



LUND UNIVERSITY

Implementation of an Adaptive Friction Compensation

Braun, Konrad

1985

Document Version:

Publisher's PDF, also known as Version of record

[Link to publication](#)

Citation for published version (APA):

Braun, K. (1985). *Implementation of an Adaptive Friction Compensation*. (Technical Reports TFRT-7304). Department of Automatic Control, Lund Institute of Technology (LTH).

Total number of authors:

1

General rights

Unless other specific re-use rights are stated the following general rights apply:

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: <https://creativecommons.org/licenses/>

Take down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

CODEN: LUTFD2/(TFRT-7304/1-61/(1985)

Implementation of an Adaptive Friction Compensation

Konrad Braun

Department of Automatic Control
Lund Institute of Technology
May 1985

Department of Automatic Control Lund Institute of Technology P.O. Box 118 S-221 00 Lund Sweden		Document name Report	
		Date of issue May 1985	
		Document Number CODEN: LUTFD2/(TFRT-7304)/1-61/(1985)	
		Supervisor 	
Author(s) Konrad Braun		Sponsoring organisation 	
Title and subtitle Implementation of an adaptive friction compensation			
Abstract This report present an implementation of an adaptive control algorithm, which allows to compensate the friction of a DC motor. An IBM Personal Computer with the DOS operating system was used. Two issues had to be added to use this computer for a real-time application: a scheduler and driver routines for the analog and digital 1/0 board. A Pascal program was written to perform experiments with this adaptive algorithm and to compare it with conventional regulators. Some measurements show the advantages of the adaptive algorithm compared with a controller without compensation and a controller with fixed compensation.			
Key words			
Classification system and/or index terms (if any)			
Supplementary bibliographical information			
ISSN and key title		ISBN	
Language English	Number of pages 61	Recipient's notes	
Security classification			

I CONTENTS

Chapter	Page	
I		CONTENTS
II		SYMBOLS
1	1	INTRODUCTION
2	1	PROBLEM
3	4	STARTING-POINT
4	5	REAL-TIME SCHEDULER
4.1	5	Description of the Scheduler
4.2	7	Usage
4.3	8	Problems with the Scheduler
5	8	ANALOG AND DIGITAL INPUT / OUTPUT
5.1	8	Description of the Driver Routines
5.2	9	Usage
6	10	THE PROGRAM AFRICO
7	13	RESULTS
7.1	13	Open-Loop System
7.2	14	Closed-Loop System without Compensation
7.3	15	Closed-Loop System with fixed Compensation
7.4	17	Closed-Loop System with adaptive Compensation
8	18	CONCLUSIONS
9	20	REFERENCES
		APPENDICES

II SYMBOLS

a	Parameter
$\tilde{a}$	Estimated parameter
b	Parameter
$\tilde{b}$	Estimated parameter
c	Constant
e	Estimation error
r	Reference signal
t	Time
T	Torque
u	Control signal
y	Output signal
α	Parameter
δ	Estimated parameter
τ	Time constant
ω	Angular velocity

1 INTRODUCTION

Whenever possible one tries to design a linear controller because the theory is well-known and easy to use. Nonlinearities in the plant are often responsible for poor performance of linear controllers.

If the nonlinearities of some processes are known, they can be compensated. For time-constant plants a fixed compensation will yield fairly good results, but if the plant is time-varying or the nonlinearities are not exactly known the compensation should be adaptive.

An adaptive compensation, as considered in this report, consists of two parts:

- estimation of the parameters of the nonlinear characteristic
- compensation with help of the estimated parameters

2 PROBLEM

The considered plant is a DC motor as shown in Fig. 2.1. The motor is torque or current controlled but with a tachometer feedback it can be emulated like a voltage controlled DC motor.

The motor drives an inertia. Between motor and load is a gear, which transmits the speed to lower values and increases the torque. The angular velocity is measured by a tachometer.

Friction occurs in the bearings. In general the friction is a nonlinear function of the velocity and its characteristic can e.g. look like the ones in Fig. 2.2 a+b.

If mainly the moving system is considered (that is not the start - stop operation) the characteristic can be simplified as sketched in Fig. 2.2 c.

The model of the motor can be assumed as a first order model with a nonlinear feedback due to the friction. The input signal $u(t)$ is a voltage and the output signal $y(t)$ the speed of the motor. The block diagram of the continuous system is shown in Fig. 2.3.

To control the system with a computer the discrete time model should be known. This is derived in [1 p. 5]. The corresponding discrete time model has the following form:

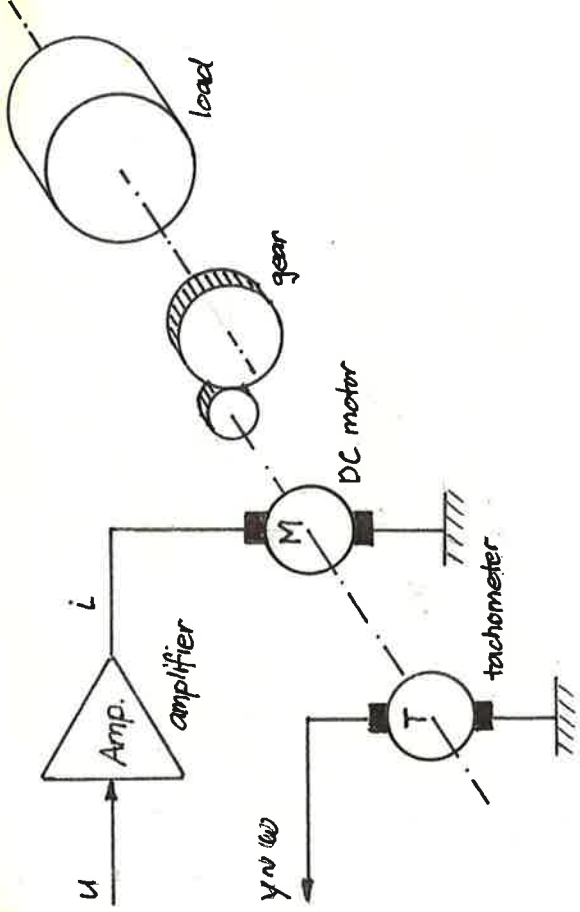


Fig. 2.1: Plant

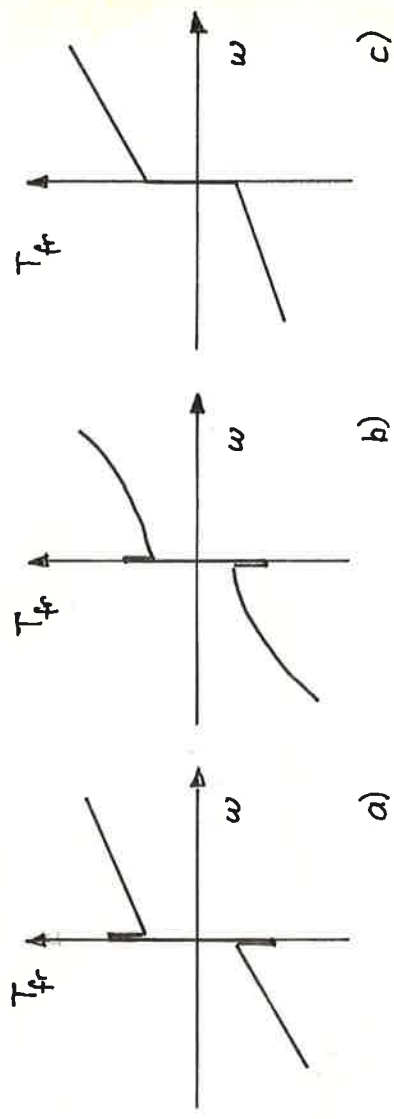


Fig. 2.2: Friction characteristics

$$y(t) = \begin{cases} (a + a_1) \cdot y(t) + b \cdot u(t) - \delta_1 & \text{if } y(t) > 0 \\ (a + a_2) \cdot y(t) + b \cdot u(t) - \delta_2 & \text{if } y(t) < 0 \end{cases}$$

The parameters a and b correspond to the linear model of the motor without friction whereas the parameters a_1 , a_2 , δ_1 , δ_2 are referred to the friction. Because in general the process parameter are not exactly known the discrete model is divided into two parts one is an approximation and the other includes the deviation to the real process.

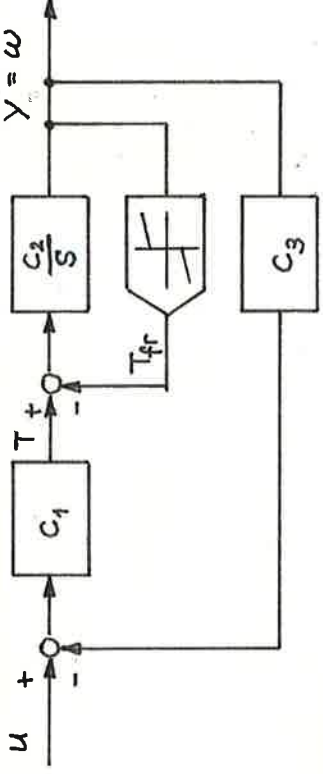


Fig. 2.3: Block diagram of the continuous system

$$y(t) = \begin{cases} (a + \tilde{a}1) \cdot y(t) + (b + \tilde{b}1) \cdot u(t) - \delta 1 & \text{if } y(t) > 0 \\ (a + \tilde{a}2) \cdot y(t) + (b + \tilde{b}2) \cdot u(t) - \delta 2 & \text{if } y(t) < 0 \end{cases}$$

The friction affects the parameters $\tilde{a}1, \delta 1, \tilde{a}2, \delta 2$ but they include also the deviation to the real parameter values of a . The parameters $\tilde{b}1, \tilde{b}2$ are introduced to correct the parameter b , since it is not exactly known.

The general idea of the adaptive compensation is the following one:

The unknown parameters are estimated by a recursive least square algorithm. The parameters $\tilde{a}1, \delta 1, \tilde{a}2, \delta 2$ are then used to compensate the friction. There are two sets of parameters to estimate. One set is estimated when the output y is greater than zero and the other one when y is less than zero. This allows to compensate an asymmetric friction characteristic. A description of this algorithm can be found in [8 p. 324].

A fixed linear controller is used. It is designed with the pole placement algorithm to perform a desired - input output characteristic [8 p. 221]. For this design the parameters a and b are used.

The compensation is done in the following way:

$$u = u_1 + u_c \quad \text{where} \quad \begin{array}{ll} u_1 : & \text{control signal from the linear controller} \\ u_c : & \text{compensation term} \end{array}$$

$$u_c = \frac{-1}{b + \tilde{b}n} (an \cdot y(t) - \delta n) \quad \begin{cases} n = 1 & \text{if } y(t) > 0 \\ n = 2 & \text{if } y(t) < 0 \end{cases}$$

This compensation does not only compensate the friction, but also the deviation of

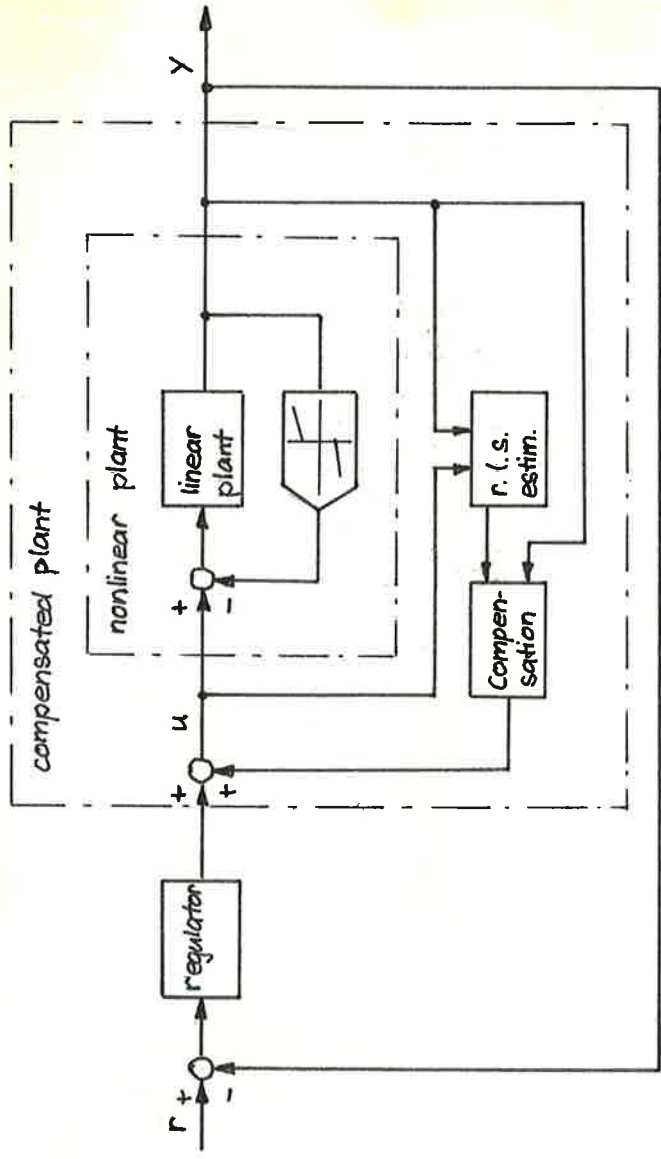


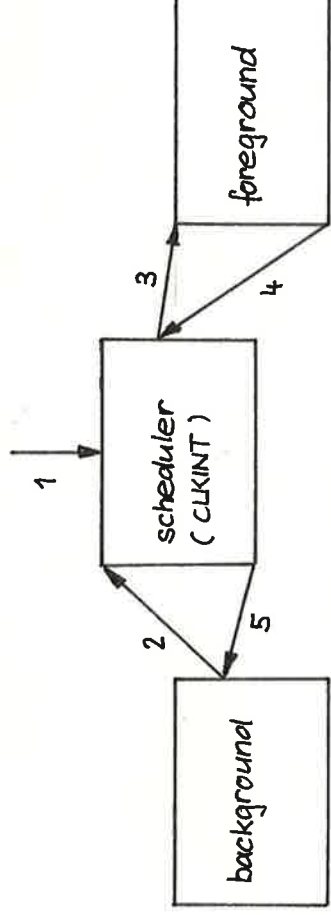
Fig. 2.4: Principle of the adaptive friction compensation

the real system to the approximated linear model described by the parameters a and b . Thus in case of convergence of the estimation the system behaves like a controlled linear system with the parameter values of the approximation. For a more detailed description of this adaptive compensation refer to [1]. The principle of this compensation is summarized in Fig. 2.4.

3 STARTING - POINT

The starting point of this project was the following one:

- The theoretical work was done as described in [1].
- A plant was available as shown in Fig. 2.1.
- An IBM PC could be used to implement the control algorithm. Because the DOS operating system is not designed for real-time applications some additional work had to be done. In the IBM PC a digital analog I/O board was available, but it could not be used with Pascal. Therefore some driver



- 1 clock interrupt
- 2 the foreground is interrupted by the interrupt routine
- 3 if the sampling period is passed the interrupt routine calls the foreground otherwise step 5 is performed directly
- 4 after execution of the foreground the program jumps back to the interrupt routine
- 5 finally the background is resumed at the point where it was interrupted

Fig. 4.1: Principle of the scheduler

routines had to be written.

- The MetaWINDOW package [5] could be used for graphic output.

4 REAL-TIME SCHEDULER

4.1 Description of the Scheduler

A normal computer program written in Pascal is performed sequentially. For real-time applications one should have the possibility to change from one task to another depending on how high the priority of these tasks are. For this implementation only a very simple scheduler is needed. It is sufficient to have a foreground - background system. In the foreground the task is implemented, which has to be performed each sampling interval. In the background some additional work can be done, which is not time critical. Normally this will be the man-machine communication. The principle of this scheduler is shown in Fig. 4.1. The scheduler must secure that the state of the background is saved after the time interrupt occurred, so that the execution of the background can be properly resumed.

The time interrupts can be triggered by an external or internal clock. For simplicity the second solution was chosen. An internal clock in the IBM causes an interrupt each 0.055 sec. After this a routine is called which starting address is stored in the interrupt vector table (s. [6 p. 3-3] Normally this routine is just the return from interrupt statement (RTI). But if another starting address is loaded in the interrupt vector table a user supplied routine can be executed after each time interrupt.

To change the interrupt vector an assembler routine SCHEDULE was written. Also the interrupt routine CLKINT, which is called each time tick is written in assembly language.

The main program of the Pascal program acts as background. It is typically a loop doing some administrative work or just waiting for the time interrupt to be interrupted. The foreground is implemented as a procedure which is called by the time interrupt routine after the sample time is passed.

One of the main problem is to handle common data of the foreground and background. If the background is changing a variable and it is interrupted during this action the foreground will get a an erroneous value accessing the same variable.

This problem can be avoided by using two sets of data. In the background one changes one of the sets and as soon as this is completed a flag signals that these data can be used. If the foreground encounters that the flag is set, it copies the new data to its own data area. The principle is shown in Fig. 4.2.

This solution is a very simple one, but it has the disadvantage that the new data is not available at the first call of the foreground, because the data is copied at the end of the foreground procedure. This disadvantage could be removed by copying at the beginning of the procedure, but this should only be done if there are few common data, otherwise the delay gets to long.

A similar scheduler is described in [2]. There is one difference to mention. In that implementation the address of the global variable with the numbers of time ticks between two calls of the foreground is passed to the scheduler. This makes it possible to change the sampling interval from the background and the foreground. In this implementation the value is passed, and therefore the sampling interval can only be changed by the background with a new call of the procedure SCHEDULE.

```

program example(input,output);

type
  comdat = record
    begin
      ..
      xy: xytype
      ..
    end;

var
  cdforegr, cdbackgr: comdat;
  cdflag,stopforegr: boolean;

procedure schedule(fg [public]: procedure, ticks: integer); extern;
procedure endsched; extern;
procedure foreground [public];

begin
  task;
  if cdflag then
    begin
      cdforegr:= cdbackgr;
      cdflag:= false;
    end;
  end;

begin
  ..
  schedule(foreground, ticks);
  repeat
    ..
    cdbackgr.xy:= newvalue;
    cdflag:= true;
  ..
  until stopforegr;
  ..
  endsched;
  ..
end.

```

Fig. 4.2: Example with the scheduler

4.2 Usage

The two assembler routines SCHEDULE and ENDSCHED must be declared external in the Pascal program as shown in Fig. 4.2. The first call of SCHEDULE will save the initial interrupt vector and the foreground - background system is started. By a subsequent call of the procedure SCHEDULE the foreground routine or the sampling interval may be changed.

The foreground - background system must be stopped by a call of procedure ENDSCHED. This routine will change the interrupt vector to its initial value.

The listing of the assembler code is added in Appendix A.

4.3 Problems with the Scheduler

A problem was encountered using the scheduler. It was not possible to use the write, writeln and read, readln procedures in connection with the digital and analog I/O board and the driver routines described in Chap. 5. When this was done the I/O board did not work properly that is either it did not get ready or the results were completely useless.

5. ANALOG AND DIGITAL INPUT / OUTPUT

5.1 Description of the Driver Routines

The analog signals of the process must be converted to digital numbers before they can be used in the computer. Conversely the computed output signals are digital numbers which must be converted to analog signals. In the IBM PC this is done with help of a single board analog and digital I/O system [3]. There are 16 channels for analog inputs and two channels for analog outputs available.

To use this board with Pascal some driver must be written. The MICROSOFT Pascal [4] allows to use units which can be compiled separately from the main program. Thus the three driver routines are collected in a unit. This unit uses four external routines which are written in assembler. This routines transfer commands and data between the Pascal program and the I/O board. The object code of the unit and the assembler routines must be added when linking the program. The listing of the assembly code is shown in Appendix B.

The three Pascal routines in the unit ADDAB are briefly described below.

Procedure INITADDA

A call of this procedure initializes the I/O board. All the activities are stopped and the two analog outputs are set to -10 Volts.

Function ADIN

This function yields the values of the specified analog input channel. There are 16 channels (0 to 15).

The range of the analog input may be between -10 and 10 volts and this voltages

```

{$include: 'addab.int'}

program addatest(input,output);
uses addab(adin, daout, initadda);

var
  val: real;

begin
  initadda;
  val:= adin(0);
  daout(0,val);
end.

```

Fig. 5.1: Program with analog input and output

are converted to real numbers in the range -1.0 to 1.0. The resolution is 12 bits, which corresponds to 5 mV for the range of -10 to 10 Volts. A single A/D conversion takes about 3 ms.

Procedure DAOUT

With this procedure one of the two analog output channels (0 or 1) may be set. The real parameter outval must be in the range -1.0 to 1.0 and this number is converted to an output voltage within the limits -10 and 10 Volts. The analog output channels have also a resolution of 12 bits. A D/A conversion takes about 3 ms.

5.2 Usage

A unit in MICROSOFT Pascal consists of two parts, that is the interface and the implementation. The interface must be included to at the very beginning of the source file to make sure that the routines are known when compiling the program.

The object files of the unit and the one of the assembly routines must be added when linking the program.

An example of a short program using the I/O board is shown in Fig. 5.1.

6 THE PROGRAM AFRICO

The Program for this adaptive friction compensation should fulfil the following specifications:

- The man - machine interface should be interactive and therefore easy to use.
- After an experiment the main variables should be graphically displayed.
- The minimal sampling rate of 0.055 sec should be possible to use. This means, that the foreground procedure with the control and estimation task should not take more time then 55 ms.
- The program should allow to make different experiments:
 - * controller without friction compensation
 - * controller with fixed friction compensation
 - * controller with adaptive friction compensation

The program is divided into four parts:

1. Initialization of the program
2. Modification of the parameters
3. Experiment
4. Evaluation of the experiment

The possible flows through the program are shown in Fig. 6.1.

Initialization of an Experiment

In this first part the I/O board and the graphic system is initialized. Then the default values are read from a specified file. Thus one can have different files with default values for special experiments. This file should be of the form like the one shown in Appendix C.

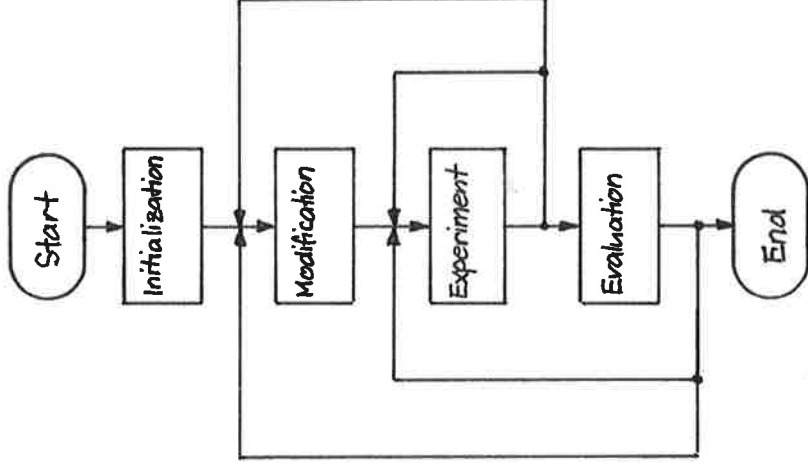


Fig. 6.1: Possible flows through the program AFRICO

Modification of the Parameters

In this part the parameters can be changed interactively. The menu for the modifications is added in Appendix D.

Experiment

This is the part where the foreground - background system is running. The background checks continuously whether the key F1 is pressed. Striking this key stops the controller.

Evaluation of the Experiment

During the experiment the traces of some variables are stored. These traces can be graphed in this part. The menu to choose between these variables is shown in

Appendix D.

The foreground has the following sequence:

1. analog input of the reference signal $r(t)$ and the output signal $y(t)$
2. computation of the control signal $u(t)$ with help of the estimated parameters from the previous call of the foreground
3. analog output of the control signal $u(t)$
4. recursive least square algorithm
5. store the variables, which may be plotted later
6. preparation of the variables used for the compensation at the next call of the foreground
7. update of the variables

This sequence has the advantage, that the delay between the call of the foreground and the output of the control signal can be neglected. To use the new estimates for the compensation the estimation has to be done before the computation of the output signal $u(t)$. But since the estimation needs the most computing time in the foreground, the delay would increase. Since the controller includes integral action, the windup problem had to be considered, and precautions were taken to avoid this phenomena.

The listing of this program is added in Appendix C.

To estimate initial values for the program AFRICO the program IDENT was written. In the main parts it is equivalent to the program AFRICO, but in the foreground the controller is omitted. The input signal is plugged in directly into the process and the parameters $\tilde{a}_1, \tilde{b}_1, \delta_1, \tilde{a}_2, \tilde{b}_2, \delta_2$ are estimated. Thus the real values of the discrete time model can be determined:

$$y(t) = \begin{cases} a1_r \cdot y(t) + b1_r \cdot u(t) - \delta1_r & \text{if } y(t) > 0 \\ a2_r \cdot y(t) + b2_r \cdot u(t) - \delta2_r & \text{if } y(t) < 0 \end{cases}$$

where

$$\begin{aligned} a1_r &= a + a\tilde{1}; & b1_r &= b + b\tilde{1}; & \delta1_r &= \delta1 \\ a2_r &= a + a\tilde{2}; & b2_r &= b + b\tilde{2}; & \delta2_r &= \delta2 \end{aligned}$$

A listing of the source code is added in Appendix E.

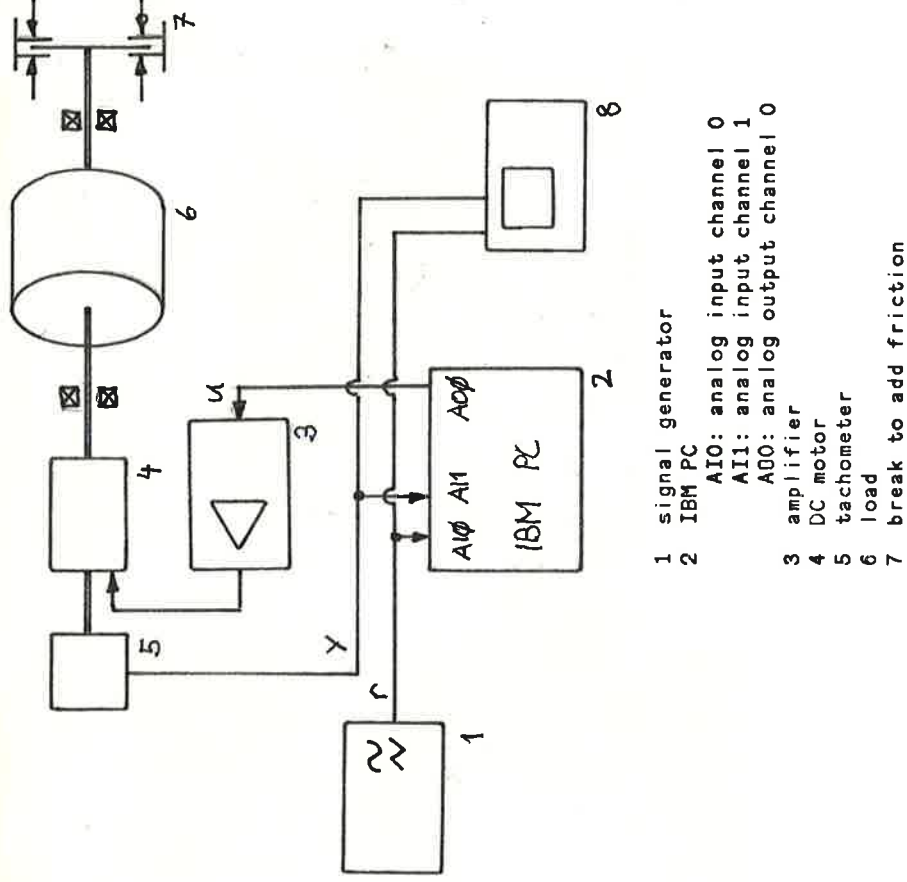


Fig. 7.1: Arrangement of the devices

7 RESULTS

The arrangement of the devices used to perform the following experiments is shown in Fig. 7.1.

7.1 Open-Loop System

To get starting values for the controller design the step response of the open-loop system was determined (s. Fig. 7.2). With this measurement an approximation of the parameters a and b can be calculated.

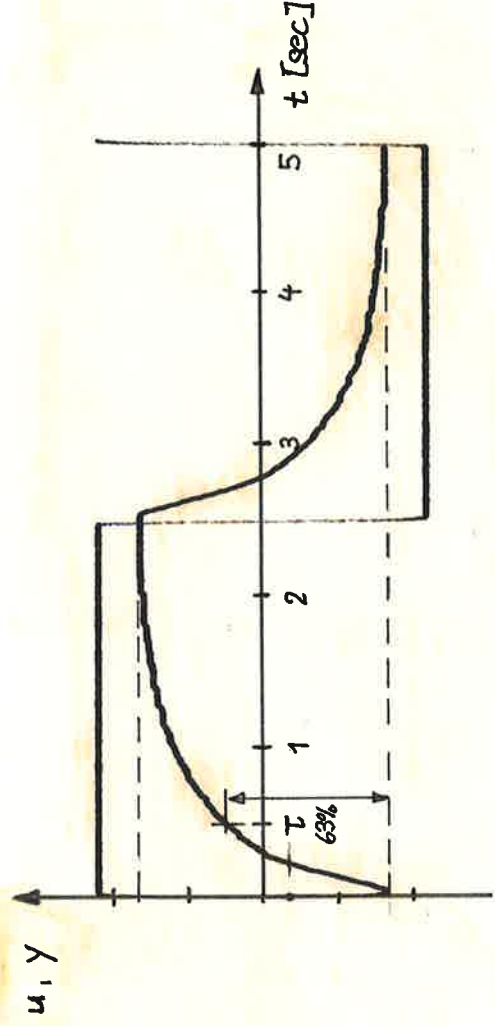


Fig. 7.2: Step response of the open-loop system

The linear approximation of the continuous time system is:

$$G(s) = \frac{\alpha}{\tau \cdot s + 1} \quad \text{where} \quad \alpha = 1.3 \\ \tau = 0.5 \text{ sec}$$

thus the discrete time system is

$$H(z) = \frac{b}{z + a} \quad \text{with} \quad a = -e^{-T/\tau} = -e^{-0.055/0.5} = -0.9 \\ b = \alpha \cdot (1 - e^{-T/\tau}) = 1.3$$

7.2 Closed-Loop System without Compensation

The approximation was used to design a controller for the closed-loop system without compensation. Two controllers were used to perform tests. They are determined by the following parameters:

	controller 1	controller 2
closed-loop polynomial:		
$A_m(z) = z - a_m$; $a_m =$	0.9	0.75
observer polynomial:		
$A_o(z) = z - a_o$; $a_o =$	0.6	0.6

The result of the step response with controller 1 and without compensation is shown in Fig. 7.3. The sampling rate is 0.055 sec.

The trace in Fig. 7.3 shows, that the step response is not completely smooth as it should be with a linear system. Especially in the cross-over point a bend can be

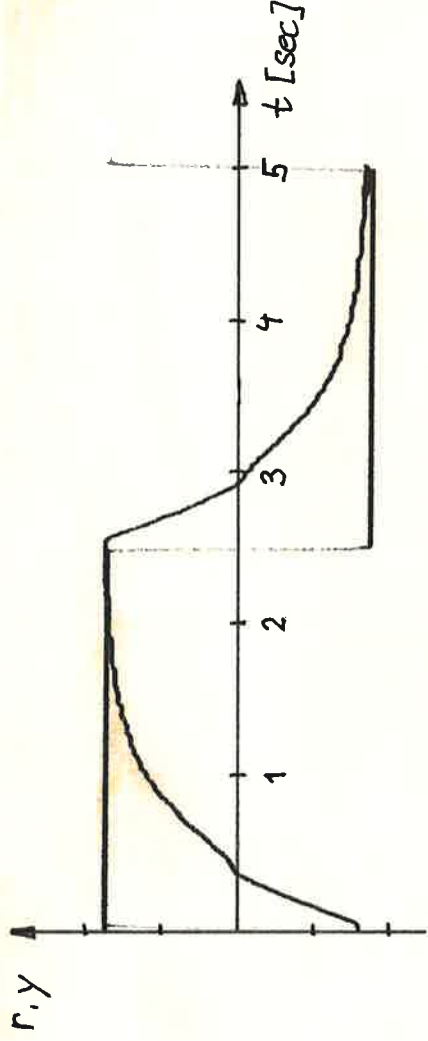


Fig. 7.3: Closed-loop system without compensation

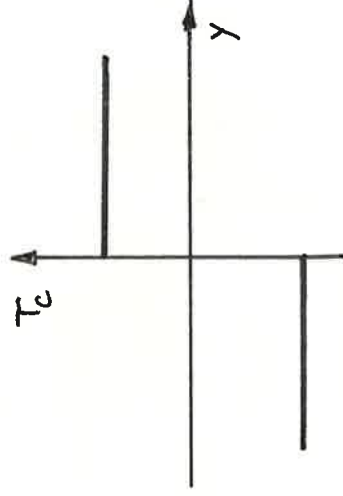


Fig. 7.4: Fixed compensation

seen. This asymmetric behaviour is due to the asymmetric friction.

The static error vanishes because the controller includes an integral action.

7.3 Closed-Loop System with fixed Compensation

To improve the behaviour of the system the nonlinearity can be compensated with a fixed compensation. This is only done for the constant part of the friction. The torque to compensate the friction is shown in Fig. 7.4. It may be different for positive and negative velocities. An approximation of the parameters for this compensation can be obtained by measuring the moment needed to move the system or by an off-line identification.

The step response with controller 1 is shown in Fig. 7.5. It is obvious that the

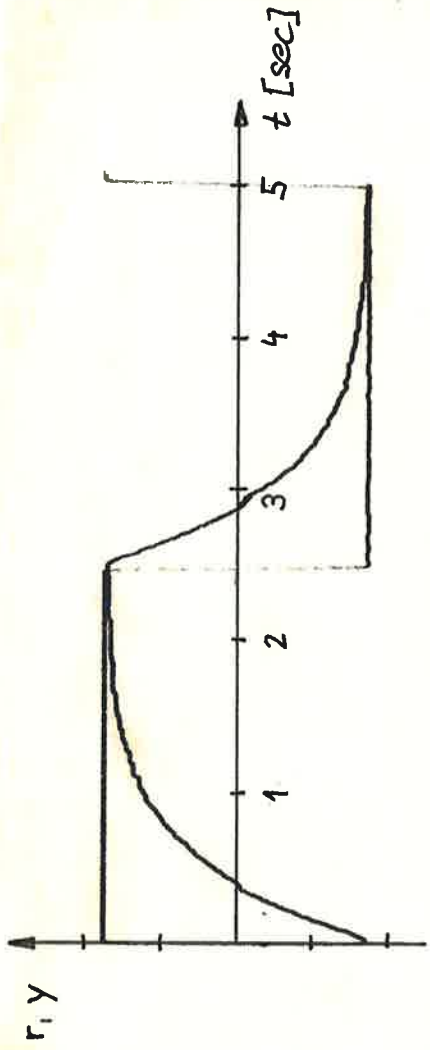


Fig. 7.5: Step response with fixed compensation

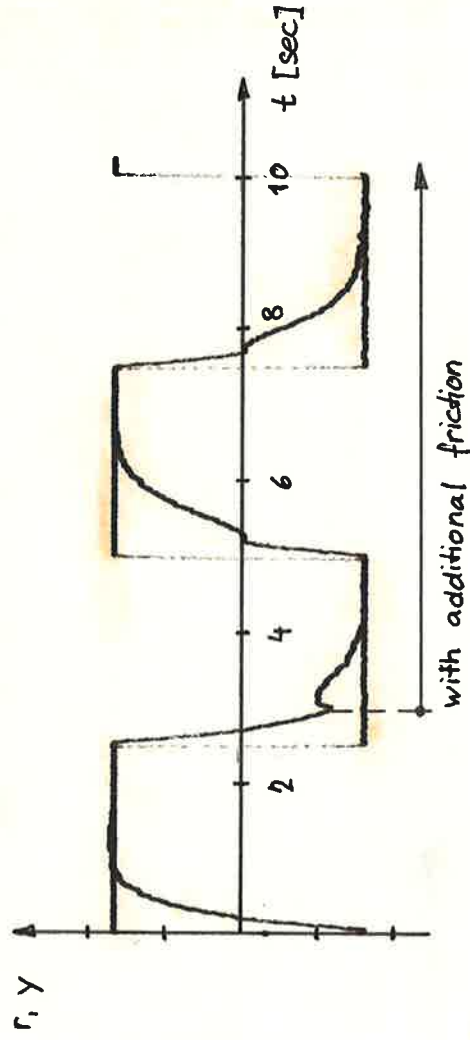


Fig. 7.6: Fixed compensation with changing friction

bend in the cross-over point became smaller.

This compensation yields fairly good results if the viscous friction is only small and the constant part of the friction does not change.

Fig. 7.6 shows what happens, if the friction changes. A break increases the friction three times. The step response after this change is different and dependent on the friction. In this example the controller 2 was used.

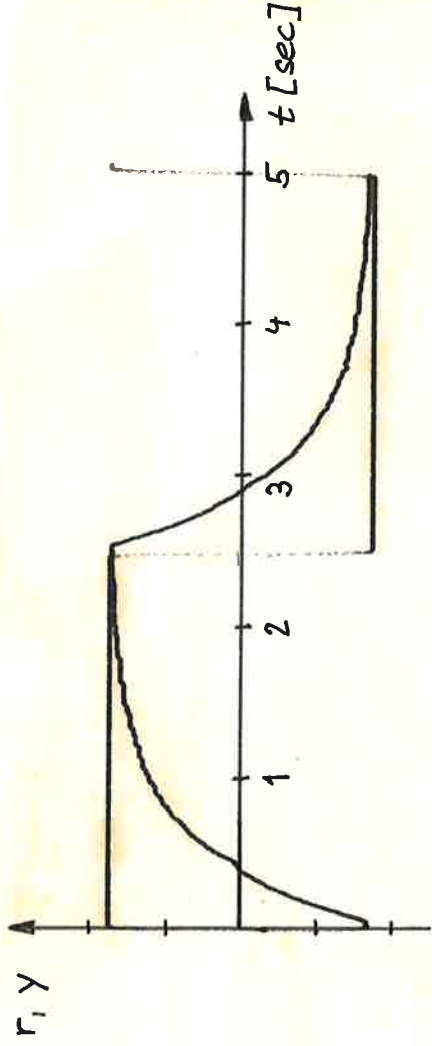


Fig. 7.7: Step response of the adaptive system

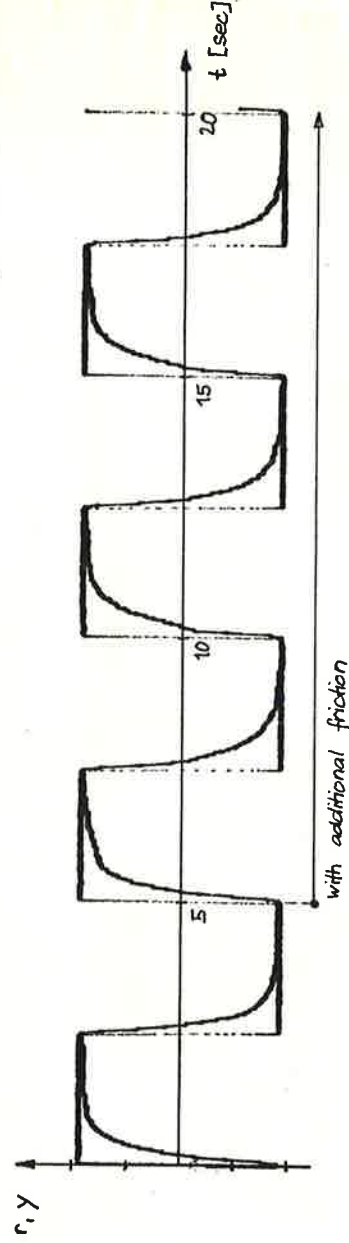


Fig. 7.8: Adaptive compensation with changing friction

7.4 Closed-Loop System with adaptive Compensation

To compensate a slowly varying friction the compensation has to be done adaptive. The step response of the adaptive system with controller 1 is shown in Fig. 7.7. The trace is recorded after a short time of adaptation.

Fig. 7.8 shows the step response with a change of the friction using controller 2. It can easily be seen, that the adaptation is working well. 10 sec after adding the break the step response has nearly the same shape as before the changing of the friction. The time needed for the adaptation is a function of the richness of the input signal and the forgetting factor.

The advantage of the adaptive system can be noticed even more obviously with a ramp as input signal. Fig. 7.9 shows a comparison of a fixed and an adaptive compensation.

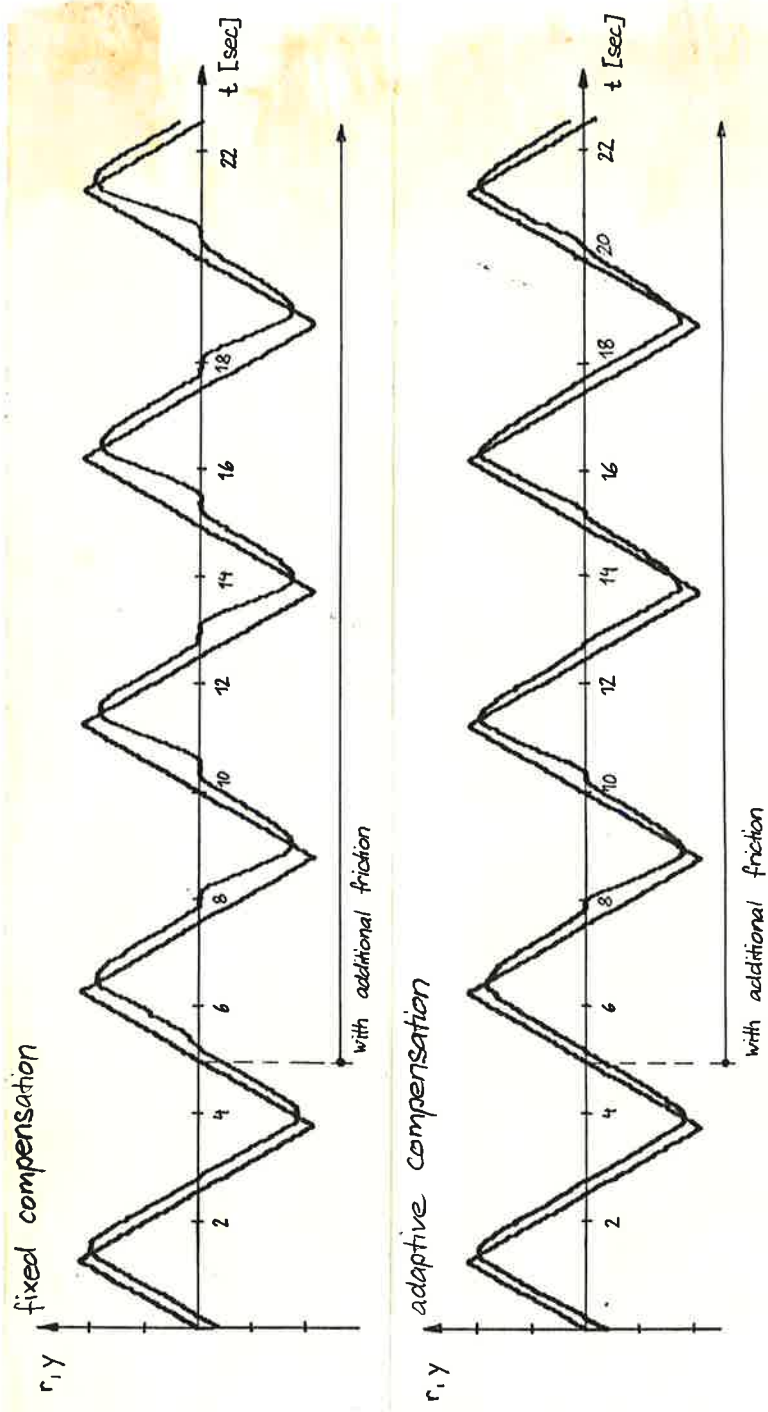


Fig. 7.9: Comparison of fixed and adaptive compensation

If the fixed compensation uses the right values, the ramp is quite smooth. When the friction is increased, the output signal is deformed.

In the adaptive system the output signal is also deformed first, but already after 5 sec one can see that the signal recovers. After some more periods there is no difference of the trace compared with the one before adding the friction.

8 CONCLUSIONS

This real-time implementation makes use of a microcomputer without a special real-time operating system. This causes some problems, because one has to know much more about the hardware, than for the normal usage. There are still some points, which could be solved in a more elegant way:

- When the scheduler is active with analog input and output in the foreground, it is impossible to do input or output from the keyboard resp. screen.

A solution should be found to make input and output possible in order to perform modifications, while the real-time activities are running.

- By using an external time interrupt with more time ticks per second the minimal sample interval could be diminished. Thus the scheduler could also be used for faster processes.
- The analog input/output board is rather slow.
- It would be an advantage to be able to make screencopies of the graphics. In principle this should be possible as described in [5 p. 91].

The results presented in Chap. 7 show the advantage of the adaptive friction compensation. The control algorithm can not only track changing friction, but also variation in the dynamics of the system e.g. additional load or change of the electrical part of the motor.

Some care has to be taken for control of small speed or stillstand. If the velocity changes from one direction to the other, the set of parameters for the compensation is changed. If the estimated parameters are not exactly the right ones or the system is disturbed by noise, chattering may occur.

This adaptive friction compensation can be used for DC motor systems, where a high performance under time-varying conditions is demanded. Further the estimation algorithm could also be used to watch over a DC motor system. It can be detected, when the friction exceeds critical values. If this happens, there is probably something wrong in the system e.g. a bearing is worn out or something is inhibited.

9 REFERENCES

- [1] Canudas de Witt, C.
Adaptive Friction Compensation in DC Motors
Report, Lund Institute of Technology, 1985
- [2] Mattsson, S. E.
A simple Real-Time Scheduler
Report, Lund Institute of Technology, 1978
- [3] DT2801 A single board digital analog I/O system
Data Translation, 1983
- [4] MICROSOFT Pascal
Users Guide and Reference Manual
Microsoft, 1983
- [5] MetaWINDOW
Users Guide for the IBM PC
- [6] IBM PC
Technical Reference
- [7] Norton, P.
Inside the IBM PC
Robert J. Brady Co, 1983
- [8] Åström, K. J.,
Computer Controlled Systems
Prentice-Hall, Inc. 1984

APPENDICES

- A:** Source Code of the Scheduler
- B:** Source Code of the analog Input/Output Routines
- C:** Source Code of the Program AFRICO
- D:** Menus of the Program AFRICO
- E:** Source Code of the Program IDENT

COMMENT *

PROLOG - SCHEDULER

PURPOSE:

This assembler program can be used together with a MICROSOFT-Pascal program in order to get a simple real-time scheduler.

The Pascal program is divided into the foreground (FG) and the background (BG). The main program acts as background whereas the foreground is declared as a procedure which is called each sampling time.

The assembler program handles the initialization, the time interrupts, calls of the foreground routine at each sampling interval and the termination of the real-time scheduler.

The time interrupts are caused by a routine in the ROM which causes an interrupt ICH. This is done 18.2 times a second or each 0.055 sec. The interrupt handler will invoke a user routine.

Therefore the address of the starting point must be placed in the table (address 0000:0070).

USAGE:

Two external procedures must be declared in the Pascal program:

```
procedure SCHEDULE(procedure FG [public]; TICKS: integer) extern;

procedure ENDSCHED; extern;
```

With a call of the procedure SCHEDULE the scheduler is initialized. Passed parameters are the foreground procedure (which must be declared 'public') and the number of time ticks between two calls of the foreground (ticks = sample time [sec] multiplied by 18.2). The foreground procedure or the sample time can be changed by recall the procedure SCHEDULE.

The scheduler should be stopped by a call of the procedure ENDSCHED. In this routine the interrupt vector of interrupt ICH is set to its initial value.

DATE:

May 14 1985

AUTHOR:

Konrad Braun *

DATAS	SEGMENT	PUBLIC
DSPAS	DW	?
FLAGIP	DW	0
INTPTR	DD	?
ADDRFG	DD	?
TICKS	DW	?
LAG	DW	0
INCT	DW	?
ACTIVE	DW	0

```
; storage for Pascal CS
; flag variable to indicate
; that initial interrupt
; pointer is stored
; storage for initial inter-
; rupt pointer
; storage for the address of
; the foreground procedure
; number of ticks between two
; calls of the foreground
; variable to store a request
; variable to count the time
; ticks
; variable to indicate that
```

; the foreground is active

DATAS ENDS

DGROUP GROUP DATAS
ASSUME CS:SCHEDS,DS:DGROUP
ASSUME SS:DGROUP,ES:DGROUP

SCHEDS SEGMENT 'CODE'

SCHEDULE PUBLIC FAR SCHEDULE

PUSH BP
MOV BP,SP
PUSH DS

CLI
MOV AL,01H
OUT 021H,AL

MOV BX,DS
MOV AX,DGROUP
MOV DS,AX
MOV DSPAS,BX
SUB AX,AX
MOV ES,AX
MOV BX,070H
CMP FLAGIP,0
JG IPSET
MOV AX,ES:[BX]
MOV INTPTR,AX
MOV AX,ES:[BX+2]
MOV INTPTR+2,AX
MOV ES:WORD PTR [BX],OFFSET CLKINT

; save BP and DS registers,
; the other registers may
; changed without to affect
; the Pascal program
; disable interrupt

; set DS to DGROUP
; save DS register of the
; Pascal program
; use the extra-segment re-
; gister to set interr. vector

; if FLAGIP > 0 goto IPSET
; else save initial inter-
; rupt vector

ES:[BX+2],CS
INC FLAGIP

IPSET:
MOV BX,[BP+10]
MOV AX,[BX+2]
MOV ADDRFG+2,AX
MOV AX,[BX]
MOV ADDRFG,AX
MOV AX,[BP+6]
MOV TICKS,AX
MOV INCT,1
MOV LAG,0
POP DS
POP BP
MOV AL,0
OUT 021H,AL
STI
RET 6
SCHEDULE ENDP

; set new interrupt vector
; procedure CLKINT
; set FLAGIP

; save code segment of the
; new foreground procedure
; save offset of the new
; foreground procedure
; save number of ticks be-
; tween two calls
; INCT := 1
; LAG := 0

; restore DS and BP

; inable interrupt
; return, clear stack

; disable interrupts

CLKINT PROC FAR
CLI
PUSH BP
MOV BP,SP
PUSH DS

```

PUSH AX
MOV AX,DGROUP
MOV DS,AX
CMP TICKS,0
JG RUN
MOV INCT,1
MOV LAG,0
JMP RETURN

```

RUN:

```

DEC INCT
JG RETURN
MOV AX,TICKS
MOV INCT,AX
INC LAG
CMP ACTIVE,0
JNZ RETURN
INC ACTIVE
PUSH BX
PUSH CX
PUSH DX
PUSH SI
PUSH DI
PUSH ES

```

LOOP1:

```

MOV LAG,0
PUSH DS
MOV AX,DGROUP
MOV ES,AX
MOV DS,DSPAS

```

```

STI
CALL ES:ADDRFG
CLI
POP DS
CMP LAG,0
JG LOOP1
MOV ACTIVE,0
POP ES
POP DI
POP SI
POP DX
POP CX
POP BX

```

RETURN:

```

PUSHF
CALL INTPTR
POP AX
POP DS
POP BP
IRET
ENDP

```

CLKINT

```

PUBLIC ENDSCHED
ENDSCHED PROC FAR
PUSH BP
MOV BP,SP
PUSH DS
CLI
MOV AL,01H

```

```

; save BP,DS, and AX reg.

```

```

; set DS to DGROUP

```

```

; if TICKS > 0 goto RUN
; else INCT :=1
; LAG := 0
; goto RETURN

```

```

; INCT := INCT - 1
; if INCT > 0 goto RETURN
; else
; INCT := TICKS
; LAG := LAG + 1

```

```

; if ACTIVE > 0 goto RETURN
; else
; ACTIVE := ACTIVE + 1

```

```

; save registers

```

```

; clear LAG
; save DS in the stack

```

```

; set ES to DGROUP
; set DS to the data segment
; in the Pascal program
; inable interrupts
; call foreground proc.
; disable interrupts
; restore DS

```

```

; if LAG > 0 loop to LOOP1
; else
; ACTIVE := 0

```

```

; restore registers

```

```

; call initial interrupt
; routine

```

```

; restore AX, DS, BP
; return from interrupt

```

```

; save BP and DS reg.
; disable interrupt

```

```
OUT      021H,AL
MOV      AX,DGROUP
MOV      DS,AX
SUB      AX,AX
MOV      ES,AX
MOV      BX,070H
MOV      AX,INTPTR
MOV      ES:[BX],AX
MOV      AX,INTPTR+2
MOV      ES:[BX+2],AX
MOV      FLAGIP,0
MOV      AL,0
OUT      021H,AL
STI
POP      DS
POP      BP
RET
ENDSCHED ENDP
SCHEDS  ENDS
END
```

```
; set DS to DGROUP
```

```
; ES := 0
```

```
; reset interrupt vector to
; initial value
; clear flagip
```

```
; enable interrupt
```

```
; restor DS and BP
; return
```

ADDAB.INT;1

29-MAY-1985 17:00

Page 1

interface (1);

unit addab(adin,daout,initadda);

function adin(channel: integer): real;

procedure daout(channel: integer; outval: real);

procedure initadda;

begin

end;


```

{$include: 'addab.int'}

implementation of addab;

procedure outcom(com: integer); extern;

procedure outdata(data: integer); extern;

procedure indata(var data: integer); extern;

procedure initio; extern;

function adin;
const cadim=12; gain=0;
var xlow, xhigh: integer;
begin
  if (Channel)=0) and (Channel<=15) then
    begin
      outcom(cadim);
      outdata(gain);
      outdata(channel);
      indata(xlow);
      indata(xhigh);
      adin:= float(xhigh*256+xlow)/2048.0-1.0
    end
  else
    adin:= 0.0;
  end;

procedure daout;
const vmax=0.9995118; cdaim=8;
var x, xhigh, xlow: integer;
begin
  if (channel=0) OR (channel=1) then
    begin
      if outval>vmax then outval:=vmax
      else if outval<-vmax then outval:=-vmax;
      outval:=outval+1.0;
      x:=trunc(outval*2048.0+0.5);
      xhigh:= x div 256;
      xlow:= x mod 256;
      outcom(cdaim);
      outdata(channel);
      outdata(xlow);
      outdata(xhigh)
    end
  end;

procedure initadda;
const command=16#2ed; datareg=16#2ec;
var val: integer;
begin
  initio;
  outcom(1);
end;

begin
end.

```

COMMENT *

PROLOG - IOPROCA

PURPOSE:

These assembler routines are used in MICROSOFT-Pascal as external procedures.

OUTCOM transfers a command to the single board analog and digital i/o system; OUTDATA transfers a data byte to the board and INDATA reads a byte from the board.

The procedure INITIO is used for initialization.

USAGE:

The four external procedures must be declared in the Pascal program in the following way:

```

procedure OUTCOM(COM: integer); extern;

procedure OUTDATA(DATA: integer); extern;

procedure INDATA(var DATA: integer); extern;

procedure INITIO; extern;

```

DATE:

May 10 1985

AUTHOR:

Konrad Braun *

```

STATUS      EQU 02EDH
COMMAND      EQU 02EDH
DATAREG      EQU 02ECH
READY        EQU 04H
DATAINFULL   EQU 02H
READREADY    EQU 05H

```

ASSUME CS:IOPROCS

IOPROCS 'SEGMENT 'CODE'

OUTCOM PUBLIC OUTCOM

PROC FAR

PUSH BP

MOV BP,SP

MOV DX,STATUS

; save BP register
; set base pointer
; set port address

LOOP1:

IN AL,DX

AND AL,READY

JZ LOOP1

MOV AX,[BP+6]

MOV DX,COMMAND

OUT DX,AL

POP BP

RET 2

; read status byte from the board

; loop until ready bit set
; mov command to the AX register
; set port address
; transfere command to port
; restore BP
; return; clear stack

OUTCOM ENDP

```

PUBLIC OUTDATA
PROC FAR
PUSH BP          ; save BP register
MOV BP,SP        ; set base pointer
MOV DX,STATUS    ; set port address

LOOP2:
IN AL,DX         ; read status byte from the board
AND AL,DATAINFULL
JG LOOP2        ; loop until datainfull bit cleared
MOV AX,[BP+6]    ; mov data to AX register
MOV DX,DATAREG   ; set port address
OUT DX,AL        ; transfere data to port
POP BP          ; restore BP register
RET 2           ; return, clear stack
ENDP

OUTDATA

PUBLIC INDATA
PROC FAR
PUSH BP          ; save BP register
MOV BP,SP        ; set base pointer
MOV DX,STATUS    ; set port address

LOOP3:
IN AL,DX         ; read status byte from the board
AND AL,READREADY
JZ LOOP3        ; loop until readready bit set
MOV DI,[BP+6]    ; set DI register
MOV DX,DATAREG   ; set port address
SUB AX,AX        ; clear AX register
IN AL,DX         ; transfere byte from port to AL
MOV [DI],AX      ; mov ax register to the variable
POP BP          ; restore BP
RET 2           ; return, clear stack
ENDP

INDATA

PUBLIC INITIO
PROC FAR
MOV DX,COMMAND   ; set port address
MOV AL,15
OUT DX,AL        ; transfere stop command to board
MOV DX,DATAREG
IN AX,DX         ; read data form the board to clear it
RET
ENDP

INITIO

IOPROCS ENDS
END

```

PROLOG - AFRICO

PURPOSE:

This program implements an adaptive friction compensation. It uses the routine SCHEDULE to get a simple real-time system. There are a foreground and a background. The foreground is called each sampling interval. The control algorithm is programed in the foreground and the man-machine interface is done by the background.

USAGE:

The reference signal from a signal generator should be connected with the analog input channel 0, the output of the process with analog input channel 1. The analog output channel 0 puts out the control signal and should be connected to the process.

The program is expected to be easy to use. After typing the name of the program a file with the default values must be specified.

This file is a text file of the following form (the comments are not needed):

```
1      ; ticks (number of ticks between calls of foregr.)
0.5    ; ouint (interval between two points of the plots)
0.8 0.11 ; a, b (parameters of the discrete time model)
0.99   ; lam (forgetting factor)
0.75   ; am (closed loop parameter)
0.6    ; a0 (observer polynomial)
0.0 0.0 -0.1 ; tef1 (values for the fixed compensation)
0.0 0.0 0.1 ; tef2 (values for the fixed compensation)
1.0 -1.0 ; umax, umin (control signal limitations)
```

The rest of the program is self-explicable.

DATE:

May 28 1985

AUTHOR:

Konrad Braun

```
{ $include: 'addab.int' }
```

```
program test4(input,output,defval);
```

```
uses addab(adin,daout,initadda);
```

```
const n=3;
```

```
      x0= 99;
```

```
      y0= 110;
```

```
      xmax= 500;
```

```
      ymax=150;
```

```

      maxpoint= 200;

type   vec1=   array [1..n] of real;

      mat1=   array [1..n,1..n] of real;

{$include: 'grconst.inc'}

var    defval: text;

      i,j,ticks,actual,strt,pt,maxpt,it,nout      : integer;

      a,b,a0,am,umax,umin,
      r,rb,rbd1,y,yd1,u,ud1,ul,uld1,e,
      t,sampint,tscale,yscale,outint,
      atilf1,btilf1,delf1,atilf2,btilf2,delf2,
      lam,s0,s1,r0,pf      : real;

      phi,te1,te1d1,te2,te2d1,tef1,tef2,
      tec1d1,tec2d1      : vec1;

      p1,p1d1,p2,p2d1      : mat1;

      stopprog,stopplt,startplt,moddone,
      startctrl,stopctrl,stopgraf,plotgrf      : boolean;

      comp      : (none, fix, adptv);

      updf1ag      : (pos, neg, nul);

      graphx,graphy      : array[1..maxpoint] of integer;

      stot,stoy,stou,stor,stoe,
      stoat11,stobtil1,stodel1,
      stoat12,stobtil2,stodel2,
      plotarr      : array[0..maxpoint-1] of real;

      plotvar      : string(20);

      lblx1,lblx2,lbly1,lbly2      : lstring(9);

      evnt      : event;

value  updf1ag:= nul; comp:= fix;

{$include: 'grextrn.inc'}

procedure schedule(procedure fg [public]; ticks: integer); extern;

procedure endsched; extern;

function sign(x: real): integer;

begin
  sign:= ord(x<0.0)-ord(x<0.0)
end;

```

AFRICO.PAS;1

29-MAY-1985 17:00

```
procedure foregnd1 [public];
```

```
var    d1,t1,t2: real;
```

```
    estim: boolean;
```

```
procedure control;
```

```
begin
  rb:= pf*r;
  ul:= uld1+(rb-a0*rbd1-s0*y-s1*yd1)/r0;
  if (y>0.0) or ((y=0.0) and (ul>0.0)) then
  begin
    t1:= tec1d1[1]*y+tec1d1[3]; t2:= b+tec1d1[2];
    u:= (b*ul-t1)/t2;
  end
  else if (y<0.0) or ((y=0.0) and (ul<0.0)) then
  begin
    t1:= tec2d1[1]*y+tec2d1[3]; t2:= b+tec2d1[2];
    u:= (b*ul-t1)/t2;
  end
  else
  begin
    t1:= 0.0; t2:= 0.0;
    u:= 0.0;
  end;
  if u>umax then u:= umax
  else if u<umin then u:= umin;
end;
```

```
procedure rls;
```

```
var    s1: real;
      kd1,v1: vec1;
      m1,m2: mat1;
```

```
procedure vsmul(var v: vec1; var s: real; var res: vec1);
```

```
var i: integer;
```

```
begin
  for i:= 1 to n do res[i]:= v[i]*s;
end;
```

```
procedure vvtmul(var v1,v2: vec1; var res: mat1);
```

```
var i,j: integer;
```

```
begin
  for i:= 1 to n do
    for j:= 1 to n do res[i,j]:= v1[i]*v2[j];
  end;
```

```
procedure vtmul(var v1,v2: vec1; var res: real);
var i: integer;
begin
  res:= 0.0;
  for i:= 1 to n do  res:= res+v1[i]*v2[i];
end;

procedure mvmul(var m: mat1; var v,res: vec1);
var i,j: integer;
begin
  for i:= 1 to n do
    begin
      res[i]:= 0.0;
      for j:= 1 to n do  res[i]:= res[i]+m[i,j]*v[j];
      end
    end;
end;

procedure mmmul(var m1,m2,res: mat1);
var i,j,k: integer;
begin
  for i:= 1 to n do
    begin
      for j:= 1 to n do
        begin
          res[i,j]:= 0.0;
          for k:= 1 to n do  res[i,j]:= res[i,j]+m1[i,k]*m2[k,j];
          end
        end
      end;
end;

procedure emsubdiv(var m: mat1; var s: real; var res: mat1);
var i,j: integer;
begin
  for i:= 1 to n do
    begin
      for j:= 1 to n do
        begin
          if i=j then
            res[i,j]:= (1.0-m[i,j])/s
          else
            res[i,j]:= -m[i,j]/s;
          end
        end
      end;
end;

procedure vvadmul(var v1,v2: vec1; var s: real; var res: vec1);
```

```

var i: integer;

begin
  for i:= 1 to n do
    res[i]:=v1[i]+v2[i]*s;
  end;

  begin
    if yd1>0.0 then
      begin
        updf1:= pos;
        vtmul(phi,te1d1,s1);
        e:= y-s1-a*yd1-b*ud1;
        mvmul(pid1,phi,v1);
        vtmul(phi,v1,s1);
        s1:= 1.0/(lam+s1);
        vsmul(v1,s1,kd1);
        vvadmul(te1d1,kd1,e,te1);
        vvtmul(kd1,phi,m1);
        emsubdiv(m1,lam,m2);
        mmmul(m2,p1d1,p1);
      end
    else
      begin
        updf1:= neg;
        vtmul(phi,te2d1,s1);
        e:= y-s1-a*yd1-b*ud1;
        mvmul(p2d1,phi,v1);
        vtmul(phi,v1,s1);
        s1:= 1.0/(lam+s1);
        vsmul(v1,s1,kd1);
        vvadmul(te2d1,kd1,e,te2);
        vvtmul(kd1,phi,m1);
        emsubdiv(m1,lam,m2);
        mmmul(m2,p2d1,p2);
      end
    end;

  procedure storeval;

  begin
    it:= it+1;
    if (it-1) mod nout = 0 then
      begin
        actual:= (actual+1) mod maxpoint;
        stot[actual]:= (it-1)*samping;
        stoy[actual]:= y*10.0;
        stor[actual]:= r*10.0;
        stou[actual]:= u*10.0;
        stoef[actual]:= e;
        stoatil1[actual]:= te1[1];
        stobtil1[actual]:= te1[2];
        stodel1[actual]:= te1[3];
        stoatil2[actual]:= te2[1];
        stobtil2[actual]:= te2[2];
        stodel2[actual]:= te2[3];
      end
    end;

```


AFRICO.PAS:1

29-MAY-1985 17:00

```
end;
end;

procedure compsel;
begin
  case comp of
    none, fix:
      begin
        tec1d1:= tef1;
        tec2d1:= tef2;
      end;
    adptv:
      begin
        tec1d1:= te1;
        tec2d1:= te2;
      end;
  end
end;

procedure update;
begin
  phi[1]:= y; phi[2]:= u; phi[3]:= 1.0;
  uld1:=(u*t2+t1)/b;
  ud1:= u;          rbd1:= rb;
  yd1:= y;
  case updf1ag of
    pos:
      begin
        te1d1:= te1;
        p1d1:= p1;
        updf1ag:= nul;
      end;
    neg:
      begin
        te2d1:= te2;
        p2d1:= p2;
        updf1ag:= nul;
      end;
    nul:
      begin
        nul;
      end;
  end;
end;

begin
  r:= adin(0); y:= adin(1);
  control;
  daout(0,u);
  estim:= (yd1<>0.0) and (sign(y) + sign(yd1) <> 0);
  if estim then rls;
  storeval;
  compsel;
  update;
end;

procedure setdefval;
```

```

begin
  reset(defval);
  readln(defval,ticks);
  readln(defval,outint);
  readln(defval,a,b);
  readln(defval,lam);
  readln(defval,am);
  readln(defval,a0);
  readln(defval,atilf1,btilf1,delf1);
  readln(defval,atilf2,btilf2,delf2);
  readln(defval,umax,umin);
  close(defval)
end;

```

```

procedure displaypar;

```

```

begin
  writeln;
  writeln('sampling interval' = ',ticks*0.055:10:3');
  writeln('output interval' = ',outint:10:3');
  writeln('a' = ',a:10:3');
  writeln('b' = ',b:10:3');
  writeln('forgetting factor lam' = ',lam:10:3');
  writeln('closed-loop parameter: am' = ',am:10:3');
  writeln('observer polynomial: a0' = ',a0:10:3');
  writeln('values of fixed compensation:');
  writeln(' atilf1 = ',atilf1:10:3,' btilf1 = ',btilf1:10:3,' delf1 = ',
    delf1:10:3);
  writeln(' atilf2 = ',atilf2:10:3,' btilf2 = ',btilf2:10:3,' delf2 = ',
    delf2:10:3);
  writeln('control signal limitations:');
  writeln(' umax = ',umax:10:3,' umin = ',umin:10:3);
  write('friction compensation: comp = ');
  if comp = none then writeln('none')
  else if comp = fix then writeln('fix')
  else writeln('adaptiv');
end;

```

```

procedure menu1;

```

```

var   answer: char;

```

```

begin
  repeat
    writeln;
    writeln('do you want to make any modifications?');
    writeln;
    write('answer [y/n]: ');
    readln(answer);
    until answer in ['y','y','n','N'];
    if answer in ['y','y'] then
      begin
        repeat
          writeln;
          writeln('choose one of the following possibilities:');
          writeln;
          writeln(' a: change sampling interval');

```

```

writeln(' b: change output intervall');
writeln(' c: change model parameter a');
writeln(' d: change model parameter b');
writeln(' e: change forgetting factor');
writeln(' f: change closed loop parameter am');
writeln(' g: change observer polynomial');
writeln(' h: change values of fixed compensation');
writeln(' i: change control signal limitations');
writeln(' j: change friction compensation mode');
writeln;
write('answer: ');
readln(answer);
until answer in ['a'..'j','A'..'J'];
writeln;
case answer of
  'a','A':
    begin
      writeln('sampling intervall = ticks * 0.055 sec = ',ticks*0.055:10:3);
      write('new value: ticks = ');
      readln(ticks);
    end;
  'b','B':
    begin
      writeln('ouput intervall = ',outint:10:3);
      write('new value: ');
      readln(outint);
    end;
  'c','C':
    begin
      writeln('a = ',a:10:3);
      write('new value: ');
      readln(a);
    end;
  'd','D':
    begin
      writeln('b = ',b:10:3);
      write('new value: ');
      readln(b);
    end;
  'e','E':
    begin
      writeln('lam = ',lam:10:3);
      write('new value: ');
      readln(lam);
    end;
  'f','F':
    begin
      writeln('am = ',am:10:3);
      write('new value: ');
      readln(am);
    end;
  'g','G':
    begin
      writeln('a0 = ',a0:10:3);
      write('new value: ');
      readln(a0);
    end;
  'h','H':
    begin

```

C.9

```

writeln('atilf1 = ',atilf1:10:3,'      btilf1 = ',btilf1:10:3,'      delf1 = ',
      delf1:10:3);
write('new values: ');
readln(atilf1,btilf1,delf1);
writeln('atilf2 = ',atilf2:10:3,'      btilf2 = ',btilf2:10:3,'      delf2 = ',
      delf2:10:3);
write('new values: ');
readln(atilf2,btilf2,delf2);
end;
'i','I':
begin
  writeln('umax = ',umax:10:3,'      umin = ',umin:10:3);
  write('new values: ');
  readln(umax,umin);
end;
'j','J':
begin
  write('comp = ');
  if comp = fix then writeln('fix') else writeln('adaptiv');
  repeat
    writeln('new mode:');
    writeln(' 1: without friction compensation');
    writeln(' 2: fixed compensation');
    writeln(' 3: adaptiv compensation');
    writeln;
    write('answer: ');
    readln(answer)
  until answer in ['1','2','3'];
  case answer of
    '1': comp:= none;
    '2': comp:= fix;
    '3': comp:= adptv;
  end;
end;
end;
end;
end
else
begin
  moddone:= true;
end
end;
end;

procedure menu2;
var   answer: char;

begin
  repeat
    writeln;
    writeln('choose one of the following possibilities:');
    writeln;
    writeln('  a: initialize another experiment');
    writeln('  b: initialize plots');
    writeln;
    write('answer: ');
    readln(answer);
  until answer in ['a'..'b','A'..'B'];
  if answer in ['b','B'] then startplt:= true;

```

```

end;

procedure menu3;
var  answer: char;

begin
  repeat
    writeln;
    writeln('choose one of the following possibilities:');
    writeln;
    writeln(' a: plot reference signal r');
    writeln(' b: plot signal y');
    writeln(' c: plot control signal u');
    writeln(' d: plot estimation error e');
    writeln(' e: plot estimated parameter atill');
    writeln(' f: plot estimated parameter btill');
    writeln(' g: plot estimated parameter dell');
    writeln(' h: plot estimated parameter atil2');
    writeln(' i: plot estimated parameter btil2');
    writeln(' j: plot estimated parameter del2');
    writeln(' k: display estimated parameters');
    writeln(' l: initialize another experiment');
    writeln(' m: stop the program');
    writeln;
    write('answer: ');
    readln(answer);
  until answer in ['a'..'m','A'..'M'];
  case answer of
    'a','A':
      begin
        plotarr:= stor;
        plotvar:= 'ref. signal r';
        plotgrf:= true;
      end;
    'b','B':
      begin
        plotarr:= stoy;
        plotvar:= 'output signal y';
        plotgrf:= true;
      end;
    'c','C':
      begin
        plotarr:= stou;
        plotvar:= 'control signal u';
        plotgrf:= true;
      end;
    'd','D':
      begin
        plotarr:= stoe;
        plotvar:= 'estimation error e';
        plotgrf:= true;
      end;
    'e','E':
      begin
        plotarr:= stoatill;
        plotvar:= 'estim. par. atill';
        plotgrf:= true;
      end;
  end;
end;

```

```

end;
'f','f':
begin
  plotarr:= stobtil1;
  plotvar:= 'estim. par. btill  ';
  plotgrf:= true;
end;
'g','g':
begin
  plotarr:= stodel1;
  plotvar:= 'estim. par. dell  ';
  plotgrf:= true;
end;
'h','h':
begin
  plotarr:= stoatil2;
  plotvar:= 'estim. par. atil2  ';
  plotgrf:= true;
end;
'i','i':
begin
  plotarr:= stobtil2;
  plotvar:= 'estim. par. btil2  ';
  plotgrf:= true;
end;
'j','j':
begin
  plotarr:= stodel2;
  plotvar:= 'estim. par. del2  ';
  plotgrf:= true;
end;
'k','k':
begin
  writeln;
  writeln('estimated parameters:');
  writeln;
  writeln(' atil1 = ',te1[1]:10:4,'      btill = ',te1[2]:10:4,'      del1 = ',
    te1[3]:10:4);
  writeln(' atil2 = ',te2[1]:10:4,'      btill2 = ',te2[2]:10:4,'      del2 = ',
    te2[3]:10:4);
end;
'l','l':
begin
  stopplt:= true;
end;
'm','m':
begin
  stopprog:= true;
end;
end;
end;

procedure paraminit;

var i,j: integer;

begin
  for i:= 1 to n do phi[i]:= 0.0;

```

```

tel:= phi;      tec1d1:= phi;
te2:= phi;      tec2d1:= phi;
for i:= 1 to n do p1d1[i,i]:= 1000.0;
p2d1:= p1d1;
case comp of
  none:
  begin
    tef1[1]:= 0.0;  tef1[2]:= 0.0;  tef1[3]:= 0.0;
    tef2[1]:= 0.0;  tef2[2]:= 0.0;  tef2[3]:= 0.0;
  end;
  fix:
  begin
    tef1[1]:= atilf1;  tef1[2]:= btilf1;  tef1[3]:= delf1;
    tef2[1]:= atilf2;  tef2[2]:= btilf2;  tef2[3]:= delf2;
  end;
  adptv:
  end;
  s0:= -(a0+am)+(a+1.0);
  s1:= a0*am-a;
  r0:= b;
  pf:= 1.0-am;
  it:= 0;
  actual:= 0;
  sampint:= ticks*0.055;
  nout:= round(outint/0.055);
end;

```

```

procedure prepareplot;

```

```

var   i,j,expo10: integer;
      min,max,maxabs,pot10,ygmax,ygmin: real;

begin
  if (it-1) div nout < maxpoint then
  begin
    strtp:=1; maxpl:= actual;
  end
  else
  begin
    strtp:= (actual+1) mod maxpoint; maxpl:= maxpoint;
  end;
  min:= plotarr[1]; max:= plotarr[1];
  for i:= 2 to maxpl do
  begin
    j:= i mod maxpoint;
    if plotarr[j] > max then max:= plotarr[j];
    if plotarr[j] < min then min:= plotarr[j];
  end;
  if abs(max) > abs(min) then maxabs:= abs(max) else maxabs:= abs(min);
  if maxabs < 0.0 then
  begin
    expo10:= trunc(ln(maxabs)/ln(10.0));
    if expo10 <= 0 then expo10:= expo10-1;
  end
  else
  begin
    expo10:= 0;
  end;
end;

```

```

pot10:= exp(expo10*ln(10.0));
ygmaz:= (trunc(maxabs/pot10)+1)*pot10;
ygmin:= -ygmaz;
yscale:= ymax/(2.0*ygmaz);
tscale:= xmax/(stot[actual]-stot[strtp]);
for i:= 1 to maxpl do
begin
  graphx[i]:= round(tscale*(stot[(strtp+i-1) mod maxpoint]-stot[strtp]))+x0
  graphy[i]:= round(yscale*plotarr[(strtp+i-1) mod maxpoint])+y0;
end;
if not encode(lbly1,stot[strtp]:10:3) then writeln('error in encode 1');
if not encode(lbly2,stot[(strtp+maxpl-1) mod maxpoint]:10:3) then
  writeln('error in encode 2');
if not encode(lbly1,ygmaz:10:3) then writeln('error in encode 3');
if not encode(lbly2,ygmin:10:3) then writeln('error in encode 4');
end;

procedure plotgraph;

var   i: integer;

begin
  setdisplay(graphg0,grafpg0,grafpg0);
  moveto(x0,y0-ymax div 2);
  line(xmax,0); line(0,ymax); line(-xmax,0); line(0,-ymax);
  moveto(x0,y0); line(xmax,0);
  fontgr(2);
  moveto(0,186); writegr('press key F1 to get control');
  moveto(0,171); writegr(lbly1);
  moveto(0,35); writegr(lbly2);
  moveto(60,20); writegr(lbly1);
  moveto(500,20); writegr(lbly2);
  moveto(0,0); writegr('plotted variable: '); writegr(plotvar);
  moveto(graphx[1],graphy[1]);
  for i:= 2 to maxpl do
  begin
    lineto(graphx[i],graphy[i]);
  end;
  eventqueue(true);
  stopgraf:= false;
  repeat
    if keyevent(false,evnt) then
    begin
      if evnt.scancode = 59 then stopgraf:= true;
    end
  until stopgraf;
  stopevent;
  setdisplay(0,0,0);
end;

begin
  initadda;
  daout(0,0.0);
  daout(1,0.0);
  initgrafix(ibmhires);
  setdefval;
  stopprog:= false;

```



```
repeat
  startplt:= false;
repeat
  moddone:= false;
repeat
  displaypar;
  menu1;
until moddone;
paraminit;
writeln;
writeln('strike key F1 to start the controller');
eventqueue(true);
startctrl:= false;
repeat
  if keyevent(false,evt) then
  begin
    if evt.scancode = 59 then startctrl:= true;
  end
until startctrl;
writeln;
writeln('      controller is running');
writeln('strike key F1 to stop the controller');
writeln;
stopctrl:= false;
schedule(foregnd1,ticks);
repeat
  if keyevent(false,evt) then
  begin
    if evt.scancode = 59 then stopctrl:= true;
  end
until stopctrl;
endsched;
stopevent;
daout(0,0.0);
menu2;
until startplt;
stopplt:= false;
repeat
  menu3;
  if plotgrf then
  begin
    prepareplot;
    plotgraph;
    plotgrf:= false;
  end;
until stopplt or stopprog;
until stopprog;
end.
```

DEFVAL5.:1

29-MAY-1985 17:03

Page 1

```
1      ; ticks
0.055  ; ouint
0.9    0.13 ; a, b
0.99   ; lam
0.75   ; am
0.8    ; a0
0.00 0.0 -0.01 ; tef1
0.00 0.0 0.01 ; tef2
1.0 -1.0      ; umax, umin
```

```

sampling intervall =      0.055
output intervall =
a =      0.055
b =      0.900
forgetting factor lam =   0.130
closed-loop parameter: am = 0.990
observer polynomial: a0 = 0.900
values of fixed compensation:
  atilf1 =      0.000  btilf1 =      0.000
  atilf2 =      0.000  btilf2 =      0.000
control signal limitations:
  umax =      1.000  umin =     -1.000
friction compensation: comp = fix
                        delf1 =     -0.010
                        delf2 =      0.010

```

do you want to make any modifications?

answer [y/n]: y

choose one of the following possibilities:

- a: change sampling intervall
- b: change output intervall
- c: change model parameter a
- d: change model parameter b
- e: change forgetting factor
- f: change closed loop parameter am
- g: change observer polynomial
- h: change values of fixed compensation
- i: change control signal limitations
- j: change friction compensation mode

answer: c

```

a =      0.900
new value: 0.8

```

a: initialize another experiment
b: initialize plots

answer: b

choose one of the following possibilities:

a: plot reference signal r
b: plot signal y
c: plot control signal u
d: plot estimation error e
e: plot estimated parameter atil1
f: plot estimated parameter btill
g: plot estimated parameter dell
h: plot estimated parameter atil2
i: plot estimated parameter btill2
j: plot estimated parameter del2
k: display estimated parameters
l: initialize another experiment
m: stop the program

answer: c

IDENT.PAS:1

29-MAY-1985 17:03

Page 1

PROGLOG - IDENT

PURPOSE:

This program is to make an open-loop identification of a DC motor system.

USAGE:

The input signal from a signal generator is connected to the analog input channel 0 and the output of the process to the analog input channel 1. The analog output channel 1 is the input signal to the system to identify and should be connected to the process.

After typing the name of the executable file the name of the file with the default values should be specified. This file must have the following form:

```
1      ; ticks (number of ticks between two calls of foregr.  
0.5    ; ouint (output interval for plots)  
0.0 0.0 ; a, b  (initial values for model parameters)  
0.99   ; lam   (forgetting factor)
```

The rest of the program is self-expllicable.

DATE:

May 28 1985

AUTHOR

Konrad Braun

{\$include: 'addab.int'}

program test4(input,output,defval);

uses addab(adin,daout,initadda);

```
const  n=3;  
       x0= 99;      y0= 110;      xmax= 500;      ymax=150;  
       maxpoint= 200;
```

type vec1= array [1..n] of real;

mat1= array [1..n,1..n] of real;

{\$include: 'grconst.inc'}

```

var    defval: text;

i,j,ticks,actual,strt,p,maxpl,it,nout      : integer;
a,b,y,yd1,u,ud1,e,lam,                    : real;
t,sampint,tscale,yscale,outint,            : real;
phi,te1,te1d1,te2,te2d1,tef1,tef2,         : vec1;
tec1d1,tec2d1                               : mat1;
p1,p1d1,p2,p2d1                             : boolean;
stopprog,stopplt,startplt,moddone,         : (pos, neg, nul);
startctrl,stopctrl,stopgraf,plotgrf
updflag                                     : array[1..maxpoint] of integer;
graphx,graphy                               : array[0..maxpoint-1] of real;
stot,stoy,stou,stoe,                      : string(20);
stoat11,stobt11,stodel1,                  : lstring(9);
stoat1l2,stobt1l2,stodel2,                : lstring(9);
plotarr                                    : event;

plotvar
lblx1,lblx2,lbly1,lbly2
evnt

value  updflag:= nul;
{$include: 'grextrn.inc'}

procedure schedule(procedure fg [public]; ticks: integer); extern;

procedure endsched; extern;

function sign(x: real): integer;
begin
  sign:= ord(x>0.0)-ord(x<0.0)
end;

procedure foregnd1 [public];
var    d1,t1,t2: real;
        estim: boolean;

procedure rls;
var    s1: real;
        kd1,v1: vec1;

```

IDENT.PAS:1

29-MAY-1985 17:03

```
      m1,m2: mat1;

procedure vsmul(var v: vec1; var s: real; var res: vec1);
var i: integer;
begin
  for i:= 1 to n do res[i]:= v[i]*s;
end;

procedure vvtmul(var v1,v2: vec1; var res: mat1);
var i,j: integer;
begin
  for i:= 1 to n do
    for j:= 1 to n do res[i,j]:= v1[i]*v2[j];
    end;
end;

procedure vtmul(var v1,v2: vec1; var res: real);
var i: integer;
begin
  res:= 0.0;
  for i:= 1 to n do res:= res+v1[i]*v2[i];
end;

procedure mvmul(var m: mat1; var v,res: vec1);
var i,j: integer;
begin
  for i:= 1 to n do
    begin
      res[i]:= 0.0;
      for j:= 1 to n do res[i]:= res[i]+m[i,j]*v[j];
      end
    end;
end;

procedure mmmul(var m1,m2,res: mat1);
var i,j,k: integer;
begin
  for i:= 1 to n do
    begin
      for j:= 1 to n do
        begin
          res[i,j]:= 0.0;
          for k:= 1 to n do res[i,j]:= res[i,j]+m1[i,k]*m2[k,j];
          end
        end
      end;
    end;
end;
```

```
procedure emsubdiv(var m: mat1; var s: real; var res: mat1);
```

```
var i,j: integer;
```

```
begin
  for i:= 1 to n do
    begin
      for j:= 1 to n do
        begin
          if i=j then
            res[i,j]:= (1.0-m[i,j])/s
          else
            res[i,j]:= -m[i,j]/s;
          end
        end
      end
    end;
end;
```

```
procedure vvadmul(var v1,v2: vec1; var s: real; var res: vec1);
```

```
var i: integer;
```

```
begin
  for i:= 1 to n do
    res[i]:=v1[i]+v2[i]*s;
  end;
```

```
begin
  if yd1>0.0 then
    begin
      updf1ag:= pos;
      vtmul(phi,te1d1,s1);
      e:= y-s1-a*yd1-b*ud1;
      mvmul(p1d1,phi,v1);
      vtmul(phi,v1,s1);
      s1:= 1.0/(lam+s1);
      vsmul(v1,s1,kd1);
      vvadmul(te1d1,kd1,e,te1);
      vvtmul(kd1,phi,m1);
      emsubdiv(m1,lam,m2);
      mmmul(m2,p1d1,p1);
    end
  else
    begin
      updf1ag:= neg;
      vtmul(phi,te2d1,s1);
      e:= y-s1-a*yd1-b*ud1;
      mvmul(p2d1,phi,v1);
      vtmul(phi,v1,s1);
      s1:= 1.0/(lam+s1);
      vsmul(v1,s1,kd1);
      vvadmul(te2d1,kd1,e,te2);
      vvtmul(kd1,phi,m1);
      emsubdiv(m1,lam,m2);
      mmmul(m2,p2d1,p2);
    end
  end;
```


IDENT.PAS;1

29-MAY-1985 17:03

end;

procedure storeval;

begin

it:= it+1;

if (it-1) mod nout = 0 then

begin

actual:= (actual+1) mod maxpoint;

stot[actual]:= (it-1)*sampint;

stoy[actual]:= y*10.0;

stou[actual]:= u*10.0;

stoe[actual]:= e;

stoat11[actual]:= te1[1];

stobtil1[actual]:= te1[2];

stodel1[actual]:= te1[3];

stoat12[actual]:= te2[1];

stobtil2[actual]:= te2[2];

stodel2[actual]:= te2[3];

end;

end;

procedure update;

begin

phi[1]:= y; phi[2]:= u; phi[3]:= 1.0;

ud1:= u;

yd1:= y;

case updf flag of

pos:

begin

te1d1:= te1;

p1d1:= p1;

updf flag:= nul;

end;

neg:

begin

te2d1:= te2;

p2d1:= p2;

updf flag:= nul;

end;

nul:

end;

end;

begin

u:= adin(0); y:= adin(1);

daout(0,u);

estim:= (yd1<0.0) and (sign(y) + sign(yd1) < 0);

if estim then rls;

storeval;

update;

end;

procedure setdefval;

```

begin
  reset(defval);
  readln(defval,ticks);
  readln(defval,outint);
  readln(defval,a,b);
  readln(defval,lam);
  close(defval)
end;

procedure displaypar;
begin
  writeln;
  writeln('sampling intervall = ',ticks*0.055:10:3);
  writeln('output intervall = ',outint:10:3);
  writeln('a = ',a:10:3);
  writeln('b = ',b:10:3);
  writeln('forgetting factor lam = ',lam:10:3);
end;

procedure menu1;
var   answer: char;

begin
  repeat
    writeln;
    writeln('do you want to make any modifications?');
    writeln;
    write('answer [y/n]: ');
    readln(answer);
  until answer in ['y','Y','n','N'];
  if answer in ['y','Y'] then
    begin
      repeat
        writeln;
        writeln('choose one of the following possibilities:');
        writeln;
        writeln(' a: change sampling intervall');
        writeln(' b: change output intervall');
        writeln(' c: change model parameter a');
        writeln(' d: change model parameter b');
        writeln(' e: change forgetting factor');
        writeln;
        write('answer: ');
        readln(answer);
      until answer in ['a'..'e','A'..'E'];
      writeln;
      case answer of
        'a','A':
          begin
            writeln('sampling intervall = ticks * 0.055 sec = ',ticks*0.055:10:3);
            write('new value: ticks = ');
            readln(ticks);
          end;
        'b','B':
          begin
            writeln('ouput intervall = ',outint:10:3);

```

29-MAY-1985 17:03

IDENT.PAS:1

```
    write('new value: ');
    readln(outint);
end;
'c','c':
begin
    writeln('a = ',a:10:3);
    write('new value: ');
    readln(a);
end;
'd','d':
begin
    writeln('b = ',b:10:3);
    write('new value: ');
    readln(b);
end;
'e','e':
begin
    writeln('lam = ',lam:10:3);
    write('new value: ');
    readln(lam);
end;
end;
end
else
begin
    moddone:= true;
end
end;
end;
```

procedure menu2;

var answer: char;

```
begin
  repeat
    writeln;
    writeln('choose one of the following possibilities:');
    writeln;
    writeln(' a: initialize another identification');
    writeln(' b: initialize plots');
    writeln;
    write('answer: ');
    readln(answer);
  until answer in ['a','b','A'..'B'];
  if answer in ['b','B'] then startplt:= true;
end;
```

procedure menu3;

var answer: char;

```
begin
  repeat
    writeln;
    writeln('choose one of the following possibilities:');
    writeln;
    writeln(' a: plot control signal u');
```

IDENT.PAS;1

29-MAY-1985 17:03

```
writeln(' b: plot signal y');
writeln(' c: plot estimation error e');
writeln(' d: plot estimated parameter atil1');
writeln(' e: plot estimated parameter btill');
writeln(' f: plot estimated parameter dell');
writeln(' g: plot estimated parameter atil2');
writeln(' h: plot estimated parameter btil2');
writeln(' i: plot estimated parameter del2');
writeln(' j: display estimated parameters');
writeln(' k: initialize another experiment');
writeln(' l: stop the program');
writeln;
write('answer: ');
readln(answer);
until answer in ['a'..'l','A'..'L'];
case answer of
  'a','A':
    begin
      plotarr:= stou;
      plotvar:= 'control signal u';
      plotgrf:= true;
    end;
  'b','B':
    begin
      plotarr:= stoy;
      plotvar:= 'output signal y';
      plotgrf:= true;
    end;
  'c','C':
    begin
      plotarr:= stoe;
      plotvar:= 'estimation error e';
      plotgrf:= true;
    end;
  'd','D':
    begin
      plotarr:= stoatill;
      plotvar:= 'estim. par. atill';
      plotgrf:= true;
    end;
  'e','E':
    begin
      plotarr:= stobtill;
      plotvar:= 'estim. par. btill';
      plotgrf:= true;
    end;
  'f','F':
    begin
      plotarr:= stodel1;
      plotvar:= 'estim. par. dell';
      plotgrf:= true;
    end;
  'g','G':
    begin
      plotarr:= stoatill2;
      plotvar:= 'estim. par. atil2';
      plotgrf:= true;
    end;
  'h','H':
```

```

begin
  plotarr:= stobtil2;
  plotvar:= 'estim. par. btil2  ';
  plotgrf:= true;
end;
'i','I':
begin
  plotarr:= stodel2;
  plotvar:= 'estim. par. del2  ';
  plotgrf:= true;
end;
'j','J':
begin
  writeln;
  writeln('estimated parameters:');
  writeln;
  writeln('  atil1 = ',te1[1]:10:4,'      btil1 = ',te1[2]:10:4,'      dell = ',
    te1[3]:10:4);
  writeln('  atil2 = ',te2[1]:10:4,'      btil2 = ',te2[2]:10:4,'      del2 = ',
    te2[3]:10:4);
end;
'k','K':
begin
  stopplt:= true;
end;
'l','L':
begin
  stopprog:= true;
end;
end;
end;

```

```

procedure paraminit;

```

```

  var i,j: integer;

```

```

begin
  for i:= 1 to n do phi[i]:= 0.0;
  tel:= phi;   tec1d1:= phi;
  te2:= phi;   tec2d1:= phi;
  for i:= 1 to n do p1d1[i,i]:= 1000.0;
  p2d1:= p1d1;
  it:= 0;
  actual:= 0;
  sampint:= ticks*0.055;
  nout:= round(outint/0.055);
end;

```

```

procedure prepareplot;

```

```

  var      i,j,expo10: integer;
          min,max,maxabs,pot10,ygmax,ygmin: real;

```

```

begin
  if (it-1) div nout < maxpoint then
  begin
    strtpr:=1; maxpl:= actual;

```

```

end
else
begin
  strtpr:= (actual+1) mod maxpoint; maxpl:= maxpoint;
end;
min:= plotarr[1]; max:= plotarr[1];
for i:= 2 to maxpl do
begin
  j:= i mod maxpoint;
  if plotarr[j] > max then max:= plotarr[j];
  if plotarr[j] < min then min:= plotarr[j];
end;
if abs(max) > abs(min) then maxabs:= abs(max) else maxabs:= abs(min);
if maxabs < 0.0 then
begin
  expol0:= trunc(ln(maxabs)/ln(10.0));
  if expol0 <= 0 then expol0:= expol0-1;
end
else
begin
  expol0:= 0;
end;
potl0:= exp(expol0*ln(10.0));
ygmax:= (trunc(maxabs/potl0)+1)*potl0;
ygmin:= -ygmax;
yscale:= ymax/(2.0*ygmax);
tscale:= xmax/(stot[actual]-stot[strtp]);
for i:= 1 to maxpl do
begin
  graphx[i]:= round(tscale*(stot[strtp+i-1] mod maxpoint)-stot[strtp])+x0;
  graphy[i]:= round(yscale*plotarr[(strtp+i-1) mod maxpoint])+y0;
end;
if not encode(lbly1,stot[strtp]:10:3) then writeln('error in encode 1');
if not encode(lbly2,stot[(strtp+maxpl-1) mod maxpoint]:10:3) then
  writeln('error in encode 2');
if not encode(lbly1,ygmax:10:3) then writeln('error in encode 3');
if not encode(lbly2,ygmin:10:3) then writeln('error in encode 4');
end;

procedure plotgraph;
var
  i: integer;

begin
  setdisplay(grafpg0,grafpg0,grafpg0);
  moveto(x0,y0-ymax div 2);
  line(xmax,0); line(0,ymax); line(-xmax,0); line(0,-ymax);
  moveto(x0,y0); line(xmax,0);
  fontgr(2);
  moveto(0,186); writegr('press key F1 to get control');
  moveto(0,171); writegr(lbly1);
  moveto(0,35); writegr(lbly2);
  moveto(60,20); writegr(lbly1);
  moveto(500,20); writegr(lbly2);
  moveto(0,0); writegr('plotted variable: '); writegr(plotvar);
  moveto(graphx[1],graphy[1]);
  for i:= 2 to maxpl do
  begin

```

29-MAY-1985 17:03

IDENT.PAS;1

```

lineto(graphx[i],graphy[i]);
end;
eventqueue(true);
stopgraf:= false;
repeat
if keyevent(false,evnt) then
begin
if evnt.scancode = 59 then stopgraf:= true;
end
until stopgraf;
stopevent;
setdisplay(0,0,0);
end;

begin
initadda;
daout(0,0,0);
daout(1,0,0);
initgrafix(ibmhires);
setdefval;
stopprog:= false;
repeat
startplt:= false;
repeat
moddone:= false;
repeat
displaypar;
menu1;
until moddone;
paraminit;
writeln;
writeln('strike key F1 to start the identification');
eventqueue(true);
startctrl:= false;
repeat
if keyevent(false,evnt) then
begin
if evnt.scancode = 59 then startctrl:= true;
end
until startctrl;
writeln;
writeln('identification is running');
writeln('strike key F1 to stop the identification');
writeln;
stopctrl:= false;
schedule(foregnd1,ticks);
repeat
if keyevent(false,evnt) then
begin
if evnt.scancode = 59 then stopctrl:= true;
end
until stopctrl;
endsched;
stopevent;
daout(0,0,0);
menu2;
until startplt;
stopplt:= false;

```

```
repeat
  menu3;
  if plotgrf then
    begin
      prepareplot;
      plotgraph;
      plotgrf:= false;
    end;
  until stopplt or stopprog
until stopprog;
end.
```