STUPID – Implementation of a Self-Tuning PID-Controller

Wittenmark, Björn; Hagander, Per; Gustavsson, Ivar

1980

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

CODEN:LUTFD2/(TFRT-7201)/1-030/(1980)

STUPID -
IMPLEMENTATION OF A SELF-TUNING PID-CONTROLLER

WITTENMARK, B
HAGANDER, P
GUSTAVSSON, I

LUND INSTITUTE OF TECHNOLOGY
DEPARTMENT OF AUTOMATIC CONTROL
JUNE, 1980

Organization
LUND INSTITUTE OF TECHNOLOGY
Dept of Automatic Control
Box 725
S-220 07 LUND 7
Sweden

Document name

Date of issue
June, 1980

CODEN: LUTFD2/(TFRT-7201)/1-030/(1980)

Author(s)
Wittenmark, B, Hagander, P and
Gustavsson, I

Sponsoring organization

Title and subtitle
STUPID- Implementation of a self-tuning PID-controller

A4 A5

Abstract
A self-tuning PID-controller based on pole-placement has been implemented on an LSI-11. The controller and the operator communication are written in Pascal. The program works with the controller as a foreground job and the operator communication as a background job. The operator communication is command driven with nine different commands.

The program is tested on systems simulated on analog computers and on some laboratory processes.

Key words
A4 A5

Classification system and/or index terms (if any)

Supplementary bibliographical information

Language
English

ISSN and key title

ISBN

Recipient's notes

Number of pages
30

Price

Security classification

Distribution by (name and address)

STUPID

Implementation of a self-tuning PID-controller

Bjorn Wittenmark
Per Hagander
Ivar Gustavsson

Department of Automatic Control
Lund Institute of Technology

June 1980

ABSTRACT

A self-tuning PID-controller based on pole placement has been implemented on an LSI-11. The controller and the operator communication are written in Pascal. The program works with the controller as a foreground job and the operator communication as a background job. The operator communication is command driven with nine different commands.

The program is tested on systems simulated on analog computers and on some laboratory processes.

TABLE OF CONTENTS

1. INTRODUCTION

The aim of the work has been to implement a self-tuning PID-controller based on pole placement on an LSI-11 computer. The control algorithm is described in Wittenmark (1979) and Wittenmark-Astr m (1980).

The control algorithm is based on estimation of the parameters of a second order process model. Based on the model the parameters in the PID-controller are chosen such that the closed loop system will have its poles at desired locations.

The parameters of the controller are obtained by solving the polynomial identity:

$$A(q^{-1})R(q^{-1})(1-q^{-1}) + q^{-1}B(q^{-1})S(q^{-1}) = D(q^{-1})$$

where B/A is a second order process model, D the desired characteristic polynomial, and

$$R(q^{-1}) = 1+r1*q^{-1}$$

$$S(q^{-1}) = s0+s1*q^{-1}+s2*q^{-2}$$

are defining the regulator. The controller can be written as

$$(1-q^{-1})R(q^{-1})*u(t) = S(1)*y_r(t) - S(q^{-1})*y(t) \qquad (1)$$

where u, y_r and y are the control signal, the reference value and the process output respectively. The four parameter values r1, s0, s1 and s2 are natural to use internally in the computer.

A conventional PID-controller has four parameters also: gain, K, reset time, TI, derivation time, TD, and a filter constant for derivation, ALFA. This parametrization is natural for manual tuning. The transformation between the conventional PID-controller and the controller (1) is described in Wittenmark (1979).

In the self-tuning PID-controller the operator choses the closed loop system parameters and some parameters in the estimation routine:

TSAMP	- the sampling interval
OM	- the natural frequency of the desired closed loop system
DAMP	- the damping of the desired closed loop system
LAM	- the forgetting factor in the estimation routine
PO	- the initial covariance in the estimation routine

TH1-TH4 - the initial values of the parameters in the second order process model.

It is possible to introduce an automatic choice of the sampling time, see Åström (1979)

The program is written in Pascal and a listing is given in the Appendix.

Design considerations

The controller contains some parameters that the operator might want to change on line. Especially during the evaluation phase it is also desirable to be able to on line start and stop the estimation and controller parts and to change to an operator tuned PID-controller. Such operator communication is done using an alfanumeric display and a tastature (Beehive B100).

To obtain convenience for the operator when changing parameters and operating mode it was decided that the communication should be command driven. Our decision is based mainly on the modularity in the implementation of command driven communication and on the greater flexibility for the user. The alternative would be to use block mode communication like in Andersson-Åström (1978) with prewritten forms on the display and windows for the parameters to be changed.

The scheduling of the controller is done using a foreground background scheduler described in Mattsson (1978). The computer and the process communicates via D/A and A/D routines.

2. THE CONTROLLER

The foreground procedure Fg is given in Appendix. The procedure contains the following routines:

```
Regulator
  Adin (external)
  Output
    Ushape
  Daout (external)
  Update
    LS (external)
    Pardet
Stateset
   Pardet
Trigosc
Loguy
```

In the following a brief description of the different routines will be given.

Regulator
This procedure reads the process output and the reference value from user specified A/D channels. The input to the process is then computed by calling Output. The process input is then sent to the process via a D/A converter using Daout. The process parameter estimates are finally updated and the regulator parameters are recalculated and are used at the next sampling time.

Output
The controller given in Section 1 is implemented using the following state space representation

$$x(t+1) = \begin{bmatrix} 1-r1 & 1 \\ r1 & 0 \end{bmatrix} x(t) + \begin{bmatrix} -S(1)(r1-1) \\ S(1)r1 \end{bmatrix} y_r(t) - \begin{bmatrix} s1-s0(r1-1) \\ s2+s0r1 \end{bmatrix} y(t)$$

$$u(t) = (1 \quad 0)x(t) + S(1)y_r(t) - s0y(t) \qquad (2)$$

The output u(t) is now calculated and limited to the interval (lolim,hilim) using Ushape. Notice that the state variable x[1] is recalculated to correspond to the limited value. This is done in order to eliminate reset windup.

Update
The main computations are made within Update, where the process and regulator parameters are updated using LS and Pardet.

The process parameters of a second order model are estimated in LS using differences of the input and the output of the process. When new process parameters are obtained the regulator parameters are updated in Pardet. The state of the regulator is then updated using equation (2).

LS
The parameter estimates are obtained using a recursive least squares method implemented according to Bierman (1977). The covariance matrix is updated in square root form. Essentially a one dimensional information update is used for the diagonal in order to minimize the risk for non positiveness inherent in a covariance update. An exponential forgetting factor is also included.

Pardet
In this procedure the regulator parameters are computed from the process parameters (theta) and the desired characteristic polynomial (dpol).

Some tests are done in order to prevent numerical difficulties. First the coefficients in the B-polynomial (theta[3] and theta[4]) should not both be equal to zero. If the absolute values of both are less than thmin (thmin=1E-4) then the largest one is put equal to thmin. Further there is a test for pole-zero cancellation between the polynomials (q-1)A(q) and B(q). If this is the case the common factor is cancelled. The computation of the regulator parameters (rpol, spol and tpol) is done differently depending on the tests.

Stateset
The procedure Stateset changes the regulator state when the operator has changed parameters. This is indicated with the switch newpar. The state is not changed using Stateset when the estimator changes the estimated process parameters. New regulator parameters are computed, and the stationary value of the state corresponding to the regulator parameters and the current values of the input, output and reference value is computed.

Trigosc
On the D/A channel trich a signal is generated which is 0.5 or -0.5 depending on the sign of the reference value. This signal can be used to trig an oscilloscope.

Loguy
A backlog of the control variable, the process output and the reference value is stored in a cyclic file. The file will contain tmax values (tmax=20). This file is displayed using the command LOG.

3. THE OPERATOR COMMUNICATION

The operator communication works in the background, and it is interrupted each sampling instant by the controller foreground routine. When the program is loaded from the diskette storage, the available command menue is written on the display (Fig 1.), and a ready sign,>, indicates that the machine is ready to accept a command. Any of the nine commands can be given at any time.

The command list in Fig 1. will be displayed by the command HELP. A command EXIT is included to terminate the execution of the whole program. Two commands, RUN and STOP, are used to start and stop the controller in the foreground, while FIX controls the update of the controller parameters from the identified process parameters. A log of the 20 most recent values of the input, output, and reference signal is displayed by the command LOG.

```
   SELF-TUNING PID-CONTROLLER

The commands have the following form:

     Command Argument Value

The following commands are implemented:

   COMMAND  ARGUMENT
     HELP
     RUN    CONT,PID, No argument
     STOP
     DISP
     FIX    ON,OFF
     INIT   TH1,TH2,TH3,TH4,PO,DEF
     PAR    OM,DAMP,TSAMP,LAM,K,TI,TD,ALFA,
            HILIM,LOLIM,YRCH,YCH,UCH,TRICH,DEF
     LOG
     EXIT
```

Fig 1: The command list in STUPID displayed by the command HELP.

The commands PAR and INIT are used to change the parameters of the controller, and DISP gives a display of the database, i.e. the current parameters, the identified model and the controller polynomials (see Fig 2).

The Pascal program is structured such that the decoding of the command line starts in the main part of the operator communication procedure Comcom, and depending on the command code a call is generated to the procedure for that command (or to the error message procedure). It was thus easy to implement the commands one by one, replacing the error message "Command not implemented". The commands RUN and FIX may have one argument on the same command line, and INIT and PAR may have two, i.e. an argument name and a value (see Fig 1). The argument decoding is done in the respective command procedures, so error messages are easily produced. The argument values of the command lines are checked for consistency with the argument names, i.e. sign etc.

The communication between the background and the foreground program may be critical. The display command DISP is reading from the database, and if the foreground controller changes some values during the readout the display will be erroneous, but this is not considered to be dangerous.

```
>DISP

TSAMP=      3.000    LOLIM =     -1.000    HILIM =       1.000
YRCH =      0        YCH   =      1        UCH   =       0
TRICH=      1

INITIAL VALUES
PO   =      100.0    LAM   =      0.980
TH   =     -1.500      0.700      0.100      0.000
CURRENT ESTIMATION
PO   =      14.45      104.1      104.1      104.1
TH   =     -0.205      0.700      0.100      0.000
STUPID PARAMETERS
OM   =      1.000    DAMP  =      0.700
DPOL =      1.000      0.132      0.015
PID PARAMETERS
K    =      1.000    TI    =      900.0    TD    =       0.000
ALFA =      0.300

REGULATOR POLYNOMIALS
TPOL =      11.47      0.000      0.000
SPOL =      13.37     -8.898      7.000
RPOL =      1.000     -1.000      0.000
```

Fig 2: The database displayed by the command DISP.

The changes from the background of the foreground parameters are done in a safer way. Two sets of parameters are used, one for the background and one for the foreground. When for instance a command line like "PAR DAMP 0.7" is given, a new desired D-polynomial is calculated, and the change is indicated by setting the flag newpar equal to true. The foreground Regulator senses newpar at the next sampling instant, and performs the transfer to its own D-polynomial at the end of its execution. The controller state is also modified for bumpless transfer.

In order to inhibit the foreground update during unfinished calculations of for instance the D-polynomial, newpar is made false during the background update. In case of slow sampling it might otherwise happen, that newpar is still true in response to the previous parameter change, when a new change is interrupted by the foreground procedure.

Some of the parameters (YRCH,YCH,UCH and TRICH) are possible to change only when no Regulator is active. "PAR DEF" gives default values to all the parameters except these channel numbers.

The initial values (TH1,TH2,TH3,TH4, and PO) to the estimation routine are changed by the command INIT, and the estimation is reinitialized, when the command RUN (without

arguments) is given. INIT DEF gives the default values (-1.5,0.7,0.1,0,100).

The command RUN signals to the Scheduler, by setting the parameter period >0 , that sampling and control should be performed in the foreground. The flag newpar is made true, as well, updating the foreground parameters and the controller state. The argument CONT indicates that the old estimation state should be used, so that no initialization of the estimation is performed. The argument PID means that the conventional PID-regulator should be used, which is signalled to the foreground using the flag ipid. Similarly FIX ON is flagged by ifix, indicating no controller parameter update due to the process parameter changes. When the log readout (LOG) is going on a flag logpar prohibits the foreground update of the log vectors.

4. MEMORY AND TIME REQUIREMENTS

The written code is about 16 pages. The amount of work to get the program working has been about 3-4 manweeks, including the documentation. The memory requirements to run the program is totally 19.6 bytes. The following table gives a feel for how large the different parts are:

Foreground program	5.4 kB
Operator communication	9.3 kB
Supporting runtime routines	5.7 kB

It is notable that the main part is the operator communication.

We have not been able to measure the execution time for the foreground program. The smallest available sampling time 1 tick = 20 ms is on the edge of the capacity of the computer. This is noticed for instance if the command LOG is given when TSAMP=0.02 s.

5. EXPERIMENTS

The program is available as an execute file STUPID.SAV. The program will be started by giving the command

```
RUN DX1:STUPID.SAV.
```

The parameter in the controller will be given default values and the available commands will be shown on the display.

The controller has been tested against processes simulated on an analog computer and against some laboratory processes. One example will be given here. More examples are given in Wittenmark-Åström (1980).

Example

This is an example showing how the controller can adapts to changing process dynamics. The process is changed abruptly as

$$1/(s+1)^2 \longrightarrow 1/(s(s+1)) \longrightarrow 1/s^2$$

These are very drastic changes. Also the changes are done such that the new system each time is more difficult to control than the previous. Fig 3 shows the output and the control signal. The controller is started with default values on the different parameters except that TSAMP=0.5, OM=1.5 and LAM=0.95. It can be seen that the controller very quickly adapts to the new dynamic of the process.

To make the simulation the following commands were given:

```
>PAR TSAMP 0.5
>PAR OM 1.5
>PAR LAM 0.95
>RUN
```

6. CONCLUSIONS

The project shows that Pascal has been a very useful tool to implement the self-tuning PID-controller and to get a very flexible operator communication. The time for development was reasonable small. This was to a large extent due to the excistence of the scheduler, the input and output routines and most of all that a good editing system PAGED, see Egardt and Elmqvist (1979), was available.

The authors want to express their thanks to Leif Andersson and Sven Erik Mattsson for their valuable assistance during the whole project.

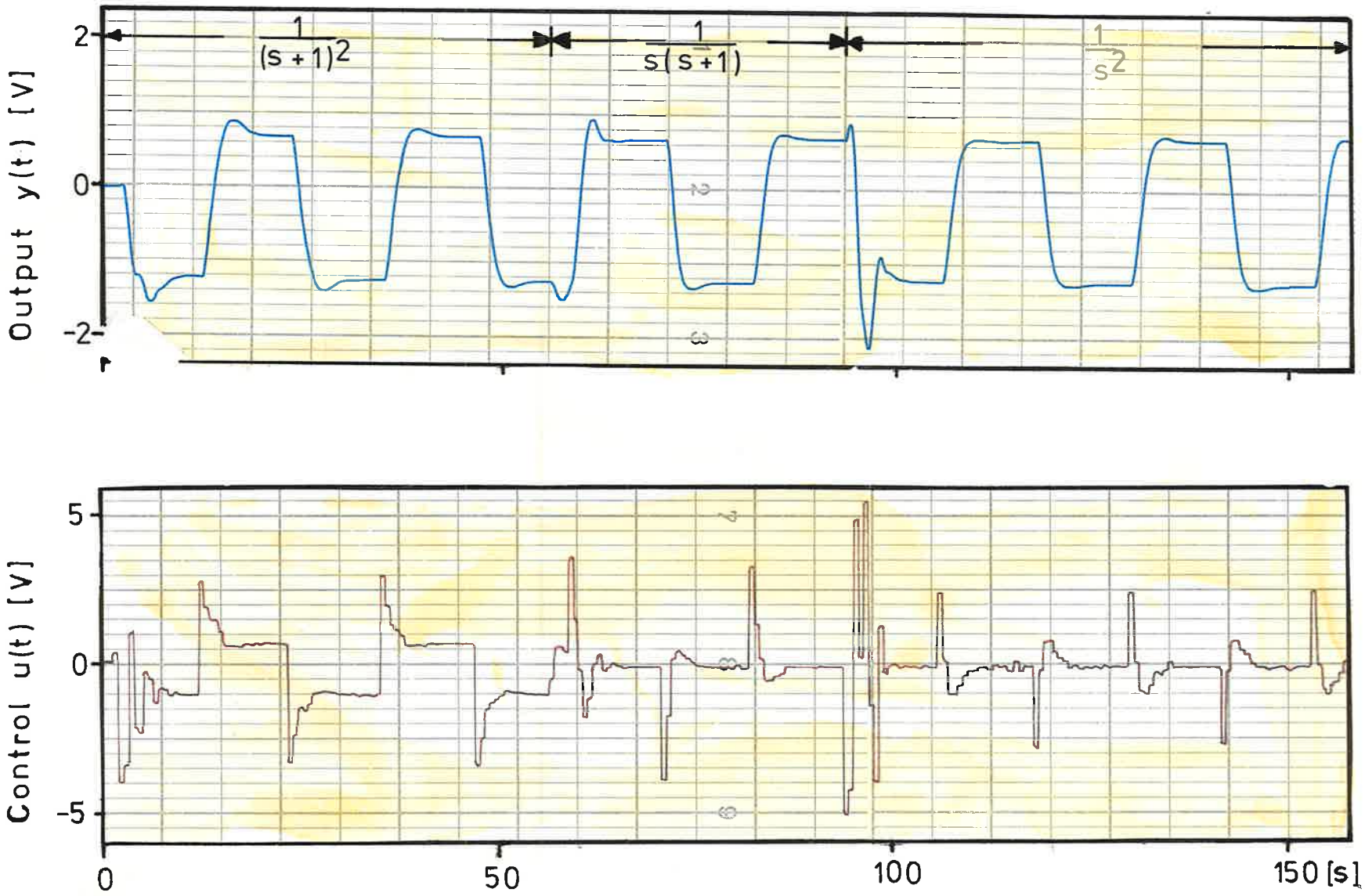

Fig 3: The output from the process, y(t), and the control signal, u(t), when controlling a process simulated on an analogue computer.

7. REFERENCES

Andersson L, Åström K J (1978): An interactive MISO regulator, Department of Automatic Control, Lund Institute of Technology, CODEN:LUTFD2/(TFRT-7154)/1-034/(1978)

Åström K J (1979): Simple self-tuners I, Department of Automatic Control, Lund Institute of Technology, CODEN:LUTFD2/(TFRT-7184)/1-063/(1979)

Bierman G J (1977): Factorization Methods for Discrete Sequential Estimation, Mathematics in Science and engineering, Vol 128, Academic Press

Egardt B, Elmqvist H (1979): PAGED - User's manual, Internal report, Department of Automatic Control, Lund Institute of Technology

Mattsson S E (1978): A simple real-time scheduler, Department of Automatic Control, Lund Institute of Technology, CODEN:LUTFD2/(TFRT-7156)/1-010/(1978)

Wittenmark B (1979): Self-tuning PID-controllers based on pole-placement, Department of Automatic Control, Lund Institute of Technology, CODEN:LUTFD2/(TFRT-7179)/1-037/(1979)

Wittenmark B, Åström K J (1980): Simple self-tuning controllers, Preprint International Symposium on Adaptive Control, March 20-21, 1980.

APPENDIX: LISTING OF PROGRAMS

```
Program stupid;
{SELF-TUNING PID-CONTROLLER
 Authors PH BW IG 1979-11-13 Revised BW 1980-05-27 ph 800612
 The program implements a self-tuning PID-controller based on
 pole-placement. The algorithm is described in
 Wittenmark: Self-tuning PID-controllers based on pole-placment,
    TFRT-7179
 Wittenmark-Astrom: Simple self-tuning controllers, TFRT-7191.
 The program is divided into the files:
    GLOBAL-global data declarations
    LS    -least squares estimator
    FG    -foreground program
    STUPID-operator communication and main program}

{GLOBAL DATA DECLARATIONS}

const tmax=20; {length of log}
      eps=0.01; {test quantity for pole-zero cancellation}
      thmin=1.E-4; {min absolute value of theta[3] and theta[4]}
      nlog=3; {number of variables stored in the log}
Type  estpartyp= record
        theta:array[1..4] of real;
        fi:array[1..4] of real;
        diag:array[1..4] of real;
        offdiag:array[1..6] of real
        end;
      regpartyp= record
        tpol:array[0..2] of real;
        spol:array[0..2] of real;
        rpol:array[0..2] of real;
        end;
      pidtyp=record
        k,ti,td,alfa:real
        end;
var
      {To be changed from background only when period=0}
       yrch,ych,uch,trich:integer;
       y,yold,yr,u,uold,lolim,hilim,lambda:real;
       x:array[1..2] of real;
       eststate:estpartyp;
       actregpar:regpartyp;
       dpol: array[0..2] of real;

      {to be changed from background only when newpar=false}
       newlolim,newhilim,newlambda,tsamp:real;
       newdpol:array[0..2] of real;
       pidpar:pidtyp;

      {to be changed from foreground only when logpar=false}
       backlog:array[0..tmax,1..nlog] of real;
       tlog:integer;

      {flags etc}
       ifix,ipid,newpar,logpar:boolean;
```

```
        period:integer;

{EXTERNAL PROCEDURE DECLARATIONS}

Function Rform(r:real; size:integer):integer;external;
Function Adin(chan:integer):real;external;
Procedure Daout(chan:integer;value:real);external;
Procedure Schedule(procedure fore;var period:integer);external;
Procedure Clksav;external;
Procedure Clkrestore;external;

{End of Global}

{Stupid Author PH BW IG 1979-11-13, Revised BW 1980-05-27 ph 800612}

Procedure Fg;external;

{=============================================================Comcom}
Procedure Comcom;

{Declarations for Comcom}

label 999;

const tsampdef=1.;omdef=1.;dampdef=0.7;lambdadef=0.98;hilimdef=1.;
      lolimdef=-1.;kdef=1.;tidef=9999.;tddef=0.;alfadef=0.3;
      p0def=100.; th1def=-1.5;th2def=0.7;th3def=0.1;th4def=0.;
      dyrch=0; dych=1; duch=0; dtrich=1;
type  errors=(illcom,fewarg,manyarg,illarg,illval,runact);
      names=array[1..6] of char;
      opindex=(xhelp,xrun,xstop,xdisp,xfix,xinit,xpar,xlog,xexit,
      xlastop);
      fixargindex=(xon,xoff,lastindex);
      runargindex=(xpid,xcont,lastx);
      parargindex=(xtsamp,xom,xdamp,xlam,xk,xti,xtd,xalfa,xhilim,
      xlolim,xpdef,xyrch,xych,xuch,xtrich,parlast);
      initargindex=(xp0,xth1,xth2,xth3,xth4,xidef,initlast);

var   op:array[opindex] of names;
      opx:opindex;
      fixarg:array[fixargindex] of names;
      fixargx:fixargindex;
      runarg:array[runargindex] of names;
      runargx:runargindex;
      pararg:array[parargindex] of names;
      parargx:parargindex;
      initarg:array[initargindex] of names;
      initargx:initargindex;
      command,text:names;
      exitpar:boolean;
      value:real;
      om,damp:real;
      estinit:estpartyp;
```

```
  {period must be 0}

  begin
    eststate:=estinit;
    yold:=0.;
    uold:=0.;
  end; {of Initest}
  {-------------------------------------------------------------Initreg}
  Procedure Initreg;
  {Initializes the regulator}
  {period must be 0}

  var n:integer;
  begin
    y:=0.;u:=0.;yr:=0.;
    with actregpar do
    begin
      for n:=1 to 2 do
      begin
        tpol[n]:=0.;rpol[n]:=0.;spol[n]:=0.;
        x[n]:=0.;
        dpol[n]:=newdpol[n]
      end;
      tpol[0]:=0.; spol[0]:=0.
    end;
  end; {of Initreg}
  {-----------------------------------------------------------Setdpol}
  Procedure Setdpol;
  {Computes new d-polynomial}
  {newpar must be false}

  var r,r1:real;
  begin
    if damp<1. then
    begin
      r:=exp(-damp*om*tsamp);
      r1:=om*tsamp*sqrt(1.-damp*damp);
      newdpol[1]:=-2.*r*cos(r1);
      newdpol[2]:=r*r
    end
    else
    begin
      r1:=-(damp+sqrt(damp*damp-1.))*om;
      r:=-(damp-sqrt(damp*damp-1.))*om;
      newdpol[1]:=-exp(r1*tsamp)-exp(r*tsamp);
      newdpol[2]:=exp(-2.*damp*om*tsamp)
    end
  end; {of Setdpol}
  {-----------------------------------------------------------Pardefault}
  Procedure Pardefault;
  {Sets default values for the parameters}
  {newpar must be false}

  begin
```

```
  {procedures for Comcom}

 {The procedures are used in the following way:

 Initialize
    Estdefault
    Initest
    Pardefault
       Setdpol
    Initreg
    Help
 Help
 Run
    Initest
    Initreg
    Error
 Stop
    Daout (external)
 Disp
    wxx
       Rform (external)
 Fix
    Error
 Init
    Estdefault
    Error
 Par
    Pardefault
       Setdpol
    Setdpol
    Error
 Log
    Rform (external)  }

 {-------------------------------------------------------------------Error}
 Procedure Error(err:Errors);
 {Produces error messages}

 begin
   case err of
     illcom:   write('illegal command ');
     fewarg:   write('too few arguments ');
     manyarg:  write('too many arguments ');
     illarg:   write('illegal argument ');
     illval:   write('illegal value ');
     runact:   write('run is active');
   end;{of case err}
   writeln;
   goto 999;
 end; {of Error}
 {-----------------------------------------------------------------Initest}
 Procedure Initest;
 {Initializes the estimator}
```

```
    pidpar.k:=kdef;
    pidpar.ti:=tidef;
    pidpar.td:=tddef;
    pidpar.alfa:=alfadef;
    newlambda:=lambdadef;
    newhilim:=hilimdef;
    newlolim:=lolimdef;
    tsamp:=tsampdef;
    om:=omdef;
    damp:=dampdef;
    Setdpol;
  end; {of Pardefault}
  {-------------------------------------------------------Estdefault}
  Procedure Estdefault;
  {Gives default values to estimator}

  var  n:integer;
  begin
  with estinit do
  begin
    theta[1]:=th1def;
    theta[2]:=th2def;
    theta[3]:=th3def;
    theta[4]:=th4def;
    for n:=1 to 4 do
    begin
      diag[n]:=p0def;
      offdiag[n]:=0.;
      fi[n]:=0.
    end;
    offdiag[5]:=0.;offdiag[6]:=0
  end;
  end; {of Estdefault}
  {-----------------------------------------------------------Help}
  Procedure Help;
  {Writes information and list of commands}

  var i:integer;
  begin
    for i:=1 to 6 do writeln;
    writeln('   SELF-TUNING PID-CONTROLLER');writeln;
    writeln('The commands have the following form:');
    writeln;
    writeln(' ':5,'Command Argument Value ');
    writeln;
    writeln('The following commands are inplemented:');
    writeln;
    writeln('   COMMAND  ARGUMENT');
    writeln(' ':5,'HELP');
    writeln(' ':5,'RUN',' ':4,'CONT,PID, No argument');
    writeln(' ':5,'STOP');
    writeln(' ':5,'DISP');
    writeln(' ':5,'FIX',' ':4,'ON,OFF');
    writeln(' ':5,'INIT',' ':3,'TH1,TH2,TH3,TH4,P0,DEF');
```

```
    writeln(' ':5,'PAR',' ':4,'OM,DAMP,TSAMP,LAM,K,TI,TD,ALFA,');
    writeln(' ':12,'HILIM,LOLIM,YRCH,YCH,UCH,TRICH,DEF');
    writeln(' ':5,'LOG');
    writeln(' ':5,'EXIT');
    writeln
end; {of Help}
{-------------------------------------------------------------Initialize}
Procedure Initialize;
begin
  op[xhelp]:='HELP  ';
  op[xrun]:= 'RUN   ';
  op[xstop]:='STOP  ';
  op[xdisp]:='DISP  ';
  op[xfix]:= 'FIX   ';
  op[xinit]:='INIT  ';
  op[xpar]:= 'PAR   ';
  op[xlog]:= 'LOG   ';
  op[xexit]:='EXIT  ';
  runarg[xcont]:= 'CONT  ';
  runarg[xpid]:= 'PID   ';
  fixarg[xon]:=   'ON    ';
  fixarg[xoff]:=  'OFF   ';
  pararg[xom]:='OM    ';
  pararg[xdamp]:='DAMP  ';
  pararg[xtsamp]:='TSAMP ';
  pararg[xlam]:='LAM   ';
  pararg[xk]:='K     ';
  pararg[xti]:='TI    ';
  pararg[xtd]:='TD    ';
  pararg[xalfa]:='ALFA  ';
  pararg[xhilim]:='HILIM ';
  pararg[xlolim]:='LOLIM ';
  pararg[xyrch]:='YRCH  ';
  pararg[xych]:='YCH   ';
  pararg[xuch]:='UCH   ';
  pararg[xtrich]:='TRICH ';
  pararg[xpdef]:='DEF   ';
  initarg[xth1]:='TH1   ';
  initarg[xth2]:='TH2   ';
  initarg[xth3]:='TH3   ';
  initarg[xth4]:='TH4   ';
  initarg[xp0]:='P0    ';
  initarg[xidef]:='DEF   ';
  tlog:=0;
  logpar:=true;
  exitpar:=false;
  ifix:=false;
  ipid:=false;
  actregpar.rpol[0]:=1.;
  dpol[0]:=1.;
  newdpol[0]:=1.;
  yrch:=dyrch;
  ych:=dych;
  uch:=duch;
```

```
  trich:=dtrich;
  lambda:=lambdadef;
  hilim:=hilimdef;
  lolim:=lolimdef;
  Estdefault;Initest;Pardefault;Initreg;newpar:=true;
  Help
end; {of Initialize}
{-------------------------------------------------------------Disp}
Procedure Disp;
{Displays current parameter values}

var i,k:integer;
    r  :real;

Procedure wxx(r:real);
begin
k:=Rform(r,6);write(r:10:k)
end;

begin
  writeln;
  write('TSAMP=');wxx(tsamp);write('  LOLIM =');wxx(newlolim);
  write('  HILIM = ');wxx(newhilim);writeln;
  writeln('YRCH =',ych:6,'       YCH   =',ych:6,'       UCH   = ',
  uch:6,'       TRICH=',trich:6);writeln;
  writeln('INITIAL VALUES');
  with estinit do
  begin
    write('PO   =');wxx(diag[1]);
    write('    LAM = ');wxx(newlambda);writeln;
    write('TH   =');for i:=1 to 4 do wxx(theta[i]) ;writeln
  end;{with}
  writeln('CURRENT ESTIMATION');
  with eststate do
  begin
    write('PO   =');for i:=1 to 4 do  wxx(diag[i]);writeln;
    write('TH   =');for i:=1 to 4 do  wxx(theta[i]);writeln
  end;{with}
  writeln('STUPID PARAMETERS');
  write('OM   =');wxx(om);  write('   DAMP = ');wxx(damp);writeln;
  write('DPOL =');for i:=0 to 2 do  wxx(newdpol[i]);  writeln;
  writeln('PID PARAMETRS');
  with pidpar do
  begin
    write('K    =');wxx(k);  write('      TI = ');wxx(ti);
    write(' TD  = ');wxx(td);
    write('   ALFA =');wxx(alfa);writeln;
  end;{with}
  writeln;
  writeln('REGULATOR POLYNOMIALS');
  with actregpar do
  begin
    write('TPOL =');for i:=0 to 2 do  wxx(tpol[i]);writeln;
    write('SPOL =');for i:=0 to 2 do  wxx(spol[i]);writeln;
```

```
    write('RPOL =');for i:=0 to 2 do  wxx(rpol[i]);writeln
  end{with}
end; {of Disp}
{-------------------------------------------------------------Init}
Procedure Init;
{Changes initial values in the estimator}

var n:integer;
begin
   if eoln then Error(fewarg);
   read(text);
   initarg[initlast]:=text;
   initargx:=xp0;
   while initarg[initargx]<>text do initargx:=succ(initargx);
   if initargx=initlast then Error(illarg);
   if initargx=xidef then Estdefault else
   begin
     if eoln  then Error(fewarg);
     read(value);
     with estinit do
     begin
       case initargx of
         xp0  :if value<0. then Error(illval) else
                begin for n:=1 to 4 do diag[n]:=value end;
         xth1 :theta[1]:=value;
         xth2 :theta[2]:=value;
         xth3 :theta[3]:=value;
         xth4 :theta[4]:=value;
       end;{of case}
     end;{if initarg=}
   end;
end;{of Init}
{-------------------------------------------------------------Par}
Procedure Par;
{Changes parameter values}

begin
  if eoln then Error(fewarg);
  read(text);
  pararg[parlast]:=text;
  parargx:=xtsamp;
  while pararg[parargx]<>text do parargx:=succ(parargx);
  if parargx=parlast then Error(illarg);
  if parargx=xpdef then
    begin newpar:=false;Pardefault;newpar:=true end else
    begin
    if eoln  then Error(fewarg);
    read(value);
    case parargx of
      xtsamp  :if (value<0.) or (value>600.)  then Error(illval) else
               begin newpar:=false;
               tsamp:=round(value/0.02)*0.02;Setdpol;newpar:=true end;
      xom     :if value<0. then Error(illval) else
               begin om:=value;newpar:=false;Setdpol;newpar:=true end;
```

```
      xdamp    :if value<0. then Error(illval) else
                begin damp:=value;newpar:=false;Setdpol;newpar:=true
                end;
      xlam     :if (value <=0.) or (value>1.) then Error(illval) else
                begin newpar:=false;
                newlambda:=value;newpar:=true end;
      xk       :if value<0. then Error(illval) else
                begin newpar:=false;
                pidpar.k:=value; newpar:=true end;
      xti      :if value<=0. then Error(illval) else
                begin newpar:=false;
                pidpar.ti:=value; newpar:=true end;
      xtd      :if value<0. then Error(illval) else
                begin newpar:=false;
                pidpar.td:=value; newpar:=true end;
      xalfa    :if value<=0. then Error(illval) else
                begin newpar:=false;
                pidpar.alfa:=value; newpar:=true end;
      xhilim   :if abs(value)<=1. then begin newpar:=false;
                newhilim:=value;newpar:=true end else Error(illval);
      xlolim   :if abs(value)<=1. then begin newpar:=false;
                newlolim:=value;newpar:=true end else Error(illval);
      xyrch    :if (value<0.) or (value>7.) then Error(illval)
                else if period>0 then Error(runact)
                  else  yrch:=trunc(value);
      xych     :if (value<0.) or (value>7.) then Error(illval)
                else if period>0 then Error(runact)
                  else  ych:=trunc(value);
      xuch     :if (value<0.) or (value>7.) then Error(illval)
                else if period>0 then Error(runact)
                  else  uch:=trunc(value);
      xtrich :if (value<0.) or (value>7.) then Error(illval)
                else if period>0 then Error(runact)
                  else  trich:=trunc( value);
    end;
  end;{if parargx=}
end; {of Par}
{-------------------------------------------------------------Log}
Procedure Log;
{Writes log of past values of inputs,outputs and reference values}

var i,j,k:integer;
var r:real;
begin
  logpar:=false;
  writeln(' Time',' ':7,'u',' ':9,'yr',' ':8,'y');
  j:=-tmax;
  for i:=tlog+1 to tmax do
  begin
    write(j:5,' ');
    for k:=1 to nlog do
    begin
      r:=backlog[i,k];write(' ':4,r:6:Rform(r,6))
    end;
```

```
    writeln; j:=j+1
  end;
  for i:= 0 to tlog do
  begin
    write(j:5,' ');
    for k:=1 to nlog do
    begin
      r:=backlog[i,k];write(' ':4,r:6:Rform(r,6))
    end;
    writeln;j:=j+1
  end;
  logpar:=true
end; {of Log}
{-------------------------------------------------------------Stop}
Procedure Stop;
{Stops the control and puts the control signal equal to zero
 or within the limits}

begin
  period:=0;
  if lolim>0. then u:=lolim else
    begin if hilim<0. then u:=hilim else u:=0. end;
  Daout(uch,u)
end; {of Stop}
{-------------------------------------------------------------Fix}
Procedure Fix;
{Fixes or unfixes  the regulator parameters.
 The estimator is not influenced}

begin
  if eoln then Error(fewarg);
  read(text);
  fixarg[lastindex]:=text;
  fixargx:=xon;
  while fixarg[fixargx]<>text do fixargx:=succ(fixargx);
  case fixargx of
    xon       : ifix:=true;
    xoff      : ifix:=false;
    lastindex: Error(illarg);
  end;{of case}
end; {of Fix}
{-------------------------------------------------------------Run}
Procedure Run;

begin
  if eoln then
    begin
      period:=0;
      ipid:=false;
      Initest;
      Initreg;
    end
  else
    begin
```

```
        read(text);
        runarg[lastx]:=text;
        runargx:=xpid;
        while runarg[runargx]<>text do runargx:=succ(runargx);
        case runargx of
          xpid : begin period:=0;ipid:=true;Initreg end;
          xcont: ipid:=false;
          lastx: Error(illarg);
        end;{of case runargx}
      end;{of else}
    newpar:=true;
    period:=round(tsamp/0.02);
  end; {of Run}

  {-------------------------------------------------Code of Comcom}
  begin{Comcom}
    Initialize;
    repeat
      write('>');
      read(command);
      op[xlastop]:=command;
      opx:=xhelp;
      while op[opx]<>command do opx:=succ(opx);
      case opx of
        xhelp   :Help;
        xrun    :Run;
        xstop   :Stop;
        xdisp   :Disp;
        xfix    :Fix;
        xinit   :Init;
        xpar    :Par;
        xlog    :Log;
        xexit   :exitpar:=true;
        xlastop :Error(illcom);
      end;{of case opx}
        {test if more on the line in each procedure}
  999: readln
    until exitpar;
  end; {of Comcom}


  {=============================================================MAIN}
  {CODE MAIN PROGRAM}
  begin
    period:=0;
    Clksav;
    Schedule(Fg,period);
    Comcom;
    Clkrestore
  end. {of Stupid}
```

```
Program Foreground;
{SELF-TUNING PID-CONTROLLER
 Authors PH BW IG 1979-11-13 Revised BW 1980-05-27 ph 800612
 The program implements a self-tuning PID-controller based on
 pole-placement. The algorithm is described in
 Wittenmark: Self-tuning PID-controllers based on pole-placment,
    TFRT-7179
 Wittenmark-Astrom: Simple self-tuning controllers, TFRT-7191.
 The program is divided into the files:
    GLOBAL-global data declarations
    LS    -least squares estimator
    FG    -foreground program
    STUPID-operator communication and main program}

{GLOBAL DATA DECLARATIONS}

const tmax=20; {length of log}
      eps=0.01; {test quantity for pole-zero cancellation}
      thmin=1.E-4; {min absolute value of theta[3] and theta[4]}
      nlog=3; {number of variables stored in the log}
Type  estpartyp= record
        theta:array[1..4] of real;
        fi:array[1..4] of real;
        diag:array[1..4] of real;
        offdiag:array[1..6] of real
        end;
      regpartyp= record
        tpol:array[0..2] of real;
        spol:array[0..2] of real;
        rpol:array[0..2] of real;
        end;
      pidtyp=record
        k,ti,td,alfa:real
        end;
var
      {To be changed from background only when period=0}
       yrch,ych,uch,trich:integer;
       y,yold,yr,u,uold,lolim,hilim,lambda:real;
       x:array[1..2] of real;
       eststate:estpartyp;
       actregpar:regpartyp;
       dpol: array[0..2] of real;

      {to be changed from background only when newpar=false}
       newlolim,newhilim,newlambda,tsamp:real;
       newdpol:array[0..2] of real;
       pidpar:pidtyp;

      {to be changed from foreground only when logpar=false}
       backlog:array[0..tmax,1..nlog] of real;
       tlog:integer;

      {flags etc}
       ifix,ipid,newpar,logpar:boolean;
```

```
        period:integer;

{EXTERNAL PROCEDURE DECLARATIONS}

Function Rform(r:real; size:integer):integer;external;
Function Adin(chan:integer):real;external;
Procedure Daout(chan:integer;value:real);external;
Procedure Schedule(procedure fore;var period:integer);external;
Procedure Clksav;external;
Procedure Clkrestore;external;

{End of Global}

{Fg Author PH BW IG 1979-11-13, Revised BW 1980-05-27 ph 800612}

Procedure LS(delu,dely:real;var eststate:estpartyp);external;


{=============================================================Fg}
Procedure Fg;

{The procedures are used in the following way
  Regulator
    Adin(external)
    Output
      Ushape
    Daout(external)
    Update
      LS(external)
      Pardet
  Stateset
    Pardet
  Trigosc
  Loguy}

{-----------------------------------------------------------Pardet}
Procedure Pardet(estate:estpartyp;var rpar:regpartyp);
{Determines polynomial representation of regulator from
 estimated parameters and desired characteristic polynomial}

var n,r,max,b1,b2,be,a1,a2,r1:real;
    test:boolean;
begin
  with estate do
  begin
    {Guarantee that not both theta[3] and theta[4] are almost zero}
    if (abs(theta[3])<thmin) and (abs(theta[4])<thmin) then
    begin
       if abs(theta[3])>abs(theta[4])
       then
       if theta[3]>0. then theta[3]:=thmin else theta[3]:=-thmin
       else
       if theta[4]>0. then theta[4]:=thmin else theta[4]:=-thmin
```

```
    end;
    {Test if cancellation of pole and zero in (q-1)A(q) and B(q)}
    n:=theta[4]*theta[4]-theta[1]*theta[3]*theta[4];
    n:=n+theta[2]*theta[3]*theta[3];
    r:=theta[4]*theta[4];
    max:=theta[3]*theta[3];
    if max<r then max:=r;
    r:=theta[3]+theta[4];
    if (abs(n)<eps*max) or (r*r<eps*max)
       then test:=true else test:=false;
    if test then {Common factor}
      begin
        b1:=theta[3];
        b2:=0.0;
        if r*r<eps*max then
        begin {a1 and a2 define the polynomial A(q)}
          a1:=theta[1];
          a2:=theta[2];
        end
        else
        begin {a1 and a2 define the polynomial (q-1)*(q-a)}
          a1:=theta[1]-theta[4]/theta[3]-1.;
          a2:=-a1-1.
        end;
      end
    else {No common factor}
      begin
        b1:=theta[3];
        b2:=theta[4];
        a1:=theta[1];
        a2:=theta[2]
      end;
  end; {with estate do}
  with rpar do
  {Determine the polynomial representation of the regulator}
  begin
    if abs(b1)<thmin then {solve as if b1=0}
    begin
      r1:=dpol[1]-a1+1.;
      spol[0]:=(dpol[2]+a1-a2-r1*(a1-1.))/b2;
      spol[1]:=(a2-r1*(a2-a1))/b2;
      spol[2]:=a2*r1/b2;
    end
    else
    if test then            {common factor}
    begin
      r1:=0.;
      spol[0]:=(dpol[1]-a1)/b1;
      spol[1]:=(dpol[2]-a2)/b1;
      spol[2]:=0.
    end
    else                    {no common factor and b1 large enough}
    begin
      be:=b2/b1;
```

```
      r:=-a2-be*(a2-a1-be*(a1-1.-be));
      {r=0 only if test=true}
      r1:=-be*(a2-be*(dpol[2]-a2+a1-be*(dpol[1]-a1+1.)))/r;
      spol[0]:=(dpol[1]-a1+1.-r1)/b1;
      spol[1]:=(dpol[2]-a2+a1-(a1-1.)*r1-b2*spol[0])/b1;
      spol[2]:=(a2-(a2-a1)*r1-b2*spol[1])/b1;
    end;
    tpol[0]:=spol[0]+spol[1]+spol[2];
    tpol[1]:=0.;
    tpol[2]:=0.;
    rpol[1]:=r1-1.;
    rpol[2]:=-r1;
  end; {with rpar do}
end; {of Pardet}

{-------------------------------------------------------Regulator}
Procedure Regulator;
{Calculates and limits control signal and updates regulator states}

{---------------------------------------------------------Ushape}
Function Ushape(u,lolim,hilim:real):real;
{Limits the control signal}

begin
  ushape:=u;
  if u<lolim then ushape:=lolim;
  if u>hilim then ushape:=hilim;
end; {of Ushape}

{----------------------------------------------------------Output}
Procedure Output;
{Computes control signal from measurements and states}

var w:real;
begin
  with actregpar do
  begin
    w:=x[1]+tpol[0]*yr-spol[0]*y;
    u:=Ushape(w,lolim,hilim);
    x[1]:=u-w+x[1];
  end;
end; {of Output}

{----------------------------------------------------------Update}
Procedure Update;
{Updates process parameters and regulator}

var delu,dely:real;
    nextx:array[1..2] of real;
begin
  delu:=u-uold;
  dely:=y-yold;
  LS(delu,dely,eststate);
  uold:=u;
```

```
  yold:=y;
  if (not ipid) and (not ifix) then Pardet(eststate,actregpar);
  with actregpar do
  {Update the state of the regulator}
  begin
    nextx[1]:=-rpol[1]*x[1]+x[2]+(tpol[1]-tpol[0]*rpol[1])*yr;
    nextx[1]:=nextx[1]-(spol[1]-spol[0]*rpol[1])*y;
    nextx[2]:=-rpol[2]*x[1]+(tpol[2]-tpol[0]*rpol[2])*yr;
    nextx[2]:=nextx[2]-(spol[2]-spol[0]*rpol[2])*y;
  end;
  x[1]:=nextx[1];
  x[2]:=nextx[2];
end; {of Update}

{-------------------------------------------------Code Regulator}
{Code Regulator}
begin
  y:=Adin(ych);
  yr:=Adin(yrch);
  Output;
  Daout(uch,u);
  Update;
end; {of Regulator}

{-----------------------------------------------------Stateset}
Procedure Stateset;
{Change regulator state after parameter change}

var al1,be,ga:real;
begin
  if ipid then
  begin
    with actregpar,pidpar do
    begin
      if ti<1000. then al1:=k*(tsamp/ti-1.) else al1:=-k;
      if tsamp<0.01 then be:=k/alfa else begin if td>0. then
        be:=k*td*(1.-exp(-tsamp/alfa/td))/tsamp else be:=0. end;
      if td>0. then ga:=-exp(-tsamp/(alfa*td)) else ga:=0.;
      tpol[0]:=k;
      tpol[1]:=al1+k*ga;
      tpol[2]:=al1*ga;
      spol[0]:=tpol[0]+be;
      spol[1]:=tpol[1]-2.*be;
      spol[2]:=tpol[2]+be;
      rpol[1]:=ga-1.;
      rpol[2]:=-ga;
    end;
  end
  else
  begin
    dpol:=newdpol;
    if (not ifix) then Pardet(eststate,actregpar);
  end;
     {note that it is not possible to change the controller
```

```
      by changing the D-polynomial during FIX ON}
  with actregpar do
  begin
    x[1]:=u-tpol[0]*yr+spol[0]*y;
    x[2]:=(1.+rpol[1])*x[1]-(tpol[1]-tpol[0]*rpol[1])*yr;
    x[2]:=x[2]+(spol[1]-spol[0]*rpol[1])*y
  end;
end; {of Stateset}

{-------------------------------------------------------------Trigosc}
Procedure Trigosc;
{Generates signal to trig oscilloscope}

var r:real;
begin
  if yr>0. then r:=0.5 else r:=-0.5;
  Daout(trich,r);
end; {of Trigosc}

{--------------------------------------------------------------Loguy}
Procedure Loguy;
{Store control variable, u, output, y, and reference value, yr, in a
 cyclic file called backlog[i,j]. Current time is at i=tlog previous
 sampling at i=tlog-1}

var i:integer;

begin
  tlog:=(tlog+1) mod (tmax+1);
  backlog[tlog,1]:=u;
  backlog[tlog,2]:=yr;
  backlog[tlog,3]:=y;
  if nlog>6 then for i:=1 to 4 do
    backlog[tlog,3+i]:=eststate.theta[i];
  if nlog>10 then for i:=1 to 4 do
    backlog[tlog,7+i]:=eststate.diag[i]
end; {of Loguy}

{-------------------------------------------------------------Code FG}
{CODE FG}
begin
  Regulator;
  Trigosc;
  if newpar then begin
     period:=round(tsamp/0.02);
     lambda:=newlambda;
     hilim:=newhilim;
     lolim:=newlolim;
     Stateset;
     newpar:=false
  end;
  if logpar then Loguy;
end; {of Fg}
```

```
Program LS;
{SELF-TUNING PID-CONTROLLER
 Authors PH BW IG 1979-11-13 Revised BW 1980-05-27 ph 800612
 The program implements a self-tuning PID-controller based on
 pole-placement. The algorithm is described in
 Wittenmark: Self-tuning PID-controllers based on pole-placment,
    TFRT-7179
 Wittenmark-Astrom: Simple self-tuning controllers, TFRT-7191.
 The program is divided into the files:
    GLOBAL-global data declarations
    LS     -least squares estimator
    FG     -foreground program
    STUPID-operator communication and main program}

{GLOBAL DATA DECLARATIONS}

const tmax=20; {length of log}
      eps=0.01; {test quantity for pole-zero cancellation}
      thmin=1.E-4; {min absolute value of theta[3] and theta[4]}
      nlog=3; {number of variables stored in the log}
Type  estpartyp= record
        theta:array[1..4] of real;
        fi:array[1..4] of real;
        diag:array[1..4] of real;
        offdiag:array[1..6] of real
        end;
      regpartyp= record
        tpol:array[0..2] of real;
        spol:array[0..2] of real;
        rpol:array[0..2] of real;
        end;
      pidtyp=record
        k,ti,td,alfa:real
        end;
var
      {To be changed from background only when period=0}
       yrch,ych,uch,trich:integer;
       y,yold,yr,u,uold,lolim,hilim,lambda:real;
       x:array[1..2] of real;
       eststate:estpartyp;
       actregpar:regpartyp;
       dpol: array[0..2] of real;

      {to be changed from background only when newpar=false}
       newlolim,newhilim,newlambda,tsamp:real;
       newdpol:array[0..2] of real;
       pidpar:pidtyp;

      {to be changed from foreground only when logpar=false}
       backlog:array[0..tmax,1..nlog] of real;
       tlog:integer;

      {flags etc}
       ifix,ipid,newpar,logpar:boolean;
```

```
        period:integer;

{EXTERNAL PROCEDURE DECLARATIONS}

Function Rform(r:real; size:integer):integer;external;
Function Adin(chan:integer):real;external;
Procedure Daout(chan:integer;value:real);external;
Procedure Schedule(procedure fore;var period:integer);external;
Procedure Clksav;external;
Procedure Clkrestore;external;

{End of Global}

Procedure LS(delu,dely:real;var eststate:estpartyp);
{Computes the least squares estimate of the parameters of a second
order system using the U/D method after Bierman and Thornton.
Author IG 1979-11-13}

const n=4;

var kf,ku,i,j:integer;
    perr,fj,vj,alphaj,ajlast,pj,w:real;
    k:array[1..4] of real;

begin
  perr:=dely;
  with eststate do
  begin
    for i:=1 to n do perr:=perr-theta[i]*fi[i];
    fj:=fi[1];
    vj:=diag[1]*fj;
    k[1]:=vj;
    alphaj:=1.0+vj*fj;
    diag[1]:=diag[1]/alphaj/lambda;
    if n>1 then
    begin
      kf:=0;
      ku:=0;
      for j:=2 to n do
      begin
        fj:=fi[j];
        for i:=1 to j-1 do
        begin
          kf:=kf+1;
          fj:=fj+fi[i]*offdiag[kf]
        end; {i}
        vj:=fj*diag[j];
        k[j]:=vj;
        ajlast:=alphaj;
        alphaj:=ajlast+vj*fj;
        diag[j]:=diag[j]*ajlast/alphaj/lambda;
        pj:=-fj/ajlast;
        for i:=1 to j-1 do
```

```
        begin
          ku:=ku+1;
          w:=offdiag[ku]+k[i]*pj;
          k[i]:=k[i]+offdiag[ku]*vj;
          offdiag[ku]:=w
        end; {i}
      end; {j}
    end; {if n>1 then}
    for i:=1 to n do theta[i]:=theta[i]+perr*k[i]/alphaj;
    {Updating of fi-vector. Must be changed if n is changed}
    fi[4]:=fi[3];
    fi[3]:=delu;
    fi[2]:=fi[1];
    fi[1]:=-dely
  end {with eststate do}
end; {LS}
```