Projekt i Adaptiv reglering VT 93

Wittenmark, Björn

1993

Document Version:
Förlagets slutgiltiga version

ISSN 0280–5316
ISRN LUTFD2/TFRT--7508--SE

Projekt i Adaptiv reglering VT 93

Björn Wittenmark
Redaktör

Institutionen för Reglerteknik
Lunds Tekniska Högskola
Juni 1993

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

Document name
INTERNAL REPORT

Date of issue
June 1993

Document Number
ISRN LUTFD2/TFRT--7508--SE

Author(s)
Björn Wittenmark (Editor)

Supervisor

Sponsoring organisation

Title and subtitle
Projekt i Adaptiv reglering VT 1993. (Projects in Adaptive Control)

Abstract

This report contains the project papers from the course Adaptive Control given during the spring semester 1993.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280–5316

ISBN

Language
Swedish

Number of pages
285

Recipient's notes

Security classification

Projekt i Adaptiv Reglering

Vårterminen 1993

Björn Wittenmark
Redaktör

Förord

Kursen "Adaptiv reglering" innehåller en projektdel där teknologerna skall utföra ett projekt under en vecka. Dessa projekt är viktiga inslag i kursen och hjälper institutionen att **utbilda elever med en stark teoretisk grund och en god ingenjörsmässig förmåga.** Denna rapport innehåller redogörelser för de projekt, som genomfördes under vårterminen 1993. Som lärare och handledare har det varit mycket tillfredsställande att se hur bra studenterna är att genomföra projekt och att på den relativt korta tiden kunna prestera goda resultat.

Teknologerna kan välja projekt från ett antal förslag, som institutionen ger. De kan också själva komma med egna förslag. Projekten genomförs individuellt eller i grupp. Projekten kan också kombineras med projekt i kursen "Realtidssystem", som går under samma termin. Flera av årets projekt var av detta slag. Många av projekten är "öppna" i den meningen att det inte finns ett färdigt svar eller lösning till det förelagda problemet. Detta medför att projekten inte alltid kan genomföras enligt de ursprungliga planerna. Detta ger emellertid en viss realism åt projektarbetena.

Projekten redovisas i en kort rapport, samt med en muntlig redovisning eller demonstration. Under våren 1993 genomfördes 16 projekt, varav 7 var gemensamt med Realtidssystem, dessa projekt är markerade med * i innehållsförteckningen.

Björn Wittenmark

Innehållsförteckning

* Gemensamt projekt med Realtidssystem

Adaptive Control 1993

A project on Extremum Control

Performed by Per Tunestål

Tutored by Björn Wittenmark

Department of Automatic Control
Lund Institute of Technology
May 1993

Extremum Control

Goal: To compare some methods of adaptive extremum control.

1. Introduction

A control problem normally concerns controlling a process around a given reference value. In extremum control the reference value is not given. Instead the goal is to find the reference value that minimizes/maximizes some property of the system. This property can be the measurement signal, but can as well be a derived value such as efficiency. Extremum control is only relevant if the static response from the input to the output has at least one extremum. The task of an extremum controller is to make the output follow this extremum.

Extremum control has many applications. In spark ignition engines the torque has a maximum for a certain spark time, the MBT (Maximum Brake Torque) time. A certain Air/Fuel ratio gives maximum efficiency.

In this paper two types of extremum controllers are compared. One controller is based on the method of stochastic approximation from numerical optimization, and the other is based on the idea of self tuning control.

The system investigated in this paper is a time discrete system of Hammerstein type

$$A^*(q^{-1})y(t) = B_0^* + B_1^*(q^{-1})u(t-1) + B_2^*(q^{-1})u^2(t-1) + C^*(q^{-1})e(t) \quad (1.1)$$

where B_0^* is a constant.

2. Stochastic Approximation

Concider a static nonlinear system corrupted by noise with zero mean and bounded variance.

$$y(u,t) = f(u) + e(t) \quad (2.1)$$

Now assume that f has one minimum for some $u=u_0$. This implies that f_u is a nondecreasing function, and that

$$f_u(u_0) = 0 \quad (2.2)$$

One way of finding the minimum is to measure f_u, and in each step update u with an amount proportional to f_u in the direcction opposite to f_u.

$$u(t+1) = u(t) - \alpha(t) f_u(u(t)) \quad (2.3)$$

It can be shown that if the gain sequence $\alpha(t)$ is chosen such that

$$\lim_{t\to\infty} \alpha(t) = 0 \tag{2.4}$$

$$\sum_{t=1}^{\infty} \alpha(t) = \infty \tag{2.5}$$

$$\sum_{t=1}^{\infty} \alpha(t)^2 < \infty \tag{2.6}$$

then algorithm (2.3) will converge. (2.4) is necessary for convergence due to the noise present in the measured signal. Condition (2.5) assures that the algorithm converges to the right value, and (2.6) is necessary to guarrantee convergence.

One problem is that f_u usually isn't measurable. This is where the method of stochastic approximation enters the picture. f_u can be approximated with the central difference

$$\hat{f}_u(u_k, \beta_k) = \frac{y(u_k + \beta_k, 2k) - y(u_k - \beta_k, 2k+1)}{2\beta_k} \tag{2.7}$$

The algorithm is now modified to

$$u_{k+1} = u_k - \alpha_k \hat{f}_u(u_k, \beta_k) \tag{2.8}$$

Necessary conditions for convergence are (2.4) - (2.6) and

$$\sum_{k=1}^{\infty} (\alpha_k / \beta_k)^2 < \infty \tag{2.9}$$

The method described above is designed for static systems. The method can, however, be applied to dynamic systems as well. In the dynamic case it takes some time before a perturbation in the control signal can be seen in the output. This means that the sampling interval of the controller must be of the same order of magnitude as the slowest time constant of the process.

3. Model Based Extremum Control

In the model based approach the parameters of the Hammerstein model (1.1) are identified recursively with the extended least squares (ELS) method. The identified parameters are then used to calculate the constant control signal that gives minimal output in stationarity. The algorithm for estimating the process parameters with ELS is given by

$$
\begin{aligned}
\hat{y}(t) &= \varphi^T(t-1)\hat{\theta}(t-1) \\
\hat{\theta}(t) &= \hat{\theta}(t-1) + K(t)(y(t) - \hat{y}(t)) \\
K(t) &= P(t)\varphi(t-1)(\lambda + \varphi^T(t-1)P(t)\varphi(t-1))^{-1} \\
P(t+1) &= (P(t) - K(t)\varphi^T(t-1)P(t))/\lambda
\end{aligned}
\tag{3.1}
$$

For the Hammerstein model (1.1) the regressors and parameters are given by

$$
\begin{aligned}
\varphi^T(t-1) &= [-y(t-1) \quad \dots \quad -y(t-n_a) \quad u(t-1) \quad \dots \quad u(t-n_{b1-1}) \\
&\qquad u^2(t-1) \quad \dots \quad u^2(t-n_{b2-1}) \quad \hat{e}(t-1) \quad \dots \quad \hat{e}(t-n_c) \quad 1] \\
\theta^T &= [a_1 \quad \dots \quad a_{na} \quad b_{10} \quad \dots \quad b_{1nb1} \quad b_{20} \quad \dots \quad b_{2nb2} \quad c_1 \quad \dots \quad c_{nc} \quad b_0] \\
\hat{e}(t) &= y(t) - \varphi^T(t-1)\hat{\theta}(t)
\end{aligned}
\tag{3.2}
$$

In stationarity the (1.1) is given by

$$
A^*(1)y_0 = B_0^* + B_1^*(1)u_0 + B_2^*(1)u_0^2 \tag{3.3}
$$

The constant controller that minimizes y_0 is thus given by

$$
u_e = -\frac{B_1^*(1)}{2B_2^*(1)} \tag{3.4}
$$

This controller gives an output mean value of

$$
Ey = \frac{B_0^*}{A^*(1)} - \frac{B_1^*(1)^2}{4A^*(1)B_2^*(1)} \tag{3.5}
$$

Since there is no feedback, the output variance will be the same as the open loop variance.

4. Simulations

The system to be studied in the simulations is described by the model (1.1) with

$$
\begin{aligned}
&A^*(q^{-1}) = 1 + aq^{-1} = 1 - 0.9q^{-1} \\
&B_0^* = b_0 = 0.25 \\
&B_1^*(q^{-1}) = b_{10} + b_{11}q^{-1} = 0.5 - 0.25q^{-1} \\
&B_2^*(q^{-1}) = b_{20} = 0.5 \\
&C^*(q^{-1}) = 1 + cq^{-1} = 1 - 0.5q^{-1}
\end{aligned}
\tag{4.1}
$$

where $e(t)$ has the standard deviation $\sigma = 0.2$. This means that the minimum stationary output value given by (3.5) is

$$\min Ey = 2.19 \tag{4.2}$$

This is achieved by using the constant controller given by (3.4)

$$u_e = -0.25 \tag{4.3}$$

The loss presented in the simulations is defined as

$$L(k) = L(k-1) + \begin{cases} y - y_{opt}, y > y_{opt} \\ 0, y \le y_{opt} \end{cases}$$

where y_{opt} is the optimal output in stationarity.

In the case of Stochastic approximation, experiments have shown that for the studied process the sampling interval in the controller should be about 9 times the sampling intervall of the process. As is seen below this is about half the rise time of the process. It is also seen that the LP-Filter used in the simulations is about twice as fast as the process. Simulations have been made for the different controllers with 7 different noise sequences. Simulations with the first two noise sequences are presented below. Loss per sample for samples 1000 - 10000 are tabulated for all noise sequences in table 4.1.

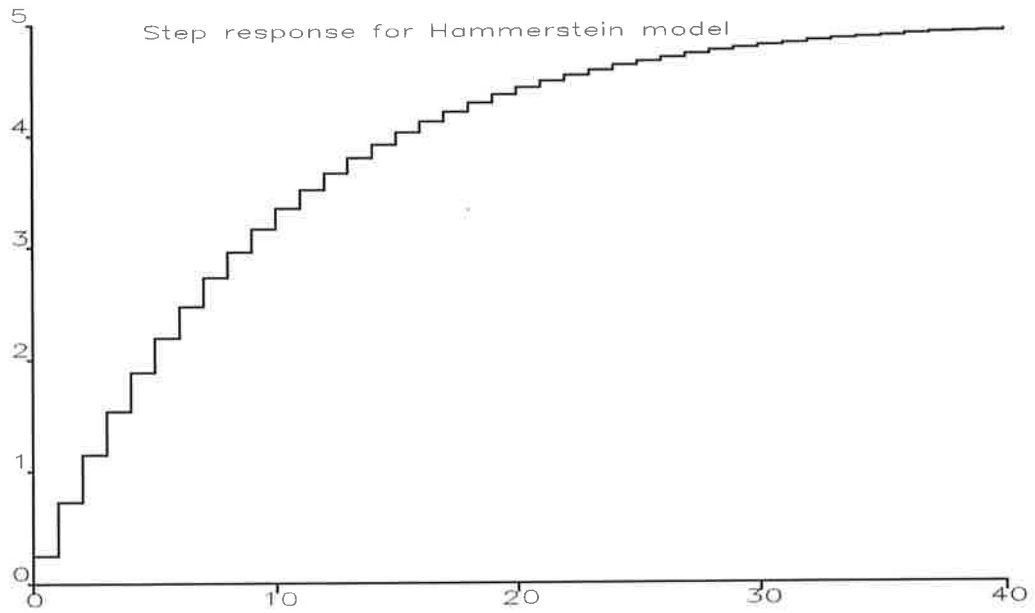


Fig. 1: Step response of the studied process.

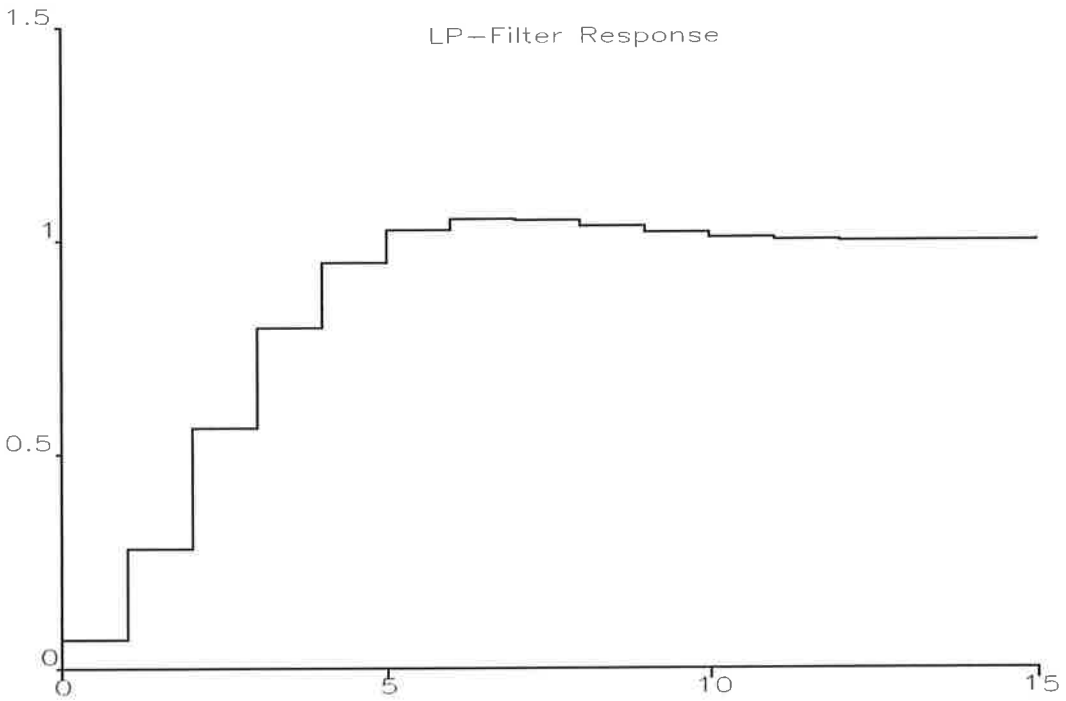
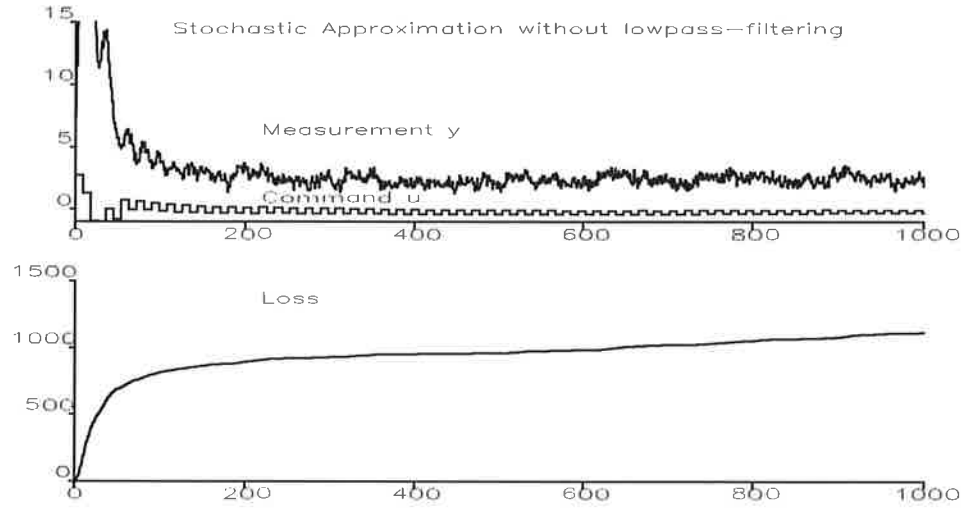
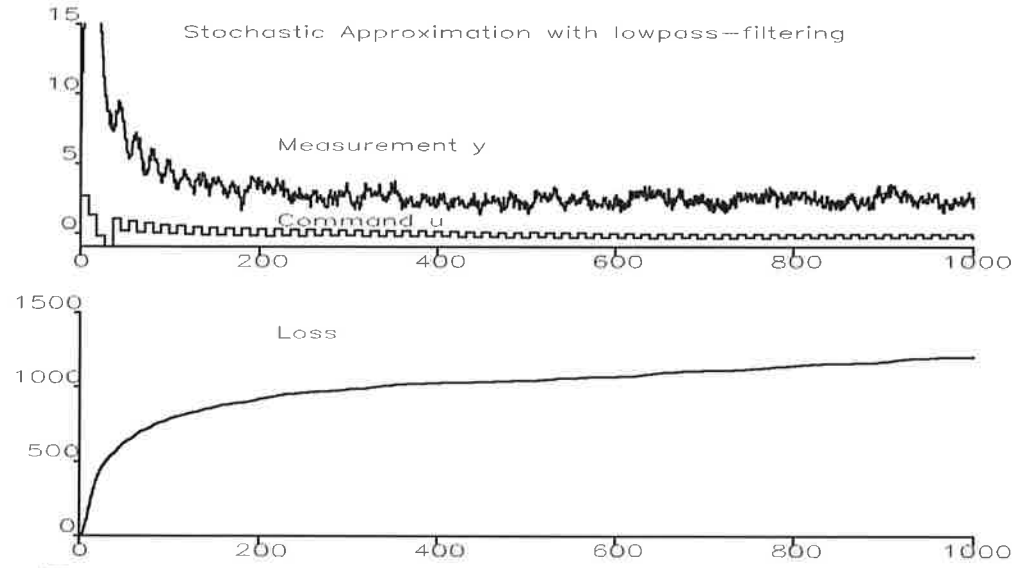


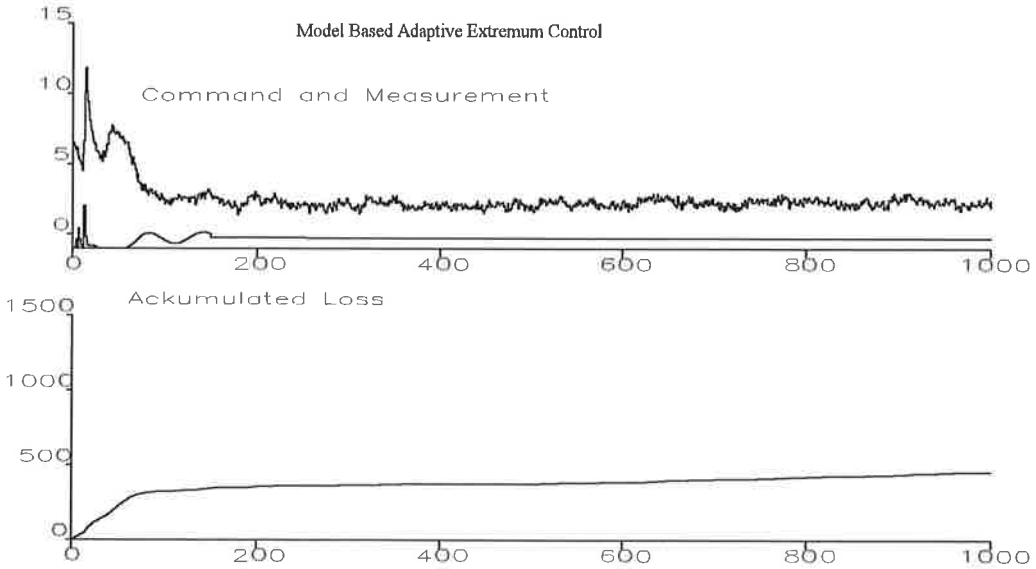
Fig. 2: Step response of the LP-Filter.

Noise Sequence 1 (sample 0 - 1000):

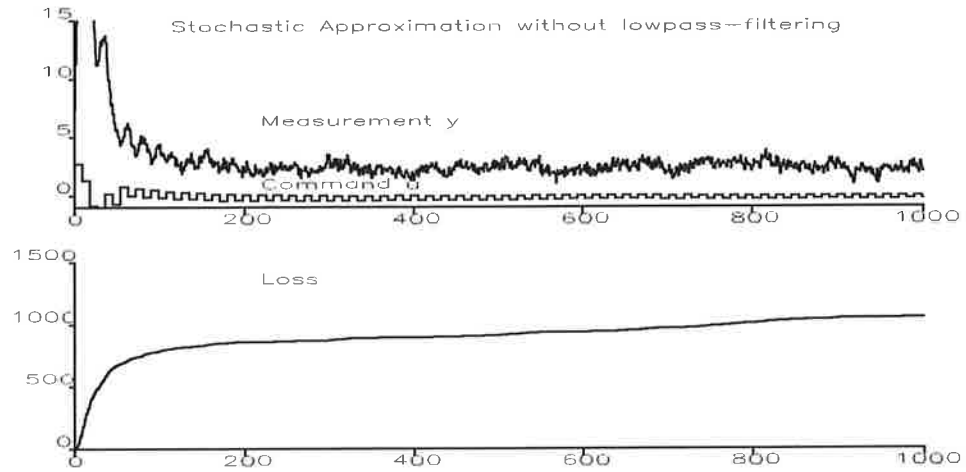


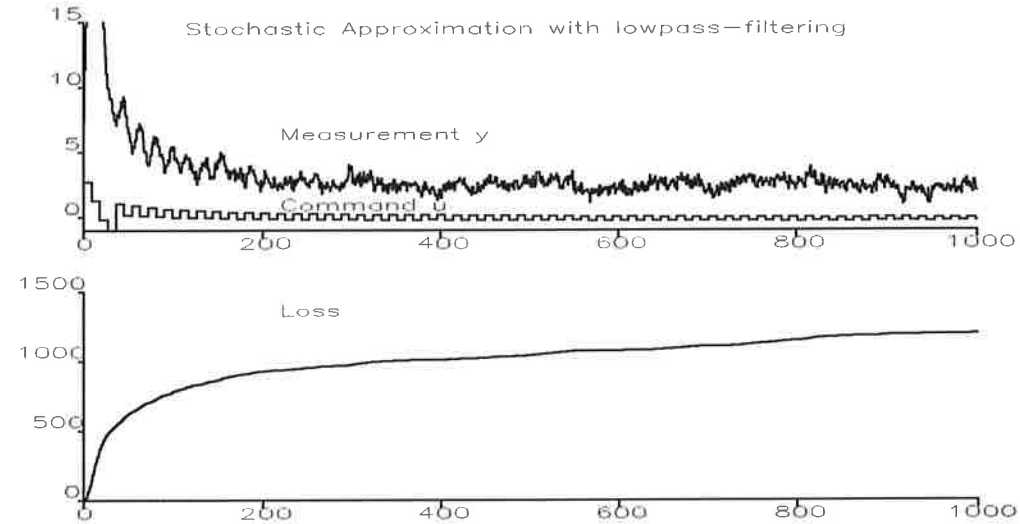


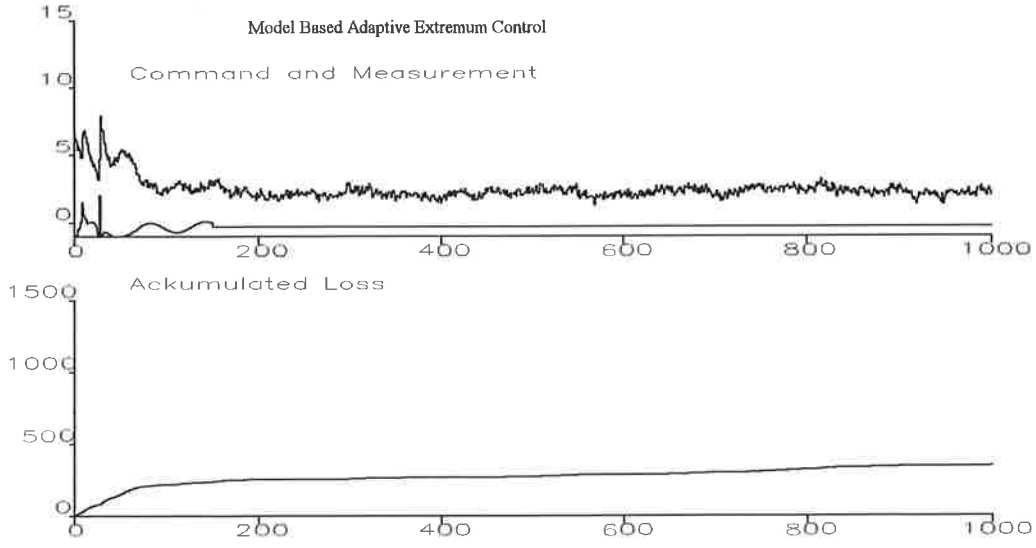
Model Based Adaptive Extremum Control



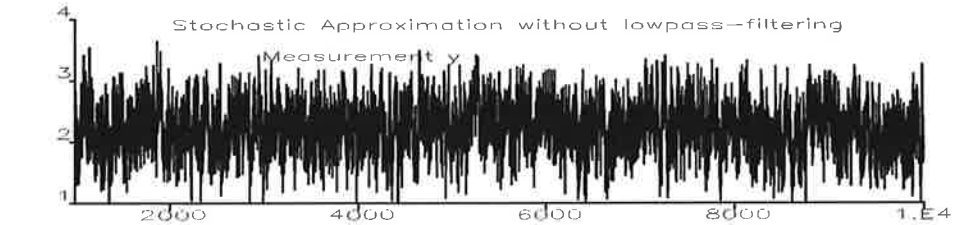
Noise Sequence 2 (Sample 0 - 1000):

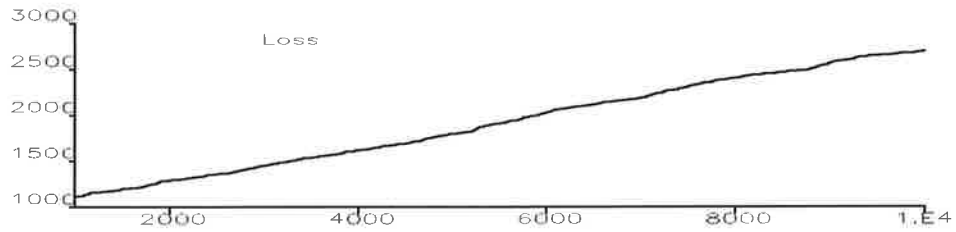


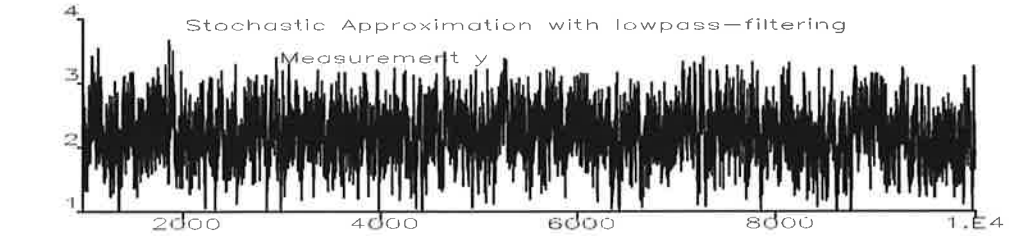


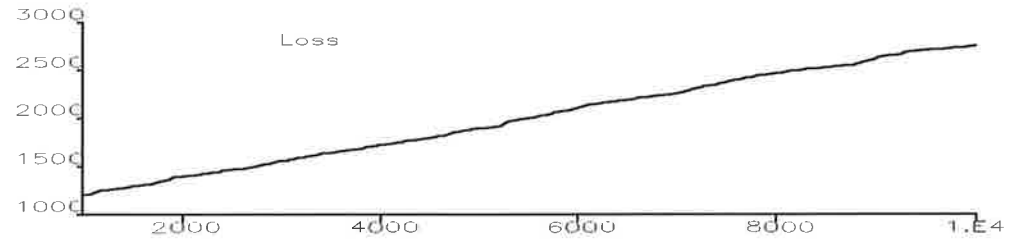


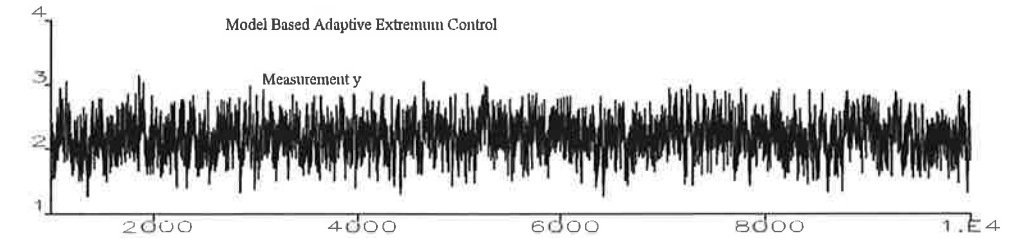
Noise Sequense 1 (Sample 1000 - 10000):

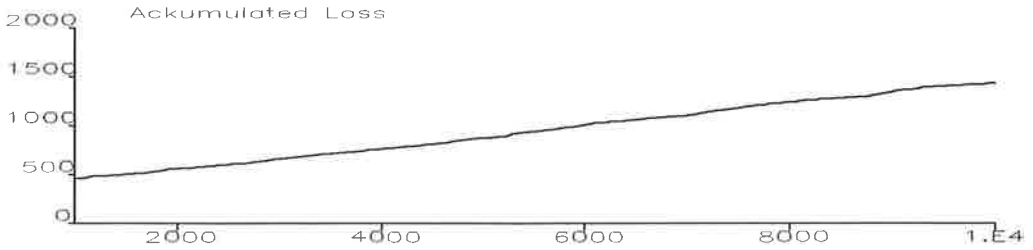




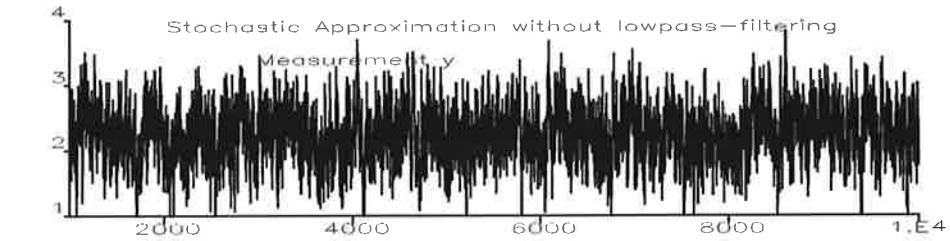


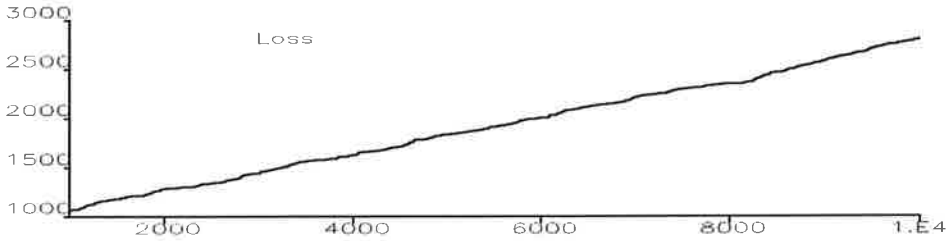


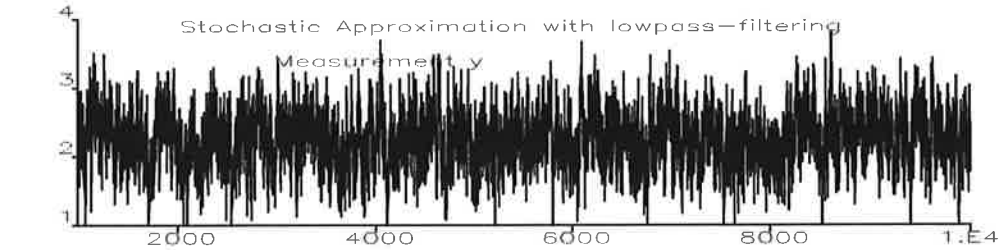


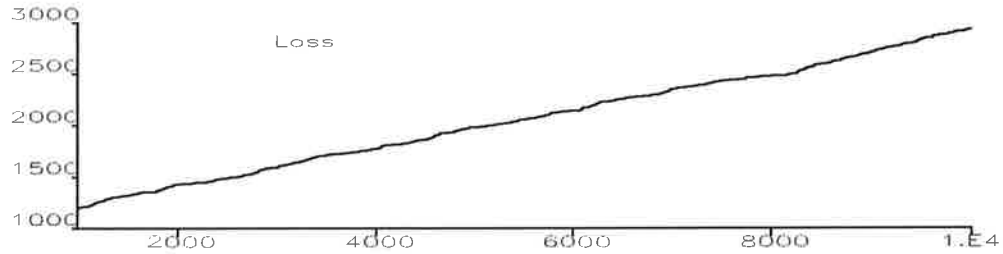


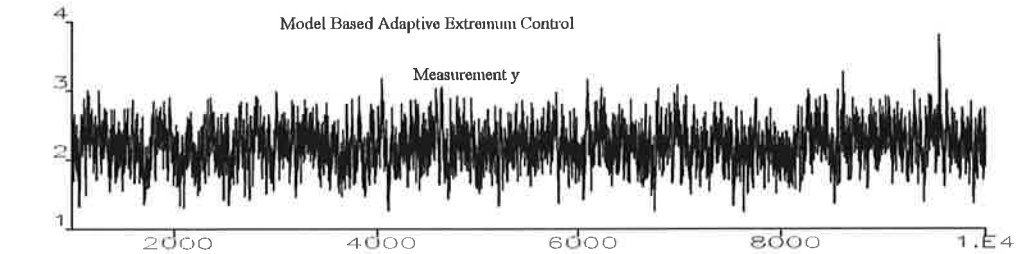
Noise Sequence 2 (Sample 1000 - 10000):

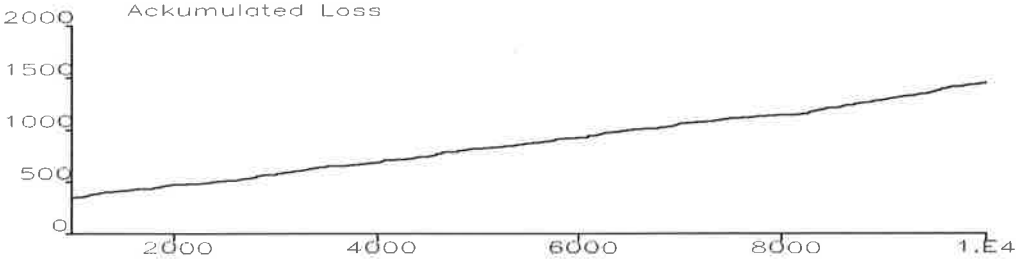












	Without LP-Filter	With LP_Filter	Adaptive
Noise sequence 1	0.176	0.172	0.109
Noise sequence 2	0.194	0.193	0.123
Noise sequence 3	0.182	0.176	0.112
Noise sequence 4	0.196	0.186	0.125
Noise sequence 5	0.186	0.188	0.120
Noise sequence 6	0.167	0.178	0.106
Noise sequence 7	0.178	0.173	0.110
Average	**0.183**	**0.181**	**0.115**

Table 4.1: Loss per sample for different methods and noise sequences.

5. Conclusions

The simulations show that the LP-filtering of the process output in the case of stochastic approximation doesn't significantly improve the convergence to the optimal output. The slopes of the loss function is almost identical for all the cases of stochastic approximation. A possible reason for this is that the errors made in the numerical differentiation due to noise are averaged out by the method itself and the LP-character of the process, thus the filtering isn't needed. After a short period of excitation the model based controller converges to the optimal value. The loss function for the model based controller has only about half the slope compared with the controllers based on stochastic approximation.

References:

"Model Based Adaptive Control. A Nonlinear Example."
Björn Wittenmark
Presented at 2nd Workshop on Adaptive Control, Cancun, Mexico, December 9 - 11, 1992

"Extremum Control"
Björn Wittenmark

Extremalsökande adaptiv reglering
Projekt i Adaptiv Reglering vt 93

Handledare: Björn Wittenmark

Fredrik Bistedt E89
Tony Sandberg E87

INLEDNING

Extremalsökande reglering används då man vill reglera utsignalen från en process så att denna ligger på en extrempunkt, t ex hög verkningsgrad eller lägsta bensinförbrukning hos en bensinmotor.

I ett extremalsökande reglersystem har den statiska utsignalen relaterad till insignal minst en extrempunkt, dvs maximum eller minimum. Detta betyder att systemet är olinjärt.
Extremalsökande reglering är nära besläktad med de optimeringstekniker som finns idag och många av idéerna har kommit från numerisk optimering.

Detta projekt kommer att behandla två sätt att för att hitta extrempunkten hos en process nämligen:

- Numerisk optimering
- Beräkning av styrsignal baserad på estimerade process-parametrar.

Projektet kommer naturligtvis inte att vara helt uttömmande angående ämnet med ger en god inblick i problem och tillvägagångssätt.

PROCESSBESKRIVNING

Processen som används i simuleringarna är en *Hammerstein process* . En typisk diskret Hammerstein process är

$$A^*(q^{-1})y(t) = B^*(q^{-1})f(u(t)) + C^*(q^{-1})e(t)$$

där $A^*(q^{-1})$, $B^*(q^{-1})$ och $C^*(q^{-1})$ är polynom i bakåtskiftoperatorn q^{-1}, f är en olinjär funktion och e är Gaussiskt vitt brus.
I den fortsatta analysen och i simuleringarna kommer vi att använda följande enkla olinjära stokastiska system

$$A^*(q^{-1})y(t) = B_0^* + B_1^*(q^{-1})u(t-1) + B_2^*(q)u^2(t-1) + C^*(q^{-1})e(t) \qquad (1)$$

B_0 är en konstant. Vidare förutsätts att första koefficienten i B_2^* är positiv, dvs > 0. Modellen har den fördelen att den är linjär i parametrarna.

Vid simulering används modellen (1) med polynomen

$$\begin{aligned} A^*(q^{-1}) &= 1 + aq^{-1} = 1 - 0.9q^{-1} \\ B_0^* &= b_0 = 0.25 \\ B_1^*(q^{-1}) &= b_{10} + b_{11}q^{-1} = 0.5 - 0.25q^{-1} \\ B_2^*(q^{-1}) &= b_{20} = 0.5 \\ C^*(q^{-1}) &= 1 + cq^{-1} = 1 - 0.5q^{-1} \end{aligned} \qquad (2)$$

Processen har följande stegsvar

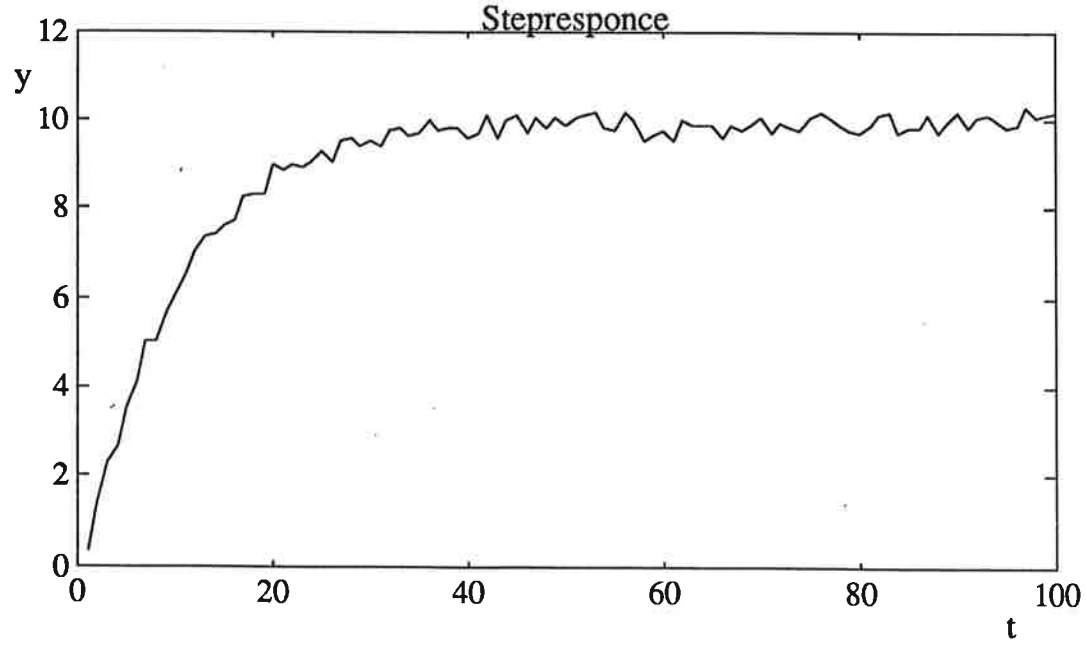


Fig1. Processtegsvar

ANALYS

Analysen av ovanstående process bygger på två olika metoder, dels att rent numeriskt söka extrempunkten dels genom att estimera processparametrarna och utifrån detta beräkna styrsignalen till processen.

Simuleras processen som ges av (2), se fig 2, finner man ett minima då u=-0.25, utsignalen y=2.20, samma värden fås då processens utsignal minimeras map styrsignalen analytiskt.

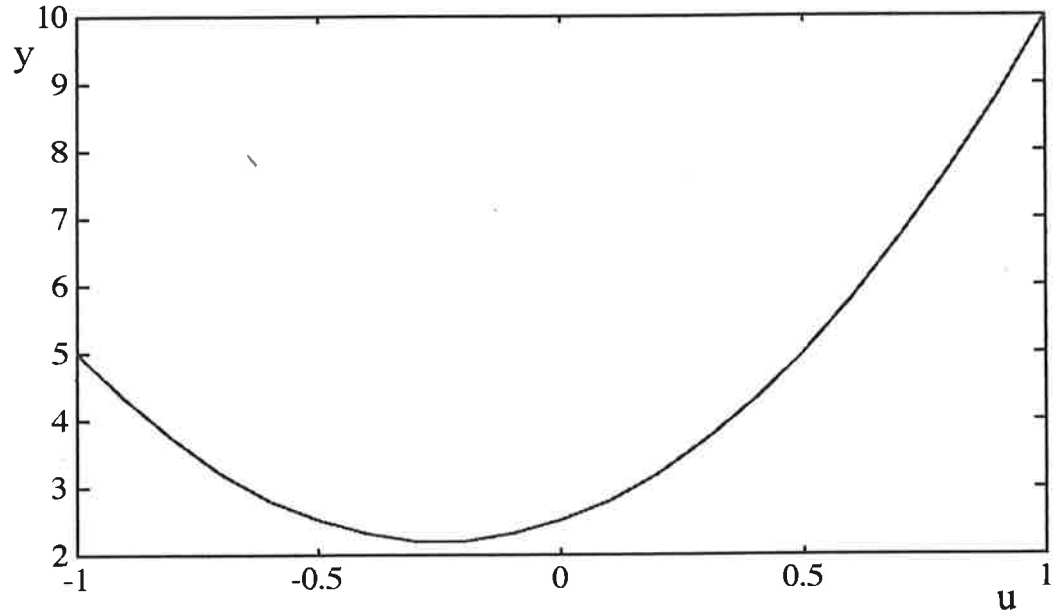


Fig 2. Simulering av process

Figuren visar utsignalen y's stationära värde då u varit konstant.
I simuleringen har vi ansett att stationärt värde nåts efter 100 sample,
Vid simuleringen har C-polynomet satts till noll för att eliminera brusets inverkan. Styrsiganlen har varit en styckvis konstant, 100 sample, rampad signal från -1 till 1 med steg om 0.1.

Numerisk optimering av styrsignalen

Metoden vi använder bygger på att man undersöker derivatan i en punkt och beslutar i vilken riktning extrempunkten ligger.I vårt fall har utsignalen som funktion av insignalen en extrempunkt, ett minimum, fig 3 visar ett schematiskt utseende.

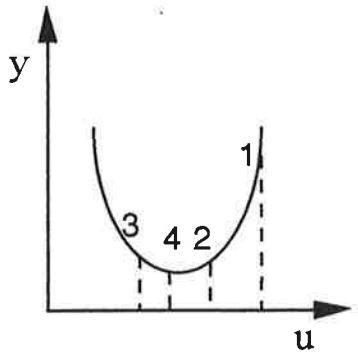


Fig 3. Schematisk bild av utsignal i förhållande till insignal

Metoden arbetar efter följande algoritm:

- avläs utsignalen för given insignal - 2
- jämför detta värde med föregående avläsning - 1
- om det nya värdet är mindre än det gamla ta ett nytt steg med bibehållen steglängd - 3.
- om inte ta ett steg i andra riktningen med halverad steglängd - 4.
- fortsätt tills steglängden < ε

I vårt fall måste utsignalen medelvärdesbildas eftersom den innehåller brus. Vi medelvärdesbildar över 20 sampel.
Då processen har ändlig snabbhet i fråga om stegvar, se fig 1, används inte de första 20 samplen eftersom utsignalen inte nått sitt stationära värde.
Fig 4 visar en simuleing i SIMNON av processen.

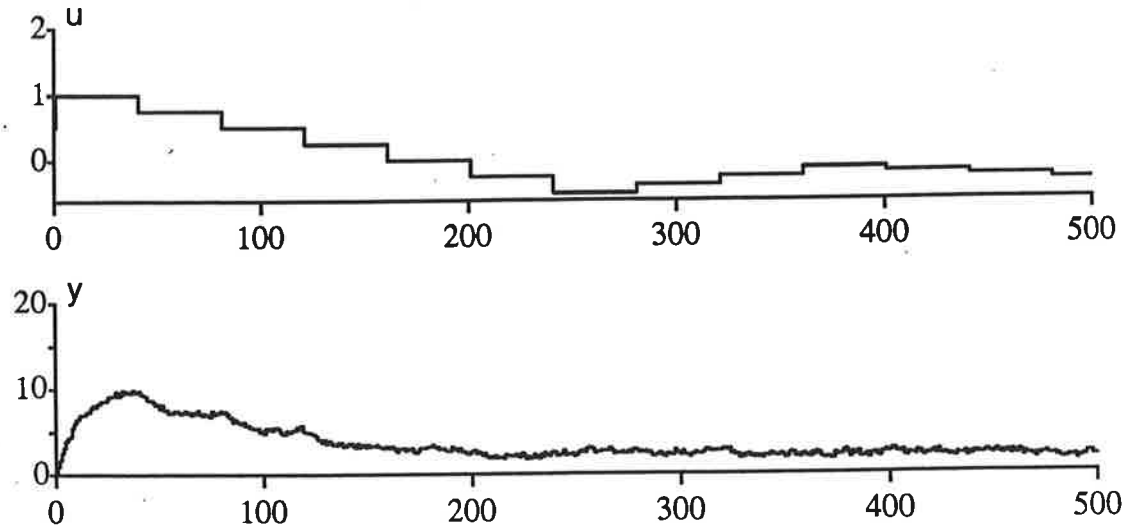


Fig 4. Numerisk optimering av styrsignal

Denna metod har sin svaghet i processvariationer.
Optimeringen görs en gång och man reglerar efter detta värde.
Fördelen är att man inte behöver någon kunskap om processen.

En möjlig utveckling av algoritmen är att inte hela tiden förkasta de 20 första värdena för att uppnå stationaritet då steglängden minskar.
Stegändringarna kommer att bli mindre och mindre, därmed kommer ochså variationerna i utsignalen att bli mindre.
Optimeringen kommer att gå fortare.

Adaptiv optimering

Optimeringen av styrsignalen bygger på estimerade processparametrar.
Styrsignalen beräknas analytiskt. Detta görs vid varje sampel.

Vid den analytiska beräkningen av minimal utsignal används

$$u(t) = -(b_{10} + b_{11})/(2*b_{20}) \qquad (3)$$

Parametrarna b_{10}, b_{11} och b_{20} kommer ifrån estimatorn.

I simuleringarna använder vi en 'Extended Least Square'-estimator där sex parametrar skattas.
Modellen är av samma form som processen, dvs vi känner strukturen på processen.

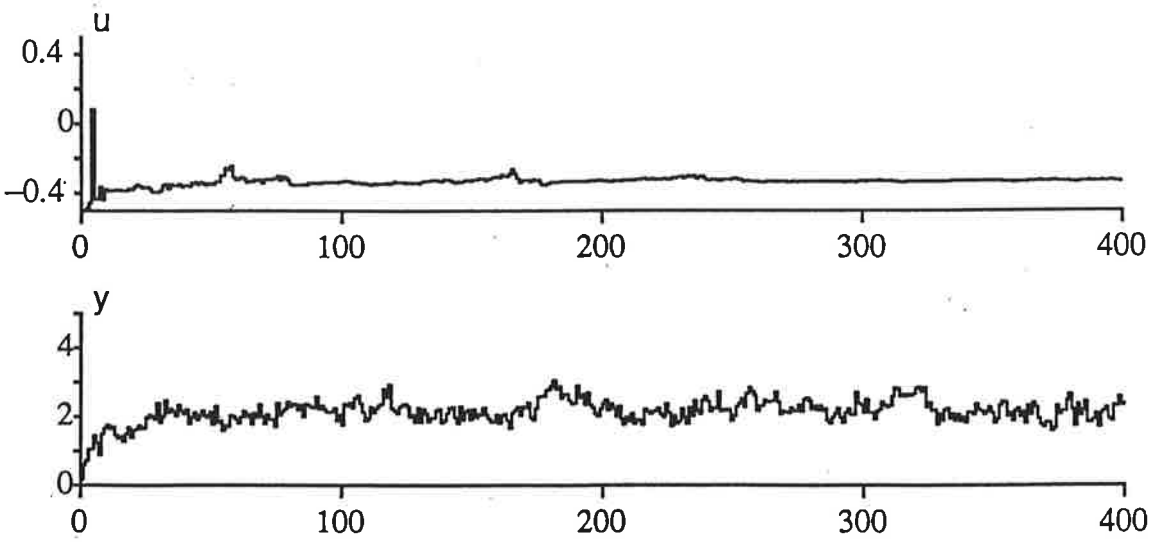


Fig 5. Simuleing av adaptiv optimering

Metodens stora fördelar ligger i okänsligheten för processvariationer. Nackdelarna däremot är att man måste ha ganska stor kunskap om processen då man skapar estimatorn.
Problem uppstår då b_{20} närmar sig noll, då kommer styrsignalen att gå mot oändligheten.

RESULTAT, SAMMANFATTNING

Oavsett vilken metod man använder kommer man ungefär fram till att en styrsignal u=-0.29 i det numeriska fallet och u=-0.33 i det adaptiva fallet, ger en minimal utsignal y, teoretiska värdet är då u=-0.25.

Plottar man $\Sigma(y-y_0)^2$, förlustfunktionen, kan man lättare jämföra motoderna sinsemellan, se fig 6.
man kan konstatera att den adaptiva metoden konvergerar snabbare och förlustfunktionen liggen inom rimliga värden.
I det numerioska fallet kommer felet $y-y_0$ att vara stort och därför kommer summan att bli väldigt stor.
I fig 6 har vi börjat summeringen vid 200 sample.
Man kan konstatera att när väl metoderna konvergerat finns där ingen större skillnad av förlustfunktionerna.

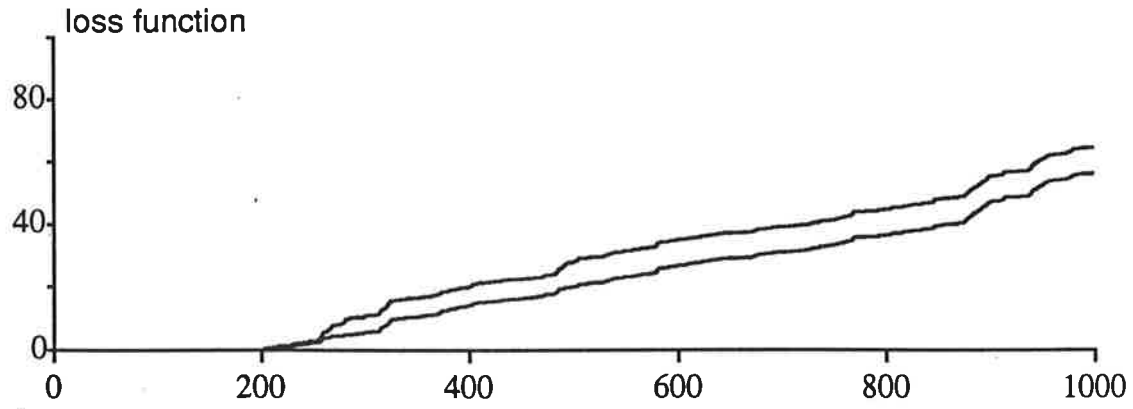


Fig 6. Plottning av förlustfunktion; övre - numerika metod, undre adaptiv metod.

Båda metoderna har sina för och nackdelar som allt annat men vi förordar den adaptiva lösningen pga av dess okänslighet för processvariationer samt att den är betydligt snabbare än den numeriska.
Visserligen finns den andra numerisk optimeringsmetoder som har en högre konvergens hastighet men den adaptiva lösningen är at föredra. Metoden kräver dock en del arbete med identifiering av processen, men detta kommer att löna sig i längden.

REFERENSER

[1] Åström, Wittenmark, Adaptive Control, Addison Wesley, 1990

[2] Wittenmark, Adaptive control of a stochastic nonlinear system: An Example, presented at Workshop on Adaptive Control, Cancun Mexico Dec 1992.

```
macro numsol

syst process num con1

turn seed

init count:0
,step:0.5
,uold[num]:2
,mean[num]:0
,direc[num]:1


store y[process] u[process] step count mean direc

split 3 1

axes h 0 600 v -0.6 5
axes v 0 50
axes v 0 50


area 1 1
text 'u'
plot u[process]
simu 0 600
area 2 1
text 'y'
show y[process]
area 3 1
text 'E[y]'
show mean

end
```



```
connecting system con1

"connecting system

u[process]=u[num]
y[num]=y[process]

end
```

```
discrete system num

"numerisk algoritm

state count mean oldmean  direc step uold
new ncount nmean noldmean ndirec nstep nuold

input y
output u
time t
tsamp ts


u=if (count>0 and count<2) then uold+step*direc else uold

nuold= u
nmean=if count<21  then 0 else mean+y/20
noldmean= if count>39 then mean else oldmean
ndirec= if count>39 then (if mean<oldmean then direc else -direc) else direc
nstep= if count>39 then (if mean<oldmean then step else step/2) else step
ncount= if count<40 then count+1 else 1

ts=t + h

h:1

end
```



```
discrete system EX95

"First order Hammerstein model

state yold uold uoold eold
new nyold nuold nuoold neold
input u
output y
time t
tsamp ts

e=sig*norm(t)
y=-a1*yold+b0+b10*uold+b11*uoold+b20*uold*uold+e+c1*eold

nyold=y
nuold=u
nuoold=uold
neold=e

ts=t+h

a1:-0.9
b0:0.25
b10:0.5
b11:-0.25
b20:0.5
c1:-0.3
sig:0.2
h:1

end
```

```
macro FIG4

"True parameter values and adaptive controllers
"Reference value y0=3
"Illustration of self-tuning property

syst dexc ex95 els6 cadapt
turn seed
par lam:0.999
let p0=100
init p11:p0
,p22:p0
,p33:p0
,p44:p0
,p55:p0
,p66:p0
init th1[els6]:1
,th2[els6]:1
,th3[els6]:1
,th4[els6]:1
,th5[els6]:1
,th6[els6]:1
par y0:3
,umin:-2
,umax:2
,eps:0
,tloss:10
,c1:-0.5
split 3 1
axes h 0 400 v  -0.5 0.5
axes v 0 5
axes v -2 1
store th1 th2 th3 th4 th5 th6 y[ex95] u[ex95] y0
area 1 1
text 'u'
plot u[ex95]
simu 0 400
area 2 1
text 'y'
show y[ex95]
area 3 1
text 'estimates'
show th1 th2 th3 th4 th5 th6
end
```



```
discrete system DEXC

"Discrete time extremum controller
"Hammerstein model
" A*y(t)=B0+B1*u(t-1)+B2*u(t-1)^2+C*e(t)


input y a b0 b10 b11 b20 c
output u
time t
tsamp ts

tu = -(b10+b11)/(2*b20)
u = if tu>umax then umax else if tu<umin then umin else tu
ts=t+h

h:1
y0:3
eps:0.05
sw:1
swp:1
swm:1
umin:-5
umax:5
tloss:100

end
```

```
discrete system EX95

"First order Hammerstein model

state yold uold uoold eold
new nyold nuold nuoold neold
input u
output y
time t
tsamp ts

e=sig*norm(t)
y=-a1*yold+b0+b10*uold+b11*uoold+b20*uold*uold+e+c1*eold

nyold=y
nuold=u
nuoold=uold
neold=e

ts=t+h

a1:-0.9
b0:0.25
b10:0.5
b11:-0.25
b20:0.5
c1:-0.3
sig:0.2
h:1

end
```



```
discrete system els6

"Makes extended least squares estimation of
"six parameters

state th1 th2 th3 th4 th5 th6
state p11 p12 p13 p14 p15 p16 p22 p23 p24 p25 p26
state p33 p34 p35 p36 p44 p45 p46 p55 p56 p66
state uold uold2 yold eold
new nth1 nth2 nth3 nth4 nth5 nth6
new np11 np12 np13 np14 np15 np16 np22 np23 np24 np25 np26
new np33 np34 np35 np36 np44 np45 np46 np55 np56 np66
new nuold nuold2 nyold neold
input y u
time t
tsamp ts

"Model
fi1=-yold
fi2=1
fi3=uold
fi4=uold2
fi5=uold*uold
fi6=eold

"Residual
eps=y-(th1*fi1+th2*fi2+th3*fi3+th4*fi4+th5*fi5+th6*fi6)

"P*fi
Pfi1=p11*fi1+p12*fi2+p13*fi3+p14*fi4+p15*fi5+p16*fi6
Pfi2=p12*fi1+p22*fi2+p23*fi3+p24*fi4+p25*fi5+p26*fi6
Pfi3=p13*fi1+p23*fi2+p33*fi3+p34*fi4+p35*fi5+p36*fi6
Pfi4=p14*fi1+p24*fi2+p34*fi3+p44*fi4+p45*fi5+p46*fi6
Pfi5=p15*fi1+p25*fi2+p35*fi3+p45*fi4+p55*fi5+p56*fi6
Pfi6=p16*fi1+p26*fi2+p36*fi3+p46*fi4+p56*fi5+p66*fi6

"Denominator
den=lam+fi1*Pfi1+fi2*Pfi2+fi3*Pfi3+fi4*Pfi4+fi5*Pfi5+fi6*Pfi6

"Gain vector
k1=Pfi1/den
k2=Pfi2/den
k3=Pfi3/den
k4=Pfi4/den
k5=Pfi5/den
k6=Pfi6/den

"Covariance update
np11=(p11-Pfi1*Pfi1/den)/lam
np12=(p12-Pfi1*Pfi2/den)/lam
np13=(p13-Pfi1*Pfi3/den)/lam
np14=(p14-Pfi1*Pfi4/den)/lam
np15=(p15-Pfi1*Pfi5/den)/lam
np16=(p16-Pfi1*Pfi6/den)/lam
np22=(p22-Pfi2*Pfi2/den)/lam
np23=(p23-Pfi2*Pfi3/den)/lam
np24=(p24-Pfi2*Pfi4/den)/lam
np25=(p25-Pfi2*Pfi5/den)/lam
np26=(p26-Pfi2*Pfi6/den)/lam
np33=(p33-Pfi3*Pfi3/den)/lam
np34=(p34-Pfi3*Pfi4/den)/lam
np35=(p35-Pfi3*Pfi5/den)/lam
np36=(p36-Pfi3*Pfi6/den)/lam
np44=(p44-Pfi4*Pfi4/den)/lam
```

```
np45=(p45-Pfi4*Pfi5/den)/lam
np46=(p46-Pfi4*Pfi6/den)/lam
np55=(p55-Pfi5*Pfi5/den)/lam
np56=(p56-Pfi5*Pfi6/den)/lam
np66=(p66-Pfi6*Pfi6/den)/lam

"Parameter update
nth1=th1+k1*eps
nth2=th2+k2*eps
nth3=th3+k3*eps
nth4=th4+k4*eps
nth5=th5+k5*eps
nth6=th6+k6*eps

"fi vector update
nuold=u
nuold2=uold
nyold=y
neold=eps
ts=t+h

lam:1
h:1

end
```



```
connecting system CADAPT

"Connection system

u[ex95]=u[dexc]
y[dexc]=y[ex95]
u[els6]=u[ex95]
y[els6]=y[ex95]
a[dexc]=th1[els6]
b0[dexc]=th2[els6]
b10[dexc]=th3[els6]
b11[dexc]=th4[els6]
b20[dexc]=th5[els6]
c[dexc]=th6[els6]

end
```

APPENDIX

SIMNON-program som använts vid simulerinarna.

```
macro numsol

syst process num con1

turn seed

init count:0
,step:0.5
,uold[num]:2
,mean[num]:0
,direc[num]:1


store y[process] u[process] step count mean direc

split 3 1

axes h 0 600 v -0.6 5
axes v 0 50
axes v 0 50


area 1 1
text 'u'
plot u[process]
simu 0 600
area 2 1
text 'y'
show y[process]
area 3 1
text 'E[y]'
show mean

end
```

```
connecting system con1

"connecting system

u[process]=u[num]
y[num]=y[process]

end
```

```
discrete system num

"numerisk algoritm

state count mean oldmean  direc step uold
new ncount nmean noldmean ndirec nstep nuold

input y
output u
time t
tsamp ts


u=if (count>0 and count<2) then uold+step*direc else uold

nuold= u
nmean=if count<21  then 0 else mean+y/20
noldmean= if count>39 then mean else oldmean
ndirec= if count>39 then (if mean<oldmean then direc else -direc) else direc
nstep= if count>39 then (if mean<oldmean then step else step/2) else step
ncount= if count<40 then count+1 else 1

ts=t + h

h:1

end
```

```
discrete system EX95

"First order Hammerstein model

state yold uold uoold eold
new nyold nuold nuoold neold
input u
output y
time t
tsamp ts

e=sig*norm(t)
y=-a1*yold+b0+b10*uold+b11*uoold+b20*uold*uold+e+c1*eold

nyold=y
nuold=u
nuoold=uold
neold=e

ts=t+h

a1:-0.9
b0:0.25
b10:0.5
b11:-0.25
b20:0.5
c1:-0.3
sig:0.2
h:1

end
```

```
macro FIG4

"True parameter values and adaptive controllers
"Reference value y0=3
"Illustration of self-tuning property

syst dexc ex95 els6 cadapt
turn seed
par lam:0.999
let p0=100
init p11:p0
,p22:p0
,p33:p0
,p44:p0
,p55:p0
,p66:p0
init th1[els6]:1
,th2[els6]:1
,th3[els6]:1
,th4[els6]:1
,th5[els6]:1
,th6[els6]:1
par y0:3
,umin:-2
,umax:2
,eps:0
,tloss:10
,c1:-0.5
split 3 1
axes h 0 400 v  -0.5 0.5
axes v 0 5
axes v -2 1
store th1 th2 th3 th4 th5 th6 y[ex95] u[ex95] y0
area 1 1
text 'u'
plot u[ex95]
simu 0 400
area 2 1
text 'y'
show y[ex95]
area 3 1
text 'estimates'
show th1 th2 th3 th4 th5 th6
end
```

```
discrete system DEXC

"Discrete time extremum controller
"Hammerstein model
" A*y(t)=B0+B1*u(t-1)+B2*u(t-1)^2+C*e(t)


input y a b0 b10 b11 b20 c
output u
time t
tsamp ts

tu = -(b10+b11)/(2*b20)
u = if tu>umax then umax else if tu<umin then umin else tu
ts=t+h

h:1
y0:3
eps:0.05
sw:1
swp:1
swm:1
umin:-5
umax:5
tloss:100

end
```

```
discrete system EX95

"First order Hammerstein model

state yold uold uoold eold
new nyold nuold nuoold neold
input u
output y
time t
tsamp ts

e=sig*norm(t)
y=-a1*yold+b0+b10*uold+b11*uoold+b20*uold*uold+e+c1*eold

nyold=y
nuold=u
nuoold=uold
neold=e

ts=t+h

a1:-0.9
b0:0.25
b10:0.5
b11:-0.25
b20:0.5
c1:-0.3
sig:0.2
h:1

end
```

```
discrete system els6

"Makes extended least squares estimation of
"six parameters

state th1 th2 th3 th4 th5 th6
state p11 p12 p13 p14 p15 p16 p22 p23 p24 p25 p26
state p33 p34 p35 p36 p44 p45 p46 p55 p56 p66
state uold uold2 yold eold
new nth1 nth2 nth3 nth4 nth5 nth6
new np11 np12 np13 np14 np15 np16 np22 np23 np24 np25 np26
new np33 np34 np35 np36 np44 np45 np46 np55 np56 np66
new nuold nuold2 nyold neold
input y u
time t
tsamp ts

"Model
fi1=-yold
fi2=1
fi3=uold
fi4=uold2
fi5=uold*uold
fi6=eold

"Residual
eps=y-(th1*fi1+th2*fi2+th3*fi3+th4*fi4+th5*fi5+th6*fi6)

"P*fi
Pfi1=p11*fi1+p12*fi2+p13*fi3+p14*fi4+p15*fi5+p16*fi6
Pfi2=p12*fi1+p22*fi2+p23*fi3+p24*fi4+p25*fi5+p26*fi6
Pfi3=p13*fi1+p23*fi2+p33*fi3+p34*fi4+p35*fi5+p36*fi6
Pfi4=p14*fi1+p24*fi2+p34*fi3+p44*fi4+p45*fi5+p46*fi6
Pfi5=p15*fi1+p25*fi2+p35*fi3+p45*fi4+p55*fi5+p56*fi6
Pfi6=p16*fi1+p26*fi2+p36*fi3+p46*fi4+p56*fi5+p66*fi6

"Denominator
den=lam+fi1*Pfi1+fi2*Pfi2+fi3*Pfi3+fi4*Pfi4+fi5*Pfi5+fi6*Pfi6

"Gain vector
k1=Pfi1/den
k2=Pfi2/den
k3=Pfi3/den
k4=Pfi4/den
k5=Pfi5/den
k6=Pfi6/den

"Covariance update
np11=(p11-Pfi1*Pfi1/den)/lam
np12=(p12-Pfi1*Pfi2/den)/lam
np13=(p13-Pfi1*Pfi3/den)/lam
np14=(p14-Pfi1*Pfi4/den)/lam
np15=(p15-Pfi1*Pfi5/den)/lam
np16=(p16-Pfi1*Pfi6/den)/lam
np22=(p22-Pfi2*Pfi2/den)/lam
np23=(p23-Pfi2*Pfi3/den)/lam
np24=(p24-Pfi2*Pfi4/den)/lam
np25=(p25-Pfi2*Pfi5/den)/lam
np26=(p26-Pfi2*Pfi6/den)/lam
np33=(p33-Pfi3*Pfi3/den)/lam
np34=(p34-Pfi3*Pfi4/den)/lam
np35=(p35-Pfi3*Pfi5/den)/lam
np36=(p36-Pfi3*Pfi6/den)/lam
np44=(p44-Pfi4*Pfi4/den)/lam
```

```
np45=(p45-Pfi4*Pfi5/den)/lam
np46=(p46-Pfi4*Pfi6/den)/lam
np55=(p55-Pfi5*Pfi5/den)/lam
np56=(p56-Pfi5*Pfi6/den)/lam
np66=(p66-Pfi6*Pfi6/den)/lam

"Parameter update
nth1=th1+k1*eps
nth2=th2+k2*eps
nth3=th3+k3*eps
nth4=th4+k4*eps
nth5=th5+k5*eps
nth6=th6+k6*eps

"fi vector update
nuold=u
nuold2=uold
nyold=y
neold=eps
ts=t+h

lam:1
h:1

end
```

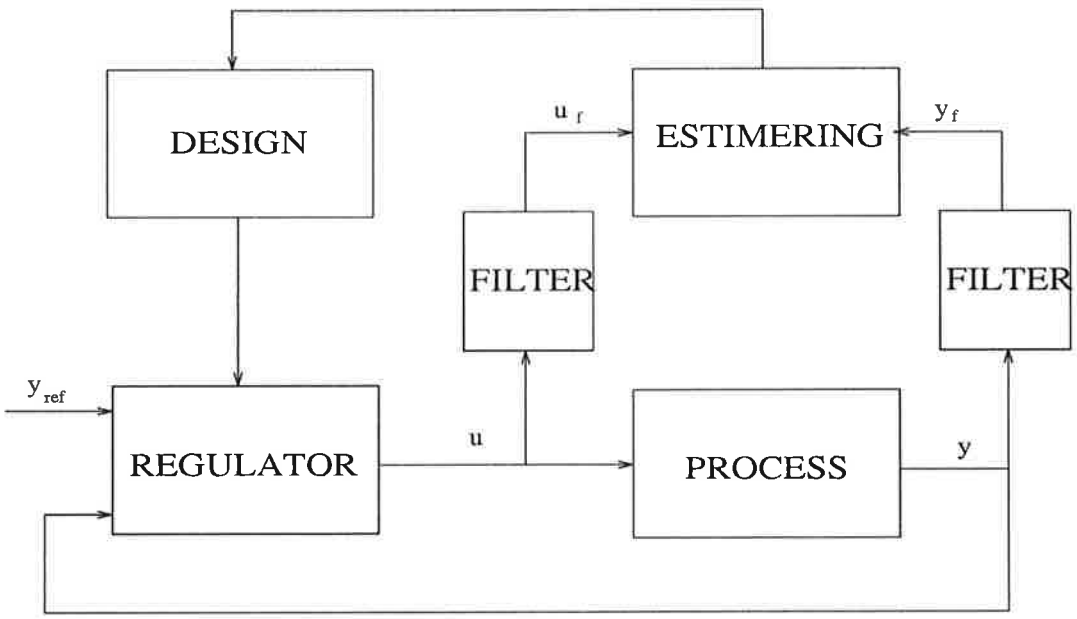
```
connecting system CADAPT

"Connection system

u[ex95]=u[dexc]
y[dexc]=y[ex95]
u[els6]=u[ex95]
y[els6]=y[ex95]
a[dexc]=th1[els6]
b0[dexc]=th2[els6]
b10[dexc]=th3[els6]
b11[dexc]=th4[els6]
b20[dexc]=th5[els6]
c[dexc]=th6[els6]

end
```

Filtrering av regressorerna till estimeringsdelen i en indirekt STR

Roger Granath E89 och Fahed Saouan E89

Handledare: Björn Wittenmark

Inst. för Reglerteknik



Innehåll

1 Sammanfattning

Vår uppgift var att undersöka verkan av att filtrera insignalerna till estimeringsblocket i en indirekt självinställande regulator.
För att kunna skatta processens utseende mäter man processens insignal respektive utsignal. Dessa båda signaler låtes passera två likadana filter innan de når estimeringsblocket.
Simuleringar gjordes för att se hur dessa filter påverkar parameterkonvergensen samt hur stor styrsignal som krävs för att få denna så god som möjligt. Valet av samplingsintervall och glömskefaktorn påverkar i vissa fall uppförandet hos parametrarna och utsignalen från processen. Vi studerade även denna utsignals förmåga att följa referenssignalen i fallen med och utan filter.

2 Indirekt STR utan filtrering

Processen som ska regleras beskrivs av pulsöverföringsfunktionen

$$G(s) = \left[\frac{1}{s^2 + s}\right]$$

Beroende på vilken hastighet man samplar överföringsfunktionen G(s) med kommer det samplade systemets parametrar att anta olika värden. Vi använde oss av följande värden på samplingstiden h:

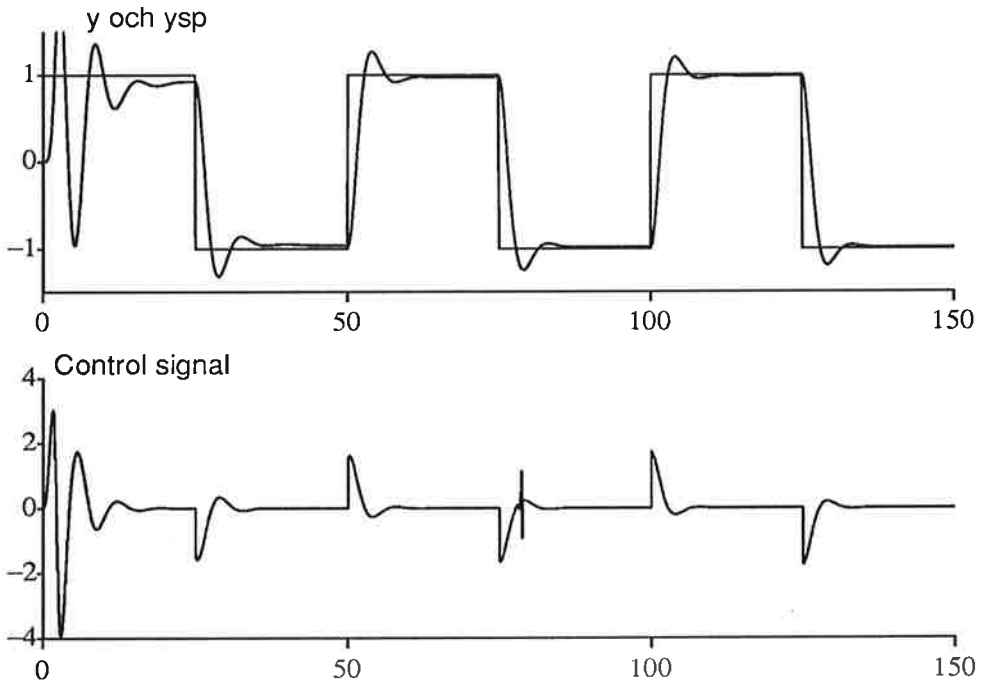
$$H(q) = \left[\frac{b_1 q + b_2}{q^2 + a_1 q + a_2}\right]$$

h	a_1	a_2	b_1	b_2
0.1	-1.9048	0.9048	0.0048	0.0047
0.3	-1.7408	0.7408	0.0408	0.0369
0.5	-1.6065	0.6065	0.1065	0.0902

Estimeringsdelen använder sig av en least-squares estimator with exponential forgetting för att skatta processparametrarna medan regulatorn använder en polplaceringsdesign utan förkortning av nollställen.
Vi började att undersöka parameterkonvergensen och signalutseende då inget filter användes. Resultatet blir en mycket långsam parameterkonvergens mot de rätta värdena oberoende av värdet på h och styrsignalens maxvärde. Då h=0.1 och maxvärdet=4 kommer parametrarna att anta följande värden vid olika simuleringstidpunkter.

t	a_1	a_2	b_1	b_2
50	-1.8412	0.8409	0.0071	0.0189
100	-1.8722	0.8720	0.0020	0.0088
200	-1.8827	0.8826	0.0035	0.0070
600	-1.8944	0.8944	0.0045	0.0055

I figur 1 visas hur utsignalen följer referensvärdet samt styrsignalens utseende då h=0.1 och styrsignalen är begränsad till 4. Som man ser kommer utsignalen att göra en översläng och en undersläng innan den svänger in mot referensvärdet. Eftersom parametrarna konvergerar så långsamt kommer man att få ett fel i början innan dessa har svängt in mot korrekt värde. Styrsignalen blir ganska stor hela tiden för att parametrarna ska konvergera eftersom detta kräver excitation av processen. Glömskefaktorn λ kan ändras från 1 ned till cirka 0.95 utan att parameterkonvergensen ändras. Under dessa värden kommer parametrarna att röra sig oroligare.



Figur 1. Utsignal och styrsignal i fallet utan filter.Observera det numeriska felet som uppstår på styrsignalen vid t=80.Detta beror på initialvärdena på parametrarna.

3 Indirekt STR med filtreringsfall 1

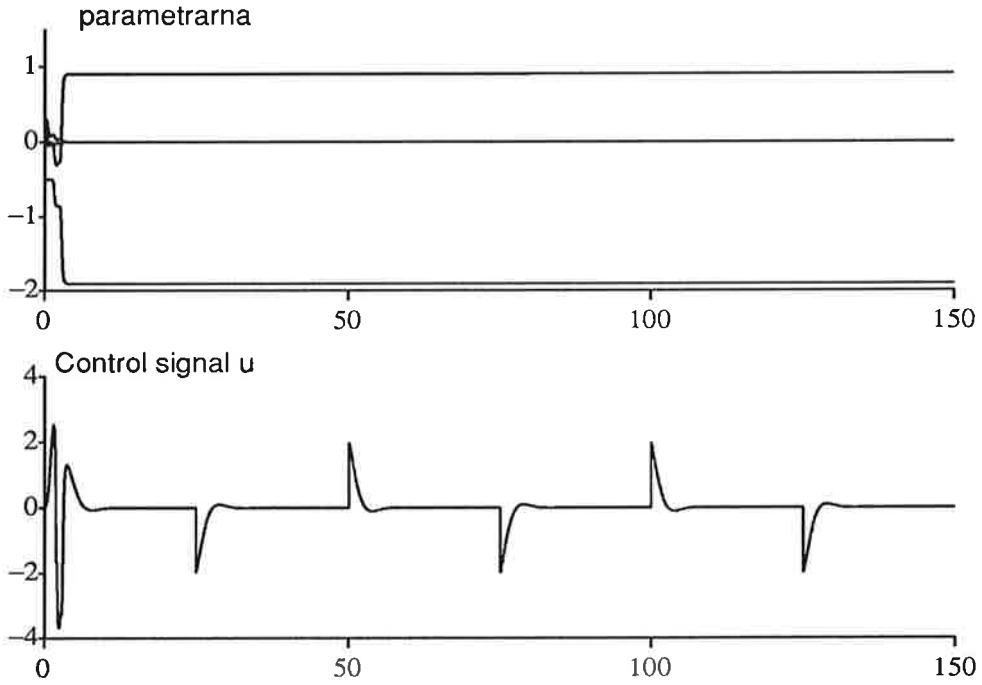
Signalerna in till estimeringsblocket filtreras nu genom ett filter med överföringsfunktionen:

$$H_{filter1}(q) = \left[\frac{1}{A_m(q)A_o(q)}\right]$$

Parametrarna kommer nu att konvergera mycket snabbare än i fallet utan filter. Med h=0.1 och styrsignalen begränsad till 4 kommer parametrarna att anta följande värden.

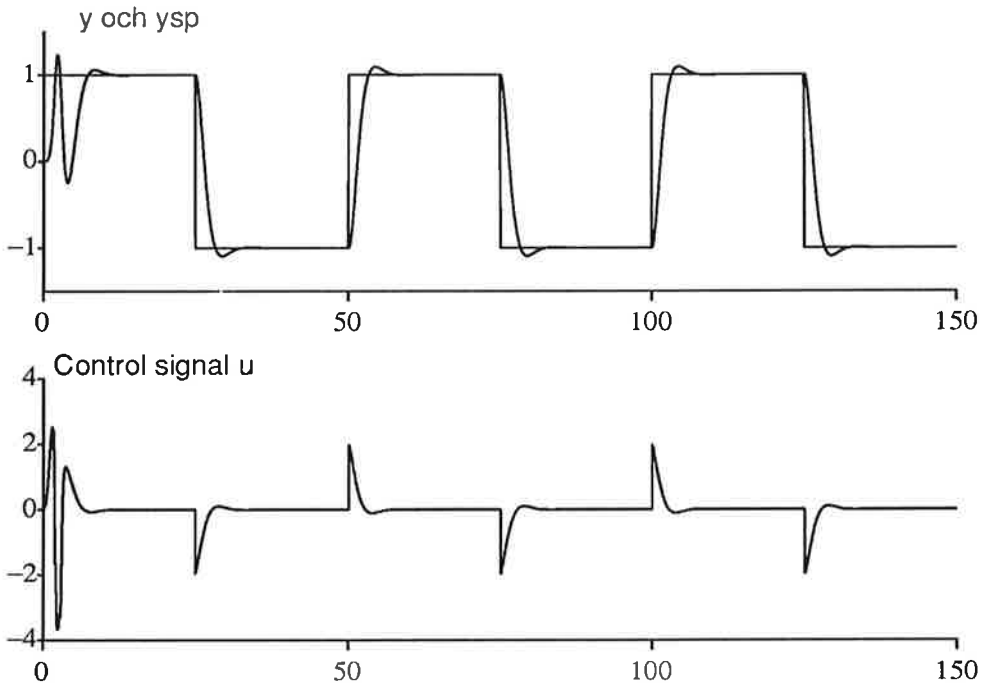
t	a_1	a_2	b_1	b_2
4	-1.9036	0.9036	0.0047	0.0048
5	-1.9048	0.9048	0.0048	0.0047

Redan vid t=5 har parametrarna svängt in mot sina rätta värden vilket visas i figur 2. Att använda filter är ur denna synvinkel en klar fördel.



Figur 2. Parametrar och styrsignal i fallet med filter1.

Utsignalen kommer att följa referenssignalen relativt snabbt efter en liten översläng vilket visas i figur 3 då h=0.1 och styrsignalens maxvärde är 4.



Figur 3. Utsignal och styrsignal i fallet med filter1

Variationer i glömskefaktorn fungerar bra ned till cirka 0.9 utan att regleringen påverkas. Även h och styrsignalens maxvärde kan varieras inom rimliga gränser utan att väsentliga förändringar sker.

4 Indirekt STR med filtreringsfall 2

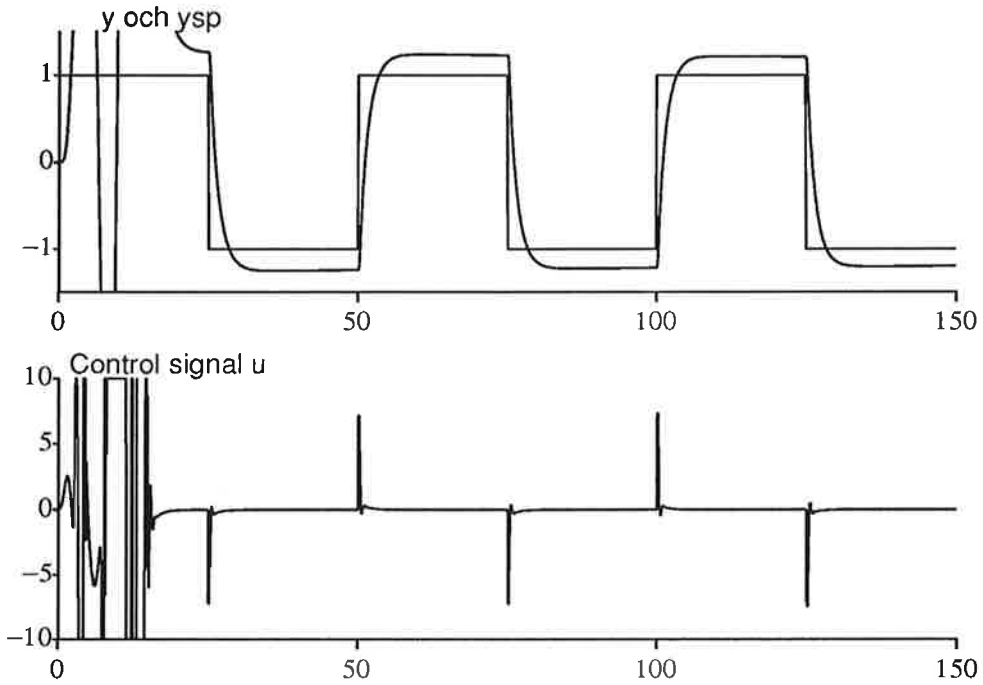
Filtret modifieras nu med ett täljarpolynom så att överföringsfunktionen istället blir

$$H_{filter2}(q) = \left[\frac{R_1(q)}{A_m(q)A_o(q)}\right]$$

Eftersom observerarpolynomet $A_o(q)$ i vårt fall är av första graden kommer polynomet $R_1(q)$ att ha utseendet q+r , dvs vi saknar integratorverkan i vår regulator. Vid simulering erhölls ett stationärt fel mellan processens utsignal och referenssignalen. Detta beror på att transienten i början medför att P-matrisens element blir mycket små. Detta medför i sin tur att förstärkningsvecktorn i estimatorn blir liten och estimeringen avstannar nästan helt. Parametrarna tycks dårför att konvergera mot fel värden. Vid olika simuleringstidpunkter då h=0.1 och styrsignalens maxvärde 10 antar dessa värdena:

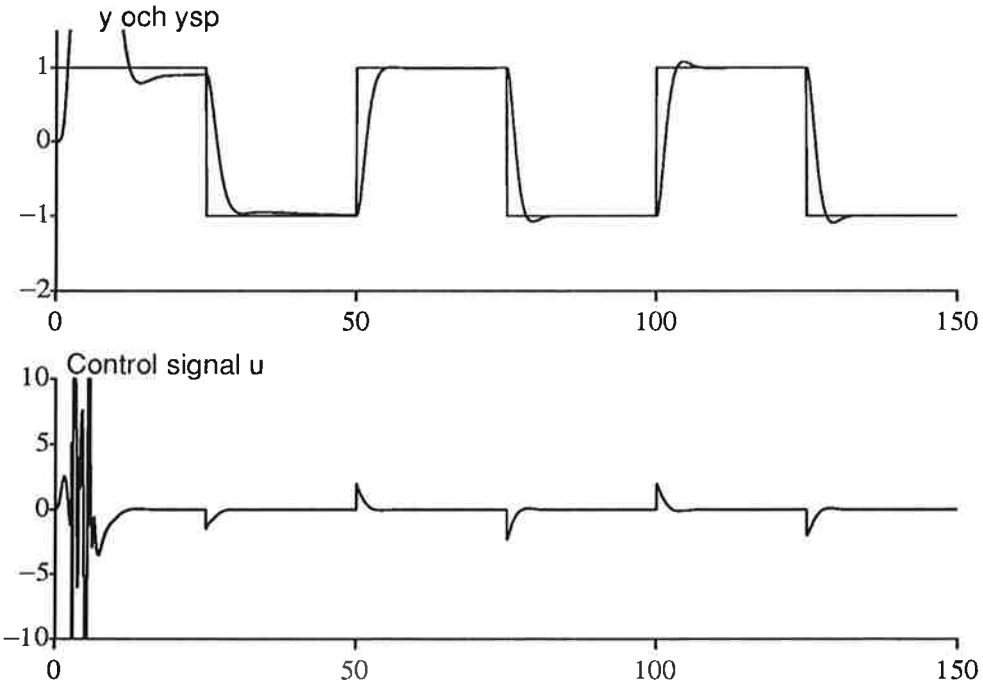
t	a_1	a_2	b_1	b_2
15	-2.0155	1.0199	0.0093	-0.0078
30	-1.9739	0.9763	0.0072	-0.0046
120	-1.9752	0.9773	0.0088	-0.0062

Vid ungefär t=120 har parametrarna konvergerat men mot fel värden. Högre styrsignal ger väldiga parametervariationer medan lägre styrsignal ger långsammare parameterkonvergens. Därmed inte sagt att maxvärdet 10 är ett optimalt val. Utsignalen och styrsignalen ges i figur 4 där man ser att styrsignalen består av ganska höga spikar och en utsignal som hamnar fel.



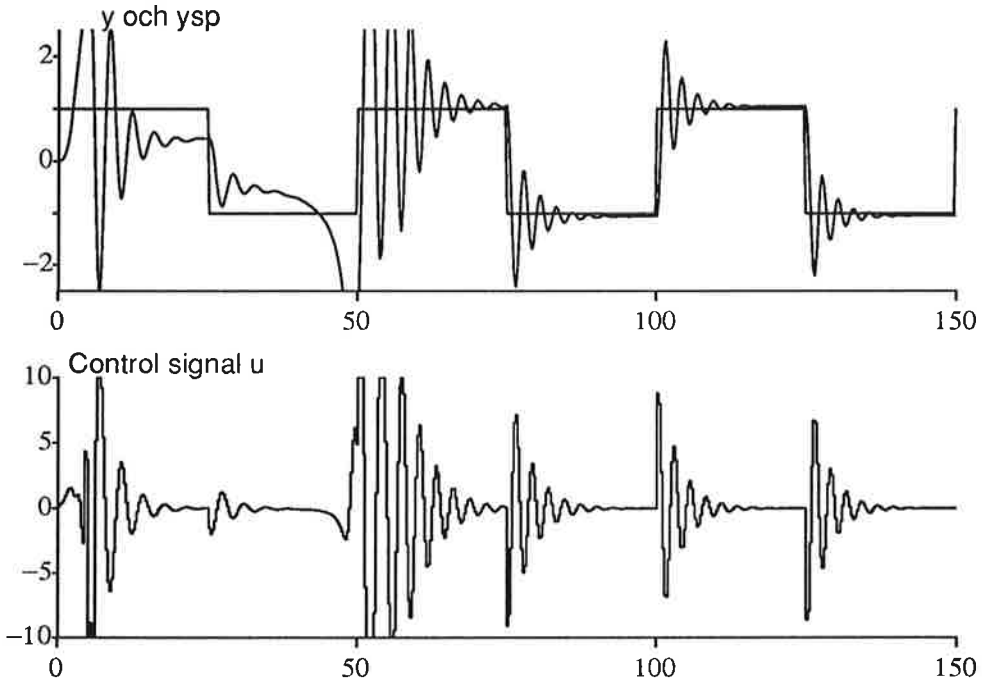
Figur 4. Utsignal och styrsignal i fallet med filter2

Det stationära felet i utsignalen försvinner om man väljer λ=0.99 vilket ges enligt figur 5. För övriga λ-värden erhålles dock inga förbättringar i detta avseende. Styrsignalen kommer att bli mindre än om λ=1 och regleringen fungerar bra. Parametrarna kommer att konvergera lite fortare än tidigare men fortfarande mycket långsammar: om man jämför med det första filtreringsfallet.



Figur 5. Filter2 med λ=0.99

Filtrering med $H_{filter2}(q)$ och h=0.3 resulterar i kraftiga ringningar i styrsignal och utsignal vilket ges i figur 6. Utan filtrering och filtrering med $H_{filter1}(q)$ ger signaler utan ringningseffekt.



Figur 6. Filter2 med h=0.3

5 Kommentarer

Vi kom fram till att indirekt STR med filtrering ger bättre resultat än utan filtrering. Parameterkonvergensen blir snabbare , styrsignalen får lägre amplitud och utsignalen följer referenssignalen snabbare. Vid filtrering med täljarpolynom (filter2) konvergerar dock parametrarna mot fel värde. Detta åtgärdade vi genom att ändra glömskefaktorn λ till 0.99 istället för 1. Det stationära felet mellan utsignalen och referenssignalen försvann , vilket vi konstaterade ur simuleringen.
Regleringen hade kunnat förbättras ytterligare genom att öka regulatorns gradtal.

Inverkan av filtrering vid indirekt STR

AV: Anders Bygren
Mats Welander

För att felet i estimering av regulatorparametrar vid indirekt STR ska vara jämförbart med regulatorfelet bör regressorerna som används i estimatorn filtreras. Man kan teoretiskt visa att ett filter av typen $R_1 / A_o A_m$ ger ett parameterskattningsfel som är identiskt med reglerfelet. (Se häftet till kap.5 "Deterministic Self-tuning regulators" i "Adaptive control")
Vår uppgift var att undersöka, genom simulering, hur olika filter påverkar resultatet i parameterskattningen.

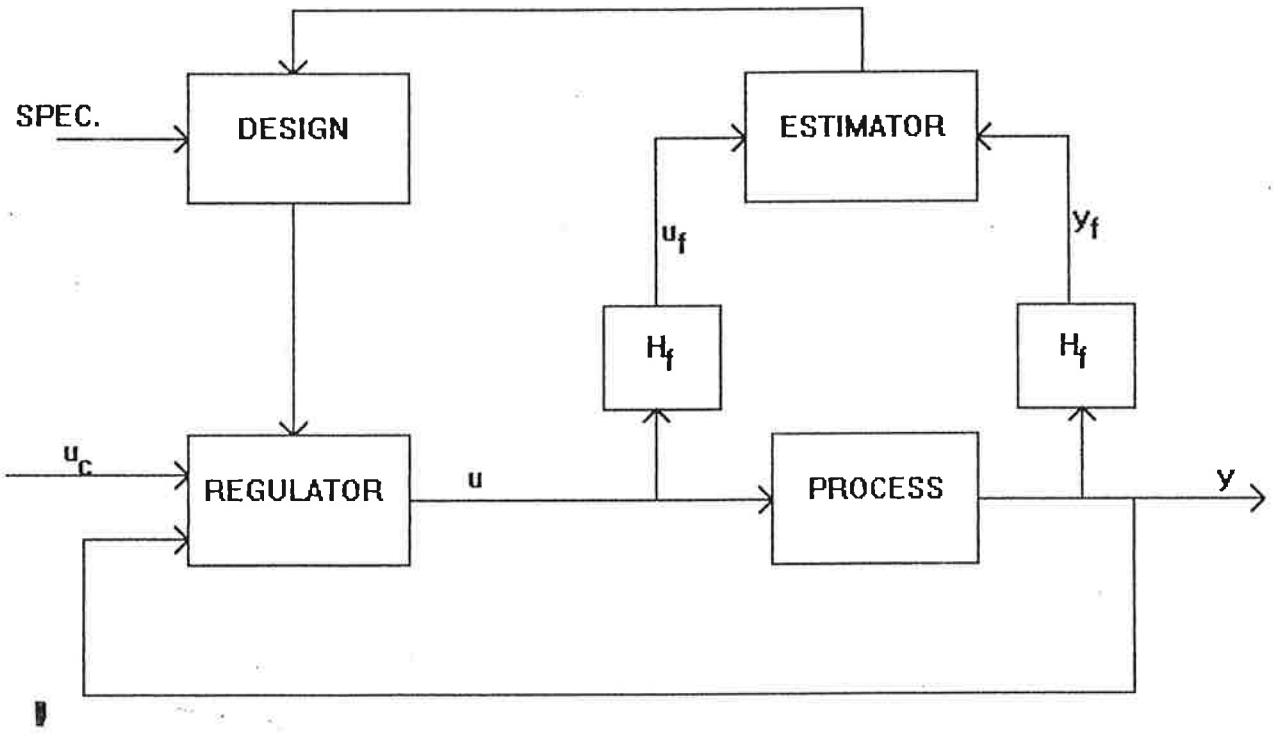


Fig. Blockschema över en ISTR med filtrering av regressorerna.
Regulatorn är en RST-regulator som skall reglera processen 1/s(s+1).
Estimatorn är en LMS-estimator med "exponential forgetting".

Filtret

Vi har baserat vårt filtersystem i simnon på

$$H_f = R_1(q) / A_m(q)A_o(q) = (q + r) / (q^2 + am1*q + am2)*(q + ao)$$

och utifrån det kan vi med små förändringar realisera ett flertal snarlika filter.

```
1  DISCRETE SYSTEM  xfilter
2  "filtrering av signalen x med H = R1/AmAo vid ISTR
3  STATE xf1 xf2 xf3 x1 x2 x3
4  NEW nxf1 nxf2 nxf3 nx1 nx2 nx3
5  INPUT x r
6  TIME t
7  SAMP ts
8  ao= IF cancel > 0.5 THEN 0 ELSE -0 8
9  bjorn =  -( ao+am1 ) * xf1 - ( am2+am1 * ao )*xf2 - ao*am2*xf3+x2+r * x3
10 xf = IF cancel > 0.5 THEN ( - am1 * xf1 - am2 * xf2 + x2) ELSE bjorn
11 nxf1 = xf
12 nxf2 = xf1
13 nxf3 = xf2
14 nx1 = x
15 nx2 = x1
16 nx3 = x2
17 ts =t + h
18 h: 0.5
19 cancel: 1
20 am1: -0.7504942
21 am2: 0.2465969
22 END
```

Som utgångspunkt för simuleringarna använder vi oss av macro fig52 med tillhörande system, ett ISTR-simnonsystem för processen 1/s(s+1). Signalerna som skall filtreras är för vår del x=y och x=u.

SIMULERINGAR OCH RESULTAT

Till varje simulering har genererats en plottning av styrsignal från regulatorn och de estimerade parametrarna. Dessa diagram finns som bilaga till resp. punkt nedan. Simuleringarna är gjorda utan nollställesförkortningar.

1. $H_f = 1 / A_m A_o$. Parameterinsvängningen är mycket snabb och till de rätta parametervärdena. Styrsignalen gör en relativt hög initieringssläng.[1]

2. $H_f = 1 / A_m^* A_o^*$. Parameterinsvängningen är mycket snabb, korrekta slutvärden. Styrsignalen har ett mycket litet startsving.[2]

3. $H_f = R_1 / A_m A_o$. Långsam parameterinsvängning, en faktor 100 längre insvängningstid än i 1. och 2., men korrekta värden till slut. Styrsignalen mycket kraftig och svängig i initieringsfasen. [3]

4. $H_f = R_1^* / A_m^* A_o^*$. I stort sett identiskt uppförande med filtret i 3.[4]

5. Utan H_f. Tiden för insvängning till korrekta parametervärden är längre än för 1 och 2 men mycket snabbare än för 3 och 4.

I dessa fem simuleringar är initialvärdena: th1: -0.5 th2: 0 th3: 0.1 th4: 0.1
λ: 0.99 ulim: 5

6. Här har vi kört en simulering med filtret $H_f = R_1 / A_m A_o$ men med initialvärden mycket avvikande från rätta parametervärden."ulim"är satt till 3 för att begränsa styrsignalen ytterligare.Även här kan vi se att de rätta parametervärdena uppnås ,dock efter en längre tid än i de övriga simuleringarna. (initialvärden: th1:-3 th2:3 th3:1 th4:1 λ: 0.99)

Utöver dessa simuleringar genomförde vi ett antal tester med initialvärden mer eller mindre överensstämmande med de exakta värdena och vi uppnår

---

[1] Ändringar i xfilter : rad 9 : x2 + r * x3 ändras till x3
[2] ------- " ------- : rad 9 : x2 + r * x3 ändras till x
[3] xfilter enligt schema
[4] Ändringar i xfilter : rad 9 : x2 + r * x3 ändras till x + r * x1

parameterinsvängning i de flesta fall. Problem med detta uppstod vid stora avvikelser hos initialvärdena , i förhållande till rätta processparametrar, och då glömskefaktorn =1 och ulim =5.

SAMMANFATTNING

Med filtrering av insignalerna till estimatorn vid ISTR kan vi med våra simuleringar se att processparametrarna uppnår rätta värden. Vi kan även se att den avsevärt snabbaste insvängningen till rätta parametervärden sker med filter utan R_1-polynom i täljaren.Utsignalen i fallet $H_f = 1 / A_m^* A_o^*$ är överensstämmande med den i det ofiltrerade fallet. Med de övriga filtren hade utsignalen en transient i första perioden men därefter följde den styrsignalen väl.

övrigt

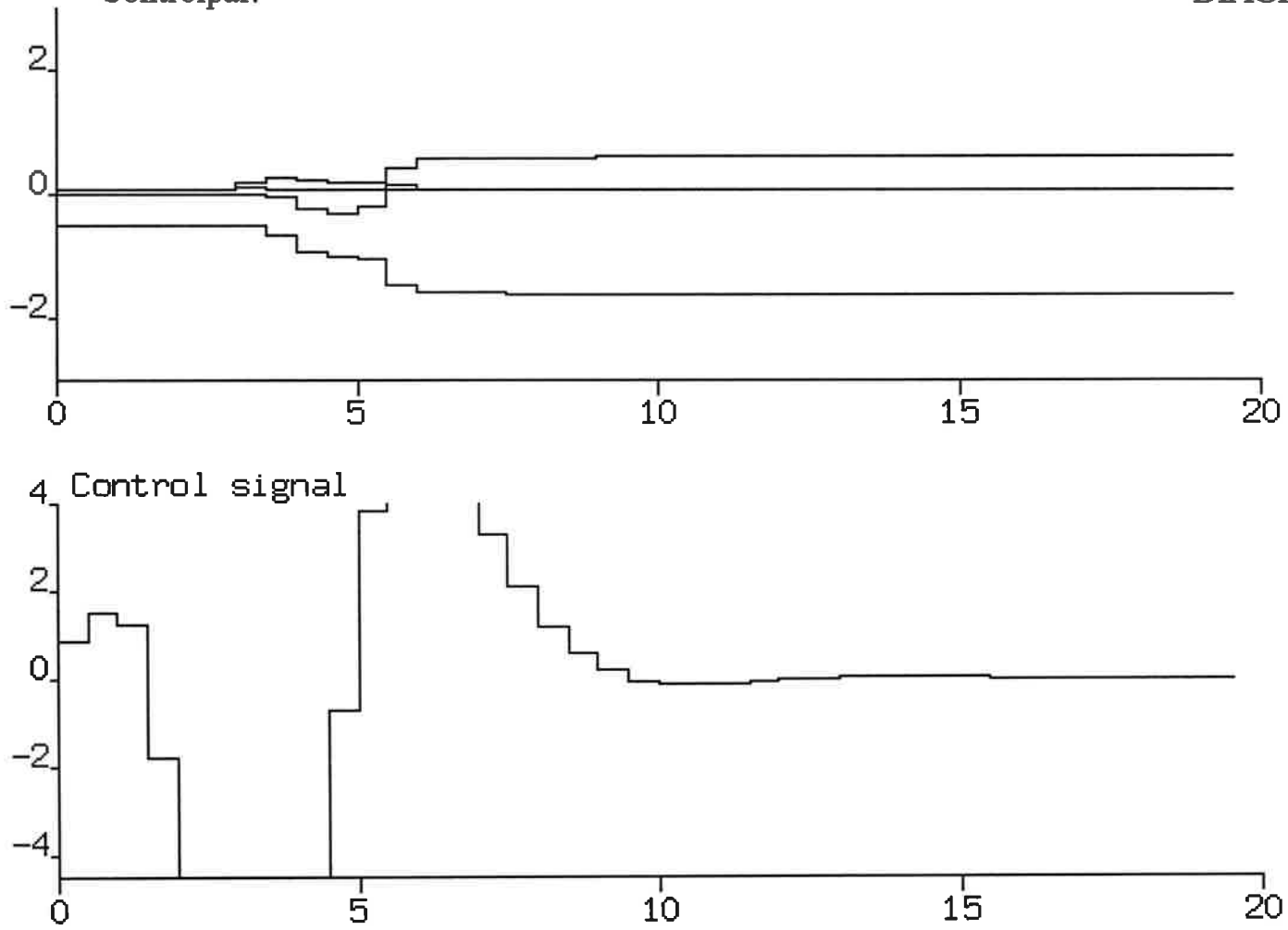
De förändringar vi gjort i redan befintliga system är följande:

Tillägg/Ändringar: Fig52: SYST reg proc ufilter yfilter con51
PAR cancel[reg]:0
PAR cancel[ufilter]:0
PAR cancel[yfilter]:0
"PAR aop:0
Strind0: INPUT ysp y yf uf
eps=yf -f1*th1-f2*th2-f3*th3-f4*th4
nf1=-yf
nf3=uf
Con51 u[ufilter]=u[reg]
y[yfilter]=y[proc]
uf[reg]=uf[ufilter]
yf[reg]=yf[ufilter]
r[ufilter]=r[reg]
r[yfilter]=r[reg]

ISTR utan nollställes-
förkortning
$H_f = 1 / A_o A_m$

Controlpar.

DIAGRAM 1



9

ISTR utan nollställes-
förkortning
$H_f = 1 / A_o * A_m *$

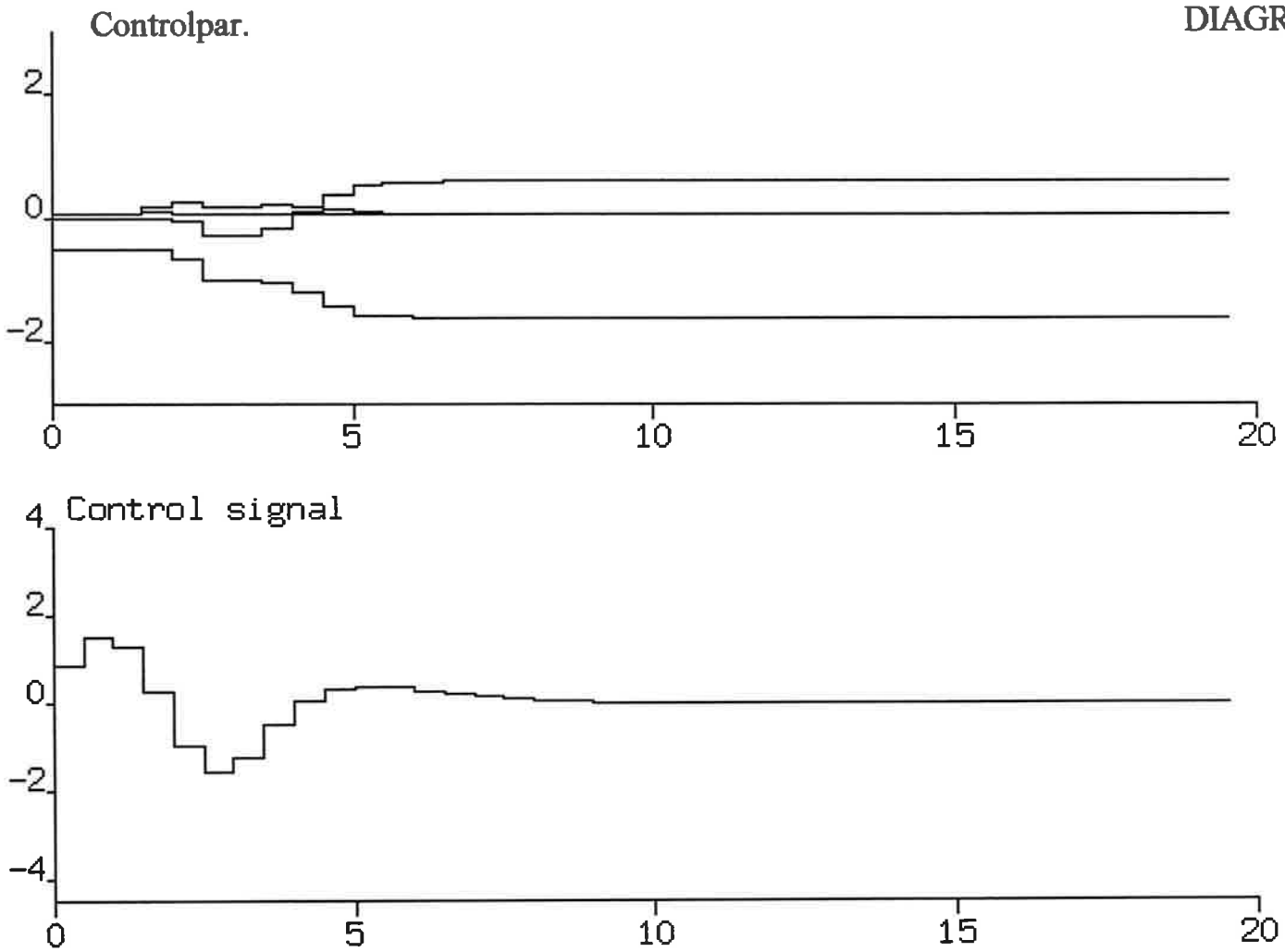
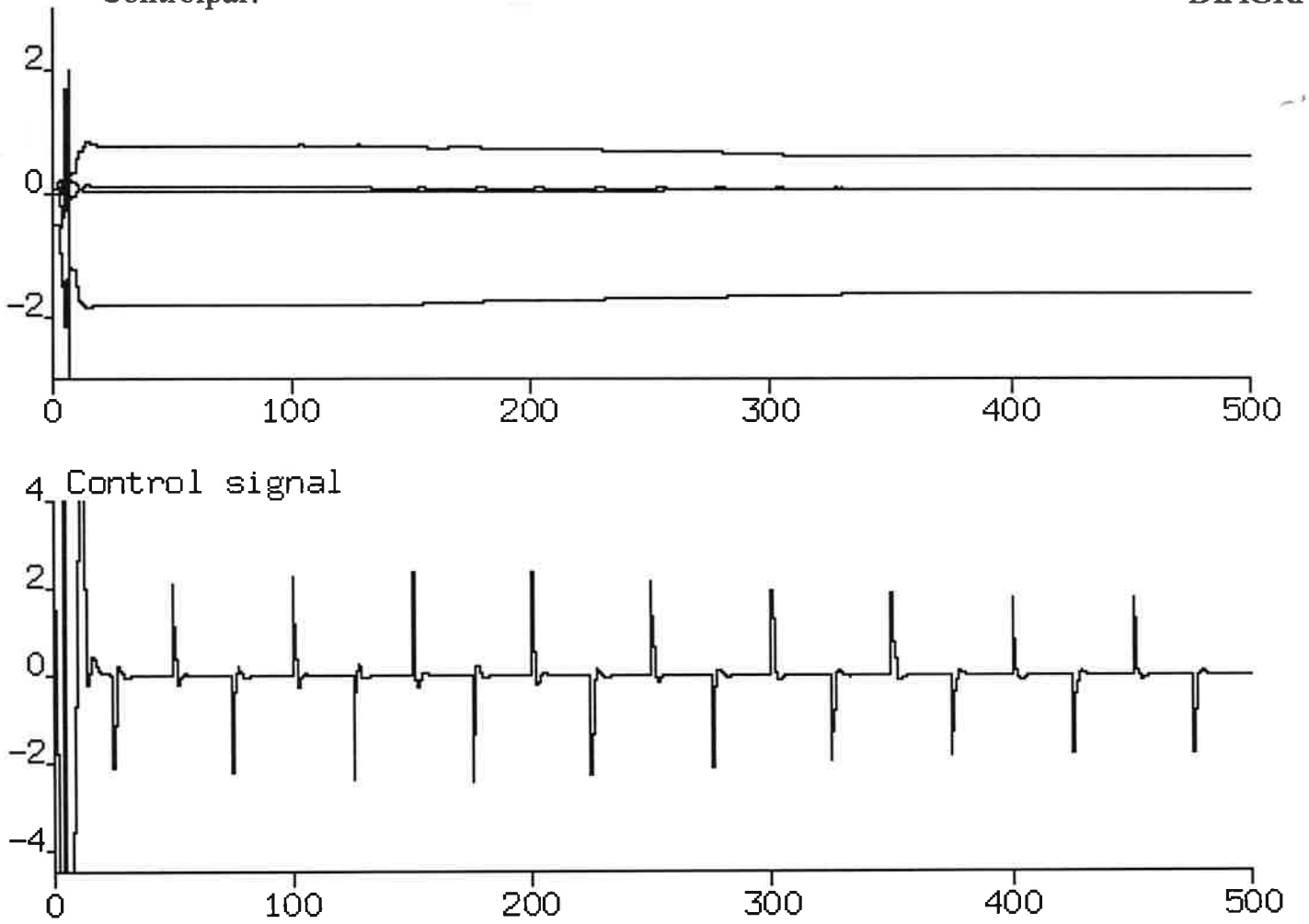


DIAGRAM 2

ISTR utan nollställes-
förkortning
$H_f = R_1 / A_o A_m$

Controlpar.

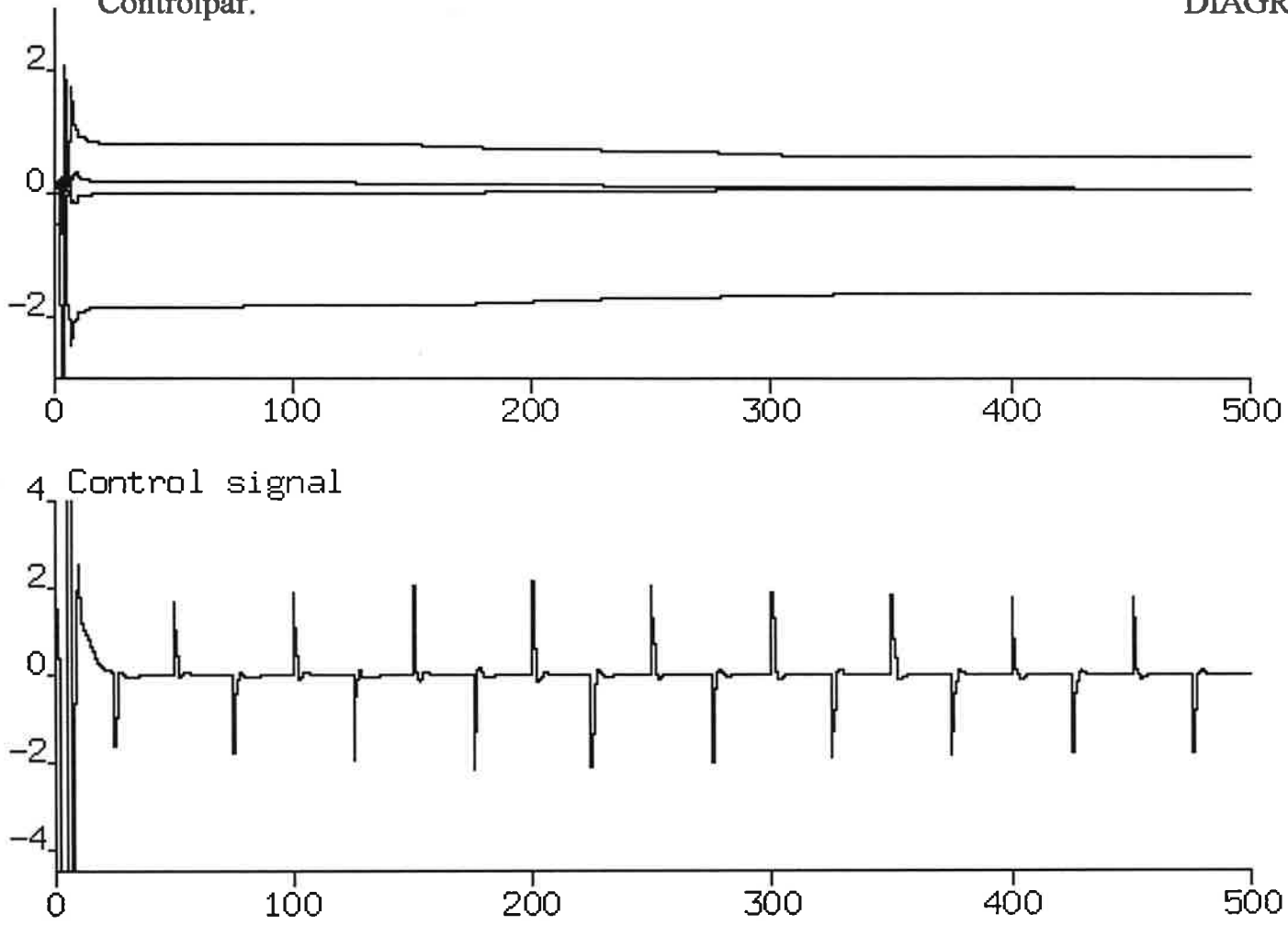
DIAGRAM 3



ISTR utan nollställes-
förkortning
$H_f = R_1^* / A_o^* A_m^*$

Controlpar.

DIAGRAM 4



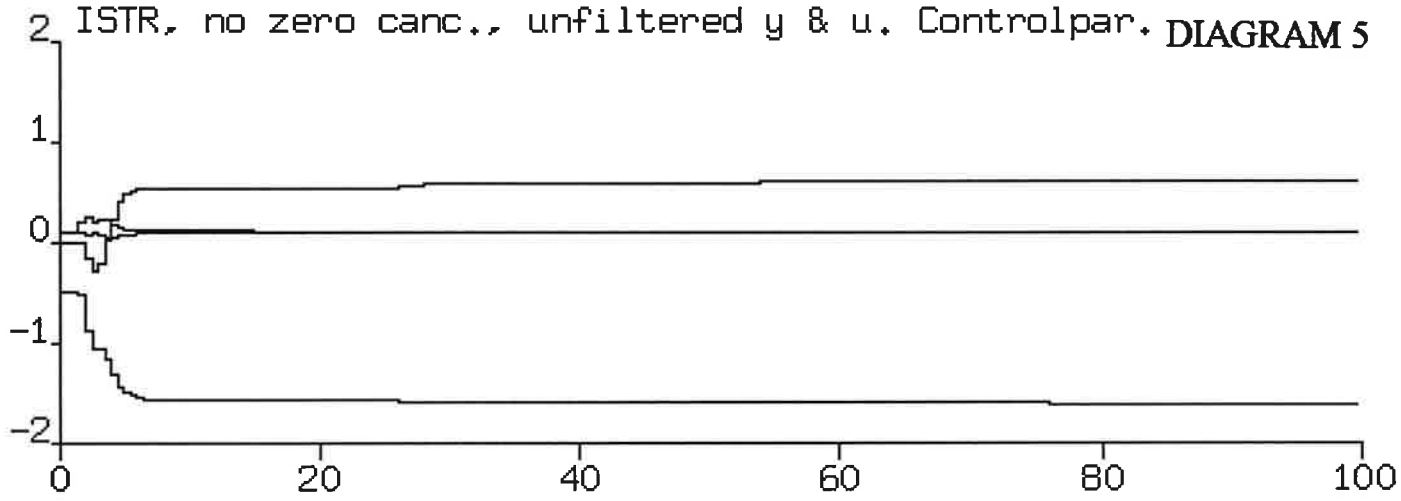
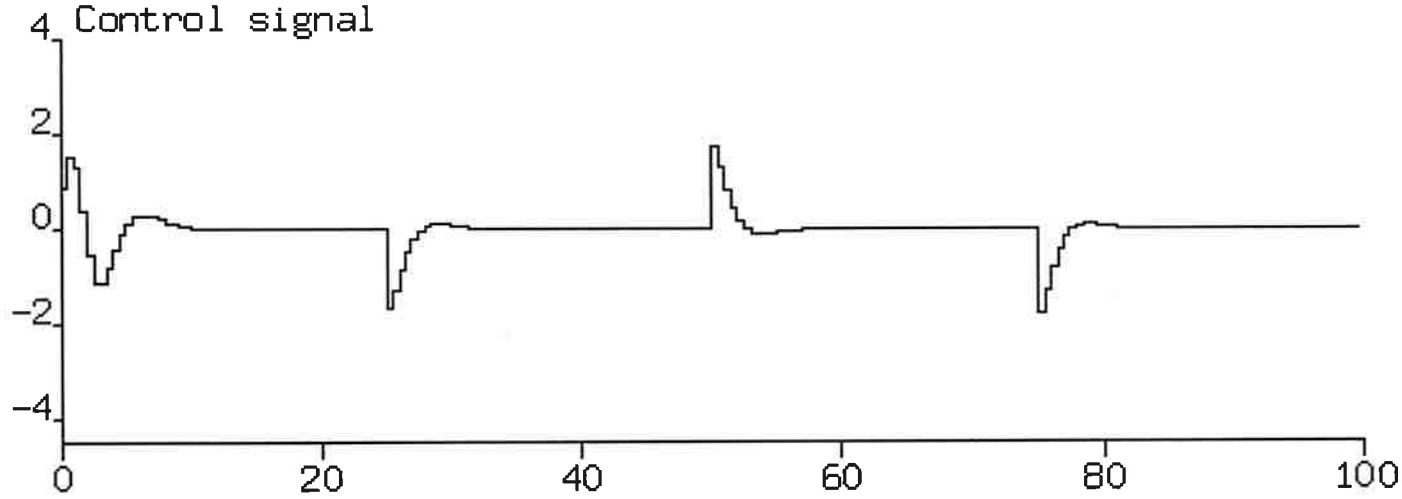

ISTR, no zero canc., unfiltered y & u. Controlpar.
2
1
0
-1
-2
0
20
40
60
80
100

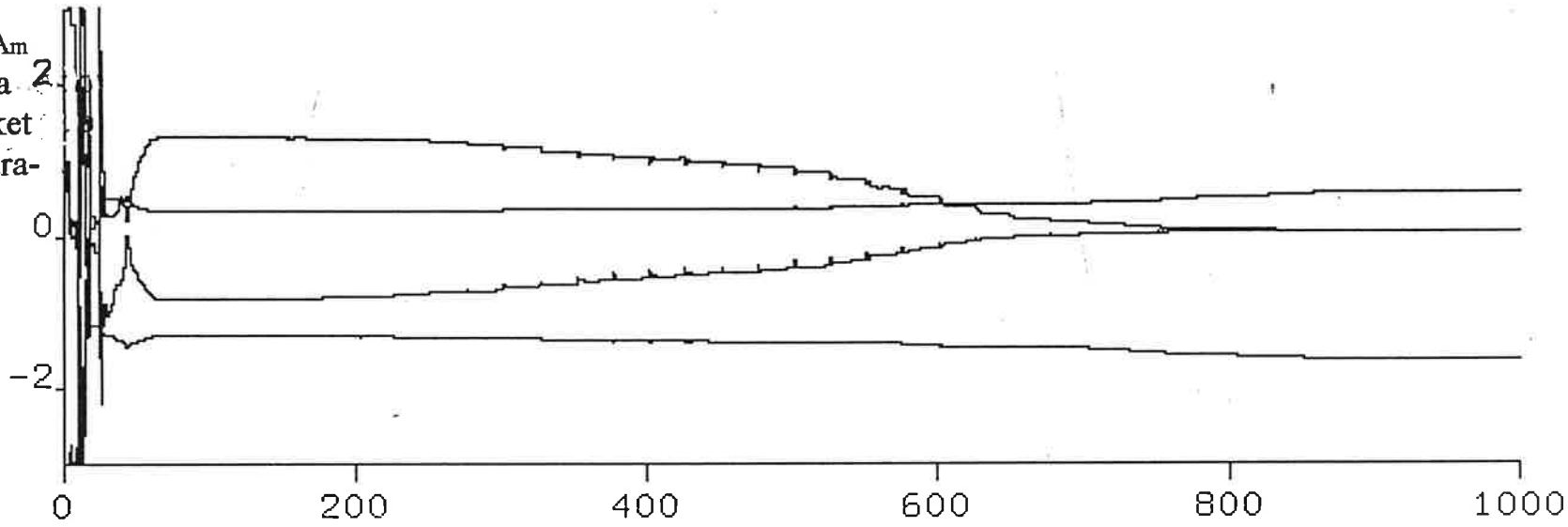
DIAGRAM 5

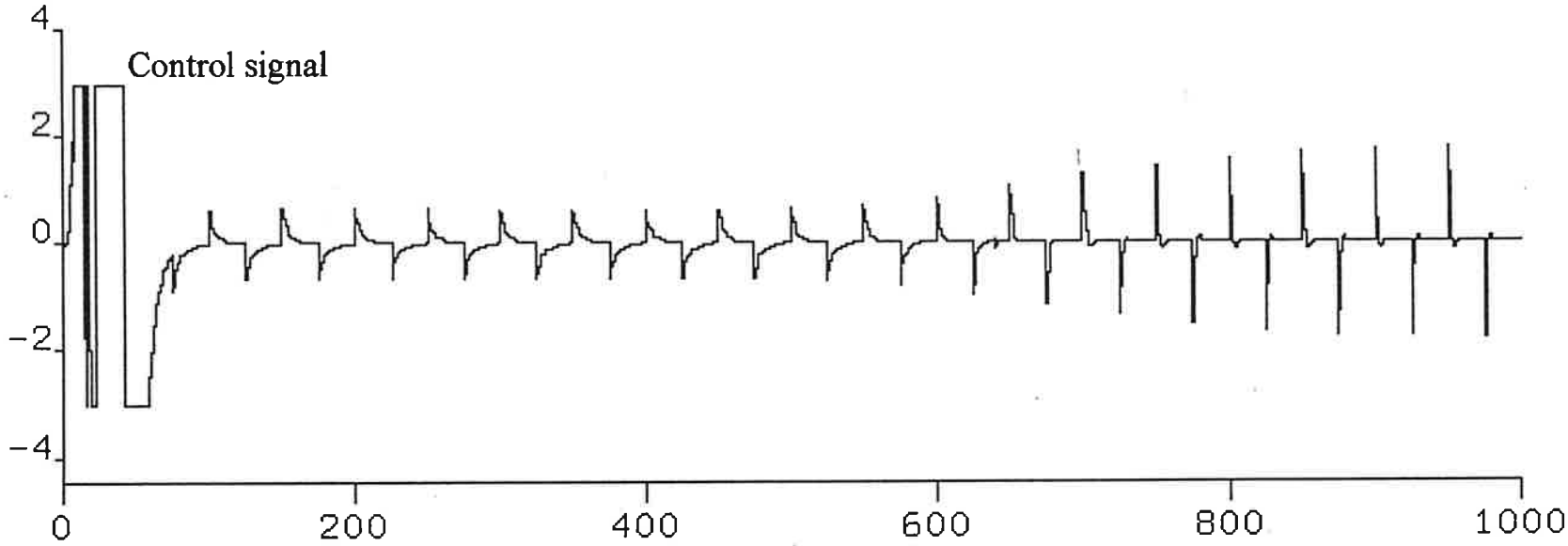

Control signal
4
2
0
-2
-4
0
20
40
60
80
100

ISTR utan nollställes-
förkortning
$H_f = R_1 / A_o A_m$
Initialvärdena
avviker mycket
från rätta para-
metervärden.

Controlpar.

DIAGRAM 6





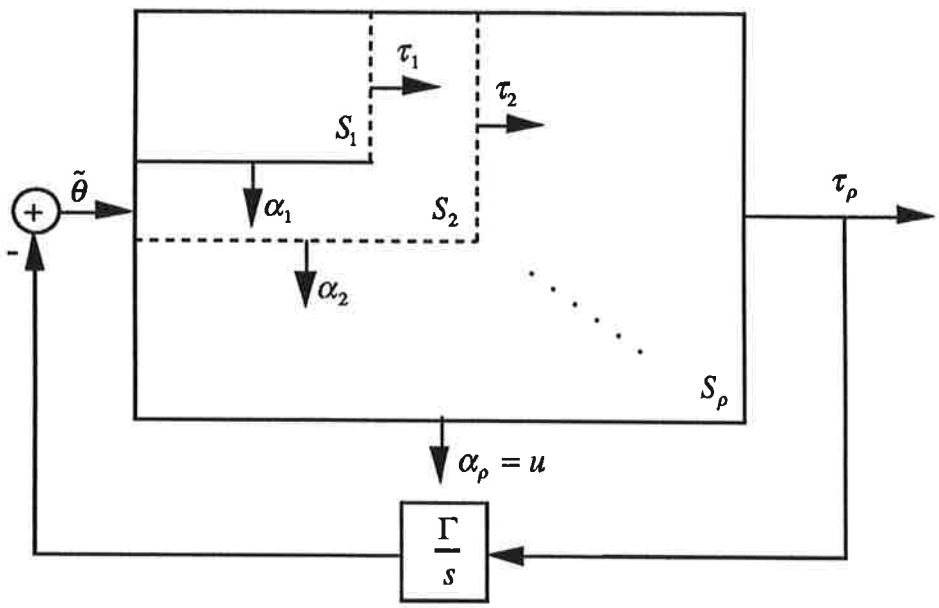
Backstepping
Projekt i Adaptiv reglering

Mikael Johansson E-89
Mats Jonsson E-89
Anna Tengvall E-89

Handledare : Ulf Jönsson

12 maj 1993

Schematisk bild över designproceduren



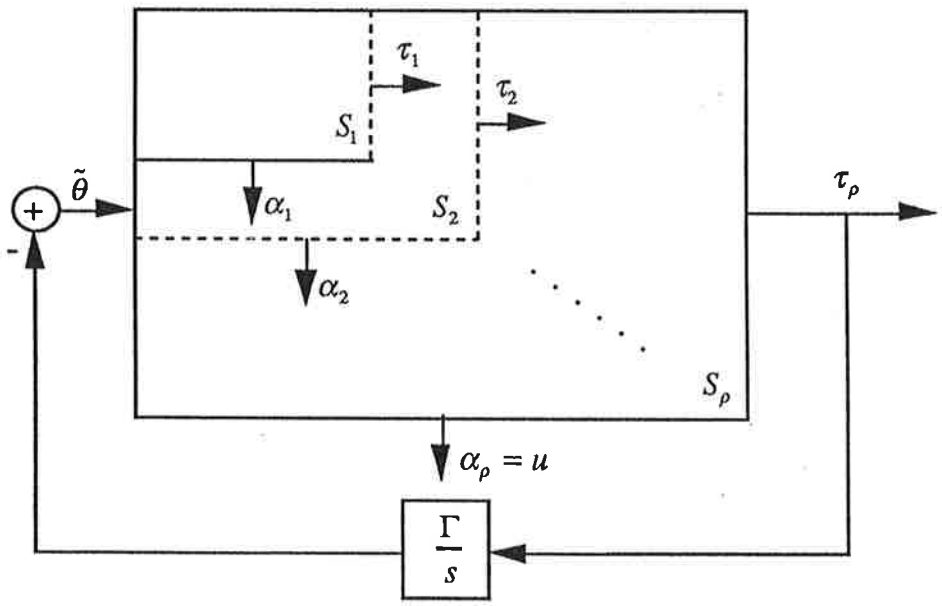
Backstepping
Projekt i Adaptiv reglering

Mikael Johansson E-89
Mats Jonsson E-89
Anna Tengvall E-89

Handledare : Ulf Jönsson

12 maj 1993

Schematisk bild över designproceduren



1 Inledning

Backstepping är en rekursiv designmetod som baserar sig på passivitetsteori. Metoden är relativt ny och har bland annat presenterats i två artiklar skrivna av Krstic, Kanellakopoulos och Kokotovic (1,2).

Backsteppingprincipen kan exemplifieras på ett generellt linjärt system med överföringsfunktionen

$$G(s) = \frac{b_m s^m + \cdots + b_1 s + b_0}{s^n + a_{n-1} s^{n-1} \cdots + a_1 s + a_0}$$

Systemet kan skrivas på tillståndsform med parametriseringen

$$\dot{x}_1 = x_2 - a_{n-1} x_1$$

$$\vdots$$

$$\dot{x}_{\rho-1} = x_\rho - a_{m+1} x_1$$

$$\dot{x}_\rho = x_{\rho+1} - a_m x_1 + b_m u$$

$$\vdots$$

$$\dot{x}_n = -a_0 x_1 + b_0 u$$

$$y = x_1$$

där $\rho = n - m$ betecknar systemets relativa gradtal.

Målet med regulatordesignen är att bestämma en styrlag u sådan att tillståndsfelen konvergerar exponentiellt mot noll.

Vi definierar felet i tillståndet $x_1 = y$ som

$$z_1 \triangleq y - y_r$$

Där y_r betecknar önskat utsignabeteende. Derivation med avseende på tiden ger

$$\dot{z_1} = \dot{y} - \dot{y_r} = \dot{x_1} - \dot{y_r} = x_2 - a_{n-1} x_1 - \dot{y_r}$$

Vi kan tänka oss denna ekvation som ett första ordningens system med x_2 som insignal och tänka oss $x_2 = \alpha_1$ som en "virtuell" styrlag, som stabiliserar felet z_1. Nu kan vi inte påverka x_2 direkt utan endast indirekt genom u. Detta medför att styrlagen α_1 inte kan uppfyllas helt, vilket ger ett nytt fel z_2 i tillstånd x_2.

$$x_2 = \alpha_1 + z_2$$

I nästkommande designsteg bestämms sedan en motsvarande styrlag α_2 för tillståndet x_3.

Det antal designsteg som måste genomföras är lika många som systemets relativa gradtal. I varje steg bestäms en ny virtuell styrlag α_i som stabiliserar felekvationen $\dot{z}_i$. I det sista steget bestäms en verklig styrlag u sådan att alla virtuella styrlagar α_i uppfylls.

I och med att processparametrarna här är okända måste vi använda parameterestimering tillsammans med backstepping. Resultatet blir en adaptiv reglermetod.

2 Härledningar

För att underlätta förståelsen för den relativt tunga teorin följer här ett tillämpat exempel.

2.1 Problemformulering

Vi skall reglera ett linjärt andra ordningens system med tre okända parametrar och relativt gradtal två:

$$G(s) = \frac{b_0}{s^2 + a_1 s + a_0}$$

Systemet kan skrivas på tillståndsform med parametriseringen:

$$\dot{x_1} = x_2 - a_1 x_1$$

$$\dot{x_2} = -a_0 x_1 + b_0 u$$

$$y = x_1$$

Önskat uppförande ges av modellsystemet

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

eller på tillståndsform, med r som referenssignal:

$$\dot{x}_{m,1} = x_{2,m}$$

$$\dot{x}_{m,2} = -\omega^2 x_{m,1} - 2\omega\zeta x_{m,2} + \omega^2 r$$

$$y_r = x_{m,1}$$

Vi har endast utsignalen tillgänglig och måste därför skatta övriga tillstånd. Vi definierar nu följande vektorer:

$$a = \begin{bmatrix} -a_1 \\ -a_0 \end{bmatrix}$$

$$b = \begin{bmatrix} b_0 \end{bmatrix}$$

2.2 Tillståndsskattningar

Kärnan i tillståndsskattaren är filterna:

$$\dot{\zeta_2} = A_0\zeta_2 + ky$$

$$\dot{\zeta_1} = A_0\zeta_1 + e_1 y$$

$$\dot{\zeta_0} = A_0\zeta_0 + e_2 y$$

$$\dot{v_0} = A_0 v_0 + e_2 u$$

A_0 väljs som

$$A_0 = \begin{bmatrix} -k_1 & 1 \\ -k_2 & 0 \end{bmatrix}$$

A_0 har karaktäristiska ekvationen $\lambda(s) = s^2 + k_1 \cdot s + k_2$. Om A_0 är stabil enligt Hurwitz kriterium fås exponentiell konvergens hos tillståndsskattningen

$$\hat{x} = \zeta_2 - a_0\zeta_0 - a_1\zeta_1 + b_0 v_0$$

Signalerna ζ och v kan fås ur filterna

$$\dot{\eta} = A_0\eta + e_2 y$$

$$\dot{\lambda} = A_0\lambda + e_2 u$$

enligt

$$\zeta_0 = I \cdot \eta$$

$$\zeta_1 = A_0\eta$$

$$\zeta_2 = -A_0^2\eta$$

$$v_0 = I \cdot \lambda = \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix}$$

Vi kan nu börja designa en adaptiv regulator enligt backsteppingmetoden. Målet är att bestämma en styrlag u så att tillståndsfelen z_1 och z_2 konvergerar exponentiellt mot noll, samt en passiv uppdateringsregel för de okända parametrarna.

2.3 Designsteg 1

Vi definierar felet i tillståndet $x_1 = y$ som

$$z_1 \triangleq y - y_r$$

Derivation med avseende på tiden ger

$$\dot{z_1} = \dot{y} - \dot{y_r} = \dot{x_1} - \dot{y_r} = x_2 - a_1 y - \dot{y_r}$$

Om detta vore ett första ordningens system med x_2 som styrsignal skulle vi kunna designa en styrlag som stabiliserar felsystemet. Eftersom tillståndet x_2 inte är direkt mätbart använder vi skattningen $\hat{x}_2$ och noterar att

$$x_2 = \hat{x}_2 + \varepsilon_2 = \zeta_{2,2} + \zeta_{(2)}a + v_{(2)}b + \varepsilon_2$$

Index (2) betecknar de filtersignaler som används för skattningen av tillstånd 2, dvs.

$$x_2 = \zeta_{2,2} + [\zeta_{1,2}, \zeta_{0,2}] \begin{bmatrix} -a_1 \\ -a_0 \end{bmatrix} + v_{0,2}b_0 + \varepsilon_2$$

Vi kan nu skriva om felekvationen som

$$\dot{z_1} = x_2 - a_1 y - \dot{y_r} = \zeta_{2,2} + \zeta_{(2)} \begin{bmatrix} -a_1 \\ -a_0 \end{bmatrix} + v_{0,2}b_0 + \varepsilon_2 - a_1 y - \dot{y_r} = \zeta_{2,2} + \omega^T\theta - \dot{y_r} + \varepsilon_2$$

där vi har infört parametervektorn θ och regressorvektorn ω enligt

$$\theta = \begin{bmatrix} -a_1 \\ -a_0 \\ b_0 \end{bmatrix}$$

$$\omega^T = [\zeta_{1,2} + y, \zeta_{0,2}, v_{0,2}]$$

$v_{0,2}$ skattar u:s påverkan på x_2 och får därför verka som virtuell styrsignal till systemet. Med

$$\overline{\omega}^T = [\zeta_{(2)} + e_1^T y, \overline{v_2}]$$

där

$$\overline{v_2} = 0$$

kan felekvationen skrivas som

$$\dot{z_1} = b_0 v_{0,2} + \zeta_{2,2} + \overline{\omega}^T \theta - \dot{y_r} + \varepsilon_2$$

Vi definierar nu felet i den virtuella styrsignalen α_1 som

$$z_2 = v_{0,2} - \alpha_1$$

eftersom $v_{0,2}$ verkar som virtuell styrsignal.
Då kan felekvationen skrivas om till

$$\dot{z_1} = -c_1 z_1 - d_1 z_1 + b_0 z_2 + b_0(\alpha_1 + p(c_1 z_1 + d_1 z_1 + \zeta_{2,2} - \dot{y_r}) + p\overline{\omega}^T \theta) + \varepsilon_2$$

där p är inversen av b_0.
Sätt

$$\varphi = c_1 z_1 + d_1 z_1 + \zeta_{2,2} - \dot{y_r} + \overline{\omega}^T \theta$$

Ett lämpligt val av α_1 som stabiliserar $\dot{z_1}$ är

$$\alpha_1 = -\hat{p}\varphi = -\hat{p}(c_1 z_1 + d_1 z_1 + \zeta_{2,2} - \dot{y_r} + \overline{\omega}^T \hat{\theta})$$

De passiva uppdateringarna väljs som

$$\dot{\hat{p}} = \gamma \varphi z_1 sgn(b_0)$$

respektive

$$\dot{\hat{\theta}} = \tau_1$$

där

$$\tau_1 = \Gamma \overline{\omega} z_1$$

För detaljer i härledningen, se (1,2). Man kan tolka τ_1 som en "virtuell" utsignal. Observera att tecknet på b_0 måste vara känt.

2.4 Designsteg 2

Vi har redan definierat felet z_2 som

$$z_2 = v_{0,2} - \alpha_1$$

Derivation med avseende på tiden ger

$$\dot{z_2} = u + \beta_2 - \frac{\partial \alpha_1}{\partial y}\omega^T \theta - \frac{\partial \alpha_1}{\partial y}\varepsilon_2 - \frac{\partial \alpha_1}{\partial \hat{\theta}}\dot{\hat{\theta}}$$

där

$$\beta_2 = -k_2 v_{0,1} - \frac{\partial \alpha_1}{\partial y}\zeta_{2,2} - \frac{\partial \alpha_1}{\partial y_r}\dot{y_r} - \frac{\partial \alpha_1}{\partial \dot{y_r}}\ddot{y_r} - \frac{\partial \alpha_1}{\partial \zeta_2}(A_0\zeta_2 + ky) - \sum_{i=0}^{1}\frac{\partial \alpha_1}{\partial \zeta_i}(A_0\zeta_i + e_{2-i}y) - \frac{\partial \alpha_1}{\partial \hat{p}}\gamma\varphi z_1 sgn(b_0)$$

innehåller enbart kända termer.
För att kunna fortsätta måste vi först bestämma de partiella derivatorna i ovanstående uttryck.

$$\frac{\partial \alpha_1}{\partial y} = -\hat{p}(c_1 + d_1 - \hat{a}_1)$$

$$\frac{\partial \alpha_1}{\partial \hat{\theta}} = -\hat{p}\overline{\omega}^T$$

$$\frac{\partial \alpha_1}{\partial y_r} = \hat{p}(c_1 + d_1)$$

$$\frac{\partial \alpha_1}{\partial \dot{y}_r} = \hat{p}$$

$$\frac{\partial \alpha_1}{\partial \zeta_2} = -\hat{p}\,[01]$$

$$\sum_{i=0}^{1} \frac{\partial \alpha_1}{\partial \zeta_i} = \hat{p}\begin{bmatrix} 0 & \hat{a}_0 \end{bmatrix} + \hat{p}\begin{bmatrix} 0 & \hat{a}_1 \end{bmatrix}$$

$$\frac{\partial \alpha_1}{\partial \hat{p}} = -\varphi$$

Vi kan nu designa den verkliga styrlagen u och den verkliga uppdateringen $\dot{\hat{p}}$ och $\dot{\hat{\theta}}$ enligt tidigare stabilitets- och passivitetsresonemang.

Välj styrlagen u som

$$u = -c_2 z_2 - d_2(\frac{\partial \alpha_1}{\partial y})^2 z_2 - b_0 z_1 - \beta_2 + \frac{\partial \alpha_1}{\partial y}\omega^T \hat{\theta} + \frac{\partial \alpha_1}{\partial \hat{\theta}}\tau_2$$

och uppdateringen som

$$\dot{\hat{p}} = \gamma \varphi z_1 sgn(b_0)$$

respektive

$$\dot{\hat{\theta}} = \tau_2 = \Gamma(\omega \begin{bmatrix} 1 & -\frac{\partial \alpha_1}{\partial y} \end{bmatrix} z + \begin{bmatrix} 0 & 0 & z_1 \hat{p} \varphi \end{bmatrix}^T)$$

För detaljerade härledningar samt stabilitets och passivitetsbevis, se (1,2).

3 Simulering

3.1 Inledning

För att studera uppförandet av backsteppingregleringen utfördes ett antal simuleringar där γ och referenssignalens utseende varierades. Försök gjordes på både en stabil process, $G(s) = \frac{1}{s^2+2s+1}$ och en instabil, $G(s) = \frac{1}{s^2-s+1}$. Referensmodellens överföringsfunktion ges av $G_m(s) = \frac{1}{s^2+1.4s+1}$. Som referenssignal användes både en fyrkantvåg och en sinussignal ($r = \sin t$).

3.2 Implementering

Följande uttryck implementerades i Simnonkod :
Process :

$$G(s) = \frac{b_0}{s^2 + a_1 s + a_0}$$

Referensmodell :

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

η-filter :

$$\dot{\eta} = A_0\eta + e_2 y$$

λ-filter :

$$\dot{\lambda} = A_0\lambda + e_2 u$$

Styrlag :

$$u = -c_2 z_2 - d_2(\frac{\partial\alpha_1}{\partial y})^2 z_2 - b_0 z_1 - \beta_2 + \frac{\partial\alpha_1}{\partial y}\omega^T\hat{\theta} + \frac{\partial\alpha_1}{\partial\hat{\theta}}\tau_2$$

Uppdatering :

$$\dot{\hat{p}} = \gamma\varphi z_1 sgn(b_0)$$

$$\dot{\hat{\theta}} = \tau_2 = \Gamma(\omega \begin{bmatrix} 1 & -\frac{\partial\alpha_1}{\partial y} \end{bmatrix} z + \begin{bmatrix} 0 & 0 & z_1\hat{p}\varphi \end{bmatrix}^T)$$

En schematisk bild över hur implementeringen är gjord visas sist i rapporten.

Utöver dessa måste en del designparametrar bestämmas. k_1 och k_2 placerar skattningsfilternas poler. c_1, c_2 bestämer felsystemets poler och d_1, d_2 bestämmer systemets olinjära dämpning. Vidare har vi 10 stycken adaptionsförstärkningar att sätta, och för enkelhetens skull valdes Γ-matrisen för uppdatering av $\hat{a_0}$, $\hat{a_1}$, $\hat{b_0}$ diagonal, $\gamma \cdot I$, med samma γ som för uppdateringen av $\hat{p}$.

3.3 Resultat av simuleringarna

Inverkan av adaptionsförstärkninen γ :
Med stort γ blev utsignalföljningen och tillståndsfelen bra i jämförelse med de simuleringar då γ valdes liten. Däremot gjorde parameterestimeringen mindre "ryck" då γ var liten, och dessutom blev styrsignalen bättre i detta fall (jämför figur 1-4 med figur 5–8). Eftersom vi inte kunde inse varför styrsignalen blev så brusig, prövade vi med att ändra integrationsmetod till Runge-Kutta. Styrsignalen blev då betydligt jämnare.

Möjligen skulle man kunna minska rycken i estimeringen genom att införa en glömskefaktor, det vill säga modifiera backsteppingmetoden något.
En annan anmärkning som gjorts är att parametrarna ej konvergerar mot rätt värden.Ett försök gjordes också med nollställd estimator, men detta misslyckades helt.

Det finns dock teori som pekar på att detta skulle fungera.

Inverkan av referenssignal :
Vid byte från fyrkant- till sinussformad referenssignal märktes ingen större skillnad vad beträffar utsignalföljningen, men tillståndsfelen, parameterestimeringen och styrsignalen blev bättre, det vill säga jämnare då sinussignal användes (jämför figur 1-4 med figur 13-16).

Inverkan av process :
Då processen ändrades från stabil till instabil märktes en liten skillnad i utsignalföljningen och tillståndsfelen, speciellt då referenssignalen var en fyrkantvåg (jämför figur 1-4 med figur 9-12).

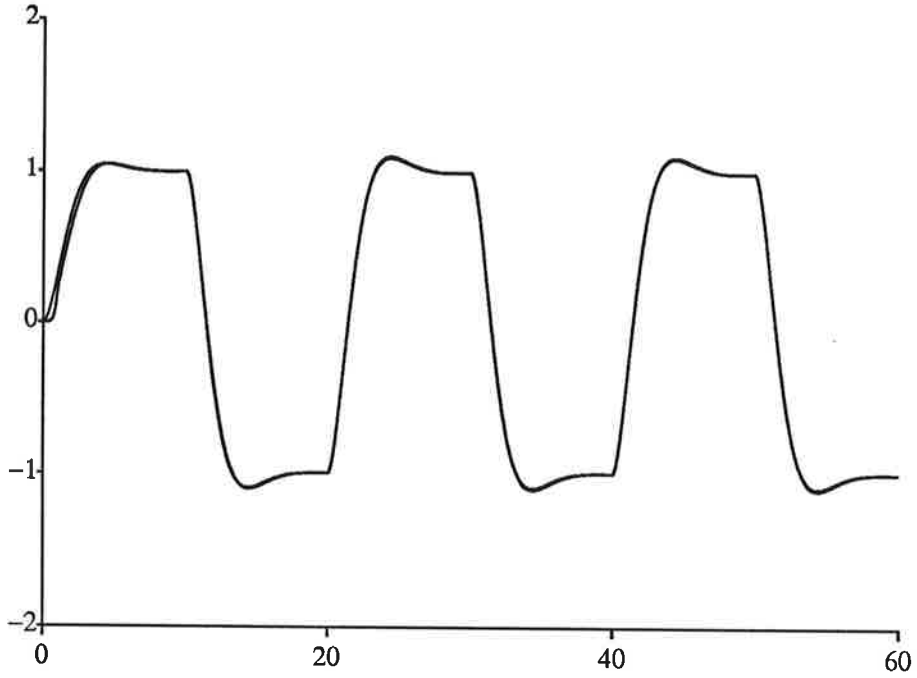
3.4 Kommentarer till simuleringsresultaten

Säkerligen skulle man kunna hitta en parameterinställning som ger bättre simuleringsresultat, men då tiden varit begränsad och vi ej metodiskt kunna välja parametrarna stannade vi här. Naturligtvis finns det saker att förbättra i vårt arbete och förmodligen också i utvecklingen av backsteppingmetoden. Förhoppningsvis kan vår rapport ligga till grund för fortsatta projekt om backstepping.

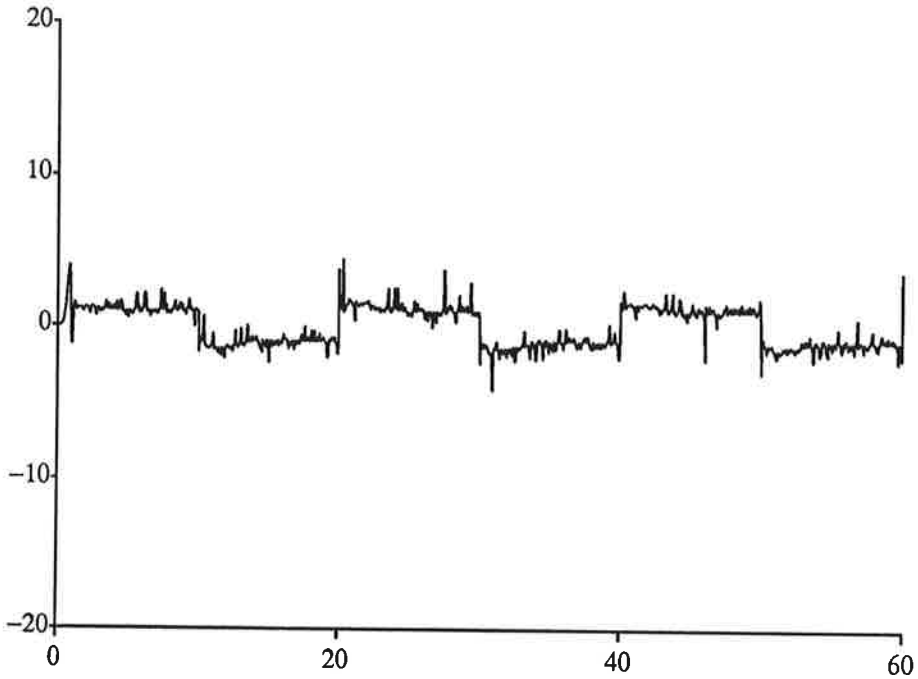
4 Referenser

(1) P V Kokotovic, M Krstic, I Kanellakopoulos, Backstepping to Passivity: Recursive Design of Adaptive Systems
(2) P V Kokotovic, M Krstic, I Kanellakopoulos, A New Generation of Adaptive Controllers for Linear Systems

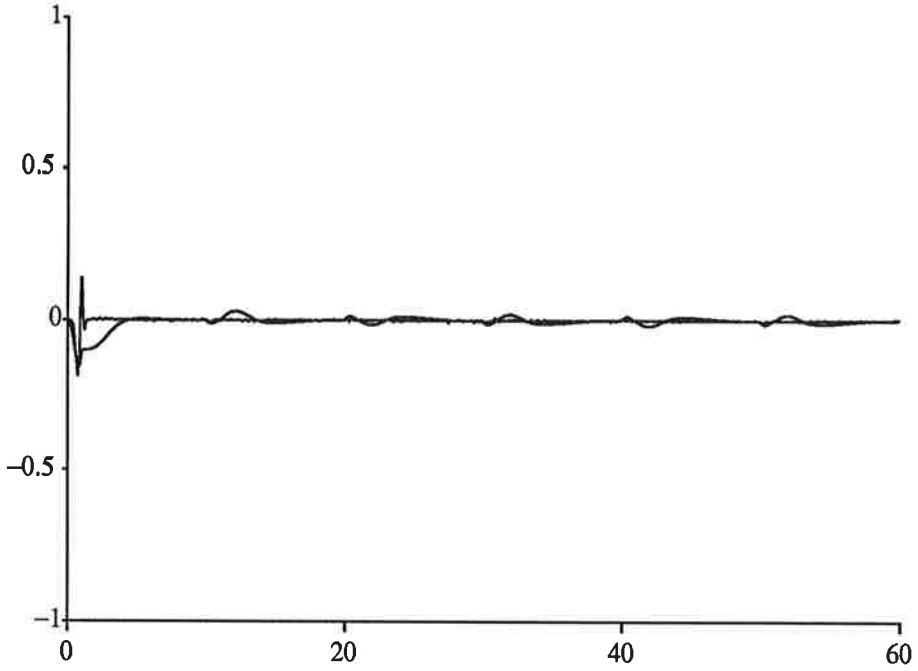
Figur 1: Simulering av stabil process, $\gamma = 10, k_1 = k_2 = 3, c_1 = c_2 = 5, d_1 = d_2 = 0.5$, referenssignal fyrkantvåg. Bilden visar utsignalföljning, y respektive y_r.



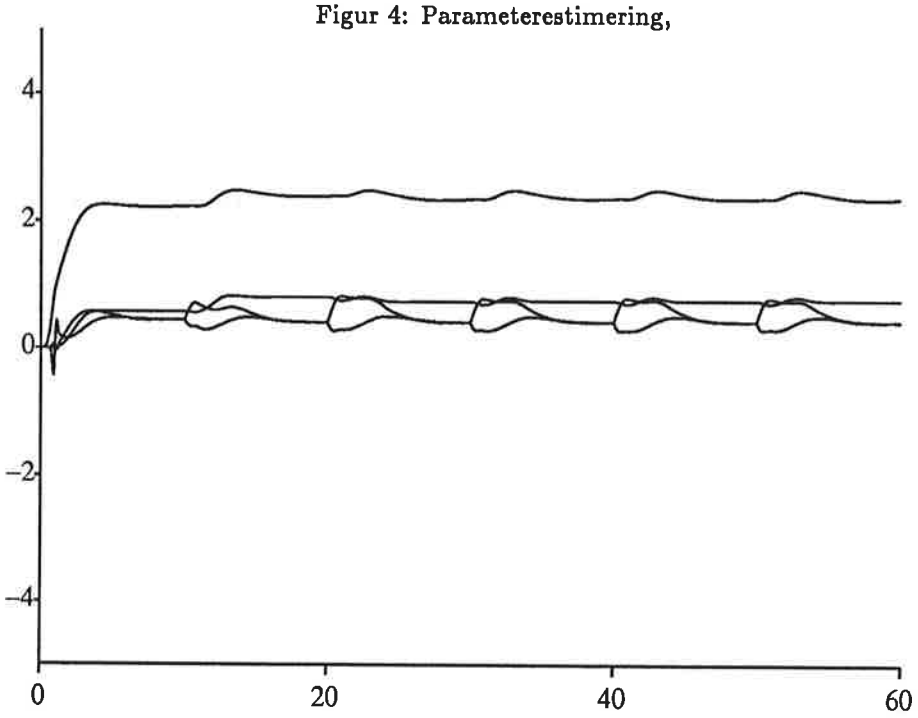
Figur 2: Styrsignal u.



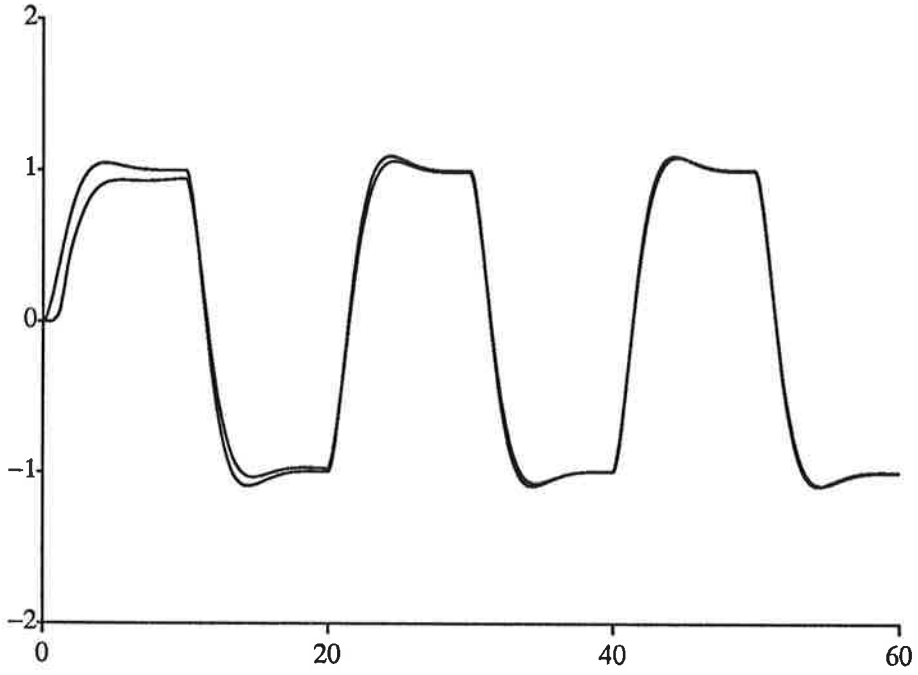
Figur 3: Tillståndsfelen z_1 och z_2



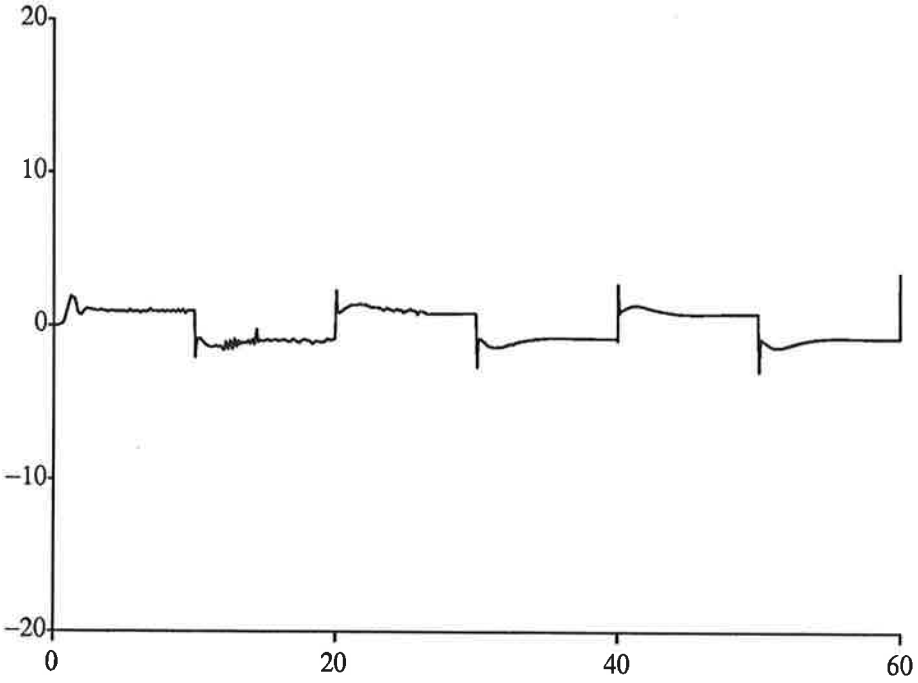
Figur 4: Parameterestimering,



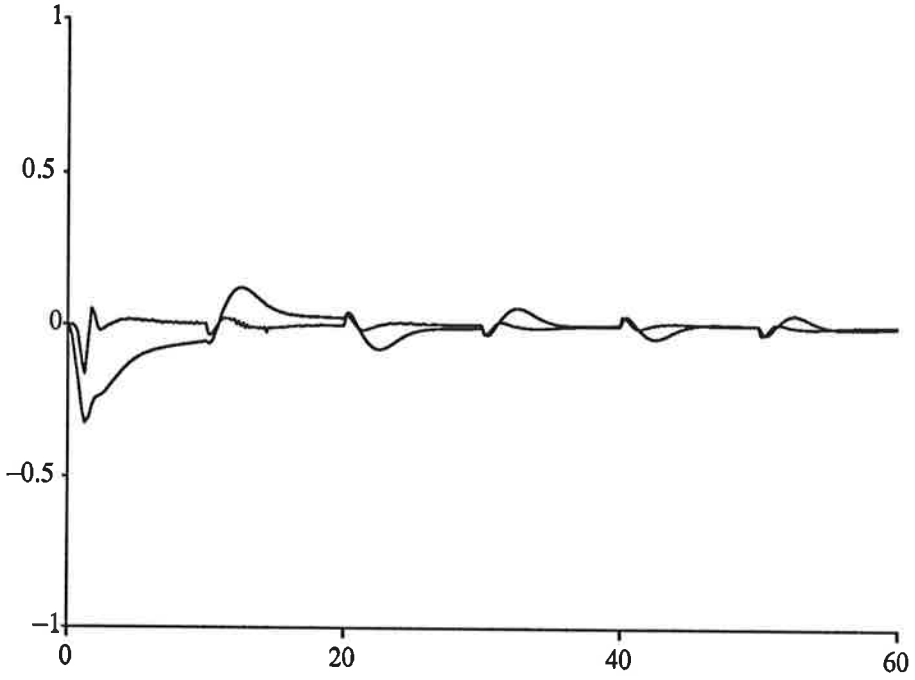
Figur 5: Simulering av stabil process, $\gamma = 1, k_1 = k_2 = 3, c_1 = c_2 = 5, d_1 = d_2 = 0.5$, referenssignal fyrkantvåg. Bilden visar utsignalföljning, y respektive y_r.



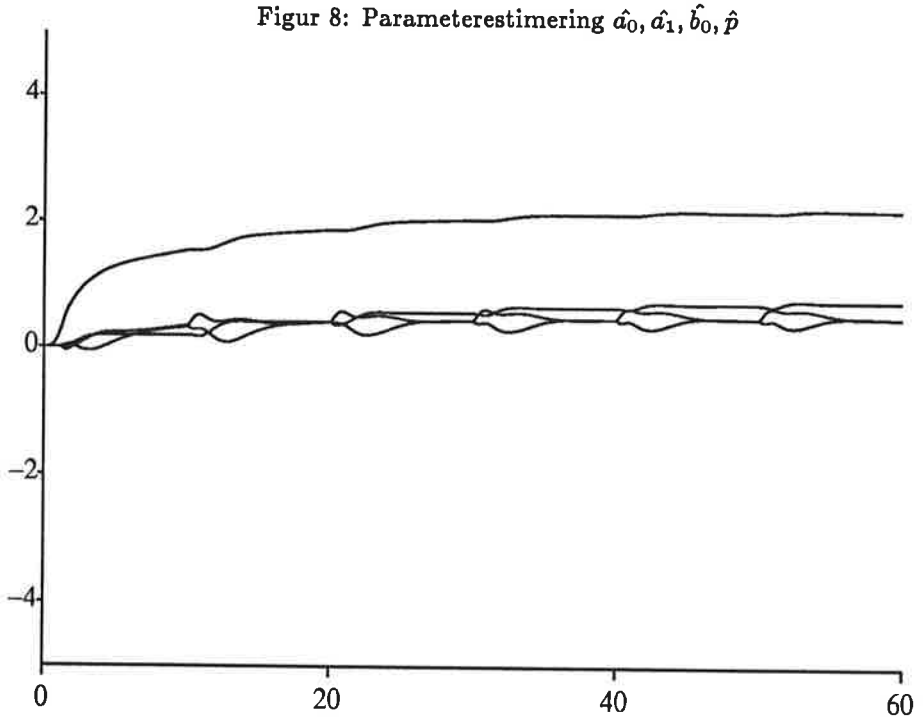
Figur 6: Styrsignal u.



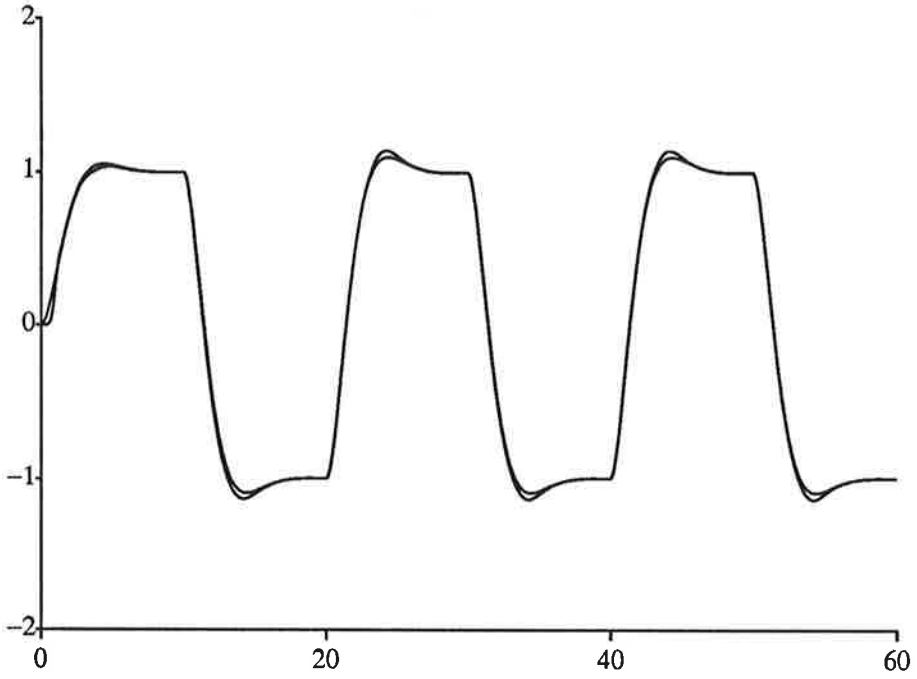
Figur 7: Tillståndsfelen z_1 och z_2



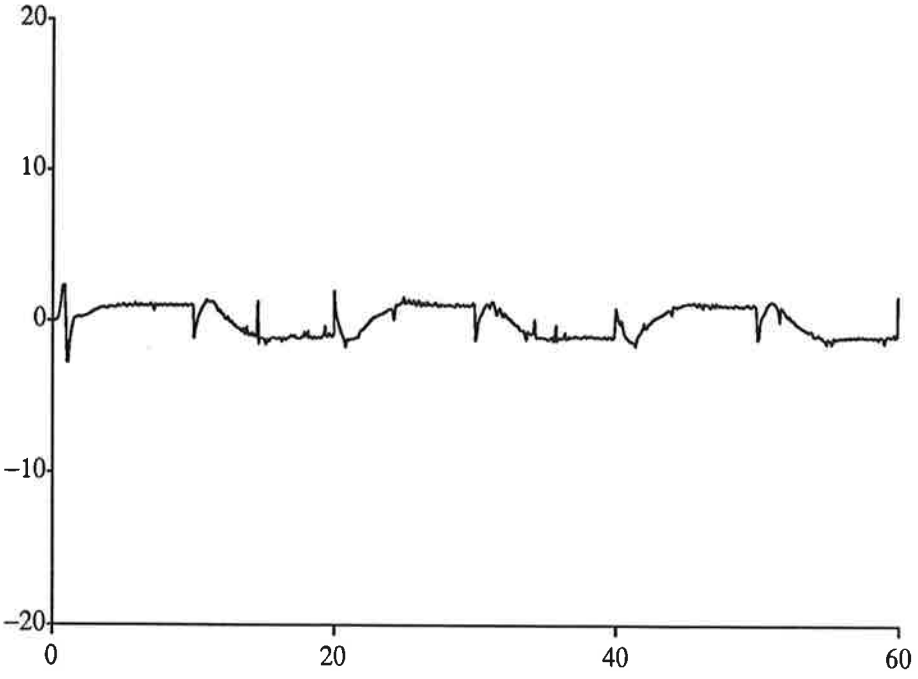
Figur 8: Parameterestimering $\hat{a_0}, \hat{a_1}, \hat{b_0}, \hat{p}$



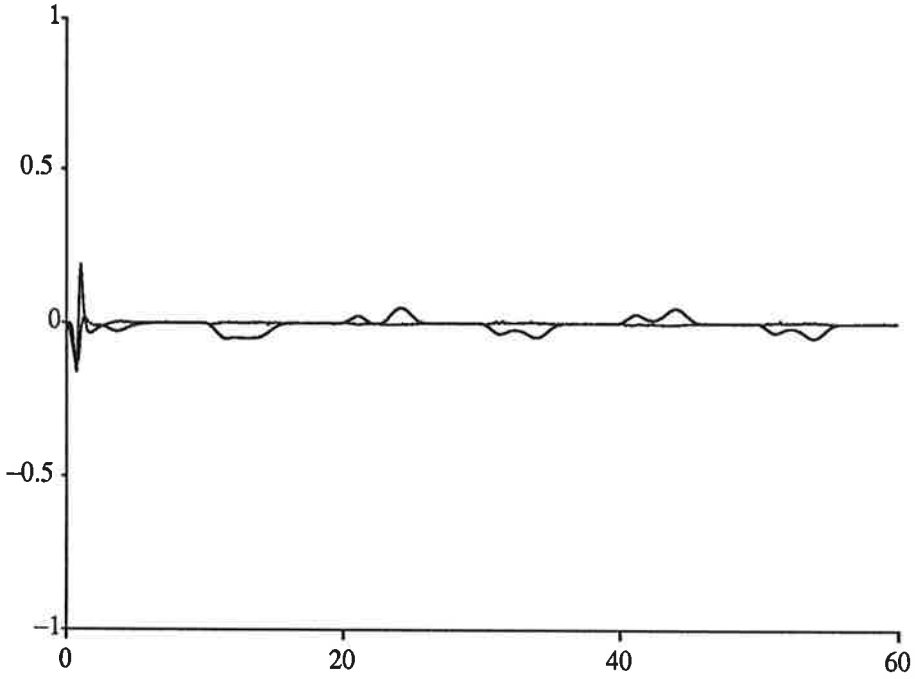
Figur 9: Simulering av instabil process, $\gamma = 10, k_1 = k_2 = 3, c_1 = c_2 = 5, d_1 = d_2 = 0.5$, referenssignal fyrkantvåg. Bilden visar utsignalföljning, y respektive y_r.



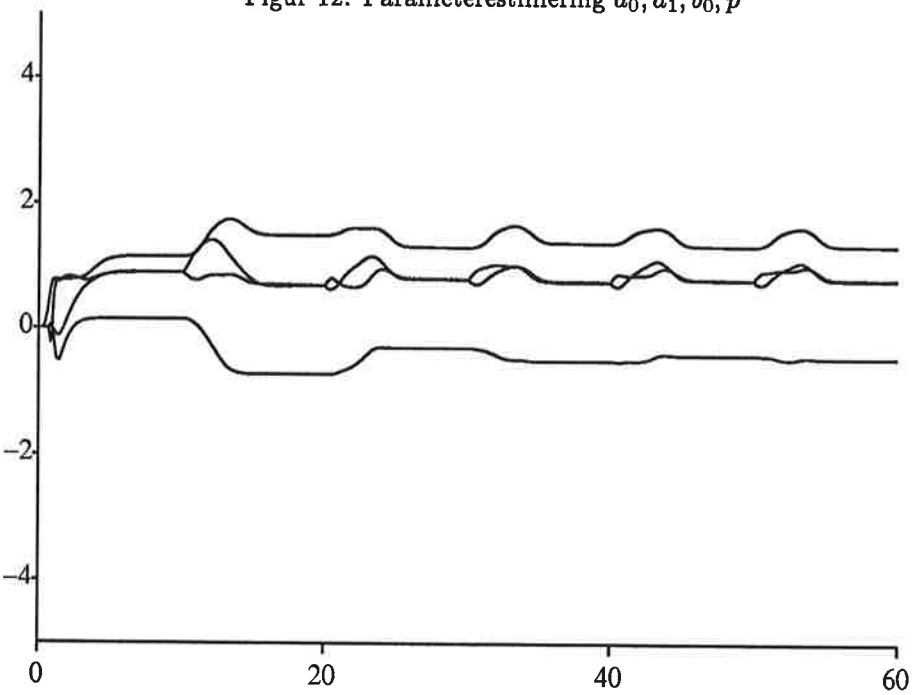
Figur 10: Styrsignal u.



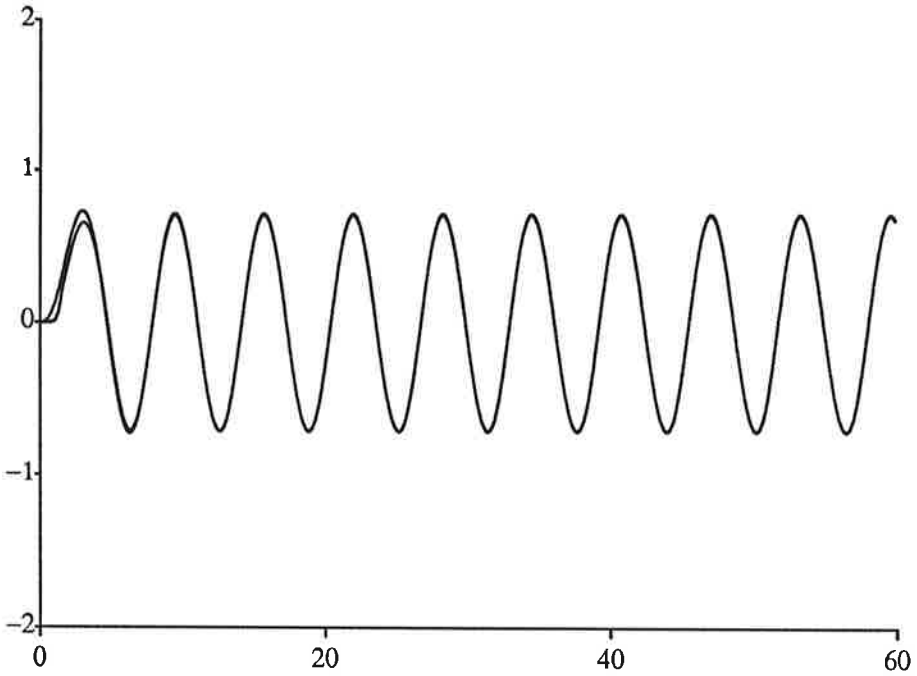
Figur 11: Tillståndsfelen z_1 och z_2



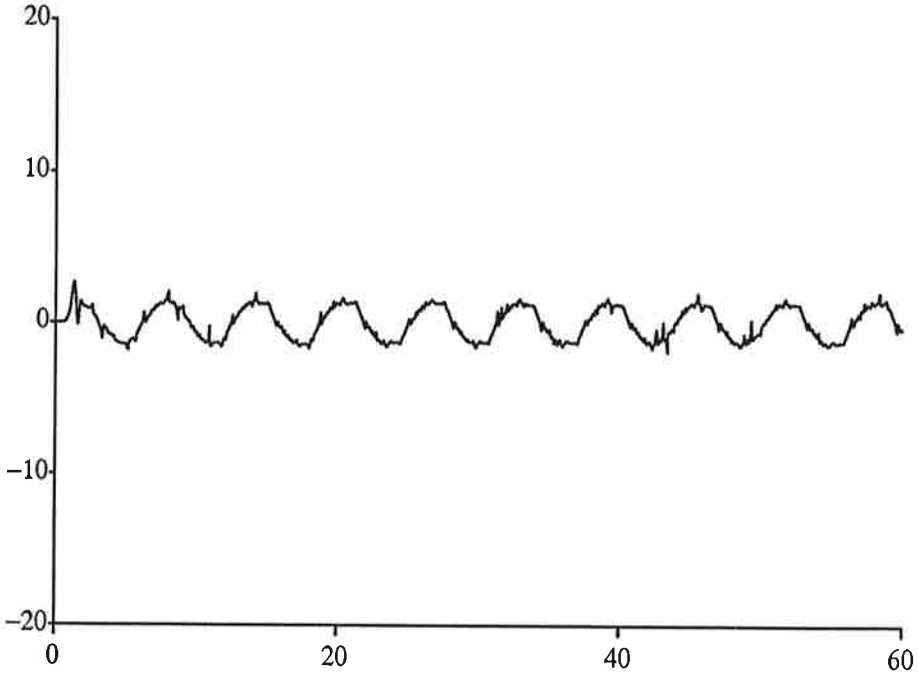
Figur 12: Parameterestimering $\hat{a_0}, \hat{a_1}, \hat{b_0}, \hat{p}$



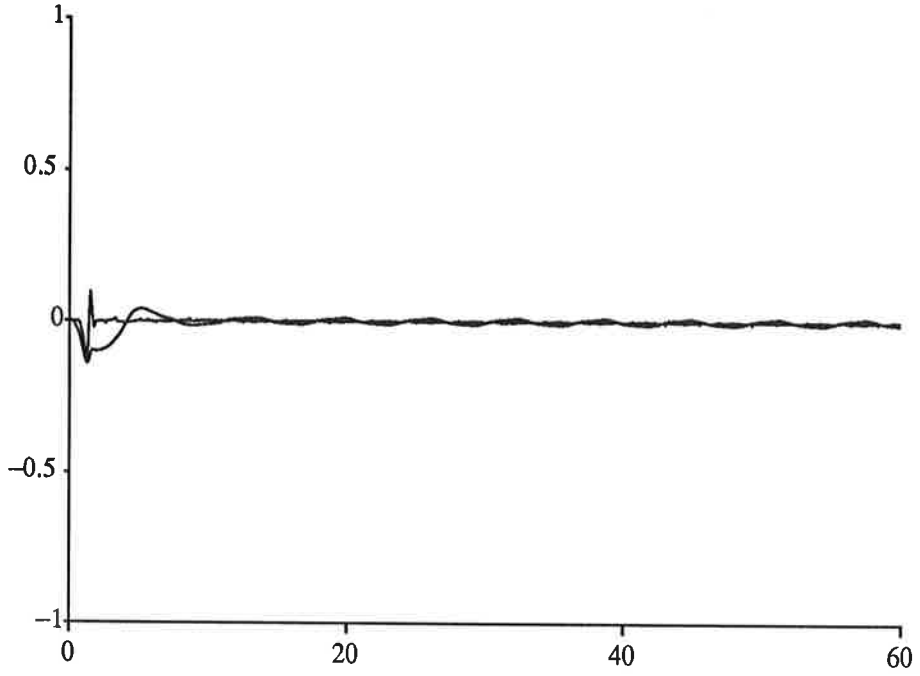
Figur 13: Simulering av stabil process, $\gamma = 10, k_1 = k_2 = 3, c_1 = c_2 = 5, d_1 = d_2 = 0.5$, sinusformad referenssignal. Bilden visar utsignalföljning, y respektive y_r.



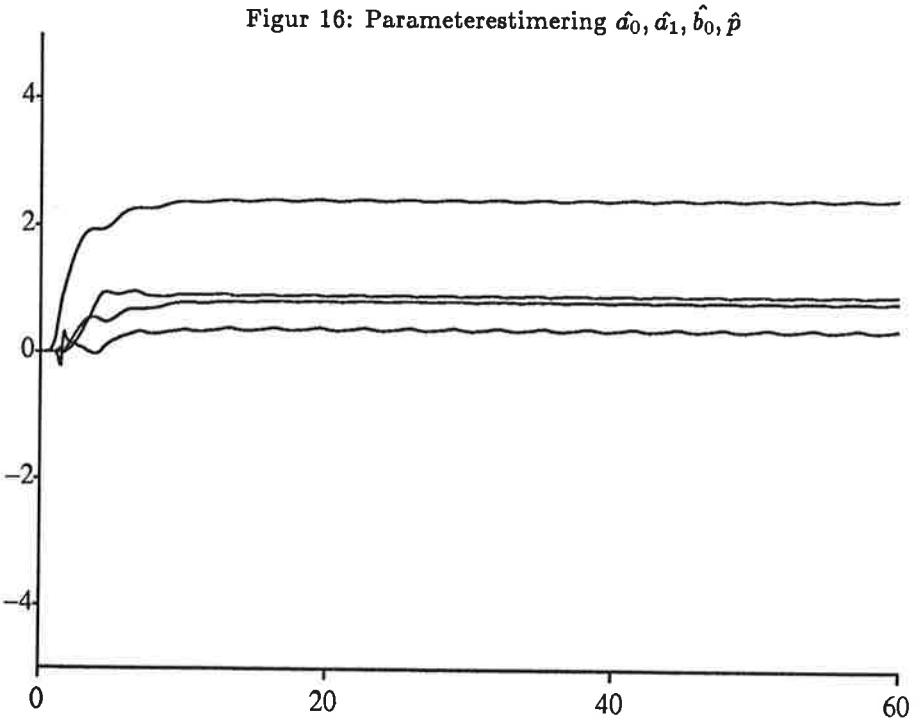
Figur 14: Styrsignal u.



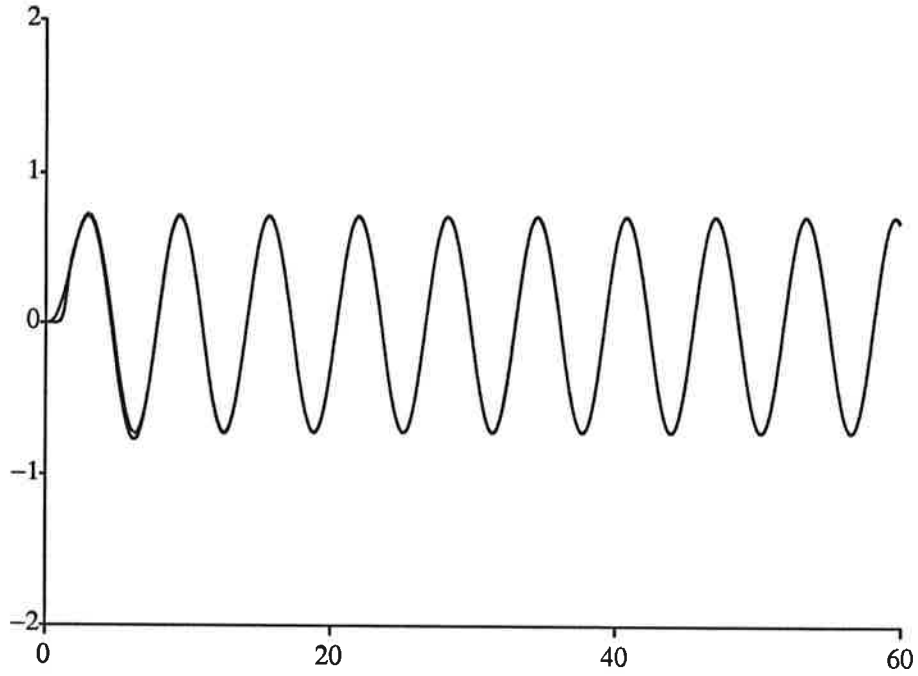
Figur 15: Tillståndsfelen z_1 och z_2



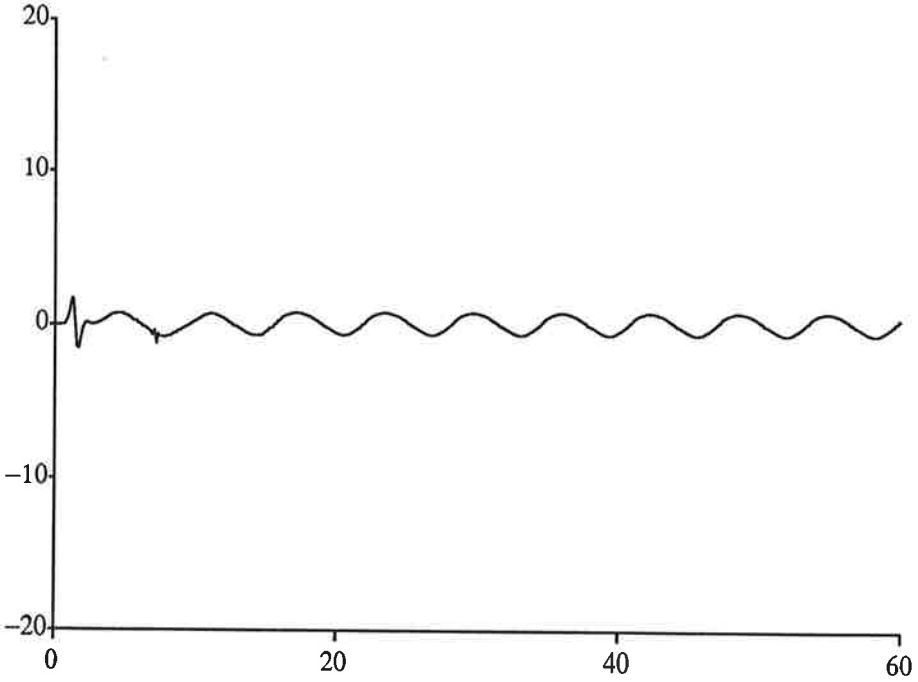
Figur 16: Parameterestimering $\hat{a_0}, \hat{a_1}, \hat{b_0}, \hat{p}$



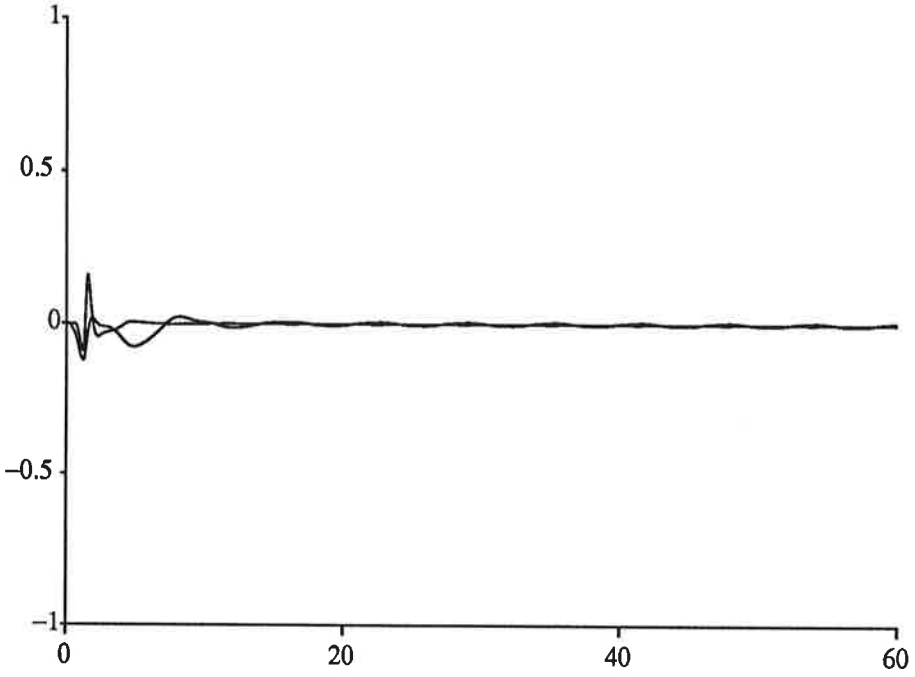
Figur 17: Simulering av instabil process, $\gamma = 10, k_1 = k_2 = 3, c_1 = c_2 = 5, d_1 = d_2 = 0.5$, sinusformad referenssignal. Bilden visar utsignalföljning, y respektive y_r.



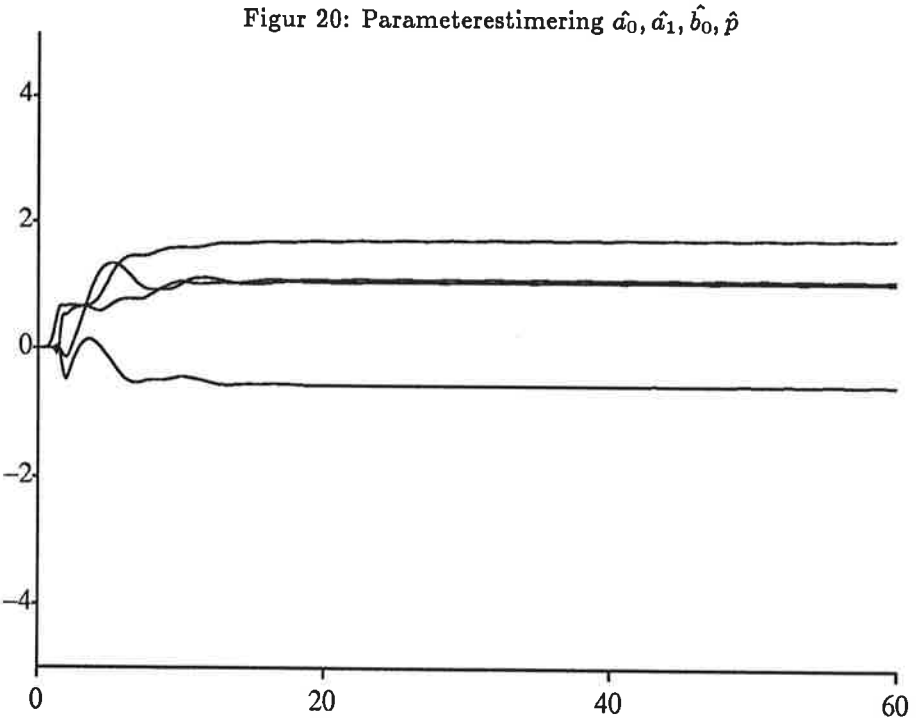
Figur 18: Styrsignal u.



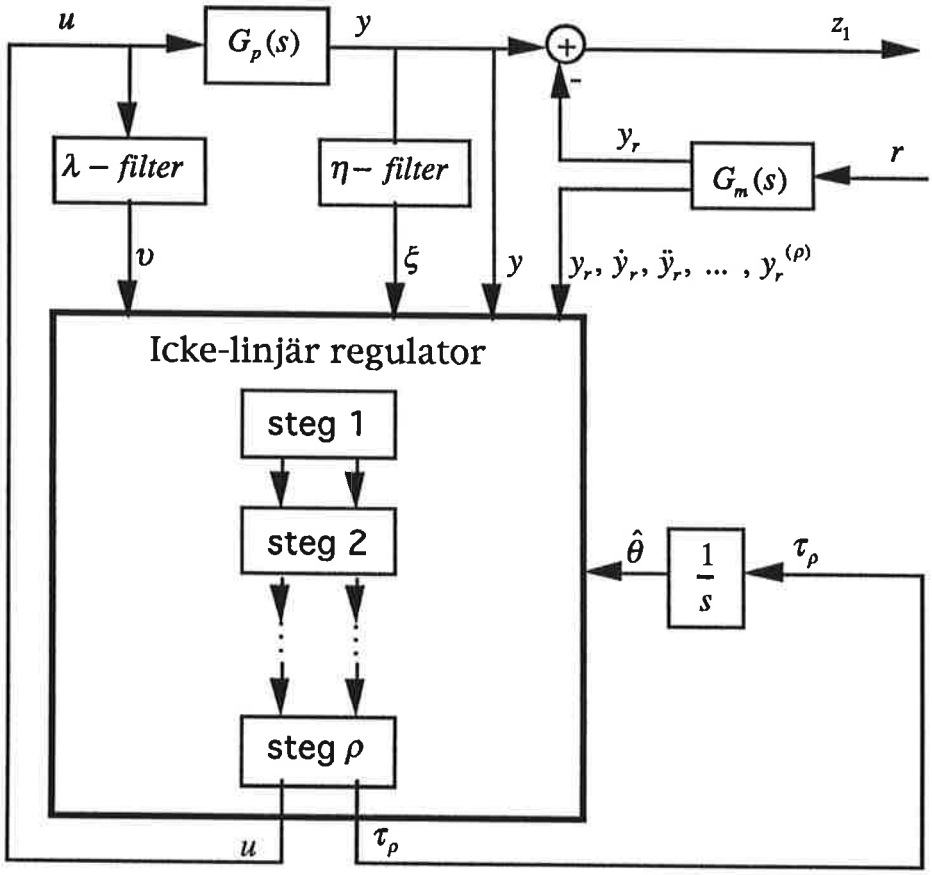
Figur 19: Tillståndsfelen z_1 och z_2



Figur 20: Parameterestimering $\hat{a_0}, \hat{a_1}, \hat{b_0}, \hat{p}$



Backstepping adaptiv system.



Projekt i adaptiv reglering.

Simulering av "backstepping"-metoden.

Lund 19 maj, 1993

Pär Larsson, d88pl
Niclas Olsson, d88no

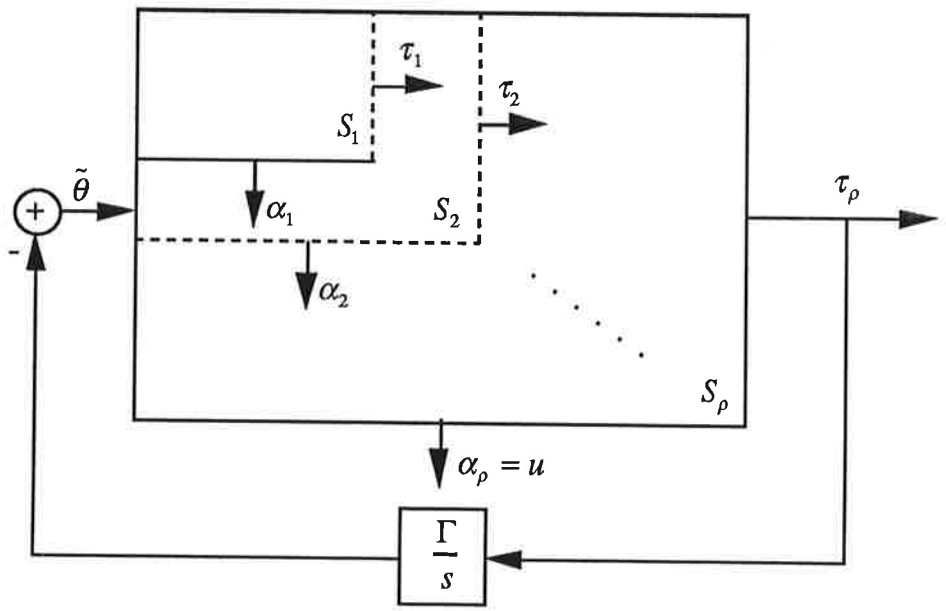
Handledare: Ulf Jönsson

1. Inledning.

I reglerteknikkurserna har vi nästan uteslutande studerat linjära processer och linjära regulatorer. Vi skall nu ta ett steg in i den olinjära världen och titta på en ny designmetod som heter backstepping-metoden.
Backstepping är en metod att, i ett antal steg, designa en regulator för en process som är antingen linjär eller olinjär, där själva regulatorn faktiskt blir olinjär. Man kan förenklat säga att man betraktar hela problemet som ett olinjärt problem, vare sig man har en linjär eller en olinjär process, och resultatet blir en regulator som är olinjär.
Backstepping är en ny metod. De artiklar vi har läst för att lära oss metoden är från december 1992. Författare är tre herrar från University of California i Los Angeles och Santa Barbara (se referenser). Målet med projektet var att implementera en "backstepping-regulator" i SIMNON för en process, som i vårt fall var en tredje ordningens process, och jämföra denna med en självinställare (STR). I denna rapport presenterar vi resultaten av denna jämförelse samt ett kortfattat avsnitt om teorin bakom backstepping-metoden.

2. Backstepping-metoden.

Backstepping bygger på passivitetsbegreppet. Passivitet är en egenskap hos ett system. Ett strikt passivt system återkopplat med ett passivt system resulterar i ett stabilt system, och det är detta man utnyttjar i backstepping-metoden.

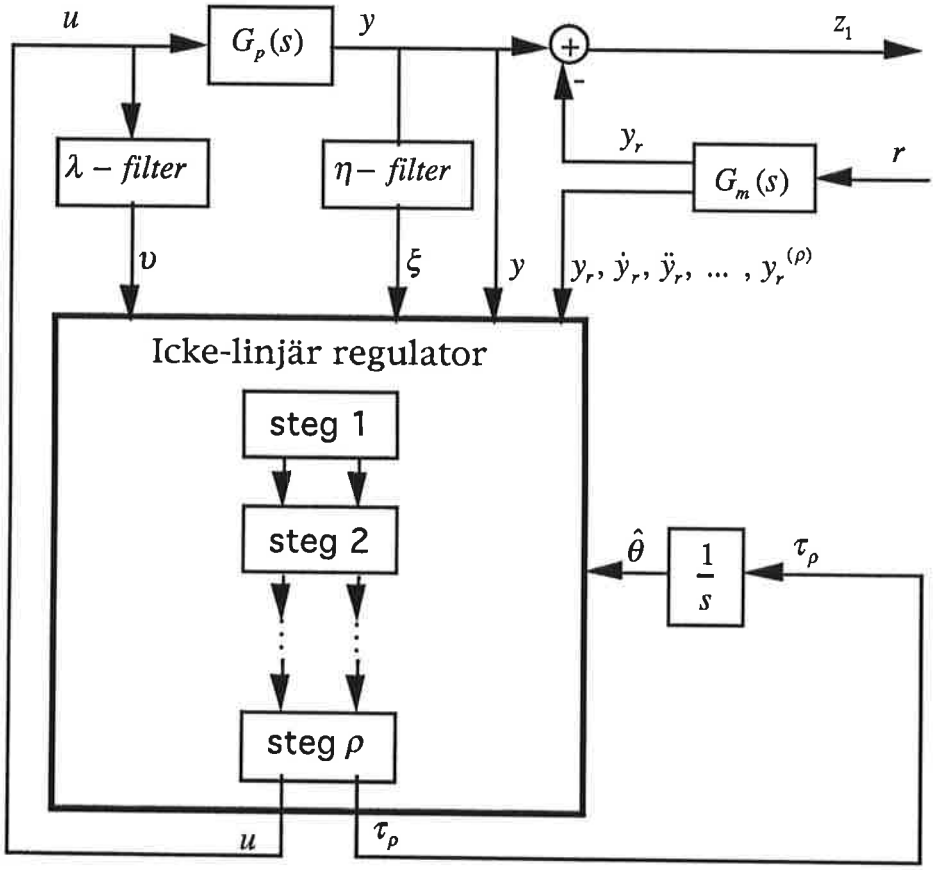


Idén med backstepping, som visas i figuren ovan, är att designa en sekvens av så kallade virtuella system, S_i, med relativt gradtal ett, där man avslutar med det verkliga systemet som den sista medlemmen i denna sekvens. För varje virtuellt system, S_i, reduceras det relativa gradtalet ett steg genom att välja den virtuella styrsignalen, α_i, och den virtuella utsignalen, τ_i, så att systemet, S_i, blir strikt passivt. Valet av dessa signaler är inte entydigt, men genom att välja dem smart, kan man få en systemmatris till systemen, S_i, som är skevsymmetrisk med negativa diagonalelement och som dessutom innehåller en olinjär dämpterm, $-d_i(\frac{\partial \alpha_{i-1}}{\partial y})^2 z_i$, vilken tillskrivs en hel del av de trevliga egenskaper som dessa system uppvisar. Den sista virtuella utsignalen, τ_p, används för att sluta den adaptiva återkopplingsloopen via en passiv parameteruppdateringslag. Den sista virtuella styrsignalen, α_p, blir den verkliga styrsignalen, u, till processen.

3. Implementering.

När man implementerar regulatorn behöver man lyckligtvis inte implementera de olika systemen, S_i. Det räcker med att man skriver kod för alla signalerna, α_i och τ_i, vilket är tillräckligt jobbigt ändå. Figuren på nästa sida visar hur det kompletta systemet ser ut i blockform. Denna figur beskriver också väldigt bra hur SIMNON-koden ser ut. Varje block i figuren representerar ett CONTINUOUS SYSTEM i SIMNON och alla pilar mellan blocken blir koden i CONNECTING SYSTEM.
Om vi börjar med de mest triviala blocken, så finns där process och referensmodell. Det man bör observera här, är att man behöver ett antal derivator av referensmodellens utsignal. Antalet bestäms av ordningen på processen. Dessutom finns där ett antal filter, benämnda med λ och η, som används för att skatta tillstånden i processen och de parametrar i processen som är okända. Det exakta antalet filter avgörs dels av antalet tillstånd, och dels av antalet okända parametrar. Till sist återstår det största blocket av dem alla, den olinjära regulatorn (med parameteruppdatering). Här implementerar man uttrycken för alla signalerna α_i och τ_i, som man fått fram i de olika stegen. Dessa uttryck blir i regel

komplicerade, och exempel på hur uttrycken ser ut för ett tredje ordningens system hittar man längre fram i denna rapport.



4. Design av regulator.

Vår uppgift var att, för ett tredje ordningens system, göra en kontinuerlig adaptiv regulator baserad på backstepping-metoden . Vi utgick då från ett system skrivet på tillståndsform innehållande en generell funktion, $\varphi(y)$:

$$\begin{cases} \dot{x}_1 = x_2 + a\varphi(y) \\ \dot{x}_2 = x_3 \\ \dot{x}_3 = u \end{cases}$$

$$y = x_1$$

Sättes $\varphi(y)$ lika med y, d.v.s. x_1, erhålles en linjär process med följande överföringsfunktion:

$$y(s) = \frac{1}{s^2(s-a)}u(s)$$

Processen innehåller alltså en okänd parameter, a, som måste skattas. Därefter gällde det att designa regulatorn i de steg, som ovan har beskrivits för en generell process. För vår del blev det tre steg, eftersom vår process var av tredje ordningen. Vi gjorde här ej själva härledningen av de olika stegen utan utnyttjade ett exempel som fanns i artikeln "A New Generation of Adaptive Controllers for Linear Systems *" (se referenser). Nedan visas uttrycken i de olika stegen. Det bör här påpekas att formlerna innehåller ett generellt $\varphi(y)$.

Steg 1:
$$z_1 = y - y_r$$
$$\tau_1 = \gamma\omega z_1$$
$$\alpha_1 = -c_1 z_1 - d_1 z_1 - \xi_{3,2} + \dot{y}_r - \omega\hat{a}$$

Steg 2:
$$z_2 = \upsilon_2 - \alpha_1$$
$$\tau_2 = \tau_1 - \gamma\frac{\partial\alpha_1}{\partial y}\omega z_2$$
$$\alpha_2 = -c_2 z_2 - d_2(\frac{\partial\alpha_1}{\partial y})^2 z_2 - z_1 + k_2\upsilon_1 + \frac{\partial\alpha_1}{\partial y}(\upsilon_2 + \xi_{3,2}) + \frac{\partial\alpha_1}{\partial y_r}\dot{y}_r$$
$$+ \frac{\partial\alpha_1}{\partial\dot{y}_r}\ddot{y}_r + \frac{\partial\alpha_1}{\partial\xi_3}(A_o\xi_3 + ky) + \frac{\partial\alpha_1}{\partial\xi_2}(A_o\xi_2 + e_1\varphi(y)) + \frac{\partial\alpha_1}{\partial y}\omega\hat{a} + \frac{\partial\alpha_1}{\partial\hat{a}}\tau_2$$

Steg 3:
$$z_3 = \upsilon_3 - \alpha_2$$
$$\tau_3 = \tau_2 - \gamma\frac{\partial\alpha_2}{\partial y}\omega z_3$$

$$u = -c_3 z_3 - d_3(\frac{\partial \alpha_2}{\partial y})^2 z_3 - z_2 + k_3 \upsilon_1 + \frac{\partial \alpha_2}{\partial y}(\upsilon_2 + \xi_{3,2}) + \frac{\partial \alpha_2}{\partial y_r}\dot{y}_r$$

$$+ \frac{\partial \alpha_2}{\partial \dot{y}_r}\ddot{y}_r + \frac{\partial \alpha_2}{\partial \ddot{y}_r} y_r^{(3)} + \frac{\partial \alpha_2}{\partial \xi_3}(A_o \xi_3 + ky) + \frac{\partial \alpha_2}{\partial \xi_2}(A_o \xi_2 + e_1 \varphi(y))$$

$$+ \frac{\partial \alpha_2}{\partial \upsilon_1}(\upsilon_2 - k_1 \upsilon_1) + \frac{\partial \alpha_2}{\partial \upsilon_2}(\upsilon_3 - k_2 \upsilon_1) + \frac{\partial \alpha_2}{\partial y}\omega \hat{a} + \frac{\partial \alpha_2}{\partial \hat{a}}\tau_3 - \gamma z_2 \frac{\partial \alpha_1}{\partial \hat{a}}\frac{\partial \alpha_2}{\partial y}\omega$$

I dessa uttryck ingår bl.a. η-filtrets två ξ-vektorer och λ-filtrets υ-vektor samt matrisen

$$A_o = \begin{bmatrix} -k_1 & 1 & 0 \\ -k_2 & 0 & 1 \\ -k_3 & 0 & 0 \end{bmatrix},$$

vilka används för att estimera de icke mätbara tillstånden.

$$e_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \omega = \xi_{2,2} + \varphi(y)$$

Uppdateringen av den estimerade parametern, $\hat{a}$, ser ut enligt följande:

$$\dot{\hat{a}} = \tau_3$$

Ovanstående formler kanske verkar ganska beskedliga, men studerar man dem noggrannare ser man att de innehåller en stor mängd derivator. En stor del av vårt arbete bestod därför av att beräkna dessa, vilket underlättades med hjälp av matematik-programmet MATHEMATICA. Vi redovisar inte alla derivator men visar ett exempel på att uttrycken blev ganska långa, vilket gjorde att den slutliga algoritmen blev tämligen omfattande.

$$\frac{\partial \alpha_2}{\partial y} = c_2 \frac{\partial \alpha_1}{\partial y} + 2d_2 \hat{a} \frac{\partial \alpha_1}{\partial y} \frac{\partial^2 \varphi(y)}{\partial y^2} z_2 + d_2 (\frac{\partial \alpha_1}{\partial y})^3 - 1 - \hat{a}(\upsilon_2 + \xi_{3,2}) \frac{\partial^2 \varphi(y)}{\partial y^2}$$
$$- k_2 - \hat{a}^2 \omega \frac{\partial^2 \varphi(y)}{\partial y^2} + \hat{a} \frac{\partial \alpha_1}{\partial y} \frac{\partial \varphi(y)}{\partial y} - 2\omega z_1 \gamma \frac{\partial \varphi(y)}{\partial y} - \omega^2 \gamma - \gamma \frac{\partial^2 \varphi(y)}{\partial y^2} \hat{a} \omega^2 z_2$$
$$+ 2\gamma \omega \frac{\partial \alpha_1}{\partial y} \frac{\partial \varphi(y)}{\partial y} z_2 - (\frac{\partial \alpha_1}{\partial y})^2 \omega^2 \gamma$$

Ekvationerna innehåller också utsignalen från referensmodellen samt dess derivator och som önskat slutet system valde vi

$$y_r(s) = \frac{1}{(s+1)^3} r(s) \ .$$

Förutom att precisera denna modell finns det som tidigare påpekats ett antal designparametrar som påverkar regulatorns beteende. I vårt fall var dessa tio stycken till antalet , nämligen k_1, k_2, k_3, c_1, c_2, c_3, d_1, d_2, d_3 samt γ. d_1-, d_2- respektive d_3-parametern ingår i de ovan nämnda olinjära dämptermerna, som ju är kännetecknande för denna regleralgoritm. Samtliga parametrar skall vara positiva, men kan annars varieras mycket och det krävs därför åtskilligt arbete för att uppnå en optimal reglering. I våra simuleringar som följer nedan har vi ingalunda åstadkommit den perfekta regulatorn, men kan likväl studera den nya reglermetodens egenskaper.

5. Kontinuerlig indirekt STR.

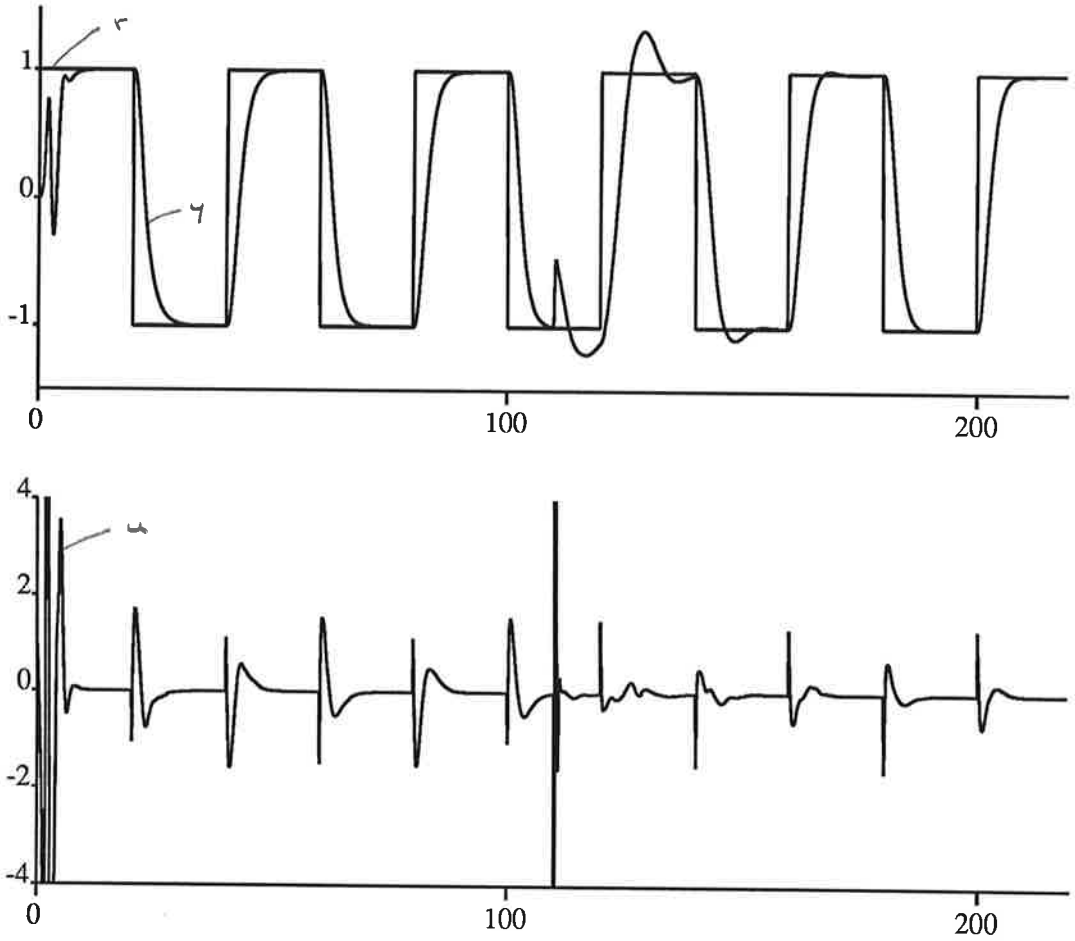
För att ha något att jämföra vår "backstepping-regulator" med, designade vi också en kontinuerlig indirekt STR utan integrator. Vid estimeringen av den linjära processens okända a-parameter använde vi kontinuerlig minsta-kvadrat-estimering, vilken finns beskriven i "ADAPTIVE CONTROL", kapitel 3 (se referenser). För att kunna göra detta skrev vi om systemet på följande sätt:

$$\frac{s^3}{(s+a_f)^3} Y(s) - \frac{1}{(s+a_f)^3} U(s) = \frac{s^2}{(s+a_f)^3} Y(s) a \quad ,$$

där a_f är en designparameter i estimeringsfiltret. Vi betecknade sedan högerledet med y_f och vänsterledet med $\varphi^T\theta$ där $\theta = a$, och skickade in y_f och φ^T till estimatorn för att skatta $\hat{a}$. Vi designade sedan en kontinuerlig RST-regulator med samma önskade slutna system som för "backstepping-regulatorn". Andra designparametrar var a_{o1} respektive a_{o2} i ett andra ordningens observerarpolynom samt p i minsta-kvadrat-skattningen av a-parametern. Liksom i fallet för föregående regulator lade vi även här ner ett begränsat arbete på att ställa in ovanstående designparametrar, men erhöll likväl en regulator som passade bra som jämförelse till den med backstepping-metoden designade adaptiva regulatorn.

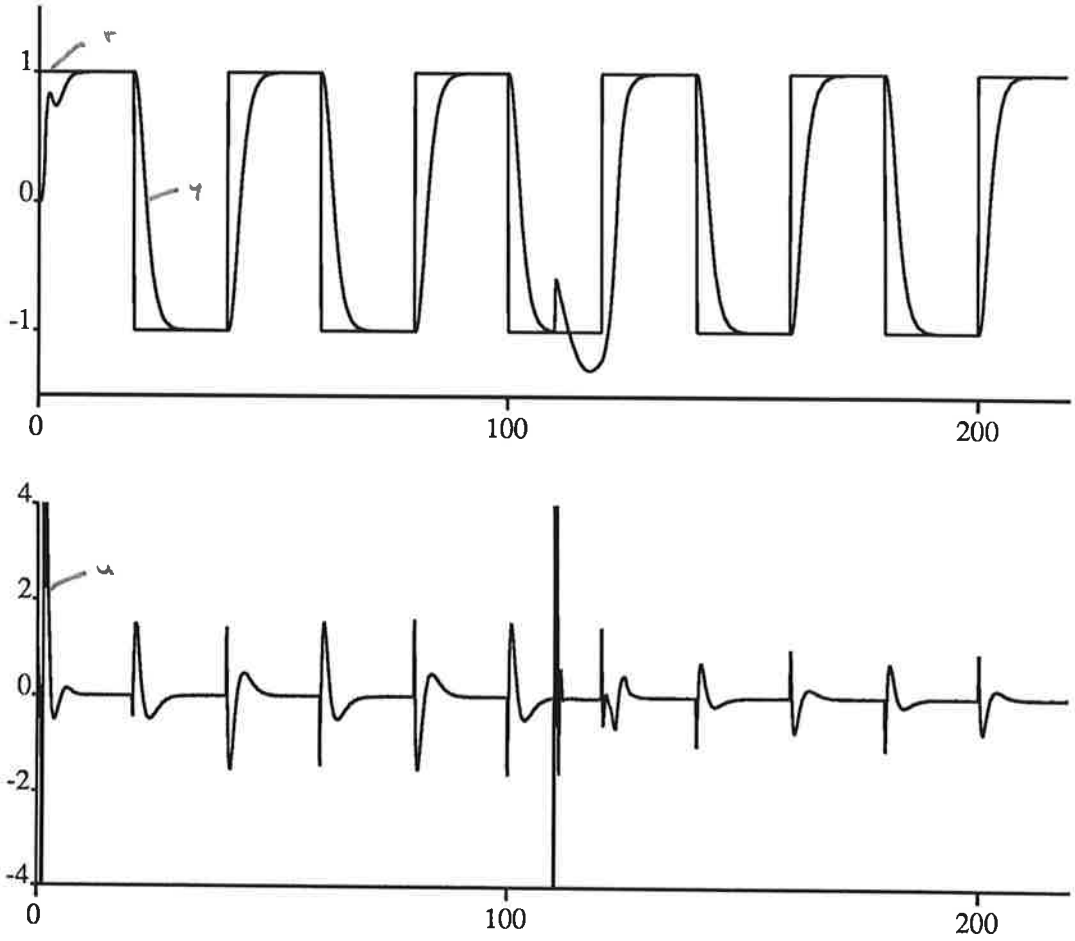
6. Simuleringar.

Det återstår nu att redovisa ett antal simuleringar som visar hur "backstepping-regulatorn" fungerar jämfört med den indirekta STR:n. Vi utförde dessa simuleringar med simuleringsprogrammet SIMNON, och de filer vi använt finns listade i appendix nedan. Vi använde oss genomgående av en fyrkantvåg som referenssignal, r. Samtliga tillstånd och skattningen av a-parametern, $\hat{a}$, var vid simuleringens början noll. Vi började med att sätta processparametern $a = 3$, och ändrade denna till $a = 1$ efter halva simuleringstiden, vilket medförde att vi hela tiden hade en instabil process. Nedan visas resultatet med STR:n, med en glömskefaktor, $\alpha = 0.05$. Övriga designparametrar var $a_f = 1$, $p = 100$, $a_{o1} = 14$, , samt $a_{o2} = 100$.



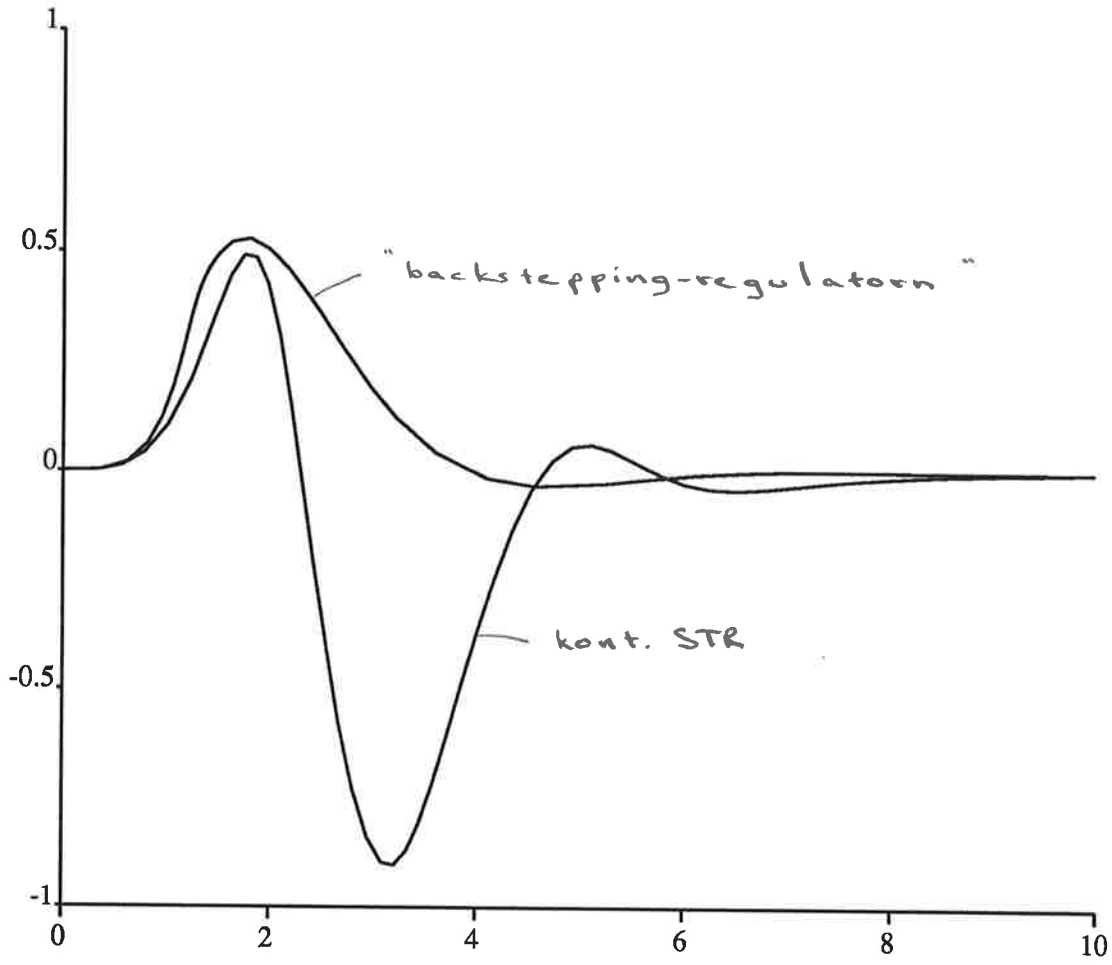
Kontinuerlig indirekt STR med $\alpha = 0.05$. Linjär process.

Man ser här att både utsignalen, y, och styrsignalen, u, varierar en hel del i början. Det tar också en stund för utsignalen att konvergera efter att a-parametern ändrat värde. Figuren på nästa sida visar nu hur simuleringen såg ut för "backstepping-regulatorn". Våra designparametrar hade här följande värden: $\gamma = 1.5$, $c_1 = c_2 = c_3 = 0.8$, $d_1 = d_2 = d_3 = 0.01$, $k_1 = 6$, $k_2 = 12$ samt $k_3 = 8$.
Man kan här se att utsignalen, y, konvergerar något snabbare, men framförallt är insvängningsförloppet betydligt mjukare. Även styrsignalen, u, klingar av betydligt fortare än för STR:n, dock skall här påpekas att styrsignalens amplitud är ganska mycket större hos "backstepping-regulatorn", vilket var genomgående vid våra olika simuleringsförsök.

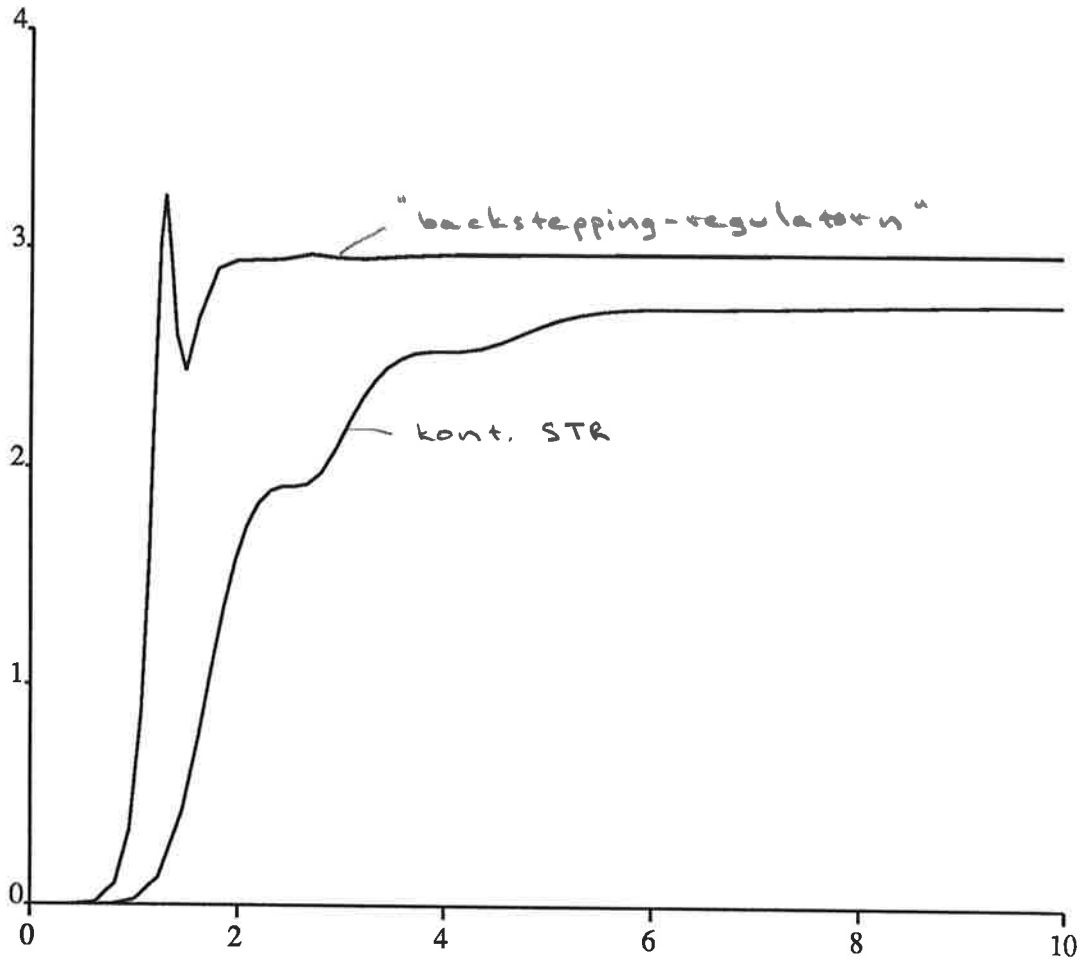


Adaptiv regulator baserad på backstepping-metoden. Linjär process.

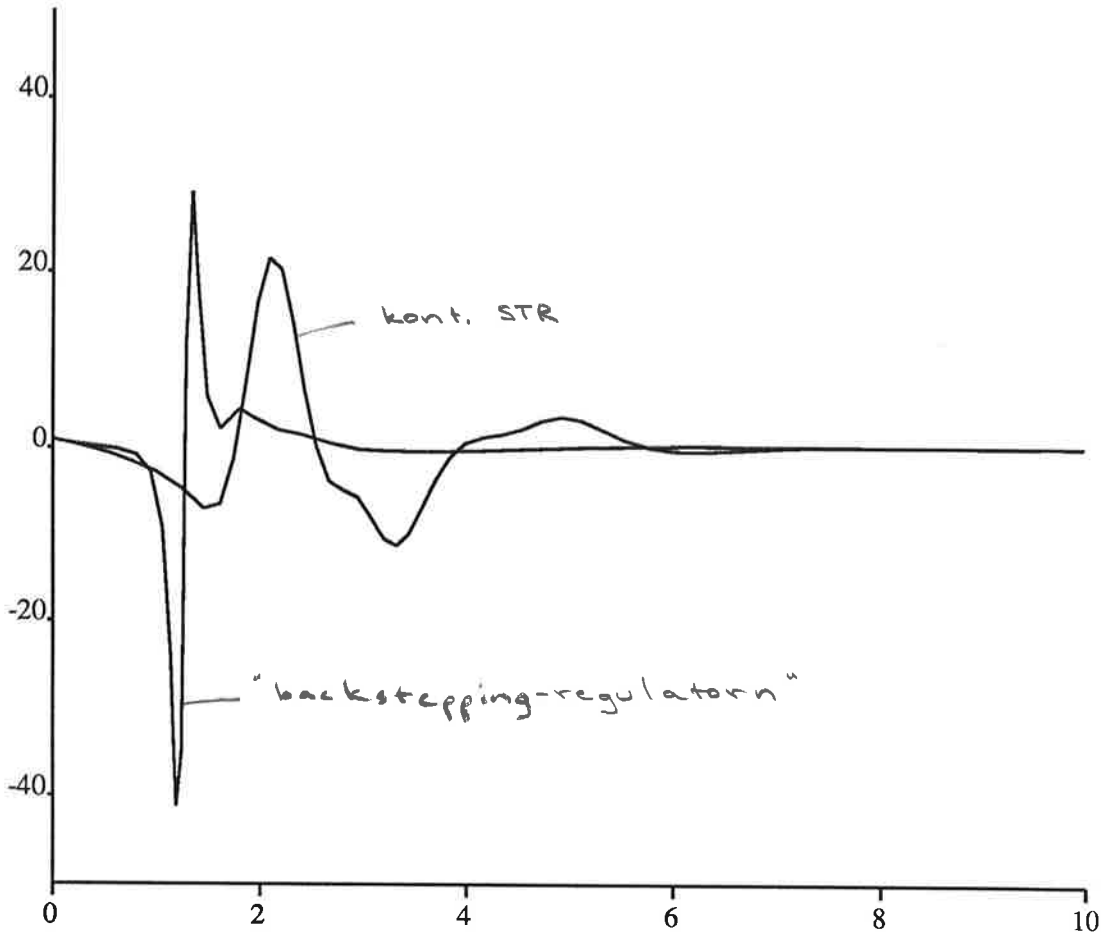
Nedanstående bilder visar mer detaljerade jämförelser mellan STR:n och "backstepping-regulatorn" under simuleringens första tio sekunder. Första bilden visar felet, $z_1 = y - y_r$, där y_r är utsignalen från referensmodellen, d.v.s. den önskade processutsignalen. Man ser här att detta fel är mindre och avtar betydligt snabbare för "backstepping-regulatorn". Nästa bild visar hur pass bra a-parametern skattas. Även här konvergerar "backstepping-regulatorn" betydligt snabbare än STR:n. Här skulle nog en bättre inställd STR kunna ge ett bättre resultat. Vad gäller "backstepping-regulatorn" medför de olinjära dämptermerna att konvergenshastigheten för parameterestimaten ej är så kritisk för utsignalföljningen. Den därpå följande figuren visar styrsignalens utseende i de båda fallen, och man kan här tydligt se den ganska stora skillnaden i amplitud.



Jämförelse av felet, $z_1 = y - y_r$.

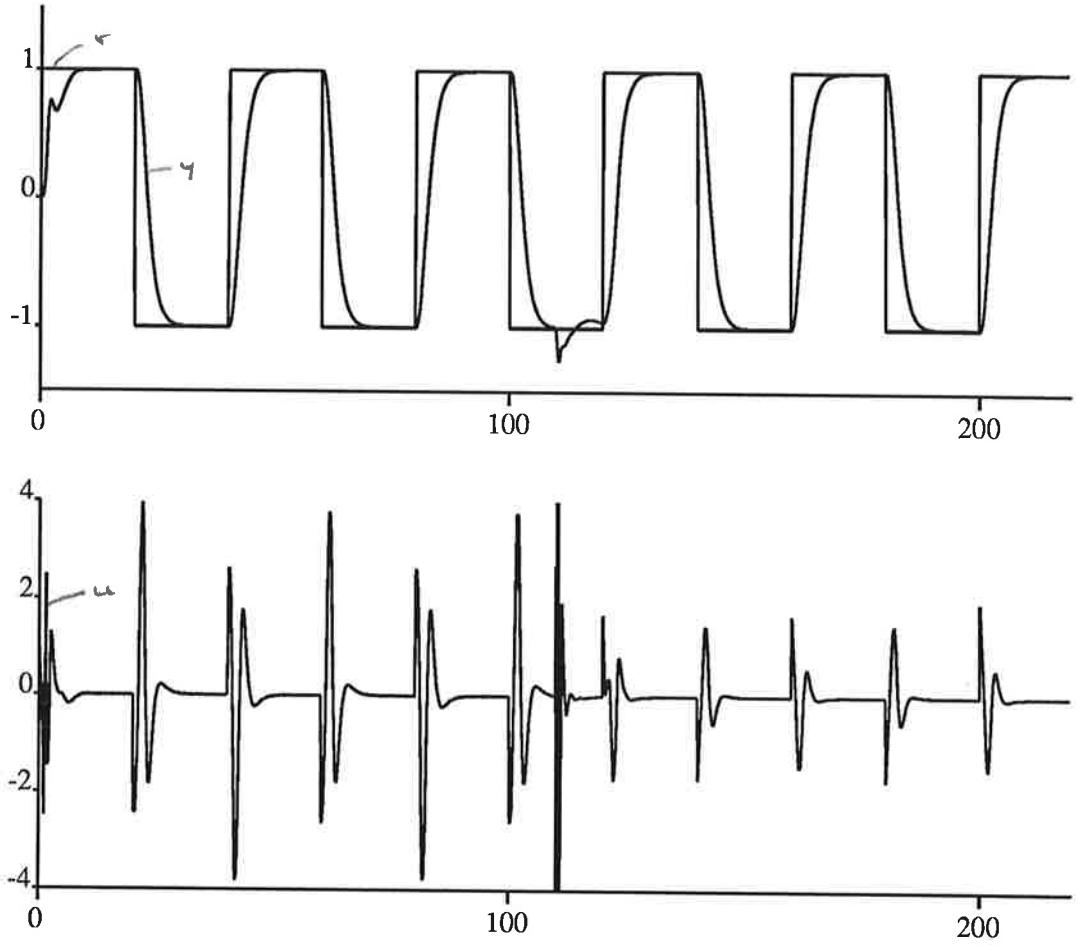


Jämförelse av konvergenshastigheten för parameterskattningen, $\hat{a}$.



Jämförelse av styrsignalen, u.

Efter simuleringen med denna linjära process ville vi även undersöka hur vår "backstepping-regulator" klarade av att reglera en olinjär process. Vi satte då den generella funktionen, $\varphi(y)$ (se ovan), lika med $\sin(y)\cos(2y)$. Då erhölls en, om än ganska "snäll", olinjär process. I nedanstående figur ser man att regulatorn klarar detta alldeles utmärkt, t.o.m. något bättre än i det linjära fallet. På samma sätt som i det linjära fallet byter vi värde på a-parametern från $a=3$ till $a=1$ efter halva tiden, och även detta fungerar bra. Styrsignalen är större i det olinjära fallet, men det är ju naturligt att så blir fallet. Våra designparametrar var desamma som i det linjära fallet d.v.s. $\gamma=1.5$, $c_1=c_2=c_3=0.8$, $d_1=d_2=d_3=0.01$, $k_1=6$, $k_2=12$ samt $k_3=8$.



Adaptiv regulator baserad på backstepping-metoden. Olinjär process.

7. Avslutning.

I artiklarna nämns att "backstepping-regulatorn" har fina transientegenskaper. Detta är något som även vi har kunnat verifiera med hjälp av våra simuleringar. Det sker dock på bekostnad av större styrsignaler. Antalet designparametrar är fler än för den kontinuerliga indirekta STR:n. Detta faktum samt att regulatorn är olinjär gör att det är svårt att finna optimala parametervärden. Artikeln ger heller ingen direkt vägledning om hur parametrarna skall ställas in. En nackdel med backstepping-metoden är att man får komplicerade och långa uttryck för regulatorn, och vi hade emellanåt svårt att få SIMNON att acceptera alla dessa uttryck. För att reglera i realtid krävs därför att beräkningarna utförs på ett effektivt sätt.

Referenser.

Petar V. Kokotovic, Miroslav Krstic, Ioannis Kanellakopoulos, "Backstepping to Passivity: Recursive Design of Adaptive Systems *" 31st Conference on Decision and Control, Tucson, Arizona, Dec. 1992

Petar V. Kokotovic, Miroslav Krstic, Ioannis Kanellakopoulos, "A New Generation of Adaptive Controllers for Linear Systems" 31st Conference on Decision and Control, Tucson, Arizona, Dec. 1992

K. J. Åström, B. Wittenmark, ADAPTIVE CONTROL, Addison Wesley, 1989

Appendix.

Process.

```
continuous system process

"Simulates a system with the transfer function
"
"
"                1
"     G(s) =  -------
"              2
"             s (s-a)
"

state x1 x2 x3
der dx1 dx2 dx3
input u
output y fi dfi d2fi
time t

dx1=a*fi+x2
dx2=x3
dx3=u

y=x1

fi=x1
dfi=1
d2fi=0

a=if t>110 then 1 else 3

end
```

Modell.

```
continuous system model

"Reference model with transfer function
"
"                1
"     Gm(s) =  -------
"                   3
"             (s+a)
"

state x1 x2 x3
der dx1 dx2 dx3
input r
output yr yrp yrpp yrppp
time t

dx1=x2
dx2=x3
dx3=-a*a*a*x1-3*a*a*x2-3*a*x3+r

yr=x1
yrp=x2
yrpp=x3
yrppp=dx3

a:1

end
```

Kontinuerlig indirekt STR.

```
continuous system cfilter

state x111 x112 x113 x121 x122 x123 x1 x2 x3
der dx111 dx112 dx113 dx121 dx122 dx123 dx1 dx2 dx3
input u y yr
output yf fi

dx111=-3*af*x111+x112-3*af*y
dx112=-3*af*af*x111+x113-3*af*af*y
dx113=-af*af*af*x111-af*af*af*y

yf1=x111+y

dx121=-3*af*x121+x122
dx122=-3*af*af*x121+x123
dx123=-af*af*af*x121+u

uf1=x121

yf=yf1-uf1

dx1=-3*af*x1+x2+y
dx2=-3*af*af*x1+x3
dx3=-af*af*af*x1

fi=x1

z1=y-yr

af:1

end
```

```
continuous system cestim

state theta p
der dtheta dp
input yf fi
output ahat

e=yf-fi*theta
dtheta=p*fi*e
dp=alpha*p-p*fi*fi*p

ahat=theta
p:100
alpha:0.05

end

continuous system cpolplac

state x11 x12 x21 x22
der dx11 dx12 dx21 dx22
input ahat r y
output u

r11=3+ao1+ahat
r12=3+3*ao1+ao2+ahat*r11
s0=3*ao1+3*ao2+1+ahat*r12
s1=3*ao2+ao1
s2=ao2

dx11=-r11*x11+x12+(r11*s0-s1)*y
dx12=-r12*x11+(r12*s0-s2)*y

yf=x11-s0*y

dx21=-r11*x21+x22+(ao1-r11)*r
dx22=-r12*x21+(ao2-r12)*r

rf=x21+r
u=yf+rf

ao1:14
ao2: 100

end
```

```
connecting system cconnect

time t

r=if mod(t,per)<per/2 then step else -step
r[cpolplac]=r
y[cpolplac]=y[process]
ahat[cpolplac]=ahat[cestim]
u[cfilter]=u[cpolplac]
y[cfilter]=y[process]
u[process]=u[cpolplac]
yf[cestim]=yf[cfilter]
fi[cestim]=fi[cfilter]
yr[cfilter]=yr[model]
r[model]=r

per:40
step:1

end
```

Adaptiv regulator baserad på backstepping-metoden.

```
continuous system vfilter

state x1 x2 x3
der dx1 dx2 dx3
input u
output v1 v2 v3

dx1=-k1*x1+x2
dx2=-k2*x1+x3
dx3=-k3*x1+u

v1=x1
v2=x2
v3=x3

k1:6
k2:12
k3:8

end
```

```
continuous system ksifilt

state x21 x22 x23 x31 x32 x33
der dx21 dx22 dx23 dx31 dx32 dx33
input y fi
output ksi21 ksi22 ksi23 ksi31 ksi32 ksi33

dx21=-k1*x21+x22+fi
dx22=-k2*x21+x23
dx23=-k3*x21

dx31=-k1*x31+x32+k1*y
dx32=-k2*x31+x33+k2*y
dx33=-k3*x31+k3*y

ksi21=x21
ksi22=x22
ksi23=x23
ksi31=x31
ksi32=x32
ksi33=x33

k1:6
k2:12
k3:8

end


continuous system estim

state a
der da
input tau3
output ahat

da=tau3

ahat=a

end
```

```
continuous system backstep

input v1 v2 v3 ksi21 ksi22 ksi23 ksi31 ksi32 ksi33
input y yr yrp yrpp yrppp fi dfi d2fi ahat
output u tau3

sort
w=ksi22+fi

z1=y-yr

tau1=gamma*w*z1

a1=-c1*z1-d1*z1-ksi32+yrp-w*ahat

da1dy=-c1-d1-dfi*ahat

da1dahat=-w

da1dyr=c1+d1

z2=v2-a1

tau2=tau1-gamma*da1dy*w*z2

a21=-c2*z2-d2*da1dy*da1dy*z2-z1+k2*v1+da1dy*(v2+ksi32)
a22=da1dyr*yrp+yrpp+k2*ksi31-ksi33-k2*y+ahat*k2*ksi21
a23=-ahat*ksi23+da1dy*w*ahat+da1dahat*tau2
a2=a21+a22+a23

da2dy1=c2*da1dy-1-k2+2*d2*ahat*da1dy*d2fi*z2
da2dy2=d2*da1dy*da1dy*da1dy-ahat*(v2+ksi32)*d2fi
da2dy3=-ahat*ahat*w*d2fi+ahat*da1dy*dfi-2*w*z1*gamma*dfi
da2dy4=-w*w*gamma-gamma*d2fi*ahat*w*w*z2
da2dy5=2*gamma*w*da1dy*dfi*z2-da1dy*da1dy*w*w*gamma
da2dy=da2dy1+da2dy2+da2dy3+da2dy4+da2dy5

da2dyr1=c2*da1dyr+1+gamma*w*w+d2*da1dy*da1dy*da1dyr
da2dyr2=-gamma*w*w*da1dy*da1dyr
da2dyr=da2dyr1+da2dyr2

da2dyrp=c2+d2*da1dy*da1dy+da1dyr-gamma*w*w*da1dy

da2dx221=-c2*ahat-d2*ahat*da1dy*da1dy+da1dy*ahat
da2dx222=-2*gamma*w*z1+2*gamma*da1dy*w*z2
```

```
da2dx223=gamma*w*w*ahat*da1dy
da2dx22=da2dx221+da2dx222+da2dx223

da2dx32=-c2-d2*da1dy*da1dy+da1dy+gamma*w*w*da1dy

da2dv1=k2

da2dv2=da2dx32

da2dah1=c2*da1dahat+2*d2*da1dy*dfi*z2-d2*da1dy*da1dy*w
da2dah2=-dfi*(v2+ksi32)+k2*ksi21-ksi23-ahat*dfi*w+w*da1dy
da2dah3=-gamma*w*w*dfi*z2+gamma*w*w*w*da1dy
da2dahat=da2dah1+da2dah2+da2dah3

z3=v3-a2

tau3=tau2-gamma*da2dy*w*z3

u1=-c3*z3-d3*da2dy*da2dy*z3-z2+k3*v1+da2dy*(v2+ksi32)
u2=da2dyr*yrp+da2dyrp*yrpp+yrppp+k3*ksi31-k3*y
u3=k2*(k1*y+ksi32-k1*ksi31)+da2dx32*(k2*y+ksi33-k2*ksi31)
u4=ahat*k3*ksi21+da2dx22*(ksi23-k2*ksi21)
u5=ahat*k2*(fi+ksi22-k1*ksi21)+da2dv1*(v2-k1*v1)
u6=da2dv2*(v3-k2*v1)+da2dy*w*ahat+da2dahat*tau3
u7=-gamma*z2*da1dahat*da2dy*w
u=u1+u2+u3+u4+u5+u6+u7

gamma:2.5
c1:1
c2:1
c3:1
d1:0.1
d2:0.1
d3:0.1
k1:6
k2:12
k3:8

end
```

```
connecting system connect

time t

r=if mod(t,per)<per/2 then step else -step
r[model]=r
u[process]=u[backstep]
u[vfilter]=u[backstep]
y[ksifilt]=y[process]
fi[ksifilt]=fi[process]
v1[backstep]=v1[vfilter]
v2[backstep]=v2[vfilter]
v3[backstep]=v3[vfilter]
ksi21[backstep]=ksi21[ksifilt]
ksi22[backstep]=ksi22[ksifilt]
ksi23[backstep]=ksi23[ksifilt]
ksi31[backstep]=ksi31[ksifilt]
ksi32[backstep]=ksi32[ksifilt]
ksi33[backstep]=ksi33[ksifilt]
y[backstep]=y[process]
yr[backstep]=yr[model]
yrp[backstep]=yrp[model]
yrpp[backstep]=yrpp[model]
yrppp[backstep]=yrppp[model]
ahat[backstep]=ahat[estim]
fi[backstep]=fi[process]
dfi[backstep]=dfi[process]
d2fi[backstep]=d2fi[process]
tau3[estim]=tau3[backstep]

per:40
step:1

end
```

Indirect STR applied to simple MIMO system

A project in Adaptive Control

Arnar Gestsson, Nordtec exch stud.
Department of Automatic Control
Lund Institute of Technology
Supervisor: Björn Wittenmark
May 13 1993

Contents

Preface

I would like to thank my supervisor Björn Wittenmark for his guidance. I would also like to thank Rolf Johansson for his help on problems in identification.

1 Abstract

The main theme of the project is to examine some of the possibilities in controlling MIMO processes. Sections 2 and 3 give a short overview over certain methods used to tread these kind of processes. The main interest here is the use of decoupling of the process. Section 4 describes simulation of MIMO process under various control situations. Thus is appears that decoupling based on recursivly estimated parameters of the process gives fairly good behavior of the closed loop response. It even shows superior advances considering no decoupling. The main difficulty approaching MIMO process with adaptive controller is the choice of estimator and the regression model parameterization.

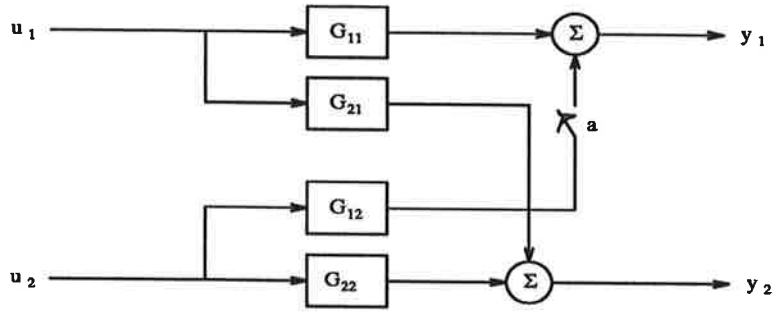


Figure 1: Block diagram of the system.

2 MIMO control of dynamic systems

In most cases when to apply a controller we are interested in finding the most suitable change in dynamics between one input and one output, i.e. we have a certain process which is SISO (single output single input) and are interested in moving the poles of the closed loop system, to minimize certain optimum criterion. In practice it can be somewhat unrealistic to treat all processes as if they were SISO, a process can have many outputs and many inputs, where excitation on one input might influence the behavior of more than one output. It is therefore of major concern to be able to decide if the coupling between different inputs and outputs is strong, i.e. pairing of inputs and outputs is therefore very important. In this rapport we will base our treatment on system with two inputs and two outputs as shown in figure 1, were G_{ij} is the transfer function between input i and output j. We will thus refer to

$$G(s) = \begin{bmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{bmatrix}$$

as a transfer function matrix (TFM). Sampled system description of the TFM will then be

$$H(q) = \begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix}$$

where define sampled subsystems as $H_{ij}(q) = ZOH(G_{ij}(s))$, and call the $H(q)$ the pulse transfer operator matrix or PTOM.

2.1 Interaction compensation

Here we shall discuss the question of how to control MIMO systems. How should we treat the coupling between different in- and outputs? We make the assumption that coupling is a disturbance acting on the system. So we assume that the compensator should minimize the effect of the coupling acting on a certain output. In the ideal case the most straight forward way to handle interaction that

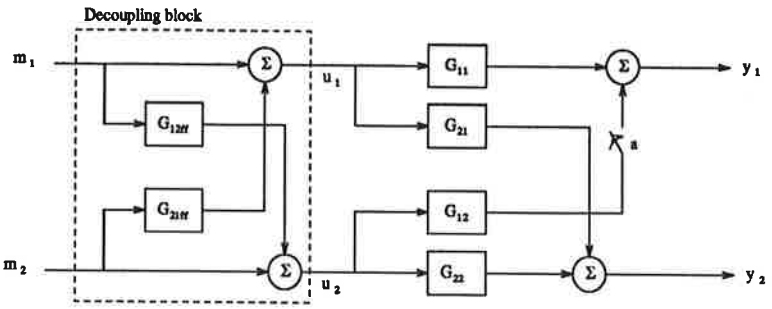


Figure 2: Block diagram of the system with feedforward interaction compensation.

are undesirable is to apply feedforward compensation, i.e. the input disturbance is measurable. As seen in figure 1 a natural way of incorporating feedforward compensation will consist of

$$G_{12ff}(s) = -G_{22}^{-1}(s)G_{21}(s) \qquad \text{and}$$

$$G_{21ff}(s) = -G_{11}^{-1}(s)G_{12}(s).$$

This can be presented as a MIMO compensator $D(s)$ with 2 inputs and 2 outputs and by introducing a new control signal $\mathbf{m}(t)$, see figure 2, where

$$G(s)D(s) = \Lambda(s) \qquad \text{and} \qquad \mathbf{m}(t) = \begin{bmatrix} m_1(t) & m_2(t) \end{bmatrix}^T$$

where $\Lambda(s)$ is a diagonal TFM. We can then make a separate regulator for each transfer function $\Lambda_{ii}(s)$ in the same manner as for SISO systems. To be able to resolve the above calculations, $G(s)$ has to be of a full rank and it also implies complete knowledge of the system. But on the other hand this gives a certain freedom in choosing the elements in either $\Lambda(s)$ or $D(s)$.

2.2 Indirect Self tuning decoupler

By using the decoupling as described above, we can use the Λ matrix to specify the closed loop response, because of

$$H(q)D(q) = \Lambda(q)$$

where the parameters of $H(q)$ are estimated by RLS and parameters of $\Lambda(q)$ are given according to certain closed loop specification. Then

$$D(q) = H^{-1}(q)\Lambda(q) = \frac{1}{detH(q)} \begin{bmatrix} H_{22}\Lambda_{11} & -H_{12}\Lambda_{22} \\ -H_{21}\Lambda_{11} & H_{11}\Lambda_{22} \end{bmatrix}$$

where $deg(detH(q)) \leq H_{jj}\Lambda_{ii}$, so $D(q)$ is realizable. This in turn restricts the $\Lambda_{ii}(q)$ which can be used and implies certain knowledge of the process other than the order of the subsystem. It is also obvious that $deg(\Lambda_{ii}) > deg(detH(q))$ will impose an extra time delay in the closed loop system. In later sections we will concentrate on the case discussed in section 2.1.
The main disadvantage here is how complicated $D(q)$ will be for systems with $n \geq 3$, $n = min(inputs, outputs)$.

3 Identification of MIMO system

When applying identification to multivariable system many problems arise. The fundamental problem is how to parameterize the regression model, which gives rise to problems relating to minimal realization. This can lead to numerical problems with the covariance matrix, such as rank defection. This problem imposes the question about the priori knowledge of the TFM or PTOM. Thus, do we just know the order of the part systems or do we know the poles of these systems as well? Here we will discuss two method for estimating the parameters of the PTOM,

- regression model based on MISO (multiple input single output) description on the process. Applying this needs as many estimators as the outputs of the process, but each of them can be the "ordinary" SISO estimator.
- regression model based on the MIMO description. This approach uses just one estimator but more complicated than the one for the SISO-cases.

3.1 MISO estimator

We refer to the PTOM of the process where

$$H_{ij} = \frac{B_{ij}}{A_{ij}}$$

where we have the MISO description as

$$y_i(k) = \sum_{j=1}^{m} \frac{B_{ij}}{A_{ij}} u_j(k)$$

with m inputs. In order to be able to find the regression model we must find the common denominator and thus the regression model looks like this

$$(\prod_{j=1}^{m} A_{ij}) y_i(k) = \sum_{j=1}^{N} (\prod_{j=1 j\neq i}^{m} A_{ij}) B_{ij} u_j(k)$$

or for the process in question the regression model is

$$A_{11}A_{12}y_1(k) = B_{11}A_{12}u_1(k) + B_{12}A_{11}u_2(k)$$

$$A_{21}A_{22}y_2(k) = B_{21}A_{22}u_1(k) + B_{22}A_{21}u_2(k).$$

But this method can lead to serious problems in the estimation. The main difficulty here is the question: "*How much do we know about the process?*". If we assume that we just know the order of the subsystems we can end up with

trying to estimate poles that does not exist in the system. This can be explained as follows, consider

$$y(k) = \frac{b_{11}}{q + a_{11}} u_1(k) + \frac{b_{12}}{q + a_{12}} u_2(k)$$

and $a_{11} = a_{12}$. Then we have regression model which assumes two poles in a_{11} or equivalently in a_{12}, but the minimal realization indicates that we just have one. Thus the model expects 2nd order dynamic in the system which just has 1st. This leads to numerical problems in the covariance matrix and the parameters converge to wrong values or no values.
In this case the RLS estimator will be implemented in the same manner as for the SISO case, with the modification that the regression vector contains the two inputs instead of one, but only one output.

3.2 MIMO estimator

If we refer again to the PTOM of the process we consider

$$H(q) = A^{-1}(q)B(q)$$

with m inputs and outputs, where

$$A(q) = q^n I + A_1 q^{n-1} + \cdots + A_n,$$

$$B(q) = B_1 q^{n-1} + \cdots + B_n$$

with $A_i, B_i \in R^{m \times m}$. This method has one characteristic problem, which is the factorization $(A(q), B(q))$ of the PTOM. For example multiplication of a tridiagonal matrix $Q(q)$ with $detQ(z) = 1$ does not change the PTOM but has different description on $A(q)$ and $B(q)$. This will in general not disturb the covariance matrix, i.e. it will not suffer indefinity.
The RLS algorythm will have to be modified more extensively because now we have the regression model

$$\mathbf{y}_k = \phi_k^T \theta_{k-1}$$

where ϕ_k is array instead of vector but the θ_k is a vector thus

$$\phi_k = \begin{bmatrix} \phi_1 \\ \phi_2 \end{bmatrix}_k^T = \begin{bmatrix} -y_1 & -y_2 & u_1 & u_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -y_1 & -y_2 & u_1 & u_2 \end{bmatrix}_{k-1}^T$$

and

$$\theta_k = \begin{bmatrix} a_{11} & a_{12} & b_{11} & b_{12} & a_{21} & a_{22} & b_{21} & b_{22} \end{bmatrix}^T$$

for the 2×2 model with first order subsystems. To update the P-matrix we have to apply the *matrix inversion lemma* because of $\phi_k^T P_{k-1} \phi_k$ not being scalar. Thus the covaiance update would look like

$$P_k = P_{k-1} - [I + \phi_k^T P_{k-1} \phi_k]^{-1} (P_{k-1} \phi_k \phi_k^T P_{k-1}).$$

In return we expect not to get any difficulties relating the realization of the process model and invertibility of the covariance matrix, i.e. this requires just knowledge about the order of the subsystem. But as mentioned before the parameters of the A and B matrices are not unique, but we are in fact not interested in them just the PTOM.

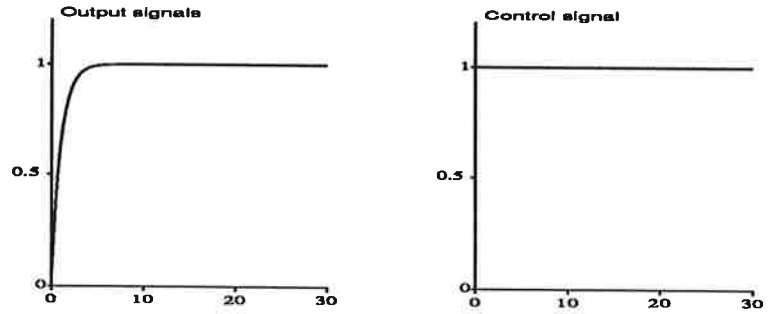


Figure 3: Step put on u_1 and u_2=0 with a=1, the figure indicates that both outputs have same response from u_1.

4 Adaptive controller for simple MIMO system

4.1 The process

The process that will be used here has 2 inputs and 2 outputs, with strong coupling from each input to both outputs. The transfer function matrix is given as

$$G(s) = \begin{bmatrix} \frac{1}{s+1} & \frac{2a}{s+3} \\ \frac{1}{s+1} & \frac{1}{s+1} \end{bmatrix}$$

where a can be used to make the transfer function matrix (TFM) tridiagonal. As can been seen on the TFM, change in a does not affect the coupling between u_1 and y_2, where the input and the output vectors are defined as

$$\mathbf{u}(t) = \begin{bmatrix} u_1(t) & u_2(t) \end{bmatrix}^T$$

$$\mathbf{y}(t) = \begin{bmatrix} y_1(t) & y_2(t) \end{bmatrix}^T.$$

Below can we see step response of the system, due to step to either of the inputs. As can be expected figure 3 shows no difference between y_1 and y_2 when step is applied to u_1, but on the other hand, as can by seen in figure 4 and in figure 5, much difference is in the step response from u_2 to the both outputs.

A pulse transfer operator matrix (POTM) of the process is given by

$$H(q) = \begin{bmatrix} \frac{0.3935}{q-0.6065} & \frac{0.5179a}{q-0.2231} \\ \frac{0.3935}{q-0.6065} & \frac{0.3935}{q-0.6065} \end{bmatrix}$$

where h=0.5 was chosen.

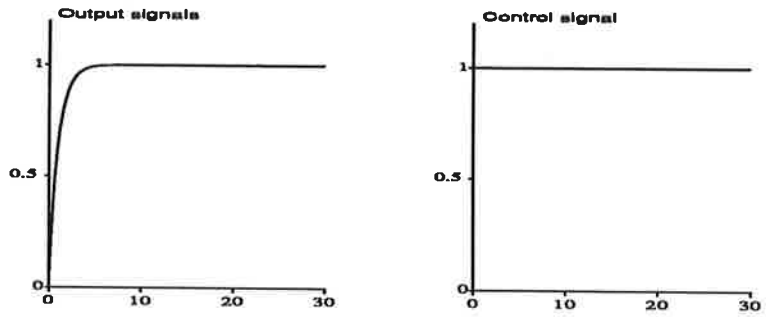


Figure 4: Step put on u_2 and u_1=0, with a=0.

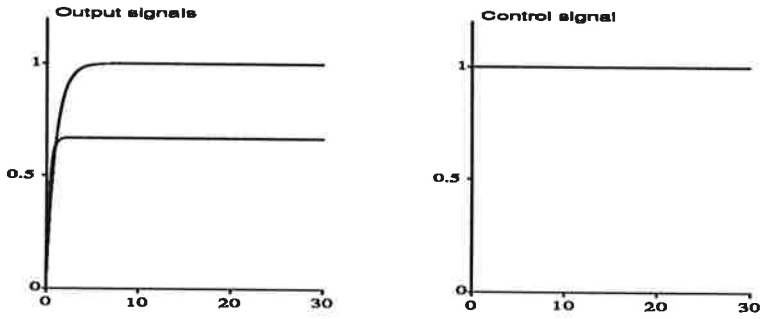


Figure 5: Step put on u_2 and u_1=0, with a=1, where y_1 is the lower and y_2 is the upper curve.

4.2 The estimator

The estimator that will be used on the system is MISO as described in section 3.1 where

$$y_1(k) = \frac{b_{11}}{q + a_{11}} u_1(k) + \frac{b_{12}}{q + a_{12}} u_2(k)$$

$$y_2(k) = \frac{b_{21}}{q + a_{21}} u_1(k) + \frac{b_{22}}{q + a_{22}} u_2(k)$$

thus finding the common denominator gives the following regression model

$$y_1(k) = \phi_1^T(k)\theta_1(k-1)$$

$$y_2(k) = \phi_2^T(k)\theta_2(k-1)$$

with

$$\phi_1(k) = \begin{bmatrix} -y_1(k-1) & -y_1(k-2) & u_1(k-1) & u_1(k-2) & u_2(k-1) & u_2(k-2) \end{bmatrix}^T$$

$$\theta_1(k) = \begin{bmatrix} (a_{11} + a_{12}) & a_{11}a_{12} & b_{11} & b_{11}a_{12} & b_{12} & b_{12}a_{11} \end{bmatrix}^T$$

and the same for the other output. When this regression model was estimated with the RLS estimator, see A.2, there were all kinds of difficulties that appeared. It even showed to depend on the sampling interval. This will be blamed on the implementation of the estimator and indicates how the condition of the covariance matrix is dependent on good numerical properties of the estimator. The simulations shows that, when using the sampling intervals $h = 0.5$, the estimator worked as expected. Even though it is worrying that it had bad numerical properties for other sampling intervals. Further discussions on this point will though be left out, because of it being somewhat outside the scope of this project.

4.3 Applying STR without decoupler

The regulator is designed for each of $H_{11}(q)$ and $H_{22}(q)$ in the PTOM, as

$$R(q)u(k) = T(q)u_c(k) - S(q)y(k)$$

with integration in R(q) and $B^+(q) = 1$ the design equation becomes

$$AR_1 + b_1 S = A_o A_m$$

which has the minimal solution with

$$R(q) = R_1'(q-1)$$

$$deg A_o = 1$$

$$degR_1' = 0$$
$$degS = 1$$

and gives

$$(q + a_1)(q - 1) + b_1(s_0 q + s_1) = (q + a_o)(q + a_m)$$

Then the controller parameters are given by

$$s_0 = \frac{a_o + a_m + 1 - a_1}{b_1}$$
$$s_1 = \frac{a_o a_m + a_1}{b_1}$$
$$b_m = \frac{1 + a_m}{b_1}$$

and $T = b_m A_o$. Figure 6 shows simulation on the system with following parameters

$$a_o = -0.3$$
$$a_m = -0.5.$$

Notice that no feedback is from y_2 and the no input acts on u_2. As could be expected this situation gives quite good response. But if we on the other hand try to use single feedback loop for each of the diagonal elements of $H(q)$ for same observer and model parameter the results can be seen in figure 7. This figure shows very well how severe the coupling is and that the SISO designed regulators do not cope with the situation. The spikes in the control signal depends on the fact that the STR tries to make 2nd order system follow first order model.

4.4 Applying decoupler to the system

If we implement the decoupler discussed in section 2, and chose the elements in $D(q)$ as follows

$$D(q) = \begin{bmatrix} 1 & \frac{-H_{12}}{H_{11}} \\ -\frac{b_{21}}{b_{22}} & 1 \end{bmatrix}$$

which in turn gives

$$\Lambda(q) = \begin{bmatrix} H_{11} - H_{12} & 0 \\ 0 & H_{22} - H_{12} \end{bmatrix}$$

where the part systems have different dynamic compared to initial part systems. This is seen in the figure 8, and the spikes indicates that the closed loop system has unstable zeros. Thus a better choice of $D(q)$ or equivalently $\Lambda(q)$ could possibly xdvi improve the system. As seen on the $\Lambda(q)$ new design has to be made if applying STR for the decoupled system. It is also noticable that the decoupling is not perfect for Λ_{11} which had a dynamic feedforward compensation on the coupling from u_2, but for Λ_{22} it shows no signs of coupling from u_1. This part of the decoupler is just gain adjustment.

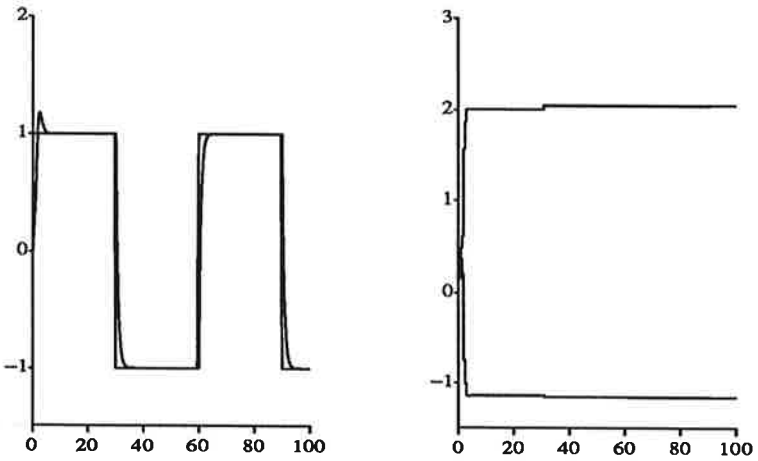


Figure 6: Input u_{c1} and output of $H_{11}(q)$ with STR regulator(left) with no feedback or input to $H_{22}(q)$. Parameter convergens of s_0 and s_1 is to the right.

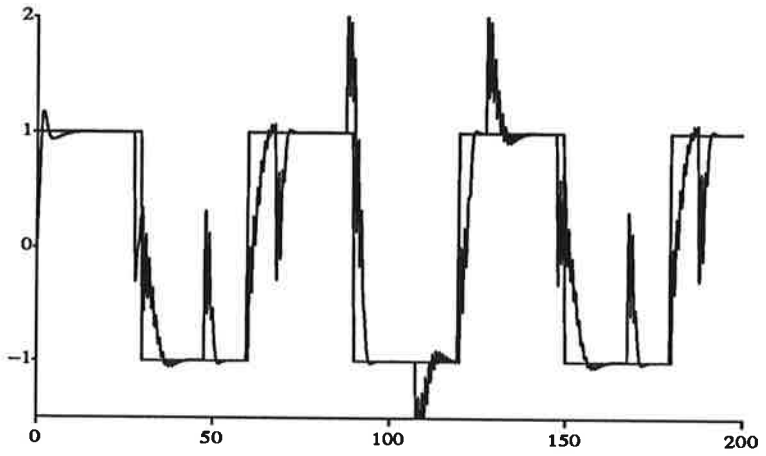


Figure 7: Input u_{c1} and output of $H_{11}(q)$ with two separate STR regulators regulating H_{11} and $H_{22}(q)$.

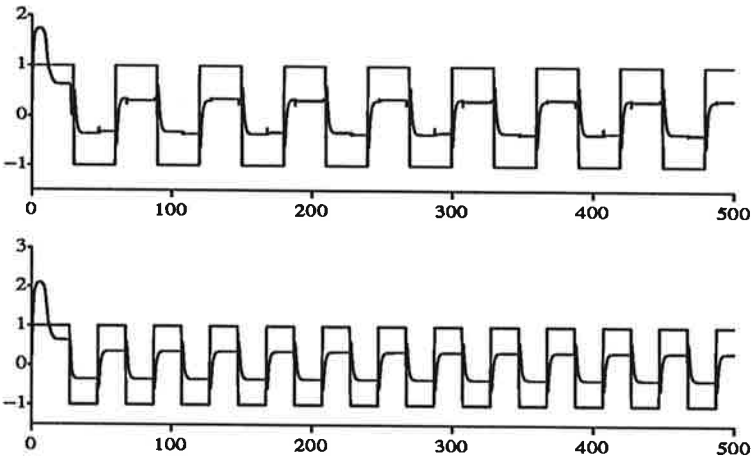


Figure 8: Input and output of the system with decoupling, u_1 and y_1 to the upper, u_2 and y_2 the lower. This is open loop system response.

4.5 Indirect STR for decoupled system

Here we will design a indirect STR for the decoupled system. As before we will have integrator in the regulator and design a separate regulator for each element of

$$\Lambda(q) = \begin{bmatrix} H_{11} - H_{12} & 0 \\ 0 & H_{22} - H_{12} \end{bmatrix} =$$

$$= \begin{bmatrix} \frac{(b_{11}-b_{12})q+a_{12}b_{11}-a_{11}b_{12}}{q^2+(a_{12}+a_{11})q+a_{11}a_{12}} & 0 \\ 0 & \frac{(b_{22}-b_{12})q+a_{12}b_{22}-a_{22}b_{12}}{q2+(a_{12}+a_{22})q+a_{22}a_{12}} \end{bmatrix}.$$

Thus we chose not to cancel the zero of the process, i.e. the closed loop system will contain the zero of the process. This gives the following design equations[1], thus

$$B^- = q + \frac{a_{12}b_{11} - a_{11}b_{12}}{b_{22} - b_{12}}$$

$$deg A_o = 2$$

$$deg R_1' = 1$$

$$deg S = 2.$$

We then solve the DAB

$$A(q)(q-1)(q+r_1) + B^-(s_0q^2 + s_1q + s_2) = A_oA_m.$$

[1] we will just consider the element $\Lambda_{11}(q)$ because the design is the same for $\Lambda_{22}(q)$ except for the indexes in the pulse transfer operator

By putting $q = -(a_{12}b_{11} - a_{11}b_{12})/(b_{22} - b_{12})$ we can solve for r_1 and get

$$r_1 = \frac{A_o(-\frac{\beta_{12}}{\beta_{11}})A_m(-\frac{\beta_{12}}{\beta_{11}})}{A(-\frac{\beta_{12}}{\beta_{11}})(-\frac{\beta_{12}}{\beta_{11}} - 1)} + \frac{\beta_{12}}{\beta_{11}}$$

where

$$\beta_{11} = b_{22} - b_{12}$$

$$\beta_{12} = a_{12}b_{11} - a_{11}b_{12}.$$

If we then put $q = 0$ we get

$$s_2 = \frac{a_{o2}a_{m2} + a_2 r_1}{\beta_{12}}$$

then we solve at last for s_0 and s_1 and that gives

$$s_0 = \frac{a_{o1} + a_{m1} + 1 - a_1 - r_1}{\beta_{11}}$$

$$s_1 = \frac{a_{o2} + a_{m2} + a_{o1}a_{m1} - a_2 + r_1 + (1 - r_1)a_1 - s_0\beta_{12}}{\beta_{11}}$$

and

$$T(q) = A_o b_m,$$

$$b_m = \frac{A_m(1)}{B(1)}.$$

Simulation on the system is shown in figure 9, thus the regulator seem to work fine but the spikes due to the choice of $D(q)$ are still there. Thus if we put in to the pulse transfer operator the values obtained in 4.1 we get $B(q) = -0.1244q + 0.2263$ or zero at $z = -1.8188$. This emphasizes the fact that the choice of $D(q)$ is important.

4.6 Conclusions

As can be seen from figure 5 and from the process description there is a severe cross-coupling between the inputs and outputs of the system. Using SISO regulator on H_{11} and H_{22} shows not acceptable behavior of the closed loop system. On the other hand when applying the decoupler to the system and then using SISO regulator to each of the remaining subsystems of the process, it gives quite good response. But as implemented here the resulting system from the decoupling is not minimum phase. It would be interesting to look further into the choice made on the $D(q)$ or the $T(q)$ in order to get better closed loop response. Another interesting thing that came up was the difficulties in the estimation. This point could also form another project. Thus look further into the numerical properties of the certain implementations of various estimators, thought maybe

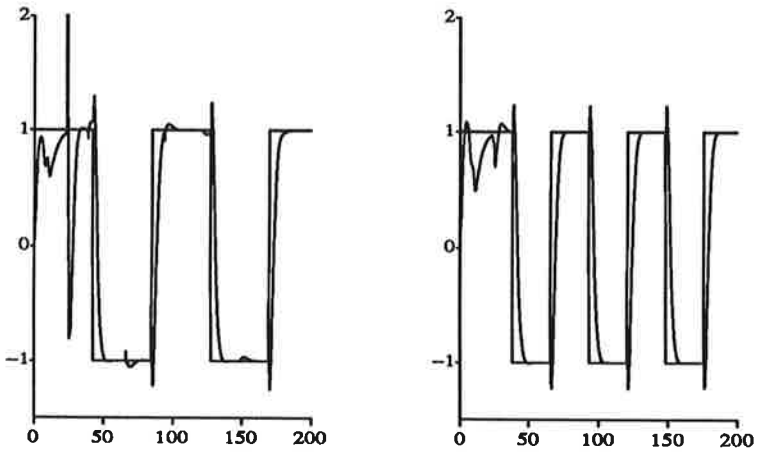


Figure 9: Reference and output of the system with decoupler and STR regulator, Λ_{11}(left) and Λ_{22}(right).

of most interest those based on RLS.

Arnar Gestsson
May 12, 1993

A Simnon programs

A.1 Process model

```
CONTINUOUS SYSTEM kerfi
INPUT u1 u2
OUTPUT y1 y2
STATE x1 x2 x3 x4
DER dx1 dx2 dx3 dx4

dx1=-x1 + u1          " G(s)=1/(s+1)
dx2=-3*x2 + 2*a*u2    " G(s)=2a/(s+3)
dx3=-x3 + u2          " G(s)=1/(s+1)
dx4=-x4 + u1

y1=x1+x2
y2=x4+x3

a : 1

END
```

A.2 The estimator

```
discrete system mimoest

"Estimates parameters of the transfer function matrix
"       b11/(q+a11) b12/(q+a12)
"H(q)=
"       b21/(q+a21) b22/(q+a22)


"The regression vector is defined  as:
" fi(k)=[-y1(k-1) -y1(k-2) u1(k-1) u1(k-2) u2(k-1) u2(k-2)]^T
" fii(k)=[-y2(k-1) -y2(k-2)]^T
" and the parameter matrix
" th=[[a11+a12 a11a12 b11  b11a12 b12 b12a11]
"      [  0       0     a21+a22 a22a21 b21  b21a22 b22 b22a21]]^T
"denoted as
"th=[[t11 t12 t13 t14 t15 t16]
"    [t21 t22 t23 t24 t25 t26]]
"where we have used the regression model
" y(k)=th^T*fi(k)

state fi1 fi2 fi3 fi4 fi5 fi6 fii1 fii2
state p11 p12 p13 p14 p15 p16 p22 p23 p24 p25 p26
state p33 p34 p35 p36 p44 p45 p46 p55 p56 p66
state c11 c12 c13 c14 c15 c16 c22 c23 c24 c25 c26
state c33 c34 c35 c36 c44 c45 c46 c55 c56 c66
state t11 t12 t13 t14 t15 t16
state t21 t22 t23 t24 t25 t26
new nfi1 nfi2 nfi3 nfi4 nfi5 nfi6 nfii1 nfii2
new np11 np12 np13 np14 np15 np16 np22 np23 np24 np25 np26
new np33 np34 np35 np36 np44 np45 np46 np55 np56 np66
```

```
new nc11 nc12 nc13 nc14 nc15 nc16 nc22 nc23 nc24 nc25 nc26
new nc33 nc34 nc35 nc36 nc44 nc45 nc46 nc55 nc56 nc66
new nt11 nt12 nt13 nt14 nt15 nt16
new nt21 nt22 nt23 nt24 nt25 nt26
input y1 y2 u1 u2
time t
tsamp ts

"Compute P*fi
"
pfi1=(p11*fi1+p12*fi2+p13*fi3+p14*fi4+p15*fi5+p16*fi6)
pfi2=(p12*fi1+p22*fi2+p23*fi3+p24*fi4+p25*fi5+p26*fi6)
pfi3=(p13*fi1+p23*fi2+p33*fi3+p34*fi4+p35*fi5+p36*fi6)
pfi4=(p14*fi1+p24*fi2+p34*fi3+p44*fi4+p45*fi5+p46*fi6)
pfi5=(p15*fi1+p25*fi2+p35*fi3+p45*fi4+p55*fi5+p56*fi6)
pfi6=(p16*fi1+p26*fi2+p36*fi3+p46*fi4+p56*fi5+p66*fi6)

"Compute C*fi
"
cfi1=(c11*fii1+c12*fii2+c13*fi3+c14*fi4+c15*fi5+c16*fi6)
cfi2=(c12*fii1+c22*fii2+c23*fi3+c24*fi4+c25*fi5+c26*fi6)
cfi3=(c13*fii1+c23*fii2+c33*fi3+c34*fi4+c35*fi5+c36*fi6)
cfi4=(c14*fii1+c24*fii2+c34*fi3+c44*fi4+c45*fi5+c46*fi6)
cfi5=(c15*fii1+c25*fii2+c35*fi3+c45*fi4+c55*fi5+c56*fi6)
cfi6=(c16*fii1+c26*fii2+c36*fi3+c46*fi4+c56*fi5+c66*fi6)

den1=lam1+(fi1*pfi1+fi2*pfi2+fi3*pfi3+fi4*pfi4+fi5*pfi5+fi6*pfi6)

den2=lam2+
    (fii1*cfi1+fii2*cfi2+fi3*cfi3+fi4*cfi4+fi5*cfi5+fi6*cfi6)

eps1=y1-fi1*t11-fi2*t12-fi3*t13-fi4*t14-fi5*t15-fi6*t16

eps2=y2-fii1*t21-fii2*t22-fi3*t23-fi4*t24-fi5*t25-fi6*t26

"K for y1(k)
g1=pfi1/den1
g2=pfi2/den1
g3=pfi3/den1
g4=pfi4/den1
g5=pfi5/den1
g6=pfi6/den1

"K for y2(k)
k1=cfi1/den2
k2=cfi2/den2
k3=cfi3/den2
k4=cfi4/den2
k5=cfi5/den2
k6=cfi6/den2

"Update the th matrix
nt11=t11+g1*eps1
nt12=t12+g2*eps1
nt13=t13+g3*eps1
```

```
nt14=t14+g4*eps1
nt15=t15+g5*eps1
nt16=t16+g6*eps1

nt21=t21+k1*eps2
nt22=t22+k2*eps2
nt23=t23+k3*eps2
nt24=t24+k4*eps2
nt25=t25+k5*eps2
nt26=t26+k6*eps2

"Update the P matrix, the covariance matrix for th1
np11=(p11-pfi1*g1)/lam1
np12=(p12-pfi1*g2)/lam1
np13=(p13-pfi1*g3)/lam1
np14=(p14-pfi1*g4)/lam1
np15=(p15-pfi1*g5)/lam1
np16=(p16-pfi1*g6)/lam1
np22=(p22-pfi2*g2)/lam1
np23=(p23-pfi2*g3)/lam1
np24=(p24-pfi2*g4)/lam1
np25=(p25-pfi2*g5)/lam1
np26=(p26-pfi2*g6)/lam1
np33=(p33-pfi3*g3)/lam1
np34=(p34-pfi3*g4)/lam1
np35=(p35-pfi3*g5)/lam1
np36=(p36-pfi3*g6)/lam1
np44=(p44-pfi4*g4)/lam1
np45=(p45-pfi4*g5)/lam1
np46=(p46-pfi4*g6)/lam1
np55=(p55-pfi5*g5)/lam1
np56=(p56-pfi5*g6)/lam1
np66=(p66-pfi6*g6)/lam1

"Update the C matrix, the covariance matrix for th2
nc11=(c11-cfi1*k1)/lam2
nc12=(c12-cfi1*k2)/lam2
nc13=(c13-cfi1*k3)/lam2
nc14=(c14-cfi1*k4)/lam2
nc15=(c15-cfi1*k5)/lam2
nc16=(c16-cfi1*k6)/lam2
nc22=(c22-cfi2*k2)/lam2
nc23=(c23-cfi2*k3)/lam2
nc24=(c24-cfi2*k4)/lam2
nc25=(c25-cfi2*k5)/lam2
nc26=(c26-cfi2*k6)/lam2
nc33=(c33-cfi3*k3)/lam2
nc34=(c34-cfi3*k4)/lam2
nc35=(c35-cfi3*k5)/lam2
nc36=(c36-cfi3*k6)/lam2
nc44=(c44-cfi4*k4)/lam2
nc45=(c45-cfi4*k5)/lam2
nc46=(c46-cfi4*k6)/lam2
nc55=(c55-cfi5*k5)/lam2
nc56=(c56-cfi5*k6)/lam2
```

```
nc66=(c66-cfi6*k6)/lam2

"Update the fi vector
nfi1=-y1
nfi2=fi1
nfi3=u1
nfi4=fi3
nfi5=u2
nfi6=fi5
nfii1=-y2
nfii2=fii1

ts=t+h  "Update the next sample time

"Internal parameters

lam1:1   "Exponential forgetting factor
lam2:1   "Exponential forgetting factor
h:0.2      "Sampling period

end
```

A.3 STR for first order system

```
discrete system adreg

"Adaptive controller to first order system
"The controller is based on The Indirect STR-method

state uck1 yk1 us
new nuck1 nyk1 nus
input y uc alfa1 alfa2 beta1
output u
time t
tsamp ts

b1=beta1
bm=(1+am)/b1
t0=bm
t1=ao*bm
e1=alfa1*alfa1/4-alfa2
e2=if e1 < 0 then 0 else e1
a1=alfa1/2-sqrt(e2)
s0=(ao+am+1-a1)/b1
s1=(am*ao+a1)/b1

v=t0*uc+t1*uck1-s0*y-s1*yk1+us

u=if v < -utlim then -utlim else (if v < utlim then v else utlim)

nyk1=y
nuck1=uc
nus=u
```

```
ts=t+h

h:1     "Sampling time
am:-0.5   "Discrete pole of the model
ao:-0.3   "observer pole
utlim:5   "Limit on control signal
end
```

A.4 Decoupling block

```
discrete system decouple

"Decouples the interaction in kerfi
"uses estimated values from RLS estimator

state m1k1 u1k1 u2k1 m2k1
new nm1k1 nu1k1 nu2k1 nm2k1
input m1 m2 alfa11 alfa12 beta11
input beta12 alfa21 alfa22 beta21 beta22
output u1 u2
time t
tsamp ts


b11=beta11
b12=beta12
e1=alfa11*alfa11/4-alfa12    "checking for valid values
e2=if e1 < 0 then 0 else e1
a11=alfa11/2-sqrt(e2)
a12=alfa11/2+sqrt(e2)
b21=beta21
b22=beta22
e3=alfa21*alfa21/4-alfa22
e4=if e3 < 0 then 0 else e3
a21=alfa21/2-sqrt(e4)
a22=alfa21/2+sqrt(e4)

uff1=-b12/b11*(m2+a11*m2k1)-a12*u1k1
u1=m1+a*uff1

uff2=-b21/b22*m1
u2=m2+b*uff2

"Update state
nu1k1=uff1
nm1k1=m1
nu2k1=uff2
nm2k1=m2

ts=t+h

h:1      "Sampling time
a:1      "Switch for decoupler
b:1      "Switch for decoupler
```

```
end
```

A.5 STR controller for the decoupled system

```
DISCRETE SYSTEM styr

STATE yk1 yk2 uk1 uk2 uck1 uck2
NEW nyk1 nyk2 nuk1 nuk2 nuck1 nuck2
INPUT y uc beta1 beta2 a1 a2
OUTPUT u
TIME t
TSAMP ts

INITIAL
om=om0*sqrt(1-zeta*zeta)
omob=omo*sqrt(1-zetao*zetao)
alfa=exp(-zeta*om0*h)
alfao=exp(-zetao*omo*h)
beta=cos(om*h)
betao=cos(omo*h)
am1=-2*alfa*beta
am2=alfa*alfa
ao1=-2*alfao*betao
ao2=alfao*alfao
amsum=1+am1+am2

SORT
b1=beta1-beta2
b2=a2*beta1-a1*beta2
bm=amsum/(b1+b2)
bp=b2/b1
tp0=bm
tp1=bm*ao1
tp2=bm*ao2
rneft=(1+bp)*(bp*bp-a1*bp+a2)
"rnefn=IF rneft<1 THEN 1 ELSE rneft
r1=bp-((bp*bp-ao1*bp+ao2)*(bp*bp-am1*bp+am2))/rneft
rp1=r1-1
rp2=-r1
s2=(ao2*am2+a2*r1)/b2
s0=(ao1+am1+1-a1-r1)/b1
s1=(am2+ao2+am1*ao1+r1-a2+(1-r1)*a1-s0*b2)/b1

ut=tp0*uc+tp1*uck1+tp2*uck2-s0*y-s1*yk1-s2*yk2-rp1*uk1-rp2*uk2

u=IF ut< -utlim THEN -utlim ELSE (IF ut<utlim THEN ut ELSE utlim)

nyk2=yk1
nyk1=y
nuk2=uk1
nuk1=u
nuck2=uck1
nuck1=uc
```

```
ts=t+h

h:0.2
omo:3           "Observer natural frequency
om0:1.5         "Model's natural frequency
zeta:0.7        "damping of the model
zetao:0.7        "damping of the observer
utlim:6
END
```

References

[1] Åström, K. J. and Wittenmark, B. *Adaptive Control.* Addison Wesley 1989.

[2] Åström, K. J. and Wittenmark, B. *Computer controlled system, 2nd ed.* Prentice-Hall 1990.

[3] Johansson, R. *System Identification.* Kompendium KF-Lund 1992.

[4] Wittenmark, B., Åström, K. J. and Jörgensen, S. B. *Process Control.* Kompendium KF-Lund 1991.

Implementering av adaptiv regulator i DSP med LabView och Think C

Martin Hägerdal, E88
John Erik Persson, E89
Jean-Marc Pfeiffer, E89
Jörgen Rosengren, E89

Handledare: Anders Carlsson

Sammanfattning

Vi har reglerat en blackboxprocess med en adaptiv regulator i Macintoshmiljö m.h.a. den digitala signalprocessorn DSP TMS 320 C, C++ och LabView. Tack vare denna uppställning har vi haft möjlighet att arbeta med en samplingsfrekvens på upp till 2 kHz för både reglering och adaptering. Vi har kunnat konstatera att LabView förkortar utvecklinstiden för realtidstillämpningar avsevärt.

1. Förord

Vi har under två veckors tid genomfört ett kombinerat projekt i Adaptiv reglering och Realtidssystem. Projektet gick ut på att med en mycket snabb signalprocessor (Texas Instruments TMS 320C30) och med det grafiska programmet LabView implementera ett realtidssystem med en adaptiv regulator. Vid projektets början hade vi inga krav på vilken process som skulle användas resp. efter vilken metod den adaptiva regulatorn skulle arbeta.

Vi vill ge vår handledare Anders Carlsson ett stort tack. Han har gett oss många värdefulla råd och tips under projektets gång och dessutom tillhandahållit underprogrammen i LabView, Signalgenerator, DSPConsole, Read shared FP data och Write shared FP data.

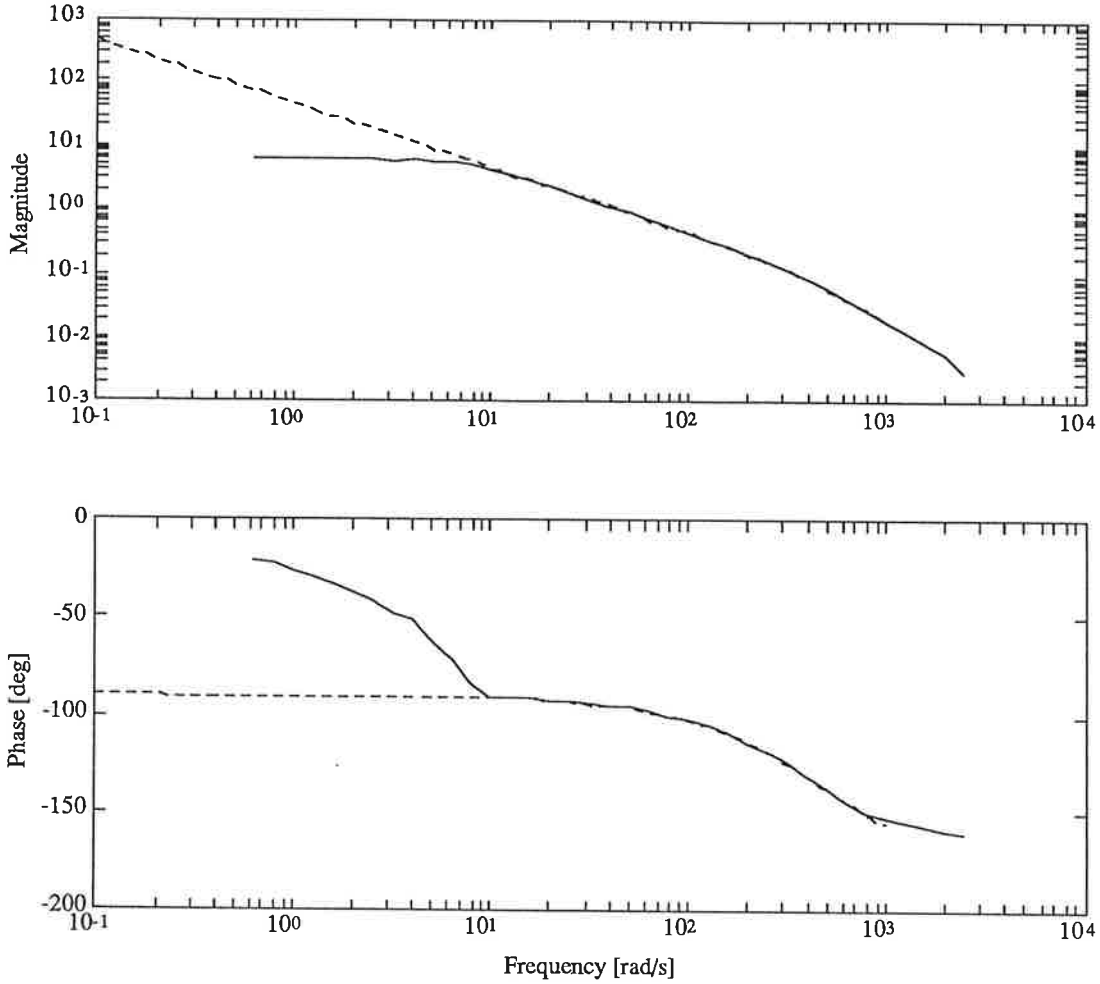
Tack dessutom till Björn Wittenmark och Ulf Jönsson, som hjälpte oss runt en del reglertekniska blindskär.

2. Metod

2.1. Identifiering av processen

För att göra oss en bild av hur vår process såg ut började vi med att göra en identifiering med hjälp av en frekvensanalysator (av märket Solartron). Med frekvensanalysatorn samlade vi in mätdata på amplitud och fas inom frekvensintervallet 0.1 Hz till 1000 Hz. Till det uppmätta frekvenssvaret anpassade vi en andra ordningens modell som med god noggrannhet beskrev systemets uppförande kring skärfrekvensen. När vi senare undersökte systemets uppförande i *tidsplanet* visade det sig att det innehöll en integrator—stegsvaret var en ramp. Man kan alltså misstänka att frekvensanalysen inte blev speciellt tillförlitlig för låga frekvenser eftersom integratortillståndet "driver iväg" under estimeringen. Trots det ansåg vi oss kunna utläsa att systemet var av andra ordningen, med en långsam pol (nämligen integratorn) och en mycket snabb i området kring 400 Hz.

Som framgår av figur 1.1 var detta en riktig slutsats; det "verkliga" frekvenssvaret (som tagits fram analytiskt, se Appendix A) ansluter mycket väl till det uppmätta utom vid de låga frekvenserna.



Figur 1.1: Bodediagram över det med frekvensanalys upptagna frekvenssvaret och det analytiskt uträknade. Det senare representeras av de streckade kurvorna.

2.2. Regulatorn

2.2.1. Allmänt

Regulatorn implementerades i programspråket C på grund av att LabView är ett relativt långsamt program och att C-program går att ladda ner direkt i den snabba signalprocessorn (DSP:n). Detta möjliggör *mycket* snabb sampling (1000 - 2000 Hz). Vi har provat att sampla med 2 kHz med gott resultat, men gick sedan över till 1 kHz pga bättre numeriska egenskaper. Observera att inte endast reglering, utan även hela adapteringsprocessen utförs i denna hastighet.

Inledningsvis ville vi implementera en *direkt* adaptiv regulator, dvs en där man utifrån processens ut- och insignal estimerar de önskade regulatorparametrarna. Vi designade en sådan och simulerade den i Simnon (se Appendix B), där det fungerade bra, förutom att den var väldigt bruskänslig. När vi skulle implementera den i C gick det emellertid inte lika bra (troligen pga att vi specificerade för låg önskad bandbredd, se nedan) varför vi gick över till en enklare struktur: den indirekta regulatorn, där estimeringen kan ske oberoende av regleringen.

Vi har implementerat två regulatorer, en P-regulator och en indirekt RST-regulator. P-regulatorn användes vid uppstart och kan även användas om man bara vill undersöka estimatorns egenskaper.

2.2.2. Estimering

Eftersom vi bestämt oss för en indirekt regulator kunde estimeringen utföras oberoende av regleringen. Vi använde oss av enkel RLS-algoritm med glömskefaktor:

$$\theta = (a_1 \quad a_2 \quad b_0 \quad b_1)^T$$

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

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

Estimatet uppdaterades enligt

$$\theta(t) = \theta(t-1) + K(t)(y(t) - \varphi^T(t)\theta(t-1))$$

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

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

Vi använde inga "skyddsnät" för att förhindra att P-matrisen exploderade e.dyl. Trots det fungerade estimeringen relativt bra, även om man var tvungen att justera glömskefaktorn beroende på om processen exciterades mycket eller lite.

Det stora problemet vid estimeringen var istället att polerna i processen låg så långt ifrån varandra. Vid låg exciteringsgrad erhölls ingen information om den snabba polens uppträdande. Vi trodde att detta kunde resultera i att estimeringen gav en gemensam faktor i A och B, vilket omöjliggör en exakt lösning av designekvationen. För att undvika detta problem införde vi ett test av gemensam faktor: om lösningen till B(q) = 0, insatt i A(q), gav ett resultat som var mindre än en viss tolerans, så räknade vi istället ut ett reducerat system av första ordningen.

Till slut visade det sig dock att problemet *inte* var att en pol och ett nollställe tog ut varandra. Istället verkade det som om estimeringen (av fyra parametrar) helt enkelt blev dålig i allmänhet när exciteringsgraden var låg. Detta skulle man kunna lösa genom att estimera en lägre ordningens modell eller t.ex. använda "directional forgetting". Av tidsbrist fick vi dock avstå från vidare utvecklingar i denna riktning.

Problemet som beskrivs ovan blev särskilt akut när man kopplade in adapteringen och ställde in en önskad bandbredd som var mindre än c:a 100 rad/s. A_m-polynomet agerade då som ett filter och tog bort väsentlig infor-

mation om den snabba polen från estimeringen. Detta skulle man eventuellt kunna lösa genom att införa någon form av automatisk excitation.

Typiska värden på de estimerade polynomen när regleringen fungerade bra är

$$B_{est}(q) = (8{,}978q + 7{,}996) \cdot 10^{-3} \text{ samt}$$

$$A_{est}(q) = q^2 - 1{,}628q + 0{,}6283,$$

vilket stämmer väl överens med de i Appendix A teoretiskt framräknade värdena:

$$B(q) = (8{,}9282q + 7{,}675) \cdot 10^{-3} \text{ samt}$$

$$A(q) = q^2 - 1.6347q + 0{,}6347.$$

Här ser man ett annat problem som man i en ideal värld borde ta hänsyn till: B-polynomets parametrar är flera tiopotenser mindre än A-polynomets. Någon form av skalning i estimeringen vore alltså på sin plats

2.2.3. Design

Designen baserades på den med frekvensanalys identifierade modellens *ordning* —någon annan information ur denna analys användes inte. Vi antog alltså att systemet kunde beskrivas av

$$H(q) = \frac{B(q)}{A(q)} = \frac{b_0 q + b_1}{q^2 + a_1 q + a_2}$$

Utifrån de kända polynomen B(q) och A(q) och styrlagen

$$Ru = -Sy + Tu_c$$

dimensionerades en regulator genom att först lösa designekvationen

$$AR + BS = A_o A_m$$

$$\deg A_o \geq 2 \deg A - \deg A_m - \deg B^+ - 1$$

Beroende på om man vill att täljarens nollställe skall ingå i det slutna systemets överföringsfunktion eller inte uppstår två fall:

a) Man förkortar processens nollställe

Att förkorta nollställe innebär att $B = B^- B^+$ med $B^- = b_0$ och $B^+ = q + b_1/b_0$. Detta ger att $R = R_1 B^+$. I vårt fall gäller dessutom pga polynomens gradtal att $R_1 = A_o = 1$. Man får alltså:

$$A + B^- S = A_m$$

med lösningen

$$R = q + b1/b_0$$

$$S = s_o q + s_1, \text{ där } s_0 = (a_{m1} - a_1)/b_0 \text{ och } s_1 = (a_{m2} - a_2)/b_0$$

b) Man behåller processens nollställe

Beräkningarna i detta fall blir mer krävande. Genom att ansätta

$$A_o = q, R = q + r_1 \text{ och } S = s_0 q + s_1$$

får vi ett linjärt ekvationssystem:

$$\begin{pmatrix} 1 & b_0 & 0 \\ a_1 & b_1 & b_0 \\ a_2 & 0 & b_1 \end{pmatrix} \begin{pmatrix} r_1 \\ s_0 \\ s_1 \end{pmatrix} = \begin{pmatrix} a_{m1} - a_1 \\ a_{m2} - a_2 \\ 0 \end{pmatrix}$$

vars lösning ger oss:

$$r_1 = \frac{a_{m2} + a_o a_{m1} - a_2 - b_1(a_{m1} - a_1)}{1 - b_0 a_2/b_1 - b_1/b_0}$$

$$s_0 = (a_{m1} - a_1 - r_1)/b_0$$

$$s_1 = -a_2 r_1 / b_1$$

För båda regulatorna gäller dessutom att

$$T = A_o B_m / B^-$$

Regulatorerna implementerades sedan genom att ersätta systemets parametrar med motsvarande skattningar.

Kommentar: I vårt fall är det olämpligt att förkorta systemets nollställe, eftersom detta ligger på negativa reella axeln. Som bekant leder förkortning av ett sådant nolltsälle att styrsignalen byter tecken en gång per samplingsintervall. Vi implementerade ändå båda regulatorerna för demonstrationsändamål.

2.3. Kommunikation med omvärlden

För att kommunicera med omvärlden använde vi programmet LabView. LabView är ett helt grafikbaserat program för programmering och kommunikation med instrument utanför datorn. Vi använde programmet för att styra och reglera en för oss okänd process.

På LabViews panel (se fig 2.1) har vi presenterat insignaler och utsignaler från processen samt vissa intressanta beräkningar. I diagramfönstret längst ner till höger presenteras referenssignal u_c och utsignalen y. Referenssignalen genereras av en signalgenerator implementerad i LabView (se Appendix). Man kan välja frekvens, amplitud och fyra olika kurvformer (sinus-, fyrkant-,triangel- och tangensvåg).

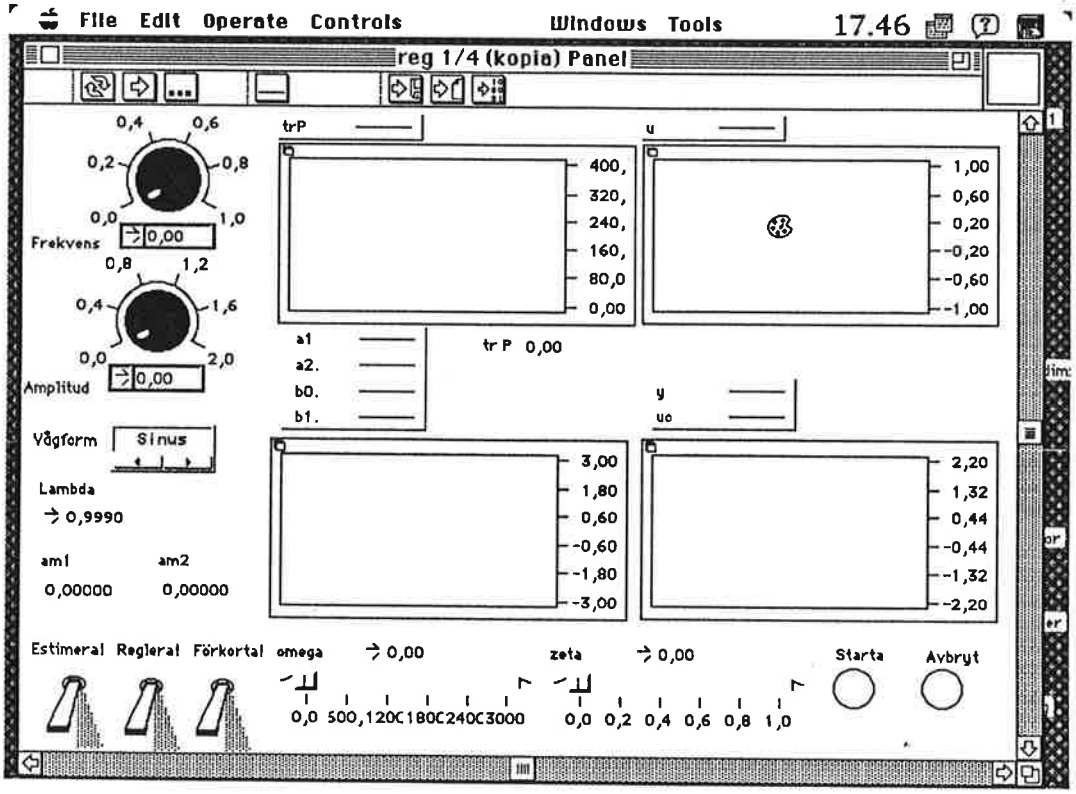


Fig 2.1: Regulatorns användargränssnitt i LabView.

I diagramfönstret längst upp till höger presenteras styrsignalen u. Denna läses från DSPn. Modellparametrarna estimeras i DSPn och presenteras i diagramfönstret längst ner till vänster. Med knappen "Estimera!" väljer man om man vill estimera eller ej.

Spåret av kovariansmatrisen P (märkt "trP") visas i diagramfönstret längst upp till vänster. Spåret är en bra indikator på dels om estimeringen ligger rätt och dels om det finns tillräckligt mycket information i insignalen. Det finns även möjlighet att välja glömskefaktorn λ (märkt "Lambda").

Med omkopplaren "Reglera" väljer man om man vill använda en enkel P-regulator eller en adaptiv RST-regulator. P-regulatorn används för att estimera parametrarna innan den adaptiva RST-regulatorn sätts igång.

Det finns möjlighet att välja att förkorta processens nollställe eller ej. Detta görs med omkopplare "Förkorta".

Längst ner på panelen kan man välja ω och ζ i det önskade nämnarpolynomet, A_m. Dess parametrar a_{m1} och a_{m2} presenteras i två rutor i den vänstra kanten på panelen.

Med tryckknapparna "Start" och "Avbryt" laddas C programmet ner till DSP respektive stannas DSPn. Detta görs i ett underprogram till vår panel som heter DSP console som endast syns som en ikon i huvudprogrammet.

Ytterligare underprogram som används är "Write Shared Data FP" och "Read Shared Data FP" vilka skriver samt läser data från DSPn.

3. Realtidsaspekter

3.2. Processer

Eftersom vi har implementerat de funktioner som traditionellt utförs av speciella processer för grafik och operatörskommunikation i LabView, har vårt C-program blivit relativt enkelt. Två processer återstår: en regulatorprocess och huvudprocessen.

Regulatorprocessen, som har prioriteten 3[1], innehåller estimering och adaptering för en indirekt RST-regulator samt en vanlig P-regulator. Allt detta körs en gång per sampelintervall.

Huvudprocessen (prioritet 12) sköter huvudsakligen kommunikationen mellan signalprocessorn och Macen. Denna kommunikation sker via ett gemensamt minne med 64 platser. Här föreligger ingen risk för kollision eftersom en viss minnesplats bara kan läsas från DSP:n och skrivas från Macen eller vice versa.

3.3. Semaforer?

De två processerna i vårt program delar en mängd data. I vanliga fall skulle det behövas (minst) en semafor eller monitor för att garantera ömsesidig uteslutning. Emellertid har vi tur i detta avseende; pga av realtidskärnans klockupplösning kan det aldrig uppstå en sådan konflikt.

Med utgångspunkt i att ingen process kör är tre situationer tänkbara:

a) Regulatorprocessen startar först

Eftersom nu huvudprocessen har lägre prioritet, kan det aldrig avbryta den pågående processen.

---

1 Prioriteterna i den använda realtidskärnan ligger mellan 1 och 15. Lägre nummer ger högre prioritet. Prioriteterna 1 och 15 är upptagna av kärnans egana funktioner.

b) Huvudprocessen startar först

Eftersom regulatorprocessen har högre prioritet, skulle den kunna avbryta den pågående processen mitt i under en operation på gemensamma data. Men det första gången det ges tillfälle till det är vid nästa "klocktick", dvs 250 µs efter det att huvudprocessen startar. Huvudprocessen tar dock bara c:a 30 µs att köra[2] och alltså är den klar när regulatorn kommer in!

c) Processerna startar samtidigt

Detta fall kan dock aldrig uppstå pga av regulatorprocessens högre prioritet.

Nu skulle man kunna anmärka att ett längre huvudprogram skulle kräva en monitor. I vårt fall skulle denna konstruktion bli onödigt klumpig eftersom nästan alla data är delade. Man fick i så fall organisera om programmet helt och hållet.

4. Resultat

4.1. Processen

Den inledande processidentifieringen mha frekvensanalys gav delvis missvisande resultat. Dock kunde man utläsa systemets ordning och polernas ungefärliga lägen, vilket var allt som behövdes för att konstruera en indirekt regulator.

Den adaptiva regulatorn innehöll också en identifiering av processparametrarna. Resultatet här stämde mycket väl överens med de teoretiska värdena i Appendix A.

4.2. Regulatorn

Vi implementerade först en direkt regulator, men lyckades inte få den att fungera. Därför gick vi över till en indirekt, som har den fördelen att man kan separera estimator- och regulatordelarna.

Det största problem vi har haft har varit att processens dynamik spänner över ett så stort frekvensintervall. Om man specificerar en för låg bandbredd i systemets önskade uppförande, blir det omöjligt att estimera så många parametrar som fyra, eftersom excitationen inte räcker till; systemet uppför sig väsentligen som ett första ordningens system. Vi försökte lösa det genom att välja en reducerad modell i de fall där estimeringen gav gemensamma faktorer i A- och B-polynomen. Detta visade sig inte fungera.

Ett bättre sätt hade troligen varit att ha två parallella estimatorer och välja mellan dem beroende på det önskade systemets bandbredd. Dessutom borde man skala elementen i regressorvektorn till ungefär samma storlek och

---

2 C:a 2 µs per anrop av C30toIEEE eller IEEEtoC30, samt mindre än en µs per anrop av write_shared_data resp. read_shared_data.

eventuellt införa någon sorts test på om processen exciteras tillräckligt. Pga tidsbrist kunde vi inte prova dessa metoder.

Trots dessa problem lyckades vi åstadkomma en adaptiv regulator med ett användarvänligt gränssnitt. Systemets uppförande reglerades genom två s.k. *performance-related knobs*[3], med vilka man ställde in systemets önskade bandbredd resp. dämpning. Regulatorn fungerade mycket bra i området 200–3000 rad/s—stegsvaren såg ut som man kunde förvänta sig.

De viktigaste erfarenheterna från projektet har varit

— att det verkligen går att med så liten ansträngning köra en adaptiv regulator med sampeltakter upp till 2 kHz. Detta är möjligt tack vare DSP:ns prestanda, men också den utmärkta programmeringsmiljön.

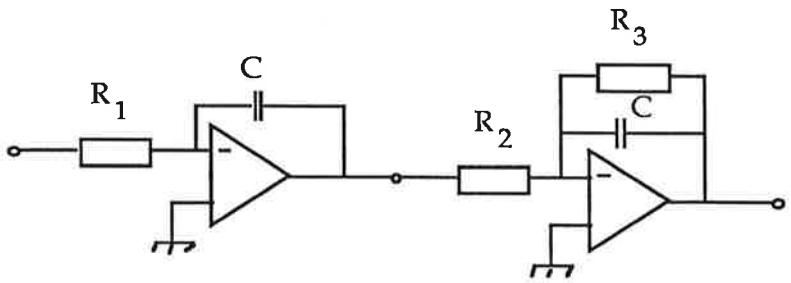
— hur mycket grafiska programmeringshjälpmedel kan förkorta utvecklingstiderna för komplicerade projekt av denna typ genom att låta programmerarna koncentrera sig på de väsentliga delarna av programmet, speciellt eftersom en mängd realtidsrelaterade problem undviks.

---

3 Ett reglage som är graderad i en för användaren praktisk storhet.

Appendix A

Efter projektets slut fick vi reda på hur den process vi reglerat verkligen såg ut. Kopplingsschemat framgår av figur A.1



Figur A.1: Kopplingsschema för processen.

Man kan visa att denna koppling har (spännings-)överföringsfunktionen

$$G(s) = \frac{1}{R_2 R_1 Cs\,(Cs + \frac{1}{R_3})}$$

Med insatta komponentvärden

$$R_1 = 100\ k\Omega,\ R2 = 10\ K\Omega \text{ och } C = 0{,}22\ \mu F$$

får man

$$G(s) = \frac{20661}{s\ (s + 454{,}55)}$$

Om detta system samplas med frekvensen 1 kHz får man den tidsdiskreta motsvarigheten

$$H(q) = \frac{8{,}9282q + 7{,}675}{q^2 - 1.6347q + 0.6347} \cdot 10^{-3},$$

vilket som synes stämmer väl överens med de praktiska resultaten i avsnitt 2.2.2.

Appendix B

B.1. Simnonprogram

```
DISCRETE SYSTEM dirreg

STATE       th1 th2 th3 th4
STATE       p11 p12 p13 p14
STATE       p22 p23 p24
STATE       p33 p34
STATE       p44
STATE       uf ufold
STATE       yf yfold
STATE       uold1 uold2
STATE       yold1 yold2
STATE       u

NEW   nth1 nth2 nth3 nth4
NEW   np11 np12 np13 np14
NEW   np22 np23 np24
NEW   np33 np34
NEW   np44
NEW   nuf nufold
NEW   nyf nyfold
NEW   nuold1 nuold2
NEW   nyold1 nyold2
NEW   nu

INPUT       ucy
OUTPUT      uout
TIME   t
TSAMP       ts

"Bilda am enlig CCS, s. 65
INITIAL
zeta = if zeta0<0 then 0 else if zeta0>1 then 1 else zeta0
wok  = if w0<1 then 1 else if w0>3000 then 3000 else w0
w    = sqrt(1-zeta*zeta)*wok
alfa = exp(-zeta*wok*h)
beta = cos(w*h)
am1  = -2*alfa*beta
am2  = alfa*alfa

SORT

" Bilda uf enligt AoAmuf = qAmuf = u <=>
" quf = -am1uf(t) - am2uf(t-1) + u(t-2).

nuold1 = u
nuold2 = uold1
nufold = uf
nuf   = -am1*uf - am2*ufold  + uold2

" Bilda yf enligt AoAmyf = qAmyf = y <=>
" qyf = -am1yf(t) - am2yf(t-1) + y(t-2).
nyold1 = y
nyold2 = yold1
```

```
nyfold = yf
nyf   = -am1*yf - am2*yfold   + yold2

"Bilda  phi-vektorn:
fi1=uf
fi2=ufold
fi3=yf
fi4=yfold

" Bilda phit * P
trP=p11+p22+p33+p44
c1p=if itr<0.5 then 1 else c1/trP
pf1=c1p*(p11*fi1+p12*fi2+p13*fi3+p14*fi4)
pf2=c1p*(p12*fi1+p22*fi2+p23*fi3+p24*fi4)
pf3=c1p*(p13*fi1+p23*fi2+p33*fi3+p34*fi4)
pf4=c1p*(p14*fi1+p24*fi2+p34*fi3+p44*fi4)

"Bilda inv(1 +  fit P fi)
den1=lam+(fi1*pf1+fi2*pf2+fi3*pf3+fi4*pf4)
den=den1+c2*(fi1*fi1+fi2*fi2+fi3*fi3+fi4*fi4)


"Bilda  felet:
eps=y-fi1*th1-fi2*th2-fi3*th3-fi4*th4

g1=pf1/den
g2=pf2/den
g3=pf3/den
g4=pf4/den

" Bilda nytt estimat
nth1=th1+g1*eps
nth2=th2+g2*eps
nth3=th3+g3*eps
nth4=th4+g4*eps

" Bilda ny p-matris:
np11=(p11-pf1*pf1/den)/lam
np12=(p12-pf1*pf2/den)/lam
np13=(p13-pf1*pf3/den)/lam
np14=(p14-pf1*pf4/den)/lam
np22=(p22-pf2*pf2/den)/lam
np23=(p23-pf2*pf3/den)/lam
np24=(p24-pf2*pf4/den)/lam
np33=(p33-pf3*pf3/den)/lam
np34=(p34-pf3*pf4/den)/lam
np44=(p44-pf4*pf4/den)/lam

"Bilda  parametrar:
r0=if abs(th1)<0.001 then sign(th1)*0.001 else th1
r1=th2
s0=th3
s1=th4

"Bilda  utsignalen
v = (-r1*uold1 -s0*y -s1*yold1 +(am1+am2+1)*uc)/r0
ulim = if v>vmax then vmax else if v<-vmax then -vmax else v
uout = ulim
```

```
"Uppdatera  utsignaltillstandet:
nu = ulim
ts = t + h
h:  0.001
w0 :628
zeta0:0.7
lam: 0.98
vmax:2
c1:10   "Parametrar for saker estimering --- se Ad Ctrl, s. 411-412
c2:0.001
itr:1

END
```

B.2. C-program

```
#include <kern:thread.h>
#include <libc:stdio:stdio.h>
#include <kern:timer_event.h>
#include <drivers:nb_mio_16.h>
#include  <peripherials:shared_data.h>
#include <peripherials:tolEEEtoC30.h>

#define MIO_16_SLOT 5
#define h 0.001
#define umax 1
#define tol 0.1

float r1,s0,s1,t0,t1;
float estim = 0.0,reg = 0.0, zero=1.0;
float u,uold1,y,yold1,uref,urefold;

float  fi1,fi2,fi3,fi4,g1,g2,g3,g4;
float  pf1,pf2,pf3,pf4,th1,th2,th3,th4;
float eps,lambda,trP,c1p,den;
float p11,p12,p13,p14;
float p22,p23,p24;
float p33,p34;
float p44;

float bs, n, amsum, tr1, ts0, ts1, zeta, w;
float  a1,a2,b0,b1,am1,am2;

void controller() {
        mio_16          mio16brd;
        event_t control_timer;
        timevalue       firsttime,interval;

        IEEEfloat fpInput;

        long    channels[] = {0};
        float   data[1];

        gettimeofday(&firsttime);
        interval.tv_sec = 0;
        interval.tv_usec = 1000;
```

```
control_timer = create_timer_event(NULL,&firsttime,&interval);

mio16brd = mio_16_init(MIO_16_SLOT,28);

/* Initerar param */

lambda = 1;

th1 = th2 = th3 = th4 = 0;
fi1 = fi2 = fi3 = fi4 = 0;

/*Initialvärden för P-matrisen*/

p11 = 10000;
p12 = 0;
p13 = 0;
p14 = 0;
p22 = 10000;
p23 = 0;
p24 = 0;
p33 = 10000;
p34 = 0;
p44 = 10000;

u = 0;
y = 0;

/* P-regulator som default */
r1=0;
t0=0.3;
t1=0;
s0=0.3;
s1=0;

do {
        read_shared_data(0,fpInput,IEEEfloat);
        uref= IEEEtoC30(fpInput)/10;

        /* Läs från A/D */

        mio_16_analog_in(mio16brd,channels,1,data);
        y = data[0];

        /* Bilda styrsignalen */

        if (reg>0.5)  /* Om adaptering påslagen... : */
          u = -r1*u -s0*y -s1*yold1 + t0*uref + t1*urefold;
        else u = 3*(uref-y);   /* ... annars P-reglering.*/

        /* Begränsa utsignalen: */
        if (u > umax)
          u = umax;
        else if (u<-umax)
          u = -umax;

        /* Skriv till D/A: */

        mio_16_analog_out(mio16brd,0,u);
```

```
if (estim>0.5)  { /* Estimera regparam */

        trP = p11 + p22 + p33 + p44;

        /* Bilda P*fi */
        pf1 = p11*fi1 + p12*fi2 + p13*fi3 + p14*fi4;
        pf2 = p12*fi1 + p22*fi2 + p23*fi3 + p24*fi4;
        pf3 = p13*fi1 + p23*fi2 + p33*fi3 + p34*fi4;
        pf4 = p14*fi1 + p24*fi2 + p34*fi3 + p44*fi4;

        /* Bilda prediktionsfelet */
        eps = y - fi1*th1 - fi2*th2 - fi3*th3 - fi4*th4;

        /* Bilda hjäpstorheter */
        den = lambda + (fi1*pf1 + fi2*pf2 + fi3*pf3 + fi4*pf4);
        g1 = pf1/den;
        g2 = pf2/den;
        g3 = pf3/den;
        g4 = pf4/den;

        /* Uppdatera estimatet */
        th1 = th1 + g1*eps;
        th2 = th2 + g2*eps;
        th3 = th3 + g3*eps;
        th4 = th4 + g4*eps;

        /* Uppdatera P-matrisen */
        p11 = (p11 - pf1*pf1/den)/lambda;
        p12 = (p12 - pf1*pf2/den)/lambda;
        p13 = (p13 - pf1*pf3/den)/lambda;
        p14 = (p14 - pf1*pf4/den)/lambda;
        p22 = (p22 - pf2*pf2/den)/lambda;
        p23 = (p23 - pf2*pf3/den)/lambda;
        p24 = (p24 - pf2*pf4/den)/lambda;
        p33 = (p33 - pf3*pf3/den)/lambda;
        p34 = (p34 - pf3*pf4/den)/lambda;
        p44 = (p44 - pf4*pf4/den)/lambda;

        /* Uppdatera regressorerna */
        fi4 = fi3;
        fi3 = u;
        fi2 = fi1;
        fi1 = -y;

        a1 = th1;
        a2 = th2;
        b0 = th3;
        b1 = th4;

        /* Bilda hjälpstorheter */
        bs = b0 + b1;
        n  = a1 - b0*a2/b1 - b1/b0;
        amsum = 1 + am1 + am2;
        tr1 = (am2  - a2 - b1*(am1 - a1)/b0)/n;
        ts0 = (am1 - a1 - tr1)/b0;
        ts1 = -a2*tr1/b1;
```

```
                    if (zero<0.5) {    /* Regulator med förkortning av nollställe */
                            t0=amsum/b0;
                            t1=0;
                            r1=b1/b0;
                            s0=(am1-a1)/b0;
                            s1=(am2-a2)/b0;
                    } else {   /* Dito utan dito */
                            t0=amsum/bs;
                            t1=0;
                            r1=tr1;
                            s0=ts0;
                            s1=ts1;
                    }
            }
            else {     /* P-regulator */
                    r1=0;
                    t0=0.3;
                    t1=0;
                    s0=0.3;
                    s1=0;
                    th1=th2=th3=th4=0;
                    p11=p22=p33=p44=100;
                    p12=p13=p14=p23=p24=p34=0;
            }

            /* Uppdatera (egentligen onödiga) signaltillstånd */
            uold1 = u;
            yold1 = y;
            urefold=uref;

            /* Vänta på sampelögonblick */
            event_wait(control_timer,0);

    } while (1); /* ... och börja om igen */
}

void main() {
    thread_t regul;
    IEEEfloat fpInput;

    event_t timer;
    timevalue    firsttime,interval;

    estim = reg =0;
    uref=0;
    trP=0;

    /* Skapa regulatorprocessen */
    regul = thread_create(controller,3);
    thread_resume(regul);

    /* Timer som bestämmer hur ofta vi läser refsignal etc. från    LabView */
    gettimeofday(&firsttime);
    interval.tv_sec = 0;
    interval.tv_usec = 10000;
    timer = create_timer_event(NULL,&firsttime,&interval);
```

Adaptiv friktionskompensering av elektriskt positionsservo Projektarbete i Adaptiv reglering och Realtidssystem, VT 1993

Jonas Eborn, F89
Jeppa Grosshög, F89
Henrik Värendh, F89
Reglerteknik, LTH
Handledare: Johan Nilsson, Henrik Olsson

7 maj 1993

Sammanfattning

Detta är en rapport av ett projekt i kurserna Adaptiv reglering och Realtidssystem vid Institutionen för Reglerteknik, LTH, våren 1993. Projektet behandlar adaptiv friktionskompensering av ett elektriskt servo med en olinjär och dynamisk friktionsmodell.

$$\dot{F} = \sigma v \left(1 - sign(v)\frac{F}{F_c}\right)$$

Här är F_c glidfriktionen och v servots vinkelhastighet. Modellen har vi skrivit om mha en lägeskoordinat z, samt ersatt F_c med funktionen $g(v)$. Därefter implementerat den som en observerare med en korrektionsterm från positionsfelet e:

$$\begin{cases} \hat{F} = \sigma\hat{z} + \sigma_1\dot{\hat{z}} \\ \dot{\hat{z}} = v - \frac{|v|}{g(v)}\hat{z} - Ke \end{cases}$$

Modellen har funnits beskriva friktionen bra vid sinussignaler som referens. Reglerfelet reducerades med upp till en faktor tio på det undersökta servot jämfört med samma regulator utan den friktionskompenserande modellen inkopplad. Vi har använt P- och PI-regulatorer, men även en D-del är implementerad. Mätbara signaler är positionen och vinkelhastigheten.

Innehåll

1 Inledning

I kurserna Adaptiv reglering och Realtidssystem vid LTH ingår ett projektarbete. Vi har valt att studera hur man kan kompensera för friktionen i ett positionsservo. Friktion är ett problem som ofta ger betydande fel vid positionering av reglersystem och då reglersystem ska följa en viss bana. Speciellt gäller detta för elektriska servon, mekaniska kopplingar och växlar. Utvecklingen inom robottekniken med större precisionskrav ställer också allt högre krav på bra friktionskompensering. Problemen yttrar sig främst vid låga hastigheter och vid byte av rörelseriktning. Pga friktionen fastnar t ex servot vid ett felaktigt läge. När styrsignalen blivit tillräckligt stor lossnar det och servot kan då göra en häftig knyck och förflytta sig till motsatt sida av referensvärdet och åter fastna. Har man en integrator i reglerdesignen kan den vridas upp till ett stort värde och orsaka instabilitet och reglerfel.

Friktionen har ofta temperatur- och tidsberoende. Det skapar problem genom att regulatorparametrarna för en statisk regulator kan behöva ändras mellan olika körningar. Den är också olinjär. Traditionellt har man löst problemen med höga regulatorförstärkningar, vilket dock leder till bruskänsliga system.

2 Teori

Klassiskt har friktion beskrivits som en kombination av vidhäftning och glidfriktion. Detta ger upphov till ett större motstånd att övervinna då en rörelse påbörjas och ett lägre konstant, eller eventuellt visköst, motstånd som håller emot då rörelsen pågår. Denna modell har varit förhärskande mycket länge och det är först i moderna tillämpningar med ökande prestandakrav som denna har ifrågasatts. Den klassiska modellen ger upphov till diskontinuiteter i friktionskraften då rörelsen vänder, stick i stäv med nyare experiment som visar en kontinuerlig övergång mellan glidfriktion av olika tecken.

Detta kontinuerliga friktionsbeteende kan beskrivas av en första ordningens olinjär differentialekvation[1].

$$\dot{F} = \frac{F_c}{\theta}\left|1 - sign(v)\frac{F}{F_c}\right|^i v$$

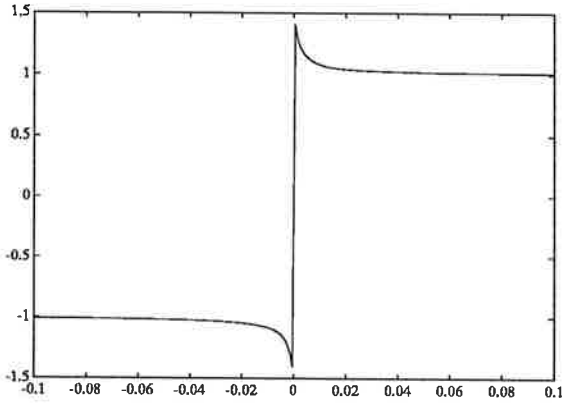
Här är F friktionen, F_c den statiska glidfriktionen, θ en rörelsekonstant, v hastigheten och i en exponentparameter. Som beskrivs i [4] har vi förenklat denna genom att välja $i = 1$ och anta att $-F_c \leq F \leq F_c$. Dessutom skriver vi om den genom att sätta $\sigma = F_c/\theta$. Detta ger den förenklade ekvationen:

$$\dot{F} = \sigma v \left(1 - sign(v)\frac{F}{F_c}\right) \tag{1}$$

Den stationära lösningen till (1) blir $F = F_c sign(v)$. För att modellera en större statisk friktion (Stribeck-effekten) ersätter vi F_c med en funktion $g(v)$ som är större för små v. Vi har valt funktionen som

$$g(v) = g\left(1 + \frac{g_{num}}{g_{den} + |v|}\right) \tag{2}$$

där g, g_{num} och g_{den} är parametrar, g kan man välja så att den har olika värde för positiva och negativa hastigheter. I figur 1 visas $g(v)$ med $g_{num} = 0.001$ och $g_{den} = 0.002$.



Figur 1: Funktionen g(v) med $g_{num} = 0.001$ och $g_{den} = 0.002$.

Idén vi försöker förverkliga är att till den linjära styrsignalen addera en friktionskompenserande term som eliminerar friktionen i servot. Eftersom vi inte kan mäta friktionen observerar vi den i en modell liknande (1). Modellen är i princip ekvivalent med att beskriva friktionen som en fjäderkraft $J\ddot{z} = -F$ för mycket små töjningar z, som om man linearisera F kring $F = 0$, $z = 0$ ger en odämpad, oscillativ modell $J\ddot{z} + \sigma z = 0$. Om vi lägger till en dämpande term $\sigma_1 \dot{z}$ kan vi skriva modellen som

$$\begin{cases} F = \sigma z + \sigma_1 \dot{z} \\ \dot{z} = v - \frac{|v|}{g(v)} z \end{cases} \tag{3}$$

Ekvation (3) har den stationära lösningen $F = \sigma\, g(v) sign(v)$. Om vi nu jämför de stationära lösningarna till våra båda modeller ser man att för hastigheter $|v| \gg g_{den}$ gäller:

$$\sigma\, g \approx F_c \tag{4}$$

3 Processmodell

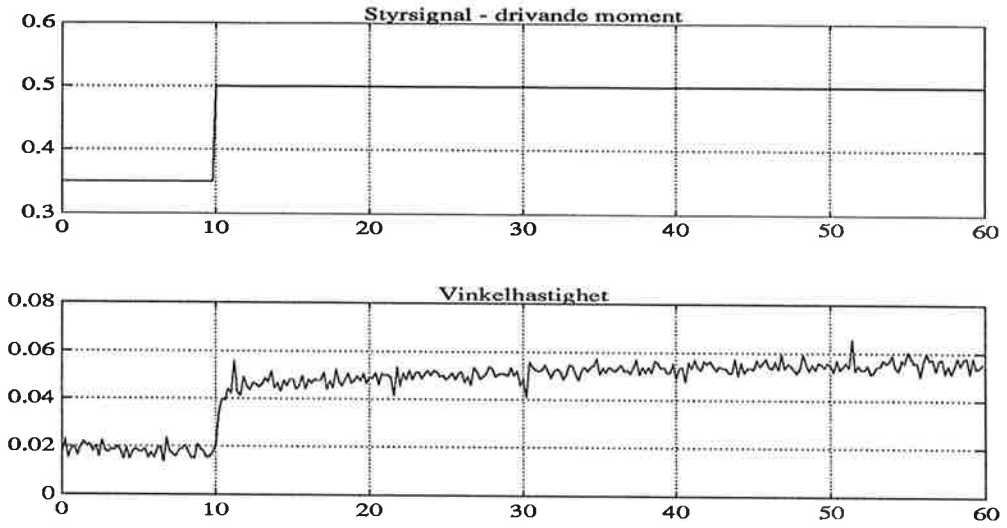
En enkel modell av ett servo är följande överföringsfunktion från insignal till hastighet:

$$G(s) = \frac{1}{Js + d} \tag{5}$$

För att få fram värden på parametrarna gjorde vi ett enkelt stegsvarsexperiment, se figur 2. Vi fick då fram följande värden:

$$d = 3.78$$

$$J = 1.99$$

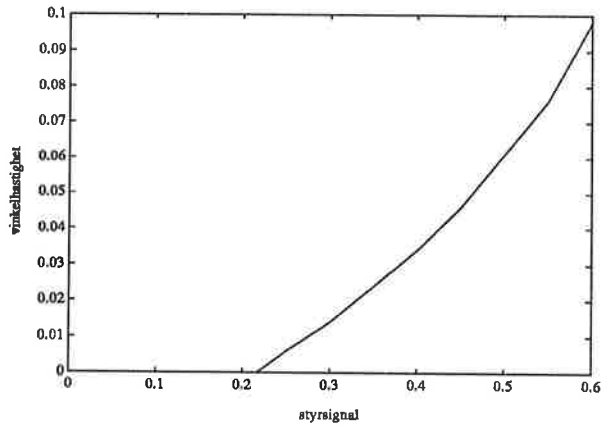


Figur 2: Stegsvarsexperiment.

Detta gav en stationär förstärkning

$$k_p = 0.265$$

Vi gjorde även ett experiment för att få fram ett ungefärligt värde på coulumb-



Figur 3: Insignalexperiment.

friktionen, dvs den statiska friktionen. Vi mätte då vinkelhastigheten hos servot för ett antal drivande moment, vilket är insignalen till servot, se figur 3. Servot börjar röra sig då insignalen är större än cirka 0.28, vilket är vårt värde på coulumb-friktionen:

$$F_c = 0.26$$

I vår friktionsmodell ska det enligt (4) gälla att $\sigma g \approx F_c$.

Experimentet visade också att processen uppför sig ungefär linjärt för insignaler upp till 0.6. För högre insignaler ökar vinkelhastigheten olinjärt. Med vår friktionsmodell ska insignalen sålunda vara maximalt 0.6.

4 Regulatordesign

För att kunna estimera friktionen i servot gör vi en observerare baserad på modellen i (3) där vi ersätter F och z med deras estimat $\hat{F}$ och $\hat{z}$ samt inför en korrektionsterm från positionsfelet $e = \varphi - \varphi_r$. Denna beskrivs av ekvationerna:

$$\begin{cases} \hat{F} = \sigma\hat{z} + \sigma_1\dot{\hat{z}} \\ \dot{\hat{z}} = v - \frac{|v|}{g(v)}\hat{z} - Ke \end{cases} \tag{6}$$

Differentialekvationen för servots vinkelläge blir med överföringsfunktionen (5) samt friktion:

$$J\ddot{\varphi} + d\dot{\varphi} = u - F \tag{7}$$

Denna kan skrivas om till dimensionslös form genom att dividera med J och införa tidskonstanten $\tau = d/J$, vilket ger ekvationen:

$$\ddot{\varphi} = -\tau\dot{\varphi} + \frac{1}{J}(u - F) \tag{8}$$

Nu kan vi genom att införa några extra termer i styrsignalen skriva om detta till en ekvation i reglerfelet e och dessutom kompensera för friktionen F. Vi väljer en styrlag som innehåller dels en linjär del, dels en framkoppling från börvärdet och servots hastighet samt friktionsestimatet ur (6) enligt:

$$u = u_{lin} + J(\ddot{\varphi}_r + \tau\dot{\varphi}) + \hat{F} \tag{9}$$

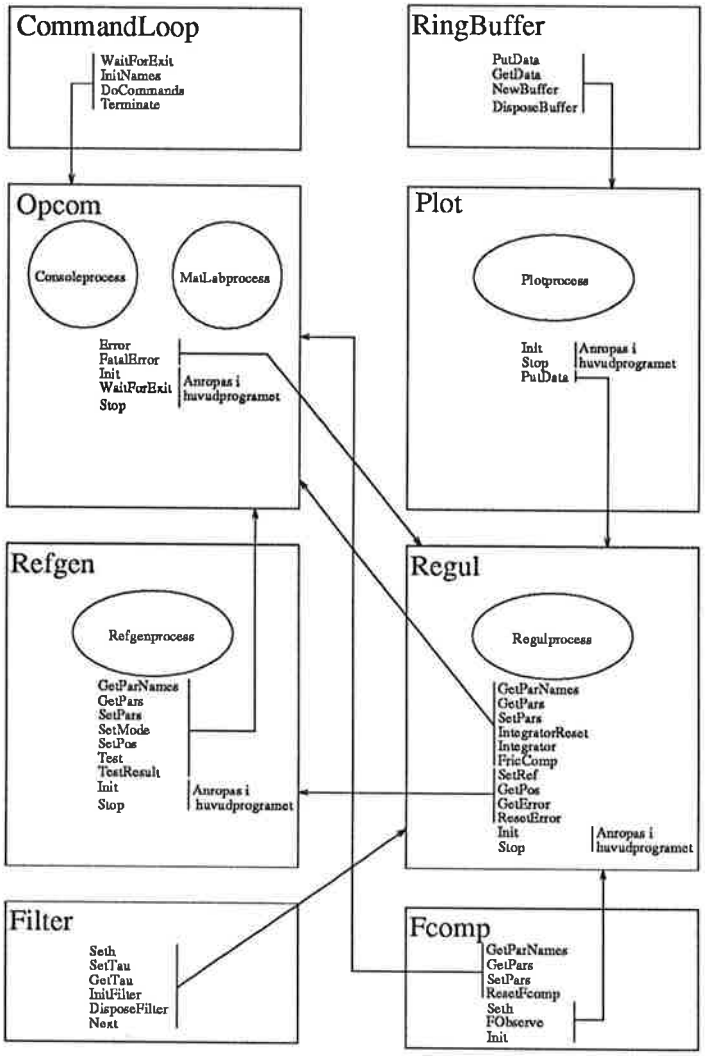
Om vi antar att vi har rätt värde på τ och J, att friktionsestimatet $\hat{F} = F$ samt bakar in ett J i u_{lin} ger detta en differentialekvationen i positionsfelet: $\ddot{e} = u_{lin}$. Den linjära återkopplingen från felet kan beskrivas med en överföringsfunktion $u_{lin} = -G_r(s)e$ och tillsammans ger detta ett slutet system där positionsfelet styrs av ekvationen.

$$(s^2 + G_r(s))e = 0 \tag{10}$$

Om (10) är stabil kommer positionsfelet att gå mot 0.

5 Implementering

Reglersystemet implementerades i modula2 i form av ett antal moduler. Reglerloopen, som består av en PID regulator finns i modulen Regul. Regulatorns börvärden genereras i modulen Refgen, och operatörskommunikationen sker via modulen Opcom. I denna modul finns två processer, en för kommunikation via console, och en för kommunikation via MatLab. Den grafiska presentationen av mät- och börvärden ombesörjs av modulen Plot. Systemets grafer presenteras i matlab. Förutom de ovan nämnda modulerna finns några hjälpmoduler.



Figur 4: Modulgraf.

Modulen RingBuffer fungerar som datakanal mellan Regul och Plot. Modulen CommandLoop tar hand om översättningen av operatörens kommandon till proceduranrop i systemet. Modulen Filter är en implementation av ett första ordningens lågpassfilter, som används för att filtrera systemets mätsignaler, och modulen Fcomp är implementeringen av vår friktionskompensering. I figur 4 visas modulgrafen för programmet och nedan ges en kort presentation av de olika modulerna.

5.1 Regul

Modulen Regul är en implementering av (9) med en PID regulator för den linjära styrsignalen. D-delen beräknas med hjälp av uppmätta värden för hastigheten, och I-delen approximeras med Eulers formel. Börvärdets position, hastighet och acceleration fås från Refgen. Förutom denna reglermod kan man

ange nödstopp, varvid styrsignalen sätts till 0 och regulatorn stannar. De parametrar som kan påverkas i Regul är sammanställda i nedanstående tabell.

parameter	namn	startvärde
Samplingstid	h	0.05
Förstärkning	k	5.0
Integratortid	ti	3.0
Derivatatid	td	0.0
Trackingtid	tr	1.0
Maximal utsignal	umax	0.6
Tröghetsmoment	J	0.27
Tidskonstant i G	tau	1.0
Filterkonstant	veltau	0.5
Förlustvikt, u	rho	0.0

Nedan presenteras de olika procedurer som kan anropas i Regul.

- *GetParNames* returnerar en vektor med namnen på de ovan beskrivna regulatorparametrarna.
- *GetPars* ger de aktuella parametervärdena i en vektor.
- *SetPars* uppdaterar hela parameteruppsättningen.
- *SetRef* används för att ange nästa referensvärde, samt derivata och andraderivata för referensvärdet. Dessutom inläses den aktuella reglermoden.
- *GetPos* returnerar den senast uppmätta positionen hos servot.
- *Init* anropas för att initiera regulatorn.
- *IntegratorReset* nollställer regulatorns I-del.
- *Integrator* kopplar till eller från I-delen i regulatorn.
- *FricComp* kopplar in eller ur friktionskompenseringen.
- *GetError* returnerar det aktuella felet (ärvärde - börvärde) samt en kvadratisk förlustsumma, som beräknas som summan av felet i kvadrat plus rho (se ovan) multiplicerat med utsignalen från regulatorn i kvadrat. Summeringen börjar från noll då ResetError anropas.
- *ResetError* nollställer förlustfunktionen.
- *Stop* avslutar regulatorprocessen.

5.2 Refgen

Refgen har till uppgift att förse regulatorn med börvärden. I Refgen genereras dessutom derivatan och andraderivatan av börvärdet. Refgen kan arbeta i tre olika moder, nämligen positionsreglering till fast position, följande av sinuskurva samt följande av fyrkantkurva. Dessutom finns möjlighet att göra nödstopp, varvid regulatorn beordras att stanna. För att undvika singulariteter i börvärdets derivator rampas hastighet och position med en av operatören specificerad maximal acceleration och hastighet. Då referensgeneratorn körs i sinus-mod är amplitud och frekvens begränsade, så att de angivna maxvärdena för acceleration och hastighet inte överskrids. I nedanstående tabell återfinns de parametrar som kan påverkas i Refgen.

parameter	namn	startvärde
Samplingstid	h	0.1
Maximal hastighet	vmax	1.0
Maximal acceleration	vdotmax	10.0
Amplitud	amp	0.5
Periodtid	per	25.0

I Refgen kan följande procedurer anropas:

- *SetMode* ändrar referensgeneratorns arbetsmod.
- *SetPos* används för att ange önskad position på servot, då man arbetar i positionsmod.
- *GetParNames* returnerar en vektor med namnen på parametrarna i ovanstående tabell.
- *GetPars* returnerar det aktuella värdet på parametrarna i referensgeneratorn.
- *SetPars* ger referensgeneratorns parametrar de angivna värdena.
- *Init* startar referensgeneratorn.
- *Stop* avslutar Refgen-processen.
- *Test* startar summering av förlustfunktionen i Regul. Summeringen pågår över tio perioder.
- *TestResult* returnerar det senast framräknade värdet på förlustfunktionen.

5.3 Opcom

Modulerna Opcom och CommandLoop svarar tillsammans för systemets användarkommunikation. I Modulen Opcom finns två processer. Den ena, som alltid startas ombesörjer kommunikation via en lokal consol. Den andra, som blir aktiv endast om man anger detta vid systemuppstarten kommunicerar med

en fjärrkontroll, som implementeras i MatLab. Användargränssnittet är detsamma i de båda consolerna. Ett typiskt kommando kan se ut som "Regpar k 10 ti 25". Systemet arbetar endast med versaler, men inmatning kan även ske med gemener. Dessa omvandlas av programmet till versaler. Procedurerna i CommandLoop anropas endast av Opcom, och de procedurer som skall anropas av systemets andra delar återfinns i definitionsmodulen för Opcom. Följande proceduranrop kan göras:

- *Init* startar Opcom. Prioriteterna för de båda consol-processerna skall anges. MatLab-kommunikationen erhåller den högsta av de båda prioriteterna.
- *WaitForExit* inväntar signal att systemet skall stängas av.
- *Stop* avslutar kommunikationen med MatLab-consolen.
- *Error* skriver ut ett felmeddelande på consolerna.
- *FatalError* skriver ut felmeddelande på consolerna och avslutar excekveringen av systemet.

5.4 Plot

Modulen Plot tar emot värden som skall presenteras grafiskt i MatLab. dessa skickas via en i modulen RingBuffer implementerad datakanal till MatLab, där värdena presenteras i lämpliga diagram. RingBuffer medger en viss fördröjning i dataöverföringen, genom att data buffras. Om fördröjningen blir för lång kommer gamla data att börja skrivas över av nya. De procedurer som kan anropas i Plot är:

- *Init* startar plottermodulen.
- *PutData* tar emot de data som skall plottas i MatLab. Dessa data skall vara organiserade i en RECORD, som definieras i Plot-Modulen.
- *Stop* avbryter excekveringen av Plot-modulen.

5.5 Fcomp

Fcomp är en implementering av friktionsobserveraren (6). Derivatan i modellen har implementerats med hjälp av Tustins approximation. Konstanten g i (2) har två olika värden, gpos och gneg, beroende på om servot körs framåt eller bakåt. Vi har använt samma värde i båda riktningarna, men för ett assymmetriskt servo kan olika värden behövas. I Fcomp kan följande parametrar påverkas:

parameter	namn	startvärde
Samplingstid	h	0.05
Modellparameter *	sigma	9.0
Modellparameter*	sigma1	4.24
Korrektionsfaktor*	k	0.1
Friktionsfunktion**	gpos	0.012
Friktionsfunktion**	gneg	0.012
Stribeckfaktor**	gnum	- 0.002
Stribeckfaktor**	gden	0.002

* ekvation (6), ** se ovan samt ekvation (2)

Parametern h skall ha samma värde som h i Regul, och bör följdaktligen ändras samtidigt i Regul och Fcomp viket också sker. I Modulen Fcomp finns följande proceduranrop:

- *SetPars* ändrar värdet på samtliga parametrar i Fcomp, utom h.
- *ResetFcomp* nollställer friktionsobserveraren, och börjar om med den estimerade friktionen 0.
- *GetPars* returnerar de aktuella värdena på Fcomps parametrar.
- *GetParNames* returnerar namnen på Fcomps parametrar.
- *Seth* ändrar värde på samplingstiden (skall vara samma som h i Regul).
- *Fobserve* beräknar nästa skattning av friktionskraften, samt uppdaterar observerarens interna tillstånd. FObserve bör anropas varje samplingsintervall även om resultatet inte skall användas.
- *Init* initierar friktionsobserveraren.

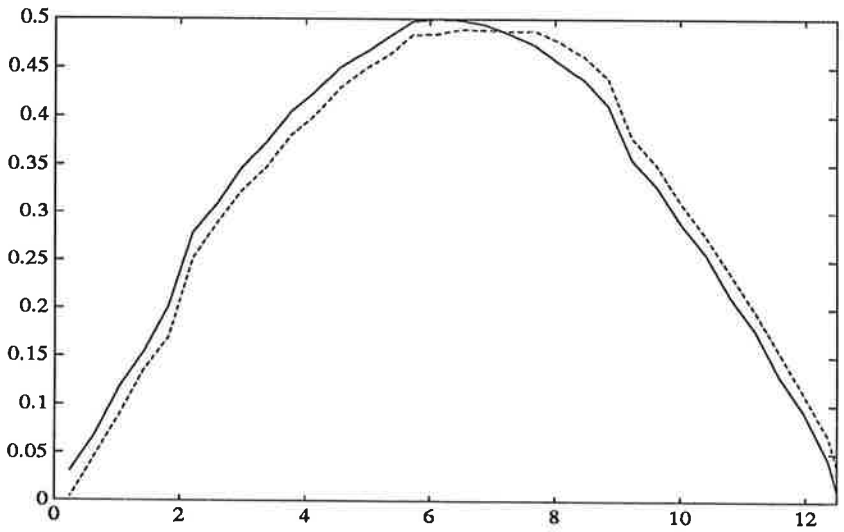
5.6 Filter

Filter är en ZOH implementering av ett första ordningens lågpassfilter. tidskonstanten avser brytpunkten för motsvarande kontinuerliga filter. Filter.def inehåller följande procedurer:

- *Seth* ändrar filtrets samplingstid. Samplingstiden bör vara densamma som i Regul.
- *SetTau* ändrar filtrets tidskonstant.
- *GetTau* returnerar filtrets aktuella tidskonstant.
- *InitFilter* skapar ett nytt filter med samplingstid 1.0 och tidskonstanten 1.0.
- *DisposeFilter* avlägsnar filtret och dess datastrukturer.
- *Next* returnerar nästa filtrerade värde. Proceduren bör anropas varje samplingsintervall, eftersom filtrets interna tillstånd annars inte uppdateras.

6 Experiment

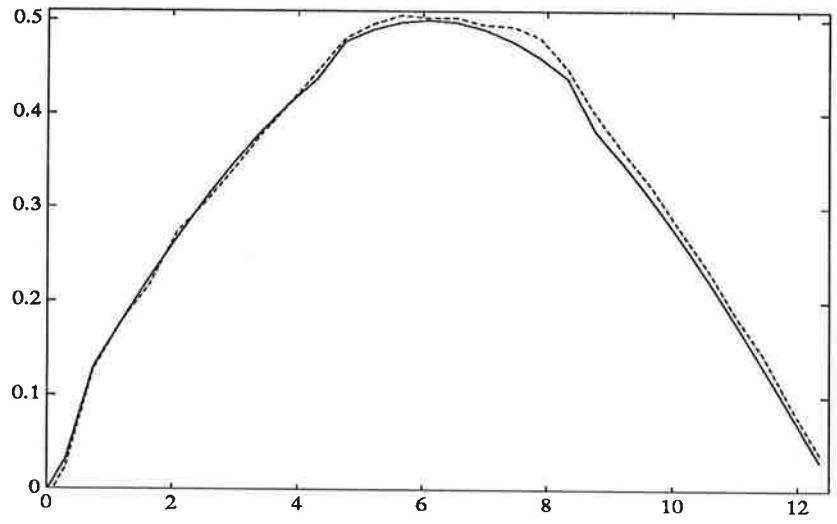
Vi har provat vår implementering på ett någorlunda stort direktdrivet positionsservo med en friktion värd att tala om. Friktionen minskade dock efterhand som vi 'körde in' servot och därför har vi inte lagt ner någon möda på att göra en perfekt nominell reglering utan använt olika PID-regulatorer med acceptabla parametrar. Typiskt kan man se att vid P-reglering ligger servot alltid lite



Figur 5: Referens och utsignal vid P-reglering.

efter referensen med ett i princip konstant fel. Vid sinuskurvans ändlägen då hastigheten är noll når inte servot ända fram utan stannar för tidigt, felet byter tecken och servot kommer lika mycket efter på vägen ner. Se figur 5. Integralverkan kompenserar för den konstanta glidfriktionen medan servot rör sig, men då rörelsen vänder medverkar I-delen och den motsatta friktionskraften till en relativt stor översläng innan I-delen har bytt tecken. Se figur 6. Då man tittar på kurvan för referensvärdet kan man se att den verkar tillknycklad på sina ställen. Detta beror antagligen på realtidsproblematiken med olika prioriteter och samplingstider för Regul och Refgen.

När vi sedan skulle koppla in friktionskompenseringen märkte vi att de modellparametrar vi mätt upp enligt kapitel 3 fungerade mindre bra. Därför provade vi oss fram till andra parametrar som fungerade bättre. Följande värden användes:

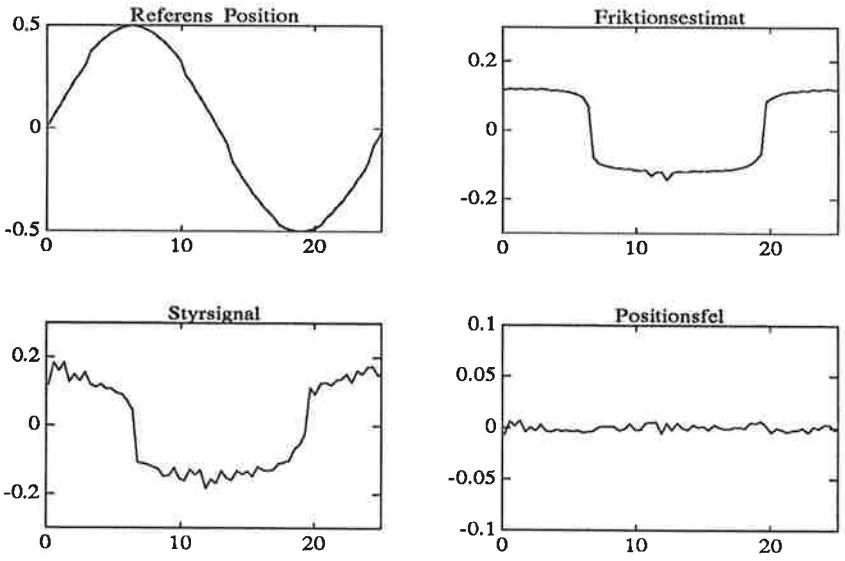


Figur 6: Referens och utsignal vid PI-reglering.

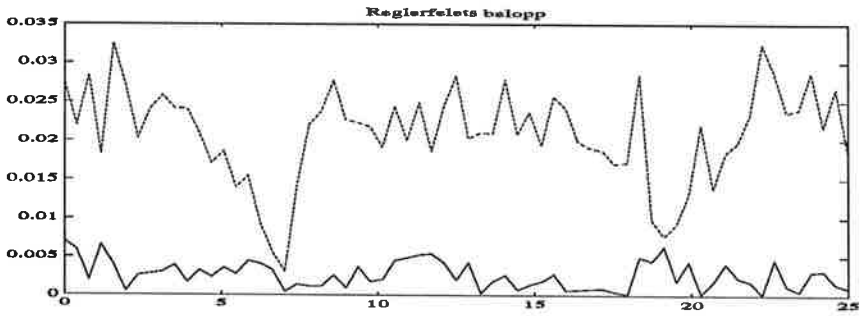
Tröghetsmoment	J	0.27
Tidskonstant	τ	1.0
Modellparameter	σ	9.0
Modellparameter	σ_1	4.24
Korrektionsfaktor	K	0.1
Friktionsfunktion	g	0.012
Stribeckfaktor	g_{num}	-0.002
Stribeckfaktor	g_{den}	0.002

Parametrarna i friktionsmodellen (3) σ och σ_1 är valda så att dämpningen i modellens karakteristiska polynom ska vara 0.7. De värden vi använt på σ och g är heller inte i överensstämmelse med den statiska friktionen F_c enligt (4). De värden vi använt ger en lägre produkt, högre värden verkade överkompensera friktionen. Anmärkningsvärt är att ett negativt värde på g_{num} gav bäst resultat. Detta innebär ju en lägre friktion vid låga hastigheter.

För att jämföra det kompenserade och okompenserade uppförandet gjorde vi experiment med att följa en sinuskurva med amplituden 0.5 och perioden 25 s. Vi använde dels rent proportionell reglering med måttlig förstärkning, dels PI-reglering. Båda dessa regulatorer utvärderades med kompenseringen från- och tillslagen. Figur 7 visar kompenserad P-reglering under en sinusperiod. Felen blir mycket små. För att kunna jämföra det okompenserade och det kompenserade uppförandet har vi dessutom plottat positionsfelet för P- och PI-reglering kontra kompenserade dito. Se figurerna 8 och 9. I figurerna kan man se att kompenseringen minskar absolutfelet med nästan 10 ggr vid proportionell reglering, vilket är anmärkningsvärt bra. Naturligtvis minskar förbättringen om man använder ett större K i den nominella regleringen, men då får man också kraftigt brus i styrsignalen vilket inte är bra för servot. Förbättringen är inte



Figur 7: Kompenserad P-reglering under en period.

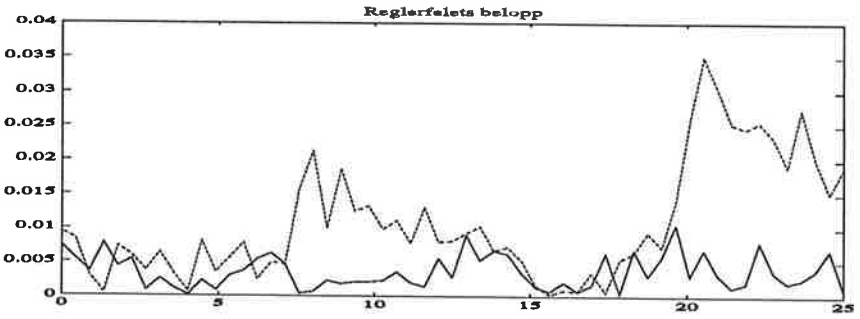


Figur 8: Positionsfelet vid P-reglering med och utan kompensering.

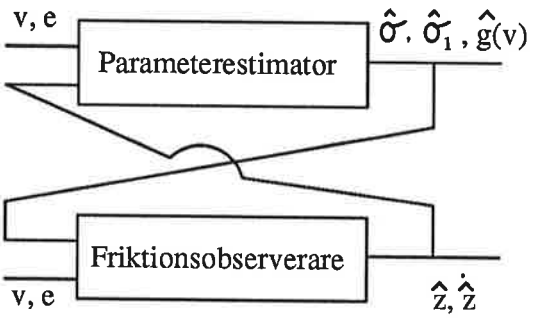
lika stor vid integrerande reglering men här lyckas kompenseringen helt ta bort det stora fel som uppstår då felet byter tecken.

7 Utvidgningar

I vår modell är parametrarna i friktionsobserveraren okända modellparametrar. För några av dessa har vi genom experiment fått en ungefärlig uppfattning om storlek och storleksförhållanden. Justeringar har vi sedan fått göra under regleringen. Parameteruppsättningar som fungerar bra vid en typ av experiment fungerar inte nödvändigtvis lika bra då experimentbetingelserna ändras. Dessutom är friktionen ett fenomen som är starkt föränderligt. Metalldelar slits, tillförseln av smörjmedel kan vara oregelbunden osv. En möjlig utvidgning är därför att adaptera på friktionsparametrarna. Insignalerna till estimatorn är utsignalerna från friktionsobserveraren, vinkelhastigheten och positionsfelet för



Figur 9: Positionsfelet vid PI-reglering med och utan kompensering.



Figur 10: Estimering av friktionsparametrar för friktionsobserveraren

servot. De skattade parametrarna är sedan insignaler till friktionsobserveraren, se figur 10. För att detta ska vara möjligt måste vi ha en bra modell av friktionsobserveraren från början, dvs bra startvärden för adapteringen.

Det går att visa att estimatorn under vissa vilkor är SPR, samt att friktionsobserveraren är passiv. Därmed borde det vara möjligt att konstruera en stabil adaptiv friktionskompensering med utgångspunkt från vår observerare.

Referenser

[1] P. R. Dahl. *Measurement of Solid Friction Parameters of Ball Bearings.* Proc. of 6th Annual Symp. on Incremental Motion Control Systems and Devices, University of Illinois, 1977.

[2] N. E. Leonard and P. S. Krishnaprasad. *Adaptive Friction Compensation for Bi-directional Low-velocity Position Tracking.* 31st IEEE Conf. on Decision and Control, December 1992.

[3] C. Canudas, K. J. Åström and K. Braun. *Adaptive Friction Compensation in DC Motor Drives.* CODEN: LUTFD2/(TFRT-7309)/1-21/1985. Department of Automatic Control. LTH, Lund, Sweden

[4] C. D. Walrath. *Adaptive Bearing Friction Compensation Based on Recent Knowledge of Dynamic Friction.* Automatica, Vol. 20, No. 6, pp. 717-727, 1984

[5] K. J. Åström and B. Wittenmark. *Adaptive Control.* Addison-Wesley, 1989.

[6] K. J. Åström and B. Wittenmark. *Computer-Controlled Systems: Theory and Design.* Prentice-Hall, 1990.

[7] L. Nielsen. *Computer Implementation of Control Systems.* ISRN: LUTFD2/TRFT–7476–SE. Department of Automatic Control. LTH, Lund, Sweden, 1992.

A Definitionsfiler

```
DEFINITION MODULE Opcom;

(*$NONSTANDARD*)

PROCEDURE Init( Prio1,Prio2 : CARDINAL );

PROCEDURE WaitForExit;

PROCEDURE Stop;

PROCEDURE Error( REF Errortext: ARRAY OF CHAR);
(* Writes Errortext on terminal *)

PROCEDURE FatalError( REF Errortext: ARRAY OF CHAR);
(* Writes Errortext on terminal, and causes Exit,
i.e system shut-down. *)

END Opcom.


DEFINITION MODULE CommandLoop;

(*$NONSTANDARD*)

TYPE
GetProc = PROCEDURE (VAR ARRAY OF CHAR);
PutProc = PROCEDURE (ARRAY OF CHAR);

CONST MaxLength=80;

PROCEDURE WaitForExit;

PROCEDURE InitNames;

PROCEDURE DoCommands (Read : GetProc; Write : PutProc);

PROCEDURE Terminate;

END CommandLoop.


DEFINITION MODULE Refgen;

(*$NONSTANDARD*)
```

```
CONST
    NrOfPar = 5;
    NameLength = 10;

TYPE
    ModeType = (Position,Square,Sine,EmergencyStop);
    NameType = ARRAY [0..NameLength] OF CHAR;
    ParNameType = ARRAY [1..NrOfPar] OF NameType;
    ParType = ARRAY [1..NrOfPar] OF LONGREAL;

PROCEDURE SetMode( REF NewMode : ModeType );
(* Changes the mode of the regulator. *)

PROCEDURE SetPos( REF NewPos : LONGREAL );
(* Requests motion to positin NewPos. *)

PROCEDURE GetParNames( VAR Names : ParNameType );
(* Gets the names of the parameters. *)

PROCEDURE GetPars( VAR Pars : ParType );
(* Gets the current values of the parameters. Notice
   that the indexing corresponds to the indexing of
   the parameter names when calling GetParNames. *)

PROCEDURE SetPars( REF NewPars : ParType );
(* Sets the entire set of parameters. *)

PROCEDURE Init( priority : CARDINAL );
(* Initializes Refgen. *)

PROCEDURE Stop;
(* Terminates Refgen. *)

PROCEDURE Test;

PROCEDURE TestResult ( VAR Loss : REAL );

END Refgen.


DEFINITION MODULE Regul;

(*$NONSTANDARD*)

CONST
    NrOfPars = 11;
    NameLength = 10;
```

```
TYPE
    ModeType = ( Position, EmergencyStop );
    NameType = ARRAY [0..NameLength] OF CHAR;
    ParNameType = ARRAY [1..NrOfPars] OF NameType;
    ParType = ARRAY [1..NrOfPars] OF LONGREAL;
    RefType = RECORD
                Mode          : ModeType;
                PosRef        : LONGREAL;
                PosRefDot     : LONGREAL;
                (*Derivative of PosRef*)
                PosRefDotDot : LONGREAL;
                (*Derivative of PosRefDot*)
              END;
    ErrorType = RECORD
                  e : LONGREAL; (* position error *)
                  V : LONGREAL; (* errorfunction *)
                END;

PROCEDURE GetParNames( VAR Names : ParNameType );
(* Returns the names of the parameters of the module
   in Names[1], ... Names[NrOfPars] *)

PROCEDURE GetPars( VAR Pars : ParType );
(* Returns the current parameter values in Pars[1], ...
   Pars[NrOfPars]. Notice that the indexing corresponds
   to the indexing of the parameter names when calling
   GetParNames *)

PROCEDURE SetPars( REF NewPars : ParType );
(* Sets all the parameters of the module to NewPars[1],...
   NewPars[NrOfPars] *)

PROCEDURE IntegratorReset;
(* Resets the integrator part of the controller *)

PROCEDURE SetRef( REF NewRef : RefType );
(* Sets the control mode to NewRef.Mode, the position
   reference to NewRef.PosRef, the velocity of the
   reference to NewRef.PosRefDot and the acceleration
   of the reference to NewRef.PosRefDotDot *)

PROCEDURE GetPos() : LONGREAL;
(* Returns the current position of the servo *)

PROCEDURE Init(priority:CARDINAL);
(* Initializes Regul *)
```

```
PROCEDURE Stop;
(* Terminates Regul *)

PROCEDURE GetError( VAR e : ErrorType );
(* Returns the current positionerror and value of the
   errorfunction *)

PROCEDURE ResetError;
(* Resets the errorfunction *)

PROCEDURE FricComp (FcompOn : BOOLEAN );
(* Sets the control to friction compensation on / off *)

END Regul.


DEFINITION MODULE Filter;

TYPE
     FilterType;

PROCEDURE Seth (h : LONGREAL; f : FilterType);
(* Sets the sampling interval to h *)

PROCEDURE SetTau (Tau : LONGREAL; f : FilterType);
(* Sets the filter time constant *)

PROCEDURE GetTau (VAR Tau : LONGREAL; f : FilterType);
(* Gets the filter time constant *)

PROCEDURE InitFilter (VAR f : FilterType);
(* Initiates a filter *)

PROCEDURE DisposeFilter (VAR f : FilterType);
(* Disposes a filter *)

PROCEDURE Next (u : LONGREAL; f : FilterType) : LONGREAL;
(* Returns the next filtered value, provided that
   u is the current unfiltered value.              *)

END Filter.


DEFINITION MODULE Fcomp;

(*$NONSTANDARD*)
```

```
CONST
  NrOfPar      = 7;
  NameLength   = 10;

TYPE
  NameType     = ARRAY[0..NameLength] OF CHAR;
  ParNameType  = ARRAY[1..NrOfPar]    OF NameType;
  ParType      = ARRAY[1..NrOfPar]    OF LONGREAL;


PROCEDURE SetPars (REF NewPars : ParType);
(* Sets the entire sets of parameters *)

PROCEDURE ResetFcomp;
(* Resets the values of the friction-observer.
   FObserve will start with 0 *)

PROCEDURE GetPars (VAR Pars : ParType);
(* Gets the current values of the parameters *)

PROCEDURE GetParNames (VAR Names : ParNameType);
(* Gets the names of the parameters *)

PROCEDURE Seth (h : LONGREAL);
(* Sets the sampling interval to h sec. *)

PROCEDURE FObserve (fi , fiprim : LONGREAL) : LONGREAL;
(* Returns an estimate of the friction-force.
   fi is the current angle-error and
   fiprim the current angular velocity *)

PROCEDURE Init;
(* Initializes Fcomp *)

END Fcomp.


DEFINITION MODULE Plot;

(*$NONSTANDARD*)

TYPE RecType = RECORD
                 Time,
                 Position,
                 Velocity,
                 PositionReference,
```

```
                Control,
                Fcomp,
                FiltVel,
                PosErr : LONGREAL;
              END;

PROCEDURE Init(Priority:CARDINAL);
(* Initiates the Plot-module *)

PROCEDURE PutData(Element:RecType);
(* Receives data to be plotted *)

PROCEDURE Stop;
(* Stops the Plot-module *)

END Plot.


DEFINITION MODULE RingBuffer;

TYPE Buffer;
     DataType = ARRAY [1..8] OF LONGREAL;

PROCEDURE PutData(data: DataType; B: Buffer);
(* Inserts ch into B *)

PROCEDURE GetData(VAR data: DataType; B: Buffer);
(* Returns ch from B *)

PROCEDURE NewBuffer(VAR B: Buffer);
(* Creates a new buffer by calling NEW and initializing
   the buffer *)

PROCEDURE DisposeBuffer(VAR B: Buffer);
(* Deletes the buffer by calling DISPOSE *)

END RingBuffer.
```

ADAPTIV FRIKTIONSKOMPENSERING ETT PROJEKT I ADAPTIV REGLERING OCH REALTIDSSYSTEM

Magnus Ericsson
Frans Hermodsson
Mats Larsson
Patrik Östberg

14 maj 1993

Innehåll

1 Inledning

Uppgiften bestod i att implementera en adaptiv regulator för friktionskompensering på ett servo. Som stöd har vi använt oss av artikeln " Adaptive Friction Compensation Applied to Positioning of a Space-Station Solar Array ", skriven av Tao, Koktovic' och Ianculescu.

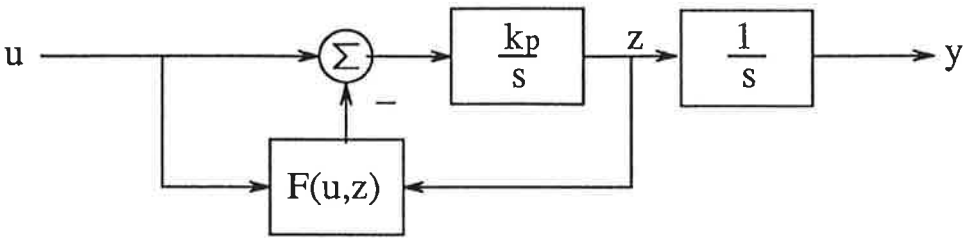
Implementeringen har gjorts på PC-datorer i reglertekniks kurslab. Som stöd för att bygga upp användarinterfacet har vi bl a använt oss av de de rutiner som finns beskrivna under Modules specific to IBM PC i läroboken Computer Implementation of Control System skriven av Lars Nielsen. Vi har satsat på att göra ett enkelt och funktionellt användargränssnitt. Här finns möjlighet att välja mellan signalplot och plot över adaptionsparametrarna. Man kan välja mellan tre olika referenssignaler samt på ett enkelt sätt sätta amplitud och frekvens på signalen under körning. Programmet kan köras i tre moder, Off, CompensationOff samt CompensationOn. I Off reglerar vi inte alls, medan vi i CompensationOff kör en RST regulator byggd på att servot har följande överföringsfunktion, G(s)=$\frac{3.5}{s(s+1.7)}$. I CompensationOn lägger vi till den adaptiva friktionskompenseringen samt kör med en RST regulator byggd på att det kompenserade servot är en dubbelintegrator med en statisk förstärkning på 3.5. Vi designar alltå RST regulatorn utifrån modellen $G_m(s) = \frac{3.5}{s^2}$, de linjära delar som finns i servot har vi lyft ut och placerat i friktionsmodellen. Eftersom vi inte känner storleken på de komponenter som ingår i friktionsmodellen använder vi oss av en adaptiv regulator för att skatta dessa. När det gäller att analysera resultatet har vi valt att inte implementera några hjälpmedel för analys i programmet, istället har vi implementerat en funktion för att logga ut alla relevanta signaler på en fil i MatLabformat. I MatLab kan man sedan utnyttja alla möjligheter för en noggrann analys av regleringsresultatet.

En användarbeskrivning för programmet finns beskriven längre fram i rapporten.

Målet med projektet var att se om vi kunde få en bättre reglering med en adaptiv regulator, byggd på den friktionsmodell man återfinner i ovan nämnda artikel, jämfört med en RST regulator byggd på polplacering. Vi ville också på något sätt kunna mäta en eventuell förbättring.

2 Teori

2.1 Processmodell

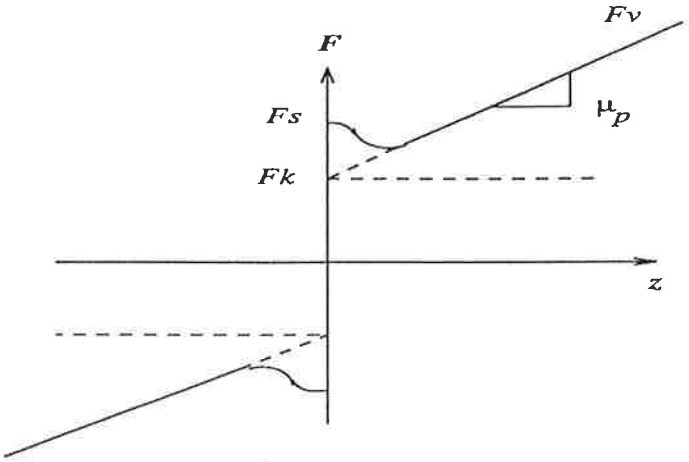


Figur 1: Processmodellen som används. $F(u, z)$ modellerar friktionen i servot. u är styrsignal till servot och z servots rotationshastighet.

I modellen ansätts friktionen som en olinjär additiv störning. Denna gör vi sedan ett försök att skatta och kompensera för med hjälp av ett MRAS. I $F(u, z)$ kan eventuellt en del av processens dynamik ingå. In- utsignalkarakteristiken för servot ges av

$$y(t) = \frac{k_p}{s^2}[u(t) - F(u(t), z(t))] \tag{1}$$

2.2 Friktionsmodell



Figur 2: Friktionsmodellen som används. $F(u,z)$ modellerar friktionen i servot. u är styrsignal till servot och z servots rotationshastighet.

Vår friktionsmodell visas i figur 2.2. Friktionsmodellen tar hänsyn till kinetisk friktion F_k, viskös friktion F_v, statisk friktion F_s och Stribeck-friktionen $(F_s - F_k)e^{-(\frac{z}{\beta_s(z)})^2}$ och kan skrivas enligt :

$$F(u,z) = \begin{cases} F_k(z) + F_v(z) + (F_s(z) - F_k(z))e^{-(\frac{z}{\beta_s(z)})^2} & z \neq 0 \\ F_s(u) & z = 0 \end{cases} \tag{2}$$

där

$$F_k(z) = \begin{cases} -\theta_1 & z < 0 \\ \theta_4 & z > 0 \end{cases}$$

$$F_v(z) = \begin{cases} \theta_2 z & z < 0 \\ \theta_5 z & z > 0 \end{cases}$$

$$\beta_s(z) = \begin{cases} \beta_{sn} & z < 0 \\ \beta_{sp} & z > 0 \end{cases}$$

$$F_s(z) = \begin{cases} -\theta_3 & z < 0 \\ \theta_6 & z > 0 \end{cases}$$

Vi parametriserar 2 enligt :

$$F(u,z) = \varphi^T(t)\Theta(t) \tag{3}$$

där

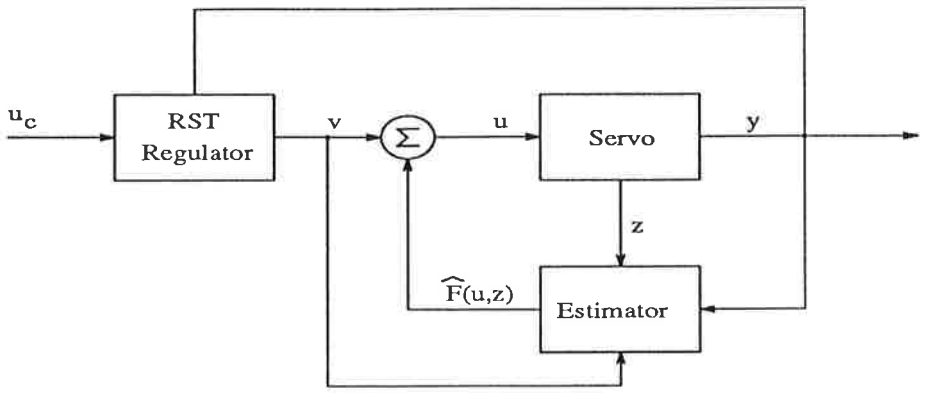
$$\varphi^T(t,z) = \begin{cases} (e^{-(\frac{z}{\beta_s(z)})^2} - 1, z, -e^{-(\frac{z}{\beta_s(z)})^2}, 0, 0, 0) & z < 0 \\ (0,0,0,1 - e^{-(\frac{z}{\beta_s(z)})^2}, z, e^{-(\frac{z}{\beta_s(z)})^2}) & z > 0 \\ (0,0,-1,0,0,0) & z = 0, u < 0 \\ (0,0,0,0,0,1) & z = 0, u > 0 \end{cases}$$

och

$$\Theta(t) = (\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6)$$

Parametrarna β_{sn} och β_{sp} antas vara kända medan Θ estimeras rekursivt.

2.3 Design av fix RST-regulator



Figur 3: Blockschema över slutna systemet.

Styrsignalen till servot sätts som

$$u(t) = v(t) + \hat{F}(u(t), z(t))$$

vilket kombinerat med ekvation 1 blir

$$y(t) = \frac{k_p}{s^2}[v(t) + \hat{F}(u(t), z(t)) - F(u(t), z(t))]$$

Om vi förutsätter att estimatorn skattar friktionen i servot perfekt, dvs. $\hat{F}(u(t), z(t)) = F(u(t), z(t))$, kommer det friktionkompenserade servot alltså att uppföra sig som en ren dubbelintegrator med överföringsfunktionen $G(s) \doteq \frac{k_p}{s^2}$. En fix RST-regulator baserad på denna processmodell bör då fungera utmärkt.

Det önskade slutna systemet ansätts som $G_m(s) = \frac{\omega^2}{s^2+2\omega\zeta s+\omega^2}$.

då får vi :

$$\begin{array}{lll} B(s) = k_p & B_m(s) = \omega^2 & A_o(s) = s + \lambda \\ A(s) = s^2 & A_m(s) = s^2 + 2\omega\zeta s + \omega^2 & \\ R(s) = s + r_1 & S(s) = s_0 s + s_1 & T(s) = \frac{A_o(s)B_m(s)}{B(s)} = t_0 A_o(s) \end{array}$$

Diofantiska ekvationen ger :

$$AR + BS = A_o A_m \Leftrightarrow s^2 (s + r_1) + k_p \, (s_0 \, s + s_1) = (s + \lambda) \left(s^2 + 2\,\omega\,\zeta\,s + \omega^2\right) \Rightarrow$$

$$r_1 = \lambda + 2\,\omega\,\zeta \quad s_0 = \frac{2\,\lambda\,\omega\,\zeta+\omega^2}{k_p} \quad s_1 = \frac{\lambda\,\omega^2}{k_p} \quad t_0 = \frac{\omega^2}{k_p}$$

Vilket ger styrlagen

$$Rv = -Sy + Tu_c \Leftrightarrow v = \frac{t_0 \, (s + \lambda) \, u_c - (s_0 \, s + s_1) \, y}{s + r_1} \tag{4}$$

2.4 Estimering av parametrar

För att driva vårat MRAS bildar vi felsignalen

$$e(t) = y(t) - y_m(t) = \frac{BT}{AR+BS}u_c + \frac{BR}{AR+BS}(\hat{F} - F) - y_m =$$

$$\frac{B_m}{A_m}u_c + \frac{BR}{A_0 A_m}(\hat{F} - F) - \frac{B_m}{A_m}u_c =$$

$$H(p)[\varphi^T(t)(\hat{\Theta}(t) - \Theta_0)] \Rightarrow H(p) = \frac{B(p)R(p)}{A_o(p)A_m(p)}$$

vidare bildar vi

$$\xi(t) = H(p)[\varphi^T(t)]\hat{\Theta}(t) - H(p)[\varphi^T(t)\hat{\Theta}(t)]$$

och

$$\epsilon(t) = e(t) + \xi(t)$$

Parameterskattningarna $\hat{\Theta}(t)$ uppdateras enligt

$$\dot{\hat{\Theta}}(t) = -\frac{\Gamma H(p)[\varphi^T(t)]\epsilon(t)}{1 + H(p)[\varphi^T(t)]H(p)[\varphi(t)] + \xi^2(t)}$$

Detta är uppdateringslag baserad på SPR-regeln.

$$\Gamma = \begin{bmatrix} \gamma_1 & & & & & \\ & \gamma_2 & & & & \\ & & \gamma_3 & & & \\ & & & \gamma_4 & & \\ & & & & \gamma_5 & \\ & & & & & \gamma_6 \end{bmatrix},$$

2.5 Approximation av derivator

Eftersom designen är gjord i kontinuerlig tid måste derivatorna i styrlagen och uppdateringslagen approximeras. Detta gör vi med hjälp av trapetsmetoden, dvs. substitution av $s = \frac{2}{h}\frac{z-1}{z+1}$ i våra kontinuerliga styr- och uppdateringslagar.

3 Experiment

Vi utförde först fyra experiment där vi beräknade medelkvadratfelet. I de fyra olika fallen användes

$$G_m(s) = \frac{3.0^2}{s^2 + 2 * 0.8 * 3.0s + 3.0^2}$$

och

$$A_o = s + 4.5$$

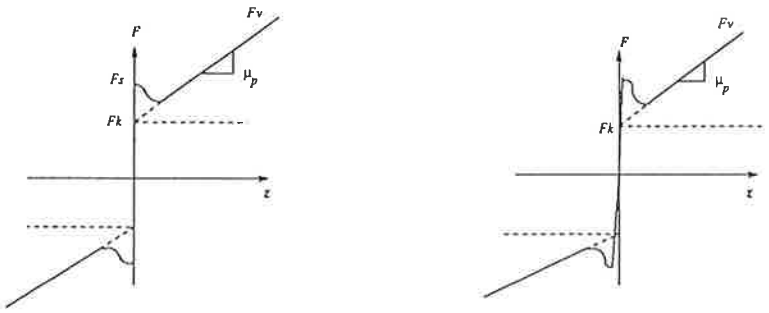
Referenssignalen som användes var fykantsvåg med amplituden 0.5 och frekvensen 0.5 rad/s. De fyra experimenten utfördes enligt följande

1. I det första fallet reglerade vi med friktionskompensering enligt artikeln. Samplingsintervallet $h = 0.08$ användes. Regleringen uppförde sig lite ryckigt när hastigheten var nära noll. Detta uppförandet var väntat eftersom friktionskompenseringen är diskontinuerlig för hast z=0. I övrigt följer utsignalen den önskade mycket bra. Medelkvadratfelet beräknat över tio perioder blev 0.00031 i detta fallet.

2. Detta experiment utfördes enligt första fallet men med samplingsintervallet $h = 0.02$. Med detta samplingsintervallet blev följningen mycket sämre än i första fallet. Detta berodde troligen på att estimationen av parametrarna inte fungerar. Medelkvadratfelet blev nu 0.0025.

3. Det tredje experimentet utfördes med en vanlig RST-regulator men med en systemmodell som $\frac{k_p}{s(s+b)}$ där vi valde $b = 1.7$ med hjälp av resultat i de två första experimenten. Vi fick en ganska bra följning. Medelkvadratfelet blev 0.0053.

4. Experiment fyra utfördes enligt fall ett men med en modifierad friktionsmodell. Istället för att sätta friktionskompenseringen till $F_s(v)$ vid låg hastihet sattes denna till noll. Följningen blev nu jämnare vid låg hastighet men lite sämre. Medelkvadratfelet blev nu 0.0015.



Figur 4: Till vänster friktionsmodellen enligt artikeln och till höger den modifierade.

Vi utförde ytterligare ett experiment för att studera hur parametrarna svängde in sig. I detta fallet använde vi oss av en sinusformad referenssignal.

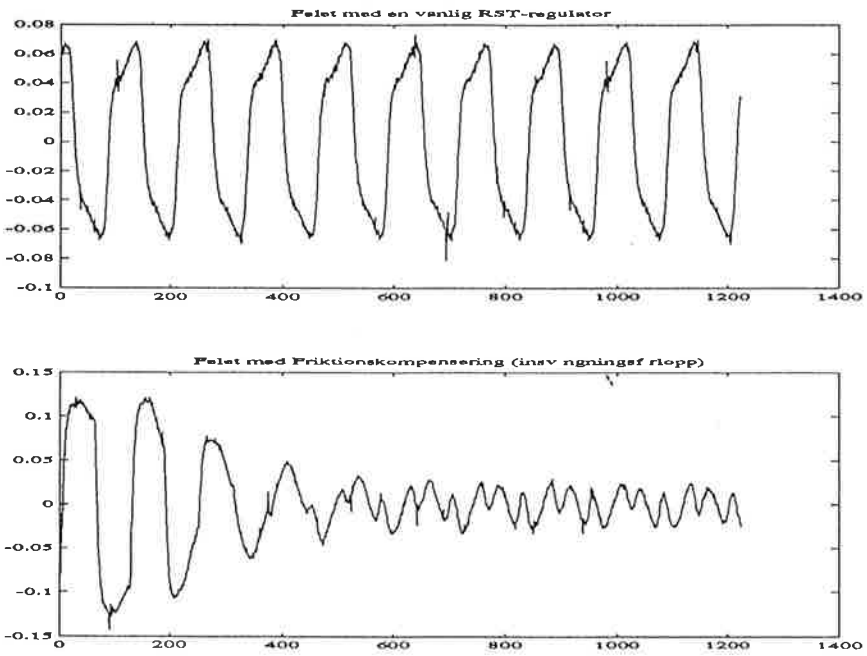
Vi satte

$$\Theta(0) = \begin{pmatrix} 0.2 & 2.0 & 0.2 & 0.2 & 2.0 & 0.2 \end{pmatrix}$$

och

$$\Gamma = \begin{pmatrix} 1.0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3.0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 \end{pmatrix}$$

Vi fick resultatet enligt figur 5.



Figur 5: Felet vid parameterinsvängning och felet vid vanlig RST

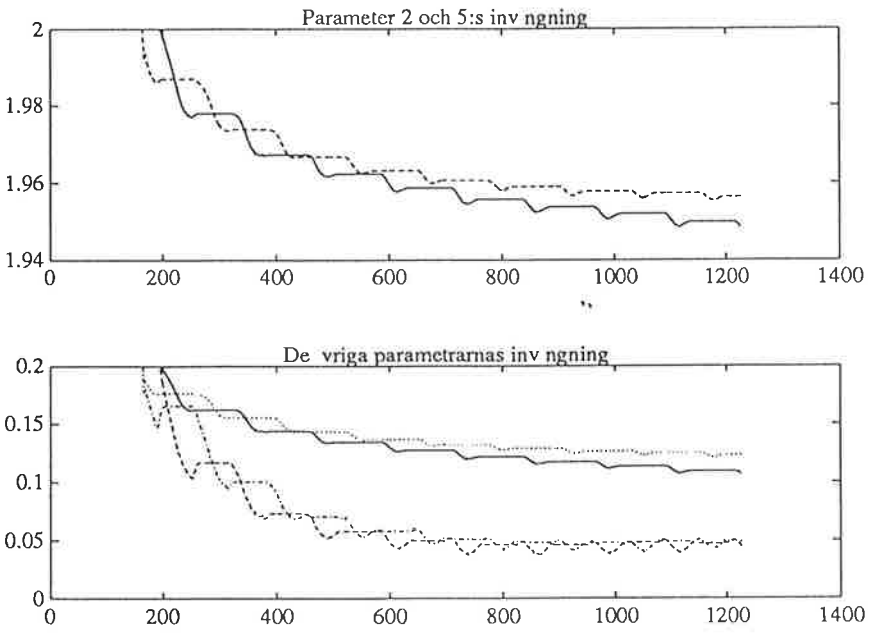
3.1 Resultat och Slutsatser

Under experimenten kom vi fram till att $K_p = 3.5$. Valet av denna parameter var mycket viktigt för resultatet. β_{sn} och β_{sp} satte vi till 0.1. Hur dessa sattes var inte så kritiskt. När parametrarna svängt in sig blev

$$\Theta = \begin{pmatrix} 0.107 & 1.95 & 0.0447 & 0.123 & 1.96 & 0.0492 \end{pmatrix}$$

Man kan notera assymmetrin mellan parameter 1 och 4
Medelkvadratfelet för de olika regleralgoritmerna blev:

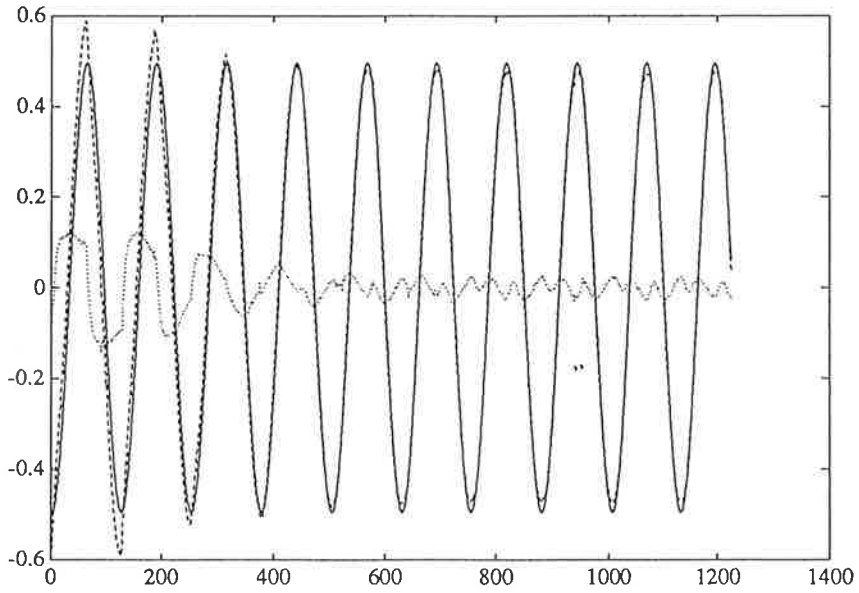
regleralgoritm	samplingsint	medelkvadratfel
frik.	0.08	0.00031
frik.	0.02	0.0025
vanlig RST	0.08	0.0053
mod. frik.	0.08	0.0015



Figur 6: Parametrarnas invängning.

Försöken visar att:

- Regulatorn med friktionskompensering ger bättre följning än en vanlig RST-regulator. Se figur 2.
- Samplingsintervallet kan inte väljas för litet eftersom då fungerar estimeringen ej. I vårt fall fick vi ingen parameterkonvergens för samplingsintervall mindre än 0.08.
- Friktionsmodellen enligt artikeln ger bättre följning än den modifierade. Dock är den modifierade mindre ryckig vid låga hastigheter.
- Parametrarna i Θ vektorn svänger in sig på tre perioder enligt figur refinsvang.



Figur 7: y_m - heldragen kurva, y - streckad, e - prickad

4 Operatörskommunikation

4.1 Handhavande

Vid uppstart av programmet får användaren en fråga om data skall loggas, och i så fall under vilket filnamn. Data består av signal och parametervärden som loggas var 10s till vald fil. Processen startas genom att klicka med musen på off.

1. Plotfönstret plottar Ym, Y, e, u, F och Z, då SignalPlot figur 8 är vald. Då parameterPlot figur 9 är vald plottas R,S och T parametrarna. Dessa parametrar kan väljas bort eller fram genom ett klick med musen i respektive fönster.

   Ym Modellens utsignal.

   Y Processens utsignal.

   e = Ym-Y.

   u Motorns insignal = RST-regulatorns + friktionskompenseringens utsignal.

   F Den skattade friktionen.

   Z Motorns hastighet.

2. Off/CompOn/CompOff. CompOn/CompOff kopplar in/ut friktionskompenseringen. I läge Off försvinner plotfönstret. Knappen tyrs via musen.

3. Sine/Ramp/Square ger referenssignalens form, styrs via musen.

4. Diverse viktiga parametrar.

   Omega och Zeta är modellparametrar som ger användaren möjlighet att välja modellens utseende med avseende på resonasfrekvens och dämpning. Och på så sätt påverka reulatorns egenskaper.

   Lambda är obseverarens pol som påverkar estimeringens konvergenshastighet.

   Gamma viktar det nya estimatet med exponentiellt dämpade gamla värden.

   h är samplingshastighet.

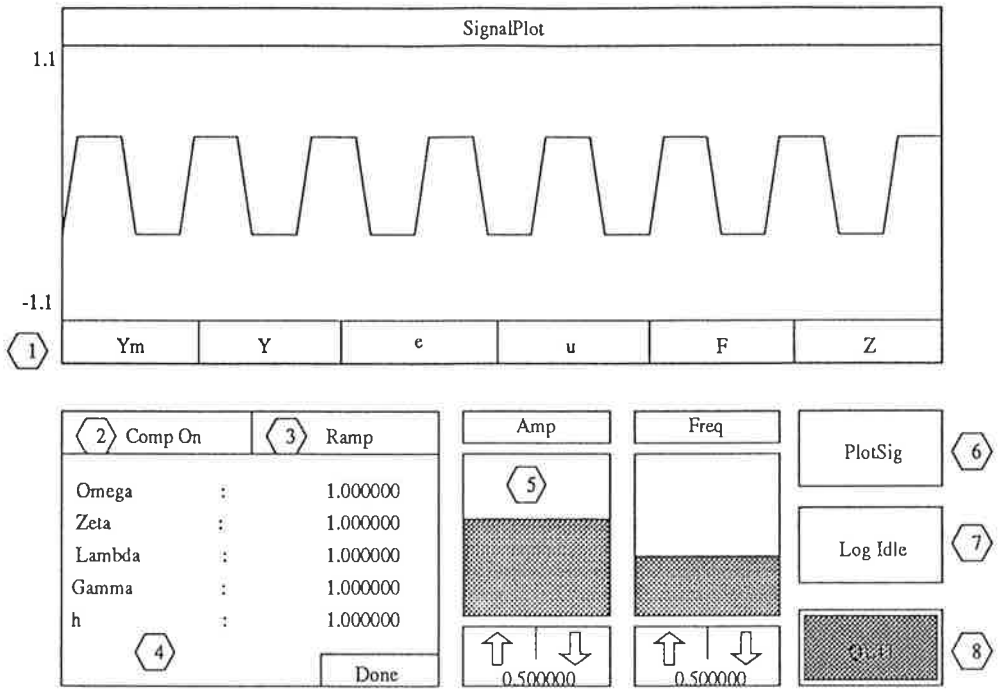
   Man ändrar dessa värden genom att klicka på de värden eller det värde som skall ändras. Skriver in de/det via tangentbordet och trycker på return. När alla ändringar är gjorda klickar man i Done med musen.

5. Referenssignalens egenskaper kan ändras med avseende på amplitud (Amp) och frekvens (Freq). Antingen genom att grafiskt styra stolpdiagrammet. Genom att sätta pilen på önskad höjd och klicka till. Eller klicka på pilarna för ändring i fasta steg av värdet. Eller numeriskt genom att klicka på värdet och skriva in ett nytt och trycka på return.

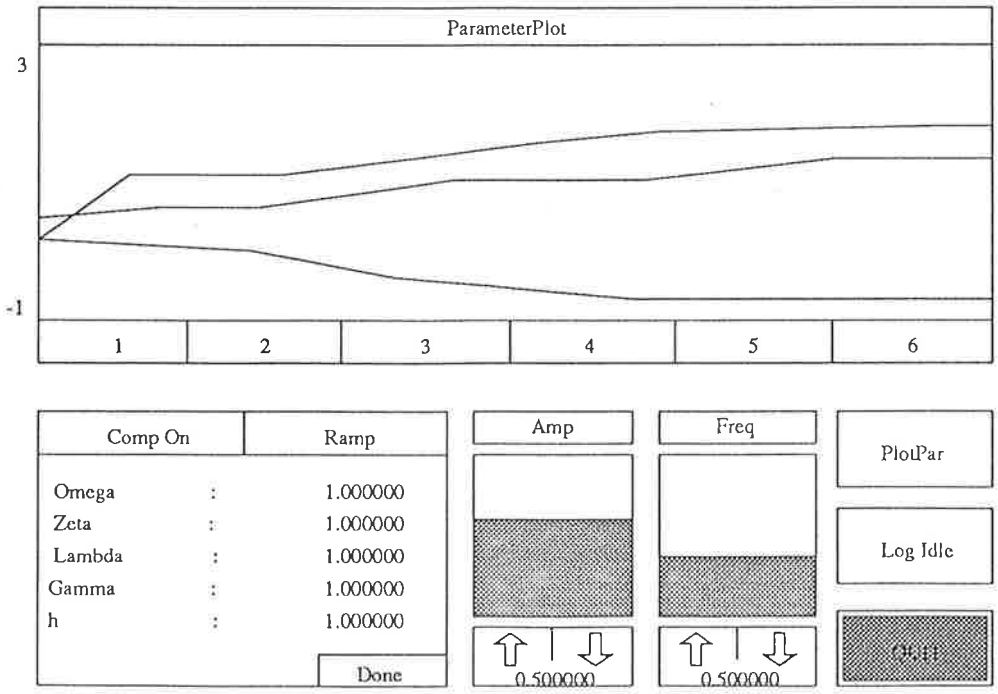
6. PlotSig/PlotPar ger olika plotfönster, antingen SignalPlot eller ParameterPlot.

7. LogIdle/LoggerRunning används för att ej logga/logga värden till tidigare vald fil. Om man går ur loggerRunning respektive loggar flera gånger under körning skrivs inte de gamla värdena över så länge man inte går ur programmet.

8. Quit stoppar programmet.



Figur 8: användargränssnitt SignalPlot



Figur 9: användargränssnitt ParameterPlot

5 Sammanfattning

Målet med hela projektet var att implementera en adaptiv regulator för friktionskompensering byggd på en given friktions modell, som står beskriven i teoriavsnittet längre fram i rapporten.

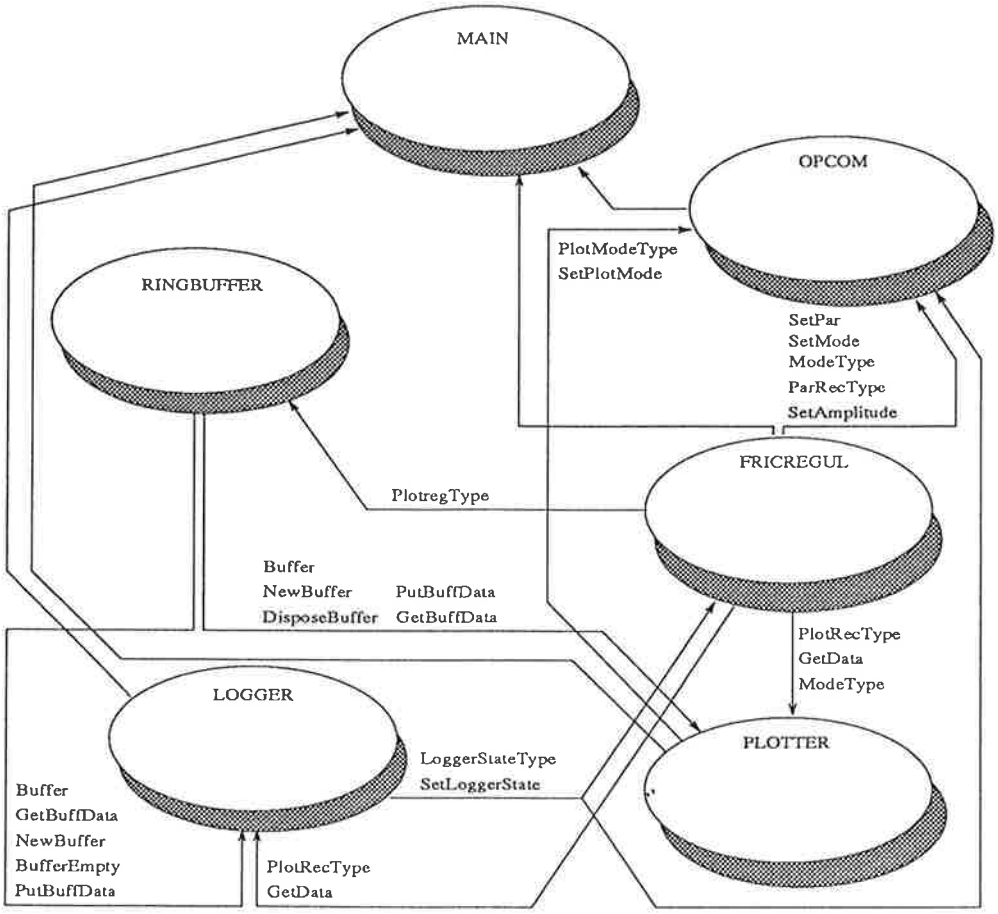
Implementeringen gick smärtfritt, däremot fick vi en hel del problem med själva trimmningen av regulatorn. Det visade sig att vi var tvugna att känna förstärkningen i servot ganska väl för att kunna få en hygglig reglering. Valet av samplingsintervall är också mycket kritiskt för att åstkomma en bra reglering, här får man göra en kompromiss mellan bra reglering med snabb sampling och bra estimation av adaptionsparametrar med långsammare samlingshastighet. När väl den adaptiva regulatorn var intrimmad visade det sig att vi fick nästan 20 gånger mindre medelkvadratfel när vi körde denna jämfört med då vi körde den vanliga RST regulatorn, byggd på att servot har överföringsfunktionen $G(s)=\frac{3.5}{s(s+1.7)}$. Överföringsfunktionen stämmer nog ganska väl överens med verkligheten och i så fall skulle jämförelsen vara rättvis.

En förbättring som skulle kunna göras för att göra programmet mer generellt är att modularisera reglermodulen. Denna skulle då bestå av en modul med konventionella regulatorer och en modul för adaption och estimation. Andra utbyggnader som ligger nära till hands är att införa någon form av clear funktion som rensar plot fönstret. Enkla funktioner för analys av regleringsresultatet skulle också vara önskvärt, t ex en funktion för att räkna ut medelkvadratfelet. Detta skulle möjliggöra en snabbare men inte lika noggrann analys som den i Matlab.

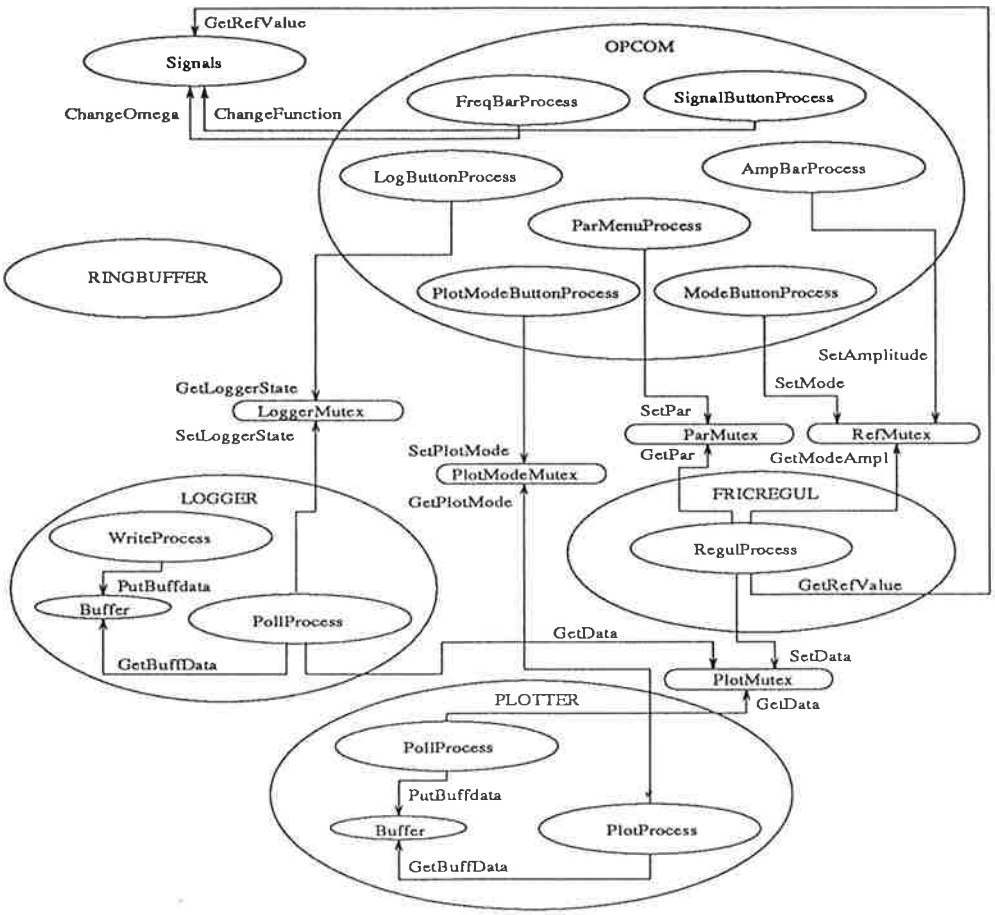
6 Avslutning

Vi vill härmed tacka våra handledare Henrik Olsson och Mats Andersson för all hjälp och ett trevligt projekt.

A Modulgraf


MAIN
OPCOM
RINGBUFFER
FRICREGUL
LOGGER
PLOTTER
PlotModeType
SetPlotMode
SetPar
SetMode
ModeType
ParRecType
SetAmplitude
PlotregType
Buffer
NewBuffer
DisposeBuffer
PutBuffData
GetBuffData
PlotRecType
GetData
ModeType
LoggerStateType
SetLoggerState
Buffer
GetBuffData
NewBuffer
BufferEmpty
PutBuffData
PlotRecType
GetData

B Processgraf



C Definitionsmoduler

C.1 Regulatorn

```
DEFINITION MODULE FricRegul;

TYPE
  VectorType = ARRAY[1..6] OF REAL;

  ModeType    = (Off, Compensation_Off, Compensation_On);
  (* De olika driftsfallen : Avstängd, Friktionskompensering från resp.
  tillslagen *)

  ParRecType  = RECORD
                  Lambda, Zeta, Omega, Gamma,h : REAL;
                END;
  (* Parametrar som styr det önskade systemets egenskaper :
     Observerarpol, dämpning, egenfrekvens, uppdateringsförstärkning och
     sampeltid *)

  PlotRecType = RECORD
                   Mode : ModeType;
                   uc, y, e, u, F, z : REAL;
   Theta : VectorType;
                END;
  (* Data som exporteras till Logger och Plotter *)

  PROCEDURE Init(Priority: CARDINAL);
  (* Initierar regulatorn, regulatorn startar i driftsfallet 'Off' *)

  PROCEDURE SetMode (Mode : ModeType);
  (* Byter mellan regulatorns driftsfall *)

  PROCEDURE SetAmplitude(Amplitude : REAL);
  (* Sätter amplituden på den av 'Signals' genererade referenssignalen *)

  PROCEDURE SetPar (ParRec : ParRecType);
  (* Ändrar driftsparametrar och tillhörande filterkonstanter räknas ut *)

  PROCEDURE GetData (VAR PlotRec: PlotRecType);
  (* Hämtar data för loggning och plottning *)

  PROCEDURE Stop;
  (* Stoppar regulatorn *)
END FricRegul.
```

C.2 Mätvärdesloggern

```
DEFINITION MODULE Logger;

TYPE
  LoggerStateType = (Running, Idle);
  (* Loggerns möjliga tillstånd  *)

PROCEDURE SetLoggerState(State : LoggerStateType; TimeStep : REAL);
(* Ändrar loggerns tillsånd och sätter tidsavstånd mellan loggade värden *)

PROCEDURE GetLoggerState(VAR TS : REAL) : LoggerStateType;
(* Ger loggerns tillsånd och tidsavstånd mellan loggade värden *)

PROCEDURE Init(LogPriority, PollPriority : CARDINAL);
(* Initierar Logger. (PollingPriority skall vara mindre än  LogPriority) *)

PROCEDURE Stop();
(* Stoppar Logger *)

END Logger.
```

C.3 Mätvärdesplotter

```
DEFINITION MODULE Plotter;
TYPE
  PlotModeType = (PlotPar, PlotSig);

  PROCEDURE Init (PollingPriority, PlotPriority: CARDINAL);
  (* Initierar Plotter. (PollingPriority skall vara mindre än  PlotPriority) *)

  PROCEDURE SetPlotMode (PlotMode : PlotModeType);
  (* Byter mellan plottning av parametrar och signaler *)

  PROCEDURE Stop;
  (* Stoppar plotter *)

END Plotter.
```

C.4 Operatörskommunikation

```
DEFINITION MODULE OpCom;

VAR
  RefSigName : ARRAY[0..7] OF CHAR;

  PROCEDURE Init (Priority: CARDINAL);
  (* Öppnar grafikfönstret och startar operatörskommunikationen *)

  PROCEDURE WaitForQuit;
  (* Anropande process fördröjs tills Quit-knappen väljs *)

  PROCEDURE Stop;
  (* Stoppar operatörskommunikationen och stänger grafikfönstret *)
END OpCom.
```

C.5 Ringbuffert

```
DEFINITION MODULE RingBuffer;

FROM FricRegul IMPORT PlotRecType;

TYPE
  Buffer;

PROCEDURE BufferEmpty(B : Buffer):BOOLEAN;
(* Kollar ifall bufferten är tom, returnera TRUE vid tom buffert *)

PROCEDURE PutBuffData (ch : PlotRecType; B : Buffer);
(* Lägger data i bufferten, anropande process fördröjs om bufferten är full *)

PROCEDURE GetBuffData (VAR ch : PlotRecType; B : Buffer);
(* Hämtar data från bufferten, anropande process fördröjs om bufferten är tom *)

PROCEDURE NewBuffer(VAR B:Buffer);
(* Skapar och initierar tom buffert *)

PROCEDURE DisposeBuffer(VAR B:Buffer);
(* Tömmer och förstör buffert *)

END RingBuffer.
```

SKVALPPROCESSEN

ETT PROJEKT I ADAPTIV REGLERING och REALTIDSSYSTEM

Tor Göransson E-88
Henrik Nilsson E-88
Johan Lindberg E-88
Jonas Fors E-88

Handledare: Ola Dahl, Ulf Jönsson
Institutionen för Reglerteknik, Lunds Tekniska Högskola

3 maj 1993

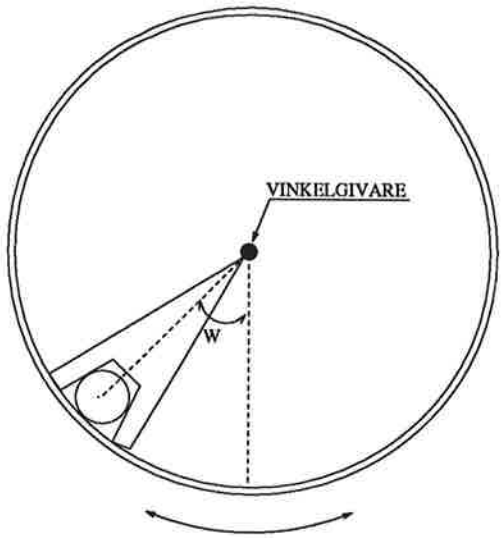
Innehåll

1 Inledning

Detta projekt utfördes under 3 veckor våren 1993 som ett samprojekt mellan kurserna adaptiv reglering och realtidssystem. Vi valde den så kallade skvalpprocessen för att den verkade intressant och för att man kunde se tillämpningar i verkligheten. Rapporten är indelad i 4 delar i vilka man kan följa hur projektet framskridit. Vi börjar med att beskriva processen för att få den fysikaliska bakgrunden till projektet. Fortsättningsvis behandlar vi den programmeringstekniska biten för att visa hur vi löste de realtidsmässiga aspekterna. Därefter behandlas den reglertekniska biten: först enkel PID-reglering och sedan adaptiv reglering där vi först behandlar teori och simuleringar, sedan realisering på den verkliga processen och resultat. Slutligen en användarmanual som beskriver hur programmet exekveras.

2 Beskrivning av processen

Processen vi valde går under många namn. Vi valde att kalla den "skvalpprocessen" eftersom det finns klara paralleller med en båt som gungar på havet, men även namn som "ekorrhjulet" och "ball and hoop" förekommer. Processen består av ett svänghjul, som är monterat på en axel från en motor (se figur 1). På axeln är även en arm med en vinkelgivare monterad. Armens uppgift är att hålla kvar kulan inuti svänghjulet samt att ge kulans position. Reglerproblemet blir att försöka reglera ner kulan så snabbt som möjligt till nolläget efter att man puttat till kulan.



Figur 1: Skiss på processen.

Som hjälp för att modellera processen fick vi ta del av rapporten ekorrhjulet, vilken utarbetats i kursen processidentifiering. Då vi själva är måttligt insatta i identifiering, valde vi den till synes enklaste modellen av processen (armax-modellen):

$$G_m(s) = \frac{ks}{s^3 + a_1 s^2 + a_2 s + a_3}$$

I det frekvensområde som vi är intresserade av så beskrivs processen mycket väl av ett andra ordningens system. Detta gör vi för att reducera antalet parametrar i regulatorn och för att förenkla programmeringsarbetet. Den förenklade modellen blir:

$$G_m(s) = \frac{ks}{s^2 + a_1 s + a_2}$$

Den samplade modellen får utseendet:

$$H_m(z) = \frac{k(z-1)}{z^2 + a_1 z + a_2}$$

3 Problemlösning

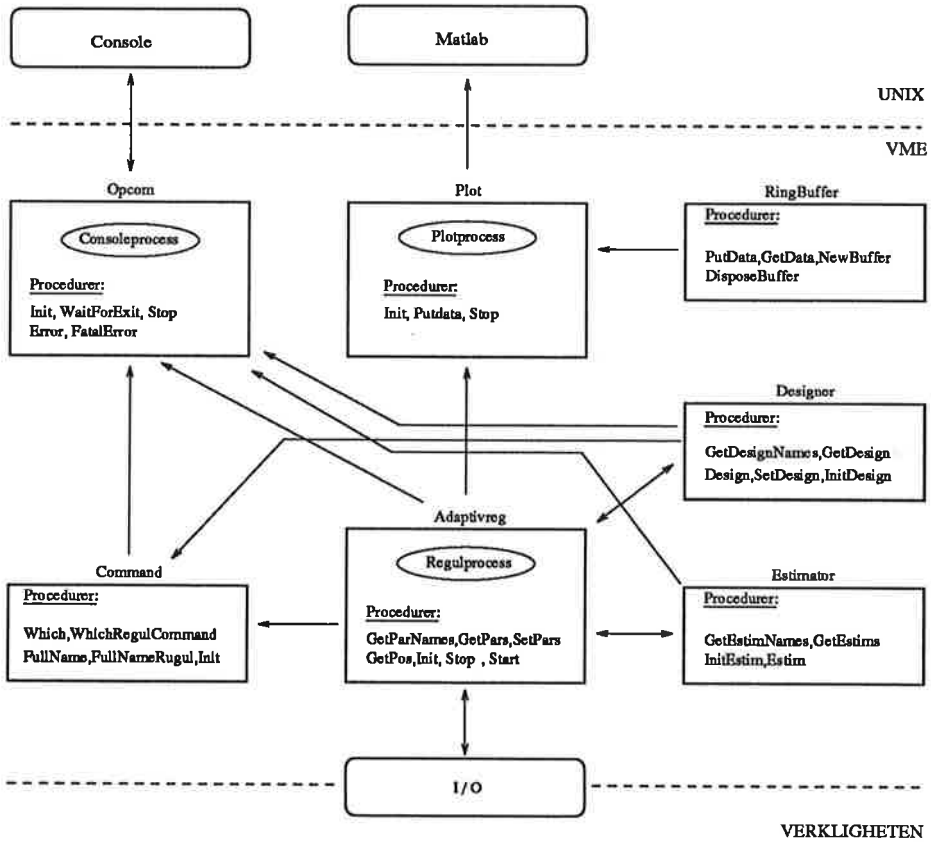
3.1 Inledning

I detta avsnitt behandlas projektets tyngdpunkt, det vill säga själva regleringen av processen. Avsnittet är uppdelat i 3 delar:

- Realtid: Här behandlas de progammeringstekniska problemen och dess lösningar.
- PID-regulator: Vi började med att reglera med en PID-regulator för att få en känsla för hur processen bör regleras.
- Adaptiv-regulatorn: Projektets huvudmål, att använda teorin inom Adaptiv reglering på vår process.

3.2 Realtid

Vi använde UNIX–VME som hårdvara för projektet.Datorprogrammet skrevs i realtidsspråket modula-2. Programmet är mycket generellt skrivet, vilket gör att det är mycket enkelt att gå in och ändra exempelvis regulatortyp. All synkronisering är implementerad med hjälp av monitorer.



Figur 2: Processgraf.

3.2.1 Moduler

Vi valde att bygga upp programmet i 7 st moduler (se figur 2).Här följer en kort beskrivning av vad respektive modul gör:

- Opcom : Har hand om all operatörskommunikation på consolen (se appendix).
- Command : Hjälpmodul till opcommodulen som bl.a. tolkar kommando (se appendix).
- Plot : Kommunicerar med Matlab och sköter plottningen av data (se appendix).
- RingBuffer : Buffrar data till plotmodulen så att man skickar data i paket, istället för ett och ett (se appendix).
- Adaptivreg : Denna modul innehåller själva regulatorn (se appendix).
- Designer : Hjälpmodul till regulatorn, gör en regulatordesign (se appendix).
- Estimator : Hjälpmodul till regulatorn, estimerar processen (se appendix).

3.2.2 Realtidsprocesser

I programmet behövde vi använda 3 st processer.

- Consoleprocess : Ligger hela tiden och väntar på kommando. När kommando kommer tolkas det, varefter lämplig åtgärd vidtas.
- Plotprocess : Buffrar data som skickats från regulatorprocessen i en buffert. När bufferten är full skickas den till matlab, vilken sköter plottningen.
- Regulprocess : När regulprocessen startas sker automatiskt en "tuning" av processen, därav namnet auto-tuner. Därefter sker en design av regulatorparametrar. Regulatorn kommer sedan vid varje sampel att reglera med just den här regulatorinställnnigen tills ny "tuning" görs.

3.3 PID-regulator

För att få en uppfattning om problemets svårighetsgrad och för att stilla vår nyfikenhet började vi med att försöka reglera processen på enklast tänkbara sätt, dvs att mäta vinkeln w (se figur 1) och låta en PID-regulator beräkna styrsignalen till motorn. Efter endast en kort tids justerande av parametrarna fann vi en regulator som fungerade väldigt bra. Parametrarna för denna regulator var: $k = 0.1$, $T_i = \infty$ och $T_d = 0.1$. Att ingen I-del behövdes är ju självklart, ty att ett stationärt fel skulle kunna uppstå är ju en fysikalisk omöjlighet. Resultatet var faktiskt så bra att en adaptiv regulator ironiskt nog framstod som helt onödig. Vi skulle aldrig klara av att få en avsevärt bättre regulator, hur avancerad vi än gjorde den.

3.4 Adaptiv-regulator

Efter att ha lyckats att reglera processen med PID-regulatorn var det dags för projektets verkliga uppgift. Visa av kunskapen från föreläsningarna i kursen Adaptiv reglering bestämde vi oss för en indirekt självinställare[1]. Tanken var att vid varje reglerbehov, det vill säga att vid varje upplyftning och nedsläppning av kulan, estimera parametrarna och reglera inom varje sampel.

[1] se Adaptiv Control kap 5

3.4.1 Simulering

För att kunna få ett någorlunda grepp om processen började vi med att simulera estimering och reglering av den. Simuleringen gjordes i programmet Simnon som vi hade tidigare erfarenhet av. Studium av tidigare projekt på processen gav som tidigare nämnts den diskreta modellen:

$$H_m(z) = \frac{k(z-1)}{z^2 + a_1 z + a_2}$$

För att först och främst få en vettig reglering simulerades processen med enbart regulatorn inkopplad, alltså utan estimering. Vi använde oss av en vanlig RST-regulator och för att få fram R- och S-polynomen gick vi till väga på känt manér. Som önskat A-polynom Am valdes den samplade versionen av:

$$A_m = s^2 + 2\zeta_m \omega_m s + \omega_m^2$$

Som observerarpolynom valdes den samplade versionen av:

$$A_o = s + \omega_o$$

Specifikationen sker alltså i s-planet genom angivelse av önskat ζ och ω. Processens typ (referensvärde=0) ger att T-polynomet försvinner. Gradtalsanalys gav följande Diofantiska ekvation:

$$(z^2 + a_1 z + a_2)(z + r_1) + k(z-1)(s_0 z + s_1) = (z + a_o)(z^2 + a_{m1} z + a_{m2})$$

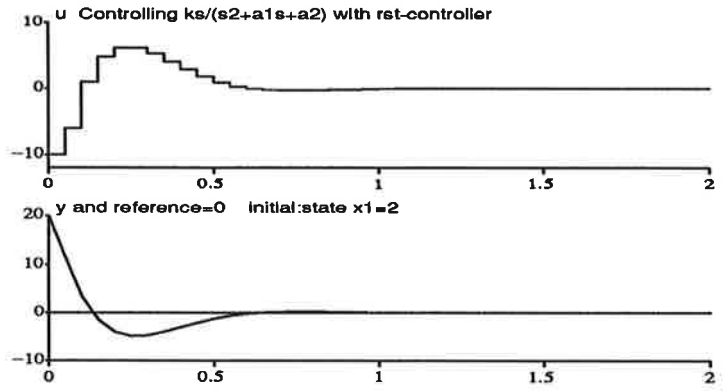
Evaluering av denna gav följande polynom:

$$R = z + \frac{(1 + a_o)(1 + a_{m1} + a_{m2})}{1 + a_1 + a_2} - 1$$

$$S = (\frac{a_{m1} + a_o - r_1 - a_1}{k})z + \frac{a_2 r_1 - a_o a_{m2}}{k}$$

För att kunna simulera utan estimeringen behövdes parametrarna k, a1 och a2. Ungefärliga värden[2] på dessa fick vi från tidigare nämnd rapport i processidentifiering. Simnon progammet kunde sedan skrivas och det delades upp i 4 delar: processen (kontinuerliga), regulatorn, sammanlänkning och själva simnon-macrot. Samplingsfrekvensen valdes till 50 Hz. För att få en realistisk reglersituation som motsvarar händelsen att kulan lyfts upp och sedan släpps (reglerproblemet är ej av servotyp) fick vi initialt sätta något tillstånd i processen skiljt från noll. Resultatet av simuleringen kan ses i figur 3. Resultatet verkade vara tillfredställande och vi kunde nu gå vidare med estimeringen inkopplad.

[2] k=0.8915, a1=-1.791, a2=1.165



Figur 3: Regulatorn utan estimering.

För att kunna estimera de 3 parametrarna användes stokastisk approximation med glömskefaktor. Man använder då att

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

$$P(t) = (\sum_{i=1}^{t} \lambda^{t-i}\varphi(i)^T\varphi(i))^{-1}$$

där

$$\Theta(t) = \begin{pmatrix} a_1 \\ a_2 \\ k \end{pmatrix}$$

$$\varphi^T(t) = \begin{pmatrix} -y(t-1) & -y(t-2) & (u(t-1) - u(t-2)) \end{pmatrix}$$

Genom att beräkna P(t) på ovan förenklade sätt, sparade vi mycket processortid. Till det tidigare simnonprogrammet tillfogades alltså nu en estimering och för att få tillräcklig excitation till estimeringen fick en referenssignal kopplas in. Enligt teorin[3] är en summa av 2 sinussignaler tillräckligt för att estimera 4 parametrar, alltså mer än tillräckligt även för vårt behov. Det ska dock nämnas att valet av sinussignalernas frekvenser är av största betydelse. Med ett felaktigt val kan det faktiskt bli så att exciteringen inte räcker. Efter en stunds testande fann vi att tex $\omega = 1$ och 3 rad/s var utmärkta val. Resultatet kan ses i figur 4.

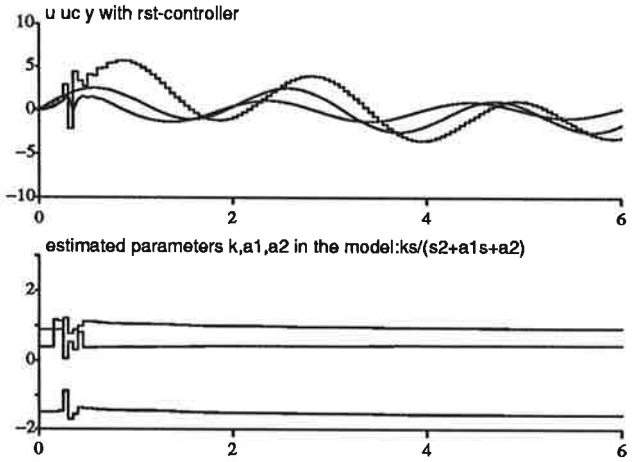
Estimeringen konvergerar till acceptabla värden och fungerar alltså. Man kan dock se att regulatorn ej lyckas följa referensen men som tidigare nämnts var detta ej heller varken reglerproblemet som skulle lösas eller sättet som regulatorn skulle arbeta på.

När vi fått simuleringen att fungera kände vi oss redo att realisera våra ansträngningar i verkligheten, väl medvetna om att denna ofta ej överensstämmer med den simulerade dito, men dock fyllda av gott mod.

3.4.2 Verkligheten

Efter simuleringen var det dags att koppla upp sig mot den verkliga processen. Vi försökte först att reglera utan estimeringen inkopplad. Regulatorn och estimeringen är givetvis de samma som i simnonprogammet så för det teoretiska hänvisas dit. Programmskelettet fanns ju redan efter PID-regleringen så allt som behövdes var att modifiera regulatormodulen och att tillfoga

[3] tex kap3,Adaptiv Control



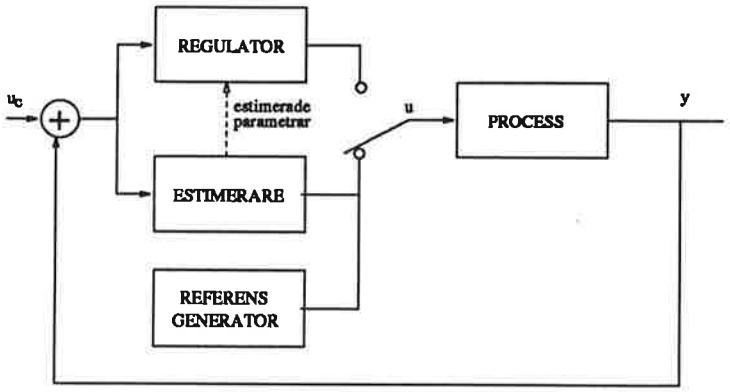
Figur 4: Estimering av processparametrar.

estimeringen för senare bruk. Modula-2-koden påminner starkt om simnonkoden så denna översättning var enkel att göra eftersom grovjobbet gjorts i simnon.

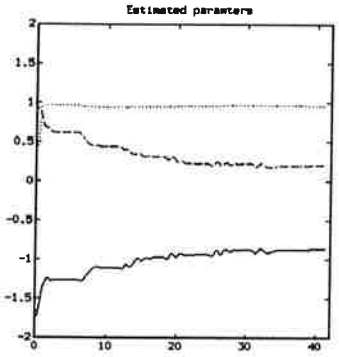
När kodningen skett försökte vi reglera, men det fungerade dåligt. Regulatorn uppträdde minst sagt underligt och inga variabeländringar hjälpte. Vi fruktade nu att våra teoretiska rön var felaktiga, men efter en del tänkande insåg vi att felet måste ligga i att vi skattat modellen allt för dåligt (se simuleringen). Vad göra? Att försöka skatta parametrarna bättre hade blivit tidsödande. Efter närmare eftertanke kom vi på att vi ju redan hade ett verktyg att skatta parametrarna med, nämligen estimeringen. Vi kopplade in denna i öppen loop och lade till en referenssignal i form av en summa av 3 sinussignaler för att säkert kunna skatta de för regleringen nödvändliga parametrarna. Efter ett par försök lyckades vi få fram en lämplig referenssignal där frekvenserna ligger runt systemets resonansfrekvens, som enligt rapporten ekorrhjulet är 1,5 Hz och amplituder som fullt ut utnyttjar systemets variationer. Med rätt inställd referens lät vi estimeringen pågå under 40 sekunder, varefter parametrarna konvergerat. Därefter använde vi dessa estimaten i tidigare nämda regulator. Bingo! Genast hade vi en regulator som fungerade som vi ville. Efter lite finjusteringar av de önskade parametrarna i Am och Ao hade vi fått en regulator som reglerade minst lika bra som PID-regulatorn. Estimeringen och regulatorn kopplades nu samman i sluten loop till den indirekta självinställare som vi tidigare hade simulerat. Tyvärr lyckades vi aldrig få detta att fungera och vi blev nu tvungna att tänka om. Våra tidigare teorier mötte en viss skeptiscism från våra handledare, vilka menade att processens natur ej skulle möjliggöra vår konstruktion. Vid närmare eftertanke insåg vi att detta naturligtvis var rätt. Att exitera processen genom att lyfta upp kulan och därefter släppa den är felaktigt. Upplyftningen är egentligen en laststörning och beskrivs ej av $Y = GU$. Vår nya lösning på problemet var att göra en slags tuningregulator, som först estimerar modellens parametrar i öppen loop och därefter använder estimaten för att beräkna parametrarna i en RST-regulator. Regulatorn arbetar alltså på samma sätt som då vi tog fram processparametrarna, fast nu sker omslag mellan estimering och reglering automatiskt efter 40 sekunder.(Se figur 5.)

Vår regulator har därmed hamnat i utkanten av ämnet Adaptiv reglering men för att kunna nå vårt mål, som var att reglera minst lika bra som med PID-regulatorn, fann vi denna angreppsstrategi lämplig.

Figur 6 visar en plottning av estimaten som fås vid den inledande estimeringen och här ser man tydligt att konvergens sker inom de 40 sekunder som estimeringen tar. Vid upprepade estimeringar visar det sig att man får värden som ligger nära varandra, vilket ju också är ett tecken på att funktionen är tillfredställande.



Figur 5: Blockschema över systemet.



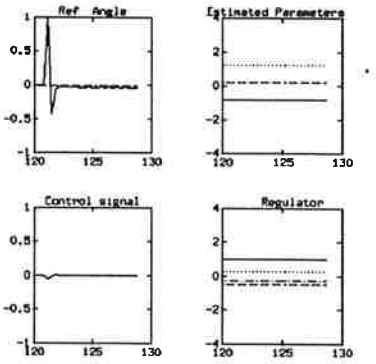
Figur 6: Estimering av verkliga processen.

Figur 7 visar regulatorns uppträdande när kulan lyfts upp en bit och släppts en gång. Kulan regleras nästan direkt ned till sitt nollläge utan några konstigheter. Dessutom visas regulatorparametrarna i ett eget diagram.

Parametrar	*Namn*	*Värde*
Sampelperiod	h	0.05 s
Initialkovarians	P_0	100
Glömskefaktor	λ	0.95
Modell	ω_m, ζ_m	8.0 0.7
Observerare	ω_0	18
Maximal utsignal	u_{max}	5 V

I tabellen redovisas de ideala parametervärden vilka vi prövade fram.

Regulatorns beteende med inkopplad estimering var mycket tillfredställande för oss. Att översätta teori till verkligheten är ju inte alltid det lättaste. Speciellt efter vårt lyckade försök att estimera parametrarna, kändes arbetet med att ta sig igenom den inte helt enkla teorin bakom projektet genast mer nyttigt. Vårt mål var att få vår regulator lika bra som PID-regulatorn. Detta var nu tillfullo uppfyllt och vi ansåg oss vara nöjda med vårt program!



Figur 7: Plottning vid reglering av processen.

4 Förbättringar

4.1 Realtid

Det behövs en del förbättringar för att vi skall kunna känna oss riktigt nöjda. Till exempel måste hårdvaran förbättras. Vi syftar då på Appletalk-kommunikationen mellan VME och UNIX. Om programmet ej fungerar beror det med största sannolikhet på detta.
Uppstartsrutinen är också något krånglig[4]. Den borde kanske ha varit konstruerad så att endast *ett* kommando behövs för uppstart av hela programmet.
Programmet borde kanske också visa när tuning pågår och när den är avslutad.

4.2 Regulator

Några egentliga förbättringar på vår adaptiva regulator behövs ej. Om man vill och har tid kan man för nöjes skull prova med en högre ordningens modell, men att detta skulle ge en märkbart bättre reglering är föga troligt.
Man skulle även kunna prova andra reglermetoder.
En sådan är att man använder sig av en vanlig PID-regulator, vars parametrar trimmas mha reläåterkoppling[5]. Detta diskuterade vi, men tyvärr hann vi ej genomföra detta i praktiken.
Kanske finns det metoder där man utnyttjar att man kan mäta vinkeln och vinkelhastigheten på hjulet, signaler som vi inte alls har använt.
Oberoende av vilken reglermetod och vilka mätsignaler man använder bör man tänka på en sak: Gör det inte svårare än vad det är.

5 Manual

5.1 Inledning

Denna manual utgör en lathund för att kunna exekvera reglerprogrammmet och identifiera styrsignaler, mätsignaler och parametrar. Första delen behandlar hur programmet laddas ned

[4] se Manual
[5] kap 8.4 i Adaptiv Control

till styrdatorn och hur exekveringen startas. I efterföljande del beskrivs vilka kommandon som styr och övervakar regulatorn under igångkörning och i drift.

5.2 Uppstart av regulatorn.

Utgående från en arbetsstation med tillgång till "x-windows", används *tre* fönster. Ett fönster för UNIX-kommandon, ett för operatörskommunikationen och det sista för Matlab. Följande kommandon används i nedan angiven ordning:

1. **initregler** i samtliga fönster.
2. **matlab** i fönstret avsett för matlab. Här kommer då matlab att exekveras och en prompt erhålles.
3. **realtid VME133 X** i unix fönstret, där X anger numret på vilken applikationsdator som används.
4. **simcom** i det fönster som är avsett för operatörskommunikation. Simcom exekveras och dess prompt erhålles.
5. **g fff20000** anges som startadress efter prompten i simcom-fönstret.
6. **ago VME/Main.sr** ges i unix-fönstret varvid programmet kommer att laddas ned i applikationsdatorn. Nedladdningsprocessen visualiseras i simcom-fönstret och därefter startar operatörskommunikationen i samma fönster.
7. **plotter** i matlab-fönstret varefter applikationsdatorns nummer efterfrågas och plot-fönstret erhålles.

Reglerprogrammet styrs efter ovanstående inledningen av operatörskommunikationen och i matlabs grafiska fönster erhålls signaler och parametrar. Exekveringen startar automatiskt och inleds med en tuning av regulatorn. Denna pågår i ca 40s och därefter träder regulatorn i funktion. Möjlighet finnes att estimera om parametrarna.

5.3 Styrningen och övervakning av regulatorn

Genom kommandot **help** erhålls operatörkommunikationens kommandon med vars hjälp styrning och övervakning kan ske. Följande kommandon och deras funktion ges nedan.

- **setpars** Kommandot används då man vill ändra någon av följande parametrar: sampeltiden (h), egenfrekvens (ω_m) och dämpning (ζ_m) hos det önskade systemet, egenfrekvens (ω_o) hos observerarpolynomet, initialkovariansen (P_0), glömskefaktor (λ) och maximal utspänning (u_{max}).
- **tune** Kommandot ställer in regulatorparametrarna genom estimering av processen. Därefter är regulatorn aktiverad och reglerar därmed processen.
- **getpos** Kommandot ger ett värde som motsvarar kulans läge.
- **showregpar** Värdena för aktuella regulatorparametrar i RST-polynomet erhålls genom kommandot.
- **showestimates** Estimaten till överföringsfunktionen erhålls via detta kommando.

- **showpar** Kommandot används till att visa de aktuella värdena för sampeltid, egenfrekvens, dämpning, observerarpolynomets egenfrekvens, kovariansen, glömskefaktor och maximal utspänning.
- **quit** Kommandot stoppar exekveringen.

6 Källförteckning

- Kolbjörn Gripne, Lars Hermanson. *Ekorrhjulet, ett projekt i processidentifiering.* Institutionen för Reglerteknik LTH 1991.
- Lars Nielsen. *Computer Implementation of Control System.* Institutionen för Reglerteknik LTH 1992.
- Karl Johan Åström, Björn Wittenmark. *Adaptive Control.* Addison Wesley 1989.

7 Appendix

```
DEFINITION MODULE Opcom;

(*$NONSTANDARD*)

PROCEDURE Init( Prio : CARDINAL );
(* Initializes the Opcom-module *)

PROCEDURE WaitForExit;
(* The program continous until this function is called *)

PROCEDURE Stop;
(* Close down the opcom-module *)

END Opcom.


DEFINITION MODULE Command;

TYPE
  Commands = (cUnknown, cAmbiguous, cTune, cHelp, cQuit,cGetPosition,
              cSetPars,cShowRegPar,cShowEstimates,cShowPar);
  (* Avaliable commands in the controller *)

PROCEDURE Which(name : ARRAY OF CHAR) : Commands;
(* Returns command connected to name *)


PROCEDURE WhichRegulCommand(name : ARRAY OF CHAR) : CARDINAL;
(* Same as function "Which" for Controller-commands *)

PROCEDURE FullName(VAR name : ARRAY OF CHAR; key : Commands);
(* Search for fullname for command when shortname is used *)
(* e.g showe -> showestimates                             *)

PROCEDURE FullNameRegul(VAR Name : ARRAY OF CHAR);
(* Same as function "FullName" for Controller-commands *)

PROCEDURE Init;

END Command.


DEFINITION MODULE Plot;

(*$NONSTANDARD*)

FROM Adaptivreg IMPORT EstimType,Polynomtype,NrOfRegpars,NrOfEstims;
```

```
CONST NrToPlot=(4+NrOfRegpars+NrOfEstims);

TYPE RecType = RECORD
                 Time,
                 Position,
                 Reference,
                 Control : LONGREAL;
                 Estimate       : EstimType;
                 Rpolynom       : Polynomtype;
                 Spolynom       : Polynomtype;
               END;

PROCEDURE Init(Priority:CARDINAL);
(* Initialize the plot-module *)

PROCEDURE PutData(Element:RecType);
(* Accepts data to be plotted  *)

PROCEDURE Stop;
(* Stops execution of plot-module *)

END Plot.



DEFINITION MODULE RingBuffer;

FROM Plot IMPORT NrToPlot;

TYPE Buffer;
     DataType = ARRAY [1..NrToPlot] OF LONGREAL;

PROCEDURE PutData(data: DataType; B: Buffer);
(* Insert ch into B *)

PROCEDURE GetData(VAR data: DataType; B: Buffer);
(* Returns ch from B *)

PROCEDURE NewBuffer(VAR B: Buffer);
(* Creates a new buffer by calling NEW and initializing
   the buffer *)

PROCEDURE DisposeBuffer(VAR B: Buffer);
(* Deletes the buffer by calling DISPOSE *)

END RingBuffer.



DEFINITION MODULE Adaptivreg;

(*$NONSTANDARD*)
```

```
CONST
    NrOfPars = 7;
    NameLength = 10;
    NrOfEstims = 3;
    NrOfRegpars = 4;
    NrOfStates = 4;
    PolynomLength = 2;

TYPE
    NameType = ARRAY [0..NameLength] OF CHAR;
    ParNameType = ARRAY [1..NrOfPars] OF NameType;
    ParType = ARRAY [1..NrOfPars] OF LONGREAL;
    PosRefType = LONGREAL;
    EstimNameType = ARRAY[1..NrOfEstims] OF NameType;
    EstimType = ARRAY[1..NrOfEstims] OF LONGREAL;
    RegNameType = ARRAY[1..NrOfRegpars] OF NameType;
    RegType = ARRAY[1..NrOfRegpars] OF LONGREAL;
    Statetype=ARRAY[1..NrOfStates] OF LONGREAL;
    Estimpartype = RECORD
                       lambda,p0:LONGREAL;
                       estim:EstimType;
                       state:Statetype;
                   END;
    Polynomtype = ARRAY[1..PolynomLength] OF LONGREAL;

PROCEDURE GetParNames( VAR parnames : ParNameType );
(* Returns the names of the parameters of the module in parnames[1], ...
   parnames[NrOfPars] *)

PROCEDURE GetPars( VAR pars : ParType );
(* Returns the current parameter values in Pars[1], ...Pars[NrOfPars].
   Notice that the indexing corresponds to the indexing of the parameter
   names when calling GetParNames *)

PROCEDURE SetPars( REF newpars : ParType );
(* Sets all the parameters of the module to newpars[1],...
   newpars[NrOfPars] *)

PROCEDURE GetPos() : LONGREAL;
(* Returns the current position of the servo *)

PROCEDURE Init(priority:CARDINAL);
(* Initializes Adaptivreg *)

PROCEDURE Stop;
(* Terminates Adaptivreg *)

PROCEDURE Start;
(* Starts Adaptivreg after termination *)

END Adaptivreg.
```

```
DEFINITION MODULE Designer;

FROM Adaptivreg IMPORT EstimType,Polynomtype,RegNameType,RegType;

(*$NONSTANDARD*)

PROCEDURE GetDesignNames(VAR regnames:RegNameType);
(* Returns the names of the controller parameters *)

PROCEDURE GetDesign(VAR reg:RegType);
(* Returns the actual controller parameters *)

PROCEDURE Design(VAR r,s:Polynomtype; estim:EstimType);
(* Calculates the R and S polynomials of the design *)

PROCEDURE SetDesign(ao,am:Polynomtype);
(* Specify the design *)

PROCEDURE InitDesign();
(* Initializes the Design-module *)

END Designer.


DEFINITION MODULE Estimator;

FROM Adaptivreg IMPORT Estimpartype,Statetype,EstimNameType,EstimType;

PROCEDURE GetEstimNames(VAR estimnames:EstimNameType);
(* Returns the names of the estimated parameters *)

PROCEDURE GetEstim(VAR estim:EstimType);
(* Returns the values of estimated parameters *)

PROCEDURE InitEstim(estimpar:Estimpartype);
(* Initializes the estimator-module *)

PROCEDURE Estim(VAR estimpar:Estimpartype;y:LONGREAL);
(* Estimates the parameters in the sampled version of the continous *)
(* transfer function ks/(s2+a1s+a2)                                 *)

END Estimator.
```

Reglering av skvalpprocess VT –93
Ett kombinerat projekt i
Realtidssystem och Adaptiv reglering

Projektgrupp M
Christian Nilsson D89
Patrik Olofsson D89
Stefan Larsson D89

27 april 1993

Sammanfattning

Uppgiften var att, i realtid, reglera processen med en adaptiv regulator. Innan vi använde adaption reglerade vi processen med en PID- och en RST-regulator. Vi fann att den bästa regulatorn var en PD-regulator och att skalpprocessen inte lämpar sig för adaptiv reglering, eftersom processens dynamik inte förändras nämnvärt.

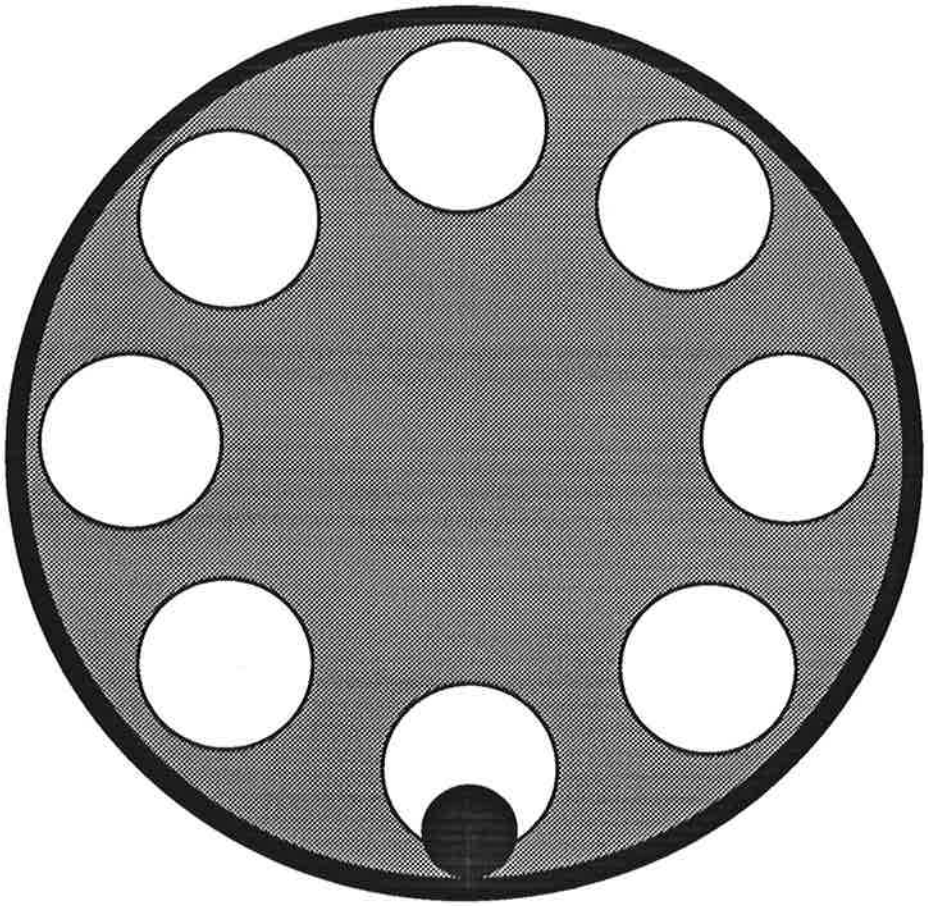
Tack till våra handledare Johan och Ulf.

Innehåll

1 Uppgiften

Uppgiften består i att implementera ett dataprogram som reglerar en process. Regulatorn ska implementeras generellt så att den är lätt att byta ut. Vi har som uppgift att reglera processen med en PID-regulator och en adaptiv regulator. Programmet ska också innehålla operatörskommunikation och plotterfunktion.

Processen som ska regleras är en kula som rullar inuti ett hjul. Meningen är att kulan ska ligga stilla och regleras tillbaka vid en yttre lastpåverkan. Denna process är lik den som finns i fartyg för att stabilisera fartyget, därav namnet "skvalpprocess".



Figur 1: Processuppställning

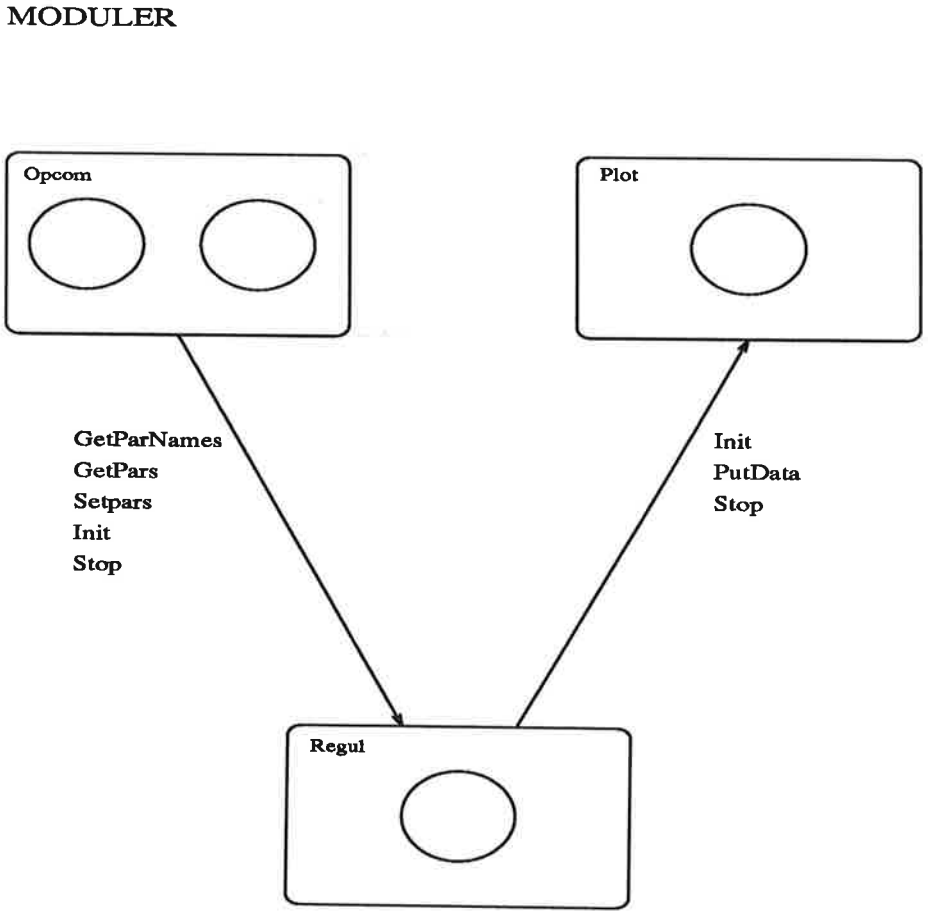
2 Processer

Programmet innehåller fyra processer, vilka i korthet har följande uppgifter:

- operatörskommunikation via Simcom och Matlab
- plottning av signaler
- regulator — sköter in- och utsignaler från processen

2.1 Moduler

Följande indelning har gjorts av processerna i moduler:



Figur 2: Ingående moduler

2.2 Operatörskommunikation

Följande kommando finns tillgängliga för operatören:

- Help — hjälpkommando
- Quit — stoppa programmet
- SetRegPar — sätt nya värden på regulatorparametrar
- ShowRegPar — visa värden på regulatorparametrar

Ex. för att byta värde på K och TI
≫setregpar K 2.0 TI 5.0

2.3 Plotterfunktion

Plotterfunktionen gör det möjligt att se intressanta signaler på skärmen via Matlab. Vi har valt att följande signaler skall plottas:

- utsignal från systemet — aktuell vinkel
- styrsignal till systemet — utsignal från regulator
- estimerat K-värde — processens förstärkning enl ekv (1)

Eftersom vi samplar vårt system relativt fort har vi valt att bara skicka vart 20:e sampel till plottern. Detta gör vi för att plottningen skall kunna hänga med och plotta aktuella värden.

Se appendix A hur datastrukturen ser ut.

2.4 Regulator

Definitionsmodulen för regulatorn är generell så att man lätt kan byta typ av regulator. Vi startade projektarbetet med att reglera processen med en befintlig PID-regulator. När denna regulator fungerade övergick vi till att implementera en RST-regulator som skulle ligga till grund för vår adaptiva regulator. Detta gjorde vi för att bli övertygade om att denna regulatorstruktur fungerar för vår process.

2.4.1 PID-regulator

Denna regulator utvecklades i ett tidigare övningsprojekt i kursen Realtidssystem, och har en struktur som förklaras i [3]. Efter en viss tids intrimning fungerade regleringen mycket bra. Regulatorn hade då följande parametervärden:

- K=-0.2
- Ti=1000
- Td=0.2
- Tr=0.45
- h=0.01

Det höga värdet på Ti innebär att vi inte har någon integralverkan i regulatorn.

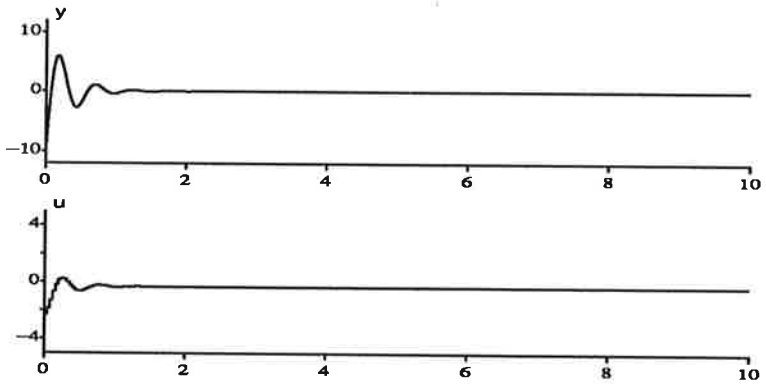
2.4.2 Adaptiv regulator

I uppgiften ingick även att implementera en adaptiv regulator. Vi började med att analysera processen för att få en uppfattning om modellordning. Vi utförde ett experiment där vi lät kulan självsvänga och mätte då periodtid och dämpning. Eftersom processen inte har någon statisk förstärkning så finns det ett nollställe i nollan. Följande modell antas:

$$G(s) = \frac{sK}{s^2 + 2\zeta\omega s + \omega^2} \tag{1}$$

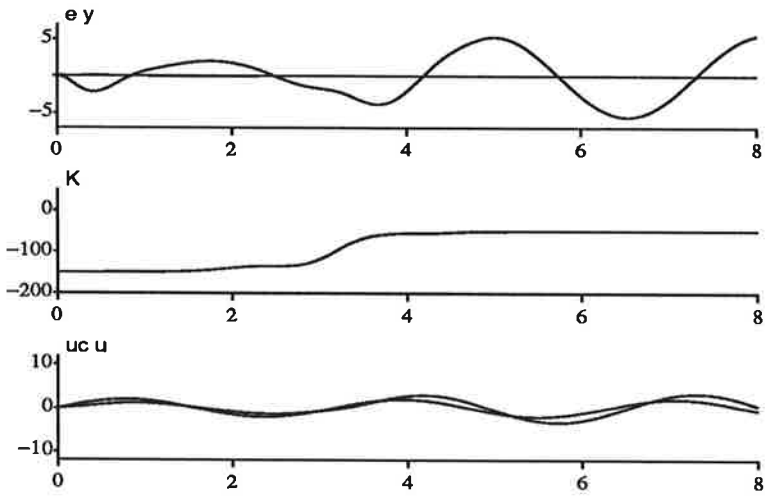
$$\omega = 9 \text{ och } \zeta = 0.06$$

Detta system simulerades i Simnon tillsammans med en RST-regulator som framtagits i Matlab. Systemet initieras med K=-50. Här följer ett resultat av simuleringen.



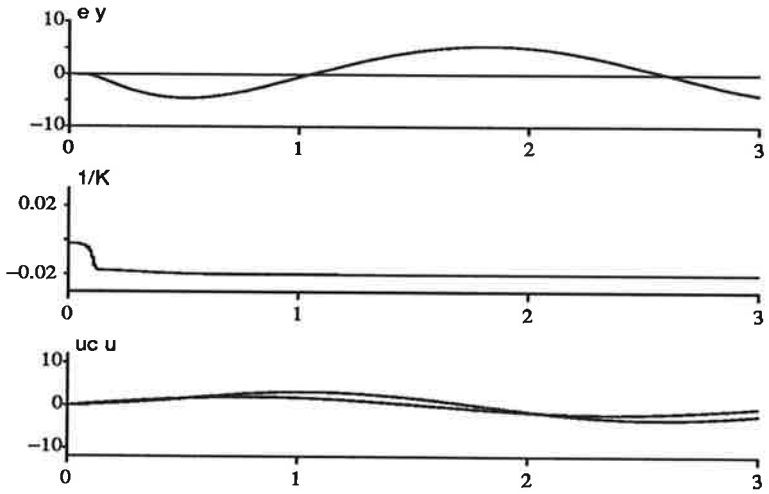
Figur 3: Simulering av systemet i Simnon med K=-50, där kulan vridits upp till sitt ena ändläge och släppts

Denna RST-regulator implementerades innan vi försökte konstruera en adaptiv algoritm. När vi fått denna att fungera tillfredsställande tog vi oss an den adaptiva biten. Vi försökte skatta K-värdet i processen och använde de experimentiellt framtagna ω och ζ. Detta kunde vi inte få att fungera pga för dålig excitation i systemet. För att öka excitationen satt vi och slog till kulan, men det visade sig olämpligt eftersom man tillför en oönskad laststörning på utgången som inte beskrivs av processens in- utsignal samband. För att undvika detta beslöt vi oss för att införa en referenssignal i avsikt att ställa in regulatorn innan vi börjar reglera. Denna ide' simulerade vi först i Simnon. Resultatet nedan visar att det fungerar men att det tar relativt lång tid innan K-värdet har konvergerat.

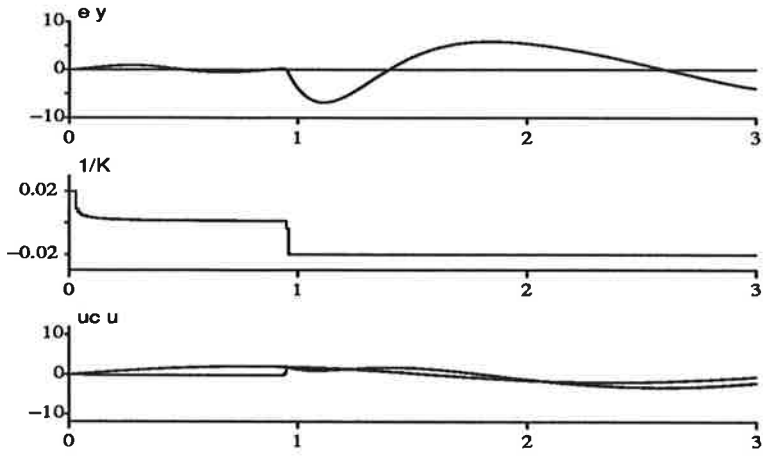


Figur 4: Estimering av K med sinusformad referenssignal

Detta visade sig inte fungera på den verkliga processen, dels konvergerar K mot ett felaktigt värde, 0, dels måste man känna tecknet på K eftersom estimeringen inte klarar en teckenväxling. För att undvika detta försökte vi istället estimera 1/K. Vi simulerade även detta med följande resultat.



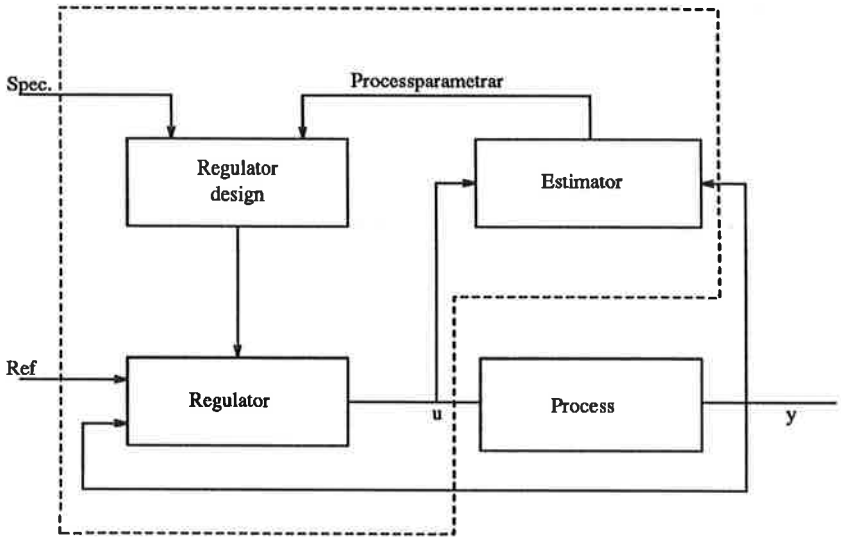
Figur 5: Estimering av 1/K med sinusformad referenssignal. 1/K initierat till ett litet värde med rätt tecken.



Figur 6: Estimering av 1/K med sinusformad referenssignal. 1/K initierat till ett värde med fel tecken.

Av simuleringarna framgår det att detta fungerar utmärkt i teorin, men tyvärr visade det sig inte fungera i praktiken. Konvergensen gick mycket snabbare, men tyvärr mot ett felaktigt slutvärde, noll. Anledningen till detta kunde varken vi eller vår handledare utreda.

För att skatta K och 1/K använde vi oss av RLS (Recursive Least Square) med glömskefaktor enligt teorem 3.4 i [1]. Den adaptiva regulatorn är en STR (Self Tuning Regulator).



Figur 7: Skiss över en STR

Referenser

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

[2] Karl Johan Åström, Björn Wittenmark. *Computer Controlled Systems.* Prentice-Hall 1984.

[3] Lars Nielsen. *Computer Implementation of Control Systems.* KF-Sigma 1993.

[4] Per Foreby. *Att skriva rapport med* LaTeX. Datordriftgruppen LTH 1992.

A Definitionsmoduler

A.1 Regul.def

DEFINITION MODULE Regul;

```
DEFINITION MODULE Regul;

(*$NONSTANDARD*)

CONST
    NrOfPars = 6;
    NameLength = 10;

TYPE
    NameType = ARRAY [0..NameLength] OF CHAR;
    ParNameType = ARRAY [1..NrOfPars] OF NameType;
    ParType = ARRAY [1..NrOfPars] OF LONGREAL;

PROCEDURE GetParNames( VAR Names : ParNameType );
(* Returns the names of the parameters of the module in Names[1], ...
   Names[NrOfPars] *)

PROCEDURE GetPars( VAR Pars : ParType );
(* Returns the current parameter values in Pars[1], ...Pars[NrOfPars].
   Notice that the indexing corresponds to the indexing of the parameter
   names when calling GetParNames *)

PROCEDURE SetPars( REF NewPars : ParType );
(* Sets all the parameters of the module to NewPars[1],...
   NewPars[NrOfPars] *)

PROCEDURE Init(priority:CARDINAL);
(* Initializes Regul *)

PROCEDURE Stop;
(* Terminates Regul *)

END Regul.
```

A.2 Opcom.def

```
DEFINITION MODULE Opcom;

(*$NONSTANDARD*)

PROCEDURE Init( Prio1,Prio2 : CARDINAL );

PROCEDURE WaitForExit;

PROCEDURE Stop;

PROCEDURE Error( REF Errortext: ARRAY OF CHAR);
(* Writes Errortext on terminal *)

PROCEDURE FatalError( REF Errortext: ARRAY OF CHAR);
(* Writes Errortext on terminal, and causes Exit, i.e system shut-down. *)

END Opcom.
```

A.3 Plot.def

```
DEFINITION MODULE Plot;

(*$NONSTANDARD*)

TYPE RecType = RECORD
                 Time,
                 Position,
                 Reference,
                 Control        : LONGREAL;
               END;


PROCEDURE Init(Priority:CARDINAL);
(* Initierar Plotter-modulen *)

PROCEDURE PutData(Element:RecType);
(* Tar emot data som skall plottas *)

PROCEDURE Stop;
(* Stoppar Plotter-modulen *)

END Plot.
```

Projekt i Adaptiv reglering och Realtidssystem

Lennart Andersson
Morten Hemmingsson
Carl-Johan Ivarsson
Michael Lekman

Inst. för Reglerteknik

Handledare : Klas Nilsson

Sammanfattning

Till en ASEA IRB–6 robot har rutiner för estimering av robotens trögheтsmoment implementerats liksom en diskret PI–regulator. Skattade tröghetsmoment används indirekt för lastestimering. De nya programavsnitten har lagts till i den mjukvara som utvecklats under tidigare projekt.

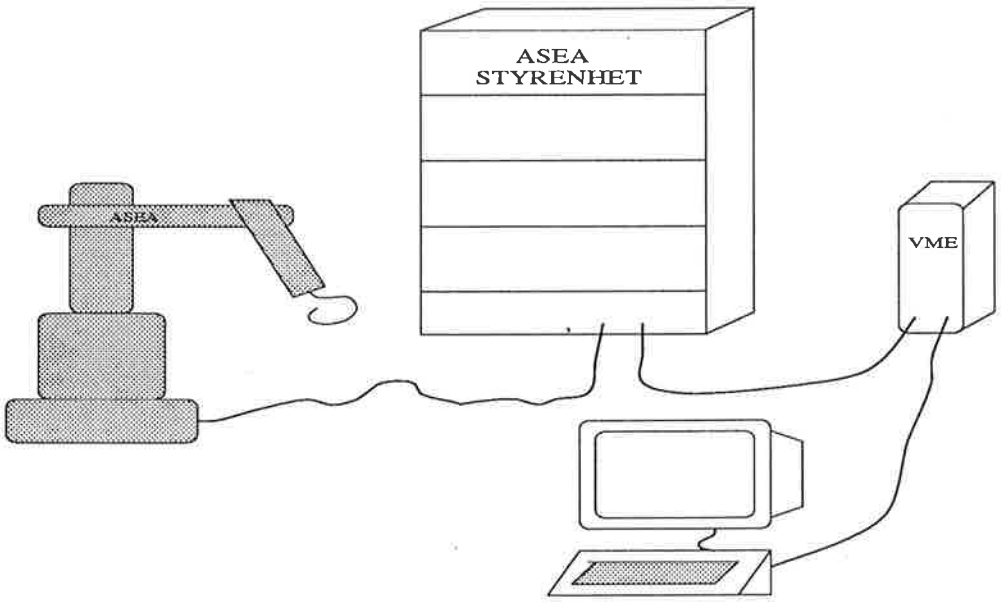
Inledning

I detta avsnitt beskrivs det problem vi skall lösa och det bakomliggande system som vi använt oss av.

Bakgrund

Den robot vi utgår ifrån är en ASEA robot på vilken vissa modifieringar gjorts. Säkerhetssystem och kraftelektronik är kvar medan styrsystemet är utbytt mot en VME-dator. Denna dator går att kommunicera med från en arbetsstation.

Redan tidigare fanns färdig mjukvara för styrning av roboten, inkluderande operatörs gränsnssnitt och ploter rutin. I mjukvaran fanns också ett enkelt robotspråk. Koden körs i VME-datorn medan den grafiska presentationen och användarkommunikationen sker på arbetsstationen, se figur 1



Figur 1. Uppställning.

Uppgift

Utgångspunkten är ASEA roboten. En av dess uppgifter är material hantering. Det kan då vara intressant att veta vikten på den upplyfta lasten. Vår uppgift var att via en skattning av robotens tröghetsmoment, med avseende på en av lederna, få fram ett värde på lastens massa. Lösningen skall översättas till exekverbar kod som passar in i befintlig mjukvara.

Lösning

För att lösa uppgiften krävs följande

1. Konstruera en modell för robotarmens dynamik. Modellen skall vara sådan att tröghetsmomentet J är en parameter som går att skatta. Metoden för hur skattningen skall göras måste också fastställas.
2. Robotens olika armar regleras idag av kaskadkopplade P– och PI–regulatorer, varav den ena är diskret och exekveras i VME datorn och den andra är kontinuerlig och finns i styrskåpet. Vi behöver direkt tillgång till alla regulatorernas utsignaler för att kunna estimera. Lösningen är att ersätta de kontinuerliga med diskreta motsvarigheter som även de exekveras i VME datorn.
3. För att operatören skall kunna utnyttja de nya faciliteterna krävs ett utökat robotspråk.

Teorin nedan ger löningen till första uppgiften.

Teori

Den teori som har använts berör estimeringens modell och metod. Metoden har valts till rekursiv minsta medelkvadrat med dyadisk dekomposition för att uppfylla implementeringskrav. Modellen baseras på likströms motorns förenklade momentekvation och lägre termer i en utveckling av friktionen. Modellen kommer nu att beskrivas närmre.

Rörelse med avseende på en led kan beskrivas med följande modell

$$T = F + J\ddot{\theta} + D\dot{\theta}$$

där T är det drivande momentet, J är tröghetsmomentet m a p aktuell led och D den viskösa dämpningen. Övrig friktion beskrivs av:

$$F = (F_0 + F_1\theta)sign(\dot{\theta})$$

För att estimeringen ska ske med tyngdpunkt på de intressanta låga frekvenserna bör en lågpass filtrering ske. Efter filtrering av modellekvationen och en approximation erhålls följande modell

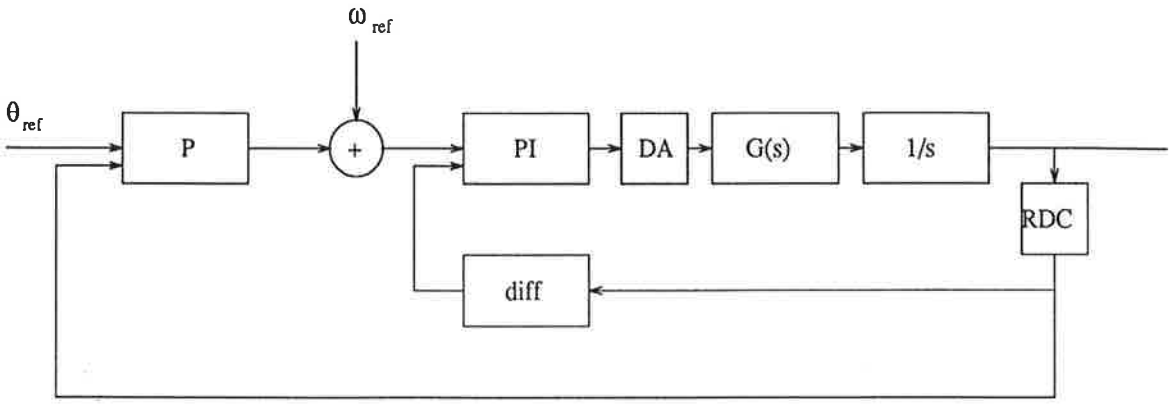
$$T_f = (F_0' + F_1'\theta_f)sign(\dot{\theta}_f) + J\ddot{\theta}_f + D\dot{\theta}_f$$

På regressionsform blir modellen

$$T_f = \begin{bmatrix} sign(\dot{\theta}_f) & \theta_f sign(\dot{\theta}_f) & \dot{\theta}_f & \ddot{\theta}_f \end{bmatrix} \begin{bmatrix} F_0 \\ F_1 \\ D \\ J \end{bmatrix} \qquad (1)$$

som kan användas direkt i den rekursiva estimatorn eftersom sammhörande $T, \theta, \dot{\theta}$ och $\ddot{\theta}$ är tillgängliga för estimatorn. För att erhålla tillräcklig

excitation kommer aktuell robotarm att exciteras med en PRBS signal. Punkt två har lösts genom att regulatorerna för varje led byggts upp av två kaskadkopplade regulatorer, se figur 2. Hastighetsregulatorn är av PI typ och positionsregulatorn är av P typ. Återkoppling sker från



Figur 2. Blockschema för regulatorn. Det finns en regulatoruppsättning för varje robotled. RDC betyder Resolver to Digital Converter.

vinkelhastigheten. Då denna ej finns tillgänglig måste den approximeras. Differentiering av vinkelsignalen ger en approximativ vinkelhastighet. Diffrentieringen görs utan några former av filtrering.

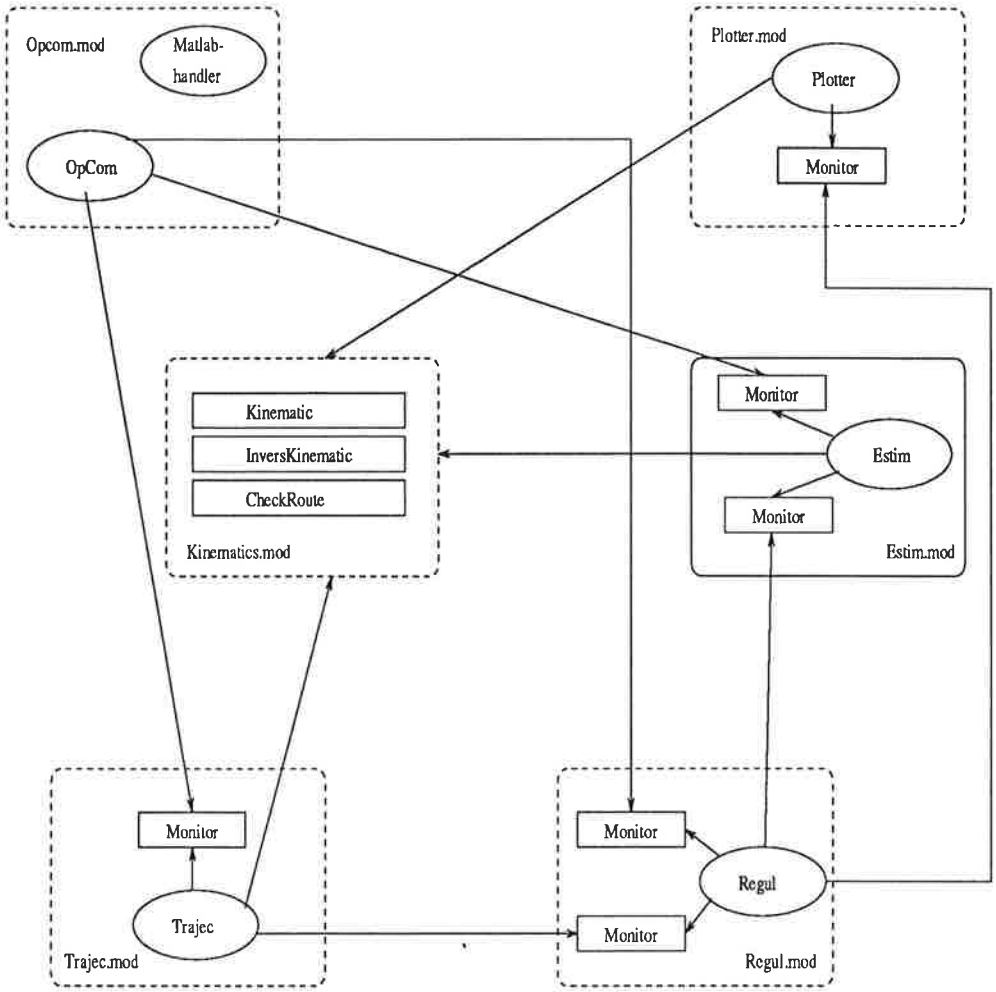
De tre nya robotkommandon i punkt tre har lagts till i det befintliga robotspråket. Kommandona beskrivs i sektionen Nya Robotkommadon på sidan 6.

Implementering

Implemeteringen av de ovan anvisade lösningarna har gjorts på två sätt. Dels har tidigare kod utökats/förändrats (gäller modulerna OPCOM och REGUL) och dels har en ny modul skrivits (ESTIM). Resulterande fullständiga processgraf visas i figur 3.

OPCOM Den struktur som tidigare fanns med kommandon i ett språk har utökats med de tre kommandona INITEST, ESTIM och DISPR.

- INITEST startar en estimering av robotarmens eget tröghetsmoment.
- ESTIM utnyttjar informationen från INITEST om tröghetsmomentet, gör en ny estimering men med lasten upplyft och returnerar en skattning av lastens massa.
- DISPR visar innehållet i valt register.



Figur 3. Processgraf för systemet. Streckade moduler är gamla som modifierats medan heldragna moduler är nya. Processerna Matlabhandler och Plotter kommmunicerar med Matlab-processer i värddatorn.

Förändringarna har praktiskt genomförts med tillägg i tidigare gjord kod. Det är främst i processen Interpretor förändringar gjorts. Tre nya procedurer har skrivits för att avkoda de nya kommandona. Detta var nödvändigt eftersom de i sin uppbyggnad skiljer sig något från de tidigare implementerade kommandona. Avkodningen tar hjälp av programpaketet Lexical Analyzer, detta ger en mängd möjligheter att avkoda kommandon med olika parametrar, dessutom hanterar det förkortningar av kommandona.

REGUL Programkoden för regulatorerna följer uppbyggnaden i Computer Implementattion of Control Systems,sidan 29.

I ursprunglig kod är det i reglerlopen förändringar gjorts. Då REGUL modulen även skall hantera genereringen av excitationssignalen har ett par nya procedurer skrivits. PRBS signalen genereras från ett 13 bitars

skiftregister konfigurerat enligt System identification. För att uppnå bra excitation som inte motarbetas av regulatorn väljs under estimeringen en sämre regulator för de aktuella leden.

För att kunna stänga av och sätta på PRBS signalen exporterar REGUL två nya procedurer, StartNoise respektive StopNoise.

För att förändringarna av OPCOM inte skulle leda för långt kan man bara ändra två parametrar därifrån. Övriga parametrar är hårdkodade in i REGUL modulen.

ESTIM Estimerings algoritmen finns beskriven i teori avsnittet ovan, se sid 3. Den rekursiva minsta kvadrat metoden baseras på Dyadic reduction. Koden är hämtad ur Adaptive Control(Karl Johan Åström och Björn Wittenmark). Den rekursiva algoritmen initieras och startas från OPCOM modulen. Data tillhandahålls perodiskt från REGUL modulen. Estimeringen fortsätter tills trace P, där P är ett kovariansmått i estimerings algoritmen, understiger ett givet värde eller att en timeout aktiveras.

För att förbättra konvergensen används en tidsvariabel lambda faktor. Den är 0.995 inledningsvis och ökas efter ca 20 s till 1.0. Tiden 20 s är vald efter erfarenhet.

När estimeringen är färdig kan OPCOM fortsätta exekvera och meddelar då REGUL att bruset skall stängas av.

Nya robotkommandon

För användaren kommer förändringarna att märkas genom tre nya kommandon:

- INITEST joint Jreg
- ESTIM joint Jreg Mreg
- DISPR reg

Parametrarna har följande betydelser:

joint Den led med avseende på vilken estimeringen utförs.

Jreg Registret i vilket skattningen av tröghetsmomentet lagras eller är lagrat. Då INITEST anropas fungerar Jreg som en destination medan det fungerar som en källa vid anropa av ESTIM.

Mreg I detta register kommer lastens skattade massa att lagras.

reg Register.

Det kommer att vara operatörens ansvar att se till att rätt register används vid respektive anrop. Dessutom ansvarar operatören för att samma position och led väljs både vid INITEST och ESTIM. Orsaken till detta

är att det skattade tröghetsmoment beror både av leden och postitionen. Genom att lagra skattat J från INITEST finns det möjlighet att göra upprepade ESTIM i samma position och för samma led utan att behöva upprepa INITEST.

Robotens arbetssätt vid estimering:

INITEST I den aktuella postitionen fås roboten att skaka genom en bra exiterande PRBS–signal. Detta fortsätter tills en bra skattning av tröghetsmomentet uppnåtts. Detta värde används sedan för bestämning av massan i ESTIM.

ESTIM Samma procedur som i INITEST med undantaget att massan lämnas som resultat istället för tröghetsmomentet.

Slutsatser

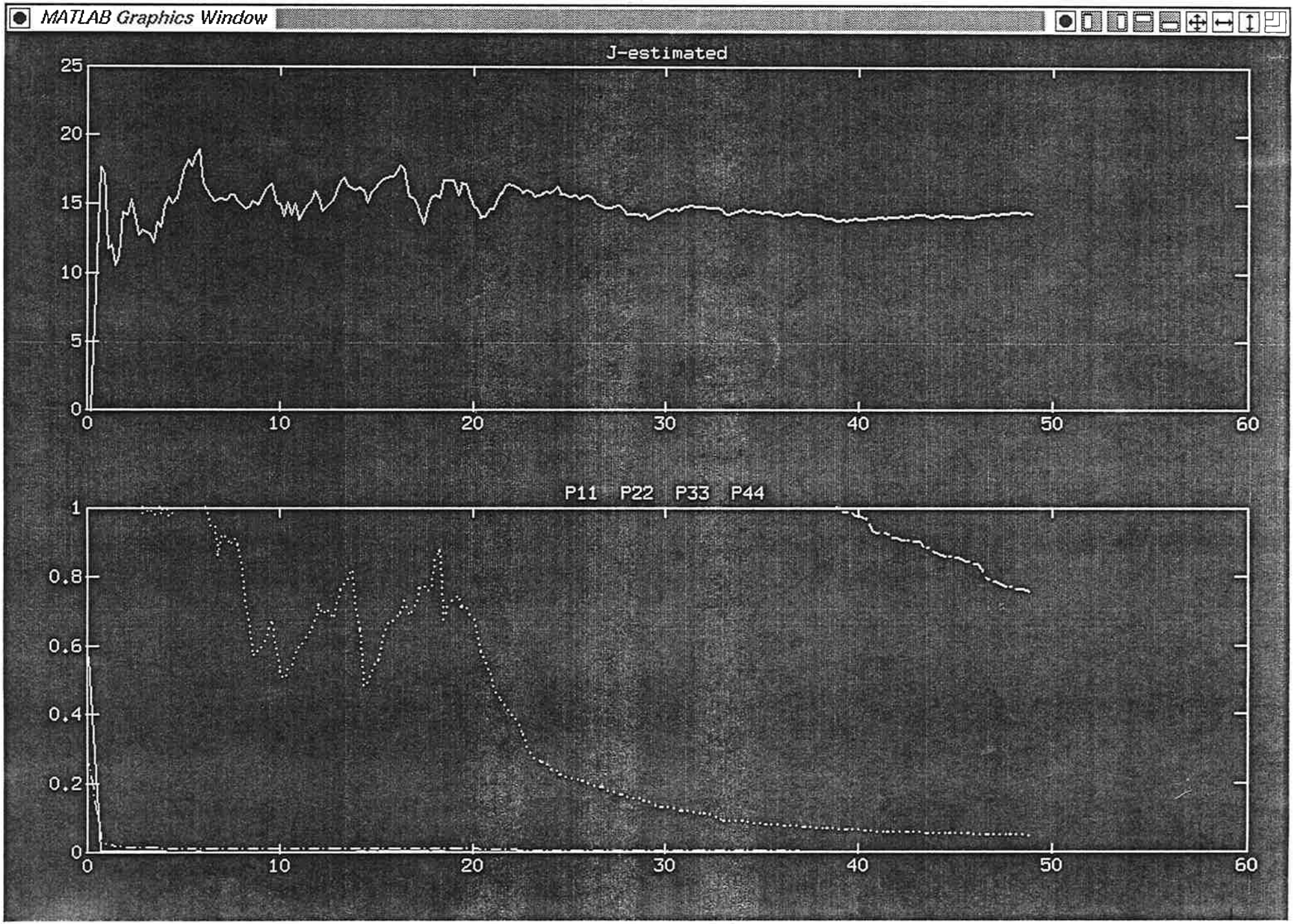
Arbetet har gått bra och roboten klarar av att separera två olika vikter med en viktskillnad på ungefär ett kilo. De problem vi har stött på har i huvudsak haft att göra med det tidigare systemet och inte speciellt med de delar vi implementerat. Mycket av vår tid har därför runnit iväg på sådant som borde ha fungerat från början. Detta har medfört att estimeringen bara har prövats för led ett. Programmet är dock skrivet så att det finns möjlighet att fritt välja led.

För att erhålla bra skattningar var det nödvändigt att under estimeringen koppla bort regulatorernas I–delar. Vid skattning av tunga laster är gripdonet för klent och bidrar till avsevärda skattningsfel.

För att förevisa programmet har ett demonstartionsprogram skrivits som gör en skattning av två olika vikter. Tillämpningen fungerar väl.

Bilaga 1 Skattningen av J och elementen i P–matrisen som funktion av tiden.

Bilaga 2 Demonstrationsprogram.


MATLAB Graphics Window
J-estimated
25
20
15
10
5
0
0
10
20
30
40
50
60
P11 P22 P33 P44
1
0.8
0.6
0.4
0.2
0
0
10
20
30
40
50
60

```
1 POS 950.0 0.0 1150.0 0.0 0.0
3 OPENGRIPPER
5 POS 570.0 870.0 850.0 0.0 0.0
7 WAITTIME 2000.0
9 INITEST 1 R1
13 WAITTIME 2000.0
15 POS 570.0 870.0 700.0 0.0 0.0
17 WAITTIME 2000.0
19 POS 570.0 870.0 650.0 0.0 0.0
21 WAITTIME 2000.0
23 CLOSEGRIPPER
24 WAITTIME 2000.0
25 POS 570.0 870.0 700.0 0.0 0.0
26 WAITTIME 999.0
29 POS 570.0 870.0 850.0 0.0 0.0
30 POS 570.0 870.0 850.0 0.0 3.1
31 WAITTIME 3000.0
32 ESTIM 1 R1 R2
33 DISPR R2
34 SUB R2 1.5 R3
35 JUMPPOS R3 60
37 POS 470.0 890.0 700.0 0.0 0.0
39 WAITTIME 999.0
41 POS 470.0 890.0 655.0 0.0 0.0
43 WAITTIME 2000.0
45 OPENGRIPPER
46 WAITTIME 999.0
47 POS 470.0 890.0 720.0 0.0 0.0
49 WAITTIME 999.0
55 JUMPA 80
60 POS 250.0 920.0 720.0 0.0 0.0
62 WAITTIME 999.0
64 POS 250.0 920.0 690.0 0.0 0.0
66 WAITTIME 999.0
68 OPENGRIPPER
70 WAITTIME 999.0
72 POS 250.0 920.0 750.0 0.0 0.0
74 WAITTIME 999.0
80 POS 950.0 0.0 1150.0 0.0 0.0
90 STOP
```

```
1 POS 950.0 0.0 1150.0 0.0 0.0
3 OPENGRIPPER
5 POS 570.0 870.0 850.0 0.0 0.0
7 WAITTIME 2000.0
9 INITEST 1 R1
13 WAITTIME 2000.0
15 POS 570.0 870.0 700.0 0.0 0.0
17 WAITTIME 2000.0
19 POS 570.0 870.0 650.0 0.0 0.0
21 WAITTIME 2000.0
23 CLOSEGRIPPER
24 WAITTIME 2000.0
25 POS 570.0 870.0 700.0 0.0 0.0
26 WAITTIME 999.0
29 POS 570.0 870.0 850.0 0.0 0.0
30 POS 570.0 870.0 850.0 0.0 3.1
31 WAITTIME 3000.0
32 ESTIM 1 R1 R2
33 DISPR R2
34 SUB R2 1.5 R3
35 JUMPPOS R3 60
37 POS 470.0 890.0 700.0 0.0 0.0
39 WAITTIME 999.0
41 POS 470.0 890.0 655.0 0.0 0.0
43 WAITTIME 2000.0
45 OPENGRIPPER
46 WAITTIME 999.0
47 POS 470.0 890.0 720.0 0.0 0.0
49 WAITTIME 999.0
55 JUMPA 80
60 POS 250.0 920.0 720.0 0.0 0.0
62 WAITTIME 999.0
64 POS 250.0 920.0 690.0 0.0 0.0
66 WAITTIME 999.0
68 OPENGRIPPER
70 WAITTIME 999.0
72 POS 250.0 920.0 750.0 0.0 0.0
74 WAITTIME 999.0
80 POS 950.0 0.0 1150.0 0.0 0.0
90 STOP
```

REGLERING AV ASYNKRONMOTOR

Projekt i Adaptiv reglering och Realtidssystem

VT 1993

Utfört av:
Lars Arvastson
Gustaf von Friesendorff
Hans Olsson
Anders Svensson

Handledare:
Mats Alaküla
Anders Blomdell
Anders Carlsson
Björn Wittenmark

SAMMANFATTNING

Detta kombinerade projekt i Adaptiv reglering och Realtidssystem har handlat om reglering av asynkronmotorer. Att styra asynkronmotorer bra går med fördel att göra med en signalprocessor. I projektet har använts en snabb digital signalprocessor med en specialskriven realtidskärna. Regleringen bygger på att man kan skatta magnetiska flöden i motorn. En enkel metod för skattning av statorflödet bygger på att man känner statortidskonstanten, τ_s. Utifrån dessa uppskattningar kan man beräkna styrsignaler för att reglera motorns moment. Tidskonstanten är dock inte konstant utan ändras med tiden vilket medför att uppskattningarna av de magnetiska flödena inte blir korrekta. Styrsignalerna blir alltså inte optimala.

Vi har estimerat tidskonstanten i realtid med en rekursiv minstakvadratskattare som använder glömskefaktor. Därefter kan man adaptera det filter som ger uppskattningar av flödena. Regulatorn får då bättre utgångsvärden för att beräkna styrsignalerna och regleringen blir bättre.

Innehållsförteckning

1 Inledning

Detta är en projektrapport för ett kombinerat projekt i kurserna Adaptiv reglering och Realtidssystem. Projektet gick ut på att vi skulle förbättra regleringen av en asynkronmotor genom att estimera och adaptera en viktig parameter.

Den regulator vi använder är beroende av en uppskattning av det magnetiska flödet i asynkronmotorn. Uppskattningen får man genom att filtrera asynkronmotorns drivspänningar i ett analogt lågpassfilter med en styrbar tidskonstant τ, som ska motsvara motorns statortidskonstant $\tau_s = L_s/R_s$. Det är denna tidskonstant som vi ska estimera och adaptera.

2 Asynkronmaskinen

Asynkronmaskinen är den vanligaste typen av elektrisk maskin. Den används mest som motor för både små och stora effekter t ex i bandspelare och spårvagnar. Avsaknad av borstar och släpringar gör att maskinen, förutom smörjning av lager, inte behöver något underhåll. Den har dock nackdelen att vara besvärlig att använda i tillämpningar som kräver noggrann varvtalsreglering eller goda servoegenskaper. Detta beror på att det krävs väldigt snabba regulatorer eftersom asynkronmaskinen styrs med växelstorheter med frekvenser upp till 150 Hz. Digitala regulatorer måste då ha sampelfrekvenser i kHz-området, vilket är möjligt med den digitala signalprocessor som har använts under projektet.

2.1 Konstruktion

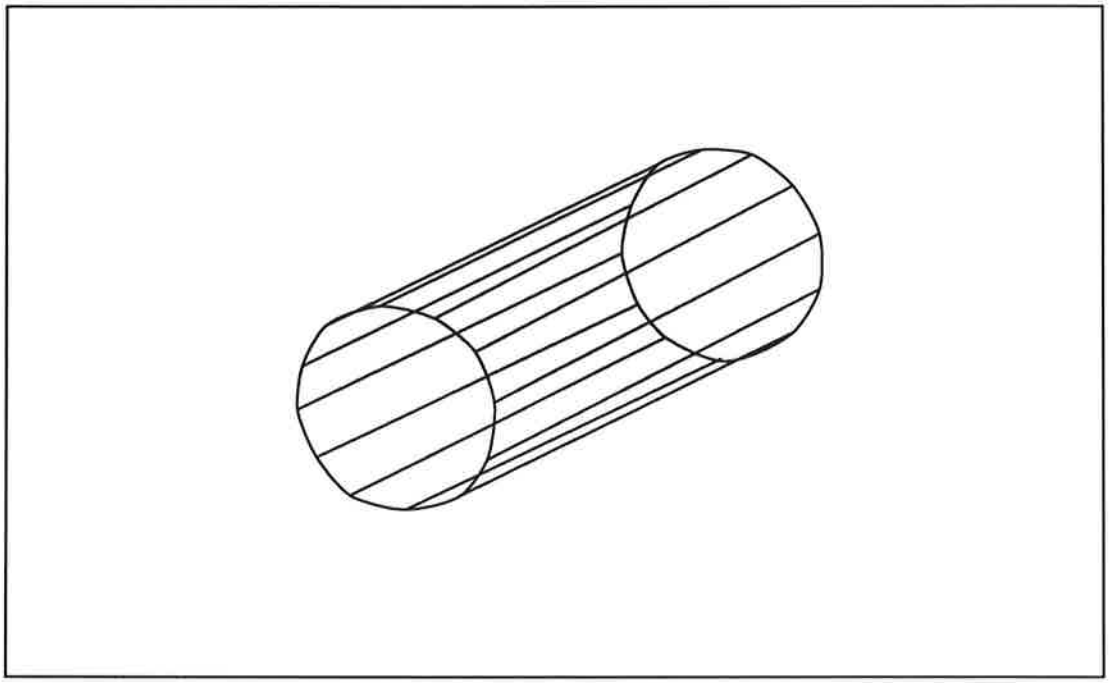
Den vanligaste typen av asynkronmaskin består av en rotor med en kortsluten burlindning, placerad i en stator med tre lindningshärvor. Rotorns lindning liknar en cylindrisk gallerbur eller ett ekorrhjul. Den består oftast av stavar av någon bra ledare som läggs i spår runt rotorn. Därefter löds det på ringar i kortändarna så att ledarna blir kortslutna. Se figur 1.

Statorns lindningshärvor är förlagda så att de bildar tre spolar, fysiskt förskjutna 120 grader från varandra runt motoraxeln. Se figur 2. Normalt ansluts lindningarna till sinuspänningar med 120 graders inbördes fasförskjutning. Detta ger i stationaritet ett roterande flöde, som inducerar strömmar i rotorn. Dessa drar med sig rotorn i flödets rotation.

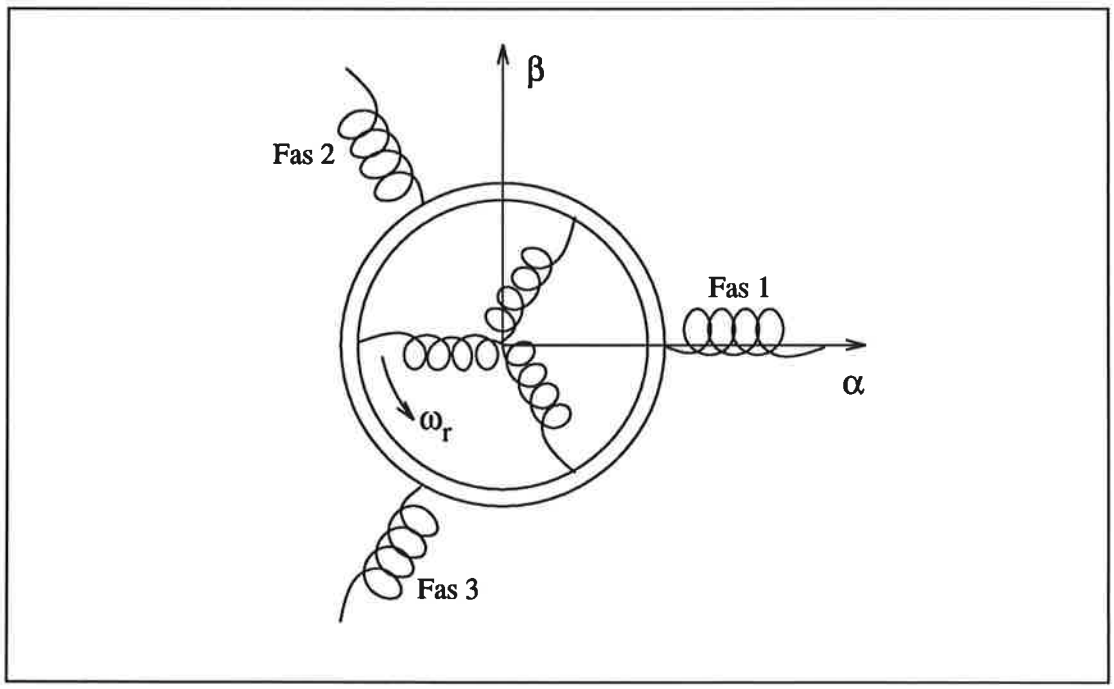
Skillnaden i rotationshastighet mellan statorns magnetfält och rotorn ger upphov till dessa strömmar. Rotorns strömmar bildar i samverkan med luftgapsflödet motorns moment. Rotorn kommer att rotera asynkront dvs med en annan vinkelfrekvens än det roterande flödet, därav namnet. För utförligare beskrivning av asynkronmotorn hänvisas till läroböcker från Institutionen för Industriell Elektroteknik och Automation, [1] och [2].

2.2 Matematisk modell

Vi kan beskriva systemet av flöden och strömmar i asynkronmotorn med följande matematiska modell:



Figur 1: Rotorns burlindning består av stavar av någon bra ledare som kortsluts av ringar i ändarna.



Figur 2: Asynkronmotorns lindningar och statorns koordinatsystem.

$$
\begin{aligned}
\dot{\psi}_{s\alpha} &= u_{s\alpha} - R_s i_{s\alpha} \\
\dot{\psi}_{r\alpha} &= -\omega_r \psi_{r\alpha} - R_r i_{r\alpha} \\
\dot{\psi}_{s\beta} &= u_{s\beta} - R_s i_{s\beta} \\
\dot{\psi}_{r\beta} &= \omega_r \psi_{r\beta} - R_r i_{r\beta}
\end{aligned}
$$

Där index s står för statorn och r för rotorn. Indexen α och β står för ett koordinatsystem som ligger i statorns referenssystem där α är en av fasriktningarna. Se figur 2. R är resistanserna i statorns respektive rotorns lindningar och L är dess induktans. U_s är den pålagda resulterande spänningen och I strömmarna. De olika flödena, Ψ kan bestämmas ur strömmarna på följande sätt:

$$
\begin{aligned}
\psi_{s\alpha} &= L_s i_{s\alpha} + L_m i_{r\alpha} \\
\psi_{r\alpha} &= L_r i_{r\alpha} + L_m i_{s\alpha} \\
\psi_{s\beta} &= L_s i_{s\beta} + L_m i_{r\beta} \\
\psi_{r\beta} &= L_r i_{r\beta} + L_m i_{s\beta}
\end{aligned}
$$

Denna modell var dock alltför stor för våra syften och vi tog därför och försummade rotorströmmarna I_r. Detta är en rimlig approximation speciellt då motorn går på tomgång, eftersom en av våra modellparametrar delvis tar hand om felen som beror på I_r. Vi får då följande förenklade modell:

$$
\begin{aligned}
\dot{\psi}_{s\alpha} &= u_{s\alpha} - R_s i_{s\alpha} \\
\dot{\psi}_{s\beta} &= u_{s\beta} - R_s i_{s\beta} \\
\psi_{s\alpha} &= L_s i_{s\alpha} \\
\psi_{s\beta} &= L_s i_{s\beta}
\end{aligned}
$$

Induktansernas tidsmässiga variation är liten utom då man försöker ändra magnetiseringen. Försummas tidsberoendet fås följande:

$$
\dot{i}_{s\alpha} = -\frac{R_s}{L_s} i_{s\alpha} + u_{s\alpha}/L_s \tag{1}
$$

$$
\dot{i}_{s\beta} = -\frac{R_s}{L_s} i_{s\beta} + u_{s\beta}/L_s \tag{2}
$$

Man kan skriva ekvation 1 på följande sätt i kontinuerlig tid:

$$
u_{s\alpha} = R_s i_{s\alpha} + L_s \dot{i}_{s\alpha} \tag{3}
$$

Man kan även gör en direkt tidsdiskretiseringen av ekvation 1:

$$i_{k+1} = e^{-h/\tau_s} i_k + \Gamma u_k \tag{4}$$

där $\tau_s = L_s/R_s$ vilket är tidskonstanten vi är intresserade av försöka estimera. Vi har här utelämnat indexen $s\alpha$.

Om vi nu tar hänsyn till ω_r kan man se att vi får ett visst bidrag från $u_{s\beta}$ i $i_{s\alpha}$. Eftersom $u_{s\beta}$ är 90° fasförskjutet jämfört med $u_{s\alpha}$ kommer det att medföra viss påverkan på Γ, men knappt någon påverkan på τ_s.

Vid samplingen av ekvation 1 bör man även ta hänsyn till att den regulator som används lägger ut styrsignal två gånger per sampel. Mer om detta kommer i stycke 4.

3 Utrustning

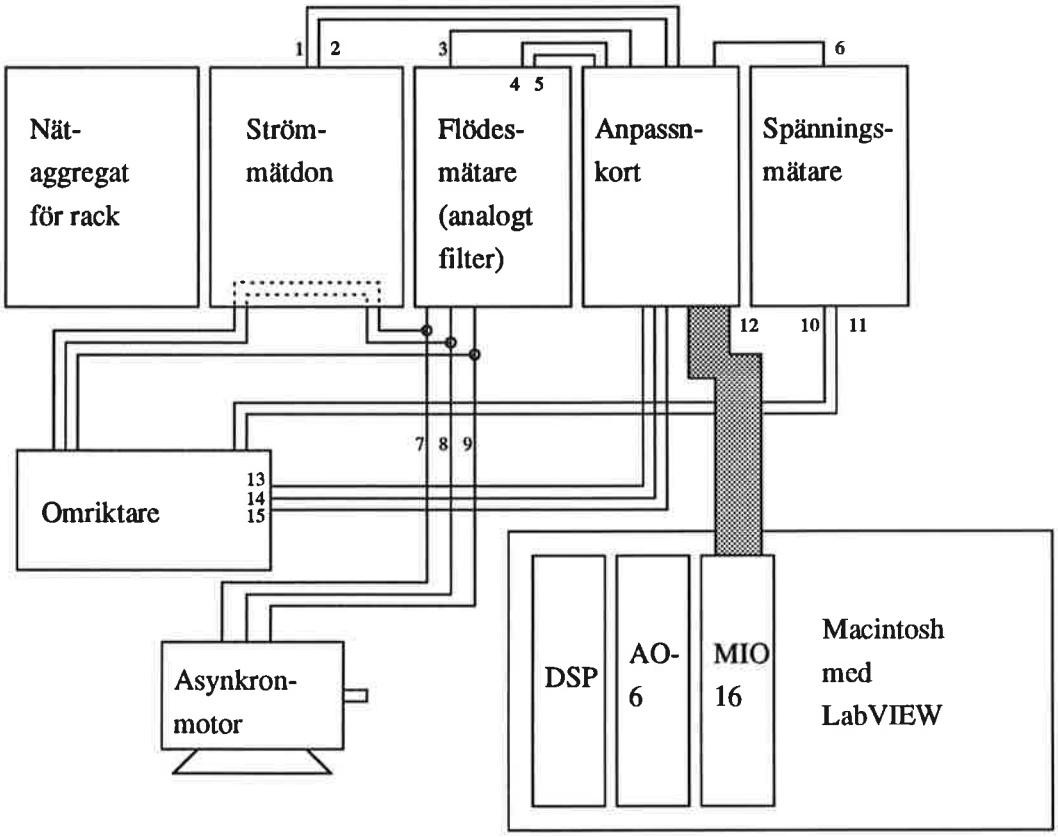
För inställning av regulatorparametrar och presentation av mätdata från asynkronmotorn har vi använt ett program, LabVIEW, som vi har kört från en Macintosh IIfx. Till denna var en digital signalprocessor, med ett antal I/O- och signalanpssningsmoduler kopplad. Dessutom krävs det en del övrig utrustning för att mäta olika storheter och driva asynkronmotorn. Se figur 3.

3.1 LabVIEW

LabVIEW är ett grafiskt programmeringssystem för datainsamling och reglering, dataanalys och datapresentation. Det ger en metod att grafiskt koppla ihop programmoduler som kallas virtuella instrument. Man bygger upp system av virtuella instrument som samlar in data från I/O-kort och därefter analyseras och presenteras dessa data i andra virtuella instrument. Resultatet blir en skärmbild som liknar en verklig instrumentpanel med olika sorters visare och reglage.

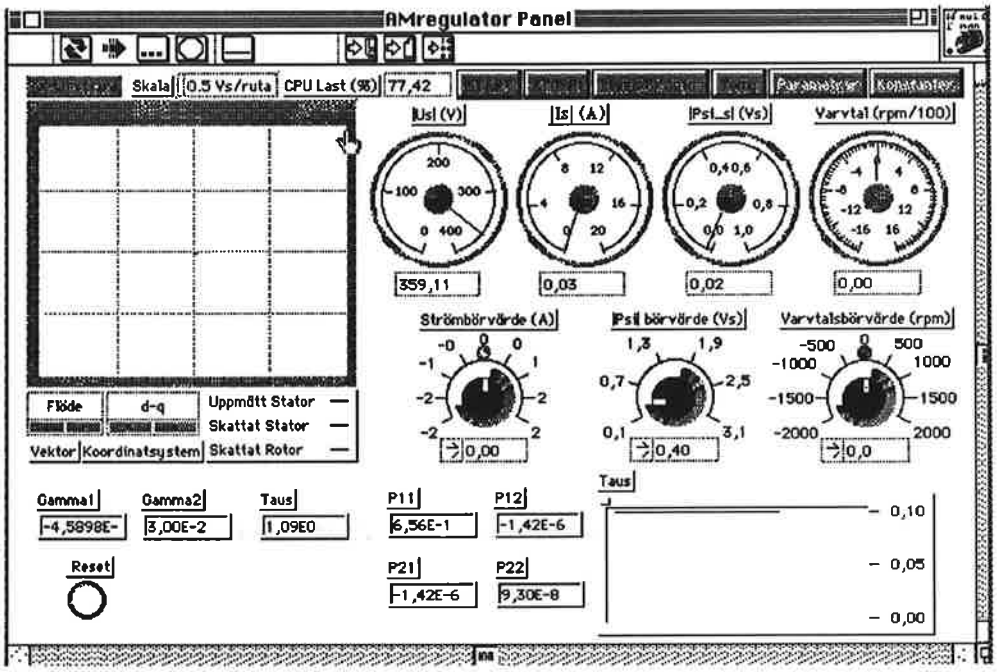
Man skapar ett virtuellt instrument genom att på skärmen först bygga en panel med knappar, rattar, spakar, diagram, visare mm. Denna panel tjänar som ett interaktivt gränssnitt för att ta emot insignaler från användaren och visa utsignaler från systemet. Med LabVIEW blir det lika enkelt att skapa ett virtuellt instrument som att rita en bild. Se fig4. När det virtuella instrumentet är färdigt använder man panelen som sitt styrsystem, även när man kör programmet, genom att röra på reglagen och mata in nya värden och realtidsmässigt studera utsignalerna från systemet.

När man har byggt upp sin egen panel och är nöjd med den, tittar man "bakom" panelen och bygger där upp blockdiagram, helt fria från syntaktiska detaljer i konventionell programmering som t ex i C. Bland olika funktioner, såsom t ex inhämtning av signaler från olika kort, utläggande av signaler, de vanliga räknesätten, sinusgenerering, fouriertransformering, olika sorters filtrering, plocker man ut de som man behöver från olika palettmenyer och binder samman dem med trådar för att överföra data från ett block till ett annat. Se fig5.

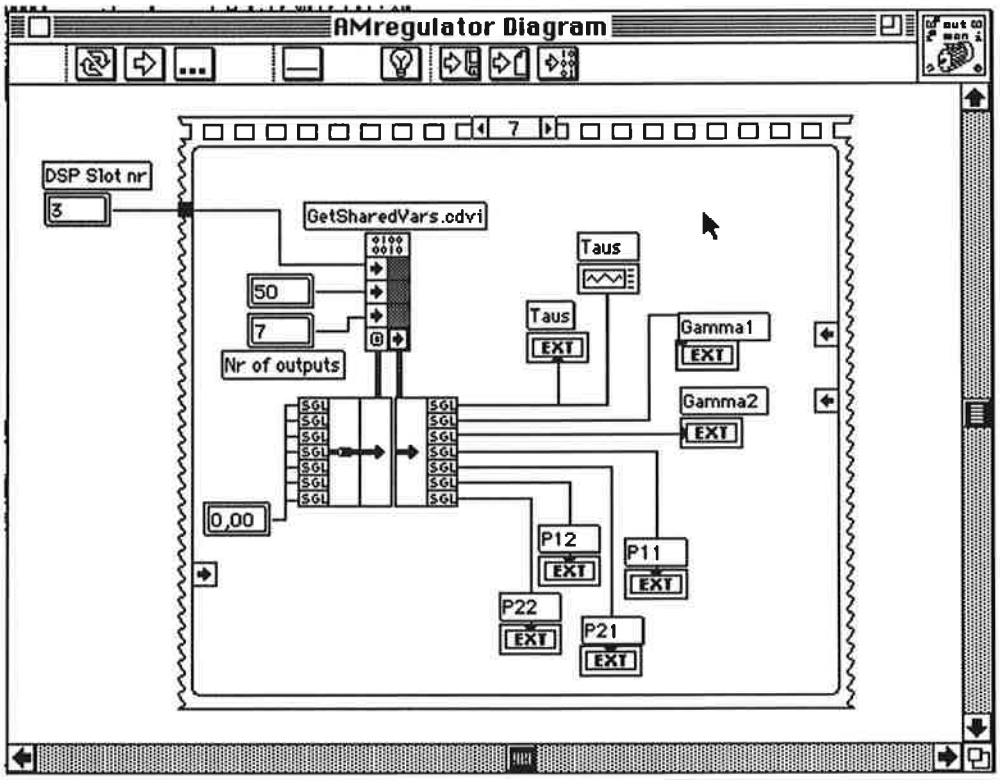


1. Ström i fas 1, Ia. Från strömmätare till anpassningskort.
2. Ström i fas 2, Ib. Från strömmätare till anpassningskort.
3. Styrsignal för tidskonstant. Från anpassningskort tillflödesmätare.
4. Skattat flöde i alfa-led. Från flödesmätare till anpassningskort.
5. Skattat flöde i beta-led. Från flödesmätare till anpassningskort.
6. Spänningen Ud. Från spänningsmätare till anpassningskort.
7. Fas 1. Från omriktare, via strömmätare och flödesmätare till asynkronmotorn.
8. Fas 2. Från omriktare, via strömmätare och flödesmätare till asynkronmotorn.
9. Fas 3. Från omriktare via flödesmätare till asynkronmotorn.
10. Maxspänning hos omriktaren. Till spänningsmätare.
11. Jordnivå hos omriktaren. Till spänningsmätare.
12. Flatkabel mellan anpasssningskort och I/O-kort.
13. Styrsignal för fas 1. Från anpassningskort till omriktare.
14. Styrsignal för fas 2. Från anpassningskort till omriktare.
15. Styrsignal för fas 3. Från anpassningskort till omriktare.

Figur 3: Uppställningen.



Figur 4: Vår frontpanel.



Figur 5: Bakom en del av panelen.

Man kan även skapa egna funktioner och bygga upp underpaneler för att studera hur signalen behandlas innan den kommmer upp till översta nivån. På det här sättet kan man flexibelt bygga upp moduler av virtuella instrument. I vårt projekt har vi modifierat en befintlig frontpanel där vi kan starta asynkronmotorn och studera våra olika skattningar samt en panel där vi kan ändra olika parametrar hos regulator och estimator.

3.2 Signalprocessor med realtidskärna

För att klara av tillräckligt snabb reglering har en snabb digital signalprocessor, DSP, använts. Signalprocessor är en TMS320C30, tillverkad av Texas Instruments och sitter på ett instickskort tillsammans med en del nödvändig kringutrustning bl a NuBus-gränssnitt, RAM-minne till signalprocessorn och gränssnitt till I/O-korten.

De I/O-kort som används är ett NB-MIO-16 och ett NB-AO-6 kort. MIO-16-kortet är ett 12-bitars multifunktions I/O-kort som har 16 analoga ingångar, två analoga utgångar samt åtta TTL-kompatibla digitala I/O och tre 16-bitars counter/timer kanaler. Eftersom vi behöver mer än två analoga utsignaler används även ett AO-6-kort med maximalt sex analoga utgångar. Såväl kortet med signalprocessorn som I/O-korten är tillverkade av National Instruments.

Signalprocessornn har en specialskriven realtidskärna kallad Ärkenöt som innehåller hjälpfunktioner för synkronisering och kommunikation vid användande av parallella processer. Dessa hjälpfunktioner är Semaforer, Händelser, Monitorer och Brevlådor. För mer information om Ärkenöt hänvisas till A. Carlssons kommande exjobbsrapport [3]. I början var det problem med avbrottshanteringen, men detta åtgärdades under projektets gång. En orsak till att detta problem inte hade upptäckts innan var att vi hade högre belastning (CPU-utnyttjande) på signalprocessorn än vid tidigare tester.

3.3 Kringelektronik (burkarna bredvid)

Utöver signalprocessorn och I/O-korten finns det en del övrig elektronik för att driva asynkronmotorn och utföra olika mätningar. De flesta av dessa moduler sitter samlade i ett rack och har gemensam strömförsörjning via ett gemensamt nätaggregat. Härmed har de även samma jordnivå för signalerna.

I detta rack sitter ett anpassningskort som anpassar signaler från de olika mätarna till I/O-kortens krav. Det anpassar även I/O-kortens utsignaler till styrelektroniken.

I racket sitter ett strömmmätdon som mäter strömmen i två av de tre faserna som används för drivning av asynkronmotorn. Den tredje strömmen behöver inte mätas eftersom den kan beräknas ur de båda andra med Kirchoffs lag. Mätvärdena skickas till anpassningskortet för att efter anpassning skickas till I/O-korten.

Racket innehåller även ett variabelt filter som ger en uppskattning av den magnetiska flödesvektorn i asynkronmotorns stator. Det som filtreras är de tre faserna hos drivspänningarna. Ut får man skattningen av värdet på det magnetiska flödet i två ortogonala riktningar vilka skickas till anpassningskortet.

För att skattnigen ska bli bra måste filtrets tidskonstant stämma överens med den aktuella asynkronmotorns statortidskonstant, τ_s.

För att regulatorn ska veta vilken spänning som maximalt kan användas för att driva asynkronmotorn, mäts den maximala spänning U_d som omriktaren kan ge av en spänningsmätare. Spänningsmätaren skickar värdet som en analog signal till anpassningskortet.

Utanför racket finns en omriktare som ger drivspänning till asynkronmotorn. Denna styrs av tre olika signaler från anpassningskortet, en för varje fas. Signalerna är pulsbreddsmodulerade. Dess utseende beräknas redan i signalprocessorn. Omriktaren ger en pulsbreddsmodulerad trefas drivspänning till asynkronmotorn.

Sist men inte minst finns naturligtvis även en asynkronmotor i uppställningen. Vi har testkört med två asynkronmotorer med olika effekter, 2,2 kW respektive 0,12 kW. Estimeringen av τ_s fungerade lika bra i båda fallen.

4 Estimering

För att estimera $\tau_s = L_s/R_s$, dvs statorns tidskonstant har vi använt en rekursiv minstakvadrat-skattare. Med denna har vi estimerat parametrar i ett antal olika matematiska modeller för asynkronmaskinen.

4.1 Modeller

Vi började med att försöka estimera R_s och L_s utgående från ekvation 3:

$$u_{s\alpha} = R_s i_{s\alpha} + L_s \dot{i}_{s\alpha} \tag{5}$$

där vi först använde Ψ-skattningar från det analoga filtret för att räkna ut $u_{s\alpha}(k) = (\psi_{s\alpha}(k) - \psi_{s\alpha}(k-1))/h$. Eftersom vi hade filtrerade värden gick det bra att bestämma $\dot{i}_{s\alpha}$ med zero order-hold sampling. Vi hade nu problem med att det analoga filtrets tidskonstant inte var helt känd, och vi hade därmed något olika filtrering av $u_{s\alpha}$ och $i_{s\alpha}$. Detta gav inga bra skattningar, så vi gick över till att använda den styrsignal som användes i början av samplet eftersom denna bör vara ganska nära det verkliga $u_{s\alpha}$. Vi fick nu ett τ_s som var någorlunda korrekt medan R_s och L_s troligen är multiplicerade med en okänd konstant. Eftersom skattningarna inte var särskilt bra skrev vi om formeln till

$$\dot{i}_{s\alpha} = \frac{1}{L_s} u_{s\alpha} - \frac{R_s}{L_s} i_{s\alpha}. \tag{6}$$

Därefter samplade vi denna med zero-order hold vilket gav

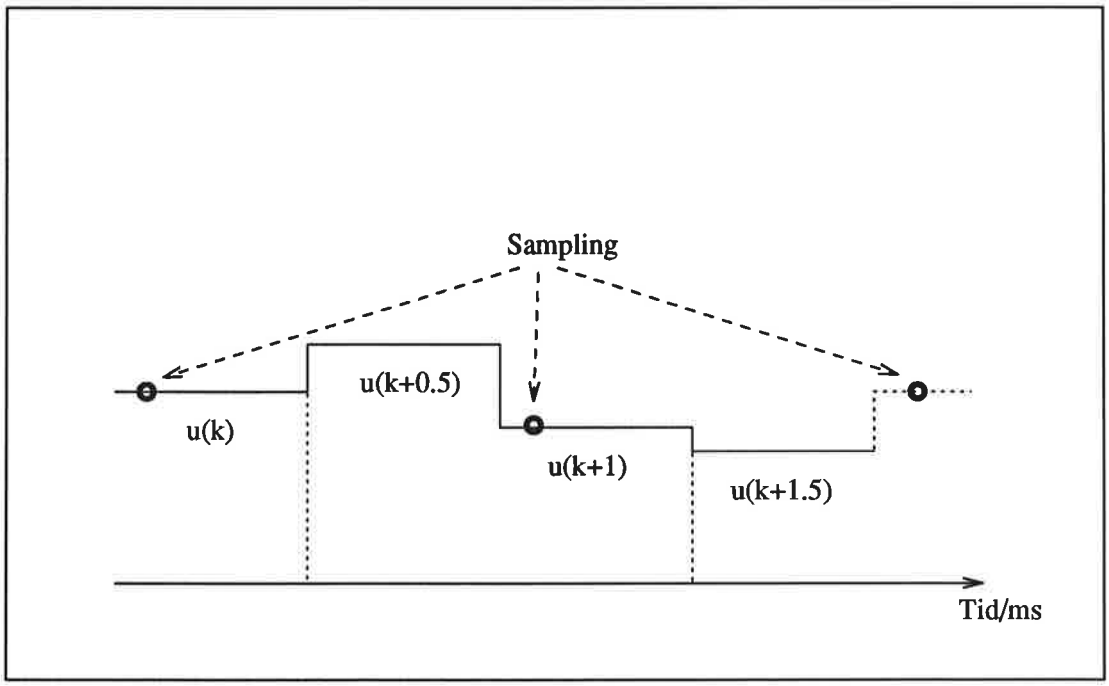
$$i_{k+1} = e^{-h/\tau_s} i_k + \Gamma u_k. \tag{7}$$

Denna ekvation gav inte heller perfekta estimat, men vi var inte längre tvungna att filtrera $i_{s\alpha}$. Att estimaten inte blir så bra beror på att vi ändrar styrsignalen mellan mätningarna.

Vi fastnade därför slutligen för en modell som tar hänsyn till detta.

$$i_{k+1} = e^{-h/\tau_s} i_k + \Gamma_0 u_k + \Gamma_1 u_{k+0.5} \tag{8}$$

där vi använder en gammal mätning av i och två gamla styrsignaler u_k och $u_{k+0.5}$. Den nuvarande styrsignalen u_{k+1} har inte hunnit verka så lång tid innan samplingen sker , så vi försummar dess inverkan. Se figur 6.



Figur 6: Sampling och utsignaler.

Vi använder alltså

$$i_{k+1} = (\theta_1 \quad \theta_2 \quad \theta_3) \cdot \begin{pmatrix} i_k \\ u_k \\ u_{k+0.5} \end{pmatrix}. \tag{9}$$

Ur estimatet av θ_1 kan man få τ_s genom

$$\tau_s = -\frac{h}{\ln \theta_1}. \tag{10}$$

Vi misstänker att det skulle vara bra att även utnyttja u_β, eftersom vi vid vissa driftsituationer får värden på Γ som tyder på att det finns ett bidrag som är 90° fasförskjutet. Innan vi tar med u_β bör vi dock hitta ett samband mellan Γ_1 och Γ_2 eftersom vi annars kan få problem med att skatta fyra Γ. Det är inte säkert att vi får tillräcklig excitation i systemet.

4.2 Estimatorn

Den metod som vi har använt är rekursiv minstakvadrat-metod med glömskefaktor. Den kan i matrisform skrivas enligt

$$\hat{\theta}(t) = \hat{\theta}(t-1) + K(t)(y(t) - \phi^T(t)\hat{\theta}(t-1)) \tag{11}$$

$$K(t) = P(t)\phi(t) = P(t-1)\phi(t)(\lambda I + \phi^T(t)P(t-1)\phi(t))^{-1} \tag{12}$$

$$P(t) = (I - K(t)\phi(t))P(t-1)/\lambda. \tag{13}$$

Dessa formler har implementerats enligt en optimerad algoritm på sidan 90 i [4] där ett teckenfel vid uppdateringen av skattningen ger problem. Vi hade i början också vissa problem med att P-matrisen exploderade efter en tid, vilket berodde på att vår kod gav en aning asymmetrisk P-matris (beroende på avrundningsfel) och algoritmen är tydligen instabil för asymmetriska P-matriser. Vi misstänker att det troligen hade fungerat längre om signalprocessorn hade räknat med fler värdesiffror. Problemet kan lösas genom att göra uträkningen av P-matrisen symmetrisk, vilket vi gjorde, eller genom att bara räkna ut den övre triangulära matrisen och sätta de nedre elementen lika med de övre.

5 Regulatorn

Regulatorn som reglerar asynkronmotorn är egentligen två olika PI-regulatorer. Det som regleras är momentet hos motorn, dvs vektorprodukten av det magnetiska flödet i statorn, Ψ_s, och strömmen i statorn, I_s. Den ena regulatorn reglerar flödet i statorn och den andra strömmen vinkelrätt mot flödet. Momentet M blir

$$M = \Psi_s \times I_s. \tag{14}$$

Det som används som indata till flödesregulatorn är en uppskattning av det magnetiska flödet uppdelat i två mot varandra vinkelräta riktningar ($\psi_{s\alpha}$ och $\psi_{s\beta}$) vilka fås ur ett analogt filter med en tidskonstant som ska vara så nära motorns statortidskonstant τ_s som möjligt. Det är denna tidskonstant som vi estimerar och justerar.

Indata till strömregulatorn är statorströmmen. Den mäts i två av de tre faserna(i_a och i_b) och omvandlas sedan till tvåfasrepresentation i två vinkelräta riktningar enligt

$$i_{s\alpha} = \sqrt{\frac{3}{2}} \cdot i_a \tag{15}$$

$$i_{s\beta} = \sqrt{\frac{1}{2}} \cdot (i_a + 2i_b). \tag{16}$$

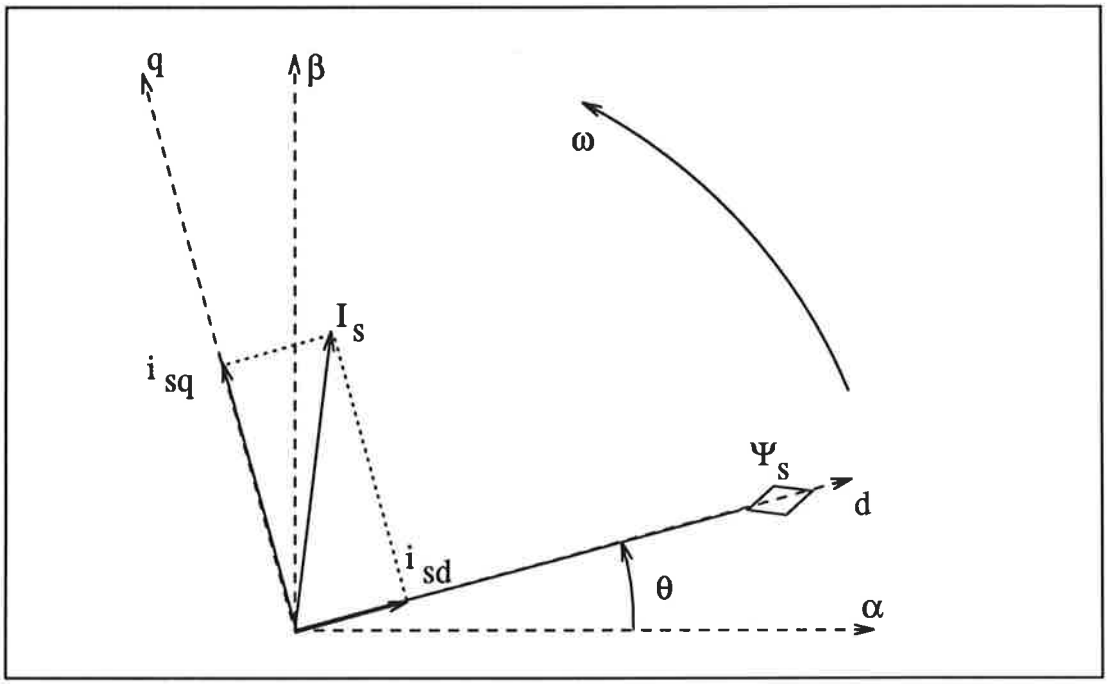
För att underlätta beräkningarna i regulatorn transformeras det magnetiska flödet och strömmen till ett koordinatsystem, d-q-systemet , där flödesvektorn sammanfaller med d-koordinataxeln. Se figur 7. Strömvektorn transformeras enligt

$$i_{sd} = i_{s\alpha} \cdot \cos\theta + i_{s\beta} \cdot \sin\theta \tag{17}$$

$$i_{sq} = i_{s\beta} \cdot \cos\theta - i_{s\alpha} \cdot \sin\theta, \tag{18}$$

där θ är flödesvektorns vinkel i α-β-koordinater. Den intressanta strömkomponenten är i_{sq}, som är ortogonal mot flödesvektorn.

Regulatorerna beräknar styrspänning i d-q-systemet, flödesregulatorn en spänningskomponent i d-led och strömregulatorn spänningskomponeneten i q-led. Den resulterande styrspänningsvektorn transformeras tillbaka till α-β-koordinater och vrids en vinkel som motsvarar den förväntade vridningen av



Figur 7: Koordinatsystemen i asynkronmotorn.

den önskade styrsignalen. Efter halva sampelintervallet läggs en ny styrspänning ut. Denna är då vriden en vinkel som motsvarar mittpunkten för den önskade styrsignalen under det intervallet.

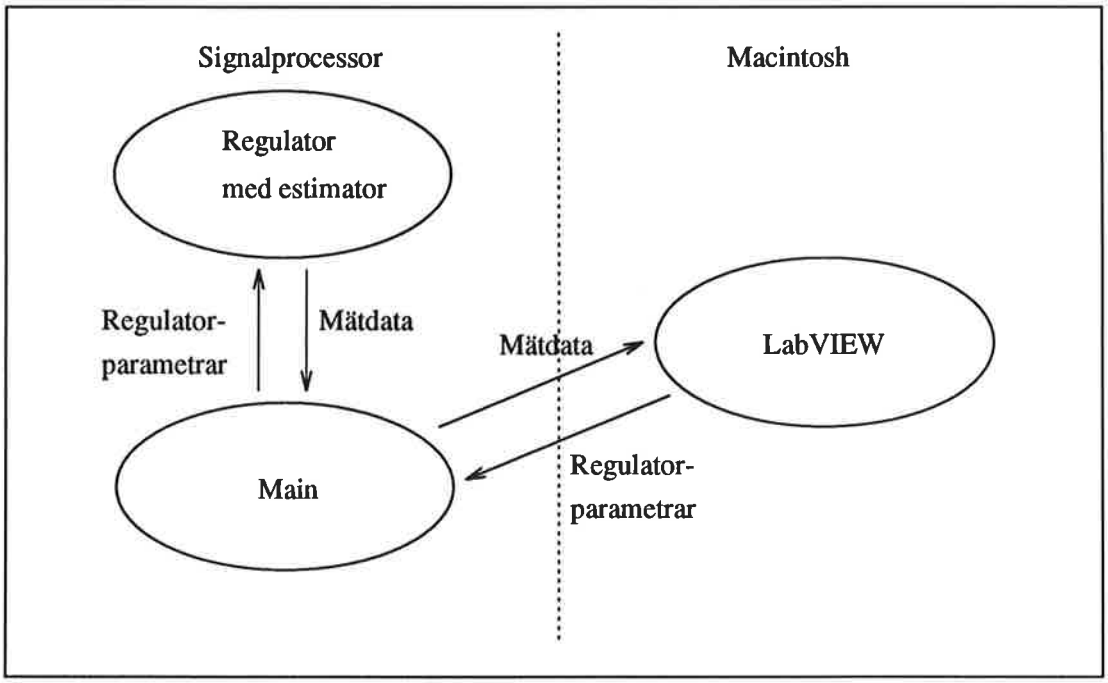
Styrspänningarna transformeras från tvåfas-representation till trefasrepresentation och pulsbreddsmoduleras innan de skickas ut för att styra omriktaren.

6 Program

De delar av programmet som vi har modifierat är skrivna i C. Utöver C's standardkommandon finns en realtidskärna som beskrivs i [3]. I signalprocessorn har vi två parallella processer, huvudprocessen som sköter kommunikationen med LabVIEW och regulatorprocessen som reglerar asynkronmotorn. Parallellt med dessa två processer körs LabVIEW i Macintoshen. Processgrafen kan ritas enligt figur 8 där LabVIEW innehåller flera processer.

Huvudprogrammet main.c innehåller uppstart av regulatorn och överföringen av data mellan regulator och LabVIEW. Det börjar med att kontrollera att LabVIEW har startat och hämtar därefter en del initieringsdata och startar regulatorn. I huvudloopen läser main in börvärden, regulatorparametrar och omvandlingsfaktorer från LabVIEW samt skriver diverse processdata och estimatorparametrar till LabVIEW.

Överföringen av parametrar från main till AMcontroller sköts genom att AMcontroller uppdaterar vissa viktiga interna parametrar en gång per loop om de skiljer sig från de globala. Övriga data klarar sig eftersom tilldelningssatsen är atomär.



Figur 8: Systemets processgraf.

7 Resultat

När vi hade fått systemet att fungera testade vi estimatorn på två olika asynkronmotorer. Den större av de två motorerna var på 2,2 kW och den mindre på 0,12 kW. Dessa hade olika statortidskonstanter. Den större motorns tidskonstant estimerades till ungefär 100 ms och den mindre motorns till 10 ms. Den mindre motorn hade en teoretisk tidskonstant på 8 ms, vilket stämmer väl med vårt estimat. När vi plötsligt ändrade rotationsriktning hos motorn syntes en topp och en dal i estimatet. En orsak är troligen att vi inte tar hänsyn till påverkan av den ortogonala spänningen när vi estimerar.

8 Avslutande kommentarer

Efter många om och men lyckades vi få vår uppställning att fungera och få fram modeller som gav oss rimliga resultat. Nästa steg är att adaptivt förbättra uppskattningen av flödesvektorn, antingen genom att använda ett från datorn styrbart filter eller genom att ersätta det analoga filtret med ett digitalt filter i signalprocessorn.

Vi vill avslutningsvis tacka våra handledare, framför allt Anders Carlsson som visade att mätkorten inte klarade 220V och därmed gav vårt projekt ett väldefinierat slut.

Referenser

[1] Gustaf Olsson (1991) *Elmaskinsystem.* Lunds Tekniska Högskola.

[2] Mats Alaküla, Gustaf Olsson och Tore Svensson (1990) *Styrning av elektriska drivsystem.* Lunds Tekniska Högskola.

[3] Anders Carlsson (1993) *Ärkenöt.* Examensarbete vid Lunds Tekniska Högskola.

[4] Karl Johan Åström, Björn Wittenmark (1989) *Adaptive Control.* Addison-Wesley.

A Programkod

93-04-26 16.46 Archimedes-1:User:DSPprojekt932:main.c

```
/*
 * main.c
 *
 * Denna fil innehåller huvudprogrammet, som sköter om uppstartsförlopp,
 * samt överföring av data mellan LabVIEW och DSP:n
 */

#include <peripherials:toIEEEtoC30.h>
#include <peripherials:shared_data.h>
#include <peripherials:shared_variables.h>
#include <kern:time.h>
#include <kern:sys_clock.h>
#include <kern:timer_event.h>
#include <libc:stdio:stdio.h>
#include <math.h>
#include "AMcontroller.h"

void main() {
    event_t     mainTimer;
    timevalue   firsttime,interval;
    IEEEfloat   fpInput;
    long        intInput;
    float       Kp,Ti,x,fpOutput;

    /*
     * Signallera till LabVIEW att kärnan har bootat färdigt.
     */
    shared_variable_give(0);

    /*printf("Hit kom jag!, väntar på signal\r");
    */

    /*
     * Vänta tills LabVIEW har överfört alla omvandlingsfaktorer
     */
    while (shared_variable_read(1) == 0);

    /*printf("Hit kom jag!, har just fått signal\r");
    */
    /*
     * Hämta Omvandlingsfaktorerna
     */
    read_shared_data(20,fpInput,IEEEfloat);
    Ky = IEEEtoC30(fpInput);
    read_shared_data(21,fpInput,IEEEfloat);
    Kpsi = IEEEtoC30(fpInput);
    read_shared_data(22,fpInput,IEEEfloat);
    Ki = IEEEtoC30(fpInput);
    read_shared_data(23,fpInput,IEEEfloat);
    Ku = IEEEtoC30(fpInput);
    read_shared_data(24,fpInput,IEEEfloat);
    Komega = IEEEtoC30(fpInput);

    /*
     * Starta regulatorerna
     */

    /*printf("Startar regulatorerna, ström/flöde först\r");
    */
    AMControllerInit();

    gettimeofday(&firsttime);

    interval.tv_sec = 0;
    interval.tv_usec = 10000; /* 10 ms */

    timeradd(&firsttime,&interval);
    mainTimer = create_timer_event(NULL,&firsttime,&interval);

    /*printf("Går in i huvudloopen\r");
    */
    for(;;){
        event_wait(mainTimer,0);


        /*
         * Läs in börvärden från LabVIEW
         */

        read_shared_data(1,fpInput,IEEEfloat);
        SetFluxSetpoint(IEEEtoC30(fpInput));


        /* read_shared_data(5,intInput,long);
        start_plots = IntInput; */
        read_shared_data(5,intInput,long);
        if (IntInput) ResetRLS();

        /*
         * Läs in regulatorparametrar
         */
        read_shared_data(10,fpInput,IEEEfloat);
        Kp = IEEEtoC30(fpInput);
        read_shared_data(11,fpInput,IEEEfloat);
        Ti = IEEEtoC30(fpInput);
        SetCurrentControllerParams(Kp,Ti);

        read_shared_data(12,fpInput,IEEEfloat);
        Kp = IEEEtoC30(fpInput);
        read_shared_data(13,fpInput,IEEEfloat);
        Ti = IEEEtoC30(fpInput);
        SetFluxControllerParams(Kp,Ti);

        {
            int algor;
            float lambda;
            read_shared_data(16,intInput,long);
            algor=intInput;
            read_shared_data(17,fpInput,IEEEfloat);
            lambda=IEEEtoC30(fpInput);
            SetRLSParams(algor,lambda);
        }

        /*
         * Hämta Omvandlingsfaktorerna
         */
        read_shared_data(20,fpInput,IEEEfloat);
        Ky = IEEEtoC30(fpInput);
        read_shared_data(21,fpInput,IEEEfloat);
        Kpsi = IEEEtoC30(fpInput);
        read_shared_data(22,fpInput,IEEEfloat);
        Ki = IEEEtoC30(fpInput);
        read_shared_data(23,fpInput,IEEEfloat);
        Ku = IEEEtoC30(fpInput);
        read_shared_data(24,fpInput,IEEEfloat);
        Komega = IEEEtoC30(fpInput);
```

```
            /*
             * Skriv ut plotdata, först alla i alfa-beta koordinater
             */
            write_shared_data(0,C30toIEEE(PsiS_beta),IEEEfloat);
            write_shared_data(1,C30toIEEE(PsiS_alfa),IEEEfloat);

            write_shared_data(6,C30toIEEE(IS_beta),IEEEfloat);
            write_shared_data(7,C30toIEEE(IS_alfa),IEEEfloat);

            write_shared_data(12,C30toIEEE(Y_beta),IEEEfloat);
            write_shared_data(13,C30toIEEE(Y_alfa),IEEEfloat);

            /*
             * Sedan alla i D-Q koordinater (d.v.s koordinater relativt huvudflödet)
             */
            write_shared_data(22,C30toIEEE(PsiS_q),IEEEfloat);
            write_shared_data(23,C30toIEEE(PsiS_d),IEEEfloat);
            write_shared_data(26,C30toIEEE(IS_q),IEEEfloat);
            write_shared_data(27,C30toIEEE(IS_d),IEEEfloat);
            write_shared_data(32,C30toIEEE(Yq),IEEEfloat);
            write_shared_data(33,C30toIEEE(Yd),IEEEfloat);

            /*
             * RMS-värden på spänning ström och flöde, samt varvtalet
             */
            write_shared_data(40,C30toIEEE(sqrt(Yq*Yq + Yd*Yd)/Tsamp),IEEEfloat);
            write_shared_data(41,C30toIEEE(sqrt(IS_d*IS_d + IS_q*IS_q)),IEEEfloat);
            write_shared_data(42,C30toIEEE(PsiS_d),IEEEfloat);

            /*
             * CPU load average kan också vara intressant att kunna visa
             */
            write_shared_data(45,C30toIEEE(100.0*cpu_load_average()),IEEEfloat);

        /* Skattaren av parametrar */
            write_shared_data(50,C30toIEEE(My_Taus),IEEEfloat);
            write_shared_data(51,C30toIEEE(Gamma1),IEEEfloat);
            write_shared_data(52,C30toIEEE(Gamma2),IEEEfloat);
            write_shared_data(53,C30toIEEE(P11),IEEEfloat);
            write_shared_data(54,C30toIEEE(P12),IEEEfloat);
            write_shared_data(55,C30toIEEE(P21),IEEEfloat);
            write_shared_data(56,C30toIEEE(P22),IEEEfloat);
        }
    }
```

```
/*
 * AM controller.h
 *
 * Den här filen defenierar gränssnittet till asynkronmaskinsregulatorn
 * vars implementation finns i AMcontroller.c
 */

#ifndef __AM_controller
#define __AM_controller

#include <kern:types.h>

/*
 * Samplingsintervall, i µs.
 */
#define SAMPLE_INTERVAL 1000

/*
 * Globala variabler, som skall vara synliga utåt
 */

/*
 * Omvandlingsfaktorer när vi hämtar dem från LabVIEW.
 */
extern float    Ky,Kpsi,Ki,Ku,Komega;

/*
 * Uppmätta spänningar/flöden
 */
extern float    PsiS_sp;
extern float    PsiS_alfa;
extern float    PsiS_beta;
extern float    PsiS_d_qsc;
extern float    PsiS_q_qsc;

/*
 * U-t ytor
 */
extern float    Y_alfa,Y_beta;
extern float    Yd,Yq;

/*
 * Statorström i alfa-beta och d-q -led
 */
extern float    IS_alfa,IS_beta;
extern float    IS_d,IS_q,IS_q_sp;

/*
 * Skattade Stator och rotorflöden, även dessa både i alfa-beta och d-q -led
 */
extern float    PsiS_d,PsiS_q;

/* Skattade parametrar i motorn */
extern float Gamma1,Gamma2;
extern float    My_Taus;
extern float    P11;
extern float    P12;
extern float    P21;
extern float    P22;
/*
 * Övrigt
 */
extern float    Sin_x,Cos_x;
extern float    Ud;     /* Strömriktarens mellanledsspänning */
extern float    Tsamp;  /* samplingsintervall i sekunder */

extern boolean_t    start_plots;

#define SetCurrentSetpoint(_I) (IS_q_sp = (_I))
#define SetFluxSetpoint(_Psi) (PsiS_sp = (_Psi))

/*
 * Funktioner
 */

void AMControllerInit();
void SetFluxControllerParams(float Kp,float Ti);
void SetCurrentControllerParams(float Kp,float Ti);
void SetRLSParams(int algor,float lambda);
void ResetRLS(void);
#endif
```

```
/*
 * AMcontroller.c
 *
 * Denna fil innehåller den tråd som sköter insamling och enhetsomvandling
 * av mätdata, beräkning av ledvärden utifrån estimerade flöden
 * samt enhetsomvandling och utmatning av nya ledvärden
 */


/* om definerad så är regulatorn igång (annars bara sinus) */

#include <kern:thread.h>
#include <kern:time.h>
#include <kern:timer_event.h>
#include <drivers:nb_mio_16.h>
#include <drivers:nb_ao_6.h>
#include <math.h>
#include "config.h"
#include "machine_constants.h"
#include "AMcontroller.h"
#include <peripherials:dsp2300_regs.h>

#define __USE_NUBUS_ATOMICS

/*
 * Några konstanter som vi har användning för
 */
#define Sqrt_three_half 1.224744871
#define Sqrt_one_half   0.7071067812
#define Sqrt_two_thirds 0.8164965809
#define Sqrt_three      1.732050808

#ifndef PI
#define PI  3.1415926535
#endif

/*
 * Omvandlingsfaktorer när vi hämtar dem från LabVIEW.
 */
float   Ky,Kpsi,Ki,Ku,Komega;

/*
 * Uppmätta spänningar/flöden
 */
float   PsiS_sp = 0.2;
float   PsiS_alfa = 0.0;
float   PsiS_beta = 0.0;
float   PsiS_d_qsc = 0.0;
float   PsiS_q_qsc = 0.0;

float   PsiS_d = 0.0;
float   PsiS_q = 0.0;
float   Gamma1=0;
float   Gamma2=0;
float   My_Taus=0.070;
float   P11=1;
float   P12=1;
float   P21=1;
float   P22=1;

/*
 * U-t ytor
 */
float   Y_alfa = 0.0,Y_beta = 0.0;
float   Yd,Yq;

/*
 * Parametrar till flödesregulatorn
 */
float   Kp_psi = 0.2;
float   Ti_psi,Tr_psi,Ki_psi = 0.0,Kr_psi = 0.0;

/*
 * Statorström i alfa-beta och d-q -led
 */
float   IS_alfa,IS_beta;
float   Is_d,IS_q,IS_q_sp = 0.0;

/*
 * Parametrar till tvärströmsregulatorn
 */
float   Kp_I = 0.0;
float   Ti_I,Tr_I,Ki_I = 0.0,Kr_I = 0.0;

/*
 * Övrigt
 */
float   Sin_x = 0.0,Cos_x = 0.0;
float   Ud;
float   Tsamp,Tsamp_inv;    /* samplingsintervall i sekunder */
float   efactor,efactor2; /* Borde egentligen analysera två olika U */
float   TauS = 0.010; /* Som uppmätt */
float   TauS_inv;

static float lambda_inv;
static float lambda=0.9999;
static float newlambda=0.9999;
static int algorithm=0; /* Bit 0: ger order, Bit 1: ger om styrsignal eller konstant */

boolean_t   start_plots = FALSE;

mio_16  inboard;
ao_6    outboard;

event_t controlTimer;


/*
 * Konstanter som hör till modulatorn
 */
#define FUN_CODE        0x0062  /* PWM = 0x0062, One-shot = 0x0042 */
#define SOURCE_CODE     0x0100  /* 5 MHz external source */
#define PULSE_PERIOD    2500    /* 500 µs */
#define MIN_INTERVAL    10      /* minimumintervall 20µs */
#define MIN_LENGTH      10
#define PULSE_DELAY     50      /* 20 µs */

void UpdateOutputs(float Y_alfa,float Y_beta) {
    register nb_mio_16_register_file    *board_registers = inboard->register_file;
    register float Ya,Yb,Yc,Ymax,Ymin,Yz;
    float Ytot_inv;
    int length_a,length_b,length_c;
    int delay_a,delay_b,delay_c;
    static int pulse_period[] = {2500,1250,832,625};
    int pidx = rand() & 0x03;
    /*
     * 2 till 3-fas omvandling, samt skalning med
     * omvandlingsfaktorer
     */
    Ya = Y_alfa * Sqrt_two_thirds;
    Yb = (Sqrt_three*Y_beta - Y_alfa)*Sqrt_two_thirds/2.0;
    Yc = (0.0 - Sqrt_three*Y_beta - Y_alfa)*Sqrt_two_thirds/2.0;

    /*
     * Maximera utnyttjandet av strömriktaren - gör
     * "rumpsinus" av ledvärdena.
     */
    Ymax = (Ya > Yb) ? Ya : Yb;
    Ymax = (Ymax < Yc) ? Yc : Ymax;
```

```
    Ymin = (Ya < Yb) ? Ya : Yb;
    Ymin = (Ymin > Yc) ? Yc : Ymin;
    Yz = (Ymax + Ymin) / 2.0 ;

    Ytot_inv = 2.0/Ud/Tsamp;
    length_a = pulse_period[pidx]*(0.5+0.5*(Ya-Yz)*Ytot_inv);
    length_a = (length_a < MIN_LENGTH) ? MIN_LENGTH :length_a;
    length_a = (length_a > pulse_period[pidx] - MIN_INTERVAL) ? pulse_period[pidx] - MIN_INTERVAL : length_a;
    delay_a = (pulse_period[pidx]-length_a);
    length_b = pulse_period[pidx]*(0.5+0.5*(Yb-Yz)*Ytot_inv);
    length_b = (length_b < MIN_LENGTH) ? MIN_LENGTH :length_b;
    length_b = (length_b > pulse_period[pidx] - MIN_INTERVAL) ? pulse_period[pidx] - MIN_INTERVAL : length_b;
    delay_b = (pulse_period[pidx]-length_b);
    length_c = pulse_period[pidx]*(0.5+0.5*(Yc-Yz)*Ytot_inv);
    length_c = (length_c < MIN_LENGTH) ? MIN_LENGTH :length_c;
    length_c = (length_c > pulse_period[pidx] - MIN_INTERVAL) ? pulse_period[pidx] - MIN_INTERVAL : length_c;
    delay_c = (pulse_period[pidx]-length_c);

    event_wait(controlTimer,0);

#ifdef __USE_NUBUS_ATOMICS
    activate_NuBus_locking(); /* Lock NuBus when accessing board registers */
#endif

    board_registers->am9513_command = 0xFFD3 << 16; /* Disarm counters*/
    board_registers->am9513_command = (0xFF01) << 16;       /* Set data pointer */
    board_registers->am9513_data = (FUN_CODE | SOURCE_CODE) << 16;      /* Mode G */
    board_registers->am9513_data = ((delay_a >> 1) + PULSE_DELAY) << 16;    /* Load */
    board_registers->am9513_data = length_a << 16;                      /* Hold */
    board_registers->am9513_data = (FUN_CODE | SOURCE_CODE) << 16;      /* Mode G */
    board_registers->am9513_data = ((delay_b >> 1) + PULSE_DELAY) << 16;    /* Load */
    board_registers->am9513_data = length_b << 16;                      /* Hold */
    board_registers->am9513_command = (0xFF05) << 16;       /* Set data pointer */
    board_registers->am9513_data = (FUN_CODE | SOURCE_CODE) << 16;      /* Mode G */
    board_registers->am9513_data = ((delay_c >> 1) + PULSE_DELAY) << 16;    /* Load */
    board_registers->am9513_data = length_c << 16;                      /* Hold */
    board_registers->am9513_command = 0xFF73 << 16; /* Load & arm counters*/
    board_registers->am9513_command = 0xFFE1 << 16;     /* Reset output 1*/
    board_registers->am9513_command = 0xFFE2 << 16;     /* Reset output 2*/
    board_registers->am9513_command = 0xFFE5 << 16;     /* Reset output 5*/
    board_registers->am9513_command = 0xFF09 << 16;     /* Set data pointer */
    board_registers->am9513_data = delay_a << 16;   /* Load */
    board_registers->am9513_command = 0xFF0A << 16;     /* Set data pointer */
    board_registers->am9513_data = delay_b << 16;   /* Load */
    board_registers->am9513_command = 0xFF0D << 16;     /* Set data pointer */
    board_registers->am9513_data = delay_c << 16;   /* Load */

#ifdef __USE_NUBUS_ATOMICS
    deactivate_NuBus_locking(); /* No need for bus locking */
#endif
}

static float Pr1[]={10,0,0};static float Pr2[]={0,10,0};
static float Pr3[]={0,0,10};static float *(P[])={Pr1,Pr2,Pr3};
static float theta[]={0.98,0.0775,0.03};

void ResetRLS(void) {
    int i,j;
    for(i=0;i<3;i++) for(j=0;j<3;j++) P[i][j]= i==j;
    theta[0]=0.98;
    theta[1]=0.0775;
    theta[2]=0.03;
}

void RLS(float **P,float *theta,int n,float y,const float *phi,float lambda,
        float lambda_inv,float *w) {
    int i,j;
    float err,den,dummy;

    /* Page 90, Adaptive Control */

    /* Compute residual */
    /* e= y-fi *theta */
    err=y;
    for(i=0;i<n;i++) err-=phi[i]*theta[i];

    /* Update estimate */
    /* w=P*fi */
    for(i=0;i<n;i++) {
        float *Prow=P[i];
        dummy=0;
        for(j=0;j<n;j++) dummy+=Prow[j]*phi[j];
        w[i]=dummy;
    }

    /* den=w*fi+lambda */
    den=lambda;
    for(i=0;i<n;i++) den+=w[i]*phi[i];

    /* theta=theta+w*e/den (note: error in book) */
    den=1/den;
    err*=den;

    for(i=0;i<n;i++) theta[i]+=w[i]*err;

    /* Update covariance matrix */
    /* P=(P-w*w/den)/lambda */
    for(i=0;i<n;i++) {
        float *Prow=P[i];
        for(j=0;j<n;j++) {
            Prow[j]=(Prow[j]-(w[j]*w[i])*den)*lambda_inv;
            /* Note that w*den cannot be precalculated because roundoff errors */
        }
    }
}

void Controller_thread() {

    /*
     * Förberedelser
     */

    float   PsiS_int = 0.0; /* PsiS_int Måste vara noll vid start */
    float   PsiS_d_inv;

    float   Cos_theta,Sin_theta;
    float   Sin_2x,Cos_2x;

    float   IS_qint = 0.0;  /* Integraldel till strömregulatorn */

    float   Yd_sp = 0.0,Yq_sp = 0.0,Y_max = 0.0,Yd_max = 0.0;
    float   Y_alfa1,Y_beta1,Y_alfa2,Y_beta2;
    float   Yq_filt = 0.0;

    float   OmegaR_old = 0.0;
    int     OmegaR_cut_count = 0;

    float   Sin_omegaR,Cos_omegaR,Sin_2omegaR,Cos_2omegaR;


    float w[3];
    float phi[3];
    int order=2;


    long    channels1[] = {1,1,0,2,3,4,5};
    struct {
        float   fubar,
                PsiS_alfa0,
```

```
            PsiS_beta0,
            Ia,
            Ib;
    float   OmegaR,Ud;
} samples1;

float   US_alfa_hat,US_beta_hat,UR_alfa_hat,UR_beta_hat;
float   IS_alfa_ff=0.0;
float   US_alfa_ff=0.0;
unsigned time=0;
static float sin_tab[256],cos_tab[256];

boolean_t buffer_plots = FALSE;
int     nr_of_points = 0;

lambda_inv=1/lambda;

/* Precompute sin-table, for pwmtest. */
{
    float fakt=2*PI/256;
    for(time=0;time<256;time++) {
        sin_tab[time]=0.06*sin(time*fakt);
    }
}
for(time=0;time<256;time++) cos_tab[time]=sin_tab[(time+64)&255];

time=0;
phi[0]=0;
phi[1]=0;

for(;;) {
    /*
     * Läs in Analoga värden
     */
    mio_16_analog_in(inboard,channels1,5,(float *) &samples1);

    /*
     * Normalisera till samma sampeltidpunkt, samt räkna om
     * till reella enheter (Vs,A,rad/s)
     * Strömmen har även en Tre- till Tvåfasomvandling.
     */
    PsiS_alfa = Kpsi * samples1.PsiS_alfa0;
    PsiS_beta = Kpsi * samples1.PsiS_beta0;
    IS_alfa = samples1.Ia * Ki * Sqrt_three_half;
    IS_beta = Sqrt_one_half * Ki * (samples1.Ia + 2*samples1.Ib);

    mio_16_analog_out(inboard,0,IS_alfa/Ki);
    mio_16_analog_out(inboard,1,IS_beta/Ki);


    /* Beräkna d-q koordinatsystemet */
    PsiS_d = sqrt(PsiS_alfa*PsiS_alfa + PsiS_beta*PsiS_beta);
    PsiS_d_inv = 1/PsiS_d;
    PsiS_q = 0;
    Cos_theta = PsiS_d_inv * PsiS_alfa;
    Sin_theta = PsiS_d_inv * PsiS_beta;

    /* Transformera strömmen till d-q koordinater */
    IS_d = IS_alfa * Cos_theta + IS_beta * Sin_theta;
    IS_q = 1.5*(IS_beta * Cos_theta - IS_alfa * Sin_theta) - 0.5*IS_q;

    /*
     * Flödes- och strömregulator:
     */

    /* Beräkna önskade ledvärden */
    Yd_sp = Kp_psi*((PsiS_sp - PsiS_d) + PsiS_int);
    Yq_sp = Kp_I*( (( (IS_q_sp > (IS_q_max)) ? (IS_q_max) : ( (IS_q_sp < -(IS_q_max)) ? -(IS_q_max):IS_q_sp) ) - IS_q) + IS_qint);

    /* Beräkna maximalt tillgänglig spännings-tidyta |Y|max: */
    Ud = 311.0; /* Ku * samples2.Ud; */
    Y_max = Tsamp * Ud * Sqrt_two_thirds;

    /* Begränsa ledvärdena, först i tvärled. */
    Yq = (Yq_sp > Y_max) ? Y_max : Yq_sp;
    Yq = (Yq < -Y_max) ? -Y_max : Yq;
    Yd = (Yd_sp > Y_max) ? Y_max : Yd_sp;
    Yd = (Yd < -Y_max) ? -Y_max : Yd;

    /*
     * Uppdatera de bägge regulatorernas integraldelar,
     * notera att vi nu kan använda skillnaden mellan önskat
     * och verkligt ledvärde till att begränsa integraldelens storlek
     */
    PsiS_int += Ki_psi*(PsiS_sp - PsiS_d) + Kr_psi*(Yd - Yd_sp);
    IS_qint  += Ki_I*(IS_q_sp - IS_q) + Kr_I*(Yq -Yq_sp);

    /*
     * Transformera till alfa-beta koordinater
     */
    Y_alfa = Yd * Cos_theta - Yq * Sin_theta;
    Y_beta = Yd * Sin_theta + Yq * Cos_theta;


    /*
     * Skatta hur mycket flödesvektorerna kommer att vrida sig
     * till nästa gång vi samplar. egentligen: beräkna sinus & cosinus för
     * fjärdedelen av denna vinkel samt även för halva samma vinkel.
     */

    Yq_filt += 0.5*(Yq-Yq_filt);
    Sin_x = (PsiS_d > 0.3) ? 0.125*Yq_filt/PsiS_sp : 0.0;
    Cos_x = sqrt(1.0-Sin_x*Sin_x);

    /*
     * Vrid fram styrvektorn med denna fjärdedels vinkel (för att hamna
     * mitt i intervallen)
     */
    Y_alfa1 = Y_alfa * Cos_x - Y_beta * Sin_x;
    Y_beta1 = Y_alfa * Sin_x + Y_beta * Cos_x;

    /*
     * Initiera uppdatering och invänta uppdateringstidpunkten
     * (Fram hit får det högst ta Tsamp/2, annars missar vi
     * uppdateringstillfället).
     */
    if (algorithm&2)
        UpdateOutputs(Y_alfa1,Y_beta1);
    else {
        time=(time+1) & 255; /* = mod 256 */
        UpdateOutputs(cos_tab[time],sin_tab[time]);
        Y_alfa1=cos_tab[time];
    }


    IS_alfa_ff=efactor*IS_alfa_ff+(1-efactor)*IS_alfa;


    /* Including the effect of the previous half-sample */
    phi[2]=US_alfa_ff;

    RLS(P,theta,order,IS_alfa_ff,phi,lambda,lambda_inv,w);

    /* We are using a regressor with the old IS_alfa_ff. */
    phi[0]=IS_alfa_ff;

    /* Compute phi[1] for next sample */
    US_alfa_ff=efactor2*US_alfa_ff+(1-efactor2)*Y_alfa1*Tsamp_inv;
    phi[1]=US_alfa_ff;
```

```
        Gamma1=theta[1];
        Gamma2=theta[2];
        P11=P[0][0];
        P21=P[1][0];
        P12=P[0][1];
        P22=P[1][1];

        if (theta[0]>0)
            My_Taus=-Tsamp/log(theta[0]);

        PsiS_d_qsc = PsiS_d;
        PsiS_q_qsc = 0.0;
        /*
         * Beräkna styrvärden till nästa intervall
         */

        Sin_2x = 2.0*Sin_x*Cos_x;
        Cos_2x = 1.0-2.0*Sin_x*Sin_x;

        Y_alfa2 = Y_alfa1 * Cos_2x - Y_beta1 * Sin_2x;
        Y_beta2 = Y_alfa1 * Sin_2x + Y_beta1 * Cos_2x;


        /*
         * Initiera uppdatering och invänta nästa uppdateringstidpunkt
         */
        if (algorithm&2)
            UpdateOutputs(Y_alfa2,Y_beta2);
        else {
            time=(time+1)&255; /* mod 256 */
            UpdateOutputs(cos_tab[time],sin_tab[time]);
            Y_alfa2=cos_tab[time];
        }

        US_alfa_ff=efactor2*US_alfa_ff+(1-efactor2)*Y_alfa2*Tsamp_inv;

        /* Utnyttjar att =operationen är atomär, för newlambda och algorithm */
        if ((algorithm&1)&&(order==2)) {
            order=3;
            ResetRLS();
        } else if ((!(algorithm&1))&&(order==3)) {
            order=2;
            ResetRLS();
        }

        if (lambda!=newlambda) {
            lambda=newlambda;
            lambda_inv=1/lambda;
        }

    } /* Så tar vi allt från början igen */
}


void to_do_atexit() {
    mio_16_reset(inboard);
    ao_6_analog_out(outboard,AO_6_chan0|AO_6_chan1|AO_6_chan2|AO_6_chan3|AO_6_chan4|AO_6_chan5|AO_6_update,0.0);
}

void AMControllerInit() {
    thread_t    th;
    timevalue   firsttime,interval;

    inboard = mio_16_init(INBOARD_SLOT,27);
    outboard = ao_6_init(OUTBOARD_SLOT);
    atexit(to_do_atexit);

    Tsamp = (float) SAMPLE_INTERVAL/1.0e+6;

    Tsamp_inv = 1/Tsamp;
    TauS_inv = 1/TauS;
    efactor=exp(-Tsamp*TauS_inv);
    efactor2=exp(-0.5*Tsamp*TauS_inv);


    gettimeofday(&firsttime);

    interval.tv_sec = 0;
    interval.tv_usec = SAMPLE_INTERVAL/2;

    timeradd(&firsttime,&interval);
    controlTimer = create_timer_event(NULL,&firsttime,&interval);

    th = thread_create(Controller_thread,7);
    thread_resume(th);
}

void SetFluxControllerParams(float Kp,float Ti) {
    Kp_psi = Kp;
    Ki_psi = Tsamp/Ti;
    Kr_psi = Tsamp/Ti/2.0/Kp;
}

void SetCurrentControllerParams(float Kp,float Ti) {
    Kp_I = Kp;
    Ki_I = Tsamp/Ti;
    Kr_I = Tsamp/Ti/2.0/Kp;
}
void SetRLSParams(int algor,float l) {
    algorithm=algor;
    newlambda=l;
}
```

Institutionen för Reglerteknik
Lunds Tekniska Högskola

Återkopplingslinjärisering av inverterad pendel

Håkan Persson, E-88
Vinh Tiet, E-88
Maria Wittrup, E-89

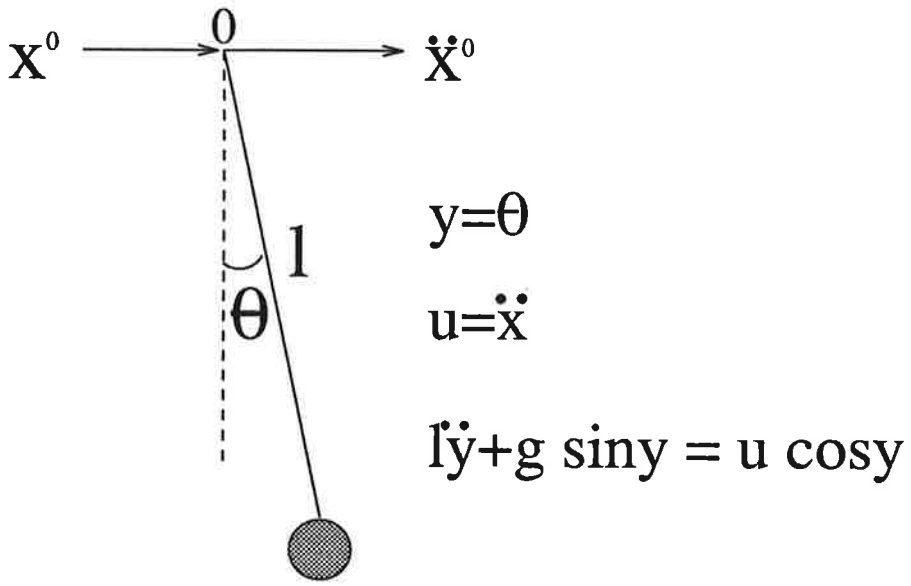
Handledare: Per-Olof Källén
maj 1993

Innehåll

1 Inledning

Detta projekt går ut på att med hjälp av en olinjär återkoppling linjärisera modellen för den inverterade pendeln. Pendelns lutning och vinkelhastighet är mätbara medan pendellängden anses okänd. I återkopplingen används en skattning av pendellängden. Man erhåller därmed en adaptiv regulator baserad på tillståndslinjärisering.

2 Systembeskrivning



Figur 1: Bild av pendeln som skall styras.

Pendelvinkeln θ skall regleras. Som insignal används accelerationen i upphängningspunkten. Systemet beskrivs då av följande ekvationer.

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -\frac{g}{l} sin x_1 + \frac{u}{l} cos x_1 \\ y = x_1 \end{cases}$$

Detta olinjära system görs linjärt med en olinjär regulator s.k. återkopplingslinjärisering. Se exempel 9.5 sidan 353, [1].

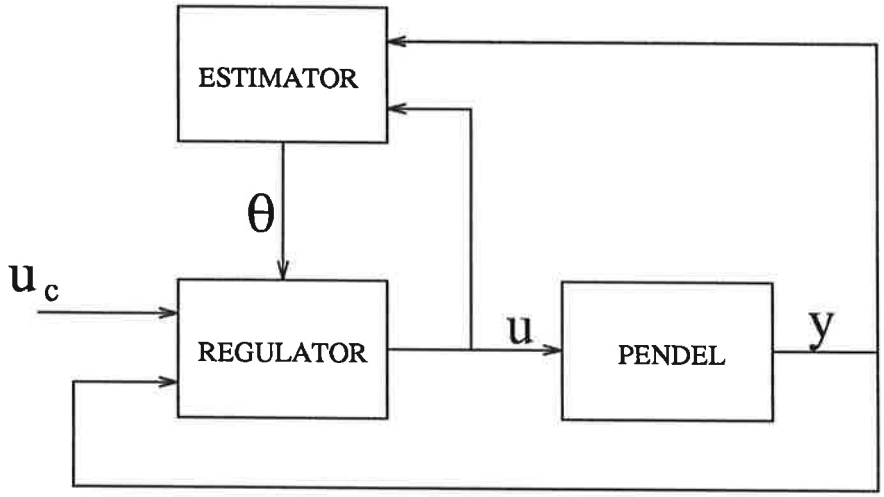
Ett andra ordningens olinjärt system

$$\frac{dx_1}{dt} = f_1(x_1, x_2)$$

$$\frac{dx_2}{dt} = f_2(x_1, x_2, u)$$

kan med hjälp av återkoppling transformeras till ett system med överföringsfunktionen

$$\frac{\omega^2}{s^2 + 2\zeta\omega s + \omega^2}$$



Figur 2: Blockschema på modell.

Inför tillstånden

$$\begin{aligned} z_1 &= x_1 \\ z_2 &= \frac{dx_1}{dt} = f_1(x_1, x_2) \end{aligned}$$

och en fiktiv styrsignal

$$v = F(x_1, x_2, u) = \frac{\delta f_1}{\delta x_1} f_1 + \frac{\delta f_1}{\delta x_2} f_2$$

Det resulterande linjära systemet blir

$$\frac{dz_1}{dt} = z_2$$

$$\frac{dz_2}{dt} = v$$

Det önskade slutna systemet erhålles genom att välja

$$v = F(x_1, x_2, u) = \omega^2(u_c - x_1) - 2\zeta\omega f_1(x_1, x_2)$$

För det aktuella fallet med pendeln fås då

$$u = \frac{l}{cosx_1}[\omega^2(u_c - x_1) - 2\zeta\omega x_2] + \frac{g}{cosx_1} sinx_1$$

Pendellängden l skall skattas och u_c är referensvinkeln.

Tre olika skattningsmodeller studeras

1. $\underbrace{y}_{y} = \underbrace{\frac{1}{l}}_{\theta} \underbrace{\frac{1}{p^2}(u\,cosy - g\,siny)}_{\varphi}$

2. $\underbrace{\frac{p^2 y}{A_f(p)}}_{y} = \underbrace{\frac{1}{l}}_{\theta} \underbrace{\frac{1}{A_f(p)}(u\,cosy - g\,siny)}_{\varphi}$

3. $\underbrace{\frac{1}{A_f(p)}(u\,cosy - g\,siny)}_{y} = \underbrace{l}_{\theta} \underbrace{(\frac{p^2}{A_f(p)}(y))}_{\varphi}$

Inför ett filter

$$A_f(p) = (1 + \tfrac{p}{a})^2$$

Detta ger ett andra ordningens lågpass filter med brytning vid a rad/s.
Estimatorerna är av kontinuerlig minsta-kvadrat typ och beskrivs av

$$\frac{d\hat{\theta}}{dt} = P(t)\,\varphi(t)\,e(t)$$

$$e(t) = y(t) - \varphi^T(t)\hat{\theta}(t)$$

$$\frac{dP}{dt} = \alpha P(t) - P(t)\varphi(t)\varphi^T(t)P(t)$$

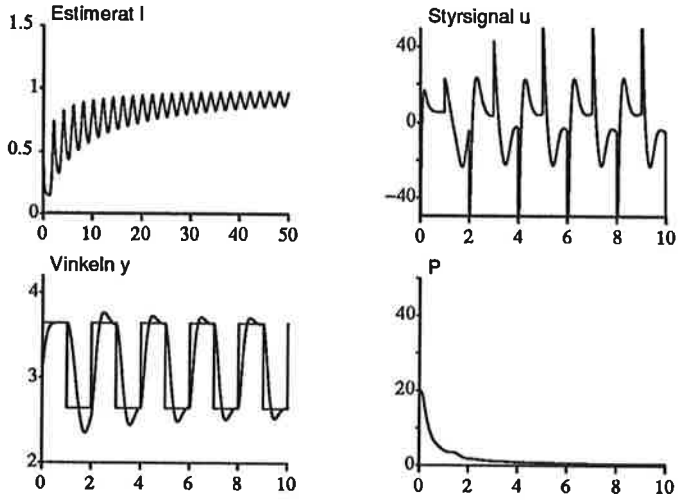
3 Simuleringar

Simuleringarna är gjorda i simnon. Macrona finns beskrivna i appendix. De tre estimatorerna ligger i tre olika filer. När simuleringen ska köras kopieras rätt estimatorfil till filen `estim.t`.

I vårt önskade slutna system är $\omega = 10$ och $\zeta = 0.7$.

3.1 Estimator 1

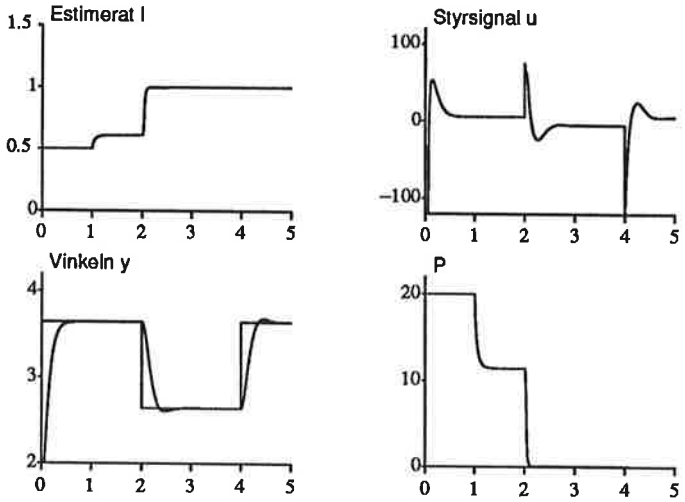
I Figur 3 ser vi att skattningen av l konvergerar långsamt mot det verkliga värdet, 1. Avsaknad av LP-filter i estimatorn ger högfrekventa svängningar i skattningen. Det är viktigt att poängtera att initieringsläget för pendeln är π, dvs lodrätt. Om detta ej är fallet kommer skattningen inte att bli rätt. Denna estimator fungerar alltså inte tillfredställande.



Figur 3: Simulering med estimator 1. $\alpha = 0.0001, l = 1$.

3.2 Estimator 2

I följande skattare används ett filter. För att ej få estimatorn att arbeta med felaktiga värden, som bildas i insvängningsförloppet hos filtret, startas inte estimatorn förrän efter 1 s.



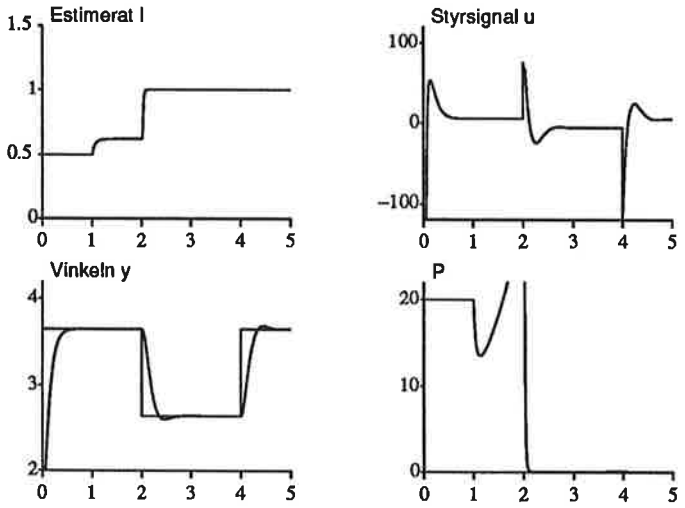
Figur 4: Simulering med estimator 2. $\alpha = 0.0001, a = 10, l = 1$.

När α ökas till 1 exploderar korrelationsmatrisen P, se Figur 5, vilket man också kan se i ekvation 1.

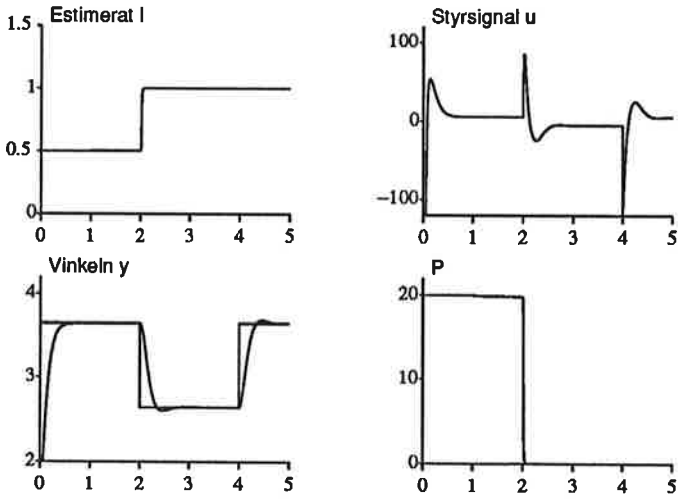
$$\frac{dP}{dt} = \alpha P(t) - P(t)\varphi(t)\varphi^T(t)P(t) \qquad (1)$$

Härur ser vi att P är avtagande om $\alpha < P \cdot \varphi^2$. När estimatorn startas är styrsignalen 0 och därför växer P som dock konvergerar mot noll då systemet exiteras. Av samma orsak konvergerar inte l mot ett under första halvperioden.

Om estimatorfiltrets brytfrekvens sätts till 28 rad/s blir estimatorn mycket snabbare, se Figur 6.



Figur 5: Simulering med estimator 2. $\alpha = 1, a = 10, l = 1$.



Figur 6: Simulering med estimator 2. $\alpha = 0.0001, a = 28, l = 1$.

3.3 Estimator 3

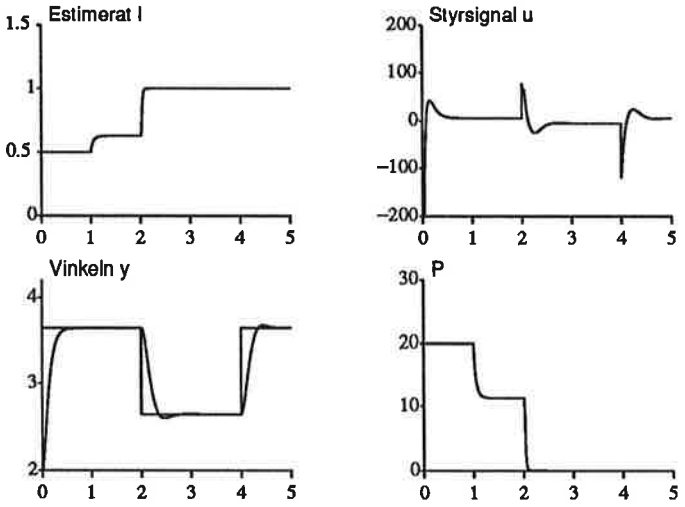
Samma parametrar som i estimator 2 används här.

Även här ser vi att skattningen inte konvergerar mot ett under första halvperioden på grund av att styrsignalen är noll. Figur 8 visar att om α väljs tillräckligt stor kommer P att explodera och estimatet att påverkas precis som för estimator 2, se ekvation 1.

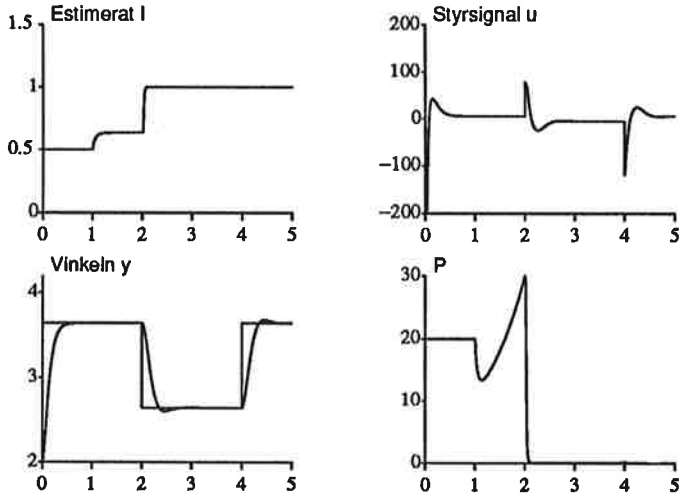
Genom att öka a, filtrets bandbredd, får man en snabbare insvängning hos estimatet, jämför Figur 7 och 9.

4 Slutsatser

Den adaptiva regulatorn beter sig väl då lämplig modellstruktur användes. Skattningen av l konvergerar efter 1-2 steg varför slutna systemet snabbt närmar sig det önskade beteendet.



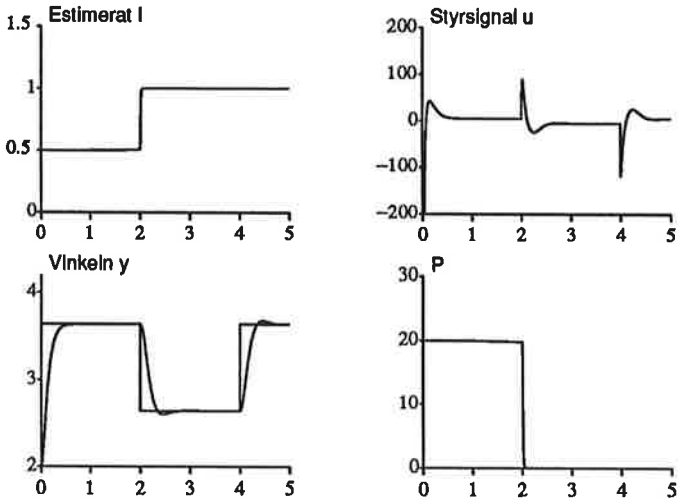
Figur 7: Simulering med estimator 3. $\alpha = 0.0001, a = 10, l = 1$.



Figur 8: Simulering med estimator 3. $\alpha = 1, a = 10, l = 1$.

Ju större bandbredd filtret har desto snabbare konvergerar estimatet, men blir mer känsligt för högfrekventa störningar.

Estimator 2 och 3 fungerar likvärdigt men estimator 3 skattar l direkt. Det kan därför vara bäst att välja estimator 3.



Figur 9: Simulering med estimator 3. $\alpha = 0.0001, a = 28, l = 1$.

Referenser

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

A Simnonfiler

```
MACRO proj2 "Huvudprogram för användning av estimator 2
ALGOR rkf23 "Ändring av integrationsmetod
SYST estim reg syst conn
PAR a[estim]:10
, alfa[estim]:0.0001
, per[conn]:2
, w[reg]:10
,l[syst]:1
INIT P:20
, theta:0.1
, x1[syst]:1.6
STORE l[estim] y[syst] u[reg] uc[reg] fi[estim] P[estim]
SIMU 0 5
SPLIT 2 2
AXES V 0 1.5 H 0 5
SHOW l[estim]
TEXT 'Estimerat l'
AXES V 2 4.2 H 0 5
SHOW y[syst] uc[reg]
TEXT'Vinkeln y'
AXES V -120 120 H 0 5
SHOW u[reg]
TEXT'Styrsignal u'
AXES V 0 22 H 0 5
SHOW P[estim]
TEXT'P'
end

connecting system conn
time t
temp=IF MOD(t,per)<per/2 THEN step ELSE -step
uc[reg]=temp+3.141592
y[reg]=y[syst]
yprim[reg]=yprim[syst]
l[reg]=if t>1 then l[estim] else 2
u[syst]=u[reg]
u[estim]=u[reg]
y[estim]=y[syst]
per:50
step:0.5
end

continuous system reg
input uc y yprim l
output u
time t
u=l/cos(y)*(w*w*(uc-y)-2*z*w*yprim)+g/cos(y)*sin(y)
g:9.81
z:0.7
w:0.1
end
```

```
continuous system syst
input u
output y yprim
time t
state x1 x2
der dx1 dx2
y=x1
yprim=x2
dx1=x2
dx2=-g/l*sin(x1)+u/l*cos(x1)
g:9.81
l:1
end

continuous system estim
"Estimator 1
input u y
output l
time t
state x1 x2 P theta
der dx1 dx2 dP dtheta
ytemp=u*cos(y)-g*sin(y)
dx1=ytemp
dx2=x1
fi=x2
" Estimator från sidan 71
dtheta=P*fi*e
e=y-fi*theta
dP=if t<1 then 0 else alfa*P-P*fi*fi*P
l=1/theta
alfa:0.01
g: 9.81
end
```

```
continuous system estim
"Estimator 2
input u y
output l
time t
state x1 x2 fi1 fi2 P theta
der dx1 dx2 dfi1 dfi2 dP dtheta
"Filtrering av y
ystreck=a*a*(-2*a*x1-a*a*x2+y)
dx1=-2*a*x1-a*a*x2+y
dx2=x1
"Filtrering av ucos(y)-gsin(y)
ytemp=u*cos(y)-g*sin(y)
fi=a*a*fi2
dfi1=-2*a*fi1-a*a*fi2+ytemp
dfi2=fi1
" Estimator från sidan 71
dtheta=P*fi*e
e=ystreck-fi*theta
dP=if t<1 then 0 else alfa*P-P*fi*fi*P
l=1/theta
alfa:0.01
a:1
g: 9.81
end

continuous system estim
"Estimator 3
input u y
output l
time t
state x1 x2 fi1 fi2 P theta
der dx1 dx2 dfi1 dfi2 dP dtheta
"Filtrering av y
fi=a*a*(-2*a*fi1-a*a*fi2+y)
dfi1=-2*a*fi1-a*a*fi2+y
dfi2=fi1
"Filtrering av ucos(y)-gsin(y)
ytemp=u*cos(y)-g*sin(y)
ystreck=a*a*x2
dx1=-2*a*x1-a*a*x2+ytemp
dx2=x1
" Estimator från sidan 71
dtheta=if t<1 then 0 else P*fi*e
e=ystreck-fi*theta
dP=if t<1 then 0 else alfa*P-P*fi*fi*P
l=theta
alfa:0.01
a:0.1
g: 9.81
end
```

Adaptation and Learning

A Comparison of AI and Control Views

Jan Eric Larsson

An Essay for the Course in Adaptive Control
Department of Automatic Control, Lund Institute of Technology
April 1993

1. *Introduction*

Once upon a time the research area of *Cybernetics* was born. In the beginning it comprised several subjects, among other both artificial intelligence and control theory. But after some years, these two fields became more and more estranged, and nowadays they form two completely separate schools of research.

There are, however, points of strong common interest. One of these is definitely *learning,* maybe the most profound area of AI and at the same time one of the least successful. Some versions of learning are quite similar to *adaptive control,* a well established area of control theory.

This report will describe some common points of AI learning and adaptive control in the area of parameter adjustment, and it will interpret the algorithms developed in AI in the terms of control theory. It was written for the course "Adaptive Control," given by Björn Wittenmark at the Department of Automatic Control, Lund Institute of Technology, Lund, in the spring of 1993.

First, AI learning and adaptive control will be described. After that, some different learning projects will be interpreted from an adaptive control viewpoint.

2. *What is Learning?*

One of the most intriguing research areas of artificial intelligence is *learning.* It is a common opinion that computers cannot be called intelligent until they are able to learn to do new things and to adapt to new situations. This is certainly wrong, as many intelligent behaviors do not include any learning aspect. In spite of this, learning *is* an important part of AI, and maybe also one of the areas that have had the least success so far.

It is difficult to define learning, (more difficult than to define adaptation, as we shall see below). A general definition is that learning is change of behavior in a given situation brought about by the repeated experiences of such situations. But such a definition is much too general to be useful, and it would include several computer programs that are not normally referred to as learning programs.

As is often the case in AI, learning is best defined by examples of what is done in the research area. At the same time, learning is a vast field which is difficult to cover. Thus, the following overview only serves to give some examples of the different subjects:

- *Rote learning.* This is the simplest kind of machine learning, and it simply means to store large quantities of data and using these data to permit a learning behavior. Once a result has been computed, it is stored in a large table and used again instead of new calculations. A well known example of this is the first of Samuel's Checkers programs, see Samuel (1963). This program used a large database of

previously encountered positions to enhance its evaluations. If a previously stored position was found during the tree search, its value was used, instead of the evaluation function or deeper search. When a significant part of the leaf-nodes could be found in the table, it gave the same effect as that of a much deeper search.

- *Parameter adjustment.* A large variety of programs rely on the procedure of weighting several features together into a single summary value. In such cases it is sometimes possible to begin with an estimate of the best value and then slowly adjust the weighting parameters according to some loss function, so that they reach or come closer to the best value. An example of this is the second of Samuel's Checkers programs. This program played games against itself and adjusted the weighting parameters of the evaluation function according to the outcome of the played games. The results were, for that time, quite impressing, and Samuel's second program is sometimes viewed as the most successful example of AI learning. The program managed to win a game against a human Checkers master. Another good example of parameter adjustment is the training of neural networks by backpropagation or other algorithms. A neural network approach has also been used in a program to play Backgammon, see Tesauro and Sejnowski (1989).

- *The General Problem Solver.* The GPS program, see Newell and Simon (1963), has been used for learning. GPS is really a sort of inference engine, using a data structure where states are represented, together with operators to change these states. One state is the initial state and some states are goal states, and GPS applies the operators in turn, (based on a difference table), to reach from the initial to a goal state. One way of using learning in GPS is to let the program try and solve several similar problems while gathering statistical information in order to construct the difference tables by itself. Another way is to view the learning task as general problem solving, and describe the initial and goal states, and the operators of learning itself in GPS, and then let the program perform the learning task. In this way, learning is equated with problem solving. This solution demands full knowledge of the learning task, though, and provides a good example of the principle that in order to learn something, one must already know a lot.

- *Dynamic Programming.* The well established technique of dynamic programming can be used in learning tasks. In Bellman (1961) such a method is used to solve the *two-armed bandit problem,* where a gambler tires to optimize his winnings when playing two slot machines. The winning probability of one of the machines is not known. Thus, dynamic programming is used to find the optimum between gathering information that will later give a better strategy, and "playing it safe" by using the current knowledge and simply making the optimal choice at every instant. This is a typical example of the fact that

testing to gather more information, or in other words, to excite the process, may be suboptimal in a short perspective but valuable in the long run.

- *Concept learning.* In many AI programs, it is important to classify objects, in order to apply reasoning rules about them. It may, however, be difficult to devise the classification mechanism. For this, learning has been used. A good example is Winston's learning *blocks world* program. This program uses a set of primitive graphical concepts, such as lines and points, and tries to invent new structural concepts, characterized by the presence of certain subsets of basic concepts. Thus, such concepts as archs, bridges, and houses may be learned, see Winston (1975). Classification can also be done by linear weighting by a scoring polynomial, and in this case, concept learning by parameter adjustment may be possible.

  The learning of concepts from a set of examples has also been treated by Haussler (1988), and Haussler *et al* (1991). Here, Valiant's learnability model and learning framework, see Valiant (1984), is used and Haussler arrives at necessary and sufficient conditions for learning to be possible. This is determined by the finiteness of the *Vapnik-Chervónenkis dimension,* a simple combinatorial parameter of the class of concepts to be learned, see Vapnik (1982), and Vapnik and Chervónenkis (1971). This theory can also be used to measure the efficiency of some different learning methods. For example, Haussler has investigated the PAC model for neural net learning, see Opper and Haussler (1991) and Haussler (1992).

  Another concept learning project is described in Ioannidis *et al* (1992), where a system uses learning for database design.

- *Learning from Examples.* Some projects use learning from examples. Pitas *et al* (1992) use examples to learn rule for a rule-based system. Gasarch and Smith (1992) describes a system that tries to learn a mathematical function from examples and is allowed to pose extra questions in order to improve the learning efficiency.

- *Learning as Discovery.* Doug Lenat's program AM, see Lenat (1977), uses a frame-based knowledge representation together with a heuristic search among some 250 rules to discover mathematical proofs. For example, it could discover the concept of prime numbers, which is quite impressing. However, AM relied heavily on its heuristics, and what it basically did, was to perform generalization from the implicit knowledge hidden in its rules.

- *Learning by Analogy.* Analogy is a powerful tool for inferencing, but it also demands a proper interpretation. Much interest has gone into this field, see for example Winston (1980). This system uses a frame representation to draw analogies between objects.

- *Learning Belief Networks and Expert Systems.* Some projects use learning to build knowledge-based systems. Hsu and Yu (1992) de-

scribes a fault diagnosis system that learns by examples, in much the same way as the BOXES algorithms to be described later. Torasso (1991) describes learning in a diagnostic expert system. Mahmoud *et al* (1992) describes a learning rule-based system for control. Neal (1992) describes a system that learns a belief network, i.e., an influence diagram by neural network techniques. the system can then be used more or less like an expert system.

- *Learning Mathematics.* Learning algorithms have been used to learning mathematical concepts. Helmbold *et al* (1992) describes a system that learns integer lattices, while the system of Arikawa *et al* (1992) learns simple formal systems.
- *Learning in Control Theory.* Learning is also used in several control applications, and the following can only server as a collection of some interesting examples. A survey is found in Moore *et al* (1992). Messner *et al* (1991) presents a new adaptive learning rule for parallel calculation. The target processes are robots. Mahadevan and Connell (1992) describes a robot learning to push boxes with learning of the BOXES (!) type. Heinzinger *et al* (1992) gives some stability results for learning control algorithms. Kalaba and Udwadia (1991) use learning for structural identification in mechanical systems. Ho *et al* (1992) describes a learning neural network for short-term load forecasting in power systems. Li and Tzou (1992) describes a learning fuzzy controller.

There are several other areas in learning, for example different versions of learning or generalization from examples, case-based reasoning, and also more connectionistic ideas, such as primitive agents that causes a global behavior to adapt or learn.

From this overview, it is clear that AI learning comprises several quite different methods. The aim of this essay is to give an adaptive control view of learning techniques, thus it will be limited to treating learning through *parameter adjustment.* The other techniques will not be further treated here.

3. *What is Adaptive Control?*

There has been difficulties in defining what is to be meant with adaptive control. The everyday meaning of the word "adapt" is to change behavior in order to better conform to new circumstances. The similarity to learning should be noted. The idea of adaptive control is to take a standard controller, such as a PID or RST controller, and change its behavior to suit changing operating conditions, i.e., to adjust its parameters, and maybe, (in case of the RST controller), its structure, to preserve a good tuning when the controlled process is changing. Thus, adaptive control may be defined as dealing with controllers *viewed as* consisting of two levels, one standard control level, and one level of parameter (and maybe structure) adaptation.

But as is the case with AI and learning, adaptive control is best defined by examples. There are two main approaches to adaptive control: model-reference adaptive systems, (MRAS), and self-tuning regulators, (STR). Both these approaches build on the general idea of using the difference between observed and wanted behavior, and to adjust parameters according to this. First, the MRAS approach will be described, as it is the simpler and more direct of the two. The following part is based on Åström and Wittenmark (1989).

Model-Reference Adaptive Systems

The model-reference adaptive system is based on a specification of a reference model, which describes how the controlled system should ideally behave. A block diagram description is given in Figure 1.

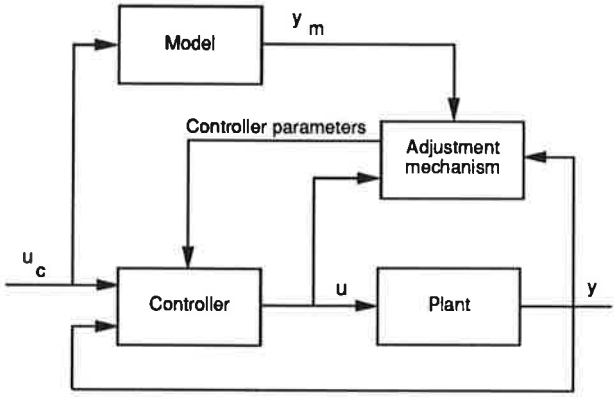


Figure 1. A block diagram of a model-reference adaptive system. From Åström and Wittenmark (1989).

The system consists of two control loops, one inner in which the controller controls the process, and one outer, where the adjustment mechanism adjusts the parameters of the controller so as to make the controller + process system behave similarly to the reference model.

The parameters of the controller are adjusted depending on the difference between the process and model outputs, i.e., the model error

$$e = y - y_m.$$

The sensitivity derivative of the model error for the parameter is also weighed in, and the value multiplied by a gain γ. The following adjustment mechanism, the MIT rule, was used in the first MRAS:

$$\frac{d\theta}{dt} = -\gamma e \frac{\partial e}{\partial \theta},$$

where e is the model error and γ the adjustment gain. In other words, the parameters of the controller are adjusted proportionally to the model error and the sensitivity derivative of the parameter in question.

In summary, a model-reference adaptive system has the following general properties:

- The processes handled are physical processes, which are continuous and usually can be described by linear or reasonably nice nonlinear equations. The equations are for example usually continuous, and the nonlinearities are often limited so that the parameters of the linear approximations do not vary within more than some orders of magnitude.
- The model error is derived as the difference between the process behavior and a reference model.
- The parameters of the controller are adjusted according to a gradient method, so that the model error is driven to zero.
- Under some reasonable assumptions, such as that the controlled process is linear, that the signals are persistently exciting, and that modeling errors and noise are small, it is possible to prove stability and convergence, or at least to hope that convergence will occur with reasonable speed in most cases.

Self-Tuning Regulators

The self-tuning regulator, (STR), is based on estimation of a parameterized model of the process. This model is then used to calculate controller parameters to achieve a specified behavior. A block diagram describing the general idea is found in Figure 2.

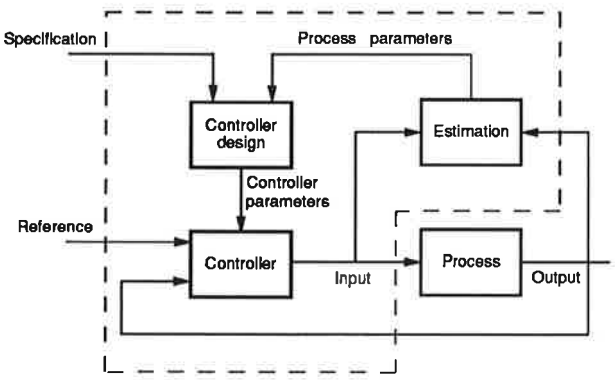


Figure 2. A block diagram of a self-tuning regulator. From Åström and Wittenmark (1989).

The *estimation* block is usually performed with a *recursive least squares* algorithm, which basically implies that the parameters of the model are updated according to the following equations:

$$\hat{\theta}(t) = \hat{\theta}(t-1) + K(t)(y(t) - \varphi^T(t)\hat{\theta}(t-1))$$
$$K(t) = P(t-1)\varphi(t)(\lambda I + \varphi^T(t)P(t-1)\varphi(t))^{-1}$$
$$P(t) = (I - K(t)\varphi^T(t))P(t-1)/\lambda$$

This equation has a strong intuitive appeal. The estimate $\hat{\theta}$ is obtained by adding a correction term to the previous estimate $\hat{\theta}(t-1)$. This correction is a function of the difference between the process output and the output predicted by the model; it is a *prediction error* method.

Here, the similarity with the MRAS approach should be noted. Both methods adjust parameters to minimize the difference between an actual value and one supplied by a model, but one uses a gradient method, while the other uses a least squares algorithm. In an MRAS, controller parameters are adjusted, as is the case in a *direct* STR, while in an *indirect* STR, the adjusted parameters describe a model, which is used to compute controller parameters. The latter is done in the *controller design* block of Figure 2.

It is now possible to summarize the properties of a self-tuning regulator:

- The processes handled are, (once again), physical processes, which are continuous and usually can be described by linear or reasonably nice nonlinear equations. The equations are for example usually continuous, and the nonlinearities are often limited so that the parameters of the linear approximations do not vary within more than some orders of magnitude.
- The model error is derived as the difference between the process behavior and an estimated model.
- The parameters of the model are adjusted so that the prediction error is driven to zero.
- Under some reasonable assumptions, such as that the controlled process is linear, that the signals are persistently exciting, and that modeling errors and noise are small, it is possible to prove stability and convergence, or at least to hope that convergence will occur in most cases.

With these simple characterizations of adaptive controllers in mind, the time has come to investigate the different methods of AI learning, and compare them to the adaptive control ideas.

4. *Checkers*

The first, and also most classical example of learning by *parameter adjustment* is the second Checkers program of Samuel, see Samuel (1963). In order to interpret Samuel's ideas in control terms, let us first describe the learning algorithm.

A Description of the Program

Samuel's programs, as most other board game programs relied on a tree search. This means that from each position where a move must be made, the program investigates the possible moves and resulting new positions several moves ahead. At the end points, called leaf nodes, an evaluation

function is applied, giving a quantitative value describing how good the position is for the program.

The values of the leaf nodes are then backed up in a minimax fashion, so that whenever the program is to move, it chooses the maximal value among the successor nodes, while if the opponent is to move, the minimal value is chosen. From the root node, which corresponds to the position from which a move actually must be made, that move is chosen which leads to the position with the highest backed-up score. This minimax search has the advantage that it guarantees the program to find the best possible move within the search horizon and given that the opponent plays perfectly. If it is possible to search to the end of the game, the program will play a theoretically perfect game. The algorithm can be enhanced in several ways, for example with the $\alpha\beta$-algorithm, which means that out of a total of N nodes in the tree, only $2\times\sqrt{N}$ need be investigated. For non-trivial games, such as Chess, Checkers, and Othello, it is seldom possible to search to the end. In this case a heuristic *evaluation function*, $\hat{f}$, is used instead of the true value, f, of the leaf node positions. The better this evaluation function can approximate the correct value, the better the program will play.

The evaluation function of Samuel's second Checkers program consisted of a linear polynomial, where parameters, k_i, were used to produce a weighted score of sixteen terms, f_i, each being a quantitative measure of some feature of the current position, p, on the board:

$$\hat{f}(p) = \sum_{i=1}^{16} k_i \times f_i(p).$$

The efficiency of the evaluation function, f, and thereby the strength of the program's play depended critically on getting a good weighting of the different terms, i.e., of finding an optimal choice of values for the parameters k_i. Typical examples of terms, f_i, were the mobility, (the number of available moves), the advancement of pieces, (the number of rows the pieces had moved forward), the centrality, (the distance of the pieces from the center of the board), etc.

The evaluation function terms were 38 all in all, out of which 16 were used at a time. If one term had been given the lowest weighting for 32 moves, it was taken out of the scoring polynomial and replaced by one of the terms from the waiting pool. This strategy was used to allow any number of terms while simplifying the adaptation problem by not using more than 16 of them at a time.

The material advantage, (computed as 200 points for every man and 300 points for every king, the opponents pieces counted negatively), was deemed to be the most important term, and to give some basic stability and direction to the learning algorithm, this term was computed separately and always kept unchanged. Thus, the material advantage was in fact made the loss function for the learning. In Checkers, the gain of a piece usually leads to the win of the game; thus the assumption seem quite reasonable.

The program was run in two variants, α and β. The α version adapted parameters in its evaluation function, while β worked as an invariant opponent. Whenever α had won sufficiently many games against β, the latter took over the new parameter settings of α. If α lost three consecutive games, it was deemed to be on the wrong track and "a fairly drastic and arbitrary change was made in its scoring polynomial, (by reducing the coefficient of the leading term to zero)." This idea is somewhat similar to the *simulated annealing* methods used in neural networks. Samuel explains the problematic effect by referring to local maxima, but it is not clear that this really is the explanation.

The basic idea of the adaptation was that after each tree search the value of the evaluation function in the root node, $\hat{f}_r$, was compared to the backed up value of the search, $\hat{f}$. The difference,

$$\delta = \hat{f}_r - \hat{f},$$

was used as a kind of error, and the parameters in the evaluation function was slightly adjusted so that the evaluation in the root node, $\hat{f}_r$, matched the backed up value, $\hat{f}$, better.

The rationale for this was that under reasonable assumptions, (which are true for the game of Checkers), a value backed up by an $\alpha\beta$-search is more reliable than a direct application of the evaluation function. The program actually was trying to reach an evaluation function that approximated the value found by a deep tree search.

The inputs to the adaptation procedure were the sign of δ and the signs of the evaluation function parameters, k_i. A correlation coefficient for each parameter was updated during play, and the parameters computed using these coefficients. Specifically, if the ratio of two correlation coefficients, c_i, was bigger than n but smaller than $n + 1$, i.e., if

$$n \leq c_i < n + 1,$$

then the corresponding weighting parameter was set to 2^n, in order to give a large span between the weighting parameters.

Some stabilization measures were also taken. If δ was below a certain limit, no updating was done, while if the material balance was affected, the change in correlation coefficients was doubled, and if δ indicated a win or loss, (the largest possible values), it was quadrupled.

Results

Samuel describes two test runs with the program. The first one consisted of 28 games played, and resulted in violent changes in both which terms that were used and discarded, and in the weighting parameters. At least 20 different terms were at the leading position during different parts of the experiment. Even at the end of the run Samuel conceded that "the learning procedure was still not completely stable." The parameter setting resulted in a program that was first "tricky but beatable" and later "better than average."

Prior to the second test, several stabilization measures had been taken. The program was often fooled by bad play on behalf of the opponent, so the adaptation was made slower when α was leading. The terms were replaced every 32nd move, instead of originally after every 8th. It was also decided to demand that α should have a majority of wins over β before the replacement of scoring polynomials took place.

The results of the second test was a more stable but slower learning. Here, a seemingly semi-stable state was reached after some 30 games played. Still, after each parameter transfer, several oscillations occurred before α landed on a parameter setting that enabled it to win over β.

Samuel observed that the rote learning program efficiently learned to play opening and endgames, but never became good in the middlegame, while the parameter adaptation program quickly learned to play a good middlegame. However, it never learned to play in a conventional manner, and its openings were weak. Also, after 28 games, it still had not learned how to win with two kings against one in a double corner, a reasonably trivial task.

Interpretation

The updating mechanism used in Samuel's second Checkers program is a bit similar to an MRAS system; maybe most similar to the *sign-sign algorithm,* a simple version often used in telecommunications:

$$\frac{d\theta}{dt} = -\gamma \operatorname{sign}\left(\frac{\partial e}{\partial \theta}\right) \operatorname{sign}(e).$$

There are some important differences, however, between such an MRAS and Samuel's algorithm:

- Samuel's "process" is a minimax $\alpha\beta$ tree search. It consists of several maximum and minimum computations, together with logical selections, and as opposed to most physical processes, it is highly nonlinear and discontinuous. Thus, no stability proof is even within sight, and it is indeed not clear why an adaptation procedure should be able to converge, nor that Samuel's tests really do so.
- The "model error," δ, is discretized into four levels, while the correlation coefficients are computed by a stable and "calm" correlation algorithm, (Samuel does not give any precise details).
- Most difficult is probably the problem of *credit assignment,* i.e., the adaptation may be fooled by faults by the opponent. This corresponds somewhat to the problem of *persistent excitation.* The problem is how to find out exactly when the important choices were made, and to separate these moments for others, where faults by the opponent and consequences of earlier play are the reasons for a changing δ. Samuel's solution tries to avoid the worst effects, but all in all, this is still an unsolved problem. An obvious improvement would be to use the adaptation on lost games only.

Evaluation

Samuel's second Checker program is probably the most successful AI learning example, and for its time, (1963), it was certainly impressing. From today's horizon, however, the glory has faded somewhat:

- It is not clear that the adaptation procedure is stable or converges to a good value. The degree of calmness reached would not be deemed satisfactory in any adaptive control system.
- The level of play attained was, in spite of all rumors, not very high, neither compared to the human skill of those days, nor to the level of today's computer programs. Samuel's program won one game over one state master in the US, but that was a master of blind players, and there was still a long way to go in order to reach world class level. Today, the Checkers program Chinook, see Schaeffer *et al* (1992), plays on par with Marion Tinsley, the Checkers world champion, and is probably the world's second or third best player. This program relies on an efficient tree search, multiprocessing, and large databases of endgame positions. It uses no adaptation or learning whatsoever, but it can search up to 20 plies, (half moves), and often evaluates the leaf positions with 100% accuracy, as they can be found in the endgame databases. The 20 plies depth should be compared to the 3 or 4 ply searches of Samuel's program. With some knowledge of state of the art game programming techniques it is trivial to write a program that searches some 10 to 15 plies deep and outperforms Samuel's program by far.

Some Final Comments

Samuel's results were impressive and maybe the best example of successful learning in its time, but they are now of little importance to game-playing programs. This shows that a large portion of humility is needed in the field of learning. It is very difficult to reach working results at all, and there are some few successful results of this. To hope for better performance than with other techniques is so far unrealistic.

5. *Chess*

The *Chiptest, Deep Thought,* and *Deep Blue* programs are all part of a long term project of VLSI construction for chess processors, see Hsu (1987) and Anantharaman *et al* (1991). These programs are really a combination of software and hardware, to perform very fast minimax $\alpha\beta$ tree searches. The current version of Deep Blue is the world's strongest chess machine, but it still has some way to go before it can compete with the best human chess players.

Deep Blue and some other game-playing programs use databases of human master games to adapt their evaluation function parameters. The idea is simple:

- From information about which side that won a particular game, and maybe also who played it, the positions of the game are scored as good, bad, or even.
- The evaluation function polynomial is then tested in the different positions and the parameters adjusted to better agree with the previously computed scores. This adjustment is usually performed with a least squares or maximum likelihood criterion.

Interpretation

These methods are clearly very similar to the *least squares* and *recursive least squares* algorithms. The main difference is that the "process" in question, (the function telling the game-theoretic value of a position), is discrete, highly nonlinear and discontinuous. Making a single, seemingly insignificant move may change the value of a position from won to lost. Therefore it is unclear how well the approximations can work.

Evaluation

Several game-playing programs use methods like the one described to automatically adjust the parameters of their evaluation functions. The fact that Deep Blue uses the method has given it an unfairly good reputation, while the truth is that Deep Blue owes it first place among chess computers to its enormous speed, not its evaluation. The best evaluation functions are probably to be found in commercial programs running on slower hardware, and these evaluation functions have been produced by careful hand crafting.

As with Samuel's approaches, the method is troubled by the problem of *credit assignment,* and it is also very difficult to arrive at a reliable evaluation of the input positions. The nonlinearities of the problem may also cause troubles. It seems that the main advantage of the method is that it provides a simple way to give a hopefully reasonable tuning of a large set of parameters.

6. *Backgammon*

A well known game-playing achievement was when the program BKG 9.8 won a match of eight games of Backgammon over the then reigning world champion Luigi Villa with 7–1, see Berliner (1980). It should be noted, though, that the match did not concern the world champion title , and that post mortem analysis showed that the program did not play better than the human opponent, but was lucky with the dice rolls. However, it was the first time a computer program had won a match over a human world champion in any board or card game, and the program certainly played on par with the human opponent.

Berliner's program used no learning, but it is still interesting because Backgammon is best played without extensive tree searching. Instead, a good evaluation function is crucial. BKG relies on a well-tuned

hand crafted evaluation function where the weighting parameters, k_i, are smoothly varying with the type of position, making it nonlinear. The difference between Backgammon and other board games, such as Chess, Checkers, and Othello, is probably due to two factors. First, the dice rolls give a large degree of randomness to the game, so that any single move will not change the winning possibilities radically. Secondly, the relative simplicity of the game makes the game-theoretically correct evaluation smoother than in the other games.

Tesauro's and Sejnowski's Learning Program

The evaluation of Berliner's program was hand crafted and essentially concerned with estimating the value of different board patterns. It was therefore natural to try and use a neural network for evaluating the different patterns and selecting Backgammon moves, see Tesauro and Sejnowski (1989). This program used a three layer network to choose between generated moves. The inputs were coded into 459 nodes, of which 8 consisted of precomputed features such as number of men in different regions of the board, the number of *blots,* (pieces that may be hit, which are weak points), etc. The network had from 12 to 48 hidden nodes, used a training set of 3202 positions evaluated by a human player, (Tesauro), and needed about 100 to 200 hours of training on a SUN 3/160, using backpropagation. The resulting program played an intermediate level of Backgammon and had a 60% performance against the commercial program Gammontool, by SUN Microsystems.

Evaluation

The fact that the program managed to learn to play a respectable game of Backgammon is impressive. The domain is one of the largest that have been successfully tackled by a neural network approach. There are several weak points, however:

- Some hand crafted evaluation features were used. This helped to speed up the learning, but the program became reliant on a priori information.
- Lots of hand crafting was needed in selecting the examples, so that the program learned to handle different types of problematic positions.
- The human evaluation of the positions in the learning set was not completely reliable.
- The level of play was not so high. Berliner's BKG program plays far better using standard game-playing techniques and a hand crafted evaluation function.

The conclusion of this example is that it is indeed possible to learn a good evaluation function for Backgammon with a neural network. This is an impressive achievement in itself. On the other hand, conventional techniques still outperform the ones based on learning.

7. *Othello*

Another game that has been the target of research is Othello. Frey (1986) and Mitchell (1984) describe a project in which a large database of late middlegame Othello positions were used to tune the parameters of an evaluation function, much in the similar way to that used by Hsu *et al* for Deep Blue. The true score of the positions were computed by deep, (and thus very time-consuming), searches to the end of the game, and then the parameters of an evaluation function was adjusted to match the correct scores as well as possible.

As an important part of an Othello evaluation function seems to be to evaluate different edge and corner patterns, it is an obvious idea to try and train a neural network to perform this evaluation. Such project is described in the thesis of Steven Walker, Australia. The best Othello programs today use simple, hand crafted evaluation functions and rely on deep tree searches to play on par with the strongest human players. A project at the Department of Computer Engineering, Lund Institute of Technology, aims at building very fast hardware for Othello.

8. *Boxes*

Michie and Chambers have devised an general learning algorithm called BOXES, see Michie and Chambers (1968). It began as an algorithm for playing trivial games as 3×3 *naughts and crosses*. The idea is quite simple. A problem is broken down into sub-problems, and a score is kept over every possible action in every sub-problem. The algorithm plays a large number of games and chooses the actions of each sub-problem that has given the best outcome so far. In the case of a board game, there is a sub-problem, (a box), for every possible position of the game, and the possible actions are the possible moves from the position in question. As the number of possible positions for any non-trivial game is very large, (10^{30} for Checkers and Othello, and 10^{120} for Chess), the BOXES algorithm is not possible to use for them. However, Michie and Chambers used it to learn a somewhat different "game," that of balancing a pole on a cart.

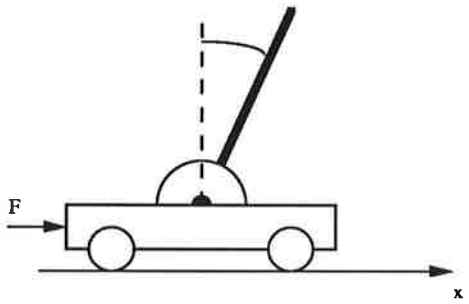


Figure 3. Michie's cart and pole system.

Figure 3 shows the *cart and pole* system used by Michie and Chambers in their experiment. A wheeled cart can move back and forth along a plane. On top of the cart there is a pole, and the task is to balance the pole by moving the cart. The input signals used by the BOXES algorithm were:

x The position of the cart on the track.

$\dot{x}$ The velocity of the cart.

θ The angle of the pole.

$\dot{\theta}$ The angular velocity of the pole.

The BOXES model was obtained by a rough quantization of the state space. Thus, x and θ were discretized into five values, and $\dot{x}$ and $\dot{\theta}$ into three. The x value intervals were three of equal size and two unbounded. The θ intervals were one quite small in the middle, two wider, and two unbounded. The $\dot{x}$ and $\dot{\theta}$ intervals consisted of one narrow interval and two unbounded. This means that the state space was divided into $5 \times 5 \times 3 \times 3 = 225$ *boxes*.

The output signal was the force F controlling the acceleration of the cart. It was discretized into only two values, + and −, or *left* and *right*. Each box contained a randomly chosen action from start. In the original experiment, the system was simulated on a computer with a sampling time of 20 Hz. It should be noted that "nice" values were chosen on the masses and length, so that the system was fairly easy to control. Even so, Michie and Chambers also put in a weak spring that tried to pull the pole towards the upright position, in order to make the system more easy to control. Later, physical systems have in fact been built, but the small number of boxes demands that the dimensions of the system be "nice" enough, otherwise the number of boxes will be too large for the learning algorithm to work in practise, see Bernhardsson and Larsson (1989), where a physical inverted pendulum was investigated. This process demands some $100 \times 40 \times 6$ boxes in order to be successfully controlled, which makes the learning impractically slow.

Learning Algorithm

The original learning algorithm was quite simple, and it is this algorithm that will be treated here. For every local box, some values are computed:

L_l The *left life*. A weighted sum of the number of sampling points during which the system managed to keep the pole from falling, (the life time), while using the action *left* in the box.

L_u The *left usage*. A weighted sum of the number of *left* actions actually used.

R_l The *right life*. A weighted sum of the life times while using the action *right* in the box.

R_u The *right usage*. A weighted sum of the number of *right* actions used.

In addition to this, two global values are updated:

G_l The *global life*. A weighted sum of the life times of the runs.

G_u The *global usage.* A weighted sum of the number of decisions taken.

All the sum are weighted so that there is a forgetting factor of 0.99. Thus, the updating rules for the global values are:

$$G_l = 0.99G_l + T_F$$
$$G_u = 0.99G_u + 1,$$

where T_F is the time in sampling points from the start of the run to when the pole fell. Thus, the quotient G_l/G_u is proportional to the average life time of the test runs.

The local values are updated according to the following formulas, (where it is assumed that the box in question is using the *left* action during the current run):

$$L_l = 0.99L_l + \sum_{i=1}^{N}(T_F - T_i)$$
$$L_u = 0.99L_u + N$$
$$R_l = 0.99R_l$$
$$R_u = 0.99R_u$$

With these values, the following calculations are performed:

$$v_L = \frac{L_l + 20\frac{G_l}{G_u}}{L_u + 20}$$
$$v_R = \frac{R_l + 20\frac{G_l}{G_u}}{R_u + 20}.$$

The control action is selected as *left* or *right* according to whether v_L is greater than v_R or not.

Results

The results of the learning tests varied within large magnitudes. After some 300 test runs the system was often able to balance the pole for 20 to 40 seconds. In one case it was never able to balance it more than in average 15 seconds, while in another case, after 600 tests it could balance it for 300 seconds in average. Michie and Chambers point out one successful run of more than 72 000 sampling points, corresponding an hour of simulated time.

As can be seen from these results, the learning process is not very stable, and it converges very slowly and unreliably. This may seem strange, as it is certainly possible to control the system with a standard state space feedback controller, and the learning algorithm should reach a discretized version of this sooner or later. The conclusion must be that the learning procedure is indeed extremely slow. Later experiments by other groups

have tried to better the speed of the learning by using more a priori information about the process.

Barto, Sutton, and Andersson improved on the experiment by designing two adaptive, neuron-like elements. They still used a predefined division of the state space and their program was able to balance the pole after some 100 trials. One reason for the better results may also have been that a higher sampling frequency of 50 Hz was used, see Barto, Sutton, and Andersson (1983).

Andersson used two predefined two-layer neural networks, one learning an evaluation function and the other learning the control actions. In this way, no predefined division of the state space was necessary. On the other hand, this program took some 10 000 trials to learn to balance the pole for an average of 140 seconds, see Andersson (1986).

In the CART experiment, the state trajectory was traced through each trial and a continuous interpolation-function replaced the state division. The program only considered states that was actually reached, and an elaborate analysis was used to pinpoint the erroneous control actions. The program was able to balance the pole indefinitely after only 16 trials, but a large amount of a priori knowledge was used in the algorithm, see Connell and Utgoff, (1987).

Interpretation

Michie's and Chambers' original algorithm is quite simple. It forms an average of the life times achieved by either *left* or *right* for each box. This average is weighted by a forgetting factor of 0.99 so that more recent observations will be more important. Then, very simply, the action with the higher life time is chosen, but the life time average is summed with the global life time average and normalized, so that when global life times increase, the adjustment of actions will be slower. For each box, the algorithm is similar to a proportional controller working on an averaged input with exponential forgetting, and the output is discretized into two levels only and the gain decreased in proportion to how successful the controller is.

Evaluation

Michie's and Chambers' BOXES algorithm is quite general, and *can* learn to control the cart and pole process. It still uses important a priori knowledge about the division of the state space, though, and given this, it is fair to say that it shows a low order of stability and an extremely slow convergence. An MRAS or STR would converge after a couple of pole movements while BOXES needs hundreds of complete test runs. Other approaches are either more general than BOXES, but then even slower in learning, or more efficient but then dependent on more specific knowledge. In the latter cases, it seems that the point of the experiment has been lost, and that the solutions are far too specific to the process. As such solutions, the proposed algorithms does not compare favorably with standard state space controllers.

Some Final Comments

Once again it is seen that the goal of learning will not be to outperform standard algorithms. Indeed, it is difficult enough to achieve successful learning at all. Michie himself has emphasized this, Michie (1990). It is also easy, it seems, to loose track of what is relevant research. The original BOXES experiment is general enough to be applied to many problems, while the solutions that are more efficient in learning use a lot of a priori information, which makes their general value doubtful.

But the most important observation is probably that a conventional model such as a state space description,

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

contains a large amount of useful a priori knowledge, and that learning without knowing even the structure of the process is very difficult.

9. *Neural Networks*

In recent years, neural networks have been successfully applied to pattern recognition and optimization tasks, see for example Hopfield (1982), Hopfield and Tank (1985), Burr (1988), Gorman and Sejnowski (1988), Sejnowski and Rosenberg (1987), and Widrow, Winter, and Baxter (1988). Two main types of neural networks have been used, the multilayer network and the Hopfield net.

The area of neural networks have grown immensely in the last years. Thus, the following is only an interpretation of the basic techniques, and no overview of the field. A good presentation of the research area, with a twist towards control, is found in Miller *et al* (1990).

Multilayer Networks

The most common neural network consists of a set, (layer), of input nodes, some layers of hidden nodes, and a layer of output nodes. Each node in the hidden and output layers receive as input a weighted sum of the nodes in the preceding layer. Some function, (often a sigmoid or truncation), is applied to this sum, and the result becomes the output of the node, to be used by the next layer, see Figure 4.

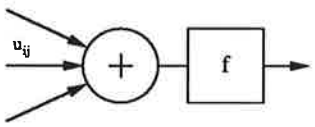


Figure 4. A single node of a multilayer neural network.

Some special types of nets have limitations on the number of layers and the functions used. Adaline is basically a one layer net, see Widrow and

Hoff, (1960). The multilayer network was originally called a Perceptron, see Rosenblatt (1962). In a CMAC network, (a special case of a Perceptron), the function applied is the identity, thus a CMAC is a linear summing device, see Albus (1975 a, b).

This general structure means that a multilayer neural network can be described as a static nonlinear map f, where

$$f(x) = F(WF(VF(Ux))).$$

Here F is the (nonlinear) function and U, V, and W the weighting matrices corresponding to the connections in the network, see Figure 5.

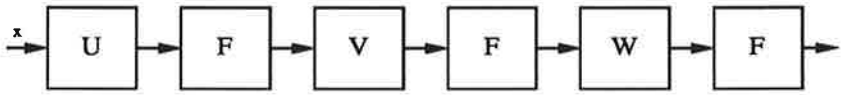


Figure 5. A multilayer neural network in block diagram form.

According to Weierstraß' theorem, every map $C(R^n, R^m)$ can be approximated to any degree by a polynomial, and it has been shown that a three layer neural network with an arbitrarily large number of nodes in the hidden layer can approximate any continuous function over a compact subset of R^n. This implies that neural networks with one hidden layer are capable of performing any characterization.

Interpretation

There is no special control interpretation of a multilayer neural network. It is simply a nonlinear map, and the use of neural networks as opposed to, say, polynomial approximations depends on whether the representation is versatile enough to provide a good basis for analysis and algorithm design. A neural network is simply one general way of "packing" a static nonlinear function.

Hopfield Networks

A Hopfield network usually consists of a layer of nodes and a feedback via time delay, see Figure 6.

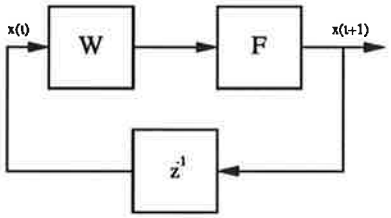


Figure 6. A Hopfield network in block diagram form.

Here, the inputs are the weights of the nodes and the output the stable states.

Interpretation

A Hopfield network is based on feedback, and in the case of a one layer network, the control interpretation is simple and well-known. In this case the Hopfield net is identical to a linear system with no inputs,

$$\dot{x} = Ax.$$

The difference from the standard use of such systems is that here the parameters of the matrix A are the "inputs" and the stable states of the system are the "outputs."

Weight Adjustment

Neural networks became popular when the *backpropagation* method for adjusting the weights was introduced, see Narendra and Parthasarathy (1988). In this method, the partial derivatives of an error criterion with respect to the weights in a multilayer neural network are determined and the weights are adjusted along the negative gradient to minimize the error function.

In order to perform backpropagation, the same network of nodes may be used, but the signals flow in the other direction, which explains the name backpropagation.

Interpretation

The backpropagation algorithm is more or less a pure MRAS method. The change of the weights, ω_i, is proportional to the partial derivative of the output error, e:

$$\frac{d\omega}{dt} = \gamma \frac{\partial e}{\partial \omega_i}.$$

This is a variant of an MRAS adjustment rule.

Thus, it can be concluded that backpropagation is a reliable method, but that it will suffer from all the same problems as the MRAS approach, the most important being that of ensuring persistent excitation.

Adaptive Control with Neural Networks

It is possible to use neural networks both for system identification and adaptive control, see Narendra (1990). Here one neural network is trained to behave as the process, (estimation), and one network is trained to use the estimated model to make the system behave as a reference model, see Figure 7.

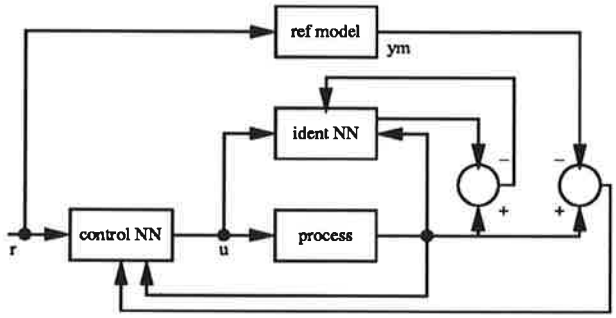


Figure 7. A self-tuning regulator based on neural networks. From Narendra (1990).

Interpretation

It is straight-forward to see that the proposed controller architecture is an indirect STR, where the estimation and design blocks have been implemented with neural networks. It is interesting to note that so far, no methods for making *direct* controllers based on neural networks exist, see Narendra (1990).

In conventional adaptive control, several assumptions are made in order to guarantee that the methods will work:

- The sign of the high frequency gain is known.
- The order of the plant is known.
- The relative degree of the plant transfer function is known.
- The zeros of the plant lie inside the unit circle.
- The reference model is linear.

These assumptions make it easier for the method to work with a successful result, and indeed they are necessary for allowing proofs of convergence and stability. It seems rather obvious that if a controller based on neural networks is based on a process that does not obey these assumptions, problems will occur. When the assumptions are obeyed, however, the neural network approach could be feasible.

Evaluation

It is hard to evaluate the success of neural networks as a whole. It is clear that they have been successful in specific tasks as pattern recognition, and that they may be useful in for example intelligent sensor methods.

Concerning control, very few results have so far been reached. When applied to problems which conventional adaptive controllers handle well, it may be supposed or at least hoped that the neural networks will also be successful. For more difficult problems, such as processes that violate some of the assumptions listed above, it should not be hoped that neural

networks will be better able to solve them, because the problems originate from the process and other circumstances, and may not be solvable whatever method is used.

An important difference between conventional control theory and neural networks is that, as neural networks are nonlinear, almost nothing of the known theory of stability and convergence can be used. So right now, using a neural network approach means that very little theoretical analysis can be done and no stability proofs given. The only way left is to use the methods and see what happens, which is a worse situation than is the case for conventional controllers.

10. A Comparison

Parameter adjustment has been used to provide a learning behavior in many domains, as can be seen from the examples above. Some general similarities and differences should be noted:

- The type of "process" may vary drastically. Adaptive controllers usually operate on reasonably nice and smooth processes, and are therefore often successful. The same is true for BOXES and neural networks for control, while the game-playing applications meet a very different process, the game-theoretic value function. This functions is discrete, but also highly nonlinear and irregular. Thus, there is no possibility to ensure stability and convergence. Indeed, it is quite surprising that good results are possible to obtain at all.
- A few learning algorithms can be proved to work under specified conditions, (Vapnik-Chervónenkis), but most of the algorithms do not provide any such proofs.
- The "difficulty" of the process greatly affects the speed of convergence. The adaptive controllers operate on the nicest processes and are thus the fastest, while Samuel's parameter adjustment is slower, even though it is still a kind of gradient method. Here, the complex process makes the convergence more difficult.
- The problem of persistent excitation varies greatly. In adaptive control, this problem has been thoroughly studied, and some countermeasures against so called "estimator windup" and other bad effects caused by lacking excitation are usually included in the algorithms. In alla the other examples mentioned, the problem is not handled at all. Instead, the methods rely on the general randomness of the experiment situations to provide enough stimulation.
- Some learning algorithms use quite much a priori information, e.g., the adaptive controllers. These algorithms are usually fast in convergence. Other algorithms, e.g., BOXES and neural networks use much less a priori information. The only predefined knowledge in these cases is a set of input signals and a failure or error signal. These algorithms are consequently slower in learning.

- Algorithms like BOXES, which rely on gathering a large statistic, (Monte Carlo-like methods), are considerably slower than gradient methods, (MRAS, backpropagation, Samuel's program), which in turn are slower than least square algorithms, (STR, Deep Blue, Frey's Othello evaluation).

11. Conclusions

Learning is a fascinating prospect, and much longed for. If a problem is difficult to solve, or demands knowledge that is not available or hard or costly to gain, how good it would be to build a machine that could learn by itself.

However, in most problem areas there are lots of knowledge, and furthermore, learning is much more difficult than one may at first guess, and as has been seen from the examples, it is often very slow and costly. Thus, learning is very seldom a viable option, and so far it has not led to any system that is *better* than a corresponding conventional non-learning system. So one conclusion is that the ideas about learning as the crown of methods, that will solve all problems, are wrong. Learning has so far not given better results than conventional techniques.

Learning is still an important research target, however, as learning processes are present in many activities. Therefore it is important to study and learn how learning works. But the goals must be modestly set. If learning is at all possible, that is a good feat in itself. In some limited areas, especially concerning neural networks for pattern recognition, learning has been quite successful, and hopefully, it will come to use in more areas and grow to a more and more successful discipline, slowly but firmly, in a step by step fashion.

The final observation is that it is important not to forget the conventional adaptive control methods, i.e., the MRAS and STR approaches. They are certainly the most successful examples of learning, and so far the only ones that have migrated from research to practical application in large numbers. They can also be used to give examples of what problems the other approaches may meet in the future. So far, the best example of learning *is* adaptive control.

12. References

ALBUS, J. S. (1975 a): "A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller, (CMAC)," *Transactions of the ASME*, September, 220–227.

ALBUS, J. S. (1975 b): "Data Storage in the Cerebellar Model Articulation Controller, (CMAC)," *Transactions of the ASME*, September, 228–233.

ANANTHARAMAN, T. S., M. S. CAMPBELL, and F.-S. HSU (1992): "Singular Extensions: Adding selectivity to Brute-Force Searching," *Artificial Intelligence*, **53**, 1, 99–109.

ANDERSSON, C. W. (1986): *Learning and Problem Solving With Multilayer Connectionist Systems*, Ph. D. Dissertation, Coins technical report 86–50, Amhearst, Massachusetts.

ARIKAWA, S., T. SHINOHARA, and A. YAMAMOTO (1992): "Learning Elementary Formal Systems," *Theoretical Computer Science*, **95**, 1, 97–113.

ÅSTRÖM, K. J. and B. WITTENMARK (1989): *Adaptive Control*, Addison-Wesley, Reading, Massachusetts.

BARTO, A. B., R. S. SUTTON, and C. W. ANDERSSON (1983): "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems," *IEEE Transactions on Systems, Man, and Cybernetics*, **13**, 5.

BELLMAN, R. (1961): *Adaptive Control Processes: A Guided Tour*, Princeton University Press, Princeton, New Jersey.

BERNHARDSSON, B. and J. E. LARSSON (1989): "Learning State Space Controllers," Technical report, Department of Automatic Control, Lund Institute of Technology, Lund.

BERLINER, H. (1980): "Computer Backgammon," *Scientific American*, June 1980, 54–62.

BURR, D. J. (1988): "Experiments on Neural Net Recognition of Spoken and Written Text," *IEEE Transactions on Acoustic, Speech, and Signal Processing*, **36**, 1162–1168.

CONNELL, M. E. and P. E. UTGOFF (1987): "Learning to Control a Dynamic Physical System," *Proceedings of the AAAI '87*, American Association for Artificial Intelligence, Seattle, Washington, pp. 456–460.

FREY, P. W. (1986): "Algorithmic Strategies for Improving the Performance of Game-Playing Programs," in Farmer, D., A. Lapedes, N. Packard, and B. Wendroff, (Eds.): *Evolution. Games, and Learning*, North-Holland, Amsterdam.

GASARCH, W. I. and C. H. SMITH (1992): "Learning via Queries," *Journal of the ACM*, **39**, 3, 649–674.

GORMAN, R. P. and T. J. SEJNOWSKI (1988): "Learned Classification of Sonar Targets Using a Massively Parallel Network," *IEEE Transactions on Acoustic, Speech, and Signal Processing*, **36**, 1135–1140.

HAUSSLER, D. (1988): "Qualifying Inductive Bias: AI Learning Algorithms and Valiant's Learning Framework," *Artificial Intelligence*, **36**, 1–2, 172–221.

HAUSSLER, D. (1992): "Decision Theoretic Generalizations of the PAC Model for Neural Net and Other Learning Applications," *Information and Computation*, **100**, 1, 78–150.

HAUSSLER, D., M. KEARNS, N. LITTLESTONE, and M. K. WARMUTH (1991): "Equivalence of Models for Polynomial Learnability," *Information and Computation*, **95**, 2, 129–161.

HELMBOLD, D., R. SLOAN, and M. K. WARMUTH (1992): "Learning Integer Lattices," *SIAM Journal on Computing*, **21**, 2, 240–266.

HO, K. L., Y. Y. HSU, and C. C. YANG (1992): "Short-Term Loaf Forecasting Using a Multilayer Neural Network with an Adaptive Learning Algorithm," *IEEE Transactions on Power Systems*, **7**, 1, 141–149.

HOPFIELD, J. J. (1982): "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proceedings of the National Academy of Sciences, USA*, **79**, 2554–2558.

HOPFIELD, J. J. and D. W. TANK (1985): "Neural Computational of Decisions in Optimization Problems," *Biological Cybernetics*, **52**, 141–152.

HSU, F.-S. (1987): "A Two Million Moves/Sec CMOS Single Chip Chess Move Generator," *1987 ISSCC Digest of Technical Papers*, p. 278.

HSU, Y. Y. and C. C. YU (1992): "A Self-Learned Fault-Diagnosis System Based on Reinforcement Learning," *Industrial and Engineering Chemistry Research*, **31**, 8, 1937–1946.

HEINZINGER C., D. FENWICK, B. PADEN, and F. MIYAZAKI (1992): "Stability of Learning Control with Disturbances and Uncertain Initial Conditions," *IEEE Transactions on Automatic Control*, **37**, 1, 110–114.

IOANNIDIS, Y. E., T. SAULYS, and A. J. WHITSITT (1992): "Conceptual Learning in Database Design," *ACM Transactions on Information Systems*, **10**, 3, 265–293.

KALABA, R. E. and F. E. UDWADIA (1991): "An Adaptive Learning Approach to the Identification of Structural and Mechanical Systems," *Computers and Mathematics with Applications*, **22**, 1, 67–75.

LENAT, D. B. (1977): "The Ubiquity of Discovery," *Artificial Intelligence*, **9**, 3.

LI, C. J. and J. C. TZOU (1992): "A New Learning Fuzzy Controller Based on the P-Integrator Concept," *Fuzzy Sets and Systems*, **48**, 3, 297–303.

MAHADEVAN, S. and J. CONNELL (1992): "Automatic Programming of Behavior-Based Robots Using Reinforcement Learning," *Artificial Intelligence*, **55**, 2–3, 311–365.

MAHMOUD, M. S., S. KOTOB, A. A. ABOUELSEOUD, H. M. ELSAYED (1992): "A Learning Rule-Based Control System," *Information and Decision Technologies*, **18**, 1, 55-66.

MICHIE, D. (1990): private communication.

MESSNER, W., R. HOROWITZ, W. W. KAO, and M. BOALS (1991): "A New Adaptive Learning Rule," *IEEE Transactions on Automatic Control*, **36**, 2, 188–197.

MICHIE, D. and R. A. CHAMBERS (1968): "Boxes: An Experiment in Adaptive Control," in Dale, E. and D. Michie, (Eds.): *Machine Intelligence 2*, Oliver & Boyd, London.

MITCHELL, D. H. (1984): "Using Features to Evaluate Positions in Expert's and Novice's Othello Games," Master's thesis, Northwestern University, Evanston, Illinois.

MOORE, K. L., M. DAHLEH, and S. P. BHATTACHARYYA (1992): "Iterative Learning Control—A Survey and New Results," *Journal of Robotic Systems*, **9**, 5, 563–594.

NARENDRA, K. S. (1990): "Adaptive Control Using Neural Networks," in Miller, W. T., R. S. Sutton, and P. J. Werbos, (Eds.): *Neural Networks for Control*, MIT Press, Cambridge, Massachusetts.

NARENDRA, K. S. and K. PARTHASARATHY (1988): "A Diagrammatic Representation of Back Propagation," Technical report 8815, Center for Systems Science, Department of Electrical Engineering, Yale University, New Haven, Connecticut.

NEAL, R. M. (1992): "Connectionist Learning of Belief Networks," *Artificial Intelligence*, **55**, 1, 71–113.

NEWELL, A. and H. A. SIMON (1963): "GPS, A Program That Simulates Human Thought," in Feigenbaum, E. A. and J. Feldman, (Eds.): *Computers and Thought*, McGraw-Hill, New York.

OPPER, M. and D. HAUSSLER (1991): "Generalization Performance of Bayes Optimal Classification Algorithm for Learning a Perceptron," *Physical Review Letters*, **66**, 20, 2677–2680.

PITAS I., E. MILIOS, and A. N. VENETSANOPOULOS (1992): "A Minimum Entropy Approach to Rule Learning from Examples," *IEEE Transactions on Systems, Man, and Cybernetics*, **22**, 4, 621–635.

ROSENBLATT, F. (1962): *Principles of Neurodynamics*, Spartan, New York.

SAMUEL, A. L. (1963): "Some Studies in Machine Learning Using the Game of Checkers," in Feigenbaum, E. A. and J. Feldman, (Eds.): *Computers and Thought*, McGraw-Hill, New York.

SCHAEFFER, J., J. CULBERSON, N. TRELOAR, B. KNIGHT, P. LU, and D. SZAFRON (1992): "A World Championship Caliber Checkers Program," *Artificial Intelligence*, **53**, 2–3, 273–290.

SEJNOWSKI, T. J. and C. R. ROSENBERG (1987): "Parallel Networks that Learn to Pronounce English Text," *Complex Systems*, **1**, 145–168.

TESAURO, G. and T. J. SEJNOWSKI (1989): "A Parallel Network that Learns to Play Backgammon," *Artificial Intelligence*, **39**, 3, 357–390.

TORASSO, P. (1991): "Supervising the Heuristic Learning in a Diagnostic Expert System," *Fuzzy Sets and Systems*, **44**, 3, 357–372.

VALIANT, L. G. (1984): "A Theory of the Learnable," *Communications of the ACM*, **27**, 11, 1134–1142.

VAPNIK, V. N. (1982): *Estimation of Dependences Based on Empirical Data*, Springer Verlag, New York.

VAPNIK, V. N. and A. Y. CHERVÓNENKIS (1971): "On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities," *Theor. Probab. Appl.*, **16**, 2, 264–280.

WIDROW, B., R. G. WINTER, and R. A. BAXTER (1988): "Layered Neural Nets for Pattern Recognition," *IEEE Transactions on Acoustic, Speech, and Signal Processing*, **36**, 1109–1118.

WIDROW, B. and M. E. HOFF, JR. (1960): *IRE WESCON Convention Record*, pt. 4, pp. 96–104.

WINSTON, P. H. (1975): "Learning Structural Descriptions from Examples," in Winston, P. H., (Ed.): *The Psychology of Computer Vision*, McGraw-Hill, New York.

WINSTON, P. H. (1980): "Learning and Reasoning by Analogy," *Communications of the ACM*, **23**, 12, 689–703.