LUND UNIVERSITY

Projekt i Adaptiv Reglering VT91

Åström, Karl Johan

1992

Document Version:
Förlagets slutgiltiga version

CODEN: LUTFD2/(TFRT-7482)/1–218/(1992)

Projekt i adaptiv reglering
VT 91

Karl Johan Åström
Redaktör

Institutionen för Reglerteknik
Lunds Tekniska Högskola
Januari 1992

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

Document name
REPORT

Date of issue
Januari 1992

Document Number
CODEN: LUTFD2/(TFRT-7482)/1–218/(1992)

Author(s)
Karl Johan Åström (Editor)

Supervisor

Sponsoring organisation

Title and subtitle
Projekt i Adaptiv Reglering (Projects in adaptive control)

Abstract

This report contains the project papers in the course on adaptive control given during spring 1991.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
Swedish

Number of pages
218

Recipient's notes

Security classification

Projekt i Adaptiv Reglering

VT 91

Karl Johan Åström
Redaktör

Förord

Denna rapport innehåller redogörelser från projekten i "Adaptiv reglering", som utfördes under vårterminen 1991. Projekten var beräknade att ta en vecka per person.

David Nilsson valde att studera det testproblem som hade lämnats ut inför European Control Conference 1991. Det har varit roligt att se att en teknolog kunde prestera en tillfredsställande lösning av detta problem. Henrik Olsson och Ulf Jönsson hade valt att undersöka några olika hybrida algoritmer med hjälp av den toolbox i MATLAB som Anders Ringdal skrivit. Detta arbete visar att det finns många frågor att undersöka innan man kan förstå dessa algoritmer. Anders Sjö hade valt att undersöka de numeriska egenskaperna hos kvadratrotsalgotitmer. Detta projekt har resulterat i ett examensarbete i numerisk analys. I vårens kurs hade vi för första gången tillgång till LabView, ett trevligt program för attt snabbt testa algoritmer. Anders Ask och Lars Sjöberg började med att göra några enkla experiment med reglering av dc-servot. Detta arbete fortsattes av Krister Tham som implementerade modellreferenssystem. Det visade sig att LabView är utmärkt för att snabbt få en prototyp med ett snyggt människa-maskin snitt.

Några mer ambitiösa projekt utfördes som gemensamma projekt i kursen datorimplementering. En fyrmannagrupp bestående av Fredrik Menander, Lars Falk, Johan Silvander och Marianne Sernevi implementerade en RST regulator i C med LabView som användargränssnitt. Tyvärr hann ej C-kompilatorn för signalprocessorn fram i tid, så systemet kunde ej köras. Projektet visade dock att det går utmärkt att använda LabView som använder gränssnitt. En annan kvartett, Rikard Berglund, Erik Apelgren, Stefan Johansson och Ola Bernersson, implementerade en adaptiv regulator med människa-maskin komkmunikation för ett DC-servo på IBM PC. I detta användes Simnon till Modula II översättaren. Programmet kördes med framgång.

Martin Kruci´nski studerade ett praktiskt problem från Tetra Pak, nämligen reglering av en fyllningsmaskin. Detta projekt fortsattes sedan som ett examensarbete. Sven Andersson och Svein Helgesen experimenterade med att använda en PC med realtids Simnon för att prova PID regulatorer med automatinställning. Detta arbete har sedan fullföljts och resulterat i en ny version av lab 3 i kursen.

Några teknologer vågade sig på att göra mer exotiska ting. Markus Jacobsson gjorde en neuronnät-composer för att komponera melodier. Melodierna var av svensktoppskaraktär, men det behövs nog bättre utorgan för att konkurrera. Tomas Carlsson och Tomas Almgren experimenterade med att använda neuronnät för att klassificera stegsvar och för att ställa in regulatorer. Ett färdigt program, Neural Works Professional, användes. Det gick utmärkt att klassificera stegsvar. Detta borde undersökas vidare.

Det har verkligen varit roligt att se hur mycket duktiga teknologer kan göra på en vecka. De 15 minuters presentationer som gavs var genomgående mycket bra.

Karl Johan Åström

Innehållsförteckning

Lunds Tekniska Högskola

Institutionen för Reglerteknik

Adaptive Control of Benchmark Example

-Projekt adaptiv reglering

av

David Nilsson E87

Handledare:
Karl Johan Åström

Lund, 21 Maj 1991

1 INLEDNING

1.1 Bakgrund

I kursen adaptiv reglering ingår ett projektarbete omfattande en veckas heltidsarbete. Som projekt valde jag att studera ett "benchmark example", se bilaga 1.

1.2 Avgränsningar

Av tidsskäl har jag tvingats göra vissa avgränsningar. Jag har slagit ihop experimenten 2-6, som behandlar adaptiva regulatorer, på så sätt att istället för att göra dem i tur och ordning, har jag valt att undersöka en kategoris regulatorernas uppträdande lite djupare; nämligen de som designas efter en andra ordningens modell. Anledningen till att jag valde en andra ordnings modell, till problemets tredje ordningens process, var att denna modell fungerade bäst när jag simulerade de digitala regulatorerna.

1.3 Metod

Jag har jobbat största delen av tiden i Matlab, där jag designat och simulerat de digitala regulatorerna. Men eftersom man bara kan simulera diskreta processer i Matlab har jag även simulerat regulatorerna med en kontinuerlig process i Simnon. Det är resultaten från de senare simuleringarna som jag redovisar. Som integrationsalgoritm har jag använt mig av rkf23 och rkf45.

När jag har placerat Ao, Am-polerna har jag använt principen "guestimates", d v s jag har provat mig fram till var jag ska placera polerna.

När det gäller val av process-modell har jag också använt mig av metoden "guestimates" när det gäller den första- och tredje ordningens digitala regulatorer, d v s man anger en modell och kontrollerar denna mot den riktiga processen i ett Bode-diagram. När jag skulle göra den andra ordningens digitala regulator använde jag mig istället av programmet visidyn[1] för att bestämma processmodellen. Att jag inte använde programmet även för de andra regulatorerna berodde på att jag inte fick reda på programmets existens innan jag hade gjort de två första regulatorerna.

1

Jag har dokumenterat alla Matlab-körningar, utom de två första dagarna då jag inte var medveten eller hade kunskap om dokumenteringens vikt. Dessa bifogas inte då de skulle ta alldeles för stor plats, men kan uppvisas om så önskas.

Som störning till processen har jag använt mig av normalfördelat brus, där man ställer in amplituden på densamma med hjälp av variabeln n. Jag har i de flesta fallen använt mig av värdet 1 på brusfaktorn, vilket motsvarar normalfördelat brus med amplituden inom området -2 till 2. Exempel på hur bruset kan se ut ges i bilaga 1.1.

Samplingstiden som jag använt är lika med 0.2 sekunder. Valet av samplingstid grundar sig på två faktorer; för det första så ska $0.1 \leq w*h \leq 0.5$, $w=1.57 \Rightarrow 0.06 \leq h \leq 0.32$ och för det andra så är tidsfördröjningen lika med 0.2.

2 BENCHMARK EXAMPLE

2.1 Experiment 1

Första experimentet går ut på att studera hur en digital regulator klarar av att reglera processen som är beskriven i bilaga 1. Det fanns ett bivillkor som sa att stigtiden fick högst vara 4 sek. Simnonprogrammen ser i stort sett likadana ut med enda skillnaden i antalet parametrar och tillstånd. Jag har därför valt att endast visa upp ett Simnonprogram som får anses vara representativt för de digitala regulatorerna i detta experimentet. Programmet hittar ni i bilaga 2.1.

2.1.1 Första ordningens regulator

Den första regulatorn som jag provade var av allpasskaraktär och av första ordningen. Processen innehåller en fördröjning, vilket motsvaras av en extra pol i nollan (samplade poler). Detta innebär att man har två poler i Am-polynomet, d v s en i nollan plus en till. Jag fann dock att egenskaperna blev bättre om jag istället placerade de kontinuerliga Am-polerna i -1 respektive -4. Anledningen till att jag kan göra på detta sättet är att tidsfördröjningens påverkan på utsignalen är liten i förhållande till den inverkan som icke-minfasen gör.

Observerarpolynomets poler placerade jag i -1 respektive -1(Ao, kont.).

Simnonsimuleringen med brusnivå n=1 hittar ni i bilaga 2.1.1

2.1.2 Andra ordningens regulator

Nästa regulator är av andra ordningen med polerna placerade på följande ställe:

Ao-poler i: -1, -1, -1 (kontinuerliga)

Am-poler i: -2, -2 (kontinuerliga)

Simuleringen av detta systemet hittar ni i bilaga 2.1.2

2.1.3 Tredje ordningens regulator

Här har jag studerat en regulator som var av samma ordning som processen själv. Vid designen gav följande poler bäst egenskaper i Matlab, se bilaga 2.1.3.1:

Ao-poler i: -1, -1, -1, -1 (kontinuerliga)

Am-poler i: -5, -5, -2, -2 (kontinuerliga)

Tyvärr så var resultatet inte lika bra när jag försökte simulera regulatorn i Simnon, se bilaga 2.1.3.2. Varken jag eller någon av de personen som jag har tillfrågat har kunnat svara på varför skillnaden blev så stor. En iakttagelse kan man dock göra och det är att regulatorn är väldigt känslig för parameter-variationer – en ändring i fjärde decimalen märks omedelbart.

En regulator som det någorlunda gick att simulera i Simnon var en med följande poler:

Ao-poler i: -1-i, -1+i, -10-10i, -10+10i (kontinuerliga)

Am-poler i: -1-i, -1+i, -5-5i, -5+5i (kontinuerliga)

Resultatet ser ni i bilaga 2.1.3.3. Observera att brusfaktorn endast är 0.1 här, vilket är en tiondel av störfaktorn som använts i de andra simuleringarna.

2.1.4 Störkänslighet

Utifrån de simuleringar jag gjort konstaterar jag att den regulator som är minst störkänslig är den andra ordningens regulator. Allpassregulatorn klarar sig något sämre men är i sin tur mycket bättre än regulatorerna av tredje ordningen.

2.2 Andra ordningens adaptiv regulator

Som modell har jag använt följande struktur:
H(q)=(b0*q+b1)/(q^3+a1*q^2+a2*q+a3)

Ni hittar Simnonprogrammen i bilaga 2.2.

2.2.1 Mätbrus

Med utgångspunkt från den digitala regulator byggde jag en adaptiv regulator. Regulatorn har samma poler som den digitala. Simuleringen i bilaga 2.2.1.1 och 2.2.1.2 visar att även den adaptiva regulatorn klarar mätbruset väl efter det att den har svängt in sig. Som mätbrus har jag använt samma brus som innan, d v s normalfördelat med brusfaktorn 1.

2.2.2 Mätbrus och laststörningar

Förutom mätbruset har jag nu även lagt till laststörningar enligt följande schema:

0 $0 \leq t < 20$
-5 $20 \leq t < 40$
0 $40 \leq t < 60$
-5 $60 \leq t < 80$

Om man studerar bilaga 2.2.2.1 så finner man att det tar ungefär åtta sekunder för regulatorn att kompensera laststörningen. Bilaga 2.2.2.2 visar estimeringen av processparametrarna.

2.2.3 Mätbrus och processparametervariationer

För att testa regulatorns förmåga att klara variationer i processens parametrar använde jag följande scenario:

$0 \leq t < 40$ Inga processvariationer
$40 \leq t < 60$ T ökade sitt värde från 1 till 1.2
$60 \leq t < 80$ Förutom att T ökade, så ökande även K från 1 till 2.5

I bilaga 2.2.3.1 ser vi att regulatorn klarar av variationerna i T hyggligt, medan regulatorn blir instabil när sen även K ökar sitt värde drastiskt. De ändrade skattningarna av parametrar ser vi i bilaga 2.2.3.2.

3 Sammanfattning

Sammanfattningsvis så skulle jag vilja säga att det går alldeles utmärkt att använda en andra ordningens modell till en tredje ordningens process. Den adaptiva regulatorn uppför sig bra så länge som processvariationerna inte blir för stora.

4 Avslutning

Matlab i kombination med visidyn är ett kraftfullt hjälpmedel för att designa regulatorer. Enda nackdelen är det tar ett tag innan man kan utnyttja systemen produktivt. Vad beträffar Simnon så är mina åsikter något kluvna, det är ett bra hjälpmedel när det fungerar som det ska. Jag har dock råkat ut för en hel del konstigheter både då det gäller felmeddelande och reproducerbarheten.

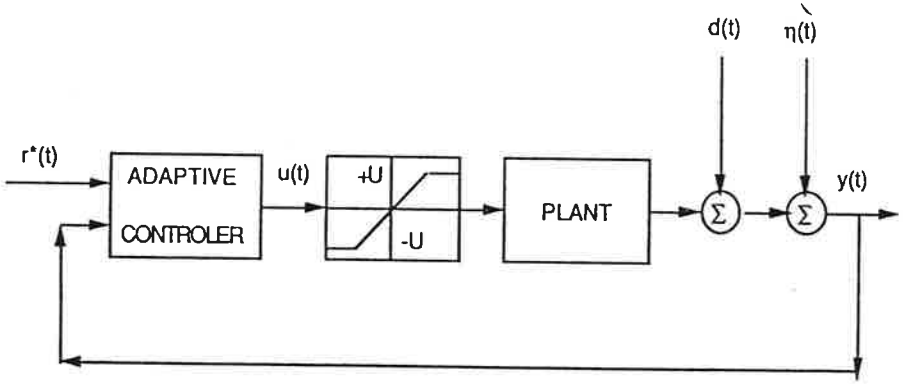
ADAPTIVE CONTROL OF BENCHMARK EXAMPLE

Our invited session, Adaptive Control of Benchmark Example, has been accepted for presentation in ECC'91. Recall that the motivation of this session is to investigate the behavior of the available adaptive control algorithms in the presence of those unavoidable features of the real world life, namely external disturbances, reduced order modeling, plant model parameter variations and input constraints.

The involved participants (or research groups) are :

Astrom (Lund), Clarke (Oxford), De Larminat (Rennes), Goodwin (Newcastle), Iserman (Darmstadt), Mosca (Firenze) and M'Saad (Grenoble).

The adaptive control scheme under consideration is depicted in the following figure



where

{r*(t)} := set point sequence.

u(t) := plant input.

y(t) := measured plant output.

{d(t)} := unmeasured load disturbances.

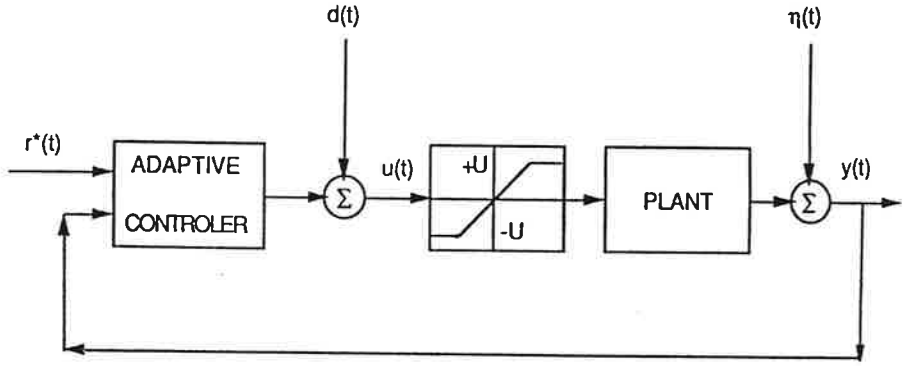
{n(t)} := sensor noise.

U := input saturation level.

ADAPTIVE CONTROL OF BENCHMARK EXAMPLE

Please, there are two changes and correction that have to be made. The changes, suggested by some participants, are the following

1. The load disturbance sequence is moved to the input.The adaptive control scheme under consideration becomes



2. The measurement disturbances {η(t)} is assumed to be a continous coloured noise with zero mean and auto-correlation function given by

$$\varphi(\tau) = \sigma\, e^{-10\,|\tau|}$$

The correction concerns the transfer function describing the plant to be controlled, and is the following

$$H(s) = K\, e^{-\tau s}\, \frac{\omega_0^{\,2}\,(sT - 1)}{(sT + 1)\,(s^2 + 2\zeta\omega_0 s + \omega_0^{\,2})}$$

PLANT DESCRIPTION.

The plant to be considered is described by the following transfer function

$$H(s) = K\, e^{-\tau s}\, \frac{\omega_o^2 (s - T)}{(s + T)(s^2 + 2\zeta\omega_o s + \omega_o^2)}$$

with

$K = K_n + \Delta K$ with $K_n = 1$ and $-0.5 \le \Delta K \le 2$

$\tau = \tau_n + \Delta\tau$ with $\tau_n = 0.2$ and $0.0 \le \Delta\tau \le 0.2$

$T = T_n + \Delta T$ with $T_n = 1$ and $-0.2 \le \Delta T \le 0.5$

$\omega_o = \omega_{on} + \Delta\omega_o$ with $\omega_{on} = 15$ and $-5 \le \Delta\omega_o \le 1$

$\zeta = 1$

CONTROL OBJECTIVE.

The control objective consists in **free-overshoot** tracking and regulation dynamics with total (5%) **rise time of about 4T**.

EXPERIMENTAL PLANNING.

- The set point sequence {r*(t)} is generated by square waves with magnitude switching between levels 10 and -5 for the tracking case. In the regulation case, the set point sequence {r*(t)} is identically equal to 10.
- The load disturbances {d(t)} are assumed to be steps of random magnitude over random time intervals.

- {η(t)} is assumed to be white noise with zero mean and variance σ.
- The parameter estimates have to be initialized to zero. A start-up procedure over a certain time interval can be used when required. This time interval will be noted To and should be indicated.
- Except when it is precised, the parameters of the plant to be controlled are kept constant during each experiment and the model order is assumed to be known.
- The input saturation level is assumed to be 25. This value can be modified if needed; the considered value should be indicated.

EXPERIMENTS.

Seven experiments have to be made according to the motivation of the benchmark example.

Experiment 1 : linear control.

This experiment is intended to show the tracking and regulation performances of the considered controller when the plant is known.

The set point sequence, load disturbances and sensor noise to be considered are generated as follows :

$r^*(t) = +10$, $d(t) = 0$ and $\sigma = 0$ for $0 \le t < 10T$
$r^*(t) = -05$, $d(t) = 0$ and $\sigma = 0$ for $10T \le t < 20T$
$r^*(t) = +10$, $d(t) = -5$ and $\sigma = 0.2$ for $20T \le t \le 30T$

Experiment 2.

This experiment is aimed to emphasize the tracking performance of the adaptive controller in the ideal case, i.e. the plant structure is assumed to be known, the unknown plant parameters are kept constant and the load disturbances are identically zero.

The set point sequence and sensor noise to be considered are generated as follows :

$r^*(t) = +10$ for $To \le t < To + 10T$
$r^*(t) = -05$ for $To + 10T \le t < To + 20T$

$r^*(t) = +10$ for $To + 20T \leq t \leq To + 30T$
$\sigma = 0.2$ for all t.

To is the length of the start-up period, if needed.

Experiment 3.

This experiment is motivated by the performance of the adaptive controller in the presence of load disturbances.

The set point sequence, load disturbances and sensor noise to be considered are generated as follows :

$r^*(t) = +10$ for all t.
$d(t) = -05$ for $0 \leq t < To + 10T$
$d(t) = +00$ for $To + 10T \leq t \leq To + 20T$
$d(t) = -05$ for $To + 20T \leq t \leq To + 30T$
$\sigma = 0.2$ for all t.

To is the length of the start-up period, if needed.

Experiment 4.

This experiment consists in illustrating the adaptive controller performance when the plant model order is overparametrized. In particular, we will use a fourth order plant model.

The set point sequence, load disturbances and sensor noise to be considered are generated as follows :

$r^*(t) = +10$ and $d(t) = 0$ for $To \leq t < To + 10T$
$r^*(t) = -05$ and $d(t) = 0$ for $To+10T \leq t < To+20T$
$r^*(t) = +10$ and $d(t) = 0$ for $To+20T \leq t \leq To+30T$
$r^*(t) = +10$ and $d(t) = -5$ for $To+30T \leq t \leq To+40T$
$\sigma = 0.2$ for all t.

To is the length of the start-up period, if needed.

Experiment 5.

This experiment aims at emphasizing the robustness of the adaptive controller with respect to reduced order modelling. To do so, the load disturbances and sensor noise are assumed to be identically zero. Both second order and first order plant model have to be considered; the (5%) rise time of the desired tracking and regulation dynamics can be slowed down if needed.

The set point sequence to be considered is generated as follows for both the second order as well as the first order plant model cases:

$r^*(t) = +10$ for $To \leq t < To + 10T$
$r^*(t) = -05$ for $To + 10T \leq t < To + 20T$
$r^*(t) = +10$ for $To + 20T \leq t \leq To + 30T$

To is the length of the start-up period, if needed.

Experiment 6.

In this experiment, the robustness of the adaptive controller with respect to reduced order modelling and state disturbances is investigated. One particularly consider the first order plant model, the (5%) rise time of the desired tracking and regulation dynamics can be slowed down if needed.

The set point sequence, load disturbances and sensor noise to be considered are generated as follows :

$r^*(t) = +10$ and $d(t) = -5$ for $To \leq t < To + 10T$
$r^*(t) = -05$ and $d(t) = -5$ for $To+10T \leq t < To+20T$
$r^*(t) = +10$ and $d(t) = 0$ for $To+20T \leq t \leq To+30T$
$r^*(t) = +10$ and $d(t) = 0$ for $To+30T \leq t \leq To+40T$
$\sigma = 0.2$ for all t.

To is the length of the start-up period, if needed.

Experiment 7.

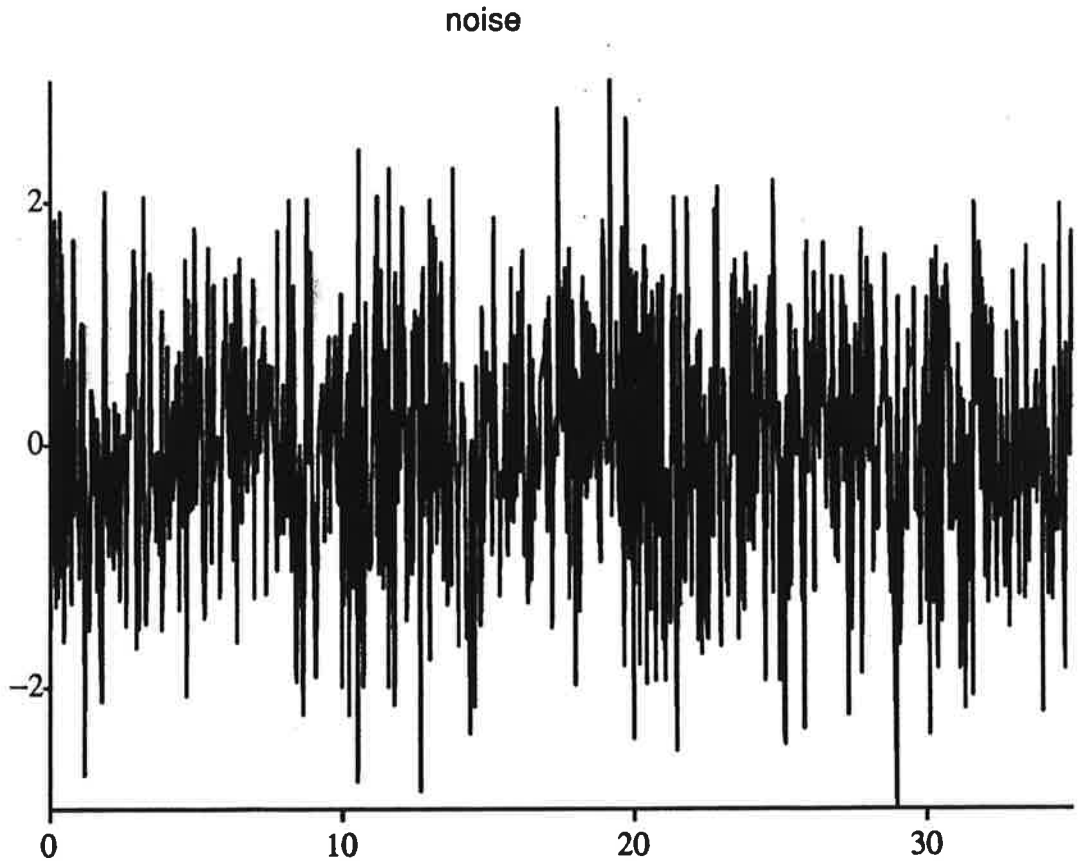
This experiment is intended to illustrate the adaptation alertness of the adaptive controller when the plant parameters K and T are allowed to vary, with respect to their nominal values, as follows :

$\Delta K = \Delta T = 0$ for $0 \leq t < To + 15T$
$\Delta K = 0$ and $\Delta T = 0.2$ for $To + 15T \leq t < To + 35T$
$\Delta K = 1.5$ and $\Delta T = 0.2$ for $To + 35T \leq t < To + 50T$

The set point sequence to be considered is generated as follows :

$r^*(t) = +10$ for $0 \leq t < To + 10T$
$r^*(t) = -05$ for $To + 10T \leq t < To + 30T$
$r^*(t) = +10$ for $To + 30T \leq t \leq To + 50T$

Bilaga 1.1



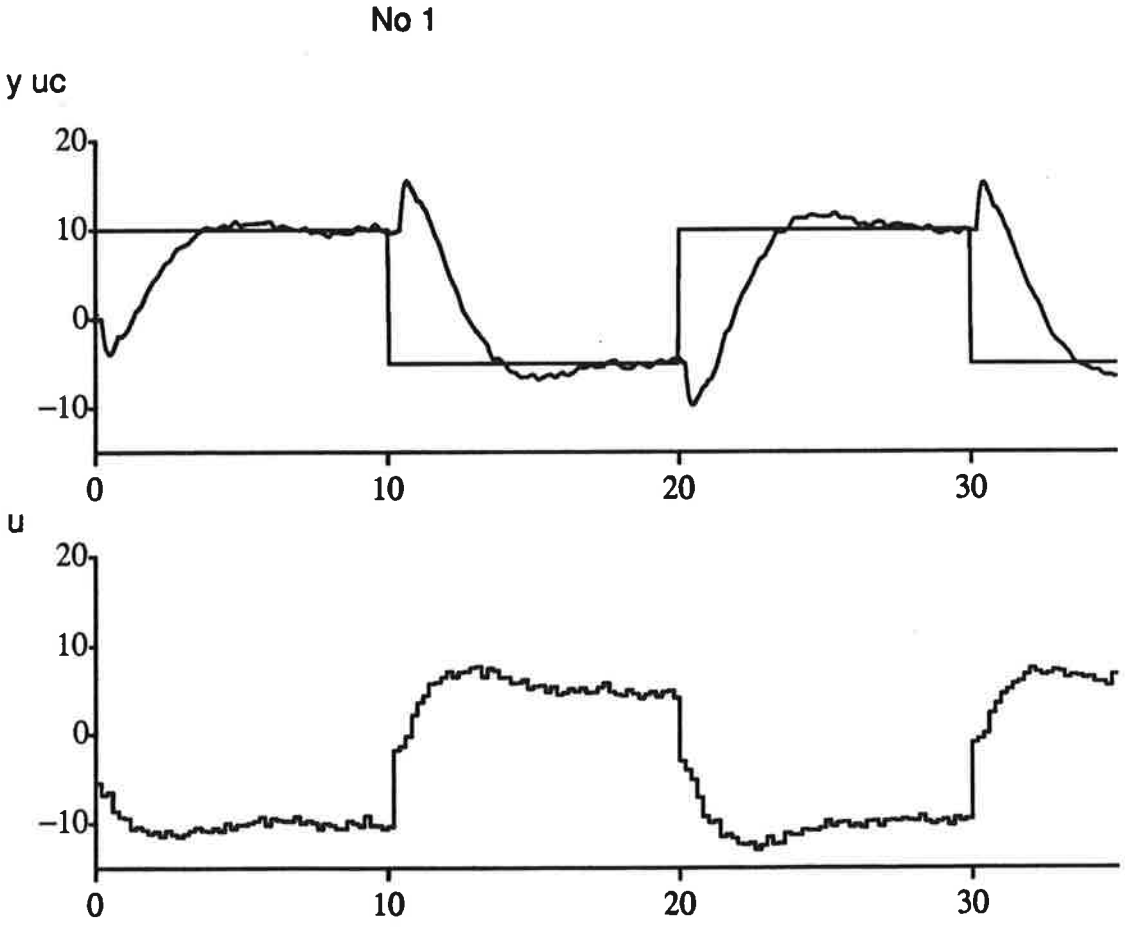
```
DISCRETE SYSTEM reg
"regulator based on H(s)=K*w0^2(sT-1)/((sT+1)*(s^2+2*w0*s+w0^2)
"                   T=1


INPUT uc y
OUTPUT u
STATE u1 u2 u3 u4 y1 y2 y3 y4
STATE uc1 uc2 uc3 uc4
NEW nu1 nu2 nu3 nu4 ny1 ny2 ny3 ny4
NEW nuc1 nuc2 nuc3 nuc4
TIME t
TSAMP ts

"control law-----------------------------------
v1=(t0*uc+t1*uc1+t2*uc2+t3*uc3+t4*uc4)
v2=(s0*y+s1*y1+s2*y2+s3*y3+s4*y4)
v3=(r1*u1+r2*u2+r3*u3+r4*u4)
u=v1 - v2 - v3

"u=IF v<-ulim THEN -ulim ELSE IF v>ulim THEN ulim ELSE v

"update controller state-------------------------

ny1=y
ny2=y1
ny3=y2
ny4=y3

nu1=u
nu2=u1
nu3=u2
nu4=u3

nuc1=uc
nuc2=uc1
nuc3=uc2
nuc4=uc3


"update sampling time---------------------------

ts=t+h


"parameters-------------------------------------

h:0.2
ulim:1


r1:-3.9641
r2:11.6207
r3:-16.5804
r4:7.9238
s0:-8.0861
s1:15.8925
s2:-8.8545
s3:1.1014
s4:-0.0551
t0:-0.4261
t1:1.3678
t2:-1.6688
t3:0.9168
t4:-0.1915
```

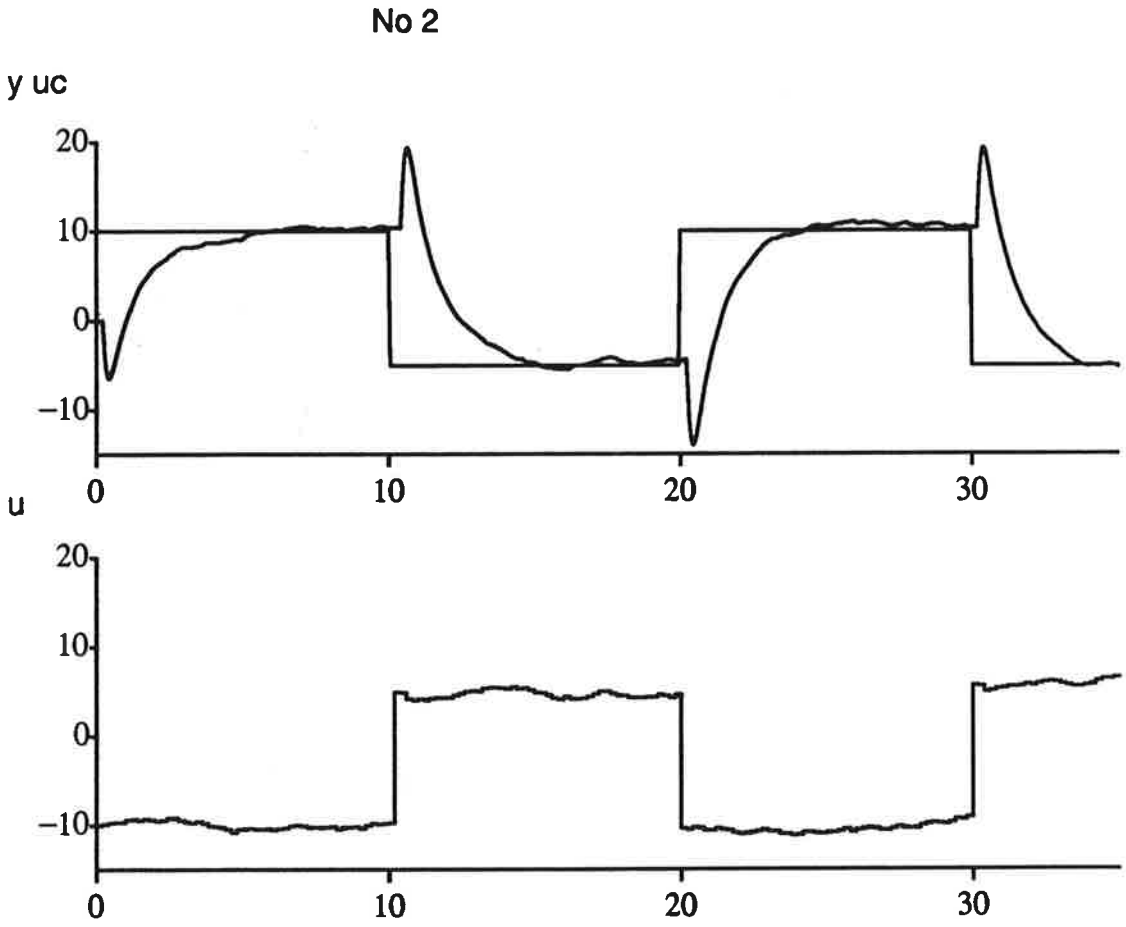
```
END
```

```
CONTINUOUS SYSTEM proc

INPUT u
OUTPUT y
STATE x1 x2 x3
DER dx1 dx2 dx3
TIME t

"Process----------------------
dx1=-(2*w0*te + 1)*x1 + x2
dx2=-(w0 + 2/te)*w0*x1 + x3 + K*w0*w0*u
dx3=-(w0*w0/te)*x1 - K*w0*w0*u/te
y=delay(x1,tdelay)

K=IF t<60 THEN 1 ELSE  2.5
te=IF t<40 THEN 1 ELSE  1.2
w0:15
tdelay:0.2

END
```

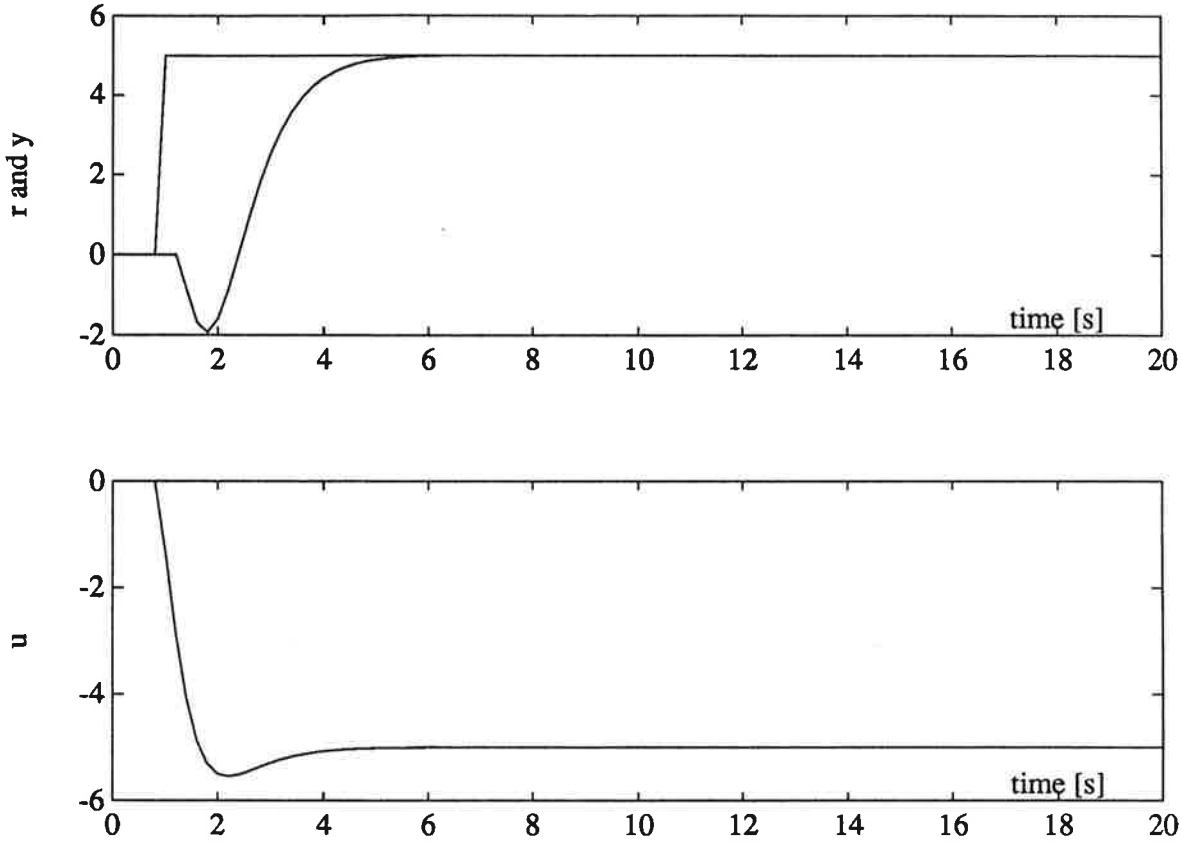
```
CONNECTING SYSTEM con

time t

noise=n*norm(t)
y[reg]=y[proc]+noise
u[proc]=u[reg]


"set point-----------------
uc[reg]=IF mod(t,per)<per/2 THEN stepup ELSE stepdown

per:20
stepup:10
stepdown:-5
n:1

END
```

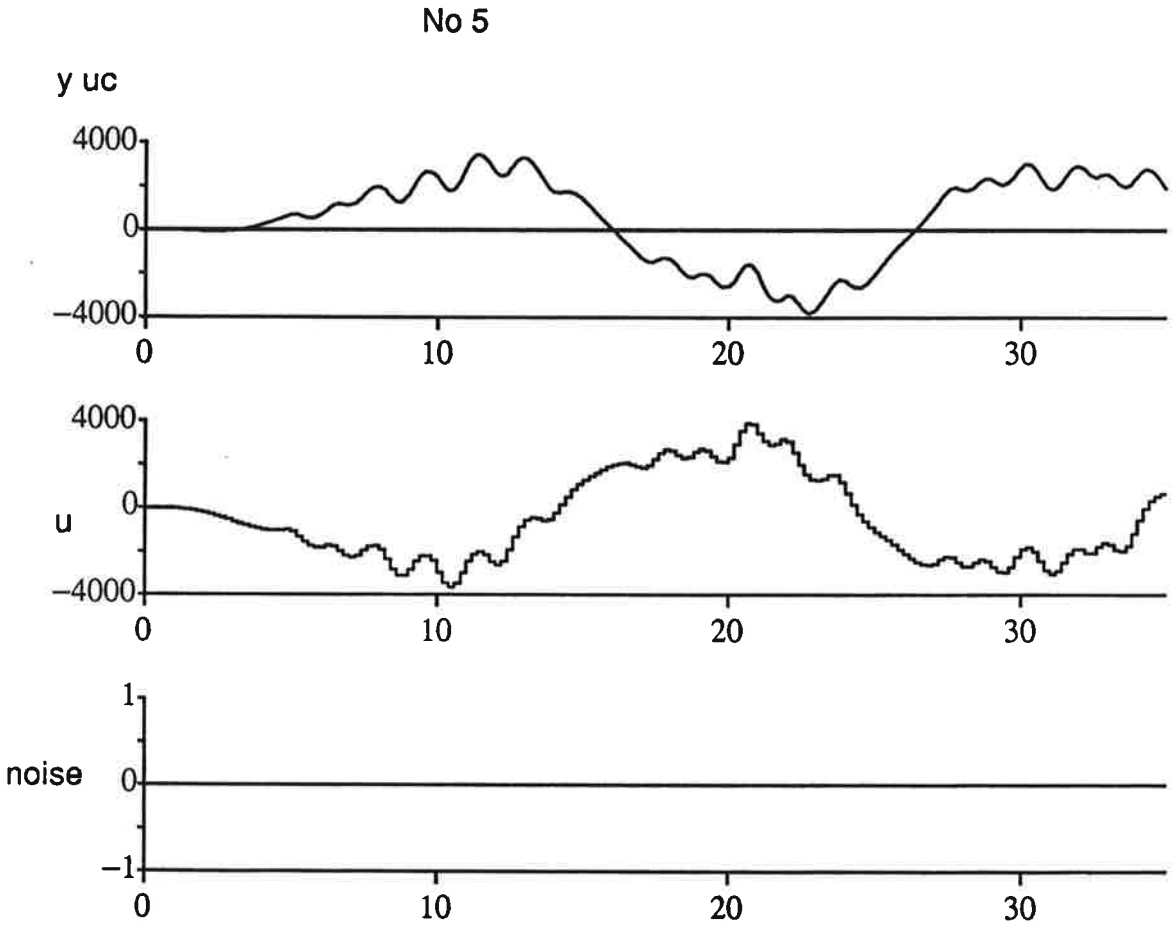
Bilaga 2.1.1



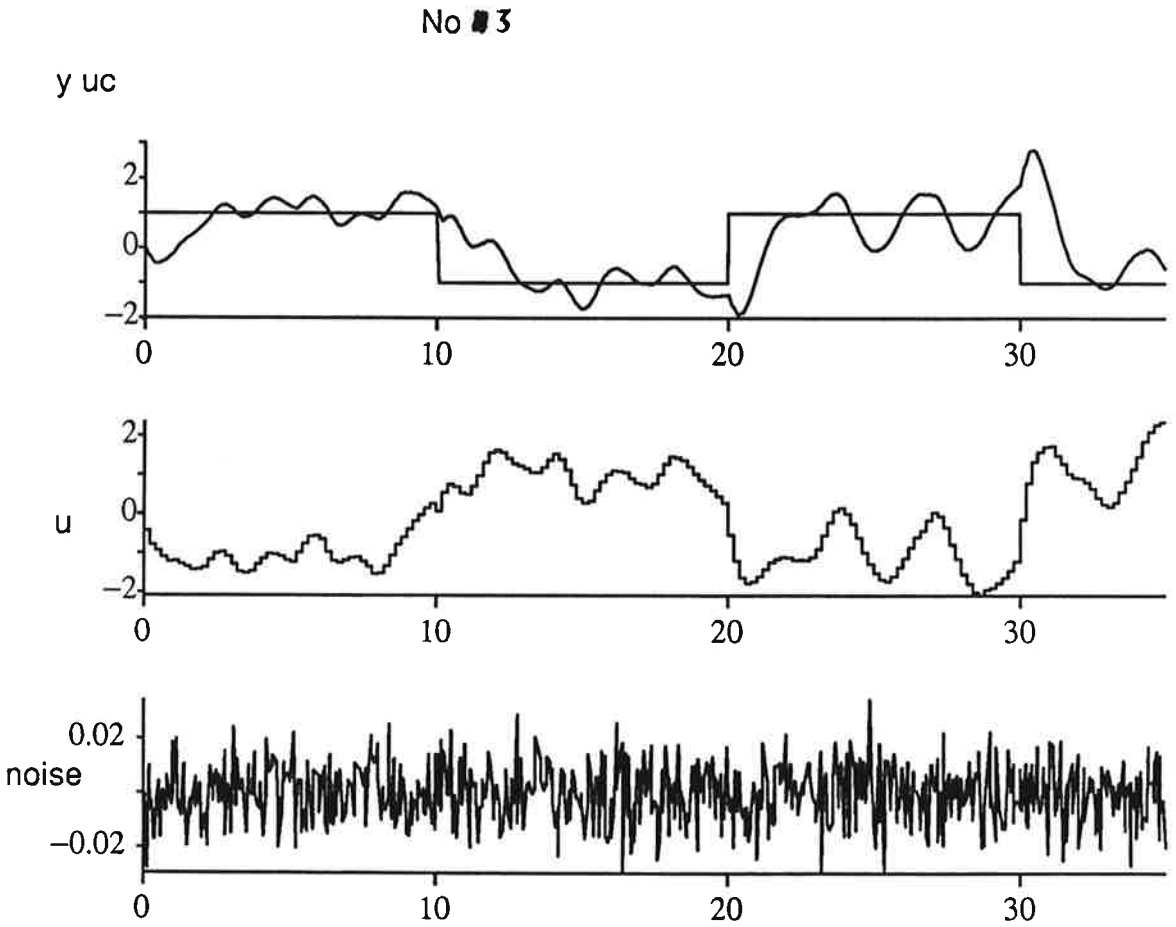

No 2
y uc
20
10
0
−10
0
10
20
30
u
20
10
0
−10
0
10
20
30

Bilaga 2.1.3.1


r and y
6
4
2
0
-2
time [s]
0
2
4
6
8
10
12
14
16
18
20
u
0
-2
-4
-6
time [s]
0
2
4
6
8
10
12
14
16
18
20


No 5
y uc
4000
0
−4000
0
10
20
30
4000
0
u
−4000
0
10
20
30
1
0
noise
−1
0
10
20
30

Bilaga 2.1.3.3



Bilaga 2.2

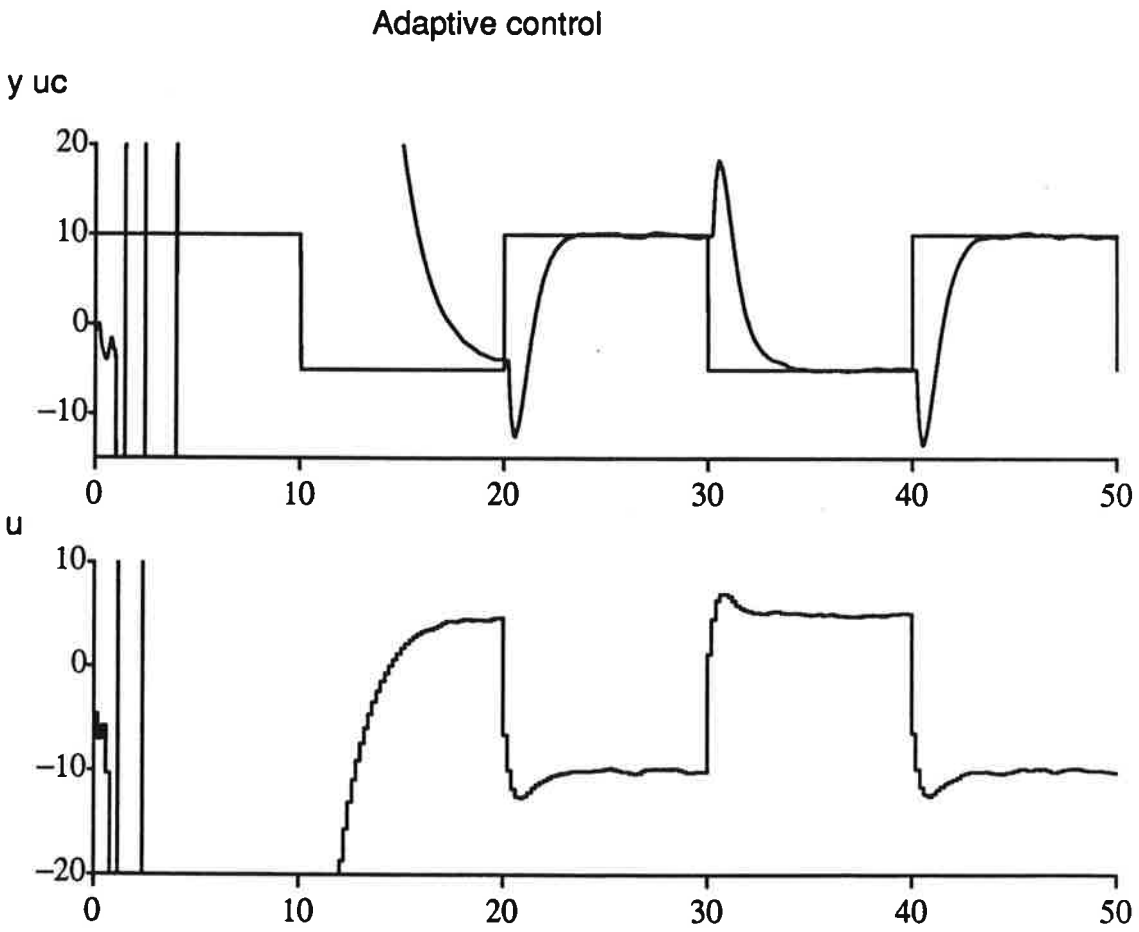
```
DISCRETE SYSTEM areg


INPUT uc y a0 a1 a2 a3 b0 b1
OUTPUT u
STATE u1 u2 u3 y1 y2 y3
STATE uc1 uc2 uc3
NEW nu1 nu2 nu3  ny1 ny2 ny3
NEW nuc1 nuc2 nuc3
TIME t
TSAMP ts


INITIAL
"model-----------------------------------------
am0=1
am1=-2*exp(-2*h)
am2=exp(-4*h)
am3=0
bm0=0.4747
bm1=-0.583393

"observer--------------------------------------
ao0=1
ao1=-3*exp(-h)
ao2=3*exp(-2*h)
ao3=-exp(-3*h)

A=am0*ao1+am1*ao0
B=am0*ao2+am1*ao1+am2*ao0
C=am0*ao3+am1*ao2+am2*ao1+am3*ao0
D=am1*ao3+am2*ao2+am3*ao1
E=am2*ao3+am3*ao2
F=am3*ao3
G=bm1+ao1*bm0
I=ao1*bm1+ao2*bm0
J=ao2*bm1+ao3*bm0

SORT
l1=(A+a0-a1)/a0
B11=B+(a0-a1)*l1+a1-a2
C1=C+(a1-a2)*l1+a2-a3
D1=D+(a2-a3)*l1+a3
E1=E+a3*l1
C2=C1-b1*B11/b0
D2=D1-b1*C2/b0
K1=a1-a0-b1*a0/b0
K2=a2-a1-b1*K1/b0
l2=(E1-b1*D2/b0)/(a3-a2+b0*a3/b1-b1*K2/b0)

"regulator design------------------------------
r1=l1-1
r2=l2-l1
r3=-l2
s0=(B11-a0*l2)/b0
s1=(C2-K1*l2)/b0
s2=(D2-K2*l2)/b0
s3=(F+a3*l2)/b1
t0=-bm0/b0
t1=-(G+t0*b1)/b0
t2=-(I+t1*b1)/b0
t3=-(J+t2*b1)/b0


"control law-----------------------------------
v1=(t0*uc+t1*uc1+t2*uc2+t3*uc3)
```

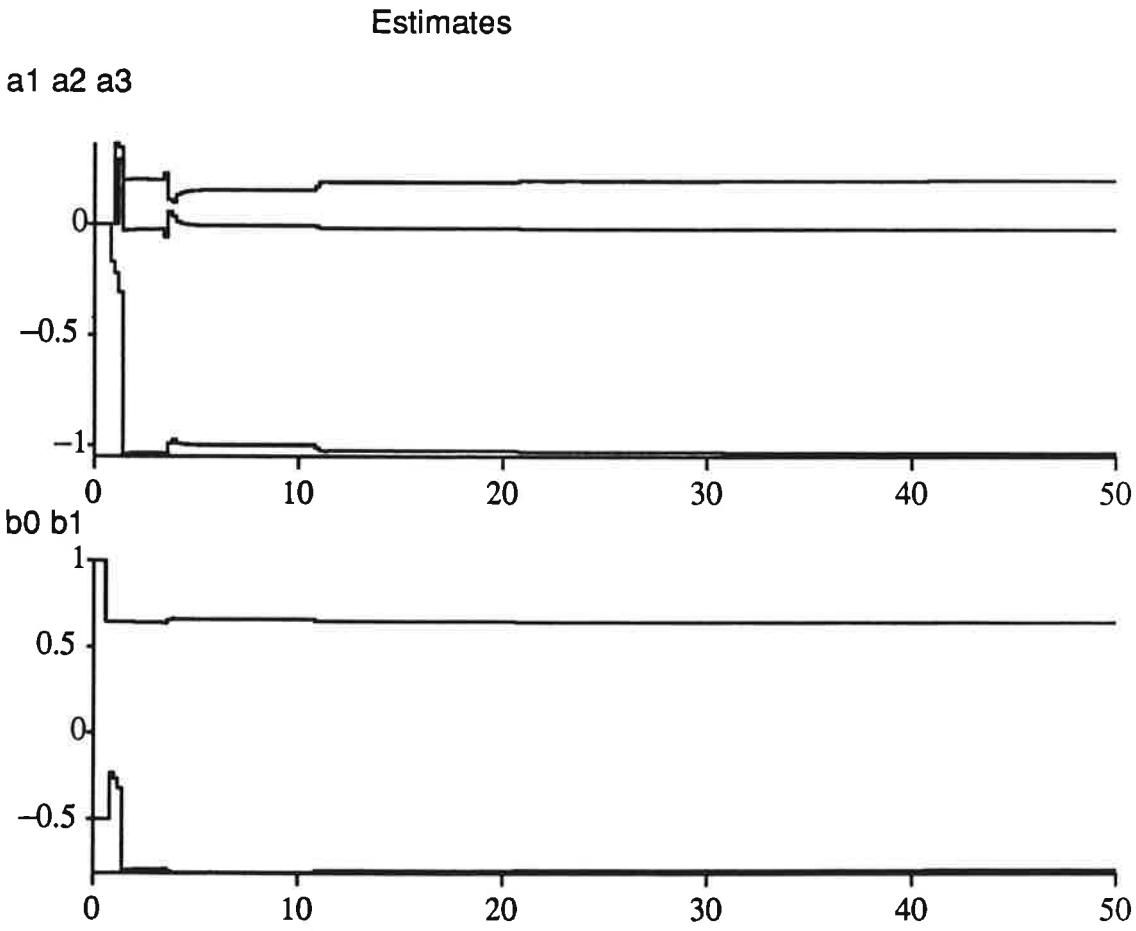
```
v2=(s0*y+s1*y1+s2*y2+s3*y3)
v3=(r1*u1+r2*u2+r3*u3)
u=v1 - v2 - v3

"u=IF v<-ulim THEN -ulim ELSE IF v>ulim THEN ulim ELSE v

"update controller state-------------------------

ny1=y
ny2=y1
ny3=y2

nu1=u
nu2=u1
nu3=u2

nuc1=uc
nuc2=uc1
nuc3=uc2


"update sampling time----------------------------

ts=t+h

"parameters--------------------------------------

h:0.2
ulim:2

END
```

```
DISCRETE SYSTEM estim

"Basic recursive least squares
"
"Estimates the parameters t1 - t5 in the model
"   H(q)=(th4*q+t5)/(q**3+th1*q**2+th2*q+th3)
"using standard recursive least squares algorithm
"
"Parameters
"   lam=exponential forgetting factor (1.0)
"   h  =sampling period

INPUT u y
OUTPUT a0 a1 a2 a3 b0 b1

STATE th1 th2 th3 th4 th5
STATE p11 p12 p13 p14 p15 p22 p23 p24 p25 p33 p34 p35 p44 p45 p55
STATE f1 f2 f3 f4 f5
STATE u1
NEW nth1 nth2 nth3 nth4 nth5
NEW n11 n12 n13 n14 n15 n22 n23 n24 n25 n33 n34 n35 n44 n45 n55
NEW nf1 nf2 nf3 nf4 nf5
NEW nu1

TIME t
TSAMP ts
"
"Computation of P*f
"
pf1=p11*f1+p12*f2+p13*f3+p14*f4+p15*f5
pf2=p12*f1+p22*f2+p23*f3+p24*f4+p25*f5
pf3=p13*f1+p23*f2+p33*f3+p34*f4+p35*f5
pf4=p14*f1+p24*f2+p34*f3+p44*f4+p45*f5
pf5=p15*f1+p25*f2+p35*f3+p45*f4+p55*f5

den=lam+f1*pf1+f2*pf2+f3*pf3+f4*pf4+f5*pf5

eps=y-f1*th1-f2*th2-f3*th3-f4*th4-f5*th5

k1=pf1/den
k2=pf2/den
k3=pf3/den
k4=pf4/den
k5=pf5/den

nth1=th1+k1*eps
nth2=th2+k2*eps
nth3=th3+k3*eps
nth4=th4+k4*eps
nth5=th5+k5*eps

n11=(p11-pf1*k1)/lam
n12=(p12-pf1*k2)/lam
n13=(p13-pf1*k3)/lam
n14=(p14-pf1*k4)/lam
n15=(p15-pf1*k5)/lam
n22=(p22-pf2*k2)/lam
n23=(p23-pf2*k3)/lam
n24=(p24-pf2*k4)/lam
n25=(p25-pf2*k5)/lam
n33=(p33-pf3*k3)/lam
n34=(p34-pf3*k4)/lam
n35=(p35-pf3*k5)/lam
n44=(p44-pf4*k4)/lam
n45=(p45-pf4*k5)/lam
n55=(p55-pf5*k5)/lam
```

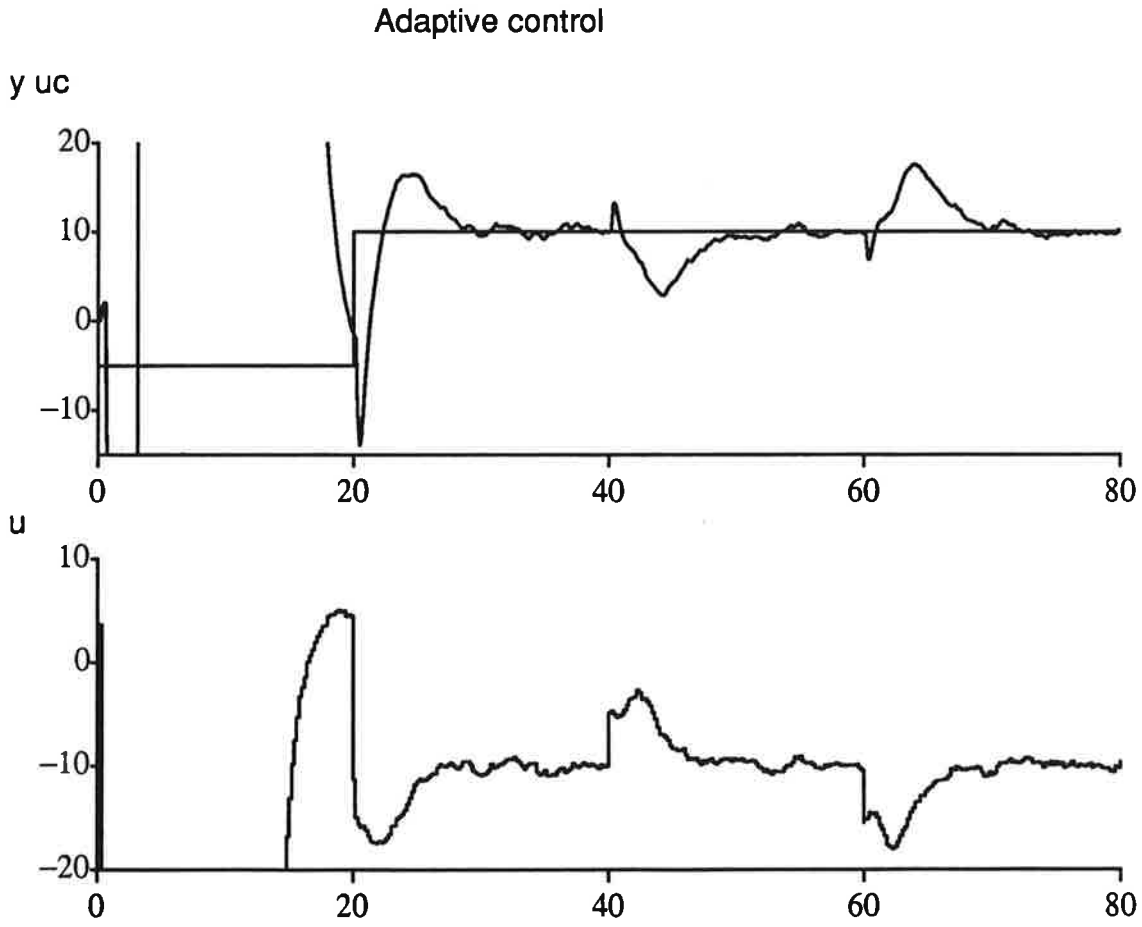
```
nf1=-y
nf2=f1
nf3=f2
nf4=u1
nf5=f4

nu1=u

a0=1
a1=th1
a2=th2
a3=th3
b0=th4
b1=th5


ts=t+h
"
lam:1
h:0.2
"
END
```

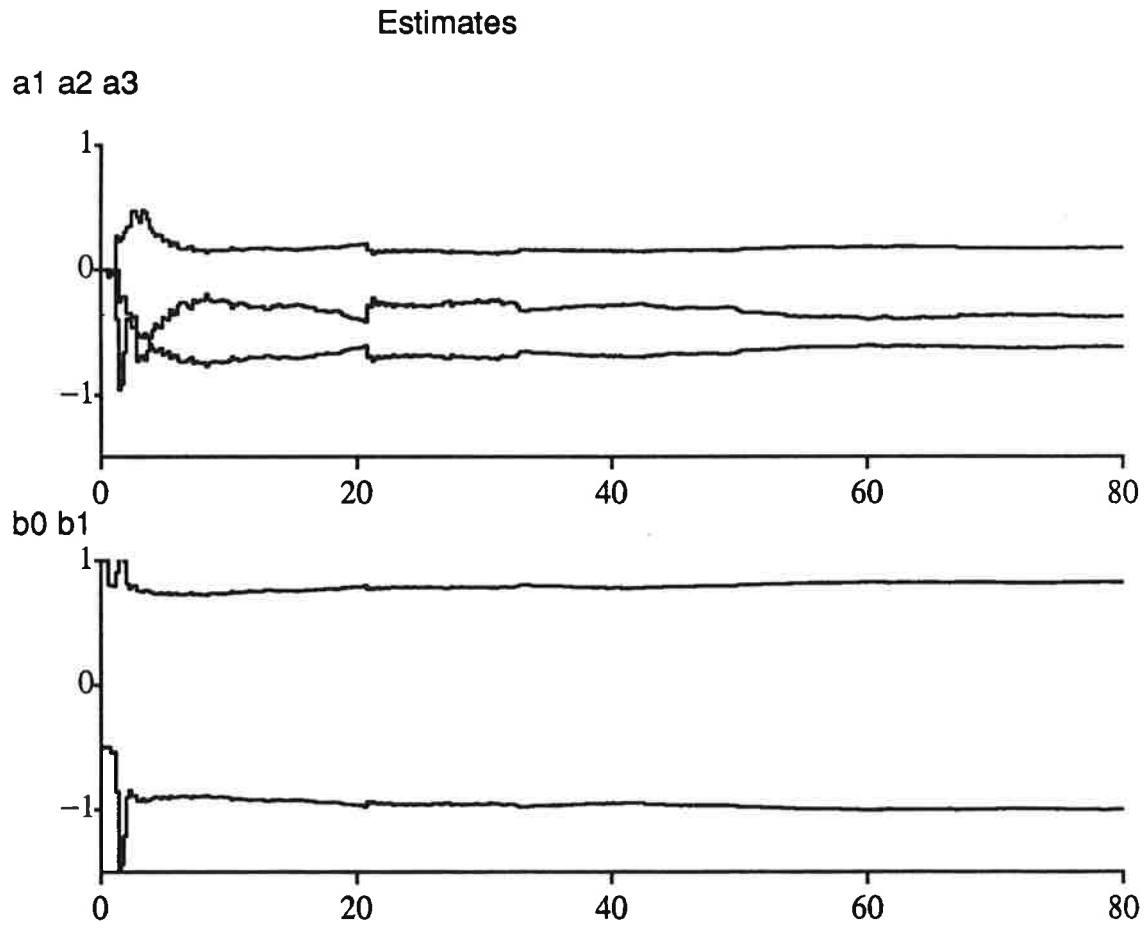
```
CONNECTING SYSTEM acon

time t

noise=n*norm(t)
y[areg]=y[proc]+noise
u[proc]=u[areg]+load
y[estim]=y[proc]+noise
u[estim]=u[areg]
a0[areg]=a0[estim]
a1[areg]=a1[estim]
a2[areg]=a2[estim]
a3[areg]=a3[estim]
b0[areg]=b0[estim]
b1[areg]=b1[estim]

"set point-----------------

uc[areg]=IF t>20 THEN stepup ELSE stepdown
load=IF mod(t,per)<per/2 THEN loadup ELSE loaddown

per:40
stepup:10
stepdown:-5
loadup:0
loaddown:-5
n:1

END
```


Adaptive control
y uc
20
10
0
−10
0
10
20
30
40
50
u
10
0
−10
−20
0
10
20
30
40
50

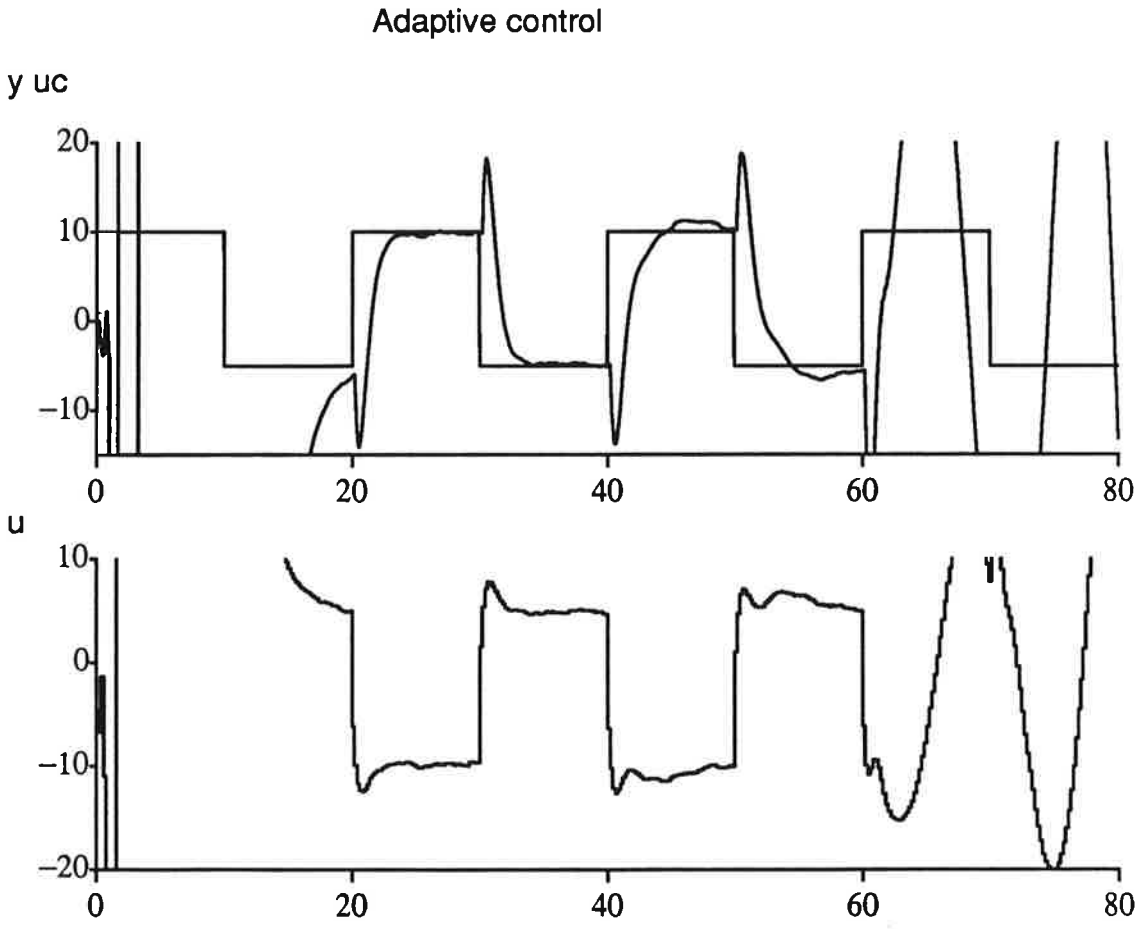

Estimates
a1 a2 a3
0
–0.5
–1
0
10
20
30
40
50
b0 b1
1
0.5
0
–0.5
0
10
20
30
40
50

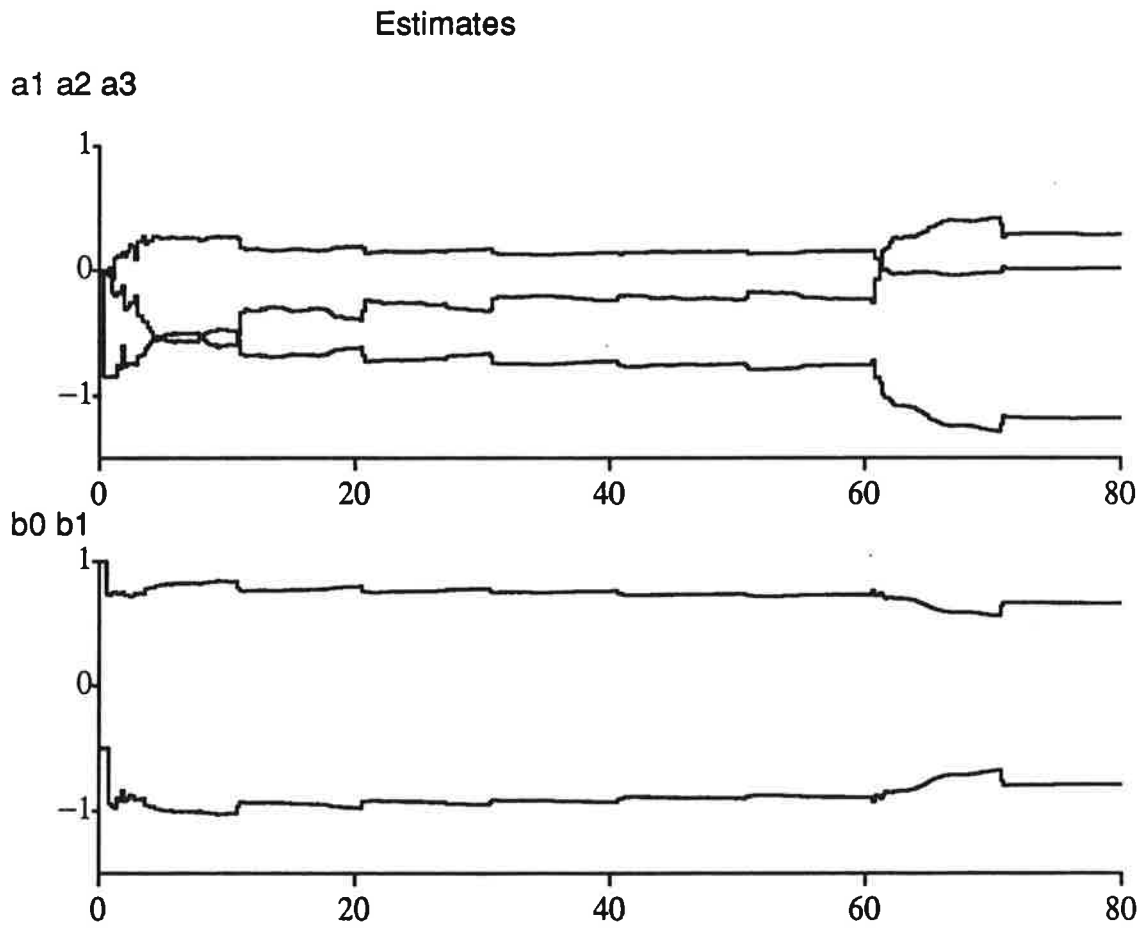
Bilaga 2.2.2.1


Adaptive control
y uc
20
10
0
−10
0
20
40
60
80
u
10
0
−10
−20
0
20
40
60
80


Estimates
a1 a2 a3
1
0
−1
0
20
40
60
80
b0 b1
1
0
−1
0
20
40
60
80

Bilaga 2.2.3.1


Adaptive control
y uc
20
10
0
−10
0
20
40
60
80
u
10
0
−10
−20
0
20
40
60
80


Estimates
a1 a2 a3
1
0
−1
0
20
40
60
80
b0 b1
1
0
−1
0
20
40
60
80

A Study of a Hybrid Direct Self-Tuning Regulator

Henrik Olsson
Ulf Jönsson

Department of Automatic Control
Lund Institute of Technology
May 1991

Table of Contents

1. Introduction

This project was done as a part of the course in adaptive control given by the department during spring 1991. The goal of the project was to investigate the behavior of so called hybrid direct self-tuning regulators. The notion of hybrid comes from the fact that both the process parameters, as in indirect self-tuners, and the controller parameters, as normal in direct self-tuners, are estimated. The algorithm is derived in Section 2 and can also be found in [2]. This type of adaptive controllers are hard to analyze analytically and therefore most of the results in this report are of observational type and more a collection of experience than new theory. The reader is assumed to be familiar with the design and analysis of RST-controllers as described in [3]. All simulations were done using an adaptive toolbox for matlab written by A. Ringdahl as a master thesis, see [4]. Some of the macros and functions however has been rewritten to suit more experimental runs.

2. Hybrid Direct Self-Tuning Regulators

Direct self-tuning regulators as opposed to indirect ones are based on a reparametrization of the process model in the controller parameters. This has the advantage that one can estimate the controller parameters directly instead of first estimating the process parameters and then do computationally complex design calculations for the controller. The algorithm for hybrid direct self-tuning is derived as follows.

Letting the Diophantine equation, with an integrator included in the controller structure, operate on the output y yields

$$A_o A_m y(t) = R_1' \Delta A y(t) + B^- S y(t) = B^-(R_1 \Delta u(t) + S y(t)) \tag{1}$$

where

$$R_1 = B^+ R_1'$$

As can be seen, this model is nonlinear in the parameters since both B^-, R_1 and S are unknown. In the case of minimum phase systems this problem is overcome by letting $B^- = 1$, thus cancelling all the process zeros. If the process is non-minimum phase or if the process zeros are poorly damped this cannot be done. The idea in hybrid direct self-tuning control is to retain the process zeros, i.e. let $B^- = B$ and use an estimate of the B polynomial in Eq. 1, i.e.

$$A_o A_m y(t) = \hat{B}(R_1 \Delta u(t) + S y(t)) \tag{2}$$

Now we define

$$\begin{cases} y_f(t) = \dfrac{\hat{B}}{A_o A_m} y(t) \\ u_f(t) = \dfrac{\hat{B}\Delta}{A_o A_m} u(t) \end{cases} \tag{3}$$

Eq. 2 then becomes

$$\begin{aligned} y(t) &= R_1 u_f(t) + S y_f(t) \\ &= \begin{pmatrix} u_f(t) & y_f(t) \end{pmatrix} \begin{pmatrix} R_1 \\ S \end{pmatrix} \end{aligned} \tag{4}$$

From Eq. 4 the polynomials R_1 and S can be estimated. Assuming a desired steady state gain of 1, we get the relation

$$T = \frac{A_m(1)}{B(1)} A_o = t_0 A_o \quad \text{where} \quad t_0 = \frac{A_m(1)}{B(1)} \tag{5}$$

which gives an estimate of T from

$$\hat{T} = \frac{A_m(1)}{\hat{B}(1)} A_o = \hat{t}_0 A_o \quad \text{where} \quad \hat{t}_0 = \frac{A_m(1)}{\hat{B}(1)} \tag{6}$$

and we have estimates of all the controller parameters. Another way to obtain the controller parameters by estimating all three polynomials together is derived by using Eq. 2 and

$$y_m(t) = \frac{B_m}{A_m} u_c(t) = \frac{TB^-}{A_o A_m} u_c(t) \tag{7}$$

where in our case B^- is substituted by $\hat{B}$. We then get the following equation

$$e(t) = y(t) - y_m(t) = \frac{\hat{B}}{A_o A_m} (R_1 \Delta u(t) + S y(t) - T u_c(t)) \tag{8}$$

where we on the left hand side use

$$y_m(t) = \frac{\hat{B}}{A_m} \frac{A_m(1)}{\hat{B}(1)} u_c(t)$$

Substituting with u_f and y_f as defined in Eq. 3 we get

$$e(t) = y(t) - y_m(t) = R_1 u_f(t) + S y_f(t) - T \frac{\hat{B}}{A_o A_m} u_c(t))$$

From here we can proceed in two different ways, either we use Eq. 5 and define

$$u_{cf}(t) = \frac{\hat{B}}{A_m} u_c(t) \tag{9}$$

to get

$$\begin{aligned} e(t) = y(t) - y_m(t) &= R_1 u_f(t) + S y_f(t) - t_0 u_{cf}(t)) \\ &= \begin{pmatrix} u_f(t) & y_f(t) & -u_{cf}(t) \end{pmatrix} \begin{pmatrix} R_1 \\ S \\ t_0 \end{pmatrix} \end{aligned} \tag{10}$$

and estimate one single coefficient, t_0, to get the T polynomial, or we estimate all coefficients of the T polynomial from

$$\begin{aligned} e(t) = y(t) - y_m(t) &= R_1 u_f(t) + S y_f(t) - T u_{cf'}(t)) \\ &= \begin{pmatrix} u_f(t) & y_f(t) & -u_{cf'}(t) \end{pmatrix} \begin{pmatrix} R_1 \\ S \\ T \end{pmatrix} \end{aligned} \tag{11}$$

where

$$u_{cf'}(t) = \frac{\hat{B}}{A_o A_m} u_c(t) \tag{12}$$

Eq's 4, 10 and 11 are all in standard linear regression form and the unknown parameters can be estimated by e.g. a recursive least squares algorithm as follows, provided that $u(t)$, $y(t)$ and $u_c(t)$ are filtered appropriately as defined in Eq's 3, 9 and 12.

ALGORITHM 1—Recursive Least Squares Estimation with Forgetting Factor
Let the data be generated by

$$y(t) = \phi^T(t)\theta$$

then an estimate of θ is obtained recursively from

$$\begin{cases} \hat{\theta}(t) = \hat{\theta}(t-1) + K(t)(y(t) - \phi^T(t)\hat{\theta}(t-1)) \\ K(t) = P(t)\phi(t) = P(t-1)\phi(t)(\lambda I + \phi^T(t)P(t-1)\phi(t))^{-1} \\ P(t) = (P(t-1) - P(t-1)\phi(t)(I + \phi^T(t)P(t-1)\phi(t))^{-1}\phi^T(t)P(t-1))/\lambda \\ \quad = (I - K(t)\phi^T(t))P(t-1)/\lambda \end{cases} \tag{13}$$

□

Now the adaptive algorithm can be summarized as follows.

ALGORITHM 2—Hybrid Direct Self-Tuning Regulator HDSTR

Step 1 Estimate the process polynomials A and B from the relation

$$Ay(t) = Bu(t) \tag{14}$$

(This relation is easily reformulated to fit the RLS-algorithm.)

Step 2 The estimate $\hat{B}$ from step 1 is now used in one of the following alternatives.

alt. a Estimate R and S from Eq. 4 and compute T from

$$\hat{T} = \frac{A_m(1)}{\hat{B}(1)} A_o$$

alt. b Estimate R , S and t_o from Eq.10 and compute T from

$$\hat{T} = \hat{t}_0 A_o$$

alt. c Estimate R, S and T using Eq. 11.

Step 3 Compute the control signal u using the estimates of the regulator parameters from step 2.

The recursive least squares algorithm given above can be used for the estimation in both steps. Repeat steps 1,2 and 3 at each sampling period. □

The algorithm is represented in a block diagram in Fig. 1

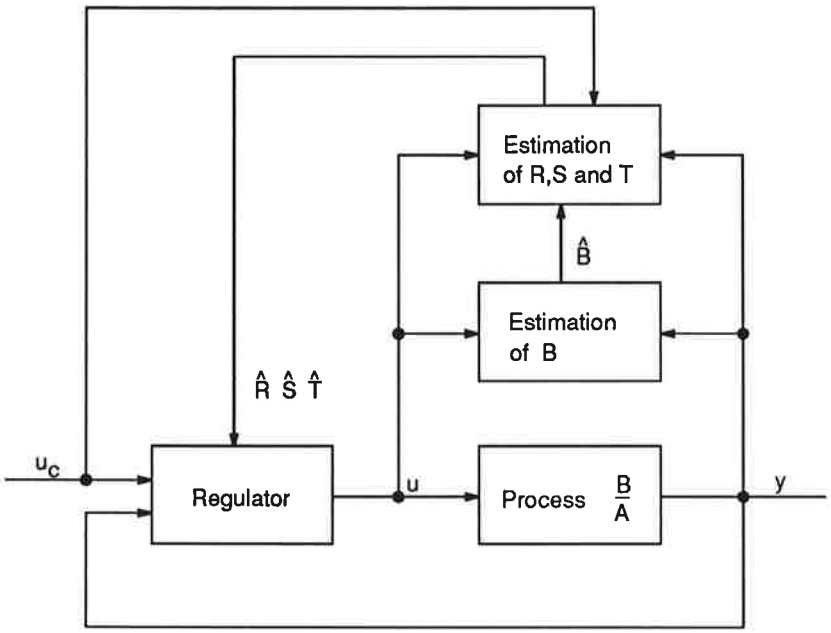


Figure 1. A block diagram of a hybrid direct self-tuning regulator.

In all simulations in this report alternative c of step 2 in the HDSTR-algorithm has been used.

3. Process and design specifications

All experiments in the following sections were done with the same process and desired closed loop specification, namely the process

$$G_p(q) = \frac{0.3648q - 0.3300}{q^2 - 1.7210q + 0.7558}$$

with desired closed loop response

$$G_m(q) = \frac{0.3648q - 0.3300}{q^2 - 1.6375q + 0.6703}$$

and observer polynomial

$$A_o(q) = q^2 - 1.6375q + 0.6703$$

An integrator was always included in the controller.

4. Start-up of the HDSTR algorithm

Start-up of a recursive estimation algorithm requires an initial value of the estimate, $\hat{\theta}$, and of the covariance matrix P, see [6]. Normally all elements of $\hat{\theta}$ are set to a small number and P is set to a large, i.e.

$$\begin{cases} \hat{\theta}(0) = \epsilon \begin{pmatrix} 1 & \dots & 1 \end{pmatrix}^T \\ P(0) = \dfrac{1}{\epsilon^2} I \end{cases}$$

The HDSTR algorithm consists of two recursive estimation algorithms one depending on the result from the other for filtering of its regressors. The start-up of these two algorithms must therefore be done with care. The question is not only how to start the algorithms but also when to start them. The type of initialization described above works for both estimations but only the process estimation can be started at once. The estimation of the regulator parameters which relies on the estimate $\hat{B}$ from the process estimate cannot be started until $\hat{B}$ can be considered certain. The reason is that the estimate $\hat{B}$ of B is used as a filter to build up the regression vector for the estimation in step 2. Since $\hat{B}$ is very uncertain during the first series of recursions, the regressors for step 2 will not be valid for estimation. RLS estimation algorithms, as described in Section 2, are sensitive when the covariance matrix is large which occurs right after initialization and during the first recursions. This sensitive period must occur when the regressors are valid, i.e. when $\hat{B}$ has converged. If the regulator parameter estimator is turned on to soon it will lead to a false certainty in the estimate and a very slow convergence to the true parameters if any. To sum this up, the following start-up procedure is suggested.

HDSTR Start-Up Procedure

Step 1 Start the process estimation, i.e. step 1 in the HDSTR algorithm, in the normal manner cf. Eq. 13 and 14. If needed, this should be done with some stabilizing controller.

Step 2 After the estimate $\hat{B}$ can be considered certain, initialize $\hat{\theta}$ and P of the controller parameter estimator and then switch to the estimated RST-regulator.

□

In Fig. 2, a system is started in the wrong way by turning both estimation recursions on at the same time. This can be compared with the same system started using the procedure above, see Fig. 3. In the case of both estimations started at the same time, the controller parameters do not converge to the desired ones.

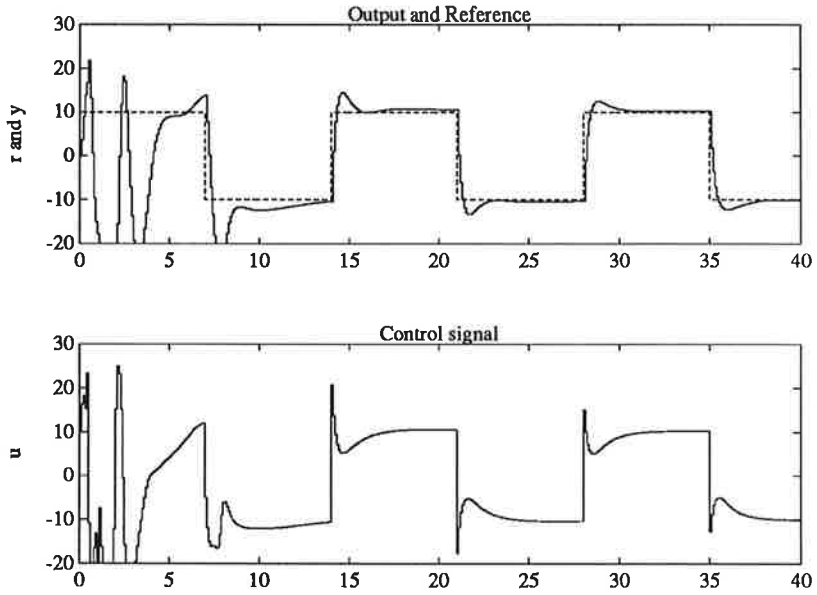


Figure 2. Start-up of both estimations at the same time. The controller does not converge to the same parameters as in deterministic design.

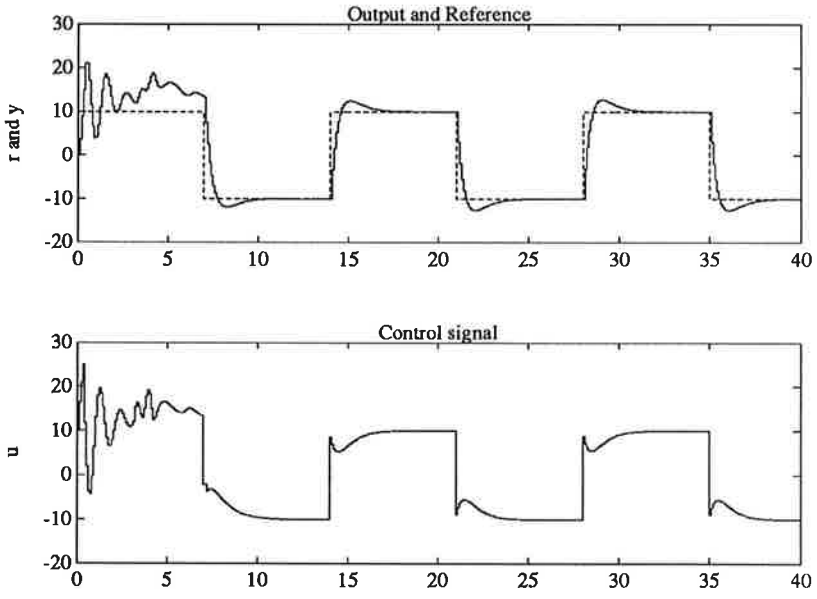


Figure 3. Start-up using a proportional controller for the first 3 seconds before turning the regulator estimator on. The controller parameters converge to the desired ones.

5. Filtering of the regressors

As described in Section 2, there are two different estimations done in the HDSTR algorithm. One of the process parameters and one of the regulator parameters.

5.1 Filtering of regressors for the process estimation

Consider step 1 of Algorithm 2. The estimation done in that step is of the process parameters and in that sense a common type of estimation. It is well known that filtering of the regressors is very important in order to obtain a successful estimate. We therefore rewrite Eq. 14 to include a band pass filter.

$$AH_{bpf}y(t) = BH_{bpf}u(t) \tag{15}$$

or with shorter notation

$$Ay_{bpf}(t) = Bu_{bpf}(t) \tag{16}$$

Filtering of the regressors y and u can be seen as a frequency domain weighting of the loss function to be minimized in the estimation, see [5]. Therefore the band pass filtering puts low weight to frequency ranges that can be corrupted by either load disturbances or high frequency noise, and by doing this emphasizes the region around the cross-over frequency, which is the most interesting part. The accuracy of the estimate of the B polynomial very much affects the achievement of the HDSTR algorithm and therefore much can be gained by careful filtering of the regressors in Eq. 15. Results from estimation of a process with and without band pass filtering is shown in Table 1. The actual input and output signals for the estimation are found in Fig 4. Both noise and a load disturbance acted on the process when the estimation was done. The frequency response of the filter in Eq. 15 and of the process are shown in Fig. 5. The pass band of the filter is chosen as to include the interesting frequency range of the process.

Polynomial	Coefficients	Roots	Stationary gain
A	$q^2 - 1.7210q + 0.7558$	$0.8605 \pm i0.1237$	0.0348
$Ae_{w/of}$	$q^2 - 1.6614q + 0.6679$	$0.9796, 0.6818$	0.0065
Ae_{wf}	$q^2 - 1.6744q + 0.7213$	$0.8372 \pm i0.1429$	0.0469
B	$0.3648q - 0.3300$	0.9047	0.0348
$Be_{w/of}$	$0.3845q - 0.3815$	0.9922	0.0030
Be_{wf}	$0.3568q - 0.3029$	0.8490	0.0539

Table 1. Results from estimation with and without filtering of the regressors. No subscript indicate the true polynomial, subscript $e_{w/of}$ means estimate without filtering and subsequently subscript e_{wf} denotes estimate with filtering.

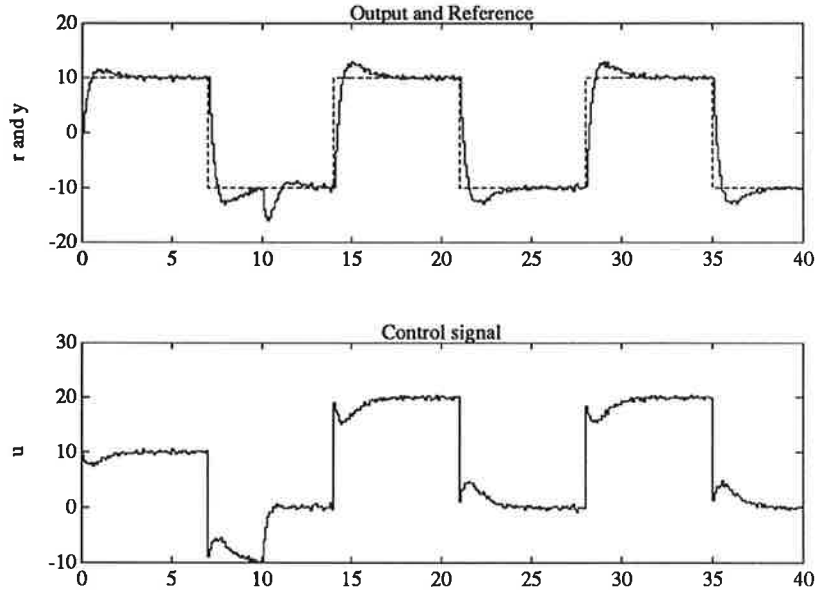


Figure 4. Input and output signals for estimation of the process parameters

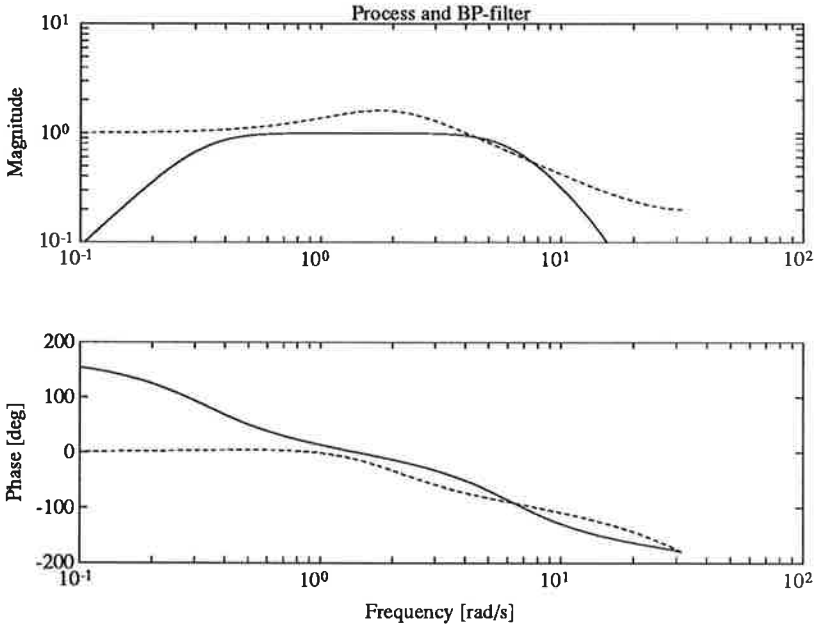


Figure 5. Frequency response for process (dashed) and band pass filter (solid)

5.2 Filtering of the regressors for the regulator estimation

The estimation done in step 2 of the HDSTR algorithm is somewhat more interesting than the one in step 1. The controller structure chosen includes an integrator. This means that the regressor u filtered as in Eq. 3 has been differentiated, i.e. constant bias terms in u such as load disturbances has been filtered out. This means that it is not necessary to filter the regressors with a band pass filter. A low pass filter should suffice. If however the regressors are band pass filtered so that the DC-levels are removed one can expect that the stationary gain, i.e. the steady state ratio between y and u_c of the controller,

can be difficult to estimate. Consider the control law

$$Ru(t) = -Sy(t) + Tu_c(t) \tag{17}$$

If we want $y(t) = u_c(t)$ in steady state we must have

$$S(1) = T(1)$$

because

$$R(1) = R_1(1)\Delta(1) = 0$$

since Δ is the differentiation operator.
The problem with estimating the steady state gain can thus be observed by checking the ratio $\frac{S(1)}{T(1)}$. Three tests were run on both a system with band pass filtering of the regressors and on a system with low pass filtering. The upper cut-off frequency for the filters was chosen with respect to the desired bandwidth of the closed loop system. The first test was run without any disturbances. The result is shown in Fig. 6 and 7. The steady state convergence (and the over-all convergence) is slightly faster for the low pass filtering system. It should however be noted that more information in the regressors is retained if only low pass filtering is done. The second test included measurement noise. The result of the simulations can be found in Fig. 8 and 9. As can be seen, the steady state convergence now is clearly faster for the system with only low pass filtering of the regressors. The third run included a load disturbance after 10 seconds. Once again the system with band pass filtering takes much longer time to find the correct value of $S(1)/T(1)$, see Fig. 10 and 11. This case shows that it is not necessary to remove the DC-level to handle load disturbances when the controller includes an integrator as discussed above, on the contrary, band pass filtering of the regressors deteriorates the performance of the algorithm. It could be an interesting task to examine the need for filtering when there is no integrator forced into the controller, however this project deals only with the case with an integrator included.

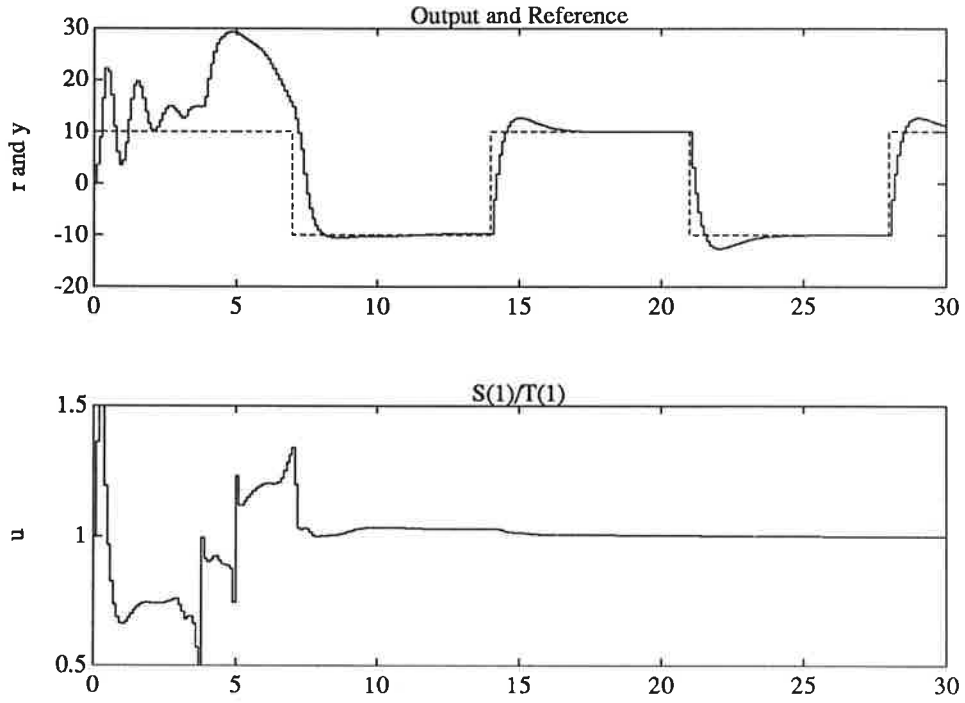


Figure 6. Test 1 (no noise, no load disturbance) for band pass filtered regressors. The algorithm is started with a proportional controller for 3 seconds.

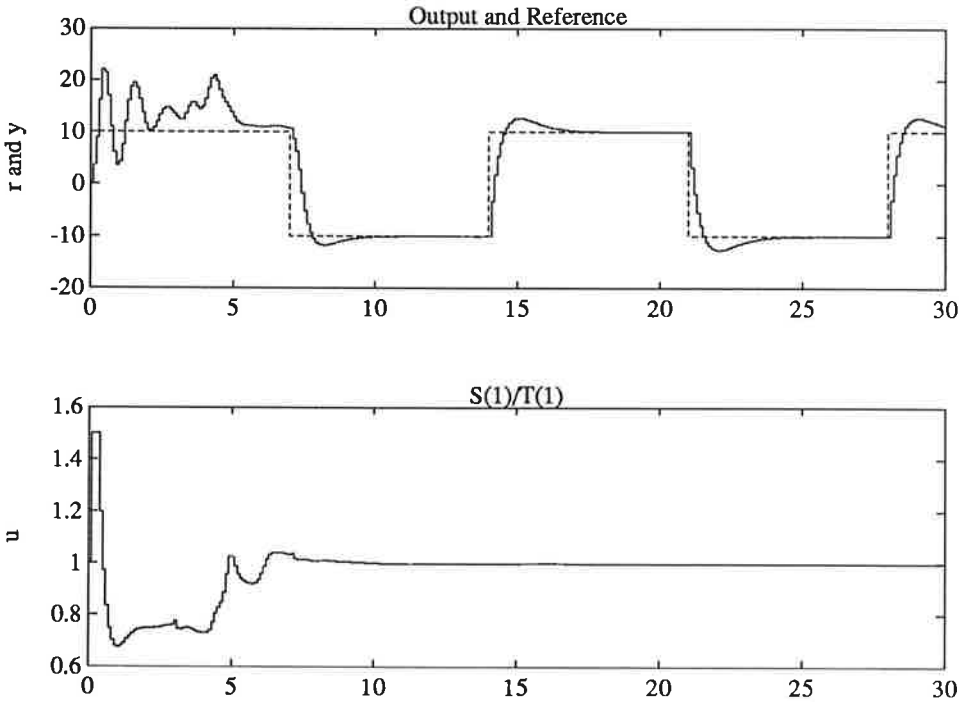


Figure 7. Test 1 (no noise, no load disturbance) for low pass filtered regressors. The algorithm is started with a proportional controller for 3 seconds.

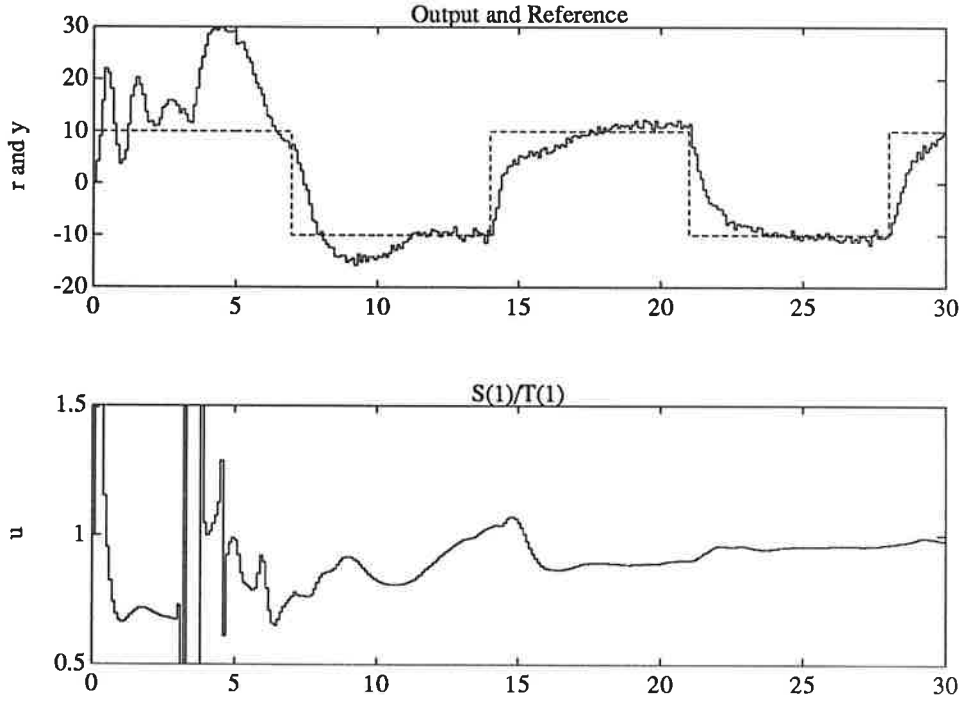


Figure 8. Test 2 including measurement noise for band pass filtered regressors. The algorithm is started with a proportional controller for 3 seconds.

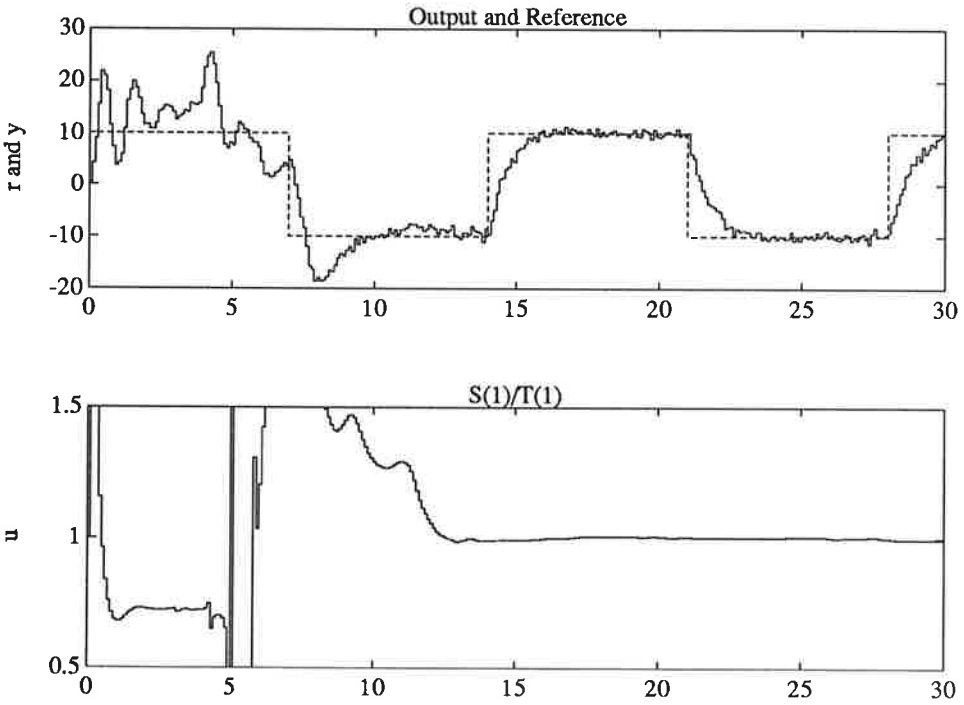


Figure 9. Test 2 including measurement noise for low pass filtered regressors. The algorithm is started with a proportional controller for 3 seconds.

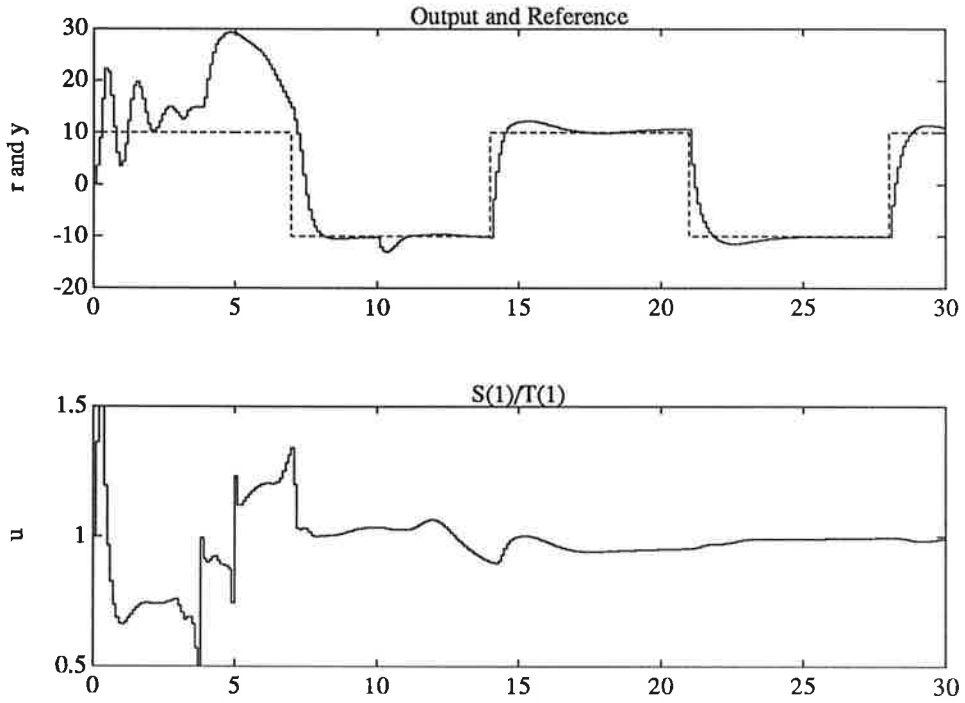


Figure 10. Test 3 for band pass filtered regressors. A load disturbance enters the system after 10 seconds. The algorithm is started with a proportional controller for 3 seconds.

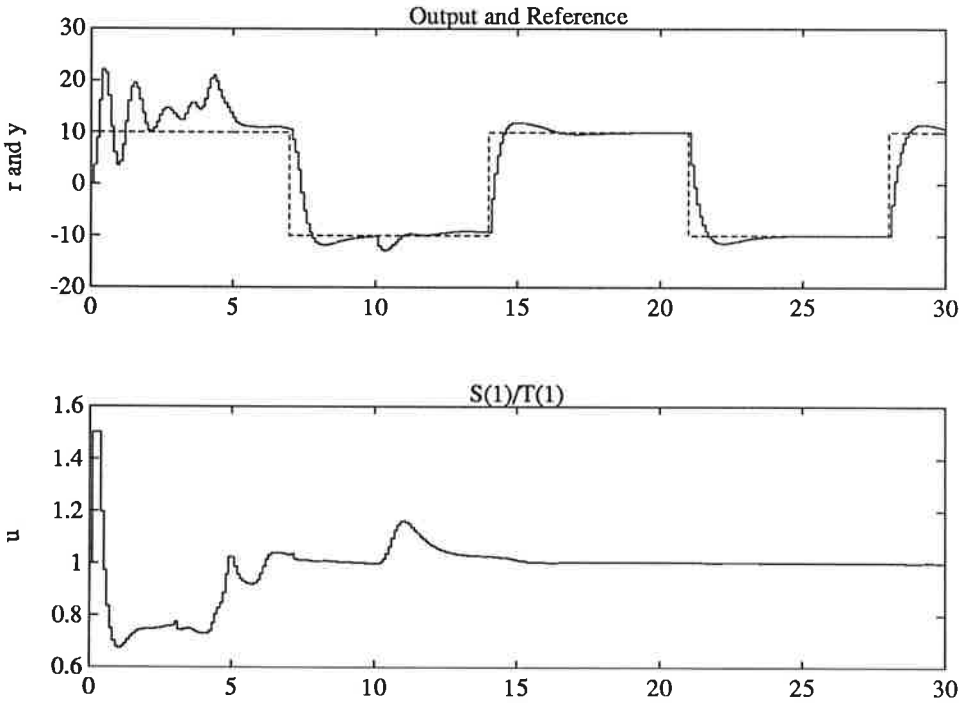


Figure 11. Test 3 for low pass filtered regressors. A load disturbance enters the system after 10 seconds. The algorithm is started with a proportional controller for 3 seconds.

6. Conclusions and suggestions for further work

We have in this report gathered some information about hybrid direct self-tuning regulators gained from simulations. The algorithms have hardly been analyzed and much more work needs to be done. The complexity of the adaptive algorithm makes it hard to study global properties such as convergence and stability. It could be an interesting project to study the algorithm when applied to a first order system. Hopefully some kind of analysis could be carried out for such a simple system. Another topic to study is the influence of the estimate $\hat{B}$ on the outcome of the algorithm. The robustness to changes or errors in $\hat{B}$ is most important. We have in this report only studied one of the three alternatives in step 2 of Algorithm 2. It would therefore be interesting to investigate the advantages and the drawbacks of all three ways to do the second step.

7. References

[1] ÅSTRÖM, K. J. and B. WITTENMARK (1989): *Adaptive Control*, Addison Wesley, Reading Mass.

[2] ÅSTRÖM, K. J. and B. WITTENMARK (1990): *2nd edition of Chapter 5 in [2].*, Unpublished manuscript.

[3] ÅSTRÖM, K. J. and B. WITTENMARK (1990): *Computer Controlled Systems*, Prentice Hall, Englewood Cliffs N.J.

[4] RINGDAHL, A. (1991): "An Adaptive Toolbox for Matlab," Forthcoming master thesis, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

[5] LJUNG, L. (1987): *System Identification - Theory For The User*, Prentice Hall, Englewood Cliffs N.J.

[6] MENDEL, J. (1987): *Lessons in Digital Estimation Theory*, Prentice Hall, Englewood Cliffs N.J.

Projekt i Adaptiv Reglering
Parameterskattning med kvadratrotsalgoritm

Anders Sjö F87

22 april 1991

Innehåll

1 Inledning

I kursen i Adaptiv Reglering tas det upp hur man kan göra parameterskattning genom att lösa minsta-kvadrat-problem. Man visar att en lösning kan erhållas genom att lösa normalekvationen

$$\Phi^T \Phi \hat{\theta} = \Phi^T Y \tag{1}$$

Att lösa normalekvationen är dock olämpligt ur numerisk synvinkel, eftersom metoden dels arbetar med $\Phi^T\Phi$ i stället för Φ och dels för att den är numeriskt instabil. Skulle dessutom matrisen $\Phi^T\Phi$ vara singulär blir problemet olösligt. Nu finns det dock andra, mer numeriskt stabila metoder att lösa minsta-kvadrat-problem, kallade kvadratrotsmetoder eftersom de arbetar direkt med Φ. Genom olika ortogonala transformationer Q erhåller man ett enklare problem att lösa. Den ortogonala transformationen Q kan fås med t ex Householder-, Givens- eller Gram-Schmidt-transformationer. Alla dessa metoder förutsätter att Φ har full rang, om så inte skulle vara fallet kan problemet lösas medelst singulärvärdesfaktorisering (SVD).

I Adaptiv Reglering är man dock inte intresserad av direkta metoder för parameterskattningar, utan endast rekursiva dito, eftersom estimatorerna jobbar i realtid. I [1] studeras en algoritm baserad på normalekvationen relativt ingående

$$\begin{cases} \hat{\theta}(t) = \hat{\theta}(t-1) + K(t)e(t) \\ K(t) = P(t)\varphi(t) = \frac{P(t-1)\varphi(t)}{\lambda + \varphi^T(t)P(t-1)\varphi(t)} \\ P(t) = \left(I - K(t)\varphi^T(t)\right) P(t-1)/\lambda \\ e(t) = y(t) - \varphi^T(t)\hat{\theta}(t-1) \end{cases} \tag{2}$$

Kortfattat nämns även en rekursiv kvadratrotsmetod, som använder sig av en transformationsmetod kallad *Dyadisk transformation.*

Mitt projekt gick ut på att studera Dyadisk transformation och försöka förstå hur den kan appliceras på minsta-kvadrat-problemet, därtill skulle jag undersöka om, och i så fall, hur denna hör samman med Housholder-, Givens- och Gram-Schmidt-transformatinerna. Jag har gjort en del simuleringar på olika problem för att jämföra den Dyadiska transformationsalgoritmen med den i [1] så flitigt använda estimeringsalgoritmen. För att visa att det kan uppstå problem då man löser normalekvationen har jag med hjälp av **Matlab** löst en (av mig själv) något utvidgad upplaga av uppgift 3.15 i [1] med olika numeriska metoder.

(Projektrapportens lay-out kan kanske te sig en aning överarbetad, det anser i alla fall halva familjen Sjö, men bör ses mer som ett försök av författaren att lära sig Slatex. Dessutom har det varit så roligt att göra det här projektet, att det för mig själv inte känts som om jag varit överambitiös!)

2 Överbestämda ekvationssystem

2.1 Det ursprungliga problemet

Vi antar att vi har en modell som kan skrivas på följande form

$$y(t) = \varphi_1(t)\theta_1 + \varphi_2(t)\theta_2 + ... + \varphi_n(t)\theta_n = \varphi^T(t)\theta \tag{3}$$

Antag vidare att vi har värden på $y(t)$ för $t = 1, 2, ..., t$. Då kan vi skriva

$$Y(t) = \Phi(t)\theta \tag{4}$$

där

$$Y(t) = [\ y(1) \quad y(2) \quad \dots \quad y(t)\ ]^T$$

$$\Phi(t) = \begin{pmatrix} \varphi^T(1) \\ \varphi^T(2) \\ \\ \varphi^T(t) \end{pmatrix}$$

(4) är ett överbestämt ekvationssystem, vilket löses med minsta-kvadrat-metoden. Bilda residualen

$$E = Y - \Phi\theta \tag{5}$$

Minimera därpå normen av residualen

$$min||E||_2 \tag{6}$$

I [1] visas det att normalekvationen (1) ger en lösning till (6), men som tidigare påpekats är denna metod mycket störningskänslig och ger dålig precission. Jag kommer härför att presentera en del andra och i detta fall bättre metoder för lösning av det överbestämda ekvationssystemet i (4).

2.2 Gram-Schmidt (klassisk)

Klassisk Gram-Schmidt är ett exempel på en metod som *inte* är numeriskt stabil, och ger inte mycket bättre värden än lösning av normalekvationen. Det finns dock en modifierad variant med återkommande reortogonaliseringar som är numeriskt stabil och används i praktiken. Att lösa (6) med klassisk Gram-Schmidt påminner dock en del om att lösa normalekvationen och kan därför vara av intresse.

Gram-Schmidt transformerar m×n-matrisen Φ till formen

$$\Phi = QR \tag{7}$$

där R är en högertriangulär n×n-matris och Q är en m×n-matris med ortogonala kolonner,

dvs

$$Q^T Q = D$$

där D är en n×n-diagonalmatris. Lösningen till (6) ges då av

$$\hat{\theta} = R^{-1} D^{-1} Q^T Y \tag{8}$$

Vi kan skriva (7) som

$$\begin{pmatrix} \varphi_1 & \varphi_2 & \dots & \varphi_n \end{pmatrix} = \begin{pmatrix} q_1 & q_2 & \dots & q_n \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ & r22 & \dots & r_{2n} \\ & & & \\ & & & r_{nn} \end{pmatrix} \tag{9}$$

vilket ger att

$$\begin{array}{lll} \varphi_1 = r_{11} q_1 & \Rightarrow \quad r_{11} = \|\varphi_1\|_2 & \text{(Unitär invarians)} \\ \varphi_2 = r_{12} q_1 + r_{22} q_2 & \Rightarrow \quad q_1^T \varphi_2 = r_{12} & (q_1 \text{ och } q_2 \text{ ortogonala}) \\ q_2 = (\varphi_2 - r_{12} q_1) / r_{22} & & \\ \text{osv...} & & \end{array}$$

Vi får den generella formen för transformationen

$$\begin{cases} r_{jj} = \|\varphi_j\|_2 \\ r_{ij} = q_i^T \varphi_j \\ q_j = \left(\varphi_j - \sum_{i=1}^{j-1} r_{ij} q_i \right) / r_{jj} \end{cases} \tag{10}$$

2.3 Ortogonala transformationer

Jag skall nu ta upp två metoder som i grund och botten använder sig av samma princip. Det gäller att överföra m×n-matrisen Φ till en triangulär m×n-matris R, dvs de m-n sista raderna i matrisen är nollor, genom att från vänster multiplicera på ortogonala transformationsmatriser Q_i så att

$$\Phi_{i+1} = Q_i \Phi_i \tag{11}$$

med

$$\Phi_1 = \Phi \quad \text{och} \quad \Phi_n = R$$

och

$$Q_i = \begin{pmatrix} I_{i-1} & 0 \\ 0 & Q \end{pmatrix}$$

Q_i:s uppgift är att nollställa element i Φ-matrisens i:te kolonnvektor. I själva verket multiplicerar man hela ekvation (5) med ortogonala transformatorer Q_i tills man har fått den på formen

$$\begin{pmatrix} \tilde{e}_1 \\ \tilde{e}_2 \end{pmatrix} = \begin{pmatrix} \tilde{y}_1 \\ \tilde{y}_2 \end{pmatrix} - \begin{pmatrix} \tilde{\Phi} \\ 0 \end{pmatrix} \theta \tag{12}$$

där $\tilde{\Phi}$ är en högertriangulär n×n-matris.

Problemet vi har att lösa är:
Avbilda en given vektor u på en multipel av e_1 med en ortogonal matris Q

$$\alpha e_1 = Qu \quad \Rightarrow \quad \alpha = \pm \|u\|_2 \tag{13}$$

eftersom vi har unitär invarians.

2.3.1 Givens plana rotationer

Att det kallas plana rotationer beror på att Q vrider vektorn u så att man nollställer ett element i vektorn åt gången. Matrisen Q har följande utseende

$$Q = \begin{pmatrix} 1 & & & & & & \\ & 1 & & & & & \\ & & c & & & s & \\ & & & 1 & & & \\ & & & & 1 & & \\ & & -s & & & c & \\ & & & & & & 1 \end{pmatrix}$$

där

$$c = \cos\gamma \quad \text{och} \quad s = \sin\gamma$$

Två element i vektorn u kommer att påverkas. Låt dessa ha värdena a resp b och antag att man vill nolla ut det andra elementet. Detta innebär att man skall välja

$$as = bc \quad \Rightarrow \quad \begin{cases} c = \frac{a}{\sqrt{a^2+b^2}} \\ s = \frac{b}{\sqrt{a^2+b^2}} \end{cases}$$

Vidare är Q en skevsymmetrisk matris med egenskaperna

$$Q^T Q = I \quad \text{och} \quad Q^{-1} = Q^T$$

2.3.2 Householder-transformationer

Ibland kallas Householder-transformationerna för Householder-reflektorer, ity transformationen innebärdet i själva verket en spegling. Antag att vektorerna u och v är givna så att

$$\|u\|_2 = \|v\|_2 \tag{14}$$

Det gäller att finna en ortogonal matris Q sådan att

$$Qu = v$$

Normalvektorn w till planet mellan u och v beräknas som

$$w = \frac{u - v}{\|u - v\|_2}$$

varvid v kan beräknas som

$$v = u - 2ww^T u = \left(I - 2ww^T\right) u$$

Householder-reflektorn blir alltså

$$Q = I - ww^T$$

och har egenskaperna

$$Q^T = Q, \quad Q^T Q = I \quad \text{och} \quad Q^{-1} = Q^T$$

Man ser dock att det finns två möjliga Householder-speglingar som uppfyller (14), dvs förutom den ovan beskrivna transformationen existerar även

$$\bar{Q}u = -v$$

där

$$\bar{Q} = I - \bar{w}\bar{w}^T$$

med

$$\bar{w} = \frac{u + v}{\|u + v\|_2}$$

Man finner att det är numerisk mest fördelaktigt att välja Q så att det blir en "trubbig vinkel" mellan u och v.

2.3.3 QR-metoden

En tredje metod, vilken också kan användas för ortogonal transformation av (5) så att (12) erhålles, är QR-metoden. Den transformerar dock Φ på ett något annorlunda sätt än vid Gram-Schmidt. Q blir i detta fall en ortogonal m×m-matris, sådan att

$$Q^T Q = I_m$$

och R blir en m×n-matris på formen

$$R = \begin{pmatrix} \tilde{\Phi} \\ 0 \end{pmatrix}$$

där $\tilde{\Phi}$ är en högertriangulär n×n-matris. Den ortogonala transformationen i (11) ges således av Q^T.

2.4 Singulärvärdesfaktorisering (SVD)

Då matrisen Φ inte har full rang går det inte att använd någon av de ovan nämnda metoderna, utan då får man tillgripa singulärvärdesfaktorisering (SVD). Även om Φ har full rang men rangen är väldigt "dålig" kan det vara lämpligt att använda sig av SVD.

Vid SVD faktoriserar man m×n-matrisen Φ enligt

$$\Phi = U\Sigma V = U_r \Sigma_r V_r \tag{15}$$

där U och V är unitära m×m- resp n×n-matriser och Σ är en m×n-matris med följande utseende

$$\Sigma = \begin{pmatrix} \Sigma_r & 0 \\ 0 & 0 \end{pmatrix}$$

där Σ_r är en r×r-diagonalmatris med diagonalelementen

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$$

vidare är $r = rang(\Phi)$ och U_r och V_r är unitära m×r- resp n×r-matriser. Att matriserna är unitära innebär att

$$U_r^T U_r = I_r \;\; ; \;\; U_r U_r^T \neq I$$
$$V_r^T V_r = I_r \;\; ; \;\; V_r V_r^T \neq I$$

Ställer vi upp normalekvationen (1) och sätter in SVD av Φ får vi

$$V_r \Sigma_r^2 V_r^T \theta = V_r \Sigma_r U_r^T Y$$

Multiplikation med $\Sigma_r^{-2} V_r^T$ från vänster ger

$$V_r^T \theta = \Sigma_r^{-1} U_r^T Y \tag{16}$$

Vi får nu två fall av lösningar

1. Om $r = n$, dvs Φ har full rang, så blir

$$V_n V_n^T = I_n$$

$$\hat{\theta} = V_n \Sigma_n^{-1} U_n^T Y$$

2. Om $r < n$ så delas θ upp i

$$\theta = V_r \tilde{\theta} + \bar{V}_r \bar{\theta} \tag{17}$$

där $\bar{V}_r$ är de kolonner i V som inte ingår i V_r. Vi får då eftersom V_r och $\bar{V}_r$ är ortogonala

$$V_r \tilde{\theta} = V_r \Sigma_r^{-1} U_r^T Y \quad \Rightarrow \quad \tilde{\theta} = \Sigma_r^{-1} U_r^T Y$$

medan $\bar{V}_r \bar{\theta}$ är obestämd, dvs vi har flera lösningar. Man finner dock att normen av θ i (17) minimeras då

$$\bar{\theta} = 0 \quad \Rightarrow \quad \hat{\theta} = V_r \tilde{\theta} = V_r \Sigma_r^{-1} U_r^T Y \tag{18}$$

Den allmäna lösningen vid SVD ges alltså av

$$\hat{\theta} = V_r \tilde{\theta} = V_r {\Sigma_r}^{-1} {U_r}^T Y = \Phi^{\dagger} Y$$

där $\Phi^{\dagger}$ kallas *Moore-Penrose-inversen* och är en *pseudo-invers.*

I vissa fall kan Φ ha dålig rang, dvs

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_i \gg \sigma_{i+1} \geq \ldots \geq \sigma_r > 0$$

Eftersom Σ_r inverteras kommer de små singulärvärdena att helt dominera över de stora, trots att de kan ha uppkommit genom brus eller störningar. Det kan därför vara på sin plats att anta att rangen är mindre än r, dvs vi sätter helt sonika de små singulärvärdena till noll. Därefter löser man (6) på nytt med den nya rangen.

2.5 Dyadisk transformation

En del av projektet var att undersöka huruvida de ovan nämda transformationerna har något gemensamt med den Dyadiska transformationen. Dyadisk transformation tillgår på följande sätt:
Antag att vektorna a och b är givna enligt

$$a = [\ 1 \quad a_2 \quad \ldots \quad a_n\ ]^T$$

$$b = [\ b_1 \quad b_2 \quad \ldots \quad b_n\ ]^T$$

och att även skalärerna α och β är givna.

Beräkna nya vektorer

$$\tilde{a} = [\ 1 \quad \tilde{a}_2 \quad \dots \quad \tilde{a}_n\ ]^T$$

och

$$\tilde{b} = [\ 0 \quad \tilde{b}_2 \quad \dots \quad \tilde{b}_n\ ]^T$$

och skalärer $\tilde{\alpha}$ och $\tilde{\beta}$ enligt

$$\tilde{\alpha}\tilde{a}\tilde{a}^T + \tilde{\beta}\tilde{b}\tilde{b}^T = \alpha a a^T + \beta b b^T \tag{19}$$

Efter en halv matsked identifikation av termer och en åttnapart av en fjäradels kaffekopp ekvationsmanipulation, vilket utförs i [1], kommer man fram till att de nya termerna i (19) kan beräknas enligt

$$\begin{cases} \tilde{\alpha} = \alpha + \beta b_1^2 \\ \tilde{\beta} = \alpha\beta/\tilde{\alpha} \\ \tilde{b}_k = b_k - b_1 a_k & k = 2, \dots, n \\ \tilde{a}_k = a_k - \beta b_k \tilde{b}_k/\tilde{\alpha} & k = 2, \dots, n \end{cases}$$

Vad har då Dyadisk transformation gemensamt med de övriga transformationerna? Vid en första anblick finner man det svårt att se några likheter över huvud taget. Av denna anledning beslöt jag mig för att undersöka ett mindre och förhoppningsvis enklare problem: Finn en ortogonal transformation sådan att

$$QA = \tilde{A} \tag{20}$$

där

$$Q = \begin{pmatrix} q_{11} & q_{12} \\ q_{21} & q_{22} \end{pmatrix} = \begin{pmatrix} q_1 & q_2 \end{pmatrix}$$

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

och $\tilde{A}$:s koefficienter beräknas enligt (19).

Kan man lösa detta problem, kan man förhoppningsvis gå vidare och finna en ortogonal transformation för en allmän matris A.

Vi försöker transfomera A till en högertriangulär matris. Vi har då

$$A = \begin{pmatrix} \sqrt{\alpha} & \sqrt{\alpha}a \\ \sqrt{\beta}b_1 & \sqrt{\beta}b_2 \end{pmatrix}$$

$$\tilde{A} = \begin{pmatrix} \sqrt{\tilde{\alpha}} & \sqrt{\tilde{\alpha}}\tilde{a} \\ 0 & \sqrt{\tilde{\beta}}\tilde{b} \end{pmatrix} = \begin{pmatrix} \tilde{a}_1 & \tilde{a}_2 \end{pmatrix} \tag{21}$$

Identifierar vi sedan koefficienterna i (20) får vi

$$\begin{cases} \tilde{a}_1 = a_{11}q_1 + a_{21}q_2 \\ \tilde{a}_1 = a_{11}q_1 + a_{21}q_2 \end{cases}$$

Vi vill bl a att Q:s kolonner skall vara ortogonala, varför vi multiplicerar med q_1^T resp q_2^T från vänster och får därmed

$$\begin{cases} a_{11} = q_1^T \tilde{a}_1 \\ a_{12} = q_1^T \tilde{a}_2 \\ a_{21} = q_2^T \tilde{a}_1 \\ a_{22} = q_2^T \tilde{a}_2 \end{cases}$$

Löser vi därpå ut elementen i Q får vi

$$\begin{cases} q_{11} = a_{11}/\tilde{a}_{11} = \sqrt{\alpha/\tilde{\alpha}} \\ q_{12} = a_{21}/\tilde{a}_{11} = b_1\sqrt{\beta/\tilde{\alpha}} \\ q_{21} = (a_{12} - a_{11}\tilde{a})/\tilde{a}_{22} = \ldots = -b_1\sqrt{\beta/\tilde{\alpha}} \\ q_{22} = (a_{22} - a_{21}\tilde{a})/\tilde{a}_{22} = \ldots = \sqrt{\alpha/\tilde{\alpha}} \end{cases}$$

Vi ser då till vår stora glädje att Q är en ortoganal matris. Genomför man sedan matrismultiplikationen i (20), finner man att man verkligen erhåller matrisen $\tilde{A}$ i (21).

Q är skevsymmetrisk, vilket också var fallet för transformatinsmatrisen vid Givens-transformation. Tittar man närmare på Q så ser man att den är identisk med den transformationsmatris man skulle fått vid Givens-transformation. Q:s utseende bestäms bara av A:s första kolonnvektor, vilken alltså motsvarar u i (13). Man kan tycka att man borde ha en viss frihet vid valet av β, b_1 och b_2, vilket i och för sig är sant, men man finner att i räkningarna ovan förekommer dessa bara som $\sqrt{\beta}b_1$ och $\sqrt{\beta}b_2$, varför denna uppdelning saknar betydelse.

2.6 Lösning av ett överbestämt ekvationssystem

2.6.1 Uppgift 3.15

I uppgift 3.15 i [1] gäller det att skatta de två parametrarna i

$$y(t) = b_1u(t) + b_2u(t-1)$$

med utgående från tre värden på insignalen och två värden på utsignalen, först genom att lösa normalekvationen och sedan med någon kvadratrotsmetod. För att tydligare kunna se nackdelarna med att lösa normalekvationen utvidgade jag problemet till följande:
Antag processen

$$y(t) = b_1u(t) + b_2u(t-1) + b_3u(t-2) + b_4u(t-3)$$

Följande mätvärden är genererade med b_1, b_2, b_3 och b_4 samtliga lika med ett

t	u	y
1	1000000000	-
2	1000000001	-
3	1000000000	-
4	1000000000	4000000001
5	1000000001	4000000002
6	1000000000	4000000001
7	999999999	4000000000
8	999999999	3999999999
9	1000000001	3999999999
10	999999999	3999999998

Vi får då

$$\Phi(10) = \begin{pmatrix} 1000000000 & 1000000000 & 1000000001 & 1000000000 \\ 1000000001 & 1000000000 & 1000000000 & 1000000001 \\ 1000000000 & 1000000001 & 1000000000 & 1000000000 \\ 999999999 & 1000000000 & 1000000001 & 1000000000 \\ 999999999 & 999999999 & 1000000000 & 1000000001 \\ 1000000001 & 999999999 & 999999999 & 1000000000 \\ 999999999 & 1000000001 & 999999999 & 999999999 \end{pmatrix}$$

$$Y(10) = \begin{pmatrix} 4000000001 \\ 4000000002 \\ 4000000001 \\ 4000000000 \\ 3999999999 \\ 3999999999 \\ 3999999998 \end{pmatrix}$$

2.6.2 Resultat

Med hjälp av **Matlab** löste jag därpå (6) medelst Normalekvationen, klassisk Gram-Schmidt, Givens-transformation, Householder-transformation, QR-metoden samt SVD. De uträknade parametrarna och $\|E\|_2$ finns återgivna i tabellen nedan. För SVD har jag tagit med resultaten för olika antaganden om Φ:s rang r. Φ:s singulärvärden är som följer

$$\begin{aligned} \sigma_1 &= 5.29150 \cdot 10^9 \\ \sigma_2 &= 2.55165 \\ \sigma_3 &= 2.22187 \\ \sigma_4 &= 1.12547 \end{aligned}$$

vilket antyder att $r = 1$ snarare än 4. Med **Matlab**:s kommando **Rank**(Φ) får man dock $r = 4$.

	b_1	b_2	b_3	b_4	$\|\|E\|\|_2$
Normalekvationen	1.8757155	−0.3318719	1.6267925	0.0029297	$2.18654 \cdot 10^9$
Gram-Schmidt	1.0000002	1.0000001	0.99999999	1.0000001	$1.02306 \cdot 10^3$
Givens	1.0000001	0.9999999	1.0000003	0.9999997	$3.12683 \cdot 10^{-6}$
Householder	0.9999997	1.0000001	0.9999999	1.0000004	$8.25906 \cdot 10^{-7}$
QR-metoden	1.0000000	1.0000000	1.0000000	1.0000000	$1.26159 \cdot 10^{-6}$
SVD $r = 1$	1.0000000	1.0000000	1.0000000	1.0000000	0.00000
SVD $r = 2$	0.9999999	1.0000001	1.0000003	0.9999999	78.8495
SVD $r = 3$	0.9999998	1.0000002	1.0000003	1.0000001	$1.10956 \cdot 10^3$
SVD $r = 4$	0.9999996	1.0000001	1.0000000	1.0000004	$2.15142 \cdot 10^2$

Nu kan man observera en del saker:

1. Alla metoder lyckas skatta parametrarna bra utom Normalekvationen, som har en katastrofal felnorm ($\sim 10^9$).
2. SVD med $r = 1$ ger $||E||_2 = 0$.
3. Givens-, Householder- och QR-metoderna ger bra resultat, dvs liten felnorm ($\sim 10^{-6}$).
4. Gram-Schmidt och SVD med $r = 2$, 3 och 4 ger bra parametervärden men felnormen är ändå stor ($\sim 10^2 - 10^3$).

2.6.3 Varför blev det så?

Konditionstalet för en matris A definieras som

$$\kappa_2[A] = ||A||_2||A^{-1}||_2$$

och används för att bestämma hur välkonditionerat ett prolblem är. Normen för en matris är enkel att beräkna om man har den på SVD-form. Matriserna U och V i (16) är unitära och påverkar inte normen, vilket ger

$$\left.\begin{aligned} ||\Phi||_2 = ||\Sigma||_2 = \sigma_{max} \\ ||\Phi^{-1}||_2 = ||\Sigma^{-1}||_2 = 1/\sigma_{min} \end{aligned}\right\} \quad \Rightarrow \quad \kappa[\Phi] = \sigma_{max}/\sigma_{min}$$

I vårt fall får vi

$$\kappa[\Phi] \sim 10^9$$

vilket är ett stort konditionstal, och problemet är alltså *inte* välkonditionerat.

Nu är det dock inte problemet allena som bestämmer hur bra en lösning blir, algoritmen har också stor betydelse för hur korrekt en lösning kan anses vara. För normalekvationen, som använder sig av $\Phi^T\Phi$, blir konditionstalet för algoritmen

$$\kappa[\Phi^T\Phi] = {\sigma_{max}}^2/{\sigma_{min}}^2 \ (\sim 10^{19})$$

medan den för kvadratrotsalgoritmerna är i samma storleksordning som konditionstalet för problemet, dvs $\kappa[\Phi]$.

Man kan då förundras över hur bra man lyckas skatta parametrarna med olika kvadratrotsmetoder. Övre gränsen för relativa felet hos utdata kan beräknas som

$$\frac{\|\delta_\theta\|}{\|\theta + \delta_\theta\|} \leq \kappa_P \left(\frac{\|\delta_\Phi\|}{\|\Phi\|} + \kappa_A u \right)$$

där u är avkortningsenheten hos datorn (i **matlab** räknas det med 15 siffror), κ_P är konditionstalet för problemet och κ_A är konditionstalet för algoritmen. Om relativa felet i indata antas vara $\sim 10^{-9}$ finner man att för kvadratrotsalgoritmerna kommer κ_P att ha störst betydelse medan κ_A inverkar mest vid beräkningar med normalekvationen.

Hur förklarar man resultaten för Gram-Schmidt, SVD med full rang och SVD med $r = 1$? Gram-Schmidt och SVD löser i princip normalekvationen, men faktoriseringen gör att vi löser ett mer välkonditionerat problem än då man löser normalekvationen direkt. Φ:s rang är, vilket tidigare nämnts, snarare $r = 1$ än $r = 4$. Då man löser minsta-kvadrat-problemet med SVD och antagandet att $r = 1$ kommer det största singulärvärdet inte att undertryckas vid invertering av Σ_r och man kommer därmed att erhålla den bästa tänkbara lösningen.

2.6.4 Bandpassfiltrering av indata

I problemet ovan var avvikelsen från indatas medelvärden väldigt liten. Om avvikelsen blir stor kommer problemets konditionstal att bli κ^2 i stället för κ och varför algoritmens konditionstal inte kommer att spela en lika avgörande roll; kvadratrotsmetoderna och normalekvationen kommer att ge ungefär lika bra resultat.

För att verifiera detta försöker vi filtrera indata innan vi beräknar parametrarna, så att endast variationen i sista siffran kvarstår. Detta kan vi uppnå genom att *bandpssfitrera* eller *differentiera* indata. Vi får då

$$\Phi_f(10) = \begin{pmatrix} 1 & 0 & -1 & 1 \\ -1 & 1 & 0 & -1 \\ -1 & -1 & 1 & 0 \\ 0 & -1 & -1 & 1 \\ 2 & 0 & -1 & -1 \\ -2 & 2 & 0 & -1 \end{pmatrix}$$

$$Y_f(10) = [\ 1 \quad -1 \quad -1 \quad -1 \quad 0 \quad -1\ ]^T$$

Problemet är nu mer välkonditionerat ($\kappa[\Phi_f] \approx 4.45$), och vi finner att samtliga metoder lyckas skatta parametrarna bra. Felnormen blir noll för kvadratrotsmetoderna medan den för SVD, Gram-Schmidt och Normalekvationen blir $\sim 10^{-6}$.

Φ_f:s rang *är* nu verkligen fyra, varför man inte behöver försöka sig på SVD med något annat antagande om rangen (de ger dessutom helt fel värden). Singulärvärdena för Φ_f är

$$\begin{aligned}\sigma_1 &= 3.99719\\ \sigma_2 &= 2.68623\\ \sigma_3 &= 1.73205\\ \sigma_4 &= 0.89814\end{aligned}$$

vilket kan jämföras med singulärvärdena för den ofiltrerade Φ-matrisen.

3 Rekursiv estimation

Som tidigare nämnts arbetar en adaptiv regulator i realtid. Detta får till följd att man vid parameterskattning i de adaptiva regulatorerna endast är intresserade av rekursiva estimeringsalgoritmer. I detta avsnitt kommer jag att undersöka hur Dyadisk transformation kan användas i en rekursiv skattningsalgoritm och jämföra denna med den i [1] så flitigt använda algoritmen (2).

3.1 Rekursiv estimation ur ett sannolikhetsteoretiskt perspektiv

Problemet vi vill lösa är följande:

- Givet den linjära observationen

$$y = \varphi^T\theta + e \qquad \theta \in N(\theta^0, P),\ e \in N(0, \sigma^2) \tag{22}$$

- Skatta θ med hjälp av

$$\hat{\theta} = \theta^0 + K(y - \varphi^T\theta) \tag{23}$$

Detta ger följande uttryck för kovariansmatrisen för y och θ

$$R = \begin{pmatrix} R_y & R_{y\theta} \\ R_{\theta y} & R_\theta \end{pmatrix} = \begin{pmatrix} \varphi^T LDL^T\varphi + \sigma^2 & \varphi^T LDL^T \\ LDL^T\varphi & LDL^T \end{pmatrix} =$$

$$= \begin{pmatrix} 1 & \varphi^T L \\ 0 & L \end{pmatrix} \begin{pmatrix} \sigma^2 & 0 \\ 0 & D \end{pmatrix} \begin{pmatrix} 1 & 0 \\ L^T\varphi & L^T \end{pmatrix} \tag{24}$$

där

$$P = LDL^T$$

och L är en lågtriangulär matris med ettor på diagonalen och D är en diagonalmatris med icke-negativa diagonalelement.

I [1] visas det att R i (24) kan transformeras till formen

$$R = \begin{pmatrix} \tilde{\sigma}^2 & \tilde{\sigma}^2 K^T \\ \tilde{\sigma}^2 K & \tilde{\sigma}^2 KK^T + \tilde{L}\tilde{D}\tilde{L}^T \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ K & \tilde{L} \end{pmatrix} \begin{pmatrix} \tilde{\sigma^2} & 0 \\ 0 & \tilde{D} \end{pmatrix} \begin{pmatrix} 1 & K^T \\ 0 & \tilde{L}^T \end{pmatrix} \tag{25}$$

där K motsvarar K:et i ekvation (23).

Vad man gör efter att ha LDL-faktoriserat P är alltså, att uttrycka kovariansmatrisen R på formen (24), transformera den till formen (25) och därefter uppdatera parametrarna enligt ekvation (23). Därpå börjar man om på nytt med

$$P = \tilde{L}\tilde{D}\tilde{L}^T$$

Den svåra biten är att göra transformationen från (24) till (25), och det är här den i detta projekt så omtalade Dyadiska transformationen kommer in...

3.2 Rekursiv estimation med Dyadisk transformation

I [1] återfinns två **modula-2**-procedurer: en för Dyadisk transformation kallad *Dyadic-Reduction* och en för konstruktion av matriserna

$$\begin{pmatrix} 1 & \varphi^T L \\ 0 & L \end{pmatrix} \quad \text{och} \quad \begin{pmatrix} \sigma^2 & 0 \\ 0 & D \end{pmatrix}$$

utgående från vektorerna θ, D, φ, matrisen L och skalären σ^2 samt transformationen från (24) till (25) kallad *LDFilter*. Ett närmare studium av dessa uppdagar att uppdateringsformeln för P, K och $\hat{\theta}$ är densamma som i (2) med $\lambda = \sigma^2$ och $P = LDL^T$. Vidare finner man vad gäller den Dyadiska transformationen att

$$\begin{cases} \alpha_n a_n a_n^T + \beta_n b_n b_n^T = \tilde{\alpha}_n \tilde{a}_n \tilde{a}_n^T + \tilde{\beta}_n \tilde{b}_n \tilde{b}_n^T \\ \alpha_{n-1} a_{n-1} a_{n-1}^T + \beta_{n-1} b_{n-1} b_{n-1}^T = \tilde{\alpha}_{n-1} \tilde{a}_{n-1} \tilde{a}_{n-1}^T + \tilde{\beta}_{n-1} \tilde{b}_{n-1} \tilde{b}_{n-1}^T \\ \dots \\ \alpha_1 a_1 a_1^T + \beta_1 b_1 b_1^T = \tilde{\alpha}_1 \tilde{a}_1 \tilde{a}_1^T + \tilde{\beta}_1 \tilde{b}_1 \tilde{b}_1^T \\ \\ \alpha_k = \tilde{\alpha}_{k+1} \\ a_k = \tilde{a}_{k+1} \end{cases}$$

vilket ger

$$\alpha_n a_n a_n^T + \sum_{k=1}^{n} \beta_k b_k b_k^T = \tilde{\alpha}_1 \tilde{a}_1 \tilde{a}_1^T + \sum_{k=1}^{n} \tilde{\beta}_k \tilde{b}_k \tilde{b}_k^T \tag{26}$$

vilket i sin tur motsvarar

$$\begin{pmatrix} \sigma^2 & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} \varphi^T LDL^T \varphi & \varphi^T LDL^T \\ LDL^T \varphi & LDL^T \end{pmatrix} = \tilde{\sigma}^2 \begin{pmatrix} 1 & K^T \\ K & KK^T \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & \tilde{L}\tilde{D}\tilde{L}^T \end{pmatrix} \tag{27}$$

Nollorna i den sista matrisen sätts inte explicit i procedurerna, men det är ju detta som transformationen går ut på, dvs att nollställa första elementet i vektorn $\tilde{b}$.

Identifikation av matriserna i (26) och (27) ger att

$$\begin{cases} \alpha_n = \lambda = \sigma^2 \\ a_n = [\ 1 \quad 0 \quad \dots \quad 0\ ]^T \\ \beta_k = d_k \\ b_k = [\ \varphi_k{}^T L_k \quad L_k\ ]^T \\ \\ \tilde{\alpha}_1 = \tilde{\sigma}^2 \\ \tilde{a}_1 = [\ 1 \quad K\ ]^T \\ \tilde{\beta}_k = \tilde{d}_k \\ \tilde{b}_k = [\ 0 \quad \tilde{L}_k\ ]^T \end{cases}$$

där L_k, osv... betyder k:te kolonnen i matrisen L, osv...

Man måste påstå att matriserna i (24) och (25) är som gjorda för Dyadisk transformation, enär

- Matriserna är symmetriska
- Det är bara första elementen i kolonnvektorerna som behöver nollas ut, vilket gör problemet enklare och naturligt ger vektorerna b
- Matrisen

$$\begin{pmatrix} \sigma^2 & 0 \\ 0 & D \end{pmatrix}$$

  ger naturligt skalärerna α och β
- Den första kolonnvektorn ger naturligt vektorn a

3.3 Testning av skattningsalgoritmer

Jag provkörde slutligen Dyadisk Transformations-algoritmen med hjälp av **Matlab** på följande process

$$y(t) = -a_1 y(t-1) - a_2 y(t-2) + b_1 u(t-1) + b_2 u(t-2)$$

och jämförde resultaten med resultaten från motsvarande simulering med RLS-algoritmen. Processen är väldigt snäll, vilket får till följd att man inte kan se någon skillnad mellan simuleringarna.

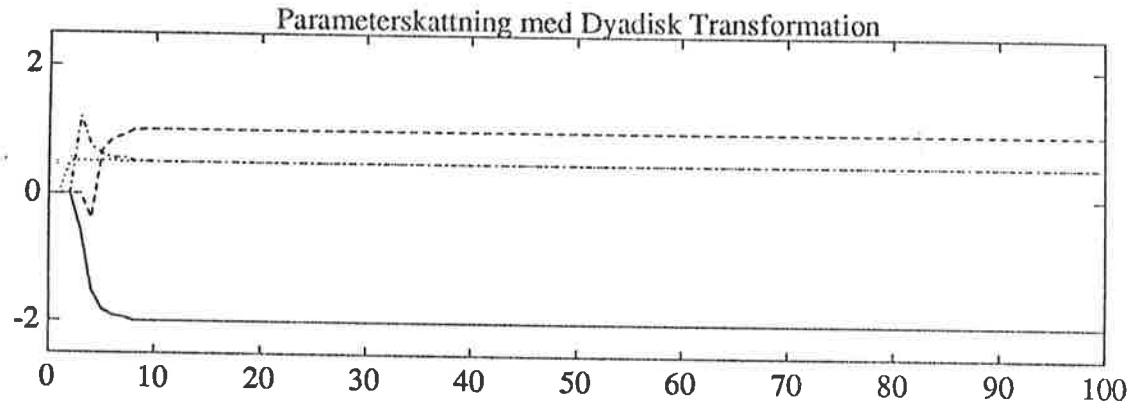
Plottar och programlistor finns återgivna i Appendix.

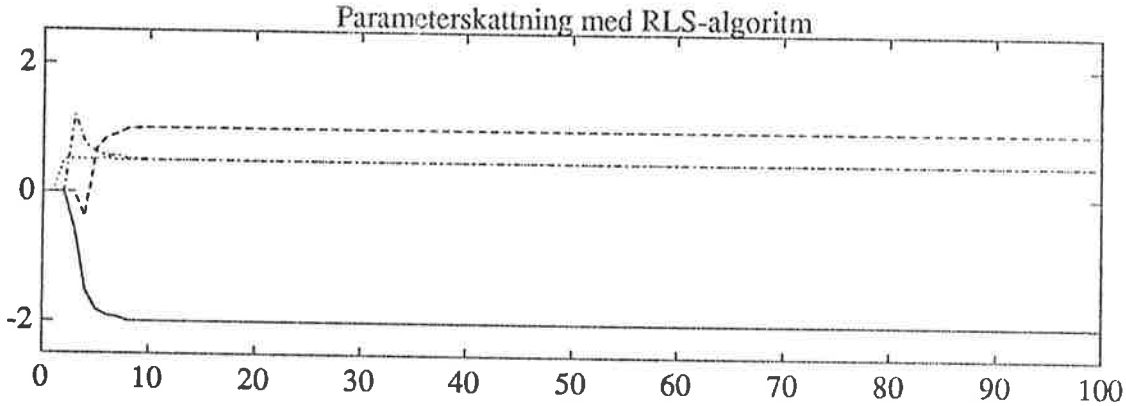
Referenser

[1] Karl Johan Åström & Björn Wittenmark *Adaptive Control* Addison-Wesley 1989.

[2] Bengt Lindberg *Kompendium i tillämpad numerisk analys för FK II vid KTH, version nr 6* 1989.

Appendix





a1 = <—>, a2 = <– –>, b1 = <···> och b2 = <–·–>

```
%macro Main.m
%
%Skattar parametrarna till systemet Sys.m med Dyadisk Transformations-algoritm
%

clc
hold off;
clg;
subplot(211);
title('Parameterskattning med Dyadisk Transformation')
axis([0, 100, -2.5, 2.5]);
plot(0, 0, 'i');
pause(0.1);
hold on;

t = 0;
h = 1;
THETA = [];
T = [];
T1 = [];

[uc, y, x, u, v, theta, L, D, phi, Phi, Gamma] = Init(h);

while t < 100,
  t = t + h;
  [uc, y, u, phi, x] = UpDate(uc, y, u, phi, x, Phi, Gamma, t);
  [u, v] = Regulator(y, uc, u, v, theta, h);
  [theta, L, D] = LDFilter(theta, y(1), phi, L, D);
  THETA = [THETA ; theta'];
  T = [T ; t];
  if t==5|t==10|t==20|t==40,
    T1 = [T1 ; theta'];
  end
end

subplot(211);
axis([0, 100, -2.5, 2.5]);
plot(T, THETA(:,1), '-');
plot(T, THETA(:,2), '--');
plot(T, THETA(:,3), ':');
plot(T, THETA(:,4), '-.');

L
D
theta
Main2

*************************************************************************

function [uc, y, x, u, v, theta, L, D, phi, Phi, Gamma] = Init(h)

uc = 0;
y = [0 ; 0 ; 0];
u = [0 ; 0];
x = [0 ; 0];
v = 0;
phi = [-y([2,3]) ; u];

theta = [0 0 0.01 0.01]';

A = [0 1 ; 0 0];
B = [0 1]';
[Phi, Gamma] = c2d(A, B, h);

L = eye(length(phi));
D = 100*ones(length(phi), 1);
```

```
function [theta, L, D] = LDFilter(theta, y, phi, L, D)

n = 4;
lambda = 1;

alpha = lambda;
e = y - phi'*theta;
w = phi'*L;
K = zeros(n,1);
for i = n : -1 : 1,
  [K, L(:,i), alpha, D(i)] = DyadTransf(K, L(:,i), alpha, D(i), w(i));
end

theta = theta + K*e;
D = D/lambda;

*************************************************************************

function [a, b, alpha, beta] = DyadTransf(a, b, alpha, beta, b1)

mzero = 1e-10;

if beta < mzero,
  beta = 0;
end
w1 = alpha;
w2 = beta*b1;
alpha = alpha + w2*b1;
if alpha > mzero,
  beta = w1*beta/alpha;
  gam = w2/alpha;
  b = b - b1*a;
  a = a + gam*b;
end
```

```
%macro Main2.m
%
%Skattar parametrarna till systemet Sys.m med RLS-algoritm med glomskefaktor
%

hold off;
subplot(212);
title('Parameterskattning med RLS-algoritm')
axis([0, 100, -2.5, 2.5]);
plot(0, 0, 'i');
pause(0.1);
hold on;

t = 0;
h = 1;
THETA = [];
T = [];
T2 = [];

[uc, y, x, u, v, theta, P, phi, Phi, Gamma] = Init2(h);

while t < 100,
  t = t + h;
  [uc, y, u, phi, x] = UpDate(uc, y, u, phi, x, Phi, Gamma, t);
  [u, v] = Regulator(y, uc, u, v, theta, h);
  [theta, P] = estim(theta, y(1), phi, P);
  THETA = [THETA ; theta'];
  T = [T ; t];
  if t==5|t==10|t==20|t==40,
    T2 = [T2 ; theta'];
  end
end

subplot(212);
axis([0, 100, -2.5, 2.5]);
plot(T, THETA(:,1), '-');
plot(T, THETA(:,2), '--');
plot(T, THETA(:,3), ':');
plot(T, THETA(:,4), '-.');
xlabel('a1 = <_>, a2 = <__>, b1 = <..> och b2 = <_.>')

P
theta

*************************************************************************

function [uc, y, x, u, v, theta, P, phi, Phi, Gamma] = Init2(h)

uc = 0;
y = [0 ; 0 ; 0];
u = [0 ; 0];
x = [0 ; 0];
v = 0;
phi = [-y([2,3]) ; u];

theta = [0 0 0.01 0.01]';

A = [0 1 ; 0 0];
B = [0 1]';
[Phi, Gamma] = c2d(A, B, h);

P = 100*eye(length(phi));
```

```
function [u, v] = Regulator(y, uc, u, v, theta, h)
%
%RST-Regulator med anti-wind-up och forkortning av nollstallen
%

a1 = theta(1);
a2 = theta(2);
b1 = theta(3);
b2 = theta(4);

z = 1;
w = 0.7;
a = exp(-z*w*h);
am1 = -2*a*cos(w*h*sqrt(1-z^2));
am2 =  a^2;
ulim = 1;

as = 1 + am1 +am2;

t0 = as/b1;
r1 = b2/b1;
s0 = (am1-a1)/b1;
s1 = (am2-a2)/b1;

u(2) = u(1);
v = t0*uc - s0*y(1) - s1*y(2) - r1*u(2);

if v < -ulim,
  u(1) = -ulim;
elseif v > ulim,
  u(1) = ulim;
else
  u(1) = v;
end
```

```
function [uc] = Refgen(t)

step = 1;
per = 50;

if t < per/2 | (t > per & t < 3*per/2),
  uc = step;
else
  uc = -step;
end

*************************************************************************

function [y, x] = Sys(u, x, Phi, Gamma)

C = [1 0];

x = Phi*x + Gamma*u(1);
y = C*x;

*************************************************************************

function [uc, y, u, phi, x] = UpDate(uc, y, u, phi, x, Phi, Gamma, t)

uc = Refgen(t);
y(3) = y(2);
y(2) = y(1);
[y(1), x] = Sys(u, x, Phi, Gamma);
phi = [-y([2,3]) ; u];
```

SIMULERING

i LabVIEW 2 av
tillståndsåterkopplat dc-servo

ett projekt i Adaptiv reglering av

Anders Ask
Lars Sjöberg

22 mars 1991

Innehållsförteckning

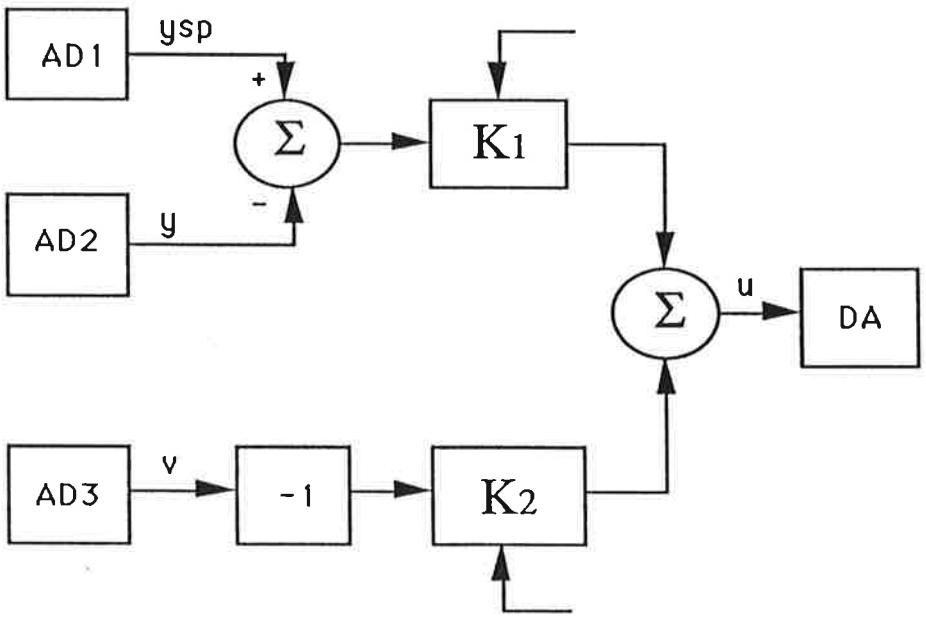
1 Inledning

Denna rapport är en dokumentering av ett projekt i kursen Adaptiv Reglering. Projektets omfattning har varit ca 40 timmar. Målet för projektet var högt ställt och inom den givna tidsramen har vi inte lyckats genomföra alla steg. De resultat vi kommit fram till redovisas här tillsammans med synpunkter och kommentarer. Rapporten är upplagd så att vi i kapitel 2 beskriver det problem som gavs till oss av kursansvarige Karl Johan Åström. Problemet beskrivs här i sin helhet samtidigt som en avgränsning görs av det som vi hunnit med. Kapitel 3 beskriver det vi gjort och torde utgöra tillräckligt med information för att kunna ge en oinvigd möjlighet att köra vårt simuleringsaprogram SIMU på institutionens Macintosh datorer. Vi har använt programpaketet LabVIEW 2 för att realisera vårt simuleringsprogram. Eftersom ingen av oss hade någon erfarenhet av LabVIEW 2 har vi i kapitel 4 sammanställt några kommentarer och synpunkter på detta programpaket.

2 Problemformulering

I programpaketet LabVIEW 2 skall vi för ett tillståndsåterkopplat dc-servo bygga ett MRAS m h a Lyapunovteori[1].För att få ett snabbt program så att vi skall kunna simulera med en analogmaskin kommer vi att skicka ut programmet till processorkortet DSP2300. A/D- och D/A-omvandling till datorn sköts med hjälp av korten MIO16 respektive AO6.

Den styrlag som vi skall använda är $u = K_1(ysp - y) - K_2v$.



Figur 2.1 Regulatorstruktur

Vi insåg ganska snart att projektets storlek var för omfattande.Vi har därför begränsat oss till tillståndsåterkoppling med kända parametrar[2] samt att all simulering sker inne i datorn.

[1] se appendix A

[2] se appendix B

3 Implementering av simuleringsprogrammet för tillståndsåterkopplat dc-servo

3.1 Implementeringsspråket G

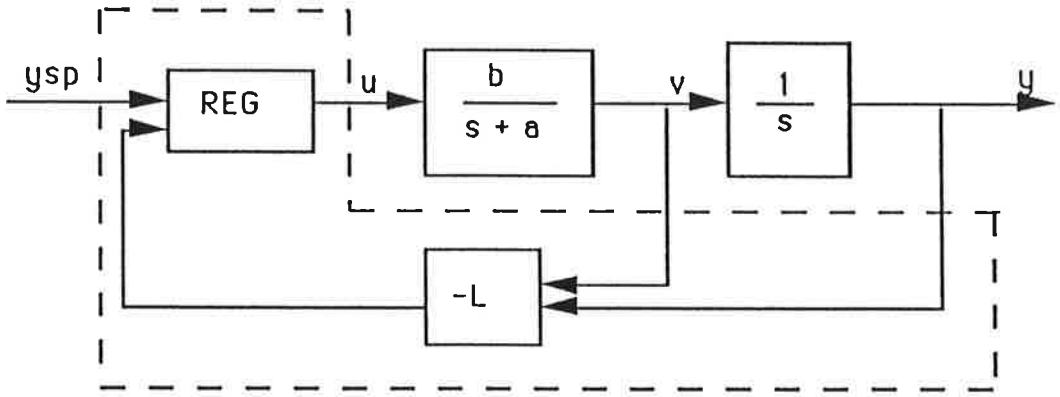
Vi har implementerat vårt simuleringsprogram i ett program som heter LabVIEW 2. I det här programmet används ett grafiskt programmeringsspråk som kallas G. För en utförlig beskrivning av detta språk hänvisar vi till de manualer som medföljer detta programpaket. I denna rapport kommer vi endast att ta upp de instruktioner som vi använt för att realisera vårt simuleringsprogram. Först lite allmänna förklaringar:

- Det fönster i vilket vi programmerar kallas **Block diagram** och den grafiska programkod som vi genererar i detta fönster kan sägas motsvara en procedur i t ex PASCAL.
- Kommunikationen med proceduren, dvs sättandet av inparametrar och visningen av utparametrar, sker via **Front Panel.** Detta kan liknas vid en operatörspanel och kan också byggas upp grafiskt som en sådan.
- Procedurens namn definieras i en s k **Icon.** Ikonen är ett block som sedan kan användas som vilken annan grafisk instruktion som helst. Det enda vi behöver göra är att editera ikonen, dvs ge den ett namn samt tala om hur många in- resp utparametrar blocket ska ha och tala om var någonstans de ska anslutas grafiskt.

Många reglertekniska problem visualiseras med hjälp av blockscheman. Med det grafiska programmeringsförfarandet blir därför överföringen av problemet i blockschemaform till programkod enkel. Vi bygger helt enkelt upp de ingående blocken och sammanbinder dessa grafiskt.

3.2 Simuleringsprogrammet SIMU

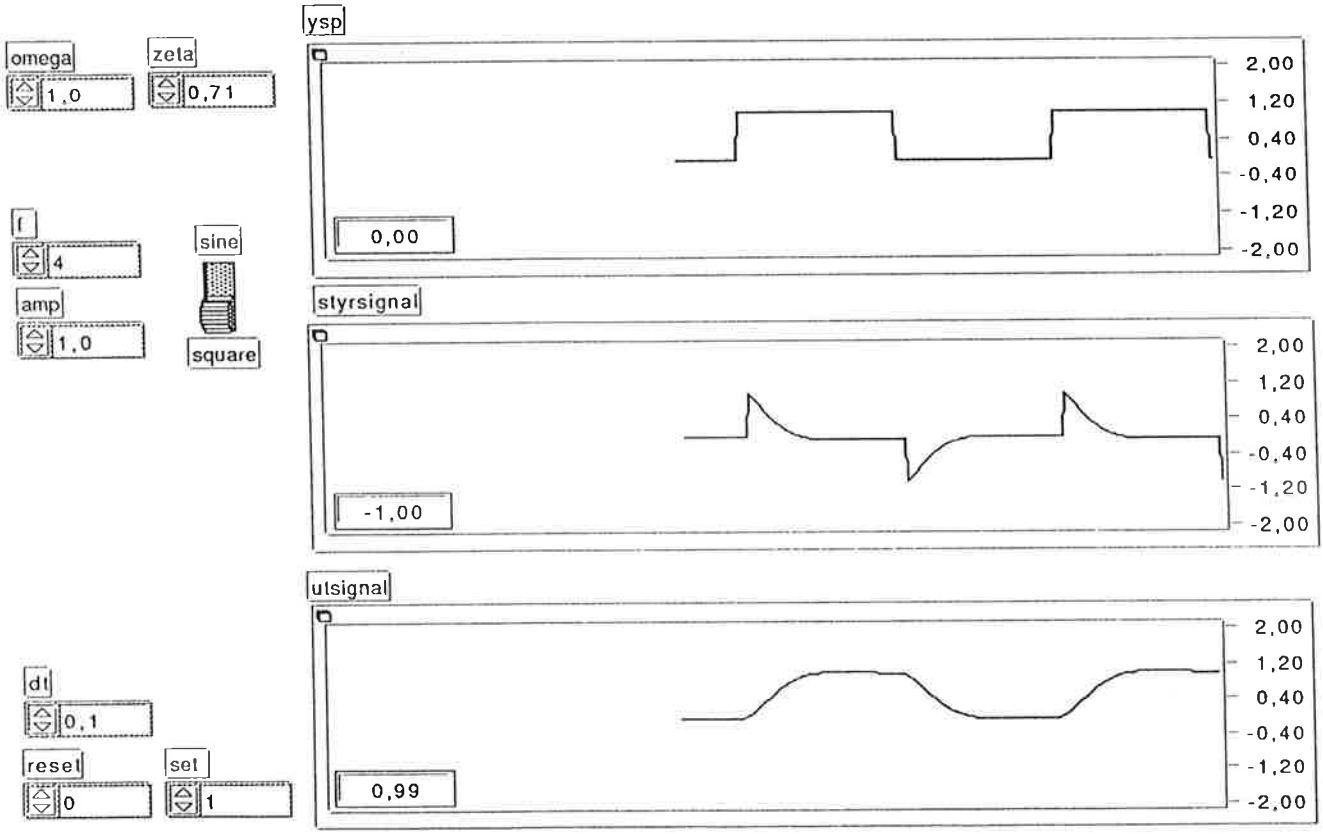
Ett förenklat blockschema för ett tillståndsåterkopplat dc-servo kan se ut enl figur 3.1.



figur 3.1 Blockschema

Låt allt inom den streckade linjen tillhöra regulatorn. Det innebär att vi betraktar regulatorn som ett block med tre insignaler och en utsignal.

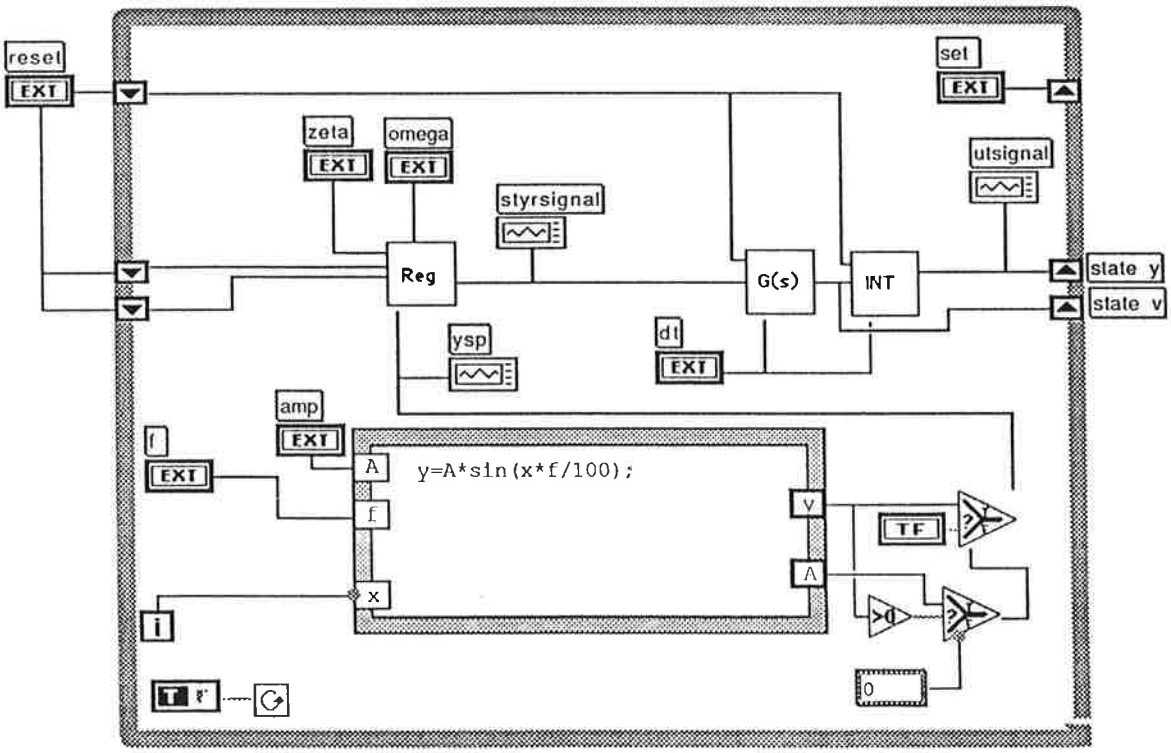
Simuleringsprogrammet styrs från SIMU:s **Front Panel**.



figur 3.2 SIMU:s **Front Panel**

På panelen visas kommunikationen med simuleringsprogrammet. Styrparametrarna, eller inparametrarna till proceduren, kallas **Controls**. Utparametrarna kallas **Indicators**. Tre signaler visas samtidigt på panelen, nämligen ysp (=börvärdet), styrsignalen samt utsignalen. De tillhör naturligtvis alla klassen indicators. Med hjälp av omega och zeta ställer vi in önskat stegsvar, dvs önskad snabbhet och dämpning. Börvärdet kan vara antingen en sinussignal eller en fyrkantvåg vars frekvens och amplitud ställs in med hjälp av f resp amp. Vidare kan det tilläggas att inga begränsningar av styrsignalens amplitud har lagts in.

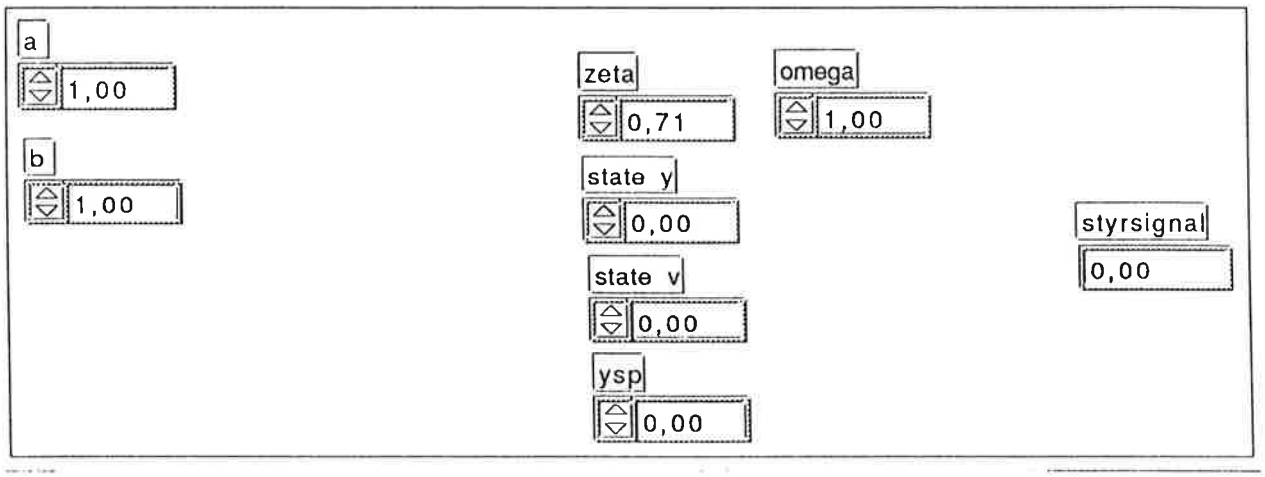
Själva programmeringen av systemet görs i SIMU:s **Block Diagram**, se figur 3.3. Jämför vi detta med blockschemat i figur 3.1 ser vi att det finns stora likheter. Det enda som egentligen skiljer dem åt är att vi inte kan sluta återkopplingen direkt utan måste använda ett skiftregister för vardera tillstånd. Skiftregistren är markerade med svarta trekanter på whileloopens kant. Koden i den nedre halvan av whileloopen är till för att generera sinus- och fyrkantvåg.Vidare finner vi de av oss definierade blocken REG, G(s) och INT. Set, Reset och dt förklaras i kapitel 3.5.



figur 3.3 SIMU:s **Block Diagram**

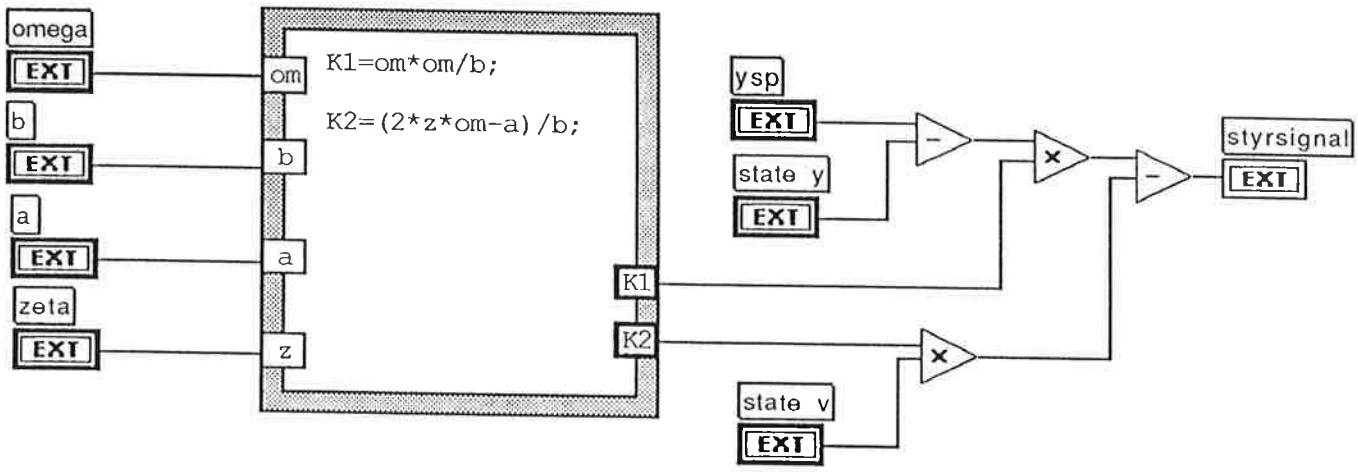
3.3 Regulatorn REG

I regulatorn är styrlagen $u = K_1(ysp - y) - K_2v$ förverkligad. Servots parametrar a och b ställs i Reg:s **Front Panel,** se figur 3.4. Här är även insignalerna samt utsignalen definierade.



figur 3.4 Reg:s **Front Panel**

Reg:s **Block Diagram** finns i figur 3.5. Här framträder ett av problemen med att programmera grafiskt, nämligen tydligheten. Vi hade kunnat lägga in alla beräkningarna inuti formelrutan, men frågan man ställer sig då är om det blivit tydligare. Någonstans får man dra en gräns när det gäller användningen av grafiska symboler.

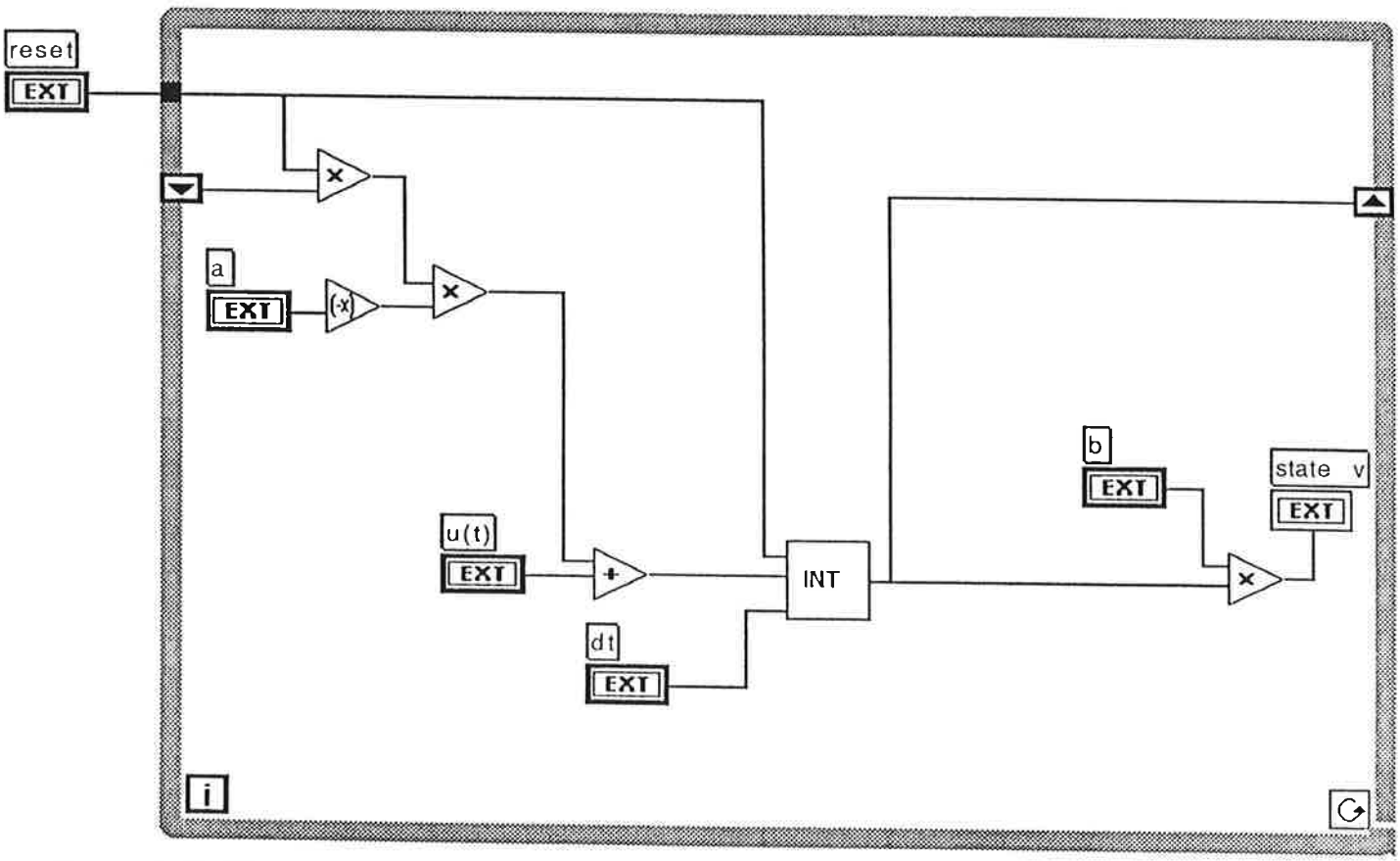


figur 3.5 Reg:s **Block Diagram**

3.4 Processen G(s)

På processens **Front Panel** sätter vi parametrarna a och b och i processens blockdiagram har vi realiserat systemet G(s) = b/(s + a), se figur 3.6 och 3.7 nedan.

figur 3.6 Processens **Front Panel**



figur 3.7 Processens **Block diagram**

Skriv systemet på styrbar kanonisk form.

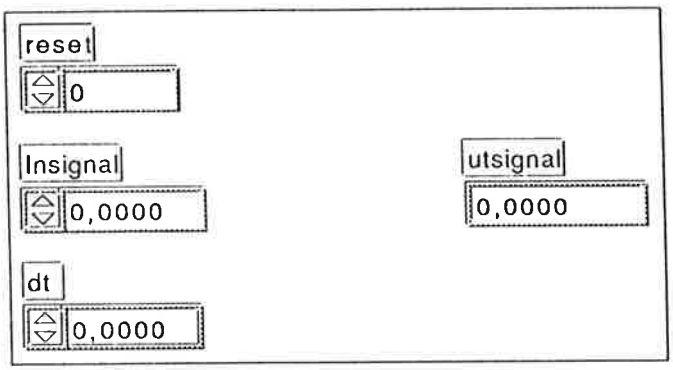
$$\frac{dx}{dt} = -ax + u$$
$$y = bx$$

Om vi tänker oss att vi realiserar denna differentialekvation med hjälp av analog teknik så behöver vi en integrator. Vårt angripssätt har varit att approximera en integrator med en summa. Programmeringsspråket i LabVIEW 2 tillåter inte slutna slingor och därför uppstår här ett problem. Vi blir tvungna att ta till ett skiftregister och skifta det återkopplade värdet istället för att sluta slingan direkt. Detta medför naturligtvis approximationer men vi antar att vi samplar så fort att de blir försummbara.

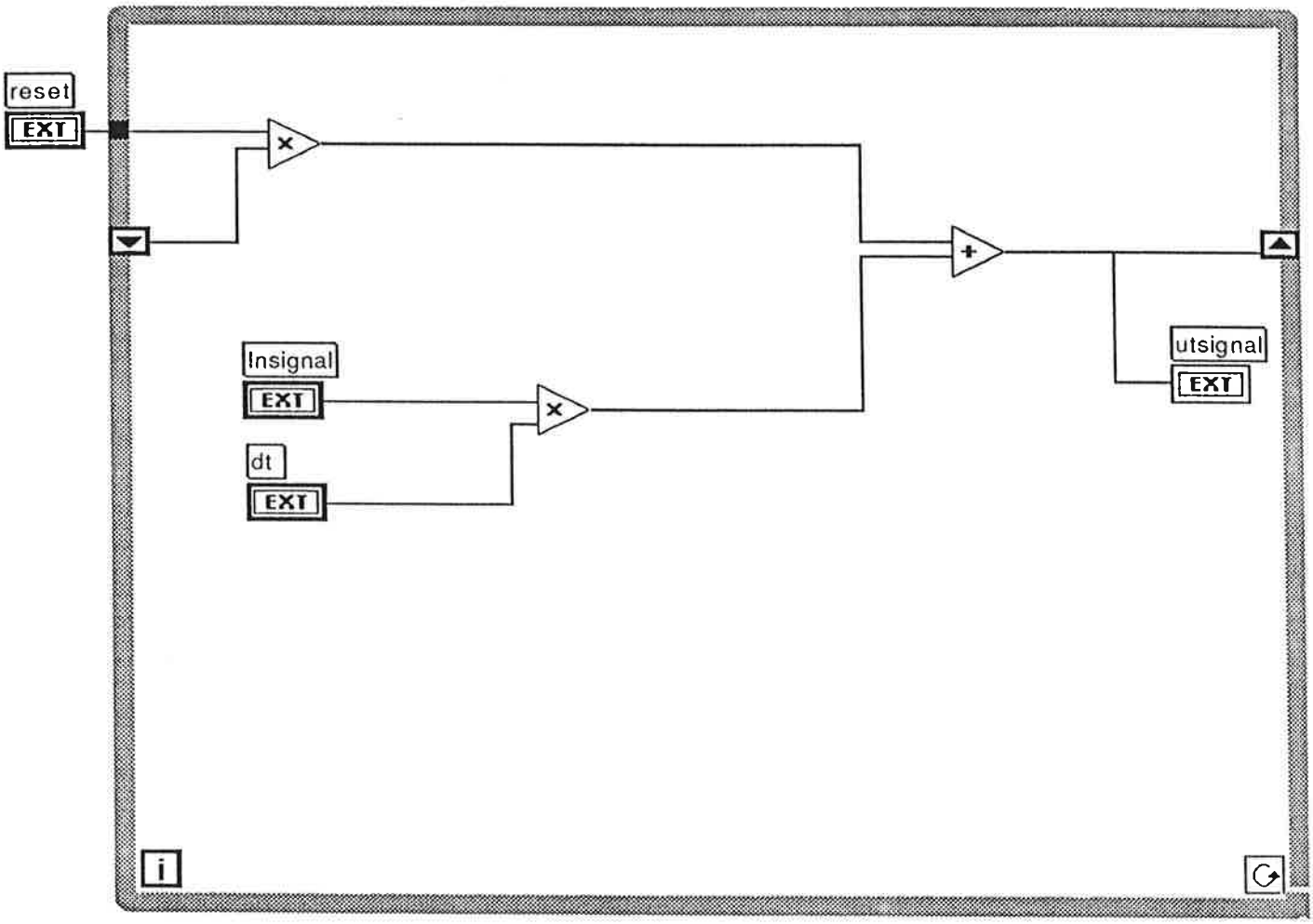
Vi är tvungna att göra en summaapproximation av integralen eftersom LabVIEW 2 inte innehåller någon on-line integration. Ett mera rättframt sätt torde dock vara att göra någon form av differensapproximation av överföringsfunktionen. Ett sådant angripssätt vore särskilt lämpat p g a LabVIEW 2:s skiftregisterfunktioner.

3.5 Integralen INT

INT är en summaapproximation av en integral. Det är här vi använder signalerna dt, set och reset. Dt är sampelinkrementet. Vi borde ha bättre kontroll på detta genom att sätta in någon slags timerfunktion som hela tiden ligger och mäter tiden mellan två sampel. Hur som helst, om vi antar att vi har konstant intervall mellan samplen då kommer dt bara in som en extra förstärkning framför intgralen. Denna extra förstärkning regleras emellertid ut eftersom vi ansatt att vårt slutna systems statiska förstärkning ska vara ett. Bäst vore naturligtvis att mäta men det problemet har vi inte haft tid att lösa. Reset är en signal som är noll första gången simuleringsprogrammets whileloop löps igenom. Därefter antar den samma värde som set, nämligen ett. Reset multipliceras varje gång med integralen. På detta sätt nollställer vi integralen varje gång vi vill göra en ny simulering. Om vi avslutat en simulering och vill fortsätta simulera där den förra slutade behöver vi således bara sätta reset till ett och exekvera igen.



figur 3.8 INT:s **Front Panel**



figur 3.9 INT:s **Block Diagram**

4 Kommentar till LabVIEW 2

Under den veckan som vi jobbade med LabVIEW 2 har vi stött på en del problem. Det första och kanske vanligaste problemet när man ska börja jobba med ett programpaket av den här storleken är att det tar ganska lång tid att komma in i systemet.

När det gäller programmeringstekniken kan vi säga att det grafiska språket gör att programmet blir tydligt, lättläst och någorlunda lätt att använda. Det bör dock påpekas att formler bör räknas ut i de formelrutor som LabVIEW 2 innehåller. Räknas formlerna ut med hjälp av grafiska symboler blir programmet svårläst och tydligheten går förlorad.

Ett annat problem vi hade var att vi tänkte "analogt" då vi realiserade våra överföringsfunktioner. Vi tycker att man bör realisera överföringsfunktioner diskret, d v s arbeta direkt med samplade modeller. På det viset undviker man onödiga approximationer. Det blir också lättare att implementera eftersom LabVIEW 2 innehåller skiftregisterfunktioner som är idealiska att använda när man arbetar med diskreta modeller.

Ett ständigt återkommande och irriterande fenomen var att kablarna som sammanbinder de grafiska symbolerna var svåra att dra. Efter nästan varje ny tråddragning var vi tvungna att gå upp i menyn för att eliminera lösa trådstumpar som kommit dit till synes utan orsak.

Avslutningsvis vill vi tillägga att det har varit kul att jobba med LabVIEW 2.

Appendix A

MRAS m h a Lyapunovteori för tillståndsåterkoppling

Modell :

$$G_m(s) = \frac{\omega^2}{s^2 + 1.5\omega s + \omega^2}$$

$$\frac{dx_m}{dt} = \begin{bmatrix} -1.5\,\omega & \omega^2 \\ 1 & 0 \end{bmatrix} x_m + \begin{bmatrix} \omega^2 \\ 0 \end{bmatrix} u_c = A_m x_m + B_m u_c$$

$$y_m = [\,0 \;\; 1\,]\, x_m$$

System:

$$G(s) = \frac{b}{s(s+a)}$$

$$\frac{dx}{dt} = \begin{bmatrix} -a & 0 \\ 1 & 0 \end{bmatrix} x + \begin{bmatrix} b \\ 0 \end{bmatrix} u$$

$$y = [\,0 \;\; 1\,]\, x$$

Styrlag:

$$u = K_1 ysp - K_1 x_2 - K_2 x_1 = \theta_3 ysp - \theta_2 x_2 - \theta_1 x_1$$

Slutna systemet:

$$\frac{dx}{dt} = \begin{bmatrix} -a & -b\theta_1 & -b\theta_2 \\ 1 & 0 \end{bmatrix} x + \begin{bmatrix} -b\theta_3 \\ 0 \end{bmatrix} u_c = A(\theta)x + B(\theta)u_c$$

$$y = [\,0 \;\; 1\,]\, x$$

Vår Lyapunov funktion är

$$v = e^T P e + tr(A - A_m)^T(A - A_m) + tr(B - B_m)^T(B - B_m) \geq 0$$

$$\frac{dv}{dt} = e^T(A_m^T P + P A_m)e + 2tr((A - A_m)^T(\frac{dA}{dt} + Pex^T)) + 2tr((B - B_m)^T(\frac{dB}{dt} + Peu_c^T))$$

om $\frac{dA}{dt} = -Pex^T$, $\frac{dB}{dt} = -Peu_c^T$ $\Rightarrow$ $\frac{dv}{dt} = e^T Q e \leq 0$ där P och Q är positivt definita.

För att få tag i P och Q löses ekvationen $A_m^T P + P A_m = -Q$

$$P = \begin{bmatrix} 4 & 2 \\ 2 & 16 \end{bmatrix} \qquad Q = \begin{bmatrix} 41 & 11 \\ 11 & 16 \end{bmatrix}$$

$$\frac{dA(\theta)}{dt} = \begin{bmatrix} -\frac{d\theta_1}{dt} & -\frac{d\theta_2}{dt} \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} \\ p_{12} & p_{22} \end{bmatrix}\begin{bmatrix} e_1 \\ e_2 \end{bmatrix}[x_1 \; x_2]$$

$$\frac{dB(\theta)}{dt} = \begin{bmatrix} \frac{d\theta_3}{dt} \\ 0 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} \\ p_{12} & p_{22} \end{bmatrix}\begin{bmatrix} e_1 \\ e_2 \end{bmatrix} u_c$$

Parametrarna uppdateras enligt

$$\frac{d\theta_1}{dt} = (p_{11}e_1 + p_{12}e_2)\, x_1$$

$$\frac{d\theta_2}{dt} = (p_{11}e_1 + p_{12}e_2)\, x_2$$

$$\frac{d\theta_3}{dt} = -(p_{11}e_1 + p_{12}e_2)\, u_c$$

Appendix B

Tillståndsåterkoppling med kända parametrar

Modell:

$$G_m(s) = \frac{\omega^2}{s^2 + 1.5\omega s + \omega^2}$$

System:

$$G(s) = \frac{b}{s(s+a)}$$

$$\frac{dx}{dt} = \begin{bmatrix} -a & 0 \\ 1 & 0 \end{bmatrix} x + \begin{bmatrix} b \\ 0 \end{bmatrix} u$$

$$y = \begin{bmatrix} 0 & 1 \end{bmatrix} x$$

Styrlag:

$$u = K_1(ysp - y) - K_2 v$$

Slutna systemet:

$$\frac{dx}{dt} = \begin{bmatrix} -a-bK_2 & -bK_1 \\ 1 & 0 \end{bmatrix} x + \begin{bmatrix} bK_1 \\ 0 \end{bmatrix} u_c$$

$$y = \begin{bmatrix} 0 & 1 \end{bmatrix} x$$

$$G_c(s) = \frac{bK_1}{s^2 + (a + bK_2)s + bK_1}$$

Jämförelse mellan modell och det slutna systemet ger parametrarna

$$K_1 = \frac{\omega^2}{b}$$

$$K_2 = \frac{1.5\omega - a}{b}$$

Adaptiv reglering av Servomotor implementerad i LabView

Krister Tham, E-86

2 maj 1991

Innehåll

96

1 Inledning

I kursen Adaptiv Reglering ingår ett projektarbete som skall utföras under en veckas tid. Detta är en sammanställning över målsättning och genomförande samt slutsatser och övriga synpunkter på projektet.

2 Problemspecifikation

Målet med projektet är att med en enkel adaptiv regulator kunna styra en servomotor. Regulatorn skall vara av MRAS-typ, baserad på stabilitetsteori (Lyapunov). Regulatorprogrammet skall implementeras i programpaketet LabView.

Servomotorn har överförinsfunktionen $G(s) = \frac{b}{s(s+a)}$, där a och b är okända parametrar (bild 2.1). Utsignaler från servomotorn är vinkelhasighet (y_1) och position (y_2).

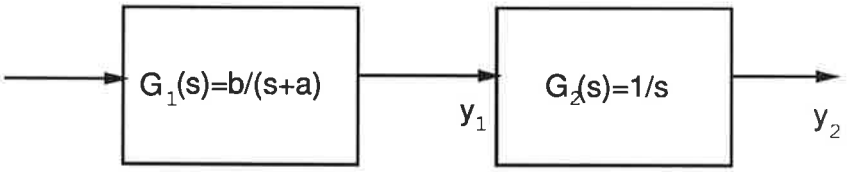


bild 2.1 Servomotorns överföringsfunktion.

3 Teoretisk lösning - översikt

Eftersom det i LabView finns en skiftregisterfunktion som lämpar sig utmärkt för implementering av samplade modeller, resonerade vi från början helt i diskreta termer (se appendix A). Detta visade sig mynna ut i ett komplicerat uttryck. För att få uttrycket på en mer hanterbar form, fick vi förutsätta att samplingsintervallet h var litet.

Eftersom vi ändå blev tvingade till denna approximation kunde vi lika gärna förutsätta ett litet h redan från början av vår modelluppställning. Vår slutgiltiga lösning fick vi alltså genom att betrakta systemet som tidskontinuerligt och sedan differentialapproximera derivatorna (se appendix B). Vi ansåg att detta var en rimlig medelväg mellan teori och verklighet.

Efter utprovning av regulatorn gjordes modifieringen enligt Appendix C för att minska regulatorns bruskänslighet.

4 Implementering i LabView

4.1 Kort beskrivning av LabView

Användargränssnittet i Labview är uppbyggt av två fönstertyper; "Front Panel" och "Block Diagram". "Front Panel" är själva operatörsfönstret. Där görs inställning av parametervärden och där visualiseras in- och utsignaler. I "Block

97

Diagram" implementeras själva programmet. Programmet består av olika figurer som var och en representerar en funktion. Figurerna knyts samman av ett nätverk av "signaltrådar".

Ett program kan byggas upp med hierakisk struktur med huvud-och underprogramstruktur. I huvudprogrammet representeras ett underprogram av en "ikon". Ikonen har formen av en liten kvadrat med "input/output"-anslutningar.

4.2 Huvudprogram

Huvudprogrammets "Front Panel" (bild 4.1) är uppbyggt av tre delar.

I den första delen bestäms önskad typ av referenssignal. Referenssignalen kan genereras som en fyrkantsvåg eller styras manuellt med markören. Ändrad signaltyp fås genom att klicka på switchknappens "skugga". Fyrkantsvågen har varierbar frekvens och amplitud.

I regulatordelen bestäms systemets önskade uppförande. Omega och zeta anger systemets önskade snabbhet respektive dämpning. Regulatorns samplingsintervall h, kan minimalt väljas till 1/60-dels sekund. Gamma står för regulatorparametrarnas förstärkningsfaktor. Till höger visas det momentana värdet av modellfelet $e = y_m - y_1$. Under exekvering kan t0 och s0 nollställas genom att klicka på "Reset t0/s0".

I den undre delen av "Front Panel" plottas referenssignalen, servomotorns styrsignal och position samt regulatorparametrarna t0 och s0.

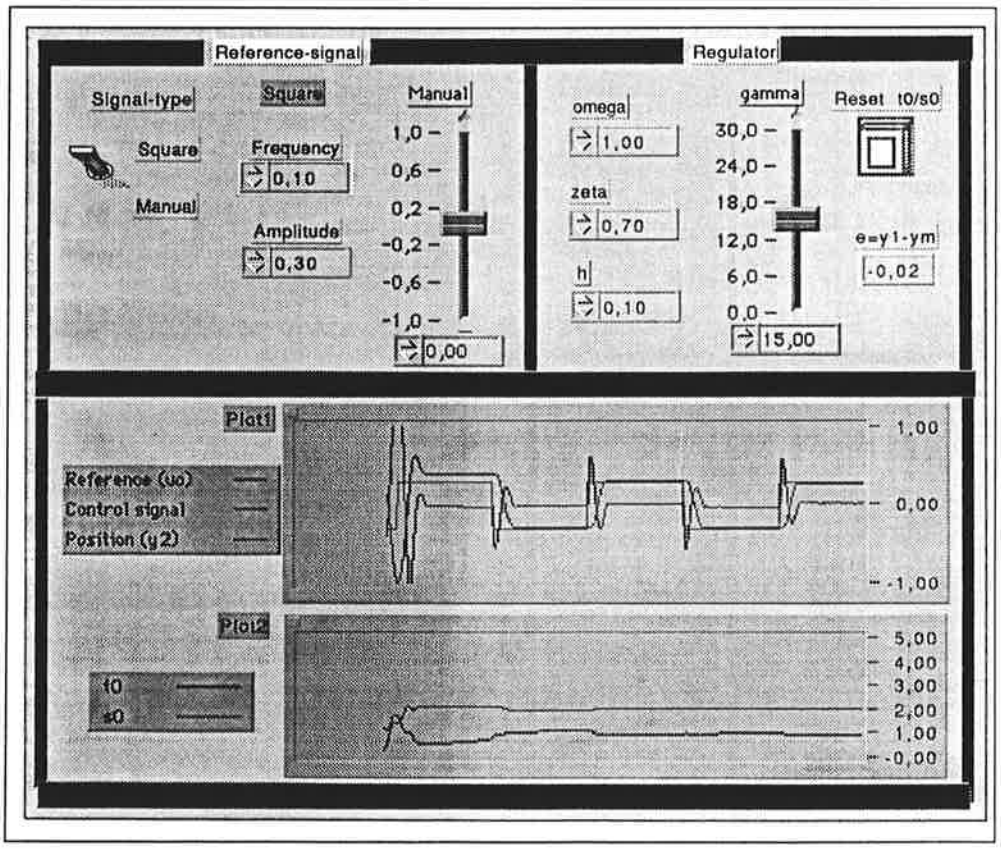


bild 4.1 Huvudprogrammets Front Panel.

Huvudprogrammets "Block Diagram" (bild 4.2) är uppbyggt kring en "formelnod". Det är i denna som själva regulatoralgoritmen implementeras. Från vänster kommer inkommande signaler (från "Front Panel" och AD-omvandlaren) och till höger går utgående signaler (till "Front Panel" och DA-omvandlaren).

För att implementera skiftregisterfunktionen lägger man en "while-loop" runt om formelnoden. Ett skiftregister symboliseras av två rektanglar, innehållande varsin triangel. Vid ett skift antar den vänstra rektangeln det värde som den högra senast antog. Högerrektangelns nya värde definieras i formelnoden. Utgående från Appendix B och C har vi definierat beteckningarna enligt:

$y_m(t+h) \equiv$ny, $y_m(t) \equiv$ym
$t_o(t+h) \equiv$nt, $t_o(t) \equiv$t0
$s_o(t+h) \equiv$ns, $s_o(t) \equiv$s0
$y_d(t+h) \equiv$nf, $y_d(t) \equiv$f1

Insignalen "re" används för att nollställa t0 och s0.

Skifthastigheten styrs av en "metronom-funktion". Denna befinner sig innanför "while-loopen". För att kunna variera skifthastigheten (samplingsfrekvensen) ges ett heltal (n0) som insignal till "metronomen". Metronomen gör $\frac{60}{n0}$ svängningar per sekund. Detta innebär att maximal samplingsfrekvens begränsas till 60 Hz. För att undvika avrundningsfel av samplingsperioden h använder vi h0= $\frac{n0}{60}$ vid beräkningar i regulatoralgoritmen.

I LabView måste varje variabel i formelnoden vara deklarerad som antingen in- eller utsignal. Detta innebär att även lokala variabler måste deklareras som utsignaler. Dessa "Unused variables" har placerats i det översta högra hörnet på formelnoden.

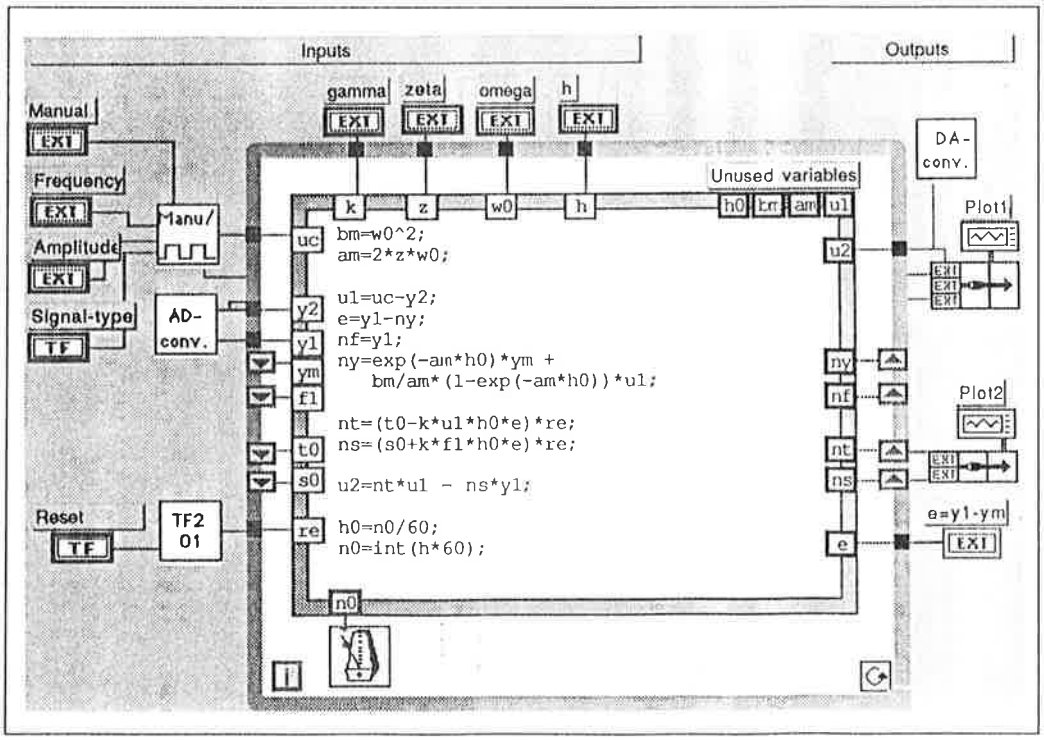


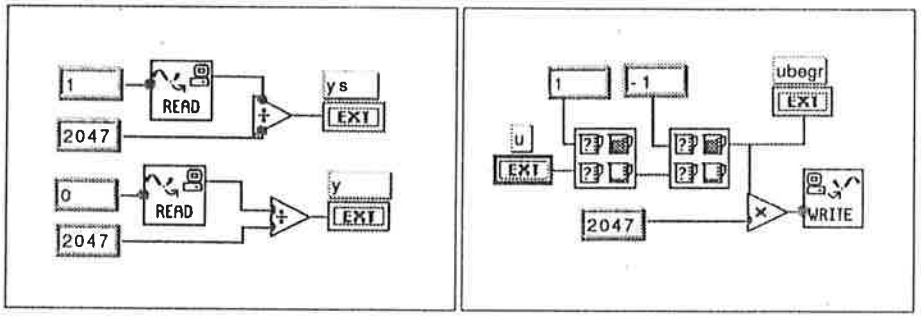
bild 4.2 Huvudprogrammets Block Diagram.

4.3 Ikonen AD-conv

I underprogrammet "AD-conv" (bild 4.3a) samplas de analoga insignalerna i "READ" på port 0 och 1. På interface-kortet motsvaras dessa utgångar av AI 0 resp AI 2. Det diskreta värdet normeras sedan med en faktor 2047.

4.4 Ikonen DA-conv

I underprogrammet "DA-conv" (bild 4.3b) begränsas först styrsignalen u till $|\ u\ | < 1$. Den begränsade signalen DA-omvandlas sedan i "WRITE". Styrsignalen kommer ut på utgång AO 0. Dessutom ges den begränsade styrsignalen som en utsignal från ikonen, för att kunna plottas i "Front Panel".



a b

bild 4.3 Ikonerna AD-conv:s och DA-conv:s Block Diagram.

4.5 Ikonen Manu/Square

Underprogrammet "Manu/Square" (bild 4.4) har insignalerna Ampl, Freq, ucin och Signal-type. Den logiska insignalen Signal-type avgör om utsignalen (ucout) ska vara identisk med ucin eller om en fyrkantsvåg skall genereras.

Fyrkantsvågen fås efter omvandling av det kontinuerligt växande tal som genereras av "metronom-funktionen.

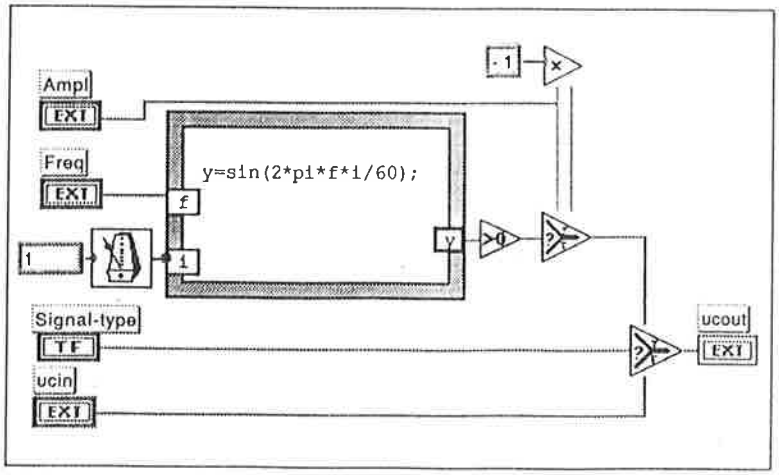


bild 4.4 Ikonen Manu/Square:s Block Diagram.

100

4.6 Ikonen TF201

Underprogrammet "TF201" omvandlar en logisk variabel till noll resp ett.

5 Utprovning av regulatorn

Vid utprovningen utgick vi från den regulatorn som ej är modifierad enligt Appendix C. Regulatorns grundinställningar var: w0=1, zeta=0.7, gamma=2, h=0.1 och fyrkantsvåg (f=0.2 , Ampl=0.3) som referenssignal.

Detta gav till en början ett mycket svängigt system. Efter ett tag verkade dock parametrarna svänga in sig och systemet uppförde sig så som w0 och z angav. Parametrarna t0 och s0 ville dock inte riktigt konvergera, utan "drev" en aning, speciellt när referenssignalen ändrades.

Därefter ökades gamma till 15. Parametrarna uppvisade då ett snabbare insvängningsförlopp men med en tilltagande drivning av s0.

Sedan gjordes modifieringen enligt Appendix C. Åtgärden medförde visserligen en minskad drivning av s0 vid referenssignalsändring, men både t0 och s0 fortsatte att driva något vid konstant referenssignal. Att drivningen härrör från bruseffekter kunde vi också verifiera genom att ändra referenssignalens amplitud. En ökad amplitud gav en ökad drivning av parametrarna och vice versa.

Därefter varierades samplingsintervallet h. Först halverades h till 0.05. Detta medförde en drastisk försämring av regleringen. Regulatorparametrarna verkade visserligen konvergera, men stegsvaret hade ett stort statiskt fel och en översläng som inte överensstämde med vårt önskade system. Detta beror antagligen på att datorn inte hinner med alla beräkningar under samplingsintervallet h. I praktiken medför detta att det verkliga h kommer att variera från skift till skift, vilket gör att vår modell inte stämmer överens med verkligheten. De mest tidsödande beräkningarna upptas troligen av plottningsfunktionerna. Därefter sattes h till 0.2. Detta gav ett stabilt system men stegsvaret överensstämde inte heller riktigt med vårt önskade system. Förklaringen till detta är nog att differentialapproximationen blir för grov vid för stort h.

Efter detta undersökte vi effekten av ändrat w0. En halvering av w0 gav ett långsammare stegsvar (rimligt), men gav dock upphov till ett statiskt fel (?). Fördubblades w0 till 2.0 blev systemet helt instabilt.

6 Slutsatser

Målet med projektet var att konstruera en enkel men hyfsad reglering av en servomotor. Detta mål får vi anse oss ha uppnått.

Problemet med denna MRAS-implementerade regulator är att den är störningskänslig. Regulatorparametrarna konvergerar visserligen först till lämpliga värden men driver sedan bort från dessa. En lösning vore att filtrera utsignalerna. Detta medför dock en så pass komplex regulator att tex "selftuning-teori" ger samma komplexitet men bättre resultat.

En annan erfarenhet från projektet är att differentialapproximering av analog

teori ger ett bra resultat om samplingsfrekvensen är tillräckligt snabb. Approximationen medför en mycket enkel implementering i den diskreta regulatorn.

Kommentarer angående LabView ges i Appendix D.

A Teori (diskret resonemang)

Tidskontinuerlig modell:

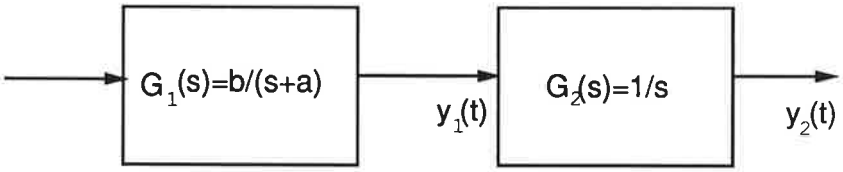


bild 7.1 Blockchema över tidskontinuerlig modell.

Om vi samplar processen får den utseendet:

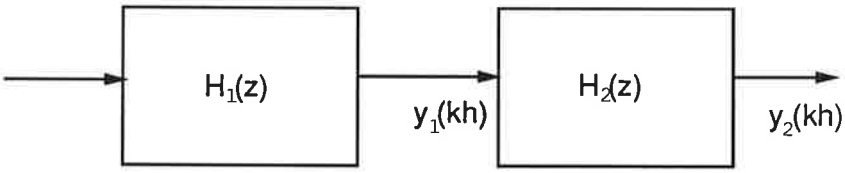


bild 7.2 Blockchema över samplad modell.

Vi sätter att $H_{tot}(z) = sampl(G_{tot}(s))$. Eftersom $y_1(t)$ inte är styckvis konstant så gäller ej att $H_{tot}(z) = sampl(G_1(s)) * sampl(G_2(s))$.

Vi får istället beräkna $H_2(z)$ genom att först beräkna $H_1(z)$ och $H_{tot}(z)$ och sedan bestämma $H_2(z)$ utifrån dessa.

$$H_1(z) = \frac{1-\alpha}{z-\alpha} * \frac{b}{a}$$

$$H_{tot}(z) = \frac{\beta_{11}z + \beta_{12}}{(z-1)(z-\alpha)} * \frac{b}{a}$$

där $\alpha = e^{-ah}$, $\beta_{11} = ah - 1 + \alpha$, $\beta_{12} = 1 - \alpha - ah\alpha$ och h är samplingsintervallet.

Ansätt att:

$$H_2(z) = \frac{\beta_{21}z + \beta_{22}}{z-1}$$

Vi kan sedan beräkna β_{21} och β_{22} ur sambandet:

$$H_{tot}(z) = H_1(z) * H_2(z)$$

Detta ger att:

$$\beta_{21} = \frac{\beta_{11}}{1-\alpha} = \frac{ah-1+\alpha}{a(a-\alpha)} = \frac{ah-1+e^{-ah}}{a(a-e^{-ah})}$$

$$\beta_{22} = \frac{\beta_{12}}{1-\alpha} = \frac{1-\alpha-ah\alpha}{a(a-\alpha)} = \frac{1-e^{-ah}-ahe^{-ah}}{a(a-e^{-ah})}$$

Om vi nu antar att h är litet och serieutvecklar α så blir:

$$\beta_{21} = \beta_{22} = \frac{h}{2}$$

Vi får då överföringsfunktionen:

$$H_2(z) = \frac{h}{2} * \frac{z+1}{z-1}$$

Om vi sätter styrlagen till:

$$U = -l_1 Y_1 + l_2(U_c - Y_2)$$

där $Y_1 = \frac{bh}{z-a}U$ och $Y_2 = \frac{h}{2} * \frac{z+1}{z-1}Y_1$ får vi för det slutna systemet överföringsfunktionen:

$$Y_2 = \frac{\frac{1}{2}bh^2 l_2(z+1)}{z^2 + (l_2h^2b + l_1bh - a - 1)z + a - l_1bh + \frac{l_2h^2b}{2}}U_c$$

Regulatorparametrarna l_1 och l_2:s uppförande kan sedan beräknas mha stabilitetsteori.

B Teori (analogt resonemang med differentialapprox.)

Om vi gör en ren återkoppling av utsignalen y_2 så får vår tidskontinuerliga modell följande utseende:

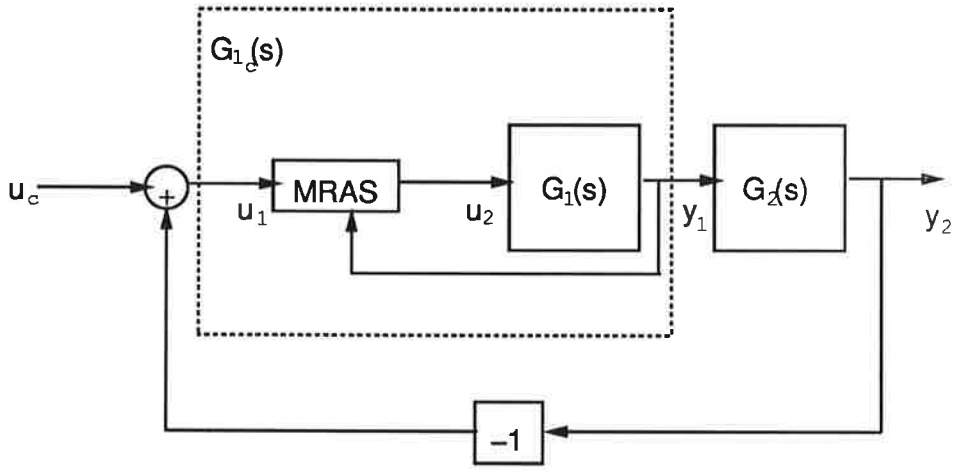


bild 7.3 Blockchema.

Om vi betraktar G_{1_c} som en separat överföringsfunktion blir det slutna systemets överföringsfunktion:

$$G_c = \frac{G_{1_c}}{s + G_{1_c}}$$

Det slutna systemets önskade överföringsfunktionen är:

$$G_m = \frac{\omega_o^2}{s^2 + 2\zeta\omega_o s + \omega_o^2}$$

Detta blir uppfyllt om vi reglerar G_1 så att:

$$G_{1_c} = \frac{\omega_o^2}{s + 2\zeta\omega_o} = \frac{b_m}{s + a_m}$$

Detta innebär att vi endast behöver reglera en funktion av första ordningen!

Vi sätter styrlagen till:

$$u_2(t) = t_o u_1(t) - s_o y_1(t)$$

där alltså $u_1 = u_c - y_2$. Enligt Lyapanovteorin varierar regulatorparametrarna t_o och s_o då enligt:

$$\frac{dt_o}{dt} = -\gamma u_1 e$$

$$\frac{ds_o}{dt} = \gamma y_1 e$$

104

där $e = y_1(t) - y_m(t)$.

Om vi nu på ett enkelt sätt vill implementera detta i en tidsdiskret regulator så differentialapproximerar vi derivatorna. Vi antar att samplingsintervallet h är litet och får då att:

$$\frac{dt_o}{dt} \approx \frac{t_o(t+h) - t_o(t)}{h} = -\gamma u_1 e$$

$$\frac{ds_o}{dt} \approx \frac{s_o(t+h) - s_o(t)}{h} = \gamma y_1 e$$

$$\Rightarrow$$

$$t_o(t+h) = t_o(t) - \gamma u_1 e h$$

$$s_o(t+h) = s_o(t) + \gamma y_1 e h$$

Referensfunktionen $G_m(s)$ i vår MRAS-regulator implementerar vi enklast genom vanligt tidsdiskret resonemang:

$$H_m(z) = sampl(G_m(s)) = \frac{1 - e^{-a_m h}}{z - e^{-a_m h}} * \frac{b_m}{a_m}$$

$$\Rightarrow y_m(t+h) = -e^{-a_m h} y_m(t) + \frac{b_m}{a_m} * (1 - e^{-a_m h}) u_1(t)$$

C Brusinverkan på regulatorparametrarna

Vi approximerar bruset på utsignalen till att vara vitt. Utsignalen y_1 kan då skrivas som:

$$y_1 = y_{1_{t(-true)}} + n(t)$$

där $n(t)$ är vitt brus. Om vi sätter in detta i resultatet från Appandix B får vi att:

$$s_0(t+h) = s_0(t) + \gamma(y_{1_t} + n(t))eh = s_0(t) + \gamma(y_{1_t} + n(t))(y_{1_t} + n(t) - y_m)h$$

Vi ser att uttrycket bla innehåller brusfaktorn $n^2(t)$. Ett enkelt sätt att elemenera denna är att fördröja $y_1(t)$ i den ena parantesen. Brusfaktorn elemineras då eftersom produkten av brusbidragen blir: $n(t-h)n(t) = 0$. Om vi inför beteckningen:

$$y_{1_{d(-elayed)}}(t) = y_1(t-h)$$

så blir då den modifierade parameteruppdateringen:

$$s_o(t+h) = s_o(t) + \gamma y_{1_d}(t) e(t) h$$

D Kommentarer och tips angående LabView

Efter ca två veckors arbete med LabView skulle jag här vilja komma med några synpunkter på programpaketet.

LabView:s största tillgång är dess grafiska snitt. Det är lätt att få en snygg och "häftig" "FrontPanel" som kan göras mycket användarvänlig. Det finns oändliga möjligheter att kombinera knappar, spakar, skyltar, grafer osv, som dessutom kan färgläggas efter behag. Detta får ses som LabView:s största fördel.

Vad gäller implementering av programvara i "BlockDiagram" verkar det vid första anblicken vara ganska finurligt med "wire- och symbol strukturen". Problemet är dock att programmet kan bli fullständigt oöverskådligt när dess storlek växer. Detta var något vi visste redan vid projektets början. Därför tänkte vi utnyttja "formelnoden" i möjligaste mån. Formelnoden visade sig dock ha vissa brister. Till exempel kan inte lokala variabler och konstanter användas. Dessa får istället deklareras som output-variabler. Dessutom kan man inte skriva ledtext inne i formelnoden.

För övrigt kan ett rörigt program undvikas genom att utnyttja "ikon-funktionen". Ikonerna kan användas för att bygga upp ett välstrukturerat hierarktiskt program. En annan fördel med ikonerna är att de kan exekveras och provköras var för sig. I manualerna framgår inte helt klart hur en LabView-fil editeras till en ikon. Efter att ha definierat in-och utsignaler i ikonen sparas filen med kommandot "file/ Save VI et al. As...". Sedan infogas ikonen i huvudprogrammet genom kommandot "functions/ VI...".

En annan nackdel med LabView är att det uppstår problemem vid parallell exikvering av två while-loopar. Om den ena loopen genererar en inparameter till den andra så blir programmet omöjligt att exekvera. Även "metronom-funktionen" uppför sig konstigt när flera sådana används samtidigt. Ändrar man den ena "metronom-hastigheten" så ändras även den andra.

Den största nackdelen är dock att maximala samplingsfrekvensen endast är 60 Hz när man använder sig av "metronom-funktionen", trots att DA/AD-omvandlaren har betydligt större kapacitet. Att sampla med teoretiskt möjliga 60 Hz visade sig dessutom vara omöjligt eftersom LabView:s grafiska beräkningsalgoritmer är alltför tidskrävande. Även vid relativt stora samplingsintervall (h=0,3 s) uppstår variationer i "metronomhastigheten".

Ytterligare en erfarenhet från projektet är att LabView:s utskrifter av FrontPanel och Blockdiagram ej går att översätta till postskriptfiler i TEX.

Två andra tips är dels tangentfunktionen "tab" som ändrar aktuell "tool-funktion" och dels "edit/ Remove bad wires"-funktionen som är helt oumbärlig vid "tråddragning".

RST-REGULATOR IMPLEMENTERAD I C MED LABVIEW SOM ANVÄNDARGRÄNSSNITT

Gjort för kurserna Datorimplementering av Reglersystem och Adaptiv Reglering

Handledare: Anders Blomdell och Karl-Johan Åström

Utfört av: Fredrik Menander E-87
Lars Falk E-87
Johan Silvander E-87
Marianne Sernevi E-86

Sammanfattning

Problemet var att implementera en RST- regulator i C kod.
Som användargränssnitt utnyttjade vi LabVIEW och undersökte på så sätt dess brister och fördelar. Vi kom fram till att det smidiga sätt på vilket man kan skapa ett bra användargränssnitt inte väger upp nackdelarna då den används som controller.

Innehållsförteckning

1 Inledning

Uppgiften var från början att implementera en indirekt självinställande regulator med hjälp av en signalprocessor. Vårt användar gränssnitt skulle göras med hjälp av LabVIEW. Programmet skulle skrivas i en extern C-fil som sedan skulle kompileras och laddas ner i signalprocessorn. Nu hör det till saken att den så omtalade kompilatorn till signalprocessorn ännu inte har påträffats i Lund utan för en undanskymd tillvaro i något transport medel. Vår uppgift har därför fått en del modifieringar och innebär numera att vi ska se hur gränssnittet mellan en C-fil och LabVIEW fungerar.

Speciella tack till Anders Blomdell och Kjell Gustavsson.

2 Utförande

2.1 Regulatorn

Regulatorn är byggd för att styra ett av laborationsservona. Servot beskriv med en andra ordningens modell, G(s) = b/(s(s+a)). Efter sampling fås H(z) = (b1*z+b2)/(z*z+a1*z+a2). Processen får då ett nollställe i z=-b2/b1 som ej ska förkortas av regulatorn. Vi valde att konstruera den med RST-polynom med intergrator. Polynomsyntes enligt Appendix A ger en regulator med andra ordningens R-, S-, T-, observerar- och modell-polynom.

Genom estimering av a1, a2, b1 och b2 kan parametrarna i RST-polynomen ändras on-line. Eftersom estimatorn kan ge skattningar som gör att A och B-polynomen har gemensamma faktorer eller att $b1 \approx 0$, så måste vi behandla olika fall i beräkningen av RST-polynomen. Dessa olika fall återfinnes i Appendix A, där fall:
a) beskriver normalfallet.
b) beskriver då $b1 \approx 0$ alt. $b1 << b2$.
c) beskriver då A och B har gemensamma faktorer.

Modell och observerare specificeras med parametrarna zeta och omega. Zeta väljes genomgående till 0.71 och omega till ca 4. Samplingsintervallet, h, har inte kunnat väljas fritt utan är beroende av programkodens omfattning. Vid mätning med oscilloskop uppskattades h till ca 0.06 sekunder. Regulatorkoden är skriven i programspråket C och finns i Appendix B.

2.2 Estimatorn

För att realisera den indirekta adaptiva regulatorn samt estimatorn så använde vi en metod med konstant spår. Denna metod bygger på att man dividerar sin kovarians matris med sitt spår och på så sätt försöker att undvika estimator uppvridning.

Följande formler användes:

$$q(t)=q(t-1) + a(t)*K(t)*(y(t)-j^T(t)*q(t-1))$$

$$K(t)=P(t-1)*j(t) / (1+j^T(t)*P(t-1)*j(t))$$

$$\bar{P}(t)=\bar{P}(t-1)-a(t)*(P(t-1)*j(t)*j^T(t)*P(t-1) / (1+ j^T(t)*P(t-1)*j(t))$$

$$P(t)=c1*\bar{P}(t) / \text{trace } \bar{P}(t) +c2*I$$

$$a(t)= a \text{ om } |y(t) - j^T(t)*q(t-1)| > \text{delta annars } 0$$

Lite förklaringar är kanske på sin plats så de kommer här nedan.

$$q = [\, u(t)\ u(t-1)\ -y(t)\ -y(t-1)\,]$$

$$j = [\, b1\ b2\ a1\ a2\,]$$

Angående konstanterna så valdes a=0.5 , c1=1E-2 och c2=1E-6. Då delta ska representera störningar så ger vi den värdet av ett fel på två bitar i AD-omvandlaren. Kovarians matrisen P och P initieras till 100 i diagonalelementen.De flesta valen har skett på inrådan av boken "Adaptive Control" och en av dess författare, nämligen K J Åström.

2.3 Gränssnitt

2.3.1 LabVIEW

LabVIEW är ett programpaket där man grafiskt kan skapa olika slags mätinstrument, filter och andra enheter som behandlar mätdata. Här finns också bra I/O-möjligheter. Ett instrument byggs med en kombination av "controls" och "indicators". "Controls" används för att generera insignaler till ditt instrument och "indicators" visar utsignalerna från instrumentet. Exempel på "controls" är vridknappar, switchar och grafer. Exempel på "indicators" är visardisplayer, sifferindikatorer och grafer. När man bygger sitt instrument arbetar man i två olika slags fönster, panel- och diagram-fönstret. I panel-fönstret visas "controls" och "indicators" och i diagram-fönstret bygger man sitt instrument.

Vi använder LabVIEW som ett gränssnitt mellan människa - maskin och mellan regulatorkod - I/O. I vårt panel-fönster visas RST-parametrarna, skattningarna av A och B, referensvärde, styrsignal och utsignal. Här bestämmer man också hur regulatorn ska fungera t ex adaptera, estimera eller bara reglera. Detta bestäms med logiska switchar. Se bild i Appendix B. I vårt diagram-fönster finns kopplingen mellan regulatorkoden och våra "controls" och "indicators". Kopplingen sker via en Code Interface Node (CIN), som kommunicerar med en extern C-fil. Regulatorkoden finns i den externa C-filen.

2.3.2 Code Interface Node

I LabVIEW ser en CIN ut som en kopplingsplint dit man kan dra insgnaler från t ex switchar. Dessa insignaler kan behandlas i ett program för att sedan skickas tillbaka som utsignaler och visas i t ex en graf. Vi har genomgående använt pekarvariabler när vi skickat variabler mellan CIN och LabVIEW.

När man bygger en CIN krävs det minst två filer, en "header"-fil och en program-fil. I bägge filerna finns det sex stycken förutbestämda underprogram. I dessa specificeras t ex variablernas initialvärden eller vad som ska hända då instrumentet stoppas. Underprogrammet CINRun anropas då instrumentet exekveras. I "header"-filen deklareras de variabler som används av respektive underprogram. Se Appendix B fil CIN2.h. I programfilen finns själva C-koden. Här kan man i CINRun skriva sin exekverbara kod. Vi har valt att i CINRun anropa en extern C-fil där regulatorkoden finns. Se Appendix B fil CIN2.c. Hur man länkar filer och skapar sin källkod finns beskrivet (någorlunda) i kap. 19, LabVIEW2 User Manual, National Instruments.

2.3.3 Extern C-fil

I den externa C-filen finns ett antal procedurer samt ett huvudprogram, main. I huvudet i main finns de variabler som skickades i anropet av regulatorn i filen CIN2.c. Insignaler till programmet är referenssignal, mätvärde samt en del logiska variabler. Utsignaler är styrsignal, RST-parametrar och de skattade koefficienterna i A- respektive B-polynomet. Filen anropas och exekveras så fort som processorn hinner med. Detta gör att sampeltiden blir både svårbestämd och varierande. För att tillstånden ska kunna uppdateras mellan exekveringarna måste de vara deklarerade som statiska variabler utanför main. Regulatorkoden finns i sin helhet i Appendix B fil Regul2.c.

2.4 Digital Signal Processor

Den processor vi skulle ha använt kommer från Texas Instruments och bär det fantasieggande namnet TMS320.Denna DSP har inbyggda timers vilka kan styras med hjälp av speciella register. Detta innebär att man kan få en bestämd samplingshastighet genom att koppla timern till ett hårdvaruavbrott. Då avbrottet inträffar så sker inläsning från AD-omvandlaren.Vi hade tänkt använda denna teknink för våran implementering.

Ett annat problem som vi ännu inte funderat på hur vi ska lösa är hanteringen av de gemensamma variablerna. Vad som krävs är ett gemensamt minnesutrymme eller någon annan överföring via en port. I stort sett är dessa problemen de samma som för annan realtidsprogrammering. Ett bra hjälpmedel hade varit en realtidskärna i C.

3 Resultat och slutsatser

Eftersom kompilatorn till processorn aldrig anlände reglerar vi nu servot med hjälp av LabVIEW som anropar en extern C-fil som innehåller koden för regulatorn och estimatorn. LabVIEW är ett utmärkt hjälpmedel för att göra ett snyggt användargränssnitt.Tyvärr kan man med dess hjälp aldrig uppnå någon snabbhet i regleringen och man har ingen flexibilitet vad det gäller hårdvaran.I vårt fall så var problemet att det var svårt att veta den exakta samplingstiden då denna beror på antalet uträkningar som ska utföras. Det finns dock möjlighet att undgå dettta men då vi byggde upp vårt program för att styras av avbrott så lämnade vi denna lösning därhän. Faktum kvarstår att maximal sampling med LabVIEW är ett tick, dvs 1/60 sekund, vilket kan medföra problem. En för del jämfört med PC miljö är att länkningen av externa filer är mycket behändig och snabb i alla fall med de referenser som vi har. Med hänsyn till ovanstående anser vi därför att det är önskvärt att ha LabVIEW enbart som användargränssnitt.

Vad gäller själva regleringen så lämnar den en hel del övrigt att önska. Reglering utan adaption klarar regulatorn utan större problem, dock kunde snabbheten var lite bättre.Detta beror förmodligen på att vi har räknat ut de "optimala" parametrarna för en given sampeltid men LabVIEW låter denna variera beroende på ovanstående. Då adaptionen används så verkar regleringen bli lite snabbare men i gengäld så får vi en svängigare utsignal, alltså inget för Barsebäck. En titt på de estimerade a- och b-parametrarna visar att dessa ställer in sig relativt fort och ligger sedan kvar på dessa värden.

I det stora hela så har vi fått bekräftat vad vi redan visste, nämligen att det går åt mycket tid så fort man ska sätta sig in i nya saker .

REGULATORDIMENSIONERING

Proc.: $y_t + a_1 y_{t-1} + a_2 y_{t-2} = b_1 u_{t-1} + b_2 u_{t-2}$

$$AR + BS = A_0 A_m = (z^2 + a_{01} z + a_{02})(z^2 + a_{m1} z + a_{m2})$$

a) $$\underbrace{(z^2 + a_1 z + a_2)}_{A(z)}\underbrace{(z-1)}_{\Delta}\underbrace{(z + r_1)}_{R(z)} + \underbrace{(b_1 z + b_2)}_{B(z)}\underbrace{(s_0 z^2 + s_1 z + s_2)}_{S(z)} = A_0(z) A_m(z)$$

$\deg A_0 \geq 2 \deg A - \deg A_m - \deg B^+ - 1 + 1 = 4 - 2 - 1 + 1 = 2$

1) sätt $z = z_1 = -\frac{b_2}{b_1}$ OBS: $b_1 \neq 0$

$$\Rightarrow r_1 = -z_1 + \frac{A_0(z_1) A_m(z_1)}{A(z_1)(z_1 - 1)} \quad \text{OBS: } A(z_1), (z_1 - 1) \neq 0$$

2) sätt $z = 1$

$$\Rightarrow (b_1 + b_2)(s_0 + s_1 + s_2) = A_0(1) A_m(1)$$

3) z^0: $-a_2 r_1 + b_2 s_2 = a_{02} a_{m2}$

z^1: $-a_1 r_1 + a_2 r_1 - a_2 + b_1 s_2 + b_2 s_1 = a_{01} a_{m2} + a_{02} a_{m1}$

z^2: $-r_1 - a_1 + a_1 r_1 + a_2 + b_1 s_1 + b_2 s_0 = a_{m2} + a_{01} a_{m1} + a_{02}$

z^3: $r_1 - 1 + a_1 + b_1 s_0 = a_{m1} + a_{01}$

$$(b_1+b_2)s_0 + (b_1+b_2)s_1 + (b_1+b_2)s_2 = 1 + am_1 + am_2 + ao_1 + ao_1 am_1 + ao_1 am_2 + ao_2 + ao_2 am_1 + ao_2 am_2$$

$$b_2 \cdot s_2 = ao_2 am_2 + a_2 r_1$$

$$b_2 s_1 + b_1 s_2 = ao_1 am_2 + ao_2 am_1 + a_1 r_1 - a_2 r_1 + a_2$$

$$b_2 s_0 + b_1 s_1 = am_2 + ao_1 am_1 + ao_2 + r_1 + a_1 - a_1 r_1 - a_2$$

$$b_1 s_0 = am_1 + ao_1 - r_1 + 1 - a_1$$

given effe koll : $b_1 \neq 0$, $b_2 \neq -b_1$

$$\begin{cases} s_0 = \frac{1}{b_1}\left(am_1 + ao_1 - r_1 + 1 - a_1\right) \\ s_1 = \frac{1}{b_1}\left(am_2 + ao_1 am_1 + ao_2 + r_1 + a_1 - a_1 r_1 - b_2 s_0 - a_2\right) \\ s_2 = \frac{1}{b_1}\left(ao_1 am_2 + ao_2 am_1 + a_1 r_1 - a_2 r_1 - b_2 s_1 + a_2\right) \\ r_1 = \frac{b_2^3(a_1 - 1 - am_1 - ao_1) + b_1 b_2^2(a_1 - a_2 + am_2 + ao_1 am_1 + ao_2)}{(b_2^2 - a_1 b_1 b_2 + a_2 b_1^2)(-b_2 - b_1)} \\ \quad + \frac{-b_1^2 b_2(a_2 + ao_1 am_2 + ao_2 am_1) + b_1^3 ao_2 am_2}{(b_2^2 - a_1 b_1 b_2 + a_2 b_1^2)(-b_2 - b_1)} \end{cases}$$

$B_m = B^- \cdot B'_m$ $\quad B^- = B = b_1 z + b_2$

$T = A_0 \cdot B'_m$ $\quad \frac{B_m}{A_m} = B'_m \cdot \frac{b_1 z + b_2}{z^2 + am_1 z + am_2}$

$T = b_m(z^2 + ao_1 z + ao_2)$ $\quad \Rightarrow B'_m = b_m = \frac{1 + am_1 + am_2}{b_1 + b_2}$

$$\begin{cases} t_0 = b_m \\ t_1 = b_m ao_1 \\ t_2 = b_m ao_2 \end{cases}$$

b) $b_1 = 0$, $b_1 << b_2$

$$\underbrace{(z^2 + a_1 z + a_2)}_{A(z)}\underbrace{(z-1)(z+r_1)}_{R(z)} + b_2\underbrace{(s_0 z^2 + s_1 z + s_2)}_{S(z)} = A_0(z)\cdot A_m(z) \qquad B$$

$\deg A_0 = 2$ $\qquad B_m = b_m$

z^0 : $-a_2 r_1 + b_2 s_2 = a_{0_2} a_{m_2}$

z^1 : $-a_1 r_1 + a_2 r_1 - a_2 + b_2 s_1 = a_{0_1} a_{m_2} + a_{0_2} a_{m_1}$

z^2 : $-r_1 + a_1 r_1 - a_1 + a_2 + b_2 s_0 = a_{m_2} + a_{0_1} a_{m_1} + a_{0_2}$

z^3 : $r_1 - 1 + a_1 = a_{m_1} + a_{0_1}$

$$\begin{cases} r_1 = 1 - a_1 + a_{m_1} + a_{0_1} \\ s_0 = \frac{1}{b_2}\left(r_1 - a_1 r_1 + a_1 - a_2 + a_{m_2} + a_{0_1} a_{m_1} + a_{0_2}\right) \\ s_1 = \frac{1}{b_2}\left(a_1 r_1 - a_2 r_1 + a_2 + a_{0_1} a_{m_2} + a_{0_2} a_{m_1}\right) \\ s_2 = \frac{1}{b_2}\left(a_2 r_1 + a_{0_2} a_{m_2}\right) \end{cases}$$

$B = b_2 = B^-$

$B_m = B^- B'_m$

$$\frac{B_m}{A_m} = \frac{B'_m \cdot b_2}{z^2 + a_{m_1} z + a_{m_2}} \Rightarrow B'_m = b_m = \frac{1 + a_{m_1} + a_{m_2}}{b_2}$$

$T = A_0 B'_m = b_m\left(z^2 + a_{0_1} z + a_{0_2}\right)$

$$\begin{cases} t_0 = b_m \\ t_1 = b_m a_{0_1} \\ t_2 = b_m a_{0_2} \end{cases}$$

c) B- och A-polynomen har gemensamt nollställe

$$B = b_1 z + b_2 = b_1 \left(z + \frac{b_2}{b_1} \right)$$

nollställe : $z = -\frac{b_2}{b_1}$

$$A = z^2 + a_1 z + a_2 = \left(z + a\right)\left(z + \frac{b_2}{b_1}\right) = z^2 + \left(a + \frac{b_2}{b_1}\right) z + \frac{a b_2}{b_1}$$

identificering :
$$\begin{cases} a = \frac{b_1}{b_2} a_2 \\ a = a_1 - \frac{b_2}{b_1} \end{cases}$$

$$\Rightarrow \quad \frac{B}{A} = \frac{b_1}{z + a}$$

$$\underbrace{(z+a)(z-1)}_{\cdot R(z)} + b_1 \cdot (s_1 z + s_2) = A_0(z) A_m(z)$$

deg $A_0 = 1$ väl: $A_0 = z + a_0$ $\quad A_m = z + a_m$

z^0 : $\quad -a + b_1 s_2 = a_m \cdot a_0$

z^1 : $\quad a - 1 + b_1 s_1 = a_m + a_0$

$$\begin{cases} s_1 = \frac{1}{b_1}(a_m + a_0 + 1 - a) \\ s_2 = \frac{1}{b_1}(a + a_m a_0) \\ a = \frac{b_1}{b_2} \cdot a_2 \quad \text{alt} \quad a_1 - \frac{b_2}{b_1} \end{cases}$$

$B = b_1 = B^-$ $\qquad \frac{B_m}{A_m} = B_m' \frac{b_1}{z + a_m}$

$B_m = B^- B_m'$

$B_m' = b_m = \frac{1 + a_m}{b_1}$

$T = A_0 B_m' = b_m (z + a_0)$

$$\begin{cases} t_1 = b_m \\ t_2 = b_m a_0 \end{cases}$$

Beräkning av styrsignal :

Antiwindup :

$$A_0 \cdot v = T u_c - S y + (A_0 - R) u$$

$$(1 + a0_1 z^{-1} + a0_2 z^{-2}) v = (t_0 + t_1 z^{-1} + t_2 z^{-2}) u_c$$

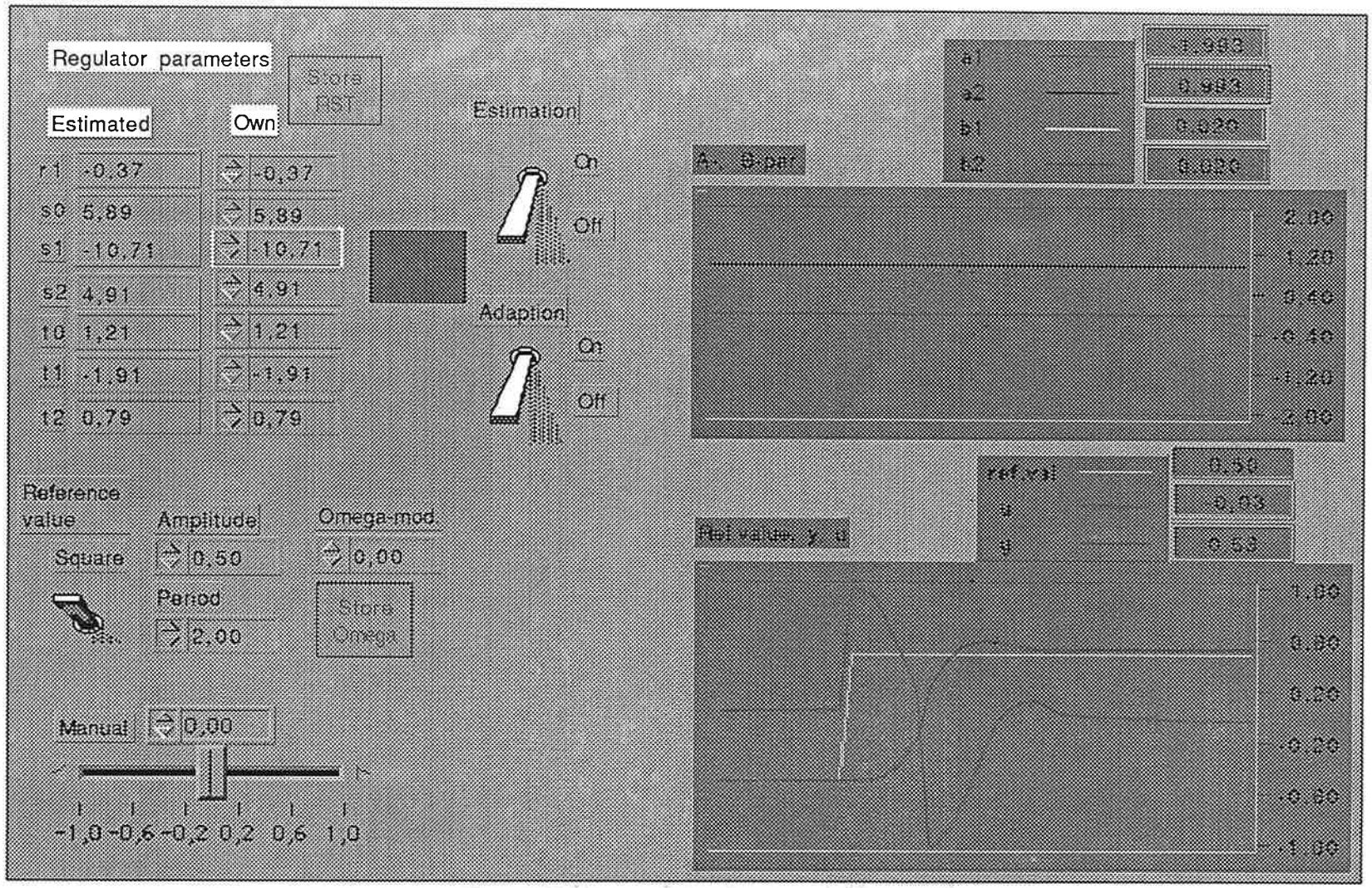
$$- (s_0 + s_1 z^{-1} + s_2 z^{-2}) y + \left((a0_1 - r_1 + 1) z^{-1} + (a0_2 + r_1) z^{-2} \right) u$$

$$x = -a0_1 v(t-1) - a0_2 v(t-2) + t_1 u_c(t-1) + t_2 u_c(t-2)$$

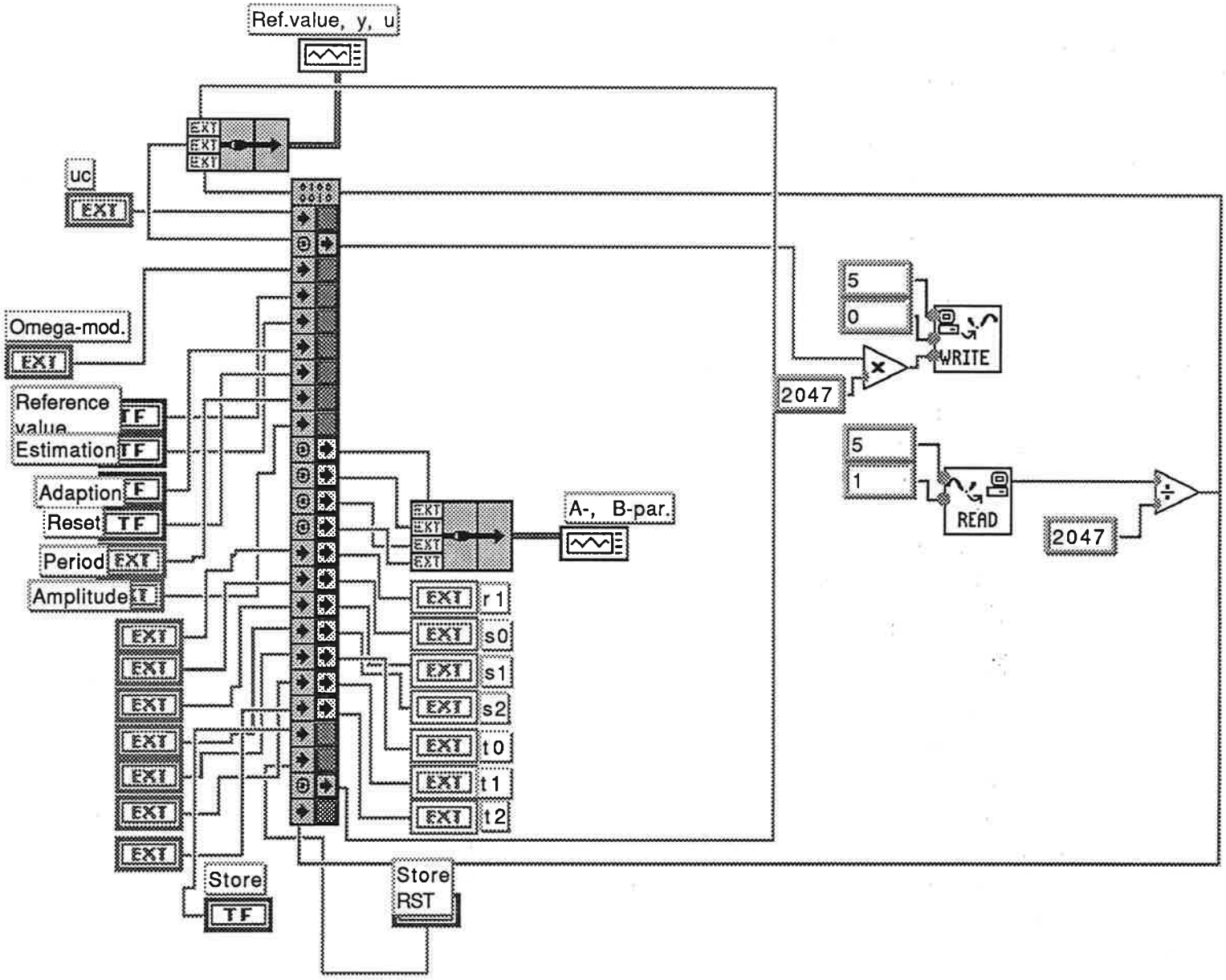
$$- s_1 y(t-1) - s_2 y(t-2) + (a0_1 - r_1 + 1) u(t-1) + (a0_2 + r_1) u(t-2)$$

$$v(t) = t_0 u_c(t) - s_0 y(t) + x$$

$$u(t) = sat\left(v(t) \right)$$


Regulator parameters
Store PST
Estimated
Own
r1 -0,37
s0 5,89
s1 -10,71
s2 4,91
t0 1,21
t1 -1,91
t2 0,79
-0,37
5,89
-10,71
4,91
1,21
-1,91
0,79
Estimation
On
Off
Adaption
On
Off
Reference value
Square
Amplitude
0,50
Period
2,00
Omega-mod
0,00
Store Omega
Manual
0,00
-1,0 -0,6 -0,2 0,2 0,6 1,0
a1
a2
b1
b2
-1,993
0,993
0,020
0,020
2,00
1,20
0,40
-0,40
-1,20
-2,00
ref_val
u
y
0,50
0,03
0,53
1,00
0,80
0,20
-0,20
-0,60
-1,00

Block Diagram



```
/* CIN2 header file */

pascal void CINInit(double *ua);
pascal void CINDispose(void);
pascal void CINAbort(double *ua);
pascal void CINRun(double *uca, double *ua,double *wma,
                   int *refvala, int *estima, int *adapta, int *reseta,
                   double *perioda, double *ampa, double *a1a, double *a2a,
                   double *b1a, double *b2a, double *r1a,double *s0a,
                   double *s1a,double *s2a, double *t0a, double *t1a,
                   double *t2a,int *storea ,int *storeRSTa,double *ucuta, double *ya)
pascal void CINLoad(void);
pascal void CINsave(void);
```

```
/* CIN.
*  Indirect adaptive controller for DC-servo using RST-design.
*   Lars Falk
*   Fredrik Menander
*   Marianne Sernevi
*   Johan Silvander */


/*declarations*/


#include "CIN2.h"
#include <SetUpA4.h>

pascal void CINInit(ua)
    double *ua;
{
    *ua = 0;
    Regul2(1);
}

pascal void CINDispose() {}

pascal void CINAbort(ua)
    double *ua;
{
    *ua = 0;
    Regul2(1);
}

pascal void CINRun(uca, ua, wma,refvala,estima,adapta,reseta,perioda,ampa,
                   a1a,a2a,b1a,b2a, r1a, s0a, s1a, s2a, t0a, t1a, t2a, storea,
                    storeRSTa,ucuta, ya)

    int *refvala,*estima,*adapta,*reseta,*storeRSTa,*storea;
    double *uca, *ua,*wma, *r1a, *s0a, *s1a, *s2a, *t0a,*t1a, *t2a,
           *perioda,*ampa,*a1a,*a2a,*b1a,*b2a,*ucuta, *ya;



{
    Regul2(0,uca,ua,wma,refvala,estima,adapta,reseta,perioda,ampa,a1a,a2a,b1a,b2a,
           r1a,s0a,s1a,s2a,t0a,t1a,t2a,storea,storeRSTa,ucuta, ya);
}

pascal void CINLoad(){}
pascal void CINSave (){}
```

```
/*  Indirect adaptive controller for DC-servo using RST-design.*/
/*  Lars Falk*/
/*  Fredrik Menander*/
/*  Marianne Sernevi*/
/*  Johan Silvander */


#include <SetUpA4.h>
#include "CIN881.h"

static int refval,adapt,estim,storeRST,store,reset;
static double   r1e, s0e, s1e, s2e, t0e, t1e, t2e;
static double uc1,uc2,u1,u2,v,v1,v2,y1,y2,x1;
static double a1,a2,b1,b2;
static double a,c1,c2, delta;
static double K,aa;
static double am1,am2,bm,zm,wm,ao1,ao2,zo,wo,am,ao,ac,h;
static double umin,umax,z1,den,nom,eps;
static double P11,P12,P13,P14,P22,P23,P24,P33,P34,P44,Pt11,Pt12,Pt13,
              Pt14,Pt22,Pt23,Pt24,Pt33,Pt34,Pt44,th1,th2,th3,th4;
static double   r1, s0, s1, s2, t0, t1, t2;
double  f1,f2,f3,f4,K1,K2,K3,K4,delfel;
double u, uc, y,tick,amp,period,ucut;

InitGlobals()
{
        h = 0.06;
        zm = 0.71;
        wm = 4.00;
        zo = 0.71;
        wo = 5.00;
        K = 11.2;               /* G(s) = K/(s*(s+aa)) */
        aa = 0.12;
        umax = 1;
        umin = -1;
        eps = 1E-6;
        a = 0.5;
        c1 = 1E-2;
        c2 = 1E-6;
        delta = 1E-6;
        tick = 0;
        refval = adapt =  store = estim = storeRST = 0;

        ModUpdate();
        ResetEst();
        ResetStates();
        ControllerA();
        Est2Contr();
};

ModUpdate()
{
        am1 = -2*exp(-zm*wm*h)*cos(sqrt(1-zm*zm)*wm*h);
```

```
        am2 = exp(-2*zm*wm*h);

        ao1 = -2*exp(-zo*wo*h)*cos(sqrt(1-zo*zo)*wo*h);
        ao2 = exp(-2*zo*wo*h);

        am = exp(-wm*h);
        ao = exp(-wo*h);
};

Est2Contr()
{
        r1e = r1;
        s0e = s0;
        s1e = s1;
        s2e = s2;
        t0e = t0;
        t1e = t1;
        t2e = t2;
};

ResetStates()
{
    uc = uc1 = uc2 = 0;
    u = u1 = u2 = 0;
    v = v1 = v2 = 0;
    y = y1 = y2 = 0;
    x1 = 0;
};

ResetEst()
{
    P11 = P22 = P33 = P44 = 100;
    Pt11 = Pt22 = Pt33 = Pt44 = 100;
    P12 = P13 = P14 = P23 = P24 = P34 = 0;
    Pt12 = Pt13 = Pt14 = Pt23 = Pt24 = Pt34 = 0;

    a1= th3 = -(1+exp(-aa*h));                                  /*-1.998*/
    a2= th4 = exp(-aa*h);                                       /*0.998*/
    b1= th1 = K/aa*(aa*h-1+exp(-aa*h))/aa;                      /*1.55E-3*/
    b2= th2 = K/aa*(1-exp(-aa*h)-aa*h*exp(-aa*h))/aa;           /*1.55E-3*/
};

Square()
{
    tick += 1;
    if ((tick)<(period*60/2)) {
        uc = amp;
    } else {
        uc = -amp;
    };
    if ((tick)>(period*60)) {
        tick = 0;
    }
```

```
};

Checkz1()
{
        z1 = - (b2/b1);

        if (fabs(z1-1) < eps) {
            UpdateStates();
            x1WhenAB();
        } else if (fabs(b1/b2) < eps) {
            ControllerB();
            UpdateStates();
            x1WhenAB();
        } else if (fabs(z1*z1 + a1*z1 + a2) < eps) {
            ControllerC();
            UpdateStates();
            x1WhenC();
        } else {
            ControllerA();
            UpdateStates();
            x1WhenAB();
        };
};



Estimate ()
{
        double term1,term2,term3,term4,e,denom,trPt;

        f1 = u;
        f2 = u1;
        f3 = -y;
        f4 = -y1;

        delfel = f1*th1+f2*th2+f3*th3+f4*th4;

        e = y-delfel;

        if (fabs(e) > delta) {
            a = 0.1;
        } else {
            a = 0;
        }

        term1 = P11*f1+P12*f2+P13*f3+P14*f4;
        term2 = P12*f1+P22*f2+P23*f3+P24*f4;
        term3 = P13*f1+P23*f2+P33*f3+P34*f4;
        term4 = P14*f1+P24*f2+P34*f3+P44*f4;

        denom = 0.99+(term1*f1+term2*f2+term3*f3+term4*f4);
```

```
        K1 = term1/denom;
        K2 = term2/denom;
        K3 = term3/denom;
        K4 = term4/denom;


        th1 = th1 + K1*e;
        th2 = th2 + K2*e;
        th3 = th3 + K3*e;
        th4 = th4 + K4*e;



        Pt11 = Pt11-a*(term1*term1)/denom;
        Pt12 = Pt12-a*(term1*term2)/denom;
        Pt13 = Pt13-a*(term1*term3)/denom;
        Pt14 = Pt14-a*(term1*term4)/denom;
        Pt22 = Pt22-a*(term2*term2)/denom;
        Pt23 = Pt23-a*(term2*term3)/denom;
        Pt24 = Pt24-a*(term2*term4)/denom;
        Pt33 = Pt33-a*(term3*term3)/denom;
        Pt34 = Pt34-a*(term3*term4)/denom;
        Pt44 = Pt44-a*(term4*term4)/denom;

        trPt = Pt11+Pt22+Pt33+Pt44;

        P11 = c1*Pt11/trPt +c2;
        P12 = c1*Pt12/trPt;
        P13 = c1*Pt13/trPt;
        P14 = c1*Pt14/trPt;
        P22 = c1*Pt22/trPt+c2;
        P23 = c1*Pt23/trPt;
        P24 = c1*Pt24/trPt;
        P33 = c1*Pt33/trPt+c2;
        P34 = c1*Pt34/trPt;
        P44 = c1*Pt44/trPt+c2;

        b1 = th1;
        b2 = th2;
        a1 = th3;
        a2 = th4;
};


ControllerA()
{
        den =  (b2*b2-a1*b1*b2+a2*b1*b1)*(-b2-b1);
        nom =  b2*b2*b2*(a1-1-am1-ao1) + b1*b2*b2*(a1-a2+am2+ao1*am1+ao2);
        nom = nom + (- b1*b1*b2*(a2+ao1*am2+ao2*am1) + b1*b1*b1*ao2*am2);
        r1 = nom/den;
        s0 = (am1+ao1-r1+1-a1)/b1;
        s1 = (am2+ao1*am1+ao2+r1+a1-a1*r1-a2-b2*s0)/b1;
```

```
        s2 = (ao1*am2+ao2*am1+a1*r1-a2*r1+a2-b2*s1)/b1;
        bm = (1+am1+am2)/(b1+b2);
        t0 = bm;
        t1 = bm*ao1;
        t2 = bm*ao2;
};

ControllerB()
{
        r1 = 1-a1+am1+ao1;
        s0 = (r1-a1*r1+a1-a2+am2+am1*ao1+ao2)/b2;
        s1 = (a1*r1-a2*r1+a2+ao1*am2+ao2*am1)/b2;
        s2 = (a2*r1+ao2*am2)/b2;
        bm = (1+am1+am2)/b2;
        t0 = bm;
        t1 = bm*ao1;
        t2 = bm*ao2;
};

ControllerC()
{
        r1 = 0;
        ac = b1*a2/b2;
        s0 = (am+ao+1-a)/b1;
        s1 = (a+am*ao)/b1;
        s2 = 0;
        bm = (1+am)/b1;
        t0 = bm;
        t1 = bm*ao;
        t2 = 0;
};


Sat_v()
{
        if (v<umax && v>umin) {
            u = v;
        } else if (v>umax) {
            u = umax;
        } else {
            u = umin;
        };
};

UpdateStates()
{
        u2 = u1;
        u1 = u;
        uc2 = uc1;
        uc1 = uc;
        y2 = y1;
        y1 = y;
        v2 = v1;
```

```
        v1 = v;
};

x1WhenAB()
{
    /*  x1 = (ao1-r1e+1)*u1+(ao2+r1e)*u2+(t1e-(r1e-1)*t0e)*uc1+(t2e+r1e*t0e)*uc2-ao1*v
        -ao2*v2-(s1e-(r1e-1)*s0e)*y1-(s2e+r1e*s0e)*y2;  */

        x1 = (ao1-r1e+1)*u1+(ao2+r1e)*u2-ao1*v1-ao2*v2+t1e*uc1+t2e*uc2-s1e*y1-s2e*y2;
};

x1WhenC()
{
        x1 = (ao+1)*u1+t1e*uc1-ao*v1-s1e*y1;
};


main (init, uca, ua, wma,
        refvala, estima,adapta, reseta,perioda,ampa,a1a,a2a,b1a,b2a, r1a, s0a,
        s1a, s2a, t0a, t1a, t2a,storea, storeRSTa,ucuta,ya)


    int init,*storeRSTa,*refvala,*estima,*adapta,*reseta,*storea;
    double *uca, *ua, *wma,*r1a, *s0a, *s1a, *s2a, *t0a,*t1a, *t2a,
            *perioda,*ampa,*a1a,*a2a,*b1a,*b2a,*ucuta, *ya;
{

    RememberA0 ();
    SetUpA4 ();

    if (init) {
        InitGlobals();
    } else {
        if (refval) {
            Square();
        } else {
            uc = *uca;
            tick = 0;
        };

        y = *ya;

        v = t0e*uc - s0e*y + x1;

        Sat_v();

        *ucuta = uc;
        *ua = u;

        refval = *refvala;
        estim = *estima;
        adapt = *adapta;
        reset = *reseta;
```

```
        storeRST = *storeRSTa;
        amp = *ampa;
        period = *perioda;
        store = *storea;

        if (adapt) {
            Estimate();
            Checkz1();
            Est2Contr();
            if (reset) {
                ResetEst();
                ControllerA();
            }
        } else if (estim) {
                Estimate();
                Checkz1();
                if (reset) {
                    ResetEst();
                    ControllerA();
                };
        } else {
            UpdateStates();
            x1WhenAB();
        }
        if (storeRST) {
            r1e = *r1a;
            s0e = *s0a;
            s1e = *s1a;
            s2e = *s2a;
            t0e = *t0a;
            t1e = *t1a;
            t2e = *t2a;
        }


        if (store) {
            wm = *wma;
            wo = wm;
            ModUpdate();
            ControllerA();
            Est2Contr();
        }

        if ((reset) && (!estim) && (!adapt)){
            *ua = 0;
            ResetStates();
            ControllerA();
        }

        *a1a = th3;
        *a2a = th4;
        *b1a = th1;
        *b2a = th2;
```

```
        *r1a = r1;
        *s0a = s0;
        *s1a = s1;
        *s2a = s2;
        *t0a = t0;
        *t1a = t1;
        *t2a = t2;
    };

    RestoreA4 ();
}
```

Reglering av DC-servo med Adaptiv Regulator

Rikard Berglund
Erik Apelgren
Stefan Johansson
Ola Bernersson

VT-91

Innehåll

1 Inledning

Vår uppgift var att reglera ett DC-servo. På grund av att friktionen värmer upp oljan i lagren flyttar sig en av systemets poler när servot körs. Detta löste vi med en adaptiv regulator. M h a utsignal och styrsignal skattas processen. De skattade parametrarna används sedan för att med jämna mellanrum designa en optimal regulator. Ett blockschema över reglersystemet finns i figur 1.

2 Systembeskrivning

Servot som skulle regleras kan beskrivas med momentekvationen

$$J * \frac{d^2\phi}{dt^2} = M = k * i - D * \frac{d\phi}{dt}$$

där

J=tröghetsmomentet
ϕ=axelns vinkel
i=ankarströmmen
D=friktionen

Vi Laplacetransformerar ovanstående ekvation

$$\Phi(s) = \frac{k}{J * s^2 + D * s} * I(s) = \frac{k/J}{s(s + D/J)} * I(s)$$

Alltså kan systemets överföringsfunktion skrivas som

$$G(s) = \frac{b}{s(s + a)}$$

Vi gjorde antagandet att polen i origo ligger stilla, medan den andra polen rör sig bort från origo längs negativa reella axeln.

3 Estimering

Om systemet samplas blir den diskreta överföringsfunktionen

$$H(q) = \frac{b_1 * q + b_2}{q^2 + a_1 * q + a_2}$$

Med samplingstiden h ser koefficienterna ut på följande sätt:

$b_1 = \frac{b}{a^2} * (ah - 1 + e^{-ah})$
$b_2 = \frac{b}{a^2} * (1 - (1 + ah) * e^{-ah})$
$a_1 = -(1 + e^{-ah})$
$a_2 = e^{-ah}$

För att estimera processen konstruerade vi en indirekt skattare som bygger på minsta-kvadrat-metoden. Algoritm och formler för estimeringen finns i appendix A.

4 Design och Regulator

Det slutna systemet i figur 2 kan beskrivas m h a de två ekvationerna

$Ru = Tu_c - Sy$
$Ay = Bu$

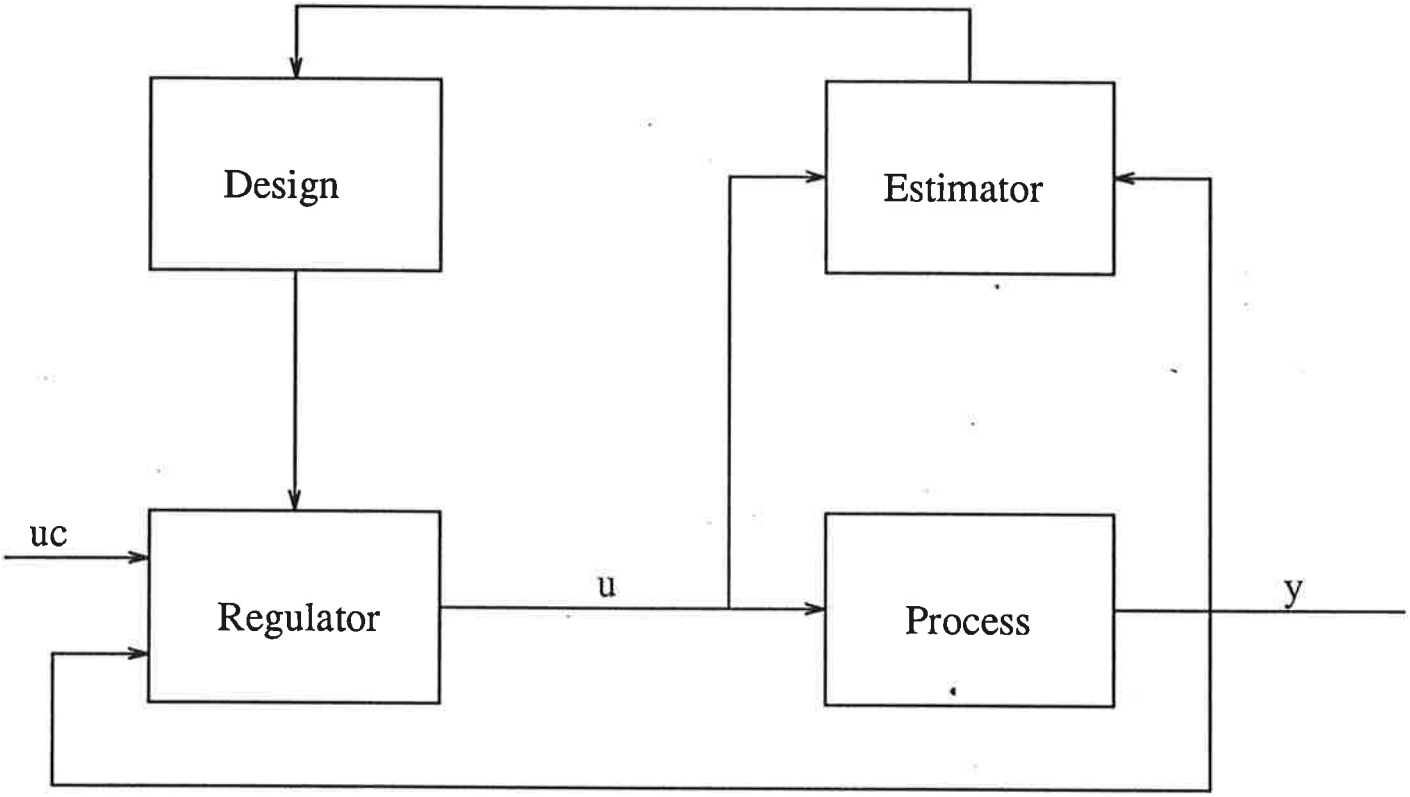
Genom att eliminera styrsignalen u i dessa ekvationer fås slutna systemet

$$\frac{BT}{AR + BS}$$

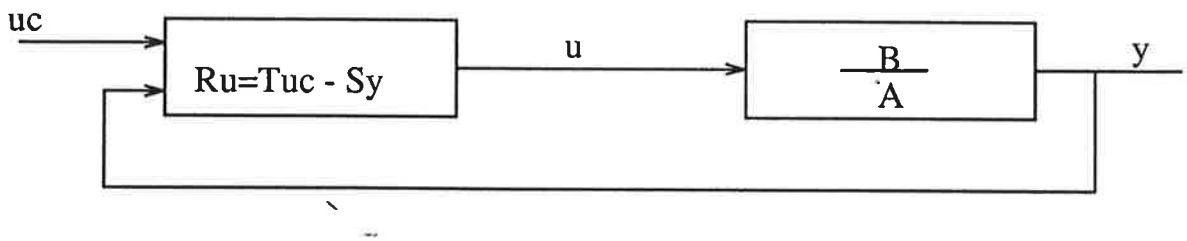
Om vi kräver att denna funktion skall vara lika med en önskad överföringsfunktion får vi

$$\frac{BT}{AR + BS} = \frac{B_m}{A_m}$$

där B_m och A_m är den önskade överföringsfunktionens täljare respektive nämnare. Om vi dessutom kräver att observerarpolynomet Ao skall vara en faktor i slutna systemets



figur 1 blockshema pa indirekt sjalvinstallande regulator



figur 2 blockshema pa slutna systemet

karakteristiska ekvation samt att inga nollställen förkortas får vi följande Diophantinska ekvation

$AR+BS=A_oA_m$

Med ett tidsintervall som är en multipel av regulatorns sampeltid görs en ny design i en speciell procedur. Designproceduren löser diophantinska ekvationen genom att ställa upp sylvestermatrisen för systemet och invertera denna. På detta sätt fås R och S-polynomen. I designproceduren ser vi också till att regulatorn får integratorverkan. T-polynomet beräknas så att den stationära förstärkningen blir lika med ett.

M h a de designade koefficienterna beräknas sedan styrsignalen ur ekvationen

$Ru=Tu_c - Sy$

5 Operatörskommunikation

I en särskild modul, Opcom, sker all kommunikation med användaren. Via en huvudmeny kan operatören bestämma vilken parameter som ska ändras och får därefter upp en undermeny i vilken parametervärden kan matas in. Menysystemet ser ut på följande sätt: (undermenyer markeras med kursiv stil)

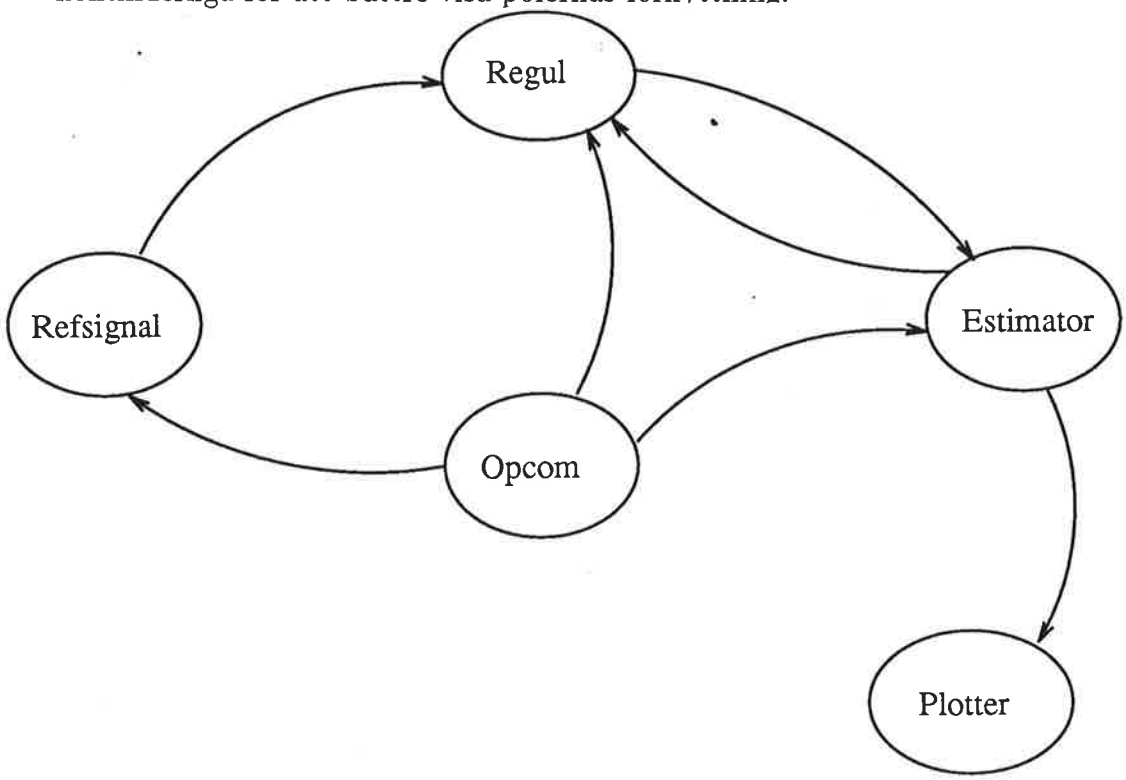
1. Sampeltid
   (a) *sampeltid för regulator*
   (b) *delningsfaktor för estimator*
2. Observerarpolynomets parametrar
   (a) *ange gradtal*
   (b) *observerarpolynomets koefficienter*
3. Modellens parametrar
   (a) *snabbhet (w0)*
   (b) *dämpning (z)*
4. Glömskefaktor
   (a) *lambda*
5. Referensvärdet
   (a) *maxläge och minläge*

(b) *periodtid i ms*

Har man valt en undermeny, men ångrar sig, finns möjlighet att återvända till huvudmenyn genom att slå en bokstav (vilken som helst) istället för en siffra. Oavsett i vilken meny man befinner sig är det alltid möjligt att stoppa processen genom att trycka på tangenten s. Regleringen av servot startas med tangenten p.

6 Implementering

Reglersystemet är uppbyggt av fem processer (se Figur 3). I Opcom matar man in bl.a sampeltid, modellens parametrar och observerarens parametrar. Opcom lägger sedan parametrarna i variabler som de olika processerna kan nå genom proceduranrop. Estimatorprocessen skattar processens överföringsfunktion och gör regulatordesignen genom att anropa proceduren Design. För att göra bytet av regulator så snabbt som möjligt använder vi två regulatoruppsättningar och byter till en ny regulator genom att ställa om två pekare. Plottermodulen har vi fått av reglerinstitutionen, plottningen görs i matlabs grafiska fönster. Från modulen Estimator lägger vi ut styrsignal, utsignal och polens parametrar för plottning. Innan plottningen räknas de diskreta polparametrarna om till kontinuerliga för att bättre visa polernas förflyttning.



figur 3 processgraf

7 Resultat

För att få stabil reglering av servot var vi tvungna att välja observerarens bandbredd stor och dämpningen liten. För låg sampelfrekvens leder också till instabil reglering. Glömskefaktorn bör väljas större än 0.995 annars börjar de skattade parametrarna att driva, vilket leder till att regulatorparametrarna ej blir stabila. När vi satte upp systembeskrivningen gjorde vi antagandet att ena polen rör sig bort från origo när växeloljan värms upp, detta visade sig helt fel, polen rör sig mot origo. Slutligen tycker vi att regulatorn fungerar som vi hade tänkt oss. Projektet har varit både intressant och lärorikt.

8 Appendix A

Estimatorns skattningsalgoritm:

$$\hat{\Theta}(t) = \hat{\Theta}(t-1) + K(t)\varepsilon(t)$$

$$\varepsilon(t) = y(t) - \varphi^T(t-1)\hat{\Theta}(t-1)$$

$$K(t) = P(t-1)\varphi(t-1)(\lambda + \varphi^T(t-1)P(t-1)\varphi(t-1))^{-1}$$

$$P(t) = (I - K(t)\varphi^T(t-1))P(t-1)/\lambda$$

där

$$\varphi^T(t-1) = (\ u(t-1)\ \ u(t-2)\ \ -y(t-1)\ \ -y(t-2)\ )$$

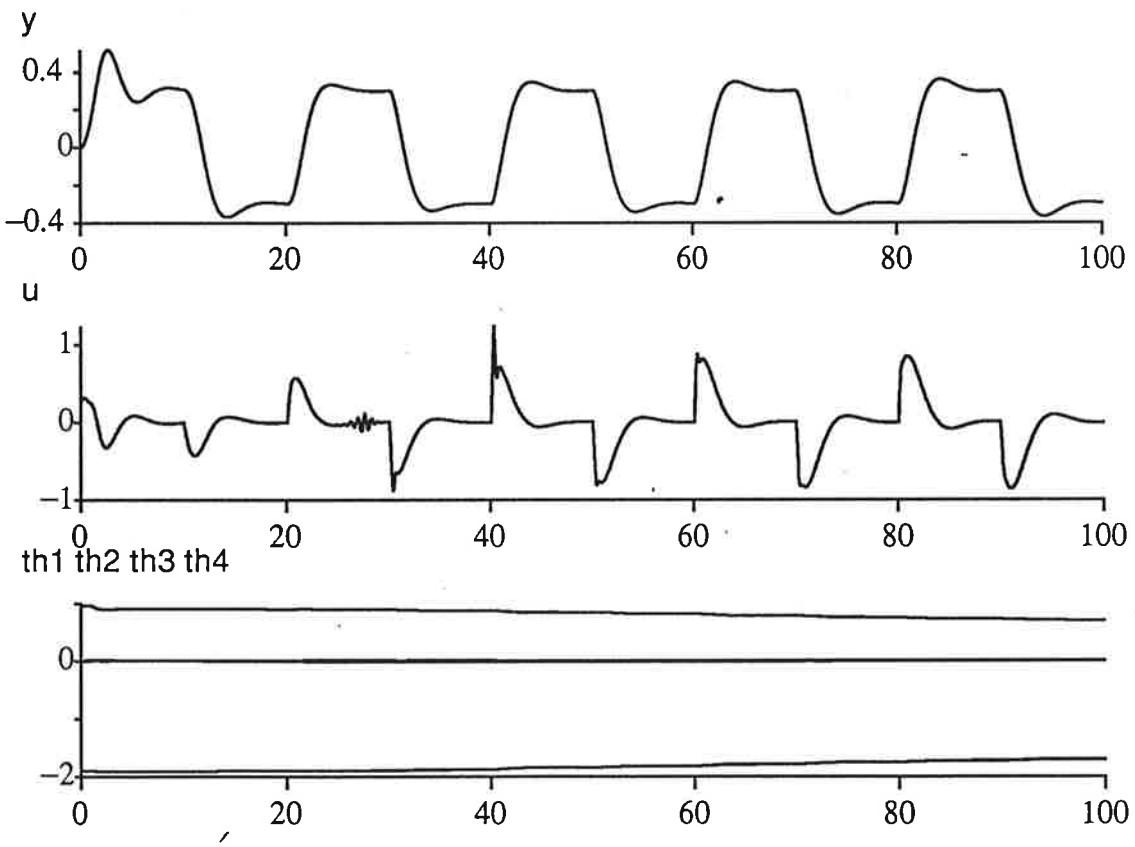
och

$$\Theta^T = (b_0\ \ b_1\ \ a_1\ \ a_2)$$

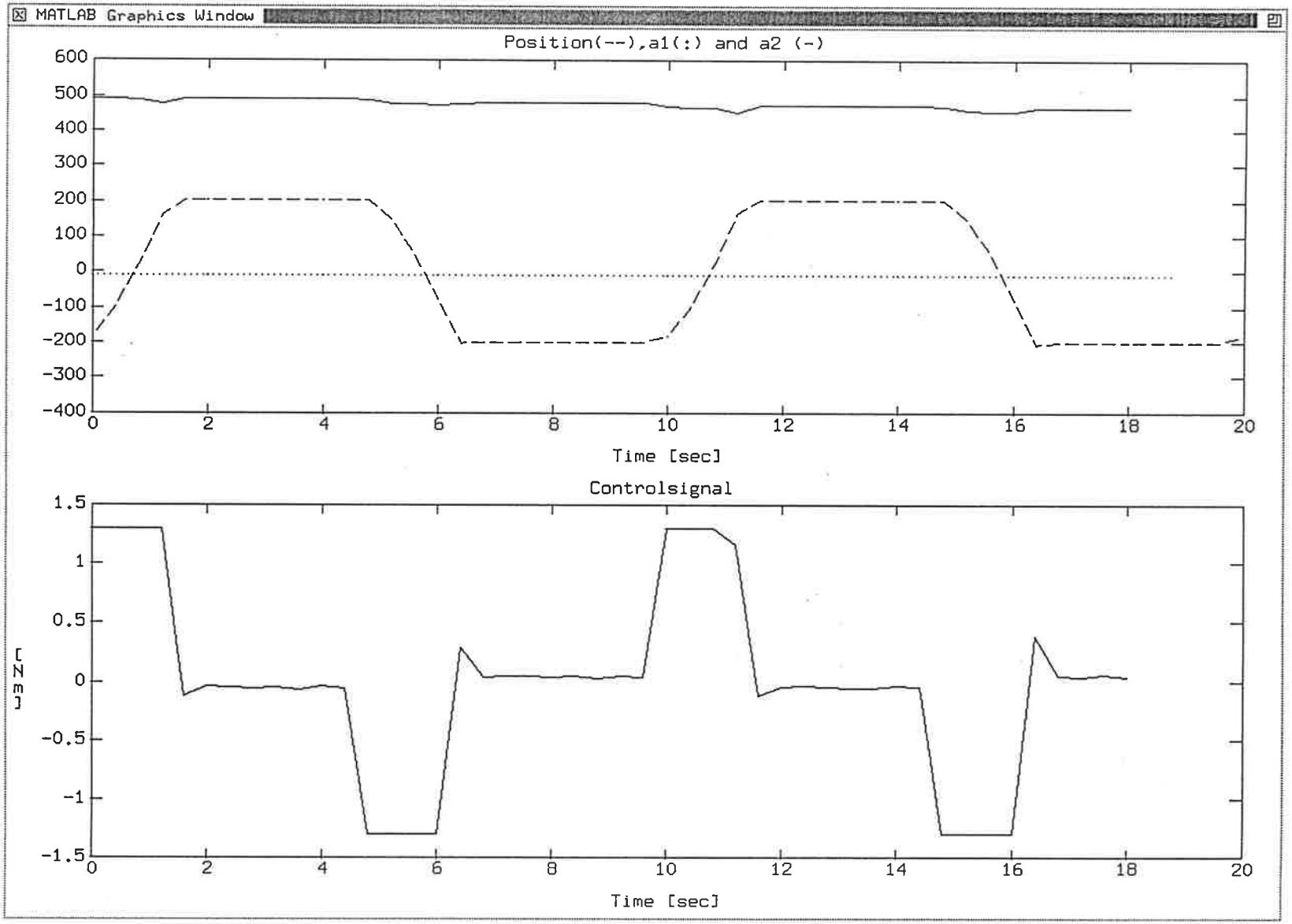
9 Appendix B

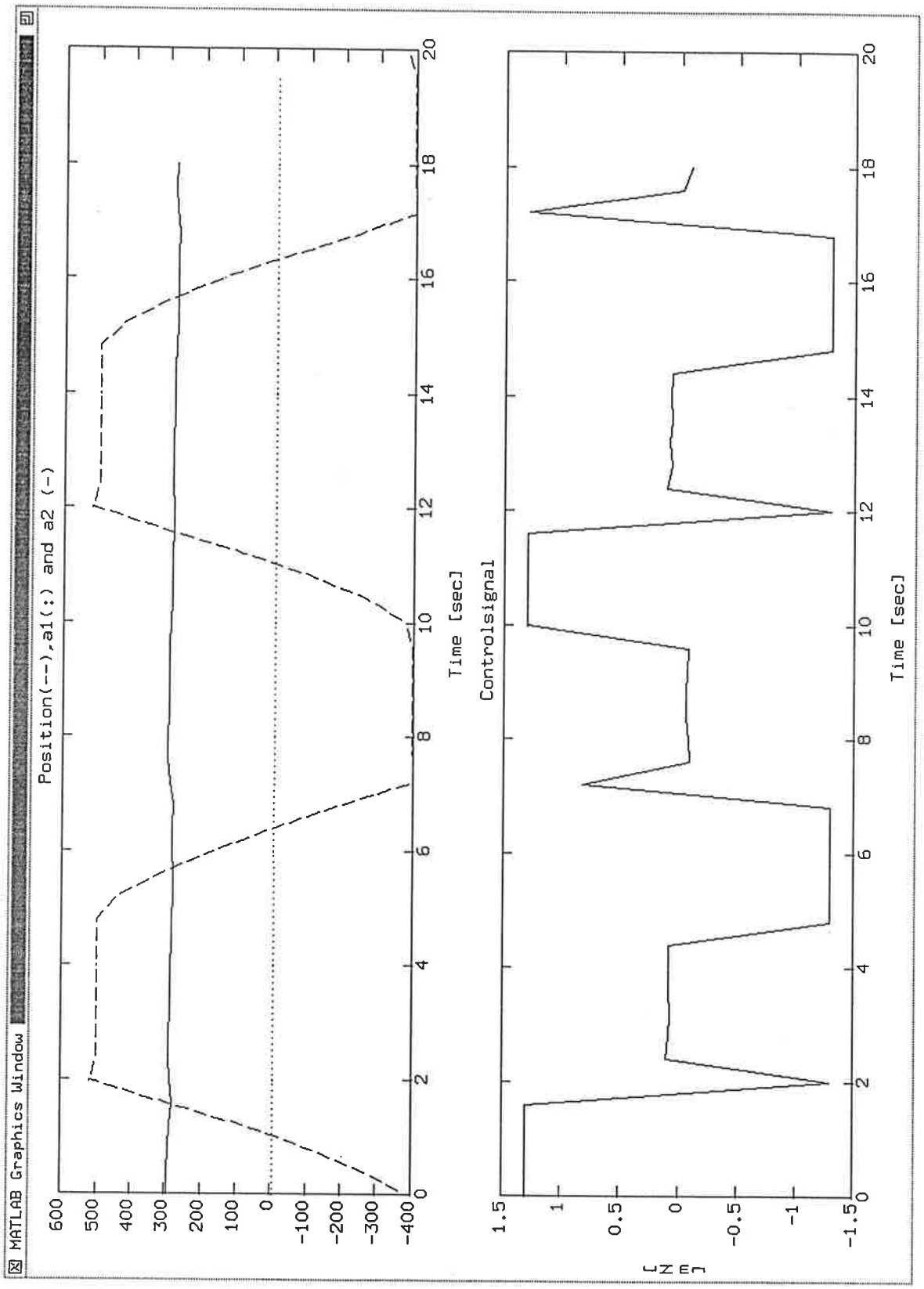
Plottningarna nedan visar en simulering av systemet m h a simnon . När vi simulerade systemet antog vi att polen flyttade sig exponentiellt längs negativa reella axeln. Observeraren är av andra ordningen och regulatorn är dimensionerad för integralverkan. Eftersom den skattade modellen har dåligt dämpade nollställen så förkortade vi inte dessa. Därför är styrsignalen fri från ringningar.

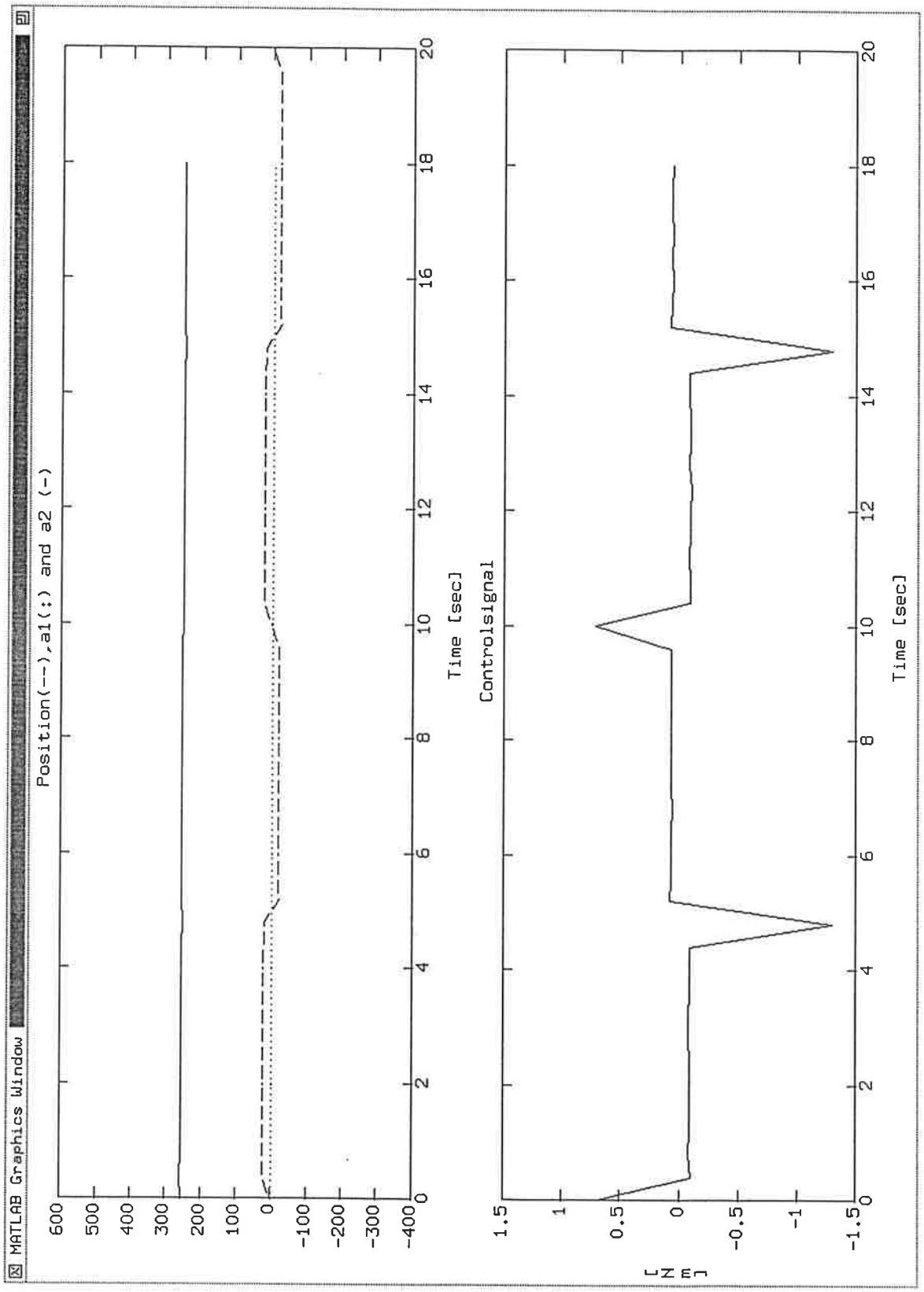
Simnonsimmulering av dc servo



140


MATLAB Graphics Window
Position(--),a1(:) and a2 (-)
600
500
400
300
200
100
0
-100
-200
-300
-400
0
2
4
6
8
10
12
14
16
18
20
Time [sec]
Controlsignal
1.5
1
0.5
0
-0.5
-1
-1.5
[ENE]
0
2
4
6
8
10
12
14
16
18
20
Time [sec]


MATLAB Graphics Window
Position(--),a1(:) and a2 (-)
600
500
400
300
200
100
0
-100
-200
-300
-400
0
2
4
6
8
10
12
14
16
18
20
Time [sec]
Controlsignal
1.5
1
0.5
0
-0.5
-1
-1.5
[N m]
0
2
4
6
8
10
12
14
16
18
20
Time [sec]


MATLAB Graphics Window
Position(--),a1(:) and a2 (-)
600
500
400
300
200
100
0
-100
-200
-300
-400
0
2
4
6
8
10
12
14
16
18
20
Time [sec]
Controlsignal
1.5
1
0.5
0
-0.5
-1
-1.5
[N m]
0
2
4
6
8
10
12
14
16
18
20
Time [sec]

```
IMPLEMENTATION MODULE Opcom;

FROM MathLib0 IMPORT exp,cos,sin,sqrt;

FROM Kernel IMPORT Init, CreateProcess,SetPriority;

FROM Monitors IMPORT InitMonitor,EnterMonitor,LeaveMonitor,
                     MonitorGate;

FROM Console IMPORT GetChar,GetString,CharAvailable,PutChar,
                    PutString,PutLn,Trap;

FROM Conversions IMPORT IntToString,CardToString,StringToCard,
                        StringToInt,StringToReal,RealToString;

FROM Strings IMPORT Compare;

FROM Storage IMPORT ALLOCATE, DEALLOCATE;

FROM Messages IMPORT MailBox,InitMailBox,SendMessage,ReceiveMessage;

CONST     Stack=10000;

TYPE      String=ARRAY [1..30] OF CHAR;

VAR        NewRs      :Refsig;
           NewEst     :EstPar;
           M,Mutex    :MonitorGate;
           z,w0       :REAL;
           Newh       :htype;
           OpBox      :MailBox;
           p          :POINTER TO INTEGER;


PROCEDURE GetRef(VAR Rs:Refsig);
BEGIN
   Rs:=NewRs;
END GetRef;

PROCEDURE GetEstPar(VAR Re:EstPar);
BEGIN
  EnterMonitor(M);
  Re:=NewEst;
  LeaveMonitor(M);
END GetEstPar;

PROCEDURE GetRegh(VAR h:htype);
BEGIN
  h:=Newh;
END GetRegh;

PROCEDURE Stopp();
BEGIN
   ReceiveMessage(OpBox,p);
END Stopp;

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

PROCEDURE Design();
VAR h,alfa,omega,beta,gamma:REAL;
    Am,Bm:cpoly;
BEGIN
   h:=FLOAT(Newh.h)/1000.0;
   alfa:=exp(-z*w0*h);
   omega:=w0*(sqrt(1.0-z*z));
   beta:=cos(omega*h);
   gamma:=sin(omega*h);
   Am.degree:=2;
   Am.coeffs[0]:=1.0;
   Am.coeffs[1]:=-2.0*alfa*beta;
   Am.coeffs[2]:=alfa*alfa;
   Bm.degree:=1;
   Bm.coeffs[0]:=1.0-alfa*(beta+z*w0*gamma/omega);
   Bm.coeffs[1]:=alfa*alfa+alfa*(z*w0*gamma/omega-beta);
   EnterMonitor(M);
   NewEst.Am:=Am;
   NewEst.Bm:=Bm;
   LeaveMonitor(M);
END Design;


PROCEDURE Stop(VAR Op:String;VAR Ok:BOOLEAN);
VAR s,p:INTEGER;
BEGIN
   Ok:=TRUE;
   s:=Compare('s',Op);
   p:=Compare('p',Op);
   IF s=0 THEN
      Newh.Stop:=TRUE;Ok:=FALSE;
   ELSIF p=0 THEN
      Newh.Stop:=FALSE;Ok:=FALSE;SendMessage(OpBox,p);
   END; (*IF*)
END Stop;

PROCEDURE Meny1();
VAR Op:String; Ok:BOOLEAN; op:INTEGER;temp:CARDINAL;
BEGIN
   PutString('1. Sampeltid for regulator');PutLn;
   PutString('2. Delnings faktor for estimator');PutLn;
   GetString(Op); Stop(Op,Ok);op:=StringToInt(Op);
   CASE op OF
        1 :PutString('hreg= ');
           GetString(Op);Stop(Op,Ok);
           temp:=StringToCard(Op);Design();
           IF Ok AND (temp<10000)THEN
              Newh.h:=temp;
              PutLn;PutLn;
           END;(*IF*)
         2 :PutString('del= ');
            GetString(Op);Stop(Op,Ok);
            op:=StringToInt(Op);
           IF Ok AND (op>0) THEN
              Newh.delf:=op;
              PutLn;PutLn;
           END;(*IF*)
   ELSE
      PutLn;
   END;(*CASE*)
END Meny1;

PROCEDURE Meny2();
VAR Op:String; Ok:BOOLEAN; grad,i:CARDINAL;
    Ao:cpoly;temp:REAL;
BEGIN
   PutString(' Ange gradtalet ');GetString(Op);Stop(Op,Ok);
   PutLn;
   grad:=StringToCard(Op);
   IF Ok AND (grad<50) THEN
```

```
      Ao.degree:=grad;
      i:=0;
      temp:=1.0;
      WHILE Ok AND (grad<>i) AND (temp<100.0) DO
         i:=i+1;
         PutString('koefficient framfor s');CardToString(Op,grad-i,2);
         PutString(Op);PutLn;
         GetString(Op);Stop(Op,Ok);temp:=StringToReal(Op);
         IF Ok AND (temp<100.0) THEN
            Ao.coeffs[i]:=temp;
         END;(*IF*)
      END;(*WHILE*)
      EnterMonitor(M);
        NewEst.Ao:=Ao;
      LeaveMonitor(M);
   END;(*IF grad*)
END Meny2;

PROCEDURE Meny3();
VAR Op:String; Ok:BOOLEAN; op:INTEGER;temp:REAL;
BEGIN
   PutString('1.w0');PutLn;
   PutString('2. z');PutLn;
   GetString(Op);Stop(Op,Ok);op:=StringToInt(Op);
   CASE op OF
         1 :PutString('w0=  ');
            GetString(Op);Stop(Op,Ok);PutLn;temp:=StringToReal(Op);
            IF Ok AND (temp<1000.0)THEN
               w0:=temp;Design();
             END|
         2 :PutString('z=  ');
            GetString(Op);
            Stop(Op,Ok);PutLn;
            IF Ok THEN
               z:=StringToReal(Op);
               IF z >= 1.0 THEN
                  PutString(' Felaktigt val -1< z < 1 ');PutLn;PutLn;
               ELSE
                  Design();
               END;(*IF*)
             END;
    ELSE
       PutLn;
    END; (*CASE*)
END Meny3;

PROCEDURE Meny4();
VAR Op:String; Ok:BOOLEAN;
     x:REAL;
BEGIN
   PutString('lambda= ');
   GetString(Op);Stop(Op,Ok);PutLn;
   IF Ok THEN
      x:=StringToReal(Op);
      IF  x <= 1.0 THEN
         EnterMonitor(M);
         NewEst.lam:=x;
         LeaveMonitor(M);
      ELSE
         PutString('felaktigt val av lambda, (lambda < 1)');PutLn;
      END;(*IF x > 1.0*)
   END; (*IF Ok*)
END Meny4;
```

```
PROCEDURE Meny5();
VAR Op:String; Ok:BOOLEAN;op:INTEGER;temp:REAL;temp2:CARDINAL;
BEGIN
   PutString('1. Maxlage och Minlage');PutLn;
   PutString('2. Periodtid T i ms');PutLn;
   GetString(Op);Stop(Op,Ok);op:=StringToInt(Op);
   CASE op OF
        1 : PutString(' Maxlage= ');
            GetString(Op);Stop(Op,Ok);PutLn;temp:=StringToReal(Op);
            IF Ok AND (temp<100000.0) THEN
                NewRs.Ucmax:=temp;
            END;(*IF*)
            PutString(' Minlage= ');
            GetString(Op);Stop(Op,Ok);PutLn;temp:=StringToReal(Op);
            IF Ok AND (temp<100000.0) THEN
                NewRs.Ucmin:=temp;
            END|(*IF*)
         2: PutString(' T= ');
            GetString(Op);Stop(Op,Ok);PutLn;temp2:=StringToCard(Op);
            IF Ok AND (temp2<1000000) THEN
                NewRs.Periodtid:=temp2;
            END;(*IF*)
   ELSE
      PutLn;
   END;(*CASE*)
END Meny5;


PROCEDURE Meny;
VAR Op:String;Ok:BOOLEAN;op:INTEGER;
BEGIN
   SetPriority(950);
   LOOP
      PutString('Valj parameter (1,2,3,4) eller s=stopp och p= start');
      PutLn;
      PutString('1.Sampeltid ');PutLn;
      PutString('2.Observerar polynomets parametrar');PutLn;
      PutString('3.Modellens parametrar');PutLn;
      PutString('4.Glomskefaktor');PutLn;
      PutString('5.Referensvardet');PutLn;PutLn;
      GetString(Op);Stop(Op,Ok);op:=StringToInt(Op);
      IF Ok THEN
        CASE op OF
             1 :Meny1()|
             2 :Meny2()|
             3 :Meny3()|
             4 :Meny4()|
             5 :Meny5();
        ELSE
             PutString('Felaktigt val');PutLn;
        END;(*CASE*)
      END;(*IF*)
   END; (*LOOP*)
END Meny;

PROCEDURE InitOpcom();
BEGIN
  NEW(p);
  p^:=1;
  InitMailBox(OpBox,1,'OpBox');
  InitMonitor(M,'M');
  Newh.h:=20;z:=0.7;w0:=20.0;Design();
  NewEst.Ao.coeffs[0]:=1.0; NewEst.Ao.coeffs[1]:=0.1;NewEst.Ao.coeffs[2]:=0.1;
  NewEst.Ao.degree:=2;
```

```
  Newh.Stop:=TRUE;Newh.delf:=3;NewRs.Periodtid:=10000;
  NewRs.Ucmax:=100.0;NewRs.Ucmin:=-100.0;
  NewEst.lam:=0.995;
  CreateProcess(Meny,Stack,'Meny');
END InitOpcom;
END Opcom.
```

```
DEFINITION MODULE Regul;

FROM Opcom IMPORT cpoly;

FROM Monitors IMPORT MonitorGate;

EXPORT QUALIFIED InitRegul,GetLsPar,Design,Vector,RegParPek,Protect,Operation,
                 Swap,Box;
TYPE Vector=ARRAY[0..3] OF REAL;

     LSType=RECORD
              FiMon   :MonitorGate;
              FiVektor:Vector;
              ynew    :REAL;
            END;

      RegParPek= POINTER TO RegParData;
              RegParData = RECORD
                               R,S,T  :cpoly;
                           END;
      Operation=(In,Ut);

PROCEDURE Swap(VAR P:RegParPek);

PROCEDURE Design(Am,Bm,Ao,A,B:cpoly; VAR RegPar:RegParPek);

PROCEDURE GetLsPar(VAR Fi:Vector; VAR y:REAL);

PROCEDURE Protect(Op:Operation);

PROCEDURE Box();

PROCEDURE InitRegul();

END Regul.
```

148

```
IMPLEMENTATION MODULE Regul;

FROM Kernel IMPORT Time,CurrentTime,IncTime,WaitUntil,CreateProcess;

FROM Calc IMPORT PolyMult,Diophantine;

FROM Refsignal IMPORT Ref;

FROM ServoIO IMPORT RefOut,JointType,Init,AnalogOutputs,ReadMode,absPos,
                    ResetServo,RefScale;

FROM Monitors IMPORT MonitorGate,EnterMonitor,LeaveMonitor,InitMonitor;

FROM Semaphores IMPORT InitSem,Wait,Signal,Semaphore;

FROM Kernel IMPORT SetPriority;

FROM Opcom IMPORT cpoly,GetRegh,htype,Stopp;

FROM AnalogIO IMPORT ADin,CardType,ChannelType;

FROM Storage IMPORT ALLOCATE, DEALLOCATE;

FROM Messages IMPORT MailBox,InitMailBox,SendMessage,ReceiveMessage;

CONST Stack =10000;

VAR LsPar           :LSType;
    AktRegPek       :RegParPek;
    u,y,uc          :Vector;
    Mutex           :Semaphore;
    Op              :ARRAY [1..30] OF CHAR;
    Int             :cpoly;
    p               :POINTER TO INTEGER;
    b               :MailBox;

PROCEDURE Box();
BEGIN
  ReceiveMessage(b,p);
END Box;

PROCEDURE Swap(VAR P:RegParPek);
VAR hpek:RegParPek;
BEGIN
   hpek:=AktRegPek;
   AktRegPek:=P;
   P:=hpek;
END Swap;

PROCEDURE Protect(Op:Operation);
BEGIN
   CASE Op OF
      In: Wait(Mutex)|
      Ut: Signal(Mutex)|
   END;
END Protect;

PROCEDURE Design( Am,Bm,Ao,A,B:cpoly; VAR RegPar:RegParPek);
VAR   i             :INTEGER;
      Am1,B1        :REAL;
      r,s,t,Ac      :cpoly;
      Rtemp,Atemp   :cpoly;
BEGIN
      PolyMult(Am,Ao,Ac);
      PolyMult(A,Int,Atemp);
      Diophantine(Atemp,B,Ac,Rtemp,s);
      PolyMult(Rtemp,Int,r);
      Am1:=0.0;
      B1:=0.0;
      FOR i:=0 TO Am.degree DO
        Am1:=Am1+Am.coeffs[i];
      END;
      FOR i:=0 TO B.degree DO
        B1:=B1+B.coeffs[i];
      END;
      FOR i:=0 TO Ao.degree DO
        t.coeffs[i]:=Am1/B1*Ao.coeffs[i];
      END;
      t.degree:=Ao.degree;
      WITH RegPar^ DO
         R:=r;
         S:=s;
         T:=t;
      END; (*WITH*)
END Design;

PROCEDURE UpDateFi(u,y:REAL);
VAR i:INTEGER;
BEGIN
   EnterMonitor(LsPar.FiMon);
   WITH LsPar DO
     FiVektor[1]:=FiVektor[0];
     FiVektor[0]:=-ynew;
     FiVektor[3]:=FiVektor[2];
     FiVektor[2]:=u;
     ynew:=y;
   END;(*WITH*)
   LeaveMonitor(LsPar.FiMon);
END UpDateFi;

PROCEDURE Shift(VAR X:Vector;VAR xnew:REAL);
VAR i:INTEGER;
BEGIN
   FOR i:=3 TO 1 BY -1 DO
     X[i]:=X[i-1];
   END; (*FOR*)
   X[0]:=xnew;
END Shift;

PROCEDURE GetLsPar(VAR Fi:Vector;VAR y:REAL);
BEGIN
   EnterMonitor(LsPar.FiMon);
   WITH LsPar DO
      Fi:=FiVektor;
      y:=ynew;
   END; (*WITH*)
   LeaveMonitor(LsPar.FiMon);
END GetLsPar;

PROCEDURE Regulator;
VAR x,v,z,ynew,utemp,ucnew,unew:REAL;
    h                   :htype;
    t                   :Time;
    k,i                 :INTEGER;
BEGIN
   SetPriority(30);
   k:=0;
   LOOP
```

```
      GetRegh(h);CurrentTime(t);IncTime(t,h.h);
      ynew:=absPos(0);
      ucnew:=Ref();
      Shift(y,ynew);
      Shift(uc,ucnew);
      x:=0.0;v:=0.0;z:=0.0;
      Wait(Mutex);
      WITH AktRegPek^ DO
         FOR i:=0 TO T.degree DO
            x:=x+uc[i]*T.coeffs[i];
         END;
         FOR i:=0 TO S.degree DO
            v:=v+y[i]*S.coeffs[i];
         END;
         FOR i:=1 TO R.degree DO
            z:=z+u[i]*R.coeffs[i];
         END;
      END; (*WITH*)
      utemp:=x-v-z;
      IF utemp>1.3 THEN
         unew:=1.3;
      ELSIF utemp < -1.3 THEN
         unew:=-1.3;
      ELSE
         unew:=utemp;
      END; (*IF*)
      IF h.Stop THEN
         RefOut(0,0.0);
         Stopp();
      END;(*IF*)
      RefOut(0,unew);
      Signal(Mutex);
      UpDateFi(u[0],ynew);
      Shift(u,unew);
      WaitUntil(t);
      IF k=h.delf THEN
         k:=0;
         SendMessage(b,p);
      ELSE
         k:=k+1;
      END;(*IF*)
   END; (*LOOP*)
END Regulator;

PROCEDURE InitRegul();
VAR  i :CARDINAL;
BEGIN
   NEW(p);
   p^:=1;
   InitMailBox(b,1,'b');
   Int.degree:=1;
   Int.coeffs[0]:=1.0;
   Int.coeffs[1]:=-1.0;
   NEW(AktRegPek);
   WITH AktRegPek^ DO
      T.degree:=2; T.coeffs[0]:=1.0; T.coeffs[1]:=1.0; T.coeffs[2]:=0.5;
      S.degree:=2; R.coeffs[0]:=1.0; R.coeffs[1]:=1.0; R.coeffs[2]:=1.0;
      R.degree:=2; S.coeffs[0]:=10.0; S.coeffs[1]:=10.0; S.coeffs[2]:=10.0;
   END;(*WITH*)
   InitMonitor(LsPar.FiMon,'LsPar.FiMon');
   Init(digital);
   ResetServo(0);
   RefScale(0, 1.3);(* Max vridmoment *)
   InitSem(Mutex,1,'Mutex');
   FOR i:=0 TO 3 DO
      LsPar.FiVektor[i]:=0.0;
      u[i]:=0.0;
      y[i]:=0.0;
      uc[i]:=0.0;
   END;(*FOR*)
   LsPar.ynew:=0.0;
   CreateProcess(Regulator,Stack,'Regulator');
END InitRegul;

END Regul.
```

```
DEFINITION MODULE Estimator;

EXPORT QUALIFIED InitEstim;

TYPE Matrix= ARRAY [0..3],[0..3] OF REAL;

PROCEDURE InitEstim();

END Estimator.
```

```
IMPLEMENTATION MODULE Estimator;

FROM Kernel IMPORT Time,CurrentTime,IncTime,WaitUntil,CreateProcess,
                   SetPriority;

FROM Opcom IMPORT cpoly,EstPar,GetEstPar;

FROM Regul IMPORT GetLsPar,Design,Vector,RegParPek,Protect,Operation,Swap,
                  Box;

FROM Storage IMPORT ALLOCATE, DEALLOCATE;

IMPORT Plotter;
FROM Plotter IMPORT SendToPlotter;

CONST   Stack=10000;
VAR   p                 :Matrix;
      th0,th1,th2,th3   :REAL;
      RegPek            :RegParPek;


PROCEDURE Estim;
VAR ahat,bhat          :cpoly;
    pf,K,Fi            :Vector;
    i,k,l              :CARDINAL;
    den,temp,y,eps     :REAL;
    tid                :Time;
    Par                :EstPar;
BEGIN
   SetPriority(30);ahat.degree:=2;bhat.degree:=1;ahat.coeffs[0]:=1.0;
   l:=4;
   LOOP
      Box();(*Stoppar Estimatorn nar Regul stoppar*)
      GetEstPar(Par);
      GetLsPar(Fi,y);
      pf[0]:=p[0,0]*Fi[0]+p[0,1]*Fi[1]+p[0,2]*Fi[2]+p[0,3]*Fi[3];
      pf[1]:=p[0,1]*Fi[0]+p[1,1]*Fi[1]+p[1,2]*Fi[2]+p[1,3]*Fi[3];
      pf[2]:=p[0,2]*Fi[0]+p[1,2]*Fi[1]+p[2,2]*Fi[2]+p[2,3]*Fi[3];
      pf[3]:=p[0,3]*Fi[0]+p[1,3]*Fi[1]+p[2,3]*Fi[2]+p[3,3]*Fi[3];
      temp:=0.0;
      FOR i:= 0 TO 3 DO
         temp:=temp+pf[i]*Fi[i];
      END;
      den:=Par.lam+temp;

      eps:=y-Fi[0]*th0-Fi[1]*th1-Fi[2]*th2-Fi[3]*th3;

      FOR i:=0 TO 3 DO
         K[i]:=pf[i]/den;
      END;

      th0:=th0+K[0]*eps;
      th1:=th1+K[1]*eps;
      th2:=th2+K[2]*eps;
      th3:=th3+K[3]*eps;
      FOR i:=0 TO 3 DO
        FOR k:=i TO 3 DO
          p[i,k]:=(p[i,k]-pf[i]*K[k])/Par.lam;
        END;
      END;
      bhat.coeffs[0]:=th2; bhat.coeffs[1]:=th3;
      ahat.coeffs[1]:=th0; ahat.coeffs[2]:=th1;
      Design(Par.Am,Par.Bm,Par.Ao,ahat,bhat,RegPek);
      IF l=5 THEN
         SendToPlotter(y,ahat.coeffs[1],ahat.coeffs[2],Fi[2]);
         l:=0;
      END;
      l:=l+1;
      Protect(In);
      Swap(RegPek);
      Protect(Ut);
   END; (*LOOP*)
END Estim;


PROCEDURE InitEstim();
VAR i,k      :INTEGER;
BEGIN
   th0:=2.0; th1:=-2.0; th2:=0.01; th3:=0.01;
   NEW (RegPek);
   FOR i:=0 TO 3 DO
      FOR k:=i TO 3 DO
         IF i=k THEN
            p[i,k]:=10000.0;
         ELSE
            p[i,k]:=0.0
         END;(*IF*)
      END;
   END;
   CreateProcess(Estim,Stack,'Estim');
   Plotter.Init;
END  InitEstim;

END Estimator.
```

```
DEFINITION MODULE Calc;

FROM Opcom IMPORT cpoly,maxdegree;

EXPORT QUALIFIED PolyMult,Diophantine;

TYPE  mat = ARRAY[1..maxdegree+1],[1..maxdegree+2] OF REAL;
      vec = ARRAY[1..maxdegree+1] OF REAL;


PROCEDURE PolyMult(a,b:cpoly;VAR c:cpoly);

PROCEDURE Diophantine(a, b, c :cpoly; VAR xx,yy:cpoly);

END Calc.
```

```
IMPLEMENTATION MODULE Calc;

FROM Opcom IMPORT cpoly;

VAR
 M    : mat;
 xxyy : vec;


PROCEDURE PolyMult(a,b:cpoly;VAR c:cpoly);
VAR  Temp  :cpoly;
     i,k,l :INTEGER;
BEGIN
   l:=a.degree+b.degree;
   FOR k:= 0 TO l DO
      Temp.coeffs[k]:=0.0;
   END;
   FOR k:= 0 TO a.degree DO
      FOR i:= 0 TO b.degree DO
         Temp.coeffs[k+i]:=Temp.coeffs[k+i]+a.coeffs[k] * b.coeffs[i];
      END;
   END;
   c.degree:=l;
   c.coeffs:=Temp.coeffs;
END PolyMult;



PROCEDURE gauss(n:CARDINAL; a:mat; VAR x:vec; VAR ok:BOOLEAN);
(* Gausselimination for solving Ax=B  (a= (A B]) *)
VAR amax,akk,help:REAL;
    i,j,k,ind:CARDINAL;
BEGIN
  ok := TRUE;
  FOR k:=1 TO n DO
    amax:=-1.0;
    FOR i:=k TO n DO            (* Search for largest element in column *)
      IF ABS(a[i,k])>amax THEN
        amax := ABS(a[i,k]);
        ind  := i;
      END;
    END;
    IF ind>k THEN               (* Pivoting *)
      FOR j:=k TO n+1 DO
        help     := a[ind,j];
        a[ind,j] := a[k,j];
        a[k,j]   := help;
      END;
    END;
    IF amax<1.0E-6 THEN         (* singular matrix *)
      ok := FALSE;
      RETURN;
    END;

    akk:=a[k,k];                (* elimination *)
    FOR j:=k TO n+1 DO
      a[k,j]:=a[k,j]/akk;
    END;
    FOR i:=k+1 TO n DO
      FOR j:=k+1 TO n+1 DO
        a[i,j]:=a[i,j]-a[i,k]*a[k,j];
      END;
    END;
 END;

  FOR i:=n TO 1 BY -1 DO
    x[i]:=a[i,n+1];
    FOR k:=i+1 TO n DO
      x[i]:=x[i]-a[i,k]*x[k];
    END;
  END;
END gauss;



PROCEDURE Diophantine(a, b, c :cpoly; VAR xx,yy:cpoly);
VAR ic,ir : INTEGER;
   align  : INTEGER;
    ok    : BOOLEAN;
BEGIN
  (* make Sylvester Matrix *)
  xx.degree := c.degree - a.degree; (* degR *)
  yy.degree := a.degree - 1;        (* degS *)

  FOR ir:=0 TO c.degree DO
    FOR ic:=0 TO c.degree DO
      M[ir+1,ic+1] := 0.0;
    END;
    M[ir+1,c.degree+2] := c.coeffs[ir];
  END;

  FOR ic:=0 TO xx.degree DO
    FOR ir:=0 TO a.degree DO
      M[ir+1+ic,ic+1] := a.coeffs[ir];
    END;
  END;

  align := 1+a.degree-b.degree+xx.degree-yy.degree;
  FOR ic:=0 TO yy.degree DO
    FOR ir:=0 TO b.degree DO
      M[ir+ic+align,xx.degree+2+ic] := b.coeffs[ir];
    END;
  END;

   (* solve linear equations *)
  gauss(c.degree+1, M, xxyy, ok);
  IF ok THEN
    FOR ic:=0 TO xx.degree DO
      xx.coeffs[ic] := xxyy[ic+1];
    END;
    FOR ic:=0 TO yy.degree DO
      yy.coeffs[ic] := xxyy[xx.degree+2+ic];
    END;
  ELSE    .
  END;
END Diophantine;

END Calc.
```

154

```
DEFINITION MODULE Refsignal;

EXPORT QUALIFIED InitRefsignal,Ref;

PROCEDURE InitRefsignal();

PROCEDURE Ref():REAL;

END Refsignal.
```

```
IMPLEMENTATION MODULE Refsignal;

FROM Kernel IMPORT SetPriority,CreateProcess,Time,WaitTime;

FROM Monitors IMPORT InitMonitor,EnterMonitor,LeaveMonitor,
                     MonitorGate;

FROM Opcom IMPORT GetRef,Refsig;

CONST Stack=10000;

VAR NyUc:REAL;
    t:REAL;
    R:Refsig;
    M:MonitorGate;

PROCEDURE Ref():REAL;
VAR k:REAL;
BEGIN
   k:=NyUc;
   RETURN k;
END Ref;

PROCEDURE Signal;
BEGIN
   SetPriority(40);
   LOOP
      GetRef(R);
      IF R.Periodtid > 0 THEN
         EnterMonitor(M);
         NyUc:=R.Ucmax;
         t:=FLOAT(R.Periodtid)/2.0;
         LeaveMonitor(M);
         WaitTime(TRUNC(t));
         EnterMonitor(M);
         NyUc:=R.Ucmin;
         LeaveMonitor(M);
         WaitTime(TRUNC(t));
       ELSE
         EnterMonitor(M);
         NyUc:=R.Ucmax;
         LeaveMonitor(M);
       END; (*IF*)
    END;  (*LOOP*)
END Signal;

PROCEDURE InitRefsignal();
BEGIN
   InitMonitor(M,'M');
   CreateProcess(Signal,Stack,'Signal');
END InitRefsignal;
END Refsignal.
```

```
DEFINITION MODULE Plotter;

PROCEDURE SendToPlotter(Position,
                        Velocity,
                        Referense,
                        Control  :REAL);

PROCEDURE Init;

PROCEDURE Stop;

END Plotter.
```

```
IMPLEMENTATION MODULE Plotter;

IMPORT Kernel;
FROM Kernel IMPORT     CreateProcess, Terminate, SetPriority,
                       CurrentTime, Time;
IMPORT Monitors;
FROM Monitors IMPORT   InitMonitor, EnterMonitor, LeaveMonitor,
                       InitEvent, Await, Cause, MonitorGate, MonitorEvent;
IMPORT Semaphores;
FROM Semaphores IMPORT InitSem, Wait, Signal, Semaphore;

IMPORT MatComm;
FROM MatComm IMPORT    OpenSocket, CloseSocket, Send, Socket,
                       ErrorType, DataType;

FROM Console IMPORT    PutString, PutLn;



TYPE MatrixType=ARRAY[1..5],[1..5] OF REAL;

     MonitorType=RECORD
                   Event   : MonitorEvent;
                   Mutex   : MonitorGate;
                   DoWrite: BOOLEAN;
                   Buffert: MatrixType;
                   Index   : INTEGER;
                 END;


VAR M            : MonitorType;
    StopSemaphore: Semaphore;
    Term         : BOOLEAN;
    initiated    : BOOLEAN;
    Sock         : Socket;



(*Monitor*) PROCEDURE SendToPlotter(Position,
                                   Velocity,
                                   Referense,
                                   Control:REAL);

VAR t: Time;

BEGIN
  IF NOT initiated THEN Init; END;
  WITH M DO
    EnterMonitor(Mutex);
      IF DoWrite THEN
        CurrentTime(t);
        Index:=Index+1;
        Buffert[Index, 1]:= Position;
        Buffert[Index, 2]:= Velocity;
        Buffert[Index, 3]:= Referense;
        Buffert[Index, 4]:= Control;
        Buffert[Index, 5]:=VAL(REAL, t.lo);
        IF Index=5 THEN
          Index:=0;
          DoWrite:=FALSE;
          Cause(Event);
        END;
      END;
    LeaveMonitor(Mutex);
  END;
END SendToPlotter;


(*Monitor*) PROCEDURE GetMatrix(VAR Matrix: MatrixType);

BEGIN
  WITH M DO
    EnterMonitor(Mutex);
      WHILE DoWrite DO
        Await(Event);
      END;
      Matrix:=Buffert;
      DoWrite:=TRUE;
    LeaveMonitor(Mutex);
  END;
END GetMatrix;


PROCEDURE SendMatrix(Matrix: MatrixType);

VAR Did: ErrorType;

BEGIN
  REPEAT
    Did:=Send(Sock,5,5,real,Matrix);
    IF ( Did=Closed ) THEN
      OpenSocket(Sock, 'Plot');
    END;
  UNTIL ( Did=OK );
END SendMatrix;

(*Process*) PROCEDURE Plot;

VAR Matrix: MatrixType;

BEGIN
  SetPriority(50);
  OpenSocket(Sock, 'Plot');
  LOOP
    GetMatrix(Matrix);
    SendMatrix(Matrix);
   (* Wait(StopSemaphore);*)
     (* IF Term THEN*)
       (* Terminate;*)
     (* END;*)
   (* Signal(StopSemaphore);*)
  END;
END Plot;



PROCEDURE Stop();

BEGIN
  Wait(StopSemaphore);
    Term:=TRUE;
  Signal(StopSemaphore);
  CloseSocket(Sock);
END Stop;
```

```
reg = vmeio('vme','Plot','open');

t1min=0;
t2min=0;
t1max=20;
t2max=20;
minvalue1=-400;
maxvalue1=600;
minvalue2=-1.5;
maxvalue2=1.5;
plotinterval=3;
clg;
tmax=t1max;
last=[0 0 0 0 0];

% 1 refers to position- and velocityplot.
% 2 refers to controlsignalplot

subplot(212);
title('Controlsignal ');
   xlabel('Time [sec]');
   ylabel('[Newtonmeter]');
   axis([t2min,t2max,minvalue2,maxvalue2]);
   plot(0,0,'i');
   subplot(211);
   title('Position,a1 and a2 ');
   xlabel('Time [sec]');
   ylabel('');
   axis([t1min,t1max,minvalue1,maxvalue1]);
   plot(0,0,'i');
hold on;
matrisen=[last; vmeio(reg)];
tref=matrisen(1,5);



while 1
  t=(matrisen(:,5)-tref)/1000;
  subplot(211);
  axis([t1min,t1max,minvalue1,maxvalue1]);
  plot(t,matrisen(:,1),'--');  %pos,tid
  [a,b]=koeff2poles(matrisen(:,2),matrisen(:,3),0.02);
  plot(t,100*a,':');  %vel,tid
  plot(t,100*b,'-');  %ref,tid
  subplot(212);
  axis([t2min,t2max,minvalue2,maxvalue2]);
  plot(t,matrisen(:,4),'-');  %control,tid
  last=matrisen(6,:);
  matrisen=[last; vmeio(reg)];
  [m,n] = size(matrisen);
   if t(length(t)) >= tmax
    tref=matrisen(1,5);
    hold off;
    clg;
    subplot(212);
    title('Controlsignal ');
    xlabel('Time [sec]');
    ylabel('[Nm]');
    axis([t2min,t2max,minvalue2,maxvalue2]);
    plot(0,0,'i');
    subplot(211);
    title('Position(--),a1(:) and a2 (-) ');
    xlabel('Time [sec]');
    ylabel('');
    axis([t1min,t1max,minvalue1,maxvalue1]);
    plot(0,0,'i');
    hold on;
   end;
   if exist('glob_logmatrix')==1
     [loglen,nnnn] = size(glob_logmatrix);
     glob_logmatrix = [glob_logmatrix(2:loglen,:) ; matrisen(m,:)];
   end
end; % While
```

```
MODULE Test;
FROM Kernel IMPORT Init,SetPriority;
FROM Opcom IMPORT InitOpcom;
FROM Refsignal  IMPORT InitRefsignal;
FROM Estimator IMPORT InitEstim;
FROM Regul IMPORT InitRegul,RegParPek ;
IMPORT MatComm;
BEGIN
   Init;
   MatComm.Init;
   InitOpcom();
   InitRefsignal();
   InitRegul();
   InitEstim;
   SetPriority(1000);
   LOOP END;
END Test.
```

Adaptive Control Project Report

Martin Kruciński, D-87

May 30, 1991

Adaptive Control of a Dairy Filling Machine

Thank you for all help

Tore Hägglund, LTH
Bert-Ove Bergman, Tetra Pak
Åke Blomkvist, Tetra Pak
Anders Sundberg, Tetra Pak

Contents

1 Problem presentation

I chose to see if adaptive control could be used to solve some difficulties with a dairy filling machine. The filling machine is ment to fill various products with different viscosity, from water and milk to yoghurt and raspberry cream. With todays controller, the system breaks down if the viscosity is too high because the filling time can not be held under a prespecified maximum time.

1.1 The Machine

The machine has a big tank situated quite high. The product which we want to fill enters this tank through a product pipe. The product level in the tank is kept constant by a regulator. From the tank there is a pipe leading to the filling valve. On its way to the packages right below the filling valve the product passes a flow-meter.

1.2 The Flow-meter

The flow-meter constantly monitors the flow through the filling pipe. When most of the desired volume has been filled, it sends a signal, PreBatch, to the filling valve which then closes approximately half-way. When only a small portion of the desired volume remains to be filled, a second signal, Batch, is sent. This signal closes the filling-valve totally. The small amount of product that manages to flow into the package before the filling valve closes is just enough to fill up the package to the desired volume.

1.3 The Problem

The filling time of a package is specified in advance when the machine is constructed and cannot be changed. This implies that all products have to be filled at less than a specified maximum time. When trying to fill high-viscous products, such as yoghurt or raspberry cream, the pressure losses in the machine are so high that the flow achieved under filling is not enough to fill a package in the specified maximum time.

1.4 The Goal

The goal is to suggest a regulating strategy that will fill all different products in less time that the allowed maximum time. To achieve this I must increase the product flow through the machine. The time is, as mentioned, limited. Also, I have to try to keep the filling errors to a minimum.

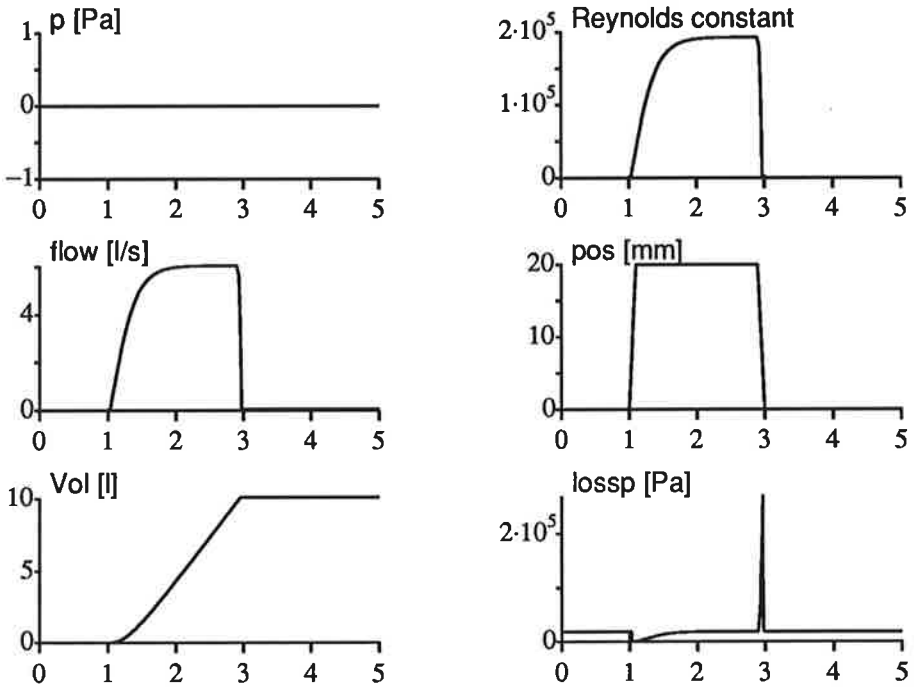


Figure 1: Simulation of flow through machine with water

2 Adaptive Control?

2.1 Viscosity

The viscosity is often an abstract thing. You do not have such a good "feeling" for it as for other physical constants as density etc. The relative difference in viscosity between the products can differ as much as 80 000! But this does not affect the flow in the machine that much. I have simulated the machine with water, see figure 1, and raspberry cream, see figure 2. I have added some pressure to the product tank at $t = 2$ in figure 2 to show that even a moderate increase in p almost doubles the low. The differences in flows without any external pressure is approximately 15 times.

2.1.1 Temperature dependency

The product properties are also affected by the temperature. The viscosity is proportional to $e^{constant/T}$, strongly temperature dependent. This can give big variations in the viscosity if the product temperature varies during filling. The temperature dependency can also be used to ones advantage. If a product is to thick to be filled one can try to fill it at a higher temperature.

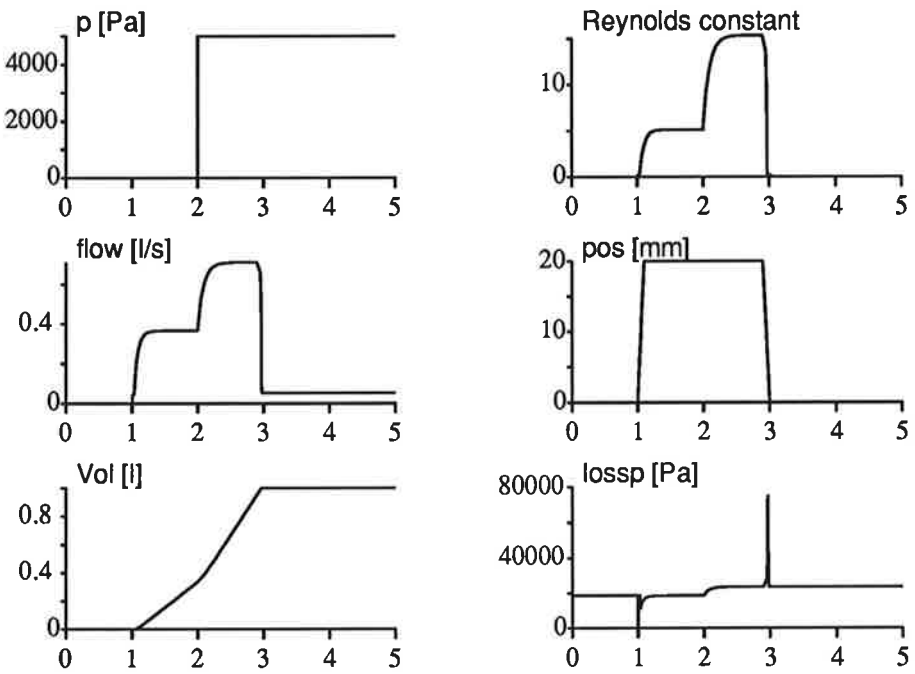


Figure 2: Simulation of flow through machine with raspberry cream

2.1.2 Time dependency

The product properties can also change with time. For example, yoghurt is mostly produced in batch. During the time the batch is packaged (maybe several hours) the viscosity changes considerably.

All these factors surely affect the flow through the filling machine and has to be compensated for.

2.2 Adaptive control is used

Adaptive control is in fact used in the controller that is used today. The flow-meter is equipped with a little computer that controls the measurements but also implements some clever "adaptive control".

2.2.1 Example of operation today

I specify the total volume I want to fill into the packages, 1000 ml, the Batch volume, 950 ml, and the PreBatch volume, 800 ml. These volumes are marked in figure 3.

During the filling process, the flow-meter computer constantly checks the volume that flows into the package after the Batch signal is given. In this way it knows the total volume filled into every package. If it does not

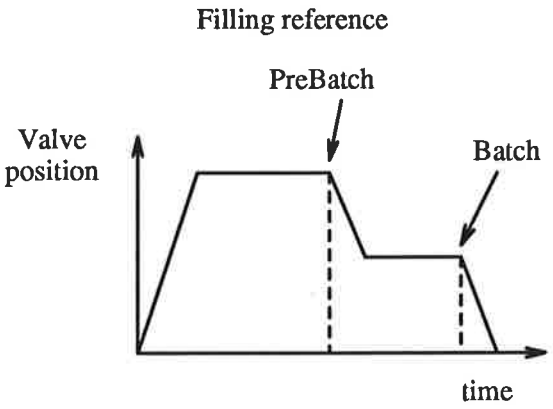


Figure 3: The filling curve used today

correspond to the prespecified filling volume it changes the Batch volume in the direction that corresponds to a correct filling volume. For example: If the Batch volume is set to 950 ml and the machine fills 985 ml it changes the Batch volume to 955 ml. These changes are made constantly every filling. In this way the filling mechanism can handle product variations and is in some way "adaptive". You may think this is not so especially advanced but it mostly works very well.

2.2.2 Problems today

The problem in the filling machine is that very viscous products give too big frictional losses. The available pressure in the tank, 1 atm, is not enough to "push" the product through the machine with the desired speed. An additional pressure increase in the product tank has to be arranged.

Another problem occurs if I have a very big pressure in the product tank. I am then sure of being able to fill all different products. But some products, e.g. yoghurt, are very sensitive to shearing. Shearing is the process when different "layers" in the product are moving relatively eachother. This is what happens when the product flows through the filling machine. This shearing damages the product, makes it less viscous. This is why I want to hold the pressure at the level that is exactly enough to be able to fill in the specified time, not lower nor higher. This requires perhaps adaptive control.

3 Different solutions

I have not had time enough to try many different adaptive regulators on the machine. The goal is to fill a package in 1 s and then do nothing for 1 s. This is because the package that was filled has to be moved away and another placed into position below the filling valve before I can start filling again. To achieve this I try to regulate the machine to follow a prespecified

flow reference. This reference is a step with a rise and fall time of 0.1 s and is zero when the packages are "indexing" (moving one step).

3.1 P-regulator

I first tried to regulate the flow with a P-regulator, controlling only the valve piston position. The product tank pressure was specified by hand. This regulator was bad since I got an error of nearly half the flow reference in stationarity.

3.2 PI-regulator

The PI-regulator i tried next also only controlled the valve piston position. Now I got rid of the error in stationarity. The simulation results were quite good, even with such a simple regulator. The filling error was around 2 – 3 %. I also got acceptable results whith such different products as water and raspberry cream without having to adjust the k and Ti parameters. The only thing I had to do was to add some extra pressure in the product tank.

3.3 PI-regulator with adaption of tank pressure

To be able to handle all kinds of products I had to modify the PI-regulator. I was searching for a simple adaptive scheme. I noticed that the PI-regulator most times manages to stabilize the valve piston position at the end of the fill cycle. I know that too low pressure in the tank is equivalent with the piston position being at its maximal 30 mm but the flow not corresponing to the reference. So I simply take a sample of the piston position at $t = 0.8\,s$ and see if it is above a prespecified value. If it is, I use this information to increase the tank pressure proportionally to the amount the position exceeds the prespecified value (a sampled P-regulator).

This scheme works good when I start with the pressure in the product tank at 0. Sadly it cannot handle the case when the viscosity decreases during filling and the tank pressure needs to be decreased.

I think one way to handle this better is try to estimate a loss-coefficient at the beginning of the filling when I open the filling valve. The information I get by seeing how fast and how much the product is accelerating I think is enough to do this. I do not think it is very wise to estimate the loss when I close the filling valve. The model is quite non-linear in this case. When I open the valve I get approximately a first-order system response. But when I close the valve fast the response is just as fast. I get a very high pressure increase when the valve closes that stops the liquid almost at once. This comes from the pipes and the valve extending a bit and then retracting back. You can study this phenomenon in the plot of lossp in figure 1.

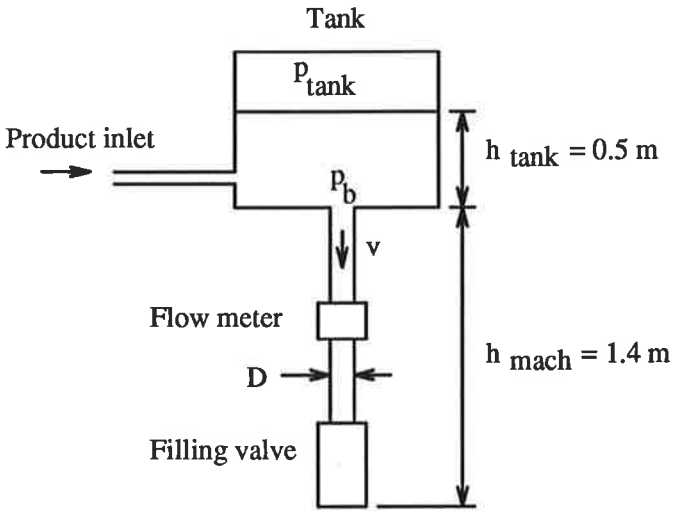


Figure 4: The simplified machine used in the model

4 The Model

I have simplified the process quite a lot. I had to understand many things about the whole system before I got a clear picture of how I should model the machine. Due to this fact I did not get time to simulate the process as detailed as I wanted. I have concentrated my efforts to model as many things concerned with the product properties as possible (viscosity and pressure losses). Other things as machine dynamics and correct machine sizes have been simplified

4.1 The machine

The pipes in the real machine are of various diameters, length and geometry. I have captured only the main outlines of it in the model. There is simply the tank with a product level of 500 mm and a pipe,1400 mm long, $D = 40$ mm, leading down to the filling valve. Figure 4 shows the simplified machine used in my simulations.

4.2 The filling valve

I have neglected the dynamics of the motor and helix screw that drive the filling valve. I have had problems of how to model the losses in the filling valve. I have tried to make some more advanced models but since these would have to be verified in some way I wait with them until future simulations. The simplified valve geometry used in the simulations is shown in figure 5.

The loss is now modeled as a one-time loss with $\zeta = 1$.

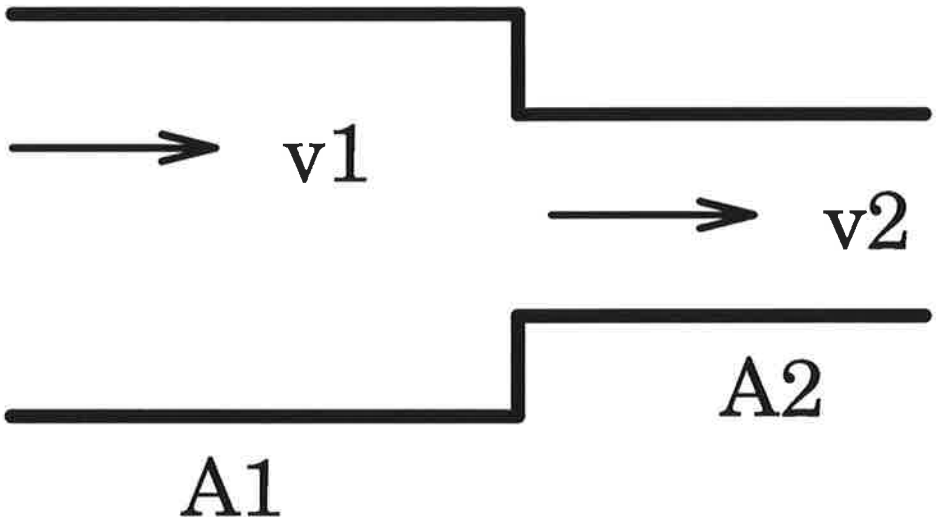


Figure 5: Simplified model of filling valve geometry

$$\Delta p = \zeta \frac{\rho (v2)^2}{2}$$

$$v2 = \frac{A1}{A2} v1$$

The area of the valve opening have been specified in detail. The above geometry is a simplification of the real valve opening.

4.3 Fluid dynamics

This part has been handled best, see Appendix for calculations. I haved modeled these situations:

4.3.1 Newtonian fluids

- Laminar flows (Re < 2300)
- Turbulent flows (2300 < Re < 100 000)

4.3.2 Ostwald-de Waele fluids

- Laminar flows (Re < 2300)

170

5 Simulation Results

5.1 Simnon Troubles

I have had severe numerical problems with Simnon. Since the pressure loss calculations with Ostwald-de Waele fluids involve logarithms of the flow I got big problems when my flow got just below zero when simulating. I solved this by introducing two pair of variables, one that was driven by the derivate and one that was copied from the first but always made positive or zero. Another problem was getting the sample time right in the adaptive PI-regulator. Also this i finally managed and I think that most variables behave as I want them to.

5.2 Simnon Systems

The total simulation system consists of 5 systems, 1 macro and 2 user defined functions. Here is a description of the parts.

model1 This system contains all the things modelling the machine, flows, accelerations and pressure losses.

reg This system is the PI-regulator with anti-windup and adaption.

preg This system helps the regulator to get samples of the valve piston position at specified time instants.

ref This system supplies all the references used in the simulations.

creg This system connects all the subsystems.

flow This macro runs the simulation.

afunc2 These are the 2 user-defined functions. They define the flow reference and the volume reference.

5.3 Simulation Results

These are the names of the different variables shown and used in the simulations:

e	–	error in flow (flow reference - actual flow)
flow	–	actual flow through the machine
flowreference	–	a step with rise- and falltimes = 0.1 s
Vol	–	actual volume filled
Volref	–	the desired volume filled at $t = 1, 3, 5, \ldots\ s$
Reynolds constant	–	used to check if turbulent or laminar flow in the machine
pos	–	filling valve piston position (pos = unfiltered pos in these simulations)
p	–	applied pressure in the product tank
lossp	–	pressure losses in the machine
k	–	gain in PI-regulator
ti	–	integration time used in PI-regulator
kpos	–	adaption gain for pressure adjustements used in the adaptive PI-regulator

Figure 6: Simulation with raspberry cream, $n = 0.35$, $k = 28$. Laminar flow in the machine. $k = 12000$, $ti = 0.04$, $p = 20\,000\,Pa$.

Figure 7: Simulation with normal water, $\eta = 0.001$. Definitely turbulent flow in the machine ($Re \gg 2300$). $k = 12000$, $ti = 0.04$.

Figure 8: Simulation with raspberry cream, as figure 6 but without any extra pressure in the product tank. $p = 0\,Pa$.

Figure 9: Simulation with raspberry cream and the adaptive PI-regulator. The pressure starts at $0\,Pa$ and then adjusts to around $10\,000\,Pa$. $k = 5000$ $ti = 0.05$ $kpos = 300$. The severe ringing in the pressure loss, lossp, is normal.

6 Results and discussion

I have shown that it is possible to model a dairy filling machine roughly and make simulations that are not too far from reality. (Verified by some experts at Tetra Pak). An acceptable control can be achived with a PI-regulator with simple adaption of pressure control.

Todays regulator follows a flow reference. This is perhaps not exactly what is wanted. I think it is better to specify a volume reference and try to follow that instead. The only thing we want is the filling error at $t = 1, 3, 5, \ldots\ s$ to be minimal, not be able to specify exactly how the filling profile should look like. I will try this scheme in future simulations with a volume reference that gives continuous movements of the valve piston. This means that I will specify the volume reference as a ramp, but with a t^3 curve at start and end.

I think also that the very simple adaption algorithm can be improved a lot. I will try to make a better estimate of the losses due to different product viscositys. The goal is to make the machine fill all kinds of products, with-

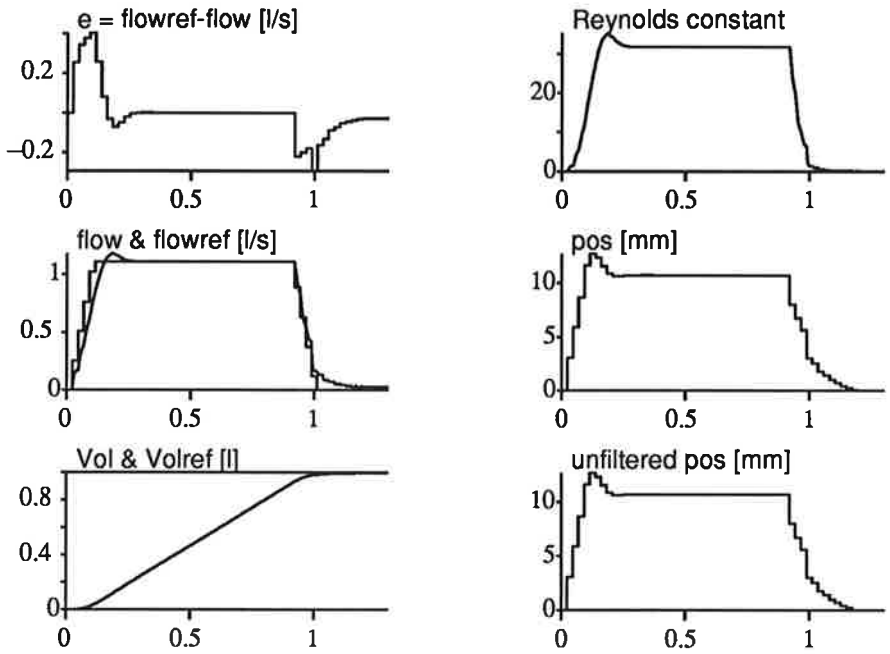


Figure 6: Simulation with raspberry cream

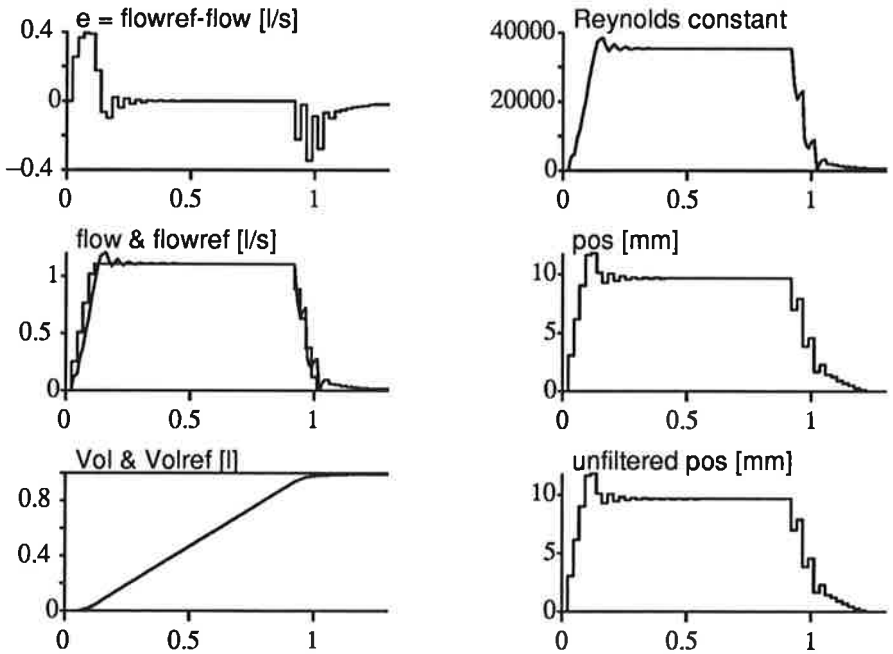


Figure 7: Simulation with water

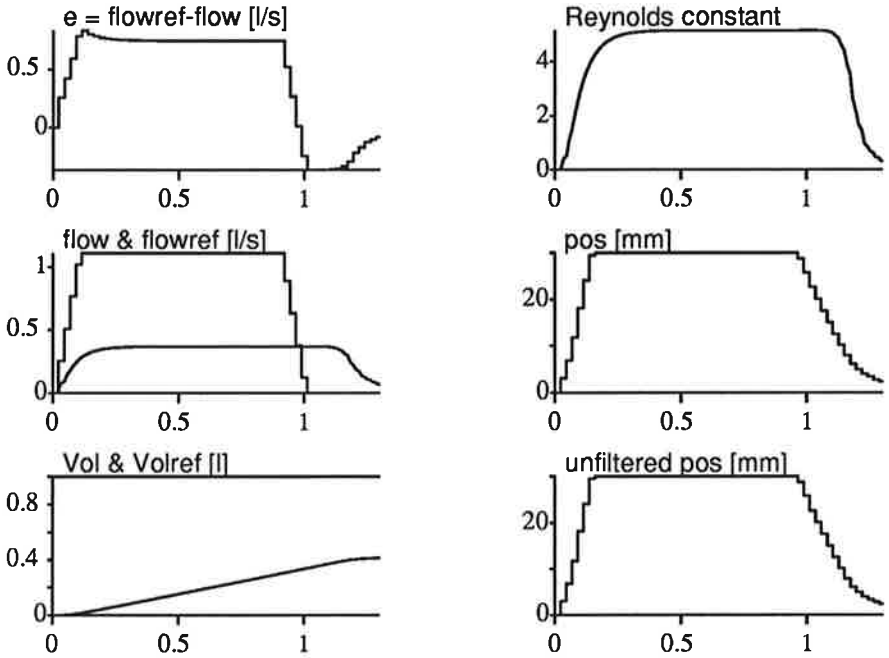


Figure 8: Simulation with raspberry cream, too low pressure in product tank

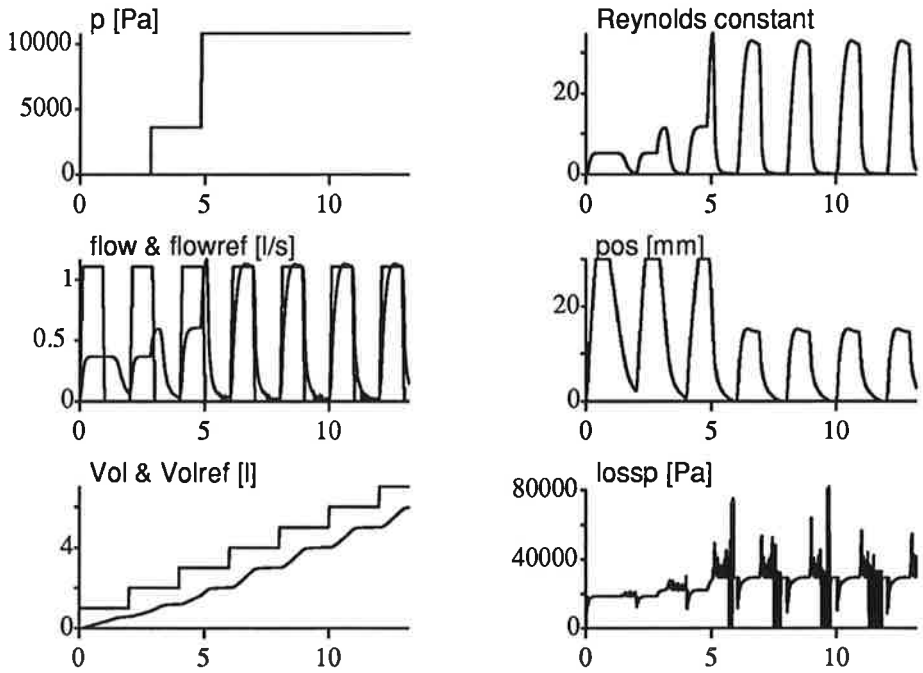


Figure 9: Simulation of adaptive PI-regulator with raspberry cream

out the user specifying a single product specific parameter. Sometimes, in foreign countries, the people who operate the machines can hardly read. So it would be ideal if the adaptive regulator can run entirely on its own.

7 Appendix: Fluid Dynamics

Calculations of frictional losses and Reynolds constant when fluids flow through circular pipes.

7.1 Newtonian fluids

Frictional losses:

$$\Delta p = \frac{\lambda L \rho v^2}{4R}$$

$$\lambda_{Re<2300} = \frac{64}{Re}$$

$$\lambda_{Re>2300} = \frac{0.316}{\sqrt[4]{Re}}$$

Reynolds constant:

$$Re = \frac{\rho v D}{\eta}$$

7.2 Ostwald-de Waele fluids

Frictional losses:

$$\Delta p = \frac{4KL}{D}\left(\frac{3n+1}{4n}\right)^n \left(\frac{32\Phi}{\pi D^3}\right)^n$$

Reynolds constant:

$$Re = \frac{D^n v^{2-n} \rho}{8^{n-1} k \left(\frac{3n+1}{4n}\right)^n}$$

7.3 Constants

η	–	Viscosity (Newtonian fluid)
n k	–	Viscosity constants (Ostwald-de Waele fluids)
R	–	Pipe radius
D	–	Pipe diameter
L	–	Pipe length
v	–	Average product velocity
Φ	–	Flow
ρ	–	Density

Lunds Tekniska Högskola
Institutionen för Reglerteknik

TEST AV AUTOTUNER ECA40

Sven Andersson
Svein Helgesen

handledare
Professor Karl Johan Åström

Lund 910502

Innehållsförteckning

Inledning

Projektet var en förstudie inför vårt kommande examensarbete, där vi skall testa regulatorer vars parametrar ställs in med hjälp av autotuning.I detta projekt begränsade vi oss dock till att testa en autotuner, ECA40 från SATTCONTROL

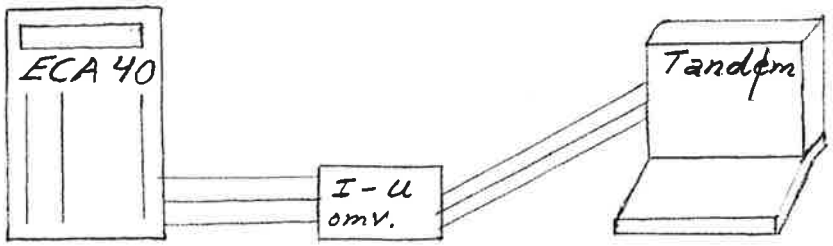
En autotuner är en regulator, i vårt fall en PID-regulator, som själv ställer in sina parametrar k,Ti och Td. För att finna lämpliga parametrar används en metod som kallas relä-återkoppling vilket innebär att regulatorn lägger ut en styrsignal ,ett antal fyrkantspulser,och genom att avläsa ärvärdet kan parametrarna fastställas genom en algoritm.

Projektet gick ut på att testa regulatorn ECA40 mot en process, som simulerades av en dator med hjälp av simuleringsprogramet RealtidsSimnon.
Vår första uppgift blev att få kringutrustningen att fungera dvs att koppla ihop datorn med regulatorn.Därefter testade vi regulatorn mot olika processer och uppmätte olika data samt undersökte regulatorns förmåga att bemästra störningar.

1.Uppkoppling av utrustningen.

Vi började med att koppla upp utrustningen , se fig.1, som bestod av en ECA40, PC-AT samt en ström-spänningsomvandlare (den sistnämnda användes för att omvandla regulatorns strömsignaler till datorns spänningssignaler och vice versa).Nästa steg var att kontrollera uppkopplingen vilket vi gjorde genom att skicka ut en signal från regulatorn till datorn .Detta fungerade inte .Efter att ha studerat regulatorns användarbeskrivning samt prövat oss fram utan framgång fick vi så småningom veta att realtidssimnon ej var konfigurerat för den PC-AT vi hade, en ..... Därför bytte vi till en Tandem maskin och vipps då fungerade "allt"! dvs vi klarade att etablera en kontakt mellan datorn och regulatorn.

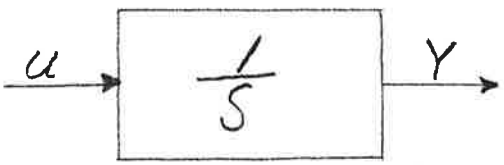
Fig.1



2.Simulering av testprocess.

Vi skapade en kontinuerlig testprocess bestående av en enkel integrator [H(s)=1/s] för att kunna gå vidare. Att vi valde en integrator berodde på att vi senare i projektet skulle testa en tredje ordningens process [H(s)=k*a^3/(s+a^3)].
Anledningen till att vi kunde använda oss av en kontinuerlig process i datorn istället för en diskret, var att processen är så mycket långsammare än tex datorn och dess kommunikations hjälpmedel med omvärlden dvs AD- och DA-omvandlarna.
Simuleringen av testprocessen skedde utan reglering och med en konstant insignal U till processen (se fig.2).

Fig.2



När vi försökte köra en simulering så bröts den efter ett tag och vi satt länge förbryllade och undrade vad det berode på ? Efter att ha studerat simnon manualen framgick det att vid integrering så används automatiskt en mycket noggrann men samtidigt mycket tidskrävande algoritm som heter Runge-Kutta. Denna algoritm tar för mycket cpu-tid och följaktligen blir tex utskriftsprocessen i datorn och den simulerade processen utsvälta dvs de får ingen körtid. Botemedlet mot detta fenomen var att använda Eulers algoritm istället ty denna tar mindre tid i anspråk. Efter initiering av den nya algoritmen plottades utsignalen , en ramp, på skärmen. Detta stämmde med teorin vilket var en kvittens på att kommunikationen mellan regulator och process fungerade.

3.Tuning

Nu var det dags att använda en mer realistisk process [H(s)=1/(s+1)].För att regulatorn ska kunna reglera måste den ha bra parametervärden.Dessa värden får den genom en autotuning (se fig.3).
Metoden som används vid ECA40´s tuning är relä-återkoppling (se fig.4).Denna metod fungerar enligt följande först kopplas PID-delen bort och ersätts av relä-delen,vilket så småningom innebär att återkopplingen självsvänger.Ur självsvängningen kan man sedan med hjälp av Ziegler-Nichols metoden ta fram parametrarna Ti,Td och k.

Fig.3

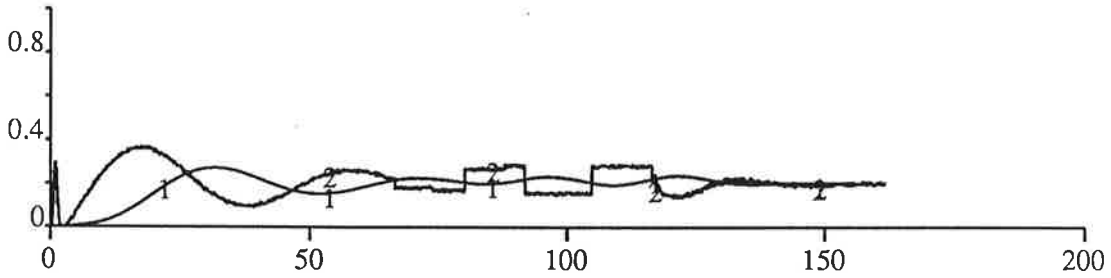
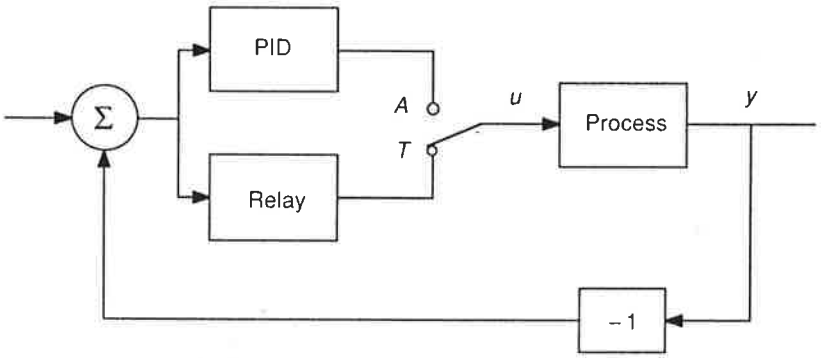


Fig.4



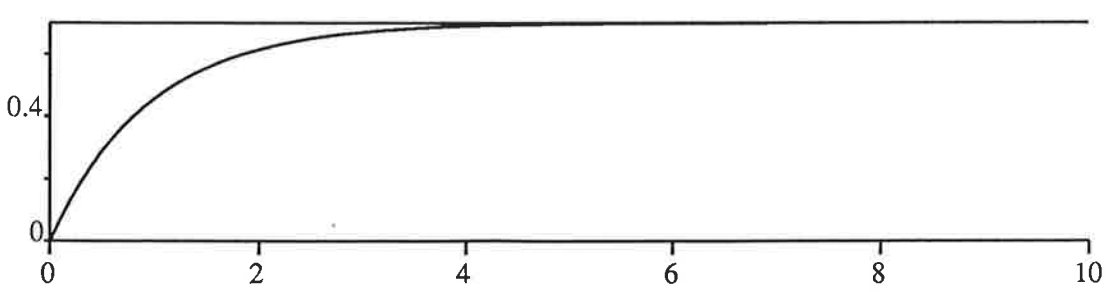
4.Reglering med internt/externt börvärde.

Nästa steg blev att istället för att ställa in börvärdet Uc på regulatorn direkt så skickades börvärdet externt från datorn.Detta implementerades i huvudprogrammet, dvs processen,(se bilaga 1).

5. Test av stegsvar för 1/(s+1)

För att kunna kontrolera att vi hade implementerat processen korrekt så kopplade vi ifrån regulatorn samt la på ett steg direkt inne i processen.Teoretiskt skall tidskonstanten nu för processen [H(s)=1/(s+1)] vara 1s , vilket den utplottade kurvan kan verifiera (se fig.5). Följaktligen var processen riktigt implementerad.

Fig.5



6.Stegsvar på det återkopplade systemet .

Nu var det dags att testa det återkopplade systemet med processen [H(s)=1/(s+1)^3], därför kopplade vi in regulatorn.Insignalen dvs börvärdet Uc var ett steg in till det återkopplade systemet.När vi plottade ut stegsvaret (se fig.6), blev vi dock mycket förundrade över hur långsamt systemet var! Ty vi förväntade oss att det återkopplade systemet skulle vara snabbare än det öppna systemet.Det återkopplade systemet visade sig vara 4-gånger långsammare och då drog vi slutsatsen att någonting var fel.Efter ett par dagars felsökning och funderingar så kom vi fram till att det nog var regulatorns konstruktion som begränsade det återkopplade systemets stegsvar.Därför beslöt vi oss för att testa systemet genom att lägga ut ett steg manuellt från regulatorn utan återkoppling.Då kunde vi isolera datorn,ledningar till och från datorn, samt AD- och DA-omvandlare för att undersöka hur enbart dessa påverkade stegsvaret.Då vi la ut steget från regulatorn manuellt så var vi tvungna att simulera en långsammare process, [H(s)=a^3/(s+a)^3] där a=0.03 , för att bli oberoende av våra begränsningar att manuellt göra ett steg ty steget blir relativt rampliknande (se fig.7).

Fig.6

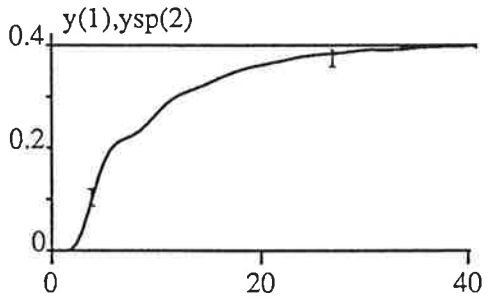
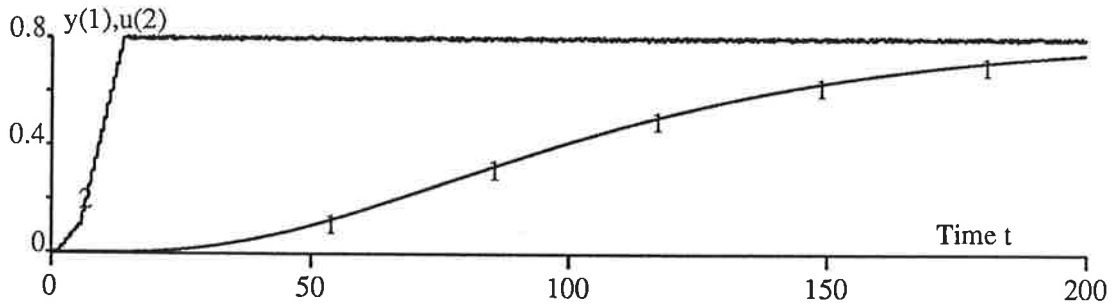


Fig.7

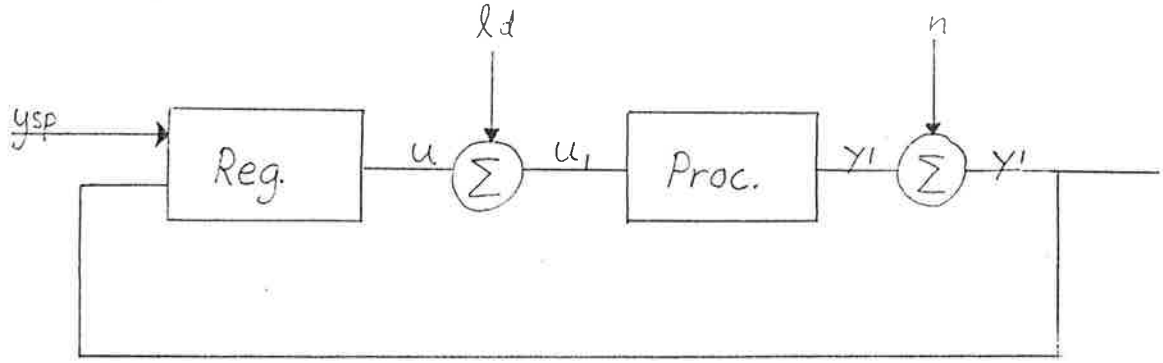


Resultatet blev att vi enligt kurvan fick en tidskonstant runt 100s vilket stämmer bra överens med tumregelns förväntade 3*1/0.03=100s.Följaktligen var det regulatorn som försämrade tidskonstanten på stegsvaret. Enligt Tore Hägglund är ECA40´s autotuning konstruerad för att ta fram relativt snälla parametrar vilket innebär att stegsvaret blir långsammare men med mindre osillativ insvängning.Därmed blir säkerhetsmarginalerna mot självsvängning onödigt stora.

7.Det återkopplade systemets störningstålighet.

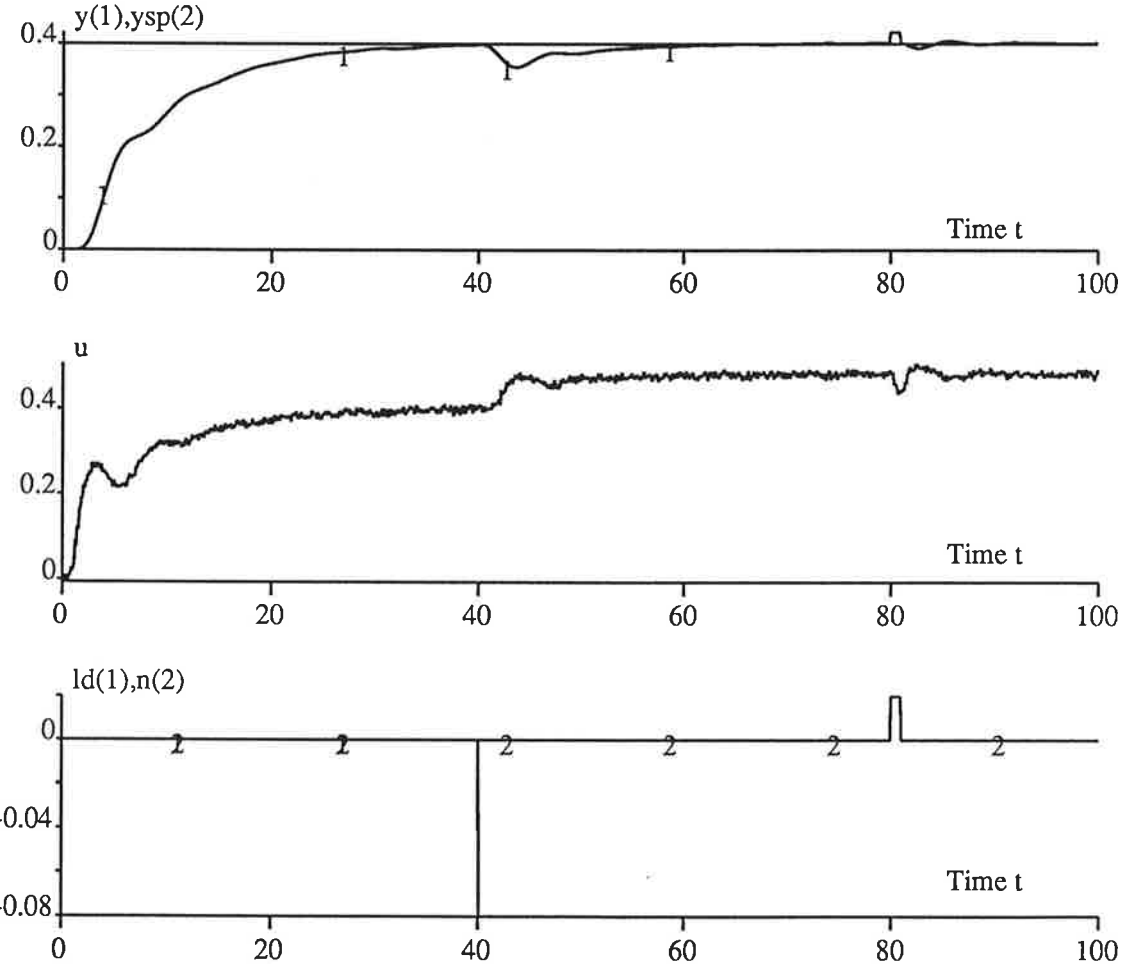
De två typer av störningar vi använde oss av var laststörningar ld och mätbrus n (se fig.8).Laststörningen var en konstant signal på 20% av börvärdet ysp medan mätbruset var en puls med längden 1s och amplituden lika med 5% av börvärdet ysp.

Fig.8



Testet gick till så att ysp var ett steg och när detta hade svängt in sig så kom en laststörning följt av mätbruset.För processen [H(s)=1/(s+1)^3] ser körningen ut enligt följande (se fig.9).

Fig.9

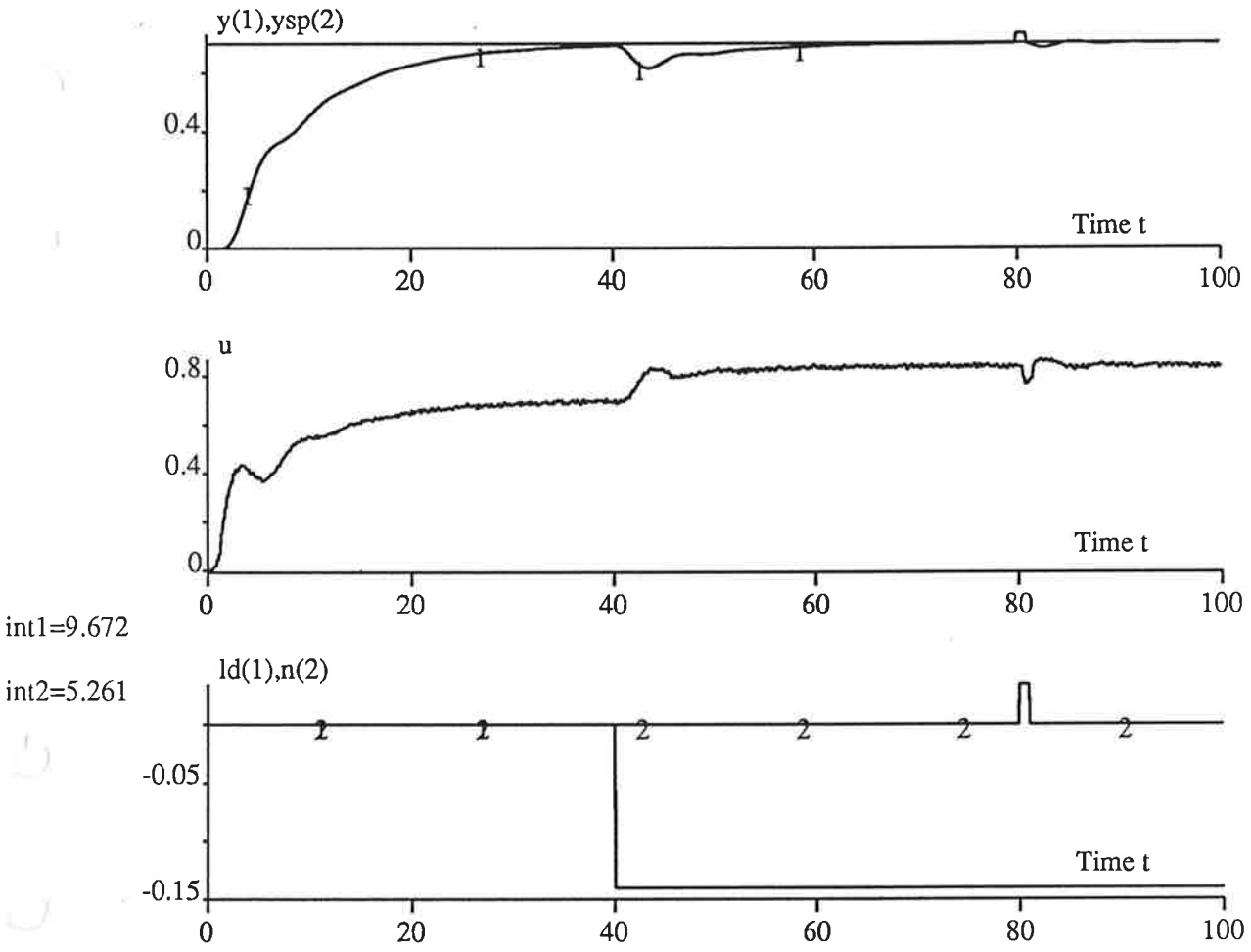


int1=9.232

int2=4.484

För att kunna relatera stegsvaren respektive laststörningståligheten kvalite gentemot andra processer/ andra börvärden räknade vi ut något som kallas det normerade absoluta integralfelet för både stegsvaret (int1) och laststörningen (int2).Det absoluta integralfelet är skillnadsytan mellan börvärde (ysp) och ärvärde (y).Detta normeras genom att man dividerar med börvärdet.En verifiering av det ovan sagda fås genom att jämföra för samma process, int1 och int2 i fig.9 ,där ysp är 0.4, med fig.10 där ysp är 0.7 .

Fig.10



int1=9.672

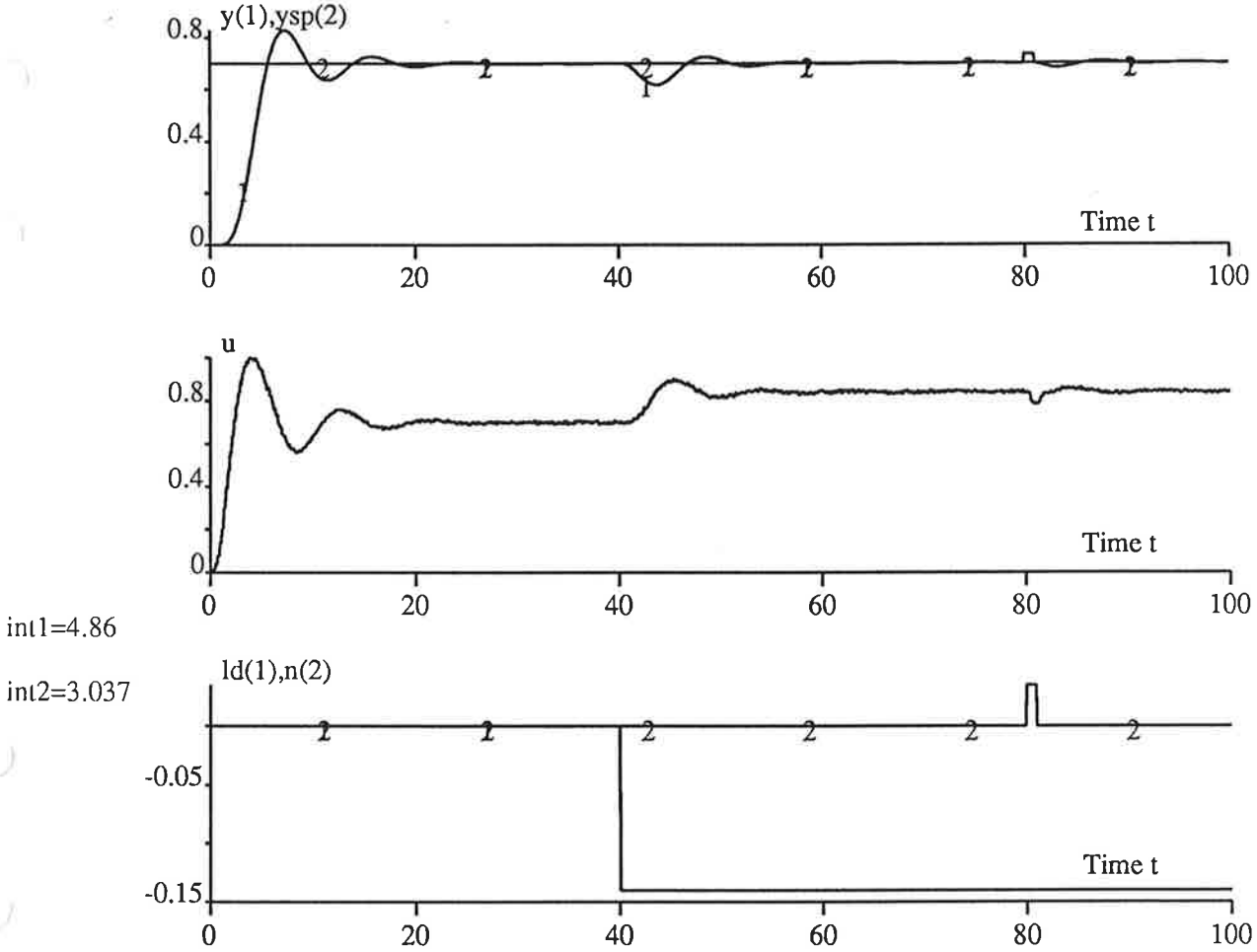
int2=5.261

Tidigare i rapporten nämnde vi att ECA40 har onödigt stora marginaler mot självsvängning och Karl Johan Åström tyckte därför att vi skulle pröva följande paramervärden istället :

	k	Ti	Td
ECA40	1.04	4.9	1.2
Åström	0.64	1.1	1.1

För processen [H(s)=1/s+1)^3] blir simuleringen med Åströms parametrar enligt fig.11 och med ECA40's parameterinställning enligt fig.10.

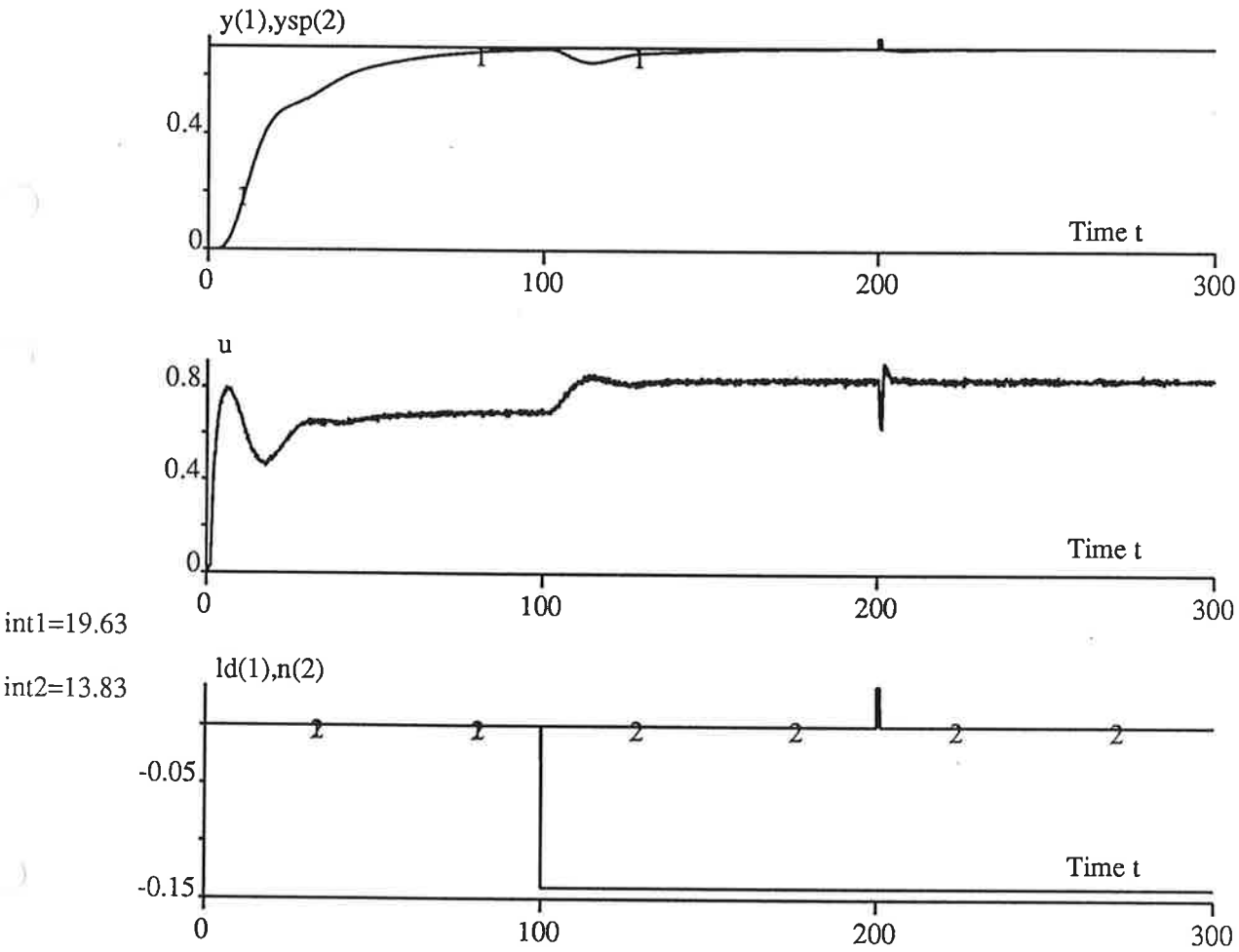
Fig.11



Man ser att snabbheten i stegsvaret är bra mycket bättre med Åströms parametrar utan att systemet för den skull blev farligt oscillativt .När man tittar på mätbrusets påverkan ser man ingen skillnad däremot blev absoluta integralfelen bättre i Åströms fall se fig.10,fig.11

Därefter körde vi en process med 5ggr längre tidskonstant och kan konstatera att den relativa tidsförsämringen pga regulatorn blir bättre än för föregående process (se fig.12).

Fig.12



186

8.Kommentar

Målsättningen med detta projekt var att sätta oss in i realtidssimnon samt att använda en på marknaden existerande autotuner mot en simulerad process.Anledingen till att projektet var utformat på detta sätt var att vi skulle göra ett examensarbete som innebar att vi gjorde ett generellt program för att testa marknadens autotuners .Vi har blivit en erfarenhet rikare och det är att saker och ting tar längre tid än vad man tror.

```
continuous system proc3
"k*a^3/(s+a)^3
time t
input u
output y ld0
state x1 x2 x3
der dx1 dx2 dx3
z=daout(chan0,ysp)
ld0=0.2*ysp
ld=if t < td1 then 0 else ld0
ld1=-ld
u1=u-ld
n0=0.05*ysp
n=if t < td2 then 0 else if t < td3 then n0 else 0
k1=a*a*a*k
dx1=x2
dx2=x3
dx3=-3*a*x3-3*a*a*x2-a*a*a*x1+k1*u1
y1=x1
y=y1+n
chan0:0
ysp:0.5
x1:0.0
a:1.0
k:1.0
td1:40
td2:80
td3:81
end
```

```
macro macro3
syst proc3 aif adc1 dac1 con3
write 'process=k*a^3/(s+a)^3'
write 'please enter k, a and ysp'
read k3 real;a3 real;ysp3 real
par k[proc3]:k3
par a[proc3]:a3
par ysp[proc3]:ysp3
par ysp[aif]:ysp3
store y[proc3] ysp[proc3] u[proc3] ld1[proc3] n[proc3]
simu  0 100 0.1 -mark/a
split 3 1
"axes h 0 40 v 0 1
ashow y[proc3] ysp[proc3] -mark /a
text 'y(1),ysp(2)'
mark a 20 11.3
mark "Time t
"axes
ashow u[proc3]/a
text 'u'
mark a 20 6.3
mark "Time t
"axes h 0 40 v -1 1
ashow ld1[proc3] n[proc3]-mark/a
text 'ld(1),n(2)'
mark a 20 1.3
mark "Time t
disp yfel1[aif]/yfel1-aif
mark a 0 5
mark:int1 = yfel1.aif
disp yfel2[aif]/yfel2-aif
mark a 0 4
mark:int2 = yfel2.aif
end
```

```
connecting system con3
time t
y[dac1]=y[proc3]
u[proc3]=u[adc1]
y[aif]=y[proc3]
ild[aif]=ld0[proc3]
end
```

```
continuous system adc1
time t
output u
u=scale1*adin(chan1,t)+offset1
chan1:1
offset1:0.0
scale1:1
end
```

```
continuous system dac1
time t
input y
z=daout(chan1,scale1*y+offset1)
"z=daout(scale1*y+offset1,chan1)
"y=scale1*daout(chan1,t)+offset1
chan1:1
offset1:0.0
scale1:1
end
```

```
continuous system aif "normerat absolut integralfel av ysp-y
time t
input  y ild
state x1 x2
der dx1 dx2
dx1=if t<t1 then sqrt(sqr(ysp-y)) else 0
dx2= if t<t1 then 0 else if t<t2 then sqrt(sqr(ysp-y)) else 0
yfel1=x1/ysp "normerat integral-fel fr[n 0 till t1
yfel2=x2/ild
x1:0
x2:0
ysp:1
t1:40
t2:80
end
```

```
continuous system proc3t
"vid tuning av process=k*a^3/(s+a)^3
time t
input u
output y
state x1 x2 x3
der dx1 dx2 dx3
dx1=x2
dx2=x3
dx3=-3*a*x3-3*a*a*x2-a*a*a*x1+k*a*a*a*u
y=x1
x1:0.0
a:1.0
k:1.0
end
```

```
macro macro3t
syst proc3t adc1 dac1 con3t
write 'process=k*a^3/(s+a)^3'
write 'please enter k and a'
read k3 real;a3 real
par k[proc3t]:k3
par a[proc3t]:a3
store y[proc3t] u[proc3t]
simu 0 200 0.1 -mark/a
ashow y[proc3t] u[proc3t]-mark/a
text'y(1),u(2)'
mark a 20 11.3
mark "Time t
end
```

```
connecting system con3t
time t
y[dac1]=y[proc3t]
u[proc3t]=u[adc1]
end
```

COMPOSER

av

Markus Jakobsson

Composer

Composer är ett program som efter att ha lyssnat på inmatade musikstycken bildar sig en uppfattning om vad musik är, vilka regler som finns för melodier - vad låter bra och vad låter falskt? - och sedan komponerar små melodier själv. Jag som har gjort programmet heter Markus Jakobsson.

Bakgrund

Programmet har gjorts som ett samprojekt mellan kurserna Adaptiv Reglering på Institutionen för Reglerteknik och Neurala Nät på Teoretisk Fysik vid LNTH. Önskan att göra ett dylikt program har jag haft länge, och när jag fick en idé hur problemet kunde lösas kändes detta som någon sorts befrielse. Ytterligare befriad kände jag mig då programmet efter två veckors hårt arbete slutligen fungerade. Jag har haft många problem av såväl praktisk som teoretisk natur, och fortfarande finns en del frågetecken.

Tack

Jag har fått hjälp att skriva in melodier av Karin Högstedt, som också har stöttat mig då nätet spelat illa.

Hur det fungerar

I programmet kan man skriva in kända melodier, spara och ladda dessa, träna ett från början helt slumpartat neuralt nät, samt spara och ladda ett nät. Dessutom kan vissa inställningar göras, temperatur och steglängd, till vilka jag återkommer.

Hur man kör programmet

Programmet startas genom att skriva COMPOSER. För att träna nätet måste du ha en melodi att träna det med. Denna kan du antingen skriva in själv eller ladda från diskett. Hur man matar in en melodi framgår i programmet.

Välj sedan **Träna ett nät** i menyn. Cirka tio varv är bra att börja med. Ett varv är en genomträning av hela melodin och tar, beroende på melodilängd och maskin, mellan cirka tio sekunder och två minuter. Du ska ange hur mycket kvadratavvikelsen får öka mellan två varv utan att träningen ska avbrytas och du larmas med en signal som påminner om ett utryckningsfordons. När det förinställda antalet varv genomgåtts tutar datorn för att påkalla din uppmärksamhet.

Om du vid något tillfälle vill avbryta träningen trycker du på en tangent, och träningen avslutas när datorn blivit klar med det varvet den för tillfället arbetar med. Du kan alltid fortsätta träningen efter ett avbrott.

När nätet ska komponera ska du mata in melodins längd (högst 240 toner lång) och det tal som utgör gräns mellan långa och korta noter. Omkring 0.7 är ofta bra. För att ta reda på exakt hur mycket, så testkör först en gång och välj sedan ett värde i mitten av utfallsrummet hos tonlängdsnoden. För att lyssna på en melodi, välj **Skapa/redigera melodi** i menyn.

Körningstips

Om man överträner nätet med en viss melodi blir resultatet sällan speciellt bra. Generellt sett försämrar överträning ett nät. I detta fall händer också att en not (den vanligaste) tvingas fram till en så kraftig dominans att de andra försvinner. Det är bättre om du efter att ha tränat nätet en stund med en melodi övergår till att träna det med en annan. Jag tränade t ex nätet först med "Du gamla du fria" omkring fem - sex varv, och fortsatte sedan ungefär lika många gånger med "Gubben Noak". Resultatet blev slående bra!

Ett annat tips är att spela melodier där inte en ton överväger kraftigt i antal, gärna melodier där olika tonlängder förekommer. Om man väljer väldigt enformiga melodier eller överträner nätet med en och samma melodi så kommer nätet lätt i ett låst läge och spelar bara en och samma not hela tiden.

En bra lösning karakteriseras av att kvadratavvikelsen mellan det rätta och det uppnådda resultatet är låg. Vid träningen kan man råka ut för att kvadratavvikelsen inte alls minskar. Om den oscillerar måste man dock låta den öka lite för att den sedan ska kunna minska motsvarande. När nätet uppnått en kvadratavvikelse under hundrafemtio är lösningen hyfsad, under cirka hälften därav är den bra.

Startläget i nätet är slumpartat, men minimeringsalgoritmen är deterministisk. Vill du t ex testa vilken steglängd som är bäst, så spara det nät som finns när du börjar experimentera med detta. Vill du prova en ny steglängd laddar du nätet på nytt.

Eftersom jag varit tvungen att begränsa problemets omfattning i största möjliga grad så finns bara tonerna i C-dur skalas första oktav. I vissa melodier har därför en icke-existerande ton ersatts med en paus av samma längd.

På disketten finns ett antal inskrivna melodier, alla med suffix TUN, ett antal framräknade nät med suffix NET, samt några melodier komponerade av dessa nät.

En lista på alla inmatade melodier följer här:

- NOAK (Gubben Noak)
- TUMAN (På tu man hand)
- KATALAN (Katalansk visa)
- HÖSTGILL (Höstgille)
- MASKROS (Maskros och tjärdoft)
- ÄNGLAMAR (Änglamarken)
- VANDRARE (Jag är en ensam vandrare)
- O_SOMMAR (O sommar)
- ANNCHARL (Ann-Charlotte)
- KLARASOL (O, klara sol)
- OXDRAGAR (Oxdragarvisan)
- GAMLAFRI (Du Gamla Du Fria)
- KUNGSÅNG (Kungssången)
- UTIVÅRHA (Uti vår hage)
- ALUNDAVI (Alundavisan)
- JUNGFRUN (Och jungfrun hon går..)
- ROSLAGVÅ (Roslagsvår)
- LUCIA (Sankta Lucia)
- OPPOHOPP (Opp och hoppa)
- MODIGGOS (Modiga gossar)
- ASPELÖV (Aspelöv och lindelöv)
- MARKNADS (Marknadsvisan)
- SNABBARE (Går för långsamt)
- ALLARENA (Alla fåglar kommit..)
- FINLANDS (Finlandia)
- GRÖNFORT (Ska gå fortare)

Följande melodier har nätet komponerat:

- GAMLANOA (Tränat med Du Gamla Du Fria och Gubben Noak)
- GAMNOFIN (Ovanstående även tränat med Finlands nationalsång)
- GANOFIÄN (Ovanstående plus Änglamark)
- HÖSTTUM (Tränat med Höstgille och På tu man hand)
- MELODI (Tränat med lite av varje)
- MELODI2 (Som ovan)
- MELODI3 (Som ovan)
- SKALA (Tränat med en skala, fick en skala. Perfekt avpassning)

Följande upptränade nät finns lagrade:

- GAMLANOA
- GANOFIÄN
- HÖSTBRA
- GAMLA

Inställningar

En stor steglängd gör att man snabbt kommer till en lösning, men att man riskerar få oscillationer. En kort steglängd innbär långsammare konvergens. Ett bra värde på steglängden är förinställt. Temperaturmultiplikatorn kallar jag den proportionalitetskonstant som reglerar hur temperaturen beror av kvadratavvikelsen. En hög temperatur ger mycket brus (knuffar kulan ur lokala minima) men ger en inexakt lösning, medan en låg temperatur ger en stor beslutsamhet.

in **gömt** **ut**

Figur: Nätets uppbyggnad

Nätets uppbyggnad

Nätet består av tre lager med framkoppling, ett insignallager, ett utsignallager och ett s k gömt lager däremellan. Varje cell i dessa lager är förbunden med alla cellerna i det eller de näraliggande lagerna, och varje förbindelse har en vikt som talar om hur stark förbindelsen är. (Se figur ovan.) Utsignallagret består av tio utnoder, en för varje ton, en pausnod och en nod som indikerar om tonen som ska spelas är lång eller kort. Insignallagret består av nio sådana uppsättningar av vardera tio noder, vilka utgör ett minne över de nio senast spelade tonerna, och det gömda mellanlagret består av lika många noder som insignallagret, eftersom detta visade sig ge bäst resultat. Varje cell får in insignaler, summerar ihop dem och förstärker dem, varvid den antingen sänder eller inte sänder en utsignal. Cellen sänder en utsignal om den förstärkta summan överstiger ett visst tröskelvärde, annars är den tyst. Utifrån detta minne över de senaste spelade tonerna görs en prediktion om nästa ton som ska spelas. Om under träningsfasen en felaktig prediktion göres så får man ett fel, och nätets vikter ändrar sig så att felet minskas. Ändringen sker enligt gradientmetoden, dvs parametrarna ändras i den riktningen felfunktionens derivata är negativ. Till slut kommer ett minimum att nås.

Tänk dig en kula som rullar fram i ett kuperat landskap och hela tiden rör sig mot bottnen av dalen. Om steglängden är alltför stor kan man få en oscillerande eller rentav instabil lösning genom att man hela tiden tar för stora steg och hoppar över den grop som minimat utgör.

För att inte fastna i lokala minima har jag infört en temperatur i programmet. Vid hög temperatur får kulan en liten knuff då och då. Dessa knuffar tar den ur lokala minima. För att få en stabil lösning när vi nått det globala minimat ska då temperaturen vara mindre. Jag låter temperaturen vara proportionell mot roten ur den kvadratiska avvikelsen för systemet. Detta innbär att när jag är långt ifrån det globala minimat (kvadratavvikelsen är stor) är temperaturen hög och vice versa. Denna metod kallas för simulerad stelning, och fungerar på samma sätt som t ex stelnande metaller. Kyler man en het metall snabbt får man spänningar, men kyler man den sakta blir bindningarna bättre och energin för systemet blir lägre.

Anledningen till att nätet ibland hakar upp sig på en ton, är att om det i minnet bara finns en och samma ton i alla minnespositionerna och påföljande ton är densamma, så kommer - om rätt prediktion göres - situationen att se precis likadan ut som när resultatet propagerats fram ett steg, och resultatet upprepas i oändlighet.

I utsignalskiktet väljes den ton vars nod har störst utsignal, och tonen görs lång om utsignalen från tonlängdens nod överstiger ett bestämt värde.

Vilket temperaturberoende man bör använda sig av för att få optimala beräkningar har jag kommit fram till rent experimentellt till att vara proportionellt mot roten ur kvadratavvikelsen, med en proportionelitetskonstant omkring 3. Ett genom temperaturen är förstärkningen hos cellen i det område som sigmoidfunktionen har en stor derivata.

Matematisk bakgrund

Ändringen av vikterna sker enligt gradientmetoden, och ändringen av en viss vikt är minus steglängden (ett positivt tal) gånger den partiella derivatan av energin med avseende på den avsedda vikten. För att få energin summeras den kvadratiska avvikelsen från det uppnådda till det önskade över alla utnoder, och multipliceras med en halv (för att få en snygg derivata). Den partiella derivatan av energin med avseende på en vikt ingående till utsignallagret, är felet gånger värdet av utsignalen från den gömda nod vikten är förbunden med.

Felet är lika med differensen av den önskade utsignalen minus den uppnådda, gånger derivatan av sigmoidfunktionen i den punkten vi fått genom summeringen av alla insignaler och tröskelvärdet.

Vikterna mellan det gömda lagret och ett underliggande lager, i det här fallet insignallagret beräknas med "back propagation"-regeln. (Se t ex "Neurocomputing" av Robert Hecht-Nielsen.)

Programkoden

Här ska jag i korthet förlara vad programmets olika delar gör, med tonvikt på de delar som ändrar vikterna i nätet.

Play spelar en ton av en viss längd.

Placexy positionerar markören.

ClrEoln tömmer en bestämd rad.

Convert tar bort suffix på programnamn.

ScanDirectory låter användaren välja en fil bland alla de i aktuellt bibliotek som har ett bestämt suffix.

InitWeights ger alla vikterna i nätet ett initialvärde. Trösklarna behandlas som vikter mellan aktuell nod och en nod som alltid är påslagen, dvs som alltid sänder utsignalen ett.

Propagate gör att tonerna tickar fram i minnet och att den ton som spelats senast kommer in längst fram i minnet. Den ton som befinner sig längst bak i minnet försvinner ur minnet.

Sigmoid är en funktion som med en tabell (för snabbhets skull) beräknar tangens hyperbolicus av det värde den får in. Det får varken bli ett eller minus ett utan bara mycket nära pga att derivatan annars blir noll, varvid ingen viktsändring kan ske.

Sigmiodprim beräknar värdet av derivatan av tangens hyperbolicus för ett givet argument.

ComputeOutput gör en prediktion om vilken ton som ska spelas genom att låta signaler gå genom nätet. Signalerna går från ett undre lager mot ett övre. Det understa lagret är insignallagret, det översta utsignallagret.

Compose komponerar en melodi med det nuvarande nätet. Först lägges vitt brus i form av slumpartade toner in på minnets alla positioner, och sedan göres ett förutbestämt antal prediktioner. Då en ton predikterats går den in på den främsta minnespositionen.

200

TrainNet tränar ett nät genom att låta en melodi ton för ton predikteras i rutinen Backpropagation som sedan justerar vikterna. Sedan låts den riktiga tonen propagera in i minnet. Temperaturen, som bestämmer nätets grad av stokastik, beräknas med hjälp av det gamla värdet på kvadratavvikelsen. Temperaturen användes av Backpropagation.

Backpropagation är programmets centrala del. Här anropas ComputeOutput och den beräknade signalen jämförs med den önskade. Sedan ändras vikterna för att minska det kvadratiska felet nästa gång. s2 är kvadratavvikelsen, Delta skillnaden mellan det önskade och det uppnådda, DeltaWeightSum är det fel som ska propageras vidare till nästa lager. OutpWeight respektive HiddenWeight är vikterna som går in i utsignal- respektive det gömda lagret. I det översta lagret finns NodesPerOctave noder (tio stycken, varav åtta toner, en paus, och en nod för tonens längd), I de två andra lagrena finns NodesPerOctave gånger TheDelay noder, där TheDelay är minnets längd.

LoadNet laddar in ett nät från aktuellt bibliotek.

SaveNet sparar ett nät i aktuellt bibliotek.

InputTune låter användaren mata in en melodi.

InputTone används av InputTune för att mata in en enstaka ton.

SaveTune sparar en melodi i aktuellt bibliotek.

LoadTune laddar en melodi från aktuellt bibliotek.

ChangeEta används för att ändra steglängden.

ChangeTempMult används för att ändra temperaturens beroende av kvadratavvikelsen.

InitAll initierar alla variabler vid programstart.

MenuChoice är funktionen där menyn presenteras och tangentbordet känns av.

Hur jag testade programmet

Att veta om man har lyckats är alltid viktigt. I detta fallet kunde det vara svårt eftersom man med god vilja kan tycka att slumpartade toner av slumpartad längd stundvis liknar något man vill att det ska likna. Därför behövde jag ett mycket enkelt fall att testa programmet på. Jag valde att träna nätet med en skala där varannan ton var lång och varannan kort. Eftersom ingen tveksamhet råder - efter ett C kommer alltid ett D - kommer det vara lätt att avgöra om man lyckets. Samtidigt måste man vara medveten om att detta är ett något lättare problem.

Efter det att jag lyckats med att få datorn att "komponera" skalor, först med ett slumpartat insvängningsförlopp (beroende på minnescellernas slumpvisa innehåll vid starten av komponerandet), och senare med direkt fångande av "den musikaliska idén", gav jag mig på melodier.

Underligheter och olösta problem

När steglängden göres tillräckligt kort får jag ibland en kraftigt oscillerande lösning. Detta är ett problem som bör undersökas närmare. Om man först väljer en stor steglängd och sedan, när systemet börjar oscillera, minskar den får man då väldigt långsam konvergens. Jag tror detta beror på att många vikter hunnit ställa om sig mycket fel under den snabba konvergensen.

Jag vet inte vad den optimala temperaturen är. Mitt val av temperatur utgående från kvadratavvikelsen är ofta - men inte alltid - bra.

Jag använde först ett betydligt lägre antal gömda noder än insignalnoder eftersom detta är allmän praksis. Det visade sig emellertid att ungefär samma antal gav bästa möjliga resultat.

En begränsning på programmet är minnets längd, som i sin tur begränsas av att variabelblock i Turbo Pascal får uppgå till högst sextiofyra kilobytes. Om längre minne önskas kan man dock allokera minne dynamiskt och sedan bygga en egen vektor. Nackdelen med detta är att programmet blir ytterligare långsammare.

Insignalerna ser ut på följande sätt: De noder som ska aktiveras sätts till ett, de som inte ska aktiveras till minus ett. Jag förväntade mig att en säker positiv utsignal skulle vara ett och en säker negativ skulle vara minus ett, och noll vara gränsen mellan kort och lång i fallet med tonlängden. (Vilken ton som väljes avgörs genom att se vilken utnod som har högst värde.) Detta eftersom min sigmoidfunktion är tangens hyperbolicus, och därför är en udda funktion med störst derivata i noll. Från denna punkt borde lösningarna driva mot ett av de möjliga jämviktslägena plus eller minus ett.

Det visar sig emellertid att om jag kräver att utsignalen ska vara minus ett på alla de noder som inte ska väljas eller t om bara på tonlängdsnoden då tonen inte ska vara lång så konvergerar problemet betydligt sämre, om alls. Jag har nu att en positiv utsignal är ett, medan en negativ är noll. Detta är anledningen till att användaren vid komponeringsfasen måste välja omslagspunkt.

Att denna inte av programmet väljes till halva intervallängden, dvs 0.5 beror på att läget omkring noll är ytterst instabilt och alla lösningar driver därifrån. En bra omslagspunkt ges av ett värde mellan 0.5 och 0.8. Jag har arbetat med detta underliga resultat för att finna en anledning, men har inte lyckats. En tanke jag har haft är att andra konvergenskrav kanske har andra temperaturkrav, men då jag provat olika lösningar för olika temperaturer har det visat sig att minimat ligger på ungerfär samma ställe som innan, men konvergensen blir obetydlig. Detta bör undersökas närmare.

Tekniska fakta

COMPOSER.EXE heter det exekverbara programmet, källkoden som är skriven i Turbo Pascal 5.5 heter COMPOSER.PAS. Jag använder mig av ett programbibliotek kallat Turbo Professional för att enkelt få bra inmatningsrutiner på ställen där jag känner att detta behövs. Uniten jag använder mig av heter TPEDIT.TPU och finns med på disketten så att den som vill kan kompilera om programmet. Jag har kompilerat programmet med Range Check Off för min egen bekvämlighets skull.

Referenser

Mer finns att läsa om neurala nät, backpropagation och gradientmetoden i

- Apprentices of Wonder, William F. Allman
- Neurocomputing, Robert Hecht-Nielsen
- Adaptive Control, Karl Johan Åström & Björn Wittenmark

202

Neuronnät i regulatorer

Tomas Carlsson
och
Tomas Almgren

Lund 29 maj 1991

Reglerteknik, Lunds Tekniska Högskola

Innehåll

204

1 Inledning

Denna artikel beskriver funktionen hos en neuronnättyp och hur ett neuronnät efter inlärning kan användas för klassificering av en godtycklig signal. Efter omfattande försök och experiment med sådana 'intelligenta' nät har vi konstruerat en adaptiv PI-regulator där adapteringen bygger på neuronnätets klassificering av det slutna systemets stegsvar. Regulatorn och processen samt neuronnätet har implementerats i SIMNONkod och tillhörande simuleringsresultat har utvärderats. Inlärningsvektorerna till neuronnätet har genererats i MATLAB och konstruktionen av neuronnätet har gjorts medelst ett interaktivt neuronnätverksprogram, NeuralWorks Professional (NPW).

2 Neuronnätet

De neurona nätverken är resultat av försök att konstruera enkla datormodeller av människans hjärna. En neuron är den fundamentala enheten i nätet och utgör en microprocessor som erhåller och kombinerar signaler från många andra neuroner. Signalöverföringen mellan neuronkärnorna ske via dendriter,axoner och synapser. Om den kombinerade insignalen är tillräckligt stark producerar neuronen en utsignal genom axonen till närliggande neuroner.

Hjärnan består av flera miljarder neuroner och den fundamentala minnesmekanismen utgörs av förändrad överförings- effektivitet i synapserna och av ändrad informationsbehandling i neuronkärnan.

I ett artificiellt nätverk är analogin till den biologiska neuronen ett processelement (PE), se figur 1. Den interna funktionen i processelementen är vanligtvis en viktad summering av värdena på dess ingångar. En olinjär funktion av summan genrerar utgångsvärdet enligt:

$$I_j = \sum_i^n (x_i * w_{ji}) \tag{1}$$

$$y_j = f(I_j) \tag{2}$$

Processelementen är tämligen ointressanta i sig själv utan det är först när processelementen sammankopplas som effekterna blir intressanta. Ett typiskt nätverk består av flera lager av processelement där varje lager är i full eller slumpmässig förbindelse med nästa lager. Det finns också en inbuffert till vilken data presenteras för nätet och en utbuffert som håller nätresultatet. Det finns följdaktligen utrymme för en hel del fantasi när man konstruerar sitt nätverk!

Vid våra försök att identifiera process-stegsvar lät vi neuronnätets inbuffert bestå av 20 stycken PE, motsvarande 20 sampel av stegsvaret, ett mellanlager med fem PE och en utbuffert med två eller fyra PE. Se figur 2.

2.1 Inlärningsalgoritmen

Den typ av neuronnät som vi arbetat med arbetar i två faser – inlärningsfas och återskapandefas. Inlärningen kan ske på många olika sätt men huvudsakligen specificerar inlärningsalgoritmen hur vikterna ska adapteras efter givna inlärningsexempel. Vid inlärningen jämförs, för en viss insignal, den önskade

utsignalen med nätets utsignal. Felet mellan önskad utsignal och aktuell utsignal återkopplas till vikterna för uppdatering så att en global energifunktion minimeras. Uppdateringen sker lager för lager från utbufferten till inbufferten. Denna typ av uppdateringsalgoritm kallas back-propagation. Hela mängden inlärningspar blandas slumpmässigt innan den presenteras för nätet. Inlärningen utförs sedan tills variansen hos utsignalfelet (energifunktionen) konvergerat och förhoppningsvis blivit liten. (Om mängden inlärningsdata är stor krävs väsentlig datorkraft för att beräkningarna inte ska ta alltför lång tid.)

2.2 Back-propagation som inlärningsalgoritm

Varje processelement har följande överföringsfunktion:

$$x_j^s = f(\sum_i w_{ji}^s * x_i^{s-1}) = f(I_j^s) \tag{3}$$

,där f är någon långsamt monotont växande funktion, tex tangenshyperbolicus:

$$f(z) = \frac{e^z + e^{-z}}{e^z + e^{-z}} \tag{4}$$

, och

x_j^s	utvärde från den j:te PE i lager s
w_{ji}^s	vikt mellan i:te PE i lager (s-1) och j:te PE i lager s
I_j^s	viktad summa av ingångarna till j:te PE i lager s

2.3 Bakåtpropagering av det lokala felet

Nätet har som sagts en global felfunktion E associerat till sig. Den kritiska parametern som 'återkopplas' mellan lagrena definieras av

$$e_j^s = \frac{-\partial E}{\partial I_j^s} \tag{5}$$

,vilken kan tolkas som storleken av det lokala felet vid PE j i lager s.
Kedjeregeln ger ett samband mellan det lokala felet hos en PE i lager s och de lokala felen i lager s+1,

$$e_j^s = f'(I_j^s) * \sum_k (e_k^{s+1} * w_{kj}^{s+1}) \tag{6}$$

Observera att det i (6) finns ett lager ovan lager s.

Om f är en tangens hyperbolicus kan dess derivata skrivas på följande sätt:

$$f'(z) = (1 + f(z)) * (1 - f(z))$$

därför kan ,från (3), ekvation (6) skrivas

$$e_j^s = (1 + x_j^s) * (1 - x_j^s) * \sum_k (e_k^{s+1} * w_{kj}^{s+1})) \tag{7}$$

Summan (7) som används för att bakåt-propagera felen är analog med summan i frammåt-propageringen av insignalen genom nätet. För att bestämma de lokala felen låter man alltså insignalen propagera genom nätet, bestämmer felet vid ut-bufferten och bakåt-propagerar felen lager för lager medelst (7). Multiplicationen av felen med derivatan av överföringsfunktionen skalar felen.

2.4 Minimering av det globala felet E

Med kunskap om det lokala felet kan man modifiera vikterna för att minimera E. Detta kan göras med gradient metoden enligt följande:

$$\Delta w_{ji}^s = -k * \frac{\partial E}{\partial w_{ji}^s} \tag{8}$$

Den partiella derivatan i (8) kan beräknas direkt från de lokala felen enligt (6) och (7). Kedjeregeln ger:

$$\frac{\partial E}{\partial w_{ji}^s} = \frac{\partial E}{\partial I_j^s} * \frac{\partial I_j^s}{\partial w_{ji}^s} = -e_j^s * x_i^{s-1}$$

,tillsammans med (8) erhålls

$$\Delta w_{ji}^s = k * e_j^s * x_i^{s-1}$$

2.5 Det globala felet E

Att en global felfunktion existerar har förutsatts ovan. Värdet av felfunktionen för en given invektor $\bar{i}$, (samplad signal), ges av

$$E = 0.5 * \sum_k (u_k - a_k)^2 \tag{9}$$

,där $\bar{u}$ är önskad utsignal och $\bar{a}$ är aktuell utsignal. Det verkliga lokala felet är alltså $u_k - a_k$ hos det k:te PE i utbufferten och det skalade felet ges av (5), dvs

$$e_k = \frac{-\partial E}{\partial I_k} = \frac{-\partial E}{\partial u_k} * \frac{\partial u_k}{\partial I_k} = (u_k - a_k) * f'(I_k)$$

E i (9) definierar det globala felet hos nätet för ett specifikt in- och utsignalpar $\bar{i}$,$\bar{u}$. En *total* global felfunktion kan definieras som summan av alla mönsterigenkänningsfel. Så modifierar alltså bakåtpropagerings-algoritmen vikterna i nätet för att minska felet hos en speciell komponent av den totala globala felfunktionen.

2.6 Det statiska nätet

Resultatet av den beskrivna inlärningsalgoritmen är ett statiskt nät som ger ett utvärde motsvarande det som gavs vid inlärningen för en given insignal. Till en tidigare okänd insignal kommer nätet att producera en rimlig utsignal erhållen från en olinjär interpollation.

3 Klassificering av stegsvar med neuronnät

Dagens regleralgoritmer kräver alla ett visst mått av information om den okända processen för att fungera väl. Denna information kan erhållas antingen manuellt från en reglertekniker eller automatiskt för helt autonoma regulatorer. En vanlig metod för att erhålla information om en okänd process är att göra ett stegsvarsförsök. Den rutinerade reglerteknikern tittar på stegsvaret och kan genast placera processen i en viss grupp. För att en autonom regulator ska erhålla motsvarande information måste den automatiskt göra motsvarande stegsvarsförsök. Att automatiskt generera stegsvaret är trivialt. Svårigheten ligger i hur man ska tolka processens stegsvar. I industrin har man länge använt tabellerade stegsvar som reglertekninkern jämför med. Problemet kan då identifieras med ett mönsterigenkänningsproblem. Neuronnät har visat sig fungera väl för att lösa denna typ av problem.

Med detta som bakgrund har vi valt att med ett neuronnät identifiera följande fyra stegsvarstyper:

- Monoton stabil
- Dämpad oscillation
- Oscillation
- Instabil

I figur 3 visas exempel på de fyra olika typerna.

3.1 Neuronnätets arkitektur

Insignal till nätet har vi valt som 20 exvidistanta sampel av stegsvaren var och en kopplad till en ingångsneuron i ingångsbufferten. Nätet ska ge en utsignal som talar om vilken grupp som stegsvaret tillhör. Vi har valt att använda 4 utgångar, var och en representerande en grupp vilket ger följande kodning:

Typ av stegsvar	Y1	Y2	Y3	Y4
Monoton stabil	1	0	0	0
Dämpad oscillation	0	1	0	0
Oscillation	0	0	1	0
Instabil	0	0	0	1

Efter lite empirisk forskning valde vi mellanlagrets storlek till 5 neuroner.

3.2 Inlärning

Med hjälp av MATLAB genererade vi samplade stegsvar tillhörande de fyra olika grupperna. Processen som användes har föjande överföringsfunktion $G(s)$:

$$G(s) = \frac{\omega^2}{s^2 + 2\zeta\omega s + \omega^2}$$

Genom att variera ω och ζ genererade vi 80 stycken olika stegsvar med 20 stycken från varje stegsvarsgrupp. Varje stegsvar representeras av en vektor med 20 sampel. I varje vektor sparades också grupptillhörighet enligt ovanstående kodning. Från dessa 80 tog vi slumpmässigt ut fem ur varje grupp och sparade dessa 20 som testvektorer. De övriga 60 utgjorde vårt inlärningsmaterial.

Med hjälp av programmet NeuralWorks Professional (NWP) konstruerades nätet. Neuronernas överförningsfunktion valdas till tanh. NWP kontrollerar automatiskt inlärningsvektorerna och skalar vektorerna så att signalerna ligger inom intervallet -1 till 1. Detta sker för att nätet inte ska mättas. De önskade utsignalerna skalas om till att ligga inom intervallet -0.8 till 0.8. Valet görs så att derivatan av tanh vid ändpunkten inte blir för liten då detta minskar inlärningshastigheten.

Efter knappt 30000 iterationer hade felfunktionen konvergerat till en acceptabel nivå.

3.3 Resultat

Varje utsignal från neuronnätet är en kontinuerlig signal som kan anta värden mellan -1 till 1, eller återskalat 0 till 1. Som utvärderingsregel valde vi att den utgång som gav störst värde angav vilken grupp som neronnätet klassificerade stegsvaret till. Vid test med de 20 testvektorerna kunde nätet till 100% identifiera stegsvaren.

För att analysera nätets prestanda med avseende på bruskänslighet adderade vi normalfördelat brus med hjälp av MATLAB med standardavvikelse σ_e till testvektorerna och erhöll följande resultat:

σ_e	Felprocent
0.02	0%
0.05	5%
0.1	8%
0.2	15%

Ovanstående resultat visar att nätet har goda egenskaper även med ganska brusiga insignaler. Härur kan man dra slutsatsen att det bör finnas god möjlighet att använda neuronnät som identifierare vid konstruktion av adaptiva regulatorer.

4 En adaptiv PI-regulator med neuronnät som identifierare

PI- och PID regulatorerna är den absolut vanligaste regulatorstrukturen. Inställningen är inte matematiskt trivial. Ett av de vanligare sätten att ställa in regulatorn är med ett stegsvars-prov i ett slutet system. Stegsvaret analyseras och parametrarna justeras efter tumregler och erfarenhet.

Vi har gjort ett försök att automatisera inställningen av en PI-regulator. Stegsvaret hos det slutna systemet analyseras av neuronnätet och regulatorparametrarna justeras.

4.1 Regulatorns struktur

Regulatorn kan delas i två delar – regulatordel och parameterinställare. Parameterinställaren innehåller neuronnätet, parametrarnas tillstånd samt logik som styr när uppdatering sker. I figur 4 visas en schematisk bild av systemet.

Regulatorn är en standard PI-regulator som beskrivs med

$$u(t) = K * (u_c(t) - y(t) + \frac{1}{T_i} * \int_0^t u_c(\tau) - y(\tau) d\tau)$$

där u_c är referenssignal och y är processens utsignal. Regulatorns parametrar är K och T_i.

Neuronnätet består av 20 ingångsneuroner, åtta i mellanlagret och två utgångar. Neuronernas överföringsfunktion är tanh. Ingångarna kopplas till ett skiftregister som innehåller 20 sampel av $e(t) = u_c(t) - y(t)$. Neuronnätets två utsignaler anger med en faktor ändringen av respektive parameter. Här använder vi alltså neuronens kontinuerliga utsignal direkt.

Adaptering startar när $|e(t_0)| > A_{lim}$ vilket indikerar ett steg in vid tiden t_0. A_{lim} väljes tillräckligt stor så att brus och laststörningar inte startar en ny adaptering. Värdet av $e(t_0)$ sparas för att normera alla sampel. 20 punkter skiftas in i skiftregistret med ekvidistant tidsavstånd h_n. Vilket ger följande insignaler till nätets ingångsbuffert.

$$x_1 = e(t_0 + h_n)/e(t_0)$$

$$x_2 = e(t_0 + 2h_n)/e(t_0)$$

$$\ldots$$

$$x_{20} = e(t_0 + 20h_n)/e(t_0)$$

Neuronnätets två utgångar Y_1 och Y_2 uppdaterar sedan K och T_i enligt:

$$K(k+1) = K(k) * \lambda^{Y_1}$$

$$T_i(k+1) = T_i(k) * \lambda^{Y_2}$$

Valet av nätets samplingstid h_n är mycket viktig. Efter ett steg på insignalen ska $e(t)$ innehålla relevant information under tiden $t = t_0$ till $t = t_0 + 20h_n$. Detta

kan tolkas som att kvoten mellan processens stigtid och h_n ska vara samma som när inlärningsdata till nätet skapades. Adapteringskonstanten λ valde vi som $\lambda = \sqrt{1.6}$ vilket ska jämföras med det värde som användes vid inlärningen ($\sqrt{2}$), se nedan. Valet gjorde att parametrarnas konvergenshastighet blev lagom stor.

4.2 Inlärning av neuronnätet

Nätets önskade utsignal ska vara en faktor som talar om hur respektive regulatorparameter ska ändras. Detta implicerar att vi ska lära upp nätet med testvektorer som är genererade av ett slutet system med en medvetet felaktigt inställd PI-regulator.

En process $G(s)$ regleras med en PI-regulator.

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

Parametrarna K och T_i ställs in så att det slutna systemets stegsvar ser 'bra' ut vilket ger K_0 och T_{i0}.

I MATLAB genereras inlärningsvektorer genom att flytta regulatorns parametrar från sina 'bra' värden. För genereringen använde vi en geometrisk serie med kvot $\sqrt{2}$ för att förflytta parametrarna från sina 'bra' värden. Följande parametervärden användes:

$$K = K_0 * (\sqrt{2})^i, \quad -4 \leq i \leq 4$$

$$K = T_{i0} * (\sqrt{2})^j, \quad -4 \leq j \leq 4$$

I testvektorn lagrades förutom de 20 samplen av $e(t)$ även i och j. I figur refsteg visas exempel på tre stegsvar med bra, snabb och långsam inställning av regulatorn.

Efter inlärning i NWP och lite drygt 30000 iterationer konvergerade det globala felet till en acceptabel nivå. Kontroll av prestanda gav vid handen att de två utsignalerna maximalt devierade 10% från önskat värde.

Efter inlärning lät vi NWP generera C-kod för datorimplementering av nätets funktion.

4.3 Implementering i SIMNON

Den av NWP genererade C-koden översattes för hand till ett diskret SIMNON system och adaptionslogiken lades till. PI-regulatorn beskrevs som ett kontinuerligt system enligt ekvationerna ovan men med tillägg för att undvika integratoruppvridning.

4.4 Simulering

Med hjälp av SIMNON utvärderade vi den adaptiva PI-regulatorn genom att använda processer med olika ordningstal, tidskonstanter och stationär förstärkning.

4.4.1 Andra ordningens process

Som process använde vi följande överföringsfunktion:

$$G(s) = \frac{b\omega^2}{s^2 + 2\zeta\omega + \omega^2}$$

Vid inlärningen använde vi denna process med $b = 1$, $\omega = 1$ och $\zeta = 1$. I figur 6 finns resultatet av simuleringen av denna process. Regulatorns parametrar konvergerar till de värden K_0 och T_{i0} som användes vid inlärningen.

Simulering av samma process fast med $b = 5$ visar att regulatorns förstärkning nu konvergerar till $K_0/5$ vilket är helt korrekt. Se figur 7.

Genom att ändra ω hos processen kan man göra processen långsammare eller snabbare. Simulering med $\omega = 0.5$ ger resultat enligt figur 8. Eftersom processen är mycket långsammare än den önskade krävs stora styrsignaler med efterföljande översläng i utsignalen. Neuronnätet prioriterar här snabbheten framför dämpningen av det slutna systemet.

Ett snabbt system med $\omega = 2$ ger resultat enligt figur 9. Processen är här mycket snabbare än det önskade slutna systemet vilket ger en låg förstärkning och en styrsignal u som är långsam.

Med $\zeta = 0.2$, $\omega = 2$ och $b = 1$ erhåller man en process med ett oscillativt svagt dämpat stegsvar. Reglering med regulatorn fungerar inte alls. Parametrarna driver iväg och orsakar temporära självsvängningar, se figur 10.

Sammanfattningsvis fungerar regulatorn bra för de processer som har $\zeta = 1$ och vars förstärkning är okänd. Processens snabbhet speler dock en roll för hur det slutna systemet stegsvar uppför sig.

4.4.2 Tredje ordningens process

För att testa regulatorns robusthet mot omodellerad dynamik använde vi följande process:

$$G(s) = \frac{4}{(0.6s + 1)^3}$$

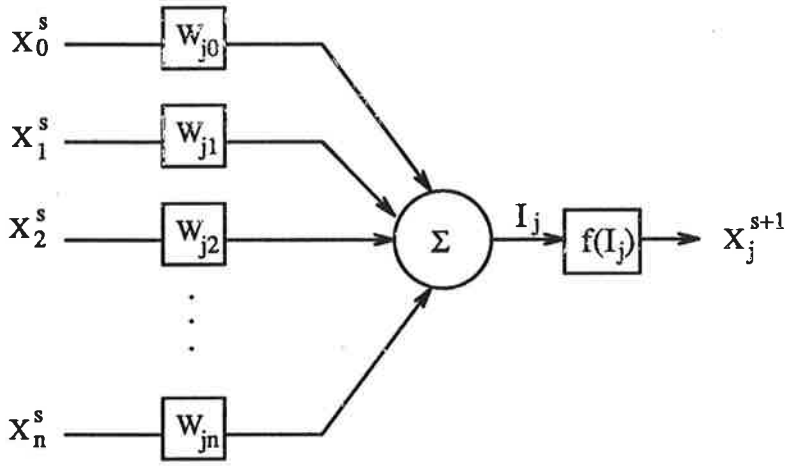
Simuleringen visar att regulatorns parametrar konvergerar. Det slutna stegsvaret är dock oacceptabelt, se figur 11.

4.5 Resultat

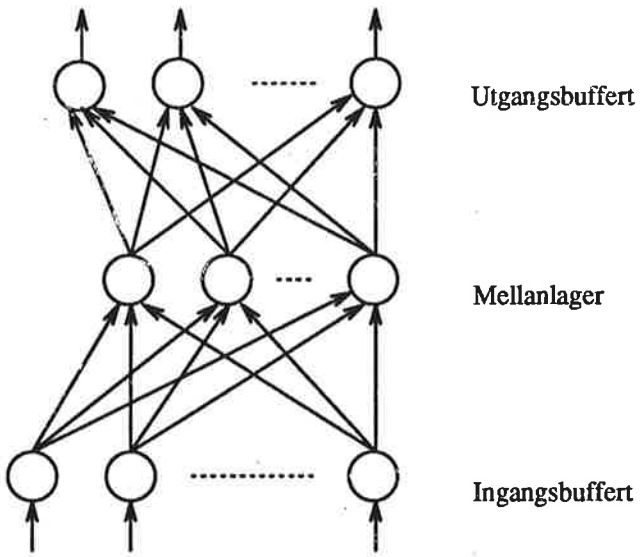
Simuleringen har visat att processens snabbhet har en inverkan på vad den slutliga stationära regulatorn konvergerar till. För att kunna använda denna typen av adaptiv regulator måste man välja neuronnätets samplingstid h_n efter snabbheten hos den process som man ska reglera.

Regulatorn är tämligen robust i det avseendet att regulatorparametrarna i alla fall utom ett har konvergerat till värden som motsvarar ett stabil slutet system.

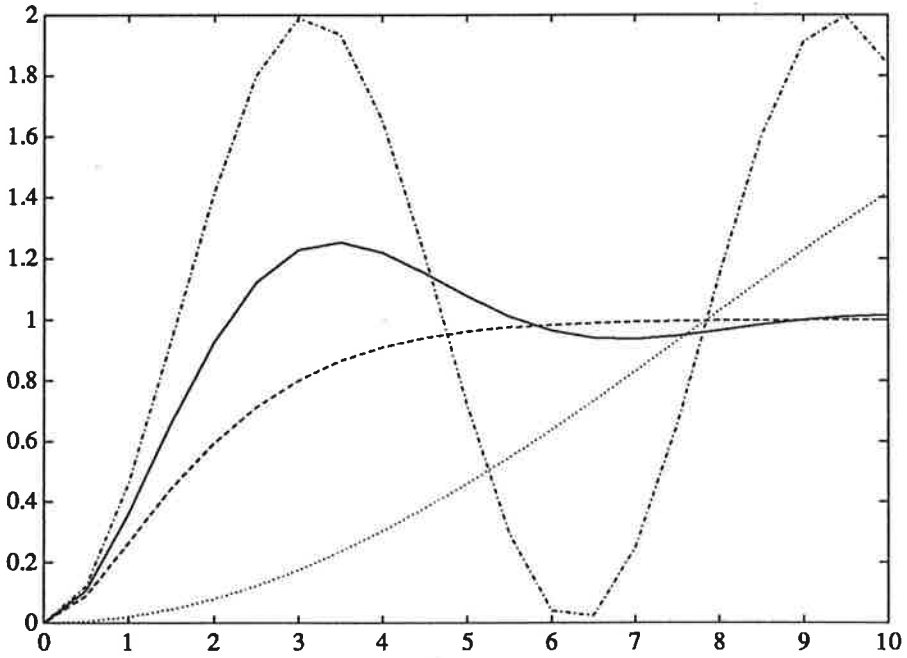
212



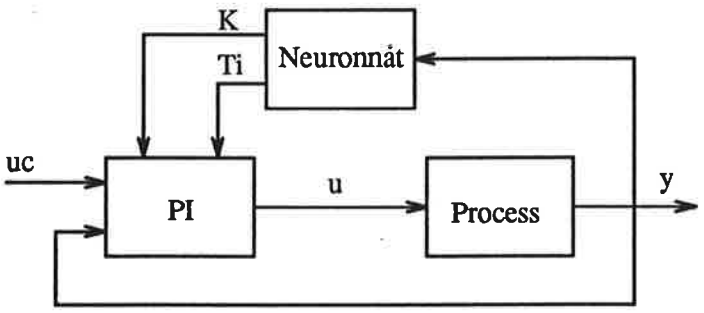
Figur 1: Strukturen hos ett processelement (PE).



Figur 2: Strukturen hos ett neuronnät.

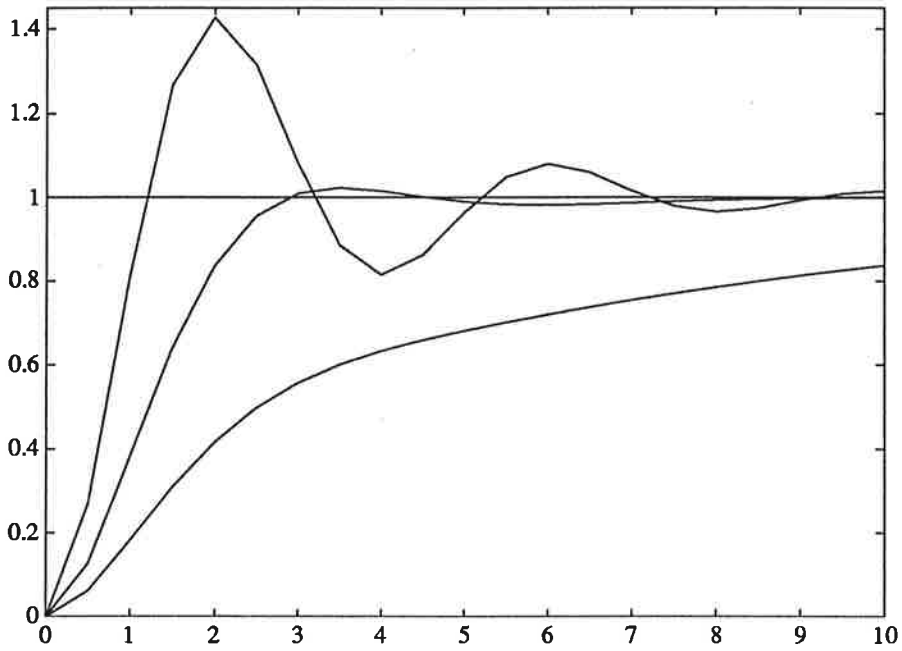


Figur 3: De fyra olika stegsvarstyperna: Monoton stabil (streckad), Dämpad oscillation (hel), Oscillation (streck-prickad) och Instabil (prickad).

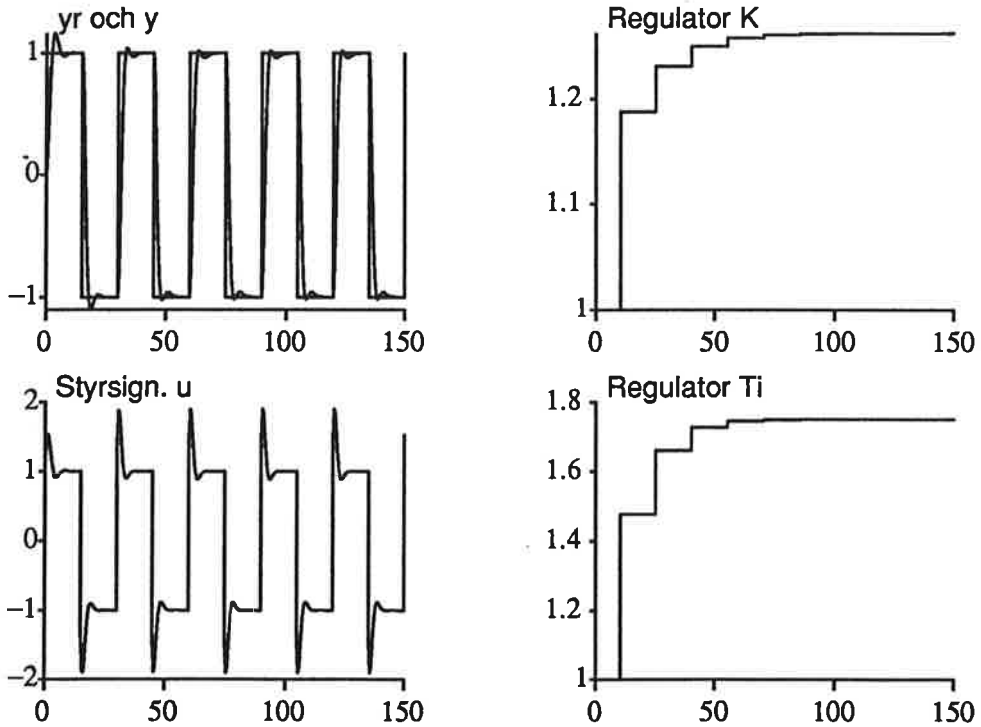


Figur 4: Principiellt blockschema för regulatorn och processen

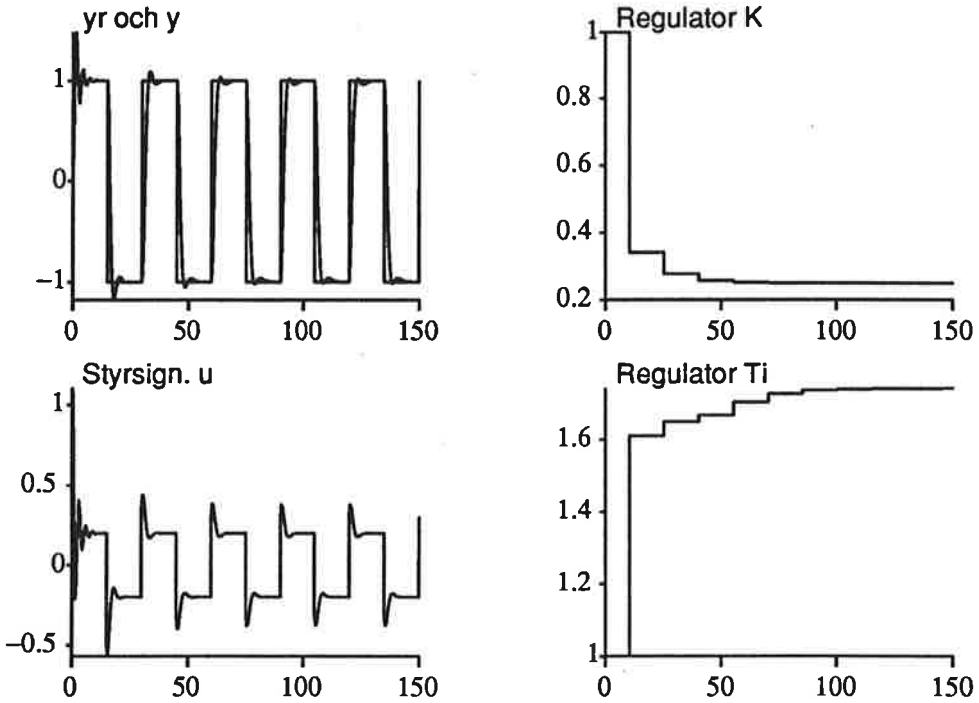
214



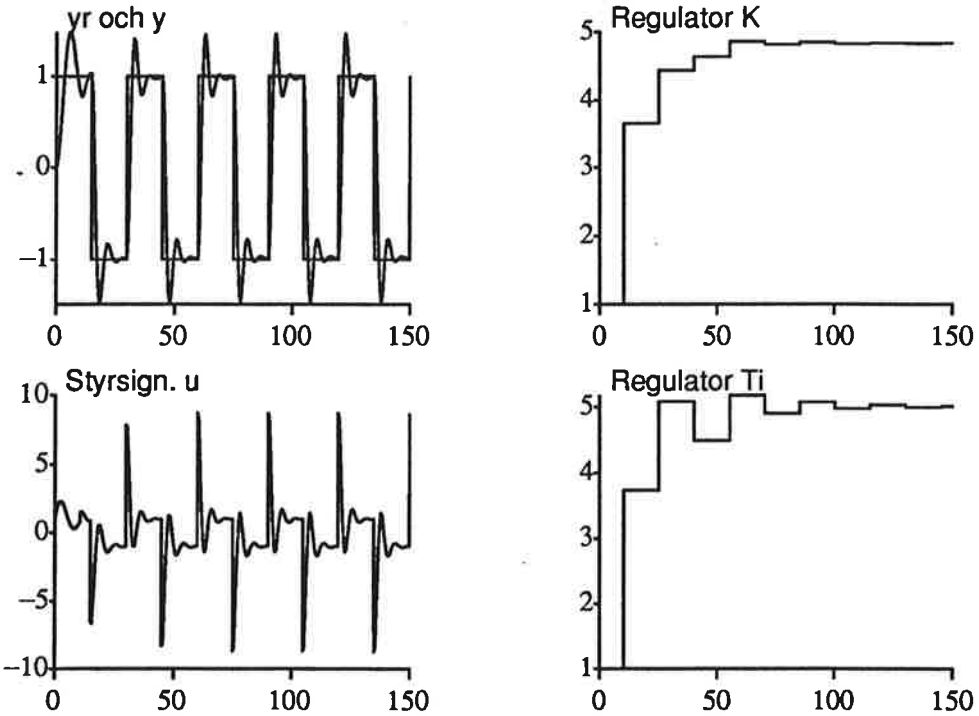
Figur 5: Stegsvar för det slutna systemet när regulatorn har bra, snabb samt långsam inställning



Figur 6: Simulering med processen $G(s) = 1/(s+1)^2$

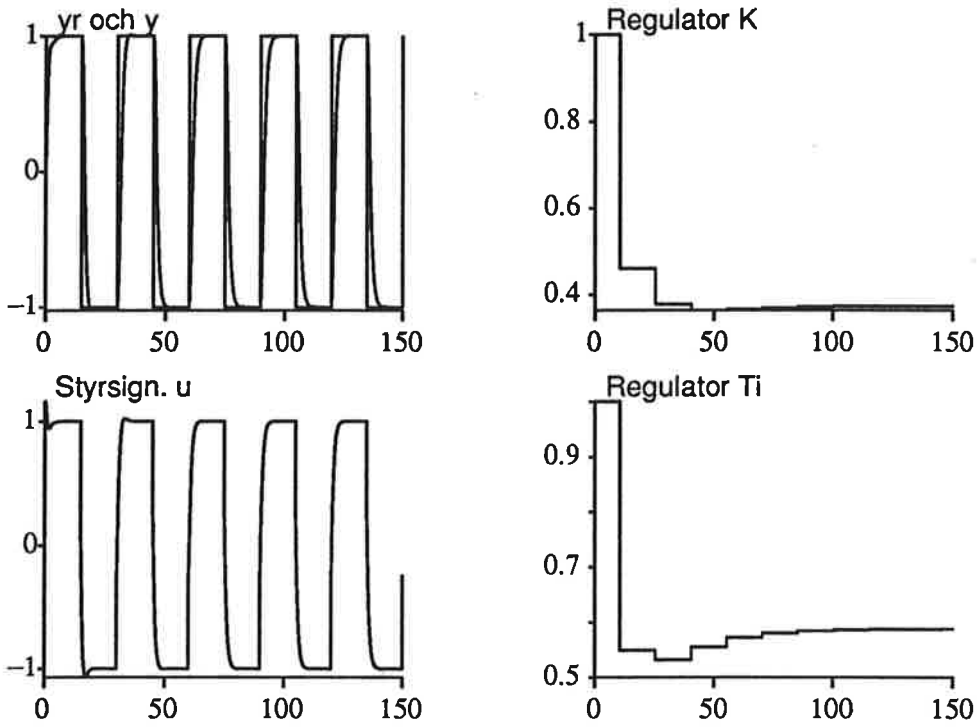


Figur 7: Simulering med processen $G(s) = 5/(s+1)^2$

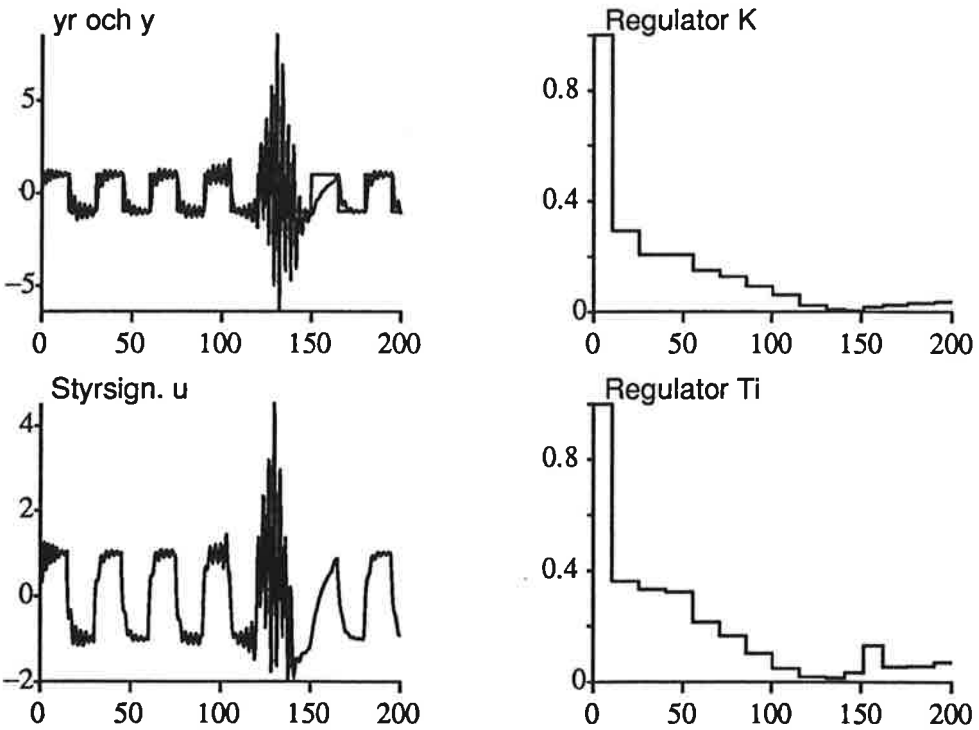


Figur 8: Simulering med processen $G(s) = 1/(2s+1)^2$

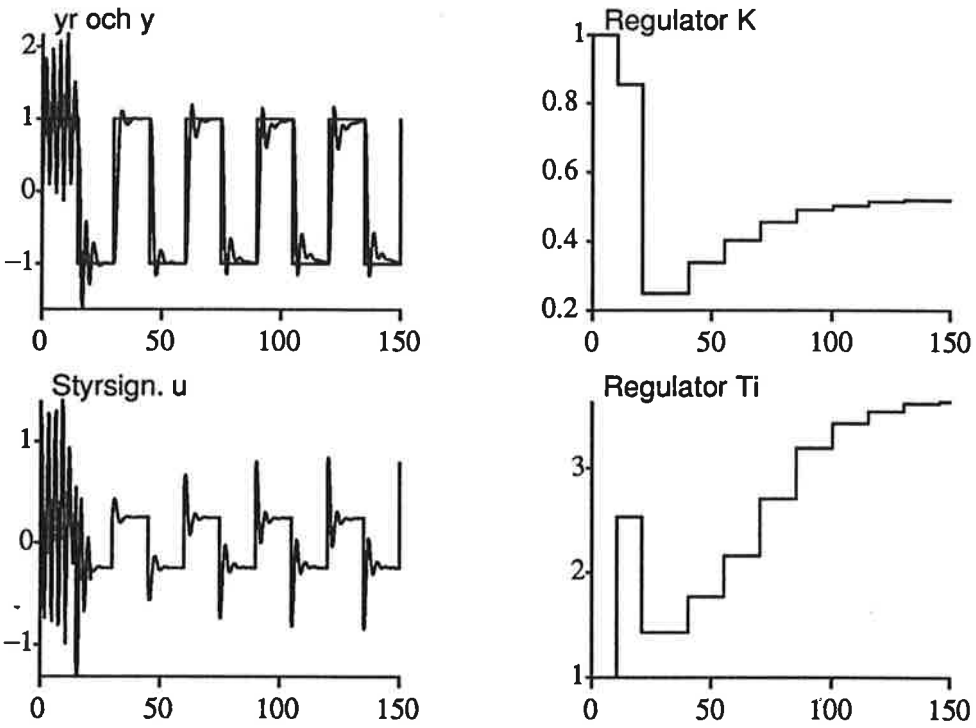
216



Figur 9: Simulering med processen $G(s) = 4/(s+2)^2$



Figur 10: Simulering med processen $G(s) = 1/(s^2 + 0.4s + 4)$



Figur 11: Simulering med processen $G(s) = 4/(0.6s + 1)^3$