Combining Auto-Tuning and Adaptation

Valentin, Claire

1989

Document Version:
Publisher's PDF, also known as Version of record

CODEN: LUTFD2/(TFRT-7431)/1–76/(1989)

Combining Auto-Tuning and Adaptation

Claire Valentin

Department of Automatic Control
Lund Institute of Technology
September 1989

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

Document name
Report

Date of issue
September 1989

Document Number
CODEN: LUTFD2/(TFRT–7431)/1-76/(1989)

Author(s)
Claire Valentin

Supervisor
Michael Lundh

Sponsoring organisation

Title and subtitle
Combining Auto-Tuning and Adaptation

Abstract

An adaptive controller needs a good initialization to converge quickly and without a too bad transient. This project develops an auto-tuning method based on the relay experiment initializing a RST controller designed with pole placement rules. This method is implemented and tested on different processes to study the limits of the algorithms.

Key words
Auto-Tuning, Relay-experiment, Pole placement design, Adaptive controller, Initialization

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
76

Recipient's notes

Security classification

ENSIEG
DEA Project in Automatic Control

Combining Auto-tuning and Adaptation

Claire Valentin

Supervisor: Michael Lundh

Department of Automatic Control
Lund Institute of Technology, Sweden

September 1989

Résumé

Un correcteur adaptatif doit être bien initialisé si l'on veut qu'il converge vite et sans un transitoire trop perturbé. Ce projet consiste en l'étude d'une méthode d'autoréglage par bouclage du procédé sur un relais initialisant un correcteur RST conçu avec la méthode du placement de pôles, en son implémentation et en l'étude de ses limites à partir d'expériences sur differents procédés.

Mots clé: autoréglage - placement de pôles - relais - correcteur adaptatif - initialisation

Abstract

An adaptive controller needs a good intialization to converge quickly and without a too bad transient. This project develops an auto-tuning method based on the relay experiment initializing a RST controller designed with pole placement rules. This method is implemented and tested on different processes to study the limits of the algorithms.

Key words: auto-tuning - relay experiment - pole placement design - adaptive controller - initialization.

Brief presentation of the Department of Automatic Control in Lund

Research at the Department concerns the following areas

- Adaptive control
- Computer aided control engineering
- Expert control
- Robotic and sensory control
- Power systems
- Control of biotechnology processes
- Modeling and control of medical systems

Also, different courses dealing with these areas of research are given each year at the Department. They are of various levels: Undergraduate courses (same level as in a french engineer school), Graduate courses (for PhD students), and even external courses for engineers in the industry.

About 30 persons work at the Department. Among them, two Professors, Associate Professors, Research engineers, Research assistant and Teaching assistant (Mostly PhD students) and secretaries. Karl Johan Åström was awarded the degree Doctor Honoris Causa from l'Institut National Polytechnique de Grenoble in 1987

It is a tradition for the Department to welcome guests from all over the world to make seminars or to work some time on a subject of common interest with researchers from here.

The main computer facilities are a Sun Workstation network, a VAX-11/780 and some IBM-AT

Contents

Preface

This project has been carried out in the Department of Automatic Control, Lund Institute of Technology, in Sweden.The purpose is to create an auto-tuned pole placement controller with good performances for a certain class of processes, among which are the stable systems with monotone step response. The initialization of the model, of the controller and of the specifications for the closed loop system are particularly studied. The operator interface allows a check of most parameters involved in the control,what makes experiments quite easy to carry out. This report is primarily written for a reader with knowledge of automatic control at the end of the engineering studies.

Acknowledgement

I would like to thank all the researchers of the Department of Automatic Control in Lund, for the time they have spent in discussions with me and for their valuable criticism on the report

The ideas and the theories used in the work about auto-tuning have been developed by Tore Hägglund and Karl Johan Åström. The implementation of the estimator and of the pole placement design has been done by Michael Lundh (see Lundh, 1988). I thank them particularly for their patience in explaining to me all the details of these theories and guiding me through the project.

The graphics part and a first simple version of the auto-tuning part of the program have been done by Ulf Andersson, Ola Persson and Mårten Åkesson during a project in an implementation course (see Persson, Andersson and Åkesson, 1989).

1. Introduction

Different kinds of controllers are useful for different kinds of users and control problems. But in any case, every operator wants to have a controller that is easy to operate.

That is the reason why self tuning regulators have been created and improved for years. The goal was to design a controller as easy as possible to use. The problem is that, if you want to use a controller you have to know the right initial parameters to enter in, in order to obtain efficient control. And even for a simple regulator as the PID, you have to know how to tune three parameters before it can be used: the integral time, the derivative time and the proportional gain of the regulator. It means that a certain knowledge is required to initialize the regulator. It would be easier for any operator to push only one button so that the controller could start without any other action from him.

With this purpose, an auto-tuning PID controller has been designed by T. Hägglund and K.J. Åström from the department of Automatic control, Lund Institute of technology. It is, now, manufactured by two companies, Satt Control AB and Fisher Controls Inc., and has been used in a lot of control loops. It is based on a relay-experiment (explained in chapter 2) which allows a measurement of the ultimate period and of the ultimate gain of the process in order to calculate the controller parameters.

The goal of this project is to initialize an adaptive controller from such a relay-experiment in order to obtain a controller with higher performance than a PID controller, but which is as easy to tune as the PID controller.

It is wise to go on with one of the simpler techniques; pole placement adaptive control. The problem with adaptive control is that you have to know the sampling period and the gain of the process as with a PID controller but also you must have an idea of the achievable performance of the process and above all, of its model to initialize the controller parameters. That is why we will now try to get some more information from the relay-experiment. An idea is to analyse the shape of the oscillations under relay feedback control and to extract a valid model from it (see Åström K.J. and Hägglund T.,1988a).

The goal of this project is to develop and implement algorithms which combine auto-tuning and pole-placement adaptive control.

This report is organised as follows: a certain knowledge is required about the Automatic tuning of simples regulators and the Pole-placement design which are described in chapters 2 and 3. Then it is discussed on the right way to combine Automatic tuning and Pole-placement design in chapter 4. Chapter 5 explains how to implement this control algorithms. Then chapter 6 deals with experiments on different processes. Finally, a conclusion on the project is given in chapter 7.

A regulator with automatic tuning is composed of four subsystems, an ordinary feedback regulator with adjustable parameters, an excitation generator, a parameter estimator and a block which performs design calculations (See fig. 2-1). The excitation generator provides a signal which makes it possible to estimate the parameters of the process.

The controlled system works in the following way. The process is excited from the excitation generator, then the process dynamics are estimated from the response of the process to the excitation and the regulator parameters are calculated from the dynamics. Finally, the estimator, the control design and the generator are disconnected and the system operates like a closed-loop process with regulator.

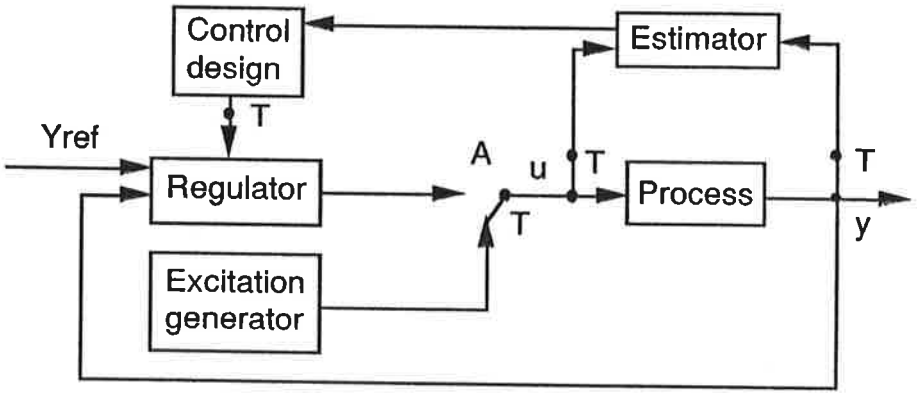

Figure 2-1: Block diagram of a regulator with automatic tuning.
T: connections on during the tune and off after.

Many schemes of this type have been proposed. The following scheme has been developed by T.Hägglund and K.J.Åström from the Department of Automatic control, Lund Institute of Technology (See K.J. Åström and T.Hägglund, 1984 and before). The characteristic feature of this approach is that it gives a very simple system which does not require prior information.

The idea is to estimate the ultimate point which is described by the ultimate gain, k_c and the ultimate frequency, ω_c. This point represents the stability boundary, that is the reason why it is so important to know it. On that point the closed loop system oscillates.

The original Ziegler-Nichols method proposes to determine these values by connecting a proportional regulator to the process in a closed loop and to increase the gain gradually until an oscillation is obtained. Then, the frequency observed is ω_c. But it is very difficult to perform this experiment automaticaly in such a way that the amplitude of the oscillation is kept under control. So, another method for automatic determination of the ultimate point is proposed.

The method is based on the observation that many systems may oscillate at frequency ω_c under relay control. It means that many processes will have limit cycle oscillations under relay feedback The scheme is the same as the one of figure 2-1 if the regulator is replaced by a PID regulator and the excitation generator by a relay.(See fig 2-2)

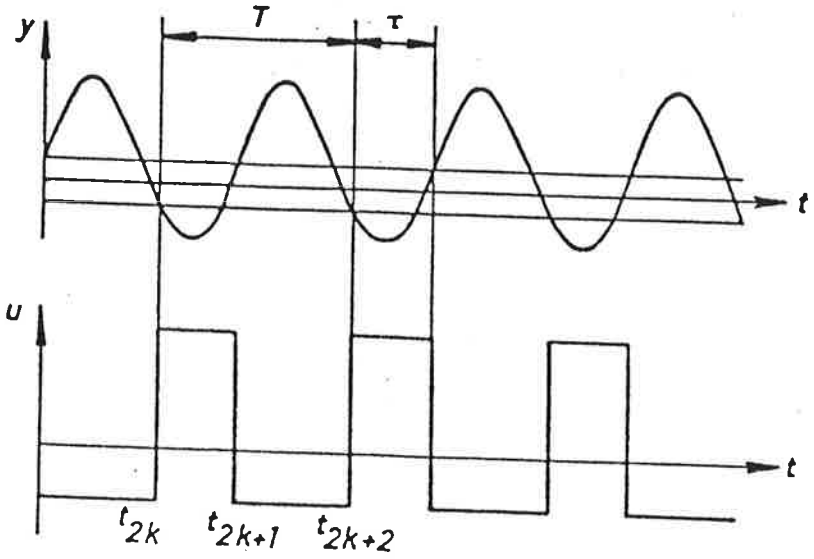

Figure 2-4: Relay output u and process output y for a system under relay feedback.

The period of the oscillation can easily be determined by measuring the times between zero-crossings. The amplitude may be determined by measuring the peak-to-peak values. This estimation method is very easy to implement because it is based on counting and comparisons only. More elaborate estimation schemes may also be used to determine the amplitude and the period of the oscillation.

Then the parameters of the PID regulator are calculated. The Ziegler-Nichols frequency response method, for example, gives simple formulas for the parameters of the regulator in terms of ultimate gain and the ultimate period (See Table 2-1).Other better types of design methods have been used too.

Controller	K	Ti	Td
P	0.5 k_c		
PI	0.4 k_c	0.8 t_c	
PID	0.6 k_c	0.5 t_c	0.12 t_c

Table 2-1: Recommended PID parameters according to Ziegler-Nichols frequency response method.

When the PID parameters are determined the PID regulator is activated.

Several improvements were made in the relay experiment: The amplitude of the oscillations obtained under relay control can be specified by adjusting the relay amplitude. A hysteresis in the relay is also useful to make the system

3. Adaptive control based on Pole placement design

3-1. Introduction

Pole placement is one of the simpler indirect design methods. It allows a control of stable or unstable systems with time delay. The techniques presented in this chapter are known by everybody dealing with adaptive control. Therefore, the purpose is not to present a deep work about pole placement design, but to settle on the different notations used in the project.

Only the procedure used for this project will be described in section 3-2. The real-time estimator is also a very important part in the adaptive controller. In section 3-3, the recursive least square estimator will be presented.Finally, the estimator and the design are combined to obtain the adaptive controller in section 3-3.

3-2. Pole placement design

The goal is to obtain the appropriate response of the process to command inputs by a simple and fast method.

Consider a process with one input u and one measured output y. v is a disturbance. The process to be controlled can be described by

$$A\,y = B\,u + v \tag{1}$$

where A and B are polynomials. It is assumed that A and B do not have any common factors i.e. that they are relatively prime. It is also assumed that A is monic, i.e., that the coefficient of the highest power in A is unity. The desired response from the reference signal u_c to the output is given by

$$A_m\,y_m = B_m\,u_c \tag{2}$$

where A_m and B_m do not have any common factors.

To get a realizable controller the model (2) must have the same or higher pole excess than the process model (1). This gives the condition

$$\deg A_m - \deg B_m \geq \deg A - \deg B \tag{3}$$

The pole placement regulator can be described by

$$R\,u = T\,u_c - S\,y \tag{4}$$

where

$$AR + BS = A_0 A_m B^+$$

It follows from this that B^+ divides R. Moreover, to make sure that low-frequency disturbances give small errors, the loop gain, H_{lg} (7), must be large for low frequencies. This may be achieved by requiring that R has the following form

$$H_{lg} = \frac{B S}{A R} \tag{7}$$

$$R(z) = (z - 1)^r R_1(z) B^+(z)$$

With a suitable r. This is a classical principle of integral control.

Therefore, it gives

$$(z - 1)^r A R_1 + B^- S = A_0 A_m \tag{8}$$

Equation (8) has a solution if A and B^- are relatively prime. It follows from equation(6) that B^- must divide B_m. It gives

$$B_m = B^- B_m^'$$

$$T = A_0 B_m^' \tag{9}$$

The pole placement design procedure can be summarized in algorithm 3-1.

Algorithm 3-1:

Data: polynomials A, B.
Specifications: Polynomials A_m, B_m, A_0.

Compatibility conditions:

B^- divides B_m.

$\deg A_m - \deg B_m \geq \deg A - \deg B$

$\deg A_0 \geq 2\deg A - \deg A_m - \deg B^+ + r - 1$

Step 1: Factor B as $B = B^+ B^-$

Step 2: Solve the equation. $(z - 1)^r A R_1 + B^- S = A_0 A_m$ with respect to R_1 and S. Choose a solution such that

$$\begin{cases} \deg R_1 = \deg A_0 + \deg A_m - \deg A - r \\ \deg S \leq \deg A + r \end{cases}$$

Step 3: The control law is then given by $R u = T u_c - S y$ where

An interpretation of these equations is given in figure 3-2.

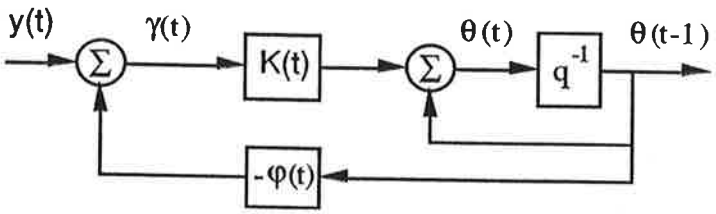

Figure 3-2: Recursive Least Square estimation diagram

Let define the prediction error

$$\gamma(t) = y(t) - \theta^{T}(t-1)\,\varphi(t)$$

Then, the estimation method can be very intuitive: $\theta(t)$ is corrected with respect to the gain $K(t)$ when the prediction error $\gamma(t)$ is different from zero.

3-4. Global scheme of the indirect adaptive regulator

The indirect adaptive regulator based on the pole placement design and on the recursive least square estimator can be expressed as the following algorithm.

Algorithm 3-2.

Data: Desired close loop transfer function: $\frac{B_m}{A_m}$

Desired observer polynomial: A_0

Step1: Estimate the coefficients of the polynomials A and B recursively using the least square method described above

Step 2: Replace A and B with the estimates obtained in step 1 and use the pole placement algorithm to find the controller polynomials R, S and T.

Step 3: Calculate the control signal u from

$$R\,u = T\,u_c - S\,y$$

Repeat step1, 2 and 3 each sampling period. ◊

This algorithm is illustrated in figure 3-3.

4. Combining auto-tuning and pole placement design

4-1. Introduction

In the two last chapters, an adaptive design method and an automatic tuning method have been presented. They both work on different processes.

The idea is to make a deeper analysis of the information given during the relay experiment described in chapter 2. Instead of only using the amplitude and the period of the oscillation, the shape of the oscillation under relay feedback can be analysed, (See Åström K.J. and Hägglund T., 1988a) and a valid model of the process can be estimated from this analysis. This model can thus be a good initial model for the pole placement control design and allows the calculation of good initial parameters for the controller

It can be expected to outperform conventional PID designs because the response speed is easily adjusted and it can also handle systems with time-delay.

The method is explained in section 4-2 and practical aspects are described in section 4-3.

4-2. Analysis of the relay experiment and control design

Chapter 2. has described how to find the period and the amplitude of the oscillations created under relay feedback. These measurements give the sampling rates and an indication of the achievable bandwidth.

In this section, it will be shown that conventional sampled data models can be determined using the wave-form of the oscillation. A measurement of some values of the process output will be computed in order to obtain this model. The following method will be used for the computation.

Parameter estimation

Remember that a relay feedback experiment in stationarity gives periodic input output signals for the process as shown in figure 4-1.

$$U(z) = -\frac{z^n + z^{n-1} + \ldots + z}{z^n + 1} = -\frac{E(z)}{z^n + 1} \tag{4}$$

$$Y(z) = \frac{y_{d+1} z^n + y_{d+2} z^{n-1} + \ldots + y_{d+n} z}{z^d (z^n + 1)} = \frac{D(z)}{z^d (z^n + 1)} \tag{5}$$

Then, the z-transform of equation (3) gives

$$Y(z) = \frac{B(z)}{A(z)} U(z) + \frac{Q(z)}{A(z)}$$

where the polynomial Q(z) corresponds to initial conditions which gives the steady state periodic output. Replacing Y(z) and U(z) by the expressions (4) and (5), it follows

$$\frac{D(z)}{z^d (z^n + 1)} = -\frac{B(z)\, E(z)}{(z^n + 1)\, A(z)} + \frac{Q(z)}{A(z)}$$

and

$$A(z)\, D(z) + z^d B(z)\, E(z) = z^d (z^n + 1)\, Q(z) \tag{6}$$

where D(z) and E(z) are known. It is thus possible to determine the polynomials A, B and Q from the (n + deg A) linear equations obtained from (6). The number of unknown parameters in the polynomials A, B and Q is (deg A + 2deg B + 1) i.e. (3deg A - 2d +1). It gives a condition for n; indeed to determine all the parameters of A, B and Q, it is necessary to have

$$n \geq 2\,(\deg A - d) + 1 \tag{7}$$

It is thus straightforward to determine the coefficients of the process model (3) from the wave-form $y_0, y_1, \ldots, y_{n-1}$ of the periodic solution. A fixed structure and a time-delay must be chosen for the process model in that method

The procedure is illustrated by one exemple where A and B are first order polynomials and the time-delay is r sampling periods.

Example 1

Consider the process model

$$y(t+1) = a\, y(t) + b_1\, u(t\text{-}rh) + b_2\, u(t\text{-}rh\text{-}1) \tag{8}$$

Where h is the sampling period. Therefore

$A = z^{r+1}(z - a)$

$B = b_1 z + b_2$ and $d = r + 1$

The model has three parameters. It gives $n \geq 3$. It is simpler to choose $n = 3$, because there is the same number of parameters and equations.

The problem is thus solved with this value.The expression (6) becomes

oscillating signal. Let t_{max} be the distance from the extremum of the output to the previous switch of the input. The value of r should then be chosen so that

$$r \geq \frac{t_{max}}{h}$$

where h is the sampling period.

It must also be noticed that the output of the system is close to sinusoidal for certain processes. It is thus not possible to determine more than two parameters in the model when three amplitude values are used.

As it was said in chapter 2, it is useful to introduce hysteresis in the relay to avoid a too important sensibility to noise. The hysteresis amplitude can be determinated by measuring the noise level in steady state. The relay amplitude can also be adjusted to avoid too large perturbations on the output signal during the relay experiment.

4-4. Conclusion

This chapter has determinated a simple method for estimating a model from a relay experiment. The information about the full wave-form is used, and not only amplitude and frequency. Moreover the algorithm can cope with systems having time-delays.

The theoretical bases are given in this chapter and the practical problems will be treated in next chapter dealing with implementation.

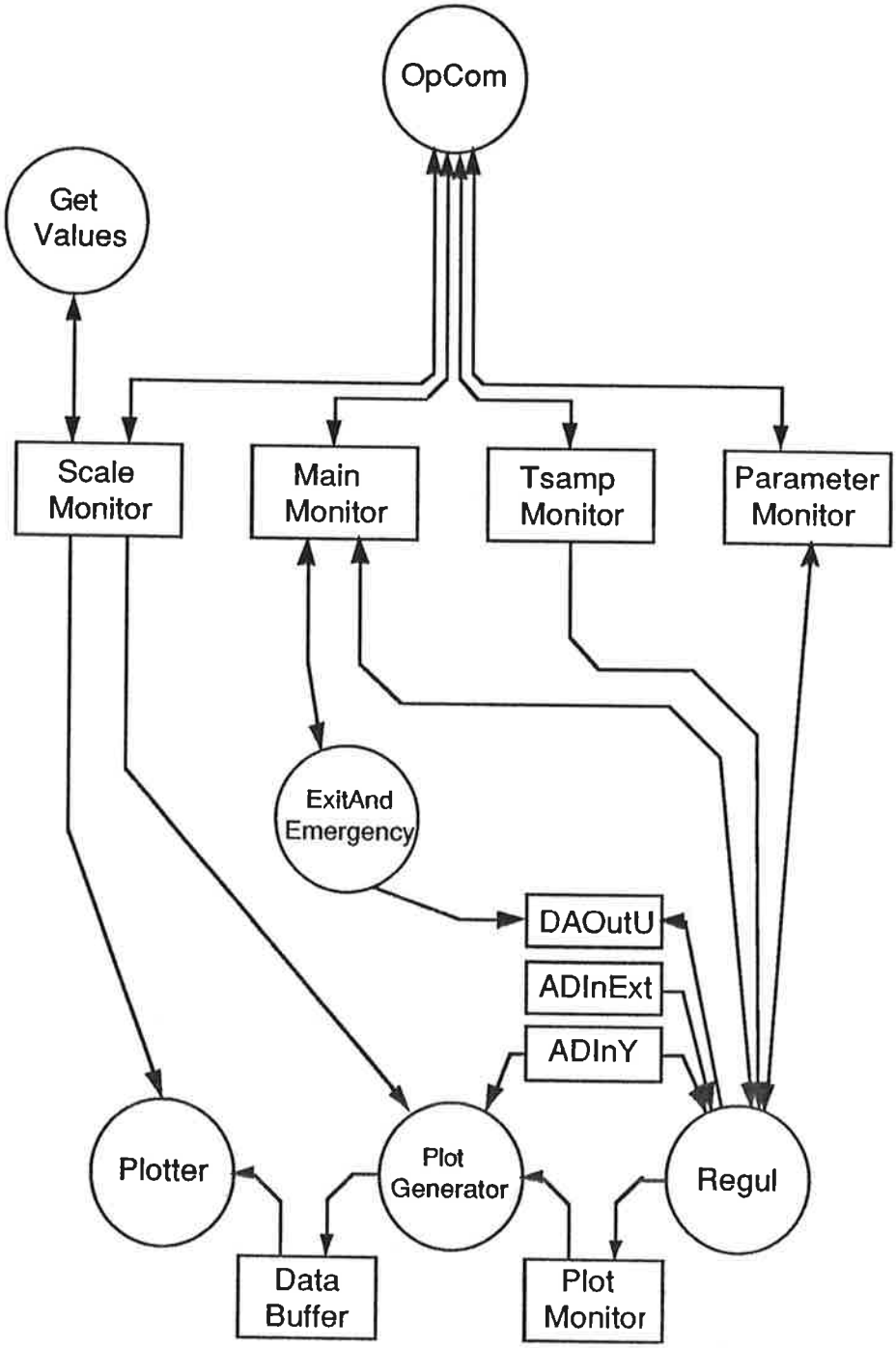

Figure 5-1: Real-time implementation structure.
6 processes and 9 monitors share the execution time.

DataBuffer : The plotter is a process with low priority which must get the signals U, Uc and Y when it has time to plot them. These signals are put in a RingBuffer while waiting to be plotted.

ScaleMonitor : The time scale of the plotting is stored in this monitor.

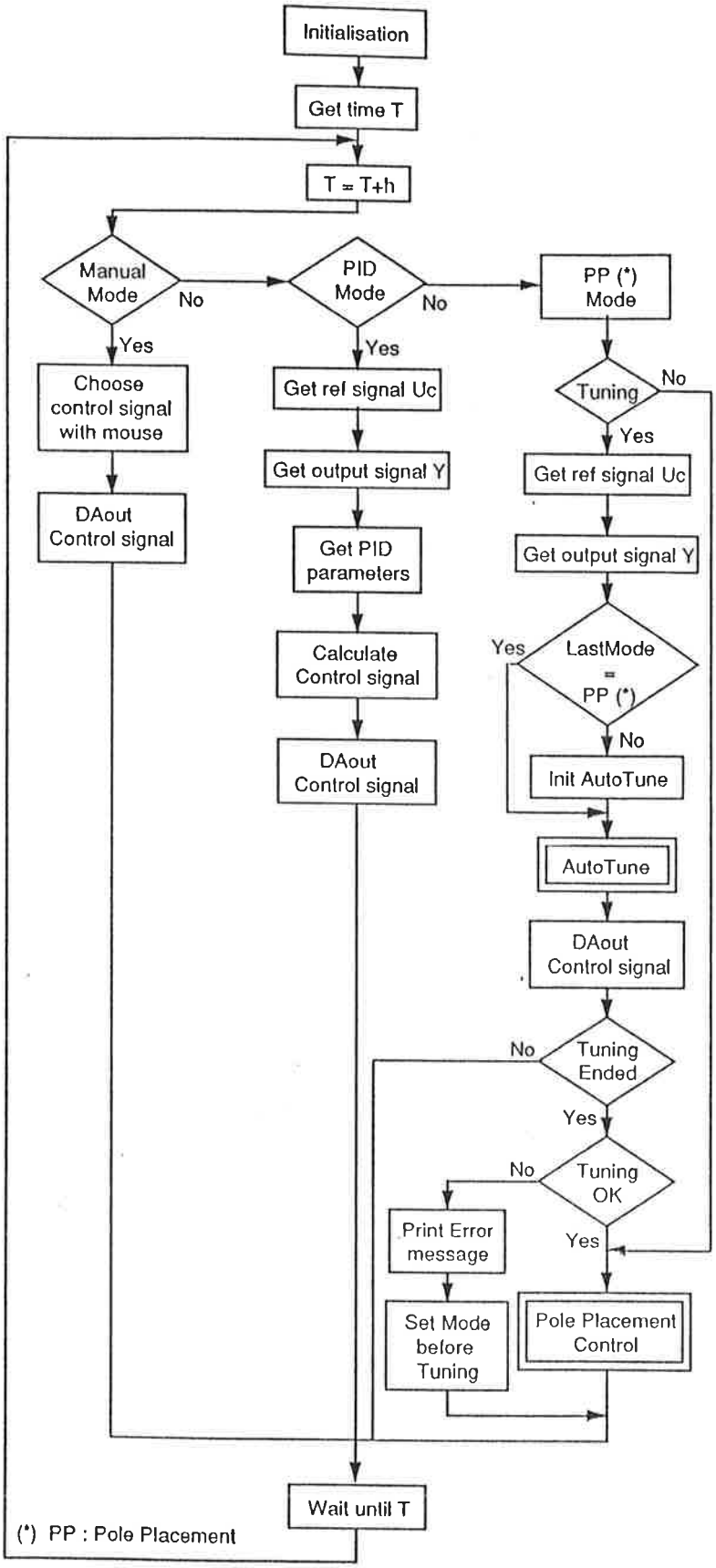

Figure 5-2: Process Regul (Priority 20)

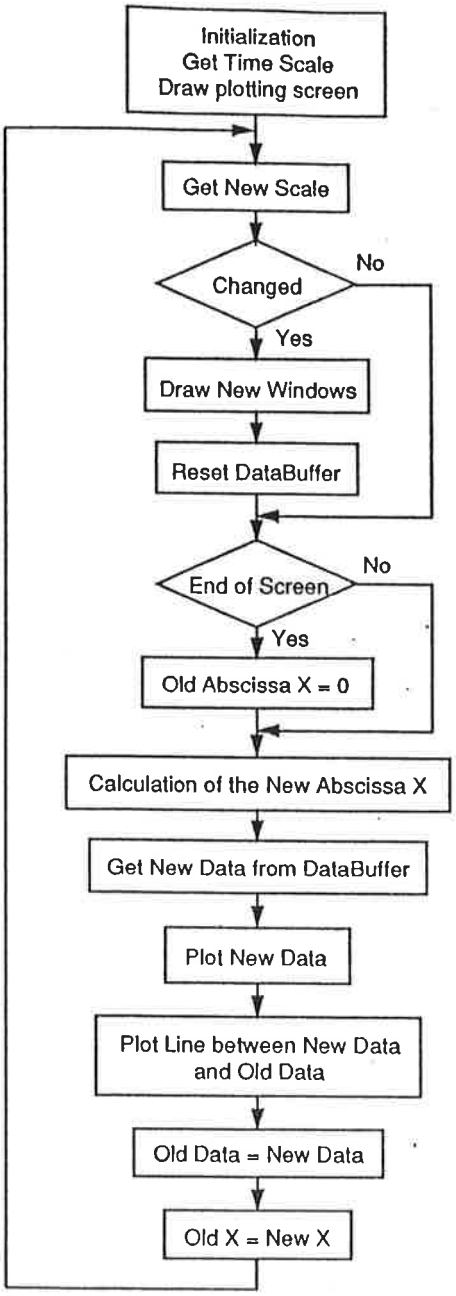

Figure 5-6: Process Plotter (Priority 50)

Presentation of the Man-Machine interface

Figure 5-7 describes the screen after start-up. On the upper half of the screen are the measurement signal Y, the reference signal Uc and the control signal U. Below, are a number of mouse sensitive buttons, whose functions are described below.

The program can handle three different modes; Man (manual), PID (control with PID regulator) or Pole Placement Control (Automatic tuning of the regulator parameters according to pole placement design). Changing of mode is done by clicking one of the squares 1, 4, or 7.

Only the most important modules dealing with theory (Auto-tuning, control) and display of variables are given (see appendix D).

The Auto-tuning part

The tune is divided in three different parts: The first one measures the noise level on the output signal in order to choose the hysteresis of the relay. The relay is then included in the closed loop in the second part, and the amplitude of its output is fixed after an analyse of the amplitude of the process output. Indeed, the tune must not disturb the process too much. In the third part, the measurement of the amplitude, of the period of the oscillation and of certain values of the process output are computed. These measurements lead to a first estimation of the process by a first order model with time delay (choice explained in 4-3), then the design specifications (second order desired dynamics, reasonably fast and with a damping of 0.7) and the parameters of the RST controller are initialized.

Directions for use

At the starting point, the mode Manual is running and the control signal ,u, is equal to zero. Then the sampling period can be choosen by clicking in the box 2, according to the time constant of the process. Then, to start the controller based on pole placement design, the reference signal amplitude must be close to the process output amplitude (Bad stationarity can imply asymetric relay oscillations). Afterwards,.you just need to click on the square 7. At the end of the relay experiment, a fixed controller is started. To make it adaptive, you can click on the square 9, type 5(Adapt)-ENTER-1 then you type -e- to exit (It is made adaptive only after exit). To get a fixed regulator again you use the same command in replacing the 1 by 0. The model and the controller can be read in the square 6 (but you can not change them by the keyboard of course). The design specifications initialized by the relay experiment can be read by clicking in the square 8 and changed by the keyboard (do not forget that they are in discrete time). The estimation specifications can be read and changed as well by clicking in square 9. Finally, the values measured in the relay experiment can be checked by clicking in square 3.

5-3 Conclusion

This chapter showed how the algorithm developed in chapter 4 has been implemented. It was tried to make a procedure for each simple task in order to decompose the complicated parts of the algorithm in understandable routines. The operator interface has not been improved as much as it could have been in order to spend more time studying the auto-tuning and the control theories involved in this controller. In appendix C, the most important routines used in the programming are described and their code is given in appendix D.

$$G(s) = \frac{1}{s+1} \tag{2}$$

Figure 6-1 shows that the relay experiment is quite symetric and that the output is quite regular. Figure 6-2 shows that the step reponse of the process is of expected type (fast enough with damping of 0.7). Big variations on the control signal are required to obtain that performance. Then, when the adaptive controller is started with these parameters for the model and the controller, there is no bad transient neither on the output signal nor on the control signal and the estimator converges, after a while, to a more accurate model that the one initialized from the relay experiment (see table 1 in appendix C). Therefore, the goal is reached.

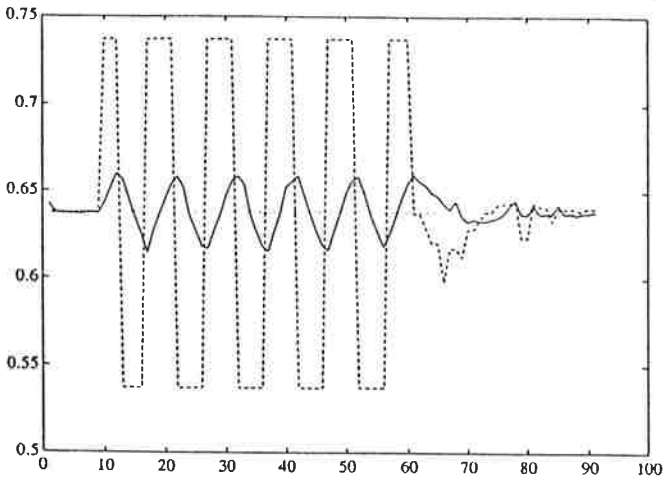

Figure 6-1: relay experiment with a first order process of type (2) without noise

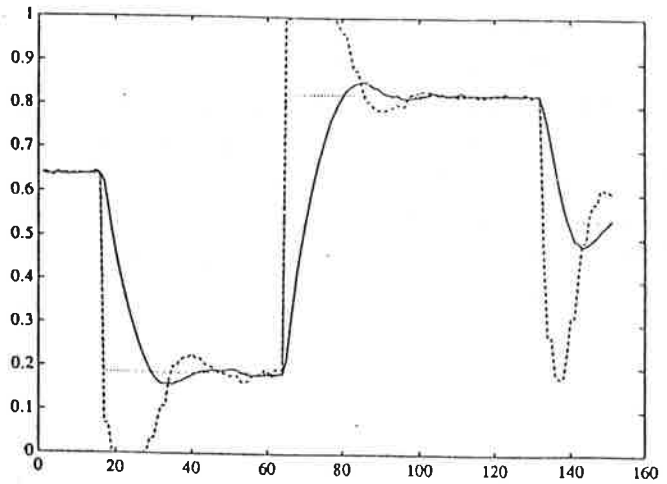

Figure 6-2: step response with a first order process of type (2) in close loop with the fixed controller initialized after the relay experiment.

6-3 Further developments

A first order process is controlled better than with a PID. But a second order process (of the kind choosen in experiment 4 and 5) is quite difficult to initialize with a relay experiment especially if there is no noise on the output. Also, with processes of higher order than one, the choice of the delay seems to be of highest importance for the control. A fixed controller can generally not be calculated from the relay experiment initializing a model of the kind (1), but, in some cases, an adaptive controller can be started from this initialization.

The method is also quite sensitive to the value of the sampling period choosen at the beginning. It must be short enough to allow the measurement of several points during one period of the relay experiment. A too big sampling period will not make the system oscillate and a too small sampling period will give a pole close to one for each process and lead to a very bad control.

The initialization performed from the relay experiment can be improved with a test determining the structure of the model to be choosen so that processes with higher order than one can be modelized by a second order or by the product of a second order by a first order.

The output could also be filtered during the relay experiment to avoid the perturbations created by the noise on the measurements.

The initialization of the desired closed loop dynamics could be also improved by extensive simulations. This choice is however based on heuristic knowledge. The observer polynomial is also fixed to a dead beat polynomial , this could be improved in choosing a first order polynomial with known dynamics.

6-4 Conclusion

A first version of an auto-tuned adaptive controller has been realized in this project. It allows a good initialization of an adaptive controller based on pole placement design for first order processes with time delay and the controller has good performances. For processes with higher order which are also stable with monotone step response the performances and even the success of the initialization depend a lot on the noise level and on the relative location of the poles of the process.

The program could be improved with intensive experiments which could allow a better choice of the empirical values. On the other hand, this tool must not be designed for a too small class of processes, and even if it does not control each particular process belonging to a certain class, with optimal performances, the purpose is that it gives average performances higher than with a PID regulator for a class as large as possible.

8. References

Analog Devices, 1986. *RTI 800/815 User's manual* and *RTI 802 User's manual.*

Andersson, L., 1986. Kursmaterial. *"Tillämpad realtidsprogrammering"*. Institutionen för Reglerteknik, Lund.(Librairy modules for real time programming).

Årzén, K.E., 1987. *Realization of Expert System Based Feedback Control.* PhD thesis held at the Department of Automatic Control, in Lund.

Åström, K.J., and Hägglund, T., 1984a. "Automatic tuning of simple regulators", *Proceedings of the IFAC 9th World Congress, Budapest, Hungary*

Åström, K.J., and Hägglund, T., 1984b. "Automatic tuning of simple regulators with specifications on phase and amplitude margins." *Automatica* **20**: 645–651.

Åström, K.J., and Wittenmark, B.,1984. *Computer Controlled Systems*. Prentice-Hall,Englewood Cliffs.

Åström, K.J., and Hägglund, T., 1988a. "A New Auto-tuning Design". *Proceedings of the IFAC international Symposium on Adaptiv Control of Chemical processes, Copenhagen.*

Åström, K.J., and Hägglund, T., 1988b. *Automatic Tuning of PID controllers.* Research Triangle Park, N.C.: Instrument Society of America.

Åström,K.J., 1989. *Assessement of achievable performance of simple feedback loops*. Internal report number 7411 of the Department of Automatic Control, in Lund

Åström, K.J., and Wittenmark, B.,1989. *Adaptive control.* Addison-wesley publishing company.

Hamel, B., 1949. "Contribution a l'étude mathématique des systemes de réglage par tout-ou-rien.".*C.E.M.V. Service Technique Aeronautique* **17**.

Lundh, M., 1988. *A TOOLBOX for Discrete Time Design and On-line Control.* Internal report number 7382 of the Department of Automatic Control, in Lund.

Martin-Sanchez, J.M. and Dumont, G.A., 1985. "Industrial Comparison of an Auto-Tuned PID regulator and a Adaptive Predictive Control System(APCS)." *Proceedings of the IFAC Workshop on Adaptive Control of Chemical Processes, Frankfurt/Main,* Pergamon Press.

Persson, O. and Andersson, U. and Åkesson, M., 1989. *Automatinställning med parameterstyrning*. Internal report of the Department of Automatic Control, in Lund.

Tsypkin, J.A., 1958. *Theorie der Relais Systeme der automatischen Regelung*. R.Oldenburg, Munich.

Wirth, N.,1988. *Programming in Modula-2*. Springer-Verlag.

9. Appendix

A. About the relay experiment

Certain conditions must be fulfilled so that the process oscillates under relay feedback. In this section, the conditions for an oscillation to occur and the period of this oscillation will be studied. Therefore a linear system under relay control has to be investigated.

There are two means to analyse the conditions for existence of a periodic solution. First a linear system under relay control will be investigated and then the describing function analysis will give a good approximation of the answer by a simpler condition.

First method: investigation of the closed-loop nonlinear system

Consider a system described by

$$\begin{cases} \dfrac{dx}{dt} = Ax + Bu \\ y = Cx \end{cases} \tag{1}$$

This system is controlled by a relay with hysteresis of the following form

$$u(t) = \begin{cases} d_1 \text{ if } e > \varepsilon \text{ or } (e > -\varepsilon \text{ and } u(t-) = d_1) \\ -d_2 \text{ if } e < \varepsilon \text{ or } (e < \varepsilon \text{ and } u(t-) = d_2) \end{cases} \tag{2}$$

where $e = -y.$, ε is the hysteresis of the relay and u is the controller output.

Conditions for relay oscillations have been given by Hamel (1949) and Tsypkin (1958). The key result is given by the following theorem.

Theorem A-1: Consider the system (1) with the feedback law (2). Assume that the matrix $\Phi - I$ is regular. A necessary condition for a limit cycle with period T is then

$$\begin{cases} C[I-\Phi]^{-1}[\Phi_2\Gamma_1 d_1 - \Gamma_2 d_2] = -\varepsilon \\ C[I-\Phi]^{-1}[-\Phi_1\Gamma_2 d_2 + \Gamma_1 d_1] = \varepsilon \end{cases} \tag{3}$$

where

τ is defined in figure 2-4, and

$$\Phi = e^{AT} \qquad \Phi_1 = e^{A\tau} \qquad \Phi_2 = e^{A(T-\tau)}$$

$$\Gamma_1 = \int_0^{\tau} e^{As}ds\, B \qquad \Gamma_2 = \int_0^{T-\tau} e^{As}ds\, B \tag{4}$$

discrete time system. (The stroboscopic transformation) The inputs are $u(t_{2k})$ and $u(t_{2k+1})$ and the outputs are $y(t_{2k+1})$ and $y(t_{2k+2})$.

$$z_k = \begin{bmatrix} x(t_{2k-1}) \\ x(t_{2k}) \end{bmatrix} \qquad u_k = \begin{bmatrix} u(t_{2k}) \\ u(t_{2k+1}) \end{bmatrix} \qquad y_k = \begin{vmatrix} C & 0 \\ 0 & C \end{vmatrix} z_k$$

The discrete time state equations of system (1) are

$$\begin{cases} x(t_{2k+1}) = \Phi_1 x(t_{2k}) + \Gamma_1 u(t_{2k}) \\ y(t_{2k+1}) = Cx(t_{2k+1}) \end{cases}$$

$$\begin{cases} x(t_{2k+2}) = \Phi_2\, x(t_{2k+1}) + \Gamma_2\, u(t_{2k+1}) \\ y(t_{2k+2}) = C\, x(t_{2k+2}) \end{cases} \qquad (5)$$

where the matrix $\Phi 1$, $\Phi 2$, $\Gamma 1$ and $\Gamma 2$ are given by (4).
Equation (5) can then be written as

$$z_{k+1} = \begin{bmatrix} \Phi & 0 \\ 0 & \Phi \end{bmatrix} z_k + \begin{bmatrix} \Gamma_1 & \Phi_1\Gamma_2 \\ \Phi_2\Gamma_1 & \Gamma_2 \end{bmatrix} u_k$$

$$y_k = \begin{bmatrix} C & 0 \\ 0 & C \end{bmatrix} z_k$$

This is a time invariant discrete time system. Let the pulse transfer function of the system be

$$H(z) = \begin{bmatrix} C\,[zI-\Phi]^{-1}\,\Gamma_1 & C\,[zI-\Phi]^{-1}\,\Phi_1\Gamma_2 \\ C\,[zI-\Phi]^{-1}\,\Phi_2\Gamma_1 & C\,[zI-\Phi]^{-1}\,\Gamma_2 \end{bmatrix} \qquad (6)$$

Putting $z = 1$ in (6), it follows that the condition (3) can be written as

$$H(1) \cdot \begin{bmatrix} d_1 \\ -d_2 \end{bmatrix} = \begin{bmatrix} \varepsilon \\ -\varepsilon \end{bmatrix}$$

Symetric oscillations

The case $d1 = d2 = d$ is of particuliar interest. It follows from (4) that

$$\Gamma_1 = \Gamma_2 = \Gamma \qquad \text{and} \qquad \Phi_1 = \Phi_2 = \Phi^{1/2}$$

In this case. Equation (3) then reduces to

$$C\left[I - \Phi\right]^{-1}\left[I - \Phi^{1/2}\right]\Gamma d = \varepsilon$$

or

But the conditions for existence of oscillations can be investigated by a simpler method which is an approximation of the exact formula given before.

Second method: The describing function analysis

This method is used to analyse systems with a nonlinear function in a control loop. To determine conditions for oscillation, the nonlinear block is described by a gain, N(a), which depends on signal amplitude,a, at the input of the nonlinearity. This gain is called the describing function. If the process has the transfer function G(iω), the condition for oscillation is simply given by

$$N(a)\, G(i\omega) = -1$$

This equation is obtained by requiring that a sine wave with frequency ω should propagate around the feedback loop with the same amplitude and phase. Since N and G may be complex numbers, this gives two equations for determining a and ω. The equations can be solved graphically by plotting -1/N(a) in the Nyquist diagram. If the negative inverse of the describing function is drawn in the complex plane (See fig A-1) together with the Nyquist curve of the linear system, an oscillation may occur if there is an intersection between the two curves.

Notice that the input U to G(s) is a square wave. In the describing function technique it is assumed that the input to the relay with hysteresis is a sine wave (Approximation of the input by its first harmonic). Therefore, it must be assumed that G(s) has a low-pass filter action, so that the amplitudes of the high frequencies in Y are small compared to the amplitude of the fundamental frequency. But, this is not a restrictive assumption, since almost all practical processes are of low-pass type.

The amplitude and the frequency of the oscillation are the same as the parameters of the two curves at the intersection point. Therefore, measuring the amplitude and the period of the oscillation, the position of one point of the Nyquist curve can be determinated.

The describing function, N(a), for the relay is given by

$$N(a) = \frac{4d}{\pi a}$$

Since this function is real, the oscillation may occur where the Nyquist curve intersects the negative real axis. Thus, the conclusion is that the ultimate point is conveniently determined by a relay feedback experiment.

To make the system less sensitive to noise a relay with hysteresis can be used. The negative reciprocal of such a relay is

The continuous time transfer function of the process, G(s), has to be sampled. In that case, a zero-order-hold is perfect to connect the continuous system to the discrete input (square waves sampled at h = T/2) (See fig. A-3).

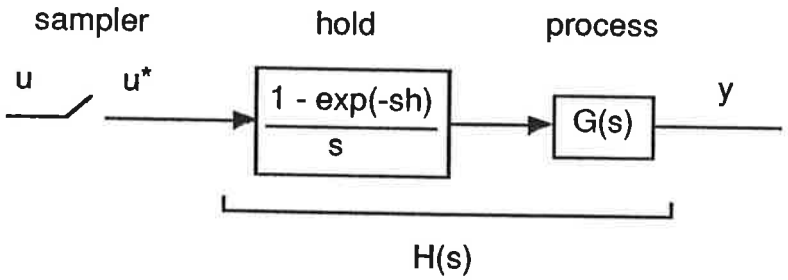

Figure A-3: Schematic diagram of the idealized model of a sample and hold circuit connected to a linear system.

The following theorem (in Åström, K.J., and Wittenmark, B.,1984) will be used to determine the sampled process description with zero-order-hold.

Theorem A-3: Let the function f have the Laplace transform F and the z-transform F',and let F* be the Laplace transform of the sampled representation f^* of f. Assume that for some $\varepsilon > 0$, $|F(s)| \leq |s|^{-1-\varepsilon}$ for large $|s|$ then

$$F^*(s) = F'(e^{sh}) = \frac{1}{h} \sum_{k=-\infty}^{\infty} F(s + ik\omega_s) \tag{9}$$

Where $\omega_S = 2\pi/h$ is the sampling frequency. ◊

Proof:

The definition of F* gives

$$F^*(s) = \int_0^{\infty} e^{-st} f^*(t)\, dt$$

$$F^*(s) = \int_0^{\infty} e^{-st} f(t) \left\{ \sum_{k=-\infty}^{\infty} \delta(t-kh) \right\} dt$$

$$F^*(s) = \int_0^{\infty} e^{-st} f(t) m(t)\, dt$$

Interchange the order of integration and the summation gives

$$F^*(s) = \sum_{-\infty}^{\infty} \int_0^{\infty} e^{-st} f(t) \delta(t-kh)\, dt$$

$$H(h,-1) = \sum_{-\infty}^{\infty} \frac{2}{i\pi(1+2n)} G(\frac{i\pi}{h}(1+2n))$$

$$H(h,-1) = \sum_{0}^{\infty} \frac{4}{i\pi(1+2n)} G(\frac{i\pi}{h}(1+2n))$$

Remember that h = T/2, so, if H(h,-1) is approximated by the first term of the series expansion the condition (8) becomes

$$\frac{4}{\pi} \text{Im } G(\frac{i2\pi}{T}) = -\frac{\varepsilon}{d}$$

which corresponds to the describing function analysis

$$\text{Im } G(\frac{i2\pi}{T}) = -\frac{\pi \quad \varepsilon}{4d}$$

This second condition is much more simple than the one obtained with theorem 2-1.

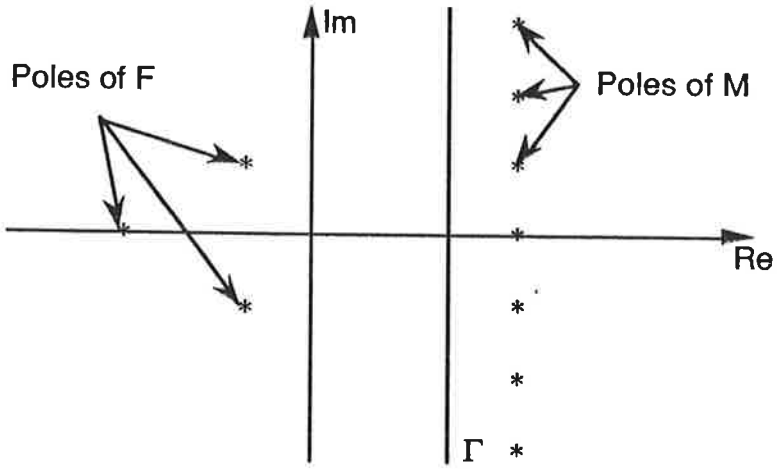

Figure A-4: Singularities of F and M and the integration contour.

Remark A-3: The Nyquist curve of a first order process, situated entirely in the right part of the Nyquist diagram will never intersects the negative inverse of the describing function, whereas a first order system oscillates under relay feedback control (That can be checked by the first method explained in this appendix).

In K.J.Åström and T.Hägglund (1984a) the conditions to get stable oscillations are discussed.

Recall about the variables used in the routines: **d** is the amplitude of the relay, **Amp** is the amplitude of the output oscillations and **Ustat** is the steady state value of U during the relay experiment

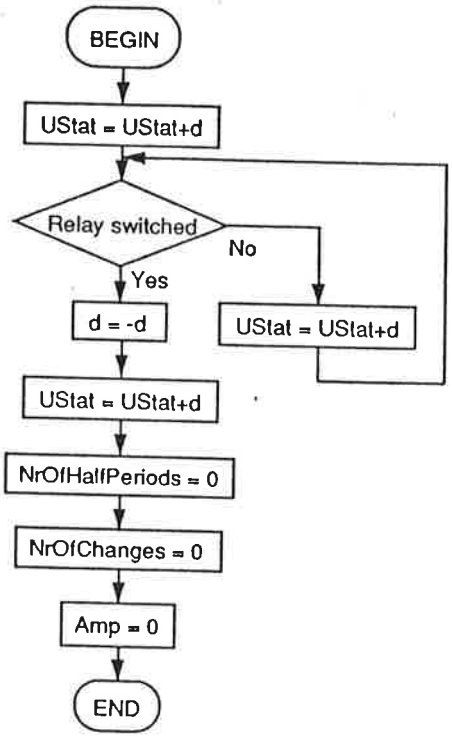

Figure B-2: routine from AutoTune: RelayStarted.

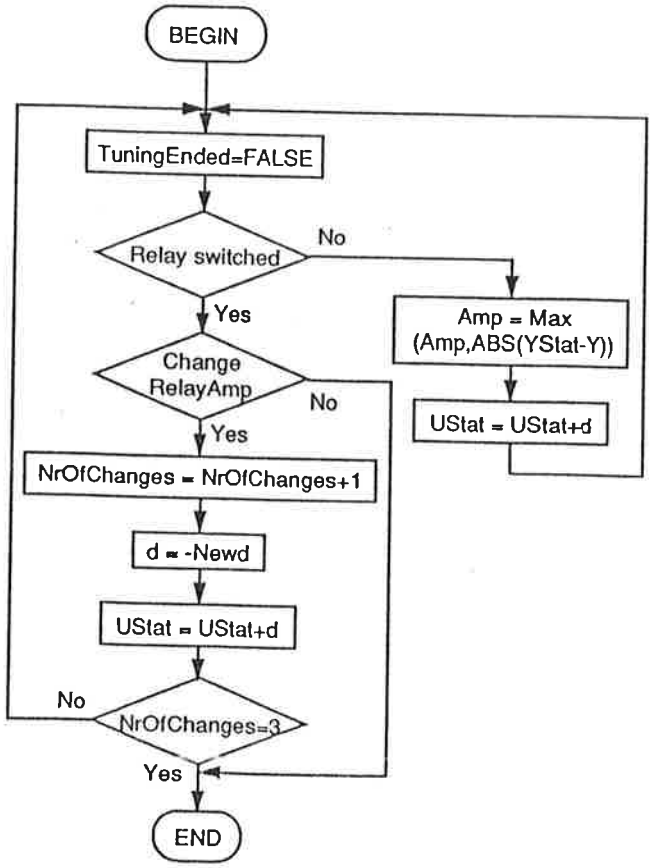

Figure B-3: routine from AutoTune: Initialization of the relay amplitude.

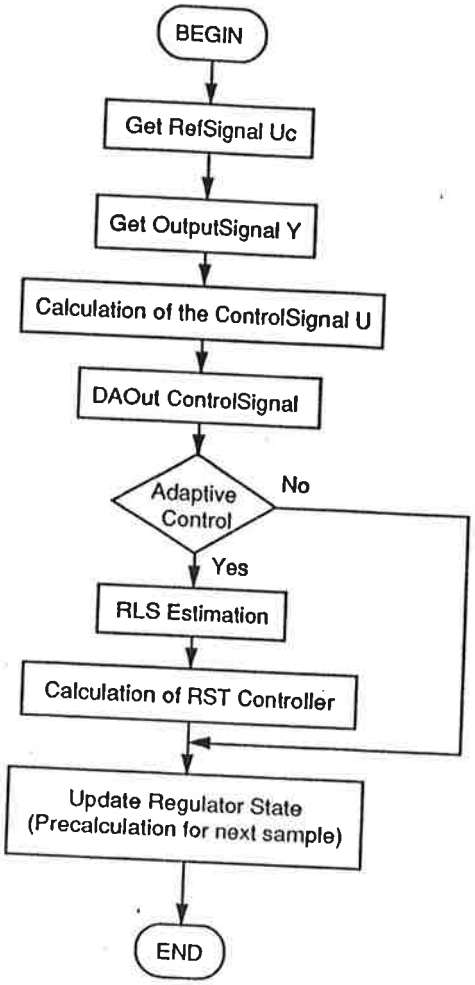

Figure B-7: routine PolePlacement Control.

Description of the table C-1.

Experiments 1 and 2 are performed on the following first order process:

$$\frac{0.113}{z - 0.887} \tag{1}$$

Experiments 4 and 5 are performed on the following second order process:

$$\frac{0.09z + 0.065}{z^2 - 1.21z + 0.368} \tag{2}$$

The first order model is described by the following transfer fonction

$$\frac{b_1 z + b_2}{z^{delay} * z(z-a)} \tag{3}$$

* h is the sampling interval
* y1, y2, y3, y4, y5 and y6 are the measurements points from the output signal
* Eps is the hysteresis of the relay
* emax and emin are respectively the maximum and the minimun values of y.
* TPP and TPN are respectively the length of the positive half period and of the negative half period.
* Tmax and Tmin are respectively the times where the maximum and the minimum.occur on the output signal (counted from the same point).
* Amp is the amplitude of the oscillation on the output signal.
* Period is the period estimated during the relay experiment.

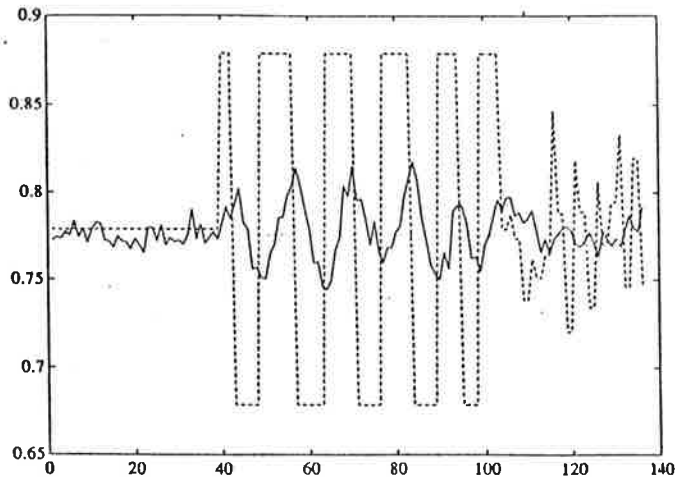

Figure C-1: First order with noise (variance 0.2). Relay experiment.

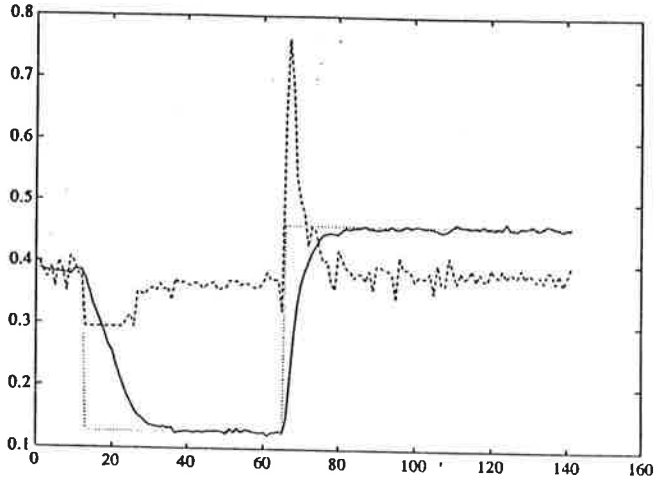

Figure C-5: DC-servo with feedback on speed (the gain is different from 1). Step responses.

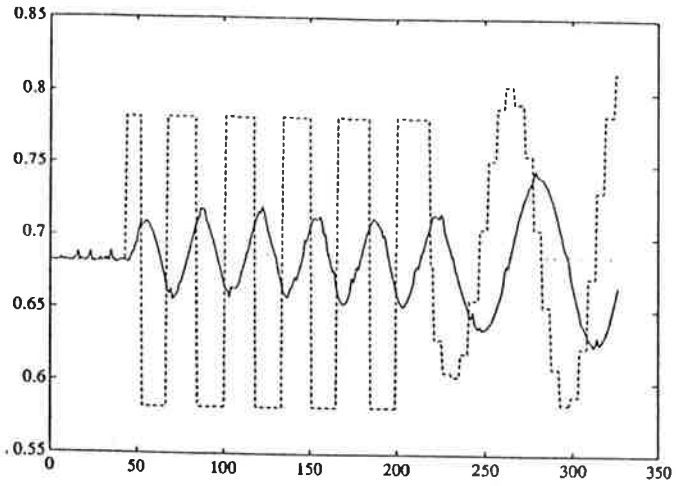

Figure C-6: second order without noise.

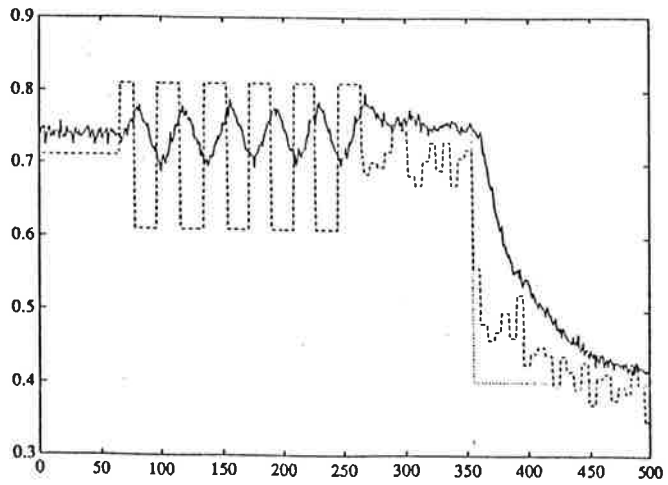

Figure C-7: second order with noise(variance 0.2).

D. Code in Modula 2

This appendix contains the following definition modules and implementation modules:

_ Autoreg
_ AutoTuner
_ AdISTR
_ DSpeBox
_ RegulModule
_ RLS

Each module is composed of a definition part which contains the declarations of the exported identifiers and an implementation module which contains the procedure body. The code of the calculations is very similar to Pascal.

* **Autoreg** is the main procedure.
* **AutoTuner** contains the implementation of the auto-tuning part and the design part.
* **AdISTR** contains the implementation of the pole placement design.
* **DSpeBox** performs the display of the design specifications on the user interaction space
* **RegulModule** supervises the three different modes of the program: Man, PID, Tune.
* **RLS** contains the implementation of the recursive least square estimator.

```
MODULE AutoReg;

FROM AutoTuner         IMPORT GlobalData;
FROM ErrorBox          IMPORT PrintError;
FROM ExitAndEmergency IMPORT WaitForExit,InitWaitForExit;
FROM IO                IMPORT InitIO;
FROM MainBuffer        IMPORT InitMainBuffer;
FROM PlotBuffer        IMPORT InitPlotBuffer;
FROM Regulmon          IMPORT InitRegulmonitor;
FROM RLS               IMPORT InitRLS;
IMPORT Graphics,Plot,RegulModule,OpCom;
IMPORT RTMouse,ExitAndEmergency,GetValues;

VAR
  i : CARDINAL;

BEGIN
   RTMouse.Init;
   InitWaitForExit;
   InitIO;
   InitPlotBuffer;
   InitMainBuffer;
   InitRegulmonitor;
   InitRLS;
   OpCom.Start;
   ExitAndEmergency.Start;
   GetValues.Start;
   Plot.Start;
   RegulModule.Start;
   WaitForExit;
(* -----creation of a file from the array "FileData",
   with the purpose to draw curves from the experiments in MATLAB----- *)
   i := 0;
   WITH GlobalData.PlotFile DO
     IF LogOn THEN
       FOR i:=1 TO NrOfData DO
         Plot.Log(FileData[i]);
(*         PrintError("Fin Log");*)
       END;
     END;
     Plot.Close;
  END;
END AutoReg.
\032
```

```
DEFINITION MODULE AutoTuner;

FROM Pcalc IMPORT cpoly;
FROM RegulModule IMPORT ArrayType,VectorType;

EXPORT QUALIFIED
  free, busy,
  TunePar, ParType, Ymean,
  PlotTypeType, SignalTypeType, AdaptRegType,
  SetupDataType, ModelType, DesignType, RegulatorType, RefSignalType,
  AdaptDataType, EstimDataType,
  GlobalData, TempData,
  InitAutoTune, AutoTune;

CONST
  free = 1;
  busy = 0;

TYPE
  ParType      = RECORD
                   K,Ti,Td:REAL;
                 END;

  ModelType    = RECORD
                   B,A,C  : cpoly;
                   AdditionalPolesInOrigin : CARDINAL;
                   delay : CARDINAL;          (* Redundant information *)
                   BodeWlow,BodeWhigh : REAL;
                 END;

  AdaptRegType    = (fixed,indstr,indstr2,dirstr);
  PlotTypeType    = (Redraw, PlotBetween,
                     PlotEvery, PlotEveryTwo, PlotEveryFour);
  SignalTypeType = (external, square, triangle, sine, step, ramp);

  SetupDataType  = RECORD
                     chanRef,
                     chanY1,
                     chanY2,
                     chanY3,
                     chanU           : CARDINAL;  (* IO channels *)
                     NumberOfInputs  : CARDINAL;
                     PlotWhen        : PlotTypeType;
                     HorizontalTime  : REAL;
                     Tsamp           : CARDINAL;
                     RealTsamp       : REAL;
                     Dt1, Dt2, Dt3   : CARDINAL;  (* For intersample plotting *)
                     Ulow, Uhigh     : REAL;
                   END;

  DesignType     = RECORD
                     Bplus,Bminus,
                     Bmprim,Am,Ac,Amw   : cpoly;
                     IntegralAction : BOOLEAN;
                   END;

  RegulatorType  = RECORD
                     RegType          : ARRAY[0..12] OF CHAR;
                     R,S1,S2,S3,T,Ac : cpoly;
                   END;

  RefSignalType  = RECORD
                     SignalType   : SignalTypeType;
                     Mean,
                     Amplitude,
                     Period       : REAL;
                     TimeInPeriod : REAL;
                   END;

  EstimDataType  = RECORD
                     Bf, Af       : cpoly;
                     D0           : REAL;
                     Lambda       : REAL;
                     HistoryTime  : REAL;
                     HistoryCard  : CARDINAL;
                     EstPurpose   : ARRAY[0..8] OF CHAR;
                                    (* MODELPOLY or REGULPOLY *)
                   END;

  AdaptDataType  = RECORD
                     AdaptReg     : AdaptRegType;
                   (*   RegulatorNr : CARDINAL; *)
                     ulimit       : REAL;
                   END;

  PlotFileType   = RECORD
                     LogOn    : BOOLEAN;
                     NrOfData : CARDINAL;
                     FileData : ARRAY[1..500],[1..4] OF REAL;
                   END;
VAR
  TunePar        : RECORD
                     Period,Time,T0,T3,TimePeriodPos,TimePeriodNeg : REAL;
                     Amp,YStat,Eps,emazmean,eminmean               : REAL;
                     TimeToMaxmean,TimeToMinmean,w,ratew           : REAL;
                     NrOfPeriods                                   : INTEGER;
                   END;
  Ymean          : RECORD
                     Ym : ARRAY [1..20] OF REAL;
                   END;
  GlobalData     : RECORD
                     SetupData       : SetupDataType;
                     Model           : ModelType;
                     Design          : DesignType;
                     (* Regulator    : RegulatorType; *)
                     RefSignal       : RefSignalType;
                     PlotWhichCurve  : BITSET;
                     EstimData       : EstimDataType;
                     AdaptData       : AdaptDataType;
                     Adapt           : BOOLEAN;
                     PlotFile        : PlotFileType;
                   END;

  TempData       : RECORD
                     SetupData       : SetupDataType;
                     Model           : ModelType;
                     Design          : DesignType;
                     Regulator       : RegulatorType;
                     RefSignal       : RefSignalType;
                     (*  PlotWhichCurve : BITSET;            *)
```

```
                AdaptData       : AdaptDataType;
                EstimData       : EstimDataType;
              END;

PROCEDURE InitAutoTune(U,Uc,Y,h:REAL);
PROCEDURE AutoTune(VAR U:REAL; Y:REAL; Uc:REAL;
              VAR TuningEnded,TuningOk:BOOLEAN);

END AutoTuner.
\032
```

```
IMPLEMENTATION MODULE AutoTuner;

FROM Debug IMPORT DebugMouse;
FROM AdISTR      IMPORT StartIndirectSTR,IndirectSTR;
FROM ErrorBox    IMPORT PrintError;
FROM MainBuffer  IMPORT Seth,Geth;
FROM MathLib     IMPORT exp,sqrt,sin,cos,arctan,entier;
FROM Pcalc       IMPORT mdegree,power;
FROM PlotWindow  IMPORT GetLimitsU, GetLimitsY;
FROM RegulModule IMPORT ArrayType;
FROM Regulmon    IMPORT UpdateRegulatorState,RegulMonitor;
FROM Storage     IMPORT ALLOCATE, DEALLOCATE;

CONST MaxNrOfHalfPeriods = 8;

VAR
    Checkh : RECORD
               ch : ARRAY[1..6],[1..3] OF REAL;
             END;
    NrOfCalls,NrOfSamples          : CARDINAL;
    NrOfHalfPeriods,NrOfChanges    : INTEGER;
    NrOfMeasurements               : CARDINAL;
    Ystore                         : ArrayType;
    UStat,UcStat,YMin,YMax         : REAL;
    Maxd,d,Delta,EpsMin,h          : REAL;
    emax,emin,Uold,hprim           : REAL;
    TimeToMax,TimeToMin            : REAL;
    TimePos,TimeNeg                : REAL;
    StartPos,StartNeg              : BOOLEAN;

(*   Remark: The variables are only defined once in the module:
     the first time they are used   *)

(* ------------------------------------------------------------------------ *)
PROCEDURE Min(A,B:REAL):REAL;
(*   Finds the minimum between two reals A and B.
       The result is in Min(A,B)   *)
VAR C : REAL;
BEGIN
  IF A<B THEN
    C := A;
  ELSE
    C := B;
  END;
  RETURN(C);
END Min;

(* ------------------------------------------------------------------------ *)
PROCEDURE Max(A,B:REAL):REAL;
(*   Finds the maximum between two reals A and B.
       The result is in Max(A,B)   *)
VAR C : REAL;
BEGIN
  IF A>B THEN
    C := A;
  ELSE
    C := B;
  END;
  RETURN(C);
END Max;

(* ------------------------------------------------------------------------ *)
PROCEDURE RelaySwitched(d,Y:REAL):BOOLEAN;
(* -----  Analyses if the relay must switch or not from the amplitude
 of the process output y.
       d       is the amplitude of the relay output
       The result is RelaySwitched which is true if the relay must switch----- *)
VAR Switched : BOOLEAN;
BEGIN
 WITH TunePar DO
  Switched := ((d<0.0) AND (Y<YStat-Eps)) OR ((d>0.0) AND (Y>YStat+Eps));
 END;
  RETURN(Switched);
END RelaySwitched;

(* ------------------------------------------------------------------------ *)
PROCEDURE ChangeRelayAmp(VAR d:REAL; VAR Changed:BOOLEAN);
(* -----  Analyses if the amplitude of the relay output must be changed.
 It must be decreased if the amplitude of the process output is too big.
       The result is a boolean, "Changed" which is true if the
       relay output must be changed -----  *)
VAR Newd : REAL;
BEGIN
  WITH TunePar DO
    Newd := d*4.0*Eps/Amp;
  END;
  IF Newd<0.0 THEN
    Newd := Max(-Maxd,Newd);
  ELSE
    Newd := Min(Maxd,Newd);
  END;
  IF ABS(Newd-d) > 0.01*d THEN
    d := Newd;
    Changed := TRUE;
  ELSE
    Changed := FALSE;
  END;
END ChangeRelayAmp;

(* ------------------------------------------------------------------------ *)
PROCEDURE AutoTune(VAR U:REAL; Y:REAL; Uc:REAL;
                   VAR TuningEnded,TuningOk:BOOLEAN);
(* -----  Realizes the relay experiment if the process output
has reached stationarity. It is the Auto-tuning part
       U             is the control signal
       Y             is the process output
       TuningEnded   is a boolean which is true if the tune
                           can not be computed
       TuningOk      is a boolean which is true when the relay
                     experiment is finished
       The outputs of the procedure are TuningEnded, TuningOk and TunePar ----- *)
CONST
    MaxNrOfChanges     = 3;
    AmpDivEps          = 2.0;
VAR
    Changed,OK       : BOOLEAN;
    Stop0            : ARRAY[1..10] OF CHAR;

BEGIN
  IF NrOfCalls<=NrOfMeasurements THEN
```

```
    IF ABS(Y - UcStat) < Delta THEN
      YMin := Min(Y,YMin);
      YMax := Max(Y,YMax);
      NrOfCalls := NrOfCalls + 1;
      NrOfChanges := 0;
      TuningEnded := FALSE;
    ELSE
      TuningEnded := TRUE;
      TuningOk := FALSE;
    END;
  ELSIF NrOfCalls=NrOfMeasurements+1 THEN
    WITH TunePar DO
      Eps := Max(EpsMin,YMax - YMin);
      YStat := (YMax+YMin)/2.0;
    END;
    d := Maxd;
    U := UStat + d;
    NrOfHalfPeriods := -1;
    NrOfCalls := NrOfCalls + 1;
    TuningEnded := FALSE;
  ELSE
     CASE NrOfHalfPeriods OF
        -1   :
             IF RelaySwitched(d,Y) THEN
                d := -d;
                WITH TunePar DO
                  Amp := 0.0;
                END;
                NrOfSamples := 1;
                NrOfChanges := 0;
                NrOfHalfPeriods := 0;
             END;
             U := UStat + d;
             TuningEnded := FALSE;|
      0    :
             IF RelaySwitched(d,Y) THEN
                ChangeRelayAmp(d,Changed);
                IF Changed THEN
                   NrOfChanges := NrOfChanges + 1;
                                  PrintError("RelayChanged");
                END;
                IF NOT Changed OR (NrOfChanges=MaxNrOfChanges) THEN
                   NrOfHalfPeriods := 1;
                END;
                NrOfSamples := 1;
                WITH TunePar DO
                  Amp := 0.0;
                  d := -d;
                  Time := 0.0;
                END;
                InitYmeasurement;
                Ymeasurement(U,Y);
             ELSE
                NrOfSamples := NrOfSamples + 1;
                WITH TunePar DO
                  Amp := Max(Amp,ABS(YStat-Y));
                END;
             END;
             Uold := U;
             U := UStat + d;
             TuningEnded := FALSE;|
       1..MaxNrOfHalfPeriods :
             IF RelaySwitched(d,Y) THEN
                NrOfHalfPeriods := NrOfHalfPeriods + 1;
                NrOfSamples := 1;
                IF (NrOfHalfPeriods <> MaxNrOfHalfPeriods+1) THEN
                   WITH TunePar DO
                     Amp := 0.0;
                   END;
                END;
                   d := -d;
                ELSE
                   NrOfSamples := NrOfSamples + 1;
                END;
                IF (NrOfHalfPeriods <> MaxNrOfHalfPeriods+1) THEN
                   U := UStat + d;
                   Ymeasurement(U,Y);
                   Uold := U;
                ELSE
                   U := UStat;
                END;
                TuningEnded := FALSE;
                WITH RegulMonitor DO
                  yl[4] := yl[3];   ref[4] := ref[3];
                  yl[3] := yl[2];   ref[3] := ref[2];
                  yl[2] := yl[1];   ref[2] := ref[1];
                  yl[1] := yl[0];   ref[1] := ref[0];
                  yl[0] := Y;       ref[0] := Uc;
                  u[4] := u[3];     v[4] := v[3];
                  u[3] := u[2];     v[3] := v[2];
                  u[2] := u[1];     v[2] := v[1];
                  u[1] := u[0];     v[1] := v[0];
                  u[0] := U;        v[0] := U;
                END;
             ELSE
                U := UStat;
                WITH TunePar DO
                  h := Period/6.0;
                END;
                Seth(h);
                InitModel;
                InitPPDesign;
                StartIndirectSTR(GlobalData.Adapt,OK);
                IF OK AND (NOT GlobalData.Adapt) THEN
                  IndirectSTR(FALSE,U,Y);
                  UpdateRegulatorState;
                ELSE
                  IF NOT OK THEN
                   PrintError("IndirectSTR non started");
                  END;
                END;
                TuningEnded := TRUE;
                TuningOk := TRUE;
          END;
    END;
  END AutoTune;

  (* ------------------------------------------------------------------------ *)
  PROCEDURE InitAutoTune(U,Uc,Y,Tsamp:REAL);
  (* ---  Initializes the variables used in the procedure AutoTune.
```

```
        Uc      is the reference signal
        Tsamp   is the sampling interval ---  *)
CONST ProcDelta = 0.03;
      ProcMaxd  = 0.1;
      ProcEpsMin = 0.02;
VAR Hi,Lo : REAL;
    I,J   : INTEGER;
BEGIN
  h := Tsamp;
  UStat := U;
  UcStat := Uc;
  GetLimitsY(Lo,Hi);
  Delta := ProcDelta*(Hi-Lo);
  EpsMin := ProcEpsMin*(Hi-Lo);
  GetLimitsU(Lo,Hi);
  Maxd := Min(ProcMaxd*(Hi-Lo),Min(UStat-Lo,Hi-UStat));
  YMax := Y;
  YMin := Y;
  NrOfMeasurements := entier(5.0/h);
  NrOfCalls := 1;
  WITH Ymean DO
    FOR I := 1 TO 20 DO
      Ym[I] := 0.0;
    END;
  END;
  WITH TunePar DO
    NrOfPeriods := 0;
    T0 := 0.0;
    T3 := 0.0;
    Time := 0.0;
    emaxmean := 0.0;
    eminmean := 0.0;
    w := 0.0;
    ratew := 0.5;
    TimePeriodPos := 0.0;
    TimePeriodNeg := 0.0;
    TimeToMaxmean := 0.0;
    TimeToMinmean := 0.0;
  END;

END InitAutoTune;

(* ------------------------------------------------------------------------ *)
PROCEDURE Ymeasurement(U,Y:REAL);
(* ---  Computes the measurement of values of the process output. ---  *)
CONST
  Pi = 3.141593;
VAR
  Time1,Time2,Time3          : REAL;

BEGIN
  WITH TunePar DO
    Time := Time + h;
  END;

  (* Measurement of 6 values of Y during the
     relay-experiment, the halfperiods and the max and min
     amplitudes reached by Y. One period is shared into 6
     equal parts and the 6 following conditions are obtained
     to measure y at the right place *)

  (* Measurement of the first point *)
  IF ((Uold-UStat)<0.0)AND((U-UStat)>0.0) THEN
    WITH TunePar DO
     IF NrOfPeriods=0 THEN
        StartPos := TRUE;
     END;
     IF NOT StartNeg THEN
        NrOfPeriods := NrOfPeriods + 1;
     END;
     IF NrOfPeriods>1 THEN
       hprim := TimePos/3.0;
     END;
     TimePos := 0.0;
     T0 := Time;
     IF NrOfPeriods>1 THEN
        Time1 := TimeNeg - TimePeriodNeg;
        TimePeriodNeg := TimePeriodNeg + Time1/FLOAT(NrOfPeriods-1);
        IF NrOfPeriods>2 THEN
           Time2 := TimeToMax - TimeToMaxmean;
           TimeToMaxmean := TimeToMaxmean + Time2/FLOAT(NrOfPeriods-2);
           Time3 := TimeToMin - TimeToMinmean;
           TimeToMinmean := TimeToMinmean + Time3/FLOAT(NrOfPeriods-2);
           emaxmean := emaxmean + (emax - emaxmean)/FLOAT(NrOfPeriods-2);
        END;
        WITH Ymean DO
         Ym[1] := Ym[1]+((Y-YStat)/ABS(d)-Ym[1])/FLOAT(NrOfPeriods-1);
        END;
        WITH Checkh DO
           ch[1,NrOfPeriods-1] := hprim;
        END;
        Ystore[1,NrOfPeriods-1] := (Y - YStat)/ABS(d);
    END;
   END;
  ELSE
    IF (U-UStat)>0.0 THEN
       TimePos := TimePos + h;
    END;
  END;

  (* Measurement of the second point *)
 WITH TunePar DO
  IF ((ABS(Time-T0-hprim)<(h/2.0)) AND (NrOfPeriods>1)) THEN
    Ystore[2,NrOfPeriods-1] := (Y - YStat)/ABS(d);
         WITH Checkh DO
           ch[2,NrOfPeriods-1] := hprim;
         END;
   WITH Ymean DO
    Ym[2] := Ym[2] + ((Y - YStat)/ABS(d) - Ym[2])/FLOAT(NrOfPeriods-1);
   END;
  END;

  (* Measurement of the third point *)
  IF ((ABS(Time-T0-2.0*hprim)<(h/2.0)) AND (NrOfPeriods>1)) THEN
    Ystore[3,NrOfPeriods-1] := (Y - YStat)/ABS(d);
         WITH Checkh DO
           ch[3,NrOfPeriods-1] := hprim;
         END;
   WITH Ymean DO
    Ym[3] := Ym[3] + ((Y - YStat)/ABS(d) - Ym[3])/FLOAT(NrOfPeriods-1);
```

```
    END;
  END;
 END;

  (* Measurement of the fourth point *)
  IF ((Uold-UStat)>0.0)AND((U-UStat)<0.0) THEN
    WITH TunePar DO
     IF NrOfPeriods=0 THEN
        StartNeg := TRUE;
     END;
     IF NOT StartPos THEN
        NrOfPeriods := NrOfPeriods + 1;
     END;
     IF NrOfPeriods>1 THEN
       hprim := TimePos/3.0;
     END;
     TimeNeg := 0.0;
     T3 := Time;
     IF NrOfPeriods>1 THEN
        Time1 := TimePos - TimePeriodPos;
        TimePeriodPos := TimePeriodPos + Time1/FLOAT(NrOfPeriods-1);
        IF NrOfPeriods>2 THEN
           eminmean := eminmean + (emin - eminmean)/FLOAT(NrOfPeriods-2);
        END;
       WITH Ymean DO
        Ym[4] := Ym[4] + ((Y-YStat)/ABS(d) - Ym[4])/FLOAT(NrOfPeriods-1);
       END;
        Ystore[4,NrOfPeriods-1] := (Y - YStat)/ABS(d);
          WITH Checkh DO
             ch[4,NrOfPeriods-1] := hprim;
          END;
     END;
    END;
  ELSE
     IF (U-UStat)<0.0 THEN
        TimeNeg := TimeNeg + h;
     END;
  END;

  (* Measurement of the fifth point *)
 WITH TunePar DO
  IF ((ABS(Time-T3-hprim)<(h/2.0)) AND (NrOfPeriods>1)) THEN
     Ystore[5,NrOfPeriods-1] := (Y-YStat)/ABS(d);
          WITH Checkh DO
             ch[5,NrOfPeriods-1] := hprim;
          END;
    WITH Ymean DO
     Ym[5] := Ym[5] + ((Y-YStat)/ABS(d) - Ym[5])/FLOAT(NrOfPeriods-1);
    END;
  END;

  (* Measurement of the last point *)
  IF ((ABS(Time-T3-2.0*hprim)<(h/2.0)) AND (NrOfPeriods>1)) THEN
     Ystore[6,NrOfPeriods-1] := (Y - YStat)/ABS(d);
          WITH Checkh DO
             ch[6,NrOfPeriods-1] := hprim;
          END;
    WITH Ymean DO
     Ym[6] := Ym[6] + ((Y - YStat)/ABS(d) - Ym[6])/FLOAT(NrOfPeriods-1);
    END;
     END;
    END;

     (* Calculation for the time where Ymax happens *)
    WITH TunePar DO
     IF (((Y-YStat)/ABS(d))>emax) AND (NrOfPeriods>1) THEN
        TimeToMax := Time - T0;
        emax := (Y - YStat)/ABS(d);
     END;

     (* Calculation for the time where the Ymin happens *)
     IF (((Y-YStat)/ABS(d))<emin) AND (NrOfPeriods>1) THEN
        TimeToMin := Time - T0;
        emin := (Y - YStat)/ABS(d);
     END;

     IF NrOfPeriods>3 THEN
       Amp := (emaxmean - eminmean)/2.0;
       Period := TimePeriodPos + TimePeriodNeg;
     END;
    END;

 END Ymeasurement;

 (* ------------------------------------------------------------------------ *)
 PROCEDURE InitYmeasurement;
 (* ---  Initializes the procedure Ymeasurement --- *)
 VAR
    i,j : INTEGER;
 BEGIN
    FOR i:= 1 TO 6 DO
      FOR j:= 1 TO 3 DO
       Ystore[i,j] := 0.0;
       WITH Checkh DO
        ch[i,j] := 0.0;
       END;
      END;
    END;
    emax      := 0.0;
    emin      := 0.0;
    TimeToMax := 0.0;
    TimeToMin := 0.0;
    TimeNeg   := 0.0;
    TimePos   := 0.0;
    StartPos := FALSE;
    StartNeg := FALSE;
    hprim :=0.0;
 END InitYmeasurement;

 (* ------------------------------------------------------------------------ *)
 PROCEDURE InitModel;
 (* ---Initialize a first order model with time delay --- *)
 VAR
   f1,f2,f3,d2                              : REAL;
   determinant,b11,b12,b21,b22              : REAL;
   indice2prim,indice3prim,indice4prim      : REAL;
   indice2,indice3,indice4                  : INTEGER;
   i,d,d1                                   : INTEGER;
 BEGIN
   (* Determination of the delay *)
```

```
  WITH TunePar DO
    d1 := entier(TimeToMinmean/h);
  END;
  WITH Ymean DO
    IF ABS(Ym[d1+1])>ABS(Ym[d1+2]) THEN
      d := d1;
    ELSE
      d := d1+1;
    END;
  END;
  (* Initialisation of the model *)
  indice2 := (d+2) MOD 3;
  IF indice2 = 0 THEN
     indice2 := 3;
  END;
  indice2prim := power(-1.0,(d+2-indice2) DIV 3);
  indice3 := (d+3) MOD 3;
  IF indice3 = 0 THEN
     indice3 := 3;
  END;
  indice3prim := power(-1.0,(d+3-indice3) DIV 3);
  indice4 := (d+4) MOD 3;
  IF indice4 = 0 THEN
     indice4 := 3;
  END;
  indice4prim := power(-1.0,(d+4-indice4) DIV 3);
 WITH Ymean DO
   IF Ym[1]<0.0 THEN
     f1 := Ym[1];
     f2 := Ym[2];
     f3 := Ym[3];
     Ym[1] := Ym[4];
     Ym[2] := Ym[5];
     Ym[3] := Ym[6];
     Ym[4] := f1;
     Ym[5] := f2;
     Ym[6] := f3;
   END;
   Ym[d+2] := indice2prim * Ym[indice2];
   Ym[d+3] := indice3prim * Ym[indice3];
   Ym[d+4] := indice4prim * Ym[indice4];
   determinant := Ym[d+3] - Ym[d+2];
 END;
  IF ABS(determinant)>0.0001 THEN
    WITH Ymean DO
     b21 := -power(Ym[d+2],2)-power(Ym[d+3],2)+power(Ym[d+4],2);
     b22 := Ym[d+2]*Ym[d+3]+Ym[d+2]*Ym[d+4]-Ym[d+3]*Ym[d+4];
     b11 := power(Ym[d+2],2)-power(Ym[d+3],2)-power(Ym[d+4],2);
     b12 := -Ym[d+2]*Ym[d+3]+Ym[d+2]*Ym[d+4]+Ym[d+3]*Ym[d+4];
    END;
     WITH GlobalData.Model.B DO          (* --- b1z+b2 --- *)
       coeffs[0] := (b11+b12) / (2.0*determinant);
       coeffs[1] := (b21+b22) / (2.0*determinant);
       FOR i:=2 TO mdegree DO coeffs[i] := 0.0; END;
       IF ABS(coeffs[1])>0.001 THEN
         degree := 1;
       ELSE
         degree := 0;
       END;
     END;
     WITH GlobalData.Model.A DO          (* --- (z-a) --- *)
       coeffs[0] := 1.0;
      WITH Ymean DO
       coeffs[1] := -(Ym[d+4] - Ym[d+3]) / determinant;
      END;
       FOR i:=2 TO mdegree DO coeffs[i] := 0.0; END;
         degree := 1;
     END;
     WITH GlobalData.Model DO
(*       AdditionalPolesInOrigin := d;*)
       IF ABS(B.coeffs[0])>0.001 THEN
         delay := d+1;
         IF ABS(B.coeffs[1])>0.001 THEN
           AdditionalPolesInOrigin := d+1;
         ELSE
           AdditionalPolesInOrigin := d;
         END;
       ELSE
         delay := d+2;
         IF ABS(B.coeffs[1])>0.001 THEN
           AdditionalPolesInOrigin := d+1;
         ELSE
           PrintError("b1=b2=0,Impossible");
         END;
       END;
     END;
  ELSE
     PrintError("Determinant = 0");
  END;
  TempData.Model := GlobalData.Model;
END InitModel;

(* ------------------------------------------------------------------------ *)
PROCEDURE InitPPDesign;
(* --- Initialization of a controller based on pole placement design --- *)
CONST
   pi=3.1416;
VAR
   i : INTEGER;
   inter1,alpha,beta : REAL;
BEGIN
  WITH GlobalData DO
(* ---if the model is a first order like k/(z-a) the desired closed loop
  denominator is also a first order--- *)
    IF ABS(Model.B.coeffs[1])<0.001 THEN
      WITH TunePar DO
        w := ratew/Period;
        alpha := exp(-w*h);
      END;
      Design.Am.degree := Model.delay+1;
      Design.Am.coeffs[0] := 1.0;                    (* Am=zdelay(z-alpha) *)
      Design.Am.coeffs[1] := -alpha;
      FOR i:=2 TO mdegree DO Design.Am.coeffs[i] :=0.0; END;
      inter1 := Model.B.coeffs[0]+Model.B.coeffs[1];
      IF ABS(inter1)>0.001 THEN
        Design.Bmprim.degree := 0;                   (* Bm' = const *)
        Design.Bmprim.coeffs[0] := (1.0-alpha)/inter1;
        FOR i:=1 TO mdegree DO Design.Bmprim.coeffs[i] :=0.0; END;
      ELSE
        PrintError("b1+b2 = 0");
```

```
      END;
    ELSE
(* ---if we are not in the previous case the closed loop desired denominator
  is a second order --- *)
      WITH TunePar DO
        w := ratew*2.0*pi/Period;
        alpha := exp(-0.7*w*h);
        beta := cos(w*h*sqrt(1.0-0.7*0.7));
      END;
      Design.Am.degree := Model.delay+2;
      Design.Am.coeffs[0] := 1.0;                    (* Am=zdelay(z2+plz+p2) *)
      Design.Am.coeffs[1] := -2.0*alpha*beta;
      Design.Am.coeffs[2] := power(alpha,2);
      FOR i:=3 TO mdegree DO Design.Am.coeffs[i] :=0.0; END;
      inter1 := Model.B.coeffs[0]+Model.B.coeffs[1];
      IF ABS(inter1)>0.001 THEN
        Design.Bmprim.degree := 0;                    (* Bm' = const *)
        Design.Bmprim.coeffs[0] := (1.0-2.0*alpha*beta+power(alpha,2))/inter1;
        FOR i:=1 TO mdegree DO Design.Bmprim.coeffs[i] :=0.0; END;
      ELSE
        PrintError("b1+b2 = 0");
      END;
    END;
    Design.Bplus.degree := 0;                         (* B+ = 1 *)
    Design.Bplus.coeffs[0] := 1.0;
    FOR i:=1 TO mdegree DO Design.Bplus.coeffs[i] :=0.0; END;
    Design.Bminus := Model.B;                         (* B- = B *)
    Design.Ao.degree := Model.delay;                  (* Ao *)
    Design.Ao.coeffs[0] := 1.0;
    FOR i:=1 TO mdegree DO Design.Ao.coeffs[i] :=0.0; END;
  END;
END InitPPDesign;

END AutoTuner.
\032
```

```
DEFINITION MODULE AdISTR;

EXPORT QUALIFIED
  StartIndirectSTR, IndirectSTR;


PROCEDURE StartIndirectSTR(Adapt:BOOLEAN;VAR ok:BOOLEAN);
PROCEDURE IndirectSTR(Adapt:BOOLEAN;u,y:REAL);

END AdISTR.
\032
```

```
IMPLEMENTATION MODULE AdISTR;

(*-------------------------------------------------------------------------*)
(*               Routines for poles placement design                       *)
(*                                                                         *)
(*               Michael Lundh            March 1988                       *)
(*-------------------------------------------------------------------------*)
FROM AutoTuner  IMPORT ModelType, DesignType, RegulatorType,
                       GlobalData, AdaptDataType, TempData;
FROM ErrorBox   IMPORT PrintError;
FROM Regulmon   IMPORT NewAdaptiveRegulator,
                       GetPolyInRegulator, NewPolyInRegulator;
FROM Pcalc      IMPORT mdegree, cpoly, Polnorm, Polmul, Diophantine, FixDegree;
FROM RLS        IMPORT RestartEstimation;
CONST
  relative =1.0E-8;

VAR
  AoAm, Integr, Ttmp,
  Aext, AextIntegr  : cpoly;
  amqeq1 : REAL;

(* ------------------------------------------------------------------------ *)
PROCEDURE StartIndirectSTR(Adapt:BOOLEAN;VAR ok:BOOLEAN);
(* Adapt=TRUE when adaptive regulator running, FALSE when initialisation *)
VAR i:CARDINAL;
BEGIN
  (* ??????????? hur ar det med modellen -- behover tester goras *)
  WITH GlobalData.Design DO
    Polmul(Ao, Am, relative, AoAm);
    IF Polnorm(AoAm)<0.0001 THEN
      PrintError(' Am * Ao = 0 ');
      ok := FALSE;
      RETURN;
    END;

    IF Polnorm(Bmprim)<0.0001 THEN
      PrintError(" Bm' = 0 ");
      ok := FALSE;
      RETURN;
    END;

    IF IntegralAction THEN
      WITH Integr DO
        degree:=1;  coeffs[0]:=1.0;  coeffs[1]:=-1.0;
        FOR i:=2 TO mdegree DO  coeffs[i]:=0.0;  END;
      END;
    ELSE
      WITH Integr DO
        degree:=0;  coeffs[0]:=1.0;
        FOR i:=1 TO mdegree DO  coeffs[i]:=0.0;  END;
      END;
    END;

    WITH Aext DO (* handle addtional poles in the origin *)
      degree:=GlobalData.Model.AdditionalPolesInOrigin;
      coeffs[0]:=1.0;
      FOR i:=1 TO mdegree DO  coeffs[i]:=0.0;  END;
    END;
    Polmul(Integr,Aext,relative,AextIntegr);

    (* degree test  deg(Ao)+deg(Am) >= 2deg(A)-deg(B+)+1-1 *)
    WITH GlobalData.Model DO
      IF AoAm.degree < 2*(A.degree+Aext.degree) -0 +Integr.degree -1 THEN
        ok:=FALSE;
        PrintError(' Degree fault1 ');
        RETURN;
      END;
    END;

    amqeq1 := 0.0;
    FOR i:=0 TO Am.degree DO (* Am(1) *)
      amqeq1 := amqeq1 + Am.coeffs[i];
    END;
    IF ABS(amqeq1) < 0.001 THEN
      ok:=FALSE;
      PrintError(' Am(1) = 0 ');
      RETURN;
    END;

    Polmul(Ao,Bmprim,relative,Ttmp);

    (* tilldela regulator polynom gradtal *)
    GetPolyInRegulator(TempData.Regulator);
    TempData.Regulator.RegType := 'ISTR';
    TempData.Regulator.T := Ttmp;
    TempData.Regulator.R.degree := AoAm.degree - Aext.degree
                                                - GlobalData.Model.A.degree;
    TempData.Regulator.S1.degree:= TempData.Regulator.R.degree;
    TempData.Regulator.T.degree := TempData.Regulator.R.degree;
    TempData.Regulator.Ao.degree:= TempData.Regulator.R.degree;

    TempData.Regulator.S2.degree:= 0;
    TempData.Regulator.S3.degree:= 0;
    FOR i:=0 TO mdegree DO
      TempData.Regulator.S2.coeffs[i] := 0.0;
      TempData.Regulator.S3.coeffs[i] := 0.0;
    END;

      NewPolyInRegulator(TempData.Regulator);

    ok:=TRUE;
  END;
    RestartEstimation(FALSE,ok);
END StartIndirectSTR;

(* ------------------------------------------------------------------------ *)
PROCEDURE IndirectSTR(Adapt:BOOLEAN;u,y:REAL);
(* Adapt=TRUE when adaptive regulator running, FALSE when initialisation *)
VAR i:CARDINAL;
    bmqeq1, t0, r0: REAL;
    a1, bm, r1: cpoly;
BEGIN
  WITH GlobalData DO
    Polmul(Model.A,AextIntegr,relative,a1);

    (* B- = B    B+ = 1 *)
    Polmul(Model.B,Design.Bmprim,relative,bm);

    bmqeq1 := 0.0;
```

```
    FOR i:=0 TO bm.degree DO (* bm(1) *)
      bmqeq1 := bmqeq1 + bm.coeffs[i];
    END;
    t0 := amqeq1/bmqeq1;

    WITH TempData.Regulator DO
      Diophantine(a1,Model.B,AoAm,r1,S1);
      Polmul(r1,Integr,relative,R);
      FixDegree(S1,R.degree);

      r0:=R.coeffs[0];  (* = 1.0 always ??? *)

      FOR i:=0 TO R.degree DO
        R.coeffs[i]  := R.coeffs[i] / r0;
        S1.coeffs[i] := S1.coeffs[i] / r0;
        T.coeffs[i]  := Ttmp.coeffs[i] / r0 * t0;
      END;
    END;
  END;

    NewAdaptiveRegulator(TempData.Regulator);
END IndirectSTR;

END AdISTR.

\032
```

```
DEFINITION MODULE DSpeBox;

FROM Graphics IMPORT rectangle;

EXPORT QUALIFIED Init,Draw,DSpe;

PROCEDURE Init(r:rectangle);

PROCEDURE Draw;

PROCEDURE DSpe;

END DSpeBox.
\032
```

```
IMPLEMENTATION MODULE DSpeBox;

FROM AdISTR            IMPORT StartIndirectSTR,IndirectSTR;
FROM AutoTuner         IMPORT GlobalData,TempData,RegulatorType;
FROM ConvReal          IMPORT RealToString,StringToReal;
FROM ErrorBox          IMPORT PrintError;
FROM GraphHelp         IMPORT GetTextColor,GetBoxColor,SetPoint,SetRectangle,
                              GetLeftIn,ClearLeft;
FROM Graphics          IMPORT handle,rectangle,color,point,VirtualScreen,SetWindow,
                              SetViewPort,WriteString,SetFillColor,SetTextColor,
                              FillRectangle,ReadString,CharacterSize,EraseChar,
                              HideCursor,ShowCursor;
FROM IO                IMPORT ADInY;
FROM MainBuffer        IMPORT ActualData;
FROM MathLib           IMPORT round;
FROM NumberConversion  IMPORT CardToString,StringToCard,
                              IntToString,StringToInt;
FROM Regulmon          IMPORT UpdateRegulatorState;
FROM Strings           IMPORT Copy,Insert;

VAR DSHandle:handle;
    Box:rectangle;
    Text:ARRAY[0..2],[0..10] OF CHAR;
    TextPoint:ARRAY[1..2] OF point;

(* ------------------------------------------------------------------------- *)
PROCEDURE Init(ViewPort:rectangle);
BEGIN
   VirtualScreen(DSHandle);
   SetViewPort(DSHandle,ViewPort);
   SetRectangle(Box,0.0,0.0,1.0,1.0);
   SetWindow(DSHandle,Box);
   SetFillColor(DSHandle,GetBoxColor());
   SetTextColor(DSHandle,GetTextColor());
   Copy("Design",0,6,Text[1]);
   SetPoint(TextPoint[1],0.1,0.6);
   Copy("Specifs",0,7,Text[2]);
   SetPoint(TextPoint[2],0.1,0.1);
END Init;

(* ------------------------------------------------------------------------- *)
PROCEDURE Draw;
BEGIN
   HideCursor;
   FillRectangle(DSHandle,Box);
   WriteString(DSHandle,TextPoint[1],Text[1]);
   WriteString(DSHandle,TextPoint[2],Text[2]);
   ShowCursor;
END Draw;

(* ------------------------------------------------------------------------- *)
PROCEDURE DSpe;
(* ---performs the display of the design specifications in the
  user interaction space --- *)
CONST NrOfStrings=7;
      un=1;
      zero=0;
VAR DSpeHandle                          : handle;
    ViewPort,Box                        : rectangle;
    width,height,Value,Y,U              : REAL;
    Test,ModifAoAm,OK                   : BOOLEAN;
    Len,Line,Pos,Valuec                 : CARDINAL;
    Text                                : ARRAY[1..NrOfStrings],[0..30] OF CHAR;
    Ans                                 : ARRAY[1..2] OF CHAR;
    NumString                           : ARRAY[0..12] OF CHAR;
    TextPoint                           : ARRAY[1..NrOfStrings] OF point;
    PromptPoint,ReadPoint1,ReadPoint2   : point;
    ReadPoint3,ReadPoint4,ClearPoint    : point;

BEGIN
   ModifAoAm := FALSE;
   VirtualScreen(DSpeHandle);
   GetLeftIn(ViewPort);
   SetViewPort(DSpeHandle,ViewPort);
   SetRectangle(Box,0.0,0.0,1.0,1.0);
   SetWindow(DSpeHandle,Box);
   SetFillColor(DSpeHandle,GetBoxColor());
   SetTextColor(DSpeHandle,GetTextColor());
   CharacterSize(DSpeHandle,width,height);
   FOR Line:= 1 TO NrOfStrings DO
      SetPoint(TextPoint[Line],0.05,1.0-FLOAT(Line)*1.2*height);
   END;
   SetPoint(PromptPoint,0.05,0.05);
   SetPoint(ReadPoint1,0.05+width,0.05);
   SetPoint(ClearPoint,0.05+2.0*width,0.05);
   SetPoint(ReadPoint2,0.05+3.0*width,0.05);
   SetPoint(ClearPoint,0.05+4.0*width,0.05);
   SetPoint(ReadPoint3,0.05+5.0*width,0.05);
   SetPoint(ClearPoint,0.05+8.0*width,0.05);
   SetPoint(ReadPoint4,0.05+9.0*width,0.05);

   REPEAT
      Copy("1 Ao.degree",0,11,Text[1]);
      Copy("2 Ao",0,4,Text[2]);
      Copy("3 Am.degree",0,11,Text[3]);
      Copy("4 Am",0,4,Text[4]);
      Copy("5 Bm'",0,5,Text[5]);
      Copy("6 IntAction",0,11,Text[6]);
      Copy("E Exit",0,6,Text[7]);
      Len:=7;
      WITH GlobalData.Design DO
        CardToString(Ao.degree,NumString,Len);
        Insert(NumString,Text[1],11);
        RealToString(Ao.coeffs[0],NumString,Len);
        Insert(NumString,Text[2],4);
        IF Ao.degree>=1 THEN
          RealToString(Ao.coeffs[1],NumString,Len);
          Insert(NumString,Text[2],13);
          IF Ao.degree>=2 THEN
            RealToString(Ao.coeffs[2],NumString,Len);
            Insert(NumString,Text[2],22);
          END;
        END;
        CardToString(Am.degree,NumString,Len);
        Insert(NumString,Text[3],11);
        RealToString(Am.coeffs[0],NumString,Len);
        Insert(NumString,Text[4],4);
        RealToString(Am.coeffs[1],NumString,Len);
        Insert(NumString,Text[4],13);
        RealToString(Am.coeffs[2],NumString,Len);
```

```
        Insert(NumString,Text[4],22);
        RealToString(Bmprim.coeffs[0],NumString,Len);
        Insert(NumString,Text[5],5);
        IF IntegralAction THEN
          IntToString(un,NumString,Len);
        ELSE
          IntToString(zero,NumString,Len);
        END;
        Insert(NumString,Text[6],11);
      END;
      HideCursor;
      FillRectangle(DSpeHandle,Box);
      FOR Line:= 1 TO NrOfStrings DO
         WriteString(DSpeHandle,TextPoint[Line],Text[Line]);
      END;
      WriteString(DSpeHandle,PromptPoint,">");
      ShowCursor;
      ReadString(DSpeHandle,ReadPoint1,Ans);
      HideCursor;
      EraseChar(DSpeHandle,ClearPoint,1);
      ShowCursor;
(* ---Ao and Am can be changed from the user interaction space
  the coefficients and the degree should be changed --- *)
      IF (Ans[1]="1") THEN
       IF NOT GlobalData.Adapt THEN
         ModifAoAm := TRUE;
         ReadString(DSpeHandle,ReadPoint2,NumString);
         Pos:=0;
         StringToCard(NumString,Valuec,Test);
         IF Test THEN
            GlobalData.Design.Ao.degree:=Valuec;
         ELSE
            PrintError("Conversion Error");
         END;
       ELSE
         PrintError("Adaptive Regulator");
         PrintError("Modification impossible");
       END;
      END;
      IF (Ans[1]="2") THEN
       IF NOT GlobalData.Adapt THEN
         ModifAoAm := TRUE;
         ReadString(DSpeHandle,ReadPoint2,NumString);
         Pos:=0;
         StringToReal(NumString,Pos,Value);
         IF Pos<>0 THEN
            GlobalData.Design.Ao.coeffs[0]:=Value;
         ELSE
            PrintError("Conversion Error");
         END;
         ReadString(DSpeHandle,ReadPoint3,NumString);
         Pos:=0;
         StringToReal(NumString,Pos,Value);
         IF Pos<>0 THEN
            GlobalData.Design.Ao.coeffs[1]:=Value;
         ELSE
            PrintError("Conversion Error");
         END;
         ReadString(DSpeHandle,ReadPoint4,NumString);
         Pos:=0;
         StringToReal(NumString,Pos,Value);
         IF Pos<>0 THEN
            GlobalData.Design.Ao.coeffs[2]:=Value;
         ELSE
            PrintError("Conversion Error");
         END;
       ELSE
         PrintError("Adaptive Regulator");
         PrintError("Modification impossible");
       END;
      END;
      IF (Ans[1]="3") THEN
       IF NOT GlobalData.Adapt THEN
         ModifAoAm := TRUE;
         ReadString(DSpeHandle,ReadPoint2,NumString);
         Pos:=0;
         StringToCard(NumString,Valuec,Test);
         IF Test THEN
            GlobalData.Design.Am.degree:=Valuec;
         ELSE
            PrintError("Conversion Error");
         END;
       ELSE
         PrintError("Adaptive Regulator");
         PrintError("Modification impossible");
       END;
      END;
      IF (Ans[1]="4") THEN
       IF NOT GlobalData.Adapt THEN
         ReadString(DSpeHandle,ReadPoint2,NumString);
         Pos:=0;
         StringToReal(NumString,Pos,Value);
         IF Pos<>0 THEN
            GlobalData.Design.Am.coeffs[0]:=Value;
         ELSE
            PrintError("Conversion Error");
         END;
         ReadString(DSpeHandle,ReadPoint3,NumString);
         Pos:=0;
         StringToReal(NumString,Pos,Value);
         IF Pos<>0 THEN
            GlobalData.Design.Am.coeffs[1]:=Value;
         ELSE
            PrintError("Conversion Error");
         END;
         ReadString(DSpeHandle,ReadPoint4,NumString);
         Pos:=0;
         StringToReal(NumString,Pos,Value);
         IF Pos<>0 THEN
            GlobalData.Design.Am.coeffs[2]:=Value;
         ELSE
            PrintError("Conversion Error");
         END;
       ELSE
         PrintError("Adaptive Regulator");
         PrintError("Modification impossible");
       END;
      END;
    UNTIL (CAP(Ans[1])="E");
    IF ModifAoAm THEN
```

```
      StartIndirectSTR(GlobalData.Adapt,OK);
      IF OK THEN
        PrintError("IndirectSTR started");
        Y := ADInY();
        U := ActualData.ut0;
        IndirectSTR(FALSE,U,Y);
        UpdateRegulatorState;
      ELSE
        PrintError("Indirect STR not started");
      END;
    END;
    ClearLeft;
END DSpe;

END DSpeBox.
\032
```

```
DEFINITION MODULE RegulModule;

EXPORT QUALIFIED Start, ArrayType, VectorType;

TYPE VectorType = ARRAY[1..11] OF REAL;

TYPE ArrayType = ARRAY [1..6],[1..3] OF REAL;

PROCEDURE Start;

END RegulModule.
\032
```

```
IMPLEMENTATION MODULE RegulModule;

IMPORT DebugPMD;
FROM AdISTR      IMPORT IndirectSTR;
FROM AutoTuner   IMPORT TunePar, ParType, InitAutoTune, AutoTune,
                        GlobalData,TempData,Ymean;
FROM ErrorBox    IMPORT PrintError;
FROM IO          IMPORT ADInY, ADInExt, DAOutU;
FROM Kernel      IMPORT Time, SetPriority, GetTime, IncTime, WaitUntil,
                        CreateProcess;
FROM MainBuffer IMPORT ModeType, GetMode, SetMode, GetPar, SetPar, GetSignal,
                        SetSignal, Geth, SetTuningOk, SetGSSignals,
                        GetGSSignal, SignalType, ActualData;
FROM PlotBuffer IMPORT PutData;
FROM PlotWindow IMPORT GetLimitsU;
FROM Regulator   IMPORT ControlSignal, InitRegulator, ResetRegulator;
FROM Regulmon    IMPORT NewControlSignal, NewAdaptiveRegulator,
                        UpdateRegulatorState;
FROM RLS         IMPORT Estimation;
FROM Storage     IMPORT ALLOCATE, DEALLOCATE;

VAR
    K,Mean      : REAL;

(* ------------------------------------------------------------------------ *)
PROCEDURE InitMean;
BEGIN
  K := 100.0;
  Mean := 0.0;
END InitMean;

(* ------------------------------------------------------------------------ *)
PROCEDURE UpdateMean(X:REAL);
(*   Computes the mean of a real signal, X   *)
CONST Lambda = 0.95;
BEGIN
  K := K/(K+Lambda);
  Mean := Mean + K*(X-Mean);
END UpdateMean;

(* ------------------------------------------------------------------------ *)
PROCEDURE GetMean():REAL;
BEGIN
  RETURN(Mean);
END GetMean;

(* ------------------------------------------------------------------------ *)
PROCEDURE Regul;
(* Process which supervises the choice of the modes Man, PID, Tune,Adapt *)
VAR Mode,LastMode,ModeBeforeTuning : ModeType;
    h,uc,y,u,ext,v,UMin,UMax,Dummy : REAL;
    Par                            : ParType;
    TuningEnded,TuningOk           : BOOLEAN;
    T                              : Time;
BEGIN
  SetPriority(20);
  LastMode := GetMode();
  GetLimitsU(UMin,UMax);
  u := 0.0;
  LOOP
    GetTime(T);
    h := Geth();
    IncTime(T,TRUNC(1000.0*h));
    Mode := GetMode();
    CASE Mode OF
      Man  : uc := GetSignal(Uc);
             y := ADInY();
             SetSignal(Y,y);
             GetPar(Par);
             Dummy := ControlSignal(uc,y,u,h,Par);
             v := GetSignal(U);
             IF v < UMin THEN
               u := UMin;
             ELSIF v > UMax THEN
               u := UMax;
             ELSE
               u := v;
             END;
             DAOutU(u);|
      PID  : uc := GetSignal(Uc);
             y := ADInY();
             SetSignal(Y,y);
             GetPar(Par);
             v := ControlSignal(uc,y,u,h,Par);
             IF v < UMin THEN
               u := UMin;
             ELSIF v > UMax THEN
               u := UMax;
             ELSE
               u := v;
             END;
             DAOutU(u);|
      Tune : y := ADInY();
             IF LastMode <> Tune THEN
               v := GetMean();
               InitAutoTune(v,GetSignal(Uc),y,h);
               SetGSSignals(v,y,ADInExt());
               ModeBeforeTuning := LastMode;
             END;
             AutoTune(v,y,uc,TuningEnded,TuningOk);
             IF v < UMin THEN
               u := UMin;
             ELSIF v > UMax THEN
               u := UMax;
             ELSE
               u := v;
             END;
             DAOutU(u);
             IF TuningEnded THEN
               IF TuningOk THEN
                 SetTuningOk;
                         NewAdaptiveRegulator(TempData.Regulator);
                 PrintError("Tuning was successful.");
                 SetMode(Control);
               ELSE
                 PrintError("Tuning wasn't successful.");
                 SetMode(ModeBeforeTuning);
               END;
             END;|
    Control : uc := GetSignal(Uc);
```

```
            y := ADInY();
            SetSignal(Y,y);
            WITH ActualData DO
               reft0 := uc;
              y1t0 := ADInY();
                   y2t0 := 0.0;
                     y3t0 := 0.0;
              ut0 := NewControlSignal(reft0,y1t0,y2t0,y3t0);
            END;
            v := ActualData.ut0;
            IF v < UMin THEN
              u := UMin;
            ELSIF v > UMax THEN
              u := UMax;
            ELSE
              u := v;
            END;
            DAOutU(u);
            Estimation(u,y);
            IF GlobalData.Adapt THEN
              IndirectSTR(TRUE,u,y);
            END;
            UpdateRegulatorState;
    END; (* case *)
    UpdateMean(u);
    SetSignal(U,u);
    PutData(u,uc);
    LastMode := Mode;
    WaitUntil(T);
  END;
END Regul;

(* ------------------------------------------------------------------------ *)
PROCEDURE Start;
BEGIN
  InitRegulator;
  InitMean;
  CreateProcess(Regul,20000);
END Start;

END RegulModule.
\032
```

```
DEFINITION MODULE RLS;

EXPORT QUALIFIED
  MaxStore, maxindex, col, matr, EstimateParameters,
  RestartEstimation, InitRLS, Estimation, DORLS;

CONST
  MaxStore = 710;
  maxindex = 10;   (* max number of estimated parameters *)

TYPE
  col  = ARRAY[0..maxindex] OF REAL;
  matr = ARRAY[0..maxindex] OF col;

  Filter2State = RECORD
                   x1,x2 : REAL;
                 END;

VAR
  EstimateParameters: RECORD
                        Theta : col;  (* estimated parameters *)
                        Phi   : col;  (* regression vector *)
                        L     : matr; (* lower triangular part of LD decompo-
                                         sition of P. Notice that the elements
                                         are stored columnwise *)
                        D     : col;  (* diag. part of LD decomposition of P *)
                        N     : CARDINAL; (* number of estimated param *)
                      END;
PROCEDURE RestartEstimation(zerotheta:BOOLEAN; VAR ok:BOOLEAN);
PROCEDURE InitRLS;
PROCEDURE Filter2(x:REAL; VAR states:Filter2State; VAR xf:REAL);
PROCEDURE Estimation(u,y:REAL);
PROCEDURE DORLS(VAR theta,d:col; VAR l:matr; phi:col; n:CARDINAL);

END RLS.
```

```
IMPLEMENTATION MODULE RLS;

(*-------------------------------------------------------------------------*)
(*          Routines for recursive least square estimation                 *)
(*                                                                         *)
(*          Michael Lundh                          March 1988              *)
(*-------------------------------------------------------------------------*)
FROM Pcalc      IMPORT mdegree. coeffvector, cpoly;
FROM AutoTuner IMPORT ModelType, GlobalData, EstimDataType;
FROM ErrorBox   IMPORT PrintError;

CONST
  relative =1.0E-8;

VAR
  uf1, uf2, uf3 : REAL;
  ufilterstate,
  yfilterstate  : Filter2State;

(*-------------------------------------------------------------------------*)

PROCEDURE Filter2(x:REAL; VAR states:Filter2State; VAR xf:REAL);
(* Second order filter for regressors *)
VAR nx1,nx2: REAL;
BEGIN
  WITH GlobalData.EstimData DO
    WITH states DO
      nx1 := -Af.coeffs[1]*x1 - Af.coeffs[2]*x2 + x;
      nx2 :=                x1;
      xf  := (Bf.coeffs[1]-Bf.coeffs[0]*Af.coeffs[1])*x1
           + (Bf.coeffs[2]-Bf.coeffs[0]*Af.coeffs[2])*x2 + Bf.coeffs[0]*x;

      x1:=nx1;
      x2:=nx2;
    END;
  END;
END Filter2;

(*-------------------------------------------------------------------------*)
(*  Routines for RLS estimation based on dyadic reduction.                 *)
(*  Ref. Peterka IFAC-86.                                                  *)
(*                                                                         *)
(*    Karl-Johan Astrom and Michael Lundh                  87.05.05        *)
(*-------------------------------------------------------------------------*)

PROCEDURE
DyadicReduction(VAR a,b:col; VAR alpha,beta:PEAL; i0,i1,i2 :CARDINAL);
CONST
  mzero = 1.0E-10;
VAR
  i            : CARDINAL;
  w1,w2,b1,gam : REAL;
BEGIN
  IF beta<mzero THEN beta:=0.0; END;
  b1 := b[i0];
  w1 := alpha;
  w2 := beta*b1;
  alpha := alpha + w2*b1;
  IF alpha > mzero THEN
    beta := w1*beta/alpha;
    gam := w2/alpha;
    FOR i:=i1 TO i2 DO
      b[i] := b[i] - b1*a[i];
      a[i] := a[i] + gam*b[i];
    END;
  END;
END DyadicReduction;

(* ------------------------------------------------------------------------- *)
PROCEDURE
LDFilter(VAR theta,d:col; VAR l:matr; phi:col; lambda:REAL; n:CARDINAL);
VAR
  i,j  : CARDINAL;
  e,w  : REAL;
BEGIN
  d[0] := lambda;
  e    := phi[0];
  FOR i:=1 TO n DO
    e:=e-theta[i]*phi[i];
    w:=phi[i];
    FOR j:=i+1 TO n DO w:=w+phi[j]*l[i,j]; END;
    l[0,i]:=0.0;
    l[i,0]:=w;
  END;
  FOR i:=n TO 1 BY -1 DO (* n.b. backward loop *)
    DyadicReduction(l[0],l[i],d[0],d[i],0,i,n);
  END;

  FOR i:=1 TO n DO
    theta[i]:=theta[i]+l[0,i]*e;
    d[i]:=d[i]/lambda;
  END;
END LDFilter;

(*-------------------------------------------------------------------------*)
PROCEDURE RestartEstimation(zerotheta:BOOLEAN; VAR ok:BOOLEAN);
VAR i,j:CARDINAL;
BEGIN
  WITH GlobalData.Model DO    (* set model polynomials *)
    IF A.degree+B.degree+1>maxindex THEN
      PrintError('Deg A + Deg B>9');
      ok := FALSE;
      RETURN;
    END;

    FOR i:=A.degree+1 TO mdegree DO  A.coeffs[i] :=0.0;  END;
    FOR i:=B.degree+1 TO mdegree DO  B.coeffs[i] :=0.0;  END;

    A.coeffs[0] := 1.0;

    IF zerotheta THEN  (* default settings of A and B *)
      FOR i:=1 TO mdegree DO  A.coeffs[i] :=0.0;  END;
      CASE A.degree OF
        1:  A.coeffs[1] := -0.5|
        2:  A.coeffs[1] := -1.5; A.coeffs[2] := 0.7;
      ELSE  A.coeffs[1] := -2.0; A.coeffs[2] := 1.4; A.coeffs[3] := 0.4;
      END;
      FOR i:=0 TO B.degree DO  B.coeffs[i] := 2.0;  END;
    END;
```

```
    WITH EstimateParameters DO  (* transfer parameters from model *)
      Theta[0] :=0.0;
      FOR i:=1 TO A.degree DO
        Theta[i] := A.coeffs[i];
      END;
      FOR i:=0 TO B.degree DO
        Theta[i+A.degree+1] := B.coeffs[i];
      END;
      N:=A.degree+B.degree+1;
      FOR i:=N+2 TO maxindex DO
        Theta[i] := 0.0;
      END;

      FOR i:=0 TO maxindex DO
        FOR j:=0 TO maxindex DO
          L[i,j] :=0.0;
        END;
        L[i,i] :=1.0;
        D[i]   := GlobalData.EstimData.D0;
        Phi[i] := 0.0;
      END;
    END;
  END;

  WITH ufilterstate DO x1:=0.0; x2:=0.0; END;
  WITH yfilterstate DO x1:=0.0; x2:=0.0; END;

  uf1 := 0.0;  uf2 := 0.0;  uf3 := 0.0;

  GlobalData.EstimData.EstPurpose:='MODELPOLY';
END RestartEstimation;

(* ------------------------------------------------------------------------ *)
PROCEDURE InitRLS;
VAR i,j : CARDINAL;
    ok  : BOOLEAN;
BEGIN
  RestartEstimation(TRUE,ok);

  WITH ufilterstate DO x1:=0.0; x2:=0.0; END;
  WITH yfilterstate DO x1:=0.0; x2:=0.0; END;

  uf1 := 0.0;  uf2 := 0.0;  uf3 := 0.0;
END InitRLS;

(* ------------------------------------------------------------------------ *)
PROCEDURE DORLS(VAR theta,d:col; VAR l:matr; phi:col; n:CARDINAL);
BEGIN
  LDFilter(theta,d,l,phi,GlobalData.EstimData.Lambda,n);
END DORLS;

(* ------------------------------------------------------------------------ *)
PROCEDURE Estimation(u,y:REAL);
VAR i      : CARDINAL;
    uf0,yf0: REAL;
BEGIN
  Filter2( u, ufilterstate, uf0);
  Filter2( y, yfilterstate, yf0);

  WITH EstimateParameters DO
    Phi[0]:=yf0;
    IF GlobalData.Adapt THEN
      DORLS(Theta,D,L,Phi,N);
    END;
    WITH GlobalData.Model DO
      (* Update regressors (phi-vector) and transfer parameters to model *)
      FOR i:=N TO 2 BY -1 DO
        Phi[i]:=Phi[i-1];
      END;
      Phi[1]          := -yf0;
      CASE delay OF
        1: Phi[A.degree+1] := uf0|
        2: Phi[A.degree+1] := uf1|
        3: Phi[A.degree+1] := uf2|
        4: Phi[A.degree+1] := uf3;
      END;
      IF GlobalData.Adapt THEN
        FOR i:=1 TO A.degree DO
          A.coeffs[i] := Theta[i];
        END;
        FOR i:=0 TO B.degree DO
          B.coeffs[i] := Theta[i+A.degree+1];
        END;
      END;
    END;
  END;
  uf3 := uf2;
  uf2 := uf1;
  uf1 := uf0;
END Estimation;

END RLS.
\032
```