FILTER : development of an off-line filter module for PRIMAL

Schoormans, C.J.C.

Award date:
1996

NR-1941 (7-5-1996) C.J.C. Schoormans

FILTER: Development of an off-line filter module for PRIMAL

C.J.C. Schoormans

AFSTUDEERVERSLAG

Begeleider: Ir. W.A. Renes
Contactdocent: Dr. Ir. R.J.P. van der Linden
Afstudeerhoogleraar: Prof. Dr. Ir. J.J. Kok

Met dank aan: Walter, Ruud, Frank, Joost, Wim, Thanh, Inge, De vakgroep Systeem & Regeltechniek (TUE-NR), Piet, Sjaan, Adriaan en de rest van familie en vrienden.

Summary

This report describes the development of an off-line filter module for PRIMAL (Package for Real-time Interactive Modelling, Analyses and Learning). This filter module is to be used for the reduction of disturbances that are present in signals collected during experiments on practical (industrial) processes.

This filtering is to be done before process identification, which is an iterative procedure in which knowledge about the dynamic behaviour of the process is collected by means of experiments, signal analysis, estimation and validation.

In order to test the usefulness of the developed filter module, filtering and identification have been done on process data of a glass feeder.

It is concluded that data conditioning of process data before identification is often necessary and that the developed filter module is a useful tool for this data conditioning.

Contents

List of Symbols

u_n	The nth sample of the signal to be filtered
y_n	The nth sample of the filtered signal
b_j	The non-recursive filter parameters
M	The non-recursive filter order
$X(z)$	z-transform of the sequence x_n
$A(z)$	z-transform of the sequence a_n
$B(z)$	z-transform of the sequence b_n
$H(z)$	z-transform
$H(s)$	z-transform
$H(\omega)$	The frequency response of $H(z)$
ω	The frequency
ω_S	The sample frequency
T	The sample time
a_j	The recursive filter parameters
N	The recursive filter order
δ_1	Delta passband
δ_2	Delta stopband
$\phi_{Causal}(\omega)$	Phase shift of the causal filter
Wd	Median width

x	The signal
x_{Trend}	A trend of the signal
S	The shaving strength
σ	A standard deviation

$A(z), B_i(z)$ *and* $F_i(z)$	The z-transformation of the model parameters
$u_i(z)$	The input signals of the model
$y(z)$	The output signal of the model
$C(z)$ *and* $D(z)$	The z-transformation of the noise filter
$e(z)$	The z-transformation of the noise
nA, nB, nC, nD, nF	Model size/order

List of Terms

anti-causal filtering	Filtering backwards in time
Bilinear Transformation	Method to convert continuous time designed filters to the discrete time
C++	Programming language used in the development of Filter
causal filtering	Filtering forwards in time
disturbance	Signal component that has no correlation with the process
DLL	Dynamic Link Library
Filter	The developed filter program
FIR-filter	Finite Impulse Response filter
identification	Estimate of a model based on experimental analysis
IIR-filter	Infinite Impulse Response filter
Nyquist frequency	Half the sample frequency
outlier	Spike: Large, temporarily (a few samples long) disturbance in a signal
PDLGBOX	C++ dialog base class in PRIMAL
PRBNS	Pseudo Random Binary Noise Signal
PRIMAL	Package for Real-time Interactive Modelling, Analyses and Learning
spike	Outlier: Large, temporarily (a few samples long) dsturbance in a signal
symmetric filter	Filter with a symmetric impulse response around zero
trend	Slow signal drift/low frequency disturbance that has no correlation with the process
window technique	Method to correct for the truncating error in FIR-filter design

1. Introduction

1.1 Preliminaries

This report is the result of my work at TNO-TPD. This work has been performed to obtain a Master of Science degree in Physical Engineering.

The goal of my work was to develop an off-line filter module. This filter module is to be used for the reduction of disturbances that are present in signals collected during experiments on practical (industrial) processes.

This filter module is (to be) part of the PRIMAL package (Package for Real-time Interactive Modelling, Analyses and Learning), which is a joined project of TNO-TPD and the System & Control group of the Physics Department at the University of Technology in Eindhoven. PRIMAL supports experimenting, data acquisition, signal processing, signal analysis, system identification, modelling and controller design.

This report describes the signal processing techniques that are used in the filter module, called Filter and a case study with process data of a glass feeder on which Filter has been tested. Further the implementation is described.

1.2 Signal Preparation before Identification

Mathematical models of the dynamic behaviour of a process can be derived in two ways:

1. One possibility is to derive a theoretical model from basic physical laws and construction data. This analytical modelling, of a complex real-world process, is often very difficult, because important process coefficients are very often hard to determine.
2. The other possibility is to estimate a model based on experimental analysis (identification). The signals of interest of an existing process are measured. Using an identification method a model, that describes the input-output behaviour of the process, may be obtained.

Of course a combination of these methods can be used.

The quality of a model obtained with an identification method will for the greater part be determined by the disturbances present in the signals. Disturbances with a bad influence on the quality of a model are, for example, trends and outliers. A trend is a slow signal drift/low frequency component that has no correlation with the process. An outlier or spike is a large, temporarily (a few samples long) disturbance in a signal, for example caused by a sensor failure, that has no correlation with the process. Further there are things like time delays and offsets, which are no disturbances, but may also lead to a bad quality of the model. Because of the disturbances negative influence on the quality of the models and because almost all industrial process data will be corrupted by these types of

"noise", it is advisable to try to reduce these disturbances as much as possible. For this purpose dedicated signal processing techniques have been developed.

At TNO-TPD (and some industrial concerns) the measurements on processes, model identification and controller design and implementation are done in/with PRIMAL. This is visualised in Figure 1.

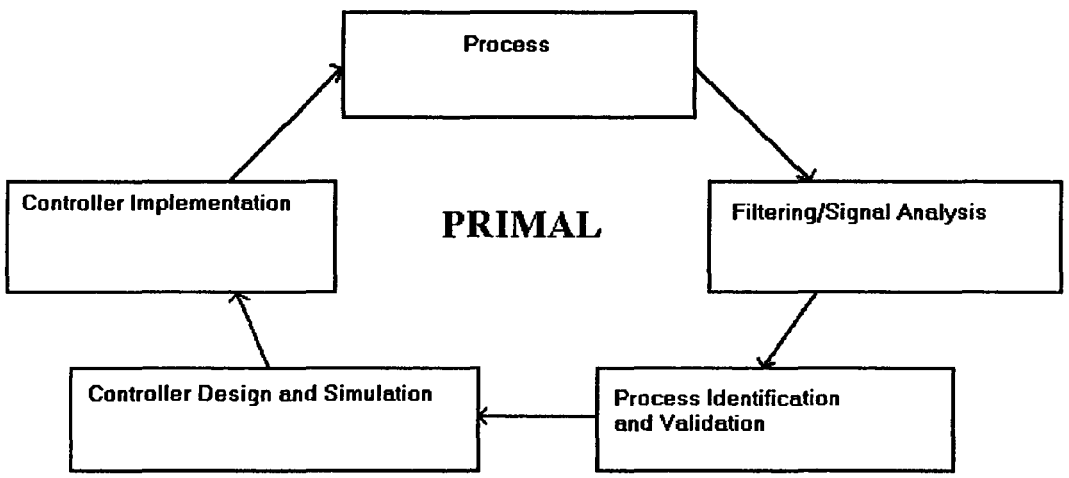

Figure 1: Role of PRIMAL in control design and implementation.

Data is collected on a process. After that the data is filtered and signal analysis is performed. With this data a process model can be identificated and validated. With the model found a controller can be designed and simulated on the collected process data. Then the controller can be implemented on the process. After the controller is implemented these steps may be taken again in order to improve the controller.

Some of these steps can also be done by programs like MATLAB but to avoid problems with data transportation between packages, and because programs like MATLAB do not present the filters in dialogs, which is thought to make a program more user friendly, own filter module has been developed for PRIMAL. Further MATLAB is very expensive for the industry and can not be used for implementation of controllers on industrial processes. So when a program which is not a part of PRIMAL is used, not all the steps taken in controller design and implementation can be done with the same program.

1.3 Specifications

The filter module to be developed must satisfy specifications on several levels. The first and most important level of specifications are the filter specifications. These specifications are (the disturbances that can be removed are):

- Time shifting, to correct for time delays the signals must be shifted in time.
- Extrapolation, to cope with starting problems of filters the signals must be extrapolated.
- "Normal" filtering, to remove trends and high frequency noise.

- Automatic and manual outlier removal and repair. To remove and repair outliers and other local (a few sample long) disturbances.
- Mathematical operations like scaling, offset correction and (non-linear) mathematical functions, like sinus and the square root.
- Data reduction; decimation.

The second level of specifications are the specifications on a software level. These are:

- Implementation in C++.
- Filter must be a frame work, so new filter algorithms can be implemented easily without any need for the source of or linking with the main program.
- The filters must be implemented independently from the main program, so they can be easily used in other programs.

The third level of specifications comes from the user interface:

- The filters must be present in dialogs, because this is thought to be user friendly.
- The result of the filter actions must be automatically visualised, so the user has a direct feedback and can decide, possibly after some attempts, to accept a filter result.

2. Theory

2.1 Introduction

A set of raw input-output measurement data, collected during experiments on practical (industrial) processes, is seldomly suited for direct use in analysis and identification. All types of disturbances appear in the data. In process modelling these are disturbances like high frequency noise, low frequency trends and outliers. What high and low frequencies, and outliers are depends on the process itself. A good example of a trend is a temperature fluctuation caused by the 24 hour day and night rhythm.

The mentioned disturbances have no correlation with the process, so in theory they should not cause any problems in the estimation of a process model. However the influence of some disturbances in the signal on the estimated parameters of a process model will only tend to zero in the limit case where the number of input and output samples is close to infinity. For example a trend may have significant components with wavelengths far below the test-signal period, and because of the low frequency components present in the input signal, the influence on the estimation will not be neglectable. And it can even be rather large compared to the influence of the process outputs without a trend, when the amplitude of the trend is large. So the trend has to be removed before identification can be performed. A mathematical analysis of this is given in [BAC87].

Another, already mentioned, disturbance is outliers, also known as spikes. There are various reasons for outliers, some of them are:

- Sensor failures
- Loss of data caused by bad wiring (loose contacts)
- Large background disturbances like switching actions of high power process machinery

Most of the time the spikes last from one to some tens of samples. If these spikes are not removed from the signals, they may form an important part of the noise energy. As a consequence the spikes can have a considerable influence on the ultimate model, although they have no relation with the process itself.

2.2 Digital Filter Theory

In this paragraph some filter theory will be treated briefly. Most of it is linear digital filter theory of which a thorough and extensive treatment is given in literature like [JAC86].

2.2.1 Introduction

The main goal of filtering is removing unwanted signal components. For example an outlier, which consists of high frequency components, can be removed with a lowpass filter or with a median filter, which takes the median of a subinterval around each sample. This also removes outliers since the median is not outlier-sensitive. Another unwanted signal component is a trend. Such a trend can be removed with a highpass filter, or by subtracting the lowpass filtered signal from the original signal.

Within the PRIMAL-package signals consist of sampled data. Therefore digital filtering is required. There are many types of digital filters. A well known type is the linear digital filter. There are two main types of linear digital filters:

- Finite Impulse Response (FIR)-filters.
- Infinite Impulse Response (IIR)-filters.

FIR-filters are also known as Moving-Average (All-Zero) filters (MA-filters). IIR-filters as AutoRegressive (All-Pole) Moving-Average filters (ARMA-filters).

2.2.2 Finite Impulse Response (FIR)-filters

In the time-domain a FIR-filter is defined by:

$$y_n = \sum_{j=0}^{M} b_j u_{n-j} \qquad (1)$$

with:

u_n	The n^{th} sample of the signal to be filtered
y_n	The n^{th} sample of the filtered signal
b_j	The non-recursive filter parameters
M	The non-recursive filter order

Or in words: The current (filtered) sample is found by adding the previous$M$ samples, multiplied with a filter parameter. But how can these filter parameters be found? A simple and obvious way to design a FIR filter is just to truncate the infinite Fourier series representation of the desired frequency response. Then the filter parameters are given by the coefficients of the (truncated) infinite Fourier series.

However, truncating of the Fourier series also produces the familiar Gibbs phenomenon, which will be manifested in the frequency response, especially if it is discontinuous. And since all frequency-selective filters are ideally discontinuous at their band edges, simple truncation of the impulse response will often result in an unacceptabel FIR design. To correct for this truncating error a method, called the Window technique, can be used. The basic problem is the abruptness of the truncation, which also can be viewed as the result of trying to obtain too narrow a transition band (the band between passband and stopband). Usage of a window will produce smaller ripples in the frequency response of the filter, at the expense of wider transition bands. The window technique is described in literature like [JAC86].

How to find the filter parameters will become much clearer once the z-transform is treated.

2.2.3 The z-transform

The z-transform $X(z)$ of the sequence x_n is defined by:

$$X(z) = \sum_{n=-\infty}^{\infty} x_n z^{-n} \qquad (2)$$

where z is a complex variable.

So the z-transform $H(z)$ of a FIR filter is:

$$H(z) = B(z) = \sum_{j=0}^{M} b_j z^{-j} \qquad (3)$$

The z-transform is to the discrete-time systems what the Laplace transform is to the continuous-time systems. For instance, the relationship between input and output of a discrete-time system involves multiplication of the appropriate z-transforms. Poles and zeros can be defined from the z-transform and have the same useful role and intuitive appeal as for the continuous-time systems (this does not mean that the conditions for stability are the same in the z-plane as in the s-plane; in the z-plane the poles must lie inside the unit circle for stability). And finally, the frequency response of the system is just as simple derived form the z-transform, as the frequency response in the continuous-time systems from the Laplace transformation. The frequency response is found by replacing z with $e^{j\omega T}$:

$$H(\omega) = H(z)\big|_{z=e^{j\omega T}} \qquad (4)$$

with:

$H(\omega)$	The frequency response of $H(z)$
ω	The frequency
T	The sample time

So given a desired frequency response, the z-transform can be calculated by expressing it in powers of $e^{j\omega T}$ and replacing these powers with z. In this z-transform the filter parameters of the FIR filter can be read of.

But FIR filters aren't the only type of linear digital filters. The other type is Infinite Impulse Response (IIR)-filters.

2.2.4 Infinite Impulse Response (IIR)-filters

In the time-domain a IIR-filter is defined by:

$$y_n = \sum_{j=0}^{M} b_j u_{n-j} - \sum_{i=1}^{N} a_i y_{n-i} \qquad (5)$$

with:

a_j	The recursive filter parameters
N	The recursive filter order

In order to calculate the previous filtered sample, not only the previous M input samples but also the previous N samples of the filtered signal are needed. So the IIR-filter is recursive. It is easy to see that the FIR-filter is just a special case of an IIR-filter.

The z-transform of the IIR-filter is:

$$H(z) = \frac{B(z)}{A(z)} = \frac{\sum_{j=0}^{M} b_j z^{-j}}{1 + \sum_{i=1}^{N} a_i z^{-i}} \qquad (6)$$

With this, given a certain frequency response and filter order, an IIR-filter can be calculated. Most often, transformation techniques are applied to the design of discrete-time filters with classical continuous-time counterparts. The IIR-filters implemented in Filter are all based on classical filter designs and satisfy constraints on the magnitude of the frequency response of the form illustrated in Figure 2 (for the lowpass case).

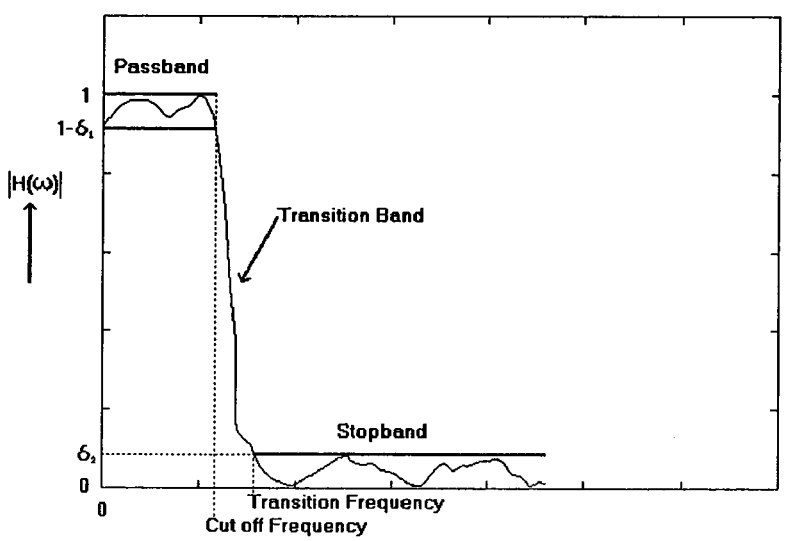

Figure 2: Design Constraints of the IIR-Filters.

That is, in the passband, the frequency response is required to satisfy:

$1-\delta_1 \le |H(\omega)| \le 1$ $\omega \le$ Cut-off Frequency.

and in the stopband:

$|H(\omega)| \le \delta_2$ $\omega \ge$ Transition Frequency.

with the response unspecified in the transition band (Cut off Frequency $\le \omega \le$ Transition Frequency).

Once a filter is designed in the continuous-time domain it can be converted from the s-domain to the z-domain by methods like the impulse-invariant transformation or the bilinear transformation. In Filter the bilinear transformation is used, for it preserves the characteristics of the frequency response.

2.2.5 The Bilinear Transformation

In the discrete-time, the highest frequency that can be retained is half the sample frequency (The Nyquist frequency). But in the continuous-time the highest frequency is infinity. So a transformation from a frequency in the continuous-time to the discrete-time is desired if all the characteristics of the frequency response are to be preserved. This transformation must provide a one-to-one mapping from a frequency in the continuous-time to a frequency in the discrete-time and a one-to-one mapping from the s-plane to the z-plane. There are many transformations which satisfy this need. The bilinear transformation is one of them and is defined by:

$$s = \frac{2}{T}\left(\frac{1-z^{-1}}{1+z^{-1}}\right) \qquad (7)$$

This formula gives the replacement of s necessary for a transformation from the s-plane to the z-plane; a filter is designed in the s-plane and then transformed to the z-plane by replacing each s with (the right side of) the given formula. For example a filter with, in the s-plane, a system function

$$H(s) = \frac{1}{s^2 + \sqrt{2}s + 1}$$

will, after applying the bilinear transformation (with $T = 0.1$ sec.), in the z-plane have a system function

$$H(z) = \frac{0.0675 + 0.1349\,z^{-1} + 0.0675\,z^{-2}}{1 - 1.1430\,z^{-1} + 0.4128\,z^{-2}}$$

The frequency responses of the continuous and discrete filter are shown in Figure 3.

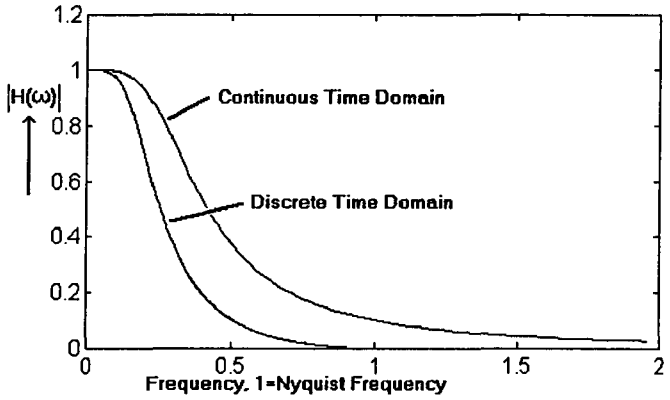

Figure 3: Frequency responses of the continuous and discrete filter.

These frequency responses are very similar, only the frequency axis of the discrete filter is compressed. But how are the frequencies in the continuous-time domain compressed into the discrete-time domain? This compression is given by:

$$\omega_{Discrete} = \frac{2}{T}\tan^{-1}\left(\frac{\omega_{Continuous}T}{2}\right) \qquad (8)$$

So if ω is almost zero the relation is linear. And for $\omega_{Continuous}$ close to infinity, $\omega_{Discrete}$ is almost the Nyquist frequency.

But how does a transformation from a design in the s-domain to the z-domain actually work? First state the desired cut-off frequency in the discrete-time. From this calculate the cut-off frequency in the continuous-time. Now design the filter in the s-domain. Then apply the bilinear transformation. This gives the design in the z-domain.

The bilinear transformation preserves the characteristics of the frequency response. If in the continuous-time the filter is a lowpass filter, the filter will still be a lowpass filter in the discrete-time. Only the cut-off frequency is different. And since the bilinear transformation is algebraic one-to-one, it is also invertible. Unlike other transformations as the impulse-invariance transformation.

2.2.6 Phase Shift Removal: Anti-causal and Symmetric Filters

The filters described in 2.2.2 and 2.2.4 are causal filters. A causal filtered signal depends only on the previous samples of the signal and has no relation with the future signal. A causal lowpass filter can be used in trend removal, by subtracting the causal lowpass filtered signal from the original signal, as is shown in Figure 4.

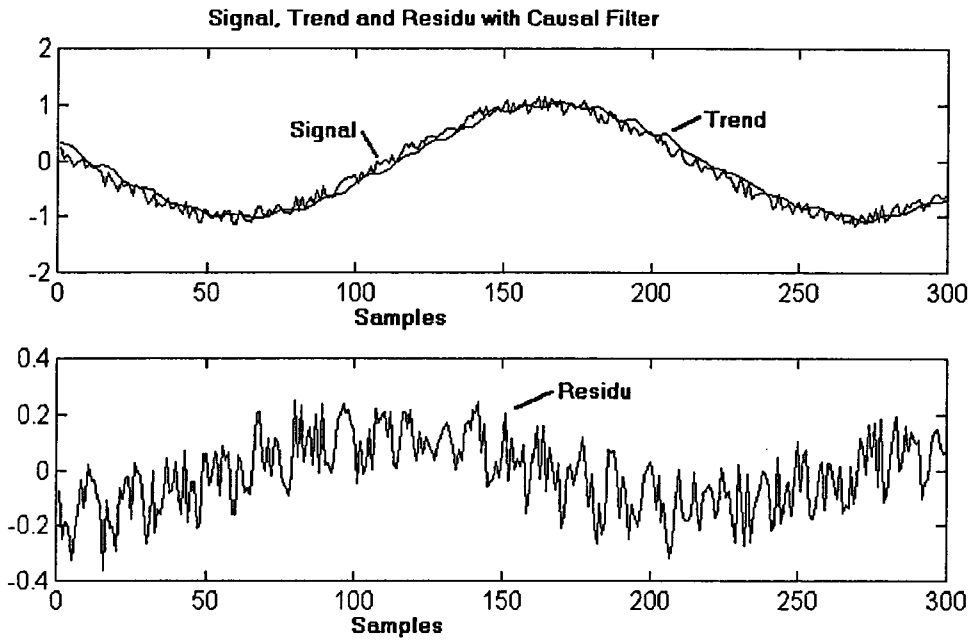

Figure 4: Phase Shift of Causal Filtered Signal

However, filtering of the signal with a causal lowpass filter introduces a phase shift between the original signal and the obtained trend from the signal. Subtraction of this shifted trend therefore does not remove this trend entirely. To overcome this problem the signal can be filtered with a symmetric filter. Such a symmetric filter can be thought of as a combination of a causal filter and a corresponding anti-causal filter.

An anti-causal filter uses samples of the future signal, so it can only be used off-line. In the time-domain the anti-causal IIR-filter is defined by:

$$y_{n,Anti-Causal} = \sum_{j=0}^{M} b_j u_{n+j} - \sum_{i=1}^{N} a_i y_{n+i} \qquad (9)$$

It is easy to see, that the same filtered signal can be obtained by filtering with the causal filter in the opposite direction (backwards). But how are the frequency response of the causal and anti-causal filter related? The causal filter has frequency response:

$$H_{Causal}(\omega) = \left. \frac{\sum_{k=0}^{M} b_k z^{-k}}{1 + \sum_{i=1}^{N} a_i z^{-i}} \right|_{z=e^{j\omega T}} = \frac{\sum_{k=0}^{M} b_k \cos(k\omega T) - j\sum_{k=0}^{M} b_k \sin(k\omega T)}{1 + \sum_{i=1}^{N} a_i \cos(i\omega T) - j\sum_{i=1}^{N} a_i \sin(i\omega T)} \qquad (10)$$

and the corresponding anti-causal filter has frequency response:

$$H_{Anti-Causal}(\omega) = \left.\frac{\sum_{k=0}^{M} b_k z^k}{1+\sum_{i=1}^{N} a_i z^i}\right|_{z=e^{j\omega T}} = \frac{\sum_{k=0}^{M} b_k \cos(k\omega T) + j\sum_{k=0}^{M} b_k \sin(k\omega T)}{1+\sum_{i=1}^{N} a_i \cos(i\omega T) + j\sum_{i=1}^{N} a_i \sin(i\omega T)} = H^*_{Causal}(\omega) \quad (11)$$

So the frequency response of the anti-causal filter is the complex conjugated of the frequency response of the causal filter. And therefore the phase shift of a causal filter will be opposite to the phase shift of the corresponding anti-causal filter. Which was to be expected, since the anti-causal filter is just the backwards version of the causal filter.

The symmetric filter has a symmetric impulse response around zero and is in Filter implemented as the average of the causal and the anti-causal filter:

$$y_{Symmetric} = \frac{y_{Causal} + y_{Anti-Causal}}{2} \quad (12)$$

The symmetric filtered signal will not be shifted in phase any more, compared to the original signal, as shown in Figure 5.

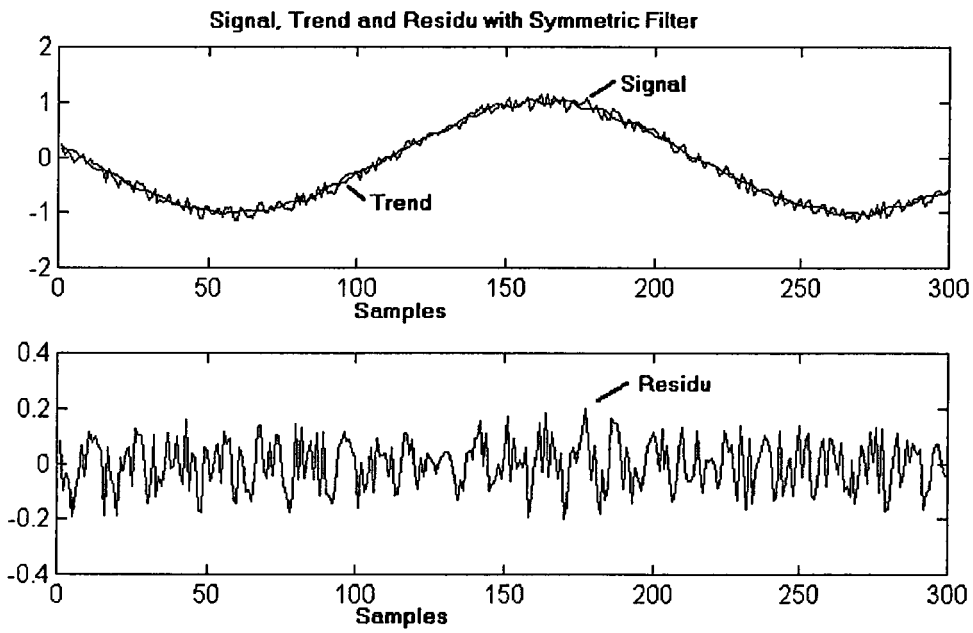

Figure 5: Symmetric Filter without a Phase Shift

Therefore the residu now contains almost no frequency components of the trend.

The symmetric filter has frequency response:

$$H_{Symmetric}(\omega) = \tfrac{1}{2}H_{Causal}(\omega) + \tfrac{1}{2}H_{Anti-Causal}(\omega) = \tfrac{1}{2}H_{Causal}(\omega) + \tfrac{1}{2}H^*_{Causal}(\omega)$$

$$= \mathrm{Re}(H_{Causal}(\omega)) = |H_{Causal}(\omega)|\cos(\phi_{Causal}(\omega))$$

with $\phi_{Causal}(\omega)$ the phase shift of the causal filter.

So, when the phase shift of the causal filter is not neglectable, the amplitude of the symmetric filtered signal will be (considerably) less then the amplitude of the causal filtered signal. The frequency response may even be zero in the passband due to a large phase shift. This may also lead to an unsatisfactory removal of the trend. However it is an effect which can be taken into account in the filter design.

2.2.7 Median Filter

A median filter takes the median of each sample, taken on a subinterval around the sample. Or in a Formula:

$$y_n = median\left\{ u_{n-\frac{1}{2}Wd}, u_{n-\frac{1}{2}Wd+1}, \cdots, u_{n+\frac{1}{2}Wd-1}, u_{n+\frac{1}{2}Wd} \right\} \qquad (13)$$

with Wd the median width.

A median filter can sometimes/often be used as (sort of) a lowpass filter. But it can also introduce high frequency components.

2.3 Outlier Detection and Removal

2.3.1 Introduction

There are many methods for the removal of outliers, such as a median filter which is shown in Figure 6.

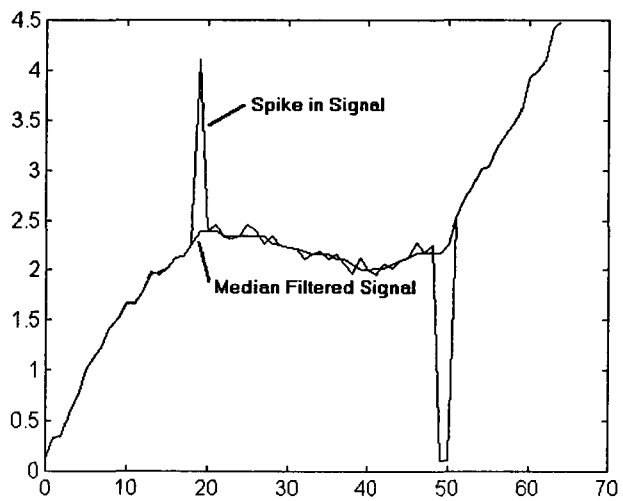

Figure 6: Outlier Removal with Median Filter.

Another outlier filter method is splitting the removal of outliers into two steps:

- An outlier detection method
- An outlier removal/repair method

This has the advantage that only the (detected) outliers are removed, whereas for example a median filter may affect the whole signal (also see Figure 6, especially between sample 20 and 48). Which in some cases can be unwanted. Moreover it is possible to combine the best of two methods.

2.3.2 Detection of Outliers

Again, there are many outlier detection methods. An important detection method is "the human eye", for a human will often have to decide if a disturbance is an outlier. Another method is the construction of an upper and a lower detection bound. A simple method for getting these bounds is just setting them to a value, of which it is known, that it will never be exceeded. So this boundaries can serve as detection bounds (see Figure 7). An example of such a known boundary can be the temperature of a vessel. If the temperature of the incoming flows is known, together with information about the processes in the vessel, the maximum and minimum temperature of the vessel may be also known.

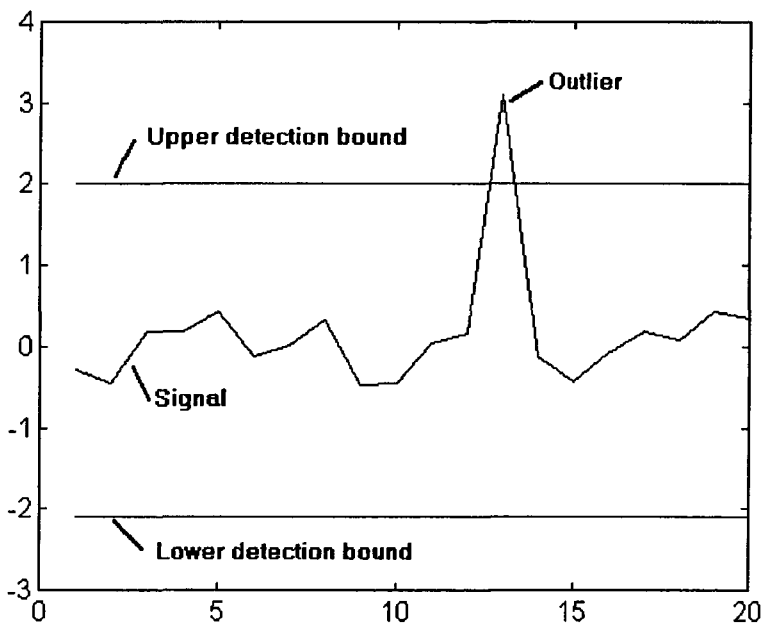

Figure 7: Upper and Lower detection bound

However it is not always that simple. An outlier may be smaller than the amplitude of the low frequency components of the signal (see Figure 8). In this context these low frequency components can be thought of as a "trend". Most of the times this is not the same trend as is described in 2.2.6 but a signal that is insensitive for outliers and follows most of the signals dynamics and possible low frequency disturbances. A possible trend is the median filtered signal, as shown in Figure 6. So the median filter can be used to remove outliers or, when the median filter affects the signal to much, it can be used to determine a possible trend. The usage of such a trend makes a more sophisticated detection method possible:

$$|x - x_{Trend}| > S\sigma \qquad (14)$$

with:

x	The signal
x_{Trend}	A trend of the signal
S	The shaving strength
σ	A standard deviation

In this formula $S\sigma$ serves as a detection bound. It is not necessary to split this bound into two parts/variables, however this splitting is done in most of the detection algorithms used in Filter. A visualisation of these detection bound(s) is shown in Figure 8.

But how does this actually work? First a trend is determined. There are various ways for doing this. Some possible trends are:

- The average value of the signal.
- The lowpass filtered signal.
- The signal minus the highpass filtered signal. This is just a difficult way of saying that $x - x_{Trend}$ is the highpass filtered signal.
- The median filtered signal (as mentioned and shown before), which is the median of each sample, taken on a subinterval around the sample.

There is no golden rule for telling which method is best, this entirely depends on the signal. After finding the best method, the variables of the method (for example a cut-off frequency of a filter) have to be fine-tuned. So the human eye and experience is needed, for finding the best method. Further a standard deviation is needed. Again there are various ways of finding a suitable standard deviation. These are methods like:

- The standard deviation of the signal.
- The standard deviation of the signal minus the trend.
- The standard deviation of the highpass filtered signal.

When all of this is done the best shaving strength has to be determined. The shaving strength determines how big a deviation has to be before it is considered to be an outlier. All this gives an upper detection bound ($x_{Trend} + S\sigma$) and a lower detection bound ($x_{Trend} - S\sigma$), as shown in Figure 8.

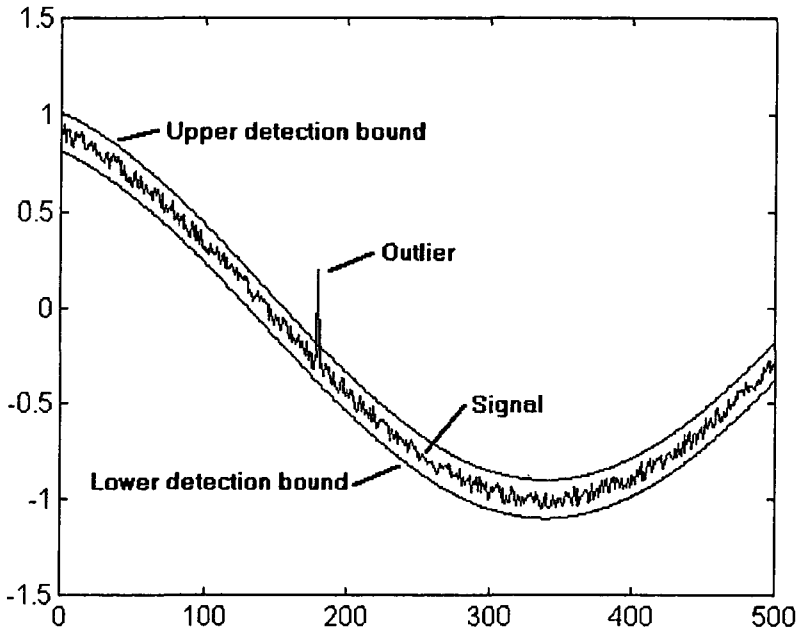

Figure 8: Detection bounds of a signal with trend.

Once the outliers are detected they have to be removed.

2.3.3 Removal of Outliers

The removal of outliers is very simple. Just replace the samples that are outliers with a repair value. There are many repair methods such as:

- Replacing the outlier with a constant (given by the user) value.
- Replacing the outlier with the average value of the signal (the outliers are excluded in the averaging).
- Add a value (given by the user) to the outlier.
- Interpolate between the sample before and after the outlier.

These methods are all very simple which makes them also very fast. However, this does not mean that only simple repair methods can be used. A very sophisticated and complex repair method is very well thinkable.

3. Filter: the implementation

3.1 Place of Filter within PRIMAL

Within PRIMAL, Filter is the operation between measurement and identification, as shown in Figure 9 and Figure 1.

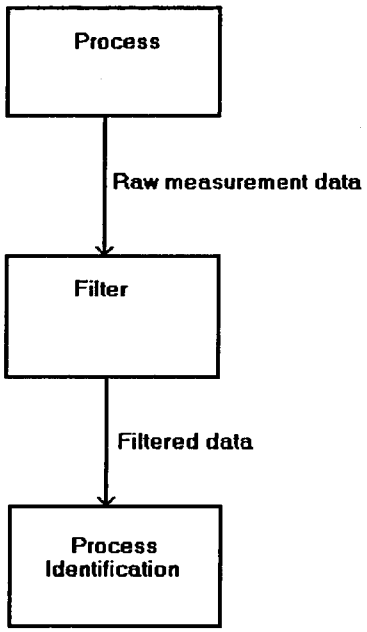

Figure 9: Place of Filter within PRIMAL.

This filter step is necessary because a set of raw measurement data is seldomly suited for direct use in analysis and identification, due to disturbances that appear in the data. These are disturbances like trends and outliers, as mentioned in 2.1.

3.2 Structure of Filter

Filter consists of six (groups of) dialogs/functions, as shown in Figure 10. First the signals, relevant to the identification, can be selected. Then the main filter dialog is entered, from which the outlier detection and repair dialog/function, the filter dialog/function and the mathematical operations dialog/function can be used. After that, the size of the dataset can be reduced.

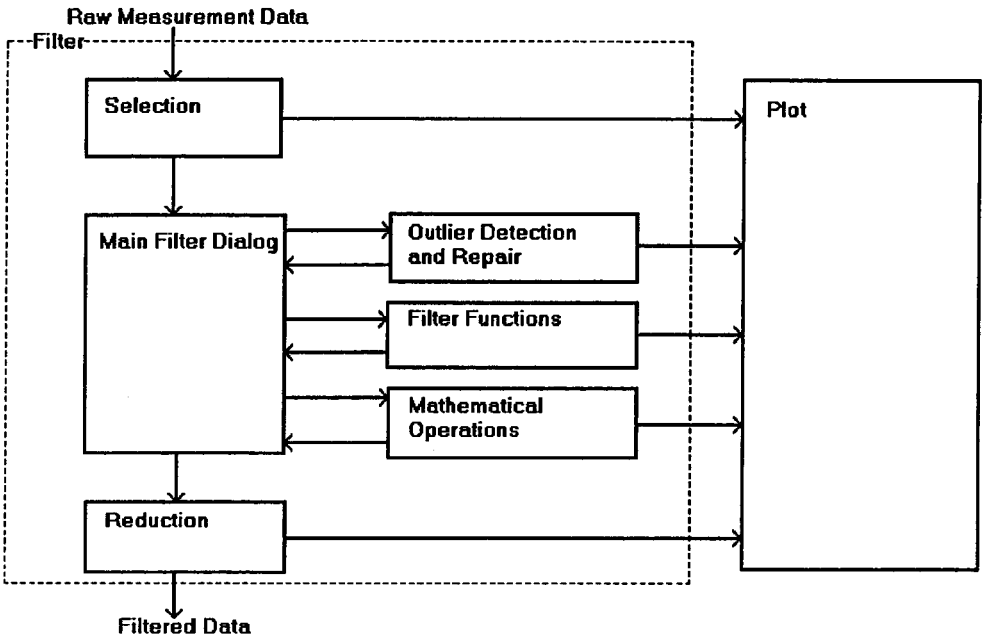

Figure 10: Structure of Filter.

All the dialogs (except the main filter dialog) use the plot module, so the filter results are visualised automatically. And all the functions have the same signaldata structure: They need a name of an input dataset, an output dataset and a work dataset, as shown in Figure 11, further they need the name of the dialogdata so the last settings can be retrieved.

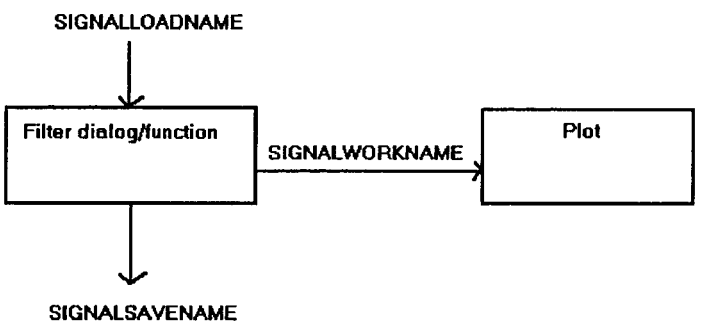

Figure 11: Structure of the filter functions.

SIGNALLOADNAME is the name of the input dataset, SIGNALSAVENAME is the name of the output dataset and SIGNALWORKNAME is the name of the work dataset. Because of the communication with plot the work dataset has to exist as a database object.

The work dataset contains the current signal dataset and the filtered signal dataset. A filter function takes the current signal dataset as an input signal, and puts the result in the filtered signal dataset. If another filter is used only the filtered signal dataset changes. So various filters or filter settings can be attempted, until the user is satisfied with the result. Once the user is satisfied, the accept button can be pressed. This replaces the current signal dataset with the filtered signal dataset, so the result becomes permanent.

3.3 Filter Functions and Dialogs

As mentioned before, Filter consists of six (groups of) dialogs/functions:

- The selection, work interval and time shift dialog.
- The main filter dialog.
- The outlier detection and repair dialog.
- The filter dialog.
- The mathematical operations dialog.
- The reduction and save interval dialog.

In the appendixes is described how each function can be used and how user defined (filter) functions can be added. In this paragraph only what the functions can do, and what the (implementation) characteristics are, will be described.

3.3.1 Selection, Work Interval and Time Shift Dialogs

Datasets, of measurements collected during experiments on a practical (industrial) process, may contain up to a 100 signals. Often not all these signals are used for (the same) identification. For example a sub process may be identificated separately. So a selection of the relevant signals must be made. This can be done with the selection dialog which is shown in Figure 12. This dialog is the first in a set of four dialogs.

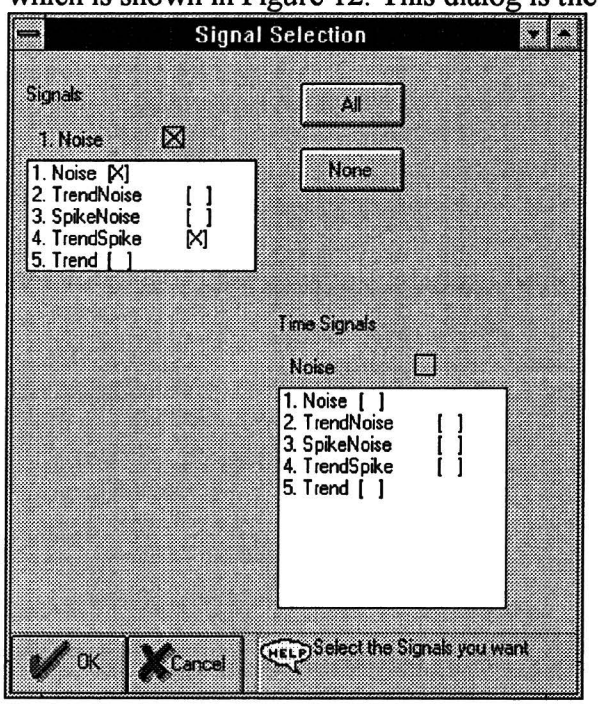

Figure 12: Signal Selection Dialog.

In this dialog the signals and the time signals can be selected. A time signal is a signal which provides another signal with a time base/axis. So when the samples of a signal occur on an irregular time base the signal can be resampled, according to the given time base. The resampling is done by linear interpolation. This resampling is necessary because some filter functions assume a equidistant discrete time base. Which time signal belongs to

which signal can be selected in the second dialog. Of course when no time signals are selected the second dialog is skipped. Once this is done the work interval and the time shift can be selected (see Figure 13).

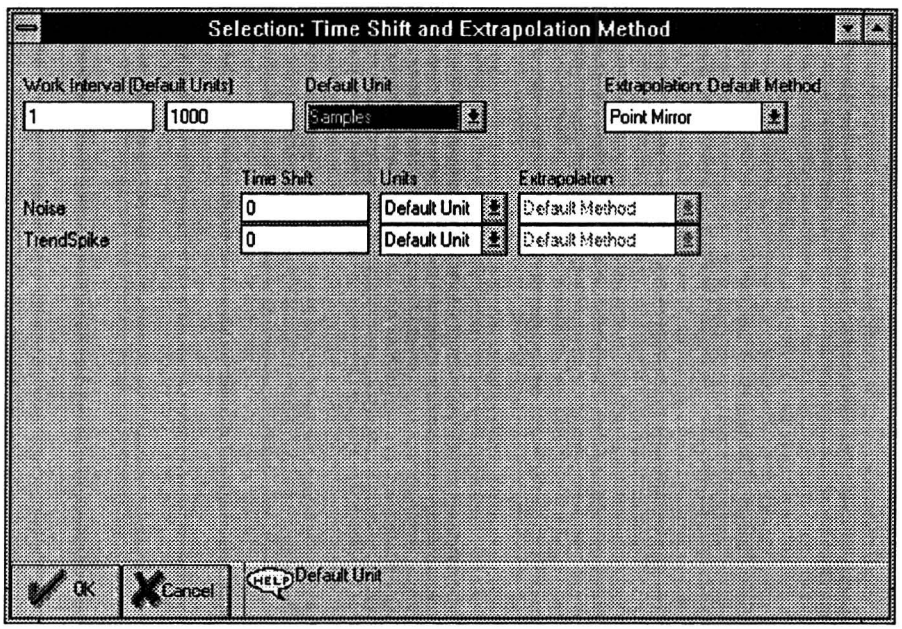

Figure 13: Time Shift, Work Interval and Extrapolation Method.

When, for example, only a piece of a very long dataset contains measurements, relevant to an identification, a work interval can be selected. This work interval can be given in samples, second, minutes and hours.

The work interval may exceed the available data, then the data will be extrapolated. This is done to cope with starting problems that some filters have (the filter needs some time to follow the dynamics of the signal). This starting problem may lead to less useful filtered data at the beginning (or ending, if filtered backwards) of the data sequence. If a filter has this problem it can be dealt with by first extrapolating the signal. The filter will start with the extrapolated part of the signal, so that the less useful part of the filtered data, lies within the extrapolation, as is shown in Figure 14. Afterwards the extrapolated part, of the filtered data, can be clipped of. (In Figure 14 the extrapolation is done with the Point Mirror method.) Of course this clipping can also be done, to remove less useful filtered data caused by starting problems of a filter, when no extrapolation has been done.

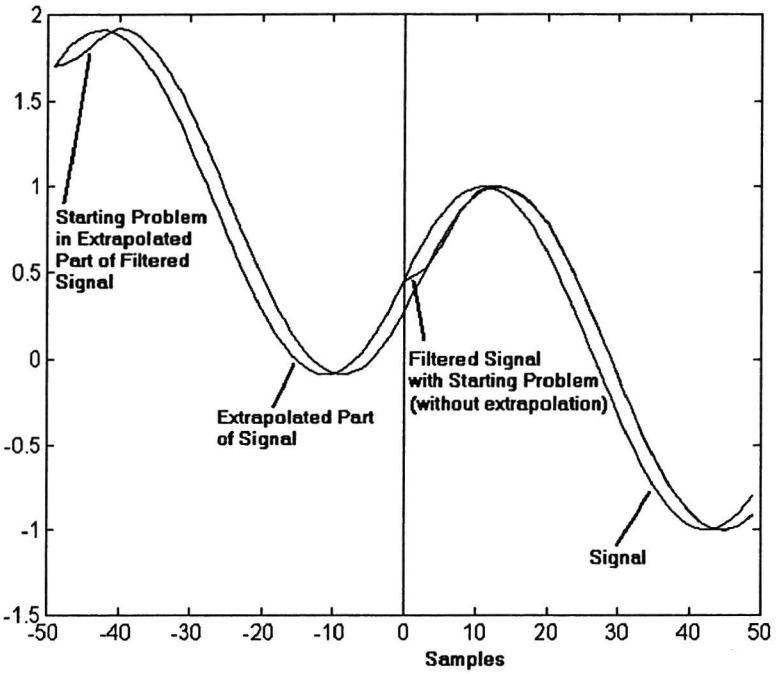

Figure 14: Starting Problem and Extrapolation.

There are signals that can only be measured with a, sometimes significant, time delay. Such a time delay may be caused by transportation times or delays in sensors. Often the delay is known and not interesting. However the time delays involve extra model parameters, since they have to be estimated as part of the model. So the number of parameters to be estimated increases when the modelling of the time delays has to be done by the identification algorithms. This leads to unnecessarily high order models. When the time delays are known, these unnecessarily high order models can be avoided by shifting the signals in time. So the time delays will not have to be modelled. Afterwards, when the model is obtained, the time delays can easily be included in the model. When these delay times are not known, they can be estimated (with correlation analysis or impulse response estimation) separately from and prior to the model parameter estimation. The information obtained can then be used as a-priori knowledge. In Filter the signals can be shifted in whole samples, so no resampling or interpolation will be done.

When samples are needed that are not available, because signals are shifted in time or the work interval exceeds the available data, an extrapolation has to be done. The time shift and the work interval may both require extrapolation, therefore they are presented in the same dialog. In Filter there are four extrapolation methods implemented:

- Point Mirror: A polynomial is estimated on the first (or last) samples of the signal, and the signal is mirrored in the first (or last) point of the estimated polynomial. This has the advantage that, in the mirror point, the signal itself and the first derivative in time fits well to the extrapolated signal.
- Constant: A constant value, given by the user, is used as an extrapolation value.

- First/Last: When data is needed before the first sample, the first sample is used. When data after the last sample is needed, the last sample is used.
- Mean: The average value of the signal is used.

The last dialog contains the extrapolation variables (for example the value of a constant). This dialog is also skipped when no extrapolation variables are needed.

3.3.2 Main Filter Dialog

The main filter dialog (shown in Figure 15) provides access to the filter functions: outlier detection and repair, filter and mathematical operations. Further some book-keeping can be done such as renaming, removing and adding (empty) signals. It is also possible to load signals. This is done by calling the selection, work interval and time shift dialog. When signals are not present in the work dataset the signals are added, if they are present the signals are reloaded. The selection dialogs can be (partly) skipped so no unwanted settings have to be done twice. Further data reduction can be chosen and the Save Name can be set.

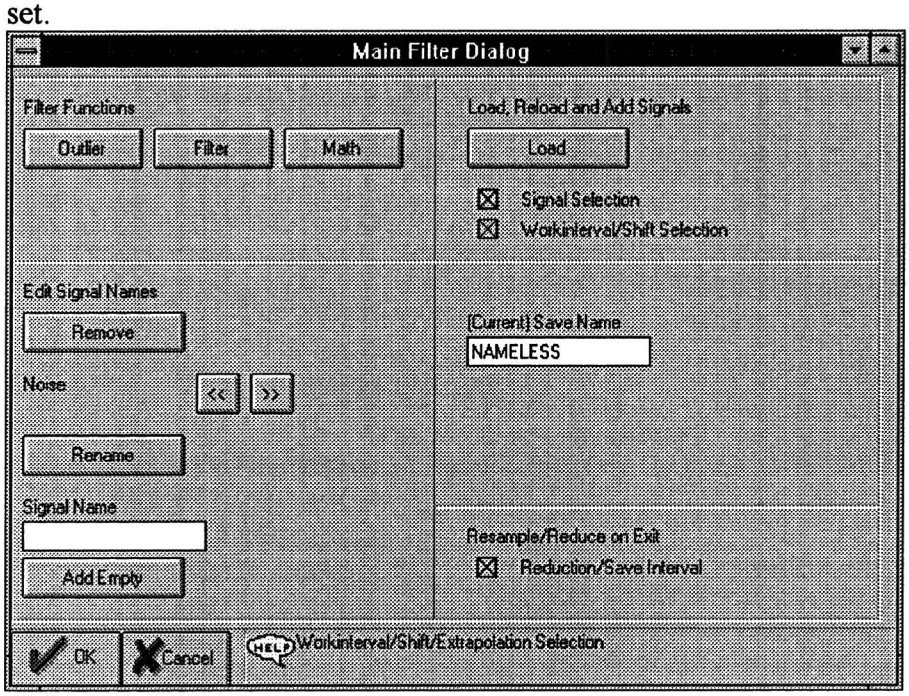

Figure 15: Main Filter Dialog.

3.3.3 Outlier Detection and Repair Dialog

This function/dialog (see Figure 16) can detect outliers/spikes and (manually) repair data. If these spikes are not removed from the signals, they may form an important part of the noise energy and have a considerable influence on the ultimate model, although they have no relation with the process itself (as mentioned in 2.1).

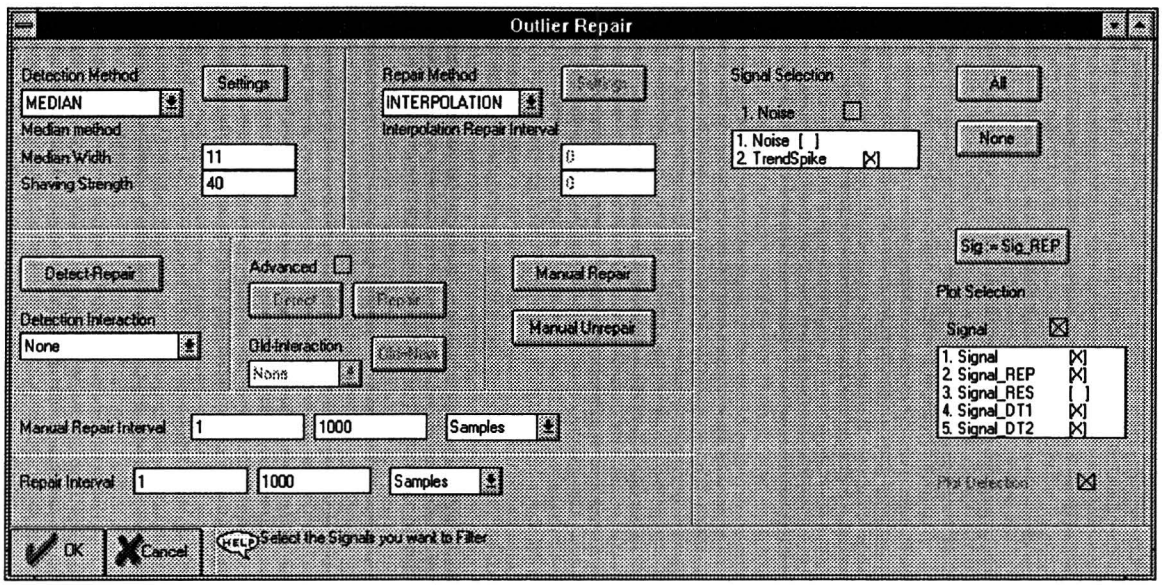

Figure 16: Outlier Detection and Repair Dialog.

The detection methods are implemented in a Dynamic Link Library (DLL). This means that they are not a part of the main program. A new method can be implemented easily without any need for the source of or linking with the main program (FILTER.EXE). How a new detection method is put in a DLL is described in the appendixes. Currently the following detection methods are implemented:

- Clip: An upper and lower detection bound can be given.
- Backx: The lowpass filtered signal (with a second order Chebyshev filter) is taken as a trend (x_{Trend}, see 2.3.2). The standard deviation (σ) of the signal minus the trend is used.
- Trend: The signal minus the highpass filtered signal is taken as a trend. The standard deviation of the highpass filtered signal is used.
- Level: The average value of the signal is taken as a trend. The standard deviation of the signal is used.
-Median: The median filtered signal is taken as a trend. The standard deviation of the highpass filtered signal is used.

The result of a detection is stored in a repair buffer. This buffer contains the (numbers of the) samples that need to be repaired: the repair samples.

Repair samples calculated by a detection method or given by the user can be repaired by one of the following methods:

- Constant: The repair samples are replaced by a constant given by the user.
- Add Value: A constant, given by the user, is added to repair samples.
- Mean: The repair samples are replaced by the average value (the repair samples are excluded in the averaging) of the signal.
- Interpolation: The repair samples are replaced by a linear interpolation between the sample before and the sample after the repair sample(s). If the repair sample is the first or last sample of the data the mean repair method is used, because interpolation is not possible for the sample before the first sample does not exist.

The repair methods are also implemented in a DLL and therefore have the same advantages as the detection DLL's.

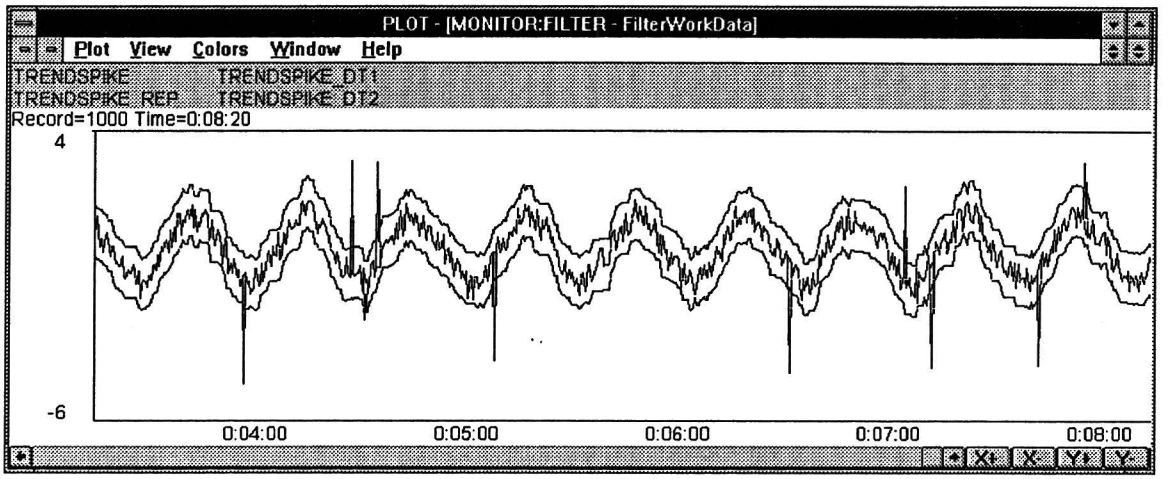

Figure 17: Signal, Repaired Signal and Detection Bounds.

The detection and repair method can be used together and separately. The result of a detect/repair is automatically visualised in a plot as shown in Figure 17. As soon as, after some attempts, the user is satisfied with the result, the accept button (Sig := Sig_REP) can be pressed. This replaces the unrepaired data with the repaired data, so the result becomes permanent, and the original data can only be retained by reloading the original dataset.

3.3.4 Filter Dialog

This function/dialog (see Figure 18) is the "real" filter function, that is a filter with one input and one output signal and some variables like a cut-off frequency or a median width. It can be used for various filter actions like lowpass filtering, the removal of trends (see 2.2.6) or the removal of spikes (see 2.3.1).

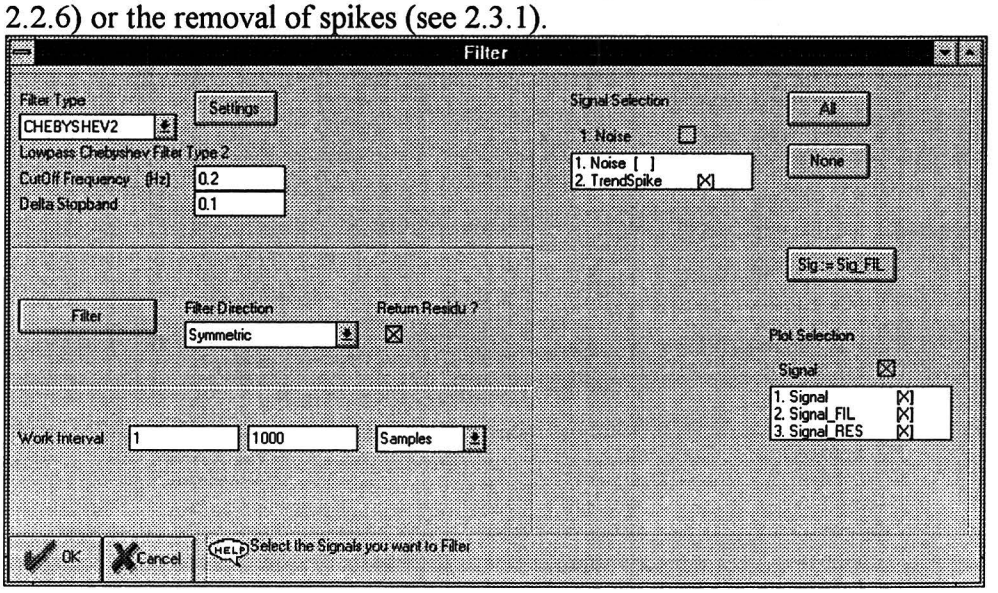

Figure 18: Filter Dialog.

Some features are the same as in the detect-repair function. Such as the automatic visual feedback, the accept button and the fact that the filters are implemented in a DLL. These filters are described in the appendixes and in literature like [JAC86], here they are only mentioned by name:

- FIR: A highpass, lowpass, bandpass and bandstop FIR-filter, with a Square (no window) Hanning, Hamming and Blackman window.
- Butterworth: A highpass, lowpass, bandpass and bandstop Butterworth filter.
- Chebyshev1: A highpass, lowpass, bandpass and bandstop Chebyshev filter of the first type.
- Chebyshev2: A highpass, lowpass, bandpass and bandstop Chebyshev filter of the second type.
- Medianfl: A median filter.

There are three filter directions:

- Causal: Filtering forwards.
- Anti-causal: Filtering backwards.
- Symmetric: The average of causal and anti-causal.

The symmetric filter can be very useful in trend determination and removal (see 2.2.6). An example of a possible trend, calculated with a symmetric filter, is given in Figure 19.

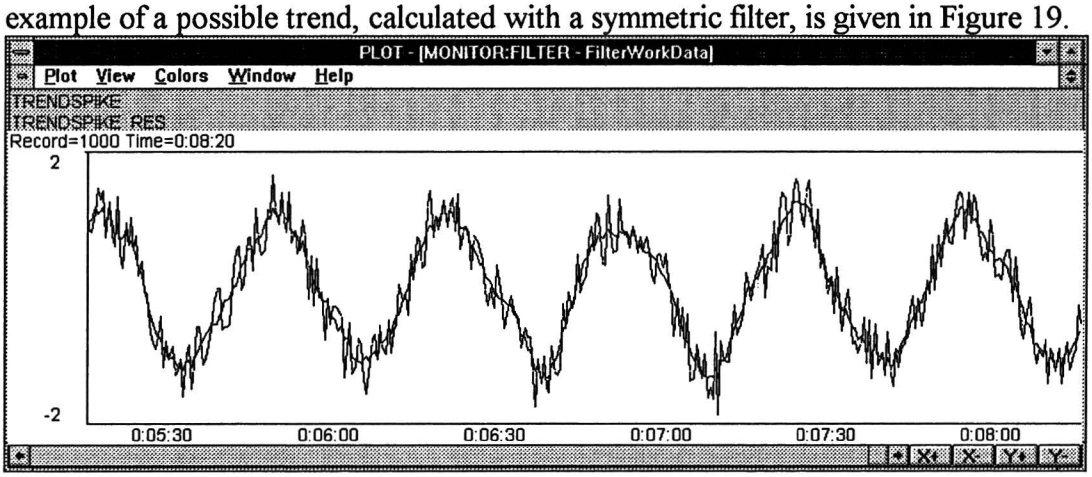

Figure 19: Possible Trend.

3.3.5 Mathematical Operations Dialog

With this function/dialog (see Figure 20) it is possible to perform mathematical functions and operations on the signal. These are operations like: scaling, offset correction, adding, multiplying, the standard mathematical functions such as sinus, inverse sinus and square root. For example scaling may be performed to increase the accuracy in a parameter estimation. Or when a linear model of the process has to be formed, it is necessary to correct for the known non-linearity's. Further manual signal repair can be performed by specifying a data range and a repair function (a mathematical expression).

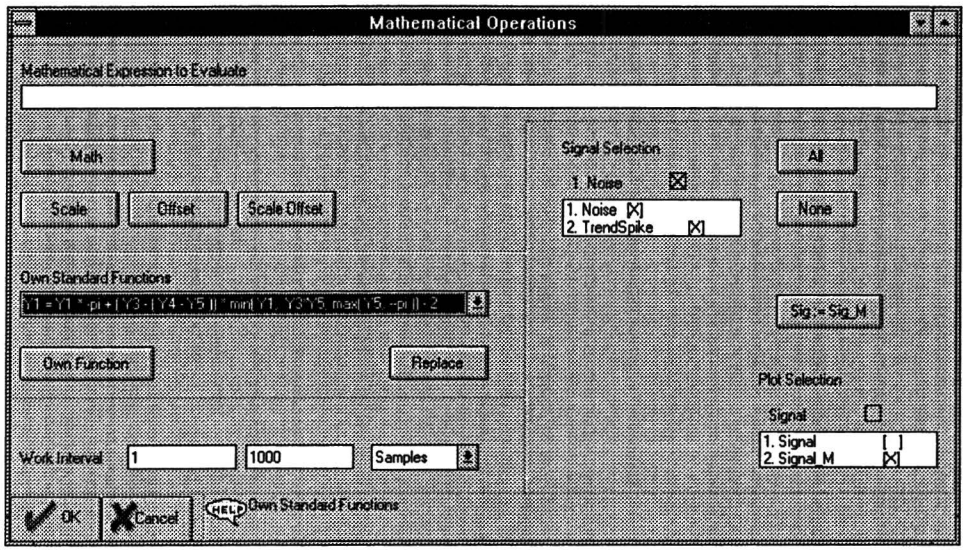

Figure 20: Mathematical Operations Dialog.

The operators and functions are implemented in a DLL so it is easy to change the syntax or to add functions (for example a function which calculates a physical value out of a voltage of a sensor). In the basic syntax of the interpreter, only brackets and the notation of functions is not flexible. The rest, the operators and functions, is not a part of the basic syntax. The currently implemented syntax supports adding (+), subtracting (-), multiplying (*), dividing (/) and 26 (standard) mathematical functions. The interpreter also supports constants (which are defined in a list like: pi=3.14159..) and single operators. Single operators take only one input value, some examples are: +, - or d/dx (not implemented). Of which the first two examples are overloaded operators because they are also defined for two input arguments. All of this gives an interpreter which has no problem with expressions like:

y=sin(min(3*4,y1+y2,y3))*---pi+(2+(3-4)*7)*mean(cos(y2))

With y, y1, y2 and y3 names of signals, sin, min, mean and cos functions and pi a constant.

Further there are two reserved variables: time, which stands for the (sample) time of the signal(s) (in seconds), and x which stands for the current selection of the signals. The usage of x makes it possible to perform a function on a group/selection of signals. And often used expressions, which can be put in an own function list, can be made signal independent by putting expressions like x=sin(x) in the function list. This expression can then be used by selecting the signal and pressing the own function button.

3.3.6 Reduction and Save Interval Dialog

With this function/dialog (see Figure 21) data reduction can be done and the save interval can be set.

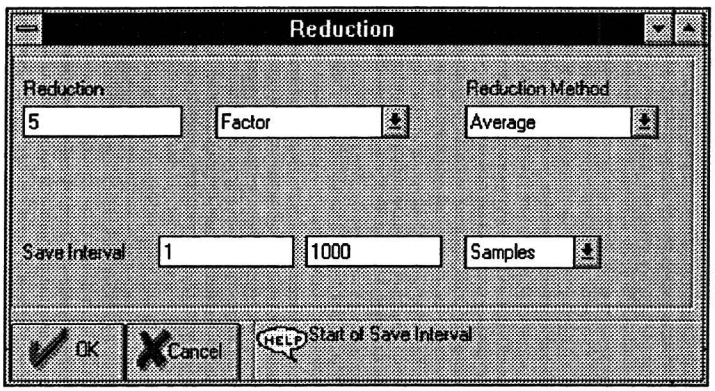

Figure 21: Data Reduction and Save Interval dialog.

Because of the desire for data conditioning, the sampling of process signals is often done at a frequency 5 to 10 times higher than the frequency essentially needed for the identification of the process. If these signals are used for parameter estimation, problems may occur at high frequencies. Because the input signals have but little power at these frequencies, while the recorded output signals sometimes have considerable components at this frequencies. To prevent unnecessary high order models and numerical problems the excess of data must be removed. This can be done with two methods:

- Average: Which calculates the average values of each group (consisting of the same number of samples as the value of the reduction factor is) of samples. These averages form the reduced signal.
- First: Which takes the first sample of each group of samples; decimation.

Further the reduction factor can be given or the number of desired samples.

At last the save interval can be set. So for example the extrapolated part of a dataset, introduced to cope with starting problems (see 3.3.1), can be removed.

4. Glass-Feeder: A Test Case

4.1 Introduction

In order to test the developed filter module, filtering and process identification have been done with process data of a glass feeder. Glass from such a feeder, for instance used for the production of television tubes, must satisfy high quality demands, such as:

- A pure chemical composition.
- No visual disturbances in the glass.
- A constant absolute temperature.
- A homogeneous temperature profile, in place.

Especially the last two demands can be satisfied with the glass feeder.

4.2 Process Description

Glass is made in a furnace out of sand and some additives. This glass pours from the furnace into the feeder (the left side in Figure 22). The feeder is a canal in which two major compartments exist where the temperature of the glass can be affected. The feeder is several decimetres deep and wide and several metres long. The height of the glass bed in the feeder is also several decimetres. During experimentation a thorn was mounted in the opening of the spout. A tube was made of the glass pouring out the spout.

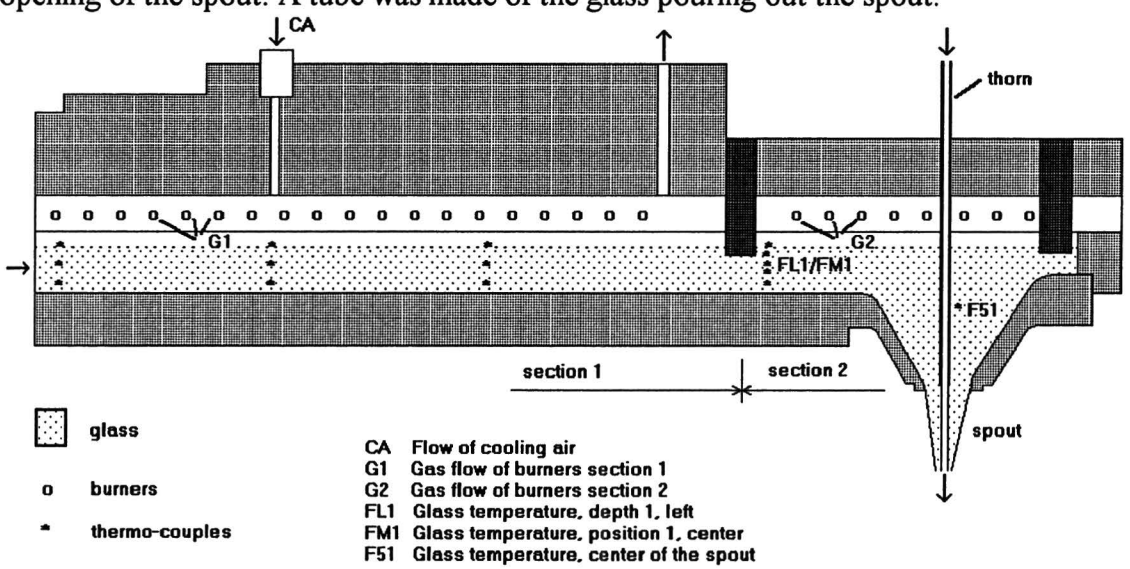

Figure 22: Schematic view of the glass-feeder.

In the feeder the glass cools down to a temperature of about 1000 °C in the spout. The average residence time of the glass in the feeder is 1-2 hours.

The process has three interesting control variables:

- The gas flow to the burners in section 1. With the burners the glass can be heated.
- The cool air flow in section 1. The glass can be cooled by blowing air above the surface of the glass bed.
- The gas flow to the burners in section 2.

During the measurements all kinds of disturbances occurred. In order to remove (most of) these disturbances before identification the data was filtered with the developed filter module.

4.3 Filtering the Glass-Feeder Data

The following signals were filtered and used for the identification:

- CA The flow of the cooling air.
- G1 The gas flow of the burners in section 1.
- G2 The gas flow of the burners in section 2.
- FL1 The glass temperature at depth 1 in the left (not the left in Figure 22) of the feeder.
- FM1 The glass temperature at position 1 in the centre of the feeder.
- F51 The glass temperature in the centre of the spout.

Of which the first three are inputs and the last three outputs. The used sample time was 50 seconds and the length of the collected dataset is 12472 samples. As inputs Pseudo Random Binary Noise Signals (PRBNS) with a width of 10 samples were used.

First offset correction and scaling was performed on the dataset. This was done to obtain numerical stability. If for example the first four digits of a number are the same the accuracy of a calculation by a filter will also decrease with four digits.

Then the outputs (F51, FL1 and FM1) were filtered with a symmetric lowpass Chebyshev filter of the second type and a cut-off frequency of 0.002 Hz (frequency ratio=1.1, delta passband=delta stopband=0.05, maximum filter order=100). This was done to remove all the frequency components higher than the frequency belonging to the width of the PRBNS. There was chosen for this filter because it has no phase shift and a high roll-off rate near the cut-off frequency, further it is reasonably flat in the passband (see Figure 42).

After that the spikes and high frequency disturbances were removed from the input signals by applying the function: max(min(x, 0.7), -0.7). Which is shown for CA in Figure 23. The removal of the high frequency disturbances was not done with an IIR-filter because

such a filter had a large starting problem/"step response" at each change in the PRBNS.

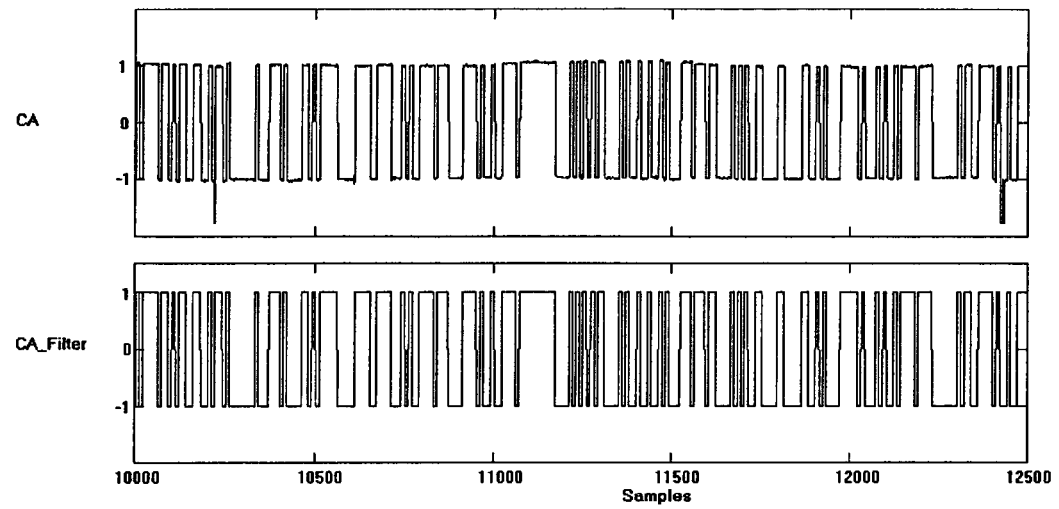

Figure 23: Cooling Air, Signal and filtered signal.

After this the data was again scaled and offset corrected, and the number of samples was reduced (decimated) with a factor 5. If no data reduction was done the lowpass filter (the next filter step) had numerical problems.

Then the trend was removed in the output signals with a symmetric highpass Chebyshev filter of the second type and a cut-off frequency of 0.00001 Hz (frequency ratio=1.1, delta passband=delta stop band=0.05, maximum filter order=100), which is shown for F51 in Figure 24. At first a symmetric lowpass Chebyshev filter was tried for the determination of the trend but this filter was not very useful. Due to the large phase shift of the causal filter the amplitude of the symmetric filter was smaller than the trend (see end of 2.2.6) so the trend could not be removed with this filter.

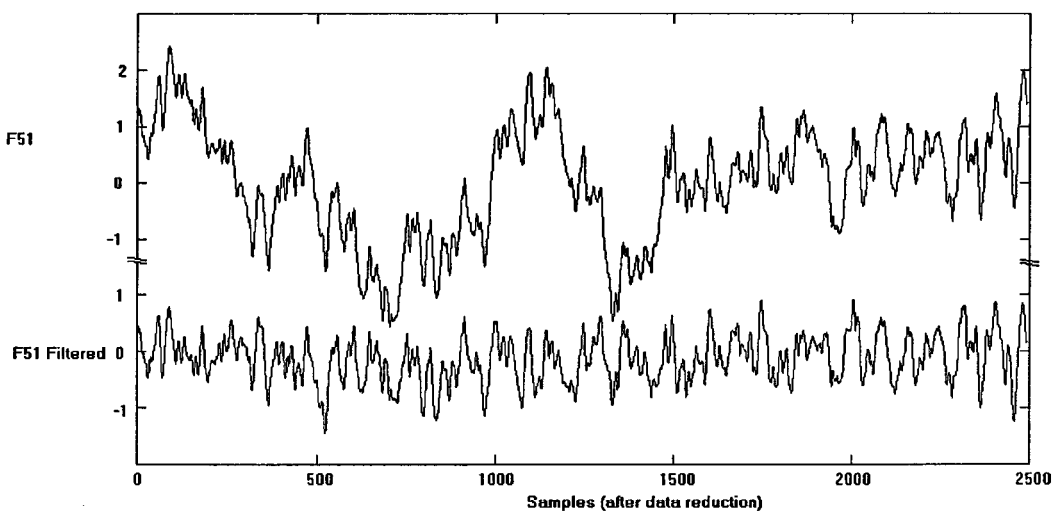

Figure 24: Temperature F51 before and after trend removal.

4.4 Identification

A general linear model has been estimated. This model had, in the z-plane, the following structure:

$$A(z)y(z) = \sum_{i=1}^{3} \frac{B_i(z)}{F_i(z)} u_i(z) + \frac{C(z)}{D(z)} e(z) \qquad (15)$$

with:

$A(z), B_i(z)$ *and* $F_i(z)$	The z-transformation of the model parameters
$u_i(z)$	The input signals (G1, G2 and CA).
$y(z)$	The output signal (F51, FL1 and FM1).
$C(z)$ *and* $D(z)$	The z-transformation of the noise filter.
$e(z)$	The z-transformation of the noise.

Presented are the impulse responses of the inputs to the outputs, simulated with the estimated models. The estimation has been performed on fully conditioned process data. With the "raw" data no model could be found (with the used estimation method) due to numerical problems.

The results form three dynamic relations are presented. For all the three estimations the first 1500 samples (after data reduction) were used.

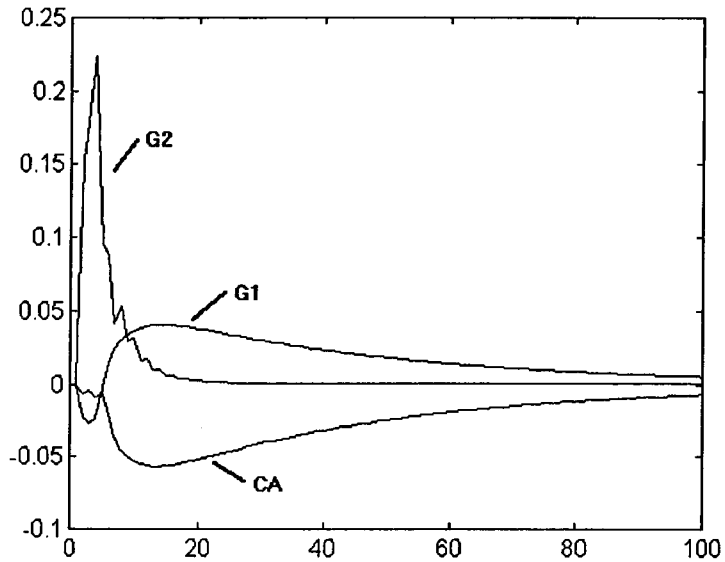

Figure 25: Impulse Responses of G1, CA and G2 to FL1.

Inputs	Output
G1 CA G2	FL1

Model Order	*nA*	*nB*	*nC*	*nD*	*nF*
	5	5	3	2	1

with nA, nB, nC, nD and nF the model orders of $A(z), B_i(z), C(z), D(z)$ *and* $F_i(z)$

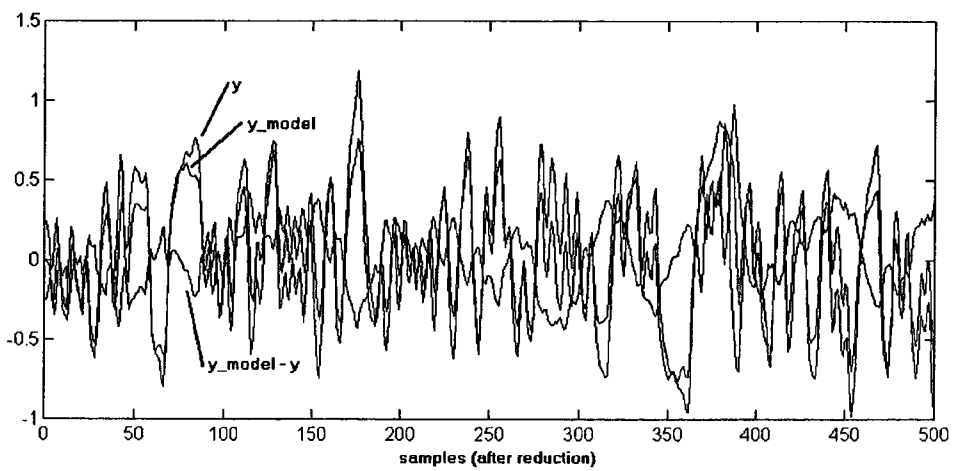

Figure 26: Output, Simulated output and Residu of FL1.

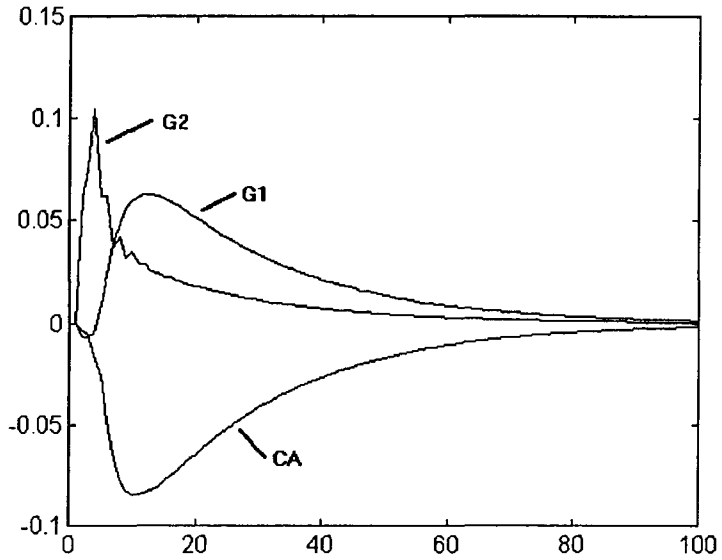

Figure 27: Impulse Responses of G1, CA and G2 to FM1.

Inputs	Output
G1 CA G2	FM1

Model Order	*nA*	*nB*	*nC*	*nD*	*nF*
	5	5	3	2	1

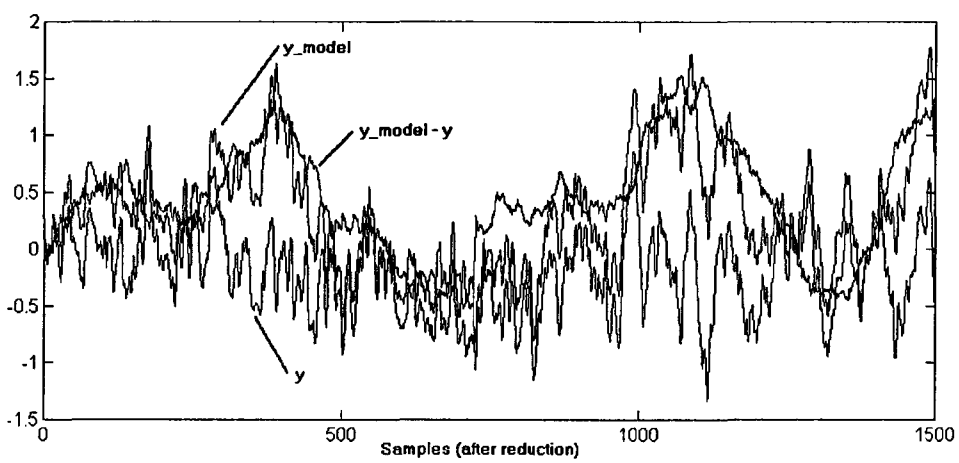

Figure 28: Output, Simulated output and Residu of FM1.

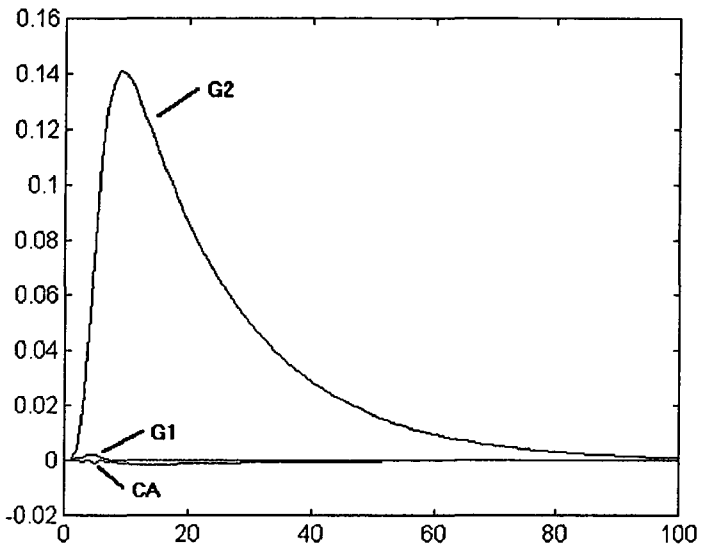

Figure 29: Impulse Responses of G1, CA and G2 to F51.

Inputs	Output
G1 CA G2	F51

Model Order	*nA*	*nB*	*nC*	*nD*	*nF*
	4	5	3	2	1

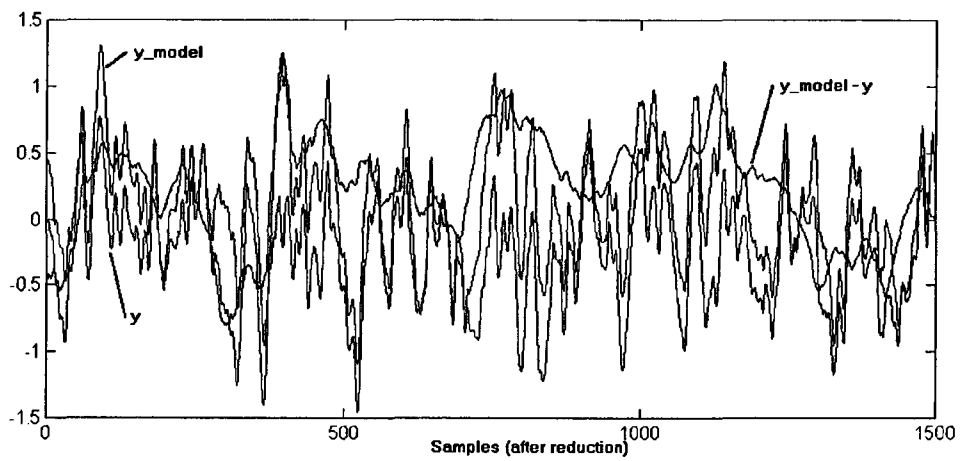

Figure 30: Output, Simulated output and Residu of F51.

The following variance ratios were found:

Output signal	$\frac{\text{var}(y - y_{Model})}{\text{var}(y)}$
FL1	0.3373
FM1	1.8850
F51	0.9684

These variance ratios are not very good. The reason for this is that the trend in the output signals was not fully removed as can be seen in Figure 26, Figure 28 and Figure 30. However this partly removed trend proved to be no problem in the model estimation. The output signal of the model does not follow the trend of the original output, because the trend is not present in the input signals.

Another way to validate the rightness of the model is to predict the signal based on the current output and the current and future inputs. This is shown for F51 in Figure 31.

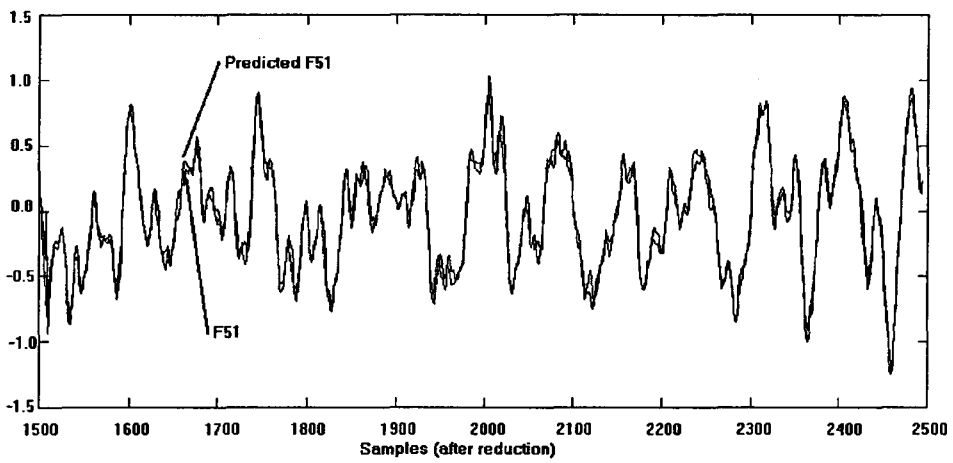

Figure 31: F51 and predicted F51.

The output signal is predicted 29 minutes (7 reduced samples) ahead. Based on the predicted signal the following variance ratios were found:

Output signal	$\frac{\text{var}(y - y_{predicted})}{\text{var}(y)}$
FL1	0.1457
FM1	0.0838
F51	0.0403

4.5 Final remarks

Since the model identification was not implemented in the PC-version of PRIMAL the identification was done in MATLAB. The major bottle-neck (in time) was the transportation of signal data between PRIMAL and MATLAB. It may be very time saving if with a filtered dataset an identification could be done without leaving the filter module. So when the user is not satisfied with the identification result more/other filters can be used.

The "best" cut-off frequency for the lowpass filter was found by visually inspection of the output signal of the process, the output signal of the model and the residu. Various cut-off frequencies were attempted until the best cut-off frequency was found.

The variance ratio of the predicted F51 and F51 was used to obtain the best model order for this signal. Again various model orders were attempted.

The filters that were not present in the previous PRIMAL filter module were used on purpose. This was done to test if they were useful for the filtering of the process data. In this case the Chebyshev filter and the matrix interpreter proved to be very useful.

The desired data conditioning could be done with the filters that are currently implemented in the developed filter module, no other filters were needed. However it is not said that they will be satisfactory for all signal conditioning. For example a trend may be determined and removed by estimating and subtracting a low order polynomial fit from the data.

Of the implemented IIR-filter the Chebyshev filter of the second type was used for it gave the best results (based on visual inspection). This filter has a very flat passband and a high roll-off rate near the cut-off frequency. Which results in a small phase shift (compared to the other implemented IIR-filters) and a sharp cut-off frequency.

5. Conclusions and Recommendations

5.1 Main Conclusions

The developed filter module for PRIMAL, which is to be used for the reduction of disturbances that are present in process data before identification, satisfies the required design specifications on the level of filter specifications, software implementation and the user interface.

Filtering of process data before identification is often necessary. For example with the unfiltered process data of the glass feeder no identification could be done. That is the identification method failed due to numerical problems. The choice of an interactive filter environment proved to be very useful. However the current computers are to slow to work comfortable with large datasets in such an interactive way. So for a large dataset a faster computer is recommended. Such a fast computer will be no problem in the future.

The framework structure of the developed filter module is very useful. During the development new outlier detection algorithms, repair algorithms, filter algorithms and mathematical functions and operations could be added easily.

5.2 Recommendations for Filter

It may be necessary/wanted to use a filtered dataset of Filter, or an intermediate dataset, while Filter is running. So for example an identification can be done simultaneously. And when a user is not satisfied with the result, more filtering can be done. Currently this is possible in the main filter dialog, but not in its sub-dialogs (Selection, Outlier, ..). Maybe an extra button like "Create Intermediate Database Object" should be added to each sub-dialog to provide for this need.

A better communication with the plot function seems useful: It should be possible to select a spike (or a work interval) in plot and then manual repair it, without giving (typing) the repair interval. This can be done with options like Copy and Paste or by selecting an area in plot, dragging it to a (repair) button and dropping it there. This is not possible with the current plot. But a new plot function is in development, so this an improvement that can soon become reality.

More feedback to the user by messages like: Busy filtering, done 10 %, is advised. This is very difficult in the current dialog base class (PDLGBOX), because in PDLGBOX it is not or at least hardly possible to open a window with only a message (and no Ok-button) that can be updated. Such a window is possible, if not with PDLGBOX then with a Windows class, but not implemented since it had a low priority.

In Filter the dialog settings are saved as a database object so they can be retrieved later. Sometimes it can be useful to save a set of dialog settings in a script. In filter this is not an option. Such a script can then be applied to a different dataset which may be very time saving when the same filter actions have to be done on similar datasets.

The filter functions are implemented in DLL's. This means that they are not a part of the main program. So new functions can be implemented easily without any need for the source of or linking with the main program. An other advantage of the implementation in DLL's is the fact that the filter functions can be used in other programs independently from Filter.

5.3 Recommendations for PRIMAL

In PRIMAL all the datasets (signaldata, dialogdata) are in the same directory. A more structured way would be a one with directories and sub-directories. So for example each filter function can have its own directory. Very recently such a directory structure has been made possible.

Some internal optimisations and some new functions in the base classes may improve the performance of some functions in Filter, and other applications, a lot. These are optimisations like a fast matrix invertation or multiplication or fast block copy functions. Such an optimisation can also be done in some of the Filter classes. A first optimisation is already done in performance-critical functions like the causal filter function.

5.4 Personal Opinion about C++

C++ is a powerful and structured way of coding, mainly for two reasons:

- C++ is object oriented, which makes it easy to re-use code. A new class can be derived from a base class. A class is, simply said, a group of functions and data that belong together. This new class then inherits all the functions of the base class. By replacing a few of these inherited functions, with new functions, a slightly different class can be created.
- C++ supports overloading of functions and operators. So functions that do the "same", but have other variable types as input variables, can all be given the same name. So a function that loads a database object can always be named "load". The overloading of operators makes it possible to define, for example, adding and multiplication on new variable types. So a compact notation for a matrix multiplication can be used.

A disadvantage can be, especially in the beginning when one is not familiar with the C++ language, the complex syntax of the C++ language. This may lead to code that will not do, what it is expected to do, although the syntax is correct. For example: if (a=3) ... will put 3 in the variable a, and execute the statements within the if statement. This if-construction

is easily confused with: if (a==3) ... which compares the variable a with 3. However, it is not said that the first if-construction never will be useful.

Another possible confusing syntax is a[2,1]. If a is an array of (C++) variables, this does the same as a[1], due to the implementation of the comma-operator. The "correct" way to use multiple arrays is: a[2][1]. However, the first array-notation can be very useful when the comma-operator is defined for a new variable type.

References

[BAC87] Backx T.
IDENTIFICATION OF AN INDUSTRIAL PROCESS:
A MARKOV PARAMETER APPROACH
PhD Thesis, Eindhoven University of Technology 1987

[JAC86] Jackson, L. B.
DIGITAL FILTERS AND PROCESSING
Kluwer Academic Publishers 1986

[JAN87] Janssen P.A.
PRACTICAL ASPECTS OF PROCESS IDENTIFICATION WITH
PRIMAL; A TOOL FOR PROCESS IDENTIFICATION
M. Sc. Thesis, Internal report NR-1470P,
Eindhoven University of Technology 1987

[LIN90] Linden, van der, R.J.P.
DESIGN AND APPLICATION OF PRIMAL: A PACKAGE FOR
EXPERIMENTAL MODELLING OF INDUSTRIAL PROCESSES
PhD Thesis, Eindhoven University of Technology 1990

[VIS93] Vissers B.A.J.
TRENDFILTERING VOOR ZWARTE MODELVORMING IN PRIMAL
Internal report NR-1827,
Eindhoven University of Technology 1993

APPENDIXES:

APPENDIX A: Filter module

Filter consists of six (groups of) dialogs/functions:

- The selection, work interval and time shift dialog.
- The main filter dialog.
- The outlier detection and repair dialog.
- The filter dialog.
- The mathematical operations dialog.
- The reduction and save interval dialog.

In this appendix is described what the function/purpose of every part (for example a button) of the dialog is. How user defined (filter) functions can be added is describe in Appendix B.

A.1 Selection, Work Interval and Time Shift Dialogs

This function consists out of a group of four dialogs. With the first dialog (Figure 32) the relevant signals and the time signals can be selected. A time signal is a signal which provides another signal with a time base/axis. So when the samples of a signal occur on an irregular time base the signal can be resampled, according to the given time base. The resampling is done by linear interpolation.

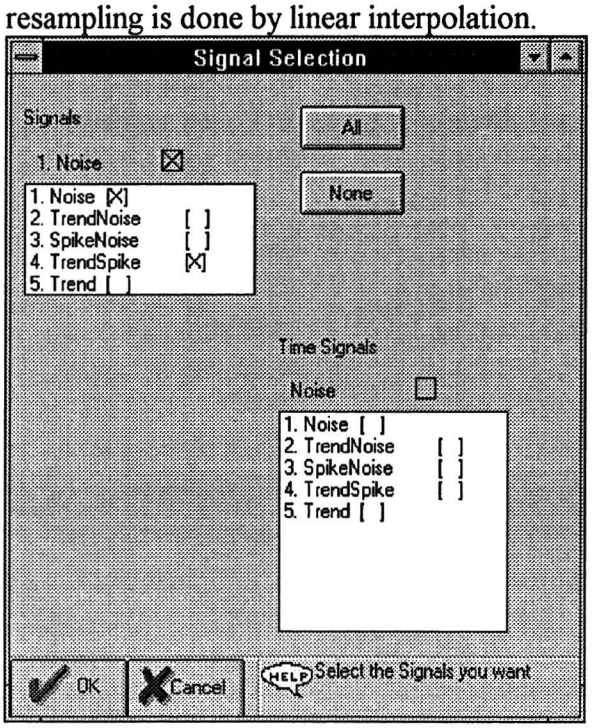

Figure 32: Signal Selection Dialog.

The left column are the signals. The column at the right bottom are the time signals. With the All-button all the signals can be selected at once. With the None-button the signals are deselected. If time signals are selected the second dialog (Figure 33) is entered.

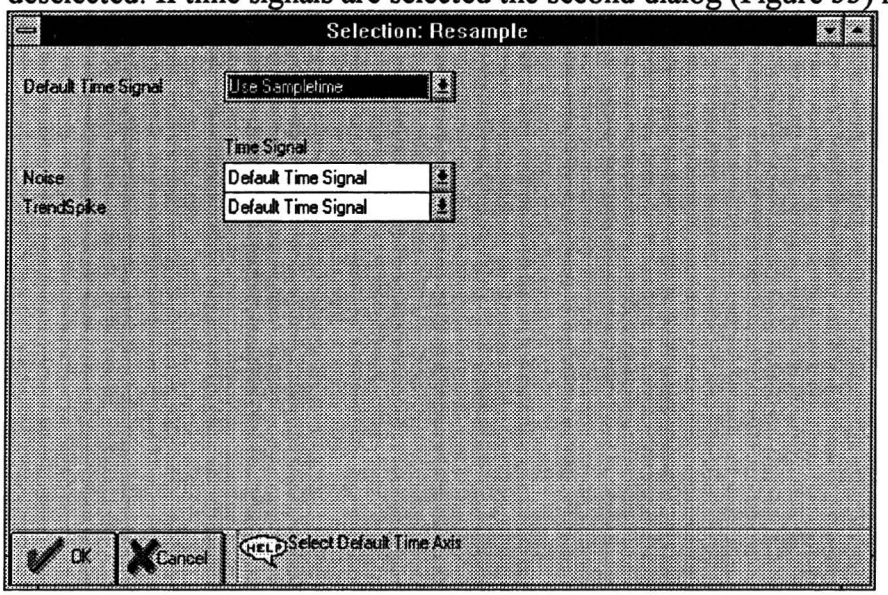

Figure 33: Resample dialog.

A Default Time Signal can be selected. This is the time signal that can be used as a time base for most of the signals. The following selection are possible:

- Use Sample time: the sample time is used as a time base, so the signal will not be resampled.
- Signal names that are selected as a time base in the first dialog.

Further the time base can be selected per signal. This can selection be:

- Default Time Signal: the default time signal is used as a time base.
- Use Sample Time.
- Time signals selected in the first dialog.

The next dialog is the Time Shift, Work Interval and Extrapolation dialog (Figure 34).

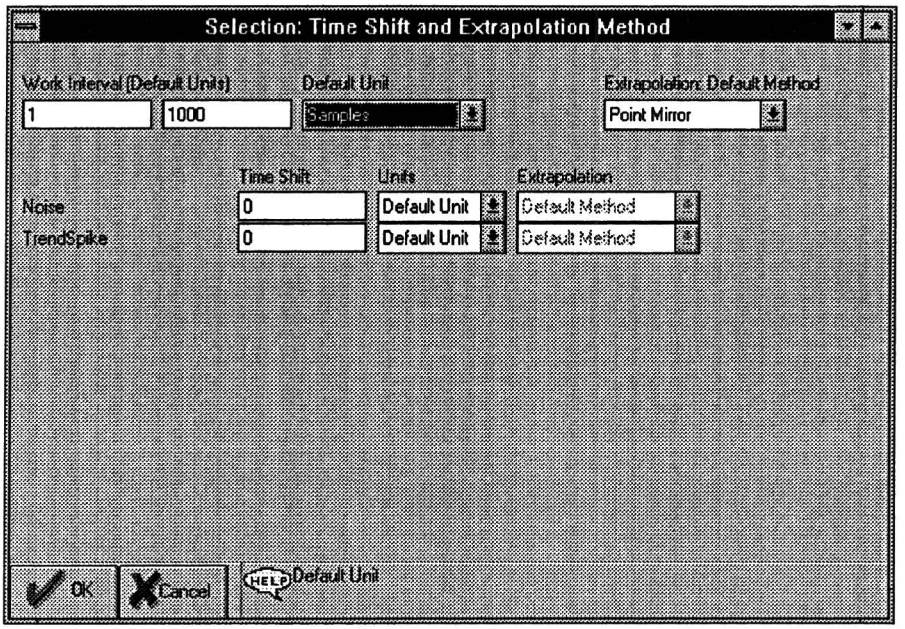

Figure 34: Time Shift, Work Interval and Extrapolation Method.

At the left-top corner the Work Interval can be selected. This is the part of the data of the original dataset that is used in the rest of the filter module. The work interval is given in Default Units. These default units can be: samples, seconds, minutes and hours. The work interval may exceed the available data, then the data will be extrapolated. At the right-top corner the Default Extrapolation Method can be selected. The following methods are available:

- Point Mirror: A polynomial is estimated on the first (or last) samples of the signal, and the signal is mirrored in the first (or last) point of the estimated polynomial. This has the advantage that, in the mirror point, the signal itself and the first derivative in time fits well to the extrapolated signal.
- Constant: A constant value, given by the user, is used as an extrapolation value.
- First/Last: When data is needed before the first sample, the first sample is used. When data after the last sample is needed, the last sample is used.
- Mean: The average value of the signal is used.

Further per signal Time Shift and Extrapolation Method (when necessary) can be given. With Time Shift the signal is shifted in time, it can be given in: Default Units, samples, seconds, minutes and hours. As an extrapolation method the Default Method, Point Mirror, Constant, First/Last and Mean can be chosen.

The last dialog (Figure 35) contains the extrapolation variables.

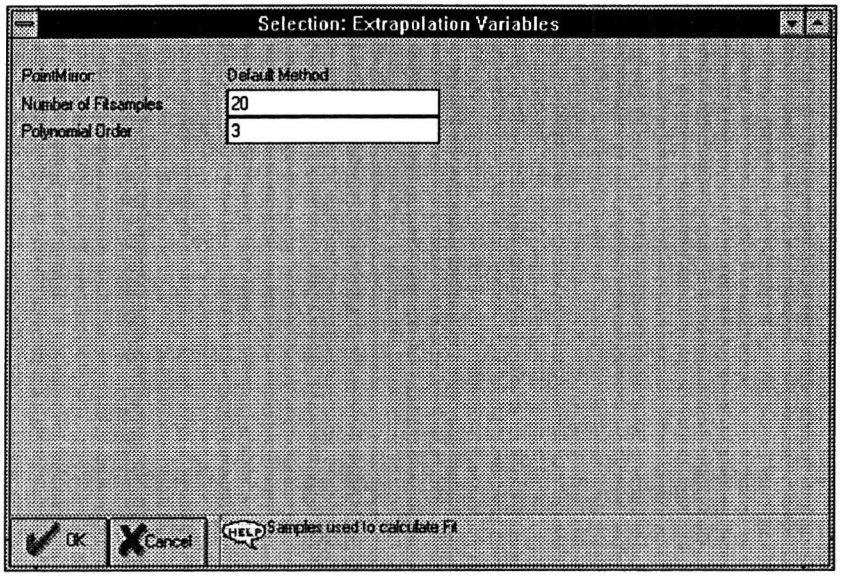

Figure 35: Extrapolation variables.

This dialog contains the variables of the Point Mirror method and the Constant method. If these methods are not used this dialog is skipped. The Point Mirror method has the following variables:

- Number of Fit samples: The number of samples at the beginning or ending of the data on which a polynomial is estimated.
- Polynomial Order: The order of the estimated polynomial.

The Constant method has a constant as a variable. this constant is used as the extrapolation value.

A.2 Main Filter Dialog

This dialog (shown in Figure 36) provides access to the filter functions, with the buttons:

- Outlier: The outlier detection and repair dialog.
- Filter: The filter dialog.
- Math: The mathematical operations dialog.

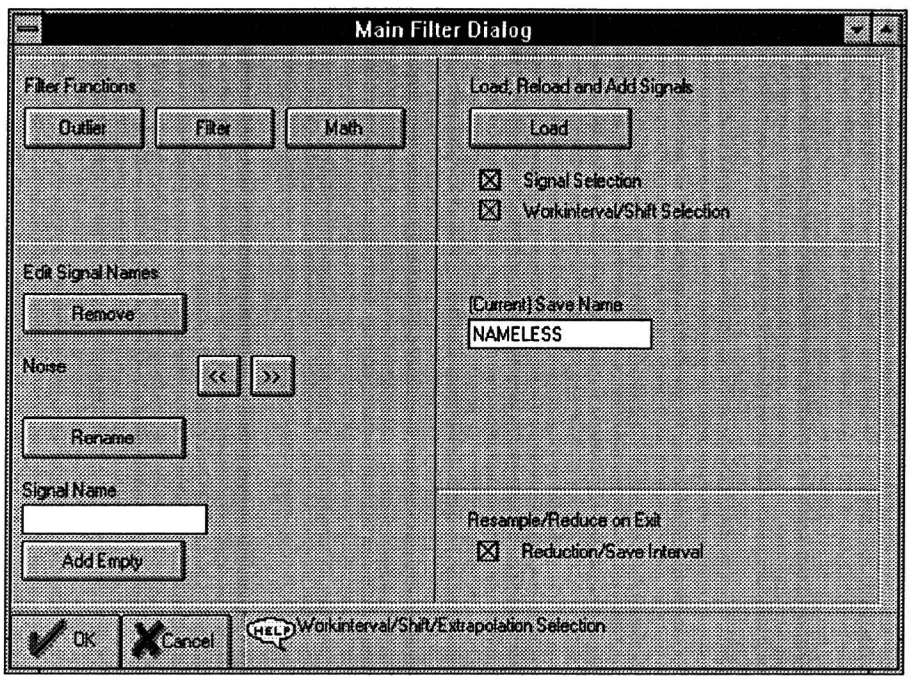

Figure 36: Main Filter Dialog.

With the Load-button signals can be loaded and reloaded from the original dataset. This is done by calling the selection, work interval and time shift dialog. When signals are not present in the work dataset the signals are added, if the are present they are reloaded. The selection dialogs can be partly skipped:

- If Signal Selection is not chosen the signal selection and the resample dialog will be skipped.
- If Workinterval/Shift Selection is not chosen the Time Shift, Work Interval and Extrapolation dialog and the Extrapolation variables dialog will be skipped.

The signals (the names) can be edited in the Edit-Signal-Names dialog part. A signal can be selected with the <<-button and the >>-button. It can be removed by pressing the Remove-button. When a Signal Name is given in the Signal Name box and when the Rename-button is pressed the selected signal will be renamed with the given name. The Add-Empty-button will add a new signal with the name given in the Signal Name Box. This signal will contain all zeros.

The save name can be altered in the (Current)-Save-Name-box.

If the Reduction/Save Interval is selected the Reduction and Save Interval Dialog will be called when the Main Filter Dialog is left. If it is not selected it will be skipped.

A.3 Outlier Detection and Repair Dialog

This function/dialog (see Figure 37) can detect outliers/spikes and (manually) repair data.

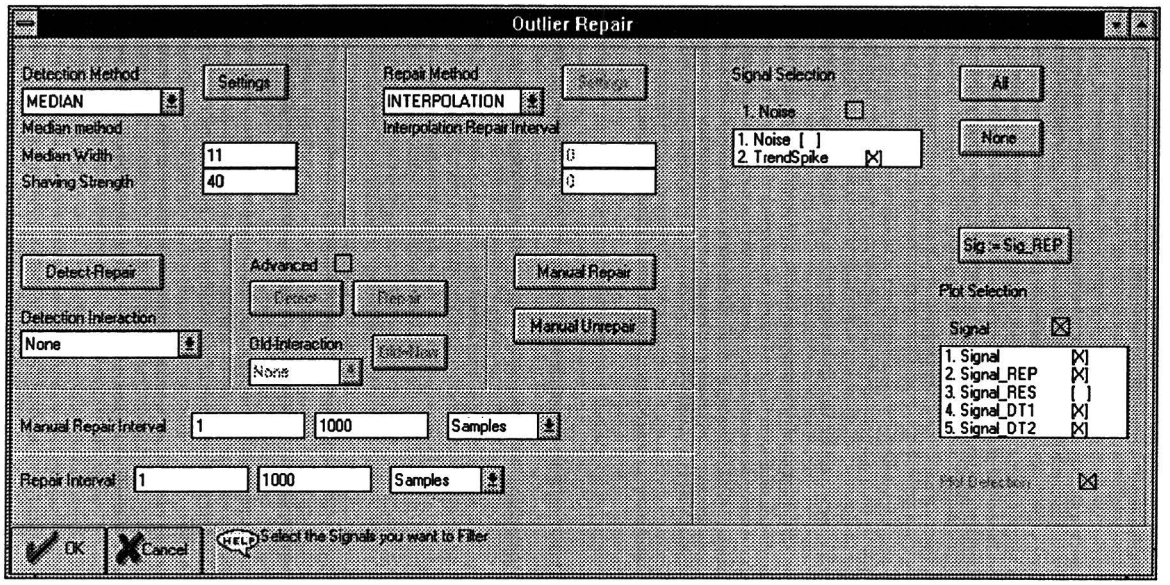

Figure 37: Outlier Detection and Repair Dialog.

In the part at the top-left corner the detection method can be selected. The dialog of the detection method can be called with the Settings-button. The following detection methods are available:

CLIP: an upper and lower detection bound can be given. This can be used when these bounds are known values.

BACKX: The lowpass filtered signal (with a second order Chebyshev filter) is taken as a trend (x_{Trend} , see 2.3.2). The standard deviation (σ) of the signal minus the trend is used. The cut off frequency of the lowpass filter and the shaving strength can be given.

TREND: The signal minus the highpass filtered signal is taken as a trend (see 2.3.2). The standard deviation of the highpass filtered signal is used. This method might be used for signals contaminated with slow drifts. The cut-off frequency of the highpass filter and the shaving strength can be given.

LEVEL: The average value of the signal is taken as a trend. The standard deviation of the signal is used. This method is recommended for process signals with no significant trends. The shaving strength can be given.

MEDIAN: The median filtered signal is taken as a trend. The standard deviation of the highpass filtered signal is used. The cut-off frequency of the highpass filter, the median width and the shaving strength can be given.

In the part at the top-middle the repair method can be selected. The dialog of the repair method can be called with the Settings-button. The following repair methods are available:

CONSTANT: The repair samples are replaced by a constant given by the user.

ADD VALUE: A constant, given by the user, is added to repair samples.

MEAN: The repair samples are replaced by the average value (the repair samples are excluded in the averaging) of the signal.

INTERPOLATION: The repair samples are replaced by a linear interpolation between the sample before and the sample after the repair sample(s). If the repair sample is the first or last sample of the data the mean repair method is used, because interpolation is not possible for the sample before the first sample does not exist.

In the part at the right side of the dialog the detect/repair signals and the plot signals can be selected. With the All-button all the signals are selected. With the None-button all the signals are deselected. The function creates a work dataset which contains signals with the following extensions:

- Signal: The signal itself.
- Signal_REP: The repaired signal.
- Signal_RES: The residu; the signal minus the repaired signal
- Signal_DT1: The lower detection bound.
- Signal_DT2: The upper detection bound.

Which of these signals is to be plot and used for detection and/or repair can be selected.

The Sig:=Sig_REP-button is the accept button. This button replaces the unrepaired data (Signal) with the repaired data (Signal_REP).

Under the plot selection the plot detection signal can be chosen to be plot. This signal is the detection signal provided by the advanced detect and repair functions.

At the left-bottom part of the dialog the repair interval can be selected. This can be done in samples, second, minutes and hours.

At the left-middle part are the Detect-Repair-button and the Detection-Interaction-select-box. With the Detect-Repair-button an automatic detect and repair can be done. The detection interaction can be:

- None: No interaction between the detections of the selected signals. Each signal has an independent detection.
- And: The detections of the selected signals are put together by means of a logical and. A detected outlier will only be considered as an outlier if it is detected in all of the selected signals. The same detection is used for the repair of all the signals.
- Or: The detections of the selected signals are put together by means of a logical or. A detected outlier will be considered as an outlier if it is detected in at least one of the selected signals. The same detection is used for the repair of all the signals.

The manual repair interval can be selected in samples, second, minutes and hours. With the Manual-Repair-button the manual repair interval can be repaired. With the Manual-Unrepair-button it can be unrepaired which means that the repaired signal is replaced by the original signal (in the manual repair interval).

When the advanced detect and repair is activated the Detect-button, Repair-button, Old=New-button, Old Interaction and Plot Detection become active. With the detect button a detection can be done independently from a repair. The result of this detection is put into a detection signal which can be shown by selecting the Plot Detection. Where an outlier is detected the signal is one, where no outlier is detected the signal is zero. A detect interaction can be done with the Old-detection. These interactions are:

- None: No interaction between the old detection and the current/new detection.
- And: The old detection and current/new detection will be put together by means of a logical and. A detected outlier will only be considered as an outlier if it is present in both detections.
- Or: The old detection and current/new detection will be put together by means of a logical or. A detected outlier will be considered as an outlier if it is present in at least one of the detections (or both).
- Remove: The detected outliers in the current/new are removed from the old detection.

The detect interaction between the selected signals with the Detect-button is the same as the interaction with the Detect-Repair-button. With the Old=New-button the old detection is replaced by the current/new detection. This is not automatically done once a detection has been made. With the Repair-button the selected signals can be repaired.

A.4 Filter Dialog

This function/dialog (see Figure 38) is the "real" filter function, that is a filter with one input and one output signal and some variables like a cut-off frequency or a median width. It can be used for various filter actions like lowpass filtering, the removal of trend or the removal of spikes.

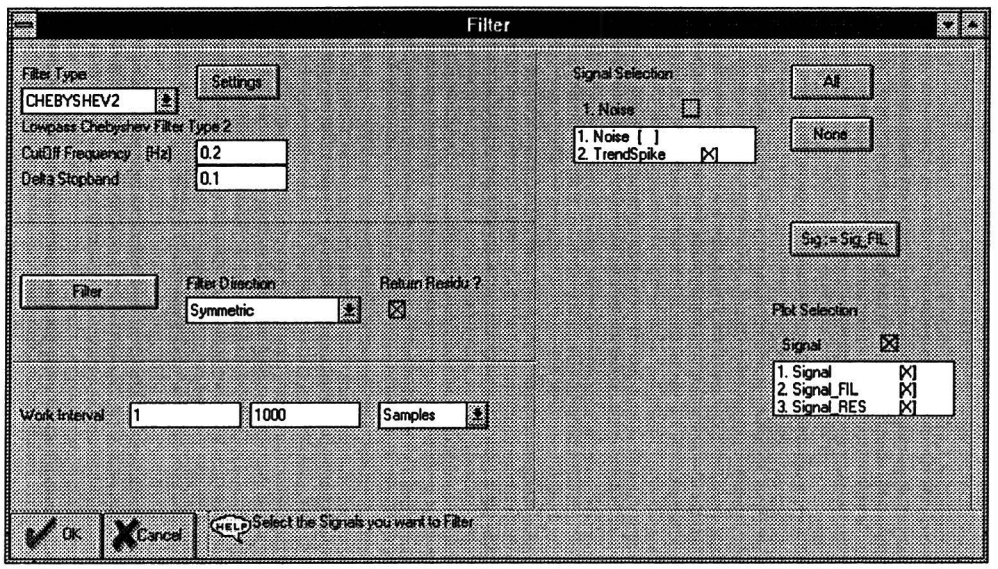

Figure 38: Filter Dialog.

In the part at the top-left corner the filter algorithm can be selected. The dialog of the filter can be called with the Settings-button. The following filters are implemented:

FIR: A finite order unit sample response sequence is obtained by truncating the infinite Fourier series and performing the z-transform (see 2.2.2 and 2.2.3). The oscillations occurring in the frequency response due to the truncation can be reduce by application of a window function. The type of window influences the maximum pass band and stop band ripple and the width of the transition band. Table 1 presents the window functions and some characteristical values of the FIR-filters designed with these windows.

Window	Maximum Stop Band Ripple (dB)	Transition Bandwidth (ω_S/M)
SQUARE	-21	0.9
HANNING	-44	3.1
HAMMING	-53	3.3
BLACKMAN	-74	5.5

Table 1: Characteristical features of the FIR-filter designed with the different window functions. For a given filter order M a smaller stopband ripple is exchanged for a larger transition bandwidth.

The following variables/settings can be given:

- Low Pass, High Pass, Band Pass and Band Stop.
- Window: Square, Hanning, Hamming, Blackman.
- Cut-off frequency (two frequencies if band pass or band stop).
- Transition range.
- Maximum filter size.

The transition range is the height of the stop band (see Figure 39). The Cut-off frequency is the cut-off frequency of the ideal (infinite) filter.

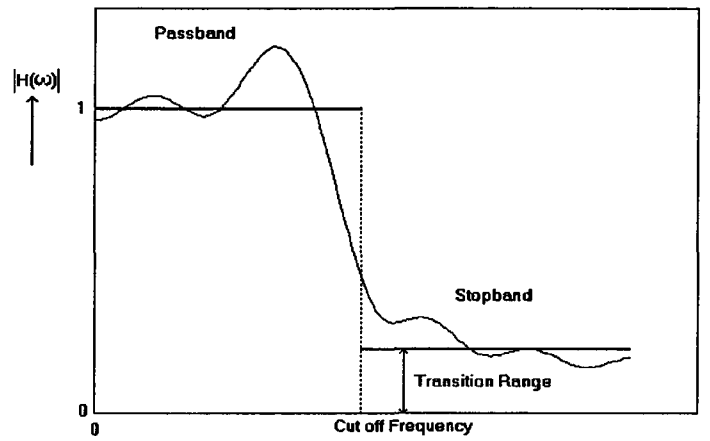

Figure 39: Frequency response of a FIR-filter.

BUTTERWORTH: The Butterworth approximation results from the requirement that the magnitude response be maximally flat in the passband. That is the first (2N-1, with N the order of the filter) derivatives of $|H(\omega)|^2$ are specified to equal zero at $\omega = 0$. This IIR-filter is designed in the s-plane and then transformed to the z-plane by use of the bilinear transformation (see 2.2.4 and 2.2.5).

The following variables/settings can be given:

- Low Pass, High Pass, Band Pass and Band Stop.
- Cut-off frequency (two frequencies if band pass or band stop).
- Delta passband.
- Delta stopband.
- Frequency Ratio, which is the ratio between the Transition frequency and the cut-off frequency in the continuous time (s-plane) design.
- Maximum filter size.

Most of these variables and a frequency response of a Butterworth filter are shown in Figure 40.

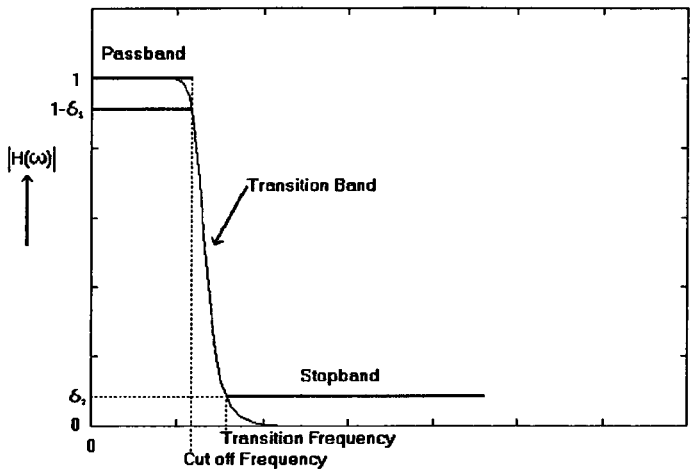

Figure 40: Frequency response of a Butterworth filter.

CHEBYSHEV1 and **CHEBYSHEV2**: A more rapid roll-off rate near the cut-off frequency than that of the Butterworth design can be achieved at the expense of a loss of monotonicity in the passband and/or the stopband. The Chebyshev design (types I and II) maintain monotonicity in one band but are equiripple in the other band as illustrated in Figure 41 and Figure 42. This IIR-filter is designed in the s-plane and then transformed to the z-plane by use of the bilinear transformation (see 2.2.4 and 2.2.5).

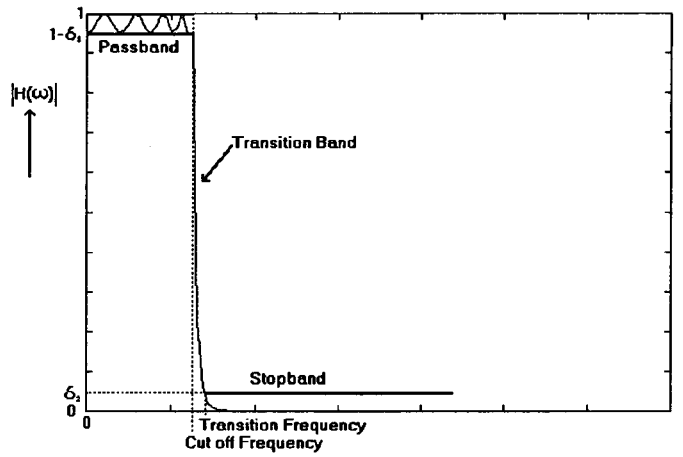

Figure 41: Frequency response of a Chebyshev Type I filter.

The Chebyshev Type I filter is equiripple in the passband.

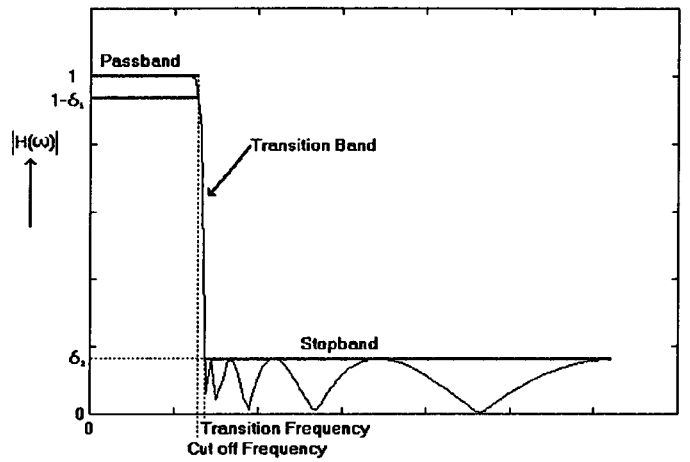

Figure 42: Frequency response of a Chebyshev Type II filter.

The Chebyshev Type II filter is equiripple in the stopband.

The variables/settings are the same as for the Butterworth filter, and are shown in Figure 41 and Figure 42.

MEDIANFL: This is a median filter, which takes the median of each sample, taken on a subinterval around the sample (see 2.2.7). It can be used for trend determination or spike removal. The median width can be given.

In the part at the right side of the dialog the filter signals and the plot signals can be selected. With the All-button all the signals are selected. With the None-button all the signals are deselected. The function creates a work dataset which contains signals with the following extensions:

- Signal: The signal itself.
- Signal_FIL: The filtered signal.
- Signal_RES: The residu; the signal minus the filtered signal

Which of these signals is to be plot and used for filtering can be selected.

The Sig:=Sig_FIL-button is the accept button. This button replaces the unfiltered data (Signal) with the filtered data (Signal_FIL).

At the left-bottom part of the dialog the Work Interval can be selected. This can be done in samples, second, minutes and hours. This is the interval on which the filtering is performed.

With the Filter-button the selected signals can be filtered. This can be done in three Filter Directions:

- Causal: Filtering forwards.
- Anti-causal: Filtering backwards.
- Symmetric: The average of causal and anti-causal.

If Return Residu is chosen the signal minus the filtered signal will be returned.

A.5 Mathematical Operations Dialog

With this function/dialog (see Figure 43) it is possible to perform mathematical functions and operations on the signal.

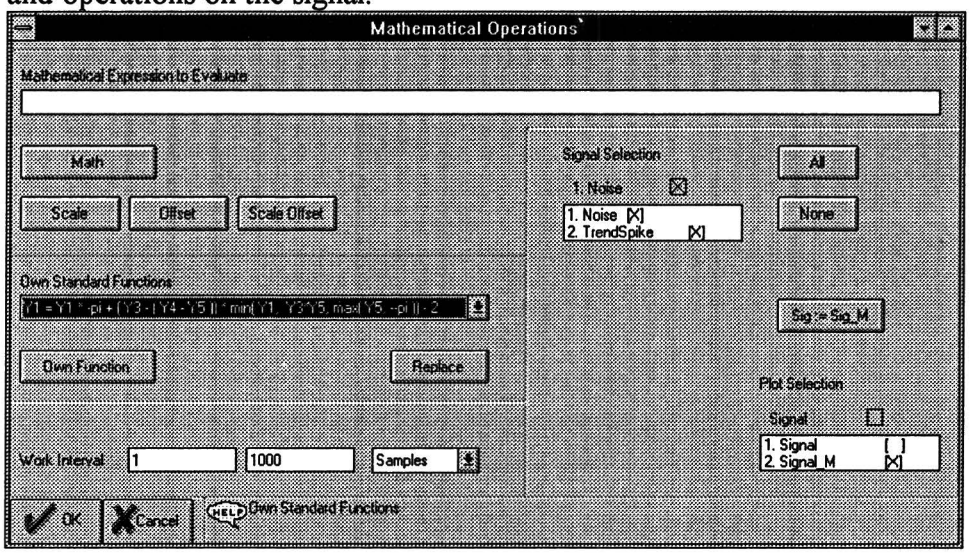

Figure 43: Mathematical Operations Dialog.

At the top of the dialog a Mathematical Expression can be entered. The syntax supports brackets, adding (+), subtracting (-), multiplying (*), dividing (/) and the following mathematical functions:

sum	mean	std	min	max	abs	ceil	floor	mod
sin	cos	tan	asin	acos	atan	atan2	sinh	cosh
tanh	sqrt	sqr	log	log10	exp	pow	div	

As a variable the signal names, x and time can be used. The variable x stands for the selected signals. The variable time stands for the (sample)time in seconds. The expression can be evaluated with the Math-button.

In the part at the right side of the dialog the math signals and the plot signals can be selected. With the All-button all the signals are selected. With the None-button all the signals are deselected. The function creates a work dataset which contains signals with the following extensions:

- Signal: The signal itself.
- Signal_M: The math-filtered signal.

Which of these signals is to be plot and are used as the variable x can be selected.

The Sig:=Sig_M-button is the accept button. This button replaces the "unfiltered" data (Signal) with the "filtered" data (Signal_M).

At the left-bottom part of the dialog the Work Interval can be selected. This can be done in samples, second, minutes and hours. This is the interval on which the mathematical expression is evaluated.

User defined expressions can be selected in the Own-Standard-Function list. These functions can be evaluated with the Own-Function-button. With the Replace-button the selected user defined expression is replaced by the expression in the Mathematical-Expression-to-Evaluate-box.

Further there are a Scale-button, a Offset-button and a Scale-Offset-button. These buttons evaluate the following expressions:

- Scale: x=x/std(x)
- Offset: x=x-mean(x)
- Scale-Offset: x=(x-mean(x))/std(x)

A.6 Reduction and Save Interval Dialog

With this function/dialog (see Figure 44) data reduction can be done and the save interval can be set.

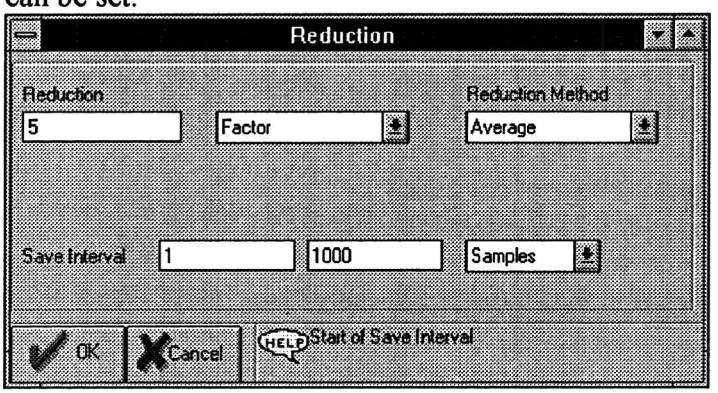

Figure 44: Data Reduction and Save Interval dialog.

At the top of the dialog the data reduction can be given with a Factor or by giving the number of desired samples. The data reduction can be done with two methods:

- Average: Which calculates the average values of each group (consisting of the same number of samples as the value of the reduction factor is) of samples. These averages form the reduced signal. The averaging is done by integrating in time and dividing trough the time interval. To make the signal continuous in time, the signal is linear interpolated.
- First: Which takes the first sample of each group of samples; decimation. If the number the first sample of a group is not an integer the signal is linear interpolated.

At the bottom of the dialog the Save Interval can be set. This can be done in samples, seconds, minutes and hours.

APPENDIX B: Dynamic Link Libraries

In Filter user defined DLL's can be used to add filter algorithms, outlier detection methods, repair methods and mathematical functions/operators. How a DLL works/is created is shown in Figure 45.

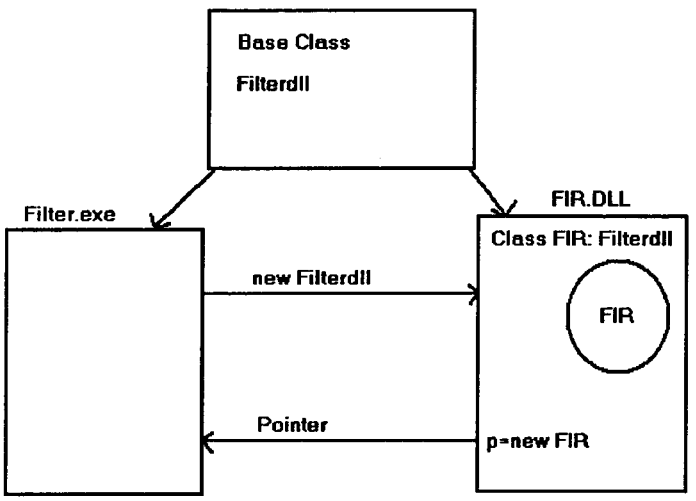

Figure 45: Creation/usage of a DLL

For starters a base class is needed (in this example Filterdll). This base class must be known in the main program (Filter.exe) and in the DLL (FIR.DLL). When the main program uses the DLL it calls the function new_Filterdll which function returns a pointer to the DLL. The main program can then use the DLL by means of this pointer.

In Filter there are 4 DLL-base classes:

- FILTERDLL For filter algorithms
- DETECTDLL For outlier detect methods
- REPAIRDLL For repair methods
- FUNCTIONDLL For mathematical functions/operations

Another important (in the Filter-DLL-context) class is Filterdialogdata.

B.1 FILTERDIALOGDATA

This data structure is used for the filter, outlier detection and repair functions. It contains the variables necessary for presentation in filter/detect/repair dialog (of the main program) and the variables of the filter/detect/repair algorithm. The following variables are defined:

```
PTEXT       HEADERTEXT;         // 3 x 32
      // HEADERTEXT(1)  Label of Own variable 1 (= variable of own
                        function in filter/outlier/repair)
      // HEADERTEXT(2)  Label of Own variable 2
```

```
      // HEADERTEXT(3)  Label/Title of DLL function        (Text
                        under filter/outlier/repair selectbox)
PMATRIX           VARUSED;           // 3
      // VARUSED(1)     Own Variable 1 used?      (1=variable is
                        used, 0=not used)
      // VARUSED(2)     Own Variable 2 used?
      // VARUSED(3)     Settings button used/own dialog defined
PMATRIX           VALUES;      // 2 x 2   (1,1)= Value var 1,
                               (1,2)= OWNINFO Position: 0=no
                               position/pointer to OWNINFO
      // VALUES(1,1)           Value of variable 1
      // VALUES(1,2)           Position/pointer to OWNINFO,
      // VALUES(2,1)           Value of variable 2
      // VALUES(2,2)           Position/pointer to OWNINFO
PMATRIX           TSAMPLE;
      // TSAMPLE(1)            SampleTime
PMATRIX           OWNINFO;           //  dim0
      // Parameters/variables of filter/detect/repair DLL-function
```

B.2 FILTERDLL

This is the base class for the filter algorithms. The following functions are defined:

checkvar	- Check variables for bounds.
dialog	- Dialog function.
sum	- Summation of one-dimensional matrix.
causal	- Causal filter, Signal, B and A are one-dimensional.
polmul	- Multiplication of (coefficients of) polynomials.
recimul	- Insert reciprocal into reciprocal.
mirror	- Mirror one-dimensional matrix.
bilin	- Bilinear transformation.
filter	- Filter function.

Checkvar checks the variables. This function is called when old variables are loaded and must check (and change) variables. This can be the case when for example the sample time has changed, which leads to a cut-off frequency that exceeds the Nyquist frequency. This function is also called by the filter dialog in order to check the boundaries of the variables that are presented in the filter dialog.

The dialog function can be replaced when the filter needs a dialog.

Filter is the filter function and can be replaced with the new filter.

The other functions are help/base functions that may be helpful to make new filters.

FIR.hpp:

This is (part of) the hpp-file of the FIR-filter. The typical DLL-statements are bold and underlined.

```
#include <filterdll.hpp>
```

```
class PMSEXPORT FIR : public FILTERDLL {
  private:
      HPMS  hApp,hWin;
      PREAL fnyq;

  public:
      virtual BOOL checkvar(int VarNr, PTEXT& ErrorText, ...
      virtual BOOL dialog(HPMS hAppl, HPMS hWindow, ...
      virtual PMATRIX filter(PMATRIX& Signal, FILTERDIALOGDATA& ....

};

extern "C" PMSEXPORT LPFILTERDLL new FILTERDLL(void)
{
      return new FIR;
}
```

B.3 DETECTDLL

This is the base class for the outlier detection methods. The following functions are defined:

checkvar	- Check variables for bounds.
dialog	- Dialog function.
detect	- The detect function.
sum	- Takes the sum of one-dimensional matrix.
causal	- A causal filter.
mirror	- Mirrors a one-dimensional matrix.
symmetric	- A symmetric filter.
mean	- Calculates the average value of (one-dimensional) signal.
std	- Calculates the standard deviation of (one-dimensional) signal.
filterIIR	- A second order Chebyshev filter.
medianfilter	- A median filter.

Checkvar and dialog do the same/have the same function as in FILTERDLL.

Detect is the outlier detect function. It must return two matrixes namely the trend (x_{Trend}, see 2.3.2) and the detection bound ($S\sigma$). It is not checked if the detection bound is negative. This can be used when detection methods do not fit into the trend-and-bound-concept. As a trend the signal can be returned. The bound can be made negative when an outlier is detected and positive when no outlier is detected.

The other functions are help/base functions that may be helpful to make new filters.

LEVEL.hpp

This is (part of) the hpp-file of the LEVEL-detection method. The typical DLL-statements are bold and underlined.

```
#include <detectdll.hpp>

class PMSEXPORT LEVEL : public DETECTDLL {
  private:
      HPMS  hApp,hWin;

  public:
      virtual BOOL checkvar(int VarNr, PTEXT& ErrorText, ...
      virtual BOOL detect(PMATRIX& Signal, PMATRIX& Trend, ...
};

extern "C" PMSEXPORT LPDETECTDLL new DETECTDLL(void)
{
      return new LEVEL;
}
```

B.4 REPAIRDLL

This is the base class for the repair methods. The following functions are defined:

checkvar	- Check variables for bounds.
dialog	- Dialog function.
SpikeLessMean	- Average without detected spikes.
repair	- Repair function.

Checkvar and dialog do the same/have the same function as in FILTERDLL.

Repair is the repair function and can be replaced with the new filter.

SpikeLessMean calculates the average of a signal but the spikes are excluded in the averaging.

INTERPOLATION.hpp

This is (part of) the hpp-file of the INTERPOLATION-repair method. The typical DLL-statements are bold and underlined.

```
#include <repairdll.hpp>

class PMSEXPORT INTERPOLATION : public REPAIRDLL {
  private:
      HPMS  hApp,hWin;

  public:
      virtual BOOL checkvar(int VarNr, PTEXT& ErrorText,
FILTERDIALOGDATA& parameters);  // VarNr=0 check/change settings on
opening, =1 Var1, =2 Var2 Check for Error, is TRUE if dispay error

      virtual PMATRIX repair(PMATRIX& Signal, PMATRIX& Buffer,
FILTERDIALOGDATA& parameters);       // repair function
};

extern "C" PMSEXPORT LPREPAIRDLL new REPAIRDLL(void)
{
      return new INTERPOLATION;
```

```
}
```

B.5 FUNCTIONDLL

This is the base class for the mathematical functions and operations. The following functions are defined:

getfunctionlist	- Returns a list of the implanted function/operators.
dofunction	- Calculates the function.

The functions are called by number not by name. This is the number of the function list (returned by getfunctionlist).

FUNCTIONS.hpp

This is (part of) the hpp-file of the functions that are currently implemented in the matrix interpreter. The typical DLL-statements are bold and underlined.

```
#include <functiondll.hpp>

class PMSEXPORT FUNCTIONS : public FUNCTIONDLL {

  public:
      virtual BOOL getfunctionlist(PTEXT& List);
      virtual BOOL dofunction(int Number, PMATRIX& inputmat ...

};

extern "C" PMSEXPORT LPFUNCTIONDLL new_FUNCTIONDLL(void)
{
      return new FUNCTIONS;
}
```

APPENDIX C: Cpp- and hpp-files of the program Filter

C.1 List of cpp/hpp-files

ADDVALUE	Repair-DLL, Add Value repair method
BACKX	Detect-DLL, Backx outlier detection method
BUTTERWORTH	Filter-DLL, Butterworth filter
CHEBYSHEV1	Filter-DLL, Chebyshev Type I filter
CHEBYSHEV2	Filter-DLL, Chebyshev Type II filter
CLIP	Detect-DLL, Clip method
CONSTANT	Repair-DLL, Constant method
DETECTDLL	Base class of DLL outlier detect methods
DETECTLINK	Dynamic link class of outlier detect methods
EXTRAPOL	Work interval and extrapolation, 3rd selection dialog
FILTER	Main program
FILTERDIALOGDATA	Structure for user defined filters, detect and repair methods
FILTERDLG	Main filter class
FILTERDLL	Base class of filter-DLL's
FILTERLINK	Dynamic link class of filters
FIR	Filter-DLL, FIR filter
FUNCTIONDLL	Base class of functions/operators matrix interpreter
FUNCTIONS	DLL with functions of matrix interpreter
FUNCTIONLINK	Dynamic link class of functions/operators matrix interpreter
INFOEXTRAPOL	Extrapolation variables, 4th selection dialog
INTERPOLATION	Repair-DLL, Linear interpolation method
INTROSELECTDLG	Signal selection, first selection dialog
LEVEL	Detect-DLL, Level outlier detect method
MEAN	Repair-DLL, Mean repair method
MEDIAN	Detect-DLL, Median outlier detect method
MEDIANFL	Filter-DLL, Median filter
OPERATORS	DLL with operators of matrix interpreter
OWNINTERP	Interpreter of PMATH
PFILTER	Filter class
PLOTUPDATE	Update plot function
PMATH	Mathematical operations class
PMATINTERP	Base class matrix interpreter
POLYNOMFIT	Function that estimates polynomial on data sequence
POLYNOMFUN	Function that calculates value the value of a polynomial
POUTLIER	Outlier detection and repair class
PREDUCE	Data reduction class
PSELECTION	Selection, work interval, extrapolation class
REPAIRDLL	Base class of repair-DLL's
REPAIRLINK	Dynamic link class of repair methods
RESAMPLEDLG	Time signal selection, second selection dialog
TREND	Detect-DLL, Trend outlier detection method

C.2 Structure cpp/hpp-files of program Filter: