Anhang B

Softwarelistings

Eigentlich hatte ich mir in der Vorbereitung meiner Arbeit gedacht, in diesem Teil meiner Arbeit die Quellen aller Softwarekomponenten einzeln aufzuführen. Inzwischen ist mir klar geworden, daß das im Nu weitere 100 Seiten füllen würde. Aus diesem Grund werde ich nur die wichtigsten Quellen hier aufführen. Der komplette Satz ist auf der beiliegenden CD-ROM enthalten.

Die aufgeführten MatLAB 4.0 - und TurboPascal 7.0 - Programme sind nach den Kapiteln geordnet, in denen sie verwendet werden. MatLAB- Programme haben die Endung ***.M***, TurboPascal- Programme die Endung ****.PAS***.

Einsatz findet die Schriftart `Courier New` , da diese eine konstante Zeichenbreite besitzt.

2.2. Simulationsmodell

Die SimuLINK- Modelle, die ich in diesem Abschnitt verwende, sind mittels Drag&Drop entstanden. Die dabei automatisch erstellten Quellen sind komplett in Englisch gehalten, es lohnt sich also nicht, diese hier aufzuführen.
Ich möchte hier nur die MatLAB- Datei aufführen, die die verwendeten Parameter für die Simulation der Belastungseinrichtung enthält.

STF_H733.M :

```
%STF_H733 Dieses Programm stellt die nötigen Parameter für
%      den Hydraulikkreislauf STFHY792 zur Verfügung,
%      da eine Simulink-Datei keine eigenen Werte setzen kann.
%
%      Die Belastungseinrichtung wird mit einem PL-Regler, der mit
%      STF_H708 (5ms-Tastzeit) bestimmt wurde, geregelt.
%      Die Strecke hat die Parameter :
%            Ts = 15 ms, Ds = 0.65, Ks = 60
%
%      Alle Parameter entsprechen STF_H723.
%
%      Hier werden die Leitwerte und Kapazitäten entprechend einer
%      Öltemperatur von 70°C korrigiert.
%            GFaktor = 1/0.53;
%            bFaktor = 1.06;
%
%      Der Regler arbeitet mit einer Tastzeit von 15ms.
%
% SIEHE STF_H702..STF_H732, STF_HY77, STF_HY78, STF_HY79, STFHY790, STF_PL
%      STFHY782, STFHY792

% Steffen Leßke
% Triebes, den 17.8.1997

% Werte für die Belastungseinrichtung
p0     = 245;              % Betriebsdruck in bar
Q0     = 200;              % Förderstrom in l/min
W0     = 18.5;             % Pumpenleistung in kW
pmax   = 250;              % Max. Druck in bar
GFaktor      = 1/0.53;     % Temp.abh. für G T=70°C
bFaktor      = 1.06;       % Temp.abh. für beta T=70°C
```

```
% Werte für die Konstantdruckpumpe
Gpleck = 0.01 * GFaktor;   % Leckleitwert in l/min*bar
Gpventil= 1000 * GFaktor;  % diff. Ventilleitwert in l/sqrt(bar)*min
Qpmax  = 100;              % max. Ventilstrom in l/min
Hp     = 4;                % Druckhysterese in bar
TtQ    = 0.05;             % Tiefpaßzeitkonstante in s
KEH    = 1/0.00167;        % Umrechnung von kW nach bar*l/min
Tp     = 0.01;             % Zeitkonstante für die Druckbegrenzung
Dp     = 0.7;              % Dämpfung für die Druckbegrenzung
pdpmax = 5;                % Druck zum maximalen Öffnen des Ventils
ppmin  = 50;               % minimaler Druck in bar

% Werte für die elastische Leitung
Gl     = 30 * GFaktor;     % Leitungsverlustleitwert in l/sqrt(bar)*min
planf  = p0-5;             % Anfangsdruck in bar
Qlmax  = 500;              % max. Leitungsstrom in l/min
V0l    = 2010600;          % Leitungsvolumen in mm³
betaS  = 3.5e-4 * bFaktor; % Kompr.faktor für den Schlauch

% Werte für die Druckbegrenzungsventile
Qvmax  = 100;              % max. Ventilstrom in l/min
Gvventil= 10 * GFaktor;    % diff. Ventilleitwert in l/sqrt(bar)*min
Hv     = 5;                % Druckhysterese in bar
Tvp    = 0.01;
Dvp    = 0.7;
pvdpmax      = 10;

% Werte für das Prop-Einbauventil 1
Ustmax = 10;               % max. Steuerspannung
Tser   = 0.010;            % 0,0007 Zeitkonstante des Schieberwegs
Dser   = 0.7;              % Dämpfung des Schieberwegs
y0     = -0.0025;          % 0.005 Überdeckung
G1max  = 143.1 * GFaktor;  % 3.38 max. Leitwert in l/sqrt(bar)*min
G3max  = G1max;
y_vek  = [0 0.1 0.3 0.7 0.9 1];  % Wertetabelle Kolbenstellung
q_vek  = [0 0.2 0.5 0.9 0.98 1]; % Wertetabelle Leitwert
Qmax   = 1450;                   % maximaler Ventilstrom in l/min

% Vorgabe der Parameter für den Arbeitszylinder 6
A1     = 49087;            % Kolbenfläche 1 in mm²
A2     = 7540;             % Kolbenfläche 2 in mm²
m      = 133;              % Masse in kg
Fan    = 500;              % 200 Anschlagfaktor in kN/mm
Fch    = 1.5;              % Abklingkonstante für Haftreibung
Frh    = 0.1;              % Haftreibungskraft in kN
Frc    = 0.1;              % coulombsche Reibung in kN
Frv    = 0.025;            % Geschw.reibung in kNs/mm
G1     = 73.6*1000 * GFaktor;     % 120 Verlustleitwert 1 in l/bar*min
G2     = 11.3*1000 * GFaktor;     % 100 Verlustleitwert 2 in l/bar*min
Gleck  = 0.005 * GFaktor;  % Leckleitwert in l/bar*min
xmax   = 250;              % max. Kolbenweg in mm
beta   = 1e-4 * bFaktor;   % Kompressibilitätsfaktor in 1/bar
V001   = 339292;           % Anfangsvolumen 1 in mm^3
V002   = 57241;            % Anfangsvolumen 2 in mm^3
p1anf  = 20;               % 52.32 Integratoranfangsdruck 1 in bar
p2anf  = p0-5;             % 106.17 Integratoranfangsdruck 2 in bar

KFa    = 1e6;              % Faktor kN -> mm/s^2
KAQ    = 6e-5;             % Faktor mm^2 -> l/min
Kpp    = 1/60;             % Faktor bar/min -> bar/s
KVV    = 1e-6;             % Faktor mm^3 -> l
KAF    = 1e-4;             % Faktor mm^2 -> kN

% Parameter der Meßsysteme
Tp     = 0.002;            % Zeitkonstante der Drucksensoren
Kp     = 1;                % Verstärkung der Drucksensoren
Qp     = 400 / (2^12);     % Druckauflösung in bar
Tx     = 0.001;            % Zeitkonstante des Wegmeßsystems
Kx     = 1;                % Verstärkung des Wegmeßsystems
Qx     = 0.01;             % Meßsystemauflösung in mm
% Parameter für den Schwingungsversuch
K_start      = 0.5;        % Verstärkung zum Einschwingen
```

```
K_min  = 1;                % min. Verstärkung
K_max  = 7.5;              % max. Verstärkung
t_anf  = 0.3;              % Beginn des Versuchs
t_max  = 5;                % Ende des Versuchs
t_step = 0.005;            % Tastperiode
xSW    = 0;                % Positions-Sollwert in mm
vSW    = 1;                % Geschwindigkeits-Sollwert in mm/s

K_step = (K_max-K_min)*t_step/(t_max-t_anf);

disp('Verstärkungsverlauf wird berechnet ...');
t_schwing = 0:t_step:t_max;
t_schwing = t_schwing';
K_schwing = zeros(size(t_schwing));
K_schwing(1,1) = K_start;
for i=2:size(K_schwing,1),
   if t_schwing(i,1)>t_anf,
      K_schwing(i,1) = K_schwing(i-1,1) + K_step;
   elseif t_schwing(i,1) == t_anf,
      K_schwing(i,1) = K_min;
   else
      K_schwing(i,1) = K_start;
   end;
end;

% Parameter für den berechneten PL-Regler
% mit Kt=2, F95=1 und Dg=0.9 und Tregler = 0.015s
%
Ttast  = 0.015;
Tregler      = 0.015;
[alpha1,alpha2,KB,c1,c2,cw1,cw2,TgPL,TfPL]=stf_pl(0.015,0.65,60,...
   0.9,2,1,Tregler);
Uoffs  = -0.024;           % ermittelte Offsetspannung

% Parameter für den automatischen Driftabgleich
Tstart = 0.6;              % Startzeit
Tstop  = 1.5;              % Stopzeit
Uschranke    = 0.02;       % Schranke für das automatische Ende
Uanstieg     = 0.002;      % Anstieg je Tastschritt

% Parameter für den Analogausgang
Qanalog      = 10 / (2^15);      % Ausgangsauflösung in V
```

Weiterhin ist die Darstellung der Stribek-Reibung interessant. Diese Funktion muß in einem SimuLINK- Block „MatLAB-Function“ aufgerufen werden. Für die Darstellung der Reibkraft über der Zeit kann die Funktion wie z.B. in ***STFREIB1.M*** aufgerufen werden.

REIBUNG.M

```
function F=reibung(Fh,Fc,Fv,vtau,v)
%REIBUNG F=reibung(Fh,Fc,Fv,vtau,v)
%        Diese Funktion berechnet die Stribek-Reibung eines
%        Hydraulikzylinders.
%
%        Dieses Reibung hat drei Komponenten :
%         - Haftreibung F=Fh bei v=0
%           fällt nach einer e-Funktion ab exp(1-(v/vtau))
%         - Coulombsche Reibung Fc
%           ist immer konstant für v<>0
%         - Geschwindigkeitsreibung Fv bei v=1
%           ist proportional zur Geschwindigkeit
%
%        Für die Bestimmung der Halftreibung wird die Abfall-
%        zeitkonstante vtau benötigt. Nach v>5.5*vtau ist die
%        Haftreibung gleich null.
%
%        Die Funktion kann nicht für Matrizen angewendet werden.
%
%        Diese Funktion ist grob nichtlinear und sollte
```

```
%        bei SIMULINK als Block "MATLAB Fcn" vereinbart werden.
%
% SIEHE ADD_BLOCK, SIMULINK, FUNCTION, STF_REIB

% Steffen Leßke
% Triebes, den 25.11.1995

F = Fc * sign(v) + ...                          % Coulombr.
    Fv * v;                                     % Geschw.r.

if abs(v) < (5.5*vtau),                         % zur Verkürzung der Rechenzeit
   F = F + Fh * exp(- (abs(v) / vtau)) * sign(v); % Haftreibung
end;
```

Auch ein turbulenter Leitwert ist mit normalen SimuLINK- Mitteln nicht darstellbar. Dafür habe ich die Funktion ***TURBULNT.M*** geschrieben.

TURBULNT.M :

```
function y = turbulnt(x);
%TURBULNT Diese Funktion berechnet den angepaßten Druckabfall
%      über einer turbulenten Drossel zur Ermittlung des durch-
%      fließenden Stromes nach der Formel :
%
%      Q = G * y (G - hydr. Leitwert)
%
%      mit y = sqrt(abs(p1-p2))*sign(p1-p2)
%
%      Mit der Funktion TURBULNT lautet die Gleichung also:
%
%      Q = G * turbulnt((p1-p2));
%
%      Die Funktion ist zum Einsatz in SIMULINK gedacht. Sie soll
%      dabei im Block "MATLAB-Fcn" eingesetzt werden, um turbulente
%      Durchstömungen zu simulieren.
%      Der Block "Function" kann nicht eingesetzt werden, da er die
%      Funktion "SIGN" nicht kennt. ABS(p1-p2)/(p1-p2) kann auch nicht
%      eingesetzt werden, da es bei einer Druckdifferenz von 0 zu falschen
%      Ausgangswerten kommt.
%
% SIEHE SIGN, SQRT, SIMULINK

% Steffen Leßke
% Triebes, den 21. Juli 1997

y = sqrt(abs(x))*sign(x);
```

2.4.2. Schwingungsversuch

Für die Auswertung des Schwingungsversuches habe ich ein kleines MatLAB- Programm geschrieben, das aus den ermittelten Kurven für Verstärkungs-, Zeit- und Stellgrößenverlauf die entsprechenden Verläufe für Periodendauer, Zeitkonstante und Dämpfung berechnet und darstellt.

STF_SW.M :

```
function [sT0,sD,sT] = stf_sw(y,stast,y_a,y_e,Kr,Ks,Tt);
%STF_SW Diese Funktion berechnet Dämpfung und Zeitkonstante
%      aus einem gegebenen Werteverlaufs eines Schwingungs-
%      versuchs und stellt sie grafisch dar.
%
%      Aufruf : [sT0,sD,sT] = stf_sw(y,stast,y_a,y_e,Kr,Ks,Tt);
%            y -   Wertefolge, die mit stast getastet wurde
%            stast -      Tastzeit
%            y_a -  Anfangstakt
%            y_e -  Endtakt
%            Kr -   kritische Reglerverstärkung
```

```
%            Ks -   Streckenverstärkung
%            Tt -   Totzeit
%
% SIEHE STF_SCHW, STF_SWMX, STF_PLVS

% Steffen Leßke
% Triebes, den 26.7.1997

T0 = zeros(size(y));             % Periodendauerfolge
D = T0;                          % Dämpfungsfolge
T = T0;                          % Zeitkonstantenfolge

disp('Das kann jetzt einige Minuten dauern ...');
for i=2:size(y,1),
   T0(i,1)=stf_swmx(stast,y,i,y_a,y_e); % Periodendauerberechnung
   [T(i,1),D(i,1)]=stf_schw(T0(i,1),Ks,Kr,Tt); % Konstantenberechnung
end;
disp('Jetzt erfolgt die Anzeige des Periodendauerverlaufs');
pause
stf_plvs(T0,stast,'Zeit in s','Periodendauer in s');
disp('Jetzt erfolgt die Anzeige des Zeitkonstantenverlaufs');
pause
stf_plvs(T,stast,'Zeit in s','Zeitkonstante in s');
disp('Jetzt erfolgt die Anzeige des Dämpfungsverlaufs');
pause
stf_plvs(D,stast,'Zeit in s','Dämpfung');

if nargout == 3,
   sT0 = T0;
   sT = T;
   sD = D;
end;
```

Wie zu sehen ist, werden weiterhin ***STF_SCHW.M*** und ***STF_SWMX.M*** benötigt.

STF_SCHW.M :

```
function [sT,sD] = stf_schw(sT0,sKs,sKr,sTt);
%STF_SCHW Diese Funktion berechnet Zeitkonstante und Dämpfung
%      eines Schwinggliedes aus den Werten des Schwingversuchs.
%
%      Aufruf : [sT,sD] = stf_schw(sT0,sKs,sKr,sTt);
%             sT -   Zeitkonstante des Schwinggliedes
%             sD -   Dämpfung des Schwinggliedes
%             sT0 -  abgelesene Periodendauer
%             sKs -  Streckenverstärkung
%             sKr -  Reglerverstärkung
%             sTt -  Streckentotzeit
%
% SIEHE STF_SW

% Steffen Leßke
% Triebes, den 5.11.1996

if sTt == 0,               % extreme Vereinfachung ohne Totzeit
   sT = sT0/(2*pi);
   sD = sKr*sKs*sT0/(4*pi);
   return;
end;
somega = 2*pi/sT0;
sT = 1/somega * sqrt(1-(sKr*sKs*sin(somega*sTt)/somega));
sD = sKr*sKs*cos(somega*sTt)/(2*sT*somega^2);
```

STF_SWMX.M :

```
function sT0 = stf_swmx(sTtast,su,si,sa,se);
%STF_SWMX Diese Funktion berechnet die Peiodendauer aus einem
%      Zeitverlauf.
%
%      Aufruf : sT0 = stf_swmx(sTtast,su,si,sa,se);
%             sT0 -  ermittelte Periodendauer
%             sTtast       Tastzeit
```

```
%            su -   gemessene Wertefolge
%            si -   absoluter Takt
%            sa -   Anfangstakt
%            se -   Endtakt
%
% SIEHE STF_SW

% Steffen Leßke
% Triebes, den 5.11.1996

global z_laeuft;          % globale Variable festlegen
global z_wechsel;
global z_zaehler;
global z_T;
if si<sa,                 % aktueller Takt vor Meßintervall
   z_laeuft = 0;
   sT0 = 0;
   return;
end;
if si>=se,                % aktueller Takt nach Meßintervall
   clear z_laeuft z_wechsel z_zaehler z_T
   sT0 = 0;
   return;
end;
if si==sa,                % definierte Anfangsbedingung
   z_laeuft = 0;
end;
if z_laeuft == 0,
   if su(si,1)<su(si-1,1),% erster Start beim Auftreten
      z_laeuft = 1;       % der ersten fallenden Flanke
      z_zaehler = 1;
      z_T = sTtast;
      z_wechsel = 0;
   else z_T = 0;
   end;
   sT0 = z_T;
   return;
end;
if su(si,1)<su(si-1,1),          % Fallende Flanke
   if z_wechsel == 0,            % Flanke fällt zum ersten Mal
      z_zaehler = z_zaehler + 1;
      sT0 = z_T;
      return;
   end;
   z_T = z_zaehler * sTtast;     % Flanke fällt zum zweiten Mal
   sT0 = z_T;             % also ist eine Periode voll
   z_zaehler = 1;         % neue Startbedingungen
   z_wechsel = 0;
   return;
end;
z_wechsel = 1;                   % wenn alles andere ausscheidet, dann
z_zaehler = z_zaehler + 1;       % muß die Flanke also steigen
sT0 = z_T;
```

3.2.1. Die sinoide Übergangsfunktion

Für den beschriebenen Sollwertgeber existiert eine MatLAB- und eine TurboPascal-Version. Die TurboPascal- Version ist komplett mit Initialisierung und Berechnung sofort ein einem Echtzeitprogramm einsetzbar.

SINOID.M :

```
function a = sinoid(a_max,v_max,x_soll,T0,t);
%STF_SIN7 Dieses Programm berechnet einen sinoiden
%     Übergangsvorgang, wie er als Lagesollwert
%     verwendet werden kann, um ein ruckfreies
%     Positionieren zu ermöglichen.
%
```

```
%      Der Lagesollwert kann über die Variable
%      x_soll vorgegeben werden.
%
%      Berechnet wird der Beschleunigungsverlauf
%      aus dem dann durch Integration die Geschw.
%      und der Weg gewonnen wird.
%      Die Beschleunigung berechnet sich nach der
%      Formel : a = (v_max/2)*w*sin(w*t)
%
%      Der Lösungsweg des Programms STF_SIN3.M kann
%      durch Rechenungenauigkeiten bei der Sinus-
%      berechnung nicht empfohlen werden.
%
%      Bei kleinen Wegen wird die Geschw. ent-
%        sprechend angepaßt, um ein Springen der
%        Beschleunigung zu verhindern.
%
% SIEHE SIF_SIN3

% Steffen Leßke
% Triebes, den 9.4.1996


% 1. Berechnung v_soll
v_soll = sqrt(2*a_max*x_soll/pi);

% 2. Test, v_soll>v_max ?
if v_soll>v_max,
   v_soll = v_max;
end;

% 3. Ta und w berechnen
Ta = pi * v_soll / (2*a_max);
w =  pi / Ta;

% 4. Tk berechnen
Tk = (x_soll/v_soll)-Ta;

% 5. Test, Tk<0
if Tk <0,
   Tk = 0;
end;

% 8. Beschleunigungsvektor berechnen
if (t<(T0+Ta)) & (t>T0),
   a = a_max * sin(w * (t-T0));
elseif (t>(T0+Ta+Tk)) & (t<=(2*Ta+Tk+T0)),
   a = a_max * (-1) * sin(w *(t-Ta-Tk-T0));
else a=0;
end;
```

REG_SIN1.PAS :

```
{*
*   Diese Unit enthält die Implementation eines Sinoiden Lagesollwertes.
*
*   In Abhängigkeit von v_max und a_max wird ein sinusförmiger Beschleu-
*   nigungsverlauf erzeugt und durch Integration ein ruckfreier Lagesoll-
*   wert erzeugt.
*
*   Für die Berechnung eines Sin-Schrittes werden auf einem 486DX4-100
*   0,013 ms benötigt.
*
*   Steffen Leßke
*   Triebes, den 5. Januar 1997
*}
unit Reg_Sin1;

{$G+,N+,D-,S-,R-,V-}

interface

uses Reg_Typ;
```

```
type PSinSollWert = ^OSinSollWert;
     OSinSollWert = object
        a_max,a_w    : zahl1;          { Max- und Ist-Beschl. }
        v_max,v_soll,v_w : zahl1;      { Max-,Soll- und Ist-Geschw. }
        x_delta,x_w  : zahl1;          { Wegdifferenz und Wegistwert }
        x_soll       : zahl1;          { Wegsollwert }
        t_tast       : zahl1;          { Tastzeit }
        Ta,Tk        : zahl1;          { Anlauf- und stationäre Zeit }
        omega        : zahl1;          { Kreisfreq. der Sinkurve }
        t            : zahl1;          { Zeitverlauf }
        laeuft       : boolean;        { läuft die Kurve gerade ? }
        negativ      : boolean;        { Verfahrrichtung negativ }
        constructor Init(Aa_max,Atast: zahl1);
        destructor  Done;virtual;
        function    SWLaeuft: boolean; virtual;
        function    GetXIst: zahl1; virtual;
        function    GetVIst: zahl1; virtual;
        function    SetXSoll(Ax_soll: zahl1): boolean; virtual;
        function    SetX(Ax_soll: zahl1): boolean; virtual;
        function    SetVMax(Av_max: zahl1): boolean; virtual;
        function    SWStart: boolean;  { Kurve starten }
        procedure   SWReset; virtual;  { Kurve rücksetzen }
        procedure   SinStep; virtual;  { Berechnung eines Schrittes }
        procedure   BerechneZeiten; virtual; { Anlaufzeiten ... }
     end;

implementation

constructor OSinSollWert.Init;
   begin
      a_max := Aa_max;
      t_tast := Atast;
      SWReset;                         { SWkurve rücksetzen }
      SetX(0);                         { Anfangswert x_w = 0 }
      SetVMax(0);                      { Anfangsgeschw. = 0 }
      SetXSoll(0);                     { Wegsollwert = 0 }
   end;
destructor OSinSollWert.Done;
   begin
   end;
function OSinSollWert.SWLaeuft;
   begin
      SWLaeuft := laeuft;
   end;
function OSinSollWert.GetXIst;         { Lageistwert }
   begin
      GetXIst:= x_w;
   end;
function OSinSollWert.GetVIst;         { Geschw.istwert }
   begin
      GetVIst := v_w;
   end;
function OSinSollWert.SetXSoll;
   begin
      if laeuft then begin
         SetXSoll := false;
         Exit;
      end;
      x_delta := Ax_soll - x_w;        { Wegdiff. für SWKurve berechnen }
      x_soll := AX_soll;               { Endsollwert }
      negativ := false;                { Verfahrrichtung }
      if x_delta < 0 then begin
         x_delta := Abs(x_delta);
         negativ := true;
      end;
      BerechneZeiten;
      SetXSoll := true;
   end;
function OSinSollWert.SetX;            { Setzt den absoluten Istwert }
   begin
      if laeuft then begin
```

```
         SetX := false;
         Exit;
      end;
      x_w := Ax_soll;
      BerechneZeiten;
      SetX := true;
   end;
function OSinSollWert.SetVMax;          { Setzen der max. Geschw. }
   begin
      if laeuft then begin
         SetVMax:= false;
         Exit;
      end;
      v_max := Av_max;
      BerechneZeiten;
      SetVMax := true;
   end;
procedure OSinSollWert.BerechneZeiten; { Werte für die Beschl.kurve berechnen }
   begin
      v_soll := sqrt(2*a_max*x_delta/pi);{ stationäre Geschw. berechnen }
      if v_soll > v_max then v_soll := v_max; { Begrenzen }
      Ta := pi * v_soll / (2*a_max);   { Anlaufzeit }
      if v_soll = 0
         then begin
            Tk := 0;
            omega := 0;
         end
         else begin
            omega := pi / Ta;
            Tk := (x_delta/v_soll) - Ta; { stationäre Phase }
            if Tk < 0 then Tk := 0;    { Begrenzen }
         end;
   end;
procedure OSinSollWert.SWReset;         { Sollwertkurve rücksetzen }
   begin
      laeuft := false;                  { Berechnung abbrechen }
      x_delta := 0;                     { Sollwertdiff = 0 setzen }
      t := 0;                           { Zeitpunkt = 0 setzen }
      a_w := 0;                         { Beschl. = 0 setzen }
      v_w := 0;                         { Geschw. = 0 setzen }
      negativ := false;                 { Verfahrrichtung }
   end;
function OSinSollWert.SWStart;          { Sollwertberechnung starten }
   begin
      if laeuft then begin
         SWStart := false;
         Exit;
      end;
      t := 0;                           { Starteinstellungen }
      a_w := 0;
      v_w := 0;
      laeuft := true;                   { Berechnung starten }
      SWStart := true;
   end;
procedure OSinSollWert.SinStep;         { Berechnet eine Schritt eines Wertes }
   begin
      if not laeuft then Exit;
      t := t + t_tast;                  { neue Abtastung }
      if t <= Ta
         then a_w := a_max * sin(omega * t)
         else if (t > (Ta+Tk)) and (t <=((2*Ta) + Tk))
            then a_w := - a_max * sin(omega * (t-((Ta)+Tk)))
            else a_w := 0;
      if negativ then a_w := - a_w;     { negative Verfahrrichtung }
      v_w := v_w + (t_tast * a_w);      { Integration der Besch. zur Geschw. }
      x_w := x_w + (t_tast * v_w);      { Integration der Geschw. zum Weg }
      if t > ((2*Ta)+Tk) then begin     { Abbruchbedingung }
         laeuft := false;
         v_w := 0;
         a_w := 0;
         x_w := x_soll;                 { Korrektur des Endwertes }
      end;
```

```
   end;

end.
```

3.3. PL-Regler

Zu diesem Punkt gibt es zwei Sachen zu unterscheiden :

1. der Reglerentwurf
2. der eigentliche Regelalgorithmus

STF_PL.M und ***STF_PL1.M*** sind zwei MatLAB- Programme für den Entwurf eines PL-Reglers. Die TurboPascal- Variante wird ausführlich im Abschnitt 5. beschrieben.

STF_PL.M :

```
function [d1,d2,K,c1,c2,cw1,cw2,Tg,Tf]=stf_pl(Ts,Ds,Ks,Dg,Kt,F95,T);
%STF_PL Diese Funktion führt einen Reglerentwurf für den diskreten
%      PL-Regler durch.
%      Der PL-Regler ist gemäß Dr. Ralf Neumann aufgebaut.
%
%      Aufruf : [d1,d2,K,c1,c2,cw1,cw2,Tg,Tf]=stf_pl(Ts,Ds,Ks,Dg,Kt,F95,T);
%             Ts  - Streckenzeitkonstante
%             Ds  - Streckendämpfung
%             Ks  - Streckenverstärkung
%             Dg  - gewünschte Dämpfung
%             Kt  - Zeitkonstantenverhältnis Tg/Tf (2...10)
%             F95 - Geschw.faktor (0.1...1)
%             T   - Tastperiodendauer
%
% SIEHE STF_PS2Z, STF_PL1

% Steffen Leßke
% Triebes, den 18.8.1997

if F95<0.1 | F95>1,              % Faktor begrenzen
   F95 = 1;
end;
Tf = 0.5 * Ts * F95 *1.1;        % Filterzeitkonstante
%if Tf<T,                        % Tf >= T
%   Tf = T;
%end;
if Kt<2 | Kt>10,                 % Verhältnis begrenzen
   Kt = 2;
end;
Tg = Kt * Tf;                            % gewünschte Zeitkonstante
[b1,b2,a1,a2] = stf_ps2z(Ts,Ds,Ks,T);    % HG(z) der Strecke
[cw1,cw2,c1,c2] = stf_ps2z(Tg,Dg,1,T);   % HG(z) gewünscht
zi = exp(-T/Tf);                         % 3fach-Pol des inneren Kreises
d2 = zi^3;                               % Filterparameter 2
K = (1-d2+(3*zi^2)-(3*zi))/(b1+b2);      % Reglerverstärkung
d1 = 1-(3*zi)-(K*b1);                    % Filterparameter 1
```

STF_PL1.M :

```
function [d1,d2,K,c1,c2,cw1,cw2]=stf_pl1
%STF_PL1 Diese Funktion führt einen interaktiven Reglerentwurf eines
%      diskreten PL-Reglers nach Dr. Ralf Neumann durch.
%
%      Benötigt werden die Variablen Ts,Ds,Ks,Dg,Kt,F95,T. Sind sie nicht
%      vorhanden, so werden sie erzeugt. In ihnen werden die interaktiven
%      Eingaben gespeichert.
%
%      Aufruf : [d1,d2,K,c1,c2,cw1,cw2]=stf_pl1;
%
% SIEHE STF_PL

% Steffen Leßke
```

```
% Triebes, den 5. August 1997


% Zuerst wird geprüft, ob die Variablen vorhanden sind, wenn nicht,
% dann werden sie auf Standardwerte gesetzt

if exist('Ts')~=1,              % Streckenzeitkonstante
   Ts = 2.8e-3;                 % auf 2.8 ms setzen
end;
if exist('Ds')~=1,              % Streckendämpfung
   Ds = 0.56;                   % auf 0.56 setzen
end;
if exist('Ks')~=1,              % Streckenverstärkung
   Ks = 25.8;                   % auf 25.8 setzen
end;
if exist('Dg')~=1,              % gewünschte Dämpfung
   Dg = 0.7071;                 % auf 0.7071 setzen
end;
if exist('Kt')~=1,              % Zeitkonst.faktor
   Kt = 2;                      % auf 2 setzen
end;
if exist('F95')~=1,             % Faktor
   F95 = 1;                     % auf 1 setzen
end;
if exist('T')~=1,               % Tastperiodendauer
   T = 0.001;                   % auf 1 ms setzen
end;

% Jetzt erfolgen die interaktiven Eingaben

disp('Interaktiver PL-Reglerentwurf');
disp('   ');
disp('ENTER - keine Änderung der Werte');
disp('   ');

a=input(['Streckenzeitkonstante Ts  (',num2str(Ts),')     :'],'s');
if size(a,2) > 0,
  Ts = str2num(a);
end;
a=input(['Streckendämpfung Ds  (',num2str(Ds),' 0...1)    :'],'s');
if size(a,2) > 0,
  Ds = str2num(a);
end;
a=input(['Streckenverstärkung Ks  (',num2str(Ks),')       :'],'s');
if size(a,2) > 0,
  Ks = str2num(a);
end;
a=input(['gewünschte Dämpfung Dg  (',num2str(Dg),' 0...1) :'],'s');
if size(a,2) > 0,
  Dg = str2num(a);
end;
a=input(['Zeitkonst.faktor Kt  (',num2str(Kt),' 2...10)   :'],'s');
if size(a,2) > 0,
  Kt = str2num(a);
end;
a=input(['Einschwingfaktor F95 (',num2str(F95),' 0.1...1) :'],'s');
if size(a,2) > 0,
  F95 = str2num(a);
end;
a=input(['Tastperiodendauer T   (',num2str(T),')          :'],'s');
if size(a,2) > 0,
  T = str2num(a);
end;

% Berechnung der Reglerkoeffizienten

[d1,d2,K,c1,c2,cw1,cw2,Tg,Tf]=stf_pl(Ts,Ds,Ks,Dg,Kt,F95,T);

disp('   ');
disp('Die gewünschte Übertragungsfunktion hat folgende Parameter :');
disp('Verstärkung   : 1');
disp(['Zeitkonstante : ',num2str(Tg)]);
```

```
disp(['Dämpfung :      ',num2str(Dg)]);
disp('    ');
disp(['Die Filterzeitkonstante beträgt : ',num2str(Tf)]);
disp(['Filterkoeffizient  d1 = ',num2str(d1)]);
disp(['Filterkoeffizient  d2 = ',num2str(d2)]);
disp('    ');
disp(['Reglerkoeffizient  c1 = ',num2str(c1)]);
disp(['Reglerkoeffizient  c2 = ',num2str(c2)]);
disp(['Reglerkoeffizient cw1 = ',num2str(cw1)]);
disp(['Reglerkoeffizient cw2 = ',num2str(cw2)]);
disp(['Reglerverstärkung   K = ',num2str(K)]);
disp('    ');
```

5. Software

Die Programme dieses Abschnitts würden viele Seiten füllen. Durch die, mit der objektorientierte Programmierung eingeführten, vererbten Eigenschaften kann eine ausreichende Beschreibung eines speziellen Reglers eigentlich nicht erfolgen.
Ich werde mich hier nur auf den PL-Regler mit sinoidem Sollwertgeber beziehen.

Sämtliche Reglertypen und die damit verbundenen Entwurfsverfahren sind in der Unit ***REG_ALGO.PAS*** enthalten.

```
{*
*   Diese Unit stellt die Regelalgorithmen PID,PL und PLI zur Verfügung.
*   Jeder Regler kann mit einen sinoiden Sollwertgenerator gekoppelt werden.
*
*   Steffen Leßke
*   Triebes, den 15. Februar 1997
*}
unit Reg_Algo;

{$G+,N+,E-,S-,V-,R-}

interface

uses Reg_Typ, Reg_Parm, Reg_Sin1;

{*
*   Das Objekt RegEntwurf stellt alle Funktionen für einen Reglerentwurf zur
*   Verfügung. In den abgeleiteten Objekten muß nur die Funktion "Entwurf"
angepaßt
*   werden.
*
*   Zur Zeit existieren folgende abgeleitete Objekte :
*      1. PID-Regler              :  PIDEntwurf = object(RegEntwurf)
*      2. PL-Regler mit I-Anteil  :  PLIEntwurf = object(RegEntwurf)
*      3. PL-Regler ohne I-Anteil :  PLEntwurf  = object(PLIEntwurf)
*}
type RegEntwurf = object               { allg. Objekt für den Reglerentwurf }
        ok          : boolean;
        EinP,AusP   : Ppara;
        EPar,APar   : Ppara;
        constructor Init(EParam,AParam: Ppara);
        function    IsOk : boolean; virtual;
        function    SetParam(ANr: byte;AMin,AMax,AP: zahl1): boolean; virtual;
        function    GetParam(ANr: byte;var AP: zahl1): boolean; virtual;
        function    Entwurf: boolean; virtual;
        destructor  Done; virtual;
     end;
     PRegEntwurf = ^RegEntwurf;
type PLIEntwurf = object(RegEntwurf)   { Entwurf eines PL-Reglers mit I-Anteil }
        TF,Tg,zi    : zahl1;
        a1,a2,b1,b2 : zahl1;
        function    Entwurf: boolean; virtual;
     end;
     PPLIEntwurf = ^PLIEntwurf;
```

```
type PLEntwurf = object(PLIEntwurf)    { Entwurf eines PL-Reglers ohne I-Anteil }
        function    Entwurf: boolean; virtual;
     end;
     PPLEntwurf = ^PLEntwurf;

{*
*   Das Objekt Algo stellt alle Funktionen zur Verwaltung eines Regelalgorithmus
zur
*   Verfügung.
*
*   Zur Zeit existieren folgende abgeleitete Objekte :
*      1. PID-Regler              :  AlgoPID = object(Algo)
*      2. PL-Regler ohne I-Anteil :  AlgoPL  = object(Algo)
*      3. PL-Regler mit I-Anteil  :  AlgoPLI = object(AlgoPL)
*}
type Algo = object
        parameter   : ppara;           { Parametersatz }
        stop        : boolean;         { Algorithmus ist gestoppt }
        u           : zahl1;           { Stellgröße U(k) }
        constructor Init(APara: Ppara);
        procedure   Algorithmus(AX,AW: zahl1;var AU: zahl1); virtual;
        procedure   BegrenzeU; virtual;
        procedure   Starten; virtual;
        procedure   Stoppen; virtual;
        destructor  Done; virtual;
     end;
     PAlgo = ^Algo;
type AlgoPL = object(Algo)
        xk_2        : zahl1 ;          { x(k-2) }
        xk_1        : zahl1 ;          { x(k-1) }
        wk_1        : zahl1 ;          { w(k-1) }
        yk_2        : zahl1 ;          { y(k-2) }
        yk_1        : zahl1 ;          { y(k-1) }
        y           : zahl1 ;          { y(k) }
        uk_1        : zahl1 ;          { u(k-1) }
        constructor Init(APara: Ppara);
        procedure   Algorithmus(AX,AW: zahl1;var AU: zahl1); virtual;
     end;
     PAlgoPL = ^AlgoPL;

{*
*   Das Objekt Regler stellt alle Funktionen zur Verfügung, die ein Regler
benötigt.
*      - Reglerentwurf
*      - Sollwertgenerator
*      - Regelalgorithmus
*      - Laden der Parameter aus einer Datei
*   In den abgeleiteten Objekten müssen nur die Funktionen zum Erzeugen der
entsprechenden
*   Zeiger angepaßt werden.
*
*   Zur Zeit existieren folgende abgeleitete Objekte :
*      1. PID-Regler              :  PIDRegler  = object(Regler)
*      2. PID-Regler mit sin. SW  :  PIDSRegler = object(PIDRegler)
*      3. PL-Regler               :  PLRegler   = object(Regler)
*      4. PL-Regler mit sin. SW   :  PLSRegler  = object(PLRegler)
*      5. PLI-Regler              :  PLIRegler  = object(PLRegler)
*      6. PLI-Regler mit sin. SW  :  PLISRegler = object(PLSRegler)
*
*   Bei den Parameter bestehen folgende Festlegungen :
*      para[0]  = Umin - minimale Stellgröße
*      para[1]  = Umax - maximale Stellgröße
*      para[11] = Xmin - Schwelle für Unterschreitung
*      para[12] = Xmax - Schwelle für Überschreitung
*}
type Regler = object
        paraok      : boolean;
        error       : byte;
        e           : PRegEntwurf;     { Zeiger auf den zugehörigen Reglerentwurf
}
        a           : PAlgo;           { Zeiger auf den zugehörigen Algorithmus }
        l           : PRegParam;       { Zeiger auf das Ladeobjekt }
```

```
        s           : PSinSollwert;    { Zeiger auf den Sollwertgeber }
        Epar,Apar   : Ppara;           { gültige Eingangsparameter stehen in Epar
}
        PDatName    : string;          { entworfene Reglerparameter stehen in
Apar }
        constructor Init(DatName: string);
        procedure   LadeParam; virtual;
        procedure   SaveParam; virtual;
        function    GetErr(AQuit: boolean): byte; virtual;
        function    GetErrStr(AQuit: boolean): string; virtual;
        function    NewEPtr(EPara,Apara: Ppara): PRegEntwurf; virtual;
        function    NewAPtr: PAlgo; virtual;
        function    NewLPtr: PRegParam; virtual;
        function    NewSPtr: PSinSollwert; virtual;
        procedure   Entwurf(Apara: Ppara); virtual;
        procedure   Algorithmus(EByte: byte;var AByte: byte;
           var AX,AW,AU: zahl1); virtual;
        destructor  Done; virtual;
     end;
     PRegler = ^Regler;
type PLRegler = object(Regler)
        function    NewEPtr(EPara,Apara: Ppara): PRegEntwurf; virtual;
        function    NewAPtr: PAlgo; virtual;
        function    NewLPtr: PRegParam; virtual;
     end;
     PPLRegler = ^PLRegler;
type PLSRegler = object(PLRegler)
        procedure   Algorithmus(EByte: byte;var AByte: byte;
           var AX,AW,AU: zahl1); virtual;
        function    NewSPtr: PSinSollwert; virtual;
     end;
     PPLSRegler = ^PLSRegler;

implementation

{*
*  Das Objekt "RegEntwurf" stellt die Funktionen zu Organisation eines beliebigen
*  Reglerentwurfs mit MaxParam-Ein- und Ausgabeparametern zur Verfügung.
*
*  In den abgeleiteten Objekten muß nur die Methode "Entwurf" entsprechend
angepaßt
*  werden.
*}
constructor RegEntwurf.Init(EParam, AParam: Ppara);
   var i : byte;
   begin
      New(EinP);
      New(AusP);
      EinP^ := Eparam^;
      AusP^ := AParam^;
      EPar := EParam;                  { Zeiger auf Eingabewerte }
      APar := AParam;                  { Zeiger auf Ausgabewerte }
      ok := true;
   end;
{* Der Fehlerstatus wird abgefragt. Nach der Abfrage wird er rückgesetzt.
*}
function RegEntwurf.IsOk: boolean;
   begin
      IsOk := ok;
      ok := true;
   end;
function RegEntwurf.SetParam(ANr: byte;AMin,AMax,AP: zahl1): boolean;
   begin
      if (ANr>MaxParam-1)or(AP<AMin)or(AP>AMax) then begin
         SetParam := false;
         Exit;
      end;
      EinP^[Anr] := AP;
      SetParam := true;
   end;
function RegEntwurf.GetParam(ANr: byte;var AP: zahl1): boolean;
   begin
```

```
      if ANr>MaxParam-1 then begin
         GetParam := false;
         Exit;
      end;
      AP := AusP^[ANr];
      GetParam := true;
   end;
{*  Diese Funktion sollte immer überschrieben werden.
*}
function RegEntwurf.Entwurf: boolean;
   begin
      Entwurf := false;
   end;
destructor RegEntwurf.Done;
   begin
      ok := false;
      if EinP<>NIL then Dispose(EinP);
      if AusP<>NIL then Dispose(AusP);
   end;

{*
*   Entwurf eines PL-Reglers mit I-Anteil
*
*   EinP-2 : Ttast 0.0001..100s
*        3 : Ts    0.001..100s
*        4 : Ds    0.1..1
*        5 : Ks    0..1e10
*        6 : Dg    Ds..1
*        7 : KT    2..10
*        8 : F95   0.2..2
*
*   AusP-2..8 : c1,c2,cw1,cw2,d1,d2,K
*}
function PLIEntwurf.Entwurf: boolean;
   const Kg : zahl1 = 1;
   begin
      if not ok then begin
         Entwurf := false;
         Exit;
      end;
      ok := SetParam(2,0.0001,100,EPar^[2]);             { Ttast }
      if ok then ok := SetParam(3,0.001,100,EPar^[3]);   { Ts }
      if ok then ok := SetParam(4,0.1,1,EPar^[4]);       { Ds }
      if ok then ok := SetParam(5,0,1e10,EPar^[5]);      { Ks }
      if ok then ok := SetParam(6,EPar^[4],1,EPar^[6]);  { Dg }
      if ok then ok := SetParam(7,2,10,EPar^[7]);        { KT }
      if ok then ok := SetParam(8,0.2,2,EPar^[8]);       { F95 }
      if not ok then begin
         Entwurf := false;
         Exit;
      end;
      TF := 1.1 * EinP^[3] * EinP^[8] / 2; { Schritt 1 }
      if TF<EinP^[2] then TF := EinP^[2];
      Tg := EinP^[7] * TF;             { Schritt 2 }
      HGzVonPTsGlied(a1,a2,b1,b2,EinP^[3],
         EinP^[4],EinP^[5],EinP^[2]);  { Schritt 3 }
      HGzVonPTsGlied(AusP^[4],AusP^[5],AusP^[2],AusP^[3],Tg,
         EinP^[6],Kg,EinP^[2]);        { Schritt 4 - c1,c2,cw1,cw2 }
      zi := Exp(-EinP^[2]/TF);         { Schritt 5 }
      AusP^[6] := Sqr(zi)*zi;          { Schritt 6 - d1 }
      AusP^[8] := (1-AusP^[6]+3*Sqr(zi)-
         3*zi)/(b1+b2);                { Schritt 7 - K }
      AusP^[7] := 1-3*zi-AusP^[8]*b1;  { Schritt 8 - d2 }
      AusP^[0] := EPar^[0];            { Umin von EPar nach Apar übertragen }
      AusP^[1] := EPar^[1];            { Umax von EPar nach Apar übertragen }
      APar^ := AusP^;
      EPar^ := EinP^;
      Entwurf := true;
   end;

{*
*   Entwurf eines PL-Reglers ohne I-Anteil. Dabei muß die Verstärkung K
```

```
*   mit 1/T multipliziert werden.
*
*   EinP-2 : T
*   AusP-8 : K
*}
function PLEntwurf.Entwurf: boolean;
   begin
      if not inherited Entwurf then begin
         Entwurf := false;
         Exit;
      end;
      AusP^[8] := AusP^[8]/EPar^[2];   { Korrektur von K }
      APar^[8] := AusP^[8];
      Entwurf := true;
   end;

{*
*  Algo ist das Grundobjekt für den Regelalgorithmus. Es beinhaltet die
Parameter-
*  verwaltung, die Start/Stop-Funktionen und die Stellsignalbegrenzung.
*  Parameter 0 und 1 entsprechen den Stellsignalbegrenzungen Umin und Umax.
*}
constructor Algo.Init(APara: Ppara);
   begin
      parameter := APara;
      stop := true;                    { Algorithmus wird angehalten }
   end;
procedure Algo.Algorithmus(AX,AW: zahl1;var AU: zahl1);
   begin
      Au := 0;
   end;
procedure Algo.BegrenzeU;
   begin
      if u<parameter^[0]
         then u := parameter^[0]
         else if u>parameter^[1] then u := parameter^[1];
   end;
procedure Algo.Starten;
   begin
      stop := false;
   end;
procedure Algo.Stoppen;
   begin
      stop := true;
   end;
destructor Algo.Done;
   begin
      stop := true;
   end;

{*
*  AlgoPL ist der von Algo abgeleitete PL-Regler.
*     parameter[2] = cw1, parameter[3] = cw2, parameter[4] = c1
*     parameter[5] = c2, parameter[6] = d1, parameter[7] = d2
*     parameter[8] = K
*}
constructor AlgoPL.Init(APara: Ppara);
   begin
      inherited Init(APara);
      xk_2 := 0; xk_1 := 0;
      wk_1 := 0;
      yk_2 := 0; yk_1 := 0; y := 0;
      uk_1 := 0;
   end;
procedure AlgoPL.Algorithmus(AX,AW: zahl1;var AU: zahl1);
   begin
      if stop then begin
         u := 0;
         Au := u;
         wk_1 := Aw;
         yk_2 := u; yk_1 := yk_2; y := yk_1;
         xk_2 := Ax; xk_1 := Ax;
```

```
         Exit;
      end;
      y := parameter^[2]*Aw + parameter^[3]*wk_1 - Ax - parameter^[4]*xk_1
         - parameter^[5]*xk_2 - parameter^[6]*yk_1 - parameter^[7]*yk_2;
         { Reglergleichung aus Abb. 3.3.3 }
      yk_2 := yk_1; yk_1 := y;          { z-Verschiebungen }
      xk_2 := xk_1; xk_1 := Ax;
      wk_1 := Aw;
      u := parameter^[8]*y + uk_1;      { Verstärkung und I-Anteil (hier=0) }
      BegrenzeU;
      Au := u;
   end;

{*
*  Das Object Regler enthält alle Teilprozesse einer Regelung :
*     - Reglerentwurf
*     - Regelalgorithmus
*     - Sollwertbildung
*     - Verwaltung
*}
constructor Regler.Init(DatName: string);
   var i : byte;
   begin
      paraok := false;
      New(Epar);
      New(Apar);
      for i:= 0 to MaxParam-1 do begin
         Epar^[i] := 0;
         Apar^[i] := 0;
      end;
      e := NIL;                         { Reglerentwurf }
      a := NIL;                         { Regelalgorithmus }
      s := NIL;                         { Sollwertgeber }
      PDatName := DatName;              { Dateiname für Reglerparameterdatei }
      writeln('Init: ',PDatName);
      error := RegErrNo;                { Fehlerbyte }
   end;
function Regler.GetErr(AQuit: boolean): byte;
   begin
      GetErr := error;
      if AQuit then error := RegErrNo;
   end;
function Regler.GetErrStr(AQuit: boolean): string;
   begin
      case GetErr(AQuit) of
         RegErrNo   : GetErrStr := '';
         RegErrFile : GetErrStr := 'Fehler beim Laden oder Speichern der
Parameterdatei';
         RegErrAlgo : GetErrStr := 'Fehler beim Algorithmus';
         RegErrPara : GetErrStr := 'keine gültigen Parameter';
         RegErrLast : GetErrStr := 'letzte Parameteränderung abgebrochen';
         RegErrNew  : GetErrStr := 'Fehler bei New';
         else GetErrStr := 'unbekannter Fehler';
      end;
   end;
procedure Regler.LadeParam;
   begin
      if error <> RegErrNo then Exit;
      l := NewLPtr;
      if l=NIL then begin
         error := RegErrNew;
         Exit;
      end;
      if not l^.GetParameter(Epar) then error := RegErrFile;
      Dispose(l,Done);
      Entwurf(Epar);
   end;
procedure Regler.SaveParam;
   begin
      if error <> RegErrNo then Exit;
      l := NewLPtr;
      if l=NIL then begin
```

```
         error := RegErrNew;
         Exit;
      end;
      if not l^.SetParameter(Epar) then error := RegErrFile;
      Dispose(l,Done);
   end;
function Regler.NewEPtr(EPara,Apara: Ppara): PRegEntwurf;
   begin
      NewEPtr := NIL;
   end;
function Regler.NewAPtr: PAlgo;
   begin
      NewAPtr := NIL;
   end;
function Regler.NewLPtr: PRegParam;
   begin
      NewLPtr := NIL;
   end;
function Regler.NewSPtr: PSinSollwert;
   begin
      NewSPtr := NIL;
   end;
procedure Regler.Entwurf(Apara: Ppara);
   begin
      if Error <> RegErrNo then Exit;
      if e<>NIL then Dispose(e,Done);
      e := NewEPtr(APara,Apar);        { entworfene Werte stehen in Apar }
      if e=NIL then begin
         error := RegErrNew;
         Exit;
      end;
      if (e^.Entwurf and e^.IsOk)
         then begin
            Epar^ := Apara^;           { gültige Eingangsparameter werden }
            error := RegErrNo;         { nach Epar übernommen }
            paraok := true;
         end
         else error := RegErrLast;
      Dispose(e,Done);
      e := NIL;
      if not paraok then Exit;
      if a<>NIL then begin
         Dispose(a,Done);
         a := NIL;
      end;
      a := NewAPtr;
      if s<>NIL then begin
         Dispose(s,Done);
         s := NIL;
      end;
      s := NewSPtr;
   end;
procedure Regler.Algorithmus(EByte: byte;var AByte: byte;var AX,AW,AU: zahl1);
   begin
      if a=NIL then begin
         error := RegErrNew;
         AByte := error;
         Au := 0;
         Exit;
      end;
      if (EByte and RegStStart)<>0
         then a^.stop := false
         else a^.stop := true;
      if (((EByte and RegStAbs)<>0) and (Ax<Abs(Epar^[11]))) or
         (((EByte and RegStAbs)=0) and (Ax<EPar^[11]))
         then AByte := SetBits(AByte,RegErrMin)
         else AByte := ResetBits(AByte,RegErrMin);
      if (((EByte and RegStAbs)<>0) and (Ax>Abs(Epar^[12]))) or
         (((EByte and RegStAbs)=0) and (Ax>EPar^[12]))
         then AByte := SetBits(AByte,RegErrMax)
         else AByte := ResetBits(AByte,RegErrMax);
      a^.Algorithmus(Ax,Aw,Au);
```

```
   end;
destructor Regler.Done;
   begin
      if Apar<>NIL then Dispose(Apar);
      Apar := NIL;
      if Epar<>NIL then Dispose(Epar);
      Epar := NIL;
      if a<>NIL then Dispose(a,Done);
      a := NIL;
      if e<>NIL then Dispose(e,Done);
      e := NIL;
      if s<>NIL then Dispose(s,Done);
      s := NIL;
      paraok := false;
   end;

{*
*  Das Object PLRegler enthält alle Funktionen zum Bedienen eines PL-Reglers
*  ohne I-Anteil.
*}
function PLRegler.NewEPtr(EPara,Apara: Ppara): PRegEntwurf;
   begin
      NewEPtr := New(PPLEntwurf,Init(EPara,APara));
   end;
function PLRegler.NewAPtr: PAlgo;
   begin
      NewAPtr := New(PAlgoPL,Init(Apar));
   end;
function PLRegler.NewLPtr: PRegParam;
   begin
      NewLPtr := New(PPLParam,Init(PDatName));
   end;

{*
*  Das Objekt PLSRegler enthält die Funktionen des PL-Reglers plus einem
*  sinoiden Führungsgrößengenerator.
*}
function PLSRegler.NewSPtr: PSinSollwert;
   begin
      NewSPtr := New(PSinSollwert,Init(Epar^[9],Epar^[2]));
   end;
procedure PLSRegler.Algorithmus(EByte: byte;var AByte: byte;var AX,AW,AU: zahl1);
   var w : zahl1;
   begin
      w := Aw;
      if (EByte and RegStSin)<>0 then begin
         if s=NIL then begin
            AByte := RegErrNew;
            Exit;
         end;
         if (EByte and RegStStart)<>0
            then begin
               if (EByte and RegStParam)<>0 then begin
                  s^.SetVMax(Epar^[10]);
                  s^.SetXSoll(Aw);
                  s^.SWStart;
               end;
               s^.SinStep;
               w := s^.GetXIst;
            end
            else s^.SWReset;
      end;
      inherited Algorithmus(EByte,AByte,Ax,w,Au);
   end;
```

Anhang A

In diesem Abschnitt erfolgt die Berechnung der Pulsübertragungsfunktion. Auch hier habe ich eine MatLAB- und eine TurboPascal- Version geschrieben.

STF_PS2Z.M :

```
function [b1,b2,a1,a2]=stf_ps2z(Ts,Ds,Ks,T);
%STF_PS2Z Diese Funktion berechnet die Koeffizienten der Pulsüber-
%     tragungsfunktion
%
%              b1z^-1 + b2z^-2
%     HG(z)=--------------------
%           1 + a1z^-1 + a2z^-2
%
%     aus der Übertragungsfunktion eines Schwinggliedes mit den
%     Parameter Ts, Ds und Ks und einem Halteglied 0.Ordnung.
%
%     Aufruf : [b1,b2,a1,a2] = stf_ps2z(Ts,Ds,Ks,T);
%           b1,b2,a1,a2 - Koeffizienten von HG(z)
%           Ts - Streckenzeitkonstante
%           Ds - Streckendämpfung
%           Ks - Streckenverstärkung
%           T  - Tastperiodendauer
%
% SIEHE auch C2DM, STF_AT1, STF_PL

% Steffen Leßke
% Triebes, den 23.12.1996

a = - (Ds/Ts);
w = sqrt(1-Ds^2)/Ts;
b = a/w;
c = exp(a*T);
d = cos(w*T);
e = sin(w*T);

b1 = Ks * (1-(c*(d-(b*e))));
b2 = Ks * c * (c-d-(b*e));
a1 = -2 * c * d;
a2 = c^2;
```

HGzVonPTsGlied (Unit STF_UTIL.PAS) :

```
procedure HGzVonPTsGlied(var a1,a2,b1,b2,Ts,Ds,Ks,T: zahl1);
   { Diese Procedure berechnet die Pulsübertragungsfunktion entsprechend
     den Gleichungen A.13 ... A.16 }
   var a,w,adw,eat,cwt,swt : zahl1;
   begin
      a := -Ds/Ts;                      { Schritt 1 }
      w := Sqrt(1-Sqr(Ds))/Ts;          { Schritt 2 }
      if Ds = 1
         then adw := 0
         else adw := a/w;               { Schritt 3 }
      eat := Exp(a*T);                  { Schritt 4 }
      cwt := Cos(w*T);                  { Schritt 5 }
      swt := Sin(w*T);                  { Schritt 6 }
      b1 := Ks*(1-eat*(cwt-adw*swt));   { Gln. A.13 }
      b2 := Ks*eat*(eat-cwt-adw*swt);   { Gln. A.14 }
      a1 := -2*eat*cwt;                 { Gln. A.15 }
      a2 := Sqr(eat);                   { Gln. A.16 }
    end;
```