Bayesian Inference for Nonlinear Dynamical Systems — Applications and Software Implementation

Jerker Nordh

LUND
UNIVERSITY

Department of Automatic Control

PhD Thesis
ISRN LUTFD2/TFRT--1107--SE
ISBN 978-91-7623-323-8 (print)
ISBN 978-91-7623-324-5 (web)
ISSN 0280–5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Media-Tryck.
Lund 2015

Abstract

The topic of this thesis is estimation of nonlinear dynamical systems, focusing on the use of methods such as particle filtering and smoothing. There are three areas of contributions: software implementation, applications of nonlinear estimation and some theoretical extensions to existing algorithms.

The common theme for all the work presented is the pyParticleEst software framework, which has been developed by the author. It is a generic software framework to assist in the application of particle methods to new problems, and to make it easy to implement and test new methods on existing problems.

The theoretical contributions are extensions to existing methods, specifically the Auxiliary Particle Filter and the Metropolis Hastings Improved Particle Smoother, to handle mixed linear/nonlinear models using Rao-Blackwellized methods. This work was motivated by the desire to have a coherent set of methods and model-classes in the software framework so that all algorithms can be applied to all applicable types of models.

There are three applications of these methods discussed in the thesis. The first is the modeling of periodic autonomous signals by describing them as the output of a second order system. The second is nonlinear grey-box system identification of a quadruple-tank laboratory process. The third is simultaneous localization and mapping for indoor navigation using ultrasonic range-finders.

Acknowledgments

First I thank the department of Automatic Control for essentially allowing me to work on whatever I found interesting, and for my supervisor Bo Bernhardsson's belief that it would eventually lead to something worthwhile, a belief I think at many times were stronger with him than with me. The feedback Thomas Schön, Uppsala University, generously took his time to provide over the years has also served a vital role in determining the direction of my work.

Additionally I wish to recognize Anders Mannesson, Jacob Antonsson and Karl Berntorp which are those at the department with whom I had the closest cooperation with for my research, not all of which is readily apparent in the form of publications.

I also thank my colleagues with whom I shared our office during my time at the department, Björn Olofsson, Jonas Dürango and Ola Johnsson, for always taking the time to discuss any problems and ideas I had, research related and otherwise, which has undoubtedly saved me countless hours of work and frustration over the years.

My thanks go out to Leif Andersson's, his assistance and knowledge of LaTeX has been invaluable during the preparation of this thesis.

Perhaps most importantly I thank Karl-Erik Årzén for not holding a grudge when I applied for the position as PhD student, after four years earlier turning down his suggestion to stay at the department as a graduate student. The years outside the academic world has made me appreciate the freedom it provides so much more.

Finally I thank all my colleagues at the department for all the interesting and fun discussions, be it during the mandatory coffee breaks or while drinking wine on a ship in the Mediterranean.

Contents

Preface

Outline and Contributions of the Thesis

This thesis consists of three introductory chapters and eight papers. This section describes the contents of each chapter and details the contributions for each paper.

Chaper 1 - Introduction

This chapter gives a brief introduction to the concept of state estimation, and gives a theoretical background for the work in this thesis, detailing the different methods that are the foundation for the work performed.

Chaper 2 - Implementation

This chapter introduces the *pyParticleEst* software that has been the common theme for all the work performed in this thesis. It also gives a quick overview of other existing software that aim to solve similar problems.

Chaper 3 - Applications

This chapter provides some background for the application areas related to the work in this thesis. These areas are system identification, autonomous systems for modeling of periodic signals and simultaneous localization and mapping in indoor environments.

Paper I

Nordh, J. (2015). "pyParticleEst: a Python framework for particle based estimation methods". *Journal of Statistical Software*. Conditionally accepted, minor revision.

This paper describes the software framework *pyParticleEst* that the author has developed. The software is an attempt to simplify the application of particle methods, such as particle filters and smoothers, to new problems. The article describes the structure of the software and breaks down a number of popular algorithms to a set of common operations that are performed on the mathematical model describing a specific problem. It also identifies some popular types of models and presents building blocks in the framework to further assist the user when operating on mathematical models of those types.

Paper II

Nordh, J. and J. Antonsson (2015, Submitted). "Quantitative evaluation of sampling based smoothers for state space models". *EURASIP Journal on Advances in Signal Processing*. Submitted.

The paper compares three different methods for computing smoothed state estimates with regards to the root mean square error of the estimate in relation to the number of operations that have to be performed. The author's contributions to this paper is the idea for how to measure the computational effort in an implementation independent way and the software implementation of the algorithms and examples.

Paper III

Ackzell, E., N. Duarte, and J. Nordh (2015, Submitted). "Nonlinear MPC and grey-box identification using PSAEM of a quadruple-tank laboratory process". Submitted.

The paper describes the application of Particle Stochastic Approximation Expectation Maximization (PSAEM) to perform greybox identification of a quadruple-tank laboratory setup. The identified model is used to perform nonlinear Model Predictive Control (MPC) by linearizing the system around the previously predicted trajectory. The work was partly performed by two students in an applied automatic control course where the author acted as supervisor. The author's contributions to the project was the initial project idea and the implementation of the system identification parts.

Paper IV

Nordh, J., T. Wigren, T. B. Schön, and B. Bernhardsson (2015, Submitted). "Particle filtering based identification for autonomous nonlinear ODE models". In: *17th IFAC Symposium on System Identification*. Beijing, China. Submitted.

The paper presents an algorithm based on PSAEM to identify a model that describes an autonomous periodic signal. It improves on previously published results that relied on linear approximations. The author's contributions to the paper are the formulation using a marginalized model, which allowed the effective sampling of the particles in the Particle Gibbs Ancestral Sampler (PGAS) kernel, and the software implementation and experimental results.

Paper V

Nordh, J. (2015, Submitted). "Metropolis-Hastings improved particle smoother and marginalized models". In: *European Conference on Signal Processing 2015*. Nice, France. Submitted.

The paper describes how to combine the Metropolis-Hastings Improved Particle Smoother (MHIPS) with marginalized models. It also demonstrates the importance of marginalization for an effective MHIPS algorithm. This work was motivated by the need to provide a cohesive set of methods in the pyParticleEst framework for all the supported model classes.

Paper VI

Nordh, J. (2014). "Rao-Blackwellized auxiliary particle filters for mixed linear/nonlinear Gaussian models". In: *12th International Conference on Signal Processing*. Hangzhou, China.

The paper extends a previously published method for using Auxiliary Particle Filters on mixed linear/nonlinear Gaussian models. It proposes two new approximations for the required density $p(y_{t+1}|x_t)$. The first approximation uses a local linearization and the second relies on the Unscented Transform. The three methods are compared for two examples to highlight the differences in behavior of the approximations.

Paper VII

Nordh, J. and K. Berntorp (2012). "Extending the occupancy grid concept for low-cost sensor based SLAM". In: *10th International IFAC Symposium on Robot Control*. Dubrovnik, Croatia.

The paper presents an extension to the occupancy grid concept in order to handle the problems associated with using ultrasound based range sensors. The motivation for the work was to be able to perform Simultaneous Localization and Mapping (SLAM) using inexpensive sensors, demonstrated by the use of a simple differential drive robot built of LEGO. The author's contributions were the idea how to extend the map to compensate for the sensor characteristics and the software implementation.

Paper VIII

Berntorp, K. and J. Nordh (2014). "Rao-Blackwellized particle smoothing for occupancy-grid based SLAM using low-cost sensors". In: *19th IFAC World Congress*. Cape Town, South Africa.

The paper continues the work of Paper VII by not only considering the filtering problem but also the smoothing problem. It also demonstrates the validity of the approach by collecting a real world dataset with ground truth for the motion obtained through the use of a VICON tracking system and shows that the proposed method gives better position estimates than relying on dead-reckoning. The author's contributions were the model formulation for the differential drive robot to avoid the degeneracy problem, the analytic solution for evaluation of the required smoothing density and the software implementation.

In addition to the work detailed above the author performed substantial parts of the writing and editing of all the articles with multiple authors.

Glossary

APF	Auxiliary Particle Filter
BSD	Berkeley Software Distribution
CPF	Conditional Particle Filter
CPFAS	Conditional Particle Filter Ancestral Sampler
EKF	Extended Kalman Filter
EM	Expectation Maximization
FFBP	Forward Filter Backward Proposer
FFBSi	Forward Filter Backward Simulator
GPL	GNU General Public License
KF	Kalman Filter
LGPL	GNU Lesser General Public License
LTV	Linear Time-Varying
MCMC	Markov Chain Monte Carlo
MHIPS	Metropolis-Hastings Improved Particle Smoother
ML	Maximum Likelihood
MLNLG	Mixed Linear/Nonlinear Gaussain
MPC	Model Predictive Control
PF	Particle Filter
PG	Particle Gibbs
PGAS	Particle Gibbs Ancestral Sampler
PIMH	Particle Independent Metropolis-Hastings

PMCMC	Particle Markov Chain Monte Carlo
PMMH	Particle Marginal Metropolis-Hastings
PS	Particle Smoother
PSAEM	Particle SAEM
RB	Rao-Blackwellized
RBPF	Rao-Blackwellized Particle Filter
RBPS	Rao-Blackwellized Particle Smoother
RTS	Rauch-Tung-Striebel
SAEM	Stochastic Approximation EM
SIR	Sequential Importance Resampler
SIS	Sequential Importance Sampler
SLAM	Simultaneous Localization and Mapping
SMC	Sequential Monte Carlo
UKF	Unscented Kalman Filter
UT	Unscented Transform

1

Introduction

This thesis deals with the concept of nonlinear estimation. Estimation in general is the process of inferring information about some unknown quantity through some form of direct or indirect measurements. It is an important topic in itself, but also serves as a central part of other topics such as control, where knowledge of the current state of a process is crucial for determining the correct control actions. This thesis will mainly deal with the estimation of discrete time dynamical systems. The rest of this chapter will introduce some basic concepts and common algorithms used within this field, for a more thorough introduction see [Särkkä, 2013] and [Lindsten and Schön, 2013].

1.1 Theoretical background

In the context of estimation, the concept of filtering refers to making estimates of a quantity at time t using information up to, and including, time t. Smoothing on the other hand refers to estimates using future information as well. The following notation is used in this thesis; $x_{t|T}$ where x denotes an estimate of the unknown quantity at time t and T is the time up to which information is used to form the estimate. So for a filtering problem $T = t$ and for a smoothing problem $T > t$.

An important class of problems that has been extensively studied are linear time-varying Gaussian (LTV) models. Such a system can be described as

$$x_{t+1} = A_t x_t + f_t + v_t \tag{1.1a}$$

$$y_t = C_t x_t + g_t + e_t \tag{1.1b}$$

$$\begin{pmatrix} v_t \\ e_t \end{pmatrix} \sim N\left(0, \begin{bmatrix} R_{xx,t} & R_{xy,t} \\ R_{xy,t}^{\mathrm{T}} & R_{yy,t} \end{bmatrix}\right). \tag{1.1c}$$

Here the states x_t occur linearly and f_t and g_t are known offsets. All the uncertainty in the system is described by the additive Gaussian noises v_t and

e_t. This class of problems is important because there exist analytical solutions to both the filtering and smoothing problem for this model, most famously the Kalman filter (KF) [Kalman, 1960]. The KF provides an algorithm for recursively computing the filtered estimate, and is summarized in Algorithm 1 where the recursive structure is evident in that the estimate is successively updated with new data from each measurement. At each time the posterior distribution of x can be completely captured by a normal distribution, that is $x_{t|t} \sim N(\bar{x}_{t|t}, P_{t|t})$. The smoothing problem can also be solved analytically by for example a Rauch-Tung-Striebel (RTS) smoother [Rauch et al., 1965]. The RTS smoother is summarized in Algorithm 2.

Algorithm 1 Kalman filter (for the case where $R_{xy} = 0$)

Initialize $\hat{x}_{0|0}, P_{0|0}$ from prior knowledge
for $t \leftarrow 0$ **to** $T - 1$ **do**
Predict step
Compute $\bar{x}_{t+1|t} = A_t \bar{x}_{t|t} + f_t$
Compute $P_{t+1|t} = A_t P_{t|t} A_t^{\mathrm{T}} + R_{xx,t}$
Update step
Let $S = C_{t+1} P_{t+1|t} C_{t+1}^{\mathrm{T}} + R_{yy,t+1}$
Compute $\bar{x}_{t+1|t+1} = \bar{x}_{t+1|t} + P_{t+1|t} C_{t+1}^{\mathrm{T}} S^{-1} (y_t - C_{t+1} \bar{x}_{t+1|t})$
Compute $P_{t+1|t+1} = P_{t+1|t} - P_{t+1|t} C_{t+1}^{\mathrm{T}} S^{-1} C_{t+1} P_{t+1|t}$

Algorithm 2 Rauch-Tung-Striebel smoother.

Initialize $x_{T|T}, P_{T|T}$ with estimates from Kalman filter
for $t \leftarrow T - 1$ **to** 0 **do**
Compute $\bar{x}_{t|T} = \bar{x}_{t|t} + P_{t|t} A_t^{\mathrm{T}} P_{t+1|t}^{-1} (\bar{x}_{t+1|T} - \bar{x}_{t+1|t})$
Compute $P_{t|T} = P_{t|t} + P_{t|t} A_t^{\mathrm{T}} P_{t+1|t}^{-1} (P_{t+1|T} - P_{t+1|t}) P_{t+1|t}^{-1} A_t P_{t|t}$

Unfortunately many interesting real world problems do not fall into the LTV class. A more general model formulation is

$$x_{t+1} = f(x_t, v_t, t) \tag{1.2a}$$

$$y_t = g(x_t, e_t, t), \tag{1.2b}$$

where f, g are arbitrary nonlinear functions and the noise v_t, e_t can be from arbitrary distributions. To be able to reuse the theory and methods from the LTV case a number of approximations have been proposed.

The Extended Kalman Filter (EKF), see for example [Julier and Uhlmann, 2004], provides an approximate solution by linearizing the func-

tions f, g around the current state estimate x_t and assumes that the noise is additive and Gaussian, which allows the application of the recursion from the normal Kalman Filter.

Another approach is the Unscented Kalman Filter (UKF) [Wan and Van Der Merwe, 2000] which assumes that the resulting estimates will be from a Gaussian distribution where the mean and covariance are computed using the Unscented Transform (UT) [Julier and Uhlmann, 2004]. The UT works by deterministically sampling points from the distribution before the nonlinearity, the so called Sigma points. These are then propagated through the nonlinear functions, after which the mean and covariance of the resulting distribution are recovered.

Both of these approaches are fundamentally limited by the fact that the resulting posterior distributions are assumed to be Gaussian. A major limitation of modeling the posterior distribution as a Gaussian is that it assumes the posterior to be unimodal, whereas many problem have multimodal distributions. Consider for example a scenario where it is possible to measure only the absolute value of an unknown variable; even if the prior belief of that variable is Gaussian the resulting posterior-probability function is multimodal as shown in Figure 1.1.

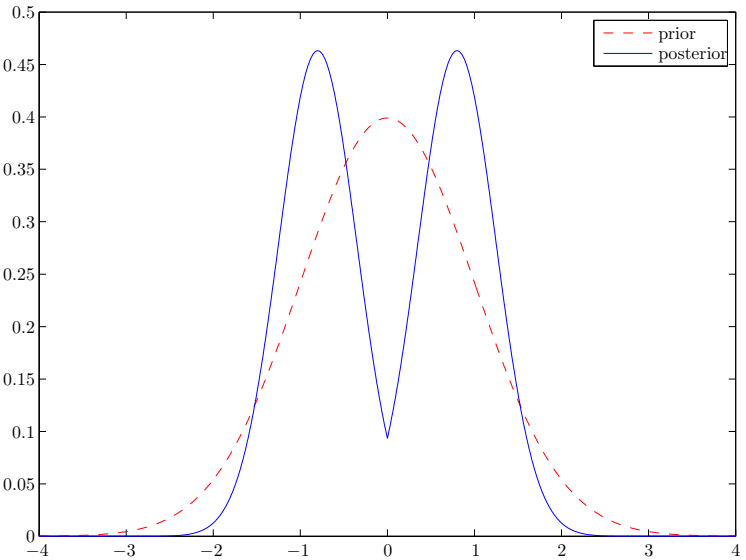

Figure 1.1 Prior probability density for an unknown variable $x \sim N(0, 1)$ and posterior after measuring $1 = |x| + e,\ e \sim N(0, 0.5)$. The prior is the dashed red line and the posterior the solid blue line.

1.2 Particle filtering

Particle Filters (PF) [Gordon et al., 1993], or Sequential Monte Carlo methods (SMC) [Liu and Chen, 1998], use a fundamentally different approach to the estimation problem; instead of assuming a pre-specified form of the posterior distribution a weighted point mass distribution is used,

$$p(x) \approx \sum_{i=1}^{N} w^{(i)} \delta(x - x^{(i)}). \tag{1.3}$$

The samples $x^{(i)}$ are typically referred to as particles which explains the name of the method, each particle is associated with a corresponding weight $w^{(i)}$, where $\sum_{i=1}^{N} w^{(i)} = 1$. The estimated mean of x could then be computed as $\bar{x} = \sum_{i=1}^{N} w^{(i)} x^{(i)}$.

The Particle Filter propagates this estimate forward in time by sampling new particles $\{x_{t+1}^{(i)}\}_{i=1..N}$ based on the values of $\{x_t^{(i)}\}_{i=1..N}$. The PF accomplishes this by sampling the particles from a proposal distribution $q(x_{t+1}|x_t^{(i)}, y_{t+1})$. The estimates are then reweighted to represent the desired probability density $p(x_{t+1}|y_{1:t+1})$, because of this it is also often referred to as a Sequential Importance Sampler (SIS).

The proposal density q is a design choice, $q = p(x_{t+1}|x_t)$ is commonly used and the resulting filter is referred to as a bootstrap Particle Filter. This simplifies the computations because when calculating the new weights for this choice the expression simplifies to $w_t^{(i)} p(y_{t+1}|x_{t+1}^{(i)})$. It is, however, important for the performance of the filter that the new particles are concentrated to areas of high probability, so depending on the nature of the particular problem to be solved and the constraints imposed by the mathematical model other choices such as $q = p(x_{t+1}|y_{t+1})$ can provide better performance.

Crucial for the PF to work in practice is that the particles are resampled, which is a method for discarding samples in regions of low probability. The algorithm is then referred to as a Sequential Importance Resampler (SIR) algorithm instead of SIS. The resampling step is important since without it the weights for all particles except one will eventually tend towards zero, a phenomenon referred to as particle depletion or degeneracy. The resampling step, however, increases the variance of the estimates and it is thus desirable to only perform it when necessary. An often used criterion for determining this is the so-called number of *effective particles*, which is computed as

$$N_{\text{eff}} = \frac{1}{\sum_{i=1}^{N} (w^{(i)})^2}. \tag{1.4}$$

The Particle Filter with resampling triggered by the number of effective particles is summarized in Algorithm 3.

Algorithm 3 Particle Filter algorithm using the effective number of particles as resampling criteria. γ_{res} is a design variable that specifies the threshold for the ratio of effective particles and the total number of particles that will trigger the resampling.

Draw $x_0^{(i)}$ from $p(x_0), \quad i \in 1..N$
Set $w_0^{(i)} = \frac{1}{N}, \quad i \in 1..N$
for $t \leftarrow 0$ **to** $T-1$ **do**
  Calculate $N_{\text{eff}} = (\sum_{i=1}^{N} {w_t^{(i)}}^2)^{-1}$
  if $(N_{\text{eff}} < \gamma_{\text{res}} N)$ **then**
    Draw new ancestor indices a_i from the categorical distribution defined by $\{w_t^{(i)}\}_{i=1..N}$
    Set $w_t^{(i)} = \frac{1}{N}, \quad i \in 1..N$
  else
    Set $a_i = i, i \in 1..N$.
  for $i \leftarrow 1$ **to** N **do**
    Sample $x_{t+1}^{(i)}$ from $q(x_{t+1}|x_t^{(a_i)}, y_{t+1})$
    Set $\hat{w}^{(i)} = w_t^{(a_i)} p(y_{t+1}|x_{t+1}^{(i)}) p(x_{t+1}^{(i)}|x_t^{(a_i)}) / q(x_{t+1}^{(i)}|x_t^{(a_i)}, y_{t+1})$
  Normalize weights, $w_{t+1}^{(i)} = \hat{w}^{(i)} / \sum_{j=1}^{N} \hat{w}^{(j)}$

The number of particles needed when using a PF typically grows exponentially with the dimension of the state-space as discussed in [Beskos et al., 2014] and [Rebeschini and Handel, 2013]. This scaling property makes it interesting to try to reduce the dimension of the problem. A common approach for this is to use so-called Rao-Blackwellized Particle Filters (RBPF) [Schön et al., 2005]. These split the state-space into two parts, $x^T = \begin{pmatrix} \xi^T & z^T \end{pmatrix}$. The first part, ξ, is the nonlinear estimation problem and the second part, z, becomes a LTV-system when conditioned on ξ. In general such a model can be written as

$$\xi_{t+1} = \begin{pmatrix} f_\xi^n(\xi_t, v_\xi^n) \\ f_\xi^l(\xi_t) \end{pmatrix} + \begin{pmatrix} 0 \\ A_\xi(\xi_t) \end{pmatrix} z_t + \begin{pmatrix} 0 \\ v_\xi^l \end{pmatrix} \tag{1.5a}$$

$$z_{t+1} = f_z(\xi_t) + A_z(\xi_t) z_t + v_z \tag{1.5b}$$

$$y_t = \begin{pmatrix} h_\xi(\xi_t, e^n) \\ h_z(\xi_t) \end{pmatrix} + \begin{pmatrix} 0 \\ C(\xi_t) \end{pmatrix} z_t + \begin{pmatrix} 0 \\ e^l \end{pmatrix} \tag{1.5c}$$

$$v_\xi^l \sim N(0, Q_\xi(\xi_t)), \; v_z \sim N(0, Q_z(\xi_t)), \; e^l \sim N(0, R(\xi_t)), \tag{1.5d}$$

where the key importance is that the parts of the model where the z-states appear are all affine in z and with additive Gaussian noise. For the remain-

ing part of the model the noise can be from any distribution and is not restricted to occur only additively. The Particle Filter is then only used to find the solution to $\xi_{1:T}$ and the estimate of $z_{1:T}$ is computed using a Kalman Filter. Interesting to note is that this becomes a non-Markovian problem when marginalizing over the z-states, i.e the values of ξ_{t+1} depend not only on (ξ_t, y_{t+1}) but on the whole history $(\xi_{1:t}, y_{1:t})$. Fortunately from an implementation point of view it is sufficient to store the filtered statistics of the z-state to capture the influence of the past states, i.e we can write the transition density as $p(\xi_{t+1}|\xi_{1:t}, y_{1:t}) = p(\xi_{t+1}|\xi_t, \bar{z}_t, P_t)$.

Importance of Proposal Distribution

The choice of proposal distribution, $q(x_{t+1}|x_t, y_{t+1})$, can have a large influence of the performance of the filter. The bootstrap Particle Filter uses $q = p(x_{t+1}|x_t)$, but if the state dynamics are uncertain, as is the case in for example system identification applications, this can result in a poor approximation.

If the model allows it the optimal choice is $q = p(x_{t+1}|x_t, y_{t+1})$, the corresponding particle weights are then computed as $w_{t+1}^{(i)} = w_t^{(i)} p(y_{t+1}|x_t^{(i)})$. In the case were the measurements are more informative than the state dynamics, the choice $q = p(x_{t+1}|y_{t+1})$ can provide good performance. However, in many cases it is not possible to directly sample from this distribution.

The Auxiliary Particle Filter (APF) [Pitt and Shephard, 1999] is a variant of the regular PF where the particles are resampled by first reweighting them according to $w_t^{(i)} p(y_{t+1}|x_t^{(i)})$ to attempt to ensure that all particles end up in regions of high probability. Since $p(y_{t+1}|x_t^{(i)})$ typically is not readily available it is often replaced by an approximation, such as using the predicted mean, e.g $p(y_{t+1}|\bar{x}_{t+1}^{(i)})$.

To illustrate these different approaches they are compared on a single realization of the trivial example of an integrator with additive Gaussian noise,

$$x_{t+1} = x_t + v_t, \; v_t \sim N(0,1) \tag{1.6a}$$

$$y_t = x_t + e_t, \; e_t \sim N(0,1) \tag{1.6b}$$

$$x_0 \sim N(0,1). \tag{1.6c}$$

The resulting particle estimates are shown in Figure 1.2 for the bootstrap Particle Filter ($q = p(x_{t+1}|x_t)$), Figure 1.3 for the case using the optimal proposal distribution ($q = p(x_{t+1}|x_t, y_{t+1})$), Figure 1.4 using $q = p(x_{t+1}|y_{t+1})$ and Figure 1.5 with an APF using $p(y_{t+1}|\bar{x}_t)$. In the end the choice of proposal distribution is often dictated by the model, but when possible it is desirable to use a proposal as close to the optimal proposal as possible.

1.3 Particle smoothing

Since each particle in a Particle Filter is influenced by its past trajectory, both in the value $x^{(i)}$ and its weight $w^{(i)}$, it actually provides a collection of weighted trajectories. Each trajectory is the ancestral path of one of the particles at time t. The ancestral path is the trajectory we obtain by taking the ancestor of each particle and iterating this procedure backwards in time. The Particle Filter thus actually targets $p(x_{1:t}) \approx \sum_{i=0}^{N} w_t^{(i)} \delta(x_{1:t} - x_{1:t}^{(i)})$ where $x_{1:t}^{(i)}$ is the ancestral path of particle i at time t. Discarding $x_{1:t-1}$ as is typically done corresponds to a marginalization over those states. In theory the Particle Filter is therefore actually a smoothing algorithm. However, due to the (necessary) resampling step, it provides a very poor approximation of the true probability density function since in practice for $t \ll T$ all the particles share a common ancestor, thus reducing the PF estimate to a single point estimate. This is commonly referred to as path degeneracy. An example of this is shown in Fig. 1.2 where it can be seen that the ancestral paths are severely degenerate for $t \ll T$.

To provide better smoothing performance than that of the Particle Filter a number of different methods have been proposed, they are all commonly referred to as particle smoothers. A very common approach is the so-called Forward Filter Backward Simulator (FFBSi) [S. J. Godsill et al., 2004]. It is a method which reuses the particles from the filter but creates new smoothed trajectories. It iterates through the forward estimates backward in time, drawing new ancestors for each particle while taking the likelihood of the future trajectory into consideration. FFBSi is summarized in Algorithm 4. Figure 1.6 revisits the example from the previous section this time showing the smoothed trajectories that were obtained using FFBSi.

Algorithm 4 Forward Filter Backwards Simulator, draws M trajectories from the smoothing distribution $p(x_{1:T}|y_{1:T})$.

Run particle filter forward in time generating $\{x_{t|t}^{(i)}, w_{t|t}^{(i)}\}_{i=1..N}, t \in 0..T$
Sample $\{x_{T|T}^{(j)}\}_{j=1..M}$ using the categorical distribution $\{w_{T|T}^{(i)}\}_{i=1..N}$
for $t \leftarrow T-1$ **to** 0 **do**
    for $j \leftarrow 1$ **to** M **do**
        Compute $\hat{w}_{t|T}^{(i)} = w_{t|t}^{(i)} p(x_{t+1|T}^{(j)}|x_{t|t}^{(i)}),\ i \in 1..N$
        Normalize $w_{t|T}^{(i)} = \hat{w}_{t|T}^{(i)} / (\sum_{k=1}^{N} \hat{w}_{t|T}^{(k)})$
        Sample a_j from the categorical distribution defined by $\{w_{t|T}\}_{i=1..N}$
        Set $x_{t|T}^{(j)} = x_{t|t}^{(a_j)}$

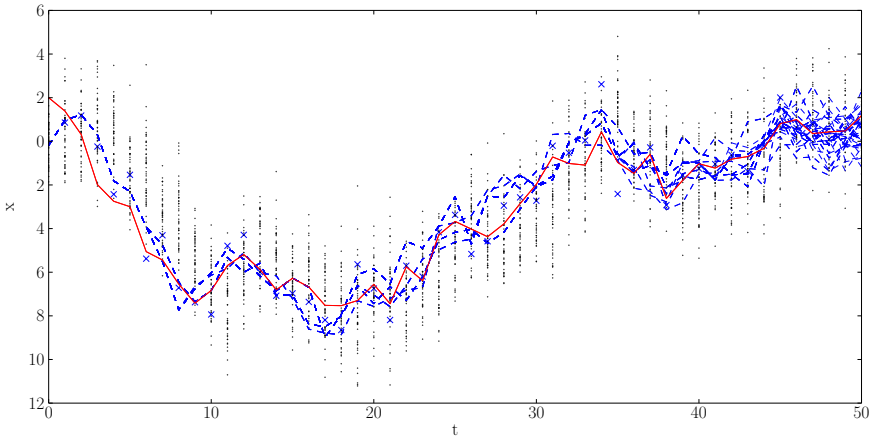

Figure 1.2 Example realization of model (1.6) estimated with a bootstrap Particle Filter. The solid red line is the true trajectory and the blue crosses are the measurements. The black points are the filtered particle estimates forward in time and the blue dashed lines are the ancestral paths of the particles at time T. The degeneracy of the ancestral paths can be seen clearly.

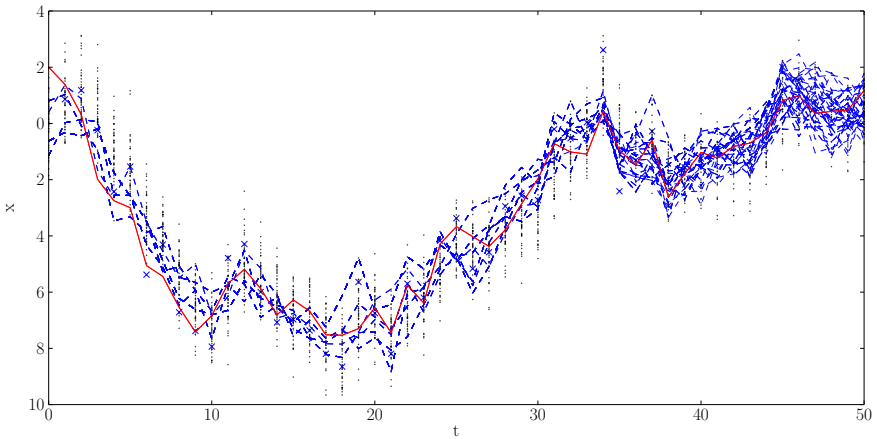

Figure 1.3 Example realization of model (1.6) estimated with a Particle Filter using the optimal proposal distribution. The notation is the same as in Figure 1.2. The problem of path degeneracy is not as big as in Figure 1.2 but still clearly present, for a longer dataset it would eventually collapse to a point estimate.

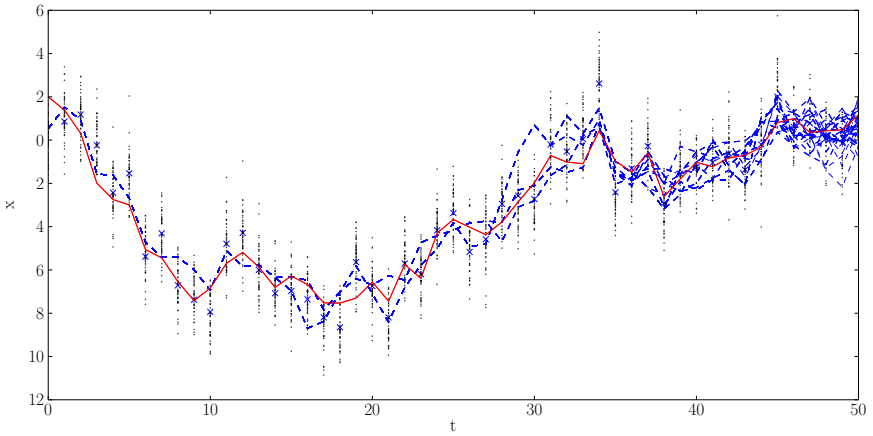

Figure 1.4 Example realization of model (1.6) estimated with a Particle Filter using $p(x_{t+1}|y_{t+1})$ as proposal distribution. The notation is the same as in Figure 1.2. Notice that all the particles occur symmetrically around the measurements, in comparison the particles in Figure 1.2 are centered around the predicted next state.

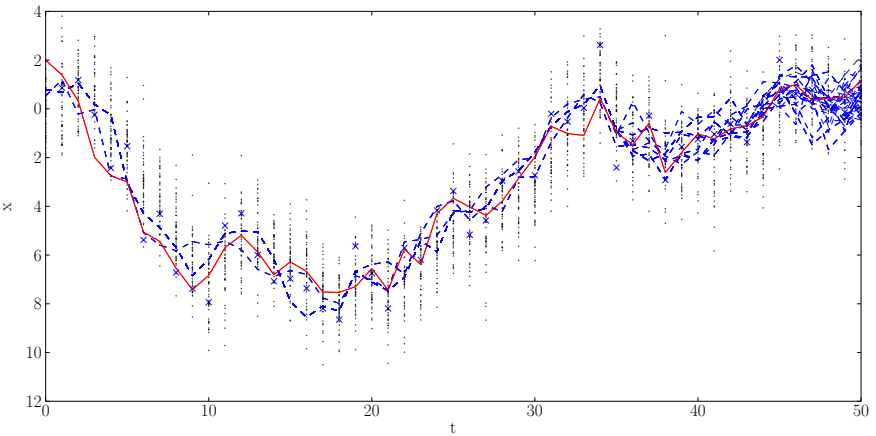

Figure 1.5 Example realization of model (1.6) estimated with an Auxiliary Particle Filter using $p(y_{t+1}|\bar{x}_{t+1})$. The notation is the same as in Figure 1.2. By attempting to predict which particles will be likely after the next measurement the APF improves the performance of the filter and slightly reduces the degenaracy compared to Figure 1.2.

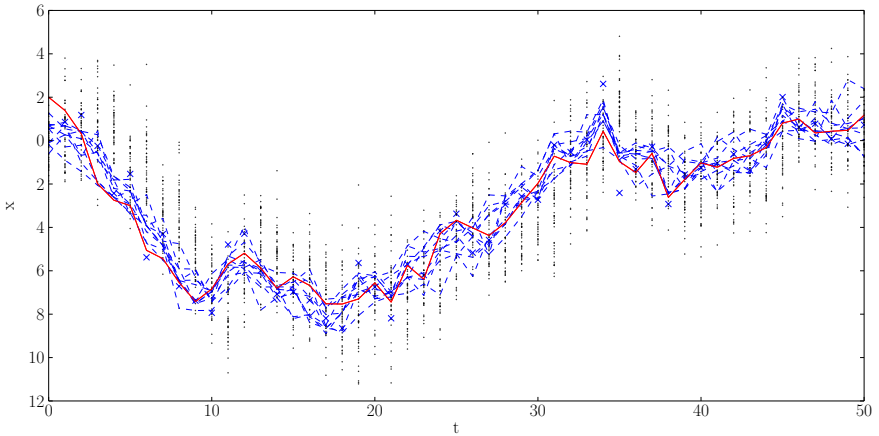

Figure 1.6 Example realization of model (1.6) estimated using a bootstrap Particle Filter combined with a FFBSi smoother using 10 backward trajectories. The notation is the same as in Figure 1.2 except for the ancestral paths which are replaced by the smoothed trajectories. The estimates using FFBSi are closer to the true trajectory when compared with the filters presented in Figure 1.2–1.5, also it can be seen that it does not suffer from path degeneracy for this example.

Looking at Algorithm 4 it is clear that the computational complexity of FFBSi scales as $\mathcal{O}(NM)$, where N is the number of particles used in the PF and M is the number of backward trajectories to be sampled. A number of different approaches have been proposed to improve this by avoiding the need to evaluate all the smoothing weights, $\{\hat{w}_{t|T}^{(i)}\}_{i=1..N}$. They include the use of rejection sampling [Doucet and Johansen, 2009] and using a Metropolis-Hastings sampler [Bunch and S. Godsill, 2013]. Further methods exist which not only reuse the forward particles but also sample new values of $x_{t|T}^{(i)}$ during the smoothing, one is the so called Metropolis-Hastings Improved Particle Smoother (MHIPS) [Dubarry and Douc, 2011], another the Forward Filter Backward Proposer (FFBP) [Bunch and S. Godsill, 2013].

In the same manner as for the Particle Filter it is of interest to reduce the dimension of the estimation problem by finding any states that represent a LTV-system when conditioned on the rest of the states. However, the resulting smoothing problem becomes more difficult to solve than the corresponding filtering problem since it becomes a non-Markovian problem. For an introduction to non-Markovian particle smoothing see [Lindsten and Schön, 2013]. There is no general solution to the marginalized smoothing problem where the computational effort required for each step does not grow with the

size of the full dataset. However, for mixed linear/nonlinear models with additive Gaussian noise (MLNLG) a method has been presented in [Lindsten et al., 2013] that propagates variables backwards during the smoothing in the same way that $\bar{z}_t$ and P_t are propagated forward in the RBPF. Another approach is that used in [Lindsten and Schön, 2011] where the solution to the smoothing problem is approximated by marginalizing over z for the filtering, but sampling the z-states during the smoothing step and later recovering the full distribution by running a constrained smoother for $z_{1:T}$ after obtaining the estimated trajectories for $\xi_{1:T}$

1.4 Parameter estimation

Another important estimation problem is that of finding the values of unknown parameters in a model. A key difference between parameters and states is that the former are constant over time whereas the latter typically are not. This property makes the problem of parameter estimation fundamentally different from that of state estimation, and much less suitable to be solved by particle filtering/smoothing. However, particle methods play an important role in several approaches for estimation of parameters in nonlinear models. By extending model (1.2) with a set of unknown parameters, θ, it now becomes

$$x_{t+1} = f(x_t, v_t, t, \theta) \tag{1.7a}$$
$$y_t = g(x_y, e_t, t, \theta). \tag{1.7b}$$

One approach that is sometimes used is to just include the unknown parameters as states in the model, but it leads to problems such as the parameters varying over time if they are modeled as for example a slowly drifting random walk process. If the parameters are assumed to be fixed the issues with particle depletion and path degeneracy lead to poor performance.

Often the quantity of interest is the maximum likelihood (ML) estimate of θ, that is

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\ p(y_{1:T}|\theta), \tag{1.8}$$

but it can also be of interest to find the full posterior distribution of the parameters, i.e $p(\theta|y_{1:T})$.

To compute the ML-estimate the Expectation Maximization (EM) algorithm [Dempster et al., 1977] is commonly used. It introduces the Q-function (1.9a) and alternates between computing the expected value (the E-step) and

finding a new estimate of θ using (1.9b) (the M-step).

$$Q(\theta, \theta_k) = \mathrm{E}_{X|\theta_k}[\mathrm{L}_\theta(X, Y|Y)] \tag{1.9a}$$

$$\theta_{k+1} = \operatorname*{argmax}_\theta Q(\theta, \theta_k) \tag{1.9b}$$

The EM-algorithm is a well studied approach for linear systems [Shumway and Stoffer, 1982][Gibson and Ninness, 2005] which finds the (local) ML-estimate. For nonlinear systems there is typically no analytic solution to the E-step, but it can be approximated using a particle smoother [Schön et al., 2011]. A drawback with this approach is that it requires a growing number of particles for the E-step for each new iteration of the algorithm [Lindsten and Schön, 2013].

When the estimation problem is computationally expensive an improvement to this approach is to use the Stochastic Approximation Expectation Maximization algorithm (SAEM) [Delyon et al., 1999] instead. It replaces (1.9a) with (1.10)

$$\widehat{Q}_k(\theta) = (1 - \gamma_k)\widehat{Q}_{k-1}(\theta) + \gamma_k \log p_\theta(x_{0:T}[k], y_{1:T}), \tag{1.10}$$

where $x_{0:T}[k]$ is the estimated trajectory of the system using θ_k. γ_k denotes the step size, which is a design parameter that must fulfill $\sum_{k=1}^{\infty} \gamma_k = \infty$ and $\sum_{k=1}^{\infty} \gamma_k^2 < \infty$. By combining this with a *Conditional Particle Filter* (CPF) [Lindsten et al., 2014] the Particle SAEM (PSAEM) [Lindsten, 2013] method is created, which avoids the need of increasing the number of particles for each iteration. Intuitively the conditioning on a previous trajectory retains information between the iterations allowing the gradual improvement of the quality of the estimate for a fixed number of particles. The use of a regular CPF, however, leads to slow mixing and therefore the use of *ancestral sampling* (CPFAS) was introduced in [Lindsten et al., 2014]. The CPFAS is summarized in Algorithm 5.

Another approach is that of the Particle Markov-Chain Monte Carlo (PMCMC) methods which, instead of computing the ML-estimate, sample from the posterior distribution $p(\theta|y_{1:T})$. This is accomplished by targeting the joint distribution $p(\theta, x_{1:T}|y_{1:T})$. Examples of such methods are Particle Marginal Metropolis-Hastings (PMMH) [Andrieu et al., 2010], Particle Gibbs (PG) [Andrieu et al., 2010] and the Particle Gibbs Ancestral Sampler (PGAS) [Lindsten et al., 2014]. The PMMH method is also known as Particle Independent Metropolis-Hastings (PIMH) when there is no unknown parameters and it thus solves a pure state-estimation problem.

Algorithm 5 Conditional Particle Filter with Ancestral Sampling (CPFAS). The conditional trajectory $x_{0:T}^{\text{cond}}$ is preserved throughout the sampling, but due to the ancestral sampling it is broken into smaller segments, thus reducing the degeneracy which otherwise leads to poor mixing when used as part of e.g. a Particle Gibbs algorithm.

Draw $x_0^{(i)}$ from $p(x_0)$, $\quad i \in 1..N-1$
Set $x_0^{(N)} = x_0^{\text{cond}}$ Set $w_0^{(i)} = \frac{1}{N}$, $\quad i \in 1..N$
for $t \leftarrow 0$ **to** $T-1$ **do**
  for $i \leftarrow 1$ **to** $N-1$ **do**
    Draw a_i with $P(a_i = j) \propto w_t^{(j)}$
    Sample $x_{t+1}^{(i)}$ from $q(x_{t+1}|x_t^{(a_i)}, y_{t+1})$
  Draw a_N with $P(a_N = i) \propto w_t^{(i)} p(x_{t+1}^{\text{cond}}|x_t^{(i)})$
  Set $x_{t+1}^{(N)} = x_{t+1}^{\text{cond}}$
  for $i \leftarrow 1$ **to** N **do**
    Set $\hat{w}^{(i)} = w_t^{(a_i)} p(y_{t+1}|x_{t+1}^{(i)}) p(x_{t+1}^{(i)}|x_t^{(a_i)}) / q(x_{t+1}^{(i)}|x_t^{(a_i)}, y_{t+1})$
  Normalize weights, $w_{t+1}^{(i)} = \hat{w}^{(i)} / \sum_j \hat{w}^{(j)}$

2

Implementation

The types of estimation methods presented in Chapter 1 can be implemented in a large number of ways, ranging from software implementations in a prototyping environment such as Matlab to dedicated hardware implementations and as a middle ground implementations running on Graphics Processing Units (GPUs). The specific target platform depends on both the application of the estimator and its performance requirements. For all cases it is of interest to be able to reuse generic parts of the implementation to reduce the time and effort needed for the development and to reduce the risk of introducing errors by relying on parts previously verified to work.

The remainder of this chapter gives a quick overview of existing software to assist in the application of these methods followed by a slightly more in-depth explanation of the pyParticleEst software that has been the foundation for the work performed in this thesis.

2.1 Related software

LibBI

LibBI [Murray, In review] is described as generating highly efficient code for performing Bayesian inference using either GPUs or multi-core CPUs. It uses a custom high level language to describe the problem and can then generate code for a number of different algorithms for solving that particular problem. It is distributed under the *CSIRO Open Source Software License*, which appears to be a derivative of the GNU General Public License version 2 (GPL v2) license.

Biips

Biips [Todeschini et al., 2014] uses the BUGS language to define a model and then solves the estimation problem using either particle filtering, PIMH or PMMH. It is licensed under GNU General Public License version 3 (GPL v3).

SMCTC

SMCTC: Sequential Monte Carlo Template Class [Johansen, 2009] is a C++ template library license under GNU General Public License v3 (GPL v3). It provides a set of low-level templates that can used for Sequential Monte Carlo methods.

Venture

Venture [Mansinghka et al., 2014] defines itself as a *higher-order probabilistic programming platform*. It defines a custom language with is used for describing the probabilistic models. It can perform inference using a number of methods including SMC. The authors currently describe it as "alpha quality research software". It is distributed under the GNU General Public License version 3 (GPL v3).

Anglican

Anglican [Wood et al., 2014] also defines a custom language for describing probabilistic models and it performs inference using PMCMC. It is distributed under the BSD license.

2.2 pyParticleEst

The work presented in this thesis has mainly revolved around a software developed by the author called pyParticleEst [Nordh, 2013]. It tries to provide a foundation for experimenting with different models and algorithms to allow the user to determine the most suitable setup for their problem.

The goal is not necessarily to provide the fastest possible implementation, but to have a very generic framework that is easy to extend and apply to new problems. To achieve this the code is split into parts that describe the algorithms and parts that describe the model for an estimation problem. The algorithmic parts operate on generic functions corresponding to different probabilistic operations on the mathematical models, such as sampling from distributions and evaluating probability densities. The model parts are then responsible for performing these operations in the correct way for that specific problem, thus achieving a separation between the generic algorithm parts and the problem specific parts. The interfaces in the library are structured to allow for estimation of non-Markovian problems, and the framework as such is not limited to state-space models. This is used for example when performing estimation using Rao-Blackwellized Particle Smoothers, but could also be used for other types of models such as Gaussian Processes as in for example [Lindsten and Schön, 2013, Chapter 4.1.3].

pyParticleEst is implemented in Python, hence the name. Python is a high level object oriented language, this is leveraged by representing the models as

Python classes. Each model class is then responsible for how the data for that particular problem is stored, and how to perform the necessary computations on that data. This requires a bit more programming knowledge from the end user compared to the approach taken by some of the softwares listed in Section 2.1, which instead take the approach of specifying the problems in a high-level language. On the other hand this allows the programmer more flexibility to perform optimizations and for example identify operations that could be performed in parallel. To reduce the effort needed by the user, a set of predefined classes that describe some common types of models are provided. These classes are then extended and specialized by the user through the inheritance mechanism in Python. This is also used to hide the additional details needed for non-Markovian smoothing when operating on standard state-space models.

The choice of Python as language combined with licensing the library as LGPL[FSF, 1999] allows pyParticleEst to be used on a wide-variety of systems without the need to pay any licensing fees for either the library itself or any of its dependencies. This also allows it to be integrated into proprietary solutions. The intended main use-case for pyParticleEst is as a prototyping tool for testing new algorithms and models, to allow quick comparison of different approaches and to serve as a reference to compare against when developing an optimized problem-specific implementation, whether that is supposed to run on a regular PC, GPU or dedicated hardware.

2.3 Discussion

As is evident from the previous sections there is a lot of recent activity in the research community when it comes to the development of generic frameworks for working within this relatively new field of estimation methods. This supports the importance of this work, but it also illustrates the variety of approaches that can be taken, depending on which level of insight of the internal workings of the algorithms the user wants and is expected to have.

The classic area of linear time-invariant systems allows a very easy representation where the user can just define the matrices that describe the model, for the more general type of models that are the target for these softwares the choice is not as trivial. Both SMCTC and pyParticleEst take the approach of letting the user define the model programmatically, which allows, and requires, more insight into the actual implementation of the algorithms. The other softwares use a high-level language closer to mathematical notation which could make them easier to use for users with little previous programming experience. The approaches could of course be combined by having a parser which takes a model specification in some high-level language and generates code for one of the lower level softwares.

3

Application areas

This section gives a brief introduction to a few areas where nonlinear estimation plays an important role and in which the author has performed research.

3.1 System identification and modeling of periodic signals

System identification is the process of obtaining a mathematical model describing an, in some sense, unknown system. The goal is to produce a model suitable for some particular purpose, since it in general is impossible to find a model that completely describes the true system [Ljung, 1999, Chapter 1]

One area where models play an important role is in control system design. Models are used both indirectly as part of the design phase and directly as is the case for Model Predictive Control (MPC). In MPC the controller has an internal model of the system and the control actions are determined by the solution of an optimization problem based on the internal model [Clarke, 1987]. Naturally it is important that the internal model provides a good approximation of the true system.

System identification methods can be categorized into two classes, black-box and grey-box. Black-box models require no prior knowledge of the structure of the model, whereas grey-box methods identify parameters in a previously defined structure. The model structure can for example be obtained through the use of physical principles such as Newton's laws of motion.

For a linear system a black-box model could for example be a canonical state-space representation [Ljung, 1999, Chaper 4], whereas a grey-box model would have a structure where the states represent well known physical quantities such as acceleration and velocity. The knowledge that the acceleration is the derivative of the velocity is then readily apparent from the structure of the model. For nonlinear systems black-box modeling could be accomplished with for example artificial neural networks [Bishop, 2006, Chapter 5]

or Gaussian processes [Rasmussen and Williams, 2005] [Lindsten and Schön, 2013, Chapter 4.1.3].

For linear models grey-box identification can be performed by using the Expectation Maximization (EM) algorithm [Dempster et al., 1977][Gibson and Ninness, 2005] to calculate the maximum likelihood estimate of the unknown parameters given the observed data. For nonlinear models the EM-algorithm can also be used with some modifications as described in Section 1.4.

Models used for control describe how an input will affect the future output of a system, it is, however, also of interest to describe autonomous systems. These are systems that are not affected by any (known) inputs. One example of their use is in the modeling of periodic signals. In [Wigren and Söderström, 2005] it is shown that any periodic signal where the phase-plane, that is the plot of $(y, \dot{y})$, does not intersect itself can be modeled as the output of a (nonlinear) second order system. If the phase-plane intersects itself it implies that $\ddot{y}$ can not be expressed as a function of $(y, \dot{y})$, thus intuitively showing the necessity of the condition, for the sufficiency part the reader is referred to the original paper.

Obtaining such a model can provide insight into the behavior that might not be immediately obvious from studying the signal directly, it could also provide a method for compressing the amount of data needed to describe a periodic signal.

3.2 Indoor navigation

The topic of navigation is commonly broken into three areas: Localization, Mapping and the combination of the former which is commonly referred to as Simultaneous Localization, and Mapping (SLAM).

Localization deals with determining the position of an agent inside a previously charted environment, it could for example be pedestrians that with the help of a phone and the strength of the WiFi-signals determine their positions inside a shopping center, or it could be a robot working on a factory floor using laser range finders to measure the distance to known walls and other obstacles.

Mapping is the process of creating a map for an unknown environment based on observations from known positions. For the shopping center example above it could correspond to measuring the WiFi-signal strengths at regular positions inside the center where the position of each measurement is determined by e.g. physically measuring distances to the wall.

SLAM combines the two previous categories and deals with creating a map of an unknown environment while at the same time relying on that map for determining the position of the mapping agent. As can be intuitively

expected this a much more challenging problem than performing either of the tasks in isolation.

Maps

A map could be any collection of information that assists in the positioning problem and is very application specific, there are, however, some common types, two of which we will discuss in more detail. For an overview of different modeling approaches see [Burgard and Hebert, 2008]

Landmark maps are collections of so called features, or landmarks, parts of the environment that are easily identifiable. A landmark map stores the location of each such feature along with the information to recognize it when observed. Whenever such a landmark is later observed it can be used to infer information about the current position of the agent. This approach could be used for the shopping center example above, each landmark would then be a WiFi access point, and the observations would be the received signal strength in the mobile phone for the signals from each access point. This utilizes the fact the signal strength among other things depend on the distance to the signal source. For an example of landmark maps see [Nordh, 2007] and [Alriksson et al., 2007]. A famous algorithm that uses landmark maps is the FastSLAM algorithm [Montemerlo et al., 2002].

A different type of map is the so called Occupancy Grid, it models the world as a regularly spaced array, and each cell stores its probability of being occupied. The observations are then made by observing if a cell is occupied or empty. A drawback with this type of map is that for a two-dimensional map the number of cells required increases quadratically with the desired resolution and quickly becomes very large if there is a need to map a larger area. Figure 3.1 shows an example of an occupancy grid. The occupancy probability, p, is typically encoded using the log-odds ratio which maps $[0, 1] \to [-\infty, \infty]$ using the following relation

$$p_{\text{logodds}} = \log \frac{p}{1-p} \tag{3.1}$$

For a detailed introduction to occupancy grid mapping see [Thrun et al., 2005, Chapter 9].

Range finder sensors

Any number of sensors could be used for localization, for example the strength of received radio signals discussed before. This section will, however, focus on so called range finder sensors, which are typically based on either light or sound. For indoor navigation laser-based range finders are very popular since they provide very high accuracy, with the drawback that they are fairly expensive. They emit short pulses of light along a narrow

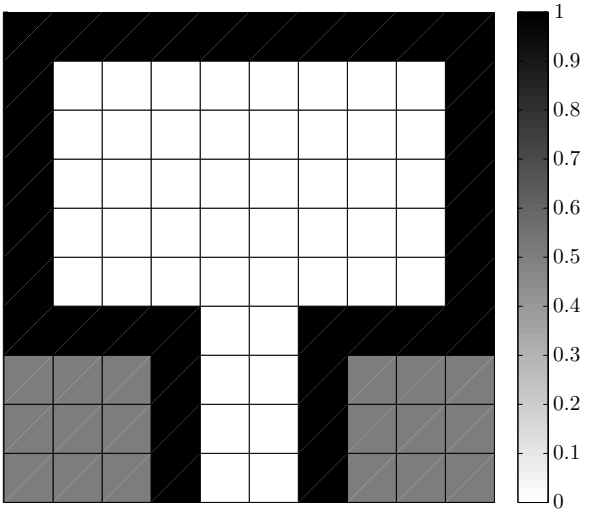

Figure 3.1 Example of an occupancy grid, the probability of being occupied is here encoded as the intensity, where black corresponds to 1 and white to 0. With some imagination this map could represent a room with an opening out to a corridor in the wall in the bottom of the map. They grey parts would correspond to areas that have not been observed and are therefore equally likely to be occupied or unoccupied.

beam and measures the time until the reflection is received, corresponding to the distance to the nearest object hit by the laser beam. This typically provides both a very accurate range measurement and a small angular uncertainty. The angular uncertainty is the accuracy with which the direction of the reflected light can be determined. Ultrasonic range finders have almost the opposite characteristics compared to a laser range finder; they are cheap but also suffer from high angular uncertainty, not as good range accuracy and they are less reliable due to the sound pulses being reflected away instead of back towards the sender when the angle of incidence increases. The same phenomenon occurs for laser range finders and mirrors, but for ultra sound this occur for almost all surfaces in a normal room. This issue is illustrated in Figure 3.2. These characteristics make ultrasonic range sensors more challenging to use for localization and/or mapping compared to laser range finders, but the lower costs makes them attractive for applications were their performance is sufficient.

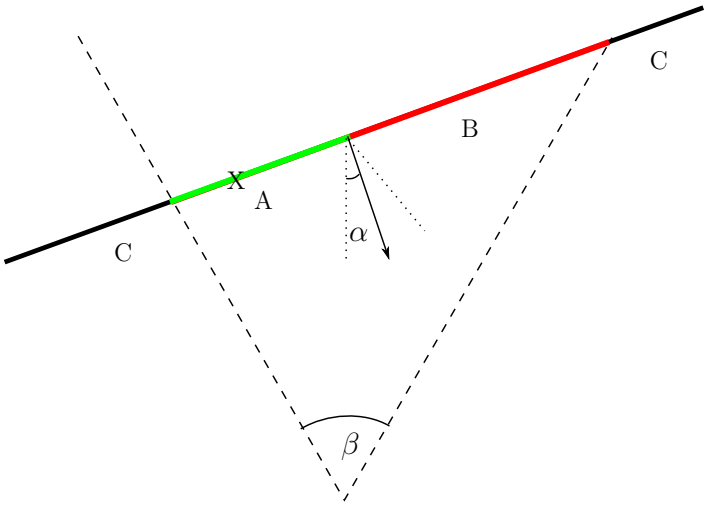

Figure 3.2 Properties of ultrasonic sensors. The sensor has an opening angle, β, which determines the size of the area that is observed. The emitted sound only reflects back to the sensor if the angle of incidence is less than α, otherwise it reflects away from the object effectively making it invisible to the sensor. The figure illustrates a straight wall, the sections marked C are outside the field of view, the section marked B is inside the field of view but the angle of incidence is so large that it is not visible to the sensor. The section marked A is inside the field of view. Also, the angle of incidence is steep enough for it to be detected by the sensor. The range returned by the sensor will be the distance to the closest point within section A, marked with an X. For this example it coincides with the actual closest point, but for more complex geometries this is not always the case. If section A of the wall was removed the sensor would not detect anything, even though sections B and C are still present.

4

Discussion

The thesis presents a collection of work that has grown from the desire to write a somewhat generic software implementation when working on the indoor navigation problem discussed in Paper VII and Paper VIII. From the start the special case of mixed linear/nonlinear Gaussian models has been an important part of the software. When expanding the framework with additional methods some minor gaps in the published methods had to be filled in order for all the algorithms to work with the MLNLG models, these extensions are presented in Paper V and Paper VI.

Having an existing software that could easily solve the same problem using multiple different methods provided an interesting opportunity to compare the performance of the different algorithms, this is the topic of Paper II. This also provided a good foundation for attempting to solve new problems, which was exploited in Paper III and Paper IV. Finally Paper I describes the actual software framework implementation, which is the foundation on which everything else in this thesis builds.

Bibliography

Alriksson, P., J. Nordh, K.-E. Årzén, A. Bicchi, A. Danesi, R. Sciadi, and L. Pallottino (2007). "A component-based approach to localization and collision avoidance for mobile multi-agent systems". In: *Proceedings of the European Control Conference*. Kos, Greece.

Andrieu, C., A. Doucet, and R. Holenstein (2010). "Particle Markov chain Monte Carlo methods". *Journal of the Royal Statistical Society, Series B* **72**:2, pp. 1–33.

Beskos, A., D. Crisan, A. Jasra, et al. (2014). "On the stability of sequential Monte Carlo methods in high dimensions". *The Annals of Applied Probability* **24**:4, pp. 1396–1445.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA. ISBN: 0387310738.

Bunch, P. and S. Godsill (2013). "Improved particle approximations to the joint smoothing distribution using Markov Chain Monte Carlo". *Signal Processing, IEEE Transactions on* **61**:4, pp. 956–963.

Burgard, W. and M. Hebert (2008). "World modeling". English. In: Siciliano, B. et al. (Eds.). *Springer Handbook of Robotics*. Springer Berlin Heidelberg, pp. 853–869. ISBN: 978-3-540-23957-4. DOI: 10.1007/978-3-540-30301-5_37. URL: http://dx.doi.org/10.1007/978-3-540-30301-5_37.

Clarke, D. (1987). "Generalized predictive control Part I. — The basic algorithm". *Automatica* **23**:2, pp. 137–148.

Delyon, B., M. Lavielle, and E. Moulines (1999). "Convergence of a stochastic approximation version of the EM algorithm". *The Annals of Statistics* **27**:1, pp. 94–128.

Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). "Maximum likelihood from incomplete data via the EM algorithm". *Journal of the Royal Statistical Society B* **39**:1, pp. 1–38.

Doucet, A. and A. M. Johansen (2009). "A tutorial on particle filtering and smoothing: fifteen years later". *Handbook of Nonlinear Filtering* **12**, pp. 656–704.

Dubarry, C. and R. Douc (2011). "Particle approximation improvement of the joint smoothing distribution with on-the-fly variance estimation". *arXiv preprint arXiv:1107.5524*.

FSF (1999). *The GNU Lesser General Public License*. See `http://www.gnu.org/copyleft/lesser.html`.

Gibson, S. and B. Ninness (2005). "Robust maximum-likelihood estimation of multivariable dynamic systems". *Automatica* **41**:10, pp. 1667–1682.

Godsill, S. J., A. Doucet, and M. West (2004). "Monte Carlo smoothing for nonlinear time series". *Journal of the american statistical association* **99**:465.

Gordon, N., D. Salmond, and A. F. M. Smith (1993). "Novel approach to nonlinear/non-Gaussian Bayesian state estimation". *Radar and Signal Processing, IEE Proceedings F* **140**:2, pp. 107–113. ISSN: 0956-375X.

Johansen, A. M. (2009). "SMCTC: sequential Monte Carlo in C++". *Journal of Statistical Software* **30**:6, pp. 1–41. ISSN: 1548-7660. URL: `http://www.jstatsoft.org/v30/i06`.

Julier, S. and J. Uhlmann (2004). "Unscented filtering and nonlinear estimation". *Proceedings of the IEEE* **92**:3, pp. 401–422. ISSN: 0018-9219. DOI: `10.1109/JPROC.2003.823141`.

Kalman, R. E. (1960). "A new approach to linear filtering and prediction problems". *Journal of Basic Engineering* **82**:1, pp. 35–45.

Lindsten, F. (2013). "An efficient stochastic approximation EM algorithm using conditional particle filters". In: *Proceedings of the 38th International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. Vancouver, Canada.

Lindsten, F. and T. Schön (2011). *Rao-Blackwellized Particle Smoothers for Mixed Linear/Nonlinear State-Space Models*. Tech. rep. URL: `http://user.it.uu.se/~thosc112/pubpdf/lindstens2011.pdf`.

Lindsten, F., P. Bunch, S. J. Godsill, and T. B. Schön (2013). "Rao-Blackwellized particle smoothers for mixed linear/nonlinear state-space models". In: *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, pp. 6288–6292.

Lindsten, F., M. I. Jordan, and T. B. Schön (2014). "Particle Gibbs with ancestor sampling". *The Journal of Machine Learning Research* **15**:1, pp. 2145–2184.

Lindsten, F. and T. Schön (2013). "Backward simulation methods for Monte Carlo statistical inference". *Foundations and Trends in Machine Learning* **6**:1, pp. 1–143. ISSN: 1935-8237. DOI: 10.1561/2200000045. URL: http://dx.doi.org/10.1561/2200000045.

Liu, J. S. and R. Chen (1998). "Sequential Monte Carlo methods for dynamic systems". *Journal of the American statistical association* **93**:443, pp. 1032–1044.

Ljung, L., (Ed.) (1999). *System Identification (2Nd Ed.): Theory for the User*. Prentice Hall PTR, Upper Saddle River, NJ, USA. ISBN: 0-13-656695-2.

Mansinghka, V. K., D. Selsam, and Y. N. Perov (2014). "Venture: a higher-order probabilistic programming platform with programmable inference". *CoRR* **abs/1404.0099**. URL: http://arxiv.org/abs/1404.0099.

Montemerlo, M., S. Thrun, D. Koller, B. Wegbreit, et al. (2002). "FastSLAM: a factored solution to the simultaneous localization and mapping problem". In: *AAAI/IAAI*, pp. 593–598.

Murray, L. M. (In review). "Bayesian state-space modelling on high-performance hardware using LibBi". URL: http://arxiv.org/abs/1306.3277.

Nordh, J. (2007). *Ultrasound-based Navigation for Mobile Robots*. Master's Thesis ISRN LUTFD2/TFRT--5789--SE. Department of Automatic Control, Lund University, Sweden.

Nordh, J. (2013). *PyParticleEst - Python library for particle based estimation methods*. URL: http://www.control.lth.se/Staff/JerkerNordh/pyparticleest.html.

Pitt, M. K. and N. Shephard (1999). "Filtering via simulation: auxiliary particle filters". *Journal of the American statistical association* **94**:446, pp. 590–599.

Rasmussen, C. E. and C. K. I. Williams (2005). *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press. ISBN: 026218253X.

Rauch, H. E., C. T. Striebel, and F. Tung (1965). "Maximum likelihood estimates of linear dynamic systems". *Journal of the American Institute of Aeronautics and Astronautics* **3**:8, pp. 1445–1450.

Rebeschini, P. and R. van Handel (2013). "Can local particle filters beat the curse of dimensionality?" *arXiv preprint arXiv:1301.6585*.

Särkkä, S. (2013). *Bayesian Filtering and Smoothing*. Cambridge University Press, New York, NY, USA. ISBN: 1107619289, 9781107619289.

Schön, T. B., A. Wills, and B. Ninness (2011). "System identification of nonlinear state-space models". *Automatica* **47**:1, pp. 39–49.

Schön, T., F. Gustafsson, and P.-J. Nordlund (2005). "Marginalized particle filters for mixed linear/nonlinear state-space models". *Signal Processing, IEEE Transactions on* **53**:7, pp. 2279–2289. ISSN: 1053-587X. DOI: 10.1109/TSP.2005.849151.

Shumway, R. H. and D. S. Stoffer (1982). "An approach to time series smoothing and forecasting using the EM algorithm". *Journal of Time Series Analysis* **3**:4, pp. 253–264.

Thrun, S., W. Burgard, and D. Fox (2005). *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press. ISBN: 0262201623.

Todeschini, A., F. Caron, M. Fuentes, P. Legrand, and P. Del Moral (2014). "Biips: software for Bayesian inference with interacting particle systems". *arXiv preprint arXiv:1412.3779*.

Wan, E. A. and R. Van Der Merwe (2000). "The unscented Kalman filter for nonlinear estimation". In: *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*. IEEE, pp. 153–158.

Wigren, T. and T. Söderström (2005). "A second order ODE is sufficient for modeling of many periodic systems". *International Journal of Control* **77**:13, pp. 982–986.

Wood, F., J. W. van de Meent, and V. Mansinghka (2014). "A new approach to probabilistic programming inference". In: *Proceedings of the 17th International conference on Artificial Intelligence and Statistics*.

Paper I

pyParticleEst: A Python Framework for Particle Based Estimation Methods

Jerker Nordh

Abstract

Particle methods such as the Particle Filter and Particle Smoothers have proven very useful for solving challenging nonlinear estimation problems in a wide variety of fields during the last decade. However, there is still very few existing tools to support and assist researchers and engineers in applying the vast number of methods in this field to their own problems. This paper identifies the common operations between the methods and describes a software framework utilizing this information to provide a flexible and extensible foundation which can be used to solve a large variety of problems in this domain, thereby allowing code reuse to reduce the implementation burden and lowering the barrier of entry for applying this exciting field of methods. The software implementation presented in this paper is freely available and permissively licensed under the GNU Lesser General Public License, and runs on a large number of hardware and software platforms, making it usable for a large variety of scenarios.

Conditionally accepted (minor revision) for publication in the *Journal of Statistical Software*

1. Introduction

During the last few years, particle based estimation methods such as particle filtering [Doucet et al., 2000] and particle smoothing [Briers et al., 2010] have become increasingly popular and provide a powerful alternative for nonlinear/non-Gaussian and multi-modal estimation problems. Noteworthy applications of particle methods include multi-target tracking [Okuma et al., 2004], Simultanous Localization and Mapping (SLAM) [Montemerlo et al., 2002] and Radio Channel Estimation [Mannesson, 2013]. Popular alternatives to the particle filter are the Extended Kalman Filter [Julier and Uhlmann, 2004] and Unscented Kalman Filter [Julier and Uhlmann, 2004], but they can not always provide the performance needed, and neither handles multimodal distributions well. The principles of the Particle Filter and Smoother are fairly straight forward, but there are still a few caveats when implementing them. There is a large part of the implementation effort that is not problem specific and thus could be reused, thereby reducing both the overall implementation effort and the risk of introducing errors. Currently there is very little existing software support for using these methods, and for most applications the code is simply written from scratch each time. This makes it harder for people new to the field to apply methods such as particle smoothing. It also increases the time needed for testing new methods and models for a given problem. This paper breaks a number of common algorithms down to a set of operations that need to be performed on the model for a specific problem and presents a software implementation using this structure. The implementation aims to exploit the code reuse opportunities by providing a flexible and extensible foundation to build upon where all the basic parts are already present. The model description is clearly separated from the algorithm implementations. This allows the end user to focus on the parts unique for their particular problem and to easily compare the performance of different algorithms. The goal of this article is not to be a manual for this framework, but to highlight the common parts of a number of commonly used algorithms from a software perspective. The software presented serves both as a proof of concept and as an invitation to those interested to study further, use and to improve upon.

The presented implementation currently supports a number of filtering and smoothing algorithms and has support code for the most common classes of models, including the special case of Mixed Linear/Nonlinear Gaussian State Space (MLNLG) models using Rao-Blackwellized algorithms described in Section 3, leaving only a minimum of implementation work for the end user to define the specific problem to be solved.

In addition to the filtering and smoothing algorithms the framework also contains a module that uses them for parameter estimation (grey-box identification) of nonlinear models. This is accomplished using an Expectation

Maximization (EM)[Dempster et al., 1977] algorithm combined with a Rao-Blackwellized Particle Smoother (RBPS) [Lindsten and Schön, 2010].

The framework is implemented in Python and following the naming conventions typically used within the Python community it has been named pyParticleEst. For an introduction to Python and scientific computation see [T. E. Oliphant, 2007]. All the computations are handled by the Numpy/Scipy [Jones et al., 2001–] libraries. The choice of Python is motivated by that it can run on a wide variety of hardware and software platforms, moreover since pyParticleEst is licensed under the LGPL [FSF, 1999] it is freely usable for anyone without any licensing fees for either the software itself or any of its dependencies. The LGPL license allows it to be integrated into proprietary code only requiring any modifications to the actual library itself to be published as open source. All the code including the examples presented in this article can be downloaded from [Nordh, 2013].

The remaining of this paper is organized as follows. Section 2 gives a short overview of other existing software within this field. Section 3 gives an introduction to the types of models used and a quick summary of notation, Section 4 presents the different estimation algorithms and isolates which operations each method requires from the model. Section 5 provides an overview of how the software implementation is structured and details of how the algorithms are implemented. Section 6 shows how to implement a number of different types of models in the framework. Section 7 presents some results that are compared with previously published data to show that the implementation is correct. Section 8 concludes the paper with a short discussion of the benefits and drawbacks with the approach presented.

2. Related software

The only other software package within this domain to the authors knowledge is LibBi [Murray, In review]. LibBi takes a different approach and provides a domain-specific language for defining the model for the problem. It then generates high performance code for a Particle Filter for that specific model. In contrast, pyParticleEst is more focused on providing an easily extensible foundation where it is easy to introduce new algorithms and model types, a generality which comes at some expense of run-time performance making the two softwares suitable for different use cases. It also has more focus on different smoothing algorithms and filter variants.

There is also a lot of example code that can be found on the Internet, but nothing in the form of a complete library with a clear separation between model details and algorithm implementation. This separation is what gives the software presented in this article its usability as a general tool, not only as a simple template for writing a problem specific implementation. This also

allows for easy comparison of different algorithms for the same problem.

3. Modelling

While the software framework supports more general models, this paper focuses on discrete time stace-space models of the form

$$x_{t+1} = f(x_t, v_t) \tag{1a}$$
$$y_t = h(x_t, e_t) \tag{1b}$$

where x_t are the state variables, v_t is the process noise and y_t is a measurement of the state affected by the measurement noise e_t. The subscript t is the time index. Both v and e are random variables according to some known distributions, f and h are both arbitrary functions.

If f, h are affine and v, e are Gaussian random variables the system is what is commonly referred to as a Linear Gaussian State Space system (LGSS) and the Kalman filter is both the best linear unbiased estimator [Arulampalam et al., 2002] and the Maximum Likelihood estimator.

Due to the scaling properties of the Particle Filter and Smoother, which are discussed in more detail in Section 4.1, it is highly desirable to identify any parts of the models that conditioned on the other states would be linear Gaussian. The state-space can then be partitioned as $x^T = \begin{pmatrix}\xi^T z^T\end{pmatrix}$, where z are the conditionally linear Gaussian states and ξ are the rest. Extending the model above to explicitly indicate this gives

$$\xi_{t+1} = \begin{pmatrix} f_\xi^n(\xi_t, v_\xi^n) \\ f_\xi^l(\xi_t) \end{pmatrix} + \begin{pmatrix} 0 \\ A_\xi(\xi_t) \end{pmatrix} z_t + \begin{pmatrix} 0 \\ v_\xi^l \end{pmatrix} \tag{2a}$$
$$z_{t+1} = f_z(\xi_t) + A_z(\xi_t) z_t + v_z \tag{2b}$$
$$y_t = \begin{pmatrix} h_\xi(\xi_t, e^n) \\ h_z(\xi_t) \end{pmatrix} + \begin{pmatrix} 0 \\ C(\xi_t) \end{pmatrix} z_t + \begin{pmatrix} 0 \\ e^l \end{pmatrix} \tag{2c}$$
$$v_\xi^l \sim N(0, Q_\xi(\xi_t)), \quad v_z \sim N(0, Q_z(\xi_t)), \quad e^l \sim N(0, R(\xi_t)) \tag{2d}$$

As can be seen all relations in (2) involving z are linear with additive Gaussian noise when conditioned on ξ. Here the process noise for the non-linear states v_ξ is split in two parts: v_ξ^l appears linearly and must be Gaussian whereas v_ξ^n can be from any distribution, similarly holds true for e^l and e^n. This is referred to as a Rao-Blackwellized model.

If we remove the coupling from z to ξ we get what is referred to as a

hierarchical model

$$\xi_{t+1} = f_\xi(\xi_t, v_\xi) \tag{3a}$$
$$z_{t+1} = f_z(\xi_t) + A(\xi_t)z_t + v_z \tag{3b}$$
$$y_t = \begin{pmatrix} h_\xi(\xi_t, e^n) \\ h_z(\xi_t) \end{pmatrix} + \begin{pmatrix} 0 \\ C(\xi_t) \end{pmatrix} z_t + \begin{pmatrix} 0 \\ e^l \end{pmatrix} \tag{3c}$$
$$v_z \sim N(0, Q_z(\xi_t)), \quad e^l \sim N(0, R(\xi_t)) \tag{3d}$$

Another interesting class are Mixed Linear/Nonlinear Gaussian (MLNLG) models

$$\xi_{t+1} = f_\xi(\xi_t) + A_\xi(\xi_t)z_k + v_\xi \tag{4a}$$
$$z_{t+1} = f_z(\xi_t) + A_z(\xi_t)z_t + v_z \tag{4b}$$
$$y_t = h(\xi_t) + C(\xi_t)z_t + e \tag{4c}$$
$$e \sim N(0, R(\xi_t)) \tag{4d}$$
$$\begin{bmatrix} v_\xi \\ v_z \end{bmatrix} \sim N\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} Q_\xi(\xi_t) & Q_{\xi z}(\xi_t) \\ Q_{\xi z}(\xi_t)^T & Q_z(\xi_t) \end{bmatrix} \right) \tag{4e}$$

The MLNLG model class (4) allows for non-linear dynamics but with the restrictions that all noise must enter additively and be Gaussian.

4. Algorithms

This section gives an overview of some common particle based algorithms, they are subdivided into those used for filtering, smoothing and static parameter estimation. For each algorithm it is identified which operations need to be performed on the model.

4.1 Filtering

This subsection gives a quick summary of the principles of the Particle Filter, for a thorough introduction see for example [Doucet et al., 2000].

The basic concept of a Particle Filter is to approximate the probability density function (pdf) for the states of the system by a number of point estimates

$$p(x_t|y_{1:t}) \approx \sum_{i=1}^{N} w^{(i)} \delta(x_t - x_t^{(i)}) \tag{5}$$

Each of the N particles in (5) consists of a state, $x_t^{(i)}$, and a corresponding weight, $w_t^{(i)}$, representing the likelihood of that particular particle. Each estimate is propagated forward in time using (1a) by sampling v_t from the corresponding noise distribution, providing an approximation of $p(x_{t+1}|y_t, ..., y_1)$.

The measurement y_{t+1} is incorporated by updating the weights of each particle with respect to how well it predicted the new measurement, giving an approximation of $p(x_{t+1}|y_{t+1}, y_t, ..., y_1)$. This procedure is iterated forward in time providing a filtered estimate of the state x.

A drawback with this approach is that typically all but one of the weights, $w_t^{(i)}$, eventually go to zero resulting in a poor approximation of the true pdf. This is referred to as particle degeneracy and is commonly solved by a process called resampling [Arulampalam et al., 2002]. The idea behind resampling is that at each time-step, or when some criteria is fulfilled, a new collection of particles with all weights equal ($w^{(i)} = \frac{1}{N}, \forall i$) is created by randomly drawing particles, with replacement, according to their weights. This focuses the particle approximation to the most likely regions of the pdf, not wasting samples in regions with low probability. This method is summarized in Algorithm 1.

Another issue with the standard Particle Filter is that the number of particles needed in the filter typically grows exponentially with the dimension of the state-space as discussed in [Beskos et al., 2014] and [Rebeschini and Handel, 2013], where they also present methods to avoid this issue. Another popular approach is to use Rao-Blackwellized methods when there exists a conditionally linear Gaussian substructure. Using the partitioning from model (2) this provides a better approximation of the underlying pdf for a given number of particles by storing the sufficient statistics for the z-states instead of sampling from the Gaussian distributions. For an introduction to the Rao-Blackwellized Particle Filter (RBPF) see [Schön et al., 2005].

A variant of the Particle Filter is the so called Auxiliary Particle Filter (APF), it attempts to focus the particles to regions of high interest by looking one step ahead by evaluating $p(y_{t+1}|x_t)$ and using this to resample the particles before the propagation stage. Since there is typically no analytic expression for this density it is often approximated by assuming that the next state will be the predicted mean; $p(y_{t+1}|x_{t+1} = \bar{x}_{t+1|t})$.

Table 1 summaries the methods needed for the two different filters.

Table 1. Operations that need to be performed on the model for the different filter algorithms. (*typically only approximately).

Operations	Methods
Sample from $p(x_1)$	PF, APF
Sample from $p(x_{t+1}\|x_t)$	PF, APF
Evaluate $p(y_t\|x_t)$	PF, APF
Evaluate* $p(y_{t+1}\|x_t)$	APF

Algorithm 1 Standard Particle Filter algorithm, this is typically improved by not performing the resampling step at every iteration, but only when some prespecified criteria on the weights is fulfilled

Draw $x_0^{(i)}$ from $p(x_0), \quad i \in 1..N$
Set $w_0^{(i)} = \frac{1}{N}, \quad i \in 1..N$
for $t \leftarrow 0$ **to** $T-1$ **do**
  for $i \leftarrow 1$ **to** N **do**
    Sample $x_{t+1}^{(i)}$ from $p(x_{t+1}|x_t^{(i)})$
    Set $w_{t+1}^{(i)} = w_t^{(i)} p(y_{t+1}|x_{t+1}^{(i)})$;
  Normalize weights, $\hat{w}^{(i)} = w_{t+1}^{(i)} / \sum_j w_{t+1}^{(j)}$
  for $i \leftarrow 1$ **to** N **do**
    Sample $x_{t+1}^{(i)} \sim p(x_{t+1}|y_{t+1})$ by drawing new particles from the categorical distribution defined by $(x_{t+1}^{(k)}, \hat{w}^{(k)}), \quad k \in 1..N$
  Set $w_{t+1}^{(i)} = \frac{1}{N}, \quad i \in 1..N$

4.2 Smoothing

Conceptually the Particle Filter provides a smoothed estimate if the trajectory for each particle is saved and not just the estimate for the current time-step. The full trajectory weights are then given by the corresponding particle weights for the last time-step. In practice this doesn't work due to the resampling step which typically results in that all particles eventually share a common ancestor, thus providing a very poor approximation of the smoothed pdf for $t \ll T$. An example of this is shown in Fig. 1

*Forward Filter Backward Simulator*s (FFBSi) are a class of methods that reuse the point estimates for $x_{t|t}$ generated by the particle filter and attempt to improve the particle diversity by drawing backward trajectories that are not restricted to follow the same paths as those generated by the filter. This is accomplished by selecting the ancestor of each particle with probability $\omega_{t|T} \sim \omega_{t|t} p(x_{t+1}|x_t)$. Evaluating all the weights $\omega_{t|T}$ gives a time complexity $O(MN)$ where N is the number of forward particles and M the number of backward trajectories to be generated.

A number of improved algorithms have been proposed that improve this by removing the need to evaluate all the weights. One approach is to use rejection sampling (FFBSi-RS) [Lindsten and Schön, 2013], this however does not guarantee a finite end-time for the algorithm, and typically spends a lot of the time on just a few trajectories. This is handled by introducing *early stopping* (FFBSi-RSES) which falls back to evaluating the full weights for a given time step after a predetermined number of failed attempts at rejection

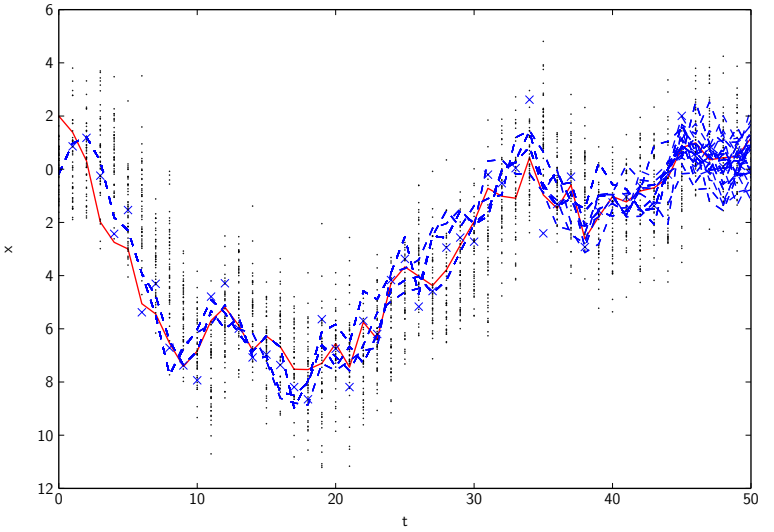

Figure 1. Example realization of a model of a simple integrator. The solid red line is the true trajectory. The black points are the filtered particle estimates forward in time, the blue dashed lines are the smoothed trajectories that result from using the particles ancestral paths. As can be seen this is severely degenerate for small values of t, whereas it works well for t close to the end of the dataset.

sampling. Determining this number ahead of time can be difficult, and the method is further improved by introducing *adaptive stopping* (FFBSi-RSAS) [Taghavi et al., 2013] which estimates the probability of successfully applying rejection sampling based on the previous successes and compares that with the cost of evaluating all the weights.

Another approach is to use Metropolis Hastings (MH-FFBsi) [Bunch and Godsill, 2013] when sampling the backward trajectory, then instead of calculating N weights, R iterations of a Metropolis-Hastings sampler are used.

All the methods mentioned so far only reuse the point estimates from the forward filter, there also exists methods that attempt to create new samples to better approximate the true posterior. One such method is the Metropolis-Hastings Backward Proposer (MHBP) [Bunch and Godsill, 2013], another is the Metropolis-Hastings improved particle smoother (MH-IPS) [Dubarry and Douc, 2011].

MHBP starts with the degenerate trajectories from the filter and while traversing them backwards proposes new samples by running R iterations of a

Metropolis-Hastings sampler targeting $p(x_t|x_{t-1}, x_{t+1}, y_t)$ for each time-step.

MH-IPS can be combined with the output from any of the other smoothers to give an improved estimate. It performs R iterations where each iteration traverses the full backward trajectory and for each time-step runs a single iteration of a Metropolis-Hastings sampler targeting $p(x_t|x_{t-1}, x_{t+1}, y_t)$.

Table 2 lists the operations needed for the different smoothing methods. For a more detailed introduction to Particle Smoothing see for example [Briers et al., 2010], [Lindsten and Schön, 2013], and for an extension to the Rao-Blackwellized case see [Lindsten and Schön, 2011]

Table 2. Operations that need to be performed on the model for the different smoothing algorithms. They all to some extent rely on first running a forward filter, and thus in addition require the operations needed for the filter. Here q is a proposal density, a simple option is to choose $q = p(x_{t+1}|x_t)$, as this does not require any further operations. The ideal choice would be $q = p(x_t|x_{t+1}, x_{t-1}, y_t)$, but it is typically not possible to directly sample from this density.

Operations	Methods
Evaluate $p(x_{t+1}\|x_t)$	FFBSi, FFBSi-RS, FFBSi-RSES, FFBSi-RSAS, MH-FFBSi, MH-IPS, MHBP
Evaluate $\text{argmax}_{x_{t+1}} p(x_{t+1}\|x_t)$	FFBSi-RS, FFBSi-RSES, FFBSi-RSAS
Sample from $q(x_t\|x_{t-1}, x_{t+1}, y_t)$	MH-IPS, MHBP
Evaluate $q(x_t\|x_{t-1}, x_{t+1}, y_t)$	MH-IPS, MHBP

4.3 Parameter estimation

Using a standard Particle Filter or Smoother it is not possible to estimate stationary parameters, θ, due to particle degeneracy. A common work-around for this is to include θ in the state vector and model the parameters as a random walk process with a small noise covariance. A drawback with this approach is that the parameter is no longer modeled as being constant, in addition it increases the dimension of the state-space, worsening the problems mentioned in Section 4.1.

PS+EM Another way to do parameters estimation is to use an Expectation Maximization (EM) algorithm where the expectation part is calculated using a RBPS. For a detailed introduction to the EM-algorithm see [Dempster et al., 1977] and for how to combine it with a RBPS for parameter estimates in model (4) see [Lindsten and Schön, 2010].

The EM-algorithm finds the maximum likelihood solution by alternating between estimating the Q-function for a given θ_k and finding the θ that maximizes the log-likelihood for a given estimate of $x_{1:T}$, where

$$Q(\theta, \theta_k) = \mathrm{E}_{X|\theta_k}[\mathrm{L}_\theta(X, Y|Y)] \tag{6a}$$
$$\theta_{k+1} = \operatorname*{argmax}_\theta Q(\theta, \theta_k) \tag{6b}$$

Here X is the complete state trajectory $(x_1, ..., x_N)$, Y is the collection of all measurements $(y_1, ..., y_N)$ and L_θ is the log-likelihood as a function of the parameters θ. In [Lindsten and Schön, 2010] it is shown that the Q-function can be split into three parts as follows

$$Q(\theta, \theta_k) = I_1(\theta, \theta_k) + I_2(\theta, \theta_k) + I_3(\theta, \theta_k) \tag{7a}$$
$$I_1(\theta, \theta_k) = \mathrm{E}_{\theta_k}[\log p_\theta(x_1)|Y] \tag{7b}$$
$$I_2(\theta, \theta_k) = \sum_{t=1}^{N-1} \mathrm{E}_{\theta_k}[\log p_\theta(x_{t+1}|x_t)|Y] \tag{7c}$$
$$I_3(\theta, \theta_k) = \sum_{t=1}^{N} \mathrm{E}_{\theta_k}[\log p_\theta(y_t|x_t)|Y] \tag{7d}$$

The expectations in (7b)-(7d) are approximated using a (Rao-Blackwellized) Particle Smoother, where the state estimates are calculated using the old parameter estimate θ_k. This procedure is iterated until the parameter estimates converge. The methods needed for PS+EM are listed in Table 3

PMMH Another method which instead takes a Bayesian approach is Particle Marginal Metropolis-Hastings (PMMH)[Andrieu et al., 2010] which is one method within the broader class known as Particle Markov Chain Monte Carlo (PMCMC) methods. It uses a particle filter as part of a Metropolis-Hastings sampler targeting the joint density of the state trajectory and the unknown parameters. This method is not discussed further in this paper. The methods needed for PMMH are listed in Table 3.

5. Implementation

5.1 Language

The framework is implemented in Python, for an introduction to the use of Python in scientific computing see [T. E. Oliphant, 2007]. The numerical computations rely on Numpy/Scipy [Jones et al., 2001–] for a fast and efficient implementation. This choice was made as it provides a free environment, both in the sense that there is no need to pay any licensing fees to use it,

Table 3. Operations that need to be performed on the model for the presented parameter estimation methods. PS-EM relies on running a smoother, and thus in addition requires the operations needed for the smoother. The maximization is with respect to θ. Typically the maximization can not be performed analytically, and then depending on which type of numerical solver is used, gradients and Hessians might be needed as well. PMMH does not require a smoothed estimate, it only uses a filter, and thus puts fewer requirements on the types of models that can be used. Here q is the proposal density for the static parameters, π is the prior probability density function. PMMH does not need a smoothed trajectory estimate, it is sufficient with the filtered estimate.

Operations	Methods
Maximize $\mathrm{E}_{\theta_k}[\log p_\theta(x_1)\|Y]$	PS+EM
Maximize $\mathrm{E}_{\theta_k}[\log p_\theta(x_{t+1}\|x_t)\|Y]$	PS+EM
Maximize $\mathrm{E}_{\theta_k}[\log p_\theta(y_t\|x_t)\|Y]$	PS+EM
Evaluate $q(\theta'\|\theta)$	PMMH
Sample from $q(\theta'\|\theta)$	PMMH
Evaluate $\pi(\theta)$	PMMH

but also that the code is open source and available for a large number of operating systems and hardware platforms. The pyParticleEst framework is licensed under the LGPL [FSF, 1999], which means that it can be freely used and integrated into other products, but any modifications to the actual pyParticleEst code must be made available. The intent behind choosing this license is to make the code easily usable and integrable into other software packages, but still encourage sharing of any improvements made to the library itself. The software and examples used in this article can be found in [Nordh, 2013].

5.2 Overview

The fundamental idea in pyParticleEst is to provide algorithms operating on the methods identified in Section 4, thus effectively separating the algorithm implementation from the problem description. Additionally, the framework provides an implementation of these methods for a set of common model classes which can be used for solving a large set of problems. They can also be extended or specialized by the user by using the inheritance mechanism in Python. This allows new types of problems to be solved outside the scope of what is currently implemented, but it also allows creation of classes building on the foundations present but overriding specific methods for increased performance, without rewriting the whole algorithm from scratch. The author believes this provides a good trade-off between generality, extensibility and

ease of use.

For each new type of problem to be solved the user defines a class extending the most suitable of the existing base classes, for example the one for MLNLG systems. In this case the user only has to specify how the matrices and functions in (4) depend on the current estimate of the nonlinear state. For a more esoteric problem class the end user might have to do more implementation work and instead derive from a class higher up in the hierarchy, for example the base class for models that can be partioned into a conditionally linear part, which is useful when performing Rao-Blackwellized filtering or smoothing. This structure is explained in more detail in sections 5.3-5.3.

The main interface to the framework is through the `Simulator` class, it is used to store the model used for the estimation toghether with the input signals and measurements, it also provides a mechanism for executing the different algorithms on the provided model and data. It is used by creating an object of the Simulator class with input parameters that specify the problem to be solved as follows

```
sim = Simulator(model, u, y)
```

Here `model` is an object defining all model specific operations, `u` is an array of all the input signals and `y` is an array of all measurements. Once the object has been created it serves as the interface to the actual algorithm, an example of how it could be used is shown below

```
sim.simulate(num, nums, res=0.67, filter='PF', smoother='mcmc')
```

Here `num` is the number of particles used in the forward filter, `nums` are the number of smoothed trajectories generated by the smoother, `res` is the resampling threshhold (expressed as the ratio of effective particles compared to total number of particles), `filter` is the filtering method to be used and finally `smoother` is the smoothing algorithm to be used.

After calling the method above the results can be access by using some of the following methods

```
(est_filt, w_filt) = sim.get_filtered_estimates()
mean_filt = sim.get_filtered_mean()
est_smooth = sim.get_smoothed_estimates()
smean = sim.get_smoothed_mean()
```

where `(est_filt, w_filt)` will contain the forward particles for each time step with the corresponding weights, `mean_filt` is the weighted mean of all the forward particles for each time step. `est_smooth` is an array of all the smoothed trajectories and `smean` the mean value for each time step of the smoothed trajectories.

5.3 Software design

The software consists of a number of supporting classes that store the objects and their relations, the most important of these are shown in Figure 2 and are summarized below.

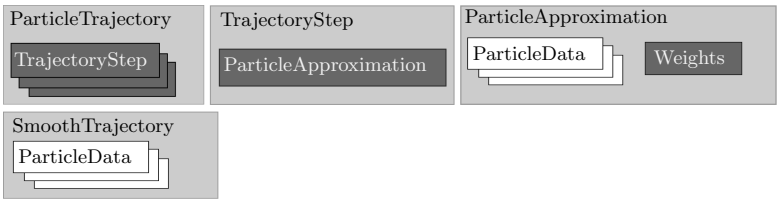

Figure 2. Overview of the classes used for representing particle estimates and their relation. The grey boxes are classes that are part of the framework, the white boxes represent objects of problem specific data-types. A box encapsulating another box shows that objects from that class contains objects from the other class. The illustration is not complete, but serves as an overview of the overall layout.

The particles are stored as raw data, where each model class is responsible for determining how it is best represented. This data is then sent as one of the parameters to each method the model class defines. This allows the model to choose an efficient representation allowing for e.g., parallell processing of all the particles for each time-step. The details of the class hierarchy and the models for some common cases are explored further in sections 5.3-5.3.

The particle data is stored using the `ParticleApproximation` class, which in addition to the raw data also stores the corresponding weights according to (5). The class `TrajectoryStep` stores the approximation for a given time instant combined with other related data such as input signals and measurement data. The `ParticleTrajectory` class represents the filtered estimates of the entire trajectory by storing a collection of `TrajectorySteps`, it also provides the methods for interfacing with the chosen filtering algorithm.

The `SmoothTrajectory` class takes a `ParticleTrajectory` as input and using a Particle Smoother creates a collection of point estimates representing the smoothed trajectory estimate. In the same manner as for the `ParticleApproximation` class the point estimates here are of the problem specific data type defined by the model class, but not necessarily of the same structure as the estimates created by the forward filter. This allows for example methods where the forward filter is Rao-Blackwellized but the backward smoother samples the full state vector.

Model class hierarchy The software utilizes the Python ABC package to create a set of abstract base-classes that define all the needed operations for

the algorithms. Figure 3 shows the complete class hierarchy for the algorithm interfaces and model types currently implemented.

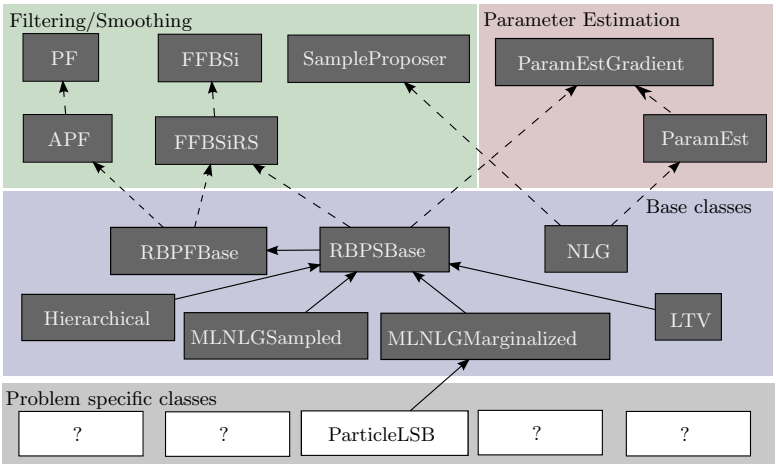

Figure 3. Class hierarchy for models that are used in the framework. The `ParticleLSB` class is presented in Section 6.3 and is an implementation of Example B from [Lindsten and Schön, 2011].

- `PF` defines the basic operations needed for performing particle filtering:
  - `create_initial_estimate`: Create particle estimate of initial state.
  - `sample_process_noise`: Sample v_t from the process noise distribution.
  - `update`: Calculate x_{t+1} given x_t using the supplied noise v_t.
  - `measure`: Evaluate $\log p(y_t|x_{t|t-1})$ and for the RBPF case update the sufficient statistics for the z-states.
- `APF` extends `PF` with extra methods needed for the Auxiliary Particle Filter:
  - `eval_1st_stage_weights`: Evaluate (approximately) the so called first stage weights, $p(y_{t+1}|x_t)$.
- `FFBSi` defines the basic operations needed for performing particle smoothing:

  - `logp_xnext_full`: Evaluate $\log p(x_{t+1:T}|x_{1:t}, y_{1:T})$. This method normally just calls `logp_xnext`, but the distinction is needed for non-Markovian models.
  - `logp_xnext`: Evaluate $\log p(x_{t+1}|x_t)$.
  - `sample_smooth`: For normal models the default implementation can be used which just copies the estimate from the filter, but for e.g., Rao-Blackwellized models additional computations are made in this method.

- `FFBSiRS` extends `FFBSi`:
  - `next_pdf_max`: Calculate maximum of $\log p(x_{t+1}|x_t)$.

- `SampleProposer` defines the basic operations needed for proposing new samples, used in the MHBP and MH-IPS algorithms:
  - `propose_smooth`: Propose new sample from $q(x_t|x_{t+1}, x_{t-1}, y_t)$.
  - `logp_proposal`: Evaluate $\log q(x_t|x_{t+1}, x_{t-1}, y_t)$.

- `ParamEstInterface` defines the basic operations needed for performing parameter estimation using the EM-algorithm presented in Section 4.3:
  - `set_params`: Set θ_k estimate.
  - `eval_logp_x0`: Evaluate $\log p(x_1)$.
  - `eval_logp_xnext`: Evaluate $\log p(x_{t+1}|x_t)$.
  - `eval_logp_y`: Evaluate $\log p(y_t|x_t)$.

- `ParamEstInterface_GradientSearch` extends the operations from the ParamEstInterface to include those needed when using analytic derivatives in the maximization step:
  - `eval_logp_x0_val_grad`: Evaluate $\log p(x_1)$ and its gradient.
  - `eval_logp_xnext_val_grad`: Evaluate $\log p(x_{t+1}|x_t)$ and its gradient.
  - `eval_logp_y_val_grad`: Evaluate $\log p(y_t|x_t)$ and its gradient.

Base classes To complement the abstract base classes from Section 5.3 the software includes a number of base classes to help implement the required functions.

- `RBPFBase` Provides an implementation handling the Rao-Blackwellized case automatically by defining a new set of simpler functions that are required from the derived class.

- `RBPSBase` Extends `RBPFBase` to provide smoothing for Rao-Blackwellized models.

Model classes These classes further specialize those from sections 5.3-5.3.

- `LTV` Handles Linear Time-Varying systems, the derived class only needs to provide callbacks for how the system matrices depend on time.
- `NLG` Nonlinear dynamics with additive Gaussian noise.
- `MixedNLGaussianSampled` Provides support for models of type (4) using an algorithm which samples the linear states in the backward simulation step. The sufficient statistics for the linear states are later recovered in a post processing step. See [Lindsten and Schön, 2011] for details. The derived class needs to specify how the linear and non-linear dynamics depend on time and the current estimate of ξ.
- `MixedNLGaussianMarginalized` Provides an implementation for models of type (4) that fully marginalizes the linear Gaussian states, resulting in a non-Markovian smoothing problem. See [Lindsten et al., 2013] for details. The derived class needs to specify how the linear and non-linear dynamics depend on time and the current estimate of ξ. This implementation requires that $Q_{\xi z} = 0$.
- `Hierarchial` Provides a structure useful for implementing models of type (3) using sampling of the linear states in the backward simulation step. The sufficient statistics for the linear states are later recovered in a post processing step.

For the LTV and MLNLG classes the parameters estimation interfaces, `ParamEstInterface` and `ParamEstInterface_GradientSearch`, are implemented so that the end-user can specify the element-wise derivative for the matrices instead of directly calculating gradients of (7b)-(7d). Typically there is some additional structure to the problem, and it is then beneficial to override this generic implementation with a specialized one to reduce the computational effort by utilizing that structure.

5.4 Algorithms

RBPF The Particle Filter implemented is summarized with pseudo-code in Algorithm 2. The predict step is detailed in Algorithm 3 and the measurement step in Algorithm 4. N_{eff} is the effective number of particles as defined in [Arulampalam et al., 2002] and is used to trigger the resampling step when a certain predefined threshold is crossed.

RBPS The main RBPS algorithm implemented in pyParticleEst is of the type JBS-RBPS with constrained RTS-smoothing from [Lindsten et al., 2013]. It simulates M backward trajectories using filtered estimates. During the backward simulation step it samples the linear/Gaussian states, but

Algorithm 2 (Rao-Blackwellized) Particle Filter

for $t \leftarrow 0$ **to** $T-1$ **do**
  for $i \leftarrow 1$ **to** N **do**
    Predict $x_{t+1|t} \leftarrow x_{t|t}$ using Alg. 3
    Update $x_{t+1|t+1} \leftarrow x_{t+1|t}, y_{t+1}$ using Alg. 4
    if $N_{eff} < N_{treshold}$ **then**
      Resample using Alg. 5

Algorithm 3 RBPF Predict, step 4-6 is only needed for the Rao-Blackwellized case

1. *Update system dynamics,* $f(x_t, v_t)$*, based on* ξ_t
2. *Sample process noise,* v_t
3. *Calculate* $\xi_{t+1|t}$ *using sampled* v_t
4. *Use knowledge of* $\xi_{t+1|t}$ *to update estimate of* z_t
5. *Update linear part of system dynamics with knowledge of* $\xi_{t+1|t}$
6. *Predict* $z_{t+1|t}$ *(conditioned on* $\xi_{t+1|t}$*)*

Algorithm 4 RBPF Measure, step 2 is only needed for the Rao-Blackwellized case

1. *Update system dynamics,* $h(x_t, e_t)$*, based on* $\xi_{t+1|t}$
2. *Calculate* $z_{t+1|t+1}$ *using* y_{t+1}
3. *Update weights* $w^{(i)}_{t+1|t+1} = w^{(i)}_{t+1|t} p(y_{t+1}|x_{t+1|t})$

later recovers the sufficient statistics, i.e., the mean and covariance, by running a constrained RTS-smoother [Rauch et al., 1965] conditioned on the nonlinear part of the trajectory. It can be combined with any of the backward simulation methods, for example FFBSi-RS or MH-FFBSi. The method is summarized in Algorithm 6.

Algorithm 5 The resampling algorithm used in the framework. Different resampling algorithms have been proposed in the literature, this one has the property that a particle, $x^{(i)}$, with $w^{(i)} \geq \frac{1}{N}$ is guaranteed to survive the resampling step.

$\tilde{\omega}_c = \text{cumsum}(\omega)$
$\omega_c = \tilde{\omega}_c / \sum \tilde{\omega}_c$
$u = ([0 : N-1] + U(0,1))/N$
for $k \leftarrow 1$ **to** N **do**
  $x^{(k)} = x^{(i)}, i = \underset{j}{\text{argmin}}\, \omega_c^{(j)} > u^{(k)}$

Additionally for MLNLG models (4) there is another RBPS algorithm implemented which fully marginalizes the linear states, it is an implementation of the method described in [Lindsten et al., 2013]. This is the statistically correct way to solve the problem, and it gives better accuracy, but it also requires more computations resulting in a longer execution time. Due to the difficulty in evaluating $\text{argmax}_{x_{t+1:T}} p(x_{t+1:T}, y_{t+1:T}|x_{t|t}, z_{t|t}, P_{t|t})$ rejection sampling is not implemented, it is anyhow unlikely to perform well due the large dimension of target variables (since they are full trajectories, and no longer single time instances). The implementation is also limited to cases where the cross covariance ($Q_{\xi z}$) between the nonlinear and linear states is zero. This method is summarized in Algorithm 7.

Algorithm 6 (Rao-Blackwellized) Particle Smoother

(0. Run RBPF generating filtered estimates)
Sample index i with probability $w^{(i)}_{T|T}$
Add $x^{(i)}_{T|T}$ to the backward trajectory
for $t \leftarrow T-1$ **to** 0 **do**
  Sample $\tilde{z}_{t+1}$ from $z^{(i)}_{t+1|T}$
  Sample index k with probability $w^{(k)}_{t|t} p(\xi^{(i)}_{t+1}, \tilde{z}_{t+1}|x^{(k)}_{t|t})$
  Update sufficent statistics of $z^{(k)}_t$ conditioned on $(\xi_{t+1}, \tilde{z}_{t+1})$
  Append $x^{(k)}_{t|T}$ to trajectory
  $i \leftarrow k$
Calculate dynamics for Rao-Blackwellized states conditioned on the nonlinear trajectory $\xi_{0:T}$
Run constrained RTS-smoothing to recover sufficient statistics for $z_{0:T}$

Parameter estimation Parameter estimation is accomplished using an EM-algorithm as presented in Section 4.3. It requires that the derived particle class implements `ParamEstInterface`. The method is summarized in Algorithm 8. The maximization step in (6b) is performed using `scipy.optimize.minimize` with the `l-bfgs-b` method [Zhu et al., 1997], which utilizes the analytic Jacobian when present.

Algorithm 7 Fully Marginalized Particle Smoother for MLNLG (4)

(0. Run RBPF generating filtered estimates)
Sample index i with probability $w_{T|T}^{(i)}$
Add $\xi_{T|T}^{(i)}$ to the backward trajectory
for $t \leftarrow T-1$ **to** 0 **do**
  Sample index k with probability $w_{t|t}^{(k)} p(\xi_{t+1:T}^{(i)}, y_{t+1:T} | \xi_{t|t}^{(k)}, z_{t|t}^{(k)}, P_{z\,t|t}^{(k)})$
  Append $\xi_{t|T}^{(k)}$ to trajectory
  $i \leftarrow k$
Calculate dynamics for Rao-Blackwellized states conditioned on the non-linear trajectory $\xi_{0:T}$
Run constrained RTS-smoothing to recover sufficient statistics for $z_{0:T}$

Algorithm 8 RBPS+EM algorithm

(0. Initial parameter guess, θ_0)
for $i \leftarrow 1$ **to** *max_iter* **do**
  Estimate $p(x_{1:T}|y_{1:T}, \theta_{i-1})$ using Alg. 6 (RBPS)
  Approximate $Q(\theta, \theta_{i-1})$ using estimated trajetory
  Calculate $\theta_i = \operatorname{argmax}_\theta Q(\theta, \theta_{i-1})$

6. Example models

6.1 Integrator

A trivial example consisting of a linear Gaussian system

$$x_{t+1} = x_t + w_t \tag{8a}$$
$$y_t = x_t + e_t, \quad x_1 \sim N(0,1) \tag{8b}$$
$$w_t \sim N(0,1), \quad e_t \sim N(0,1) \tag{8c}$$

This model could be implemented using either the `LTV` or `NLG` model classes, but for this example it was decided to directly implement the required top level interfaces to illustrate how they work. In this example only the methods needed for filtering are implemented. To use smoothing the `logp_xnext` method would be needed as well. An example realization using this model was shown in Fig. 1

```
1  class Integrator(interfaces.ParticleFiltering):
       def __init__(self, P0, Q, R):
           self.P0 = numpy.copy(P0)
           self.Q = numpy.copy(Q)
           self.R = numpy.copy(R)
6
       def create_initial_estimate(self, N):
           return numpy.random.normal(0.0, self.P0, (N,)
                                      ).reshape((-1, 1))

11     def sample_process_noise(self, particles, u, t):
           N = len(particles)
           return numpy.random.normal(0.0, self.Q, (N,)
                                      ).reshape((-1, 1))

16     def update(self, particles, u, t, noise):
           particles += noise

       def measure(self, particles, y, t):
           logyprob = numpy.empty(len(particles))
21         for k in range(len(particles)):
               logyprob[k] = kalman.lognormpdf(
                                   particles[k, 0] - y,
                                   self.R)
           return logyprob
```

Line `08` Samples the initial particles from a zero-mean Gaussian distribution with variance `P0`.

Line `13` This samples the process noise at time `t`.

Line `16` Propagates the estimates forward in time using the noise previously sampled.

Line `19` Calculates the log-probability for the measurement y_t for particles $x_t^{(i)}$.

6.2 Standard nonlinear model

This is a model that is commonly used as an example when demonstrating new algorithms, see e.g., [Lindsten and Schön, 2013], [Arulampalam et al., 2002] and [Briers et al., 2010]

$$x_{t+1} = 0.5x_t + 25\frac{x_t}{1+x_t^2} + 8\cos 1.2t + w_t \tag{9a}$$

$$y_t = 0.05x_t^2 + e_t, \quad x_1 \sim N(0,5) \tag{9b}$$

$$w_t \sim N(0,10), \quad e_t \sim N(0,1) \tag{9c}$$

For the chosen noise covariances the filtering distribution is typically multi-modal whereas the smoothing distribution is mostly unimodal. Figure 4 shows an example realization from this model, the smoothed estimates have been calculated using backward simulation with rejection sampling using adapative stopping (FFBSi-RSAS).

The corresponding model definition exploits that this is a model of the type Nonlinear Gaussian, and thus inherits the base class for that model type.

```
class StdNonLin(nlg.NonlinearGaussianInitialGaussian):
    def __init__(self, P0, Q, R):
        super(StdNonLin, self).__init__(Px0=P0, Q=Q, R=R)

    def calc_g(self, particles, t):
        return 0.05 * particles ** 2

    def calc_f(self, particles, u, t):
        return (0.5 * particles +
                25.0 * particles / (1 + particles ** 2) +
                8 * math.cos(1.2 * t))
```

Line 03 In this example the covariance matrices are time-invariant and can thus be set in the constructor. This also allows the base class to later perform optimization where the fact that the matrices are identical for all particles can be exploited.

Line 05 `calc_g` utilizes that all the particles are stored in an array to effectively evaluate $g_t(x_t^{(i)})$ for all particles in a single method call.

Line 08 `calc_f` evaluates $f_t(x_t^{(i)})$ in a similar fashion as above.

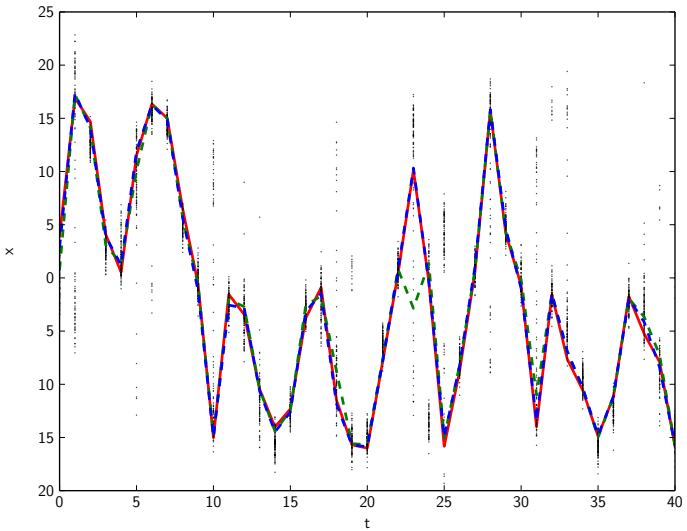

Figure 4. Example realization using the standard nonlinear model. The solid red line is the true trajectory. The black points are the filtered particle estimates forward in time, the green dashed line is the mean value of the filtered estimates, the blue dashed line is the mean value of the smoothed trajectories. The smoothing was performed using the BSi RSAS algorithm. Notice that the filtered mean does not follow the true state trajectory due to the multi-modality of the distribution, whereas the smoothed estimate does not suffer from this problem.

6.3 Lindsten and Schön, Model B

This model was introduced in [Lindsten and Schön, 2011] as an extension to the standard nonlinear model from Section 6.2. It replaces the constant 25 by the output of a fourth order linear system.

$$\xi_{t+1} = 0.5\xi_t + \theta_t \frac{\xi_t}{1+\xi_t^2} + 8\cos 1.2t + v_{\xi,t} \tag{10a}$$

$$z_{t+1} = \begin{pmatrix} 3 & -1.691 & 0.849 & -0.3201 \\ 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \end{pmatrix} z_t + v_{z,t} \tag{10b}$$

$$y_t = 0.05\xi_t^2 + e_t \tag{10c}$$

$$\theta_t = 25 + \begin{pmatrix} 0 & 0.04 & 0.044 & 0.008 \end{pmatrix} z_t \tag{10d}$$

$$\xi_0 = 0, \quad z_0 = \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix}^T \tag{10e}$$

Since this model conforms to the class from (4) it was implemented using the `MLNLG` base class. Doing so it only requires the user to define the functions and matrices as a function of the current state. The corresponding source code is listed below.

```
class ParticleLSB(
    mlnlg.MixedNLGaussianMarginalizedInitialGaussian):
    def __init__(self):
        xi0 = numpy.zeros((1, 1))
        z0 = numpy.zeros((4, 1))
        P0 = numpy.zeros((4, 4))

        Az = numpy.array([[3.0, -1.691, 0.849, -0.3201],
                          [2.0, 0.0, 0.0, 0.0],
                          [0.0, 1.0, 0.0, 0.0],
                          [0.0, 0.0, 0.5, 0.0]])

        Qxi = numpy.diag([ 0.005])
        Qz = numpy.diag([ 0.01, 0.01, 0.01, 0.01])
        R = numpy.diag([0.1, ])

        super(ParticleLSB, self).__init__(xi0=xi0, z0=z0,
                                          Pz0=P0, Az=Az,
                                          R=R, Qxi=Qxi,
                                          Qz=Qz,)

    def get_nonlin_pred_dynamics(self, particles, u, t):
        tmp = numpy.vstack(particles)[:, numpy.newaxis, :]
        xi = tmp[:, :, 0]
```

```
            Axi = (xi / (1 + xi ** 2)).dot(C_theta)
27          Axi = Axi[:, numpy.newaxis, :]

            fxi = (0.5 * xi +
                   25 * xi / (1 + xi ** 2) +
                   8 * math.cos(1.2 * t))
32          fxi = fxi[:, numpy.newaxis, :]

            return (Axi, fxi, None)

        def get_meas_dynamics(self, particles, y, t):
37          if (y != None):
                y = numpy.asarray(y).reshape((-1, 1)),
            tmp = 0.05 * particles[:, 0] ** 2
            h = tmp[:, numpy.newaxis, numpy.newaxis]

42          return (y, None, h, None)
```

Line 03 In the constructor all the time-invariant parts of the model are set

Line 22 This function calculates $A_\xi(\xi_t^{(i)}), f_\xi(\xi_t^{(i)})$ and $Q_\xi(\xi_t^{(i)})$

Line 27 The array is resized to match the expected format (The first dimension indices the particles, each entry being a two-dimensional matrix)

Line 34 Return a tuple containing A_ξ, f_ξ and Q_ξ arrays. Returning `None` for any element in the tuple indicates that the time-invariant values set in the constructor should be used

Line 37 This function works in the same way as above, but instead calculates $h(\xi_t)$, $C(\xi_t)$ and $R(\xi_t)$. The first value in the returned tuple should be the (potentially preprocessed) measurement.

7. Results

The aim of this section is to demonstrate that the implementation in pyParticleEst is correct by reproducing results previously published elsewhere.

7.1 Rao-Blackwellized particle filtering / smoothing

Here Example B from [Lindsten and Schön, 2011] is reproduced, it uses the model definition from Section 6.3 the marginalized base class for MLNLG models. The results are shown in Table 4 which also contains the corresponding values from [Lindsten and Schön, 2011]. The values were calculated by running the RBPS-agorithm on 1000 random realizations of model (10) using 300 particles and 50 smoothed trajectories. The smoothed trajectories where averaged to give a point estimate for each time step. The average was used to calculated the RMSE for a single realization. The values in this article were computed using the marginalized `MLNLG` base class, which uses the smoothing algorithm presented in [Lindsten et al., 2013]. This is a later improvement to the algorithm used in the original article, which explains why the values presented here are better than those in [Lindsten and Schön, 2011]. The mean RMSE is also highly dependent on the particular realizations, 89.8% of the realizations have a lower RMSE than the average, whereas 3.3% have an RMSE greater than 1.0. This also makes a direct comparison of the values problematic since the exact amount of outliers in the dataset will have a significant impact om the average RMSE.

Table 4. Root mean squared error (RMSE) values for ξ and θ from model 10, compared with those presented in [Lindsten and Schön, 2011].

RMSE	ξ	θ
pyParticleEst	0.275	0.545
Lindsten & Schön	0.317	0.585

7.2 Parameter estimation in MLNLG

In [Lindsten and Schön, 2010] the following model is introduced

$$\xi_{t+1} = \theta_1 \arctan \xi_t + \begin{pmatrix} \theta_2 & 0 & 0 \end{pmatrix} z_t + v_{\xi,t} \tag{11a}$$

$$z_{t+1} = \begin{pmatrix} 1 & \theta_3 & 0 \\ 0 & \theta_4 \cos\theta_5 & -\theta_4 \sin\theta_5 \\ 0 & \theta_4 \sin\theta_5 & \theta_4 \cos\theta_5 \end{pmatrix} z_t + v_{z,t} \tag{11b}$$

$$y_t = \begin{pmatrix} 0.1\xi_t^2 \operatorname{sgn}(\xi_t) \\ 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 1 & -1 & 1 \end{pmatrix} z_t + e_t \tag{11c}$$

The task presented is to identify the unknown parameters, θ_i. Duplicating the conditions as presented in the original article, but running the algorithm on 160 random data realizations instead of 70, gives the results presented in Table 5. The authors of [Lindsten and Schön, 2010] do not present the number of smoothed trajectories used in their implementation, for the results in this article 5 smoothed trajectories were used.

Table 5. Results presented by Lindsten and Schön in [Lindsten and Schön, 2010] compared to results calculated using *pyParticleEst*. The column marked with * are the statistics when excluding those realizations where the EM-algorithm was stuck in local maxima for θ_5.

	True value	Lindsten & Schön	pyParticleEst	pyParticleEst*
θ_1	1	0.966 ± 0.163	0.981 ± 0.254	1.006 ± 0.091
θ_2	1	1.053 ± 0.163	0.947 ± 0.158	0.984 ± 0.079
θ_3	0.3	0.295 ± 0.094	0.338 ± 0.308	0.271 ± 0.112
θ_4	0.968	0.967 ± 0.015	0.969 ± 0.032	0.962 ± 0.017
θ_5	0.315	0.309 ± 0.057	0.263 ± 0.134	0.312 ± 0.019

Looking at the histogram of the estimate of θ_5 shown in Figure 5 it is clear that there are several local maxima. Of the 160 realizations 21 converged to a local maximum for θ_5 thus giving an incorrect solution. This is typically handled by solving the optimization problem using several different initial conditions and choosing the one with the maximum likelihood. However since that does not appear to have been performed in [Lindsten and Schön, 2010] it is problematic to compare the values obtained, since they will be highly dependent on how many of the realizations that converged to local maxima. Therefore Table 5 contains a second column named pyParticleEst* which presents the same statistics but excluding those realizations where θ_5 converged to a local maxima.

8. Conclusion

pyParticleEst lowers the barrier of entry to the field of particle methods, allowing many problems to be solved with significantly less implementation effort compared to starting from scratch. This was exemplified by the models presented in Section 6, demonstrating the significant reduction in the amount of code needed to be produced by the end-user. Its use for grey-box identification was demonstrated in Section 7.2. The software and examples used in this article can be found at [Nordh, 2013].

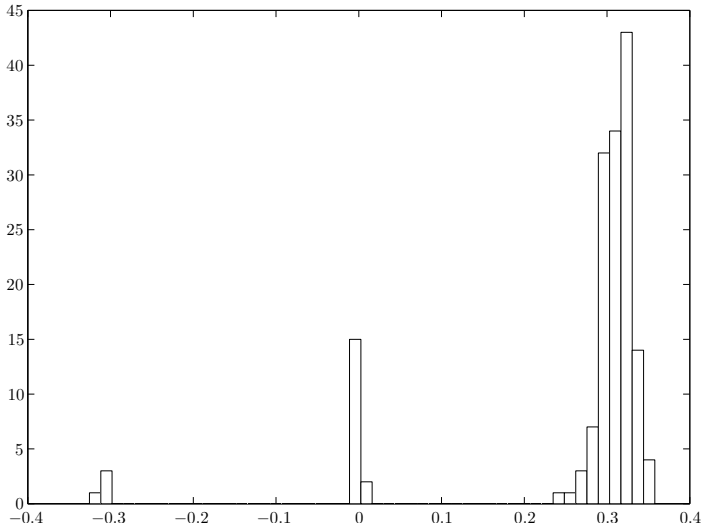

Figure 5. Histogram for θ_5. The peaks around -0.3 and 0 are likely due to the EM-algorithm converging to local maxima. Since θ_5 enters the model through $\sin\theta_5$ and $\cos\theta_5$, with cos being a symmetric function the peak around -0.3 could intuitively be expected.

There is an overhead due to the generic design which by necessity gives lower performance compared to a specialized implementation in a low-level language. For example a hand optimized C-implementation that fully exploits the structure of a specific problem will always be faster, but also requires significantly more time and knowledge from the developer. Therefore the main use-case for this software when it comes to performance critical applications is likely to be prototyping different models and algorithms that will later be re-implemented in a low-level language. That implementation can then be validated against the results provided by the generic algorithms. In many circumstances the execution time might be of little concern and the performance provided using pyParticleEst will be sufficient. There are projects such as Numba [Continuum Analytics, 2014], Cython [Behnel et al., 2009] and PyPy [Rigo, 2004] that aim to increase the efficiency of Python-code. Cython is already used for some of the heaviest parts in the framework. By selectively moving more of the computationally heavy parts of the model base classes to Cython it should be possible to use the framework directly for many real-time applications.

For the future the plan is to extend the framework to contain more algorithms, for example the interesting field of PMCMC methods [Del Moral et al., 2006]. Another interesting direction is smoothing of non-Markovian

models as examplified by the marginalized smoother for MLNLG models. This type of smoother could also be combined with Gaussian processes as shown by [Lindsten and Schön, 2013]. The direction taken by e.g., [Murray, In review] with a high level language is interesting, and something that might be worthwhile to implement for automatically generating the Python code describing the model, providing a further level of abstraction for the end user.

Acknowledgments

The author is a member of the LCCC Linnaeus Center and the eLLIIT Excellence Center at Lund University. The author would like to thank professor Bo Bernhardsson, Lund University, and professor Thomas Schön, Uppsala University, for the feedback provided on this work.

References

Andrieu, C., A. Doucet, and R. Holenstein (2010). "Particle Markov chain Monte Carlo methods". *Journal of the Royal Statistical Society, Series B* **72**:2, pp. 1–33.

Arulampalam, M., S. Maskell, N. Gordon, and T. Clapp (2002). "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking". *IEEE Trans. Signal Process.* **50**:2, pp. 174–188. ISSN: 1053-587X.

Behnel, S., R. W. Bradshaw, and D. S. Seljebotn (2009). "Cython tutorial". In: Varoquaux, G. et al. (Eds.). *Proceedings of the 8th Python in Science Conference*. Pasadena, CA USA, pp. 4–14.

Beskos, A., D. Crisan, A. Jasra, et al. (2014). "On the stability of sequential Monte Carlo methods in high dimensions". *The Annals of Applied Probability* **24**:4, pp. 1396–1445.

Briers, M., A. Doucet, and S. Maskell (2010). "Smoothing algorithms for state-space models". English. *Annals of the Institute of Statistical Mathematics* **62**:1, pp. 61–89. ISSN: 0020-3157. DOI: 10.1007/s10463-009-0236-2. URL: http://dx.doi.org/10.1007/s10463-009-0236-2.

Bunch, P. and S. Godsill (2013). "Improved particle approximations to the joint smoothing distribution using Markov Chain Monte Carlo". *Signal Processing, IEEE Transactions on* **61**:4, pp. 956–963.

Continuum Analytics (2014). *Numba, Version 0.14*. URL: http://numba.pydata.org/numba-doc/0.14/index.html.

Del Moral, P., A. Doucet, and A. Jasra (2006). "Sequential Monte Carlo samplers". *Journal of the Royal Statistical Society B (Statistical Methodology)* **68**:3, pp. 411–436.

Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). "Maximum likelihood from incomplete data via the EM algorithm". *Journal of the Royal Statistical Society B* **39**:1, pp. 1–38.

Doucet, A., S. Godsill, and C. Andrieu (2000). "On sequential Monte Carlo sampling methods for Bayesian filtering". English. *Statistics and Computing* **10**:3, pp. 197–208. ISSN: 0960-3174. DOI: 10.1023/A:1008935410038. URL: http://dx.doi.org/10.1023/A:1008935410038.

Dubarry, C. and R. Douc (2011). "Particle approximation improvement of the joint smoothing distribution with on-the-fly variance estimation". *arXiv preprint arXiv:1107.5524*.

FSF (1999). *The GNU Lesser General Public License*. See http://www.gnu.org/copyleft/lesser.html.

Jones, E., T. Oliphant, P. Peterson, et al. (2001–). *SciPy: open source scientific tools for Python*. URL: http://www.scipy.org/.

Julier, S. and J. Uhlmann (2004). "Unscented filtering and nonlinear estimation". *Proceedings of the IEEE* **92**:3, pp. 401–422. ISSN: 0018-9219. DOI: 10.1109/JPROC.2003.823141.

Lindsten, F. and T. Schön (2010). "Identification of mixed linear/nonlinear state-space models". In: *Decision and Control (CDC), 2010 49th IEEE Conference on*, pp. 6377–6382. DOI: 10.1109/CDC.2010.5717191.

Lindsten, F. and T. Schön (2011). *Rao-Blackwellized Particle Smoothers for Mixed Linear/Nonlinear State-Space Models*. Tech. rep. URL: http://user.it.uu.se/~thosc112/pubpdf/lindstens2011.pdf.

Lindsten, F., P. Bunch, S. J. Godsill, and T. B. Schön (2013). "Rao-Blackwellized particle smoothers for mixed linear/nonlinear state-space models". In: *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, pp. 6288–6292.

Lindsten, F. and T. Schön (2013). "Backward simulation methods for Monte Carlo statistical inference". *Foundations and Trends in Machine Learning* **6**:1, pp. 1–143. ISSN: 1935-8237. DOI: 10.1561/2200000045. URL: http://dx.doi.org/10.1561/2200000045.

Mannesson, A. (2013). *Joint Pose and Radio Channel Estimation*. Licentiate Thesis. Dept. of Automatic Control, Lund University, Sweden.

Montemerlo, M., S. Thrun, D. Koller, B. Wegbreit, et al. (2002). "FastSLAM: a factored solution to the simultaneous localization and mapping problem". In: *AAAI/IAAI*, pp. 593–598.

Murray, L. M. (In review). "Bayesian state-space modelling on high-performance hardware using LibBi". URL: http://arxiv.org/abs/1306.3277.

Nordh, J. (2013). *PyParticleEst - Python library for particle based estimation methods*. URL: `http://www.control.lth.se/Staff/JerkerNordh/pyparticleest.html`.

Okuma, K., A. Taleghani, N. De Freitas, J. J. Little, and D. G. Lowe (2004). "A boosted particle filter: multitarget detection and tracking". In: *Computer Vision-ECCV 2004*. Springer-Verlag, pp. 28–39.

Oliphant, T. E. (2007). "Python for scientific computing". *Computing in Science Engineering* **9**:3, pp. 10–20. ISSN: 1521-9615. DOI: `10.1109/MCSE.2007.58`.

Rauch, H. E., C. T. Striebel, and F. Tung (1965). "Maximum likelihood estimates of linear dynamic systems". *Journal of the American Institute of Aeronautics and Astronautics* **3**:8, pp. 1445–1450.

Rebeschini, P. and R. van Handel (2013). "Can local particle filters beat the curse of dimensionality?" *arXiv preprint arXiv:1301.6585*.

Rigo, A. (2004). "Representation-based Just-in-Time specialization and the psyco prototype for Python". In: *Proceedings of the 2004 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*. ACM, Verona, Italy, pp. 15–26. ISBN: 1-58113-835-0. DOI: `10.1145/1014007.1014010`. URL: `http://portal.acm.org/citation.cfm?id=1014010`.

Schön, T., F. Gustafsson, and P.-J. Nordlund (2005). "Marginalized particle filters for mixed linear/nonlinear state-space models". *Signal Processing, IEEE Transactions on* **53**:7, pp. 2279–2289. ISSN: 1053-587X. DOI: `10.1109/TSP.2005.849151`.

Taghavi, E., F. Lindsten, L. Svensson, and T. Schön (2013). "Adaptive stopping for fast particle smoothing". In: *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 6293–6297. DOI: `10.1109/ICASSP.2013.6638876`.

Zhu, C., R. H. Byrd, P. Lu, and J. Nocedal (1997). "Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization". *ACM Transactions on Mathematical Software (TOMS)* **23**:4, pp. 550–560.

Paper II

A Quantitative Evaluation of Monte Carlo Smoothers

Jerker Nordh Jacob Antonsson

Abstract

In this paper we compare the performance of several popular methods for particle smoothing to investigate if any of the algorithms can deliver better performance for a given computational complexity. We use four different models for the evaluation, chosen to illuminate the differences between the methods. When comparing the computational cost we avoid the issues of implementation efficiency by instead counting the number of evaluations required for a set of high level primitives that are common for all the algorithms. Our results give some insight into the performance characteristics of the chosen methods, even though no universal best choice can be identified since the cost/performance ratios of the methods depend on the characteristics of the problem to be solved which can be clearly seen in the results.

Submitted to the *EURASIP Journal on Advances in Signal Processing*

1. Introduction

We are concerned with the problem of inferring the latent states $x_{1:T} \triangleq \{x_t\}_{t=1}^T$, $x_t \in \mathbb{R}^n$, given a set of observations $y_{1:T} \triangleq \{y_t\}_{t=1}^T$, $y_t \in \mathbb{R}^m$ from a state-space model,

$$x_{t+1} \sim p(x_{t+1}|x_t), \tag{1a}$$
$$y_t \sim p(y_t|x_t), \tag{1b}$$
$$x_1 \sim p(x_1). \tag{1c}$$

Specifically, we want to estimate the *joint smoothing density* (JSD), $p(x_{1:T}|y_{1:T})$ for such a system. An estimate of the JSD can be found by the particle filter, which approximates the marginal densities $p(x_t|y_{1:T}), t \leq T$ of the JSD by a set of weighted samples, called particles. The particle filter is a sequential Monte Carlo (SMC) algorithm applied to state-space models. Due to the degeneracy property [Doucet and Johansen, 2009] of the particle filter, the amount of unique samples approximating the marginal densities are decreasing with decreasing t; the approximations are said to lack particle diversity. The filter density, $p(x_t|y_{1:t})$, estimates are diverse in the particles however, and they can be used to simulate realizations, usually called backward trajectories, from the JSD. This is the idea of the forward filter backward simulator (FFBSi) algorithm [S. J. Godsill et al., 2004]. Let N and M be the number of particles and backward trajectories respectively. The FFBSi has $\mathcal{O}(MN)$ complexity [S. J. Godsill et al., 2004], which can be prohibitive. Asymptotically in N, the complexity can be reduced to $\mathcal{O}(N)$ by the use of rejection sampling, giving the FFBSi-RS algorithm, proposed by [Douc et al., 2011].

Another approach, developed by [Dubarry and Douc, 2011], among others, is to use the degenerate particle approximations of the JSD in a Metropolis within Gibbs sampler to get univariate realizations from all time points of the smoothed trajectory. In each step of the Gibbs sampler, the state at time t is sampled using Metropolis Hastings (MH) sampling. This algorithm is known as the Metropolis Hastings improved particle smoother (MH-IPS).

A couple of algorithms based on using the degenerate particle filter approximation of the JSD as a pseudo likelihood in a Markov chain Monte Carlo (MCMC) sampler have also been proposed, yielding the particle Gibbs (PG), and the particle Metropolis Hastings algorithms. It was shown by [Andrieu et al., 2010] that such samplers do indeed target the JSD. The samplers originally proposed by [Andrieu et al., 2010] have been improved by the use of backward simulation by for example [Lindsten and Schön, 2012; Lindsten et al., 2014; Olsson and Ryden, 2011], giving the particle Gibbs with backward simulation (PGBS), particle Gibbs with ancestor sampling (PGAS) and

particle Metropolis Hastings (PMMH-BS) algorithms. The particle MCMC methods generally have higher complexity than the aforementioned smoothing algorithms, but it has been proposed [Lindsten and Schön, 2013] that they might be more efficient, and can give better approximations of the JSD for fewer particles.

These algorithms have a lot of subtle differences and none have been shown to clearly perform better than the others for smoothing in general state space models. We here try to investigate the performance of the different types of smoothing methods for some state space models commonly used in the literature. We also include a model whose transition kernel (1a) does not admit a density, a common case in practice. One algorithm from each type of method is chosen to, hopefully, reflect the variety between them. The methods chosen are FFBSi-RS, with an additional improvement of adaptive early stopping, proposed by [Taghavi et al., 2013], MH-IPS and PGAS. We compare the algorithms for different parameter choices using a novel approach which is independent of implementation and computer system. The theory of these algorithms will not be detailed, we refer the comprehensive introduction and review made by [Lindsten and Schön, 2013], and the original articles for further details.

Smoothing in state space models is an important topic in its own right, but it also has many applications. The smoothing distributions are for example a key component in several parameter estimation algorithms and can thus be used to find the posterior distributions of parameters in general state space models, or to learn the whole model itself. For example [Lindsten et al., 2013] uses PMCMC, and [Wills et al., 2013] uses an FFBSi-RS smoother together with an expectation maximization procedure, for identification of Hammerstein-Wiener systems.

2. Theoretical Preliminaries

In this section we give a brief review of the particle filter which is the basis for all the other algorithms. We also briefly discuss the backward kernel for state space models, as both the FFBSi-RS and PGAS uses it to simulate from the JSD.

2.1 The Particle Filter

The particle filter is an SMC sampler targeted at the marginal posterior densities $p(x_t|y_{1:t}), \forall t \in \{1, \ldots, t\}$, which are approximated as discrete distributions,

$$p(x_t|y_{1:t}) \approx \sum_{i=1}^{N} w_t^i \delta \left(x_t - x_t^i\right). \tag{2}$$

The sets of weighted particles targeting the filtering densities are calculated by sequential importance sampling. Given a weighted set of particles $\{w_{t-1}^i, x_{t-1}^i\}_{i=1}^N$ approximating $p(x_{t-1}|y_{1:t-1})$, a set of particles approximating $p(x_t|y_{1:t})$ can be found by sampling from an importance distribution, $q(x_t|x_{t-1}^i, y_t)$, chosen to satisfy the Markov property, and the weights can be calculated as

$$w_t^i \propto w_{t-1}^i \frac{p(y_t|x_t^i)p(x_t^i|x_{t-1}^i)}{q(x_t^i|x_{t-1}^i, y_t)},$$

after which they are normalized to sum to unity. The approach leads to an approximation with a variance that will increase toward infinity for increasing t [Doucet and Johansen, 2009]. This will gradually shift all weight to one particle, leading to an effective approximation size of only one sample. To mitigate this a resampling step is introduced when the number of effective particles, measured by the effective sample size statistic $ESS(t) \approx 1/\sum(w_t^i)^2$, described by [Doucet and Johansen, 2009], is too low. A rule of thumb is to resample if $ESS < 2N/3$, where N as before is the number of particles. The resampling is done by sampling new equally weighted particles from the existing set of particles, where the probability of selecting the i:th particle is w_t^i, giving samples distributed according to the empirical distribution (2). This procedure is encoded in ancestor indices a_t^i that keeps track of which previous sample the particle x_t^i originates from.

The simplest choice of importance distribution, which we use and which is often used in practice, is to use the transition kernel (1a), giving the bootstrap filter outlined in Algorithm 1. [Doucet and Johansen, 2009] gives more details of the particle filter and SMC for state-space models.

2.2 The Backward Kernel

The key to a lot of the smoothing algorithms for (1), such as FFBSi and PGAS, is the backward kernel,

$$p(x_t|x_{t+1}, y_{1:T}) = \frac{p(x_{t+1}|x_t)p(x_t|y_{1:t})p(x_{t+1}|y_{1:T})}{p(x_{t+1}|y_{1:t})}, t \leq T. \tag{3}$$

This expression is dependent on the filter distribution which can be approximated with high particle diversity by the particle filter. Mark particles representing the smoothed distribution with a tilde. Using (2) in (3), an empirical approximation to the backward kernel is given by,

$$p(x_t|\tilde{x}_{t+1}, y_{1:T}) \approx \sum_{i=1}^{N} w_{t|T}^i \delta\left(x_t - x_t^i\right), \tag{4}$$

with smoothing weights calculated as

$$w_{t|T}^i \propto w_t^i p(\tilde{x}_{t+1}|x_t^i).$$

Algorithm 1 Bootstrap Particle Filter for (1)

Input: Measurements $\{y_t\}_{t=1}^T$.
Output: Particle system $\{w_t^i, x_t^i\}_{i=1}^N, t \in \{1,\dots,T\}$ and ancestor indices $\{a_t^i\}_{i=1}^N, t \in \{2,\dots,T\}$ approximating the marginal filter- and the degenerate smoothing distributions.

for $i = 1$ to N **do**
  Sample x_1^i from $p(x_1)$.
  Compute $\tilde{w}_1^i = p(y_1|x_1^i)$
end for
Set $w_1^i = \tilde{w}_1^i / (\sum_{j=1}^N \tilde{w}_1^j), \quad \forall i \in \{1,\dots,N\}$.
for $t = 1$ to $T-1$ **do**
  for $i = 1$ to N **do**
    if $ESS(t) < 2N/3$ **then**
      Sample ancestor indices, a_{t+1}^i, with $\mathbb{P}(a_{t+1}^i = j) = w_t^j, j \in \{1,\dots,N\}$.
    else
      $a_{t+1}^i = i$
    end if
    Sample x_{t+1}^i from $p(x_{t+1}|x_t^{a_{t+1}^i})$
    Compute $\tilde{w}_{t+1}^i = p(y_{t+1}|x_{t+1}^i)$
  end for
  Set $w_{t+1}^i = \tilde{w}_{t+1}^i / (\sum_{j=1}^N \tilde{w}_{t+1}^j), \quad \forall i \in \{1,\dots,N\}$.
end for

Smoothed samples can then be drawn using the following recursive sampling scheme; starting at $t = T$ and iterating until $t = 1$, all the time conditioning on the already sampled future states $\tilde{x}_{t+1:T}$,

$$\text{Sample } \tilde{x}_T \text{ with } \mathbb{P}\left(\tilde{x}_T = x_T^i\right) = w_T^i, \tag{5a}$$

$$\text{sample } \tilde{x}_t \text{ with } \mathbb{P}\left(\tilde{x}_t = x_t^i\right) = w_{t|T}^i. \tag{5b}$$

3. Algorithms

We chose three algorithms to represent the diversity of particle smoothing algorithms: FFBSi-RS, MHIPS and PGAS. Here we briefly discuss these algorithms and list them as they are implemented. For details and theoretical results we refer to references given in the introductory section of the paper.

3.1 Forward Filtering Backward Simulation with Rejection Sampling - FFBSi-RS

The FFBSi-RS algorithm uses the recursive scheme (5) to sample smoothed trajectories. However, to avoid evaluating all the smoothing weights (2.2), it samples from the categorical distribution defined by $w_{t|T}^i,\ i \in \{1,\dots,N\}$,

using rejection sampling. It uses the filter weights, $w_t^i,\ i \in \{1, \ldots, N\}$, as proposal distribution. When the number of particles is large, this renders the complexity of the algorithm approximately linear in the number of particles [Douc et al., 2011]. However, the rejection sampler can get stuck for more improbable realizations of the backward trajectories. We therefore use a hybrid scheme where the rejection sampler is run a certain number of iterations, and if needed the final realizations are drawn by evaluating the smoothing weights explicitly. The stopping rule for the rejection sampler is adaptive and depends on an estimate of the average acceptance probability, as suggested by [Taghavi et al., 2013]. The algorithm used is outlined in Algorithm 2. The expression for the acceptance probability is derived by [Douc et al., 2011] in the original article. Our notation approximately follows that of [Lindsten and Schön, 2013], Cat$(\cdot)$ for instance, refers to the categorical distribution. Rejection sampling is treated exhaustively by [Robert and Casella, 2013].

3.2 Metropolis Hastings Improved Particle Smoother - MH-IPS

We now move over to two algorithms that combine the inference approaches of SMC and MCMC. The first one, the MH-IPS, samples from the trajectory using Gibbs sampling by sampling the state at each time-step t separately, while conditioning on the rest of the trajectory. For a Markovian model like (1), the smoothed states x_t should thus be sampled from

$$p(x_t|x_{1:t-1}, x_{t+1:T}, y_{1:T}) \propto f(x_{t+1}|x_t)g(y_t|x_t)f(x_t|x_{t-1}).$$

This is done by MH sampling. A sample x_t' is drawn from a proposal distribution
$q(x_t|x_{t-1}, x_{t+1}, y_{1:T})$ and accepted with probability,

$$1 \wedge \frac{p(x_{t+1}|x_t')}{p(x_{t+1}|x_t)} \frac{p(y_t|x_t')}{p(y_t|x_t)} \frac{p(x_t'|x_{t-1})}{p(x_t|x_{t-1})} \frac{q(x_t|x_{t+1}, y_t, x_{t-1})}{q(x_t'|x_{t+1}, y_t, x_{t-1})}, \tag{6}$$

where $\wedge$ is the min-operator. We use the proposal density

$$q(x_t|x_{t-1}, x_{t+1}, y_{1:T}) = p(x_t|x_{t-1}), \tag{7}$$

which simplifies the acceptance probability (6) and gives Algorithm 3. [Dubarry and Douc, 2011] outlines MH-IPS in greater generality, and with more theoretical details. [Robert and Casella, 2013] provides more details on MH and Gibbs sampling.

3.3 Particle Gibbs Ancestral Sampling - PGAS

We saw that MH-IPS updates each state in the trajectory univariately, which can lead to poor mixing [Lindsten and Schön, 2013]. In contrast, PG samples whole trajectories at each update. The trajectories are sampled from a set

Algorithm 2 FFBSi with rejection sampling and early stopping, for (1). For details of the stop criterion see [Taghavi et al., 2013]

Input: Forward filter particle system $\{w_t^i, x_t^i\}_{i=1}^N, t \in \{1, \ldots, T\}$.
Output: M realizations from the JSD.

Sample indices $\{b_T(j)\}_{j=1}^M$ from Cat $\left(\{w_T^i\}_{i=1}^N\right)$
for $j = 1$ to M **do**
  $\tilde{x}_T^j = x_T^{b_T(j)}$
end for
for $t = T - 1$ to 1 **do**
  $L \leftarrow \{1, \ldots, M\}$
  $\rho_t = \max_i \operatorname{argmax}_{x_{t+1}} p(x_{t+1}|x_t^i)$
  while length$(L)\neg 0$ or stop criterion not fulfilled **do**
    $n \leftarrow \text{length}(L)$
    $\delta \leftarrow \emptyset$
    Sample $\{I(k)\}_{k=1}^n$ from Cat $\left(\{w_t^i\}_{i=1}^N\right)$
    Sample $\{U(k)\}_{k=1}^n$ from $\mathcal{U}([0,1])$
    for $k = 1$ **to** n **do**
      if $U(k) \leq p(\tilde{x}_{t+1}^{L(k)}|x_t^{I(k)})/\rho_t$ **then**
        $b_t(L(k)) \leftarrow I(k)$
        $\delta \leftarrow \delta \cup \{L(k)\}$
      end if
    end for
    $L \leftarrow L \setminus \delta$
  end while
  for $j = 1$ **to** length(L) **do**
    Compute $\tilde{w}_{t|T}^{i,j} \propto w_t^i p(\tilde{x}_{t+1}^j|x_t^i),\ i \in \{1, \ldots, N\}$.
    $w_{t|T}^{i,j} = \tilde{w}_{t|T}^{i,j}/(\sum_i \tilde{w}_{t|T}^{i,j}),\ i \in \{1, \ldots, N\}$.
    Sample $b_t(L(j))$ from Cat $\left(\{w_{t|T}^{i,j}\}_{i=1}^N\right)$
  end for
  for $j = 1$ **to** M **do**
    Set $\tilde{x}_t^j = x_t^{b_t(j)}$ and $\tilde{x}_{t:T}^j = \{\tilde{x}_t^j, \tilde{x}_{t+1:T}^j\}$.
  end for
end for

of degenerate SMC approximations of the smoothed trajectory. Invariance of the sampler is achieved by doing the SMC sampling conditioned on a fixed trajectory. Due to the degeneracy of the trajectories from the SMC pass, the PG sampler can also suffer from bad mixing [Lindsten et al., 2014]. In PGAS this is alleviated by splitting the reference trajectory at each time point t, by assigning a new history to the future trajectory. This is done by sampling a new ancestor index, a_t^i, with probability given by the smoothing weights (2.2). This keeps the invariance of the PG sampler and improves mixing. The algorithm we use, with the same choices for proposal density as

Algorithm 3 MHIPS with MH proposal (7) for model (1), performing R iterations of the Gibbs sampler

Input: Degenerate smoothing trajectories $\{w_T^i, x_{1:T}^i\}_{i=1}^N$.
Output: Improved smoothing trajectories $\{\tilde{x}_{1:T}^i\}_{i=1}^M$.

Initialize $\{\tilde{x}_{1:T}^j\}_{j=1}^M$ by sampling from the ancestral paths of the forward filter, drawing each trajectory with probability w_T^i.
for $r = 1$ to R **do**
  for $j = 1$ to M **do**
    Modifed acceptance probability for last time-step
    Sample ${x'}_T^j \sim p(x_T|\tilde{x}_{T-1})$
    With probability $1 \wedge \frac{p(y_T|x'_T)}{p(y_T|\tilde{x}_T)}$, set $\tilde{x}_T^j = {x'}_T^j$
    for $t = T-1$ to 2 **do**
      Sample ${x'}_t^j \sim p(x_t|\tilde{x}_{t-1})$
      With probability $1 \wedge \frac{p(\tilde{x}_{t+1}|x'_t)}{p(\tilde{x}_{t+1}|\tilde{x}_t)} \frac{p(y_t|x'_t)}{p(y_t|\tilde{x}_t)}$, set $\tilde{x}_t^j = {x'}_t^j$
    end for
    Modifed proposal distribution for first time-step
    Sample ${x'}_1^j \sim p(x_1)$
    With probability $1 \wedge \frac{p(\tilde{x}_2|x'_1)}{p(\tilde{x}_2|\tilde{x}_1)} \frac{p(y_1|x'_1)}{p(y_1|\tilde{x}_1)}$, set $\tilde{x}_1^j = {x'}_1^j$
  end for
end for

in Algorithm 1, is listed in Algorithm 4. Further details and theory is given by [Lindsten et al., 2014]. A related sampler is the Particle Gibbs Backward Simulator (PGBS) developed by [Lindsten and Schön, 2012; Lindsten and Schön, 2013].

4. Models

We chose four different models to evaluate the smoothing algorithms. We use a standard one-dimensional nonlinear model that is commonly seen used in the particle filtering and smoothing literature for benchmarking algorithms. A model often used in two-dimensional tracking applications is also included to test how the algorithms fare in a higher dimensional scenario. Especially interesting in that regard is the FFBSi-RS algorithm, since rejection sampling notoriously gets trickier for higher dimensions.

It is not uncommon for models used in engineering to have degenerate transition kernels. This is for example true for orientation filter models, used for finding the orientation of a physical body using inertial measurement sensors, and other types of dynamical first principles models [Gustafsson, 2010]. Such models impose a further difficulty for the smoothing algorithms, and are very common in practice. As an example of such a model we therefore include a double integrator. It is included both in its degenerate formulation

Algorithm 4 PGAS kernel for (1)

Input: Measurements and conditional trajectory $\{y_t, x'_t\}_{t=1}^T$
Output: Smoothed trajectories $\{\tilde{x}_{1:T}^i\}_{i=1}^M$ and intermediate quantities such as the same particle systems given by the particle filter.

for $i = 1$ to $N-1$ **do**
  Sample $\tilde{x}_1^i$ from $p(x_1)$.
  Compute $\tilde{w}_1^{(i)} = p(y_1|\tilde{x}_1^i)$
  Set $\tilde{x}_1^N = x'_1$.
end for
Set $w_1^i = \tilde{w}_1^i / (\sum_{j=1}^N \tilde{w}_1^j), \quad \forall i \in [1, N]$.
for $t = 1$ to $T-1$ **do**
  for $i = 1$ to $N-1$ **do**
    Sample ancestor indices, a_{t+1}^i, with $\mathbb{P}(a_{t+1}^i = j) = w_t^j, j \in [1, N]$.
    Sample $\tilde{x}_{t+1}^i$ from $p(x_{t+1}|\tilde{x}_t^{a_{t+1}^i})$
  end for
  Set $\tilde{x}_{t+1}^N = x'_{t+1}$.
  Sample a_{t+1}^N with $\mathbb{P}\left(a_{t+1}^N = i\right) = \frac{w_t^i p(x'_{t+1} \mid \tilde{x}_t^i)}{\sum_{l=1}^N w_t^l p(x'_{t+1} \mid \tilde{x}_t^l)}$
  for $i = 1$ to N **do**
    Set $\tilde{x}_{1:t+1}^i = \{\tilde{x}_{1:t}^{a_t^i}, \tilde{x}_{t+1}^i\}$
    Compute $w_{t+1}^i = p(y_{t+1}|\tilde{x}_{t+1}^i)$
  end for
end for

and as a non-Markovian model, that is the result of marginalizing over the deterministic state. The following subsections state the models in some more detail.

4.1 Standard Benchmark Model

The model

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

$$y_t = 0.05x_t^2 + e_t, \quad x_1 \sim \mathcal{N}(0, 5), \tag{8b}$$

$$w_t \sim \mathcal{N}(0, 10), \quad e_t \sim \mathcal{N}(0, 1), \tag{8c}$$

where $\mathcal{N}(m, P)$ is the Gaussian distribution with mean m and covariance P, is commonly used for evaluating filter and smoother performance. It has been used by for example [Lindsten and Schön, 2013; Arulampalam et al., 2002; S. J. Godsill et al., 2004] as a benchmark model. For such a model the filtering and smoothing distributions are multi-modal, which makes algorithms that assume a fixed distribution on the posterior, such as the extended Kalman filter [Särkkä, 2013], perform very poorly.

4.2 Double Integrator

An example of a state space model with a degenerate transition kernel is the double integrator

$$x_{t+1} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} x_t + \begin{pmatrix} 0 \\ 1 \end{pmatrix} w_t, \tag{9a}$$

$$y_t = \begin{pmatrix} 1 & 0 \end{pmatrix} x_t + e_t, \tag{9b}$$

$$w_t \sim \mathcal{N}(0, Q), \quad e_t \sim \mathcal{N}(0, R). \tag{9c}$$

Due to the noise only acting on the input there is a deterministic relation between the two states making the model degenerate and not suitable for the standard particle smoothing methods. This coupling means that $p(x_{t+1}^i | x_t^j) = 0, \ \forall j \neq a_t^i$, where a_t^i is the index of the ancestor for particle x_{t+1}^i.

The model in (9) can be rewritten as a first order system with a non-Markovian structure. For notational brevity introduce the notation $x_t = (p_t \quad v_t)^T$. The model can then be rewritten as

$$v_{t+1} = v_t + w_t, \tag{10a}$$

$$y_t = p_1 + \sum_{i=1}^{t-1} v_i + e_t, \tag{10b}$$

$$w_t \sim \mathcal{N}(0, Q), \quad e_t \sim \mathcal{N}(0, R), \tag{10c}$$

and the smoothing problem can be solved using a non-Markovian particle smoother [Lindsten and Schön, 2013]. For this particular model it is possible to reduce the computational effort by propagating additional information in the forward and backward steps of the algorithms. Writing the model (10) as

$$v_{t+1} = v_t + w_t, \tag{11a}$$

$$s_{t+1} = s_t + v_t, \tag{11b}$$

$$y_t = s_t + e_t, \tag{11c}$$

$$s_0 = p_0, \tag{11d}$$

$$w_t \sim \mathcal{N}(0, Q), \quad e_t \sim \mathcal{N}(0, R), \tag{11e}$$

it is clear that during the filtering step, each particle also stores the sum of all its previous states. At a quick glance this looks like simply reintroducing the p-state from the original model, but the key distinction is that this new

variable is a function of the past trajectory, and not included as a state in the model.

The smoothing weights for the model (11) are computed using the density

$$\prod_{k=t+1}^{T} p(y_k|v_k)p(v_k|v_{1:k-1}, y_{1:k-1}) \tag{12}$$

$$\underset{v_{1:t}}{\propto} p(y_{t+1:T}|v_{1:T})p(v_{t+1}|v_t), \tag{13}$$

as noted by [Lindsten and Schön, 2013]. Evaluating this directly leads to a computational effort for each time-step that grows with the length of the full dataset. This is clearly undesirable, but using the same approach as [Lindsten and Schön, 2013], and noticing that (13) only needs to be evaluated up to proportionality (with regard to $v_{1:t}$) it is possible to propagate information backwards during the smoothing in the same way as the s_t variables propagate the sum during filtering. The first factor of (13) can be evaluated up to proportionality as follows,

$$p(y_{t+1:T}|s_t, v_{t:T}) = \prod_{k=t+1}^{T} p(y_k|s_t, v_{t:T}) \tag{14a}$$

$$\underset{s_t, v_t}{\propto} \prod_{k=t+1}^{T} e^{(s_t+v_t)^2 - 2(y_k - \sum_{j=t+1}^{k-1} v_j)(s_t+v_t)} \tag{14b}$$

$$= e^{(T-t)(s_t+v_t)^2 - 2\sum_{k=t+1}^{T}(y_k - \sum_{j=t+1}^{k-1} v_j)(s_t+v_t)}. \tag{14c}$$

Through the introduction of two new variables N_t, γ_t that are propagated backwards during the smoothing this allows (14) to be evaluated as

$$\log p(y_{t+1:T}|s_t, v_t, v_{t+1:T}) + \text{constant} = \frac{1}{2R}(N_{t+1}(s_t + v_t)^2 - 2\gamma_{t+1}(s_t + v_t)), \tag{15a}$$

$$N_t = N_{t+1} + 1, \quad N_T = 1, \tag{15b}$$

$$\gamma_t = \gamma_{t+1} + y_t - N_{t+1}v_t, \quad \gamma_T = y_T. \tag{15c}$$

Using (15) it is now possible to evaluate the required smoothing density in constant time. A more detailed derivation of these expressions for this particular model is given by [Nordh, 2015, Submitted].

4.3 Tracking Model

To test how the algorithms fare in a higher dimensional setting we have included a four dimensional model commonly used for tracking. The model

was also used to test a new FFBSi smoother, similar to MHIPS, in [Bunch and S. Godsill, 2013]. Define the state vector of positions and velocities in two dimensions, $x_t = \begin{pmatrix} x_t & y_t & \dot{x}_t & \dot{y}_t \end{pmatrix}$, let $h = 0.1$ be the sampling time, I_2 the identity matrix in $\mathbb{R}^{2\times 2}$, $0_{2\times 2}$ the null matrix in $\mathbb{R}^{2\times 2}$, and 0_n the null vector in $\mathbb{R}^n$ respectively. The model is then given by,

$$x_{t+1} = \begin{pmatrix} I_2 & hI_2 \\ 0_{2\times 2} & I_2 \end{pmatrix} x_t + w_t, \tag{16a}$$

$$y_t = \begin{pmatrix} \operatorname{atan2}(y_t, x_t) \\ \sqrt{x_t^2 + y_t^2} \end{pmatrix} + e_t, \tag{16b}$$

$$w_t \sim \mathcal{N}\left(0_4, \begin{pmatrix} \frac{h^3}{3} I_2 & \frac{h^2}{3} I_2 \\ \frac{h^2}{3} I_2 & hI_2 \end{pmatrix}\right), \tag{16c}$$

$$e_t \sim \mathcal{N}\left(0_2, \begin{pmatrix} \left(\frac{\pi}{720}\right)^2 & 0 \\ 0 & 0.1 \end{pmatrix}\right), \tag{16d}$$

where $\operatorname{atan2}(\cdot,\cdot)$ is the four-quadrant inverse tangent.

5. Method

To compare the smoothing performance of the algorithms we use the Root Mean Square Error (RMSE) metric which is a standard choice in the literature. It is however flawed in that it only compares the estimated mean value to the true value, whereas the goal of a particle smoother is to provide a good approximation of the JSD. The fact that the average of the posterior density most of the time will not coincide with the true state means that there is a problem specific lower bound for the RMSE. This is related to the Cramér-Rao Lower Bound (CRLB)[Tichavsky et al., 1998] which provides a bound on the achievable performance of any estimator for a given problem. In theory all the methods examined in this paper should reach the same RMSE since they all converge to the true posterior distribution as the quality of the approximations are increased. To reduce the variance of the measured RMSE metric we compute the average over a large number of realizations from each model. To ensure a fair comparison for the different algorithms they are all given the exact same set of realizations. Denote, as before, the number of realizations as N and the length of each realization as T, the true value as x and the estimated mean value as $\bar{x}$. The average RMSE is then calculated as

$$\frac{1}{N}\sum_{i=1}^{N}\sqrt{\frac{1}{T}\sum_{t=1}^{T}(\bar{x}_t - x_t)^2}.$$

The standard error of the average RMSE is also computed to determine if the observed differences between the methods are statistically significant.

Our goal is not only to investigate which methods can provide the lowest RMSE for a given problem, but also to analyze the computation effort required to do so. To avoid any issues arising from the efficiency of our implementation, the execution time is not used as the metric of computation effort. The methods are all based on a few common operations. We therefore use the number of times these operations are performed by the algorithms as a complexity measure. The high-level primitives are listed in Table 1. In the presented graphs all the operations are equally weighted to give a quick overview, in practice the relative cost of the operations are both problem and implementation specific. The level of parallelization that can be exploited, both in the hardware and software, and which simplifying assumptions that can be made about the model will greatly affect the cost of each operation. Therefore we chose to not include the actual execution times for the algorithms as that would be more a benchmark of our implementation than of the methods themselves. This is especially true in our case since we used a flexible generic framework to test the methods. This facilitates implementation to a great degree, but causes an overhead in terms of execution time, since the framework does not exploit model specific characteristics to increase speed, such as knowledge about which matrices are time-invariant, sparsity structures and so on. As a drastic example consider an embedded system where for performance reasons it is decided to sample from distributions by simply iterating over a pre-calculated array with numbers. The relative cost of the operations in that scenario will differ drastically compared to an implementation running on a desktop system and where accuracy and correctness might be more important than execution time. The interested reader can download the raw results together with the source code from the homepage of [Nordh and Antonsson, 2013] and make their own analysis for their particular target platform.

6. Results

The mean of the RMSE together with a band of 2 standard errors on both sides is plotted as a function of computation effort, encoded in numer of high-level primitive operations, for all the methods and a few different parameter choices are presented in Figure 1 for the standard nonlinear model, Figure 2 for the tracking example, Figure 3 for the degenerate double integrator and in Figure 4 for the marginalized double integrator.

For the marginalized double integrator model we use the normal FFBSi-smoother instead of rejection sampling. This is because we can only evaluate the required probability densities up to proportionality and we can therefore

Table 1. List of common high-levels operations for all the algorithms used. The number of times each operation is performed is compared for the different algorithms and used as the computational cost

Sample from $p(x_{t+1}\|x_t)$	Used in e.g the particle filter when propagating the estimates forward
Evaluate $p(y_t\|x_t)$	Used in e.g the particle filter when updating the particle weights
Evaluate $p(x_{t+1}\|x_t)$	Used in most of the smoothers to update the weights of particles given information about the future trajectories
Compute $\underset{x_{t+1}}{\text{argmax}}\ p(x_{t+1}\|x_t)$	Used for rejections sampling in combination with FFBSi smoothers
Sample from $p(x_1)$	Draw particles from the initial distribution

not compute the required normalization factor. Additionally, in theory the rejection sampling would work very poorly since it is actually sampling the full ancestral path in each step, $x_{1:t}$, which is a very high-dimensional problem and therefore not suitable for rejection sampling.

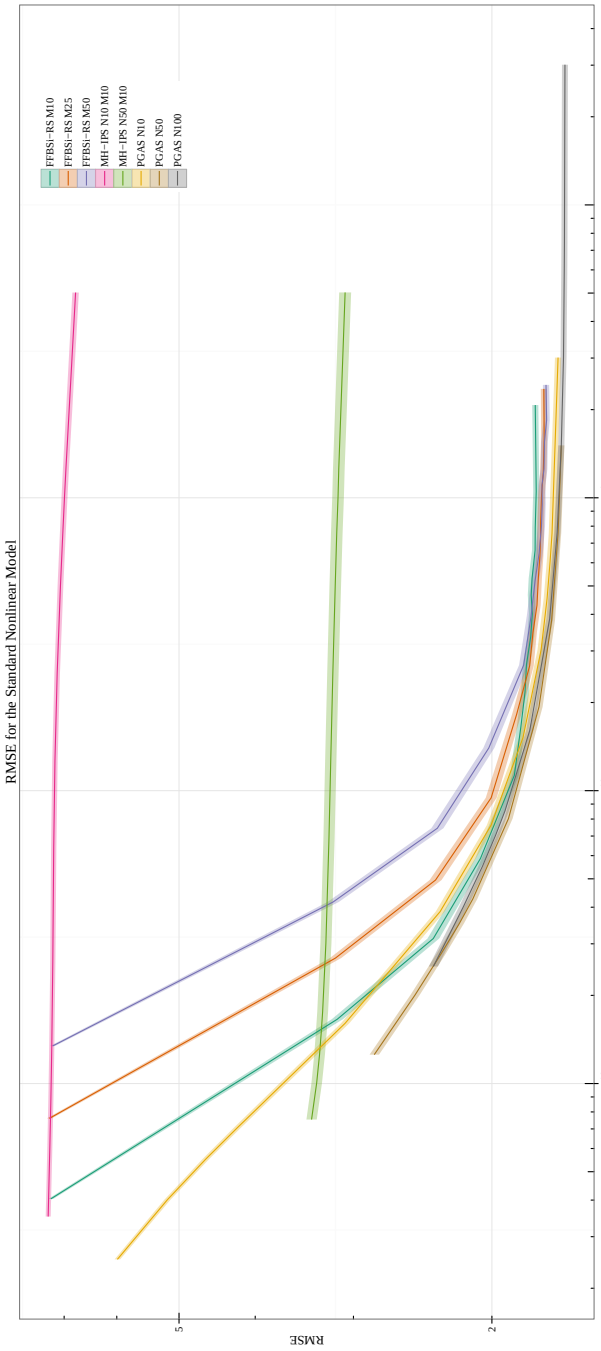

Figure 1. RMSE as a function of the computation effort for all methods applied to the standard nonlinear model (8). The mean of the RMSE is plotted together with a band of two standard errors on both sides. For the FFBSi-RS plots, the number of backward trajectories are held fixed at 10, 25 and 50, while the number of particles used in the forward filter is increased. For the two MH-IPS cases the number of particles in the forward filter is either 10 or 50. In both cases there are 10 backward trajectories. The computational complexity is varied by changing the number of performed MH-IPS passes. For the PGAS plots, the number of particles used in the filter is either 10, 50 or 100, and the computational complexity is varied by changing the number of iterations.

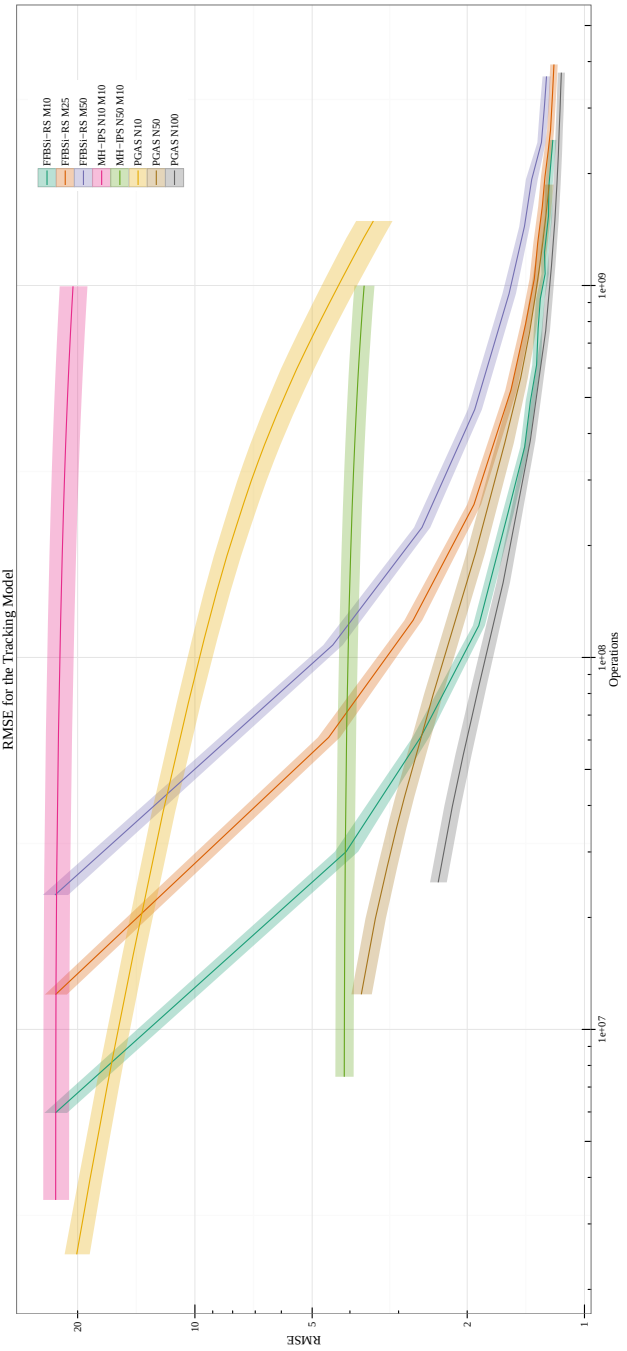

Figure 2. RMSE as a function of the computation effort for all methods applied to the tracking model (16). The notation is the same as in Figure 1. As can be seen, the MH-IPS performs poorly for this model, yielding almost no visible improvements as the number of iterations, R, is increased. In this particular case, PGAS seems to be the clear winner.

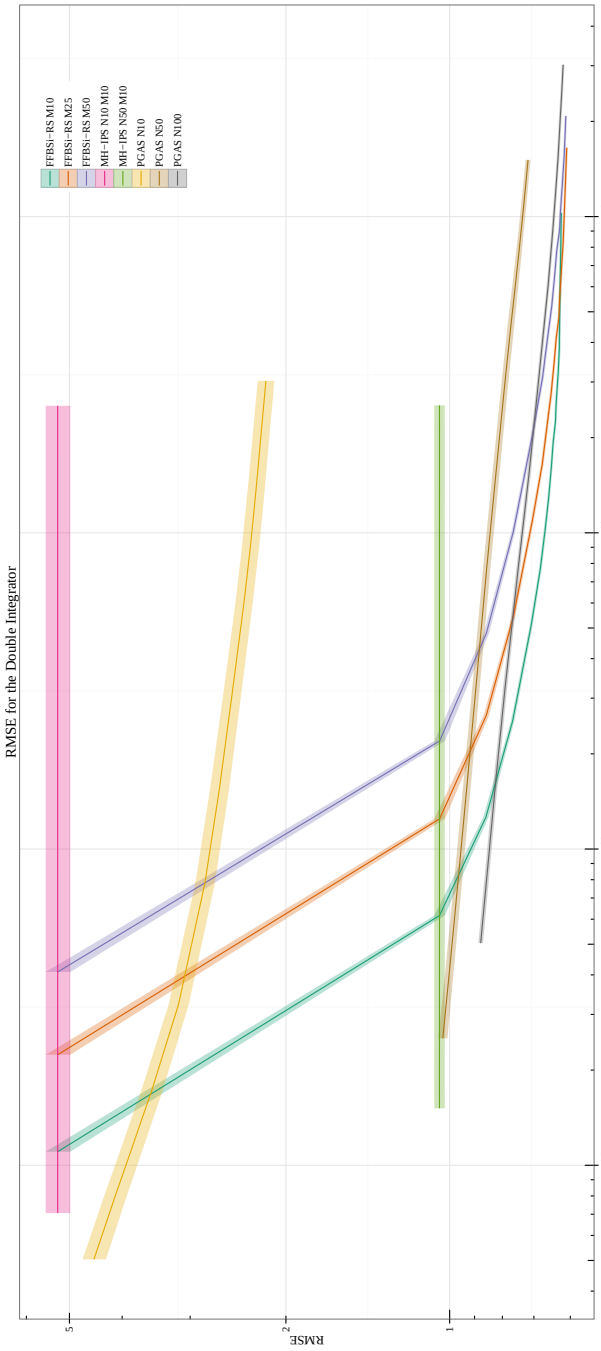

Figure 3. RMSE as a function of the computation effort for all methods applied to the degenerate double integrator model (9). The notation is the same as in Figure 1. Both PGAS and MH-IPS perform poorly for this model, which is not unexpected since they both rely on the backward kernel $p(x_t|x_{t+1})$, which due to the degeneracy of the model will only be non-zero when following the original ancestral path.

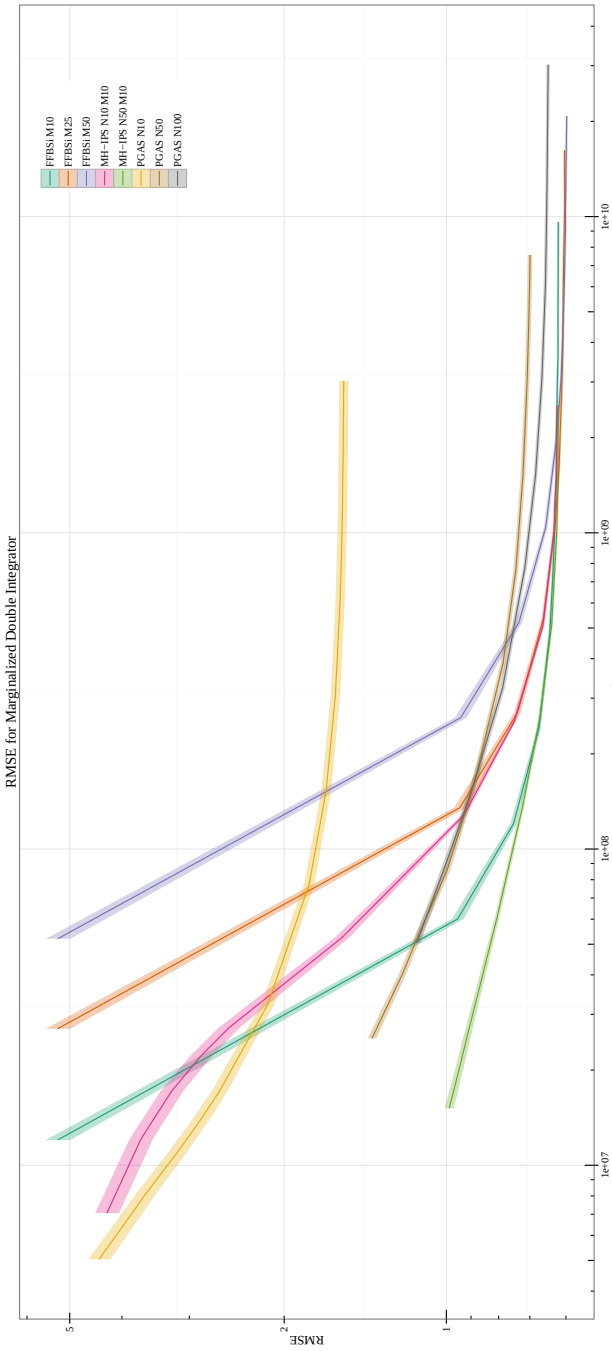

Figure 4. RMSE as a function of the computation effort for all methods applied to the marginalized double integrator model (10). Note that for this model the FFBSi smoother is used without rejection sampling. Other than that, the notation is the same as in Figure 1. For this model the MH-IPS smoother is very competitive, delivering much lower RMSE in the region of low computation effort.

7. Discussion

By studying the figures in Section 6 we see that there is no method that consistently outperforms the others, however some general trends can be observed. For the FFBSi smoothers, using a larger number of backward trajectories only gives an improvement when also increasing the number of forward particles. Thus, unless a large amount of computation effort can be spent, it is typically better to use a large number of forward particles and fewer backward trajectories.

The MH-IPS algorithm shows the most varying results. For the degenerate case it shows no improvement in the RMSE when increasing the number of iterations performed. This is expected since due to the degeneracy it will not be possible to find any new particles that give nonzero probability for $p(x_{t+1}|x_t)$ and $p(x_t|x_{t-1})$. For both the tracking example and the standard nonlinear model we see that the average RMSE only slowly decreases with the number of iterations performed, making MH-IPS a less attractive method for these models. However, for the marginalized double integrator we see that for low effort of computation MH-IPS clearly outperforms the other methods, and it also remains competitive for larger computation efforts. For the results presented in this paper the proposal was chosen as $q(x_t|x_{t-1}, y_t, x_{t+1}) = p(x_t|x_{t-1})$. This a common choice since it is typically already used in the particle filter, however using a proposal taking more information in account might improve the performance of MH-IPS. Finding such a proposal density is a model specific problem and in many cases it might not be easily obtainable.

We can see that PGAS in general performs very well, but for the degenerate model it is outperformed by the FFBSi smoother for all but the lowest computation efforts. Worth noting is that in this case the ancestral sampling part of the PGAS algorithm will not be effective and the extra computation effort required for the AS step is wasted. The end result will therefor be the same as for a regular PG smoother, which is thus preferable in this case.

8. Conclusion

In this paper we have clearly demonstrated that there is no universal best choice of method for performing particle smoothing. Assuming that the goal is to have the lowest possible RMSE for a given number of computations, all the available methods have to be compared for the particular problem. To facilitate easy comparison and experimentation it is useful to use a software that separates the model specific implementation parts from the generic parts of algorithms. One such framework is pyParticleEst[Nordh, 2013] which is what we used for the work presented in this paper.

Acknowledgements

We would like to thank Prof. Thomas Schön, Upsala University, for the suggestions and ideas that led to this work. The authors are members of the LCCC Linnaeus Center and the eLLIIT Excellence Center at Lund University.

References

Andrieu, C., A. Doucet, and R. Holenstein (2010). "Particle Markov chain Monte Carlo methods". *Journal of the Royal Statistical Society, Series B* **72**:2, pp. 1–33.

Arulampalam, M., S. Maskell, N. Gordon, and T. Clapp (2002). "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking". *IEEE Trans. Signal Process.* **50**:2, pp. 174–188. ISSN: 1053-587X.

Bunch, P. and S. Godsill (2013). "Improved particle approximations to the joint smoothing distribution using Markov Chain Monte Carlo". *Signal Processing, IEEE Transactions on* **61**:4, pp. 956–963.

Douc, R., A. Garivier, E. Moulines, J. Olsson, et al. (2011). "Sequential Monte Carlo smoothing for general state space hidden Markov models". *The Annals of Applied Probability* **21**:6, pp. 2109–2145.

Doucet, A. and A. M. Johansen (2009). "A tutorial on particle filtering and smoothing: fifteen years later". *Handbook of Nonlinear Filtering* **12**, pp. 656–704.

Dubarry, C. and R. Douc (2011). "Particle approximation improvement of the joint smoothing distribution with on-the-fly variance estimation". *arXiv preprint arXiv:1107.5524*.

Godsill, S. J., A. Doucet, and M. West (2004). "Monte Carlo smoothing for nonlinear time series". *Journal of the american statistical association* **99**:465.

Gustafsson, F. (2010). *Statistical sensor fusion*. Studentlitteratur.

Lindsten, F., M. I. Jordan, and T. B. Schön (2014). "Particle Gibbs with ancestor sampling". *The Journal of Machine Learning Research* **15**:1, pp. 2145–2184.

Lindsten, F. and T. Schön (2012). "On the use of backward simulation in the particle Gibbs sampler". In: *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*. IEEE, pp. 3845–3848.

Lindsten, F. and T. Schön (2013). "Backward simulation methods for Monte Carlo statistical inference". *Foundations and Trends in Machine Learning* **6**:1, pp. 1–143. ISSN: 1935-8237. DOI: 10.1561/2200000045. URL: http://dx.doi.org/10.1561/2200000045.

Lindsten, F., T. B. Schön, and M. I. Jordan (2013). "Bayesian semiparametric Wiener system identification". *Automatica* **49**:7, pp. 2053–2063.

Nordh, J. (2013). *PyParticleEst - Python library for particle based estimation methods*. URL: http://www.control.lth.se/Staff/JerkerNordh/pyparticleest.html.

Nordh, J. (2015, Submitted). "Metropolis-Hastings improved particle smoother and marginalized models". In: *European Conference on Signal Processing 2015*. Nice, France. Submitted.

Nordh, J. and J. Antonsson (2013). *Raw results and source code for all the experiments in this paper*. URL: http://www.control.lth.se/Staff/JerkerNordh/pscomp.html.

Olsson, J. and T. Ryden (2011). "Rao-Blackwellization of particle Markov chain Monte Carlo methods using forward filtering backward sampling". *Signal Processing, IEEE Transactions on* **59**:10, pp. 4606–4619.

Robert, C. and G. Casella (2013). *Monte Carlo statistical methods*. Springer Science & Business Media.

Särkkä, S. (2013). *Bayesian Filtering and Smoothing*. Cambridge University Press, New York, NY, USA. ISBN: 1107619289, 9781107619289.

Taghavi, E., F. Lindsten, L. Svensson, and T. Schön (2013). "Adaptive stopping for fast particle smoothing". In: *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 6293–6297. DOI: 10.1109/ICASSP.2013.6638876.

Tichavsky, P., C. Muravchik, and A. Nehorai (1998). "Posterior Cramer-Rao bounds for discrete-time nonlinear filtering". *Signal Processing, IEEE Transactions on* **46**:5, pp. 1386–1396. ISSN: 1053-587X. DOI: 10.1109/78.668800.

Wills, A., T. B. Schön, L. Ljung, and B. Ninness (2013). "Identification of Hammerstein–Wiener models". *Automatica* **49**:1, pp. 70–81.

Paper III

Nonlinear MPC and Grey-Box Identification using PSAEM of a Quadruple-Tank Laboratory Process

Erik Ackzell Nuno Duarte Jerker Nordh

Abstract

This paper describes how to apply model predictive control to a quadruple-tank laboratory process, which is a nonlinear and non-minimum phase system. It shows how to use grey-box identification to identify a nonlinear process model using the recently proposed particle stochastic approximation expectation maximization algorithm. Non-minimum phase systems pose several challenges in the MPC framework which are discussed and a method that gives good results for the system in question is presented. The nonlinearity of the process is handled by linearizing around the predicted trajectory of the previous solution of the optimization problem and the benefits of this approach is demonstrated by comparing it with using a fixed linearization point. The paper is concluded by demonstrating the validity of the approach presented by applying it to the real process.

Submitted to *Control Engineering Practice*.

1. Introduction

The objective of this paper is to describe a method to control a nonlinear process with non-minimum phase behavior. This is done by first determining the physical parameters of the process using grey-box identification and then make use of the identified model to control the process using nonlinear model predictive control, where the CVXGEN software [Mattingley and Boyd, 2012] is used to create a fast solver for a convex optimization problem.

The particle filter was introduced more than two decades ago as a solution to the nonlinear state estimation problem, see e.g. [Doucet and Johansen, 2009] for an introduction. Recently, it has also proven useful when solving the nonlinear system identification problem, see [Kantas et al., 2014] for a recent survey. The work presented in this paper uses the so-called PSAEM algorithm introduced by [Lindsten, 2013] to identify a nonlinear model for the process. It provides a solution to the nonlinear maximum likelihood problem by combining the stochastic approximation expectation maximization algorithm of [Delyon et al., 1999] with the PGAS kernel of [Lindsten et al., 2014].

Model predictive control (MPC) algorithms were first introduced almost 40 years ago in the control of processes for which a linear model could be obtained. Early work include [Richalet et al., 1978] and [Clarke, 1987]. Since the beginning of the 1990s, different approaches to control nonlinear processes using MPC have been studied, see e.g. [Kouvaritakis et al., 1999] or [Cannon et al., 2011]. In this paper, the system is linearized around the predicted trajectory of the previous solution at every time step, similar to the approach described in [Oliveira and Biegler, 1995].

A description of the process considered is given in Section 2 and the system identification method is described in Section 3. In Section 4 the MPC formulation is introduced and extensions to handle the nonlinear case are described. A method to handle the non-minimum phase behavior of the process is discussed in Section 5. Lastly, the method presented in the prior sections is used to control the real process and the results are presented in Section 6.

2. Process

The process considered in this paper, illustrated in Figure 1, consists of four water tanks, all of which have the same cross-section area A and an outlet cross-section area a_i, $i = 1, 2, 3, 4$. The objective is to control the water levels (y_1, y_2) in the two lower tanks, by adjusting the flow (q_1, q_2) from the two pumps. The flow from pump 1 is divided between tanks 1 and 4 and the flow from pump 2 is divided between tanks 2 and 3. Denote these distributive relations by γ_1 and γ_2. For $\gamma_i < 0.5$, $i = 1, 2$, the system is non-minimum

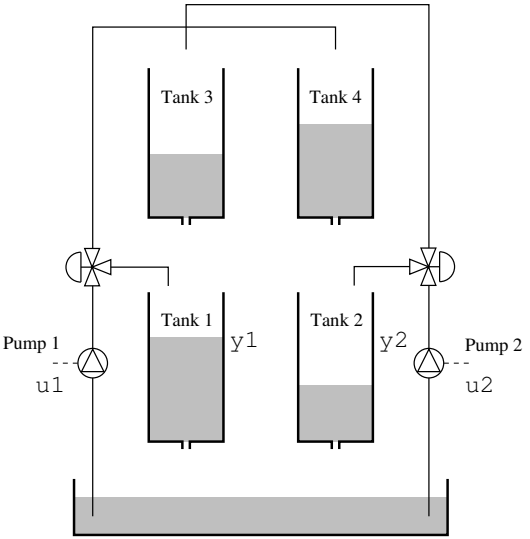

Figure 1. The quadruple-tank process.

phase [Johansson, 2000], which is the case studied in this paper. Denote by x_i the water level in tank i. By Bernoulli's law and a mass balance, the dynamics of the system is given by

$$\frac{dx_1}{dt} = -\frac{a_1}{A}\sqrt{2gx_1} + \frac{a_3}{A}\sqrt{2gx_3} + \gamma_1 q_1 \tag{1a}$$
$$\frac{dx_2}{dt} = -\frac{a_2}{A}\sqrt{2gx_2} + \frac{a_4}{A}\sqrt{2gx_4} + \gamma_2 q_2 \tag{1b}$$
$$\frac{dx_3}{dt} = -\frac{a_3}{A}\sqrt{2gx_3} + (1-\gamma_2) q_2 \tag{1c}$$
$$\frac{dx_4}{dt} = -\frac{a_4}{A}\sqrt{2gx_4} + (1-\gamma_1) q_1, \tag{1d}$$

where g is the gravitational constant. Equation (1) can then be discretized using forward Euler, yielding

$$x_{1,k+1} = x_{1,k} + h(-\alpha_1\sqrt{x_{1,k}} + \alpha_3\sqrt{x_{3,k}} + \gamma_1 q_{1,k}) \tag{2a}$$
$$x_{2,k+1} = x_{2,k} + h(-\alpha_2\sqrt{x_{2,k}} + \alpha_4\sqrt{x_{4,k}} + \gamma_2 q_{2,k}) \tag{2b}$$
$$x_{3,k+1} = x_{3,k} + h(-\alpha_3\sqrt{x_{3,k}} + (1-\gamma_2) q_{2,k}) \tag{2c}$$
$$x_{4,k+1} = x_{4,k} + h(-\alpha_4\sqrt{x_{4,k}} + (1-\gamma_1) q_{1,k}), \tag{2d}$$

where h is the step size in time and $\alpha_i = \frac{a_i}{A}\sqrt{2g}$.

3. Grey-box system identification

The identification problem is a so called grey-box identification problem, where the goal is to estimate a set of parameters for a predefined model structure, here provided by the physical modeling of the process. Knowing the inflows, q_i, the problem is to identify all the process-specific parameters, this corresponds to finding the maximum likelihood estimate (ML) of (3) given the model structure in (2) and measurements over a given time interval $(\mathbf{y}_{1:T})$.

$$\widehat{\theta}_{\mathrm{ML}} = \underset{\theta}{\operatorname{argmax}} \log p_\theta(\mathbf{y}_{1:T}) \tag{3a}$$

$$\theta = (\alpha_1 \quad \alpha_2 \quad \alpha_3 \quad \alpha_4 \quad \gamma_1 \quad \gamma_2)^{\mathrm{T}} \tag{3b}$$

Section 3.1 provides a quick introduction to ML-estimation using PSAEM and Section 3.2 gives the details for how to apply PSAEM to system identification and the specific design choices made in this paper.

3.1 Introduction to ML-estimation using PSAEM

The Expectation Maximization (EM) algorithm [Dempster et al., 1977] is an iterative algorithm to compute ML estimates of unknown parameters θ in probabilistic models involving latent variables. In this case the unknown latent variables are the true water levels in the tanks, ie. the trajectory $\mathbf{x}_{1:T}$. The EM algorithm computes the ML estimate (3) by iteratively computing the so-called *intermediate quantity*

$$\mathcal{Q}(\theta, \theta) = \int \log p_\theta(\mathbf{x}_{1:T}, \mathbf{y}_{1:T}) p_{\theta_k}(\mathbf{x}_{1:T} \,|\, \mathbf{y}_{1:T}) \mathrm{d}\mathbf{x}_{1:T} \tag{4}$$

and then maximizing $\mathcal{Q}(\theta, \theta_k)$ w.r.t. θ. For linear Gaussian state space models there exist closed form solutions for all the required expressions [Shumway and Stoffer, 1982; Gibson and Ninness, 2005]. However for nonlinear models, such as the one considered here, approximate methods have to be used, see e.g. [Lindsten, 2013; Schön et al., 2011; Cappé et al., 2005].

The sequential Monte Carlo (SMC) methods [Doucet and Johansen, 2009] or the particle Markov chain Monte Carlo (PMCMC) methods introduced by [Andrieu et al., 2010] can be exploited to approximate the joint smoothing density (JSD) with a weighted point mass distribution arbitrarily well according to

$$\widehat{p}(\mathbf{x}_{1:T} \,|\, \mathbf{y}_{1:T}) = \sum_{i=1}^{N} w_T^i \delta_{\mathbf{x}_{1:T}^i}(\mathbf{x}_{1:T}). \tag{5}$$

The quality of such an approximation increases with the number of particles (N) used. Here, $\mathbf{x}_{1:T}^i$ denotes the samples, w_T^i denotes the corresponding weights and $\delta_{\mathbf{x}}$ denotes a point-mass distribution at $\mathbf{x}$. [Schön et al., 2011] uses SMC to approximate the intermediate quantity (4). When using this approximation it is possible to improve the performance of the resulting algorithm by instead of using regular EM to make use of *stochastic approximation* EM (SAEM) [Delyon et al., 1999]. In the SAEM algorithm, the intermediate quantity (4) is replaced by the following stochastic approximation update

$$\widehat{\mathcal{Q}}_k(\theta) = (1 - \gamma_k)\widehat{\mathcal{Q}}_{k-1}(\theta) + \gamma_k \log p_\theta(\mathbf{x}_{1:T}[k], \mathbf{y}_{1:T}), \tag{6}$$

where γ_k denotes the step size, which is a design parameter that should fulfill $\sum_{k=1}^{\infty} \gamma_k = \infty$ and $\sum_{k=1}^{\infty} \gamma_k^2 < \infty$ to achieve asymptotic convergence of the approximation [Delyon et al., 1999]. Furthermore, $\mathbf{x}_{1:T}[k]$ denotes a sample from the JSD $p_{\theta_k}(\mathbf{x}_{1:T} \mid \mathbf{y}_{1:T})$.

For the problem under consideration the recently developed PMCMC methods [Andrieu et al., 2010; Lindsten et al., 2014] are useful to approximately generate samples from the JSD. In [Lindsten, 2013] SAEM is combined with the Particle Gibbs Ancestral Sampler (PGAS), a recently developed type of PMCMC method [Andrieu et al., 2010; Lindsten et al., 2014], resulting in the so-called particle SAEM (PSAEM) algorithm which is the method used in this paper.

The PGAS kernel was introduced by [Lindsten et al., 2014]. From an implementation perspective it is very similar to the standard particle filter, except for the fact that the particles are conditioned on a so-called *reference trajectory* $\mathbf{x}'_{1:T}$, which is a single trajectory estimate sampled from the particle approximation from the previous iteration. Hence, $\mathbf{x}'_{1:T}$ have to be retained throughout the sampling procedure, for details see [Lindsten et al., 2014]. Intuitively this allows for the gradual improvement of the particle approximation over the iterations for any number of particles $N > 1$. This avoids the need to increase the number of particles for each iteration of the EM-algorithm, which is a drawback when using a regular particle filter combined with (SA)EM. PGAS is summarized in Algorithm 5 where the notation is as follows: $\mathbf{x}_t = (\mathbf{x}_t^1, \ldots, \mathbf{x}_t^N)$ denotes all the particles at time t and $\mathbf{x}_{1:T} = (\mathbf{x}_1, \ldots, \mathbf{x}_T)$ the entire trajectories. $\mathbf{x}_{1:T}^i$ denotes the trajectory obtained by following the ancestral path of particle i at time T. The particles are propagated according to a proposal distribution $r_t(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{y}_t)$. The resampling step and the propagation step of the standard particle filter has been collapsed into jointly sampling the particles $\{\mathbf{x}_t^i\}_{i=1}^N$ and the ancestor indices $\{a_t^i\}_{i=1}^N$ independently from

$$M_t(a_t, \mathbf{x}_t) = \frac{w_t^{a_t}}{\sum_{l=1}^N w_t^l} r_t(\mathbf{x}_t \mid \mathbf{x}_{1:t-1}^{a_t}, \mathbf{y}_{1:t}). \tag{7}$$

Finally, W_t denotes the weight function,

$$W_t(\mathbf{x}_{1:t}, \mathbf{y}_{1:t}) = \frac{p(\mathbf{y}_t \,|\, \mathbf{x}_{1:t}) p(\mathbf{x}_t \,|\, \mathbf{x}_{1:t-1})}{r(\mathbf{x}_t \,|\, \mathbf{x}_{1:t-1}, \mathbf{y}_{1:t})}. \tag{8}$$

With minor alterations to M and W for the first time-step, where the prior distribution is used instead.

Algorithm 5 PGAS kernel

1: **Initialization** ($t = 1$)**:** Draw $x_1^i \sim r_1(\mathbf{x}_1|\mathbf{y}_1)$ for $i = 1, \ldots, N-1$ and set $\mathbf{x}_1^N = \mathbf{x}'_1$. Compute $w_1^i = W_1(\mathbf{x}_1^i)$ for $i = 1, \ldots, N$.
2: **for** $t = 2$ to T **do**
3: Draw $\{a_t^i, \mathbf{x}_t^i\} \sim M_t(a_t, \mathbf{x}_t)$ for $i = 1, \ldots, N-1$.
4: Set $\mathbf{x}_t^N = \mathbf{x}'_t$, where $\mathbf{x}'_{1:T}$ is the reference trajectory
5: Draw a_t^N with $\mathbb{P}\left(a_t^N = i\right) \propto \frac{w_{t-1}^i p(\mathbf{x}'_t \,|\, \mathbf{x}_{1:t-1}^i)}{\sum_{l=1}^N w_{t-1}^l p(\mathbf{x}'_t \,|\, \mathbf{x}_{1:t-1}^l)}$
6: Set $\mathbf{x}_{1:t}^i = \{\mathbf{x}_{1:t-1}^{a_t^i}, \mathbf{x}_t^i\}$ for $i = 1, \ldots, N$.
7: Compute $w_t^i = W_t(\mathbf{x}_{1:t}^i, \mathbf{y}_{1:t})$ for $i = 1, \ldots, N$.
8: **end for**
9: **Return** $\mathbf{x}_{1:T}, \mathbf{w}_T$.

The PSAEM algorithm for ML identification now simply amounts to making use of the PGAS kernel in Algorithm 5 to generate a particle system $\mathbf{x}_{1:T}, \mathbf{w}_{1:T}$ that is then used to approximate the intermediate quantity according to

$$\widehat{\mathcal{Q}}_k(\theta) = (1 - \gamma_k)\widehat{\mathcal{Q}}_{k-1}(\theta) + \gamma_k \log p_\theta(\mathbf{x}_{1:T}[k], \mathbf{y}_{1:T}), \tag{9}$$

where $\mathbf{x}_{1:T}[k]$ is a sampled trajectory from $\mathbf{x}_{1:T}, \mathbf{w}_{1:T}$ generated at iteration k. This trajectory is, in addition to its use in the intermediate quantity, then used as the reference trajectory ($\mathbf{x}'_{1:T}$) for the next iteration. The result is provided in Algorithm 6. Note that the initial reference trajectory $\mathbf{x}_{1:T}[0]$ is obtained by running a regular forward filter backward simulator (FFBSi) particle smoother, see [Lindsten and Schön, 2013] for details.

3.2 Nonlinear Grey-box identification using PSAEM

A stochastic model is needed to apply the methods from the previous section. Therefore model (2) is extended with additive Gaussian noise,

$$\mathbf{x}_{k+1} = F(\mathbf{x}_k, \mathbf{q}_k) + \mathbf{v}_k \tag{10a}$$
$$\mathbf{y}_k = \mathbf{C}\mathbf{x}_k + \mathbf{e}_k, \tag{10b}$$
$$\mathbf{v}_k \sim N(0, \mathbf{V}),\ \mathbf{e}_k \sim N(0, \mathbf{R}). \tag{10c}$$

Algorithm 6 PSAEM for grey-box identification

1: **Initialization:** Set $\theta[0] = \theta_0$ and compute $\mathbf{x}_{1:T}[0]$ using an FFBSi particle smoother. Set $\widehat{\mathcal{Q}}_0 = 0$ and set $\mathbf{w}[0]$ to an empty vector.
2: Draw $\mathbf{x}'_{1:T}$ using FFBSi.
3: **for** $k \geq 1$ **do**
4: Draw $\mathbf{x}_{1:T}, \mathbf{w}_T$ by running Algorithm 5 using $\mathbf{x}'_{1:T}$ as reference and with the parameters as $\theta[k]$.
5: Draw j with $\mathbb{P}(j = i) = w_T^i$.
6: Set $\mathbf{x}'_{1:T} = \mathbf{x}^j_{1:T}[k]$
7: Compute $\widehat{\mathcal{Q}}_k(\theta)$ according to (9).
8: Compute $\theta[k] = \operatorname{argmax} \widehat{\mathcal{Q}}_k(\theta)$.
9: **if** termination criterion is met **then**
10: **return** $\{\theta[k]\}$
11: **end if**
12: **end for**

Here $F(\mathbf{x}_k, \mathbf{q}_k)$ are the nonlinear dynamics from (2).

There are a few important details to make the PSAEM algorithm from the previous section work well for system identification. One important choice is the proposal density $r(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_t)$. A common choice in particle filters is the bootstrap proposal $r = p(\mathbf{x}_t|\mathbf{x}_{t-1})$. However, since the model is unknown, this would require modeling the process noise, $\mathbf{v}$, as having a very large covariance to ensure that particles end up in the regions corresponding to the measurements. If all the states can be directly measured, that is $\mathbf{C} = \mathbf{I}_{4\text{x}4}$, it is also possible to use the proposal $p(\mathbf{x}_{t+1}|\mathbf{y}_{t+1})$. For the model in question in this paper, an analytic expression for the optimal proposal can also be found, $r = p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_t)$, having the form shown in (11).

$$p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{y}_{t+1}) = N(\bar{\mathbf{x}}_{t+1}, \mathbf{P}) \tag{11a}$$
$$\bar{\mathbf{x}}_{t+1} = F(\mathbf{x}_t, \mathbf{q}_t) + \mathbf{V}\mathbf{C}^{\mathrm{T}}\mathbf{S}^{-1}(\mathbf{y}_{t+1} - \mathbf{C}\mathbf{F}(\mathbf{x}_t, \mathbf{q}_t)) \tag{11b}$$
$$\mathbf{P} = \mathbf{V} - \mathbf{V}\mathbf{C}^{\mathrm{T}}\mathbf{S}^{-1}\mathbf{C}\mathbf{V} \tag{11c}$$
$$\mathbf{S} = \mathbf{C}\mathbf{V}\mathbf{C}^{\mathrm{T}} + \mathbf{R} \tag{11d}$$

This still has the problem mentioned above that initially the model will be very inaccurate, thus requiring a large process noise uncertainty, $\mathbf{V}$. By including $\mathbf{V}$ as an unknown parameter it is possible to have a large value as an initial guess. Then, as the prediction accuracy of the model increases, the estimated process noise covariance will decrease, helping to focus the particles to regions of high probability.

A more problem specific improvement is to use the knowledge that the water tanks can not have negative water levels, and neither can they be filled more than to the height of the tank. The proposal density is therefore

further improved by sampling from a truncated Gaussian instead, which helps avoid spending time computing particles that can not represent the true state. Using the symbol $N_{trunc}(\mathbf{m}, \mathbf{P}, \mathbf{a}, \mathbf{b})$ to indicate the truncated normal distribution with mean $\mathbf{m}$, variance $\mathbf{P}$, lower limit $\mathbf{a}$ and upper limit $\mathbf{b}$ it is thus wished to sample from

$$N_{trunc}(\bar{\mathbf{x}}_{t+1}, \mathbf{P}, \mathbf{0}, \mathbf{h}_{\max}). \tag{12}$$

Due to the assumption that $\mathbf{V}$ and $\mathbf{R}$ are diagonal this is easily accomplished using rejection sampling, since the problem just corresponds to sampling four independent univariate truncated Gaussian variables. The multivariate case is in general more complicated.

The maximization step now requires the computation of the ML estimate of all the parameters given the estimated trajectories. To simplify the computations the solution is approximated as the ML estimate of $\{\alpha_{1-4}, \gamma_{1-2}\}_{k+1}$ using the estimated process noise covariance, $\mathbf{V}_k$, from the previous step, and then using the new values of $\{\alpha_{1-4}, \gamma_{1-2}\}_{k+1}$ to compute $\mathbf{V}_{k+1}$. This approximation allows the values of $\{\alpha_{1-4}, \gamma_{1-2}\}_{k+1}$ to be found as the solution to a weighted linear least squares (WLS) problem, and the estimation of $\mathbf{V}$ reduces to calculating the weighted mean value of the squared residuals. Notice that the problem is essentially a double-weighted least squares problem, for a given trajectory the states are weighted according to the $\mathbf{V}$-matrix, since this choice of weights implies that the solution of the WLS problem coincides with the maximum-likelihood solution. In addition to weighting of the individual states within a single trajectory according to the process noise covaraince, the trajectories generated from all the iterations up to the current one should also be weighted relative each other according to the sequence $\gamma_{1:k}$, which is part of the SAEM approximation (6).

Even though the process earlier was presented as providing measurements of all the four tank levels, this is not necessary using the algorithm presented above. Using the measured tank levels from only the lower tanks simply corresponds to

$$\mathbf{C} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}. \tag{13}$$

The design choices required are the choice of initial parameter guesses, θ_0, the measurement noise, $\mathbf{R}$, the sequence $\{\gamma_k\}_1^\infty$ and the termination criterion.

3.3 Grey-box identification validation data results

The performance of the identification algorithm is evaluated on 100 simulated realizations of the nonlinear model (with $\mathbf{C} = \mathbf{I}_{4x4}$) using the input signal shown in Figure 2, which also shows a sample realization. The process noise

Table 1. Parameter estimation results for 100 simulated datasets, presented is the mean values ± one standard deviation. The least squares solution is the estimated parameter values from assuming that the measurements correspond exactly to the true states (x_1, x_2, x_3, x_4). "PSAEM, all tanks" is the maximum-likelihood solution using PSAEM while measuring the water levels in all the tanks. "PSAEM, lower tanks" it the maximum-likelihood solution when only measuring the water levels in the lower tanks. As can be seen the LS solution appears to consistently over estimate the outlet areas, α_i, whereas PSAEM is much closer to the true value. There is a slight improvement in accuracy when measuring all the tank levels, which is expected since the algorithm then has access to more data.

	True	Least Squares	PSAEM, all tanks	PSAEM, lower tanks
α_1	0.27	0.2891 ± 0.0031	0.2713 ± 0.0017	0.2723 ± 0.0043
α_2	0.29	0.3102 ± 0.0031	0.2916 ± 0.0016	0.2921 ± 0.0023
α_3	0.31	0.3311 ± 0.0040	0.3074 ± 0.0022	0.3072 ± 0.0039
α_4	0.33	0.3538 ± 0.0034	0.3274 ± 0.0018	0.3270 ± 0.0036
γ_1	0.28	0.2836 ± 0.0067	0.2872 ± 0.0029	0.2782 ± 0.0050
γ_2	0.32	0.3216 ± 0.0073	0.3269 ± 0.0029	0.3178 ± 0.0049

covariance was set to $\mathbf{x}Q = 0.1^2\mathbf{I}$ and the simulated measurement noise had covariance $\mathbf{R} = 0.25^2\mathbf{I}$.

To evaluate the performance of the algorithm it is compared against the least-squares solution that is obtained by directly using all the four measurements. Included are also the results when using PSAEM and only having access to the lower tank levels ($\mathbf{C} = \mathbf{I}_{2x4}$). The initial guess for each α and γ was drawn uniformly in the range $0.15 - 0.45$. The termination criterion used was a fixed number of iterations. For the case when all tank levels were measured 500 iterations were used, and for the case when only measuring the lower tank levels 50000 iterations were used. In both cases $\gamma_k = k^{-0.51}$ was used. Figure 3 shows the convergence of the parameter estimates from the PSAEM algorithm when only measuring the lower tank levels. The results for both cases can be seen in Table 3.3. It is obvious that the PSAEM algorithm outperforms the least-squares approach. Interestingly there is only a minor degradation in performance when only using the water levels from the lower tanks.

For the case when 50000 iterations were used, only the last 4000 sampled trajectories were used in the maximization step, i.e. at iteration k the maximization problem was solved using only $\{\mathbf{x}_{1:T}[i]\}_{i=k-4000}^{k}$. This was done to reduce the computational complexity. The estimation problem was solved using the *pyParticleEst* framework [Nordh, 2013].

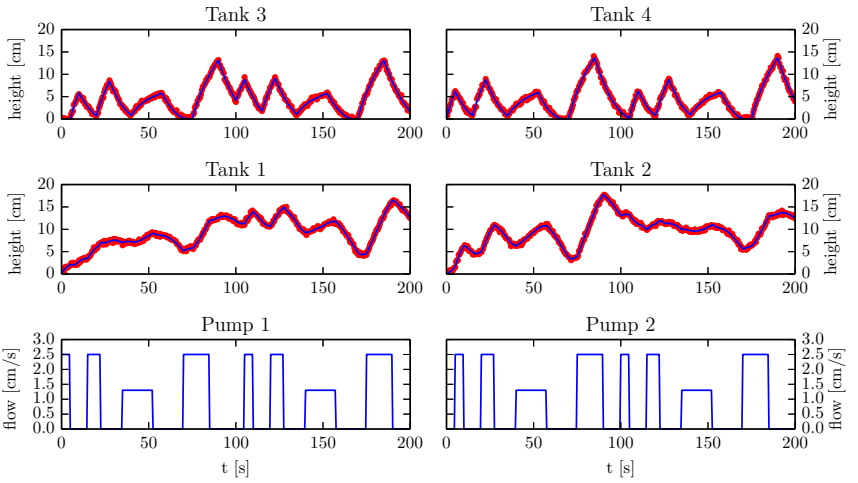

Figure 2. Example realization from those used for verifying the system identification. Notice that the tank levels for both the upper and lower tanks move over almost their full range $(0 - 20)$. This was a design goal when selecting the input signal used for the identification, as the authors believe that will provide a good scenario for distinguishing between the outlet areas, α_i, and the pump flow ratios γ_i. The red dots are the measurements and the continuous lines are the true trajectories.

4. MPC

4.1 Introduction

Let a discrete mathematical model of some physical process be given and let it be linearized around some point, yielding the state space model

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{f}_k \\ \mathbf{y}_k &= \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k, \end{aligned} \tag{14}$$

where $\mathbf{x}_k \in \mathbb{R}^n$ is the state vector and $\mathbf{u}_k \in \mathbb{R}^m$ is the input signal at time step k. Furthermore, let some state reference $\{\mathbf{x}_k^{ref}\}_1^\infty$ be given.

Let $\mathbf{W}$ be a positive definite weight matrix and let P be a positive integer. By minimizing the cost function

$$J = \sum_{t=k}^{k+P} \|\mathbf{x}_t - \mathbf{x}_t^{ref}\|_{\mathbf{W}} \tag{15}$$

at the current time step, subject to (14) and possibly to linear physical constraints, an input signal $\{\mathbf{u}_t\}_k^{k+P}$ for the following P time steps is obtained.

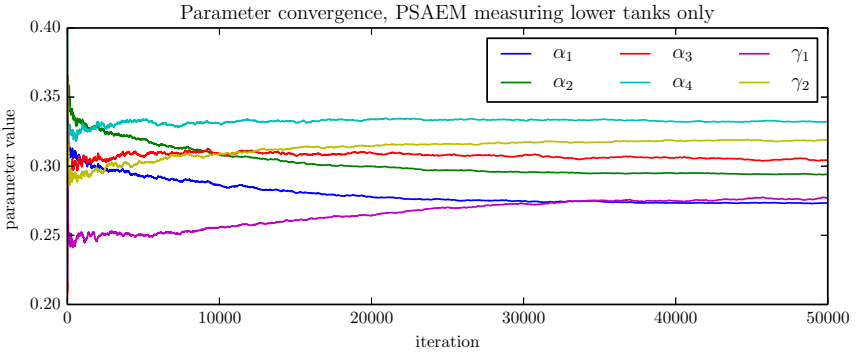

Figure 3. Parameter convergence plot for one of the example datasets using PSAEM and only measuring the heights of the lower tanks. As can be seen the parameters converge very slowly, thus requiring a large number of iterations. For this particular realization it even appears that 50000 iterations could have been slightly too few to reach stationarity.

The first input signal $\mathbf{u}_k$ is then used and (15) is then minimized again for $\{t\}_{k+1}^{k+1+P}$. Since $\mathbf{W}$ is positive definite and the constraints are all linear, minimizing (15) is a linear convex problem.

4.2 NMPC

Instead of linearizing the system around a fixed point independent of the current state of the process, the system can be linearized around the predicted trajectory, similar to the approach described in [Oliveira and Biegler, 1995]. Given that, at each time step, the predicted trajectory for the states over the following P time steps describes the states more accurately compared to if the states were assumed to remain at a fixed linearization point, this method would ensure a smaller loss in accuracy of the model. A slight difficulty is that the linearized model becomes time-varying.

In this paper, no attempt is made to use the nonlinear model of the process when solving the control problem without linearizing the model first. This is mainly due to the increased complexity of the numerical problem as well as the availability of methods and software for solving linear problems.

4.3 Controlling the quadruple-tank process

By linearizing (2) around some point $\mathbf{x}^0$, the state space model

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{A}_0\mathbf{x}_k + \mathbf{B}\mathbf{q}_k + \mathbf{f}_0 \\ \mathbf{y}_k &= \mathbf{C}\mathbf{x}_k \end{aligned} \tag{16}$$

is obtained, where

$$\mathbf{A}_0 = \frac{h}{2}\begin{bmatrix} 1-\frac{\alpha_1}{\sqrt{x_1^0}} & 0 & \frac{\alpha_3}{\sqrt{x_3^0}} & 0 \\ 0 & 1-\frac{\alpha_2}{\sqrt{x_2^0}} & 0 & \frac{\alpha_4}{\sqrt{x_4^0}} \\ 0 & 0 & 1-\frac{\alpha_3}{\sqrt{x_3^0}} & 0 \\ 0 & 0 & 0 & 1-\frac{\alpha_4}{\sqrt{x_4^0}} \end{bmatrix} \tag{17a}$$

$$\mathbf{B} = h\begin{bmatrix} \gamma_1 & 0 \\ 0 & \gamma_2 \\ 0 & 1-\gamma_2 \\ 1-\gamma_1 & 0 \end{bmatrix}, \quad \mathbf{C} = \mathbf{I}_{4\text{x}4} \tag{17b}$$

$$\mathbf{f}_0 = \frac{h}{2}\begin{bmatrix} -\alpha_1\sqrt{x_1^0} + \alpha_3\sqrt{x_3^0} \\ -\alpha_2\sqrt{x_2^0} + \alpha_4\sqrt{x_4^0} \\ -\alpha_3\sqrt{x_3^0} \\ -\alpha_4\sqrt{x_4^0} \end{bmatrix}. \tag{17c}$$

Let P denote the number of steps in the cost function. The software used to solve the minimization problem is CVXGEN [Mattingley and Boyd, 2012]. Naturally, the software has certain limitations, one of which being the maximum numbers of non-zero KKT matrix entries for the problem, see [Mattingley and Boyd, 2012] for details. Hence, the maximum number of possible steps P for this specific problem is approximately $P = 30$.

In order to increase the possible prediction horizon, limited by the number of steps in the cost function, each step in the cost function can be chosen to represent more than one time step. If the pump flow is assumed to change in each of the first P_1 time steps in the prediction horizon and thereon after only change every third time step, the prediction horizon can be increased significantly. This implies that the initial P_1 steps in the cost function represent one time step each, while the remaining $P - P_1$ represents three time steps each. If the system is stationary after the initial P_1 steps, no loss in the accuracy of the model occurs with this approach.

Let some reference for the water levels be given, $\{x_{i,k}^{\text{ref}}\}_{k=1}^{\infty}$ with $i = 1, 2$, and let $\mathbf{w}_t^{\mathbf{x}}$ be the weight on the difference between the states and their respective references at step t. In order to favor solutions where the pump flow remains constant, a penalty on the change in pump flow, denoted $\Delta\mathbf{q}_t = \mathbf{q}_t - \mathbf{q}_{t-1}$, is also included in the cost function. Let $\mathbf{w}_t^{\Delta\mathbf{q}}$ denote the weight on $\Delta\mathbf{q}_t$ at step t. Introduce the cost function

$$J = \sum_{t=k+1}^{k+1+P} (\mathbf{w}_t^{\mathbf{x}} \| (\tilde{x}_{1,t} \quad \tilde{x}_{2,t}) - (\tilde{x}_{1,t}^{\text{ref}} \quad \tilde{x}_{2,t}^{\text{ref}}) \|_2^2 + \mathbf{w}_t^{\Delta \mathbf{q}} \| \Delta \mathbf{q}_t \|_2^2) \tag{18}$$

subject to the constraints

$$\tilde{\mathbf{x}}_{t+1} = \mathbf{A}_t \tilde{\mathbf{x}}_t + \mathbf{B}\mathbf{q}_t + \mathbf{f}_t, \quad t = k, \ldots, k + P_1 - 1 \tag{19a}$$

$$\begin{aligned} \tilde{\mathbf{x}}_{k+P_1+1} &= \mathbf{A}_{k+P_1+1}(\mathbf{A}_{k+P_1} \tilde{\mathbf{x}}_{k+P_1} + \mathbf{B}\mathbf{q}_{k+P_1} + \mathbf{f}_{k+P_1}) \\ &\quad + \mathbf{B}\mathbf{q}_{k+P_1} + \mathbf{f}_{k+P_1+1} \end{aligned} \tag{19b}$$

$$\begin{aligned} \tilde{\mathbf{x}}_{t+1} &= \mathbf{A}_{k+P_1+2+3(t-P_1-k)}(\mathbf{A}_{k+P_1+1+3(t-P_1-k)}(\mathbf{A}_{k+P_1+3(t-P_1-k)} \tilde{\mathbf{x}}_t \\ &\quad + \mathbf{B}\mathbf{q}_t + \mathbf{f}_{k+P_1+3(t-P_1-k)}) + \mathbf{B}\mathbf{q}_t + \mathbf{f}_{k+P_1+1+3(t-P_1-k)}) + \\ &\quad + \mathbf{B}\mathbf{q}_t + \mathbf{f}_{k+P_1+2+3(t-P_1-k)}, \quad t = k + P_1 + 1, \ldots, k + P - 1 \end{aligned} \tag{19c}$$

$$0 \leq \tilde{x}_i \leq x_{\max} \quad i = 1, 2, 3, 4 \tag{19d}$$

$$0 \leq q_j \leq q_{\max} \quad j = 1, 2. \tag{19e}$$

Here $\mathbf{A}_t$ and $\mathbf{f}_t$ denote the matrices linearized around the predicted state $\mathbf{x}_t$. Furthermore, $\tilde{\mathbf{x}}_i = \mathbf{x}_j$, where

$$j = \begin{cases} i, & i = k+1, k+2, \ldots, k+P_1 \\ i+1, & i = k+P_1+1 \\ k+P_1+2+3(i-P_1-1-k), & i = k+P_1+2, k+P_1+3, \ldots, k+P \end{cases}.$$

With the same use of the index j, the references are chosen as

$$\tilde{\mathbf{x}}_i^{\text{ref}} = \begin{cases} \mathbf{x}_j^{\text{ref}}, & i = k+1, k+2, \ldots, k+P_1 \\ \frac{1}{3} \sum_{m=-1}^{1} \mathbf{x}_{j+m}^{\text{ref}}, & i = k+P_1+1, k+P_1+2, \ldots, k+P \end{cases}.$$

Since the last $P - P_1$ steps in the cost function represent three time steps each, the references $\tilde{\mathbf{x}}_t^{ref}$, $\quad t > k + P_1$, are chosen as the mean value of three consecutive steps in order to make changes in references smoother.

If it is of interest to put different weights on the different tanks, $\mathbf{w}_t^{\mathbf{x}} \| \cdot \|_2^2$ can be replaced by $\| \cdot \|_{\mathbf{W}}$ in the cost function for some positive definite matrix $\mathbf{W}$.

When comparing the linear MPC controller and the NMPC controller, the NMPC controller performs better, in terms of both the root mean square error (RMSE) and the integrated absolute error (IAE). Furthermore, the NMPC controller also handles the change in reference faster than the linear MPC controller. The result can be seen in Figures 4 and 5.

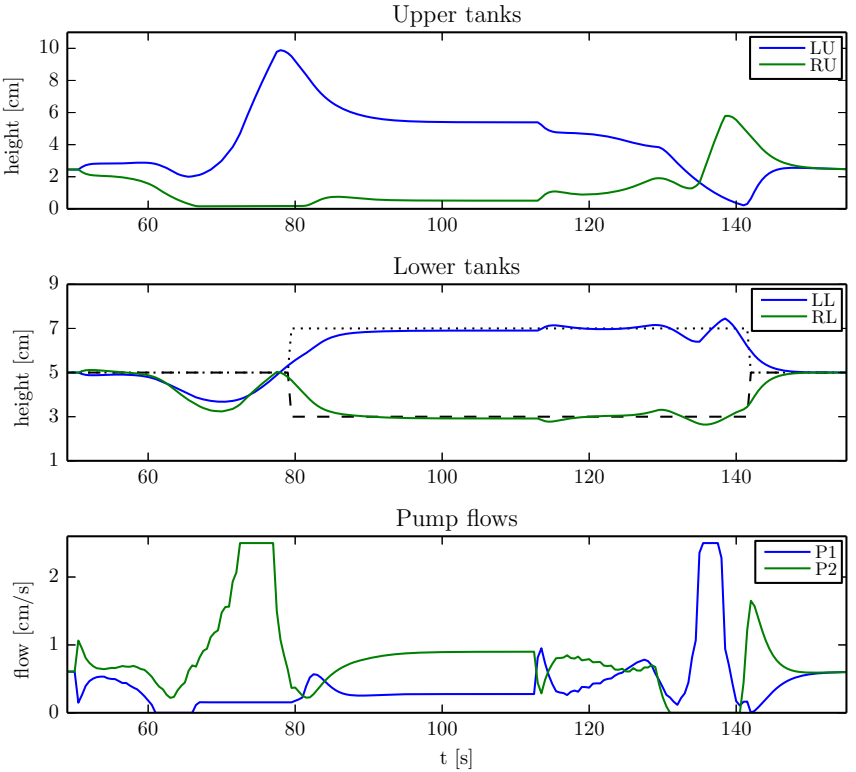

Figure 4. Control of the simulated quadruple-tank process using linear MPC. The system is linearized around a fixed state, $\mathbf{x}^0 = (5.0, 5.0, 2.45, 2.45)$. A prediction horizon of 30 seconds is used, with $P_1 = 15$ and time step $h = 0.5$ seconds. In the initial 4.5 seconds in the cost function, the states are weighted by a fixed negligible weight, the following 3 seconds they are are weighted by one unit of weight, while the remaining states are weighted by 3 units of weight. See Section 5 for a discussion about how the weights are chosen. LU and RU are the outputs from the left upper and right upper tank, LL and LR are the outputs from the left lower and right lower tank, the dotted and dashed line are the references for the lower left and lower right tank, while P1 and P2 are the flows from pump 1 and pump 2. The controller handles the change in reference slower than when using nonlinear MPC as can be seen in Figure 5 and stationary errors also occur. The RMSEs for the lower tanks are 0.59 and 0.61 for the left and right tank, respectively, and the IAEs are 50.04 and 49.81 for the left and right tank, respectively.

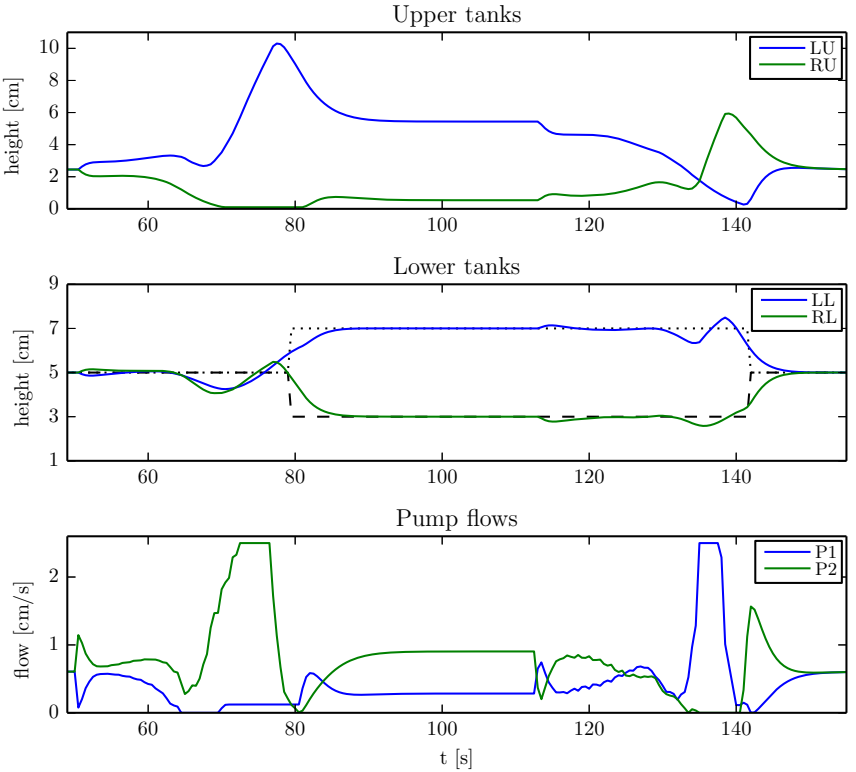

Figure 5. Control of the simulated quadruple tank process using nonlinear MPC. In each time step the system is linearized around the predicted trajectory from the previous time step. The same prediction horizon, step size, weighting and naming is used as in Figure 4. The controller handles the changes in references faster than when using linear MPC and no stationary error occurs. The RMSEs for the lower tanks are 0.52 and 0.52 for the left and right tank, respectively, while the IAEs are 35.54 and 36.86 for the left and lower tanks, respectively.

5. Improved MPC for non-minimum phase systems

5.1 Introduction

A typical behavior of a non-minimum phase system is that in order to change the value of one of the outputs in a desired direction, the value of the output initially has to change in the opposite direction. Due to the non-linearity of the process, this initial change depends on the magnitude of the desired change. When controlling a non-minimum phase system, including the initial time steps of the prediction horizon in the cost function is not the best approach [Richalet and O'Donovan, 2009]. On the other hand, it is possible to exclude too many time steps, so a good choice for the number of steps to exclude needs to be determined.

5.2 Dealing with the non-minimum phase behavior of the quadruple-tank process

Only when a change in the difference of the water levels is desired, the non-minimum phase behavior occurs. Hence, this is the case studied in this section.

In order not to exclude too many initial time steps in the cost function due to the non-minimum phase behavior, a negligible weight ought to be put on these as well. This would imply that the controller has no reason to postpone a change in the pump flow which would otherwise lead to stationary errors.

A good choice of the weights $\mathbf{w}_t^{\mathbf{x}}$ and $\mathbf{w}_t^{\Delta\mathbf{q}}$ in (18) in order to deal with the non-minimum phase behavior is desired. Three different approaches are considered in this paper. The first two use a constant weight $\mathbf{w}_t^{\Delta\mathbf{q}}$ on the difference in the pump flow and are chosen in such a way that an increasing amount of weight $\mathbf{w}_t^{\mathbf{x}}$ is put on the states. These methods increase the weight over time linearly and quadratically respectively.

The third method uses a weight $\mathbf{w}_t^{\Delta\mathbf{q}}$ on the difference in the pump flow proportional to the time $\mathbf{q}_t$ is allowed to vary. Furthermore, the weights $\mathbf{w}_t^{\mathbf{x}}$ on the states are chosen as three fixed values, a negligible weight on the initial p steps, 1 unit of weight on the $P_1 - p$ following steps and 3 units of weight on the last $P - P_1$ steps. These are selected in such a way since the first P_1 steps represent 1 time step each and the last $P - P_1$ steps represent 3 time steps each.

5.3 Handling non-minimum phase behavior

When the weights on the states are chosen to increase linearly, the controller handles the change in reference slower compared to when they are chosen to increase quadratically as can be seen in Figures 6 and 7.

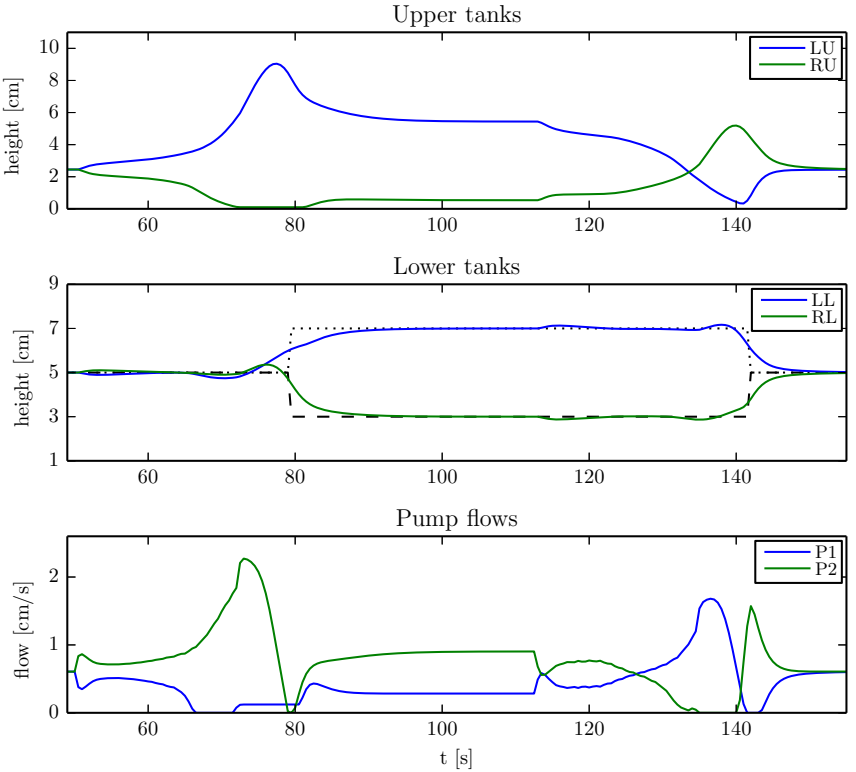

Figure 6. Control of the simulated quadruple-tank process using nonlinear MPC. The weights on the states in the cost function are chosen to increase linearly. A prediction horizon of 30 seconds is used, with $P_1 = 15$ and time step $h = 0.5$ seconds. LU and RU are the outputs from the left upper and right upper tank, LL and LR are the outputs from the left lower and right lower tank, the dotted and dashed line are the references for the lower left and lower right tank, while P1 and P2 are the flows from pump 1 and pump 2. The controller handles the change in references slower compared to when the weights are chosen to increase quadratically, as can be seen in Figure 7. The RMSEs are 0.50 and 0.49 for the left and right tank, respectively, and the IAEs are 31.88 and 29.94 for the left and right tank, respectively.

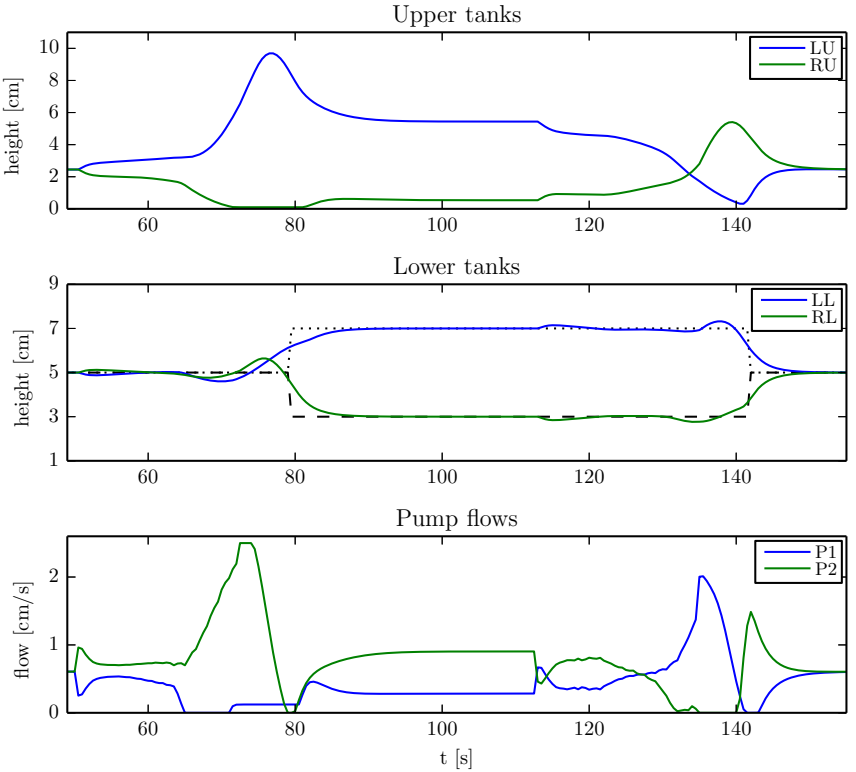

Figure 7. Control of the simulated quadruple-tank process using nonlinear MPC. The weights on the states in the cost function are chosen to increase quadratically. The same prediction horizon, step size and naming is used as in Figure 6. The controller handles the change in references faster compared to when the weights are chosen to increase linearly, but slower compared to when the weights are chosen in the way as can be seen in Figure 5. The RMSEs are 0.51 and 0.50 for the left and right tank, respectively, while the IAEs are 32.32 and 31.49 for the left and right tank, respectively.

When the third scheme of weights is used with $p = 9$, the controller handles the change in reference faster compared to both previous methods. This can be seen in Figure 5. A comparison of different values of p can be seen in Figure 8. That the water levels converge faster to the references as p is increased implies that the prediction horizon can be decreased. Since the prediction horizon is limited by the software used when solving the minimization problem, reducing the prediction horizon might be needed for more complex problems.

6. Experiment on the real process

6.1 Identification

The lab process is designed so that it is possible to block the outflows from the different tanks. Utilizing this it is possible to experimentally determine the values of α, γ. The α-values are determined measuring the time it takes for the water to decrease from one level to another in each tank isolated by itself, and the γ-values by studying the ratio of water being pumped into the different tanks when all the outlets are closed. Table 6.1 compares the values obtained in this way for a couple of lab processes by those found by the algorithm from Section 3. Worth noticing is that the real processes introduce some extra unmodeled nonlinearities. The value of γ for instance varies with the flow, when the flow is low it is essentially zero due to the way the process is constructed. Another issue is that the tubing in the process introduces time delays that are not modeled either.

Table 2. Parameter estimation results for three different labprocesses compared with the results obtained from performing manual experiments as described in Section 6.1. As can be seen the results are close, but not identical. This is expected since the real process includes additional nonlinearities and time-delays that are not present in the model.

	Process 1		Process 2		Process 3	
	Manual	PSAEM	Manual	PSAEM	Manual	PSAEM
α_1	0.32	0.33	0.46	0.44	0.40	0.38
α_2	0.40	0.40	0.43	0.44	0.45	0.45
α_3	0.45	0.40	0.45	0.42	0.42	0.42
α_4	0.44	0.42	0.44	0.42	0.42	0.41
γ_1	0.27	0.28	0.27	0.29	0.27	0.23
γ_2	0.35	0.36	0.29	0.32	0.30	0.26

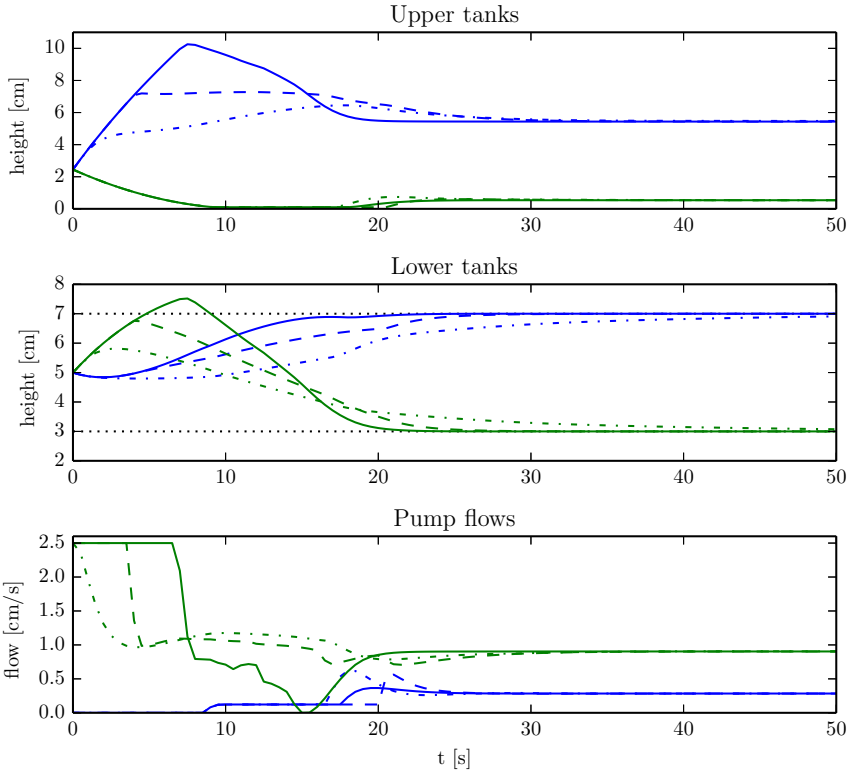

Figure 8. Control of the simulated quadruple-tank process using nonlinear MPC. The same prediction horizon and step size as in Figure 6 is used. The initial state is chosen as $\mathbf{x}_0 = (5.0 \quad 5.0 \quad 2.45 \quad 2.45)$ and the dotted line denotes the references for the lower tanks. Three different choices for how many initial states p in the cost function to only weight with a negligible weight are compared. The dash-dotted lines describe the output and pump flows when $p = 0$, the dashed lines describe the output and pump flows when $p = 2$, while the continuous line describes the output and pump flows when $p = 9$. The tank levels converge to the references significantly faster when p is increased.

6.2 Integral action

In order to deal with possible steady state errors as a result of any modeling errors and disturbances, an integral part is added to the controller using the same approach as in [Oliveira and Biegler, 1995]. The state vector $\mathbf{x}$ introduced in Section 2 is extended with the integrals i_1, i_2 of the errors of the two first states, yielding $\hat{\mathbf{x}} = (x_1 \quad x_2 \quad x_3 \quad x_4 \quad i_1 \quad i_2)^{\mathrm{T}}$. The state space model (16) can then be extended with

$$\hat{\mathbf{x}}_{k+1} = \underbrace{\begin{bmatrix} \mathbf{A} & \mathbf{0}_{2x2} \\ -\mathbf{ZA} & \mathbf{I}_{2x2} \end{bmatrix}}_{\hat{\mathbf{A}}} \hat{\mathbf{x}}_k + \underbrace{\begin{bmatrix} \mathbf{B} \\ -\mathbf{ZB} \end{bmatrix}}_{\hat{\mathbf{B}}} \mathbf{q}_k + \underbrace{\begin{bmatrix} \mathbf{f} \\ \mathbf{x}_{k+1}^{\mathrm{ref}} - \mathbf{Zf} \end{bmatrix}}_{\hat{\mathbf{f}}}, \tag{20}$$

where

$$\mathbf{Z} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

A new cost function to minimize is then introduced,

$$J = \sum_{t=k+1}^{P} (\mathbf{w}_t^{\mathbf{x}} \| (\hat{x}_{1,t} \quad \hat{x}_{2,t}) - (\hat{x}_{1,t}^{\mathrm{ref}} \quad \hat{x}_{2,t}^{\mathrm{ref}}) \|_2^2 + \mathbf{w}_t^{\Delta \mathbf{q}} \| \Delta \mathbf{q}_t \|_2^2 + \mathbf{w}_t^{i} \| (i_{1,t} \quad i_{2,t}) \|_2^2) \tag{21}$$

subject to (19) with appropriate changes.

6.3 Results

In Figure 9, the performance of the controller on the real quadruple-tank process can be seen. The physical parameters of the process was first identified using the method described in Section 3.2 and the process was then controlled using the methods described in the previous sections.

7. Conclusion

As seen in Section 3, a good approximation to the physical parameters of the simulated quadruple-tank process can be obtained using grey-box identification. Furthermore, the method yields good results when using it on the real process as well. That only measurements from two of the four tanks are needed for the algorithm to converge indicates that the method can be used on processes when not all states are measurable, which is desirable to decrease complexity and cost of the hardware.

In Section 4, it is shown that controlling the quadruple tank process using nonlinear MPC yields a much better behavior compared to when using linear

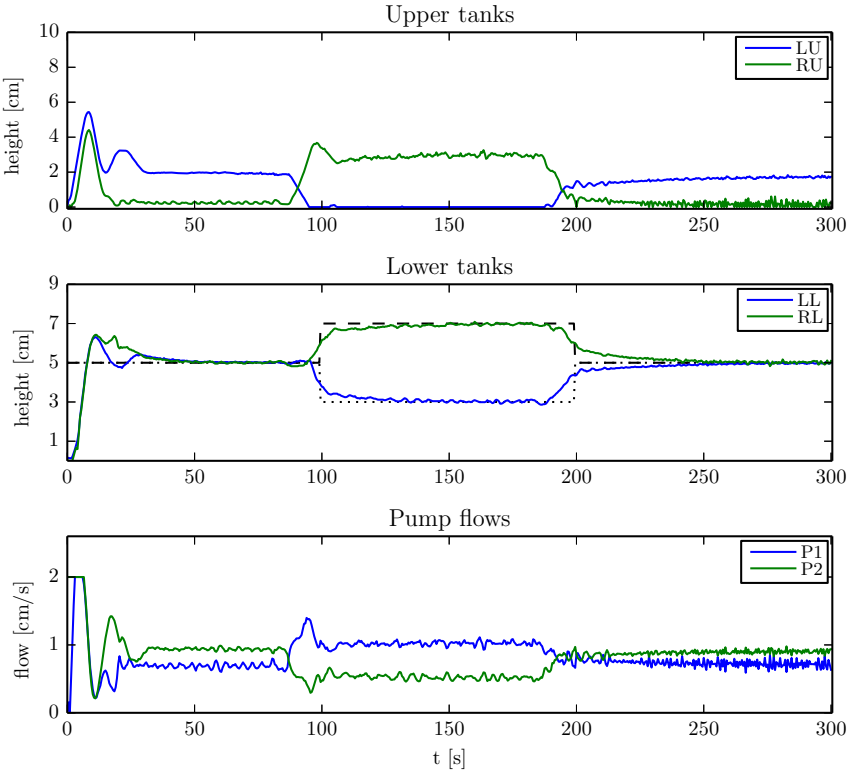

Figure 9. Control of the real quadruple-tank process using nonlinear MPC. The physical parameters of the process are first determined using the method described in Section 3.2 and the process is then controlled using the method described in Section 4.3. To deal with modeling errors and disturbances, the controller is augmented with an integral part in the way described in Section 6.2. The prediction horizon is 28.5 seconds, with $P_1 = 15$ and time step $h = 0.5$ seconds. The weights on the states in the cost function are chosen in the same way as in 4 with an extra large weight on the last state. The weight on the integral part is chosen to be fixed for the first 4.5 seconds, increased by a factor of 10 for the following 3 seconds and increased by a factor of ten for the remaining prediction horizon. LU and RU are the outputs from the left upper and right upper tank, LL and LR are the outputs from the left lower and right lower tank, the dotted and dashed line are the references for the lower left and lower right tank, while P1 and P2 are the flows from pump 1 and pump 2.

MPC. This is expected, since a better model is obtained when linearizing around the predicted trajectory compared to a fixed point.

The results presented in Section 5 show that by using different approaches when choosing the weights in the cost function, the needed prediction horizon for the minimization problem can be reduced and quicker convergence is achieved.

The real process is then controlled with good results after first determining the physical parameters of the nonlinear process using nonlinear grey-box system identification and using the parameters to construct a nonlinear MPC controller which deals with the non-minimum phase behavior of the process.

References

Andrieu, C., A. Doucet, and R. Holenstein (2010). "Particle Markov chain Monte Carlo methods". *Journal of the Royal Statistical Society, Series B* **72**:2, pp. 1–33.

Cannon, M., J. Buerger, B. Kouvaritakis, and S. Rakovic (2011). "Robust tubes in nonlinear model predictive control". *IEEE Transactions on Automatic Control* **56**:8, pp. 1942–1947.

Cappé, O., E. Moulines, and T. Rydén (2005). *Inference in Hidden Markov Models*. Springer Series in Statistics. Springer, New York, USA.

Clarke, D. (1987). "Generalized predictive control Part I. — The basic algorithm". *Automatica* **23**:2, pp. 137–148.

Delyon, B., M. Lavielle, and E. Moulines (1999). "Convergence of a stochastic approximation version of the EM algorithm". *The Annals of Statistics* **27**:1, pp. 94–128.

Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). "Maximum likelihood from incomplete data via the EM algorithm". *Journal of the Royal Statistical Society B* **39**:1, pp. 1–38.

Doucet, A. and A. M. Johansen (2009). "A tutorial on particle filtering and smoothing: fifteen years later". *Handbook of Nonlinear Filtering* **12**, pp. 656–704.

Gibson, S. and B. Ninness (2005). "Robust maximum-likelihood estimation of multivariable dynamic systems". *Automatica* **41**:10, pp. 1667–1682.

Johansson, K. H. (2000). "The quadruple-tank process: a multivariable laboratory process with an adjustable zero". *Control Systems Technology, IEEE Transactions on* **8**:3, pp. 456–465.

Kantas, N., A. Doucet, S. S. Singh, J. Maciejowski, and N. Chopin (2014). "On particle methods for parameter estimation in state-space models". arXiv:1412.8695, submitted to Statistical Science.

Kouvaritakis, B., M. Cannon, and J. Rossiter (1999). "Non-linear model based predictive control". *International Journal of Control* **72**:10, pp. 919–928.

Lindsten, F. (2013). "An efficient stochastic approximation EM algorithm using conditional particle filters". In: *Proceedings of the 38th International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. Vancouver, Canada.

Lindsten, F., M. I. Jordan, and T. B. Schön (2014). "Particle Gibbs with ancestor sampling". *The Journal of Machine Learning Research* **15**:1, pp. 2145–2184.

Lindsten, F. and T. Schön (2013). "Backward simulation methods for Monte Carlo statistical inference". *Foundations and Trends in Machine Learning* **6**:1, pp. 1–143. ISSN: 1935-8237. DOI: 10.1561/2200000045. URL: http://dx.doi.org/10.1561/2200000045.

Mattingley, J. and S. Boyd (2012). "CVXGEN: a code generator for embedded convex optimization". *Optimization and Engineering* **13**:1, pp. 1–27.

Nordh, J. (2013). *PyParticleEst - Python library for particle based estimation methods*. URL: http://www.control.lth.se/Staff/JerkerNordh/pyparticleest.html.

Oliveira, N. D. and L. Biegler (1995). "An extension of Newton-type algorithms for nonlinear process control". *Automatica: A Journal Of IFAC The International Federation Of Automatic Control* **31**:2, pp. 281–286.

Richalet, J. and D. O'Donovan (2009). *Predictive Functional Control: Principles and Industrial Applications*. Springer Science & Business Media, pp. 145–157.

Richalet, J., A. Rault, J. Testud, and J. Papon (1978). "Model predictive heuristic control: applications to industrial processes". *Automatica* **14**:5, pp. 413–428.

Schön, T. B., A. Wills, and B. Ninness (2011). "System identification of nonlinear state-space models". *Automatica* **47**:1, pp. 39–49.

Shumway, R. H. and D. S. Stoffer (1982). "An approach to time series smoothing and forecasting using the EM algorithm". *Journal of Time Series Analysis* **3**:4, pp. 253–264.

Paper IV

Particle Filtering Based Identification for Autonomous Nonlinear ODE Models

Jerker Nordh Torbjörn Wigren[1]
Thomas B. Schön[1] **Bo Bernhardsson**

Abstract

This paper presents a new black-box algorithm for identification of a nonlinear autonomous system in stable periodic motion. The particle filtering based algorithm models the signal as the output of a continuous-time second order ordinary differential equation (ODE). The model is selected based on previous work which proves that a second order ODE is sufficient to model a wide class of nonlinear systems with periodic modes of motion, also systems that are described by higher order ODEs. Such systems are common in systems biology. The proposed algorithm is applied to data from the well-known Hodgkin-Huxley neuron model. This is a challenging problem since the Hodgkin-Huxley model is a fourth order model, but has a mode of oscillation in a second order subspace. The numerical experiments show that the proposed algorithm does indeed solve the problem.

Submitted to the *17th IFAC Symposium on System Identification*, Beijing, China, October 2015

[1] Department of Information Technology, Uppsala University

1. Introduction

The identification of nonlinear autonomous systems is a fairly unexplored field. While a very large amount of work has been devoted to the analysis of autonomous systems, given an ordinary differential equation (ODE), the inverse problem has received much less attention. At the same time system identification based on particle filtering ideas is expanding rapidly. However, little attention has so far been given to identification of nonlinear autonomous systems. The present paper addresses this by presenting a new approach for identification of nonlinear autonomous ODE model based on recently developed particle filtering methods. Furthermore, the paper presents new results on neural modeling, by applying the new algorithm to data generated by the well-known Hodgkin-Huxley model. These constitutes the two main contributions of the paper.

As stated above, much work has been performed on the analysis of a given nonlinear autonomous system, see e.g. [Khalil, 1996]. The classical analysis provided by e.g. Poincaré provide tools for prediction of the existence of periodic orbits of a second order ODE. Bifurcation analysis and similar tools have also been widely applied to the analysis of chaos, inherent in nonlinear autonomous ODEs [Khalil, 1996; Li et al., 2007]. There are much less publications on and connections to the inverse problem, i.e. the identification of an autonomous nonlinear ODE from measured data alone. However, algorithms tailored for identification of second order ODEs from periodic data appeared in [Wigren et al., 2003a], [Manchester et al., 2011] and [Wigren, 2014]. A result on identifiability that gives conditions for when a second order ODE is sufficient for modeling of periodic oscillations is also available, see [Wigren and Söderström, 2005] and [Wigren, 2015]. That work proves that in case the phase plane of the data is such that the orbit does not intersect itself, then a second order ODE is always sufficient for identification. In other words, higher order models cannot be uniquely identifiable.

There is a vast literature on stable oscillations in biological and chemical systems, see e.g. [Rapp, 1987]. An important example is given by neuron spiking [Doi et al., 2002; Izhikevich, 2003; Hodgkin and Huxley, 1952]. This spiking is fundamental in that it is the way nerve cells communicate, for example in the human brain. The field of reaction kinetics provides further examples of dynamic systems that are relevant in the field of molecular systems biology, see e.g. [Ashmore, 1975]. Such kinematic equations typically result in systems of ordinary differential equations with right hand sides where fractions of polynomials in the states appear. The many authors that have dealt with identification of the Hodgkin-Huxley model have typically built on complete models, including an input signal current, see e.g. [Doi et al., 2002; Saggar et al., 2007; Tobenkin et al., 2010; Manchester et al., 2011; Lankarany et al., 2013]. With the exception of [Wigren, 2015] there does no

seem to have been much work considering the fact that identification may not be possible if only periodic data is available.

The particle filter was introduced more that two decades ago as a solution to the nonlinear state estimation problem, see e.g. [Doucet and Johansen, 2009] for an introduction. However, when it comes to the nonlinear system identification problem in general, it is only relatively recently that the particle filter has emerged as a really useful tool, see [Kantas et al., 2014] for a recent survey. The algorithm presented here is an adaptation of the so-called PSAEM algorithm introduced by [Lindsten, 2013]. It provides a solution to the nonlinear maximum likelihood problem by combining the stochastic approximation expectation maximization algorithm of [Delyon et al., 1999] with the PGAS kernel of [Lindsten et al., 2014]. This improves upon the earlier work of [Wigren et al., 2003b], since it is no longer necessary to rely on the sub-optimal extended Kalman filter and restrictive models of the model parameters.

2. Formulating the model and the problem

The periodic signal is modeled as the output of a second order differential equation, and can in continuous-time thus be represented as

$$x_t = \begin{pmatrix} p_t & v_t \end{pmatrix}^{\mathsf{T}}, \tag{1a}$$

$$\dot{x}_t = \begin{pmatrix} v_t \\ f(p_t, v_t) \end{pmatrix}. \tag{1b}$$

The discretized model is obtained by a Euler forward approximation and by introducing noise acting on the second state and on the measurement according to

$$p_{t+1} = p_t + hv_t, \tag{2a}$$

$$v_{t+1} = v_t + hf(p_t, v_t) + w_t, \qquad w_t \sim \mathcal{N}(0, Q^w), \tag{2b}$$

$$y_t = p_t + e_t, \qquad e_t \sim \mathcal{N}(0, R). \tag{2c}$$

The noise is only acting on one of the states, implying that one of the states can be marginalized resulting in the following non-Markovian model

$$\begin{aligned} p_{t+1} &= p_t + h(v_{t-1} + hf(p_{t-1}, v_{t-1}) + w_{t-1}) \\ &= 2p_t - p_{t-1} + h^2 f\left(p_{t-1}, \frac{p_t - p_{t-1}}{h}\right) + hw_{t-1}, \end{aligned} \tag{3a}$$

$$y_t = p_t + e_t, \tag{3b}$$

where the noise is still Gaussian according to

$$w_{t-1} \sim \mathcal{N}(0, Q^w), \qquad e_t \sim \mathcal{N}(0, R). \tag{3c}$$

As suggested by [Wigren et al., 2003b], the function $f(p, v)$ is parametrized according to

$$f(p, v) = \sum_{i=0}^{m} \sum_{j=0}^{m} a_{ij} p^i v^j. \tag{4}$$

Here, m is a design parameter deciding the degree of the model used for the approximation and the indices a_{ij} denote unknown parameters to be estimated together with the process noise covariance Q^w. Hence, the unknown parameters to be estimated are given by

$$\theta = \{Q^w \quad a_{00} \quad \ldots \quad a_{0m} \quad \ldots \quad a_{m0} \quad \ldots \quad a_{mm}\}. \tag{5}$$

It has been assumed that the measurement noise covariance R is known. The problem under consideration is that of computing the maximum likelihood (ML) estimate of the unknown parameters θ by solving

$$\widehat{\theta}_{\mathrm{ML}} = \operatorname*{argmax}_{\theta} \log p_\theta(y_{1:T}), \tag{6}$$

where $y_{1:T} = \{y_1, \ldots, y_T\}$ and $p_\theta(y_{1:T})$ denotes the likelihood function parameterized by θ.

3. Particle filtering for autonomous system identification

After the marginalization of the v state in model (2) the problem becomes non-Markovian, for an introduction to non-Markovian particle methods see e.g., [Lindsten et al., 2014] and [Lindsten and Schön, 2013]. The remainder of this section will go through the components required for the algorithm and note specific design choices made to apply the methods to the particular class of problems that are of interest in this paper.

3.1 Expectation maximization algorithms

The expectation Maximization (EM) algorithm [Dempster et al., 1977] is an iterative algorithm to compute ML estimates of unknown parameters (here θ) in probabilistic models involving latent variables (here, the state trajectory $x_{1:T}$). More specifically, the EM algorithm solves the ML problem (6) by iteratively computing the so-called *intermediate quantity*

$$\mathcal{Q}(\theta, \theta_k) = \int \log p_\theta(x_{1:T}, y_{1:T}) p_{\theta_k}(x_{1:T} \mid y_{1:T}) \mathrm{d}x_{1:T} \tag{7}$$

and then maximizing $\mathcal{Q}(\theta, \theta_k)$ w.r.t. θ. There is now a good understanding of how to make use of EM-type algorithms to identify dynamical systems. The linear state space model allows us to express everything in closed form [Shumway and Stoffer, 1982; Gibson and Ninness, 2005]. However, when it comes to nonlinear models, like the ones considered here, approximate methods have to be used, see e.g. [Lindsten, 2013; Schön et al., 2011; Cappé et al., 2005].

The sequential Monte Carlo (SMC) methods [Doucet and Johansen, 2009] or the particle Markov chain Monte Carlo (PMCMC) methods introduced by [Andrieu et al., 2010] can be exploited to approximate the joint smoothing density (JSD) arbitrarily well according to

$$\widehat{p}(x_{1:T} \,|\, y_{1:T}) = \sum_{i=1}^{N} w_T^i \delta_{x_{1:T}^i}(x_{1:T}). \tag{8}$$

Here, $x_{1:T}^i$ denotes the samples (also referred to as particles, motivating the name particle filter/smoother), w_T^i denotes the corresponding weights and δ_x denotes a point-mass distribution at x. [Schön et al., 2011] used the SMC approximation (8) to approximate the intermediate quantity (7). However, there is room to make even more efficient use of the particles in performing ML identification, by making use of the *stochastic approximation* developments within EM according to [Delyon et al., 1999]. In the so-called stochastic approximation expectation maximization (SAEM) algorithm, the intermediate quantity (7) is replaced by the following stochastic approximation update

$$\widehat{\mathcal{Q}}_k(\theta) = (1 - \gamma_k)\widehat{\mathcal{Q}}_{k-1}(\theta) + \gamma_k \log p_\theta(x_{1:T}[k], y_{1:T}), \tag{9}$$

where γ_k denotes the step size, which is a design parameter that must fulfill $\sum_{k=1}^{\infty} \gamma_k = \infty$ and $\sum_{k=1}^{\infty} \gamma_k^2 < \infty$. Furthermore, $x_{1:T}[k]$ denotes a sample from the JSD $p_{\theta_k}(x_{1:T} \,|\, y_{1:T})$. The sequence θ_k generated by the SAEM algorithm outlined above will under fairly weak assumptions converge to a maximizer of $p_\theta(y_{1:T})$ [Delyon et al., 1999].

For the problem under consideration the recently developed PMCMC methods [Andrieu et al., 2010; Lindsten et al., 2014] are useful to approximately generate samples from the JSD. This was realized by [Lindsten, 2013], resulting in the so-called particle SAEM (PSAEM) algorithm, which is used in this work.

3.2 The PGAS kernel

The particle Gibbs with ancestor sampling (PGAS) kernel was introduced by [Lindsten et al., 2014]. It is a procedure very similar to the standard particle filter, save for the fact that conditioning on one so-called *reference trajectory* $x'_{1:T}$ is performed. Hence, $x'_{1:T}$ have to be retained throughout the sampling

procedure. For a detailed derivation see [Lindsten et al., 2014], where it is also shown that the PGAS kernel implicitly defined via Algorithm 1 is uniformly ergodic. Importantly, it also leaves the target density $p(x_{1:T} \mid y_{1:T})$ invariant for any finite number of particles $N > 1$ implying that the resulting state trajectory $x^{\star}_{1:T}$ can be used as a sample from the JSD. The notation used in Algorithm 1 is as follows, $\mathbf{x}_t = (x_t^1, \ldots, x_t^N)$ denotes all the particles at time t and $\mathbf{x}_{1:T} = (\mathbf{x}_1, \ldots, \mathbf{x}_T)$ the entire trajectories. The particles are propagated according to a proposal distribution $r_t(x_t \mid x_{t-1}, y_t)$. The resampling step and the propagation step of the standard particle filter has been collapsed into jointly sampling the particles $\{x_t^i\}_{i=1}^N$ and the ancestor indices $\{a_t^i\}_{i=1}^N$ independently from

$$M_t(a_t, x_t) = \frac{w_t^{a_t}}{\sum_{l=1}^N w_t^l} r_t(x_t \mid x_{1:t-1}^{a_t}, y_{1:t}). \tag{10}$$

Finally, W_t denotes the weight function,

$$W_t(x_{1:t}, y_{1:t}) = \frac{p(y_t \mid x_{1:t}) p(x_t \mid x_{1:t-1})}{r(x_t \mid x_{1:t-1}, y_{1:t})}. \tag{11}$$

Algorithm 1 PGAS kernel

1: **Initialization ($t = 1$):** Draw $x_1^i \sim r_1(x_1 | y_1)$ for $i = 1, \ldots, N-1$ and set $x_1^N = x'_1$. Compute $w_1^i = W_1(x_1^i)$ for $i = 1, \ldots, N$.
2: **for** $t = 2$ to T **do**
3: Draw $\{a_t^i, x_t^i\} \sim M_t(a_t, x_t)$ for $i = 1, \ldots, N-1$.
4: Set $x_t^N = x'_t$.
5: Draw a_t^N with $\mathbb{P}\left(a_t^N = i\right) \propto \frac{w_{t-1}^i p(x'_t \mid x_{1:t-1}^i)}{\sum_{l=1}^N w_{t-1}^l p(x'_t \mid x_{1:t-1}^l)}$
6: Set $x_{1:t}^i = \{x_{1:t-1}^{a_t^i}, x_t^i\}$ for $i = 1, \ldots, N$.
7: Compute $w_t^i = W_t(x_{1:t}^i, y_{1:t})$ for $i = 1, \ldots, N$.
8: **end for**
9: **Return** $\mathbf{x}_{1:T}, \mathbf{w}_T$.

3.3 Identifying autonomous systems using PSAEM

The PSAEM algorithm for ML identification of autonomous systems now simply amounts to making use of the PGAS kernel in Algorithm 1 to generate a particle system $\{x_{1:T}^i, w_T^i\}_{i=1}^N$ that is then used to approximate the intermediate quantity according to

$$\widehat{\mathcal{Q}}_k(\theta) = (1 - \gamma_k)\widehat{\mathcal{Q}}_{k-1}(\theta) + \gamma_k \sum_{i=1}^N w_T^i \log p_\theta(x_{1:T}^i, y_{1:T}). \tag{12}$$

Note that similarly to (9) only the reference trajectory $x_{1:T}[k]$ could have been used, but in making use of the entire particle system the variance of the resulting state estimates are reduced [Lindsten, 2013]. The result is provided in Algorithm 2.

Algorithm 2 PSAEM for sys. id. of autonomous systems

1: **Initialization:** Set $\theta[0] = (Q_0^w \quad 0^\mathsf{T})$ and set $x_{1:T}[0]$ using an FFBSi particle smoother. Set $\widehat{\mathcal{Q}}_0 = 0$ and set $\mathbf{w}[0]$ to an empty vector.
2: Draw $x'_{1:T}$ using FFBSi.
3: **for** $k \geq 1$ **do**
4: Draw $\mathbf{x}_{1:T}[k], \mathbf{w}_T$ by running Algorithm 1 using $x'_{1:T}$ as reference.
5: Draw j with $\mathbb{P}(j = i) = w_T^i$.
6: Set $x'_{1:T} = x_{1:T}^j[k]$
7: Set $\mathbf{w}[k] = ((1 - \gamma_k)\mathbf{w}[k-1] \quad \gamma_k \mathbf{w}_T)$
8: Compute $\widehat{\mathcal{Q}}_k(\theta)$ according to (12).
9: Compute $\theta[k] = \operatorname{argmax} \widehat{\mathcal{Q}}_k(\theta)$.
10: **if** termination criterion is met **then**
11: **return** $\{\theta[k]\}$
12: **end if**
13: **end for**

Note that the initial reference trajectory $x_{1:T}[0]$ is obtained by running a so-called forward filter backward simulator (FFBSi) particle smoother, see [Lindsten and Schön, 2013] for details. To indicate that the trajectories were generated at iteration k, we use $\mathbf{x}_{1:T}[k]$ and analogously for the weights. The 9$^{\text{th}}$ row of Algorithm 2 will for the model (3) under consideration amount to a weighted least squares problem, which is solved in Algorithm 3. For the work presented in this article the run Algorithm 2 for a fixed number of iterations, which gives the termination criterion.

Algorithm 3 Maximizing $\mathcal{Q}$

1: For each trajectory in $\mathbf{x}_{1:T}[k]$ calculate the velocity at each time $v_t^{(i)} = (p_{t+1}^{(i)} - p_t^{(i)})/h$
2: For each time step and for each trajectory in $\mathbf{x}_{1:T}[k]$, evaluate $f(p_t^{(i)}, v_t^{(i)})$.
3: Find $a_{00} ... a_{mm}$ via the related weighted least squares (WLS) problem.
4: Find Q^w by estimating the covariance of the residuals of the WLS-problem.
5: Set $\theta[k] = \{Q^w \quad a_{00} \quad ... \quad a_{mm}\}$.

3.4 Choosing the proposal distribution

The proposal density $r(x_t|x_{1:t-1}, y_{1:t})$ constitutes an important design choice of the particle filter that will significantly affect its performance. The commonly used bootstrap particle filter amounts to making use of the dynamics

to propose new particles, i.e. $r(x_t|x_{1:t-1}, y_{1:t}) = p(x_t \,|\, x_{1:t-1})$. However, we will make use of the measurement model and the information present in the current measurement to propose new particles, i.e. $r(x_t|x_{1:t-1}, y_{1:t}) = p(y_t \,|\, x_t)$. This is enabled by the marginalization of the deterministic state in the model, since the dimension of the state-space then matches that of the measurement. As a possible further improvement, the proposal could also include the predicted state using the model. Recently, [Kronander and Schön, 2014] showed that the combined use of both the dynamics and the measurements results in competitive algorithms. In such a scenario the estimated uncertainty in the model would initially be large and thus not affect the proposal distribution significantly, but for each iteration of the PSAEM algorithm the model will be more and more accurate in predicting the future states, and its influence on the proposal distribution would increase accordingly.

4. Numerical illustrations

The performance of the proposed algorithm is illustrated using two examples, namely the Van det Pol oscillator in Section 4.1 and the Hodgkin-Huxley neuron model in Section 4.2 – 4.3. There is no prior knowledge of the model, hence it is assumed that all the parameters a_{ij} in (4) are zero. The initial covariance Q_0^w is set to a large value. The Python source code for the following examples can be downloaded from [Nordh, 2015], the implementation was carried out using the pyParticleEst software framework [Nordh, 2013].

4.1 Validation on the Van der Pol oscillator

To demonstrate the validity of the proposed solution it is first applied to the classic Van der Pol oscillator [Khalil, 1996], which belongs to the model class defined by (3). The oscillator is described by

$$\dot{x}_t = \begin{pmatrix} v_t \\ -p_t + 2(1 - p_t^2)v_t \end{pmatrix}, \tag{13}$$

where $x_t = (p_t \quad v_t)^\mathsf{T}$. Performing 50 iterations of Algorithm 2 gives the parameter convergence shown in Fig. 1. Here $N = 15$ particles were used for the PGAS sampler, and the model order was chosen as $m = 2$. For the SAEM step, the sequence $\gamma_{1:K}$ is chosen as

$$\gamma_k = \begin{cases} 1 & \text{if } k \leq 5, \\ (k-5)^{-0.9} & \text{if } k > 5. \end{cases} \tag{14}$$

It can be seen that the parameters converge to values close to those obtained from the Euler forward discretization. To analyze the behavior further

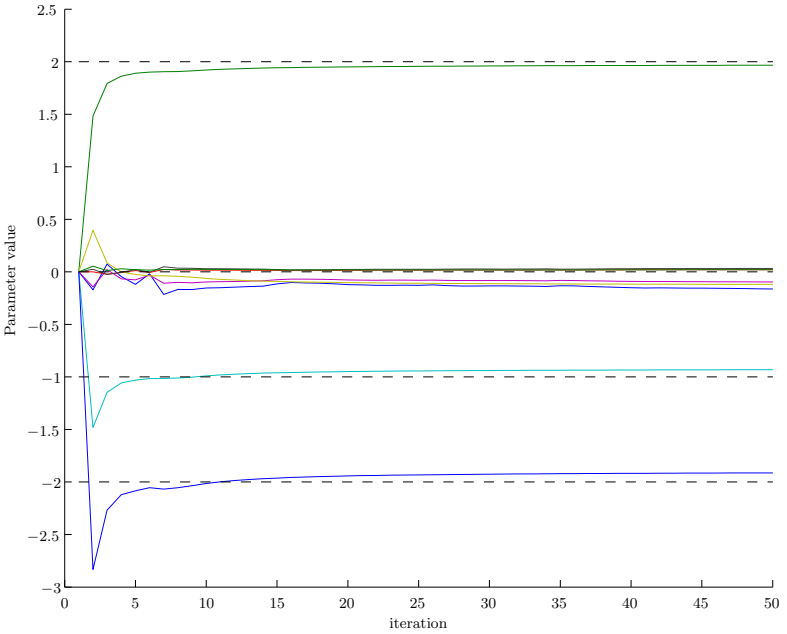

Figure 1. Parameter convergence for the Van der Pol example (13). The dashed lines are the coefficients obtained from the discretization of the model, the other lines represent the different parameters in the identified model. Note that while the parameters do not converge to the 'true' values, they provide a very accurate model for predicting the signal as shown in Fig. 3.

the phase-plane for the system is shown in Fig. 2 and the time-domain realization in Fig. 3. Here it can be seen that the identified model captures the behavior of the true continuous-time system significantly better than the model obtained from the discretization.

4.2 The Hodgkin-Huxley neuron model

The well-known Hodgkin-Huxley model uses a nonlinear ODE to describe the dynamics of the action potentials in a neuron. In this paper the model will be used for two purposes. First, simulated spiking neuron data is used to characterize the performance of the proposed algorithm when identifying a nonlinear autonomous system. It should be noted that the data does not correspond to a system that is in the model set. The ability to handle nonlinear under-modeling is therefore also assessed. Secondly, the new algorithm and the identification results contribute to an enhanced understanding of spiking neurons by providing better performance compared to previous algorithms.

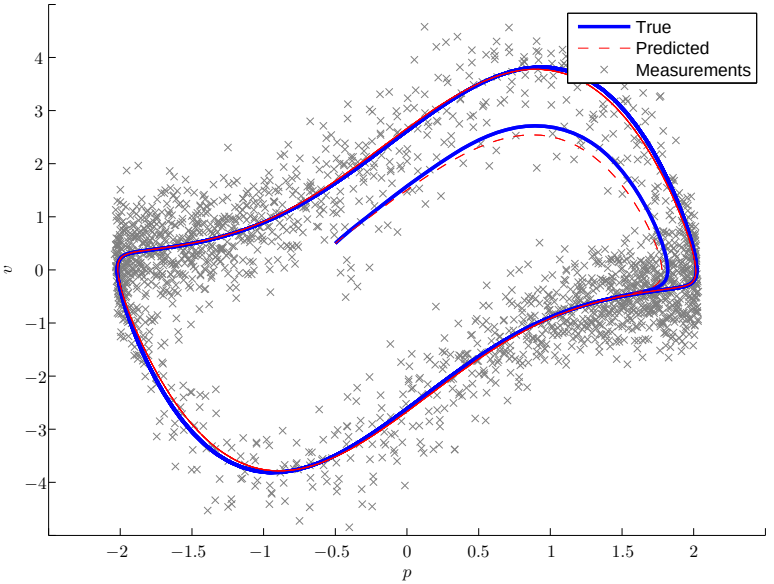

Figure 2. Phase-plane plot for the Van der Pol example (13).

The Hodgkin-Huxley model formulation of [Siciliano, 2012] is used, where the following ODE is given

$$\begin{aligned}
\frac{dv}{dt} &= \frac{1}{C_m}\left[I - g_{na}m^3h(v - E_{na})\right. \\
&\quad \left.\times g_K n^4(v - E_K) - g_l(v - E_l)\right], && \text{(15a)}\\
\frac{dn}{dt} &= \alpha_n(v)(1 - n) - \beta_n(v)n, && \text{(15b)}\\
\frac{dm}{dt} &= \alpha_m(v)(1 - m) - \beta_m(v)m, && \text{(15c)}\\
\frac{dh}{dt} &= \alpha_h(v)(1 - h) - \beta_h(v)h. && \text{(15d)}
\end{aligned}$$

Here, v denotes the potential, while n, m and h relate to each type of gate of the model and their probabilities of being open, see [Siciliano, 2012] for details. The applied current is denoted by I. The six rate variables are described

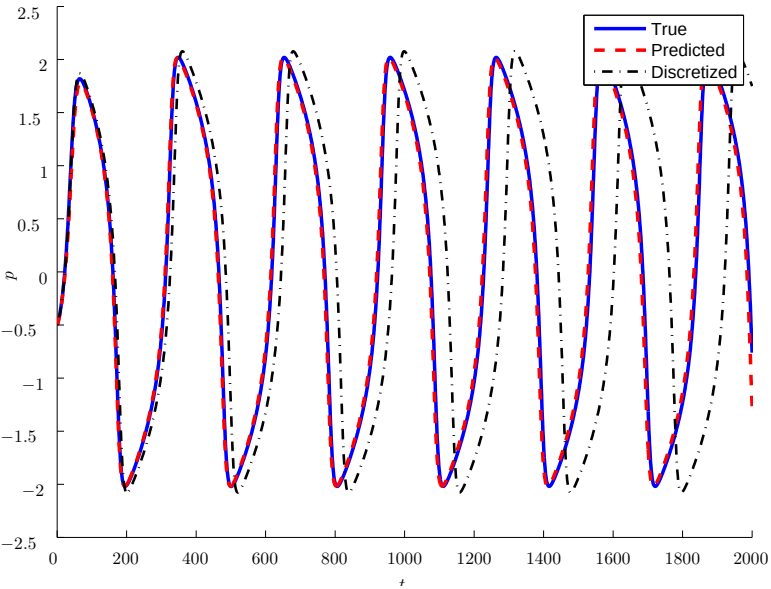

Figure 3. Predicted output using the true initial state and the estimated parameters. The plot labeled "discretized" is obtained by discretization of the true continuous-time model using Euler forward and using that to predict the future output. It can been seen that the discretized model diverges over time from the true signal, clearly the identified parameter values give a better estimate than using the values from the discretization.

by the following nonlinear functions of v

$$\alpha_n(v) = \frac{0.01(v+50)}{1-e^{-(v+50)/10}}, \tag{16a}$$

$$\beta_n(v) = 0.125e^{-(v+60)/80}, \tag{16b}$$

$$\alpha_m(v) = \frac{0.1(v+35)}{1-e^{-(v+35)/10}}, \tag{16c}$$

$$\beta_m(v) = 4.0e^{-0.0556(v+60)}, \tag{16d}$$

$$\alpha_h(v) = 0.07e^{-0.05(v+60)}, \tag{16e}$$

$$\beta_h(v) = \frac{1}{1+e^{-0.1(v+30)}}. \tag{16f}$$

The corresponding numerical values are given by $C_m = 0.01\ \mu F/cm^2$, $g_{Na} = 1.2\ mS/cm^2$, $E_{Na} = 55.17\ mV$, $g_K = 0.36\ mS/cm^2$, $E_K = -72.14\ mV$, $g_l = 0.003\ mS/cm^2$, and $E_l = -49.42\ mV$.

4.3 Identifying the spiking mode of the Hodgkin-Huxley model

Using a simulated dataset of length $T = 10\,000$, a model of the form (3) with $m = 3$ is selected. Algorithm 2 was run for 200 iterations, employing $N = 20$ particles in the PGAS kernel. For the SAEM step, the sequence $\gamma_{1:K}$ is chosen as

$$\gamma_k = \begin{cases} 1 & \text{if } k \leq 40, \\ (k-40)^{-0.5} & \text{if } k > 40. \end{cases} \tag{17}$$

The predicted phase-plane of the identified model is shown in Fig. 4 along with the true phase-plane and the measurements.

Fig. 5 shows the same dataset in the time-domain, it shows the true signal (without added noise) and the output predicted by the identified model when initialized with the correct initial state. It can be seen that the model captures the behaviour of the signal, but introduces a small error in the frequency of the signal, causing the predicted and true signal to diverge over time.

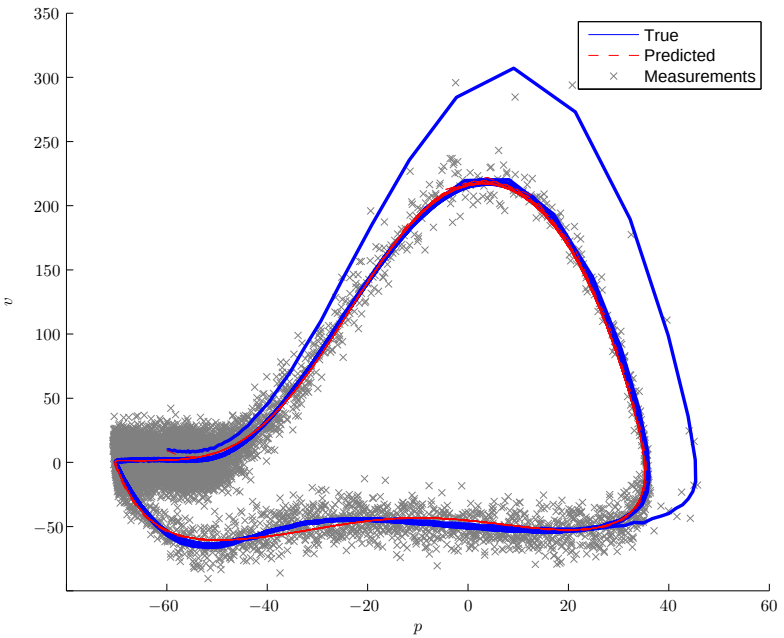

Figure 4. Phase-plane for the Hodgkin-Huxley dataset. It can be seen that the predicted model fails to capture the initial transient, but accurately captures the limit cycle.

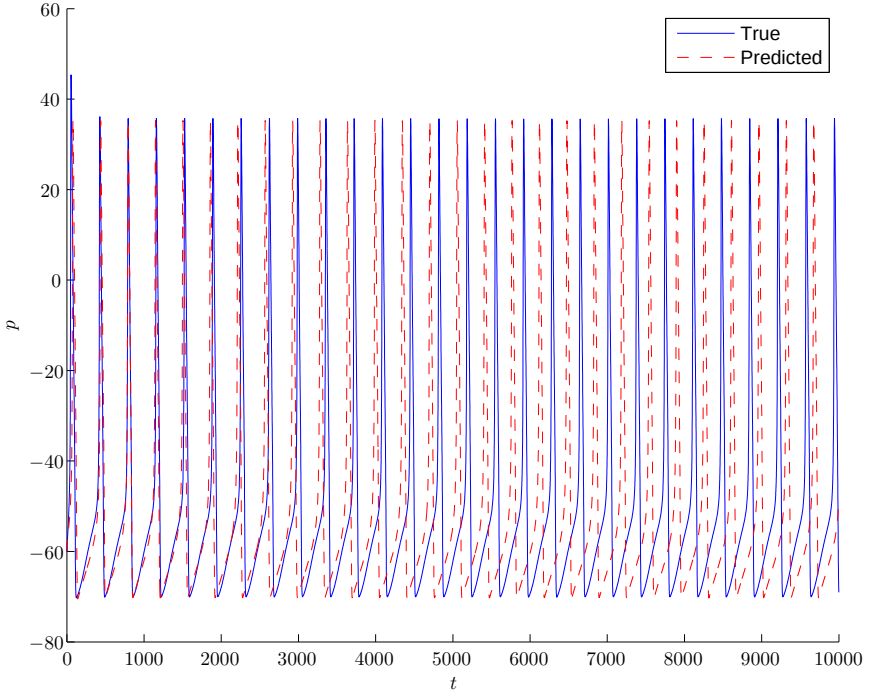

Figure 5. Predicted output using the true initial state and the estimated parameters. The overall behaviour of the signal is captured, but there is a small frequency error which leads to the predicted signal slowly diverging from the true signal.

5. Conclusions

The new identification method successfully identified models for both the Van der Pol example and for the more complex Hodgkin-Huxley model. Even though the Hodgkin-Huxley model in general cannot be reduced to a second order differential equation it is possible to identify a good second order model for its limit cycle as shown in Fig. 4.

The use of a Bayesian nonparametric model in place of (4) constitutes an interesting continuation of this work. A first step in this direction would be to employ the Gaussian process construction by [Frigola et al., 2013; Frigola et al., 2014].

Acknowledgements

This work was supported by the project *Probabilistic modeling of dynamical systems* (Contract number: 621-2013-5524) funded by the Swedish Research Council. Bo Bernhardsson and Jerker Nordh are members of the LCCC Linnaeus Center and the ELLIIT Excellence Center at Lund University.

References

Andrieu, C., A. Doucet, and R. Holenstein (2010). "Particle Markov chain Monte Carlo methods". *Journal of the Royal Statistical Society, Series B* **72**:2, pp. 1–33.

Ashmore, P. G. (1975). *Kinetics of Oscillatory Reactions*. The Chemical Society.

Cappé, O., E. Moulines, and T. Rydén (2005). *Inference in Hidden Markov Models*. Springer Series in Statistics. Springer, New York, USA.

Delyon, B., M. Lavielle, and E. Moulines (1999). "Convergence of a stochastic approximation version of the EM algorithm". *The Annals of Statistics* **27**:1, pp. 94–128.

Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). "Maximum likelihood from incomplete data via the EM algorithm". *Journal of the Royal Statistical Society B* **39**:1, pp. 1–38.

Doi, S., Y. Onada, and S. Kumagai (2002). "Parameter estimation of various Hodgkin-Huxley-type neuronal models using a gradient-descent learning method". In: *Proceedings of the 41st SICE Annual Conference*. Osaka, Japan, pp. 1685–1688.

Doucet, A. and A. M. Johansen (2009). "A tutorial on particle filtering and smoothing: fifteen years later". *Handbook of Nonlinear Filtering* **12**, pp. 656–704.

Frigola, R., F. Lindsten, T. B. Schön, and C. E. Rasmussen (2013). "Bayesian inference and learning in Gaussian process state-space models with particle MCMC". In: *Advances in Neural Information Processing Systems (NIPS) 26*, pp. 3156–3164.

Frigola, R., F. Lindsten, T. B. Schön, and C. E. Rasmussen (2014). "Identification of Gaussian process state-space models with particle stochastic approximation EM". In: *Proceedings of the 18th World Congress of the International Federation of Automatic Control (IFAC)*. Cape Town, South Africa.

Gibson, S. and B. Ninness (2005). "Robust maximum-likelihood estimation of multivariable dynamic systems". *Automatica* **41**:10, pp. 1667–1682.

Hodgkin, A. L. and A. F. Huxley (1952). "A quantitative description of membrane current and its application to conduction and excitation in nerve". *The Journal of Physiology* **117**:4, pp. 500–544.

Izhikevich, E. M. (2003). "Simple models of spiking neurons". *IEEE Transactions on Neural Networks* **14**:6, pp. 1569–1572.

Kantas, N., A. Doucet, S. S. Singh, J. Maciejowski, and N. Chopin (2014). "On particle methods for parameter estimation in state-space models". arXiv:1412.8695, submitted to Statistical Science.

Khalil, H. K. (1996). *Nonlinear Systems*. Prentice Hall, Upper Saddle River.

Kronander, J. and T. B. Schön (2014). "Robust auxiliary particle filters using multiple importance sampling". In: *Proceedings of the IEEE Statistical Signal Processing Workshop (SSP)*. Gold Coast, Australia.

Lankarany, M., W.-P. Zhu, and N. S. Swamy (2013). "Parameter estimation of Hodgkin-Huxley neuronal model using dual extended Kalman filter". In: *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*. Beijing, China, pp. 2493–2496.

Li, H.-Y., Y.-Q. Che, H.-Q. Gao, F. Dong, and J. Wang (2007). "Bifurcation analysis of the Hodgkin-Huxley model exposed to external DC electric field". In: *Proceedings of the 22nd International Symposium on Intelligent Control (ISIC)*. Singapore, pp. 271–276.

Lindsten, F. (2013). "An efficient stochastic approximation EM algorithm using conditional particle filters". In: *Proceedings of the 38th International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. Vancouver, Canada.

Lindsten, F., M. I. Jordan, and T. B. Schön (2014). "Particle Gibbs with ancestor sampling". *The Journal of Machine Learning Research* **15**:1, pp. 2145–2184.

Lindsten, F. and T. Schön (2013). "Backward simulation methods for Monte Carlo statistical inference". *Foundations and Trends in Machine Learning* **6**:1, pp. 1–143. ISSN: 1935-8237. DOI: 10.1561/2200000045. URL: http://dx.doi.org/10.1561/2200000045.

Manchester, I., M. M. Tobenkin, and J. Wang (2011). "Identification of nonlinear systems with stable oscillations". In: *Proceedings of the 50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*. Orlando, FL, USA, pp. 5792–5797.

Nordh, J. (2013). *PyParticleEst - Python library for particle based estimation methods*. URL: http://www.control.lth.se/Staff/JerkerNordh/pyparticleest.html.

Nordh, J. (2015). *Example source code for "Particle filtering based identification for autonomous nonlinear ode models"*. URL: http://www.control.lth.se/Staff/JerkerNordh/ode-id.html.

Rapp, P. (1987). "Why are so many biological systems periodic?" *Progress in Neurobiology* **29**:3, pp. 261–273.

Saggar, M., T. Mericli, S. Andoni, and R. Miikulainen (2007). "System identification for the Hodgkin-Huxley model using artificial neural networks". In: *Proceedings of the International Joint Conference on Neural Networks*. Orlando, FL, USA.

Schön, T. B., A. Wills, and B. Ninness (2011). "System identification of nonlinear state-space models". *Automatica* **47**:1, pp. 39–49.

Shumway, R. H. and D. S. Stoffer (1982). "An approach to time series smoothing and forecasting using the EM algorithm". *Journal of Time Series Analysis* **3**:4, pp. 253–264.

Siciliano, R. (2012). *The Hodgkin–Huxley model – its extensions, analysis and numerics*. Tech. rep. Department of Mathematics and Statistics, McGill University, Montreal, Canada.

Tobenkin, M. M., I. Manchester, J. Wang, A. Megretski, and R. Tedrake (2010). "Convex optimization in identification of stable non-linear state space models". In: *Proceedings of the 49th IEEE Conference on Decision and Control (CDC)*. Atlanta, USA, pp. 7232–7237.

Wigren, T. (2014). *Software for recursive identification of second order autonomous systems*. Tech. rep. Department of Information Technologt, Uppsala University, Uppsala, Sweden. URL: `http://www.it.uu.se/research/publications/%5C%5Creports/2014-014/SWAutonomous.zip`.

Wigren, T. (2015). "Model order and identifiability of non-linear biological systems in stable oscillation". *IEEE/ACM Transactions on Computational Biology and Bioinformatics*. Accepted for publication.

Wigren, T., E. Abd-Elrady, and T. Söderström (2003a). "Harmonic signal analysis with Kalman filters using periodic orbits of nonlinear ODEs". In: *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. Hong Kong, PRC, pp. 669–672.

Wigren, T. and T. Söderström (2005). "A second order ODE is sufficient for modeling of many periodic systems". *International Journal of Control* **77**:13, pp. 982–986.

Wigren, T., E. Abd-Elrady, and T. Söderström (2003b). "Least squares harmonic signal analysis using periodic orbits of ODEs". In: *Proceeding of 13th IFAC symposium on system identification (SYSID)*. Rotterdam, The Netherlands, pp. 1584–1589.

Paper V

Metropolis-Hastings Improved Particle Smoother and Marginalized Models

Jerker Nordh

Abstract

This paper combines the Metropolis-Hastings Improved Particle Smoother (MHIPS) with marginalized models. It demonstrates the effectiveness of the combination by looking at two examples; a degenerate model of a double integrator and a fifth order mixed linear/nonlinear Gaussian (MLNLG) model. For the MLNLG model two different methods are compared with the non-marginalized case; the first marginalizes the linear states only in the filtering part, the second marginalizes during both the filtering and smoothing pass. The results demonstrate that marginalization not only improves the overall performance, but also increases the rate of improvement for each iteration of the MHIPS algorithm. It thus reduces the required number of iterations to beat the performance of a Forward-Filter Backward Simulator approach for the same model.

Submitted to the *2015 European Signal Processing Conference*, Nice, France

1. Introduction

During the last decade particle filters have become popular for solving nonlinear estimation problems. For linear Gaussian systems the Kalman filter[Welch and Bishop, 1995] provides the optimal estimate, and some extensions such as the Extended Kalman Filter[Welch and Bishop, 1995] and Unscented Kalman Filter[Julier and Uhlmann, 2004] have been proposed to handle nonlinear systems. These make simplifying assumptions such as that the system can be locally approximated by a linearized model or that the resulting distribution will be Gaussian. For models where these assumptions do not hold particle filters[Doucet et al., 2000] can provide superior performance, some examples are multi-target tracking[Okuma et al., 2004] and Simultanous Localization and Mapping (SLAM) [Montemerlo et al., 2002]. Instead of assuming a Gaussian distribution the true distribution is approximated using a set of weighted point estimates, or particles,

$$\hat{p}(x_t|y_{1:t}) = \sum_{i=1}^{N} w_t^{(i)} \delta(x_t - x_t^{(i)}) \tag{1}$$

where $x_t^{(i)}$ are particles and $w_t^{(i)}$ the weights. As the number of particles increases the approximation approaches the true distribution.. A typical particle filter implementation is shown in algorithm 1, a common choice of proposal distribution is $r(x_{t+1}|x_t, y_{t+1}) = p(x_{t+1}|x_t)$ which leads to some simplifications during the computations. For a more detailed introduction to the particle filter see [Doucet et al., 2000].

Algorithm 1 Particle Filter

for $i = 1$ to N **do**
  Sample $x_1^{(i)} \sim r(x_1)$.
  Set $\tilde{w}_1^{(i)} = p(x_1^{(i)})p(y_1|x_1^{(i)})/r(x_1^{(i)})$
end for
Set $w_1^{(i)} = \tilde{w}_1^{(i)}/(\sum_{j=1}^{N} \tilde{w}_1^{(j)}), \quad \forall i \in [1, N]$.
for $t = 2$ to T **do**
  for $i = 1$ to N **do**
    Sample ancestor indices, a_t^i, with $\mathbb{P}\left(a_t^i = j\right) = w_{t-1}^{(j)}$.
    Sample $x_{t+1}^{(i)}$ from $r(x_{t+1}|x_t^{(a_t^i)}, y_{t+1})$
    Calculate weights:
    $\tilde{w}_{t+1}^{(i)} = p(y_{t+1}|x_{t+1}^{(i)})p(x_{t+1}|x_t^{(a_t^i)})/r(x_{t+1}|x_t^{(a_t^i)}, y_{t+1})$
  end for
  Set $w_t^{(i)} = \tilde{w}_t^{(i)}/(\sum_{j=1}^{N} \tilde{w}_t^{(j)}), \quad \forall i \in [1, N]$.
end for

A drawback with particle filters is that the number of particles needed increases with the dimension of the problem[Beskos et al., 2014][Rebeschini

and Handel, 2013], because of this it is of interest to marginalize over some of the states if possible to reduce the dimension of the estimation problem and thus the computational complexity. A typical example of this are models where some of the states only occur linearly and are only affected by additive Gaussian noise. Conditioned on the remainder of the states those could thus be optimally estimated using a Kalman filter. Filters exploiting this is typically referred to as Rao-Blackellized Particle Filters (RBPF)[Schön et al., 2005].

Conceptually by storing the ancestral paths of the particles the filter also provides a smoothed state estimate, i.e $p(x_t|y_{1:T})$ where $t < T$. However, due to the resampling step in the particle filter, in practice for all $t \ll T$ the ancestral path will be identical, thus providing a very poor approximation of the true posterior distribution[Lindsten and Schön, 2013]. Because of this a number of particle smoothing algorithms have been proposed, the most commonly used is the Forward Filter Backward Simulator (FFBSi). The FFBSi complements the particle filter by performing a backwards sweep where the ancestor of each particle is sampled using updated weights that depend on both the old filter weights, as well as the probability $p(x_{t+1}|x_t)$. For a more thorough introduction to particle smoothing see [Briers et al., 2010] and for the Rao-Blackwellized particle smoothing see [Lindsten and Schön, 2011].

A weakness with the FFBSi is that it only reuses the samples from the forward filter, this is something the Metropolis-Hastings Backward Proposer (MHBP) [Bunch and Godsill, 2013] and Metropolis-Hastings Improved Particle Smoother (MHIPS) addresses. The rest of this paper will focus on how to combine MHIPS with marginalized models, showing how to combine it with a previously published method[Lindsten et al., 2013][Lindsten and Schön, 2011] for marginalization of mixed linear/nonlinear Gaussian models (MLNLG). First it briefly discusses the, in general non-Markovian, filtering/smoothing problem then shows a trivial example of a model where the computational cost of the general solution can be avoided through the introduction of extra variables. These variables are propagated backwards during the smoothing step in similar manner as the mean and covariance are propagated forward in a RBPF. Section 2 introduces the example models used in this article, section 3 discussed MHIPS in detail and shows how to extend it for marginalized models and section 4 presents some results of applying the method to the example models and finally section 5 summarizes the paper.

2. Models

Two models are used in this paper, a degenerate double integrator and a fifth order MLNLG model. The double integrator is included to illustrate how the methods are affected by degeneracy and to introduce the concept used for the marginalization of MLNLG models in a simpler setting.

2.1 Double integrator

Due to the noise only acting on the input there is a deterministic relation between the two states making the model degenerate and not suitable for the standard particle smoothing methods. This coupling means that $p(x_{t+1}^{(i)}|x_t^{(j)}) = 0, \quad \forall j \neq a_i$ where a_i is the index of the ancestor for particle $x_{t+1}^{(i)}$.

$$x_{t+1} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} x_t + \begin{pmatrix} 0 \\ 1 \end{pmatrix} w_t \tag{2a}$$

$$y_t = \begin{pmatrix} 1 & 0 \end{pmatrix} x_t + e_t \tag{2b}$$

$$w_t \sim N(0, Q), \quad e_t \sim N(0, R) \tag{2c}$$

The model in (2) can be rewritten as a first order system with a non-Markovian structure, for notational brevity the notation $x_t = \begin{pmatrix} p_t & v_t \end{pmatrix}^T$ is introduced and the model can then be rewritten as

$$v_{t+1} = v_t + w_t \tag{3a}$$

$$y_t = p_0 + \sum_{i=0}^{t-1} v_i + e_t \tag{3b}$$

$$w_t \sim N(0, Q), \quad e_t \sim N(0, R) \tag{3c}$$

The estimation problem could now be solved using a non-Markovian particle smoother[Lindsten and Schön, 2013]. For this particular model it is possible to reduce the computational effort by propagating additional information in the forward and backward steps of the algorithms. During the filtering each particle also stores the sum of all its previous states. At a quick glance this looks like simply reintroducing the p-state from the original model, but the key distinction is that this new variable is a function of the past trajectory,

and not included as a state in the model.

$$v_{t+1} = v_t + w_t \tag{4a}$$
$$s_{t+1} = s_t + v_t \tag{4b}$$
$$y_t = s_t + e_t \tag{4c}$$
$$s_0 = p_0 \tag{4d}$$
$$w_t \sim N(0, Q), \quad e_t \sim N(0, R) \tag{4e}$$

The (non-Markovian) smoother will need to evaluate the probability density (5), but only up to proportionality which allows it to be rewritten as (6). Evaluating this directly leads to a computational effort for each time-step that grows with the length of the full dataset

$$\prod_{k=t+1}^{T} p(y_k|v_k)p(v_k|v_{1:k-1}, y_{1:k-1}) \tag{5}$$
$$\underset{v_{1:t}}{\propto} p(v_{t+1:T}, y_{t+1:T}|v_{1:t}, y_{1:t})$$
$$= p(y_{t+1:T}|v_{1:T}, y_{1:t})p(v_{t+1:T}|v_{1:t})$$
$$\underset{v_{1:t}}{\propto} p(y_{t+1:T}|v_{1:T})p(v_{t+1}|v_t) \tag{6}$$

This is clearly undesirable, but utilizing the same approach as the authors of [Lindsten et al., 2013] and noticing that (6) only needs to be evaluated up to proportionality (with regard to $v_{1:t}$) it is possible to propagate information backwards during the smoothing in the same way as the s_t variables propagates the sum during filtering. The first factor of (6) can be evaluated up to proportionality as follows

$$p(y_{t+1:T}|s_t, v_{t:T}) = \prod_{k=t+1}^{T} p(y_k|s_t, v_{t:T}) \tag{7a}$$
$$\underset{s_t, v_t}{\propto} \prod_{k=t+1}^{T} e^{(s_t+v_t)^2 - 2(y_k - \sum_{j=t+1}^{k-1} v_j)(s_t+v_t)} \tag{7b}$$
$$= e^{(T-t)(s_t+v_t)^2 - 2\sum_{k=t+1}^{T}(y_k - \sum_{j=t+1}^{k-1} v_j)(s_t+v_t)} \tag{7c}$$

Through the introduction of two new variables N_t, γ_t that are propagated backwards during the smoothing this allows (7) to be evaluated as

$$\log p(y_{t+1:T}|s_t, v_t, v_{t+1:T}) + \text{constant} =$$
$$\frac{1}{2R}(N_{t+1}(s_t + v_t)^2 - 2\gamma_{t+1}(s_t + v_t)) \tag{8a}$$
$$N_t = N_{t+1} + 1, \quad N_T = 1 \tag{8b}$$
$$\gamma_t = \gamma_{t+1} + y_t - N_{t+1}v_t, \quad \gamma_T = y_T \tag{8c}$$

Using (8) it is now possible to evaluate the required smoothing density in constant time.

2.2 Mixed Linear/Nonlinear Gaussian

This model was introduced in [Lindsten and Schön, 2011] as an extension to the commonly used standard nonlinear model, the difference is that the constant 25 has been replaced by the output of a fourth order linear system.

$$\xi_{t+1} = 0.5\xi_t + \theta_t \frac{\xi_t}{1 + \xi_t^2} + 8\cos 1.2t + v_{\xi,t} \tag{9a}$$
$$z_{t+1} = \begin{pmatrix} 3 & -1.691 & 0.849 & -0.3201 \\ 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \end{pmatrix} z_t + v_{z,t} \tag{9b}$$
$$y_t = 0.05\xi_t^2 + e_t \tag{9c}$$
$$\theta_t = 25 + \begin{pmatrix} 0 & 0.04 & 0.044 & 0.008 \end{pmatrix} z_t \tag{9d}$$
$$\xi_0 = 0, \quad z_0 = \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix}^T \tag{9e}$$
$$v_{\xi,t} \sim N(0, Q_\xi), \; v_{z,t} \sim N(0, Q_z) \tag{9f}$$
$$e_t \sim N(0, R) \tag{9g}$$

The z-states are not fully observable, but the affine combination θ is. The four z-states appear affinely and given the trajectory $\xi_{1:T}$ the remaining estimation problem could easily be solved using a regular Kalman filter. Several solutions have been proposed to utilize this fact when performing particle smoothing, in this paper two of these are compared. In the forward step both methods work the same way, marginalizing the linear states by computing the sufficient statistics, i.e. the mean value ($\bar{z}_t$) and the covariance (P_t), instead of sampling the z-states. This allows the densities $p(\xi_{t+1}|\xi_{1:t}, y_{1:t})$ and $p(y_t|\xi_{1:t}, y_{1:t-1})$ to be expressed as $p(\xi_{t+1}|\xi_t, \bar{z}_t, P_t)$ and $p(y_t|\xi_t, \bar{z}_t, P_t)$ respectively. The differences are in the backward smoothing recursions; the first method[Lindsten and Schön, 2011] samples the linear states during the smoothing step, the second instead fully marginalizes the model resulting in a non-Markovian smoothing problem[Lindsten et al., 2013]. The first method

thus effectively only uses marginalization during the filtering. The smoothing for the second method is accomplished using the concept of propagating information backwards that was demonstrated in section 2.1. This allows $p(\xi_{t+1:T}, y_{t+1:T}|\xi_{1:t}, y_{1:t})$ to be evaluated in constant time, for the details the reader is referred to the original paper[Lindsten et al., 2013].

3. Algorithm

This section shows how to combine MHIPS[Dubarry and Douc, 2011] with the type of marginalized models shown in section 2. Algorithm 2 summarizes the MHIPS algorithm for Markovian models. It is initialized with the ancestral trajectories, denoted $\tilde{x}_{1:T}$, from the forward particle filter, it then iterates over the trajectories from end to beginning R times, and for every time-step a new sample, x', is proposed from the proposal density $q(x'_t|\tilde{x}_{t+1}, y_t, \tilde{x}_{t-1})$. This new sample is accepted with probability given by

$$1 \wedge \frac{p_f(\tilde{x}_{t+1}|x'_t)}{p_f(\tilde{x}_{t+1}|\tilde{x}_t)} \frac{p_g(y_t|x'_t)}{p_g(y_t|\tilde{x}_t)} \frac{p_f(x'_t|\tilde{x}_{t-1})}{p_f(\tilde{x}_t|\tilde{x}_{t-1})} \frac{q(\tilde{x}_t|\tilde{x}_{t+1}, y_t, \tilde{x}_{t-1})}{q(x'_t|\tilde{x}_{t+1}, y_t, \tilde{x}_{t-1})}. \tag{10}$$

where $(a \wedge b)$ denotes the minimum value of a and b. When a sample is accepted it replaces the previous value in $\tilde{x}_{1:T}$ by setting $\tilde{x}_t = x'_t$. For the non-Markovian case the proposal and acceptance probabilities have to be extended in the same way as for the backward simulator type of smoother[Lindsten et al., 2014][Lindsten and Schön, 2013]. This results in the proposal density $q(x'_t|\tilde{x}_{t+1:T}, y_{1:T}, \tilde{x}_{1:t-1})$ and the acceptance probability is now given by

$$\begin{aligned} 1 \wedge & \frac{p_f(\tilde{x}_{t+1:T}, y_{t+1:T}|x'_t, \tilde{x}_{1:t-1}, y_{1:t})}{p_f(\tilde{x}_{t+1}, y_{t+1:T}|\tilde{x}_{1:t}, y_{1:t})} \\ & \times \frac{p_g(y_t|x'_t, \tilde{x}_{1:t-1}, y_{1:t-1})}{p_g(y_t|\tilde{x}_{1:t}, y_{1:t-1})} \\ & \times \frac{p_f(x'_t|\tilde{x}_{1:t-1}, y_{1:t-1})}{p_f(\tilde{x}_t|\tilde{x}_{1:t-1}, y_{1:t-1})} \frac{q(\tilde{x}_t|\tilde{x}_{t+1:T}, y_{1:T}, \tilde{x}_{1:t-1})}{q(x'_t|\tilde{x}_{t+1:T}, y_{1:T}, \tilde{x}_{1:t-1})}. \end{aligned} \tag{11}$$

At each time-step t, the MHIPS is choosing between two complete trajectories $x_{1:t}$, they are however identical except for the last state (x_t). Algorithm 3 gives the extended algorithm exemplified using the propagation of backwards variables as derived for the marginalized double integrator in (8), the concept is the same for other models where the necessary information can be back-propagated in a similar manner. For models where that is not possible it is of course possible to evaluate the required densities by iterating over the future parts of the trajectory, however the poor scaling properties of that approach makes it an unattractive method.

Algorithm 2 MHIPS

Sample $\{\tilde{x}_{1:T}^{(j)}\}_{j=1}^{M}$ from the ancestral paths of the forward filter, drawing each trajectory with probability $\omega_T^{(i)}$.
for $r = 1$ to R **do**
  for $t = T$ to 1 **do**
    for $j = 1$ to M **do**
      Sample $x'^{(j)}_t \sim q_t(x_t|\tilde{x}_{t+1}^{(j)}, y_t, \tilde{x}_{t-1}^{(j)})$
      With probability given by (10) set $\tilde{x}_t^{(j)} = x'^{(j)}_t$
    end for
  end for
end for

Algorithm 3 MHIPS non-Markov

Sample $\{\tilde{x}_{1:T}^{(j)}\}_{j=1}^{M}$ from the ancestral paths of the forward filter, drawing each trajectory with probability $\omega_T^{(i)}$.
for $r = 1$ to R **do**
  for $t = T$ to 1 **do**
    for $j = 1$ to M **do**
      Sample $x'^{(j)}_t \sim q_t(x_t|\tilde{x}_{t+1:T}^{(j)}, y_t, \tilde{x}_{1:t-1}^{(j)})$
      With probability given by (11) set $\tilde{x}_t^{(j)} = x'^{(j)}_t$
      Calculate backward propagating variables $(N_t^{(j)}, \gamma_t^{(j)})$
    end for
  end for
  for $t = 1$ to T **do**
    for $j = 1$ to M **do**
      Update forward propagating variables $(s_t^{(j)})$
    end for
  end for
end for

4. Results

All the experiments have been done using the proposal density $q(x'_t|\tilde{x}_{t+1:T}, y_{1:T}, \tilde{x}_{1:t-1}) = p(x'_t|y_{1:t-1}, \tilde{x}_{1:t-1})$ for the marginalized case and $q(x'_t|\tilde{x}_{t+1}, y_t, \tilde{x}_{t-1}) = p(x'_t|\tilde{x}_{t-1})$ for the Markovian case. The Python source code for the examples can be downloaded from [Nordh, 2015], all simulation have been done using the pyParticleEst[Nordh, 2013] software framework.

4.1 Double integrator

The methods are tested against an example with $Q = R = 1$ and initial state $x_1 = (-10 \quad 1)^{\mathrm{T}}$ and using 50 particles in the forward filter and 10 smoothed trajectories. Fig. 1 shows the RMSE for the double integrator example as a function of the number of iterations of the MHIPS algorithm. As expected there is no improvement for the degenerate case. For the marginalized case there is clear improvement using MHIPS and after only 9 iterations it yields better average RMSE than the FFBSi smoother. It would of course have been possible to initialize the MHIPS algorithm using the trajectories obtained from the FFBSi smoother instead of the ancestral paths from the particle filter, giving a lower initial RMSE, but to clearly demonstrate the initial rate of improvement of the MHIPS algorithm this was not done.

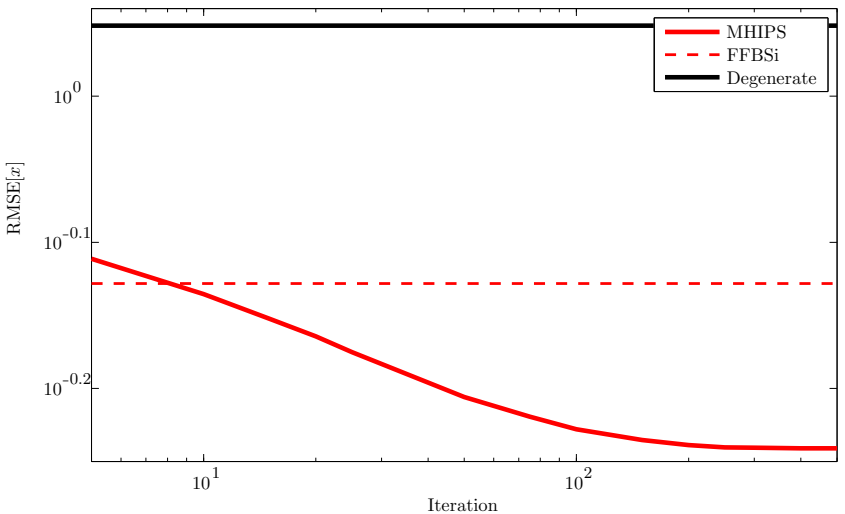

Figure 1. Average RMSE over 10000 realizations in logarithmic scale. The dotted black line is for the degenerate model, as expected it shows no improvement using MHIPS. The solid red line is for MHIPS on the marginalized model and dashed line when for using FFBSi

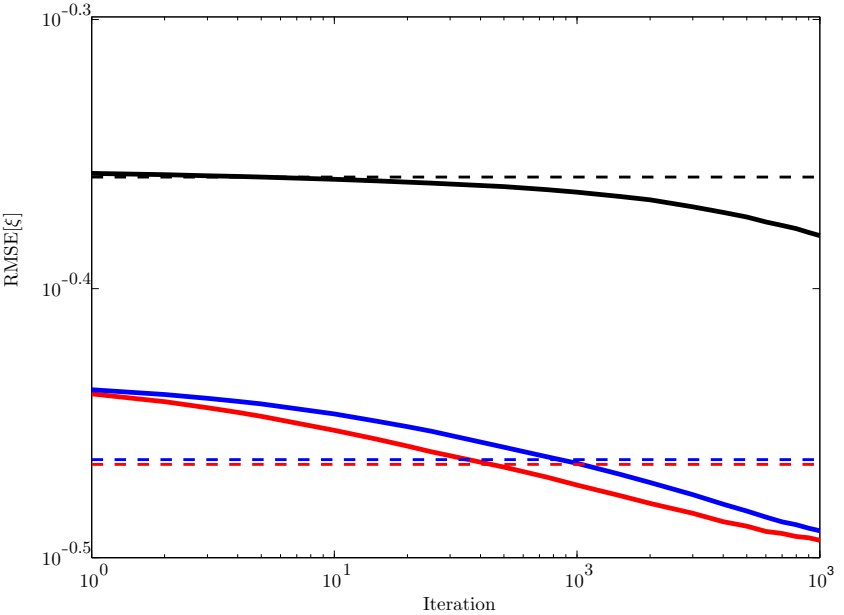

Figure 2. Average RMSE over 1500 realisation for ξ in logarithmic scale. The black (top) solid line is for the non-marginalized model, the blue (middle) solid line is for the model marginalized in the forward direction and the red (bottom) solid line is for the fully marginalized model. The dashed lines with corresponding colors (and order) are the RMSE obtained when using FFBSi.

4.2 MLNLG model

The methods are tested against an example with $Q_e = 0.005$, $Q_z = 0.01 I_{4\text{x}4}$, $R = 0.1$ and initial state $(\xi_1^{\mathrm{T}} \quad z_1^{\mathrm{T}})^{\mathrm{T}} = 0_{5\text{x}1}$ and using 100 particles in the forward filter and 10 smoothed trajectories. Fig. 2 shows the average RMSE of the ξ-state as a function of the number of iterations of the MHIPS algorithm, Fig. 3 shows the average RMSE of θ, the affine function of the z-states. Both figures include results for three different methods; the case when all the states are sampled, when the z-states are sampled during the smoothing step and finally when the z-states are fully marginalized for both the filter and smoother. The figures also include the performance of a FFBSi smoother for all three cases as a reference. It can be seen that marginalization as expected leads to a lower average RMSE for both FFBSi and MHIPS. It also shows a higher rate of improvement when using MHIPS and marginalization, requiring fewer iterations to beat the performance of FFBSi. Suggesting that proper marginalization is even more important for MHIPS than for FFBSi.

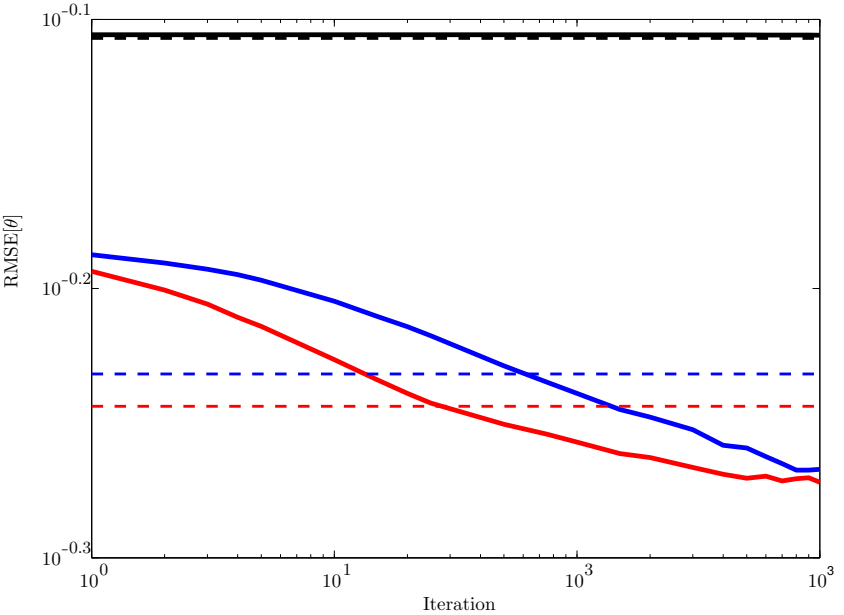

Figure 3. Average RMSE over 1500 realisation for θ in logarithmic scale. The black (top) solid line is for the non-marginalized model, the blue (middle) solid line is for the model marginalized in the forward direction and the red (bottom) solid line is for the fully marginalized model. The dashed lines with corresponding colors (and order) are the RMSE obtained when using FFBSi.

5. Conclusion

This article has demonstrated how to combine MHIPS with marginalized models and combines it with the method for fully marginalizing the linear sub-states in a MLNLG model using the approach presented in [Lindsten et al., 2013]. It compares the marginalized MHIPS with the regular MHIPS, showing the importance of marginalization with MHIPS since it enables MHIPS to outperform the FFBSI smoother using fewer iterations.

Acknowledgements

The author is a member of the LCCC Linnaeus Center and the eLLIIT Excellence Center at Lund University. The author would like to thank professor Bo Bernhardsson, Lund University, for the feedback provided on this work.

References

Beskos, A., D. Crisan, A. Jasra, et al. (2014). "On the stability of sequential Monte Carlo methods in high dimensions". *The Annals of Applied Probability* **24**:4, pp. 1396–1445.

Briers, M., A. Doucet, and S. Maskell (2010). "Smoothing algorithms for state-space models". English. *Annals of the Institute of Statistical Mathematics* **62**:1, pp. 61–89. ISSN: 0020-3157. DOI: 10.1007/s10463-009-0236-2. URL: http://dx.doi.org/10.1007/s10463-009-0236-2.

Bunch, P. and S. Godsill (2013). "Improved particle approximations to the joint smoothing distribution using Markov Chain Monte Carlo". *Signal Processing, IEEE Transactions on* **61**:4, pp. 956–963.

Doucet, A., S. Godsill, and C. Andrieu (2000). "On sequential Monte Carlo sampling methods for Bayesian filtering". English. *Statistics and Computing* **10**:3, pp. 197–208. ISSN: 0960-3174. DOI: 10.1023/A:1008935410038. URL: http://dx.doi.org/10.1023/A:1008935410038.

Dubarry, C. and R. Douc (2011). "Particle approximation improvement of the joint smoothing distribution with on-the-fly variance estimation". *arXiv preprint arXiv:1107.5524*.

Julier, S. and J. Uhlmann (2004). "Unscented filtering and nonlinear estimation". *Proceedings of the IEEE* **92**:3, pp. 401–422. ISSN: 0018-9219. DOI: 10.1109/JPROC.2003.823141.

Lindsten, F. and T. Schön (2011). *Rao-Blackwellized Particle Smoothers for Mixed Linear/Nonlinear State-Space Models*. Tech. rep. URL: http://user.it.uu.se/~thosc112/pubpdf/lindstens2011.pdf.

Lindsten, F., P. Bunch, S. J. Godsill, and T. B. Schön (2013). "Rao-Blackwellized particle smoothers for mixed linear/nonlinear state-space models". In: *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, pp. 6288–6292.

Lindsten, F., M. I. Jordan, and T. B. Schön (2014). "Particle Gibbs with ancestor sampling". *The Journal of Machine Learning Research* **15**:1, pp. 2145–2184.

Lindsten, F. and T. Schön (2013). "Backward simulation methods for Monte Carlo statistical inference". *Foundations and Trends in Machine Learning* **6**:1, pp. 1–143. ISSN: 1935-8237. DOI: 10.1561/2200000045. URL: http://dx.doi.org/10.1561/2200000045.

Montemerlo, M., S. Thrun, D. Koller, B. Wegbreit, et al. (2002). "FastSLAM: a factored solution to the simultaneous localization and mapping problem". In: *AAAI/IAAI*, pp. 593–598.

Nordh, J. (2013). *PyParticleEst - Python library for particle based estimation methods*. URL: `http://www.control.lth.se/Staff/JerkerNordh/pyparticleest.html`.

Nordh, J. (2015). *Source code*. URL: `http://www.control.lth.se/Staff/JerkerNordh/mhips-marginalized.html`.

Okuma, K., A. Taleghani, N. De Freitas, J. J. Little, and D. G. Lowe (2004). "A boosted particle filter: multitarget detection and tracking". In: *Computer Vision-ECCV 2004*. Springer-Verlag, pp. 28–39.

Rebeschini, P. and R. van Handel (2013). "Can local particle filters beat the curse of dimensionality?" *arXiv preprint arXiv:1301.6585*.

Schön, T., F. Gustafsson, and P.-J. Nordlund (2005). "Marginalized particle filters for mixed linear/nonlinear state-space models". *Signal Processing, IEEE Transactions on* **53**:7, pp. 2279–2289. ISSN: 1053-587X. DOI: `10.1109/TSP.2005.849151`.

Welch, G. and G. Bishop (1995). *An introduction to the Kalman filter*.

Paper VI

Rao-Blackwellized Auxiliary Particle Filters for Mixed Linear/Nonlinear Gaussian models

Jerker Nordh

Abstract

The Auxiliary Particle Filter is a variant of the common particle filter which attempts to incorporate information from the next measurement to improve the proposal distribution in the update step. This paper studies how this can be done for Mixed Linear/Nonlinear Gaussian models, it builds on a previously suggested method and introduces two new variants which tries to improve the performance by using a more detailed approximation of the true probability density function when evaluating the so called first stage weights. These algorithms are compared for a couple of models to illustrate their strengths and weaknesses.

1. Introduction

The particle filter (PF) [Gordon et al., 1993][Doucet and Johansen, 2009][Arulampalam et al., 2002] has become one of the standard algorithms for nonlinear estimation and has proved its usefulness in a wide variety of applications. It is an application of the general concept of Sequential Importance Resampling (SIR)[Doucet et al., 2000]. At each time the true probability density function is approximated by a number of point estimates, the so called particles. These are propagated forward in time using the system dynamics. For each measurement the weight of each particle is updated with the corresponding likelihood of the measurement. The last step is to resample the particles, this occurs either deterministically or only when some criteria is fulfilled. This criteria is typically the so called number of effective particles, which is a measure of how evenly the weights are distributed among the particles. The resampling is a process that creates a new set particles where all particles have equal weights, which is accomplished by drawing them with probability corresponding to their weight in the original set. This process is needed since otherwise eventually all the weights except one would go to zero. The resampling thus improves the estimate by focusing the particles to regions with higher probability.

The Auxiliary Particle Filter (APF)[Pitt and Shephard, 1999][Pitt and Shephard, 2001][Johansen and Doucet, 2008] attempts to improve this by resampling the particles at time t using the predicted likelihood of the measurement at time $t+1$. If done properly this helps to focus the particles to areas where the measurement has a high likelihood. The problem is that it requires the evaluation of the probability density function $p(y_{t+1}|x_t)$. This is typically not available and is therefore often approximated by assuming that next state is the predicted mean state, i.e. $x_{t+1} = \bar{x}_{t+1|t}$ and the needed likelihood instead becomes $p(y_{t+1}|x_{t+1} = \bar{x}_{t+1|t})$.

This paper focuses on the special case of Mixed Linear/Nonlinear Gaussian (MLNLG) models, which is a special case of Rao-Blackwellied models. For an introduction to the Rao-Blackwellized Particle Filter see [Schön et al., 2005]. Rao-Blackwellized models have the property that conditioned on the nonlinear states there exists a linear Gaussian substructure that can be optimally estimated using a Kalman filter. This reduces the dimensionality of the model that the particle filter should solve, thereby reducing the complexity of the estimation problem. The general form for MLNLG models is shown in (1), where the state vector has been split in two parts, $x = (\xi \quad z)^T$. Here ξ are the nonlinear states that are estimated by the particle filter, z are the states that conditioned on the trajectory $\xi_{1:t}$ are estimated using a Kalman filter.

$$\xi_{t+1} = f_\xi(\xi_t) + A_\xi(\xi_t)z_t + v_\xi \tag{1a}$$
$$z_{t+1} = f_z(\xi_t) + A_z(\xi_t)z_t + v_z \tag{1b}$$
$$y_{t+1} = g(\xi_{t+1}) + C(\xi_{t+1})z_{t+1} + e \tag{1c}$$
$$\begin{pmatrix} v_\xi \\ v_z \end{pmatrix} \sim N\left(0, \begin{pmatrix} Q_\xi(\xi_t) & Q_{\xi z}(\xi_t) \\ Q_{z\xi}(\xi_t) & Q_z(\xi_t) \end{pmatrix}\right) \tag{1d}$$
$$e \sim N(0, R(\xi_t)) \tag{1e}$$

In [Fritsche et al., 2009] an approximation is presented that can be used with the APF for this type of models, section 2 presents that algorithm and two variants proposed by the author of this paper. Section 3 compares the different algorithms by applying them to a number of examples to highlight their strengths and weaknesses. Finally section 4 concludes the paper with some discussion of the trade-offs when choosing one of theses algorithms.

2. Algorithms

2.1 Auxiliary Particle Filter introduction

$$x_{t+1} = f(x_t, v_t) \tag{2a}$$
$$y_t = h(x_t, e_t) \tag{2b}$$

Looking at a generic state-space model of the form in (2) and assuming we have a collection of weighted point estimates (particles) approximating the probability density function of $x_{t|t} \approx \sum_{i=1}^{N} w^{(i)}\delta(x_t - x_t^{(i)})$ the standard particle filter can be summarized in the following steps that are done for each time instant t.

1. (Resample; draw N new samples $\tilde{x}_t^{(i)}$ from the categorical distribution over $x_t^{(i)}$ with probabilities proportional to $w_t^{(i)}$. Set $\tilde{w}_t^{(i)} = \frac{1}{N}$.)
2. For all i; sample $v_t^{(i)}$ from the noise distribution
3. For all i; calculate $x_{t+1|t}^{(i)} = f(\tilde{x}_t^{(i)}, v_t^{(i)})$
4. For all i; calculate $w_{t+1}^{(i)} = \tilde{w}_t^{(i)} p(y_t | x_{t+1|t}^{(i)})$

The resampling step introduces variance in the estimate, but is necessary for the long term convergence of the filter. Step 1 is therefore typically not done at each time instant but only when a prespecified criteria on the weights are fulfilled.

The Auxiliary Particle Filter introduces an additional step to incorporate knowledge of the future measurement y_{t+1} before updating the point estimates $x_t^{(i)}$.

1. For all i; calculate $\tilde{w}_t^{(i)} = l^{(i)} w_t^{(i)}$, where $l^{(i)} = p(y_{t+1}|x_t^{(i)})$
2. (Resample; draw N new samples $\tilde{x}_t^{(i)}$ from the categorical distribution over $x_t^{(i)}$ with probabilities proportional to $l^{(i)} w_t^{(i)}$. Set $\tilde{w}_t = \frac{1}{N}$.)
3. For all i; sample from $v_t^{(i)}$ from the noise distribution
4. For all i; calculate $x_{t+1|t}^{(i)} = f(\tilde{x}_t^{(i)}, v_t^{(i)})$
5. For all i; calculate $w_{t+1}^{(i)} = \frac{\tilde{w}_t^{(i)}}{l^{(i)}} p(y_t|x_{t+1|t}^{(i)})$

The difference between the algorithms in the rest of section 2 is in how the first stage weights, $l^{(i)}$, are approximated for the specific class of model (1).

2.2 Algorithm 1

The first algorithm considered is the one presented in [Fritsche et al., 2009]. It computes $p(y_{t+1}|x_t)$ using the approximate model shown in (3). This approximation ignores the uncertainty in the measurement that is introduced through the uncertainty in ξ_{t+1}, thereby underestimating the total uncertainty in the measurement. Tbjs could lead to resampling of the particles even when the true particle weights would not indicate the need for resampling. Here P_{z_t} denotes the estimated covariance for the variable z_t. In (3e) the dependence on ξ_t has been suppressed to not clutter the notation.

$$p(y_{t+1}|x_t) \sim N(\bar{y}_{t+1}, P_{y_{t+1}}) \tag{3a}$$
$$\bar{\xi}_{t+1} = f_\xi(\xi_t) + A_\xi(\xi_t)\bar{z}_t \tag{3b}$$
$$\bar{z}_{t+1} = f_z(\xi_t) + A_z(\xi_t)\bar{z}_t \tag{3c}$$
$$\bar{y}_{t+1} = g(\bar{\xi}_{t+1}) + C(\bar{\xi}_{t+1})\bar{z}_{t+1} \tag{3d}$$
$$P_{y_{t+1}} \approx C(A_z P_{z_t} A_z^T + Q)C^T + R \tag{3e}$$

2.3 Algorithm 2

Our first proposed improvement to the algorithm in section 2.2 is to attempt to incorporate the uncertainty of ξ_{t+1} in our measurement by linearizing the measurement equation around the predicted mean. For the case when C

and R has no dependence on ξ the pdf of the measurement can then be approximated in the same way, except for $P_{y_{t+1}}$ which instead is approximated according to (4), here J_g is Jacobian of g. The difference compared to model 1 lies in the additional terms for the covariance $P_{y_{t+1}}$ due propagating the uncertainty of ξ_{t+1} by using the Jacobian of g.

$$\begin{aligned} P_{y_{t+1}} \approx & (J_g A_\xi + C A_z) P_{z_t} (J_g A_\xi + C A_z)^T \\ & + (J_g + C) Q (J_q + C)^T + R \end{aligned} \tag{4}$$

2.4 Algorithm 3

If the linearization scheme presented in Section 2.3 is to work well g must be close to linear in the region where ξ_{t+1} has high probability. Another commonly used approximation when working with Gaussian distributions and nonlinearities is the Unscented Transform used in e.g. the Unscented Kalman Filter[Julier and Uhlmann, 2004]. The second proposed algorithms uses the UT to approximate the likelihood of the measurement. It propagates a set of so called Sigma-point through both the time update (2a) and the measurement equation (2b) and estimates the resulting mean and covariance of the points after the transformations. This is then used as the approximation for $\bar{y}_{t+1}$ and $P_{y_{t+1}}$. This avoids the need to linearize the model, and can thus handle discontinuities and other situations where linearization cannot be used or gives a poor approximation.

The Sigma-points, $x_\sigma^{(i)}$, were chosen according to (5), using twice the number of points as the combined dimension of the state and noise vectors (N_x). Σ_x is the combined covariance matrix. $(\sqrt{N_x \Sigma_x})_i$ is the i-th column of R, where $N_x \Sigma_x = RR^T$. All the points were equally weighted. This conforms to the method introduced in section III-A in [Julier and Uhlmann, 2004].

$$\Sigma_x = \mathrm{diag}(P_{z_t},\ Q,\ R) \tag{5a}$$

$$W^{(i)} = (2N_x)^{-1}, \quad i \in 1..2N_x \tag{5b}$$

$$x_\sigma^{(i)} = \bar{x} + (\sqrt{N_x \Sigma_x})_i, \quad i \in 1..N_x \tag{5c}$$

$$x_\sigma^{(i+N_x)} = \bar{x} - (\sqrt{N_x \Sigma_x})_i, \quad i \in 1..N_x \tag{5d}$$

3. Results

To study the difference in performance of the three algorithms we will look at two models, the first one was introcuded in [Lindsten and Schön, 2011] and is an extension of a popular nonlinear model to include a linear Gaussian substructure. The second model was designed to highlight problems with the approximation used in Algorithm 1.

3.1 Model 1

$$\xi_{t+1} = \frac{\xi_t}{1+\xi_t^2} \begin{pmatrix} 0 & 0.04 & 0.044 & 0.008 \end{pmatrix} z_t + \\ + 0.5\xi_t + 25\frac{\xi_t}{1+\xi_t^2} + 8\cos 1.2t + v_{\xi,t} \tag{6a}$$

$$z_{t+1} = \begin{pmatrix} 3 & -1.691 & 0.849 & -0.3201 \\ 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \end{pmatrix} z_t + v_{z,t} \tag{6b}$$

$$y_t = 0.05\xi_t^2 + e_t \tag{6c}$$

$$\xi_0 = 0, \quad z_0 = \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix}^T \tag{6d}$$

$$R = 0.1, \quad Q_\xi = 0.005 \tag{6e}$$

$$Q_z = \mathrm{diag}(0.01 \quad 0.01 \quad 0.01 \quad 0.01) \tag{6f}$$

Notice the square in the measurement equation in (6c), depending on the magnitude of uncertainty in ξ_t a linearizion of this term might lead to a poor approximation. So we might expect that for this model the unscented transform based approximation might fare better.

The performance of the algorithms was tested by generating 25000 dataset from (6), all the algorithms were then tested on this collection for a number of different particle counts. The average RMSE values are presented in table 1, they are also shown in Fig. 1. The relative RMSE of the algorithms compared to the standard particle filter is shown in Table 2. As can be seen both algorithm 2 and 3 outperform Algorithm 1, most of the time Algorithm 3 also beats Algorithm 2. This is expected since they use a more detailed approximation of the true density of $p(y_{t+1}|x_t)$.

Table 1. Average RMSE for ξ over 25000 realizations of model 1. As can be seen the improved approximations in Algorithm 2 and 3 lead to slightly better performance, but for this model the difference in performance of the algorithms is small

N	PF	Alg. 1	Alg. 2	Alg. 3
10	1.701	1.611	1.607	1.600
15	1.395	1.338	1.322	1.322
20	1.234	1.170	1.153	1.154
25	1.121	1.061	1.060	1.052
30	1.049	0.992	0.978	0.978
40	0.955	0.893	0.885	0.880
50	0.874	0.831	0.824	0.819
75	0.782	0.737	0.732	0.730
100	0.720	0.689	0.686	0.687

Table 2. RMSE compared to the RMSE for the regular particle filter, for ξ over 25000 realizations of model 1. As can be seen the improved approximations in Algorithm 2 and 3 lead to slightly better performance, but for this model the difference in performance of the algorithms is small

N	Alg. 1	Alg. 2	Alg. 3
10	94.7%	94.5%	94.1%
15	96.0%	94.8%	94.7%
20	94.8%	93.4%	93.5%
25	94.6%	94.5%	93.8%
30	94.5%	93.2%	93.2%
40	93.5%	92.7%	92.1%
50	95.0%	94.2%	93.7%
75	94.2%	93.5%	93.3%
100	95.7%	95.2%	95.4%

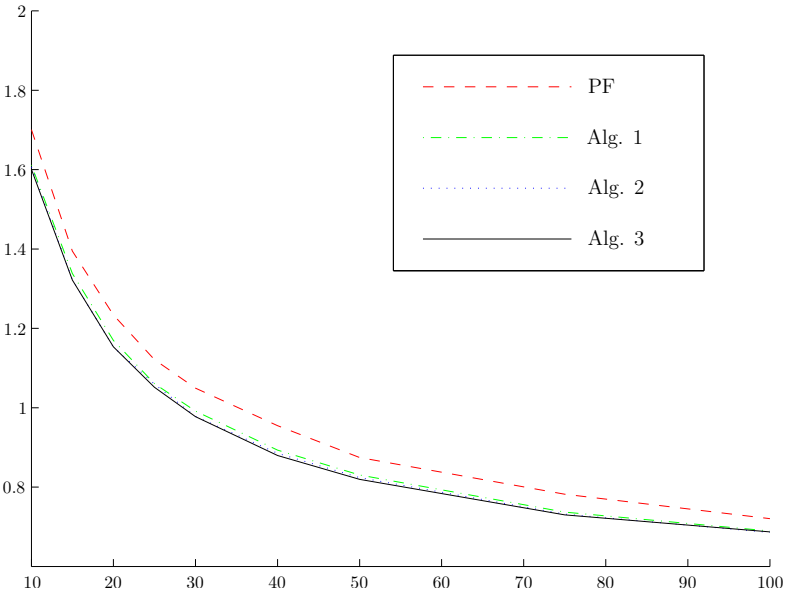

Figure 1. Average RMSE as a function of the number of particles when estimating model 1 with the three algorithms in this paper. The particle filter is included as a reference. Average over 25000 realizations. Data-points for $10, 15, 20, 25, 30, 40, 50, 75$ and 100 particles

3.2 Model 2

$$\xi_{t+1} = 0.8\xi_t + a_t z_t + v_\xi \tag{7a}$$
$$z_{t+1} = 0.8 z_t + v_z \tag{7b}$$
$$y_t = \xi_t^2 + a_t z_t + e_t \tag{7c}$$
$$v_\xi \sim N(0, 0.5a_t + 0.1(1 - a_t)) \tag{7d}$$
$$v_z \sim N(0, 0.1) \tag{7e}$$
$$R \sim N(0, 0.1) \tag{7f}$$
$$a_t = t \bmod 2$$

The model in (7) was constructed to exploit the weakness of neglecting the uncertainty introduced in the measurement due to the uncertainty of ξ_{t+1}. In order for this to affect the estimate it is also important that the trajectory leading up to the current estimate is of importance. Since z_t depends on the full trajectory $\xi_{1:t}$ it captures this aspect. The estimate of z for different particles will also influence the future trajectory, this implies that even if two particles at time t are close to the same value of ξ the path they followed earlier will affect their future trajectories. This means that a premature resampling of the particles due to underestimating the uncertainty in the measurement can lead to errors in the following parts of the dataset.

To clearly demonstrate this issue a time-varying model was chosen were the z-state only affects every second measurement and every second state update. It turns out that this model is difficult to estimate, and when using few particles the estimates sometimes diverge with the estimated states approaching infinity. To not overshadow the RMSE of all the realization were the filters perform satisfactorily they are excluded, the number of excluded realizations are presented in Table 3.

The model was also modified by moving the pole corresponding to the ξ-state to 0.85 and the one for the z-state to 0.9, this makes the estimation problem more difficult. The corresponding number of diverged estimates are shown in Table 4. It can seen that even though all three algorithms are more robust than the standard particle filter, Algorithm 1 clearly outperforms the rest when it comes to robustness.

The RMSE of the algorithms are shown in Fig. 2 for model (7) and Fig. 3 for the case with the poles moved to 0.85 and 0.9. Surprisingly the RMSE of Algorithm 1 increases as the number of particles increases. The author believes this is an effect of the particles being resampled using a too coarse approximation, thus introducing a modeling error. When the number of particles is increased the estimate of a particle filter normally converges to the true poster probability density, but in this case the estimate is converging to the true posterior for the incorrect model, leading to an incorrect estimate.

Table 3. The number of diverged estimates for 1000 realizations of model 2. A diverged realization is defined as one where the RMSE exceeded 10000.

N	PF	Alg. 1	Alg. 2	Alg. 3
25	3	0	0	2
50	2	0	0	0
75	1	0	0	0
100	0	0	0	0
125	0	0	0	0
150	0	0	0	0
200	0	0	0	0

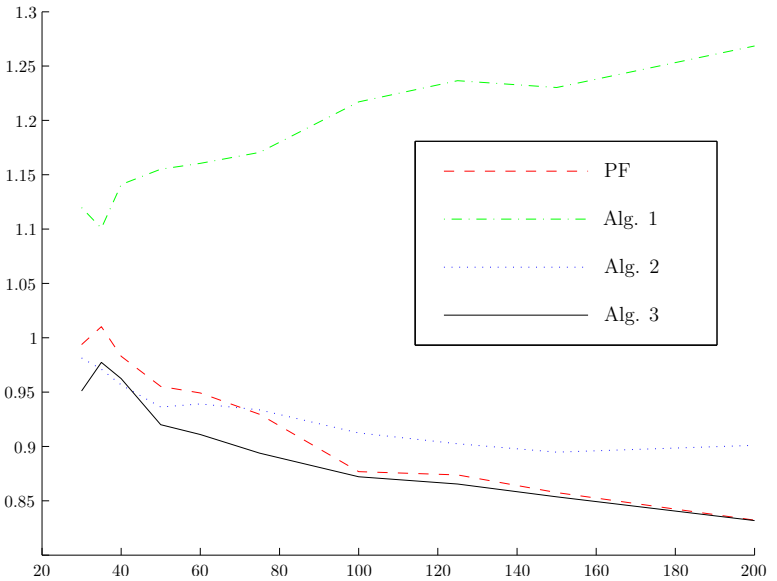

Figure 2. Average RMSE as a function of the number of particles when estimating model 2 with the three algorithms in this paper. Data-points for $30, 35, 40, 50, 60, 75, 100, 125, 150$ and 200 particles. The particle filter is included as a reference. Average over 1000 realizations, excluding those that diverged (RMSE $\geq$ 10000). Both algorithm 2 and 3 beat the PF for low particle counts, but only the algorithm 3 keeps up when the number of particles increases. Surprisingly algorithm 1's performance degrades as the number of particles increases.

Table 4. The number of diverged estimates for 5000 realizations of model 2 with the pole for ξ moved to 0.85 and the pole for z in 0.9 instead. A diverged realization is defined as one where the RMSE exceeded 10000.

N	PF	Alg. 1	Alg. 2	Alg. 3
50	124	1	40	35
75	62	0	12	13
100	45	0	7	8
125	22	0	5	4
150	17	0	4	2
200	9	0	1	2
300	1	0	0	0

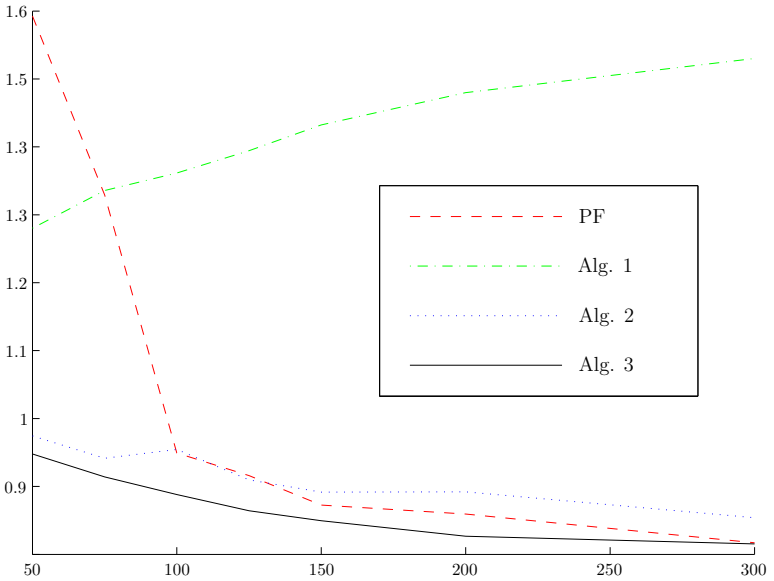

Figure 3. Average RMSE for the three algorithms as a function of the number of particles when estimating the modified model 2 (poles in 0.85 and 0.9 instead). Data-points for $50, 75, 100, 125, 150, 200$ and 300 particles. The particle filter is included as a reference. Average over 5000 realizations, except for the PF where 25000 realizations were used for $50, 75, 75, 100$ and 125 particles due to the large variance of those estimates. The RMSE was calculated excluding those realizations that diverged (RMSE $\geq$ 10000). All filters beat the PF for low particle counts, but only algorithm 3 keeps up when the number of particles increases. Surprisingly the performance of algorithm 1 degrades as the number of particles increases.

4. Conclusion

The three algorithms compared in this paper are all variants of the general Auxiliary Particle Filter algorithm, but by choosing different approximations when evaluating the first stage weights (l) they have different performance characteristics. eps As was especially evident when looking at model 2 all three auxiliary particle filters were more robust than the ordinary particle filter, particularly when using fewer particles. Algorithm 1 is especially noteworthy since it almost did not suffer at all from the problem with divergence that was noted for the other approximations when estimating model 2. However it does seem to have problems approximating the true posterior, as evident by the high RMSE that also was unexpectedly increasing when the number of particles were increased.

Algorithm 2 and 3 both beat the standard particle filter both in terms of average RMSE and robustness. Algorithm 3 has the best performance, but also requires the most computations due to the use of an Unscented Transform approximation for each particle and measurement.

In the end the choice of which algorithm to use has to be decided on a case by case basis depending on relative computational complexity for the different approximations for that particular model, so this article can not present any definitive advice for this choice. The linearization approach doesn't increase the computational effort nearly as much as using the Unscented Transform, but it doesn't always capture the true distribution with the needed accuracy. However, for models where the linearization works well this is likely the preferred approximation due to the low increase in computational effort needed.

When it is possible to increase the number of particles it could very well be most beneficial to simply use the standard particle filter with a higher number of particles, thus completely avoiding the need to evaluate $p(y_{t+1}|x_t)$. As always this is influenced by the specific model and how the uncertainty in the update step compares to the uncertainty in the measurement.

Acknowledgments

The author would like to thank professor Bo Bernhardsson, Lund University, for the feedback provided on the work presented in this article.

The author is a member of the LCCC Linnaeus Center and the eLLIIT Excellence Center at Lund University.

All simulations have been done using the pyParticleEst framework[Nordh, 2013].

References

Arulampalam, M., S. Maskell, N. Gordon, and T. Clapp (2002). "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking". *IEEE Trans. Signal Process.* **50**:2, pp. 174–188. ISSN: 1053-587X.

Doucet, A., S. Godsill, and C. Andrieu (2000). "On sequential Monte Carlo sampling methods for Bayesian filtering". English. *Statistics and Computing* **10**:3, pp. 197–208. ISSN: 0960-3174. DOI: 10.1023/A:1008935410038. URL: http://dx.doi.org/10.1023/A:1008935410038.

Doucet, A. and A. M. Johansen (2009). "A tutorial on particle filtering and smoothing: fifteen years later". *Handbook of Nonlinear Filtering* **12**, pp. 656–704.

Fritsche, C., T. Schon, and A. Klein (2009). "The marginalized auxiliary particle filter". In: *Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP), 2009 3rd IEEE International Workshop on*. IEEE, pp. 289–292.

Gordon, N., D. Salmond, and A. F. M. Smith (1993). "Novel approach to nonlinear/non-Gaussian Bayesian state estimation". *Radar and Signal Processing, IEE Proceedings F* **140**:2, pp. 107–113. ISSN: 0956-375X.

Johansen, A. M. and A. Doucet (2008). "A note on auxiliary particle filters". *Statistics & Probability Letters* **78**:12, pp. 1498–1504.

Julier, S. and J. Uhlmann (2004). "Unscented filtering and nonlinear estimation". *Proceedings of the IEEE* **92**:3, pp. 401–422. ISSN: 0018-9219. DOI: 10.1109/JPROC.2003.823141.

Lindsten, F. and T. Schön (2011). *Rao-Blackwellized Particle Smoothers for Mixed Linear/Nonlinear State-Space Models*. Tech. rep. URL: http://user.it.uu.se/~thosc112/pubpdf/lindstens2011.pdf.

Nordh, J. (2013). *PyParticleEst - Python library for particle based estimation methods*. URL: http://www.control.lth.se/Staff/JerkerNordh/pyparticleest.html.

Pitt, M. K. and N. Shephard (1999). "Filtering via simulation: auxiliary particle filters". *Journal of the American statistical association* **94**:446, pp. 590–599.

Pitt, M. K. and N. Shephard (2001). "Auxiliary variable based particle filters". In: *Sequential Monte Carlo methods in practice*. Springer, pp. 273–293.

Schön, T., F. Gustafsson, and P.-J. Nordlund (2005). "Marginalized particle filters for mixed linear/nonlinear state-space models". *Signal Processing, IEEE Transactions on* **53**:7, pp. 2279–2289. ISSN: 1053-587X. DOI: 10.1109/TSP.2005.849151.

Paper VII

Extending the Occupancy Grid Concept for Low-Cost Sensor-Based SLAM

Jerker Nordh Karl Berntorp

Abstract

The simultaneous localization and mapping problem is approached by using an ultrasound sensor and wheel encoders. To account for the low precision inherent in ultrasound sensors, the occupancy grid notion is extended. The extension takes into consideration with which angle the sensor is pointing, to compensate for the issue that an object is not necessarily detectable from all positions due to deficiencies in how ultrasonic range sensors work. A mixed linear/nonlinear model is derived for future use in Rao-Blackwellized particle smoothing.

1. Introduction

The problem of having a robot simultaneously localize itself and learn its map is commonly referred to as simultaneous localization and mapping (SLAM), and is still considered a challenging problem. The problem is often solved using odometry readings in combination with vision or range sensors. In mobile robotics it has been studied extensively over the last three decades. For surveys and tutorials of the SLAM problem and its different solutions up to recently, see for example [Thrun, 2002] or [Durrant-Whyte and Bailey, 2006].

At least since the early 1990's the approach to SLAM has been probabilistic, and one of the earliest works on this was presented in [Smith et al., 1990], where extended Kalman filtering (EKF) was used for state estimation. One of the problems with using Kalman filtering is that the nonlinearities that typically are present tend to lead to divergence of the state estimates. For example, the kinematics of a planar robot is typically nonlinear in the heading angle, and the consequent linearizations that the EKF uses for estimating the odometry may lead to instability. To remedy this, particle filtering was introduced as a means to solve the SLAM problem; the reader is referred to [Grisetti et al., 2005] and [Grisetti et al., 2007] for state of the art algorithms.

Several approaches exist of how to represent the map, where two possible approaches are metric and topological. In the metric approaches, which this paper will focus on, the maps capture the geometric properties of the environment, while the topological maps try to describe the connectivity of different places using graphs, see [Thrun, 1998]. Perhaps the most popular representative of the metric approaches is known as occupancy grid mapping. In this representation the space is described by evenly spaced grids, see [Siciliano and Khatib, 2008] for an introduction. The grids are considered to be either occupied or free, with some probability distribution associated with the grid. A possible usage of occupancy grid maps is when utilizing range sensors, such as laser sensors or sonar sensors. Both types of sensors have noise and may occasionally give severe measurement errors. Since laser sensors have very high spatial resolution, thus giving a sharp probability distribution, they appear to be the most common solution, see [Hähnel et al., 2003], [Eliazar and Parr, 2003], and [Grisetti et al., 2007] for some examples. In contrast, sonar sensors have the problem of covering a cone in space, which typically makes it impossible to determine from a single measurement whether a certain cell is occupied or not because of the low spatial resolution in the tangential direction. Also, ultrasound sensors are very sensitive to the angle of an object's surface relative to the sensor. This leads to the problem that measuring the same surface from slightly different angles may render different results. Obviously, this could potentially lead to estimation errors. See Fig. 1 and Section 2 for a more detailed description of the problems

encountered with ultrasound based range sensors.

In this paper the SLAM problem is approached using only wheel encoder readings and one ultrasonic sensor. To get rid of, or at least attenuate, the problems inherent in ultrasonic sensors described earlier, a new approach to grid mapping is developed. This should be seen as an extension to the notion of occupancy grids described in [Siciliano and Khatib, 2008], in the sense that the angle with which the sensor is facing the cell is now taken into account. Particle filtering is used for position estimation, and each particle represents a possible robot position and a map. It is known that using particle filters for SLAM tends to destroy the map over time caused by sample depletion, see [Kwak et al., 2007] for an investigation. However, an idea is that particle smoothing could be a way to get rid of this problem. To prepare for exploiting the ideas of Rao-Blackwellized particle smoothing established in [Lindsten and Schön, 2011], a mixed linear/nonlinear state-space model is developed. Compared to the regular occupancy grid this formulation also represents the variance of each probability estimate.

Using sonar sensors for SLAM has been studied before; an example is [Burgard et al., 1999], in which an offline expectation maximization algorithm was used for occupancy grid mapping using 24 Polaroid sensors with 15 degrees opening angle. An early work is [Rencken, 1993], where the SLAM problem was solved in simulation using 24 ultrasonic sensors by estimating the errors introduced in the localization and mapping parts, respectively, and correcting for them using a modified Kalman filter approach. A third example is [Leonard et al., 1992] in which the SLAM problem was solved using a feature based approach with the aid of servo-mounted ultrasonic sensors. A more recent work is [Ahn et al., 2008], where ultrasonic sensors and a stereo camera is used in an EKF-SLAM setting.

The paper is organized as follows: The SLAM problem is introduced in Section 2, where the difficulties with ultrasonic sensors is explained in more detail. Section 3 details the scope of the work presented in this article. In Section 4 the kinematics, sensor, map modeling, as well as the SLAM algorithm are introduced. Implementational aspects are discussed in Section 5. The validation results are shown in Section 6. Finally, the paper is concluded in Section 7.

2. Problem Formulation

As previously mentioned there are two major systematic issues with ultrasonic sensor that have been observed:

1. Large field of view; making it uncertain from which point a measurement originated. This leads to a fundamental limitation on the resolution of the sensor, which varies with the distance to the object. Because

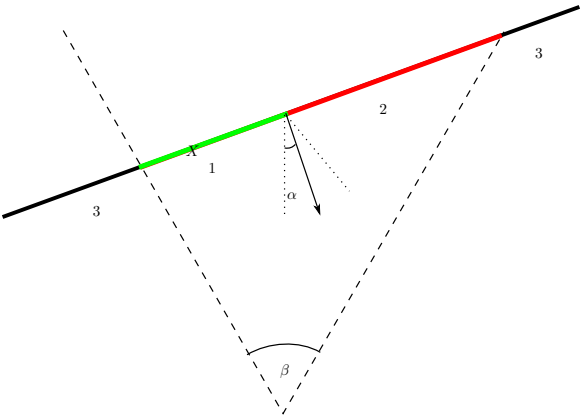

Figure 1. Properties of ultrasonic sensors. The sensor has an opening angle, β. The sound only reflect back to the sensor if the angle of incidence is less than α. The figure illustrates a straight wall. Sections marked 3 are outside the field of view, the section marked 2 is inside the field of view but the angle of incidence is so large that it is not visible to the sensor. The section marked 1 is inside the field of view. Also, the angle of incidence is steep enough for it to be detected by the sensor. The range returned by the sensor will be the distance to the closest point within section 1, marked with an X.

of the typically large opening angle of ultrasonic sensors the resolution is of the same order as the distance to the detected object. That is, when detecting an object at a range of 1 meter, the spatial resolution of the sensor is roughly 1 meter in the tangential direction.

2. Angle of incidence; for angles above a certain threshold the object becomes more or less invisible for the sensor. Thus an object can only be detected from certain directions. This problem becomes apparent when the sensor is close to a wall and measuring along it. Typically the wall will be inside the field of view of the sensor, but the sensor will not detect it because of a too narrow angle of incidence. If the same wall is then measured from another position within the room where the sensor faces it perpendicularly, it will be detected. Note that the angle is dependent on the material of the observed object.

See Fig. 1 for an illustration.

The basic premise for the SLAM problem is that it is possible to reobserve parts of the environment, thus relating the current position to those before. Therefore if it is not possible to observe objects that have been detected previously, it will lead to inconsistencies in the map as well as inaccuracy in

the position estimate. This article aims to provide a method for dealing with these systematic errors.

3. Article Scope

The method presented in this paper extends the concept of occupancy grids to take the angle of incidence into consideration and partitioning each cell of the map into several parts, where each part is visible only from certain regions within the environment. This reduces the problem of conflicting measurements. The drawback is that it also reduces the correlation between measurements, and thus the underlying SLAM algorithm will require more data to converge.

3.1 Computational Complexity

Particle methods are sensitive to the number of states in the model as the number of particles needed to represent the probability density function explodes with the number of states. Therefore a conditionally linear model is very beneficial for reducing the computational burden. The method presented in this paper is conditionally linear given the position and orientation of the sensor, thus for the planar case only 3 nonlinear states are needed. The number of linear states depends on the size and resolution of the map.

3.2 Evaluation

To evaluate the model, simulated data corresponding to different sensors characteristics were generated. The data sets were used with different levels of subdivision of the grid cells, showing the methods strengths and weaknesses for a selection of sensor characteristics. The focus is on investigating how the angle-of-incidence limitations on the sensor affect the position estimate of the SLAM algorithm. The goal is position estimation, the map is merely a tool. Therefore the map estimates are not presented.

The results presented are generated using more pessimistic noise values than what could be expected for even very inexpensive sensor, typically available from hobby electronics suppliers for a few tens of dollars. The amount of noise introduced by the wheel encoders is also believed to be exaggerated. The interesting quantity to study is the relative performance of using the simulated ultrasonic sensor for SLAM, with angle-of-incidence dependence, compared with relying only on dead reckoning. The absolute positioning error is therefore not as relevant and all the results presented in the article are normalized against the dead-reckoning scheme.

The model presented contains a large number of parameters, but the work in this paper does not address how to optimally choose them. Rather, the same set of parameters for all parametrizations of the model are used to show

the relative merits of the method. It is surely possible to improve the results by better parameter choices and more sophisticated particle methods, but that is outside the scope of this paper. More effective methods for storing the map could be implemented. This is however not done as this is currently only a proof of concept.

The concept of modeling the angle of incidence could also be applied to methods for the SLAM problem which are not particle based.

4. Modeling

4.1 State-Space Model

The robot used is a differential-driven mobile robot, with the sensor placed at the front end. Since the robot moves in a plane, only three states are needed to describe the motion. Using the position variables p_x and p_y, as well as the heading θ as state variables, the robot's kinematics is

$$\dot{x}_r = \begin{pmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{\theta} \end{pmatrix} = \frac{R}{2} \begin{pmatrix} \cos\theta & \cos\theta \\ \sin\theta & \sin\theta \\ -2B^{-1} & 2B^{-1} \end{pmatrix} \left[\begin{pmatrix} \omega_l \\ \omega_r \end{pmatrix} + \begin{pmatrix} v_l \\ v_r \end{pmatrix} \right] \tag{1}$$

which in short is written as $\dot{x}_r = f(x_r, \omega, v)$. Here, $\omega_{l,r}$ is the left and right wheel angular velocity which have noise $v_{l,r}$, R is the wheel radius, and B is the distance between them. To be implementationally useful, this model is discretized using a second order approximation.

By assuming that the map is slowly time varying, the map can be modeled as a constant position model. Every cell in the occupancy grid representation is a state. Since each state represents a probability it should only take values between zero and one, representing the probability that a cell is occupied. The discretized occupancy grid model is on the form

$$x^m_{t+1} = x^m_t + v_m, \tag{2}$$

where each x^m is a cell in the map and v_m is the noise on the states. This representation allows that the map varies over time meaning that old measurements should not be given as much weight as more recent measurements. Taking the angle of incidence into account gives rise to additional states. As an example, assume that a map with two cells is used, and assume further that the angle dependence has a resolution of 45 degrees. This means that the cell can be viewed from 8 different angles, which gives that there will be 16 states in total for representing this simple map.

4.2 Measurement Model

Assume that an ultrasound measurement returns a distance, and that the opening angle of the sensor is β degrees. The field of view is then a closed cone with aperture 2β. The cells that are inside the field of view can now be calculated, given that a position estimate exists. Assuming that a method exists for directly measuring the states of each cell inside the field of view the measurement equation would be linear, and the C-matrix in the equation $y = Cx$ would be sparse with a single 1 per row, and the same number of rows as the number of cells inside the field of view.

The question of how to generate the measurements, y, is not trivial. A common way to generate the measurements is to create a probability function with a peak in the center of the cone, depending on the distance from the robot that the sensor returned. The probability function then decays with the angle from the center of the cone and the distance from the most probable cell, reaching the nominal value of occupancy at the end of the cone. This is called evidence mapping. Here, the focus is instead on exploiting variance. Each measurement is given a variance, dependent on which distance is returned from the sensor. If the distance is short the variance should be low, since fewer cells are visible. Furthermore, the cells' angles with respect to the center of the cone is also influencing the variance. For cells at a given distance the probability is the same. This is of course an approximation, as a single distance measurement does not contain information about the individual cells. However, it provides the conditionally linear formulation that is desired.

A typical C matrix for the example in Section 4.1 could be

$$y_t = \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \end{pmatrix} x_t^m + w_t = C_t x_t^m + w_t. \qquad (3)$$

In this particular example both cells were equally probable to be occupied since they were at the same distance from the sensor. The cells not inside the field of view at time t does not generate any measurements at all for time t. The measurement noise at time t, w_t, is a tuning parameter, parametrized by the angle and distance as previously discussed.

4.3 Map Estimation

Since the approximated measurement model is linear, a Kalman filter approach will yield the optimal result. The Kalman filter equations are written out next with the notation from (2) and (3). The reader is referred to [Anderson and Moore, 1979] for a thorough investigation on linear filtering.

$$\hat{x}^m_{t|t-1} = \hat{x}^m_{t-1|t-1} \tag{4}$$
$$P_{t|t-1} = P_{t-1|t-1} + Q_n \tag{5}$$
$$S_t = C_t P_{t|t-1} C_t^T + R_n \tag{6}$$
$$K_t = P_{t|t-1} C_t^T S_t^{-1} \tag{7}$$
$$e_t = y_t - C\hat{x}^m_{t|t-1} \tag{8}$$
$$\hat{x}^m_{t|t} = \hat{x}^m_{t|t-1} + K_t e_t \tag{9}$$
$$P_{t|t} = (I - K_t C_t) P_{t|t-1} \tag{10}$$

Here, $\hat{x}^m$ are the linear states and P is the estimated covariance matrix. The matrices Q_n and R_n are the variances of the process noise and measurement noise, respectively.

This is a convenient result. First of all, it means that the solution is analytic. Secondly, the computation of the distribution function, characterized by the mean $\hat{x}^m_{t|t}$ and the variance $P_{t|t}$, is relatively cheap compared to if a particle filter would be used for map estimation.

4.4 Pose Estimation

As already mentioned, particle filtering is used to estimate the robot's pose. A good tutorial on particle filters is [Arulampalam et al., 2002]; only the parts relevant for this paper will be explained next: The key idea with the particle filter is to approximate the density function p as a weighted set of samples

$$p_N(x_t|y_t) = \sum_{i=1}^{N} w_t^i \delta(x_t - x_t^i),\ \sum_{i=1}^{N} w_t^i = 1,\ w_t^i \geq 0, \forall i, \tag{11}$$

where δ is the Dirac function used to represent the empirical probability density function with support in N discrete samples. The weights w^i reflects the importance of each particle. These values are updated when new measurements arrive.

Each of the N particles represent a filtered trajectory estimate. To proceed in the algorithm the particles are simulated forwards one step using the input signals, yielding

$$\hat{x}_t^i = f(x_{t-1}^i, \omega_{t-1}, v_{t-1}), \quad i = 1, \ldots, N. \tag{12}$$

The weights are normally calculated according to the measurement density error function $p(y_t|x_t)$. However, due to the parametrization used for the measurements, the weights are now instead updated as

$$w_t^i = (e_t^i)^T S_t^{-1} e_t^i, \quad i = 1, \ldots, N, \tag{13}$$

$$w_t^i = w_t^i / \sum_{i=1}^{N} w_t^i. \tag{14}$$

The weighting in (13) stemming from (4)-(10) reflects how likely the particle was, given the ultrasound measurement. The state estimate may now be formed as

$$\hat{x}_t = \sum_{i=1}^{N} w_t^i \hat{x}_t^i.$$

Since all particles but a few will have negligible weights after a while, new particles are drawn with replacement according to the sampling importance resampling principle

$$\mathrm{P}(x_t^i = \hat{x}_t^j) = w_t^j, \quad j = 1, \ldots, N. \tag{15}$$

Note that this step is only performed given that the effective sampling number $1/(\sum_{i=1}^{N}(w_t^i)^2) < N_{\text{eff}}$, where N_{eff} is a number between zero and N. To summarize: The SLAM algorithm is run by first calculating (3) as explained in Section 4.2, and then running (4)-(15).

5. Implementation Details

There are a number of parameters to be chosen when implementing the model described in Section 4. In addition to this a number of minor tweaks to help with the implementation have been made.

Each particle contains a Kalman filter that estimates the probability of occupancy for each cell in the grid. Instead of storing this value directly it is first converted to log-odds format and thus spans the entire range of real numbers instead of only the interval $(0, 1)$. All measurements are also converted to this format before being passed to the Kalman filter. The covariance matrix grows quadratically with the map size, therefore it is critical to exploit the sparsity that occurs. Under the assumption that all cells are independent only the diagonal elements will be nonzero.

Because of the large field of view of the sensor it rarely provides accurate information on the position of objects. However, it provides information on which cells are likely to be empty. To exploit this fact a threshold value has been added, and whenever the measured range exceeds this value only the

cells that are detected as empty are included in the measurement update. For closer distances also the cells that are on the edge of the cone and thus likely to be occupied are included in the measurement update.

Since the model not only contains an estimate of the probability of occupancy, but also an estimate of how certain that estimate is, it is possible to have a very high prior probability of occupancy since the map will quickly converge to the new estimate once a measurement has been made. This allows the algorithm to successively clear areas of the map, relying more on the information on which areas are empty rather than those that are occupied. This is a better match for the characteristics of an ultrasound sensor.

6. Results

The focus of this section is to show the relative merits of the method for different sensor characteristics, and to give an indication of what performance different number of angular subdivisions of each cell provide. Therefore the figures are normalized such that the absolute position error of the dead reckoning is 1 at time 1.

The data used for the simulations were generated by driving a robot in a simulated environment consisting of a number of walls with different orientations and lengths. The trajectory was a circle repeated roughly twenty times, with the sensor sweeping back and forth through 180° in front of the robot.

The data presented in this section are the absolute errors of the position estimates. As a reference comparison for the SLAM algorithm the same number of particles (200) was simulated using the same noise model, but only using the information from the wheel encoders. The position of the dead-reckoning estimate was then taken as the mean of these particles. For the SLAM estimate the position was taken as a weighted mean with the weight at each instant being proportional to how good the particular particle is deemed to be, with the particles being resampled whenever the weights are considered to be too unevenly distributed.

The results presented in Fig. 2 are the mean of the estimates and standard deviations calculated by repeated Monte-Carlo simulations using different noise perturbations of the correct input. As can be seen in the figure the SLAM estimate is clearly improved by subdividing each cell taking the angle of incidence into consideration. Most notable is the significant decrease in the variance of the estimate, the mean error is roughly the same for all the different subdivisions.

Unexpectedly this method also seems to improve the estimates when using a simulated sensor with no dependence on the angle of incidence, which can be seen in the right-most column in Fig. 2. It is believed that this could

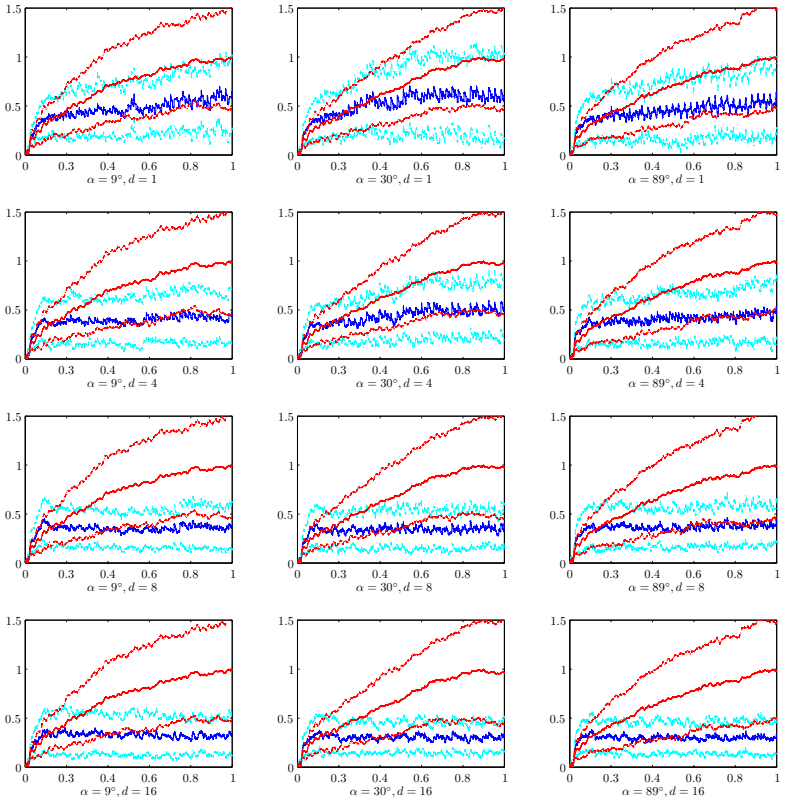

Figure 2. Comparison of the absolute position error for 3 different sensors with different characteristics and number of angular subdivisions for each grid cell. All the simulated sensors have an opening angle of 60°. The maximum angle of incidence is denoted α, the number of subdivisions is denoted d. Note that $d = 1$ is similair to a normal occupancy grid approach. All figures are normalized such that the total accumulated error of the dead reckoning is 1 at time 1. The red line is the accumulated error for the dead-reckoning. For each simulation the dead reckoning is evaluated as the mean of 200 particles simulated using the same noise model as that provided to the SLAM algorithm. The blue line corresponds to the same quantities when using the SLAM algorithm. The position of the SLAM estimate is taken as the weighted average of 200 simulated particles. The dot-dashed line is the 1 standard-deviation interval estimated by repeatedly running the Monte-Carlo simulation but with different noise realizations for each iteration. From the results shown it can be seen that for all the sensor characteristics it is beneficial to increase the number of subdivisions, d, for each cell. Counter-intuitively this also seems to hold for the 'perfect' sensor where the angle of incidence, α, does not affect the measurement.

be explained from the fact that the wide field of view of the sensor is still a limiting factor of the performance, and that using the angular subdivision scheme increases the resolution by separating the measurements from different positions. This effect, however, might not always be desirable since it also decreases the correlation between different robot poses, making it harder for the particle filter to converge.

7. Conclusions and Future Work

The results presented in this paper show that by extending the concept of the regular occupancy grid to model the fact that objects are not necessarily detected from all positions within a room, the performance for a particle filtering based SLAM algorithm can be significantly improved. The method presented here can thus be seen as a way to trade sensor performance against computational resources. In a world with computing power becoming cheaper day by day, this tradeoff will become more relevant as time progresses, perhaps allowing the use of SLAM algorithms in inexpensive consumer products.

The model presented in this paper is clearly suited for Rao-Blackwellized particle methods due to its conditional linearity, and in the future the implementation will most likely be extended with smoothing methods, such as those presented by [Lindsten and Schön, 2011]. It is believed that smoothing will decrease the position error since a single measurement contains fairly little information of the robot pose. Also, the problem with particle depletion should be possible to solve with particle smoothing. Therefore a method which takes the trajectory into consideration is likely to perform significantly better, but with larger computational burden

More extensive simulations and individual tweaking of the parameters for the different number of subdivision of the cells would be beneficial for clearly establishing the methods merits, but our initial results seem promising.

We will gather actual measurement data under conditions where we have access to a reliable ground truth. In this way it is possible to provide proper estimates on the accuracy of the method under real-world conditions.

References

Ahn, S., J. Choi, N. L. Doh, and W. K. Chung (2008). "A practical approach for EKF-SLAM in an indoor environment: fusing ultrasonic sensors and stereo camera". *Auton. Robots* **24** (3), pp. 315–335. ISSN: 0929-5593.

Anderson, B. D. O. and J. B. Moore (1979). *Optimal Filtering*. Prentice Hall, Englewood Cliffs, NJ.

Arulampalam, M., S. Maskell, N. Gordon, and T. Clapp (2002). "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking". *IEEE Trans. Signal Process.* **50**:2, pp. 174–188. ISSN: 1053-587X.

Burgard, W., D. Fox, H. Jans, C. Matenar, and S. Thrun (1999). "Sonar-based mapping with mobile robots using EM". In: *Proc. of the 16th Intl. Conf. on Machine Learning*.

Durrant-Whyte, H. and T. Bailey (2006). "Simultaneous localisation and mapping (SLAM): part 2". *IEEE Robot. Autom. Mag.* **13**, pp. 108–117.

Eliazar, A. and R. Parr (2003). "DP-SLAM: fast, robust simultaneous localization and mapping without predetermined landmarks". In: *Proc. 18th Int. Joint Conf. on Artificial Intell.* Morgan Kaufmann, pp. 1135–1142.

Grisetti, G., C. Stachniss, and W. Burgard (2005). "Improving grid-based SLAM with Rao-Blackwellized particle filters by adaptive proposals and selective resampling". In: *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*. Barcelona, Spain, pp. 2443–2448.

Grisetti, G., C. Stachniss, and W. Burgard (2007). "Improved techniques for grid mapping with Rao-Blackwellized particle filters". *IEEE Trans. Robot.* **23**, pp. 34–46.

Hähnel, D., W. Burgard, D. Fox, and S. Thrun (2003). "A highly efficient FastSLAM algorithm for generating cyclic maps of large-scale environments from raw laser range measurements". In: *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*. Las Vegas, NV, pp. 206–211.

Kwak, N., I. K. Kim, H. C. Lee, and B. H. Lee (2007). "Analysis of resampling process for the particle depletion problem in FastSLAM". *Robot and Human interactive Communication, 2007. RO-MAN 2007. The 16th IEEE Int. Symposium on*, pp. 200–205.

Leonard, J. J., H. F. Durrant-Whyte, and I. J. Cox (1992). "Dynamic map building for an autonomous mobile robot". *Int. J. Rob. Res.* **11** (4), pp. 286–298. ISSN: 0278-3649.

Lindsten, F. and T. Schön (2011). *Rao-Blackwellized Particle Smoothers for Mixed Linear/Nonlinear State-Space Models*. Tech. rep. URL: `http://user.it.uu.se/~thosc112/pubpdf/lindstens2011.pdf`.

Rencken, W. (1993). "Concurrent localisation and map building for mobile robots using ultrasonic sensors". In: *Proc. of the 1993 IEEE/RSJ Int. Conf. on IROS '93*. Vol. 3, 2192–2197 vol.3.

Siciliano, B. and O. Khatib, (Eds.) (2008). *Springer Handbook of Robotics*. Springer, Berlin, Heidelberg. ISBN: 978-3-540-23957-4.

Smith, R., M. Self, and P. Cheeseman (1990). "Estimating uncertain spatial relationships in robotics". In: Springer-Verlag New York, Inc., New York, NY, pp. 167–193. ISBN: 0-387-97240-4.

Thrun, S. (2002). "Robotic mapping: a survey". In: Lakemeyer, G. et al. (Eds.). *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann.

Thrun, S. (1998). "Learning metric-topological maps for indoor mobile robot navigation". *Artif. Intell.* **99**:1, pp. 21–71. ISSN: 0004-3702.

Paper VIII

Rao-Blackwellized Particle Smoothing for Occupancy-Grid Based SLAM Using Low-Cost Sensors

Karl Berntorp Jerker Nordh

Abstract

The simultaneous localization and mapping problem is approached by using an ultrasound sensor and wheel encoders on a mobile robot. The measurements are modeled to yield a conditionally linear model for all the map states. We implement a recently proposed Rao-Blackwellized particle smoother (RBPS) for jointly estimating the position of the robot and the map. The method is applied and successfully verified by experiments on a small Lego robot where ground truth was obtained by the use of a VICON real-time positioning system. The results show that the RBPS contributes with more robust estimates at the cost of computational complexity and memory usage.

1. Introduction

Having a robot simultaneously localizing itself and learning its map is commonly referred to as simultaneous localization and mapping (SLAM). The problem is often solved using odometry in combination with vision or range sensors. In mobile robotics it has been studied extensively over the last three decades. For surveys and tutorials of the SLAM problem and its different solutions up to recently, see for example [Thrun, 2002] or [Durrant-Whyte and Bailey, 2006].

At least since the early 1990's the approach to SLAM has been probabilistic. In [Smith et al., 1990], extended Kalman filtering (EKF) was used for state estimation. An issue with using Kalman filtering is that the nonlinearities that typically are present tend to lead to divergence of the state estimates. For example, the kinematics of a planar mobile robot is nonlinear in the heading angle, and the consequent linearizations that the EKF uses for estimating the odometry may lead to instability. To remedy this, particle filtering was introduced as a means to solve the SLAM problem. State-of-the-art particle filter algorithms when using high-resolution laser scanners are found in [Grisetti et al., 2005; Grisetti et al., 2007].

One way to describe the map is to use occupancy-grid mapping [Siciliano and Khatib, 2008]. The grids are considered to be either occupied or free, with some probability distribution associated with the grid. One usage of occupancy-grid maps is when utilizing range sensors, such as laser sensors or sonar sensors. Both types of sensors have noise and may occasionally give severe measurement errors. Since laser sensors have very high spatial resolution, thus giving a sharp probability distribution, they appear to be the most common solution, see [Hähnel et al., 2003], [Eliazar and Parr, 2003], [Grisetti et al., 2007]. In contrast, sonar sensors cover a cone in space. Because of the low spatial resolution in the tangential direction, it is impossible to determine from a single measurement whether a certain cell is occupied or not. Also, ultrasound sensors are sensitive to the angle of an object's surface relative to the sensor; that is, they have a small angle of incidence. This leads to that measurements of the same surface from slightly different angles may render different results. Obviously, this could potentially lead to estimation errors and must be handled by the algorithms.

In [Nordh and Berntorp, 2012] we performed SLAM using differential-driven mobile robots equipped with an ultrasound sensor. To handle the deficiencies with sonar sensors we extended the occupancy-grid concept by dividing every cell into subcells. Further, we developed a conditionally linear Gaussian state-space model for use in SLAM. In this paper we propose a novel measurement model aimed at sonar sensors, and extend the work in [Nordh and Berntorp, 2012] to include Rao-Blackwellized particle smoothing (RBPS) as a means for SLAM. Rao-Blackwellization takes advantage of

a linear substructure in the model, which can be handled by a Kalman filter. Rao-Blackwellized particle filtering (RBPF) has been used for a number of years, see [Andrieu and Doucet, 2000], [Doucet et al., 2000], [Schön et al., 2005]. During the last years Rao-Blackwellized particle smoothers have gained interest, see [Särkkä et al., 2012], [Lindsten and Schön, 2011], where RBPS are developed for conditionally linear Gaussian models. We utilize the methods in [Lindsten and Schön, 2011] and provide an extension to also handle uniform noise for the class of differential-drive mobile robots. Particle filters for SLAM tend to have a particle depletion problem, thus producing overconfident estimates of uncertainty and eliminating particles with low weights. This leads to that the number of particles used to perform loop closure decreases, yielding degenerate performance [Kwak et al., 2007]. Using particle smoothing could potentially eliminate this loop-closure problem since more information is available for map estimation and position localization.

We verify the method on a Lego Mindstorms mobile robot, see Fig. 1, equipped with a low-cost ultrasonic range finder. The Lego Mindstorms robot has low-performance motors, with severe backlash and highly uncertain encoder readings. Therefore, this setup represents a worst-case scenario.

1.1 Related Work

Using sonar sensors for SLAM has been studied before; an example is [Burgard et al., 1999], in which an offline expectation maximization algorithm was used for occupancy-grid mapping using 24 Polaroid sensors with 15 degrees opening angle. An early work is [Rencken, 1993], where the SLAM problem was solved in simulation using 24 ultrasonic sensors by estimating the errors introduced in the localization and mapping parts, respectively, and correcting for them using a modified Kalman filter approach. A third example is [Leonard et al., 1992], in which a feature-based approach with the aid of servo-mounted ultrasonic sensors was used. A more recent work is [Ahn et al., 2008], where ultrasonic sensors and a stereo camera is used in an EKF-SLAM setting. All of the previously mentioned work either use offline approaches and/or a vast number of sensors to perform SLAM. As such, their approaches are quite different from ours.

The state-of-the-art algorithms mentioned earlier, [Grisetti et al., 2005; Grisetti et al., 2007], are designed for use with laser scanners. Further, laser scanners, besides having high precision, are several orders of magnitude more expensive than the sensors we use. Therefore, a comparison with laser-based approaches would be unfair.

1.2 Outline

The structure of the rest of the paper is as follows: In Sec. 2 we give the preliminaries. Sec. 3 presents the state-space and measurement model. In Sec. 4 we explain the forward filter used. In Sec. 5 we summarize the RBPS, derive how to handle uniform noise in the RBPS, and discuss implementation aspects. The algorithm is evaluated in Sec. 6. Finally, we conclude the paper in Sec. 7.

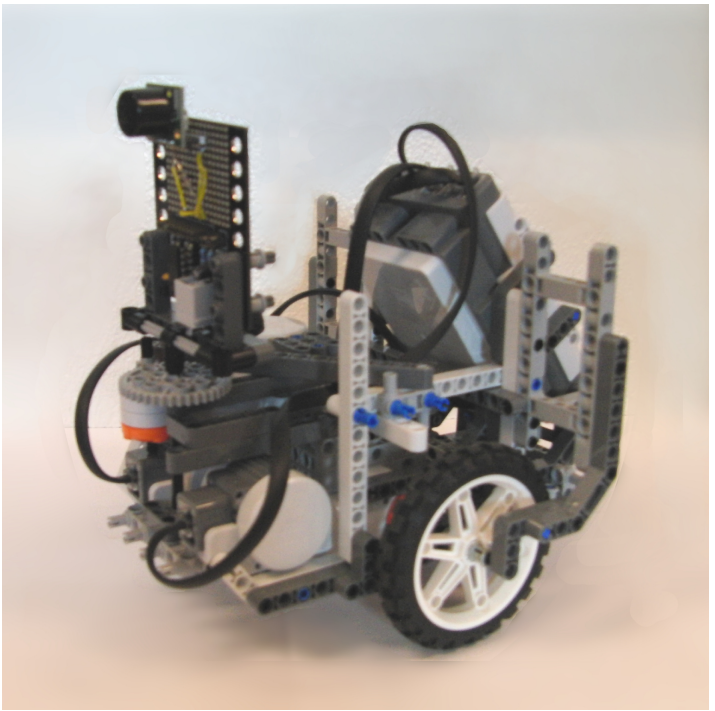

Figure 1. The Lego Mindstorm differential-driven mobile robot used in the experiments. The ultrasound sensor is placed at the front end of the robot.

2. Preliminaries

The conditional distribution density of the variable x at time index k conditioned on the variable y from time index i to time index k is denoted $p(x_k|y_{i:k})$. At each time step k the state can be partitioned into a non-linear part, ξ_k, and a linear part, z_k. The total state vector is a discrete-time

Markov process, which obeys the transition density $p(\xi_{k+1}, z_{k+1}|\xi_k, z_k)$. The robot model is written as

$$\xi_{k+1} = f(\xi_k, u_k, v_k, w_k), \tag{1}$$
$$z_{k+1} = A(\xi_k)z_k + e_{z,k}, \tag{2}$$
$$y_k^m(r_k, \xi_k) = C(r_k, \xi_k)z_k + e_{y^m,k}(r_k, \xi_k), \tag{3}$$

where $\xi \in \mathbb{R}^{7\text{x}1}$. The state vector $z \in \mathbb{R}^{n\text{x}1}$ contains the cells of the modified occupancy grid, where the size n depends on the map dimension, resolution, and the number of subcells used in every cell. The measurement function $C(\cdot)$ and measurement y^m are parametrized in the nonlinear state vector ξ and the ultrasound range measurement r. The inputs enter in the nonlinear states, and z is linear given ξ. The process noise v is independent uniform with zero mean, and w is white Gaussian with zero mean. Moreover, the process noise e_z and measurement noise e_{y^m} are assumed to be white Gaussian with zero mean.

3. Modeling

Here, we derive the robot kinematics and the conditionally linear map model. Moreover, we describe how the measurement equation is approximated as linear despite that the ultrasound sensor measurements are highly nonlinear.

3.1 State-Space Model

The robot used is assumed to be a differential-driven mobile robot, equipped with a sonar sensor. Since the robot moves in a plane, only three states are needed to describe the motion in continuous time. Using the position variables x and y, as well as the heading θ as state variables, the discretized kinematics (1) is, using a bilinear transformation, written as

$$\xi_{k+1} = f(\xi_k, u_k, v_k, w_k). \tag{4}$$

Here, $\xi_k = (x_k\ y_k\ \theta_k\ P_{k-1}^R\ P_{k-1}^L\ P_{k-2}^R\ P_{k-2}^L)^{\mathrm{T}}$ is the state vector, with $P_k^{R,L}$ being the right and left wheel encoder positions at time index k. The input u_k equals $u_k = (P_{k+1}^R \quad P_{k+1}^L)^{\mathrm{T}}$, and the wheel encoder noise vector is assumed uniformly distributed according to $v_k \sim U(-\alpha, \alpha)$. The process noise w_k only enters in θ with variance Q_w. After introducing

$$\bar{\theta}_k = \theta_k + \begin{pmatrix} \frac{1}{2l} & -\frac{1}{2l} \end{pmatrix} \begin{pmatrix} P_{k-2}^R \\ P_{k-2}^L \end{pmatrix} + \begin{pmatrix} \frac{1}{2l} & -\frac{1}{2l} \end{pmatrix} (u_k + v_k),$$

where l is the wheel axis length, the kinematics vector $f(\xi_k, u_k, v_k, w_k)$ becomes

$$f(\xi_k) = \begin{pmatrix} 1 & 0 & 0 & \frac{1}{4}a & \frac{1}{4}a & -\cos\theta_k & -\cos\theta_k \\ 0 & 1 & 0 & \frac{1}{4}b & \frac{1}{4}b & -\sin\theta_k & -\sin\theta_k \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \xi_k + \begin{pmatrix} \frac{1}{4}\cos\bar{\theta}_k & \frac{1}{4}\cos\bar{\theta}_k \\ \frac{1}{4}\sin\bar{\theta}_k & \frac{1}{4}\sin\bar{\theta}_k \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} (u_k + v_k) + \begin{pmatrix} 0 \\ 0 \\ \bar{\theta}_k + w_k \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

where $a = \cos\theta_k - \cos\bar{\theta}_k$, and $b = \sin\theta_k - \sin\bar{\theta}_k$. This model is affected both by the wheel encoder noise and the Gaussian distributed noise entering in θ. Wheel encoders typically have backlash uncertainties, which may be modeled as uniform noise. Moreover, the robot exhibits wheel slip, which provides the physical justification for the Gaussian θ-noise. As we will discuss later the Gaussian noise can also be motivated with that it helps both the smoothing and filtering to perform better, by mitigating the issue of particle depletion.

The map is modeled as a slowly time-varying linear model; that is, every cell in (2) is modeled as

$$z_{k+1} = z_k + e_{z,k}, \tag{5}$$

where z contains the probability that a cell is occupied, stored in log-odds format. The process noise $e_{z,k}$ is a tuning parameter reflecting that the uncertainty of the cell grows when no new measurements arrive.

3.2 Measurement Model

Assume that an ultrasound measurement returns a distance to an object, and that the opening angle of the sensor is β degrees. The field of view is then a closed cone with aperture 2β. The cells that are inside the field of view can now be calculated, given that a position estimate exists. Assuming that a method exists for converting a single range measurement to an observation of the map states inside the field of view, the measurement equation would be linear. Further, the C-matrix in (3) would be sparse with a single 1 per row and with the same number of rows as the number of cells inside the field

of view. A typical C matrix in (3) could be

$$y_k^m(r_k, \xi_k) = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \end{pmatrix} z_k + e_{y^m,k}(r_k, \xi_k). \tag{6}$$

The cells not inside the field of view at time index k do not generate any measurements at all for time index k.

To convert a range measurement to an observation of the map states, we set y_k^m to a high probability ($p = 0.99$) of occupancy for all the cells along the arc at the range of the distance measurement. All the cells closer than the measured distance are considered to be empty ($p = 0.01$). The additive noise is also parametrized by the distance so that when the detected object is far away the measurement is considered more noisy, thus suppressing the influence on the map. This reflects the high spatial uncertainty along the arc.

4. Forward Filtering

The particle smoothing needs a weighted particle estimate as input. However, this estimate does not necessarily have to be generated by the same model as the one used for RBPS. In our case we use a slightly modified particle filtering approach.

For an RBPF in our setup, (1)–(3), the particle weights would be updated using the marginal density function of the linear states for the measurement. Instead we use a nonlinear function of the robot position and the map with a more physically intuitive formulation, which has proved effective in both simulations and on real data. Conditioned on the robot position all the cells that are inside the field of view are collected in a list and then sorted in ascending order of the range to the robot. The list is then traversed and the probability of each cell generating that measurement is calculated and weighted by the probability that all the earlier cells were empty, thus yielding

$$p(r_k) = \sum_{j=1}^{n} p(r_k|z_k^j) p_{\text{occ}}(z_k^j) \prod_{i=1}^{j-1} (1 - p_{\text{occ}}(z_k^i)), \tag{7}$$

where z_k^j is cell j at time index k. Each individual probability in (7) is evaluated against a flat probability density centered around the center point of the cell with width equal to the diagonal of a cell. Outside the flat region it falls off linearly over another cell width. See Fig. 2 for a visualization. This models the fact that a map with a certain resolution cannot differentiate measurements with higher precision than the resolution.

In a standard RBPF the weights of each particle $i = 1, \ldots, N$ is updated by multiplying it with the new weight provided by the measurement. However, we have replaced this multiplication by a first order low-pass filter.

The resulting algorithm is no longer a true RBPF, but from experience using both simulations and experimental data we have found that it works very well for our problem formulation.

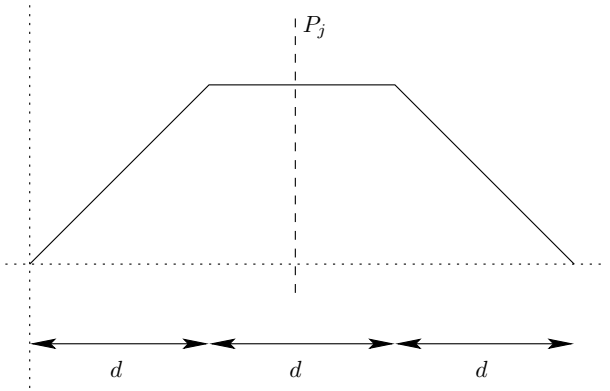

Figure 2. The probability density function at time index k, $p(r_k|z_k^j)$, for a single measurement conditioned on cell j being the origin of that measurement. P_j is the center point of cell j, and d is the diagonal width of a cell

5. Rao-Blackwellized Particle Smoothing

In this section we only summarize the RBPS algorithm and provide our extension. The reader is referred to [Lindsten and Schön, 2011] for algorithm details.

5.1 Rao-Blackwellized Particle Smoother–Summary

The RBPS in [Lindsten and Schön, 2011] consists of three main steps:

1. A backward particle smoother
2. A forward Kalman filter
3. A Rauch-Tung-Striebel (RTS) smoother

Backward Particle Smoother The backward pass starts with initializing $j = 1, \ldots, M$ backward trajectories at time T—that is, initializing $\{\xi_T^j, z_T^j\}$ using the filtered estimates at time T. Then, for the timespan of interest, rejection sampling is performed to find an index $I(j)$ corresponding to the forward filter particle that is to be appended to the j-th backward trajectory.

This index is exploited to set $\xi_k^j = \xi_k^{I(j)}$ and to draw linear samples from the Gaussian density

$$p(z_k|\xi_{1:k}^{I(j)}, \xi_{k+1}, z_{k+1}, y_{1:k}^m) \propto p(\xi_{k+1}, z_{k+1}|\xi_{1:k}^{I(j)}, y_{1:k}^m) \times p(z_k|\xi_{1:k}^{I(j)}, y_{1:k}^m,). \tag{9}$$

Finally, the backward trajectory vector is appended, yielding $\{\xi_{k:T}^j, z_{k:T}^j\} = \{\xi_k^j, \xi_{k+1:T}^j, z_k^j, z_{k+1:T}^j\}$.

Kalman Filter After the backward smoothing step the nonlinear states are assumed fixed, thus yielding a linear model suitable for Kalman filtering [Anderson and Moore, 1979]. Hence, for each $j = 1, \ldots, M$ a Kalman filter runs for the whole timespan using (6).

RTS Smoother After the Kalman filter step both linear and nonlinear estimates are available. Then, an RTS smoothing (backward pass) step is performed, concluding the RBPS-algorithm. Note that steps 2 and 3 are needed to find continuous, unsampled, conditional smoothing densities.

5.2 RBPS with Uniform Noise

In the rejection sampling step in Sec. 5.1, [Lindsten and Schön, 2011] exploits that the forward density $p(\xi_{k+1}^j, z_{k+1}^j|\xi_{1:k}^i, y_{1:k}^m)$, where j is the trajectory index and i is the particle index, is Gaussian distributed. Also, calculation of the maximum of the distribution is needed. However, since we also have uniform noise in (4), some modifications are required:

From the factorization and Markov properties of (4) we have

$$\begin{aligned} p(\xi_{k+1}, z_{k+1}|\xi_{1:k}, y_{1:k}^m) &= p(\xi_{k+1}|\xi_{1:k})p(z_{k+1}|\xi_{1:k}, y_{1:k}^m) \\ &= p(\xi_{k+1}|\xi_k)p(z_{k+1}|z_k), \end{aligned} \tag{10}$$

where the input u_k is suppressed for reasons of notation. The second factor of (10) is the linear filtering density—that is, the density resulting from a Kalman filter. However, since the map is modeled as a slowly time-varying map, we have assumed it to be constant in the implementation. We can deduce the first factor in (10) as follows: Assume that we have the estimate ξ_{k+1}^*. Then the probability $p(\xi_{k+1} = \xi_{k+1}^*|\xi_k, u_k)$ can be reformulated as

$$p(\xi_{k+1} = \xi_{k+1}^*|\xi_k, u_k) = p(\xi_{k+1}^* = f(\xi_k, u_k, v_k, w_k)|\xi_k, u_k). \tag{11}$$

Due to the form of (4) we can solve $f(.)$ for v_k and w_k. Thus (11) transforms to

$$p((v_k,\, w_k) = g(\xi_{k+1}, \xi_k, u_k)|\xi_{k+1}, \xi_k, u_k). \tag{12}$$

Using the noise independence properties yields that (12) is equal to

$$p(v_k|g_1(\xi_{k+1}, \xi_k, u_k))p(w_k|g_2(\xi_{k+1}, \xi_k, u_k)). \tag{13}$$

To find the solution to (13), we start by forming the difference of (4) between two consecutive time steps (i.e., $\Delta x = x_{k+1} - x_k$). After reshuffling the equations for the translational coordinates we get

$$\cos\theta_{k+1} = \frac{4\Delta x - (P_k^R - P_{k-1}^R + P_k^L - P_{k-1}^L)\cos\theta_k}{P_{k+1}^R - P_k^R + P_{k+1}^L - P_k^L} \tag{14}$$

$$\sin\theta_{k+1} = \frac{4\Delta y - (P_k^R - P_{k-1}^R + P_k^L - P_{k-1}^L)\sin\theta_k}{P_{k+1}^R - P_k^R + P_{k+1}^L - P_k^L} \tag{15}$$

$$\Delta\theta = \frac{1}{2l}\left(P_{k+1}^R - P_{k-1}^R - (P_{k+1}^L - P_{k-1}^L)\right). \tag{16}$$

Utilizing the trigonometric identity on (14) and (15) gives that

$$(P_{k+1}^R + P_{k+1}^L - P_k^R - P_k^L)^2 = \gamma_1 \tag{17}$$

for some constant γ_1. Likewise, we get from (16) that

$$P_{k+1}^R - P_{k+1}^L = \gamma_2, \tag{18}$$

where $\gamma_2 = 2l\Delta\theta + P_{k-1}^R - P_{k-1}^L$. Moreover, by dividing (15) with (14) we can solve for the θ_{k+1} congruent to (x_{k+1}, y_{k+1}):

$$\theta_{k+1}^{1,2} = \text{atan2}(d_1, d_2) + m\pi, \quad m = 0, 1, \tag{19}$$

where $\text{atan2}(\cdot,\cdot)$ is the four quadrant inverse tangent, and

$$d_1 = 4\Delta x - (P_k^R - P_{k-1}^R + P_k^L - P_{k-1}^L)\cos\theta_k,$$
$$d_2 = 4\Delta y - (P_k^R - P_{k-1}^R + P_k^L - P_{k-1}^L)\sin\theta_k.$$

From (17) we see that there are two input vectors that give the same (x, y)-coordinates—that is,

$$P_{k+1}^R + P_{k+1}^L = P_k^R + P_k^L \pm \sqrt{\gamma_1}. \tag{20}$$

Also, from (16), (18), and (19), we find the two possible input vectors coincident to the two solutions of (20) to be

$$P_{k+1}^R - P_{k+1}^L = 2l\Delta\theta^{1,2} + P_{k-1}^R - P_{k-1}^L. \tag{21}$$

Using (20) and (21) we can calculate the (uniformly distributed) probability, $p(P_{k+1}^{R,L})$, of the two possible combinations of input vectors that yield the coordinates Δx, Δy, and $\Delta\theta^{1,2}$. The probability for the two possible solutions $\theta_{k+1}^{1,2}$ is Gaussian. Now (12) is formed by multiplying the Gaussian probability for $\theta_{k+1}^{1,2}$ with the two distributions for the wheel encoders and

choosing the solution with the largest probability. Further, the maximum of $p(\xi_{k+1}, z_{k+1}|\xi_{1:k}, y^m_{1:k}, u_k)$ is found by setting $\theta^{1,2}_{k+1}$ to be in the middle of the distribution.

We now transform (9), which for our model yields

$$\begin{aligned} p(z_k|\xi_{1:k+1}, z_{k+1}, y^m_{1:k}) &= p(z_k|\xi_{1:k}, z_{k+1}, y^m_{1:k}) \\ &\propto p(z_{k+1}|z_k, \xi_{1:k}, y^m_{1:k}) p(z_k|\xi_{1:k}, y^m_{1:k}) \\ &= p(z_{k+1}|z_k) p(z_k|\xi_{1:k}, y^m_{1:k}). \end{aligned} \tag{22}$$

Here, the first equality follows from that there is no new measurement at time index $k+1$, whereas the second and third steps follow from Bayes' rule and the Markov property, respectively. Now, (22) may be estimated using a Kalman filter.

Remark 1
We do not use an RBPF for finding the density (10), see Sec. 4. However, this is not a restriction since for the RBPS it actually does not matter how the density was produced. □

Remark 2
The addition of the Gaussian state noise for θ can be motivated from that it can be used to model wheel slip. However, it also improves smoothing performance. Assume that we at time index k have the robot position estimate (x^0, y^0, θ^0), and that the estimate at time index $k+1$ is (x^1, y^1, θ^1). Then θ^1 is uniquely determined by the initial conditions and x^1 and y^1, since there are not enough degrees of freedom in the motion model, see Fig. 3 for an illustration. This means that the smoother will never switch between trajectories, caused by only having two degrees-of-freedom noise. This leads to that the smoother (and filter) will be unable to recover from spurious errors. However, if noise (i.e., an additional degree of freedom) is added to θ we remedy this, which is essential for the smoothing to give any impact, and for the forward filter to avoid particle depletion. □

5.3 Implementation Aspects

The computational demands when using smoothing are greater than for filtering. Not only the amount of computations needed increases, but there is also a need for storing the filtered estimate for each time instant over which the smoothing will be performed. For the problem described in this paper each particle contains an estimate of the entire map, thus it is of significant size. This means that for larger maps the memory requirements will be problematic unless some representation of the map which takes advantage of the similarities between particles at time instances is used. However, this is not an issue we investigated further in this work.

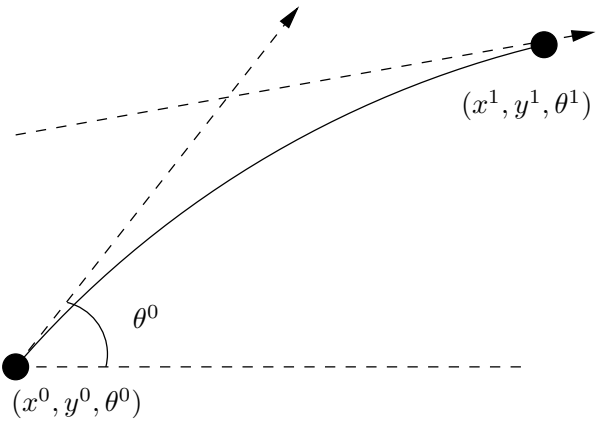

Figure 3. The bilinear transformation (the arc) is a second order approximation of the robot motion. Thus, the end point is uniquely determined from the initial conditions and (x^1, y^1). This implies that when evaluating (10) for all j trajectories conditioned on particle i, the probabilities will equal zero when $j \neq i$. By adding noise for θ this uniqueness disappears, which implies that the smoother will be able to recover from errors caused by, for example, wheel slip.

When implementing the RBPS for SLAM some parameter considerations have to be made. For example, the performance depends both on the time window used for the smoothing and the number of trajectories. Also, we will have to decide when to trigger the smoothing. Further, when the smoothing is finished it is not obvious from where we should reinitialize the forward filter. Since we are primarily interested in real-time estimation, it is not feasible to perform smoothing over the entire data set. We therefore use smoothing over a subset of the data. We then reinitialize the filtering at a point within the subset and restart from there. This provides the filtering with a "future" estimate of the map, which in theory should make it possible to perform better: Assume that the smoothing was triggered at time index k, and that we have a time window of length t. Thus, we suspend the forward filtering at time index k. Also assume that we initialize an additional forward filter halfway through the smoothing time window—that is, at time index $k-t/2$. Then we use the smoothed estimates at time index $k-t/2$ to initialize this additional forward filter, and executes it up to time index k. At time index k we may then attach these estimates and resume the original forward filter. Overlapping the filtering and smoothing in this way means that we can improve the future filtering with the smoothing. However, it increases the total amount of computations needed since we recompute the same time step several times.

The implementation as presented in this paper is available as open source software, see [Nordh and Berntorp, 2013a]. It is built on top of the open source framework pyParticleEst, see [Nordh and Berntorp, 2013b].

6. Experimental Results

For the experiments we used a differential-driven mobile robot built using Lego Mindstorms, see Fig. 1. One of the aims of the present work is to use low-cost sensors, with rather poor performance. The sensor chosen was an ultrasonic range finder XL-Maxsonar EZ4 [Inc., 2012], with a resolution of 1 cm at 10 Hz. The maximum target range is 7.65 m, but because of voltage division in the building process the range was limited to approximately 3.8 m. Further, the maximum angle of incidence was measured to be approximately 10 deg. The sensor was mounted on a motor, enabling it to sweep back and forth. The motor used for this was of the same type as those used for driving the robot, which are part of the standard Lego Mindstorms toolkit. The backlash of the motors was estimated to roughly 5 deg. When it comes to precision in odometry, we believe that this setup represents a worst-case scenario.

The ground truth, only used for evaluation, was gathered using a VICON real-time positioning system, see Fig. 4, installed at Linköping University, Sweden. The VICON system uses 10 infrared cameras and infrared lamps to track markers attached to the robot. The positioning precision provided by the system is about 1 mm.

The results were generated by effectuating the algorithm 40 times on the same data set. We set the number of backward trajectories to $M = 25$ and the number of forward particles to $N = 100$. The maximum smoothing length was set to 400. The reinitialization overlap was set to be half of the smoothing time window. Further, we chose to trigger the smoothing every fifth resample of the forward filter. These parameter choices were quite arbitrary, and most likely a better trade-off between complexity and performance can be found by tweaking the parameters.

We chose the mean of the norm of the position error at each time instance as performance measure. Figure 5 shows the results for smoothing-based SLAM, forward-filter SLAM, and dead reckoning. Included in the figure is the standard deviation for all three methods. It is clear that although the smoothing-based SLAM does not have much smaller mean error, it is more robust than only using the forward filter. The dead reckoning is deterministic since each execution was performed on the same data set. At about $t = 120$ the SLAM estimates seem to have converged to an error of about 0.4 m, which is of the same order of magnitude as the map resolution (0.2 m). The error in the odometry is clearly increasing with time, and gives substantially

Figure 4. The experimental setup. The cameras in the upper part are part of the VICON system.

larger error than the SLAM algorithms in the position estimate for $t > 140$. Note that the SLAM algorithms start with no knowledge about its environment, and, hence, they do not outperform the odometry until sufficient map knowledge has been built up. The data set used was challenging in that during the last seconds the dead reckoning diverged quickly, with an angle error of about 180 deg and a position error of approximately 3 m. This was caused by severe wheel slip. The two SLAM algorithms managed to handle this in about 2/3 of the realizations.

Figure 5 is truncated in order to more clearly visualize the differences between the filtering and smoothing. In Fig. 6 we show results from a successful execution of the whole data set, where the smoothing-based SLAM has a mean error that is roughly the same as the map resolution throughout the realization. Further, it manages to converge to the ground truth at the execution end time. Although the error of the forward-filter SLAM in Fig. 6 is from an unfortunate realization, it still shows that the smoothing-based SLAM is more robust and consistent. Results as inadequate as the forward-filter SLAM in Fig. 6 is not something we have observed when using the smoothing-based SLAM.

In Fig. 7 we visualize a map generated using the ground truth robot positions from the VICON system. In Fig. 8 we show the map obtained by the SLAM algorithm presented in this paper. For both cases each grid cell in the

map actually consists of eight sub-cells corresponding to different directions in the environment according to the occupancy-grid extension mentioned in Sec. 1, see [Nordh and Berntorp, 2012]; what is visualized is the direction from which it is most likely to observe something, for each cell. In the maps presented the color blue represents areas that are unexplored by the sensor. The red color scale, starting from black, indicates the probability of a cell being occupied, starting from probability zero. The green color scale corresponds to the uncertainty (variance) of the probability estimate, with black implying large uncertainty and green corresponding to low uncertainty. Thus a yellow cell corresponds to that it is likely to be occupied, and there is low uncertainty associated with it. A green cell is believed to be empty with low uncertainty. A red cell is believed to be occupied, but because of a lack of good measurements of that area the uncertainty is large. Similarly, black corresponds to areas believed to be empty, but with large uncertainty.

In Fig. 7 the position estimates for both the SLAM and dead reckoning are overlaid on the map, corresponding tho the last time step in Fig. 6. Also shown is the reference position from the VICON system. As seen the SLAM algorithm nearly coincides perfectly with the reference position at the end time, whereas the dead reckoning is several meters of, mainly due to the large wheel slip around $t = 210$. By visual inspection only, the map appears quite poor, and indeed it is of quite low resolution (each cell is 0.2 m wide) and rather noisy. But as can be seen from the SLAM position estimate it contains enough information to correct for the severe deficiencies in the dead reckoning. Looking for correspondences between the map and Fig. 4, we note that there is a concentration of yellow pixels roughly corresponding to the L-shaped wall in the upper right part of Fig. 4. There is also a blob of red pixels corresponding to the box in front of the L-shaped wall. The red arcs along the edges of the map are a result of the wide opening angle of the sensor; we simply have no other information other than that there somewhere along this arc exists an object. It is not possible to locate the object more precisely from the observations made.

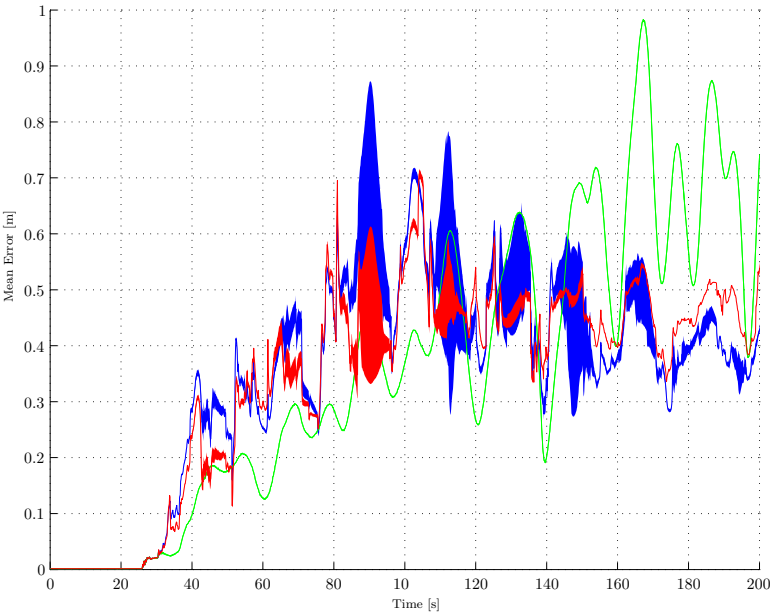

Figure 5. Mean error and standard deviation for 40 realizations. The red curve shows the smoothing-based SLAM estimate, the blue curve shows the filtered SLAM estimate, and the green curve is the dead-reckoning estimate. The smoothing-based SLAM estimate is more consistent than both forward-filter SLAM and dead reckoning. Notice that initially neither the filtered nor the smoothed SLAM estimate outperforms the odometry, which is natural since initially there is no map estimate that can correct for the error in odometry. At around $t = 210$ a wheel slip occured that caused large odometry divergence. Therefore the data set is truncated at $t = 200$ in order to more clearly visualize the differences between the filtering and smoothing.

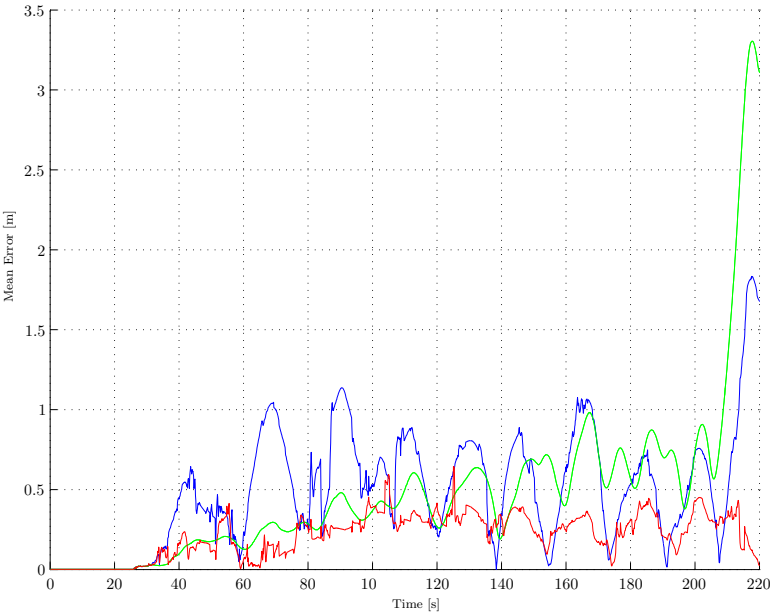

Figure 6. Mean error for a single realization with same notation as in Fig. 5. The poor performance of the filtered estimates can be explained by that the map has converged with a bias compared to the real world, an issue which is largely corrected for by using smoothing instead. This data set is slightly longer than the one presented in Fig. 5 to clearly demonstrate the presented methods' abilities to handle severe odometry errors. Note the large error also for the filtering SLAM estimate (blue), something which the smoothing-based SLAM managed to correct for.

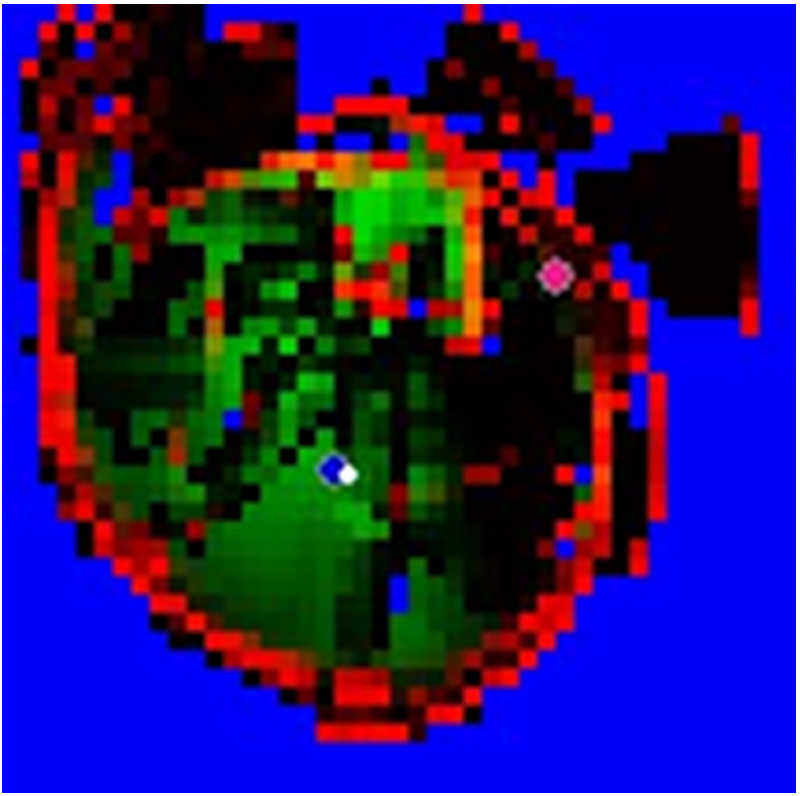

Figure 7. The map corresponding to the last time step in Fig. 6, generated using the measurement model described in Sec. 3 but with the known positions from the VICON system. Since the position is known, this corresponds to the mapping part of the problem. Overlaid on this map is the position estimates; the white dot is the VICON reference, the blue dot close to the white is the SLAM estimate, and the pink dot is the dead reckoning. As can be seen the SLAM estimate has converged to the VICON ground-truth.

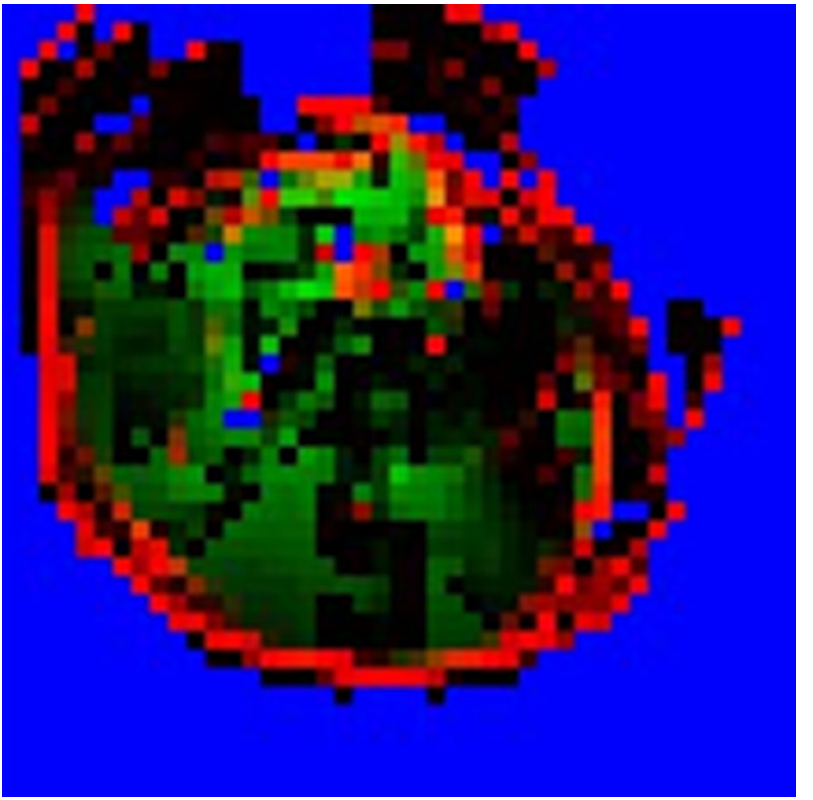

Figure 8. The map corresponding to the last time step in Fig. 6, generated using the full SLAM algorithm as described in this article. Compared to the map in Fig. 7 this one is more noisy, and not all straight walls are preserved. But as can be seen from the accuracy of the position estimate (Figs. 5–7), this information is enough to compensate for the inaccuracy in the odometry. Further, it is also able to compensate for the large wheel slip that occurred around time $t = 210$.

7. Conclusions

We presented a novel approach to the SLAM problem using an ultrasound sensor. The method uses a particle filtering inspired technique paired with a Rao-Blackwellized particle smoother. The experimental results show that the smoothing gives a substantial robustness improvement and increased predictability of the algorithm. Further, the results show that the algorithm is able to converge to the true position even for scenarios with extreme odometry uncertainty.

Acknowledgments

The authors are members of the LCCC Linnaeus Center and the ELLIIT Excellence Center at Lund University. This work was supported by the Swedish Foundation for Strategic Research through the project ENGROSS. High accuracy reference measurements are provided through the use of the VICON real-time tracking system courtesy of the UAS Technologies Lab, Artificial Intelligence, and Integrated Computer Systems Division (AIICS) at the Department of Computer and Information Science (IDA), Linköping University, Sweden. See the URL http://www.ida.liu.se/divisions/aiics/aiicssite/index.en.shtml

References

Ahn, S., J. Choi, N. L. Doh, and W. K. Chung (2008). "A practical approach for EKF-SLAM in an indoor environment: fusing ultrasonic sensors and stereo camera". *Auton. Robots* **24** (3), pp. 315–335. ISSN: 0929-5593.

Anderson, B. D. O. and J. B. Moore (1979). *Optimal Filtering*. Prentice Hall, Englewood Cliffs, NJ.

Andrieu, C. and A. Doucet (2000). "Particle filtering for partially observed Gaussian state space models". *J. R. Statist. Soc. B* **64**, pp. 827–836.

Burgard, W., D. Fox, H. Jans, C. Matenar, and S. Thrun (1999). "Sonar-based mapping with mobile robots using EM". In: *Proc. of the 16th Intl. Conf. on Machine Learning*.

Doucet, A., S. Godsill, and C. Andrieu (2000). "On sequential Monte Carlo sampling methods for Bayesian filtering". English. *Statistics and Computing* **10**:3, pp. 197–208. ISSN: 0960-3174. DOI: 10.1023/A:1008935410038. URL: http://dx.doi.org/10.1023/A:1008935410038.

Durrant-Whyte, H. and T. Bailey (2006). "Simultaneous localisation and mapping (SLAM): part 2". *IEEE Robot. Autom. Mag.* **13**, pp. 108–117.

Eliazar, A. and R. Parr (2003). "DP-SLAM: fast, robust simultaneous localization and mapping without predetermined landmarks". In: *Proc. 18th Int. Joint Conf. on Artificial Intell.* Morgan Kaufmann, pp. 1135–1142.

Grisetti, G., C. Stachniss, and W. Burgard (2005). "Improving grid-based SLAM with Rao-Blackwellized particle filters by adaptive proposals and selective resampling". In: *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*. Barcelona, Spain, pp. 2443–2448.

Grisetti, G., C. Stachniss, and W. Burgard (2007). "Improved techniques for grid mapping with Rao-Blackwellized particle filters". *IEEE Trans. Robot.* **23**, pp. 34–46.

Hähnel, D., W. Burgard, D. Fox, and S. Thrun (2003). "A highly efficient FastSLAM algorithm for generating cyclic maps of large-scale environments from raw laser range measurements". In: *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*. Las Vegas, NV, pp. 206–211.

Inc., M. (2012). *XL-MaxSonar-EZ4 Datasheet*. URL: http://www.maxbotix.com.

Kwak, N., I. K. Kim, H. C. Lee, and B. H. Lee (2007). "Analysis of resampling process for the particle depletion problem in FastSLAM". *Robot and Human interactive Communication, 2007. RO-MAN 2007. The 16th IEEE Int. Symposium on*, pp. 200–205.

Leonard, J. J., H. F. Durrant-Whyte, and I. J. Cox (1992). "Dynamic map building for an autonomous mobile robot". *Int. J. Rob. Res.* **11** (4), pp. 286–298. ISSN: 0278-3649.

Lindsten, F. and T. Schön (2011). *Rao-Blackwellized Particle Smoothers for Mixed Linear/Nonlinear State-Space Models*. Tech. rep. URL: `http://user.it.uu.se/~thosc112/pubpdf/lindstens2011.pdf`.

Nordh, J. and K. Berntorp (2012). "Extending the occupancy grid concept for low-cost sensor based SLAM". In: *10th International IFAC Symposium on Robot Control*. Dubrovnik, Croatia.

Nordh, J. and K. Berntorp (2013a). *NXT SLAM*. URL: `http://www.control.lth.se/Staff/JerkerNordh/nxt%5C_slam.html`.

Nordh, J. and K. Berntorp (2013b). *pyParticleEst - A Python Framework for Particle Based Estimation*. Tech. Report ISRN LUTFD2/TFRT--7628--SE. Department of Automatic Control, Lund Univ., Sweden.

Rencken, W. (1993). "Concurrent localisation and map building for mobile robots using ultrasonic sensors". In: *Proc. of the 1993 IEEE/RSJ Int. Conf. on IROS '93*. Vol. 3, 2192–2197 vol.3.

Särkkä, S., P. Bunch, and S. J. Godsill (2012). "A backward-simulation based Rao-Blackwellized particle smoother for conditionally linear Gaussian models". In: *Proc. of SYSID 2012*. Vol. 16. Brussels, Belgium, pp. 506–511.

Schön, T., F. Gustafsson, and P.-J. Nordlund (2005). "Marginalized particle filters for mixed linear/nonlinear state-space models". *Signal Processing, IEEE Transactions on* **53**:7, pp. 2279–2289. ISSN: 1053-587X. DOI: 10.1109/TSP.2005.849151.

Siciliano, B. and O. Khatib, (Eds.) (2008). *Springer Handbook of Robotics*. Springer, Berlin, Heidelberg. ISBN: 978-3-540-23957-4.

Smith, R., M. Self, and P. Cheeseman (1990). "Estimating uncertain spatial relationships in robotics". In: Springer-Verlag New York, Inc., New York, NY, pp. 167–193. ISBN: 0-387-97240-4.

Thrun, S. (2002). "Robotic mapping: a survey". In: Lakemeyer, G. et al. (Eds.). *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann.