USA, 1946.
- [35] S. M. Kay, *Fundamentals of Statistical Signal Processing: Estimation Theory*, vol. 1, Prentice-Hall, Inc, 1993.
- [36] E. L. Lehmann, *Theory of Point Estimation*, Probability and Mathematical Statistics. John Wiley & Sons, Ltd, 1983.
- [37] S. Kullback, J. C. Keegel, and J. H. Kullback, *Topics in Statistical Information Theory*, vol. 42 of *Lecture Notes in Statistics*, Springer-Verlag, 1987.
- [38] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Statist.*, vol. 22, no. 1, pp. 79–86, Mar. 1951.
- [39] C. Arndt, *Information Measures*, Springer-Verlag, 2001.
- [40] S. Gibson and B. Ninness, "Robust maximum-likelihood estimation of multivariable dynamic systems," *Automatica*, vol. 41, pp. 1667–1682, 2005.
- [41] J. Bröcker, "On comparing nonlinear filtering algorithms," in *Proc. 2005 Int. Symp. Nonlin. Theory App.*, Bruges, Belgium, Oct. 2005.
- [42] G. E. P. Box and G. C. Tao, *Bayesian Inference in Statistical Analysis*, Addison-Wesley, 1973.
- [43] The MathWorks, *MATLAB The Language of Technical Computing. Version 7*, 2006.
- [44] P. A. Fritzon, *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, IEEE Press, Piscataway, NJ, USA, 2004.
- [45] A. Alexandrescu, *Modern C++ Design. Generic Programming and Design Patterns Applied*, Addison-Wesley, 2001.
- [46] J. Rosén, "A framework for nonlinear filtering in MATLAB," Master's thesis no LiTH-ISY-EX-05/3733--SE, Dept. Electr. Eng, Linköpings universitet, Sweden, Oct. 2005.
- [47] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 3. edition, 1999.
- [48] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi, *GNU Scientific Library*, 1.8 edition, Mar. 2006, For GSL version 1.8.
- [49] W. D. Blair, G. A. Watson, T. Kirubarajan, and Y. Bar-Shalom, "Benchmark for radar allocation and tracking in ECM," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 34, no. 4, pp. 1097–1114, Oct. 1998.
- [50] G. Kitagawa, "Non-Gaussian state-space modeling of nonstationary time series," *JASA*, vol. 82, no. 400, pp. 1032–1041, Dec. 1987.



Rickard Karlsson was born 1970 in Örebro, Sweden. He received the M.Sc. degree in Applied Physics and Electrical Engineering in 1996, and a Ph.D. in Automatic Control in 2005, both from Linköping university, Linköping, Sweden. He worked at Saab Bofors Dynamics in Linköping between 1997–2002, with sensor fusion and target tracking, between 1999–2002 partly employed at the Automatic Control group, and from 2002 full time. He is currently a research associate at the Automatic Control group in Linköping. His research interests include positioning and tracking applications mainly using particle filters.



Gustaf Hendeby was born 1978 in Stenstorp, Sweden. He received his M.Sc. degree in Applied Physics and Electrical Engineering in 2002 and his Licentiate of Engineering degree in 2005, both from Linköpings universitet, Sweden. He is currently pursuing a Ph.D. in Automatic Control in Linköping. His major research interest lies in fundamental performance limitations for estimation and detection, and the methods used to achieve these bounds. He is a student member of the IEEE.

	Avdelning, Institution Division, Department Division of Automatic Control Department of Electrical Engineering	Datum Date 2007-01-24
---	--	--

Språk Language <input type="checkbox"/> Svenska/Swedish <input checked="" type="checkbox"/> Engelska/English <input type="checkbox"/> _____	Rapporttyp Report category <input type="checkbox"/> Licentiatavhandling <input type="checkbox"/> Examensarbete <input type="checkbox"/> C-uppsats <input type="checkbox"/> D-uppsats <input checked="" type="checkbox"/> Övrig rapport <input type="checkbox"/> _____	ISBN _____ ISRN _____ Serietitel och serienummer ISSN Title of series, numbering 1400-3902
--	---	--

URL för elektronisk version http://www.control.isy.liu.se	LiTH-ISY-R-2767
---	-----------------

Titel Title Författare Author	Target Tracking Performance Evaluation — A General Software Environment for Filtering Gustaf Hendeby, Rickard Karlsson
--	---

Sammanfattning Abstract <p>In this paper, several recursive Bayesian filtering methods for target tracking are discussed. Performance for target tracking problems is usually measured using the second-order moment. For nonlinear or non-Gaussian applications, this measure is not always sufficient. The <i>Kullback divergence</i> is proposed as an alternative to mean square error analysis, and it is extensively used to compare estimated posterior distributions for various applications. The important issue of efficient software development, for nonlinear and non-Gaussian estimation, is also addressed. A new framework in C++ is detailed. Utilizing modern design techniques an object oriented filtering and simulation framework is provided to allow for easy and efficient comparisons of different estimators. The software environment is extensively used in several applications and examples.</p>

Nyckelord Keywords	tracking, filtering, particle filter
------------------------------	--------------------------------------

