CODEN: LUTFD2/(TFRT-5367)/1-26/(1987)

Återkopplade DA-omvandlare

Einar Zettergren

Institutionen för Reglerteknik
Lunds Tekniska Högskola
Juni 1987

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

Document name
Master Thesis

Date of issue
May 1987

Document Number
CODEN: LUTFD2/(TFRT-5367)/1-26/(1987)

Author(s)
Einar Zettergren

Supervisor
Rolf Johansson

Sponsoring organisation

Title and subtitle
Återkopplade DA-omvandlare. (DA-converter with feedback.)

Abstract

The signal from a common DA-converter (DAC) is piecewise constant. The overtones may cause problems in a sensitive system. This paper describes a way to decrease the amplitude of the overtones: by interpolation of lines between the sampling points. Implementation and construction is described. Advantages and disadvantages are discussed.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
Swedish

Number of pages
26

Recipient's notes

Security classification

DA-converter with feedback

Abstract

The signal from a common DA-converter (DAC) is piecewise constant. The overtones may cause problems in a sensitive system. This paper describes a way to decrease the amplitude of the overtones: by interpolation of lines between the sampling points. Implementation and construction is described. Advantages and disadvantages are discussed.

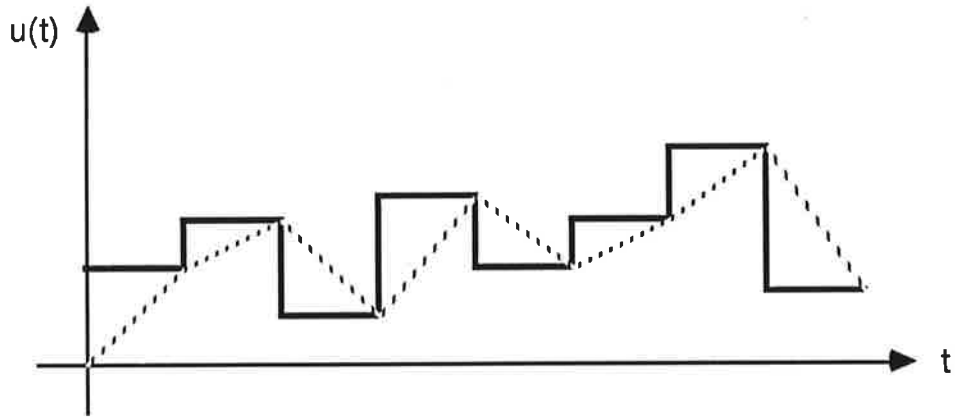
Innehållsförteckning

1. Inledning

Från en D/A-omvandlare får man, som bekant, en styckvis konstant signal. Detta är inte särskilt lyckat i känsliga system, eftersom en sådan signal ger övertoner som kan ställa till besvär. Normalt använder man lågpassfilter i sådana fall, men här skall jag presentera en helt annan metod.

1.1 Idé

Idén är att interpolera fram linjer mellan samplingspunkterna:



Figur 1.1 Fyllda linjer: vanlig D/A-omvandlare; streckade linjer: återkopplad D/A-omvandlare.

På en fyrkantsvåg avtar k:te övertonens amplitud som $\frac{1}{k}$, medan den på en triangelvåg avtar som $\frac{1}{k^2}$. Kan man genomföra detta utan allt för stora problem, är alltså mycket vunnet. Det kan man; det är det de följande kapitlen handlar om.

2. Återkopplad D/A-omvandlare

Om man i datorn deriverar styrsignalen, lägger ut den på D/A-omvandlaren och därefter integrerar den tidskontinuerligt, så får man precis den signal figur 1.1 beskriver:

$$u_{kont} = u_0 + \int_{\tau}^{\tau+h} \frac{\Delta u}{h} dt = u_0 + \Delta u \tag{2.1}$$

där u_0 är värdet integratorn har vid tiden τ, och $\Delta u = u(k) - u(k-1)$. Eftersom Δu är konstant, kommer vi att få den önskade rampen.

2.1 Inkrementell regulator

Hur bör man då beräkna Δu? Det bästa är att låta operatorn $\Delta = 1 - q^{-1}$ operera direkt på regulatorn. T ex ändras då RST-regulatorn (se t ex Åström-Wittenmark: Computer controlled systems, kap 10)

$$u(k) = -r_1 u(k-1) + t_0 u_c(k) - s_0 y(k) - s_1 y(k-1) \tag{2.2}$$

till den inkrementella formen

$$\Delta u(k) = -r_1 \Delta u(k-1) + t_0 \Delta u_c(k) - s_0 y(k) + (s_0 - s_1) y(k-1) + s_1 y(k-2) \tag{2.3}$$

vilket ju är skäligen enkelt. Här stöter man näppeligen på några problem. Men de problem man faktiskt har lyder som följer:

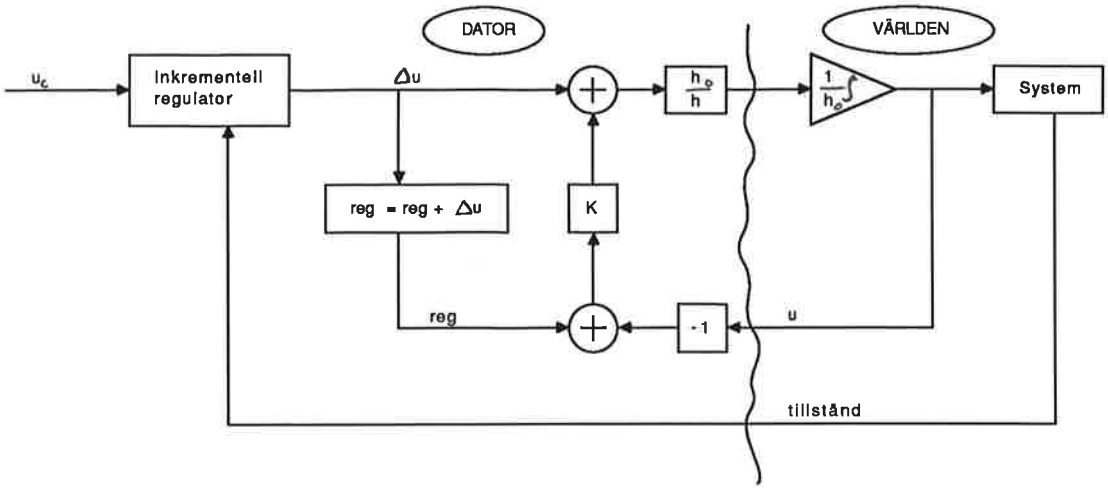
- Vid hög samplingsfrekvens blir $\frac{\Delta u}{h}$ mycket stor. Det klarar datorns D/A-omvandlare inte av.
- Integratorn driver.
- Integratorn brister i precision.

Det första klarar man av genom att multiplicera Δu med den kortaste samplingstid man tänkt använda. På så sätt blir utsignalen från datorn aldrig större än Δu. Detta måste förstås kompenseras genom att dividera utsignalen från integratorn med samma faktor.

De två senare problemen löser man genom att återkoppla utsignalen från integratorn och jämföra den med den signal man skulle haft om man använt konventionell reglering. Denna signal fås genom att addera alla Δu i ett register; det är enkelt att göra detta i datorn.

Det så erhållna felet $u_{teoretisk} - u_{verklig}$ multipliceras med en återkopplingsfaktor K och läggs till Δu. Felet regleras alltså med en P-regulator, och det räcker: återkopplar man en integrator får man en exponentiellt avtagande funktion som utsignal.

Vi är nu mogna att betrakta det hela i ett blockdiagram:



Figur 2.1 Blockdiagram över regulator/återkopplad D/A-omvandlare.

2.2 Exempel

Programmeringsarbetet under examensarbetet gjordes i Modula-2: låt oss studera en implementering av RST-regulatorn i ekvation (2.3). Antag att minsta samplingstid är 0.01 sekunder.

EXEMPEL 2.1 — RST-regulator för återkopplad D/A-omvandlare.

```
.....
(* Programhuvud, deklarationer etc *)
Register:= 0.0; OldDeltau:= 0.0; OldSetPoint:= 0.0;
y[1]:= 0.0; y[2]:= 0.0; (* y[i] = y(k-i) *)
(* startvärden *)
LOOP
    .....
    (* inläsning av parametrar *)
    Deltau:= -r1*OldDeltau + t0*(SetPoint - OldSetPoint);
    y[0]:= ADIn(0); (* tillstånd *)
    Deltau:= Deltau - s0*y[0] + (s0 - s1)*y[1] - s1*y[2];
    u:= ADIn(1); (* verklig utsignal *)
    Out:= 0.01/h*(Deltau + K*(Register - u));
    DAOut(0, Out);
    Register:= Register + Deltau; (* uppdatering börjar *)
    OldDeltau:= Deltau;
    OldSetPoint:= SetPoint;
    y[2]:= y[1]; y[1]:= y[0]; (* uppdatering klar *)
    .....
    (* Tidshantering *)
END;
```

Kommentar: Anledningen till att uppdatering av registret `Register` sker först efter beräkning av utsignalen är att integratorn alltid släpar efter ett sampel. Det tar ju tiden h för den att nå värdet $u_0 + \Delta u$ efter det att $\frac{\Delta u}{h}$ lagts ut på datorns D/A-omvandlare.

2.3 Val av parametrar

Val av återkopplingsfaktor

Om felet är e, är $u_{integrator}$ e för liten. Enligt ekvation 2.1 vill vi ha $u_0 + \Delta u$ efter tiden h, men får $u_0 + \Delta u - e$. Således borde man helt enkelt lägga e till Δu, dvs välja $K = 1$. Detta stämmer dock inte: har man fel utsignal, har man troligen också fel tillstånd. Då griper regulatorn själv in och ändrar Δu en aning. Dock inte tillräckligt; erfarenheten visar, att det är lämpligt att välja $K \approx 0.5$.

Val av samplingstid

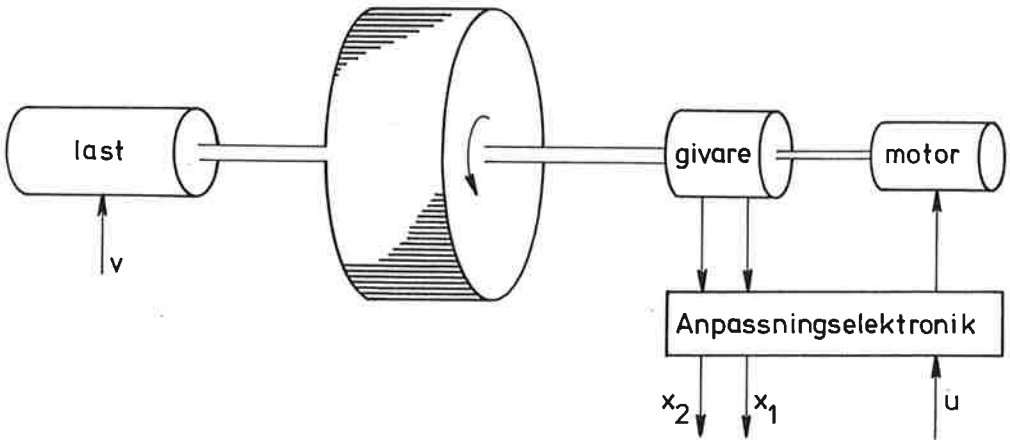
Den övre gränsen bestäms av integratorn. Om den är dåligt kalibrerad, kommer drift och dålig precision förhindra alla försök till vettig reglering. Men för de system som kan vara aktuella för återkopplad D/A-omvandling, torde fasförskjutningen få systemet i obalans innan drift och dålig precision gör det. Med vetskap om hur en normal integrator beter sig, inser man att övre gränsen ligger i storleksordningen 1 sekund.

Den nedre gränsen bestäms av realtidskärnan. På IBM-AT omöjliggjorde den körning med $h < 10$ ms. Ett annat problem är integratorns förstärkning. Det nämndes tidigare att det var lämpligt att ha denna förstärkning till den högsta samplingsfrekvensen, men det inses lätt att man får problem vid mycket höga samplingsfrekvenser. Men eftersom Δu blir mindre och mindre för ökad samplingsfrekvens, kan man välja integratorns förstärkning till ett väsentligt lägre värde, utan att signalen från datorn blir för stor. Självfallet måste motsvarande ändring ske i programmet.

3. Experiment

Under de experiment som utförts har hela tiden realtidsspråket Modula-2 på IBM-AT använts; som integrator och nödvändig förstärkare begagnades en analogimaskin. Dessutom förutsattes att den minsta samplingstiden var 10ms.

3.1 RST-regulator på DC-servo



Figur 3.1 DC-servot.

Systemet består av ett DC-servo med en motor på vars axel man satt en massiv metallcylinder, vilken är graderad i grader. Det gällde att reglera axelns vridningsvinkel. Eftersom överföringsfunktionen för systemet var given, räknades regulatorparametrar snabbt ut för olika samplingsfrekvenser.

Det första problem man stöter på är att integratorn börjar driva. Detta sker om man kopplar in den innan man startat regulatorn, eller om man har för stora regulatorparametrar. Då hjälper det inte att man ändrar tillbaka till "snälla" parametrar; då cylindern snurrar mycket snabbt, blir de vinkelvärden givaren läser fullständigt förvirrande för regulatorn.

Knepet man får ta till är att införa en extra slinga i regulatorproceduren. I den regleras utsignalen från integratorn till noll, och även där räcker det med en P-regulator. Man får också se till att det är möjligt att byta från den ena slingan till den andra interaktivt:

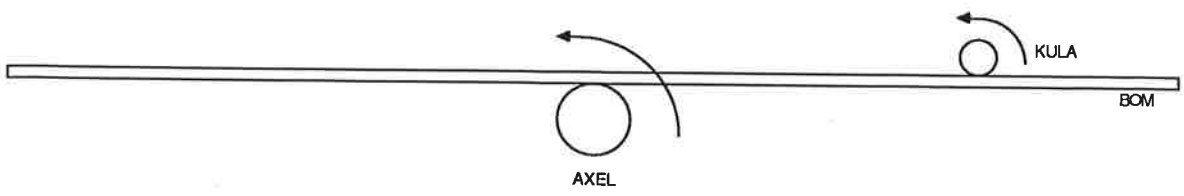
EXEMPEL 3.1 — Extraslingans insättande.

```
.....
(* Programhuvud, deklarationer etc *)
LOOP
    LOOP
        .....
        (* inläsning av parametrar *)
        u:= ADIn(1); (* utsignal från integrator *)
        DAOut(0, -0.01*u/h); (* P-reglering *)
        IF byt THEN QUIT; END; (* Villkor för loopbyte *)
        .....
        (* Tidshantering *)
    END;
    (* Upprepning av listningen i Exempel 2.1, fast med en *)
    (* loopbytessats enligt ovan efter uppdateringen *)
END;
.....
(* Avslutning *)
```

Körning

När detta väl är gjort, går regleringen utan problem. Några väsentliga skillnader mellan vanlig respektive återkopplad D/A-omvandlare kunde dock inte noteras.

3.2 Kulan och bommen



Figur 3.2 Kulan och bommen.

Detta system består av en servomotor på vilken man satt en bom: på denna bom finns ett spår, i vilket det ligger en kula. Genom att vrida bommen på lämpligt sätt kan kulans läge regleras, vilket är problemet. Bommen är nämligen ganska smäcker, så här kan man tänka sig att olämpliga övertoner leder till både transversella svängningar och torsionssvängningar.

Regleringen skedde här med PID-reglering av kulans läge: värdet på den regulatorns utsignal gavs som börvärde till vinkelregleringen. På denna räckte det med P-reglering, och en sådan är ju mycket enkel att omvandla till inkrementell form.

Även här var det nödvändigt med ovan nämnda extraslinga. Eftersom systemet är ganska känsligt, var det vettigt att använda denna när man ändrade parametrar.

Körning

Vid normal körning var den enda skillnaden att med vanlig D/A-omvandlare verkar kulan litet "nervösare" än i det återkopplade fallet, då kulan befann sig långt ut på bommen. Vid stora regulatorparametrar, däremot, betedde sig systemet med återkopplad D/A-omvandlare mycket bättre. Kulan trillade inte av så ofta. Här såg man att kulan stod och hoppade litet grand upp och ned, medan den med vanlig D/A-omvandling hade tendenser att hoppa i sidled. Detta kan bero på torsionssvängningar i bommen.

Fasförskjutning

En nollte ordningens hållkrets har överföringsfunktionen

$$G(s) = \frac{1 - e^{-sh}}{s} \approx \frac{1 - (1 - sh + \frac{(sh)^2}{2})}{s} = h(1 - \frac{sh}{2}) \tag{3.1}$$

vilket motsvarar fasförskjutningen

$$\varphi = argG(s) \approx arctan(-\frac{\omega h}{2}) \approx -\frac{\omega h}{2} \tag{3.2}$$

Eftersom A/D-omvandlare alltid har en sådan hållkrets kan vi inte undvika denna fasförskjutning. Låt oss nu betrakta den återkopplade D/A-omvandlaren. Den tidsdiskreta Δ-operatorn motsvarar $1 - e^{-sh}$ i kontinuerlig tid, och integratorn har överföringsfunktionen $\frac{1}{s}$. Totalt har alltså den återkopplade D/A-omvandlaren samma överföringsfunktion som en nollte ordningens hållkrets, och därmed samma fasförskjutning. Därför borde den totala fasförskjutningen bli dubbelt så stor som med en vanlig D/A-omvandlare.

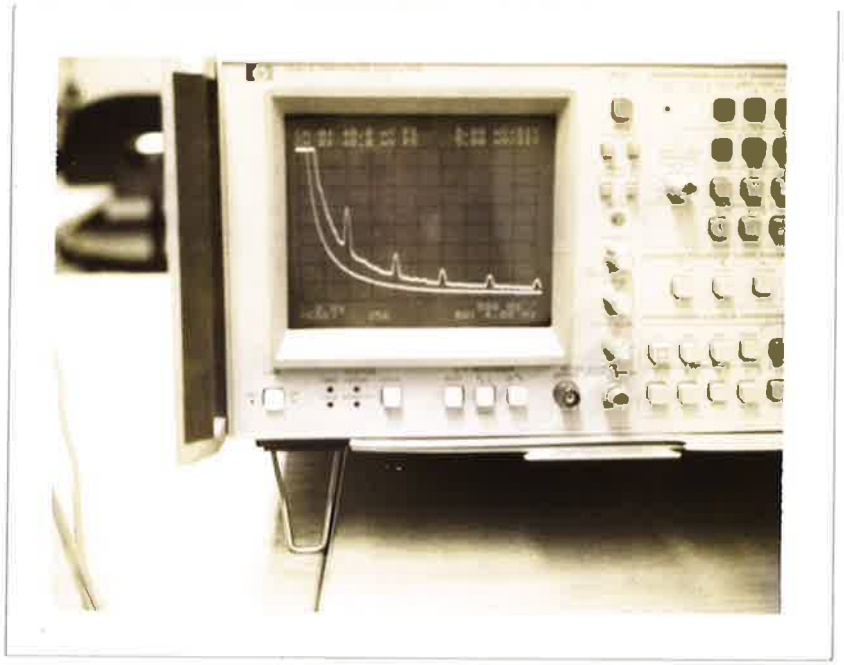
Detta verifierades experimentellt. Samplingstiden ökades sakta tills systemet kom i obalans; med återkopplad D/A-omvandlare skedde detta vid ≈140 ms, och i det vanliga fallet vid ≈220 ms.

3.3 Spektralanalys

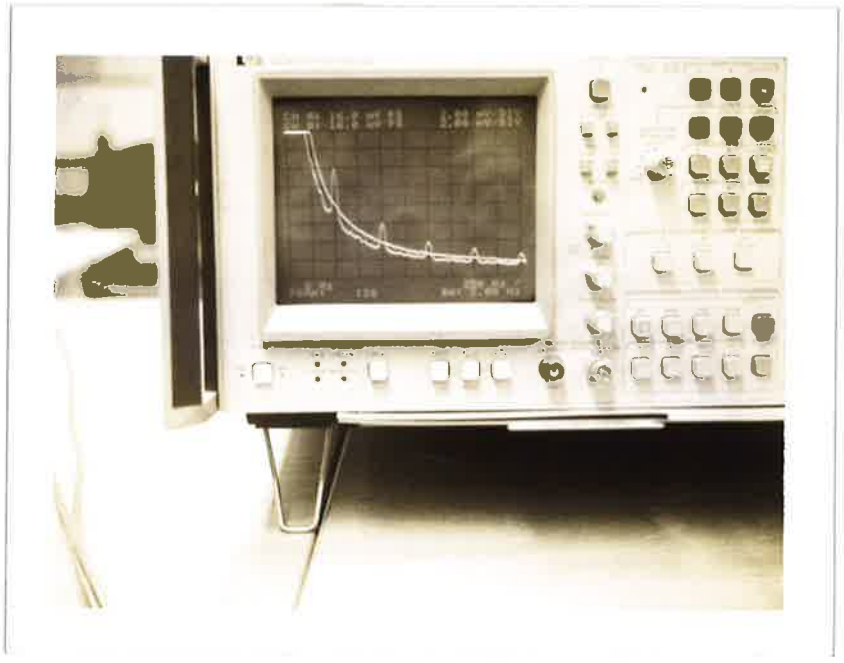
Som det nämndes i inledningen, är syftet med denna undersökning att minska övertonernas amplitud. För att verifiera att så sker, gjordes ett testprogram som ger en sinusvåg med period 40*samplingstiden och amplituden 5 volt. Denna signal dividerades med 10 innan den analyserades, så att den skulle

vara jämförbar i amplitud med utsignalen från datorn. Vid analysen användes en spektrumanalysator HP 3582A.

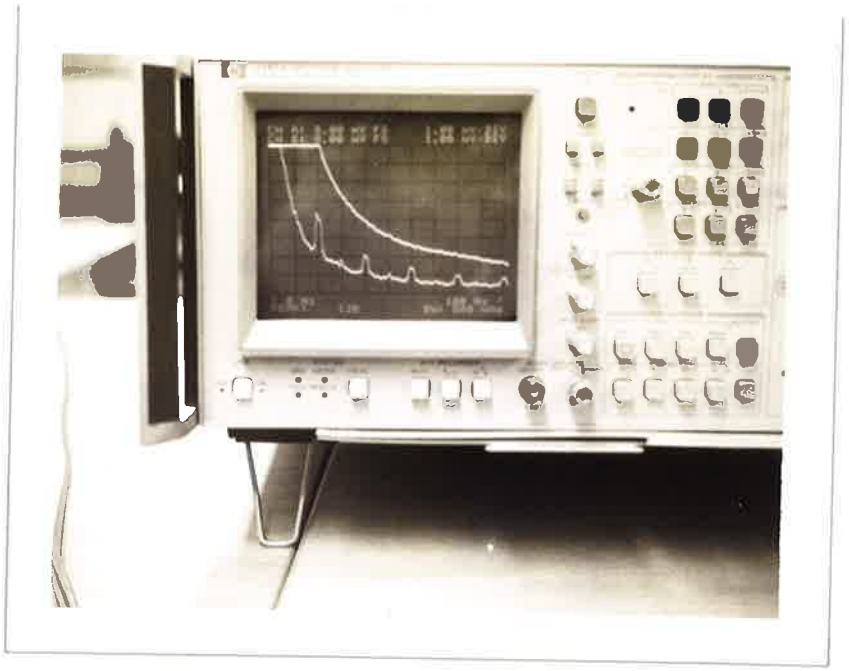
Vid tre olika val av samplingstider, nämligen 10 ms, 20 ms och 50ms, togs bilder av analysen:



Figur 3.3 Spektralanalys vid $h = 10$ ms.



Figur 3.4 Spektralanalys vid $h = 20$ ms.

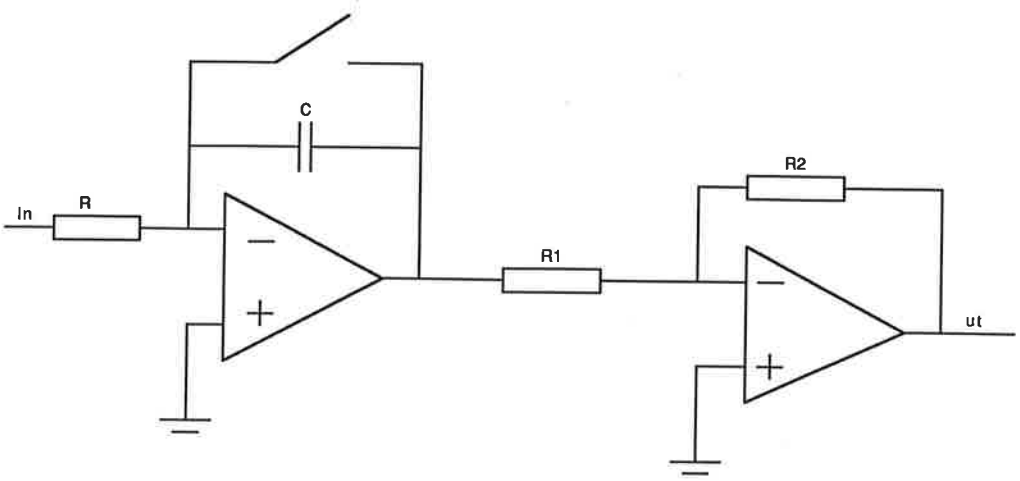


Figur 3.5 Spektralanalys vid $h = 50$ ms.

Som synes är resultatet bättre än teorin förutsade: övertonerna är *helt utplånade* i det återkopplade fallet. Förmodligen rundar integratorn av kanterna på triangelvågen. Försök har också gjorts med ännu högre samplingstid, och resultatet står sig.

4. Konstruktion av integrator

Att konstruera integratordelen av systemet är ganska okomplicerat: man tar en vanlig OP-baserad integrator följt av en lika OP-baserad inverterad förstärkare:



Figur 4.1 Integrator med förstärkare.

Det gäller nu att välja förstärkningen

$$f_0 = \frac{1}{h_0} = (-\frac{1}{RC})(-\frac{R_2}{R_1}) = \frac{R_2}{RCR_1} \tag{4.1}$$

Där lär konstruktören inte ha några problem.

Som det antytts i figuren, bör man också ha en från datorn styrd kortslutning av kondensatorn. Detta för att integreringen skall börja först när regulatorn gjort det. Man skulle kunna ta ett relä styrt av en bistabil vippa, som i sin tur styrdes av datorn. På så sätt skulle utsignalen kunna sättas på och stängas av från lämpliga ställen i programmet. Med detta skulle också den i kapitel 3 nämnda extraslingan bli överflödig.

En finess som skulle göra apparaten mer generell är att man byter R_2 mot en stegpotentiometer. Kompletterar man detta med att interaktivt kunna ändra förstärkningen av Δu på motsvarande sätt, så har man ett system som är mycket anpassningsbart. De problem som förväntas vid mycket höga samplingsfrekvenser (se kapitel 2.3) kan enkelt undvikas.

5. Sammanfattning

Det primära målet med det här examensarbetet var att kontrollera om idén gick att realisera. Som framgår av föregående kapitel gick det utmärkt. Följande för- och nackdelar har konstaterats:

Fördelar:

- Den återkopplade D/A-omvandlaren skär effektivt bort övertoner.
- Enkelt att implementera: lättare att beräkna integratorn än ett lågpassfilter.
- Oberoende av samplingsfrekvens (till en viss utsträckning). Vid långsam sampling skulle ett filter kunna ställa till problem.

Nackdelar:

- Dubbelt så stor fasförskjutning som en vanlig D/A-omvandlare.
- Ofta är $\Delta u << u$. Ev. kan kvantiseringsproblem uppstå. Detta har dock aldrig detekterats.
- Samplar man mycket långsamt kan integratorn hinna driva för mycket.

Vad skulle man då kunna göra mer? Man skulle kunna testa styrning av ett system, som är väsentligt känsligare än de som tidigare nämnts. Man skulle kunna göra en noggrannare teoretisk analys. Om det skulle kunna tillföra ytterligare meningsfull information vet jag inte; inte heller om detta bara är en lustig kuriositet utan praktisk användning. Faktum kvarstår dock, att denna metods fördelar oftast överväger dess nackdelar.

6. Referenser

ÅSTRÖM, K. J. och B. WITTENMARK (1984): *Computer Controlled Systems*, Prentice-Hall, Inc., Englewood Cliffs, N.J..

Appendix: Programlistningar

Här följer nu listningar på de program i Modula-2 som gjorts till detta examensarbete. Läsaren må ha överseende med att kommentarernas å, ä och ö ser ut som [, ' och ;. Det bör också noteras att programmen gjorts för att fylla sin funktion, och att då det estetiska fått stryka på foten. Det står läsaren fritt att själv göra de ändringar han/hon finner nödvändiga.

`MODULE Newtest` är det testprogram som användes i spektralanalysen i kap. 3.

`MODULE Reglera` är RST-regulatorn för DC-servot.

`MODULE Beam` är regulatorn för kula/bom-systemet.

```
MODULE Newtest;
FROM Kernel IMPORT
   InitKernel, CreateProcess, SetPriority,
   Time, IncTime, GetTime, WaitUntil, Semaphore, Wait, Signal, InitSem;
FROM AnalogIO IMPORT  DAOut, ADIn;
FROM NumberConversion IMPORT StringToCard;
FROM Terminal IMPORT ReadString, WriteString, WriteLn;
FROM MathLib IMPORT sin, float, cos;
FROM ConvReal IMPORT StringToReal;

TYPE radtyp= ARRAY[0..79] OF CHAR;

VAR mutex, TheEnd: Semaphore;
    h: CARDINAL;                (* Samplingstid i ms *)
    K: REAL;                    (* F;rst'rkning *)

PROCEDURE GetH(VAR value: CARDINAL; VAR changed: BOOLEAN);
BEGIN
    changed:=FALSE;                          (* Anv'nds f;r uthopp ur inre *)
    IF (value<>h) THEN changed:=TRUE; END;     (* loopen  *)
    Wait(mutex);
    value:=h;
    Signal(mutex);
END GetH;

PROCEDURE GetK(VAR varde: REAL);
BEGIN
    Wait(mutex);
    varde:=K;
    Signal(mutex);
END GetK;

PROCEDURE ChangeH(value: CARDINAL);
BEGIN
    Wait(mutex);
    h:=value;
    Signal(mutex);
END ChangeH;

PROCEDURE ChangeK(varde: REAL);
BEGIN
    Wait(mutex);
    K:=varde;
    Signal(mutex);
END ChangeK;

PROCEDURE FelUtskrift;
BEGIN
    WriteLn; WriteString(' Ej giltigt!!');
END FelUtskrift;

PROCEDURE KollaOmSlut(row: radtyp);
BEGIN
    IF (row[0]='e') OR (row[0]='E') THEN
       DAOut(0,0.0);
       Signal(TheEnd);
    END;
END KollaOmSlut;

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

```
(* Process *) PROCEDURE InUt;

VAR next: Time;
    h: CARDINAL;
    H, X, u, ut, y, yref, K: REAL;
    changed: BOOLEAN;

BEGIN
    SetPriority(10);
    WHILE TRUE DO
       yref:=0.0; X:=0.0;                    (* Startv'rden *)
       GetTime(next);
       LOOP
         GetK(K);
         GetH(h, changed);
         IF changed THEN EXIT; END;
         H:=float(h);
         X:=X+1.0; IF (X)=41.0) THEN X:=1.0; END;
         u:= 0.5*(sin(6.2832*X/40.0) - sin(6.2832*(X-1.0)/40.0));
         y:=ADIn(0);
         ut:= 10.0/H*(u + K*(yref-y));     (* Cterkoppling *)
         DAOut(0,ut);
         yref:=-0.5*sin(6.2832*X/40.0);    (* :nskad funktion *)
         IncTime(next,h);
         WaitUntil(next);
       END;
    END;
END InUt;

(* Process *) PROCEDURE Reader;
CONST hmin=10;
VAR row: radtyp;
    h, pos: CARDINAL;
    done: BOOLEAN;
    K: REAL;

BEGIN
    SetPriority(11);
    LOOP
       WriteLn; WriteString('Skriv h (minst 10) eller E f;r stopp  ');
       ReadString(row);
       KollaOmSlut(row);
       StringToCard(row, h, done);
       IF done & (h)=hmin) THEN
          ChangeH(h);
       ELSE
          FelUtskrift;
       END;
       WriteLn; WriteString('Skriv K (minst 0) eller E f;r stopp  ');
       ReadString(row);
       KollaOmSlut(row); pos:= 0;
       StringToReal(row, pos, K);
       IF (pos()0) & (K)=0.0) THEN
          ChangeK(K);
       ELSE
          FelUtskrift;
       END;
    END;
END Reader;

BEGIN
    InitKernel;
```

```
    InitSem(mutex, 1);
    InitSem(TheEnd, 0);
    h:=10; K:=1.0;                    (* Förslag på startvärden *)
    CreateProcess(InUt, 3000);
    CreateProcess(Reader, 5000);
    Wait(TheEnd);
END Newtest.
```

```
MODULE Reglera;
FROM Kernel IMPORT
   InitKernel, CreateProcess, SetPriority,
   Time, IncTime, GetTime, WaitUntil, Semaphore, Wait, Signal, InitSem;
FROM AnalogIO IMPORT DAOut, ADIn;
FROM Terminal IMPORT ReadString, WriteString, WriteLn;
FROM ConvReal IMPORT StringToReal, RealToString;
FROM MathLib IMPORT round;

TYPE radtyp = ARRAY [0..2] OF CHAR;
     statetyp = ARRAY [0..2] OF REAL;

VAR mutex, TheEnd: Semaphore;
    r1, S0, S1, H, T0, REF, G: REAL;
    quit: BOOLEAN;

(* r1, S0, S1, T0: regulatorparametrar, H: samplingstid i msN          *)
(* REF: b;rv'rde, G: ;terkopplingsf;rst'rkning och quit: villkor f;r *)
(* uthopp ur extraslingan                                              *)

PROCEDURE GetPar(row: radtyp; VAR value: REAL);
BEGIN
    Wait(mutex);
    CASE row[0] OF
       'H': value:= H   |
       'r': value:= r1  |
       '0': value:= S0  |
       '1': value:= S1  |
       'T': value:= T0  |
       'G': value:= G   |
       'R': value:= REF
    END;
    Signal(mutex);
END GetPar;

PROCEDURE ChangePar(row: radtyp; value: REAL);
BEGIN
    Wait(mutex);
    CASE row[0] OF
       'H': H:= value   |
       'r': r1:= value   |
       '0': S0:= value  |
       '1': S1:= value  |
       'T': T0:= value   |
       'G': G:= value   |
       'R': REF:= value
    END;
    Signal(mutex);
END ChangePar;

PROCEDURE GetQ(VAR value: BOOLEAN);
BEGIN
    Wait(mutex);
    value:= quit;
    Signal(mutex);
END GetQ;

PROCEDURE ChangeQ;  (* "ndrar quit till dess konjugat *)
BEGIN
    Wait(mutex);
    IF quit THEN
       quit:= FALSE;
```

```
    ELSE
       quit:= TRUE;
    END;
    Signal(mutex);
END ChangeQ;

PROCEDURE Felutskrift;
BEGIN
    WriteLn; WriteString(' Not Valid!!');
END Felutskrift;

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

(* Process *) PROCEDURE InUt;

VAR OldU, Register, u, y, ref, OldRef, H, ut, deltau, omega, G: REAL;
    State: statetyp;
    h: INTEGER;
    next: Time;
    quit: BOOLEAN;

BEGIN
    SetPriority(10);
    GetTime(next);
    LOOP
      ChangeQ;
      LOOP                                  (* Extraslingan  *)
        GetPar('H  ', H);
        u:=-10.0/H*ADIn(1);
        DAOut(0, u);                        (* enkel [terkoppling *)
        GetQ(quit);
        IF quit THEN EXIT; END;             (* loopbytestest *)
        h:= round(H);
        IncTime(next, h);
        WaitUntil(next);
      END;
        ChangeQ;
        State[0]:= 0.0; State[1]:= 0.0; State[2]:= 0.0;
        OldU:= 0.0; Register:= 0.0; OldRef:= 0.0;          (* Startv'rden *)
      LOOP                                  (* Styrslingan *)
        GetPar('r  ', r1);
        GetPar('0  ', S0);
        GetPar('1  ', S1);
        GetPar('T0 ', T0);
        GetPar('G  ', G);
        GetPar('REF', ref);
        GetPar('H  ', H);      (* inl'sning av variabler *)
        deltau:= -r1*OldU + T0*(ref-OldRef);          (* ber'kning av *)
        deltau:= deltau + (S0-S1)*State[1] + S1*State[2];  (* gamla v'rden *)
        State[0]:= ADIn(0);
        u:= ADIn(1);            (* l'sning av nya v'rden p[ u & y *)
        deltau:= deltau - S0*State[0];
        ut:= 10.0/H*(deltau + G*(Register - u));
        IF (ut ) 10.0/H) THEN ut:= 10.0/H; END;       (* skydd mot f;r *)
        IF (ut ( -10.0/H) THEN ut:= -10.0/H; END;     (* stora steg    *)
        DAOut(0, ut);
        Register:= Register + deltau;                 (* uppdatering *)
        OldU:= deltau;
        OldRef:= ref;
        State[2]:= State[1];
        State[1]:= State[0];                          (* uppdatering slut *)
        h:= round(H);
```

```
        GetQ(quit);
        IF quit THEN EXIT; END;                              (* loopbytestest *)
        IncTime(next, h);
        WaitUntil(next);
      END;
    END;
END InUt;

(* Process *)PROCEDURE Reader;

CONST w = 7;
VAR r1, S0, S1, H, T0, REF, value, G: REAL;
    pos: CARDINAL;
    rad: radtyp;
    row: ARRAY[0..8] OF CHAR;
    error: BOOLEAN;

BEGIN
    SetPriority(11);
    LOOP
      GetPar('r1 ', r1); GetPar('0  ', S0); GetPar('1  ', S1);
      GetPar('T0 ', T0); GetPar('H  ', H); GetPar('REF', REF); GetPar('G  ', G);
      WriteLn;
      RealToString(G, row, w); WriteLn; WriteString('G = ');
      WriteString(row);
      RealToString(r1, row, w); WriteLn; WriteString('r1 = ');
      WriteString(row);
      RealToString(S0, row, w); WriteLn; WriteString('S0 = ');
      WriteString(row);
      RealToString(S1, row, w); WriteLn; WriteString('S1 = ');
      WriteString(row);
      RealToString(T0, row, w); WriteLn; WriteString('T0 = ');
      WriteString(row);
      RealToString(H, row, w); WriteLn; WriteString('H = ');
      WriteString(row);
      RealToString(REF, row, w); WriteLn; WriteString('REF = ');
      WriteString(row);
      WriteLn; WriteLn; WriteString('Parameter to be changed (END for stop, ');
      WriteString('Q for stop->control or control->stop):');
      ReadString(rad); WriteLn; error:= FALSE;
      CASE rad[0] OF
         'r':|
         '0':|   (*                                          *)
         '1':|   (*      Skydd mot misstag gjorda vid        *)
         'T':|   (*                                          *)
         'H':|   (*      inl'sning av variabler              *)
         'G':|   (*                                          *)
         'Q': ChangeQ |
         'R':|
         'E': DAOut(0, 0.0); Signal(TheEnd)
      ELSE
         Felutskrift; error:= TRUE;
      END;
      IF NOT error AND (rad[0] <> 'Q') THEN
         WriteString(rad); WriteString(' = '); ReadString(row); pos:= 0;
         StringToReal(row, pos, value);
         IF (pos <> 0) THEN
            ChangePar(rad, value);
         ELSE
            Felutskrift;
         END;
      END;
```

```
    END;
END Reader;

BEGIN
    InitKernel;
    InitSem(mutex, 1);
    InitSem(TheEnd, 0);
    r1:= 0.2441;  S0:= 15.3949; S1:= -12.3355; T0:= 3.0669;
    (* motsvarar omega = 10 och z'ta = 1/sqrt(2) vid h = 40 ms   *)
    H:= 40.0;  REF:= 0.0;  G:= 0.3; quit:= TRUE;
    CreateProcess(InUt, 5000);
    CreateProcess(Reader, 5000);
    Wait(TheEnd);
END Reglera.
```

```
MODULE Beam;
FROM Kernel IMPORT
   InitKernel, CreateProcess, SetPriority,
   Time, IncTime, GetTime, WaitUntil, Semaphore, Wait, Signal, InitSem;
FROM AnalogIO IMPORT DAOut, ADIn;
FROM Terminal IMPORT ReadString, WriteString, WriteLn;
FROM ConvReal IMPORT StringToReal, RealToString;
FROM MathLib IMPORT round;

TYPE radtyp = ARRAY [0..2] OF CHAR;
     ControlType = (hold, position, angle);

VAR mutex, TheEnd: Semaphore;
    K, Td, REF, H, G, N, AK, IT: REAL;
    ControlState: ControlType;

(* K, IT, Td: PID-parametrar kulan, H = samplingstid i ms *)
(* AK: P-parameter vinkeln, N: filterkonstant derivatadel *)
(* REF: b|rv'rde, G: {terkopplingsf|rs'rkning              *)


PROCEDURE GetPar(row: radtyp; VAR value: REAL);
BEGIN
    Wait(mutex);
    CASE row[0] OF
       'K': value:= K  |
       'T': value:= Td |
       'R': value:= REF|
       'H': value:= H  |
       'N': value:= N  |
       'A': value:= AK |
       'I': value:= IT |
       'G': value:= G
    END;
    Signal(mutex);
END GetPar;

PROCEDURE ChangePar(row: radtyp; value: REAL);
BEGIN
    Wait(mutex);
    CASE row[0] OF
       'K': K:= value  |
       'T': Td:= value |
       'R': REF:= value|
       'H': H:= value  |
       'N': N:= value  |
       'A': AK:= value |
       'I': IT:= value |
       'G': G:= value
    END;
    Signal(mutex);
END ChangePar;

PROCEDURE GetState(VAR value: ControlType);
BEGIN
    Wait(mutex);
    value:= ControlState;
    Signal(mutex);
END GetState;

PROCEDURE ChangeState;
```

```
VAR  rad: radtyp;

BEGIN
    WriteLn; WriteString('Controlstate = '); ReadString(rad);
    Wait(mutex);
    CASE rad[0] OF
       'H': ControlState:= hold      |
       'P': ControlState:= position  |
       'A': ControlState:= angle
    ELSE
       Felutskrift;
    END;
    Signal(mutex);
END ChangeState;


PROCEDURE Felutskrift;
BEGIN
    WriteLn; WriteString('  Ej giltigt!');
END Felutskrift;

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

(* Process *) PROCEDURE InUt;

VAR K, Td, H, G, REF, ut, Register, u, deltau, H1, y, N, X: REAL;
    xpos, AK, IT, p, i, d, FiRef, RegUt: REAL;
    next: Time;
    ControlState: ControlType;
    h: INTEGER;
    e, ex: ARRAY[0..2] OF REAL;

BEGIN
    SetPriority(10);
    GetTime(next);
    LOOP
      LOOP                              (* Extraslingan *)
        GetPar('H  ', H);
        u:= 10.0/H*ADIn(2);
        DAOut(0, -u);                   (* Enkel P-reglering *)
        DAOut(1, 0.0);                  (* Om man anv'nder vanlig D/A-omvandling *)
        GetState(ControlState);
        IF (ControlState () hold) THEN EXIT; END;
        h:= round(H);
        IncTime(next, h);
        WaitUntil(next);
      END;
      Register:= 0.0; e[1]:= 0.0; e[2]:= 0.0;
      ex[1]:= 0.0; i:= 0.0; d:= 0.0; ex[0]:= 0.0;    (* Startv'rden *)
      LOOP                              (* Styrslingan *)
        GetPar('K  ', K);
        GetPar('Td ', Td);
        GetPar('H  ', H);
        GetPar('REF', REF);
        GetPar('G  ', G);
        GetPar('N  ', N);
        GetPar('IT ', IT);
        GetPar('AK ', AK);
        H1:= H*0.001;                   (* ms => sek *)
        X:= Td/(H1*N);                  (* mycket vanlig faktor *)
        IF (ControlState = position) THEN
            xpos:= ADIn(0);
```

```
            ex[0]:= REF - xpos;
            p:= K*ex[0];
            i:= i + K*H1/IT*ex[0];
            d:= (X*d + K*N*X*(ex[0] - ex[1]))/(1.0 + X);
            FiRef:= p + i + d;              (* PID-reglering av kulan *)
            ex[1]:= ex[0];
            IF (FiRef ) 0.2) THEN FiRef:= 0.2; END;      (* Vinkelbegr'nsning
            IF (FiRef ( -0.2) THEN FiRef:= -0.2; END;
            FiRef:= -FiRef;                 (* L'get ;kas => vinkeln minskas *)
        ELSE
            FiRef:= REF;                    (* Vid vinkelreglering *)
        END;
        y:= ADIn(1);
        u:= ADIn(2);
        IncReg(u, ex[0]);
        e[0]:= FiRef - y;
        deltau:= AK*(e[0] - e[1]);                          (* Inkrementell form *)
        ut:= 10.0/H*(deltau + G*(Register - u));
        IF (ut ) 10.0/H) THEN ut:= 10.0/H; END;      (* Begr'nsning av    *)
        IF (ut ( -10.0/H) THEN ut:= -10.0/H; END;    (* stegstorlek       *)
        DAOut(0, ut);
        Register:= Register + deltau;
        RegUt:= Register;
        IF(RegUt ) 1.0) THEN RegUt:= 1.0; END;
        IF(RegUt ( -1.0) THEN RegUt:= -1.0; END;
        DAOut(1, RegUt);                    (* Utg[ng f;r vanlig D/A-omvandlare *)
        e[2]:= e[1]; e[1]:= e[0];           (* Uppdatering *)
        GetState(ControlState);
        IF (ControlState = hold) THEN EXIT; END;
        h:= round(H);
        IncTime(next, h);
        WaitUntil(next);
      END;
    END;
END InUt;

(* Process *) PROCEDURE Reader;

CONST w = 7;
VAR   K, Td, H, G, REF, value, N, AK, IT: REAL;
      pos: CARDINAL;
      rad: radtyp;
      row: ARRAY [0..8] OF CHAR;
      error: BOOLEAN;
      ControlState: ControlType;

BEGIN
    SetPriority(11);
    LOOP
        GetPar('K  ', K); GetPar('Td ', Td); GetPar('H  ', H);
        GetPar('G  ', G); GetPar('REF', REF); GetState(ControlState);
        GetPar('N  ', N); GetPar('AK ', AK); GetPar('IT ', IT);
        WriteLn;
        RealToString(K, row, w); WriteLn; WriteString('K = ');
        WriteString(row);
        RealToString(Td, row, w); WriteLn; WriteString('Td = ');
        WriteString(row);
        RealToString(H, row, w); WriteLn; WriteString('H = ');
        WriteString(row);
        RealToString(G, row, w); WriteLn; WriteString('G = ');
        WriteString(row);
        RealToString(REF, row, w); WriteLn; WriteString('REF = ');
```

```
        WriteString(row);
        RealToString(N, row, w); WriteLn; WriteString('N = ');
        WriteString(row);
        RealToString(AK, row, w); WriteLn; WriteString('AK = ');
        WriteString(row);
        RealToString(IT, row, w); WriteLn; WriteString('IT = ');
        WriteString(row);
        WriteLn; WriteString('Controlstate = ');
        CASE ControlState OF
           hold: WriteString('hold');                          |
           position: WriteString('position: input 0')          |
           angle: WriteString('angle: input 1')
        END;
        WriteLn; WriteLn; WriteString('Parameter to be changed'); WriteLn;
        WriteString('(END for stop):');
        ReadString(rad); WriteLn; error:= FALSE;
        CASE rad[0] OF
           'K', 'T', 'H', 'G', 'R', 'N', 'A', 'I':|
           'E': DAOut(0, 0.0); Signal(TheEnd)  |
           'C': ChangeState  |
        ELSE
           Felutskrift; error:= TRUE;
        END;
        IF NOT error AND (rad[0] <>'C') THEN
           WriteString(rad); WriteString(' = '); ReadString(row); pos:= 0;
           StringToReal(row, pos, value);
           IF (pos <> 0) THEN
              ChangePar(rad, value);
           ELSE
              Felutskrift;
           END;
        END;
     END;
  END Reader;

  BEGIN
     InitKernel;
     InitSem(mutex, 1);
     InitSem(TheEnd, 0);
     K:= 0.3; Td:= 1.0; H:= 20.0; G:= 0.3; REF:= 0.0; N:= 5.0;
     AK:= 2.0; IT:= 20.0;              (* "Sn'lla" parametrar *)
     ControlState:= hold;              (* Börja med u = 0     *)
     CreateProcess(InUt, 5000);
     CreateProcess(Reader, 5000);
     Wait(TheEnd);
  END Beam.
```