Derivative free filtering using Kalmtool

Enis Bayramoglu, **Søren Hansen**,
Ole Ravn,
DTU Electrical Engineering
The Technical University of Denmark
eba, sh, or@elektro.dtu.dk

Niels Kjølstad Poulsen
DTU Informatics
The Technical University of Denmark
nkp@imm.dtu.dk

Abstract – *In this paper we present a toolbox enabling easy evaluation and comparison of different filtering algorithms. The toolbox is called Kalmtool 4 and is a set of MATLAB tools for state estimation of nonlinear systems. The toolbox contains functions for extended Kalman filtering as well as for DD1 filter and the DD2 filter. It also contains functions for Unscented Kalman filters as well as several versions of particle filters. The toolbox requires MATLAB version 7, but no additional toolboxes are required.*

Keywords: Simulation, Software tools, State estimation, Kalman filtering, Derivative free filtering Nonlinear systems

1 Introduction

Kalmtool is a collection of MATLAB implementations for simulation and estimation in connection with nonlinear dynamic systems. It can also be used as a simulation platform in mobile robotics or for other specialized domains and the toolbox development has been driven by these applications.

Many filters are implemented in the toolbox. Currently the toolbox implements the classical filters like Kalman filter, extended Kalman filter, unscented Kalman filter and some variants of particle filters. Past research has shown that the extended Kalman filter is sometimes inconvenient to use. Small modifications of the application sometimes had serious implications on the EKF implementation. The problem is the inherent linearisation of the system model, which sometimes can be hard to determine and even implement. Small changes made to the model means finding the derivatives again. Especially when temporarily including model parameters in the state vector for estimating them along with the states, it requires much extra work. Combined with the fact that the Jacobians are not allways defined in all points and the linearised model sometimes describes the model poorly, two alternative filters were developed. The *Divided Difference filters* DD1 and DD2 [2] are filters for state estimation in nonlinear systems. They are preferable due to the fact that no linearised model is needed, and the estimate is as good as the well known Kalman filter. The only drawback of the algorithms is an increased computation time. Several other newer non-linear filters have similar properties, and several of them are implemented in the toolbox for comparison.

Several toolboxes for non-linear filtering have been developed over the years and new ones are still being developed. ReBEL (Recursive Bayesian Estimation Library) [5] is a MATLAB toolkit of functions and scripts, designed to facilitate sequential Bayesian inference (estimation) in general state space models. In the area of mobile robot navigation, the CAS Robot Navigation Toolbox [1] is a tool for doing off-line off-board localisation and SLAM. The design of the CAS toolbox decouples robot model, sensor models, features and algorithms used giving the user the possibility to modify the toolbox. Initiatives like the OpenSLAM project [4], which gathers algorithms and toolboxes, show that the field is maturing.

Kalmtool 4 is a reimplementation of the algorithms in Kalmtool 3 with a new structure. The aim of the current version is to make the software modular and to ease the process of adapting the toolbox to new estimation tasks. Using a more standardised method for calling the internal functions, the recombination and the reuse of functions have been simplified. The implemented algorithms are described in more detail in [3].

The paper is organised as follows; Firstly, the overall design philosophy behind the modular platform is described. This covers the system input/output, estimation algorithm switching and user extension and modification. The estimation algorithms treated are the extended and unscented Kalman filters, divided difference filters and different particle filters. Secondly, the toolbox is benchmarked using a nonlinear system and numerical experiments are performed. The example study shows different implementations and results obtained using simulation in the toolbox.

2 The Platform and the Toolbox

The overall design philosophy has been to focus on the creation of a simple, transparent, yet powerful platform making life easier for the application developer as well as the algorithm developer.

Transparency overcomes the barrier that is often experienced when using tools, that at first sight seem very user-friendly, but when used on real problems become difficult to handle, due to the inherent complexity.

The approach taken uses MATLAB as a numerical and graphical basis for developing the platform. Execution can be done using two modes. Script execution, where the system is modelled using M-files and Graphical execution, where the program flow is modelled using Simulink, but the underlying models are based om M-files. Simulink provides a shorter path to implementation and eases understanding. Tools like Realtime Workshop makes it simple to achieve fast prototyping and use real data for comparison.

The philosophy of the Kalmtool toolbox is to provide a more open structure (see fig. 1), which is easier to use and which enables the user to investigate the inner workings of the estimation algorithms, which are listed in table 2. The functions are made to work in an on-line setting, one timestep - one update, though this does not prohibit its use in offline environments.

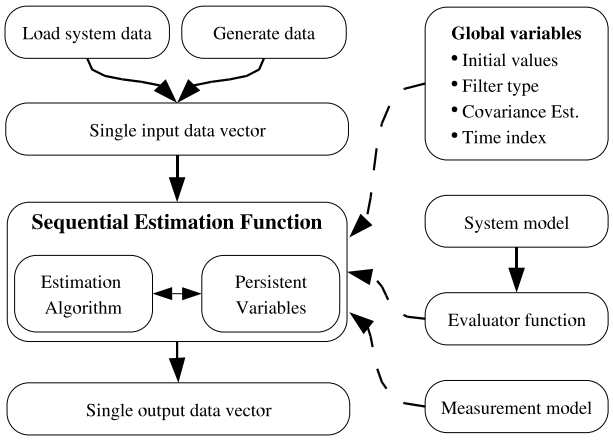

Figure 1: *The structure of Kalmtool 4.*

The toolbox is modular by design and its configuration is handled by a main file connected to each simulation or data set. Module switching is handled by a subdirectory separation of methods and switching between systems by means of the *MATLAB path*. The main file is typically placed in the main directory, and is also used to document the system being investigated.

For each system a subdirectory is used to contain the models and real data, if it is available. In this subdirectory 10 MATLAB m-files (see table 1) describe the complete system and the model to be used for the estimator. The files are standardised for all systems, making it possible to easily switch between estimators.

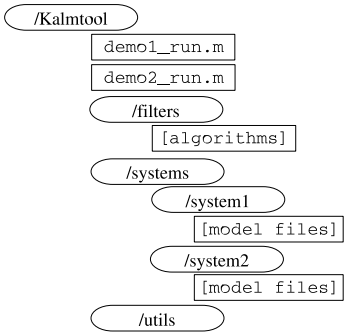

Figure 2: *The Kalmtool 4 directory structure layout.*

File	Content
initt.m	Model initialisation (x0, dimensions e.g.)
	Process model (for simulation)
stateproc.m	State update
measproc.m	Measurement model
outproc.m	State output model
	Estimation model
statemodel.m	State update
measmodel.m	Measurement model
outmodel.m	State output
statelin.m	Linearised state update
measlin.m	Linearised measurement model

Table 1: Model files in Kalmtool 4.

The toolbox operates with two different system models. The detailed process model denoted with *proc* and the estimator model denoted *model*. The process model is used for the simulation of the real system and data generation. The estimator model is used for the estimation algorithms, and can be chosen as a different model than the process model. Decoupling the two models makes it possible to investigate robustness properties and the effect of model mismatch. This also enables the use of a simpler model for estimation. The *proc* model is omitted if real data is used. From table 1 it is evident that there is also a linearised model. This is the Taylor linearisation of the estimator model, for filters needing that.

Using script based execution, the file execution order of the model is best illustrated graphically. As the toolbox is highly customisable, not all function calls are depicted in fig. 3.

In order to use Simulink for structuring the model and visualising the data, a Simulink block diagram is placed in the system directory. The diagram structures data input/output and facilitates the data visualisation. To run either a script based or Simulink based estimation, the main file preprocesses the input data, sets up the environment and executes the estimation process. Using Simulink, the systems treated can be both continuous or discrete. Examples of both

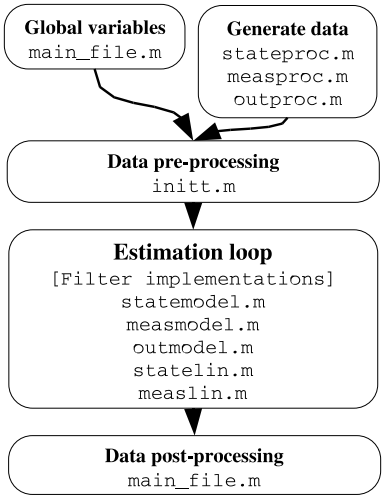

Figure 3: *Kalmtool 4 script execution function calls.*

are supplied with the toolbox.

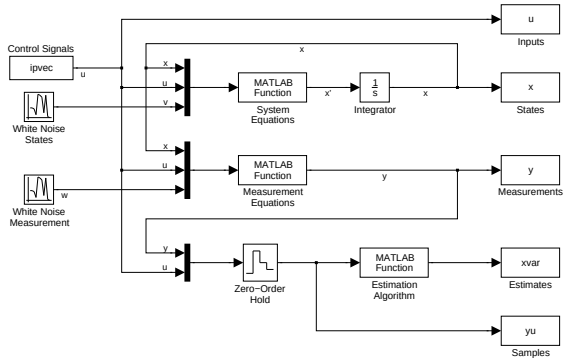

Figure 4: *The Simulink layout of a continuous system.*

As seen in fig. 4, the user can easily add new algorithms into the platform, by modifying the MATLAB function in the *Estimation* block and change the system by modifying the *System* and *Measurement* MATLAB blocks.

The toolbox currently consists of 13 filter implementations. Adding filters to the platform is easy, using a previous filter as template. The main file is then expanded with the option to execute the filter, keeping the toolbox backwards compatible.

3 Implementation

In order to use the filter routines, we first need a state space model in the form:

$$x_{k+1} = f(x_k, u_k, v_k)$$

and a measurement model in the form:

$$y_k = g(x_k, e_k)$$

Filter impl.	Filter type
dd1.m	Divided difference first order
dd2.m	Divided difference second order
ekf.m	Extended Kalman filter
klm.m	Kalman filter (time varying)
klms.m	Kalman filter (stationary)
lrkf.m	Linear regression Kalman filter
scd_ekf.m	Scaled Unscented Kalman filter
skfb.m	Kalman Particle filter
skfp.m	Kalman Particle Filter (projection based)
sslq.m	Steady state LQ controller
ukf.m	Unscented Kalman filter

Table 2: Estimation functions available in Kalmtool 4.

The noise covariance matrices

$$\begin{aligned} Q &= \mathbf{E}\Big\{v_k v_k^T\Big\} \\ R &= \mathbf{E}\Big\{e_k e_k^T\Big\} \end{aligned}$$

and the initial estimates of state and covariance matrix

$$\begin{aligned} \bar{x}_0 &= \mathbf{E}\Big\{x_0\Big\} \\ P_0 &= \mathbf{E}\Big\{(x_0 - \bar{x}_0)(x_0 - \bar{x}_0)^T\Big\} \end{aligned}$$

are also needed. Other required items are the input-output datasets

$$\begin{aligned} y &= \{y_0, y_1, y_2, \ldots\} \\ u &= \{u_0, u_1, u_2, \ldots\} \end{aligned}$$

For the application of the extended Kalman filter (EKF), we must also derive the linearized state and observation equations:

$$\begin{aligned} x_{k+1} &= d_x + Ax_k + G_x v_k + hot \\ y_k &= d_m + Cx_k + G_m e_k + hot \end{aligned}$$

where

$$\begin{aligned} A &= \left.\frac{\partial f(x, u_k, \bar{v}_k)}{\partial x}\right|_{x=\hat{x}_k} \quad G_x = \left.\frac{\partial f(\hat{x}_k, u_k, v)}{\partial v}\right|_{v=\bar{v}_k} \\ C &= \left.\frac{\partial g(x, \bar{w}_k)}{\partial x}\right|_{x=\bar{x}_k} \quad G_m = \left.\frac{\partial g(\bar{x}_k, w)}{\partial w}\right|_{w=\bar{w}_k}. \end{aligned}$$

This information has to be specified in terms of MATLAB functions. These functions must conform to a particular structure, which is discussed below. In order to exemplify we will illustrate this for a second order non-linear model given by the process equation:

$$\begin{aligned} x_k^1 &= a\frac{x_k^1}{1 + (x_k^1)^4} + bx_k^2 + v_k^1 \\ x_k^2 &= -bx_k^1 + ax_k^2 + v_k^2 \end{aligned} \tag{1}$$

and the measurements equation:

$$y_k = sin(\omega x_k^2) + e_k \tag{2}$$

Furthermore:

$$v_k^1, v_k^2, e_k \sim \mathcal{N}(0, 0.01) \tag{3}$$

and otherwise uncorellated. The parameters are $a = 1.01$, $b = 0.3, \omega = 3$.

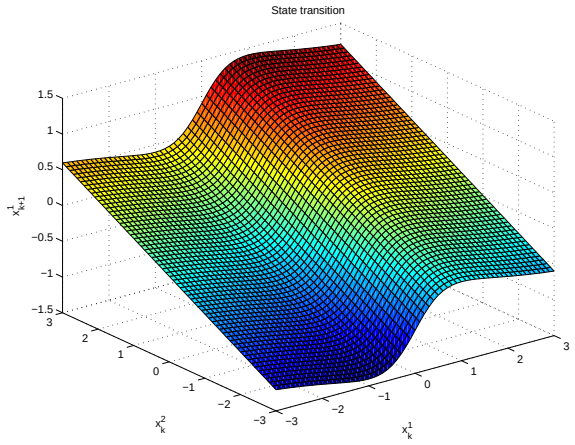

Figure 5: State transition given by (1)

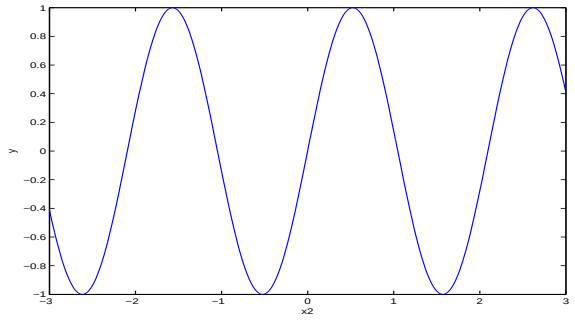

Figure 6: Measurements mapping as given by (2)

3.1 M-Functions

If the purpose is to to run one of the derivative free filters two functions have to be written. One (statemodel.m) should contain the state equation, and the other (measmodel.m) should contain the output equation. If the purpose is to use the EKF, it is necessary to write an additional function that specifies the linearization of the two equations.

In a development environment there is often a difference between the system used for generating the data and the models used in the filtration. For that reason, two procedures (stateproc.m and measproc.m) might exist.

The basic procedure supplying structural information is initt.m which has the following structure:

```
function varargout=initt(typ)
% Usage:
%
% Initialization:
% initt(0)
% Getting true system dimensions:
% [nx,nu,nv,ny,ne]=initt(1)
% Getting true system statistics:
% [x0,P0,vm,R1,em,R2]=initt(2)
% Getting model dimensions:
% [nx,nu,nv,ny,ne]=initt(1)
% Getting model statistics:
% [x0,P0,vm,R1,em,R2]=initt(2)
global nx nu nv ny ne
global Gx Gy
global em vm R1 R2
global P0 x0
global a b w
switch typ
case 0,
 nx=2; nu=0; ny=1; nv=2; ne=1;
 % State Space description
 Gx = eye(2); Gy = [1];
 R1=eye(nv)*0.01; R2=eye(ne)*0.01;
 vm=zeros(nv,1);
 em=zeros(ne,1);
 P0=eye(nx);
 x0=zeros(nx,1);
 a = 1.01;
 b = 0.3;
 w = 3;
case {1,3},
 varargout={nx; nu; nv; ny; ne};
case {2,4},
 varargout={x0; P0; vm; R1; em; r2};
end
```

Notice that the information is passed through the output arguments as well as through globals. The latter is of course a matter of taste.

3.1.1 The State Equation

The state process in (1) maps the present state, the input and the process noise to the next state. The implementation in the state model should reflect that. In derivative free filters a swarm of particles, or sigma points, are mapped through the process model. Therefore, statemodel.m should be able to process a collection of particles. The state model implementation for (1) is given below.

```
function x=statemodel(U)

global nx nu nv
global Gx
global a b
xo=U(1:nx,:); % Old states (swarm)
u=U(nx+1:nx+nv,:); % input swarm
v=U(nx+nu+1:end,:); % noise swarm

x(1,:)=a*xo(1,:)./(xo(1,:).^4+1)+b*xo(2,:);
x(2,:)=-b*xo(1,:)+a*xo(2,:);
```

```
x=x+Gx*v;
% ---------------------------------------
```

The input argument `U` is a matrix containing the sigma points as columns. In a column, the first `nx` elements are the state variables, the next `nu` elements are the control (or the known) inputs and the last `nv` elements are the process noise elements. In our example in (1), there are no inputs ($n_u = 0$). The output argument (`x`) is a matrix containing the image of the sigma points.

If there is no difference between the system generating the data and the estimation model, the `stateproc` can simply be:

```
function x=stateproc(U)
x=statemodel(U);
```

3.1.2 The Measurement Equation

The measurement equation is written in an m-function with a similar format:

```
function y=measmodel(U)

global Gy nx w

x=U(1:nx,:); e=U(nx+1:end,:);
y(1,:) = sin(w*x(2,:))+ Gy*e;
% ---------------------------------------
```

which also applies for the measurement process (true data generating process when performing simulations):

```
function y=measproc(U)
y = measmodel(U);
```

The functions should take one argument: the sigma points (or swarm) in the same format as the state process.

3.2 Linearization of the Equations (EKF only)

When using the EKF, it is necessary to write separate functions that contain the linearization of the two equations. The functions must have the following format (exemplified by (1)):

```
function [A,B,Gx]=statelin(U)

global a Gx
B= [];

x=U(1:nx,:);
v=U(nx+1:end,:);

x = x(1);
A = [a/(x^4+1)-4*x^4/(x^4+1)^2    b
     -b                           a];
% ---------------------------------------
```

The linearization of the measurements equation (2) follows a similar format as the state equation.

```
function [C,D,Gy]=measlin(U)

global Gy w

D = [];
C = [0 w*cos(w*U(2,1))];
% ---------------------------------------
```

3.3 Running the Filters

When input and observation data are available, and the appropriate m-functions have been written, it is straightforward to run the filters. The DD2 filter is (in an off-line application) invoked in the following way:

```
[xht,St,yht,rest,Rt]=dd2(flag,pred,yt,ut)
  or
[xht,St,yht,rest,Rt]=dd2(flag,pred,[yt ut])
```

The other filters follow a similar syntax. The input arguments are the measurements (`yt`) and the control inputs (`ut`) if they exist. If `pred` has the value `1`, a predictive estimate ($\hat{x}_{t|t-1}$) is produced. Otherwise an ordinary ($\hat{x}_{t|t}$) is in the output argument list. In an off-line application `flag` has to be 3. (Other values of `flag` is reserved for recursive or on line applications. As a matter of fact, the off-line is implemented as a successive use of an on-line version.)

The output argument is the state estimate, `xht` (as a function of time coloumn wise), the residual sequence (`rest`), the predicted output (`yht`), the standard deviation (`St`) (square root of the diagonal elements of the covariance matrix) of the estimation error and the standard deviation (`Rt`) of the prediction error.

4 Numerical Simulations

Kalmtool is used to simulate the system given in (1). EKF and DD2 filters are then applied to the measurement time series obtained from the simulation. The results are given in figs. 7 and 8.

The actual non-linear process used to simulate the state and the measurement time series is the same as the model used for the filters, although Kalmtool allows otherwise. The initial states are independent and have the distribution $\mathcal{N}(0, 1)$. Running the simulation repeatedly shows that for this system, DD2 is faster to recover the initial state (as in the plotted instances), while they display a similar performance after approaching the true state.

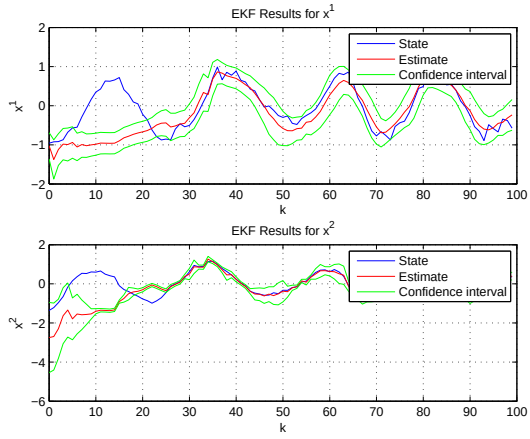

Figure 7: A simulation result using EKF to estimate the state of the system given in (1). The upper plot shows the evolution of the actual state through samples(blue), along with the estimate(red) and the confidence interval calculated from the estimated covariance(green).

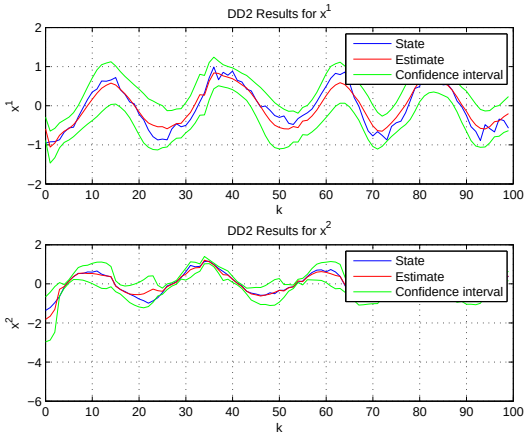

Figure 8: A simulation result using DD2 to estimate the state of the system given in (1). DD2 is run on the same measurement time series as in fig. 7.

5 Conclusion

In this paper we have presented the toolbox Kalmtool which a set of MATLAB tools for state estimation for nonlinear systems.

It contains estimation methods for extended Kalman filtering, two types of Divided Difference filters, unscented, standard and scaled Kalman filters as well as several versions of particle filters.

Examples have been presented to illustrate the methods and treatment of simulated data. The results are based on the Divided Difference filter (DD2) and the extended Kalman filter (EKF) and show the simple yet powerful way the toolbox enables evaluation and comparison of filter performance.

The toolbox is available for download at:

http://server.elektro.dtu.dk/www/or/kalmtool/

References

[1] Kai O. Arras. The cas robot navigation toolbox, quick guide. Technical report, CAS, KTH, January 2004.

[2] M. Nørgaard, N.K. Poulsen, and O. Ravn. New developments in state estimation for nonlinear systems. *Automatica*, 36(11):1627–1638, November 2000.

[3] Thomas Hanefeld Sejerøe, Niels Kjølstad Poulsen, and Ole Ravn. A new evaluation platform for navigation systems. In *16'th IFAC World Congress, Prague, Czech Republic*, pages Tu–A18–TO/2, ID: 03891, 2005.

[4] Cyrill Stachniss, Udo Frese, and Giorgio Grisetti. Open slam - give your algorithm to the community. Web site - www.openslam.org, October 2008. Authors assosiated with University of Freiburg.

[5] Rudolph van der Merwe. Quick-start guide for rebel toolkit. Technical report, Oregon Health and Science University, February 2004.