CODEN: LUTFD2/(TFRT-5354)/0–86/(1986)

Modelling and Identification of Fetal Aorta Dynamics

Anders Svensson

Department of Automatic Control
Lund Institute of Technology
September 1986

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
Master Thesis

Date of issue
September 1986

Document Number
CODEN: LUTFD2/(TFRT-5354)/0-86/(1986)

Author(s)
Anders Svensson

Supervisor
Rolf Johansson

Sponsoring organisation

Title and subtitle
Modelling and Identification of Fetal Aorta Dynamics

Abstract

This thesis treats the problem of estimating elasticity and peripheral resistance of a fetal aorta. The aorta wall dynamics is represented in a simple physical model. Radius variations used in the model are measured with an existing equipment , used in the Malmö Public Hospital , Malmö Allmänna Sjukhus (MAS).

The model parameters are identified directly in the continuous time model. This is accomplished when signal processing filters are applied to the measured radius variations and the filtered measurements are used in a Recursive Least Square algorithm.

Finally , the identification algorithm and signal processing filters are implemented in a personal computer.

Key words
Ultrasonic measurements, Dynamic model, Parameter estimation, Signal processing filters, Recursive identification, Personal computer implementation.

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
86

Recipient's notes

Security classification

Modelling And Identification Of Fetal Aorta Dynamics

Anders Svensson
Department of Automatic Control
Lund Institute of Technology
Lund , Sweden

Abstract

It is often of great importance to be able to detect fetal growth retardations. When ultrasonic equipment was introduced in this area of medicine , a major breakthrough was done.

This thesis treats the problem of estimating elasticity and peripheral resistance of a fetal aorta. The aorta wall dynamics is represented in a simple physical model. Radius variations used in the model are measured with an existing equipment , used in the Malmö Public Hospital , Malmö Allmänna Sjukhus (MAS).

The model parameters are identified directly in the continuous time model. This is accomplished when signal processing filters are applied to the measured radius variations and the filtered measurements are used in a Recursive Least Square algorithm.

Finally , the identification algorithm and signal processing filters are implemented in a personal computer.

Key words : Ultrasonic measurements , Dynamic model , Parameter estimation , Signal processing filters , Recursive identification , Personal computer implementation.

Preface

This master thesis was carried out at the Department of Automatic Control , Lund Institute of Technology , Lund , Sweden , from June to August 1986.

It has been very stimulating to work at the Department and I have learned a lot , not only in the subject of this thesis. First of all I would like to thank my supervisor Rolf Johansson at the Department for a great support. He proposed the model structure for me to work with and throughout the work he provided me with advices and hints whenever I needed them. The staff at the Department have been very helpful with hints concerning simulations , programming and this report.

In a seminar at the hospital in Malmö , I got some valuable information from professor Gerhard Gennser and the people around him. I also received some propositions for a future study.

For the reader I would like to point out that this is merely a first approach to the modelling and identification of fetal aorta dynamics. A lot of work is still to be done.

Lund , August 1986
Anders Svensson

Contents

Introduction

This study is devoted to methods for detection of fetal growth retardation. When ultrasonic measurement equipment was introduced , a great improvement was reached in this field. With ultrasonic measurements the examiner can get a good opinion of a fetals growth and detect external and internal malfunctions. Some internal malfunctions are however difficult to detect.

The subject of this master thesis was originally posed as a problem by professor Gerhard Gennser and his group at the Malmö Public Hospital (MAS). They use an ultrasonic equipment for recordings of the pulsatile motion of a fetal aorta. Their interest was to develop a method for estimation of elasticity and peripheral resistance (which is a sort of flow resistance) of a fetal vessel , mainly the aorta.

To fulfil this task a model of the vessel is first constructed. The aim is to create a simple model with as few parameters as possible. An electronic circuit analogy may be used to describe the model structure. Basic force and volume balances are used for the model. Since it is possible to measure the radius in two different positions at the same time , two models are developed. The first model exploits only one radius measurement and makes an assumption of the flow. The second uses both available measurements. Both models are linearized versions of the original nonlinear models. This is a advantageous in order to use the identification method. A number of simulations are done to verify the models. Model building is treated in chapter 1.

When the model building is completed a choice of identification method is done. Instead of using the common ARMA model identification , which requires a transformation to a discrete time model , the continuous time model is used directly. This is possible when input signals are filtered using signal processing filters with low pass characteristics. A transformation is introduced , but it is linear and simple contrary to the case with ARMA models. An ordinary Recursive Least Square algorithm is then used on the filtered input signals. To present the results of identification , several simulations are done. The identification method and its properties are treated in chapter 2.

Most of this work is spent on an implementation program. A personal computer was used. The filters are implemented in the computer as digital filters. Some graphics possibilities for data and parameter plots are included. The program runs in real time. The implementation is treated in chapter 3. In Appendix B the program listing is given , together with some comments. There is also a Users Guide in Appendix C.

In chapter 4 a discussion is held over the results in previous chapters. Some propositions for a future study in the subject of this thesis are also given.

Finally in Appendix A , listings of all simulation modules are provided. The macros for generation of figures are also given here.

1. Model Building

Motivation

There has been a number of works on modelling the dynamics of a blood vessel. Some of them treats curve fitting , see Thompson et al. (1985) , where no physical parameters are used . It may be misleading only to use the curve forms , when determining the physiological characteristics of a vessel . It is obvious that the curves changes with changes in cardiac output , does not correspond to changes in vessel parameters. There are also works treating this modelling problem that uses sophisticated models with lots of parameters , see Borgström et al. (1982). The main goal of this work , was to develop a simple model based on volume and mass balances , with only a few physical parameters. Therefore these methods and models were rejected.

Accurate modelling of vessel dynamics in all points of space requires a partial differential equation (pde) approach , but this automatically increases the complexity. When it was clear that an implementation with two points of radius measurements was possible to achieve , also a model with time delay could have been used. Here as well as with pde the complexity increases.

We have chosen to model the vessel dynamics by characteristics such as peripheral resistance and vessel wall elasticity. Hereby the model may be described by the electronic circuit analogy , see figure 1.1 , giving the model low pass characteristics. The capacitor C models volume balance in a vessel with a wall under tension and the resistor models a sort of flow resistance , called peripheral resistance R. The voltage and current of an electronic circuit are replaced by pressure and flow.

One Point Model

First a model for a one point measurement of the pulsative motion of the vessel radius was developed. The vessel is supposed to be tubelike with a circular cross section as shown in figure 1.2 . Here r stands for the radius , h for the vessel wall thickness and ℓ for the length of a considered vessel.

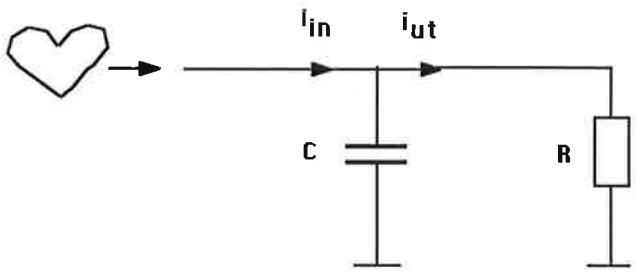


Figure 1.1 : Electronic circuit analogy.

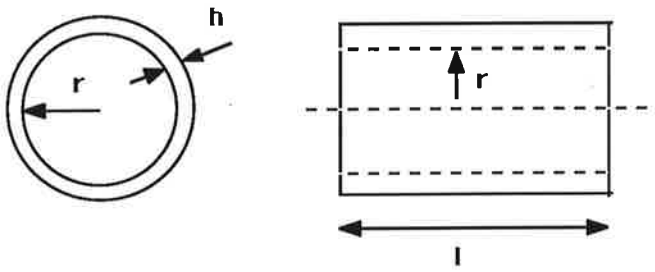


Figure 1.2 : A tubelike vessel

A pulsatile pressure denoted p is acting in the vessel. Due to this pressure there is a tension in the wall. This leads to a force equilibrium , as shown in figure 1.3 , where σ is the wall tension. Force equilibrium gives

$$2\sigma h\ell = 2r\ell p \qquad (1.1)$$

The ℓ should not be thought of as the length of a long tubelike vessel. Instead it is the length along which the pressure and therefore also the tension is supposed to be constant*) . Equation (1.1) leads to an expression for the wall tension σ

---

*) Another approach is to consider ℓ to be the equivalent distance from the heart to the measurement point. The measured radius variation is then used as an average value for the variations in a vessel with length ℓ.

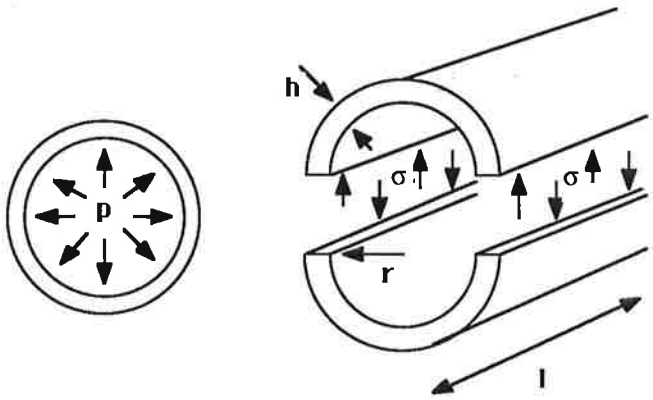


Figure 1.3 : Force equilibrium

$$\sigma = \frac{r}{h} p \tag{1.2}$$

The wall thickness h is supposed to vary during the pulsatile motion . One approach is to let the vessel wall volume be constant. This leads to the approximative equation

$$2\pi r_0 h_0 = 2\pi r h \tag{1.3}$$

where r_0 and h_0 are the radius and the thickness when there is no tension in the wall. From (1.3) the wall thickness can be written

$$h = \frac{r_0}{r} h_0 \tag{1.4}$$

When combining (1.2) and (1.4) the wall tension σ becomes

$$\sigma = \frac{r^2}{r_0 h_0} p \tag{1.5}$$

Using Hooke's law

$$\sigma = E\varepsilon \tag{1.6}$$

where E is Young's module of elasticity and ε is the relative increase in the vessel perimeter , which is approximately

$$\varepsilon = \frac{r - r_0}{r_0} \tag{1.7}$$

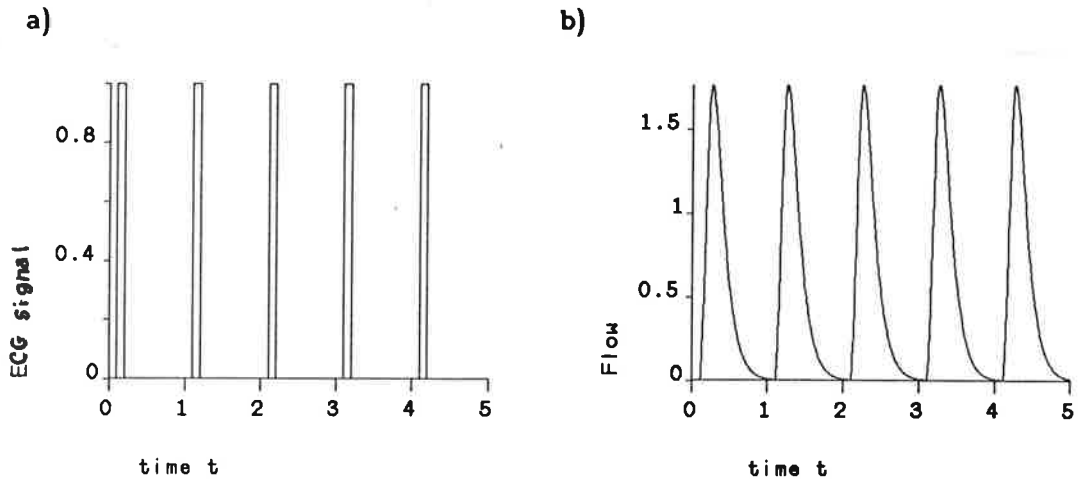


Figure 1.4 : Model signals : a) ECG signal b) Input flow

Joining (1.5) and (1.7) with the help of (1.6) leads to an expression for the pressure

$$p = \frac{Eh_0}{r^2}(r - r_0) \tag{1.8}$$

The volume stored in a tube of length ℓ is

$$V = (\pi r^2 - \pi r_0^2)\ell \tag{1.9}$$

Next a differential equation for the volume balance may be formulated

$$\frac{dV}{dt} = i_{in} - i_{out} \tag{1.10}$$

where i_{in} and i_{out} are the input and output flow , to and from a measurement point in a tubelike vessel with length ℓ. Compare with the electronic circuit analogy in figure 1.1. The input flow i_{in} is assumed to be a pulse response of the form

$$h(t) = bte^{(-at)} \tag{1.11}$$

It is triggered by an ECG signal. The time dependency is shown in figure 1.4. This figure together with all the other results of simulations in this work are generated with the program package Simnon , see Åström (1985). Simnon systems and macros for generation of figures are listed in Appendix A. If (1.10) is

evaluated using (1.9) the differential equation becomes

$$\frac{dV}{dt} = 2\pi r \frac{dr}{dt} \ell \tag{1.12}$$

The output flow i_{out} may be formed from figure 1.1 , through an equivalent to Ohm's law , i.e

$$i_{out} = \frac{p}{R} \tag{1.13}$$

where p is the pressure and R is the peripheral resistance. Using (1.8) on (1.13) leads to

$$i_{out} = \frac{Eh_0}{R} \frac{r - r_0}{r^2} \tag{1.14}$$

and when (1.14) is joined with (1.10) and (1.12)

$$2\pi r \frac{dr}{dt} \ell = i_{in} - \frac{Eh_0}{R} \frac{r - r_0}{r^2} \tag{1.15}$$

or

$$\frac{dr}{dt} = \frac{1}{2\pi\ell} \frac{i_{in}}{r} - \frac{Eh_0}{2\pi\ell R} \frac{r - r_0}{r^3} \tag{1.16}$$

When efforts were made to identify the parameters in equation (1.16) , the results were poor. This is probably due to bad excitation because of the combination of input and output signals , i.e. i_{in}/r , as a factor of the equation.

Therefore the differential equation was rewritten in terms of r^2 , i.e.

$$\frac{dr^2}{dt} = \frac{Eh_0 r_0}{\pi R\ell} \frac{1}{r^2} - \frac{Eh_0}{\pi R\ell} \frac{1}{\sqrt{r^2}} + \frac{1}{\pi\ell} i_{in} \tag{1.17}$$

Since equation (1.17) is nonlinear , identification of such a model using signal processing filters should not be possible , except for a special filter constant. This was in fact the result of some brief simulations and identifications. Therefore this model was rejected.

Next an approximation of equation (1.17) was made. The equation was linearized around $r = r_0$. This gives

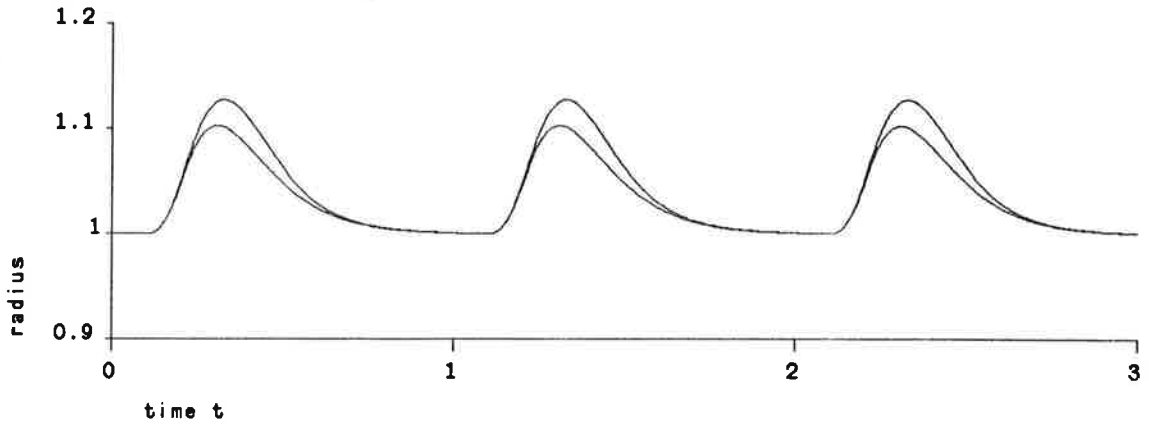


Figure 1.5 : Simulated radius of nonlinear/linear model.

$$\frac{dr^2}{dt} = - \frac{Eh_0}{2\pi R\ell r_0^3} r^2 + \frac{Eh_0}{2\pi R\ell r_0} + \frac{1}{\pi\ell} i_{in} \qquad (1.18)$$

or

$$\frac{dr^2}{dt} = - \alpha_1 r^2 + \alpha_2 + \alpha_3 i_{in} \qquad (1.19)$$

$$\alpha_1 = \frac{Eh_0}{2\pi R\ell r_0^3} \qquad \alpha_2 = \frac{Eh_0}{2\pi R\ell r_0} \qquad \alpha_3 = \frac{1}{\pi\ell}$$

Equation (1.19) is a linear model of the vessel dynamics. It is well suited for identification using low pass signal processing filters , as will be shown in the following chapter. Figure 1.5 shows simulations of , equations (1.17) and (1.18). The linear equation generates the smallest amplitude.

Two Point Model

The possibility to measure the radius at two points along the vessel at the same time , led to a somewhat different model . A weakness of the One Point Model is of course the assumption of the input flow. This problem is , as will be seen avoided in the Two Point Model. The tubelike vessel for this case is shown in figure 1.6 , where r_A and r_B are the radii , and h_A and h_B denote the wall thickness in the two measurement points A and B. It should be pointed out that point A is closer to the heart. According to equation (1.4) the wall thickness may

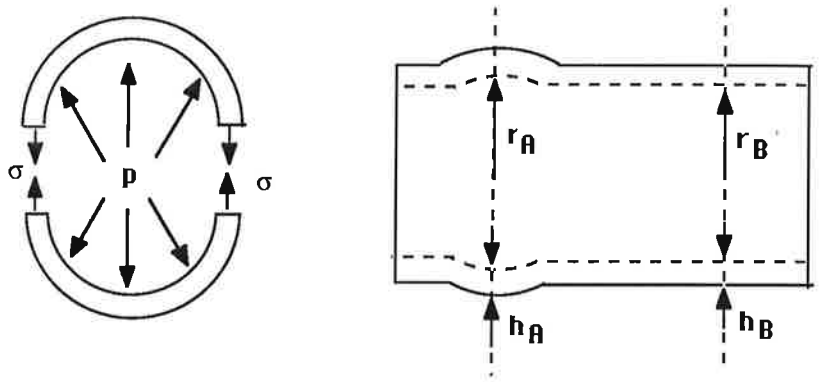


Figure 1.6 : A two point tubelike vessel.

be written

$$h_A = \frac{r_{0A}}{r_A} h_{0A} \tag{1.20}$$

$$h_B = \frac{r_{0B}}{r_B} h_{0B} \tag{1.21}$$

The distance between point A and B is not a factor in the model. Since the Two Point Model is based on the assumption that output flow of point A and input flow of point B are the same , the distance must be rather small. This is a circumstance that should be tested clinically*) .

The two differential equations are , compare with equation (1.10) , for point A

$$\frac{dV_A}{dt} = i_{inA} - i_{outA} \tag{1.22}$$

$$\frac{dV_B}{dt} = i_{inB} - i_{outB} \tag{1.23}$$

$$i_{inB} = i_{outA} \tag{1.24}$$

---

*) Another approach is to use two distances , ℓ_A and ℓ_B , and consider ℓ_A as the equivalent distance from the heart to measurement point A. ℓ_B is then the distance between the two measurement points A and B. In this case the radius measurements are used as mean values for each distance.

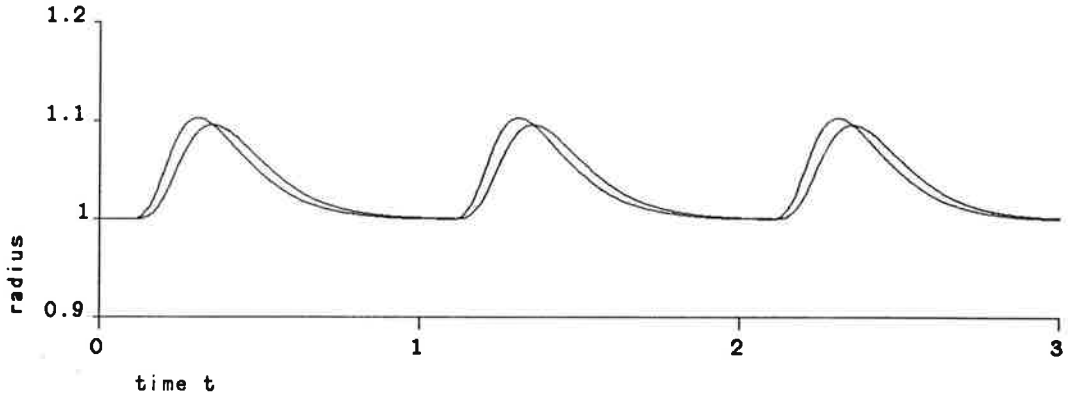


Figure 1.7 : Simulated radius variations in points A and B.

Using (1.24) and (1.18) to evaluate (1.23) leads to a differential equation in r_B^2

$$\frac{d}{dt}(r_B^2) = \frac{E_A h_{0A}}{2\pi R_A \ell r_{0A}^3}(r_A^2) - \frac{E_A h_{0A}}{2\pi R_A \ell r_{0A}} - \frac{E_B h_{0B}}{2\pi R_B \ell r_{0B}^3}(r_B^2) - \frac{E_B h_{0B}}{2\pi R_B \ell r_{0B}} \qquad (1.25)$$

or

$$\frac{d}{dt}(r_B^2) = -\beta_1(r_B^2) + \beta_2(r_A^2) + \beta_3 \qquad (1.26)$$

where

$$\beta_1 = \frac{E_B h_{0B}}{2\pi \ell R_B r_{0B}^3} \qquad \beta_2 = \frac{E_A h_{0A}}{2\pi \ell R_A r_{0A}^3}$$

$$\beta_3 = \frac{1}{2\pi\ell}\left[\frac{E_B h_{0B}}{R_B r_{0B}} - \frac{E_A h_{0B}}{R_A r_{0A}}\right]$$

Equation (1.26) was simulated using the input flow of equation (1.11). The result can be seen in figure 1.7. In this figure the low pass characteristics of the model are clear. The amplitude is smaller in point B , but the duration in time is longer than in point A. This is a typical action of a low pass filter.

2. Identification

Motivation

The most commonly used algorithms for parameter estimation are based upon ARMA models , see e.g. Söderström (1984). ARMA model identification is a widespread technique , but of course it requires a transformation to a discrete time system. Due to this transformation , a parameter in the discrete time system becomes nonlinearly dependent on all the parameters of the continuous time model. Discrete time parameters depend also on the length of the sampling interval. Parameters estimated in a blood vessel model are only meant for monitoring , and not for control system synthesis. Therefore a short linear connection between the estimated parameters and the model is preferable. It is then a good thing if identification can be done directly in the continuous time model and in fact this is the case.

Operator translation

One method for a direct identification of a continuous time model , is to use signal processing filters of low pass type , for filtering of variables. Techniques of this sort are dealt with in a number of works , see e.g. Young (1965, 1969 ,1981). In a paper by Johansson (1986) , an operator translation is suggested in order to get a system suited for identification. The linear continuous time system is translated using the low pass filter operator

$$z = \frac{1}{1 + p\tau} \tag{2.1}$$

where $p = d/dt$ and τ is the filter constant. The differential operator of (2.1) may also be written

$$p = \frac{1}{z\tau}\left[\frac{1 - z}{z}\right] \tag{2.2}$$

Identification Algorithm

The algorithm used in the simulations and in the implementation program , is a well known Recursive Least Square method , see e.g. Söderström (1984). Below , only the equations for updating of parameter vector Θ , error ϵ , weighing vector K and the P-matrix are given

$$\hat{\theta}(t) = \hat{\theta}(t-1) + K(t)\epsilon(t) \tag{2.3}$$

$$\epsilon(t) = y(t) - \varphi^T(t)\hat{\theta}(t-1) \tag{2.4}$$

$$K(t) = \frac{P(t-1)\varphi(t)}{\lambda + \varphi^T(t)P(t-1)\varphi(t)} \tag{2.5}$$

$$P(t) = \frac{1}{\lambda}\left[P(t-1) - \frac{P(t-1)\varphi(t)\varphi^T(t)P(t-1)}{\lambda + \varphi^T(t)P(t-1)\varphi(t)} \right] \tag{2.6}$$

The vector φ contains filtered variables , λ is a forgetting factor and y is the state. A special feature when signal processing filters are used , is that the start of identification is delayed in order to wait out filter transients. The delay is set to three times the filter constant τ.

One Point Model

Usage of the operator translation of (2.2) on equation (1.19) leads to the transformed model

$$r^2 = (1 - \alpha_1\tau)[zr^2] + \alpha_2\tau[z1] + \alpha_3\tau[zi_{in}] \tag{2.7}$$

The φ and Θ vectors of the Recursive Least Square algorithm are then

$$\varphi^T = [\ zr^2 \quad z1 \quad zi_{in}\ ]$$

$$\Theta^T = [\ (1 - \alpha_1\tau) \quad \alpha_2\tau \quad \alpha_3\tau\ ]$$

It is now simple to make a translation back to α parameters

$$\alpha^T = [\ (1 - \Theta_1)/\tau \quad \Theta_2/\tau \quad \Theta_3/\tau\ ]$$

If α_2 is divided by α_1 , the result is r_0^2 , so it is possible to identify the equilibrium radius as well. Since ℓ and π are the only factors in α_3 , also ℓ is

possible to identify. The resulting parameter after reduction is

$$\alpha' = \frac{Eh_0}{R}$$

To test the behaviour of the algorithm , the Recursive Least Square algorithm and the signal processing filters were programmed in Simnon together with systems describing the time dependency of the radius , i.e. equation (1.19) , and systems describing the output flow of the heart , according to equation (1.11) , see appendix A. In all simulations the P-matrix is initiated with 10000 in the diagonal elements , λ is 1 and h , the sampling increment of the Recursive Least Square algorithm is set to 0.05 time units if nothing else is stated. As can be seen in the previous chapter the period of the radius motion is approximately 1 time unit. The model parameters are given the following values

$$E = 50 \qquad h_0 = 1 \qquad r_0 = 1 \qquad l = 0.1 \qquad R = 1$$

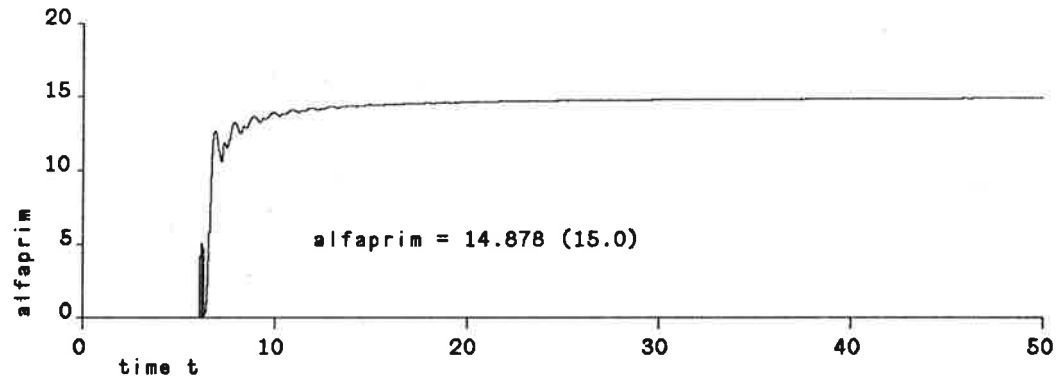
The estimated parameter is α'. Figure 2.1 shows the results of simulations , using some different values for the filter constant τ. Remember that the start of identification is delayed three filter constants. The choice of τ is not crucial. The convergence is fast and correct for at least two orders of magnitude of τ. The values of α' are given in the figure. The correct value is 15.0.

A parameters convergence is of course also dependent on the sampling increment of the Recursive Least Square algorithm. Results from simulations showing this effect can be seen in figure 2.2. τ , the filter constant is here 1. A shorter sampling increment gives a faster convergence. The sampling increment should be chosen corresponding to the desired fastness and correctness.

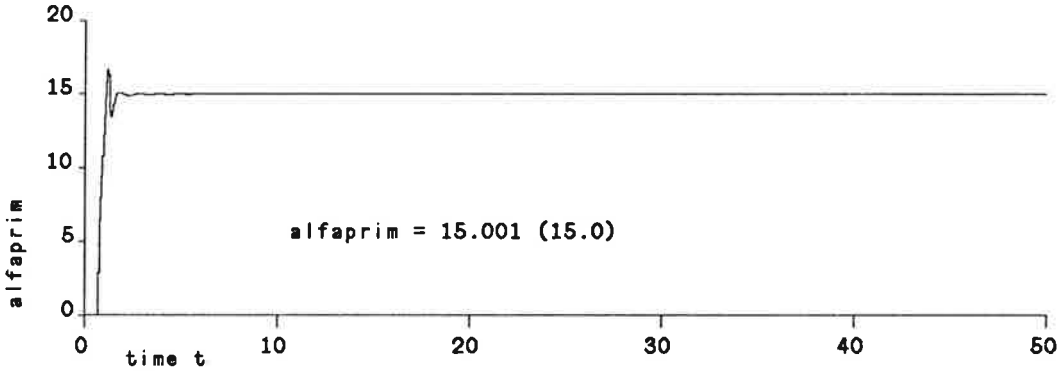
So far any influence from disturbances on the system has not been considered. There are several imaginable sources of disturbances. One is of course noise. The influence of noise on the radius measurement can be seen in figure 2.3. In figures 2.4 and 2.5 the influence of different λ and τ is shown.

Figure 2.4 shows that λ should be given a value of 1 , or perhaps a little less in order to forget other disturbances than noise. Figure 2.5 shows that as the filter constant τ is set to smaller values the influence of noise on the system increases. This is quite natural , since with smaller filter constants , more of the noise

a)



b)



c)

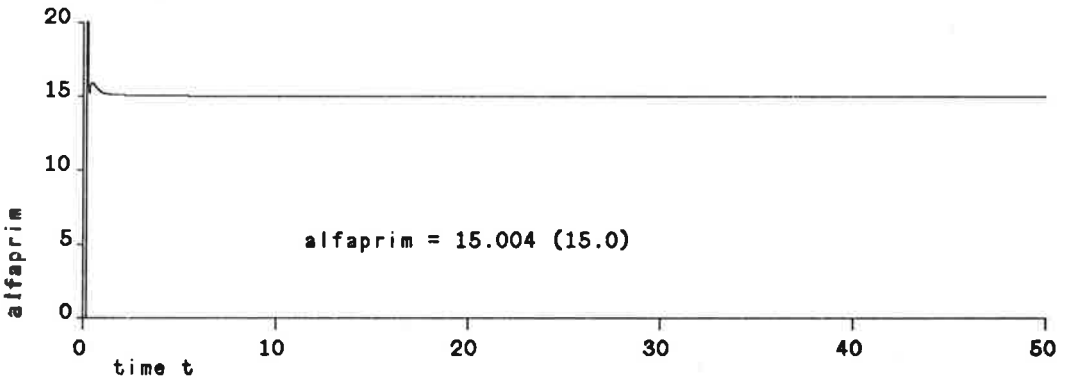


Figure 2.1 : Simulated convergence of α'. a) $\tau = 2$ b) $\tau = 0.2$ c) $\tau = 0.02$

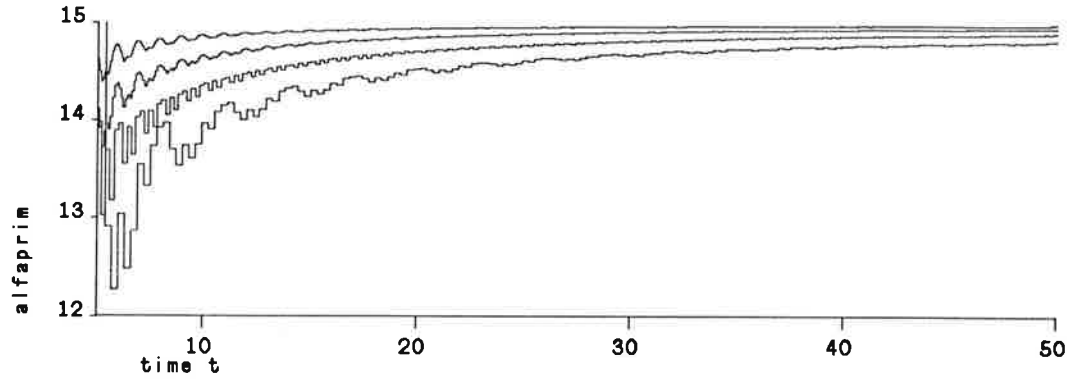


Figure 2.2 : Simulated convergence of α'. h = 0.05 , 0.1 , 0.2 , 0.3

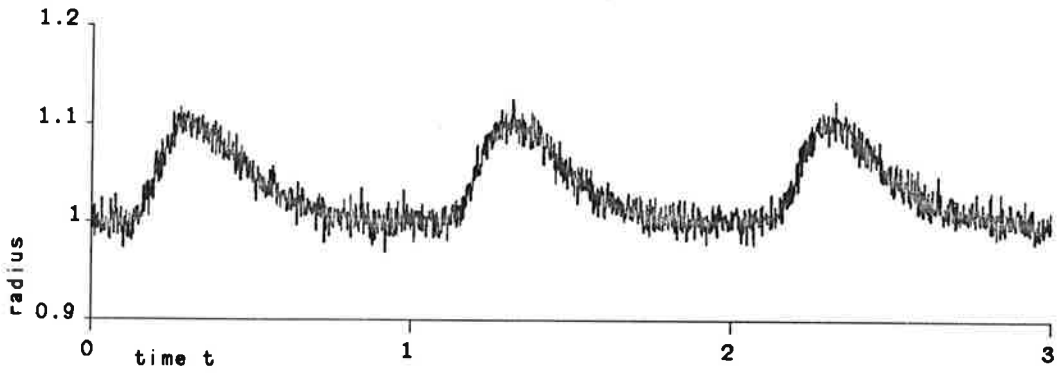


Figure 2.3 : A simulation of a noisy radius measurement.

effect is put through. Therefore there is a lower limit for τ , if the system is noisy.

Another disturbance is a bias in the measured radius , or a calibration error. The convergence of parameter α' when there is a step disturbance in the measured radius , is shown in figure 2.6. Remember that the radius varies from 1 up to approximately 1.1. Figure 2.6 clearly shows that a step error in the measurement is fatal. The decrease in an estimated parameter is dependent on the step amplitude , but even a small change strongly affects the results of identification. In order to avoid this complication an off-line version of identification is suggested for implementation use. Data , i.e. radius measurements

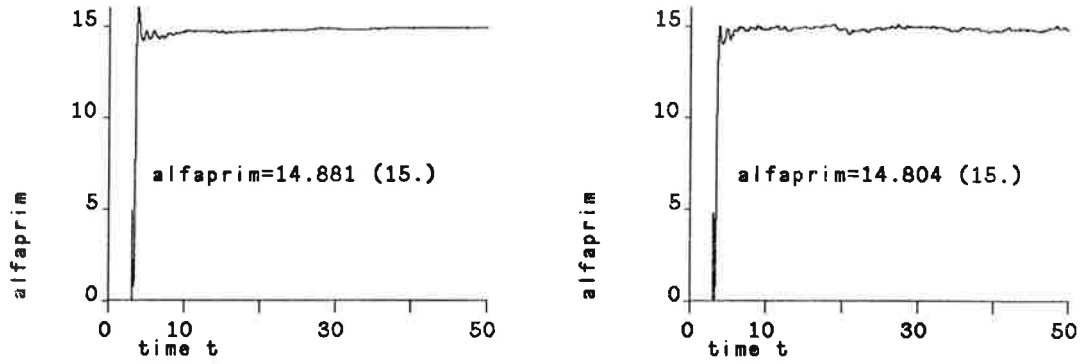


Figure 2.4 : Simulated convergence with noise. a) $\lambda = 1$ b) $\lambda = 0.99$.($\tau = 1$)

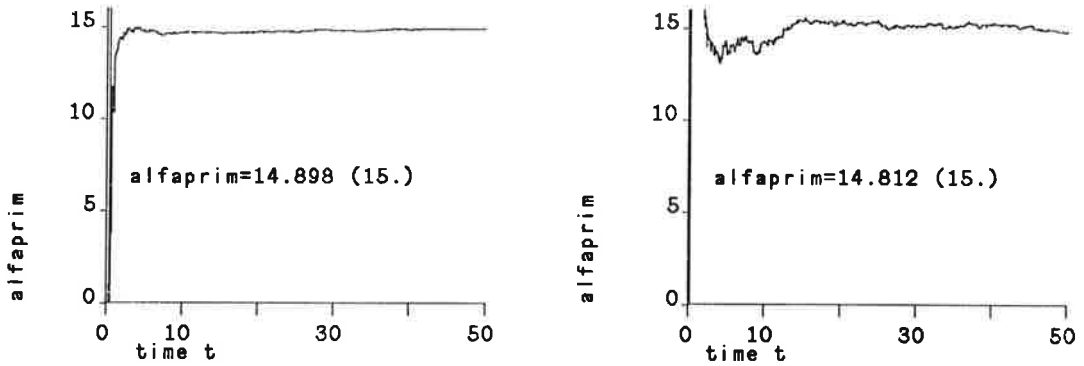


Figure 2.5 : Simulated convergence with noise. a) $\tau = 0.1$ b) $\tau = 0.01$.(λ=1)

, are first collected and displayed for the user , whom may use them for identification , or collect a new set of data.

Estimation of parameter α' in the One Point Model can be done successfully , however the assumption of the input flow is a drawback.

Two Point Model

When the operator translation of (2.2) is applied to (1.26) the result is

$$r_B^2 = (1 - \beta_1\tau)[zr_B^2] + \beta_2\tau[zr_A^2] + \beta_3\tau[z1] \qquad (2.8)$$

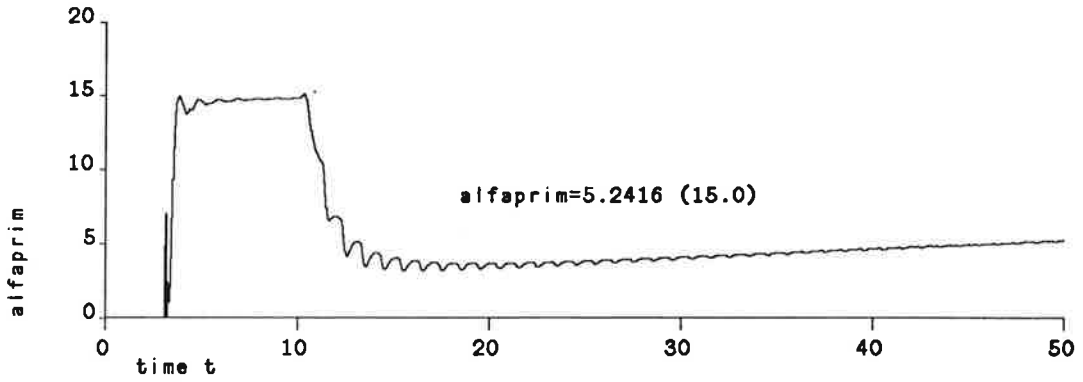


Figure 2.6 : Simulated convergence with step disturbance of amplitude 0.02 at time 10 .

The vectors used in the Recursive Least Square algorithm are then

$$\varphi^T = [\ zr_B^2 \quad zr_A^2 \quad z1\ ]$$

and

$$\theta^T = [\ (1 - \beta_1\tau) \quad \beta_2\tau \quad \beta_3\tau\ ]$$

A linear backward translation gives

$$\beta^T = [\ (1 - \theta_1)/\tau \quad \theta_2/\tau \quad \theta_3/\tau\ ]$$

To reduce these parameters they can be divided with 2 and π and also with the length ℓ , which is given a small value , e.g. 0.1 mm. The parameters that can be estimated are then

$$\beta_1' = \frac{E_B h_{OB}}{R_B r_{OB}^3} \qquad \beta_2' = \frac{E_A h_{OA}}{R_A r_{OA}^3}$$

$$\beta_3' = \left[\frac{E_B h_{OB}}{R_B r_{OB}} - \frac{E_A h_{OA}}{R_A r_{OA}} \right]$$

The identification algorithm for a Two Point Model is also implemented in Simnon. The Simnon modules are listed in Appendix A. Properties of this algorithm and

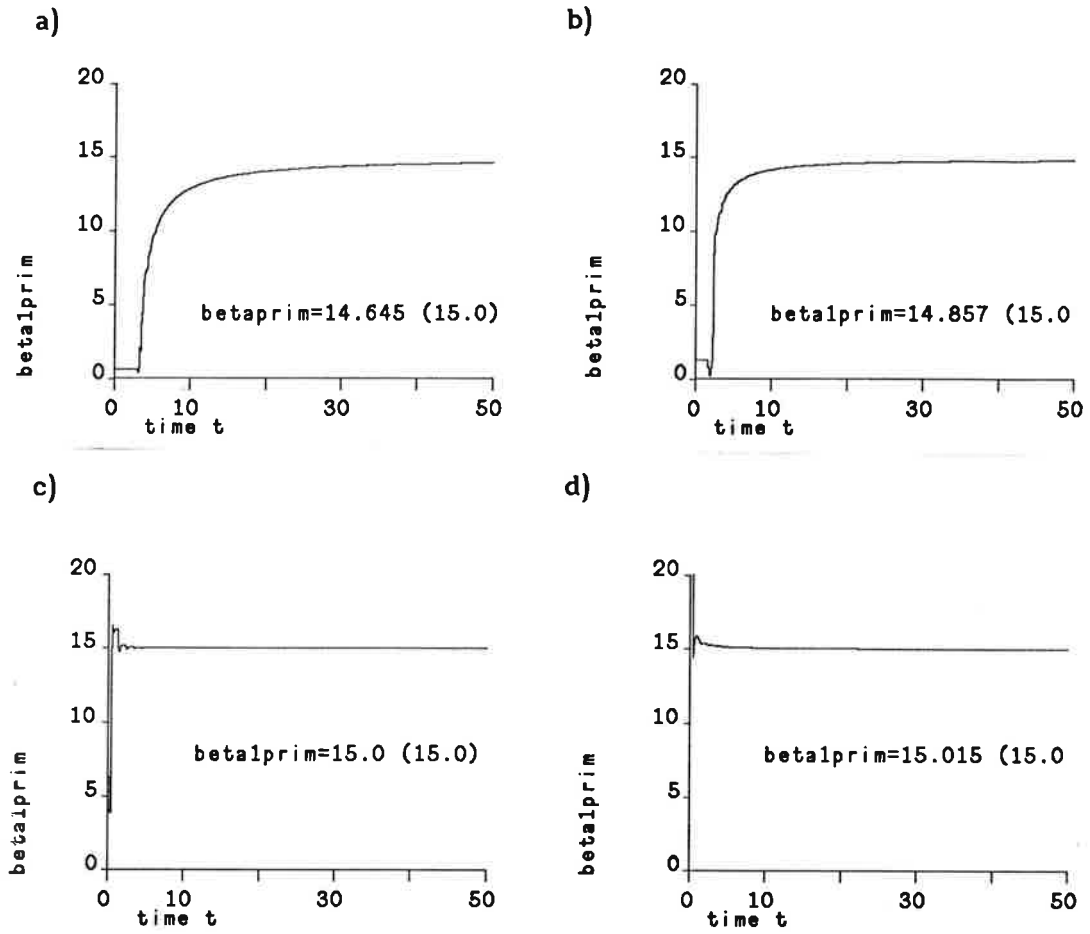


Figure 2.7 : Simulated convergence of β_1' . a) τ = 1 b) τ = 0.5 c) τ = 0.1 d) τ = 0.01

the Two Point Model were tested through a number of simulations. The setting of parameters in the Recursive Least Square algorithm is the same as for the One Point Model. Point A and point B parameters are also identical with the one point case. Figure 2.7 shows the result of simulations with various filter constants τ. The estimated parameter is β_1'. The correct value of β_1' is 15.0. It is clear from figure 2.7 that the estimated parameter settles to a value very close to the correct one for at least two orders of magnitude of τ. The filter constant τ may therefore be chosen in a rather big interval , although it should not be greater than the period of radius movement. In this case the period of the simulated model is approximately 1 time unit or 1 second. The fetal heart frequency is about twice as high as a normal persons or about 120-140 strokes per minute.

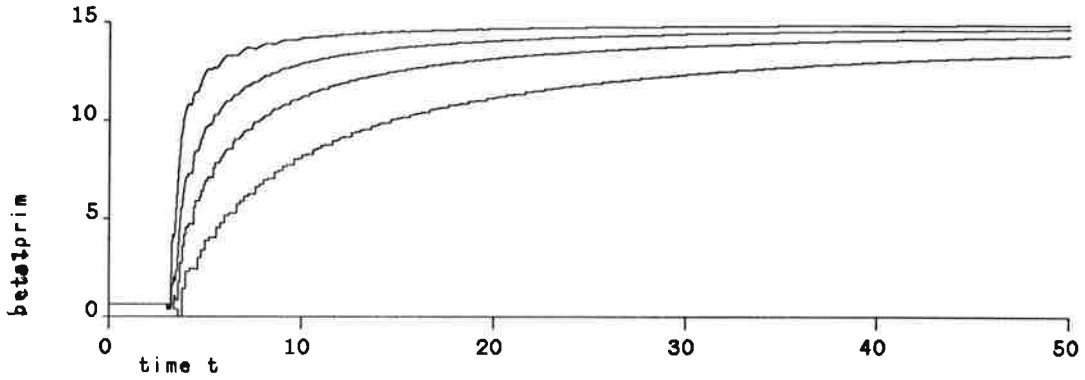


Figure 2.8 : Simulated convergence of β_1' using different sampling increments. h= 0.05 , 0.1 , 0.25 , 0.4.

Therefore the filter constant τ should be given a value of about 0.3 to 0.05 in an implementation.

Parameter convergence is also dependent on the sampling increment h of the Recursive Least Square algorithm as has been shown with the One Point Model. Figure 2.8 shows this dependency for parameter β_1'. The value of h is chosen in order to get the wanted convergence. Since the time available for identification is short , due to movements of the fetus and the ultrasonic sensor , the sampling increment must be rather small. A value of at least 20 times per period is a suggestion. This means that h should be at least 20 ms for implementation use. In fact 20 ms is the shortest available time for the implementation presented here. Another approach is to let the P-matrix in the Recursive Least Square algorithm be initialized with greater diagonal elements. Figure 2.9 shows this for a case with τ = 1 and h = 0.05.

A noise disturbance causes a bias in the estimated parameters as it does with the One Point Model. How much bias there is in the parameters depends on the noise power , the filter constant τ and also of the forgetting factor λ. In fact this sets a lower limit for the filter constant dependent on the wanted precision. Figure 2.10 shows the effect of different choices of λ for a fixed τ = 1 and figure 2.11 the effect of different τ for a fixed λ = 1. The system is affected by noise in the same way as in figure 2.3. In figure 2.10 , it is clear that the convergence is bad.

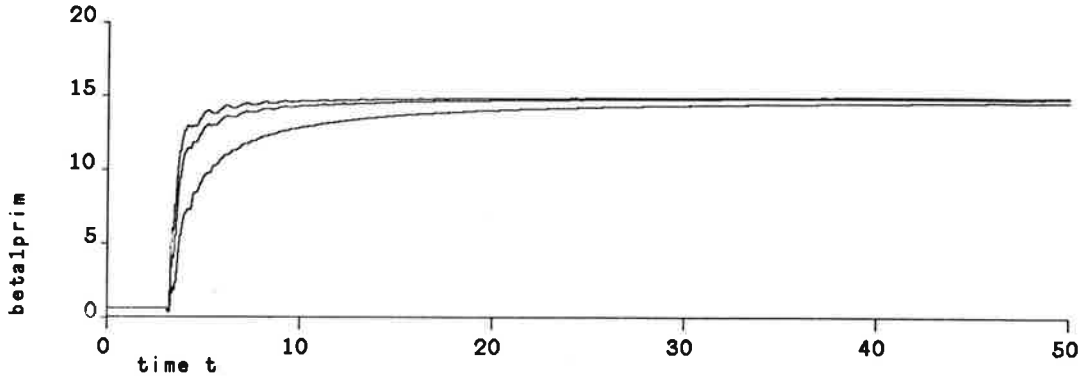


Figure 2.9 : Simulated convergence of β_1' when the P-matrix is initialized with 10000 , 30000 , 50000.

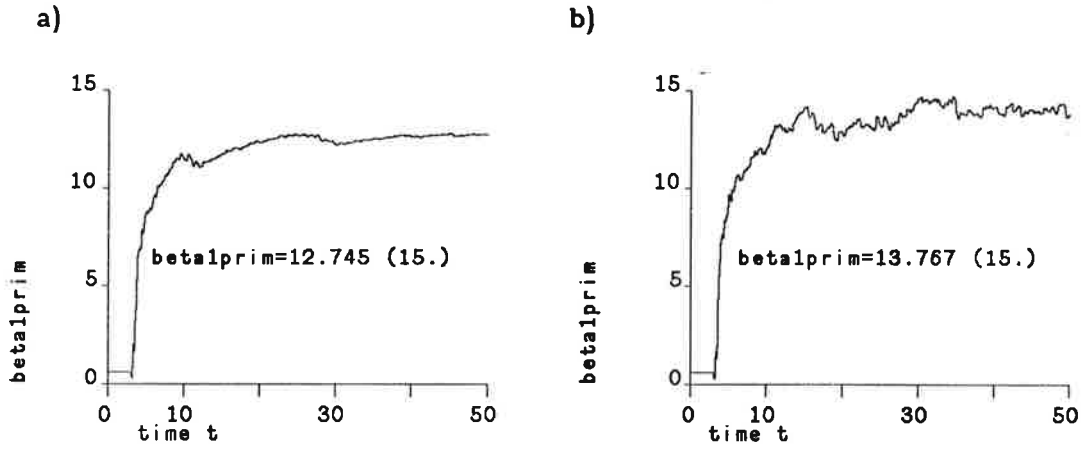


Figure 2.10 : Simulated convergence of β_1' . a) $\lambda = 1$ b) $\lambda = 0.99$.

Therefore efforts must be made to keep the noise level low. From figure 2.11 is is easy to see that a lower limit for τ is to be set in order to avoid an unsmooth parameter convergence. The final value of β_1' is however closer to the correct one (15.0) in figure 2.11 a). This might be an effect of the starting value for β_1' , which according to the linear backward translation is 10 when τ is 0.1 , but only 1 when τ is 1. Another possible cause is that the noise is not "white". In figure 2.11 b) noise influence makes the convergence very shaky.

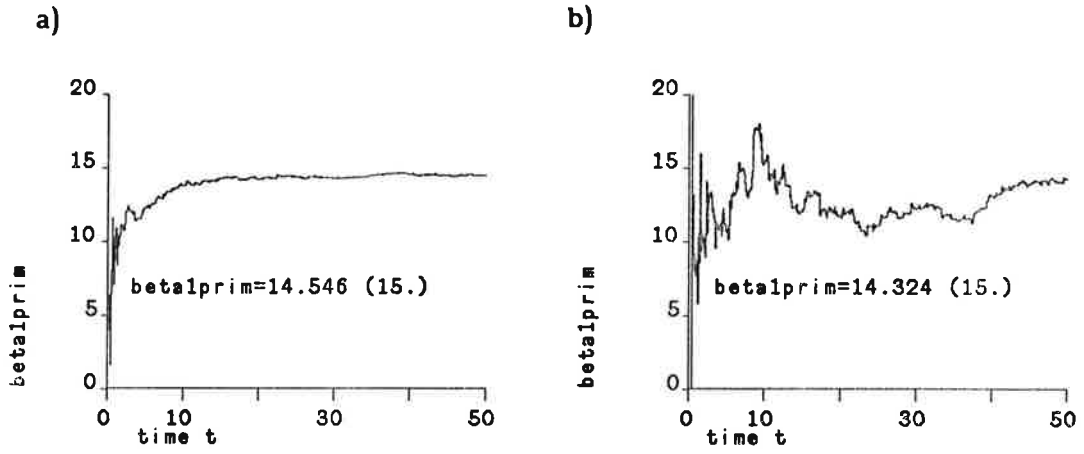


Figure 2.11 : Simulated convergence of β_1' . a) τ = 0.1 b) τ = 0.01 .

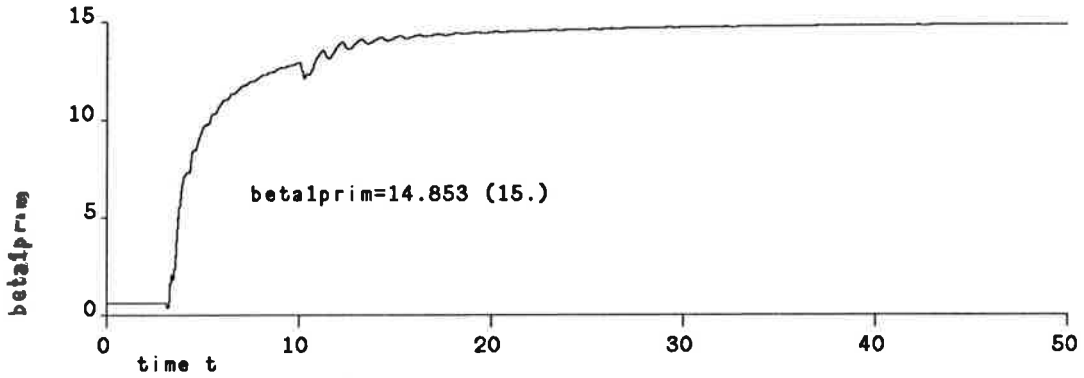


Figure 2.12 : Simulated convergence of β_1' with a step disturbance of 0.1 at time 10. Both radii are affected by the same disturbance.

An attractive feature of the Two Point Model is that when there are step disturbances in both the measured radii and they occur at the same time , the effect on parameter convergence is reduced. Figure 2.12 illustrates this for β_1'. However , if the time delay between those disturbances is only as small as 0.2 time units , or seconds , the effect is clear , see figure 2.13 a). Also when the step disturbance amplitude is different in the measured radii , the influence is marked as shown in figure 2.13 b).

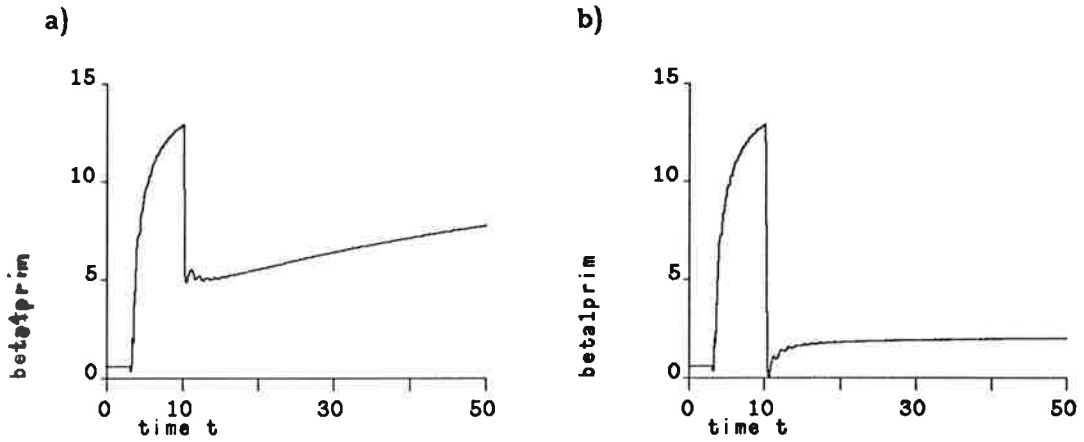


Figure 2.13 : a) Simulated convergence of β_1' with a time delay of 0.2 between step disturbances of amplitude 0.1. b) Simulated convergence with step disturbance amplitudes 0.1 and 0.05 and no time delay.

In the Two Point Model there is no assumption on the input flow , since it is not needed for identification. Instead the estimated parameters are a little more complex. The identification method using signal processing filters manages to estimate parameters well if the disturbances are small. The implementation , described in the next chapter , uses this Two Point Model. No efforts are made there to reduce influence from step disturbances by error detection and correction. Instead an off line version , as suggested in the previous section is used.

3. Implementation

Motivation

Datacollection and identification algorithm are implemented on a personal computer. This is quite natural since the computer is to be used in a hospital , which requires handy equipment. The choice of IBM PC/AT depended on the fact that it was available the Department of Automatic Control , where this work has performed. IBM PC/AT computers are used in the teaching laboratory and these machines are equiped with some useful features. The program is written in Modula-2 , see Wirth (1983) , which is a high level language , closely related to Pascal.

Since one of the program activities is to collect data from analog inputs , some procedures for handling this in real time are needed. A real time kernel written at the Department , see Elmqvist et al. (1981) or Andersson (1986) for an overview , offers these features. Other useful procedures handling e.g. conversion of numbers and strings , and analog inputs are also given in the paper by Andersson. An assemble routine using clock interrupts would have increased speed , but a real time kernel is a fast way of getting a program of this sort running. In order to get a graphic output , e.g. plots of data and parameters , a graphic program package , GEM , from Digital Research , see Andersson (1986) , is used.

Program Structure

A real time program consists of a number of processes. In this case there are three processes. The one running with the highest priority is a clock , which is used to control another process , used for data collection from analog inputs. There is also one process for handling of communication with the user.

The clock process waits one sampling increment and then activates the data collection process. In the process for operator communication , a number of activities may be selected. A choice of data collection activates a part of the data collection process , in which the analog inputs are read and their values are

stored in a matrix. If identification is chosen , then data are fetched from the storage matrix , filtered and used by a Recursive Least Square algorithm. With plot old data the contains of the storage matrix may be displayed for the user. The remaining activity is setup , in which for instance some parameters of the Recursive Least Square algorithm may be altered. A more detailed description of these activities is given in Appendix C. With a halt command the program is terminated.

Filter Implementation

Signal processing filters used to filter measurements in this continuous time system identification method are analog. It would of course be possible to use real analog filters , but an easier way is to implement them directly in the computer as digital filters. The cut off frequency of digital filters is easily changed by altering the filter constant τ.

The transfer function of the used low pass filter is $1/(1+p\tau)$, $p=d/dt$, which is denoted z in equation (2.1) , and the corresponding differential equation is therefore

$$\dot{y} = -\frac{1}{\tau}y + \frac{1}{\tau}u \tag{3.1}$$

or

$$\dot{y} = -ay + bu \tag{3.2}$$

with input u and output y. Multiplication with e^{at} and integration over the sampling increment h gives

$$e^{a(t+h)}y(t+h) - e^{at}y(t) = \int_t^{t+h} be^{as}u(s)\,ds$$

and when y(t+h) is solved

$$y(t+h) = e^{-ah}y(t) + e^{-a(t+h)}\int_t^{t+h} be^{as}u(s)\,ds \tag{3.3}$$

The input signal u was first supposed to be constant during the sampling interval , i.e. zero order hold sampling. However , when a filter of this sort was used in

the IBM PC/AT computer the results of identification were not so good. Therefore a different approximation of the input u over the sampling interval is done

$$u(s) = \frac{u(t+h) + u(t)}{2} \qquad t < s < t+h \tag{3.4}$$

Equation (3.3) is evaluated and a filter state x is introduced. Hereby two easily implemented equations are formed and when a and b are replaced the resulting equations for the filter state x and filter output y are

$$\begin{cases} x(t+h) = e^{-h/\tau}x(t) + \left[1 - e^{-h/\tau}\right] u(t) \\ \\ y(t) = \frac{1}{2}\left[1 - e^{-h/\tau}\right]\left[u(t) + x(t)\right] \end{cases} \tag{3.5}$$

A filter with this approximation of the intersampling behaviour for the input signal is used in the implementation program. It provides clearly an improvement of identification results. Even if there is more precision in this filter , than in the zero order hold sampled , a more accurate algorithm for integrating equation (3.1) would be useful. This is however left for the future.

Testing

No real radius measurements were available when the program was written and therefore a different method was used to test the program. Data , i.e. simulated radius measurements were logged from the Two Point Model , implemented in Simnon , and stored in a file. This file was then sent to and stored in another IBM PC/AT computer. A small real time program reads the file and places data in a matrix. In an everlasting loop data is then dumped from the matrix to analog outputs , which then may be read from the computer with the running implementation program. This makes it possible to check the whole program , from data collection via analog inputs to the Recursive Least Square algorithm , which as is earlier pointed out , not runs on line. It is of course a great advantage to make such a test with known model parameters and the results are quite good.

4. Discussion

When this work was presented for professor Gennser and his group at the hospital in Malmö , criticisms were formed on the time dependency of the cardiac output flow. From clinical experiments they had found the cardiac output to be more like a square wave , perhaps with a rounded top. For this purpose and to clear other doubts about the model , coming from the fact that cardiac output flow and radius curves are rather alike , some simulations were done using other flows. The results are shown in figures 4.1 and 4.2. In these figures the low pass characteristics of the Two Point Model are more obvious. The purpose of this work has been to develop a simple model. Only time dependent radius variations are measured. It is therefore desirable to get a more precise model , perhaps through a measurement of the pressure , if possible , or the speed of a

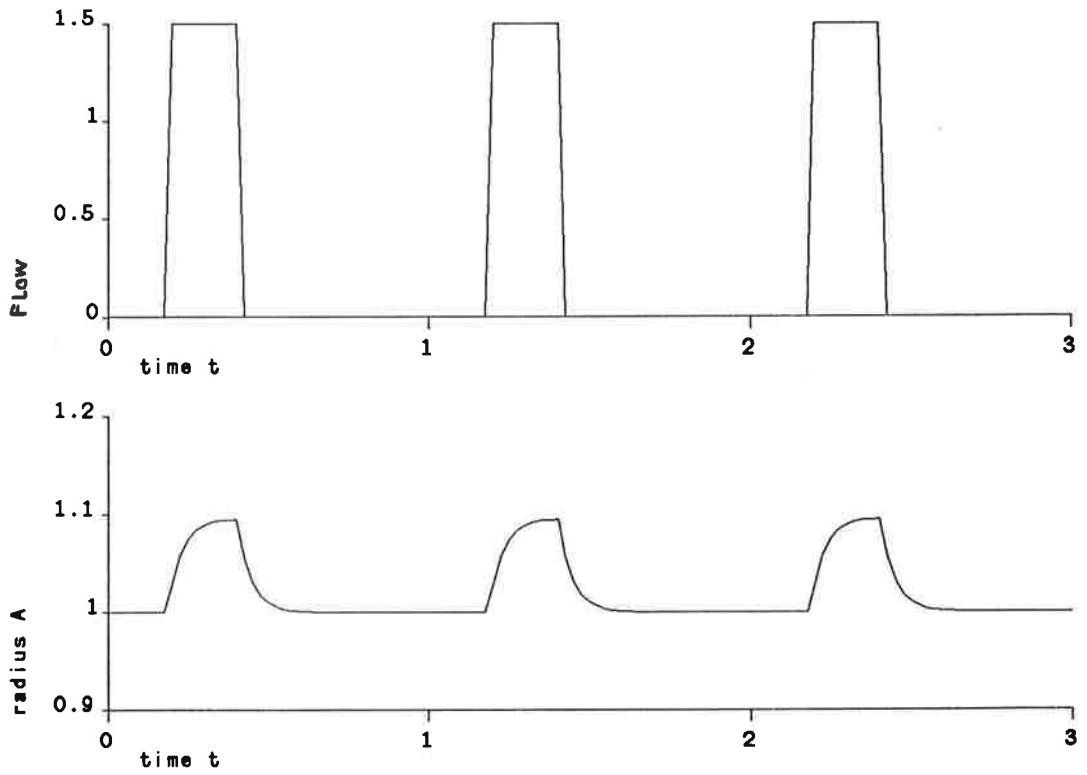


Figure 4.1 : Simulated square wave flow and radius A variations.

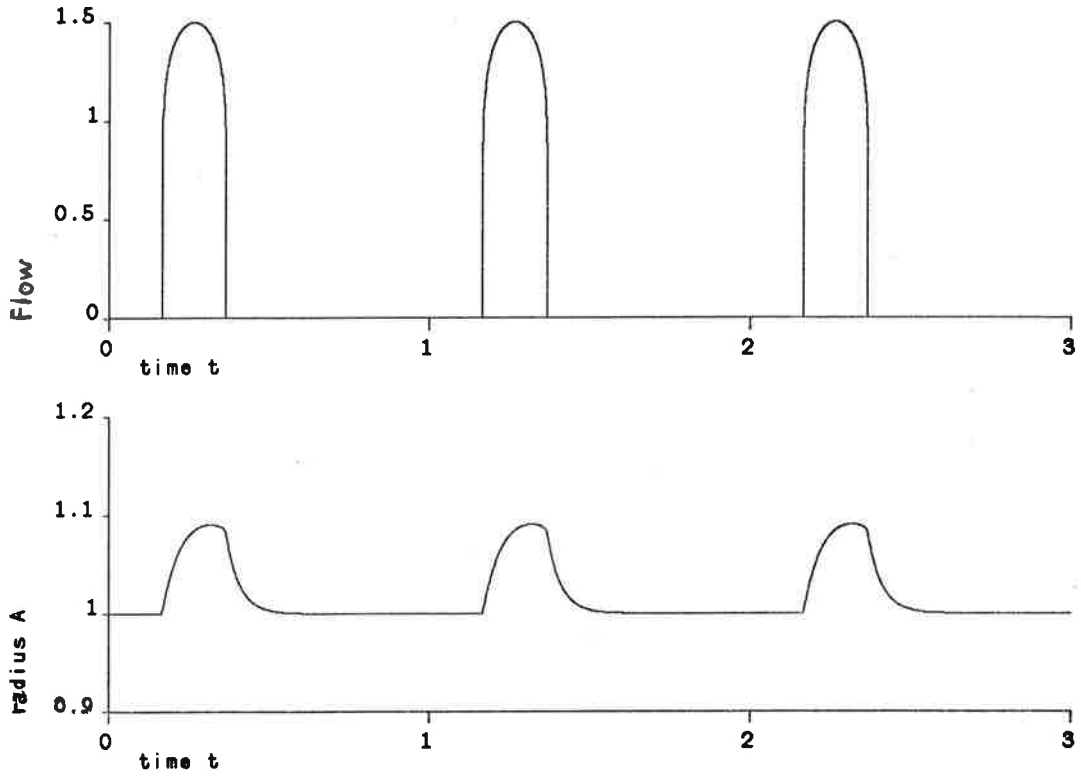


Figure 4.2 : Simulated rounded top flow and radius A variations.

pulse wave.

Identification using signal processing filters for filtering measurements shows a good result for this implementation , at least if the signals are rather free from disturbances. When the purpose is to detect fetal growth retardation , errors in estimated parameters due to disturbances must be prevented. This places a rather hard demand on the identification algorithm. Parameters must converge in a limited period of time , perhaps as little as 10 seconds. One way of reducing convergence time is to use large starting values for the P-matrix in the Recursive Least Square algorithm. Another approach is to shorten the sampling interval of the algorithm. These demands on the algorithm are of course the same for an ARMA model implementation. The properties of this continuous time system identification method are not fully explored , but so far the results are attractive.

The program written in Modula-2 may be used either directly , or as a pattern for translation to another language. A real time kernel is used to control the data collection. Since the data collection task is the only one really using the kernel , apart from the clock , a small assembler routine using clock interrupts may be used instead. Such a routine would perhaps speed up the activities on a sampling point so much , that a shorter sampling interval could be used. The idea that the user should determine a sequence of undisturbed measurements from all collected data , see Appendix C , introduces more uncertainty to the results of identification. However , this is the best method available , since a real on line method can not be used. One future expansion is to introduce possibilities for the user to save results of data collections and identifications. Another future task is to program a more precise numerical algorithm for the implementation of signal processing filters.

Appendix A

Simulation Modules

This appendix contains all Simnon modules used for the simulations , and macros used to create simulation illustrations in this thesis. For information about the simulation package Simnon , see Åström (1985). In figure A.1 a try is made to show the connection between the different modules. One of the filter inputs in figure A.1 has a 1 as input signal. This is due to the fact that there is a constant term in both model equations. This constant term is then estimated by connecting the filter input as stated above.

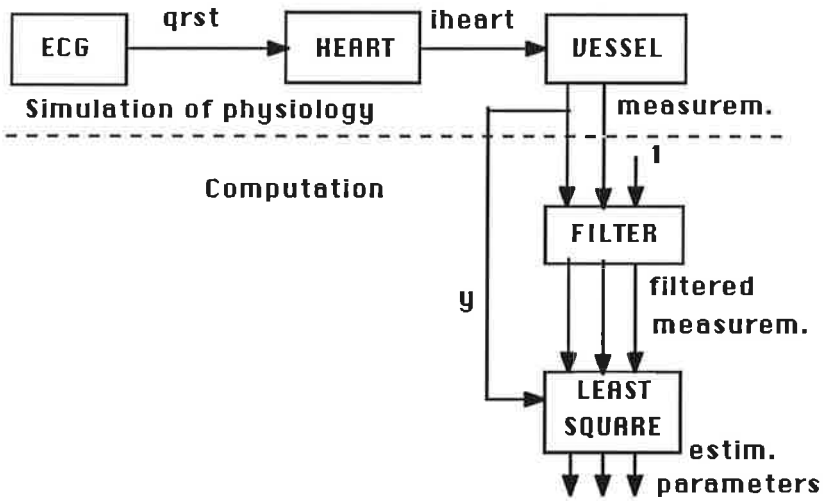


Figure A.1 : Simulation module connections.

For the nonlinear One Point Model the following systems are used

```
discrete system ecg

time t
tsamp ts
state q1 q2 q3 q4 q5 q6 q7 q8 q9 q10
new n1 n2 n3 n4 n5 n6 n7 n8 n9 n10

n1=if (q10>0) then q10 else 0
n2=q1
n3=q2
n4=q3
n5=q4
n6=q5
n7=q6
n8=q7
n9=q8
n10=q9
w: 0.5
q1: 0
q2: 0
q3: 0
q4: 0
q5: 0
q6: 0
q7: 0
q8: 0
q9: 0
q10: 1
ts=t+h
h: 0.1
end


continuous system heart
"produces pulse response of b*t*exp(-a*t)

time t
input qrst
state s1 s2
der d1 d2
output iheart

d1=-2*a*s1-a*a*s2+b*qrst
d2=s1
iheart=s2
a: 10
b: 500
```

```
end


continuous system vessel3

time t
state r2
der dr2
input i
output out0 out1 out2 out3

dr2=-a/sqrt(r2)+b/r2+c*i
a=E*h0/pi/l/pr
b=a*r0
c=1/pi/l
r=sqrt(r2)
out0=1/r
out1=1/r2
out2=i
out3=r2
pi: 3.1415927
r0: 1
pR: 1
E: 50
h0: 0.3
l: 0.1
r2: 1
end


continuous system filter

time t
state x1 x2 x3
der dx1 dx2 dx3
input y1 y2 y3
output f1 f2 f3

dx1=(y1-x1)/tau
dx2=(y2-x2)/tau
dx3=(y3-x3)/tau
f1=x1
f2=x2
f3=x3
tau: 1
end


discrete system lsq

time t
```

```
tsamp ts
state th1 th2 th3 p11 p12 p13 p22 p23 p33
new nth1 nth2 nth3 np11 np12 np13 np22 np23 np33
input fi1 fi2 fi3 y

tstart=3*tau
eps=if t>tstart then (y-fi1*th1-fi2*th2-fi3*th3) else 0

pfi1=p11*fi1+p12*fi2+p13*fi3
pfi2=p12*fi1+p22*fi2+p23*fi3
pfi3=p13*fi1+p23*fi2+p33*fi3

div=fi1*pfi1+fi2*pfi2+fi3*pfi3+lambda

np11=if t>tstart then (p11-pfi1*pfi1/div)/lambda else p11
np12=if t>tstart then (p12-pfi1*pfi2/div)/lambda else p12
np13=if t>tstart then (p13-pfi1*pfi3/div)/lambda else p13
np22=if t>tstart then (p22-pfi2*pfi2/div)/lambda else p22
np23=if t>tstart then (p23-pfi2*pfi3/div)/lambda else p23
np33=if t>tstart then (p33-pfi3*pfi3/div)/lambda else p33

k1=np11*fi1+np12*fi2+np13*fi3
k2=np12*fi1+np22*fi2+np23*fi3
k3=np13*fi1+np23*fi2+np33*fi3

nth1=th1+k1*eps
nth2=th2+k2*eps
nth3=th3+k3*eps

kf1=(-th1+1)/tau
kf2=th2/tau
kf3=th3/tau
kf4=sqrt(abs(kf2/kf1))

alfprim=if kf3>0 then kf1*2/kf3*kf4*kf4*kf4 else 0

ts=t+h
p11: 10000
p22: 10000
p33: 10000
lambda: 1
h: 0.05
tau: 1
end



connecting system vescon3

time t

i[vessel3]=iheart[heart]
qrst[heart]=q1[ecg]
y1[filter]=out0[vessel3]
```

```
y2[filter]=out1[vessel3]
y3[filter]=out2[vessel3]
fi1[lsq]=f1[filter]
fi2[lsq]=f2[filter]
fi3[lsq]=f3[filter]
y[lsq]=out3[vessel3]
end


macro sysmac3
syst vessel3 heart lsq filter ecg vescon3
end
```

In the linear One Point Model , vessel3 , vescon3 and sysmac3 are replaced with

```
continuous system vessel31

time t
state r2
der dr2
input i noi step
output out1 out2

dr2=-a*r2+b+c*i
a=E*h0/2/pi/l/pr/r0/r0/r0
b=a*r0*r0
c=1/pi/l
r=if r2>0 then sqrt(r2) else 0
rm=r+noi*on+step
nr2=rm*rm
out1=nr2
out2=i
pi: 3.1415927
r0: 1
pR: 1
E: 50
h0: 0.3
l: 0.1
r2: 1
on: 0
end


connecting system vescon31

time t

i[vessel31]=iheart[heart]
```

```
qrst[heart]=q1[ecg]
y1[filter]=out1[vessel3l]
y2[filter]=1
y3[filter]=out2[vessel3l]
fi1[lsq]=f1[filter]
fi2[lsq]=f2[filter]
fi3[lsq]=f3[filter]
y[lsq]=out1[vessel3l]
noi[vessel3l]=e1[noise1]
step[vessel3l]=if t>ton then amp else 0
amp: 0
ton: 10
end
```

```
macro sysmac3l
let n.noise1=1
let nodd.noise1=25831
syst vessel3l heart ecg lsq filter noise1 vescon3l
end
```

In the Two Point Model , vessel3l , lsq , vescon3l and sysmac3l are replaced with

```
continuous system vessel6

time t
state ra2 rb2
der dra2 drb2
input i noi1 noi2 step1 step2
output out1 out2

dra2=-a1*ra2+a2+c1*i
drb2=a1*ra2-b1*rb2+(b2-a2)

a1=Ea*h0a/2/pi/l/pra/r0a/r0a/r0a
a2=a1*r0a*r0a
c1=1/pi/l
b1=Eb*h0b/2/pi/l/prb/r0b/r0b/r0b
b2=b1*r0b*r0b

ra=if ra2>0 then sqrt(ra2) else 0
rb=if rb2>0 then sqrt(rb2) else 0
ram=ra+noi1*on+step1
rbm=rb+noi2*on+step2
nra2=ram*ram
nrb2=rbm*rbm
out1=nrb2
out2=nra2
pi: 3.1415927
```

```
r0a: 1
r0b: 1
pra: 1
prb: 1
Ea: 50
Eb: 50
h0a: 0.3
h0b: 0.3
l: 0.1
ra2: 1
rb2: 1
on: 0
end



discrete system lsq6

time t
tsamp ts
state th1 th2 th3 p11 p12 p13 p22 p23 p33
new nth1 nth2 nth3 np11 np12 np13 np22 np23 np33
input fi1 fi2 fi3 y

tstart=3*tau
eps=if t>tstart then (y-fi1*th1-fi2*th2-fi3*th3) else 0

pfi1=p11*fi1+p12*fi2+p13*fi3
pfi2=p12*fi1+p22*fi2+p23*fi3
pfi3=p13*fi1+p23*fi2+p33*fi3

div=fi1*pfi1+fi2*pfi2+fi3*pfi3+lambda

np11=if t>tstart then (p11-pfi1*pfi1/div)/lambda else p11
np12=if t>tstart then (p12-pfi1*pfi2/div)/lambda else p12
np13=if t>tstart then (p13-pfi1*pfi3/div)/lambda else p13
np22=if t>tstart then (p22-pfi2*pfi2/div)/lambda else p22
np23=if t>tstart then (p23-pfi2*pfi3/div)/lambda else p23
np33=if t>tstart then (p33-pfi3*pfi3/div)/lambda else p33

k1=np11*fi1+np12*fi2+np13*fi3
k2=np12*fi1+np22*fi2+np23*fi3
k3=np13*fi1+np23*fi2+np33*fi3

nth1=th1+k1*eps
nth2=th2+k2*eps
nth3=th3+k3*eps

b1prim=(-th1+1)/tau*2*pi*l
b2prim=th2/tau*2*pi*l
b3prim=th3/tau*2*pi*l

ts=t+h
p11: 10000
```

```
p22: 10000
p33: 10000
lambda: 1
h: 0.05
tau: 1
pi: 3.1415927
l: 0.1
end


connecting system vescon6

time t

i[vessel6]=iheart[heart]
qrst[heart]=q1[ecg]
y1[filter]=out1[vessel6]
y2[filter]=out2[vessel6]
y3[filter]=1
fi1[lsq6]=f1[filter]
fi2[lsq6]=f2[filter]
fi3[lsq6]=f3[filter]
y[lsq6]=out1[vessel]
noi1[vessel6]=e1[noise1]
noi2[vessel6]=e2[noise1]
step1[vessel6]=if t>ton1 then amp1 else 0
step2[vessel6]=if t>ton2 then amp2 else 0
amp1: 0
amp2: 0
ton1: 10
ton2: 10
end


macro sysmac6
let n.noise1=2
let nodd.noise1=25831
syst vessel6 heart ecg lsq6 filter noise1 vescon6
end
```

To generate the figures in chapter 1 the following macros are used

```
macro fig14
"generates figure 1.4
sysmac31
algor rk
store qrst iheart
simu 0 5
```

```
split 2 2
ashow qrst
area 1 2
ashow iheart
mark a 3 7
mark "time t
mark a 13.5 7
mark "time t
mark a 1 8
mark v  "ECG signal
mark a 11 8
mark v "Flow
end


macro fig15
"generates figure 1.5
sysmac3l
algor rk
store r
simu 0 3
sysmac3
split 2 1
axes h 0 3 v 0.9 1.2
plot r
simu 0 3
show r
mark a 3 7
mark "time t
mark a 1 8
mark v "radius
end


macro fig17
"generates figure 1.7
sysmac6
algor rk
store ra rb
simu 0 3
split 2 1
axes h 0 3 v 0.9 1.2
show ra
show rb
mark a 3 7
mark "time t
mark a 1 8
mark v "radius
end
```

To generate the figures in chapter 2 the following macros are used

```
macro fig21ab
"generates figure 2.1 a) and b)
sysmac31
algor rk
split 2 1
axes h 0 50 v 0 20
plot alfprim
par tau[filter]:2
par tau[ls]:2
simu 0 50
disp alfprim
axes
par tau[filter]:0.2
par tau[ls]:0.2
simu
disp alfprim
mark a 3 7
mark "time t
mark a 3 0
mark "time t
mark a 1 8
mark v "alfaprim
mark a 1 1
mark v "alfaprim
end
```

```
macro fig21c
"generates figure 2.1 c)
sysmac31
algor rk
split 2 1
axes h 0 50 v 0 20
plot alfprim
par tau[filter]:0.02
par tau[ls]:0.02
simu 0 50
disp alfprim
mark a 3 7
mark "time t
mark a 1 8
mark v "alfaprim
end
```

```
macro fig22
"generates figure 2.2
sysmac31
```

```
algor rk
split 2 1
axes h 5 50 v 12 15
plot alfprim
simu 0 50
par h[ls]:0.1
simu
par h[ls]:0.2
simu
par h[ls]:0.3
simu
mark a 1 8
mark v "alfaprim
mark a 3 7
mark "time t
end
```

```
macro fig23
"generates figure 2.3
sysmac3l
algor rk
par on:1
par stdev1[noise1]:0.01
par dt[noise1]:0.002
split 2 1
axes h 0 3 v 0.9 1.2
plot rm
simu 0 3
mark a 1 8
mark v "radius
mark a 3 7
mark "time t
end
```

```
macro fig24
"generates figure 2.4
sysmac3l
algor rk
par on:1
par stdev1[noise1]:0.01
par dt[noise1]:0.002
split 2 2
axes h 0 50 v 0 16
plot alfprim
simu 0 50
disp alfprim
par lambda:0.99
area 1 2
axes
simu
```

```
disp alfprim
mark a 1 8
mark v "alfaprim
mark a 3 7
mark "time t
mark a 11 8
mark v "alfaprim
mark a 13 7
mark "time t
end


macro fig25
"generates figure 2.5
sysmac31
algor rk
par on: 1
par stdev1[noise1]: 0.01
par dt[noise1]: 0.002
par tau[ls]: 0.1
par tau[filter]: 0.1
split 2 2
axes h 0 50 v 0 16
plot alfprim
simu 0 50
disp alfprim
par tau[ls]: 0.01
par tau[filter]: 0.01
area 1 2
axes
simu
disp alfprim
mark a 1 8
mark v "alfaprim
mark a 3 7
mark "time t
mark a 11 8
mark v "alfaprim
mark a 13 7
mark "time t
end


macro fig26
"generates fig 2.6
sysmac31
algor rk
par amp: 0.02
par ton: 10
split 2 1
axes h 0 50 v 0 20
plot alfprim
```

```
simu 0 50
disp alfprim
mark a 3 7
mark "time t
mark a 1 8
mark v "alfprim
end


macro fig27
"generates fig 2.7
sysmac6
algor rk
par tau[ls]:1
par tau[filter]:1
split 2 2
axes h 0 50 v 0 20
plot b1prim
simu 0 50
mark a 3 7
mark "time t
mark a 1 8
mark v "beta1prim
disp b1prim
area 1 2
axes
par tau[ls]:0.5
par tau[filter]:0.5
simu
mark a 11 8
mark v "beta1prim
mark a 13 7
mark "time t
disp b1prim
area 2 1
axes
par tau[ls]:0.1
par tau[filter]:0.1
simu
mark a 1 1
mark v "beta1prim
mark a 3 0
mark "time t
disp b1prim
area 2 2
axes
par tau[ls]:0.01
par tau[filter]:0.01
simu
mark a 11 1
mark v "beta1prim
mark a 13 0
mark "time t
```

```
disp b1prim
end


macro fig28
"generates figure 2.8
sysmac6
algor rk
par h[ls]:0.02
split 2 1
axes h 0 50 v 0 15
plot b1prim
simu 0 50
par h[ls]:0.05
simu
par h[ls]:0.1
simu
par h[ls]:0.2
simu
mark a 1 8
mark v "beta1prim
mark a 3 7
mark "time t
end


macro fig29
"generates figure 2.9
sysmac6
algor rk
split 2 1
axes h 0 50 v 0 20
plot b1prim
simu 0 50
init p11:30000
init p22:30000
init p33:30000
simu
init p11:50000
init p22:50000
init p33:50000
simu
mark a 1 8
mark v "beta1prim
mark a 3 7
mark "time t
end
```

```
macro fig210
"generates figure 2.10
sysmac6
algor rk
par stdev1:0.01
par stdev2:0.01
par dt:0.002
par on:1
split 2 2
axes h 0 50 v 0 15
plot b1prim
simu 0 50
disp b1prim
area 1 2
axes
par lambda:0.99
simu
disp b1prim
mark a 1 8
mark v "beta1prim
mark a 3 7
mark "time t
mark a 11 8
mark v "beta1prim
mark a 13 7
mark "time t
end


macro fig211
"generates figure 2.11
sysmac6
algor rk
par stdev1:0.01
par stdev2:0.01
par dt:0.002
par on:1
split 2 2
axes h 0 50 v 0 20
plot b1prim
par tau[ls]:0.1
par tau[filter]:0.1
simu 0 50
disp b1prim
area 1 2
axes
par tau[ls]:0.01
par tau[filter]:0.01
simu
disp b1prim
mark a 1 8
mark v "beta1prim
mark a 3 7
```

```
mark "time t
mark a 11 8
mark v "beta1prim
mark a 13 7
mark "time t
end


macro fig212
"generates figure 2.12
sysmac6
algor rk
par amp1: 0.1
par amp2: 0.1
par ton1: 10
par ton2: 10
split 2 1
axes h 0 50 v 0 15
plot b1prim
simu 0 50
disp b1prim
mark a 1 8
mark v "beta1prim
mark a 3 7
mark "time t
end


macro fig213
"generates figure 2.13
sysmac6
algor rk
par amp1: 0.1
par amp2: 0.1
par ton1: 10
par ton2: 10.2
split 2 2
axes h 0 50 v 0 15
plot b1prim
simu 0 50
par amp1: 0.05
par ton2: 10
area 1 2
axes
simu
mark a 1 8
mark v "beta1prim
mark a 3 7
mark "time t
mark a 11 8
mark v "beta1prim
mark a 13 7
```

```
mark "time t
end
```

The figures in chapter 4 required a different heart to provide the requested curve shape. For figure 4.1 , heart is replaced with heart2. This module and the macro for generating figure 4.1 are listed below.

```
continuous system heart2
"produces pulse response of square wave type

time t
input qrst
state s1 s2
der d1 d2
output iheart

d1=-2*a*s1-a*a*s2+b*qrst
d2=s1
iheart=if s2>1 then 1.5 else 0
a: 10
b: 500
end
```

```
macro fig41
"generates figure 4.1
let n.noise1=2
let nodd.noise1=25831
syst lsq6 filter heart2 ecg vessel6 noise1 vescon6
algor rk
split 2 1
axes h 0 3 v 0 1.5
plot iheart
store ra
simu 0 3
axes v 0.9 1.2
show ra
mark a 3 7
mark "time t
mark a 3 0
mark "time t
mark a 1 8
mark v  "Flow
mark a 1 1
mark v "radius A
end
```

To get a heart that delivers the flow in figure 4.2 , the module extra is introduced. A new heart module is needed as well. A new connecting system is also needed. These modules and the macro used to generate figure 4.2 are listed below.

```
discrete system extra

time t
tsamp ts
input in
output out
state x
new nx

nx=if in<0.7 then t else x
y=sin(w*(t-x))
out=if in>0.7 and y>0 then 1.5*sqrt(sqrt(sqrt(y))) else 0
ts=t+h
h: 0.001
w: 15.71
end



continuous system heart
"produces pulse response with rounded top

time t
input qrst in
output out
state s1 s2
der d1 d2
output iheart

d1=-2*a*s1-a*a*s2+b*qrst
d2=s1
iheart=in
out=s2
a: 10
b: 500
end



connecting system vescon6e

time t

i[vessel]=iheart[heart]
qrst[heart]=q1[ecg]
y1[filter]=out1[vessel]
y2[filter]=out2[vessel]
```

```
y3[filter]=1
fi1[ls]=f1[filter]
fi2[ls]=f2[filter]
fi3[ls]=f3[filter]
y[ls]=out1[vessel]
noi1[vessel]=e1[noise1]
noi2[vessel]=e2[noise1]
step1[vessel]=if t>ton1 then amp1 else 0
step2[vessel]=if t>ton2 then amp2 else 0
in[heart]=out[extra]
in[extra]=out[heart]
amp1: 0
amp2: 0
ton1: 10
ton2: 10
end


macro fig42
"generates figure 4.2
let n.noise1=2
let nodd.noise1=25831
syst lsq6 filter heart3 ecg vessel6 noise1 extra vescon6e
algor rk
split 2 1
axes h 0 3 v 0 1.5
plot iheart
store ra
simu 0 3
axes v 0.9 1.2
show ra
mark a 3 7
mark "time t
mark a 3 0
mark "time t
mark a 1 8
mark v  "Flow
mark a 1 1
mark v "radius A
end
```

Appendix B

Program Listing

Before the program listings are given some global variables and constants should be explained. For a short description of imported procedures , see Andersson (1986). Constants are

l - is the length of the considered vessel. It is given a value of 0.1 mm.

xaxislength - is the length of the x-axis in data and parameter plots. It is given in terms of window coordinates. The window is set to 350.0 in y-axis direction and 640.0 in x-axis direction.

pyaxislength and dyaxislength - are the lengths of y-axes for parameter and dataplots.

xaxisstart - is the starting point of all x-axes , counted from the left side of the screen.

maxpar - maximal number of estimated parameters.

numpar - actual number of parameters used.

maxinput - maximal number of measured inputs.

numinput - actual number of inputs used.

maxsample - maximal number of samples available for a data collection.

maxtime - maximal measurement time.

tsamp - sampling increment , 0.02 seconds.

Global variables are

mon1 - contains the filter state vector , the filtered variables in fivec and the theta vector and P-matrix of the Recursive Least Square algorithm.

mon2 - contains a data matrix , where collected data are stored. It also contains arrays used for plotting.

mon3 - contains a parmatrix , where parameter values are stored during the identification. It also contains arrays used for plotting.

mon4 - contains variables used for synchronisation of sampling.

tstart - is the delay for the Recursive Least Square algorithm , used to wait out filter transients. It is set to $3*\tau$, where τ is the filter constant.

time - used by the Recursive Least Square algorithm to keep track of time for identification start.

lambda - the forgetting factor in the Recursive Least Square algorithm.

tau - is the filter constant in a continuous time signal processing low pass filter.

filtercoeff1 and filtercoeff2 - are filter constants used in the discrete time approximation of a continuous time filter.

pinit - is the value used as starting value in the P-matrix of the Recursive Least Square algorithm.

Bupper , Blower , Aupper and Alower - are used as scale factors for data collection. B means point B and A means point A. An example : The AD converters range is -10 to +10 Volts. If the signal range is between 0 and +5 Volts , 0 Volts meaning 0 mm. and +5 Volts

meaning e.g. 50 mm. ,then Aupper and Bupper should be set to 100 (mm.). In the same way Alower and Blower should be given the value (-100 mm.).

firsttime and lasttime - selects a sequence from collected data to use for a new plot or an identification. Their values are given in seconds.

firstsample and lastsample - have the same meaning as firsttime and lasttime , only their values are given in terms of samples.

numsample - the total number of samples collected in a measurement.

handle - is used by the GEM system for graphic outputs.

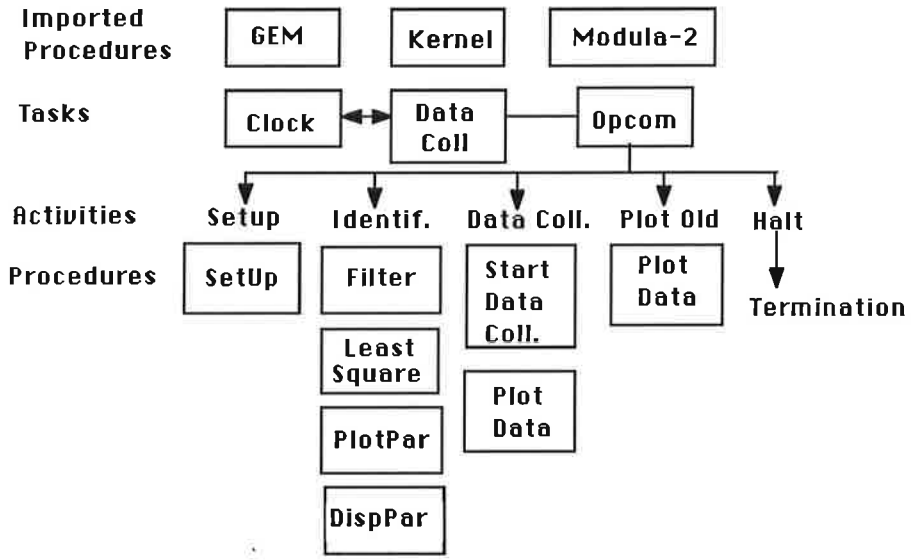


Figure B.1 : Program organization.

In figure B.1 a try is made to show the program organisation. All procedures are not shown.

```
MODULE Main;

(* THIS PROGRAM WAS CONSTRUCTED DURING A MASTER PROJECT *)
(* AT THE DEPARTMENT OF AUTOMATIC CONTROL , LUND *)
(* INSTITUTE OF TECHNOLOGY BY ANDERS J.G. SVENSSON E83 *)
(* FROM JUNE TO AUGUST 1986 *)

FROM Kernel IMPORT SetPriority,MaxPriority,CreateProcess,
                   InitKernel,WaitTime,InitSem,Wait,
                   Signal,Tick,Semaphore,Event,SetTimeOut,
                   InitEvent,Await,Cause;
FROM ConvReal IMPORT StringToReal,RealToString;
FROM GEMin IMPORT ReadString;
FROM MathLib IMPORT sqrt,exp;
FROM GEMtypes IMPORT handletype,point,colortype;
FROM GEMset IMPORT VirtualScreen,SetViewPort,SetWindow,
                   SetFillColor,Shutdown;
FROM GEMout IMPORT Text,PolyLine,FillRectangle;
FROM AnalogIO IMPORT ADIn;
FROM NumberConversion IMPORT StringToCard,CardToString;
```

```
CONST   pi=3.1415927;
        l=0.1;
        xaxislength=555.0;
        pyaxislength=90.0;
        dyaxislength=115.0;
        xaxisstart=75.0;
        maxpar=3;
        numpar=3;
        maxinput=2;
        numinput=2;
        maxsample=1600;
        maxtime=32;
        tsamp=0.02;

TYPE    arraytype1=ARRAY[1..maxpar]OF REAL;
        arraytype2=ARRAY[1..maxinput]OF REAL;
        arraytype3=ARRAY[1..maxinput]OF ARRAY[0..1]OF point;
        arraytype4=ARRAY[1..maxpar]OF ARRAY[0..1]OF point;
        arraytype5=ARRAY[1..maxsample]OF REAL;
        matrixtype1=ARRAY[1..maxpar]OF arraytype1;
        matrixtype2=ARRAY[1..maxinput]OF arraytype5;
        matrixtype3=ARRAY[1..maxpar]OF arraytype5;
        montype1=RECORD
                    mutex1: Semaphore;
                    fivec,state,thetavec: arraytype1;
                    pmatrix: matrixtype1;
                 END;
        montype2=RECORD
                    mutex2: Semaphore;
                    dyaxisstart,datamin,datamax: arraytype2;
                    datamatrix: matrixtype2;
                    datapoint: ARRAY[1..numinput]OF point;
                    datapointer: CARDINAL;
                 END;
        montype3=RECORD
                    mutex3: Semaphore;
                    pyaxisstart,parmax,parmin: arraytype1;
                    parmatrix: matrixtype3;
                    parpoint: ARRAY[1..numpar]OF point;
                    parpointer: CARDINAL;
                 END;
        montype4=RECORD
                    mutex4: Semaphore;
                    clocksync,dataready: Event;
                    num: CARDINAL;
                    collect: BOOLEAN;
                 END;

VAR     mon1: montype1;
        mon2: montype2;
        mon3: montype3;
        mon4: montype4;
        tstart,time,lambda,tau,filtercoeff1,filtercoeff2,
        pinit,Bupper,Blower,Aupper,Alower: REAL;
```

```
        firsttime,lasttime,firsample,lastsample,
        numsample,count1,count2: CARDINAL;
        handle: handletype;
        letter: CHAR;


PROCEDURE Opcom; (* Process *)
(* Handles the communication with the user *)

VAR  p: ARRAY[0..1]OF point;
     s: ARRAY[0..80]OF CHAR;
     cont: BOOLEAN;
     count: CARDINAL;

BEGIN
   InitGraphics;
   SetPriority(5);
   WHILE TRUE DO
      s[0]: ="1";
      cont: =FALSE;
      WHILE NOT cont DO
         IF (s[0]="D") OR (s[0]="I") OR (s[0]="S") OR
            (s[0]="P") OR (s[0]="H") OR (s[0]="d") OR
            (s[0]="i") OR (s[0]="s") OR (s[0]="p") OR
            (s[0]="h") THEN
            cont: =TRUE
         ELSE
            BlankOpline(p[0]);
            Text(handle,p[0],
                 "DATACOLL.(D)  IDENT.(I)  PLOLD (P)
                  SETUP (S)  HALT (H)  ?");
            p[0].h: =545.0;
            s[0]: =CHR(0);
            ReadString(handle,p[0],s);
         END
      END; (* while *)
      s[0]: =CAP(s[0]);
      BlankOpline(p[0]);
      CASE s[0] OF
         "I": Text(handle,p[0],"IDENTIFICATION");
              InitMonType1(mon1);
              InitMonType3(mon3);
              ResetDataPointer(mon2,firstsample-1);
              ResetParPointer(mon3,firstsample-1);
              time: =0.0;
              FOR count: =firstsample TO lastsample DO
                 Filter;
                 Leastsq
              END;
              ComputeParMaxMin(mon3);
              BlankThreeAxes;
              ResetParPointer(mon3,firstsample-1);
              PlotPar;
```

```
            MarkParAxes;
            DisplayPar(mon3)|
        "D": Text(handle,p[0],"DATACOLLECTION");
            InitMontype2(mon2);
            StartDataColl;
            BlankTwoAxes;
            firstsample:=1;
            lastsample:=numsample;
            firsttime:=0;
            lasttime:=numsample DIV TRUNC(1.0/tsamp+0.5);
            ResetDataPointer(mon2,0);
            ComputeDataMaxMin(mon2);
            PlotData;
            MarkDataAxes|
        "P": Text(handle,p[0],"PLOT OLD DATA");
            ResetDataPointer(mon2,firstsample-1);
            BlankTwoAxes;
            ComputeDataMaxMin(mon2);
            PlotData;
            MarkDataAxes|
        "S": SetUp|
        "H": Shutdown;
            HALT
      END; (* case *)
   END; (* while true *)
END Opcom;



PROCEDURE InitGraphics;
(* Used by Opcom to init the window of the GEM system and *)
(* set up the graphics of the window *)

VAR   p:ARRAY[0..1]OF point;
      s:ARRAY[0..79]OF CHAR;

BEGIN
   VirtualScreen(handle);
   p[0].h:=0.0;
   p[0].v:=1.0;
   p[1].h:=1.5;
   p[1].v:=0.0;
   SetViewPort(handle,p[0],p[1]);
   p[0].h:=1.0;
   p[0].v:=350.0;
   p[1].h:=640.0;
   p[1].v:=1.0;
   SetWindow(handle,p[0],p[1]);
   SetFillColor(handle,white);
   p[0].h:=5.0;
   p[0].v:=337.0;
   s:="PARAMETERS :     PAR1 =            PAR2 =
                  PAR3 = ";
   s[60]:=CHR(0);
```

```
   Text(handle,p[0],s);
   p[0].v:=330.0;
   p[0].h:=1.0;
   p[1].h:=640.0;
   p[1].v:=330.0;
   PolyLine(handle,p,2);
   p[0].h:=5.0;
   p[0].v:=7.0;
   s:="OPLINE : ";
   s[9]:=CHR(0);
   Text(handle,p[0],s);
   p[0].h:=1.0;
   p[0].v:=20.0;
   p[1].h:=640.0;
   p[1].v:=20.0;
   PolyLine(handle,p,2);
END InitGraphics;



PROCEDURE SetUp;
(* Used by Opcom to allow the user to alter parameters *)
(* and print out the P-matrix *)

CONST   numchar=7;

VAR     p:ARRAY[0..3]OF point;
        s:ARRAY[0..19]OF CHAR;
        pmatrix:matrixtype1;
        done,cont1,cont2:BOOLEAN;
        pos,meastime:CARDINAL;

BEGIN
   PrintSetUpWindow;
   cont1:=FALSE;
   WHILE NOT cont1 DO
      cont2:=FALSE;
      REPEAT
         BlankOpline(p[0]);
         Text(handle,p[0],"TYPE YOUR CHOICE (A - H) : ");
         p[0].h:=300.0;
         s[0]:=CHR(0);
         ReadString(handle,p[0],s);
         IF ((s[0]>="A") AND (s[0]<="H")) OR
            ((s[0]>="a") AND (s[0]<="h")) THEN
               cont2:=TRUE
         END
      UNTIL cont2;
      s[0]:=CAP(s[0]);
      BlankOpline(p[0]);
      pos:=0;
      CASE s[0] OF
         "A":Text(handle,p[0],
                  "forgetting factor LAMBDA : ");
```

```
        p[0].h:=300.0;
        s[0]:=CHR(0);
        ReadString(handle,p[0],s);
        StringToReal(s,pos,lambda);
        p[0].h:=262.0;
        p[0].v:=200.0;
        Text(handle,p[0],"        ");
        RealToString(lambda,s,numchar);
        s[numchar]:=CHR(0);
        Text(handle,p[0],s)|
"B":    Text(handle,p[0],"filter constant TAU :  ");
        p[0].h:=260.0;
        s[0]:=CHR(0);
        ReadString(handle,p[0],s);
        StringToReal(s,pos,tau);
        p[0].h:=222.0;
        p[0].v:=180.0;
        Text(handle,p[0],"        ");
        RealToString(tau,s,numchar);
        s[numchar]:=CHR(0);
        Text(handle,p[0],s);
        filtercoeff1:=exp(-(tsamp/tau));
        filtercoeff2:=1.0-filtercoeff1;
        tstart:=3.0*tau|
"C":    Text(handle,p[0],"A upper :  ");
        p[0].h:=160.0;
        s[0]:=CHR(0);
        ReadString(handle,p[0],s);
        StringToReal(s,pos,Aupper);
        p[0].h:=126.0;
        p[0].v:=160.0;
        Text(handle,p[0],"        ");
        RealToString(Aupper,s,numchar);
        s[numchar]:=CHR(0);
        Text(handle,p[0],s);
        BlankOpline(p[0]);
        pos:=0;
        Text(handle,p[0],"A lower :  ");
        p[0].h:=160.0;
        s[0]:=CHR(0);
        ReadString(handle,p[0],s);
        StringToReal(s,pos,Alower);
        p[0].h:=286.0;
        p[0].v:=160.0;
        Text(handle,p[0],"        ");
        RealToString(Alower,s,numchar);
        s[numchar]:=CHR(0);
        Text(handle,p[0],s)|
"D":    Text(handle,p[0],"B upper :  ");
        p[0].h:=160.0;
        s[0]:=CHR(0);
        ReadString(handle,p[0],s);
        StringToReal(s,pos,Bupper);
        p[0].h:=126.0;
```

```
    p[0].v:=140.0;
    Text(handle,p[0],"       ");
    RealToString(Bupper,s,numchar);
    s[numchar]:=CHR(0);
    Text(handle,p[0],s);
    BlankOpline(p[0]);
    pos:=0;
    Text(handle,p[0],"B lower : ");
    p[0].h:=160.0;
    s[0]:=CHR(0);
    ReadString(handle,p[0],s);
    StringToReal(s,pos,Blower);
    p[0].h:=286.0;
    p[0].v:=140.0;
    Text(handle,p[0],"       ");
    RealToString(Blower,s,numchar);
    s[numchar]:=CHR(0);
    Text(handle,p[0],s)|
"E":REPEAT
       BlankOpline(p[0]);
       Text(handle,p[0],"Measurementtime in
            seconds , max 32 : ");
       p[0].h:=390.0;
       s[0]:=CHR(0);
       ReadString(handle,p[0],s);
       StringToCard(s,meastime,done);
       IF done THEN
          IF (meastime>maxtime) OR
          (FLOAT(meastime)<tstart) THEN
             done:=FALSE
          END
       END;
    UNTIL done;
    p[0].h:=190.0;
    p[0].v:=120.0;
    Text(handle,p[0],"  ");
    Text(handle,p[0],s);
    numsample:=meastime*TRUNC(1.0/tsamp+0.5);
    CardToString(numsample,s,4);
    s[4]:=CHR(0);
    p[0].h:=570.0;
    p[0].v:=100.0;
    Text(handle,p[0],"    ");
    Text(handle,p[0],s)|
"F":Text(handle,p[0],
        "Initial value for P-matrix : ");
    p[0].h:=315.0;
    s[0]:=CHR(0);
    ReadString(handle,p[0],s);
    StringToReal(s,pos,pinit);
    p[0].h:=278.0;
    p[0].v:=100.0;
    Text(handle,p[0],"       ");
    RealToString(pinit,s,numchar);
```

```
            s[numchar]:=CHR(0);
            Text(handle,p[0],s)|
         "G": InquireFirstLastTime|
         "H": cont1:=TRUE
      END (* case *)
   END; (* while *)
   BlankMainWindow
END SetUp;


PROCEDURE InquireFirstLastTime;
(* Continuation of SetUp *)

VAR     p: ARRAY[0..1]OF point;
        s: ARRAY[0..19]OF CHAR;
        done: BOOLEAN;

BEGIN
   REPEAT
      BlankOpline(p[0]);
      Text(handle,p[0],
           "firsttime ( 0 < firsttime < lasttime ) : ");
      p[0].h:=400.0;
      s[0]:=CHR(0);
      ReadString(handle,p[0],s);
      StringToCard(s,firsttime,done);
      IF done THEN
         IF (firsttime>lasttime) THEN
            done:=FALSE
         END
      END
   UNTIL done;
   p[0].h:=142.0;
   p[0].v:=80.0;
   Text(handle,p[0],"  ");
   Text(handle,p[0],s);
   REPEAT
      BlankOpline(p[0]);
      Text(handle,p[0],"lasttime
           ( firsttime + 1 < lasttime < 32 ) : ");
      p[0].h:=435.0;
      s[0]:=CHR(0);
      ReadString(handle,p[0],s);
      StringToCard(s,lasttime,done);
      IF done THEN
         IF (lasttime<(firsttime+1)) OR
         (lasttime>maxtime) THEN
            done:=FALSE
         END
      END
   UNTIL done;
   p[0].h:=262.0;
   p[0].v:=80.0;
```

```
   Text(handle,p[0],"  ");
   Text(handle,p[0],s);
   firstsample:=firsttime*TRUNC(1.0/tsamp+0.5)+1;
   lastsample:=lasttime*TRUNC(1.0/tsamp+0.5);
   p[0].h:=500.0;
   p[0].v:=60.0;
   Text(handle,p[0],"    ");
   CardToString(lastsample-firstsample+1,s,4);
   s[4]:=CHR(0);
   Text(handle,p[0],s)
END InquireFirstLastTime;



PROCEDURE PrintSetUpWindow;
(* Used by SetUp to print the menu and graphics for the *)
(* setup session *)

CONST   numchar=7;

VAR     p:ARRAY[0..3]OF point;
        s:ARRAY[0..19]OF CHAR;
        pmatr:matrixtype1;
        count1,count2:CARDINAL;

BEGIN
   BlankMainWindow;
   p[0].v:=220.0;
   p[0].h:=30.0;
   Text(handle,p[0],"ALTER :");
   p[0].v:=218.0;
   p[1].v:=218.0;
   p[1].h:=65.0;
   PolyLine(handle,p,2);
   p[0].v:=200.0;
   Text(handle,p[0],
        "A. forgetting factor LAMBDA (       )");
   RealToString(lambda,s,numchar);
   s[numchar]:=CHR(0);
   p[0].h:=262.0;
   Text(handle,p[0],s);
   p[0].v:=180.0;
   p[0].h:=30.0;
   Text(handle,p[0],"B. filter constant TAU (       )");
   RealToString(tau,s,numchar);
   s[numchar]:=CHR(0);
   p[0].h:=222.0;
   Text(handle,p[0],s);
   p[0].v:=160.0;
   p[0].h:=30.0;
   Text(handle,p[0],
        "C. A upper (       )   A lower (       )");
   RealToString(Aupper,s,numchar);
   s[numchar]:=CHR(0);
```

```
p[0].h:=126.0;
Text(handle,p[0],s);
RealToString(Alower,s,numchar);
s[numchar]:=CHR(0);
p[0].h:=286.0;
Text(handle,p[0],s);
p[0].v:=140.0;
p[0].h:=30.0;
Text(handle,p[0],
     "D. B upper (       )   B lower (       )");
RealToString(Bupper,s,numchar);
s[numchar]:=CHR(0);
p[0].h:=126.0;
Text(handle,p[0],s);
RealToString(Blower,s,numchar);
s[numchar]:=CHR(0);
p[0].h:=286.0;
Text(handle,p[0],s);
p[0].v:=120.0;
p[0].h:=30.0;
Text(handle,p[0],"E. Measurementtime (   seconds)");
CardToString(TRUNC(FLOAT(numsample)*tsamp),s,2);
s[2]:=CHR(0);
p[0].h:=190.0;
Text(handle,p[0],s);
p[0].v:=100.0;
p[0].h:=30.0;
Text(handle,p[0],
     "F. Initial value for P-matrix (       )");
RealToString(pinit,s,numchar);
s[numchar]:=CHR(0);
p[0].h:=278.0;
Text(handle,p[0],s);
p[0].v:=80.0;
p[0].h:=30.0;
Text(handle,p[0],
     "G. firsttime (  )  lasttime (  ) : sec");
CardToString(firsttime,s,2);
s[2]:=CHR(0);
p[0].h:=142.0;
Text(handle,p[0],s);
CardToString(lasttime,s,2);
s[2]:=CHR(0);
p[0].h:=262.0;
Text(handle,p[0],s);
p[0].v:=60.0;
p[0].h:=30.0;
Text(handle,p[0],"H. Exit from SETUP");
p[0].h:=370.0;
p[0].v:=329.0;
p[1].v:=21.0;
p[1].h:=370.0;
PolyLine(handle,p,2);
p[0].h:=465.0;
```

```
   p[0].v:=280.0;
   Text(handle,p[0],"P-MATRIX = ");
   p[0].h:=405.0;
   p[0].v:=130.0;
   p[1].h:=390.0;
   p[1].v:=130.0;
   p[2].h:=390.0;
   p[2].v:=240.0;
   p[3].h:=405.0;
   p[3].v:=240.0;
   PolyLine(handle,p,4);
   p[0].h:=610.0;
   p[1].h:=625.0;
   p[2].h:=625.0;
   p[3].h:=610.0;
   PolyLine(handle,p,4);
   GetPmatrix(pmatr,mon1);
   p[0].h:=406.0;
   p[0].v:=220.0;
   FOR count1:=1 TO numpar DO
      FOR count2:=1 TO numpar DO
         RealToString(pmatr[count1,count2],s,numchar);
         s[numchar]:=CHR(0);
         Text(handle,p[0],s);
         p[0].h:=p[0].h+72.0
      END;
      p[0].v:=p[0].v-40.0;
      p[0].h:=406.0
   END;
   p[0].h:=410.0;
   p[0].v:=100.0;
   Text(handle,p[0],"Samples collected = ");
   p[0].h:=570.0;
   CardToString(numsample,s,4);
   s[4]:=CHR(0);
   Text(handle,p[0],s);
   p[0].h:=410.0;
   p[0].v:=80.0;
   Text(handle,p[0],"Samples for Ident/Oldplot = ");
   p[0].v:=60.0;
   p[0].h:=500.0;
   CardToString(lastsample-firstsample+1,s,4);
   s[4]:=CHR(0);
   Text(handle,p[0],s)
END PrintSetUpWindow;


PROCEDURE GetPmatrix(VAR pmat:matrixtype1;
                     VAR mon1:montype1);
(* Used by SetUp to copy the P-matrix from the structured *)
(* datatype mon1 *)

VAR   count1,count2:CARDINAL;
```

```
BEGIN
   WITH mon1 DO
      Wait(mutex1);
      FOR count1:= 1 TO numpar DO
         FOR count2:= 1 TO numpar DO
            pmat[count1,count2]:=pmatrix[count1,count2]
         END
      END;
      Signal(mutex1)
   END; (* with *)
END GetPmatrix;



PROCEDURE StartDataColl;
(* Used by Opcom to set a flag in the structured datatype *)
(* mon4 . The flag is then tested by DataColl *)

BEGIN
   WITH mon4 DO
      Wait(mutex4);
      collect:=TRUE;
      Await(dataready);
      Signal(mutex4)
   END (* with *)
END StartDataColl;



PROCEDURE ComputeDataMaxMin(VAR mon2: montype2);
(* Used to update max and min datavalues to get a correct *)
(* plot *)

VAR   count: CARDINAL;

BEGIN
   WITH mon2 DO
      Wait(mutex2);
      FOR count:=1 TO numinput DO
         datamax[count]:=-1.0E20;
         datamin[count]:=1.0E20
      END;
      FOR count:=firstsample TO lastsample DO
         IF datamatrix[1,count]>datamax[1] THEN
            datamax[1]:=datamatrix[1,count]
         END;
         IF datamatrix[1,count]<datamin[1] THEN
            datamin[1]:=datamatrix[1,count]
         END;
         IF datamatrix[2,count]>datamax[2] THEN
            datamax[2]:=datamatrix[2,count]
         END;
         IF datamatrix[2,count]<datamin[2] THEN
            datamin[2]:=datamatrix[2,count]
```

```
         END
      END;
      Signal(mutex2)
   END; (* with *)
END ComputeDataMaxMin;


PROCEDURE ComputeParMaxMin(VAR mon3: montype3);
(* Used to compute max and min values to get a correct *)
(* plot *)

VAR count1,count2: CARDINAL;

BEGIN
   WITH mon3 DO
      Wait(mutex3);
      FOR count1: =1 TO numpar DO
         parmax[count1]: =-1.0E20;
         parmin[count1]: =1.0E20
      END;
      FOR count1: =firstsample TO lastsample DO
         FOR count2: =1 TO numpar DO
            IF parmatrix[count2,count1]<parmin[count2] THEN
               parmin[count2]: =parmatrix[count2,count1]
            END;
            IF parmatrix[count2,count1]>parmax[count2] THEN
               parmax[count2]: =parmatrix[count2,count1]
            END
         END;
      END;
     Signal(mutex3)
   END; (* with *)
END ComputeParMaxMin;


PROCEDURE BlankMainWindow;
(* Blanks the main window of all text , graphics .. *)

VAR   p: ARRAY[0..1]OF point;

BEGIN
   p[0].h: =1.0;
   p[0].v: =329.0;
   p[1].h: =640.0;
   p[1].v: =21.0;
   FillRectangle(handle,p[0],p[1]);
END BlankMainWindow;
```

```
PROCEDURE BlankOpline(VAR p:point);
(* Blanks the text on the opline and positions cursor p *)
(* at the begining *)

VAR cursor1,cursor2:point;

BEGIN
   p.h:=80.0;
   p.v:=7.0;
   cursor1.h:=80.0;
   cursor1.v:=19.0;
   cursor2.h:=640.0;
   cursor2.v:=1.0;
   FillRectangle(handle,cursor1,cursor2)
END BlankOpline;



PROCEDURE MarkDataAxes;
(* Used by Opcom to mark axes with min and max values , *)
(* and time *)

VAR cursor:point;
    numchar:CARDINAL;
    s:ARRAY[0..19]OF CHAR;
    maxvec,minvec:arraytype2;
    number:ARRAY[0..1]OF CHAR;

BEGIN
   GetDataMaxMin(maxvec,minvec,mon2);
   cursor.h:=5.0;
   cursor.v:=300.0;
   numchar:=7;
   RealToString(maxvec[1],s,numchar);
   s[7]:=CHR(0);
   Text(handle,cursor,s);
   cursor.v:=240.0;
   Text(handle,cursor,"Radius B");
   cursor.v:=194.0;
   RealToString(minvec[1],s,numchar);
   s[7]:=CHR(0);
   Text(handle,cursor,s);
   cursor.v:=165.0;
   RealToString(maxvec[2],s,numchar);
   s[7]:=CHR(0);
   Text(handle,cursor,s);
   cursor.v:=106.0;
   Text(handle,cursor,"Radius A");
   cursor.v:=59.0;
   RealToString(minvec[2],s,numchar);
   s[7]:=CHR(0);
   Text(handle,cursor,s);
   cursor.h:=75.0;
   cursor.v:=40.0;
```

```
   numchar: =2;
   CardToString(firsttime,number,numchar);
   Text(handle,cursor,number);
   CardToString(lasttime,number,numchar);
   cursor.h: =610.0;
   Text(handle,cursor,number);
   cursor.v: =177.0;
   Text(handle,cursor,number);
   cursor.h: =75.0;
   CardToString(firsttime,number,numchar);
   Text(handle,cursor,number)
END MarkDataAxes;



PROCEDURE GetDataMaxMin(VAR maxvec,minvec: arraytype2;
                        VAR mon2: montype2);
(* Used by MarkDataAxes to copy min and max values from *)
(* structured datatype mon2 *)

VAR count: CARDINAL;

BEGIN
   WITH mon2 DO
      Wait(mutex2);
      FOR count: =1 TO numinput DO
         maxvec[count]: =datamax[count];
         minvec[count]: =datamin[count]
      END;
      Signal(mutex2)
   END (* with *)
END GetDataMaxMin;



PROCEDURE MarkParAxes;
(* Used by Opcom to mark axes with min and max values , *)
(* and time *)

VAR cursor: point;
    numchar: CARDINAL;
    s: ARRAY[0..19]OF CHAR;
    maxvec,minvec: arraytype1;
    number: ARRAY[0..1]OF CHAR;

BEGIN
   GetParMaxMin(maxvec,minvec,mon3);
   cursor.h: =5.0;
   cursor.v: =309.0;
   numchar: =7;
   RealToString(maxvec[1],s,numchar);
   s[7]: =CHR(0);
   Text(handle,cursor,s);
   cursor.v: =264.0;
```

```
   Text(handle,cursor,"Par 1");
   cursor.v:=229.0;
   RealToString(minvec[1],s,numchar);
   s[7]:=CHR(0);
   Text(handle,cursor,s);
   cursor.v:=209.0;
   RealToString(maxvec[2],s,numchar);
   s[7]:=CHR(0);
   Text(handle,cursor,s);
   cursor.v:=164.0;
   Text(handle,cursor,"Par 2");
   cursor.v:=129.0;
   RealToString(minvec[2],s,numchar);
   s[7]:=CHR(0);
   Text(handle,cursor,s);
   cursor.v:=109.0;
   RealToString(maxvec[3],s,numchar);
   s[7]:=CHR(0);
   Text(handle,cursor,s);
   cursor.v:=64.0;
   Text(handle,cursor,"Par 3");
   cursor.v:=29.0;
   RealToString(minvec[3],s,numchar);
   s[7]:=CHR(0);
   Text(handle,cursor,s);
   cursor.h:=75.0;
   cursor.v:=22.0;
   numchar:=2;
   CardToString(firsttime,number,numchar);
   Text(handle,cursor,number);
   cursor.v:=118.0;
   Text(handle,cursor,number);
   cursor.v:=218.0;
   Text(handle,cursor,number);
   cursor.h:=610.0;
   CardToString(lasttime,number,numchar);
   Text(handle,cursor,number);
   cursor.v:=118.0;
   Text(handle,cursor,number);
   cursor.v:=22.0;
   Text(handle,cursor,number);
END MarkParAxes;



PROCEDURE GetParMaxMin(VAR maxvec,minvec:arraytype1;
                       VAR mon3:montype3);
(* Used by MarkDataAxes to copy max and min values from *)
(* structured datatype mon3 *)

VAR count:CARDINAL;

BEGIN
   WITH mon3 DO
```

```
      Wait(mutex3);
      FOR count:=1 TO numpar DO
         maxvec[count]:=parmax[count];
         minvec[count]:=parmin[count]
      END;
      Signal(mutex3)
   END (* with *)
END GetParMaxMin;



PROCEDURE BlankThreeAxes;
(* Used by Opcom to draw axes for the plot of parameters *)

VAR   p:ARRAY[0..1]OF point;

BEGIN
   BlankMainWindow;
   p[0].h:=75.0;
   p[0].v:=319.0;
   p[1].h:=75.0;
   p[1].v:=229.0;
   PolyLine(handle,p,2);
   p[0].v:=219.0;
   p[1].v:=129.0;
   PolyLine(handle,p,2);
   p[0].v:=119.0;
   p[1].v:=29.0;
   PolyLine(handle,p,2);
   p[0]:=p[1];
   p[1].h:=630.0;
   PolyLine(handle,p,2);
   p[0].v:=129.0;
   p[1].v:=129.0;
   PolyLine(handle,p,2);
   p[0].v:=229.0;
   p[1].v:=229.0;
   PolyLine(handle,p,2)
END BlankThreeAxes;



PROCEDURE BlankTwoAxes;
(* Used by Opcom to draw axes for the plot of *)
(* collected data *)

VAR   p:ARRAY[0..1]OF point;

BEGIN
   BlankMainWindow;
   p[0].h:=75.0;
   p[0].v:=309.0;
   p[1].h:=75.0;
   p[1].v:=194.0;
```

```
   PolyLine(handle,p,2);
   p[0].v:=174.0;
   p[1].v:=59.0;
   PolyLine(handle,p,2);
   p[0]:=p[1];
   p[1].h:=630.0;
   PolyLine(handle,p,2);
   p[0].v:=194.0;
   p[1].v:=194.0;
   PolyLine(handle,p,2)
END BlankTwoAxes;



PROCEDURE Filter;
(* Used in Opcom under IDENTIFICATON to filter *)
(* collected data *)

VAR   fivec,oldstate,newstate,datavec:arraytype1;
      count:INTEGER;
      width:CARDINAL;
      s:ARRAY[0..9]OF CHAR;

BEGIN
   GetStateIndata(oldstate,datavec,mon1,mon2);
   FOR count:=1 TO numpar DO
      newstate[count]:=filtercoeff1*oldstate[count]
                       +(1.0+filtercoeff1)*datavec[count]
   END;
   FOR count:=1 TO numpar DO
      fivec[count]:=(datavec[count]+oldstate[count])
                    *filtercoeff2/2.0
   END;
   PutFi(fivec,newstate,mon1)
END Filter;



PROCEDURE GetStateIndata(VAR svec,datavec:arraytype1;
                VAR mon1:montype1;VAR mon2:montype2);
(* Used by Filter to copy collected data and states *)
(* of filters from structured datatypes mon1 and mon2 *)

VAR   count:INTEGER;

BEGIN
   WITH mon1 DO
      Wait(mutex1);
      FOR count:=1 TO numpar DO
         svec[count]:=state[count];
      END;
      Signal(mutex1)
   END; (* with *)
   WITH mon2 DO
```

```
      Wait(mutex2);
      INC(datapointer);
      FOR count:=1 TO numinput DO
         datavec[count]:=datamatrix[count,datapointer]*
                         datamatrix[count,datapointer]
      END;
      datavec[3]:=1.0;
      Signal(mutex2)
   END; (* with *)
END GetStateIndata;



PROCEDURE PutFi(VAR fvec,nstate:arraytype1;
                VAR mon1:montype1);
(* Used by Filter to put states of filters in structured *)
(* datatype mon1 *)

VAR   count:INTEGER;

BEGIN
   WITH mon1 DO
      Wait(mutex1);
      FOR count:=1 TO numpar DO
         fivec[count]:=fvec[count];
         state[count]:=nstate[count]
      END;
      Signal(mutex1)
   END; (* with *)
END PutFi;



PROCEDURE Leastsq;
(* Used in Opcom under IDENTIFICATION . Performs an *)
(* ordinary recursive least square identification. *)
(* The start of  identification is delayed to wait *)
(* out filter transients *)

VAR     thetavec,fivec,pfivec,kvec,parvec:arraytype1;
        pmat:matrixtype1;
        count1,count2:INTEGER;
        eps,y,divisor:REAL;

BEGIN
   GetThetaFiPY(thetavec,fivec,pmat,y,mon1,mon2);
   time:=time+tsamp;
   IF time>tstart THEN
      eps:=y;
      FOR count1:=1 TO numpar DO
         eps:=eps-fivec[count1]*thetavec[count1]
      END;
      FOR count1:=1 TO numpar DO
         pfivec[count1]:=0.0
```

```
      END;
      FOR count1:=1 TO numpar DO
         FOR count2:=1 TO numpar DO
            pfivec[count1]:=pfivec[count1]+
               pmat[count1,count2]*fivec[count2]
         END
      END;
      divisor:=lambda;
      FOR count1:=1 TO numpar DO
         divisor:=divisor+fivec[count1]*pfivec[count1]
      END;
      FOR count1:=1 TO numpar DO
         FOR count2:=count1 TO numpar DO
            pmat[count1,count2]:=
               (pmat[count1,count2]-pfivec[count1]
                  *pfivec[count2]/divisor)/lambda;
            pmat[count2,count1]:=pmat[count1,count2]
         END
      END;
      FOR count1:=1 TO numpar DO
         kvec[count1]:=pfivec[count1]/divisor
      END;
      FOR count1:=1 TO numpar DO
         thetavec[count1]:=thetavec[count1]+
                           kvec[count1]*eps
      END;
   END; (* if *)
   parvec[1]:=(-thetavec[1]+1.0)/tau*2.0*pi*l;
   parvec[2]:=thetavec[2]/tau*2.0*pi*l;
   parvec[3]:=thetavec[3]/tau*2.0*pi*l;
   PutParThetaP(parvec,thetavec,pmat,mon1,mon3)
END Leastsq;


PROCEDURE GetThetaFiPY(VAR tvec,fvec:arraytype1;
              VAR pmat:matrixtype1;VAR y:REAL;
              VAR mon1:montype1;VAR mon2:montype2);
(* Used by Leastsq to copy needed data from structured *)
(* datatype mon1 and mon2 *)

VAR   count1,count2:INTEGER;

BEGIN
   WITH mon1 DO
      Wait(mutex1);
      FOR count1:=1 TO numpar DO
         fvec[count1]:=fivec[count1];
         tvec[count1]:=thetavec[count1]
      END;
      FOR count1:=1 TO numpar DO
         FOR count2:=1 TO numpar DO
            pmat[count1,count2]:=pmatrix[count1,count2]
         END
```

```
      END;
      Signal(mutex1)
   END; (* with *)
   WITH mon2 DO
      Wait(mutex2);
      y:=datamatrix[1,datapointer]*
         datamatrix[1,datapointer];
      Signal(mutex2)
   END (* with *)
END GetThetaFiPY;


PROCEDURE PutParThetaP(VAR parvec,tvec: arraytype1;
                  VAR pmat: matrixtype1; VAR mon1: montype1;
                  VAR mon3: montype3);
(* Used by Leastsq to put updated data in structured *)
(* datatype mon1 and mon3 *)

VAR   count1,count2: INTEGER;

BEGIN
   WITH mon1 DO
      Wait(mutex1);
      FOR count1:=1 TO numpar DO
         thetavec[count1]:=tvec[count1];
         FOR count2:=1 TO numpar DO
            pmatrix[count1,count2]:=pmat[count1,count2]
         END
      END;
      Signal(mutex1)
   END; (* with *)
   WITH mon3 DO
      Wait(mutex3);
      INC(parpointer);
      FOR count1:=1 TO numpar DO
         parmatrix[count1,parpointer]:=parvec[count1]
      END;
      Signal(mutex3)
   END (* with *)
END PutParThetaP;


PROCEDURE Clock; (* Process *)
(* Used for the datacollection to generate sampling *)
(* intervals *)

BEGIN
   SetPriority(3);
   WHILE TRUE DO
      WaitTime(2);
      WITH mon4 DO
         Wait(mutex4);
```

```
         num:=num+1;
         Cause(clocksync);
         Signal(mutex4)
      END (* with*)
   END (* while *)
END Clock;


PROCEDURE DataColl; (* Process *)
(* Collects data from analog inputs *)

VAR   ra,rb:REAL;
      collect:BOOLEAN;
      width,count:CARDINAL;
      cursor:point;
      s:ARRAY[0..9]OF CHAR;

BEGIN
   SetPriority(4);
   WHILE TRUE DO
      WITH mon4 DO
         Wait(mutex4);
         IF num=0 THEN Await(clocksync) END;
         num:=num-1;
         IF collect THEN
            ra:=ADIn(1);    (* channel 1 point a *)
            rb:=ADIn(0);    (* channel 0 point b *)
            rb:=Blower+(Bupper-Blower)*(rb+1.0)/2.0;
                (* interval Bupper - Blower *)
            ra:=Alower+(Aupper-Alower)*(ra+1.0)/2.0;
                (* interval Aupper - Alower *)
            PutData(rb,ra,mon2,collect);
            IF NOT collect THEN Cause(dataready) END
         END;
         Signal(mutex4)
      END (* with *)
   END (* while true *)
END DataColl;


PROCEDURE PutData(VAR rb,ra:REAL;VAR mon2:montype2;
                  VAR coll:BOOLEAN);
(* Used by DataColl to place collected data in structured *)
(* datatype mon2 *)

BEGIN
   WITH mon2 DO
      Wait(mutex2);
      INC(datapointer);
      coll:=NOT (datapointer=numsample);
      datamatrix[1,datapointer]:=rb;
      datamatrix[2,datapointer]:=ra;
```

```
      Signal(mutex2)
   END (* with *)
END PutData;


PROCEDURE PlotData;
(* Plots the collected data *)

VAR   pointarray: arraytype3;
      count: INTEGER;
      stop: BOOLEAN;

BEGIN
   stop: =FALSE;
   WHILE NOT stop DO
      GetData(pointarray,mon2,stop);
      FOR count: =1 TO numinput DO
         PolyLine(handle,pointarray[count],2)
      END
   END
END PlotData;


PROCEDURE GetData(VAR dpointarray: arraytype3;
                  VAR mon2: montype2; VAR dstop: BOOLEAN);
(* Used by PlotData to compute points for the graphics *)

VAR   count: CARDINAL;

BEGIN
   WITH mon2 DO
      Wait(mutex2);
      INC(datapointer);
      dstop: =(datapointer=lastsample);
      FOR count: =1 TO numinput DO
         dpointarray[count,0]: =datapoint[count];
         dpointarray[count,1].h: =
            xaxislength*FLOAT(datapointer-firstsample)/
            FLOAT(lastsample-firstsample)+xaxisstart;
         dpointarray[count,1].v: =
            dyaxisstart[count]+(datamatrix[count,
            datapointer]-datamin[count])/(datamax[
            count]-datamin[count])*dyaxislength;
         datapoint[count]: =dpointarray[count,1]
      END;
      Signal(mutex2)
   END (* with *)
END GetData;
```

```
PROCEDURE PlotPar;
(* Plots parameters on the screen *)

VAR   count: INTEGER;
      stop: BOOLEAN;
      pointarray: arraytype4;

BEGIN
   stop: =FALSE;
   WHILE NOT stop DO
      GetPar(pointarray,mon3,stop);
      FOR count: =1 TO numpar DO
         PolyLine(handle,pointarray[count],2)
      END
   END (* while *)
END PlotPar;



PROCEDURE GetPar(VAR pointarray: arraytype4;
             VAR mon3: montype3; VAR stop: BOOLEAN);
(* Used by PlotPar to compute points for the graphics *)

VAR   count: CARDINAL;

BEGIN
   WITH mon3 DO
      Wait(mutex3);
      INC(parpointer);
      stop: =(parpointer=lastsample);
      FOR count: =1 TO numpar DO
         pointarray[count,0]: =parpoint[count];
         pointarray[count,1].h: =
            xaxislength*FLOAT(parpointer-firstsample)/
            FLOAT(lastsample-firstsample)+xaxisstart;
         pointarray[count,1].v: =
            pyaxisstart[count]+(parmatrix[count,
            parpointer]-parmin[count])/
            (parmax[count]-parmin[count])*pyaxislength;
         parpoint[count]: =pointarray[count,1]
      END;
      Signal(mutex3)
   END (* with *)
END GetPar;
```

```
PROCEDURE DisplayPar(VAR mon3: montype3);
(* Used in Opcom to display the last values of *)
(* parameters on the top of the screen *)

VAR   cursor: point;
      par1,par2,par3: REAL;
      sp: ARRAY[0..79]OF CHAR;
      numchar: CARDINAL;

BEGIN
   WITH mon3 DO
      Wait(mutex3);
      par1: =parmatrix[1,lastsample];
      par2: =parmatrix[2,lastsample];
      par3: =parmatrix[3,lastsample];
      Signal(mutex3)
   END; (* with *)
   cursor.h: =195.0;
   cursor.v: =337.0;
   numchar: =8;
   RealToString(par1,sp,numchar);
   sp[8]: =CHR(0);
   Text(handle,cursor,sp);
   cursor.h: =345.0;
   RealToString(par2,sp,numchar);
   sp[8]: =CHR(0);
   Text(handle,cursor,sp);
   cursor.h: =500.0;
   RealToString(par3,sp,numchar);
   sp[8]: =CHR(0);
   Text(handle,cursor,sp)
END DisplayPar;



PROCEDURE InitMonType1(VAR mon1: montype1);
(* Init structured datatype mon1 *)

VAR   count1,count2: INTEGER;

BEGIN
   WITH mon1 DO
      Wait(mutex1);
      FOR count1: =1 TO numpar DO
         fivec[count1]: =0.0;
         state[count1]: =0.0;
         thetavec[count1]: =0.0
      END;
      FOR count1: =1 TO numpar DO
         FOR count2: =1 TO numpar DO
            pmatrix[count1,count2]: =0.0
         END;
         pmatrix[count1,count1]: =pinit
      END;
```

```
      Signal(mutex1)
   END (* with *)
END InitMonType1;


PROCEDURE InitMontype2(VAR mon2: montype2);
(* Init structured datatype mon2 *)

VAR count: CARDINAL;

BEGIN
   WITH mon2 DO
      Wait(mutex2);
      dyaxisstart[1]: =194.0;
      dyaxisstart[2]: =59.0;
      FOR count: =1 TO numinput DO
         datamin[count]: =1.0E20;
         datamax[count]: =-1.0E20
      END;
      datapointer: =0;
      FOR count: =1 TO numinput DO
         datapoint[count].h: =xaxisstart;
         datapoint[count].v: =dyaxisstart[count]
      END;
      Signal(mutex2)
   END (* with *)
END InitMontype2;


PROCEDURE InitMonType3(VAR mon3: montype3);
(* Init structured datatype mon3 *)

VAR   count: CARDINAL;

BEGIN
   WITH mon3 DO
      Wait(mutex3);
      pyaxisstart[1]: =229.0;
      pyaxisstart[2]: =129.0;
      pyaxisstart[3]: =29.0;
      FOR count: =1 TO numpar DO
         parmax[count]: =-1.0E20;
         parmin[count]: =1.0E20
      END;
      parpointer: =0;
      FOR count: =1 TO numpar DO
         parpoint[count].h: =xaxisstart;
         parpoint[count].v: =pyaxisstart[count]
      END;
      Signal(mutex3)
   END (* with *)
END InitMonType3;
```

```
PROCEDURE ResetDataPointer(VAR mon2: montype2;
                           resetvalue: CARDINAL);
(* Resets data pointer to resetvalue and inits *)
(* the start of a plot *)

VAR   count: CARDINAL;

BEGIN
   WITH mon2 DO
      Wait(mutex2);
      datapointer:=resetvalue;
      FOR count:=1 TO numinput DO
         datapoint[count].h:=xaxisstart;
         datapoint[count].v:=dyaxisstart[count]
      END;
      Signal(mutex2)
   END (* with *)
END ResetDataPointer;



PROCEDURE ResetParPointer(VAR mon3: montype3;
                          resetvalue: CARDINAL);
(* Resets parameter pointer *)

VAR   count: CARDINAL;

BEGIN
   WITH mon3 DO
      Wait(mutex3);
      parpointer:=resetvalue;
      Signal(mutex3)
   END (* with *)
END ResetParPointer;




(* Main Program *)

(* First some global variables are initialized , then *)
(* Semaphores and Event are initialized *)
BEGIN
   firstsample:=1;
   lastsample:=maxsample;
   firsttime:=0;
   lasttime:=maxtime;
   Aupper:=10.0;
   Alower:=-10.0;
   Bupper:=10.0;
   Blower:=-10.0;
   lambda:=1.0;
   tau:=0.5;
```

```
    filtercoeff1:=exp(-(tsamp/tau));
    filtercoeff2:=1.0-filtercoeff1;
    tstart:=3.0*tau;
    pinit:=50000.0;
    numsample:=1600;
    time:=0.0;
    WITH mon1 DO
       InitSem(mutex1,1)
    END;
    InitMonType1(mon1);
    WITH mon2 DO
       FOR count1:=1 TO numinput DO
          FOR count2:=1 TO maxsample DO
             datamatrix[count1,count2]:=0.0
          END
       END;
       InitSem(mutex2,1)
    END;
    InitMontype2(mon2);
    WITH mon3 DO
       FOR count1:=1 TO numpar DO
          FOR count2:=1 TO maxsample DO
             parmatrix[count1,count2]:=0.0
          END
       END;
       InitSem(mutex3,1)
    END;
    InitMonType3(mon3);
    WITH mon4 DO
       InitSem(mutex4,1);
       InitEvent(clocksync,mutex4);
       InitEvent(dataready,mutex4);
       num:=0;
       collect:=FALSE
    END;
    InitKernel(0);
    CreateProcess(Clock,5000);
    CreateProcess(Opcom,25000);
    CreateProcess(DataColl,10000);
    SetPriority(MaxPriority);
END Main.
```

Appendix C

Users Guide

With this guide the user should get a good understanding of how to run the program on an IBM PC/AT computer with facilities for Modula-2. The actions to be taken are listed below

1. When the computer is switched on and the system is booted , then insert a flexible diskette with the program IDV4.LOD on.

2. Start program execution by typing the command M2 IDV4.LOD and then pressing <RETURN>.

3. The program starts and three windows are displayed. One for estimated parameters , one for plots and at the bottom there is a window marked 'OPLINE' , where commands may be entered. A suitable start is to type an 'S' for setup.

4. The main window is now split into two halves. In the right half the P-matrix in the Recursive Least Square algorithm is displayed with its current elements. Below the P-matrix the number of samples used for data collection and identification/plot of old data are listed. The left half contains a menu with some alterable parameters. To change the value of one of these parameters , just type one of the letters 'A' through 'G' on the OPLINE.

    A. forgetting factor LAMBDA is the λ used in the Recursive Least Square algorithm. Lambda should be set to 1 (default) or perhaps a somewhat lower value , e.g. 0.995.

    B. filter constant TAU is the filter constant τ in the signal processing filters. According to simulations the setting should be 0.1 to 0.5 (default 0.5) , but this is to be further investigated.

C. The parameters 'Aupper' and 'Alower' are scale factors , see Appendix B for an explanation. Default values are 'Aupper' = 10.0 and 'Alower' = -10.0 , which is the higher and lower limit for the used AD converter in Volts.

D. See above at C.

E. Measurementtime is simply the time used to collect data measured with the ultrasonic equipment. Default value is the maximal , namely 32 seconds.

F. Initial value for P-matrix. The P-matrix in the Recursive Least Square algorithm is given this value for the diagonal elements. All the other elements are set to zero. Default value is 50000.

G. The parameters 'firsttime' and 'lasttime' have values in seconds. These parameters are used to select an undisturbed sequence of the collected data , in order to get better identification results. The identification is based only on this sequence. 'firsttime' and 'lasttime' may also be used to "zoom" into collected data , when they are used with the PLOLD (P) command. Default values are 0 and 32 , selecting the longest sequence available.

By typing an 'H' on the OPLINE the setup session is ended.

5. Now everything is ready for the data collection. To start the collection type a 'D' on the OPLINE. The procedure , which reads analog inputs , is activated when <RETURN> is pressed. 'DATACOLLECTION' is echoed on the OPLINE. When the measurement time has elapsed , collected measurements are automatically plotted on the screen and the axes are marked with max and min values.

6. If the plotted sequence seems to be without disturbances , then only type an 'I' for identification and press <RETURN>. However , if the whole sequence can not be used , then a part of it must be selected.

Enter setup with a 'S' on the OPLINE and use firsttime and lasttime to select a good sequence. The resulting outcut may be plotted with the PLOLD (P) command. If the outcut is good then start the identification as described above. 'IDENTIFICATION' is echoed on the OPLINE and when it is completed , the convergence of parameters is plotted. The axes are marked with radius A and radius B for the two measurement points , see chapter 1 and 2. In the upper window the last values of these parameters are displayed. Parameter 1 is the same as β_1' , parameter 2 is the same as β_2' and parameter 3 equals β_3'. All β' parameters are introduced in chapter 2. If the parameters do not converge smoothly , then there may be an undetected disturbance in the data. One way of avoiding this is to reduce the forgetting factor lambda , another is to use a different sequence of data. Adjust the parameters and make a new identification.

7. Program execution may now be continued with more data collections and identifications or terminated with a HALT (H) command.

References

Andersson L (1986) : Kort Beskrivning Av IBM-PC och Modula-2. Department of Automatic Control , Lund Institute of Technology , Lund , Sweden.

Borgström P and Grände P-O (1979) : Myogenic Microvascular Responses To Change Of Transmural Pressure. Acta Physiol Scand 1979 , 106 : 411-423.

Borgström P and Grände P-O (1979) : A Mathematical Description Of The Myogenic Response In The Microcirculation. Acta Physiol Scand 1982 , 116 : 363-376.

Elmqvist H , Mattson S E and Olsson G (1981) : Datorer I Reglersystem - Realtidsprogrammering. Department of Automatic Control , Lund Institute of Technology , Lund , Sweden.

Johansson R (1986) : Identification of continuous time dynamic systems. Report TFRT-7318 , Department of Automatic Control , Lund Institute of Technology , Lund , Sweden.

Söderström T (1984) : Lecture Notes In Identification. UPTEC 8468 R , Automatic Control and System Analysis Group , Department of Technology , Uppsala University , Uppsala , Sweden.

Thompson R S , Trudinger B J and Cook C M (1985) : Doppler Ultrasound Waveforms In The Fetal Umbilical Artery : Quantitative Analysis Technique. Department of Obstetrics and Gynaecology , University of Sydney , Westmead Hospital , Weastmead , Australia.

Wirth N (1983) : Programming In Modula-2. Springer-Verlag

Young P C (1965) : Process parameter estimation and self-adaptive adaptive control. - The theory of self-adaptive control systems , Plenum Press , New York.

Young P C (1969) : Applying parameter estimation to dynamic systems , Control engineering , 16 , Oct 119-125 , Nov 118-124.

Young P C (1981) : Parameter estimation for continuous-time models : a survey , Automatica , 17 , 23-29.

Åström K J (1985) : A Simnon Tutorial. Report TFRT-3176 , Department of Automatic Control , Lund Institute of Technology , Lund , Sweden.

```
        p[0].h:=300.0;
        s[0]:=CHR(0);
        ReadString(handle,p[0],s);
        StringToReal(s,pos,lambda);
        p[0].h:=262.0;
        p[0].v:=200.0;
        Text(handle,p[0],"        ");
        RealToString(lambda,s,numchar);
        s[numchar]:=CHR(0);
        Text(handle,p[0],s)|
"B":    Text(handle,p[0],"filter constant TAU : ");
        p[0].h:=260.0;
        s[0]:=CHR(0);
        ReadString(handle,p[0],s);
        StringToReal(s,pos,tau);
        p[0].h:=222.0;
        p[0].v:=180.0;
        Text(handle,p[0],"        ");
        RealToString(tau,s,numchar);
        s[numchar]:=CHR(0);
        Text(handle,p[0],s);
        filtercoeff1:=exp(-(tsamp/tau));
        filtercoeff2:=1.0-filtercoeff1;
        tstart:=3.0*tau|
"C":    Text(handle,p[0],"A upper : ");
        p[0].h:=160.0;
        s[0]:=CHR(0);
        ReadString(handle,p[0],s);
        StringToReal(s,pos,Aupper);
        p[0].h:=126.0;
        p[0].v:=160.0;
        Text(handle,p[0],"        ");
        RealToString(Aupper,s,numchar);
        s[numchar]:=CHR(0);
        Text(handle,p[0],s);
        BlankOpline(p[0]);
        pos:=0;
        Text(handle,p[0],"A lower : ");
        p[0].h:=160.0;
        s[0]:=CHR(0);
        ReadString(handle,p[0],s);
        StringToReal(s,pos,Alower);
        p[0].h:=286.0;
        p[0].v:=160.0;
        Text(handle,p[0],"        ");
        RealToString(Alower,s,numchar);
        s[numchar]:=CHR(0);
        Text(handle,p[0],s)|
"D":    Text(handle,p[0],"B upper : ");
        p[0].h:=160.0;
        s[0]:=CHR(0);
        ReadString(handle,p[0],s);
        StringToReal(s,pos,Bupper);
        p[0].h:=126.0;
```