CODEN: LUTFD2/(TFRT-5401)/1-62/(1989)

Design av en estimator för en DC-motor

Lars Norrman
Ola Nilsson

Institutionen för Reglerteknik
Lunds Tekniska Högskola
Maj 1989

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

Document name
Master Thesis

Date of issue
May 1989

Document Number
CODEN: LUTFD2/(TFRT-5401)/1-62/(1989)

Author(s)
Lars Norrman and Ola Nilsson

Supervisor
Rolf Johansson

Sponsoring organisation

Title and subtitle
Design av en estimator för en DC-motor. (Design of an estimator for a DC drive.)

Abstract

We have designed an estimator of and a regulator for a DC drive. The estimator and regulator have, together with an operator interface, been implemented at an IBM PC in the language MODULA2.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

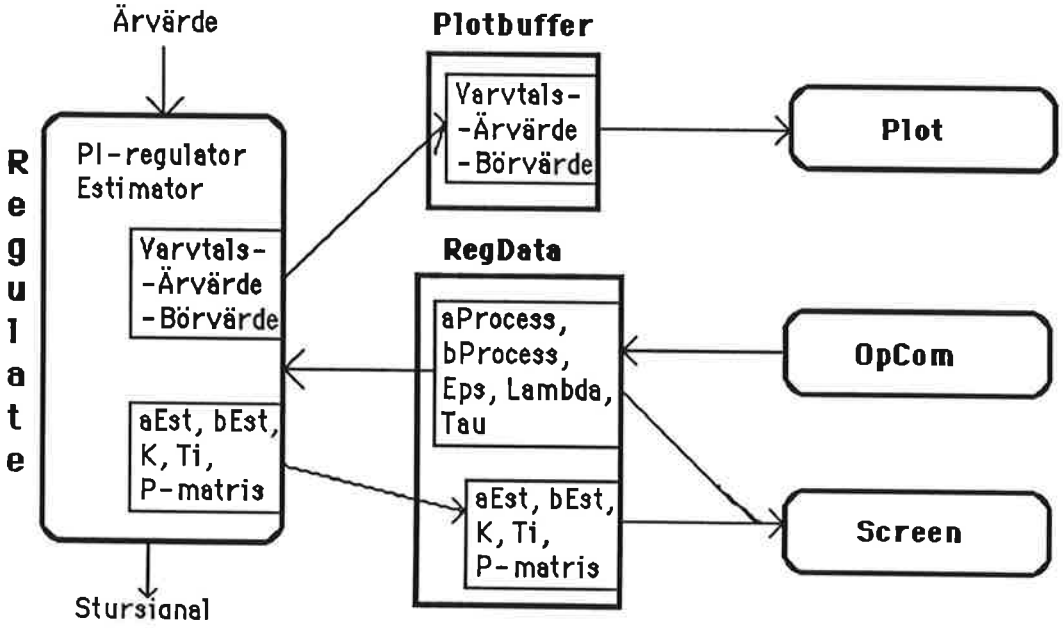
Language
Swedish

Number of pages
62

Recipient's notes

Security classification

DESIGN AV EN ESTIMATOR FÖR EN DC-MOTOR



EXAMENSARBETE VID
INSTITUTIONEN FÖR
REGLERTEKNIK, LTH

MAJ 1989

LARS NORRMAN
OLA NILSSON

FÖRORD

Syftet med detta examensarbete är att undersöka en metod för processestimering presenterad i Johansson(1986), med avseende på konvergens och störtålighet, samt att implementera denna metod i ett realtidsspråk, för styrning av en likströmsmotor.

Vårt ursprungliga syfte var att implementera metoden för styrning av ASEA Drives' växelströmsmotorer, men av olika praktiska skäl gick detta ej att genomföra.

Framförallt av på grund av att vårt ursprungliga syfte ej var genomförbart har vårt exjobb dragit ut på tiden. Vi är tacksamma mot alla på institutionen, som har hjälpt oss när det behövts, och speciellt mot Rolf Johansson, som nu inte längre kan räkna sig som handledare till "Exjobbsveteranerna".

LUND I MAJ 1989

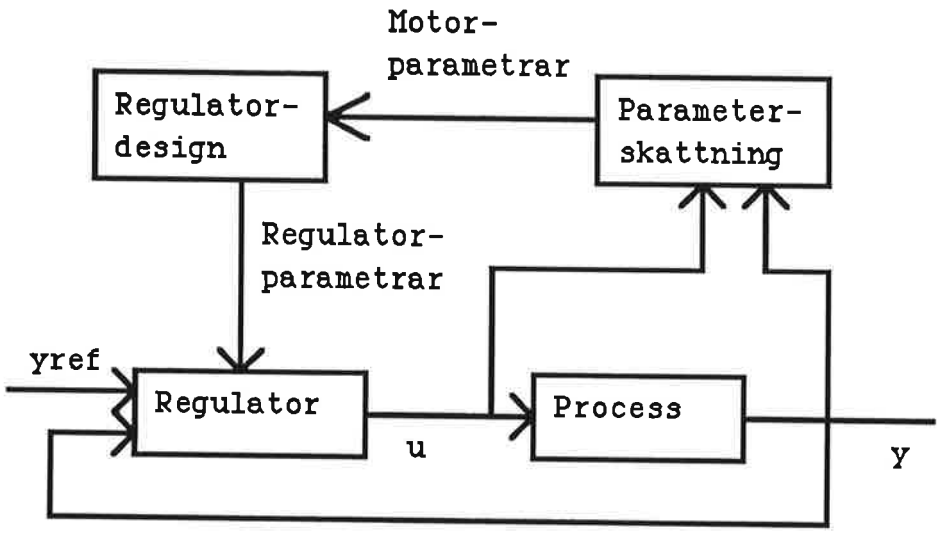
Lars Norrman
Ola Nilsson

INNEHÅLLSFÖRTECKNING

1. PROBLEM

I Johansson (1986) beskrivs en metod för att skatta parametrar i ett tidskontinuerligt system.

Metoden bygger på att processens in- och utsignal lågpassfiltreras. Genom minsta-kvadratoptimering skattas processens parametrar i kontinuerlig tid. (Se t.ex. Åström & Wittenmark, kapitel 10). Därefter beräknas K och Ti så att det slutna systemet får önskad överföringsfunktion.



Figur 1 Adaptivt reglersystem

Vid bestämning av parametrar för digital reglering av en samplad process med minsta-kvadrat-metoden finner man ofta att det tar lång tid för regulatorparametrarna att ställa in sig vid en förändring av processdynamiken.

Syftet med vårt examensarbete är att undersöka om den i Johansson(1986) beskrivna metoden (vi kommer i fortsättningen att referera till denna metod som estimatorn) för parameterskattning fungerar bättre vid styrning av en liten likströmsmotor (institutionens servo-uppställning).

Vi kommer att studera dels uppförandet vid adaptiv reglering, dels estimeringen utan återkoppling från estimatorn till

regulatorn.

Innan vi försökte styra och estimera motorn, modellerade vi motor, regulator och estimator i SIMNON, för att bättre kunna undersöka känsligheten för parameterval etc.

Eftersom vår uppgift var att undersöka om estimatorn var bättre måste vi ha något att jämföra den med. Vid en laboration i digital reglering används en adaptiv digital regulator för att styra "vattentankarna". Vi använde denna regulator för att styra vår likströmsmotor.

De erfarenheter vi gjorde då var att när väl regulatorn var trimmad till optimala parametrar, reagerade den långsamt på förändringar av motor-karaktäristiken, samtidigt som den var känslig för störningar.

I Johansson(1986) anges följande fördelar för metoden:

- Modelleringen sker i kontinuerlig tid
- Skattningen konvergerar relativt snabbt
- Störningstålig
- Möjligt att använda olika samplingsintervall för reglering och estimering vilket är en fördel om tillgången på datortid är en begränsande faktor.

Vi kommer i första hand att undersöka hur snabbt skattningen konvergerar och hur störningstålig den är.

2. BESKRIVNING AV ALGORITMEN

Innan vi tillämpar metoden på vårt problem ger vi en beskrivning i mer generella termer (Johansson).

Ett linjärt tidsinvariant system beskrivs i kontinuerlig tid av

$$A(p)y = B(p)u; \qquad A(p) = p^n+a_1p^{n-1}+...+a_n \qquad B(p) = b_1p^{n-1}+b_2p^{n-2}+...+b_n$$

där p=d/dt.

Genom att införa lågpassoperatorn z:

$$z = 1/(1+p\tau)$$

kan systemet transformeras till

$$A^*(z)y = B^*(z)u \qquad A^*(z) = 1+\alpha_1z+\alpha_2z^2+...+\alpha_nz^n \qquad B^*(z) = \beta_1z+\beta_2z^2+...+\beta_nz^n$$

Genom att införa vektorerna

$$\theta_\tau^T = (-\alpha_1 -\alpha_2 \ldots -\alpha_n \quad \beta_1 \beta_2 \ldots \beta_n)$$
$$\phi_\tau T = ([zy](t) [z^2y](t) \ldots [z^ny](t) [zu](t) \ldots [z^nu](t))$$

kan ovanstående samband formuleras som

$$y(t) = \theta_\tau^T\phi_\tau$$

Vårt mål är att skatta $\theta_\tau(t)$ och transformera den till $\theta(t)$

$$\theta^T(t) = (-a_1 -a_2 \ldots -a_n \quad b_1 b_2 \ldots b_n)$$

Relationen mellan θ_τ och θ är

$$\theta_\tau = F_\tau\theta + G_\tau$$

där

$$F_\tau = \begin{bmatrix} M_\tau & 0 \\ 0 & M_\tau \end{bmatrix} \qquad M_\tau = \begin{bmatrix} m_{11} & 0 & \cdots & 0 \\ \vdots & \ddots & & \vdots \\ & & & 0 \\ m_{n1} & \cdots & & m_{nn} \end{bmatrix}$$

$$m_{ij} = (-1)^{i-j}\binom{n-j}{}t^j$$

$$G_\tau = (g_1 \ldots g_n\, 0 \ldots 0) \qquad g_i = \binom{n}{}(-1)^i$$

$$\theta = F_\tau^{-1}(\theta_\tau - G_\tau)$$

$\theta_\tau(t)$ skattas med minsta-kvadratmetoden genom minimering av

$$J = \int|y - \theta_\tau\phi_\tau|^2dt$$

Algoritmen för att uppdatera θ_τ kan i kontinuerlig tid skrivas

$$\theta_\tau\,(t) = P_c(t)\phi_\tau(t)(y(t) - \theta_\tau^T(t)\phi_\tau(t))$$
$$P_c(t) = -P_c(t)\phi_\tau(t)\phi_\tau^T(t)P_c(t)$$

I diskret tid motsvaras detta av

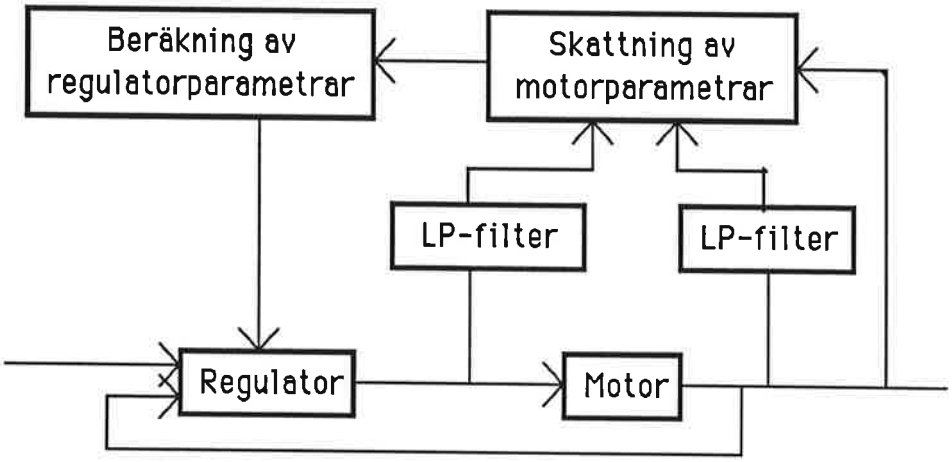
$$\theta_\tau(t) = \theta_\tau(t-h) - P(t)\phi_\tau(t)(y(t) - \theta_\tau^T(t-h)\phi_\tau(t))$$
$$P(t) = (P(t-h) - Q(t))/\lambda$$
$$Q(t) = P(t-h)\phi_\tau(t)\phi_\tau^T(t)P(t-h)/(\lambda + \phi_\tau^T(t)P(t-h)\phi_\tau(t))$$

där λ är en glömskefaktor, som normalt är $0.95< \lambda \leq 1$.

3. SIMULERING I SIMNON

Metoden för parameterskattning, som beskrivits i Kapitel 2., har vi realiserat och simulerat i SIMNON. SIMNON är ett programpaket framtaget vid Institutionen för Reglerteknik vid LTH. SIMNON tillåter att man blandar enheter beskrivna i diskret tid och i kontinuerlig tid.

I SIMNON har vi delat in systemet i block, enligt figur 2.



Figur 2 Blockindelning i SIMNON

Vi kommer att beskriva systemet genom att i tur och ordning gå igenom delsystemen. Programtexten i SIMNON bifogas i bilaga 1.

3.1. Delsystem motor

Motorn har vi realiserat som en tidskontinuerlig process i SIMNON med tillståndsbeskrivningen

$$\dot{x} = ax + bu$$
$$y = x$$

x varvtal, tillstånd
u motorström
y varvtal, utsignal

Vid våra simuleringar har vi använt kallat enheten "MOTOR", se bilaga 1. Värdena på a och b har varit i storleksordningen

$$a \approx -10 \qquad b \approx 2$$

3.2. Delsystem regulator

Regulatorn är realiserat i ett tidsdiskret system, men algoritmen motsvarar en tidskontinuerlig PI-regulator, dvs utan bakåtoperatorer. PI-regulatorn beskrivs av

$$e := y_{ref} - y$$
$$y := K*e + I$$
$$I := I + K*e*h/T_i$$

Dessutom finns utsignalbegränsning och eliminering av integratoruppvridningen.

Vid simuleringarna har regulatorn ofta varit frånkopplad. Då har motorn styrts direkt med en fyrkantvåg. Detta motsvarar trimningsmoden. Normalt driftläge motsvaras av att regulatorn är inkopplad med förinställda parametrar. Genom att hela tiden återkoppla de beräknade regulatorparametrarna får man en adaptiv regulator. Även de bägge senare fallen har simulerats med hyggligt resultat, dock med sämre estimering som följd.

Samplingstiden (h) är av någon anledning satt till 3.3 ms.

Implementeringen har kallats "REGUL" i SIMNON, se bilaga 1.

3.3. Delsystemet för beräkning av regulatorparametrar

I SIMNON har vi realiserat detta som en egen process med de skattade motorparametrarna (a och b) som indata och regulatorparametrarna (K och T_i) som utdata. K och Ti beror, förutom på motorparametrarna, på önskad överföringsfunktion för det slutna systemet. Med

$AY = BU$ $\quad A = s-a \quad B = b$

$CU = D(Y_{ref} - Y) \quad C = s \quad D = K/T_i + Ks$

får vi

$$(AC + BD)Y = BDY_{ref}.$$

Det slutna systemets poler placeras i

$$p_{1,2} = p_r \pm ip_i.$$

Genom koefficientidentifiering i

$$(s-p_1)(s-p_2) = s(s-a) + b(K/T_i+Ks)$$

erhålls

$$K = (-a -2p_r)/b$$
$$T_i = (-a-2p_r)/(p_r^2).$$

Implementeringen har vi kallat REGPAR, se bilaga 1.

3.4. Delsystem för skattning av motorparametrar

För skattningen av motorparametrarna a och b använder vi den metod som beskrivits i kapitel 2. Följande beteckningar används:

zu,zy	lågpassfiltrerade in- och utsignaler från motorn
α,β	transformerade motorparametrar

$$\phi = (zy\ zu)^T = (\phi_1\ \phi_1)^T$$
$$\theta = (-\alpha\ \beta)^T = (\theta_1\ \theta_2)^T$$

Varje beräkningscykel genomlöps följande algoritm:

$$\text{Error} := y - \theta^T\phi$$
$$P := (P - (P\phi\phi^TP)/(\lambda + \phi^TP\phi))/\lambda$$
$$\theta := \theta + P\phi\cdot\text{Error}$$
$$a_{est} := (\theta_1 -1)/\tau$$
$$b_{est} := \theta_2/\tau$$

Ett problem vid rekursiva minsta-kvadratalgoritmer av den här typen är att P-matrisen har en tendens att öka exponentiellt om

glömskefaktorn $\lambda < 1$. (Se t.ex. Ljung & Söderström). Ett sätt att undvika detta är att sätta ett tröskelvärde för skattningsfelet ERROR. Om

$$ERROR < e$$

så uppdateras inte P och θ. Ett annat sätt att undvika okontrollerad tiilväxt av P är att sätta en referensmatris P_0, som viktas in i P beroende på hur stor λ är.

I SIMNON har detta delsystem kallats MOTPAR, se bilaga1.

3.5. Delsystemet lågpassfilter

Lågpassfiltret har implementerats i diskret tid för att efterlikna en implementering i dator.

I kontinuerlig tid skrivs lågpassoperatorn

$$z = 1/(1 + p\tau), \qquad p = d/dt.$$

I diskret tid blir detta för u

$$zu_t := \exp(-h/\tau)\cdot zu_{t-h} + (1 - \exp(-h/\tau))\cdot u_t = -a_\tau \cdot zu_{t-h} + b_\tau \cdot u_t$$

Vid filtreringen av y har både y(t) och y(t-h) använts som indata för att ge bättre överensstämmelse med ett analogt filter:

$$zy_t := -a_\tau \cdot zy_{t-h} + b_0 \cdot y_t + b_1 \cdot y_{t-h}, \qquad b_0 + b_1 = b_\tau$$

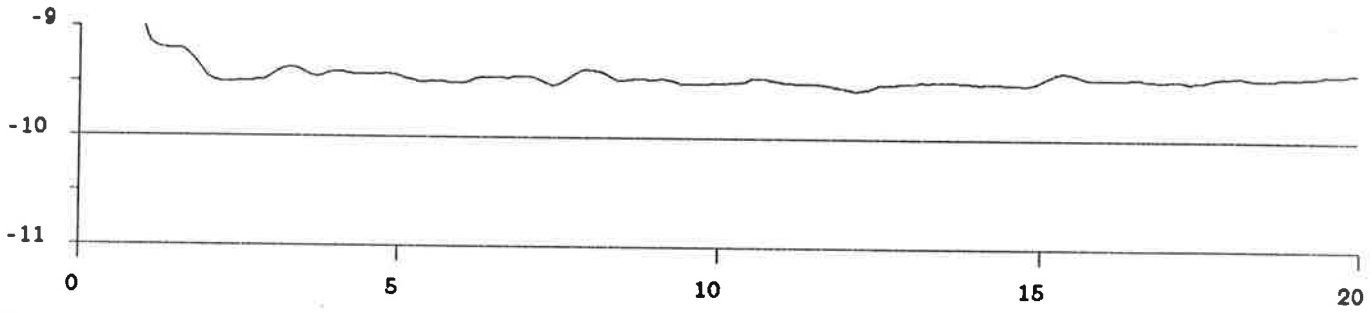
I SIMNON har implementeringen kallats LPFILT, se bilaga 1.

3.6. Sammanbindning av delsystemen

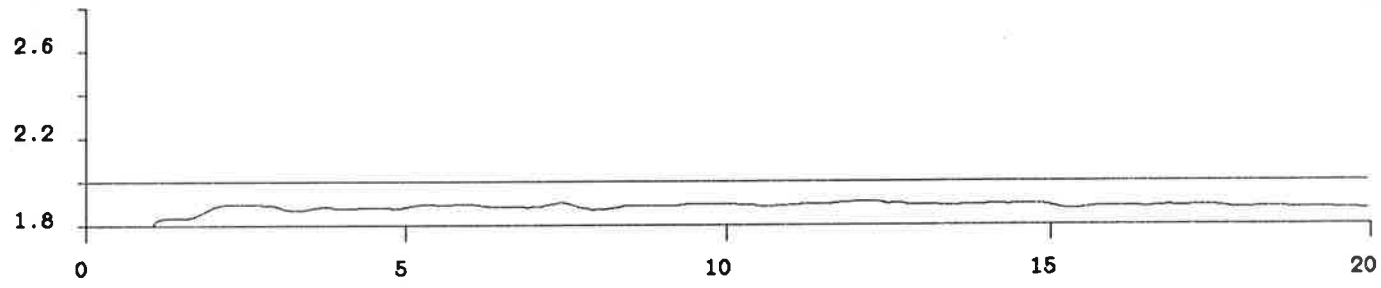
De olika delsystemen förenas med systemet CON, se bilaga 1. Signaler som förekommer i flera delsystem kopplas ihop.

3.7 Resultat av realiseringar i SIMNON

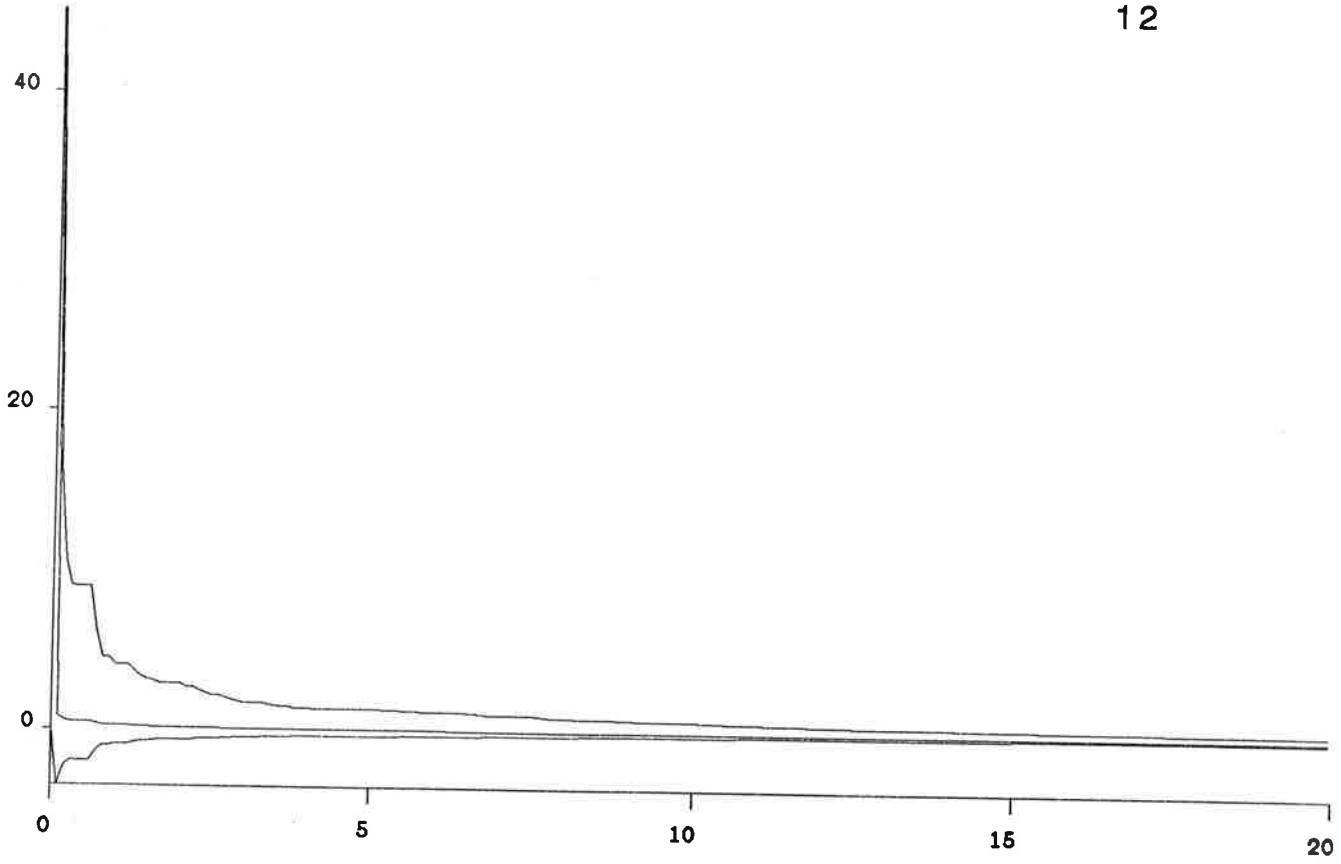
Eftersom SIMNON-övningarna i första hand var till för att ge oss en känsla för hur hur algoritmen beter sig har vi ingen utförligare resultatpresentation här.



Figur 3 *Motorparameter a, skattning och riktigt värde. Skattningen är den översta kurvan och det riktiga värdet den nedersta.*



Figur 4 *Motorparameter b, skattning och riktigt värde. Skattningen är den understa kurvan, och det riktiga värdet drn översta.*



Figur 5 P-matrisens element.

Vi ser att insvängningstiden är ca 0.2 sekund. Därefter ligger skattningen inom ±10% från det riktiga värdet. P-matrisen hamnar på en rimlig nivå, dvs den tillväxer inte exponentiellt.

Om Lambda varieras förändras skattningarnas stabilitet. Om Lambda < 0.99 blir skattningarna instabila. Storleken på Tau påverkar framförallt högfrekvensvariationer i skattningen. Ligger Tau utanför intervallet 0,05 < Tau < 1,0 får man inte meningsfulla resultat. Epsilon måste ligga inom intervallet 0,005 < Epsilon < 0,02. Om Epsilon > 0,02 "låses" skattningen vid en felaktig nivå. Om Epsilon < 0.005 växer P-matrisen exponentiellt.

Samtliga slutsatser här gäller för det system vi realiserat. Med en annan motor eller med andra störningar, hade parameterinställningarna antagligen hamnat på andra nivåer. Den viktigaste slutsatsen för oss från SIMNON-simuleringarna var att metoden bör gå att realisera på en riktig motor.

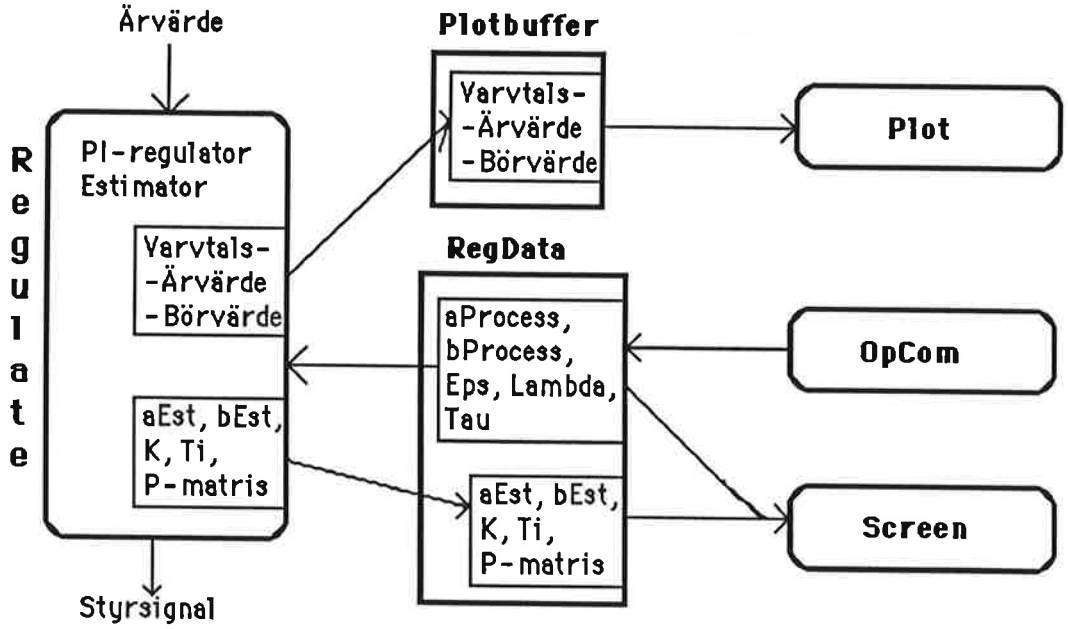
4. IMPLEMENTERING I MODULA

4.1. Inledning

Efter simuleringarna i SIMNON var nästa steg att implementera det framtagna systemet i realtidsmiljö. Realtidsspråket vi använt oss av benämns MODULA-2. Det är ett språk baserat på PASCAL, men modernare och vidareutvecklat för att kunna användas i realtid

De växelverkande och jämlöpande aktiviteter, processer, som finns i vårt program har vi kallat Regulate, Plot, Screen, och Opcom. För att garantera ömsesidig uteslutning av processer som opererar på gemensamma data låter vi all kommunikation mellan processerna ske via monitorer, RegData och PlotBuffer.

I figur 6 beskrivs programmets struktur. I avsnitt 4.2. kommenteras varje process och monitor, och i 4.3. beskrivs modulen Regulate, som innehåller regulator och estimator. Programlistan bifogas i bilaga 2.



Figur 6 Realtidsprogrammets struktur.

4.2. Beskrivning av realtidsprogrammet

Processer:

Regulate: Här finns PI-regulatorn och estimatorn. Regulate läser indata från varvtalsgivaren och hämtar parametrar från RegData. Styrsignal beräknas och skickas till motorn och PlotBuffer.

Plot: Plot hämtar data från PlotBuffer och plottar kurvor för varvtalsärvärde och varvtalsbörvärde på skärmen.

Screen: Denna modul hämtar data från RegData och presenterar informationen på skärmen. De storheter som presenteras på skärmen är aProcess, bProcess (polplacering), Epsilon, Lambda, Tau (Estimatorparametrar), aEst, bEst (estimerade motorvärden), K, Ti, P11,P12, P21, P22 och On/Off.

OpCom: Kommunikationen med operatören hanteras av denna process. Data lagras i RegData. Av de värden som presenteras av Screen kan följande ändras av operatören: aProcess, bProcess, Epsilon, Lambda, Tau och On/Off.

Monitorer:

RegData: Denna monitor innehåller alla data och de parametrar som är relevanta för regulatorn. Dessa måste lagras i en monitor, eftersom såväl Regulate som OpCom måste ha access till dem.

PlotBuffer: I denna monitor lagras data som skall plottas på skärmen inom kort. Regulate lagrar data här, som senare läses av Plot.

4.3 Beskrivning av modulen Regulate

I modulen Regulate finns förutom processen Regulate (=Process), proceduren SystemIdentification, som innehåller estimatorn. Insignaler är varvtal och styrsignal. Dessa lågpasstransformeras innan P-matrisen och Theta uppdateras. Theta invers-transformeras till aEst och bEst. Utdata är aEst och bEst, som är skattningen av motorparametrarna.

Processen Regulate läser Är-värde och hämtar Bör-värde och beräknar styrsignalen, som sänds till motorn. SystemIdentification anropas med styrsignal och Är-värde. K och Ti beräknas genom polplacering ur aEst och bEst.

5. FÖRSÖKSRESULTAT

5.1. Försöksbeskrivning

Med en PC har vi styrt en liten likströmsmotor (institutionens lab-uppställning). I processen Regulate (regulatorn) genereras en intern fyrkantvåg som bör-värde. Med hjälp av en temporär rutin har intressanta signaler i regulatorn och estimatorn samplats och skickats till en ASCII-fil, för senare plottning med hjälp av Matlab. Följande mätdata registreras i varje försök:

- P11, P12, P21, P22	P-matrisens element
- aEst, bEst	Estimerade parametrar
- Reference	Börvärde
- ProcessOutput	Ärvärde
- ControlSignal	Styrsignal
- K, Ti	Regulatorparametrar

De enda enheter som behöver förklaras är ärvärdets och styrsignalens. För båda gäller att ±1 motsvarar ±10Volt.

Registreringen har skett under 10 sekunder med samma frekvens som regleringen (=10Hz).

Följande driftsfall finns dokumenterade i bilagor:

	Adaptiv reglering	PI-reglering
Startsekvens:	Bilaga 3:1	Bilaga 3:2
Stabil sekvens:	Bilaga 3:3	Bilaga 3:4
Laststörning:	Bilaga 3:5	Bilaga 3:6

I försöken med PI-reglering redovisas inte regulatorsignalerna utan endast P-matrisens element och de skattade parametrarna.

Startsekvensen avser att visa hur snabbt estimatorn konvergerar. Registreringen av värden har påbörjats samtidigt som regulatorn startats (fram tills dess var styrsignalen =0). Den stabila sekvensen skall visa dels hur "lugn" signalen är, dels relationen mellan parametervärdena i det stabila läget och parametervärdena från startsekvensen. Registreringen har skett

ett par minuter efter start av regulatorn. De estimerade parametrarna har då inte visat några tendenser till att "glida". Laststörningen har åstadkommits genom att ett motriktat drivande moment lagts på, sedan de estimerade motorparametrarna stabiliserats. Detta moment har lagts på cirka 2 sekunder efter start av registreringen.

Värden på intressanta parametrar under försöken:
lambda = 0.99
tau = 0.2 s

5.2. Presentation av försöksresultat

5.2.1 Startsekvens, Adaptiv reglering

P-matrisen konvergerar förhållandevis snabbt. Efter 25-30 sampel har en någorlunda stabil nivå nåtts. Motorparametrarna har redan efter ca 40 sampel nått ett någorlunda stabilt läge:
aEst(40 sampel) = -0.17
bEst(40 sampel) = 1.1
Efter 80 sampel är parametervärdena:
aEst(80 sampel) = -0.19
bEst(80 sampel) = 1.1
Redan efter mindre än 10 sampel har regulatorparametrarna antagit fungerande värden:
K(8 sampel) = 3.1
Ti(8sampel) = 0.4

5.2.2. Startsekvens, PI-reglering

Efter 10-15 sampel har P-matrisen nått en någorlunda stabil nivå. Också aEst och bEst når då någorlunda stabila värden:
aEst(10sampel) = -0.05
bEst(10sampel) = 0.6
aEst(80sampel) = -0.13
bEst(80sampel) = 0.8
Regulatorns in- och utsignaler och parametrar redovisas inte vid PI-reglering.

5.2.3. Stabil sekvens, Adaptiv reglering

P-matrisen varierar med ±5% jämfört med genomsnittsvärdena. Andra försök med värden på lambda närmre 1 ger mindre svängningar mellan börvärdesändringarna.

aEst ligger inom intervallet -0.28±5%.

bEst ligger inom intervallet 1.63±4%

K varierar med ±4% och Ti med mindre än ±1%.

5.2.4. Stabil sekvens, PI-reglering

Överensstämmer med resultaten vid adaptiv reglering bortsett från att genomsnittsvärdena är:

aEst = -0.27
bEst = 1.57

5.2.5 Laständring, Adaptiv reglering

Vid ca 20 sampel lades ett moment på. Vi konstaterar att inga kraftiga svängningar i de estimerade värdena syns, men att de långsamt "glider" iväg för att efter ca 500 sampel ha stabiliserat sig på en ny nivå.

5.2.6 Laständring, PI-reglering

Vid ca 20 sampel lades ett moment på. Vi konstaterar att inga kraftiga svängningar i de estimerade värdena syns, men att de något mindre långsamt "glider" iväg för att efter ca 100 sampel ha stabiliserat sig på en ny nivå.

5.2.7 Övriga observationer

Vid en jämförelse mellan startsekvenserna med Adaptiv- resp. PI-reglering kan följande iakttagelser göras:

- Den adaptiva regulatorns parametrar varierar kraftigt under

de första 10 samplen, vilket ger dålig reglering.
- I PI-fallet hittar estimatorn tidigare en någorlunda stabil nivå, men den är längre ifrån den långsiktiga nivån.

Bortsett från transienteffekterna i uppstarten liknar insvängningsförloppet det vid laststörningar.

5.3. Slutsatser

De estimerade parametrarna konvergerar ganska snabbt (<50sampel) efter uppstart eller dynamikändring mot ett värde inom en faktor 2 från det korrekta för att sedan över 200-500 sampel långsamt glida till rätt nivå.

Valet av glömskefaktor är naturligtvis viktigt för konvergenshastigheten, men det påverkar också hur mycket P-matrisen och estimeringarna varierar vid konstant dynamik i processen.

Vi har funnit att genom att sätta Lambda = 1 kommer man att få mycket stabila och korrekta estimeringar, men om processdynamiken fluktuerar snabbt hinner inte estimatorn med. Om inte stegändringar (eller andra störningar) genereras regelbundet har estimatorn en tendens att glida iväg, vilket är normalt för en minsta-kvadrat-algoritm.

Estimatorn verkar vara mest lämpad för processer med långsamt varierande dynamik, men där små stegstörningar accepteras, eller som "tuner" vid installation/uppstart av en process (istället för manuell inställning, vilket kan vara tidsödande).

LITTERATURFÖRTECKNING

Johansson, R. (1986). Recursive Identification of Continous Time Dynamic Systems, Lunds Tekniska Högskola, Institutionen för reglerteknik, Lund, Sverige.

Ljung, L., Söderström, t. (1983). Theory and Practice of Recursive Identification. MIT Press, Cambridge, USA.

Åström, K.J., Wittenmark, B. (1983). Computer Controlled Systems. Prentice-Hall, N.J., USA

```
connecting system con
time t
yr[regul]=0.2*sign(sin(w1*t))
"u[motor]=1*(0.1*sign(sin(w1*t))+0.1*sign(sin(w2*t)))
w1:6
"w2:2.3
tau[lpfilt]=ta
tau[motpar]=ta
lambda[motpar]=la
eps[motpar]=ep
u[motor]=u[regul]
w[motor]=e1[noise1]
y[regul]=y[motor]
u[motpar]=u[motor]
y[motpar]=y[motor]
zu[motpar]=zu[lpfilt]
zy[motpar]=zy[lpfilt]
y[lpfilt]=y[motor]
u[lpfilt]=u[motor]
ap[regpar]=ap[motpar]
bp[regpar]=bp[motpar]
k[regul]=k[regpar]
ti[regul]=ti[regpar]
"k[regul]=3
"ti[regul]=0.2
h[regul]=h[lpfilt]
h[motpar]=h[regpar]
h[lpfilt]=h[motpar]
h[regpar]=0.01/3
la:0.9999
ta:0.1
ep:0.01
end
```

```
discrete system regul
input yr y k ti h
output u
state i
new ni
time t
tsamp ts
e=yr-y
v=k*e+i
u=if v<ulow then ulow else if v<uhigh then v else uhigh
ni=i+k*e*h/ti+u-v
ts=t+h
ulow:-100
uhigh:100
end
```

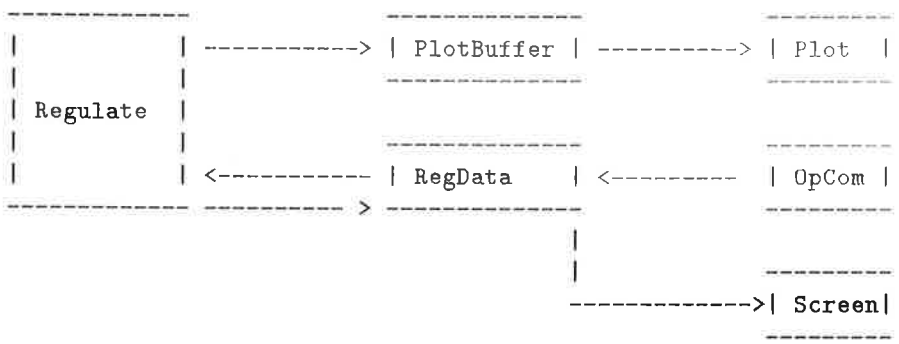
```
continuous system motor
input u w "w brus
output y
state x
der dx
time t
y=x+w
dx=a*x+b*u
a=if t<20 then -10 else -8
b=if t<90 then 2 else 4
END
```

```
discrete system lpfilt
input u y h tau
output zu zy
state intu inty oldy oldu
new intuny intyny oldyny olduny
time t
tsamp ts
zy=intyny
intyny=-a1*inty+b0*y+b1*oldy
"intuny=-a1*intu+b0*y+b1*oldu
a1=-exp(-h/tau)
b0=1-(1+a1)*tau/h
b1=(1+a1)*tau/h+a1
"b0=(1+a1)/2
"b1=b0
"intyny=exp(-h/tau)*(inty-(y+oldy)/2)+(y+oldy)/2
olduny=u
oldyny=y
zu=intuny
intuny=exp(-h/tau)*(intu-u)+u
ts=t+h
end
```

```
discrete system motpar " med eps-test | motpar0.t
input u y zu zy h tau lambda eps
output ap bp
state p11 p12 p21 p22 theta1 theta2
new p11ny p12ny p21ny p22ny theta1ny theta2ny
time t
tsamp ts
fi1=zy
fi2=zu
ap=(theta1-1)/tau
bp=theta2/tau
theta1ny=theta1+dtheta1
theta2ny=theta2+dtheta2
dtheta1=if abs(e)<eps then 0 else (p11ny*fi1+p12ny*fi2)*e
dtheta2=if abs(e)<eps then 0 else (p21ny*fi1+p22ny*fi2)*e
e=y-theta1*fi1-theta2*fi2
p11ny=if abs(e)<eps then p11 else (p11-q11/qq)/lambda
p12ny=if abs(e)<eps then p12 else (p12-q12/qq)/lambda
p21ny=p12ny
p22ny=if abs(e)<eps then p22 else (p22-q22/qq)/lambda
q11=(p11*fi1+p12*fi2)*(fi1*p11+fi2*p12)
q12=(p11*fi1+p12*fi2)*(fi1*p12+fi2*p22)
q22=(p12*fi1+p22*fi2)*(p12*fi1+p22*fi2)
qq=lambda+(fi1*p11+fi2*p12)*fi1+(fi1*p12+fi2*p22)*fi2
ts=t+h
end
```

```
discrete system regpar
input ap bp h
output k ti
time t
tsamp ts
ti=bp/(ak*ak+bk*bk)
k=if (bp<0.01 and bp)-0.01) then 100*(ap-2*ak) else -(2*ak-ap)/bp
ts=t+h
ak=polr
bk=poli
polr:-8
poli:4
end
```

```
MODULE main;

(*
        Authors :   Ola Nilsson and Lars Norrman
        Date    :   890505

  The program is divided into four processes and two monitors. Each one of
  them resides in its own separately compiled moduls.


  Monitors
  --------
  Regdata       This monitor contains all data and parameters relevant to
                the controller. They must be kept in a monitor since both
                the controller and the operator wants to have access to them.

  PlotBuffer    This monitor buffers data that are about to be plotted. The
                controller puts data here and data is fetched by the plotter.


  Processes
  ---------
  Regulate      Here we find the controller. It fetches its parameters from
                RegData, calculates a new controlsignal and sends data to
                PlotBuffer. The estimated parameters, K and Ti are sent to
                RegData.

  Plot          The plotter fetches data from PlotBuffer and plots it on the
                screen.

  Screen        This module initiates the graphics on the screen used by the
                operator and refreshes it at short intervals.


  Opcom         This module is responsible for the communication with the
                operator.
```



```
                                                                    *)


     (* MODULE MAIN *)

IMPORT RTMouse, PlotBuffer, RegData, Regulate, Plot, OpCom,ScreenEx;


BEGIN
  RTMouse.Init;

  (* initialize monitors *)
  PlotBuffer.Init;
  RegData.Init;

  (* start processes *)
  Regulate.Start;
  Plot.Start;
  ScreenEx.Start;
  OpCom.Start;

  OpCom.WaitTheEnd;
END main.
DEFINITION MODULE OpCom;

EXPORT QUALIFIED Start, WaitTheEnd;

(*
This module contains the communication with the operator. It's a process
which is started by calling "Start". The user will input data concerning
the behaviour of the regulator to this process. "OpCom" will then make this
data available to the regulator by calling "PutRegulatorData" in the module
"RegData". "WaitTheEnd" is a procedure that when it's called will block the
calling process/procedure until the operator tells "OpCom" to stop program
execution.
*)

PROCEDURE Start;

PROCEDURE WaitTheEnd;

END OpCom.
DEFINITION MODULE ScreenEx;

EXPORT QUALIFIED Start;

(*
This module initiates the graphics on the screen used by the
operator and refreshes it at short intervals.
*)

PROCEDURE Start;
```

```
END ScreenEx.

DEFINITION MODULE Plot;

EXPORT QUALIFIED Start;

(*
This module containes the plot process. It's started by calling "Start".
"Plot" fetches plotdata by calling "GetPlotData" in the module "PlotBuffer".
If the buffer is empty it will wait until there is data to fetch.
*)

PROCEDURE Start;

END Plot.
DEFINITION MODULE PlotBuffer;

EXPORT QUALIFIED PutInBuffer, GetFromBuffer, Init, PlotDataType;

(*
This module implements a buffer containing items of type "PlotDataType".
Use "PutInBuffer" to place an item in the buffer, and "GetFromBuffer" to
fetch an item. If "GetFromBuffer" is called when the buffer is empty it
will wait until there is data to fetch, i.e. until "PutInBuffer" has been
executed. If the buffer is full when "PutInBuffer" is called then
the oldest data in the buffer will be overwritten.
*)

TYPE PlotDataType = RECORD
                      ProcessOutput,Reference : REAL;
                      TimeInc : CARDINAL;
                    END;

(* Entry *) PROCEDURE PutInBuffer(Item : PlotDataType);

(* Entry *) PROCEDURE GetFromBuffer(VAR Item : PlotDataType);

PROCEDURE Init;

END PlotBuffer.

DEFINITION MODULE RegData;

FROM Graphics IMPORT
  handle,color,point,rectangle;

EXPORT QUALIFIED PutRegulatorData, GetRegulatorData, Init, RegulatorDataType,
                 RegulatorInDataType,PutRegulatorInData,GetRegulatorInData,
                 ScreenHandle,PlotAreaHandle,Color,
  aProcessPoint,bProcessPoint,SampPerPoint,LambdaPoint,TauPoint,OnOffPoint,
  StopPoint,aEstPoint,bEstPoint,KPoint,TiPoint,p11Point,p12Point,p21Point,
  p22Point,
  aProcessRect,bProcessRect,SampPerRect,LambdaRect,TauRect,OnOffRect,
```

```
  StopRect,aEstRect,bEstRect,KRect,TiRect,p11Rect,p12Rect,p21Rect,
  p22Rect,PlotAreaRect;


(*
This module keeps track of all data concerning the regulator. This data must
be accessable from both the regulator and the operator, and is therefor
places in a monitor. The data has the type "RegulatorDataType". You can get the
current values by using "GetRegulatorData", and place new data by using
"PutRegulatorData".
*)


TYPE RegulatorDataType = RECORD
                           aEst,bEst,K,Ti,p11,p12,p21,p22,p13,p23,p33  :REAL;
                         END;


    RegulatorInDataType = RECORD
                            RegulatorOn : BOOLEAN;
                            Reference : REAL;
                            aProcess,bProcess,Eps,Lambda,Tau : REAL;
                          END;


  (* global variabels for Screen and Opcom *)

VAR
  ScreenHandle,PlotAreaHandle : handle;
  Color : color;
  aProcessPoint,bProcessPoint,SampPerPoint,LambdaPoint,TauPoint,OnOffPoint,
  StopPoint,aEstPoint,bEstPoint,KPoint,TiPoint,p11Point,p12Point,p21Point,
  p22Point :       point;
  aProcessRect,bProcessRect,SampPerRect,LambdaRect,TauRect,OnOffRect,
  StopRect,aEstRect,bEstRect,KRect,TiRect,p11Rect,p12Rect,p21Rect,
  p22Rect,PlotAreaRect  :       rectangle;



(* Entry *) PROCEDURE PutRegulatorData(Item : RegulatorDataType);
(* Entry *) PROCEDURE GetRegulatorData(VAR Item : RegulatorDataType);

(* Entry *) PROCEDURE PutRegulatorInData(Item : RegulatorInDataType);
(* Entry *) PROCEDURE GetRegulatorInData(VAR Item : RegulatorInDataType);

PROCEDURE Init;

END RegData.
DEFINITION MODULE GraphUtility;

FROM Graphics IMPORT rectangle;

EXPORT QUALIFIED SetRectangleAbsolute, SetRectangleRelative;

PROCEDURE SetRectangleAbsolute(
  VAR NewRectangle : rectangle;
```

```
  xlo,ylo,xhi,yhi : REAL);

PROCEDURE SetRectangleRelative(
  VAR NewRectangle : rectangle;
  xlo,ylo,Width,Height : REAL);

END GraphUtility.

DEFINITION MODULE Regulate;

EXPORT QUALIFIED Start;

(*
This module contains the regulator. It's a process and its started by
calling "Start". The regulator gets data by calling "GetRegulatorData"
in the module "RegData", calculates a new controlsignal and sends data
to plot by calling "PutPlotData" in the module "PlotBuffer".
The estimated parameters are sent to "RegData" by calling
"PutRegulatorData".
*)

PROCEDURE Start;

END Regulate.
```

```
IMPLEMENTATION MODULE OpCom;

FROM Kernel IMPORT
  Semaphore, InitSem, Wait, Signal,
  WaitTime,
  SetPriority, CreateProcess;
FROM Graphics IMPORT
  point,color,buttonset,LeftButton,RightButton,
  WriteString,SetFillColor,
  FillRectangle,
  ShowCursor, HideCursor,
  EraseChar,GetMouse, SetMouseRectangle,WaitMouseRectangle;
FROM RegData IMPORT
  RegulatorDataType, GetRegulatorData ,
  RegulatorInDataType ,GetRegulatorInData, PutRegulatorInData,
  ScreenHandle,PlotAreaHandle,Color,
  aProcessPoint,bProcessPoint,SampPerPoint,LambdaPoint,TauPoint,StopPoint,

  OnOffPoint,
  aEstPoint,bEstPoint,KPoint,TiPoint,p11Point,p12Point,p21Point,p22Point,

  aProcessRect,bProcessRect,SampPerRect,LambdaRect,TauRect,StopRect,OnOffRect,

  aEstRect,bEstRect,KRect,TiRect,p11Rect,p12Rect,p21Rect,p22Rect,PlotAreaRect;
FROM ConvReal IMPORT
  RealToString;

CONST
  OpComPriority = 12;
  SampPer = 100.0;
TYPE
  CommandType = (ChangeReference,ChangeSampPer,OnOff,Terminate,
                 ChangeaProcess,ChangebProcess,ChangeLambda,ChangeTau);
VAR
  TheEnd : Semaphore;

PROCEDURE Start;
BEGIN
  InitSem(TheEnd,0);
  CreateProcess(Process,10000);
END Start;

PROCEDURE WaitTheEnd;
BEGIN
  Wait(TheEnd);
END WaitTheEnd;

(* Process Opcom *) PROCEDURE Process;

VAR
  MousePoint,RefPoint : point;
  button : buttonset;
  RegulatorData : RegulatorDataType;
  RegulatorInData : RegulatorInDataType;
  string1,string2 : ARRAY [0..40] OF CHAR;
  Command : CommandType;
```

```
  PROCEDURE ChangePar(VAR ParValue:REAL;Value:REAL);
  BEGIN
    IF LeftButton IN button THEN
      ParValue := ParValue-Value;
    ELSIF RightButton IN button THEN
      ParValue := ParValue+Value;
    END;
  END ChangePar;

BEGIN
  SetPriority(OpComPriority);
  SetMouseRectangle(ScreenHandle,aProcessRect,ORD(ChangeaProcess));
  SetMouseRectangle(ScreenHandle,bProcessRect,ORD(ChangebProcess));
  SetMouseRectangle(ScreenHandle,SampPerRect,ORD(ChangeSampPer));
  SetMouseRectangle(ScreenHandle,LambdaRect,ORD(ChangeLambda));
  SetMouseRectangle(ScreenHandle,TauRect,ORD(ChangeTau));
  SetMouseRectangle(ScreenHandle,PlotAreaRect,ORD(ChangeReference));
  SetMouseRectangle(ScreenHandle,OnOffRect,ORD(OnOff));
  SetMouseRectangle(ScreenHandle,StopRect,ORD(Terminate));

  LOOP
    ShowCursor;
    GetRegulatorInData(RegulatorInData);
    WITH RegulatorInData DO
      Command := VAL(CommandType,WaitMouseRectangle(ScreenHandle));
      GetMouse(ScreenHandle,MousePoint,button);
      HideCursor;
      CASE Command OF
        ChangeaProcess:
           ChangePar(aProcess,0.1); |
        ChangebProcess:
          ChangePar(bProcess,0.1); |
        ChangeSampPer:
          ChangePar(Eps,0.001); |
        ChangeLambda:
          ChangePar(Lambda,0.0001); |
        ChangeTau:
          ChangePar(Tau,20.0); |
        ChangeReference :
          (* start by getting the mouse position (= new reference) *)
          GetMouse(PlotAreaHandle,RefPoint,button);
          IF (RefPoint.v>=0.0) AND (RefPoint.v<=1.0) THEN
            (* new reference value accepted *)
            Reference := RefPoint.v;
          END; |
        OnOff :
          RegulatorOn := NOT RegulatorOn;

          (* show on screen if regulator is on or off *)
          (* red - off, green - on *)
          IF RegulatorOn THEN
            Color := green;
          ELSE
            Color := red;
          END;
```

```
          SetFillColor(ScreenHandle,Color);
          FillRectangle(ScreenHandle,OnOffRect);
          WriteString(ScreenHandle,OnOffPoint,"On/Off"); |
        Terminate :
          (* start by turning off the regulator, wait a certain time to be *)
          (* sure that the action has taken place, then stop the program *)
          RegulatorOn := FALSE;
          PutRegulatorInData(RegulatorInData);
          WaitTime(3*TRUNC(SampPer));
          Signal(TheEnd)
      END;
    PutRegulatorInData(RegulatorInData);
    END;
  END;
END Process;

END OpCom.
IMPLEMENTATION MODULE ScreenEx;

FROM Kernel IMPORT
  WaitTime,
  SetPriority, CreateProcess;
FROM Graphics IMPORT
  point, rectangle, color,
  VirtualScreen, SetViewPort, SetWindow,
  SetTextColor, SetLineColor, SetFillColor,
  WriteString,
  FillRectangle,
  EraseChar;
FROM GraphUtility IMPORT
  SetRectangleAbsolute, SetRectangleRelative;
FROM RegData IMPORT
  ScreenHandle,PlotAreaHandle,RegulatorDataType, GetRegulatorData ,
  RegulatorInDataType ,GetRegulatorInData, PutRegulatorInData,
  Color,
  aProcessPoint,bProcessPoint,SampPerPoint,LambdaPoint,TauPoint,StopPoint,

  OnOffPoint,
  aEstPoint,bEstPoint,KPoint,TiPoint,p11Point,p12Point,p21Point,p22Point,
  aProcessRect,bProcessRect,SampPerRect,LambdaRect,TauRect,
  aEstRect,bEstRect,KRect,TiRect,p11Rect,p12Rect,p21Rect,p22Rect,
  StopRect,OnOffRect,PlotAreaRect;

FROM ConvReal IMPORT
  RealToString;

CONST
  ScreenPriority = 11;
  SampPer = 100.0;


PROCEDURE Start;
BEGIN
  CreateProcess(Process,10000);
END Start;
```

```
(* Process Screen *) PROCEDURE Process;

VAR
  MousePoint : point;
  BackGroundRect,HelpRect : rectangle;
  RegulatorData : RegulatorDataType;
  RegulatorInData : RegulatorInDataType;
  string1 : ARRAY [0..40] OF CHAR;
  RefreshScreen : CARDINAL;

  PROCEDURE aInitScreenAndParameters;

  CONST
    XBase = 0.50;          (* constants used to be able to easily define *)
    YBase = 0.50;          (* rectangles on the screen *)
    RectWidth = 3.0;
    RectHeight = 1.5;
  BEGIN
    (* start with parameters concerning the complete screen              *)
    (* OpComHandle covers the whole screen since we want to be able to   *)
    (* point at the upper part (the plots) to change reference value.    *)
    VirtualScreen(ScreenHandle);
    SetRectangleAbsolute(HelpRect,0.0,0.0,1.5,1.0);
    SetViewPort(ScreenHandle,HelpRect);
    SetRectangleAbsolute(HelpRect,0.0,0.0,61.0,20.0);
    SetWindow(ScreenHandle,HelpRect);
    SetRectangleAbsolute(PlotAreaRect,0.0,10.0,61.0,20.0);
    SetRectangleAbsolute(BackGroundRect,0.0,0.0,61.0,10.0);
    SetFillColor(ScreenHandle,intensewhite);
    FillRectangle(ScreenHandle,BackGroundRect);
    SetFillColor(ScreenHandle,black);
    SetTextColor(ScreenHandle,intensewhite);

    (* define a handle that covers the plot part of the screen, and *)
    (* choose the scales so that the new reference value can be read *)
    (* directly as the mouse-location *)
    VirtualScreen(PlotAreaHandle);
    SetRectangleAbsolute(HelpRect,0.0,0.52,1.5,0.99);
    SetViewPort(PlotAreaHandle,HelpRect);
    SetRectangleAbsolute(HelpRect,0.0,-0.1,1.0,1.1);
    SetWindow(PlotAreaHandle,HelpRect);
  END aInitScreenAndParameters;

  PROCEDURE InitScreenRect;

    (* define rectangles and points that are to be used as locations for *)
    (* text and information on the screen *)
  CONST
    RectWidth = 9.0;
    RectHeight = 4.5;
    HalfRectHeight = 2.125;
  BEGIN
    SetFillColor(ScreenHandle,blue);
    SetRectangleRelative(aProcessRect,1.0,5.25,9.0,4.5);
    FillRectangle(ScreenHandle,aProcessRect);
    SetRectangleRelative(bProcessRect,11.0,5.25,9.0,4.5);
```

```
  FillRectangle(ScreenHandle,bProcessRect);
  SetRectangleRelative(SampPerRect,21.0,5.25,9.0,4.5);
  FillRectangle(ScreenHandle,SampPerRect);
  SetRectangleRelative(LambdaRect,31.0,5.25,9.0,4.5);
  FillRectangle(ScreenHandle,LambdaRect);
  SetRectangleRelative(TauRect,41.0,5.25,9.0,4.5);
  FillRectangle(ScreenHandle,TauRect);
  SetRectangleRelative(StopRect,51.0,7.625,9.0,2.125);
  FillRectangle(ScreenHandle,StopRect);
  SetRectangleRelative(OnOffRect,51.0,5.25,9.0,2.125);
  SetFillColor(ScreenHandle,Color);
  (* The OnOffRect is green when the regulator is on and red when it's off *)
  FillRectangle(ScreenHandle,OnOffRect);
  SetFillColor(ScreenHandle,magenta);
  SetRectangleRelative(aEstRect,1.0,0.25,9.0,4.5);
  FillRectangle(ScreenHandle,aEstRect);
  SetRectangleRelative(bEstRect,11.0,0.25,9.0,4.5);
  FillRectangle(ScreenHandle,bEstRect);
  SetRectangleRelative(KRect,21.0,0.25,9.0,4.5);
  FillRectangle(ScreenHandle,KRect);
  SetRectangleRelative(TiRect,31.0,0.25,9.0,4.5);
  FillRectangle(ScreenHandle,TiRect);
  SetRectangleRelative(p11Rect,41.0,2.625,9.0,2.125);
  FillRectangle(ScreenHandle,p11Rect);
  SetRectangleRelative(p12Rect,51.0,2.625,9.0,2.125);
  FillRectangle(ScreenHandle,p12Rect);
  SetRectangleRelative(p21Rect,41.0,0.25,9.0,2.125);
  FillRectangle(ScreenHandle,p21Rect);
  SetRectangleRelative(p22Rect,51.0,0.25,9.0,2.125);
  FillRectangle(ScreenHandle,p22Rect);
END InitScreenRect;

PROCEDURE DefPointToWriteString(xPoint,yPointHigh,yPointLow,pOffset,q:REAL);
BEGIN
  aProcessPoint.h :=  xPoint+1.0;
  aProcessPoint.v :=  yPointHigh;
  bProcessPoint.h :=  xPoint+11.0;
  bProcessPoint.v :=  yPointHigh;
  SampPerPoint.h :=  xPoint+21.0;
  SampPerPoint.v :=  yPointHigh;
  LambdaPoint.h :=  xPoint+31.0;
  LambdaPoint.v :=  yPointHigh;
  TauPoint.h :=  xPoint+41.0;
  TauPoint.v :=  yPointHigh;
  StopPoint.h :=  xPoint+51.0;
  StopPoint.v :=  yPointHigh;
  OnOffPoint.h := xPoint+51.0;
  OnOffPoint.v := yPointHigh-2.375+q;
  aEstPoint.h :=  xPoint+1.0;
  aEstPoint.v :=  yPointLow;
  bEstPoint.h :=  xPoint+11.0;
  bEstPoint.v :=  yPointLow;
  KPoint.h :=  xPoint+21.0;
  KPoint.v :=  yPointLow;
  TiPoint.h :=  xPoint+31.0;
  TiPoint.v :=  yPointLow;
```

```
    p11Point.h :=  xPoint+41.0;
    p11Point.v :=  yPointLow+pOffset;
    p12Point.h :=  xPoint+51.0;
    p12Point.v :=  yPointLow+pOffset;
    p21Point.h :=  xPoint+41.0;
    p21Point.v :=  yPointLow-2.375+pOffset;
    p22Point.h :=  xPoint+51.0;
    p22Point.v :=  yPointLow-2.375+pOffset;
  END DefPointToWriteString;


  PROCEDURE WriteNameInRect;
  BEGIN
    WriteString(ScreenHandle,aProcessPoint,"aProcess");
    WriteString(ScreenHandle,bProcessPoint,"bProcess");
    WriteString(ScreenHandle,SampPerPoint,"Eps");
    WriteString(ScreenHandle,LambdaPoint,"Lambda");
    WriteString(ScreenHandle,TauPoint,"Tau");
    WriteString(ScreenHandle,StopPoint,"Stop");
    WriteString(ScreenHandle,OnOffPoint,"On/Off");
    WriteString(ScreenHandle,aEstPoint,"aEst");
    WriteString(ScreenHandle,bEstPoint,"bEst");
    WriteString(ScreenHandle,KPoint,"K");
    WriteString(ScreenHandle,TiPoint,"Ti");
    WriteString(ScreenHandle,p11Point,"p11");
    WriteString(ScreenHandle,p12Point,"p12");
    WriteString(ScreenHandle,p21Point,"p21");
    WriteString(ScreenHandle,p22Point,"p22");
  END WriteNameInRect;


  PROCEDURE WritePar(ParValue:REAL;ParPoint:point);
  BEGIN
    RealToString(ParValue,string1,8);
    EraseChar(ScreenHandle,ParPoint,8);
    WriteString(ScreenHandle,ParPoint,string1);
  END WritePar;

  PROCEDURE InitOrRefreshScreen;
  BEGIN
    InitScreenRect;
    DefPointToWriteString(1.0,9.0,4.0,0.0,0.0);
    WriteNameInRect;
    DefPointToWriteString(1.5,7.0,2.0,0.85,2.0);
    RefreshScreen := 0;
  END InitOrRefreshScreen;

BEGIN
  SetPriority(ScreenPriority);
  aInitScreenAndParameters;
  InitOrRefreshScreen;
  SetPriority(14); (* lowest priority for screen after initialization *)
  LOOP
    RefreshScreen := RefreshScreen+1;
    IF RefreshScreen=100 THEN
      InitOrRefreshScreen;
```

```
      END;
      GetRegulatorData(RegulatorData);
      GetRegulatorInData(RegulatorInData);
      WITH RegulatorData DO WITH RegulatorInData DO
        SetFillColor(ScreenHandle,Color);
        WritePar(aProcess,aProcessPoint);
        WritePar(bProcess,bProcessPoint);
        WritePar(Eps,SampPerPoint);
        WritePar(Lambda,LambdaPoint);
        WritePar(Tau,TauPoint);
        WritePar(aEst,aEstPoint);
        WritePar(bEst,bEstPoint);
        WritePar(K,KPoint);
        WritePar(Ti,TiPoint);
        WritePar(p11,p11Point);
        WritePar(p12,p12Point);
        WritePar(p21,p21Point);
        WritePar(p22,p22Point);
      END; END;
    END;
  END Process;

  END ScreenEx.
```

```
  IMPLEMENTATION MODULE Plot;

  FROM Kernel IMPORT
    SetPriority, CreateProcess;
  FROM Graphics IMPORT
    handle, point, rectangle, color,
    VirtualScreen, SetViewPort, SetWindow,
    SetTextColor, SetLineColor, SetFillColor,
    WriteString,
    DrawRectangle, FillRectangle, PolyLine;
  FROM GraphUtility IMPORT
    SetRectangleAbsolute;
  FROM PlotBuffer IMPORT
    PlotDataType, GetFromBuffer;
  FROM ConvReal IMPORT
    RealToString;
  FROM Strings IMPORT
    Concat;

  CONST
    PlotPriority = 13;
    PlotLength = 30000.0; (* The plot on the screen will be 30 seconds long *)

  PROCEDURE Start;
  BEGIN
    CreateProcess(Process,10000);
  END Start;

  (* Process Plot *) PROCEDURE Process;

  VAR
    PlotData : PlotDataType;
    PlotHandle,ScaleHandle : handle;
```

```
ScaleViewRect,ScaleWindRect,PlotViewRect,PlotWindRect : rectangle;
ref,out : ARRAY [0..1] OF point;
TimeString : ARRAY [0..10] OF CHAR;
WritePoint : point;
x : REAL;

PROCEDURE InitScreenAndParameters;

VAR
  ScalePoints : ARRAY [0..1] OF point;
  i : CARDINAL;

BEGIN
  (* The screen is divided into two parts. The first one contains the  *)
  (* scale of the plot and is only drawn once. The second one contains *)
  (* the plot and is refreshed every time the plot reaches the right   *)
  (* end of the screen.                                                *)

  (* Start with scale part of screen *)
  VirtualScreen(ScaleHandle);
  SetRectangleAbsolute(ScaleViewRect,0.0,0.52,0.15,0.99);
  SetViewPort(ScaleHandle,ScaleViewRect);
  SetRectangleAbsolute(ScaleWindRect,0.0,-0.1,1.0,1.1);
  SetWindow(ScaleHandle,ScaleWindRect);
  SetLineColor(ScaleHandle,intensewhite);
  SetTextColor(ScaleHandle,intensewhite);

  (* Plot the scale *)
  FOR i := 0 TO 10 DO
    IF (i=0) OR (i=10) THEN
      ScalePoints[0].h := 0.2;
      ScalePoints[0].v := FLOAT(i)/10.0;
    ELSIF i=5 THEN
      ScalePoints[0].h := 0.6;
      ScalePoints[0].v := FLOAT(i)/10.0;
    ELSE
      ScalePoints[0].h := 0.8;
      ScalePoints[0].v := FLOAT(i)/10.0;
    END;
    ScalePoints[1].h := 1.0;
    ScalePoints[1].v := FLOAT(i)/10.0;
    PolyLine(ScaleHandle,ScalePoints,2);
  END;
  WritePoint.h := 0.2;
  WritePoint.v := 0.0;
  WriteString(ScaleHandle,WritePoint,"0.0");
  WritePoint.v := 1.0;
  WriteString(ScaleHandle,WritePoint,"1.0");

  (* Now it's time to deal with the plot part of the screen *)
  VirtualScreen(PlotHandle);
  SetRectangleAbsolute(PlotViewRect,0.15,0.52,1.5,0.99);
  SetViewPort(PlotHandle,PlotViewRect);
  SetRectangleAbsolute(PlotWindRect,0.0,-0.1,PlotLength,1.1);
  SetWindow(PlotHandle,PlotWindRect);
  SetFillColor(PlotHandle,lightgreen);
```

```
    SetTextColor(PlotHandle,white);

    (* Here comes the x-axis text on the plot *)
    WritePoint.h := PlotLength*0.90;
    WritePoint.v := -0.1;
    RealToString(PlotLength/1000.0,TimeString,5);
    Concat(TimeString," S",TimeString);
  END InitScreenAndParameters;

BEGIN
  SetPriority(PlotPriority);
  InitScreenAndParameters;
  x := PlotLength+1.0;          (* A value outside the screen to force refresh *
)
  GetFromBuffer(PlotData);
  ref[1].v := PlotData.Reference;
  out[1].v := PlotData.ProcessOutput;
  LOOP
    GetFromBuffer(PlotData);
    x := x+FLOAT(PlotData.TimeInc);
    IF x>PlotLength THEN
      (* The plot has reached the right end of the screen. Refresh the screen *
)
      (* and start from the left of it again. *)
      x := FLOAT(PlotData.TimeInc);
      ref[1].h := 0.0;
      out[1].h := 0.0;
      FillRectangle(PlotHandle,PlotWindRect);
      SetLineColor(PlotHandle,intensewhite);
      DrawRectangle(PlotHandle,PlotWindRect);
      WriteString(PlotHandle,WritePoint,TimeString);
    END;
    ref[0] := ref[1];
    out[0] := out[1];
    ref[1].h := x;
    ref[1].v := PlotData.Reference;
    out[1].h := x;
    out[1].v := PlotData.ProcessOutput;
    (* Plot reference *)
    SetLineColor(PlotHandle,red);
    PolyLine(PlotHandle,ref,2);
    (* Plot process output *)
    SetLineColor(PlotHandle,white);
    PolyLine(PlotHandle,out,2);
  END;
END Process;

END Plot.

IMPLEMENTATION MODULE PlotBuffer;

FROM Kernel IMPORT
  Semaphore, Event, Signal, Wait, Cause, Await, InitSem, InitEvent;

CONST
  BufferLength = 20;
```

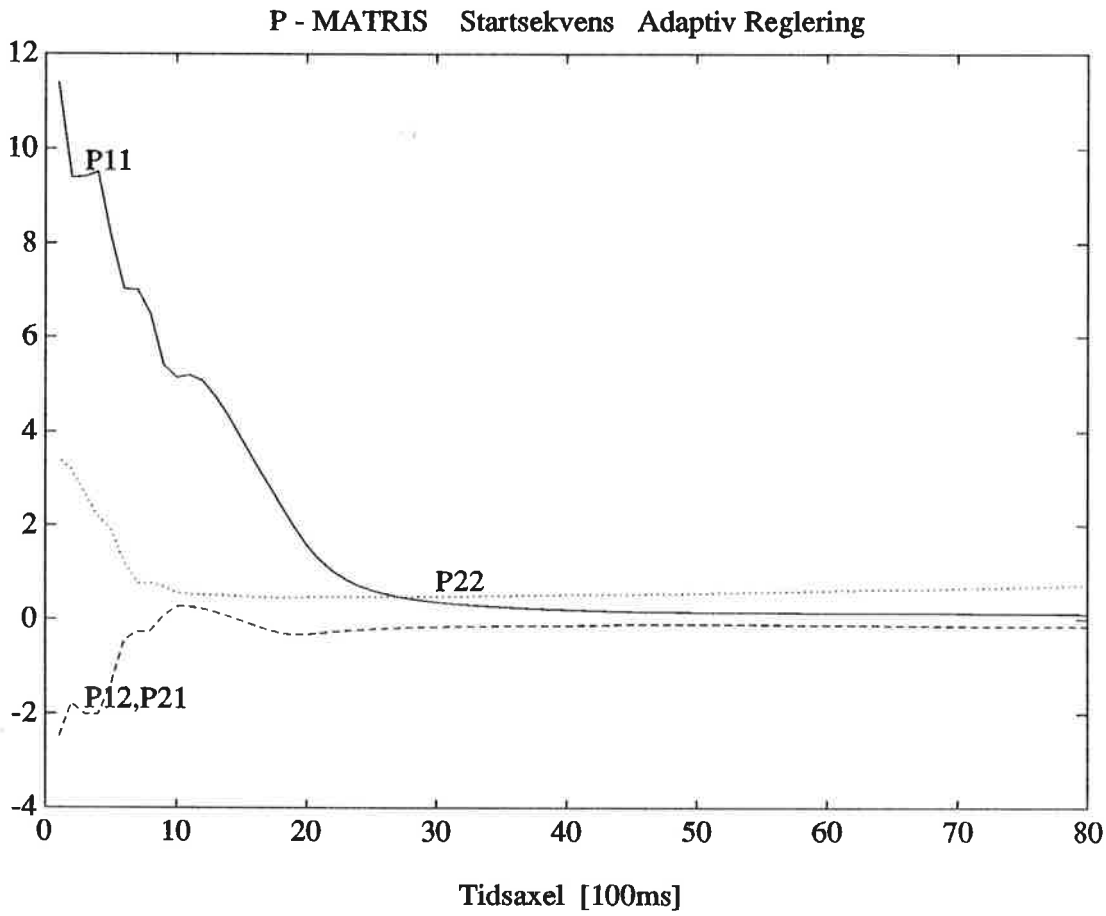
```
TYPE
  Index = [0..BufferLength-1];

VAR
  mutex : Semaphore;
  NonEmpty : Event;
  WritePos : Index;  (* The position last written *)
  ReadPos : Index;   (* The position last read *)
  DataVector : ARRAY Index OF PlotDataType;

(* Entry *) PROCEDURE PutInBuffer(Item : PlotDataType);

BEGIN
  Wait(mutex);
  WritePos := (WritePos+1) MOD BufferLength;
  DataVector[WritePos] := Item;
  IF WritePos=ReadPos THEN
    (* DATA LOST *)
    ReadPos := (ReadPos+1) MOD BufferLength;
  END;
  Cause(NonEmpty);
  Signal(mutex);
END PutInBuffer;

(* Entry *) PROCEDURE GetFromBuffer(VAR Item : PlotDataType);

BEGIN
  Wait(mutex);
  WHILE ReadPos=WritePos DO
    Await(NonEmpty);
  END;
  ReadPos := (ReadPos+1) MOD BufferLength;
  Item := DataVector[ReadPos];
  Signal(mutex);
END GetFromBuffer;

PROCEDURE Init;
BEGIN
  InitSem(mutex,1);
  InitEvent(NonEmpty,mutex);
  WritePos := 0;
  ReadPos := 0;
END Init;

END PlotBuffer.



IMPLEMENTATION MODULE RegData;

FROM Kernel IMPORT
  Semaphore, Signal, Wait, InitSem;

FROM Graphics IMPORT
  color;
```

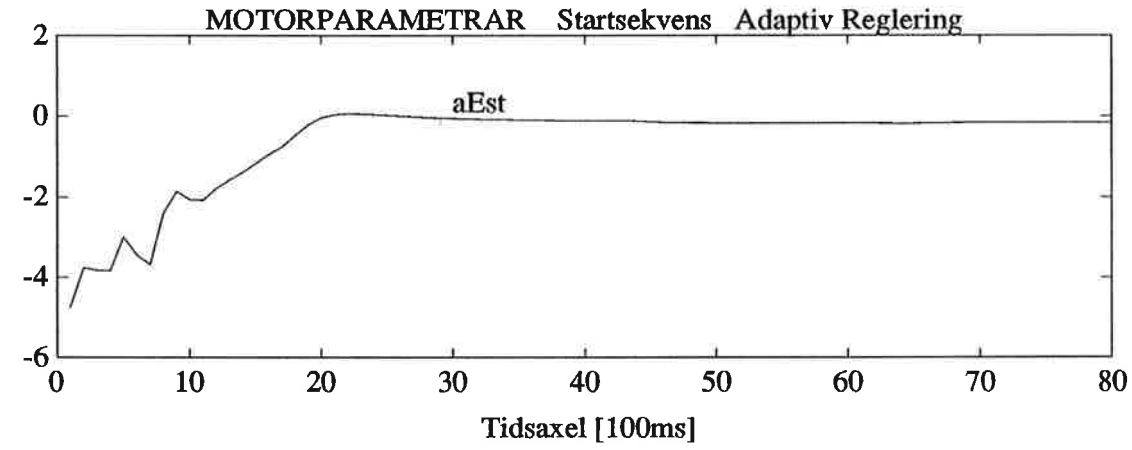
```
VAR
  mutex,mutex2 : Semaphore;
  RegulatorData : RegulatorDataType;
  RegulatorInData : RegulatorInDataType;

(* Entry *) PROCEDURE PutRegulatorData(Item : RegulatorDataType);
BEGIN
  Wait(mutex);
  RegulatorData := Item;
  Signal(mutex);
END PutRegulatorData;

(* Entry *) PROCEDURE GetRegulatorData(VAR Item : RegulatorDataType);
BEGIN
  Wait(mutex);
  Item := RegulatorData;
  Signal(mutex);
END GetRegulatorData;

(* Entry *) PROCEDURE PutRegulatorInData(Item : RegulatorInDataType);
BEGIN
  Wait(mutex2);
  RegulatorInData := Item;
  Signal(mutex2);
END PutRegulatorInData;

(* Entry *) PROCEDURE GetRegulatorInData(VAR Item : RegulatorInDataType);
BEGIN
  Wait(mutex2);
  Item := RegulatorInData;
  Signal(mutex2);
END GetRegulatorInData;

PROCEDURE Init;
BEGIN
  InitSem(mutex,1);
  InitSem(mutex2,1);
  WITH RegulatorData DO WITH RegulatorInData DO
    RegulatorOn := FALSE;
    Color := red;
    Reference := 0.5;
    aProcess  := -2.0;
    bProcess  := 0.5;
    Eps       := 0.001;
    Lambda    := 0.99;
    Tau       := 200.0;
    aEst      := -0.12;
    bEst      := 1.35;
    K         := 3.0;
    Ti        := 0.50;
    p11       := 12.00;
    p12       := 0.00;
    p21       := 0.00;
    p22       := 12.00;
  END;END;
END Init;
```

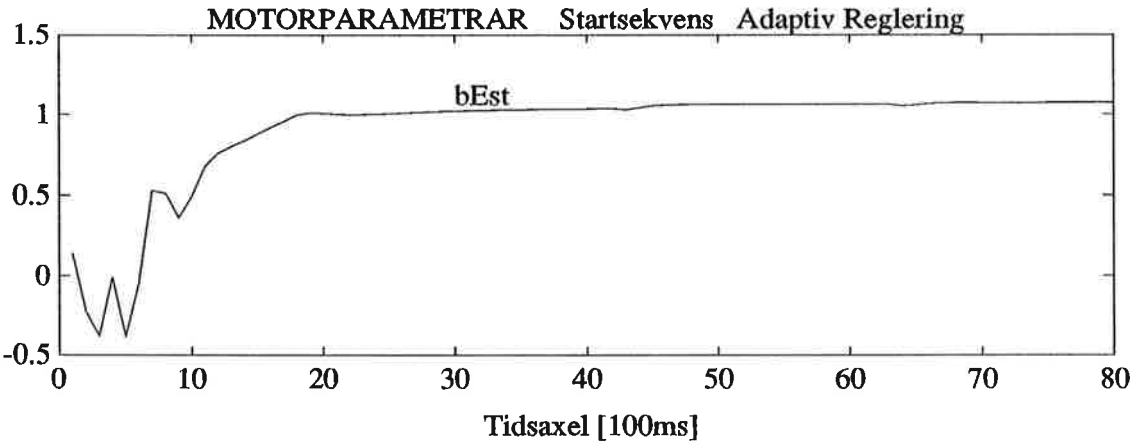
```
END RegData.

IMPLEMENTATION MODULE GraphUtility;

FROM Graphics IMPORT rectangle;

PROCEDURE SetRectangleAbsolute(
  VAR NewRectangle : rectangle;
  xlo,ylo,xhi,yhi : REAL);
BEGIN
  NewRectangle.xlo := xlo;
  NewRectangle.ylo := ylo;
  NewRectangle.xhi := xhi;
  NewRectangle.yhi := yhi;
END SetRectangleAbsolute;

PROCEDURE SetRectangleRelative(
  VAR NewRectangle : rectangle;
  xlo,ylo,Width,Height : REAL);
BEGIN
  NewRectangle.xlo := xlo;
  NewRectangle.ylo := ylo;
  NewRectangle.xhi := xlo+Width;
  NewRectangle.yhi := ylo+Height;
END SetRectangleRelative;

END GraphUtility.

IMPLEMENTATION MODULE Regulate;

FROM AnalogIO IMPORT
  ADIn, DAOut;
FROM Kernel IMPORT
  Time, GetTime, IncTime, WaitUntil,
  SetPriority, CreateProcess;
FROM MathLib IMPORT
  exp,sin;
FROM PlotBuffer IMPORT
  PlotDataType, PutInBuffer;
FROM RegData IMPORT
  RegulatorDataType, PutRegulatorData,GetRegulatorData,
  RegulatorInDataType, PutRegulatorInData,GetRegulatorInData;
CONST
  RegulatorPriority = 10;
  ProcessOutputChannel = 1;
  ControlSignalChannel = 1;
  ControlSignalLinPoint = 0.0;  (* linearization point for control signal *)
  HighLimit = 1.0;
  LowLimit = -1.0;

PROCEDURE Start;
BEGIN
  CreateProcess(Process,10000);
END Start;
```

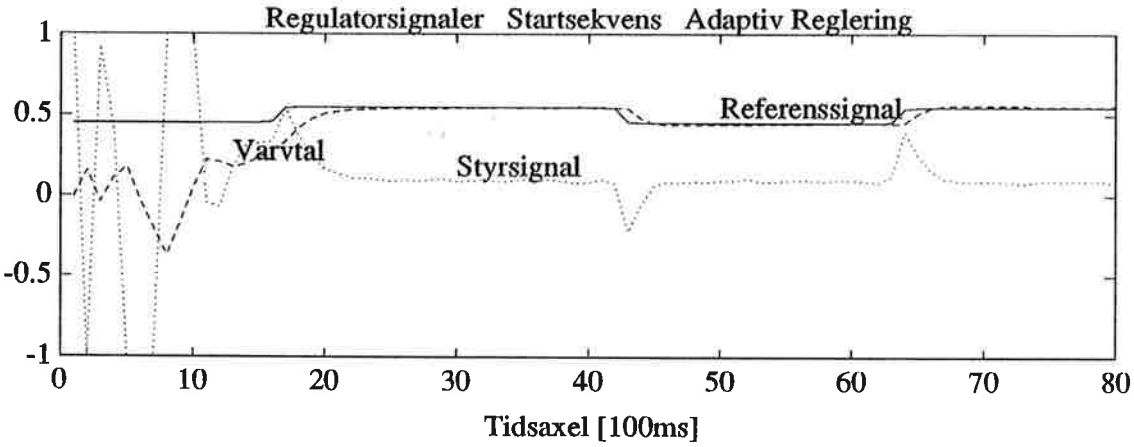
```
(* Process Regulate *) PROCEDURE Process;

CONST
SampPer = 100.0;


VAR
  PlotData : PlotDataType;
  ControlSignal,Error,oldy,theta1,theta2 : REAL;
  RegulatorData : RegulatorDataType;
  RegulatorInData : RegulatorInDataType;
  NextSamp : Time;
  I ,t : REAL;
  a1,b0,b1,fi1,fi2,e,q11,q12,q21,q22,qq : REAL;

  PROCEDURE SystemIdentification(u,y:REAL);


  BEGIN
    (* System Identification *)
    WITH RegulatorData DO WITH RegulatorInData DO
        (* the unit for SampPer and Tau is milliseconds *)
      a1 := -exp(-SampPer/Tau);
      b0 := 1.0-(1.0+a1)*Tau/SampPer;
      b1 := (1.0+a1)*Tau/SampPer+a1;
      fi1 := -a1*fi1+b0*y+b1*oldy;
      oldy := y;
      fi2 := -a1*(fi2-u)+u;
      e := y-theta1*fi1-theta2*fi2;
      IF ABS(e)>Eps THEN
        q11 := (p11*fi1+p12*fi2)*(p11*fi1+p12*fi2);
        q12 := (p11*fi1+p12*fi2)*(fi1*p12+fi2*p22);
        q22 := (p12*fi1+p22*fi2)*(p12*fi1+p22*fi2);
        qq  := p11*fi1*fi1+2.0*fi1*fi2*p12+fi2*fi2*p22+Lambda;
        p11 := (p11-q11/qq)/Lambda;
        p12 := (p12-q12/qq)/Lambda;
        p22 := (p22-q22/qq)/Lambda;
        p21 := p12;
        theta1 := theta1+e*(p11*fi1+p12*fi2);
        theta2 := theta2+e*(p12*fi1+p22*fi2);
        aEst := (theta1-1.0)/(0.001*Tau);
        bEst := theta2/(0.001*Tau);
      END;
    END; END;
  END SystemIdentification;
BEGIN
  SetPriority(RegulatorPriority);
  ControlSignal := 0.0;
  I := 0.5;
  oldy := 0.5; theta1 := 0.1; theta2 := 0.2; fi1 := 0.1; fi2 := 0.2;
  t := 0.0;
  GetTime(NextSamp);
  LOOP
    GetRegulatorData(RegulatorData);
    GetRegulatorInData(RegulatorInData);
    WITH RegulatorData DO WITH RegulatorInData DO WITH PlotData DO
```

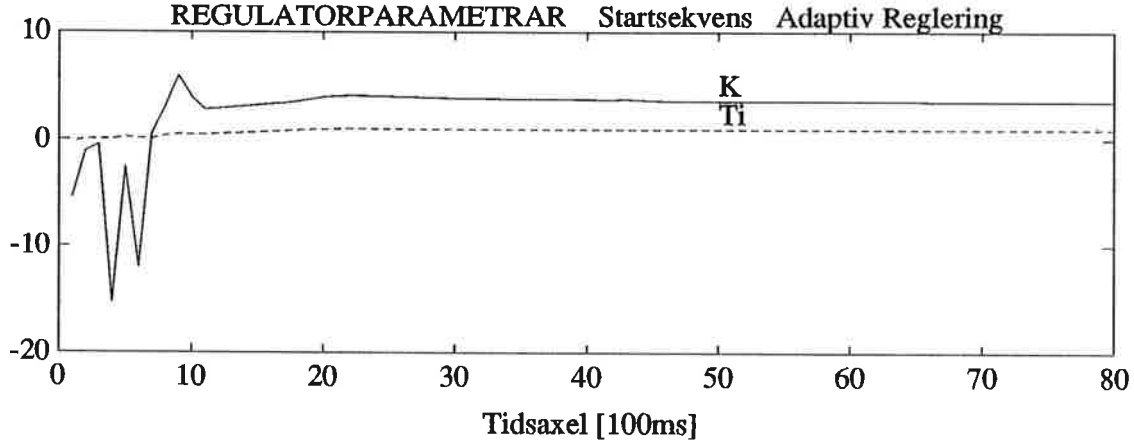
```
        IncTime(NextSamp,TRUNC(SampPer));
        t := t+1.0;
        PlotData.Reference := 0.5+0.05*sin(0.12*t)/(ABS(sin(0.12*t))+0.01);
        ProcessOutput := ADIn(ProcessOutputChannel);
        IF RegulatorOn THEN
          SystemIdentification(ControlSignal,ProcessOutput);
          Error := Reference-ProcessOutput;
          ControlSignal := K*Error+I;
          (* Limit ControlSignal *)
          IF ControlSignal>HighLimit THEN
            ControlSignal := HighLimit;
          ELSIF ControlSignal<LowLimit THEN
            ControlSignal := LowLimit;
          END;
          DAOut(ControlSignalChannel,ControlSignal);
            (* the unit for Ti is seconds *)
          I := ControlSignal+K*Error*(0.001*SampPer/Ti-1.0);
          K := -(2.0*aProcess-aEst)/bEst;
            IF K >= ABS (5.0) THEN K := 5.0 END;
          Ti := -(2.0*aProcess-aEst)/(aProcess*aProcess);
          PutRegulatorData(RegulatorData);
        ELSE
          (* When the regulator is off the controlsignal is set to zero *)
          ControlSignal := 0.0;
          DAOut(ControlSignalChannel,ControlSignal)
        END;
        TimeInc := TRUNC(SampPer);
        PutInBuffer(PlotData);
        END;END;END;
      WaitUntil(NextSamp);
    END;
  END Process;

  END Regulate.
```

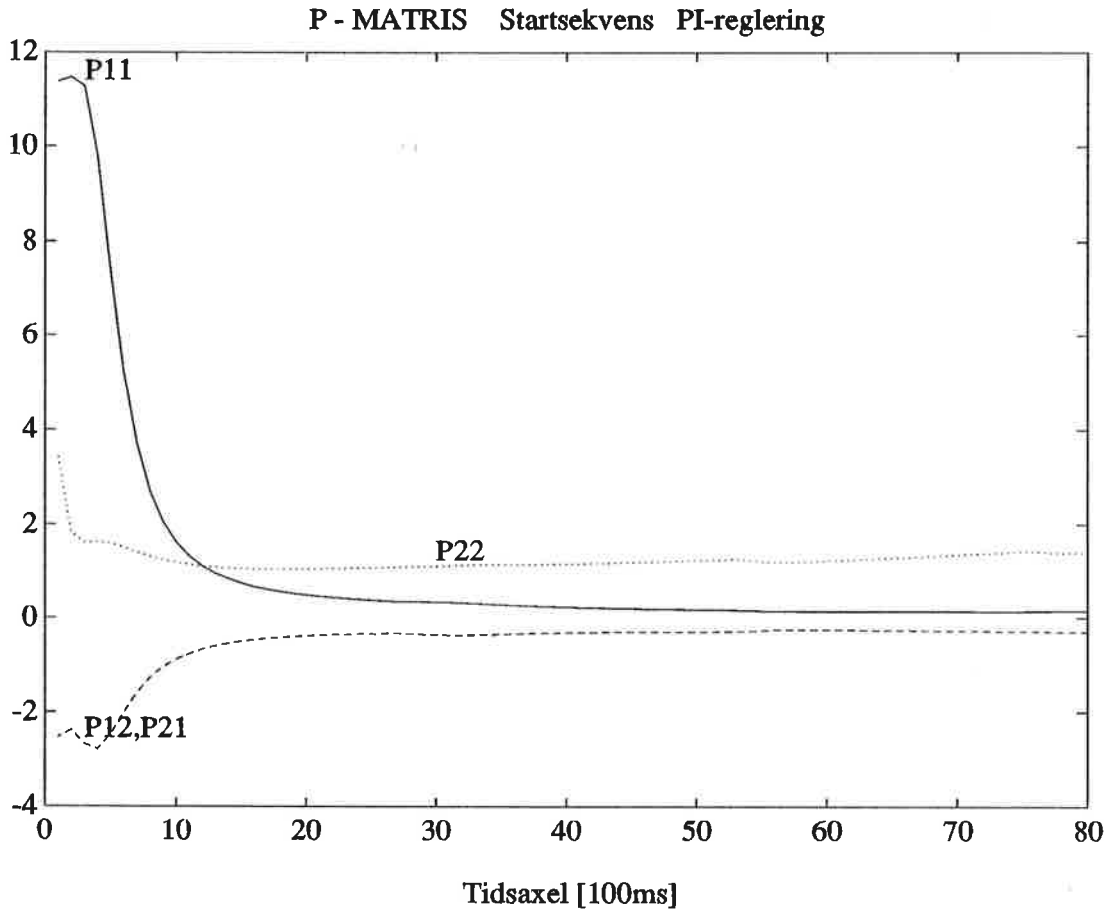

P - MATRIS Startsekvens Adaptiv Reglering
12
10
8
6
4
2
0
-2
-4
0
10
20
30
40
50
60
70
80
P11
P22
P12,P21
Tidsaxel [100ms]

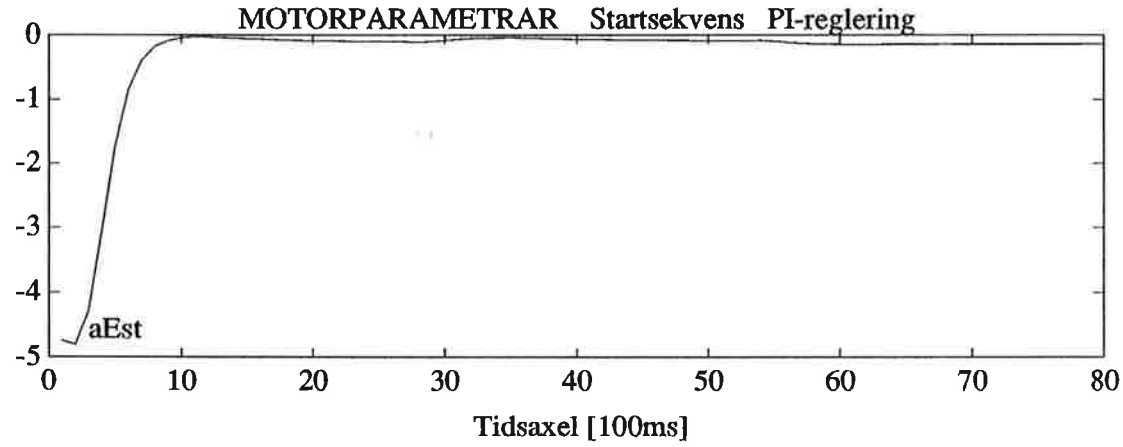

MOTORPARAMETRAR Startsekvens Adaptiv Reglering
aEst
2
0
-2
-4
-6
0
10
20
30
40
50
60
70
80
Tidsaxel [100ms]

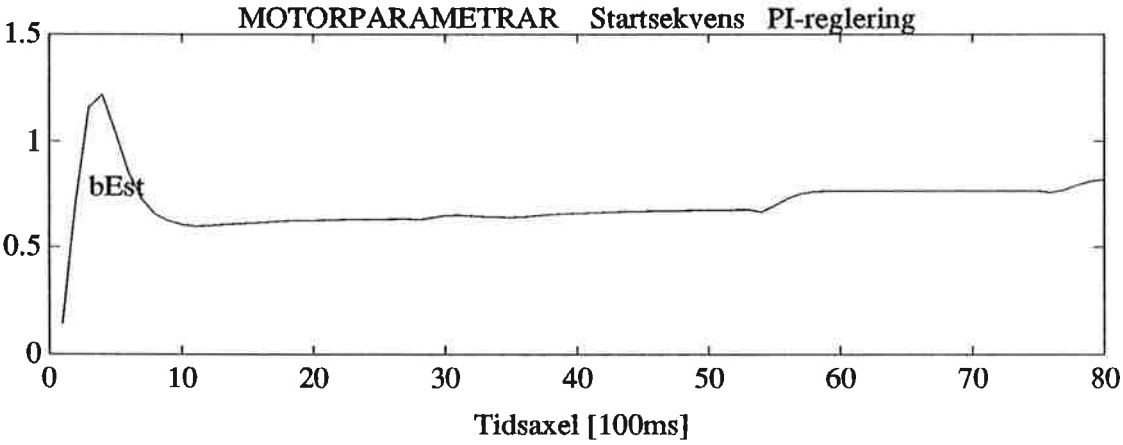

MOTORPARAMETRAR Startsekvens Adaptiv Reglering
bEst
1.5
1
0.5
0
-0.5
0
10
20
30
40
50
60
70
80
Tidsaxel [100ms]

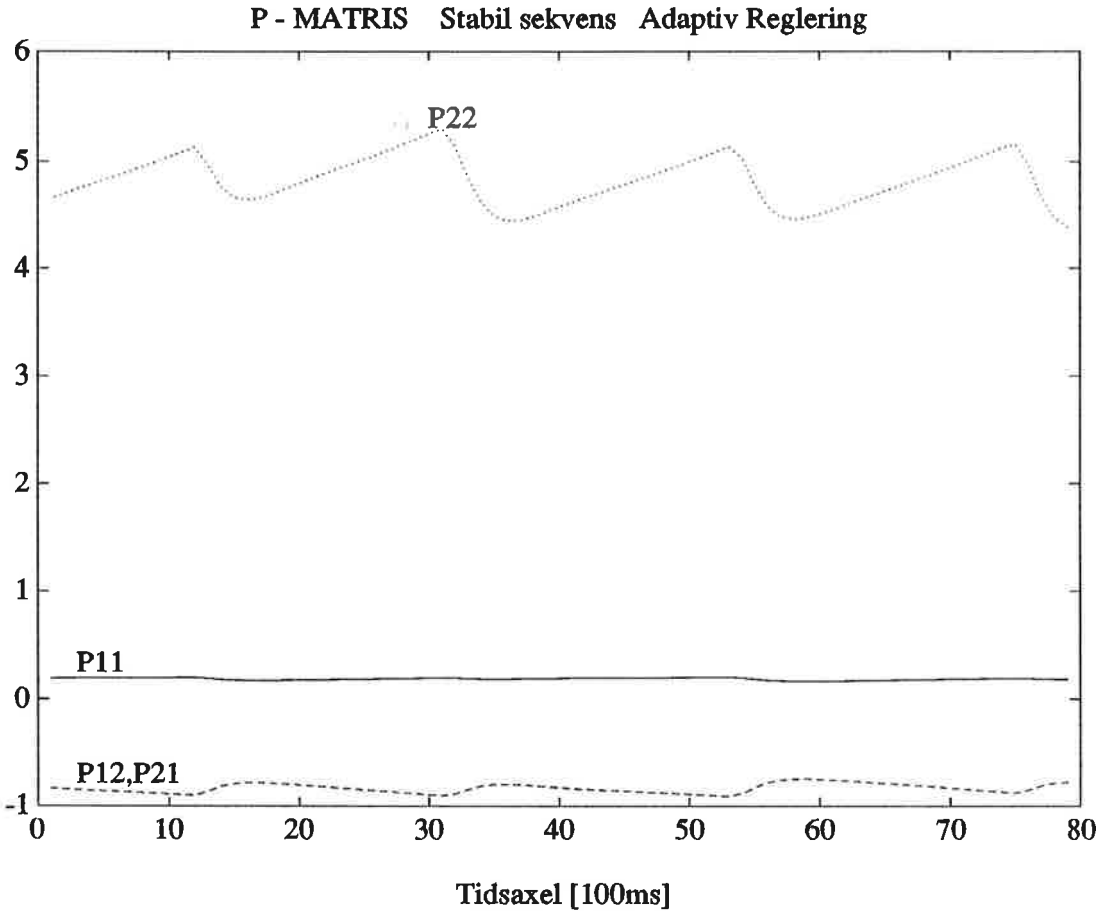

Regulatorsignaler Startsekvens Adaptiv Reglering
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
Referenssignal
Varvtal
Styrsignal
Tidsaxel [100ms]

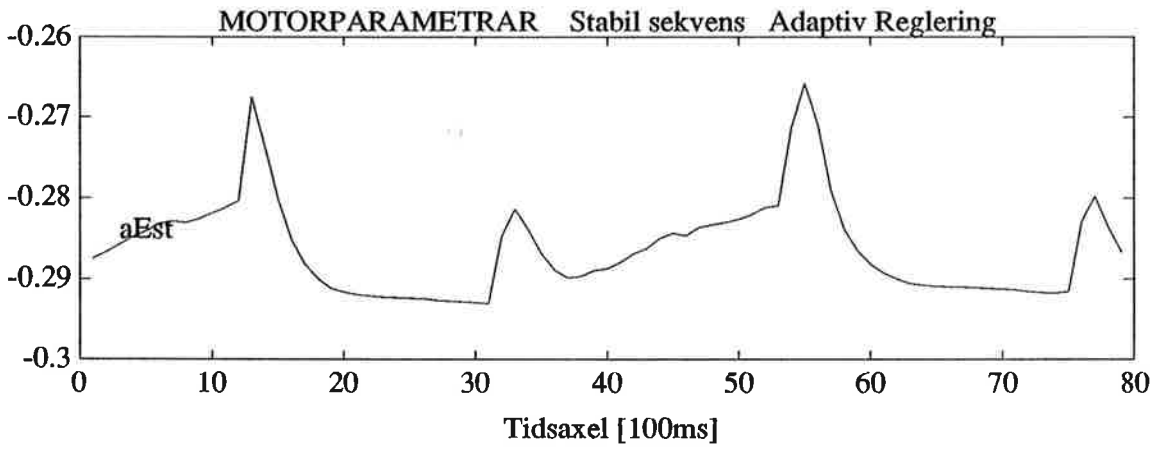

REGULATORPARAMETRAR Startsekvens Adaptiv Reglering
10
0
-10
-20
0
10
20
30
40
50
60
70
80
K
Ti
Tidsaxel [100ms]

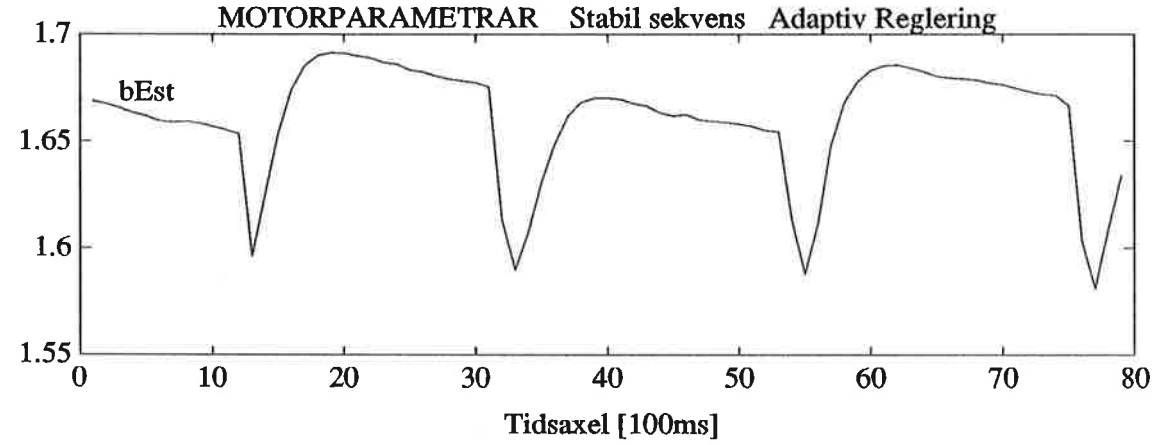

P - MATRIS Startsekvens PI-reglering
P11
P22
P12,P21
12
10
8
6
4
2
0
-2
-4
0
10
20
30
40
50
60
70
80
Tidsaxel [100ms]

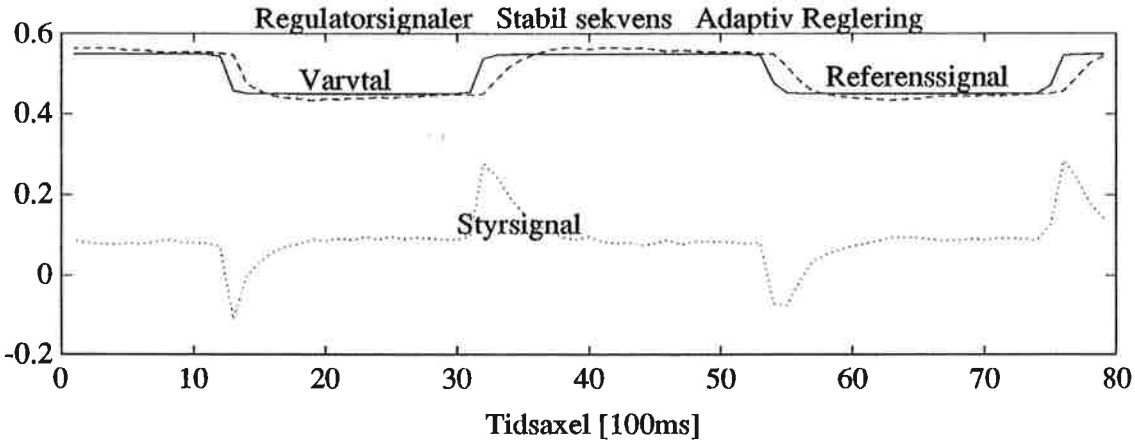

MOTORPARAMETRAR Startsekvens PI-reglering
0
-1
-2
-3
-4
-5
aEst
0
10
20
30
40
50
60
70
80
Tidsaxel [100ms]

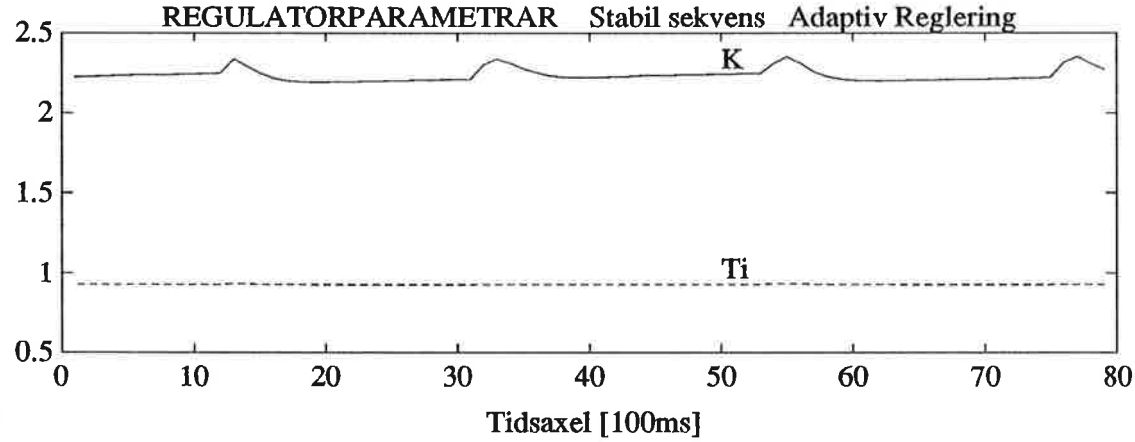

MOTORPARAMETRAR Startsekvens PI-reglering
1.5
1
0.5
0
bEst
0
10
20
30
40
50
60
70
80
Tidsaxel [100ms]

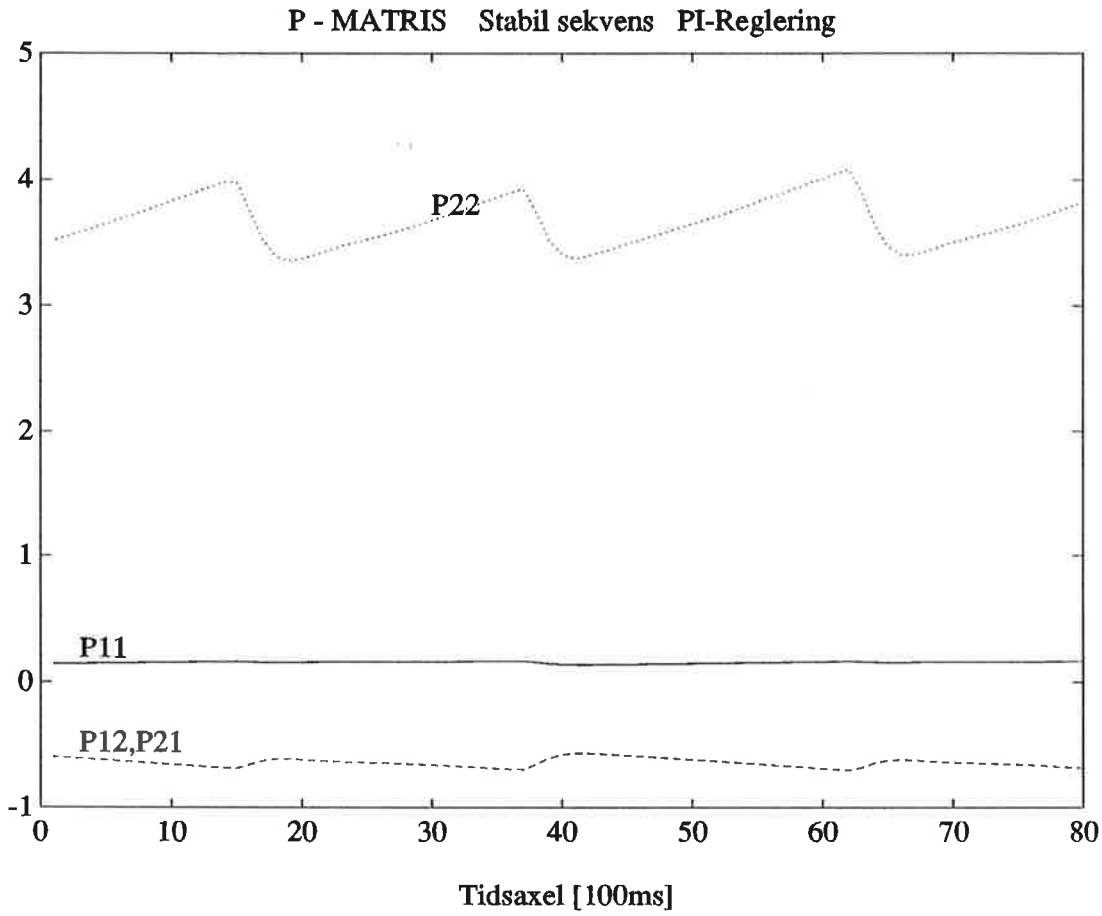

P - MATRIS Stabil sekvens Adaptiv Reglering
P22
P11
P12,P21
6
5
4
3
2
1
0
-1
0
10
20
30
40
50
60
70
80
Tidsaxel [100ms]

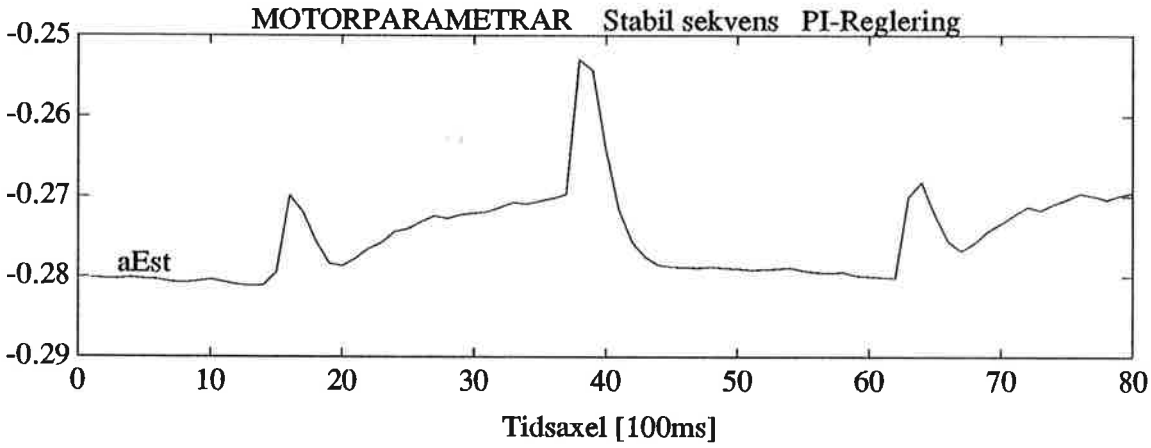

MOTORPARAMETRAR Stabil sekvens Adaptiv Reglering
-0.26
-0.27
-0.28
-0.29
-0.3
aEst
0
10
20
30
40
50
60
70
80
Tidsaxel [100ms]

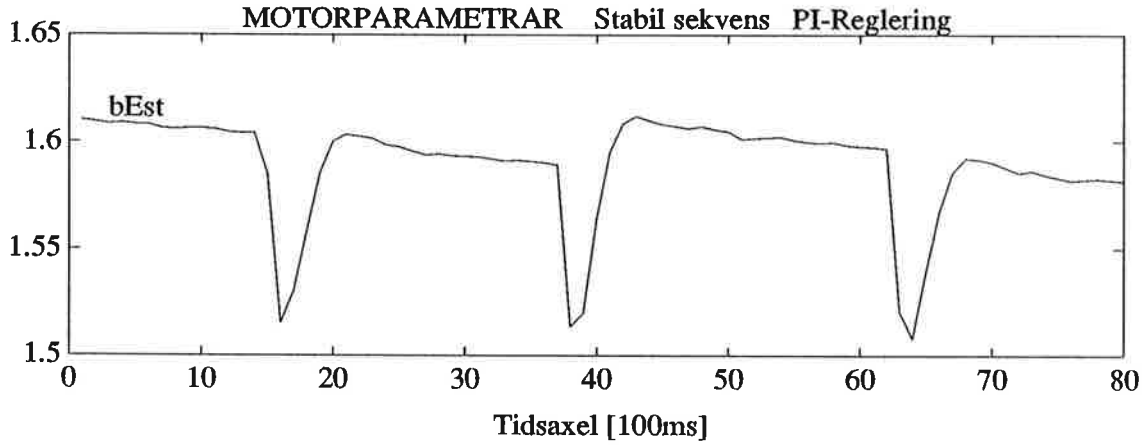

MOTORPARAMETRAR Stabil sekvens Adaptiv Reglering
1.7
1.65
1.6
1.55
bEst
0
10
20
30
40
50
60
70
80
Tidsaxel [100ms]

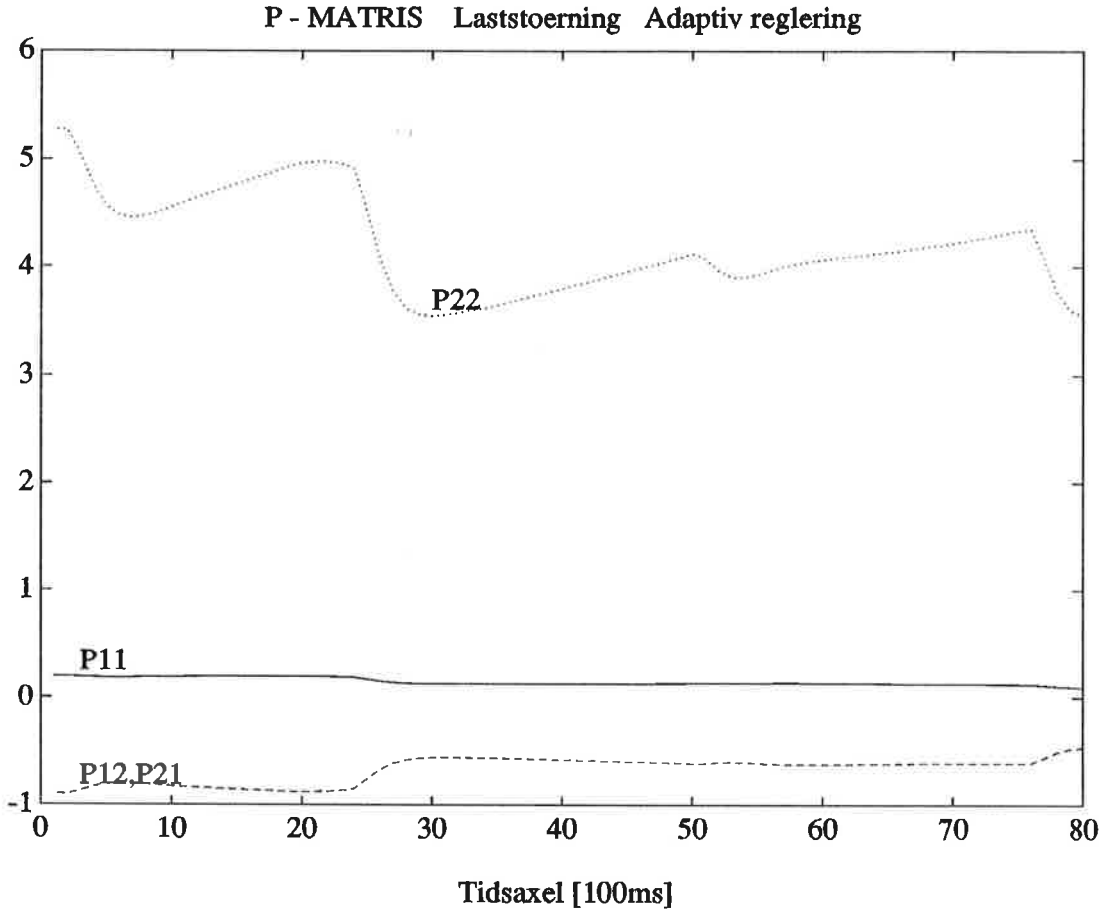

Regulatorsignaler Stabil sekvens Adaptiv Reglering
Varvtal
Referenssignal
Styrsignal
0.6
0.4
0.2
0
-0.2
0
10
20
30
40
50
60
70
80
Tidsaxel [100ms]

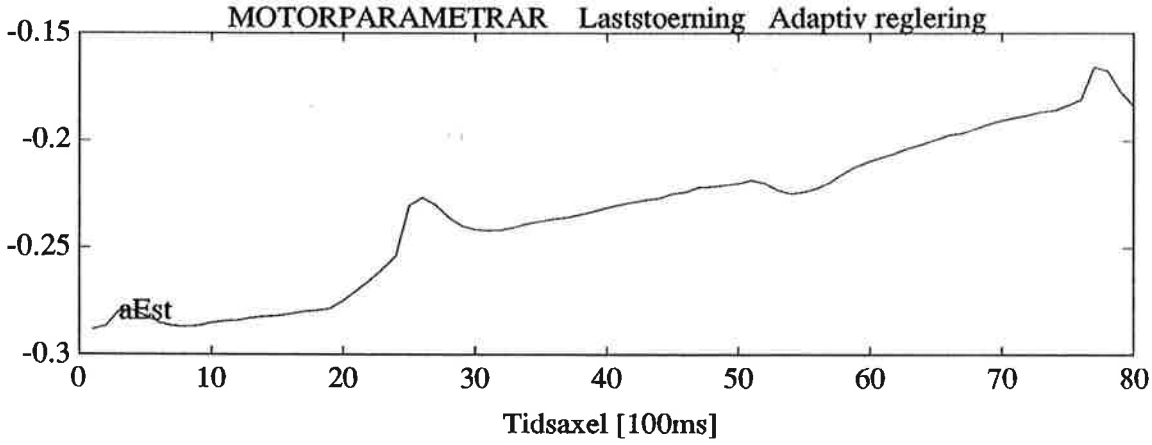

REGULATORPARAMETRAR Stabil sekvens Adaptiv Reglering
K
Ti
2.5
2
1.5
1
0.5
0
10
20
30
40
50
60
70
80
Tidsaxel [100ms]

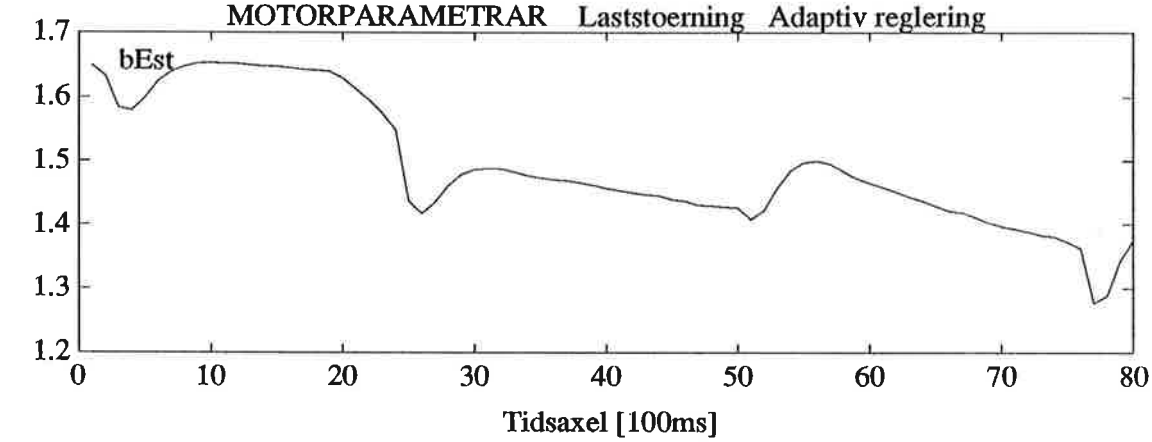

P - MATRIS Stabil sekvens PI-Reglering
P22
P11
P12,P21
5
4
3
2
1
0
-1
0
10
20
30
40
50
60
70
80
Tidsaxel [100ms]

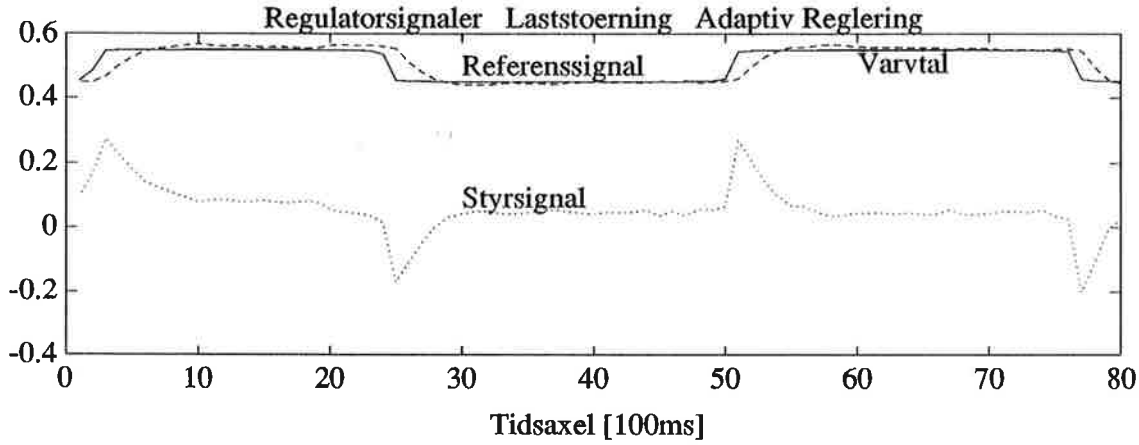

MOTORPARAMETRAR Stabil sekvens PI-Reglering
-0.25
-0.26
-0.27
-0.28
-0.29
aEst
0
10
20
30
40
50
60
70
80
Tidsaxel [100ms]

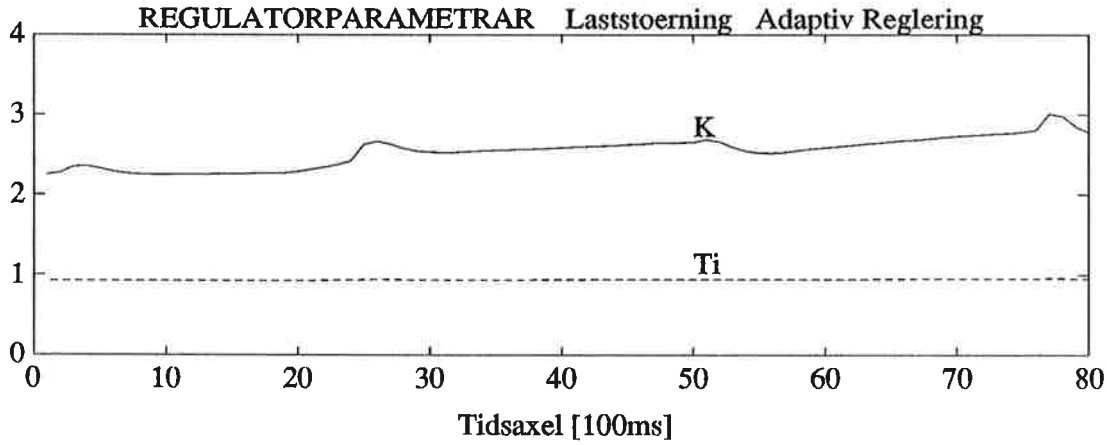

MOTORPARAMETRAR Stabil sekvens PI-Reglering
1.65
1.6
1.55
1.5
bEst
0
10
20
30
40
50
60
70
80
Tidsaxel [100ms]

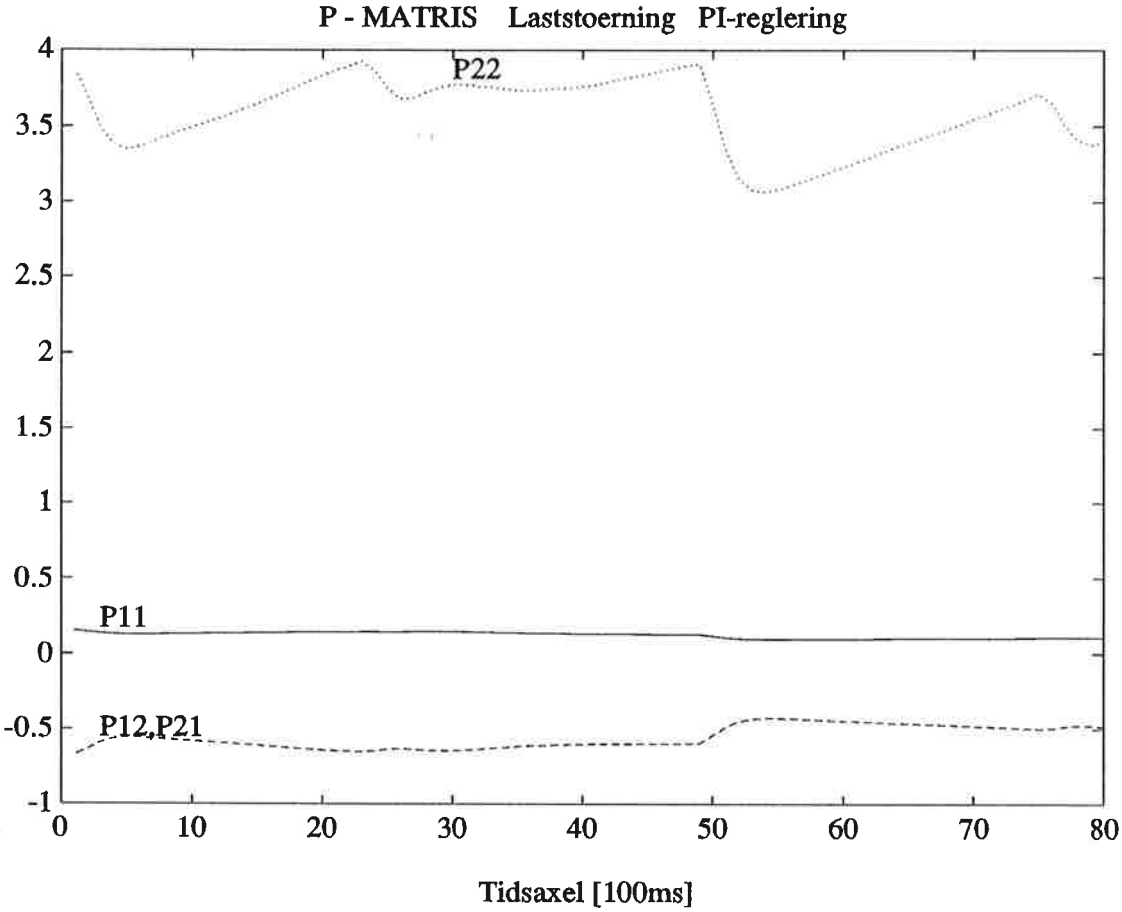

P - MATRIS Laststoerning Adaptiv reglering
6
5
4
3
2
1
0
-1
P22
P11
P12,P21
0
10
20
30
40
50
60
70
80
Tidsaxel [100ms]

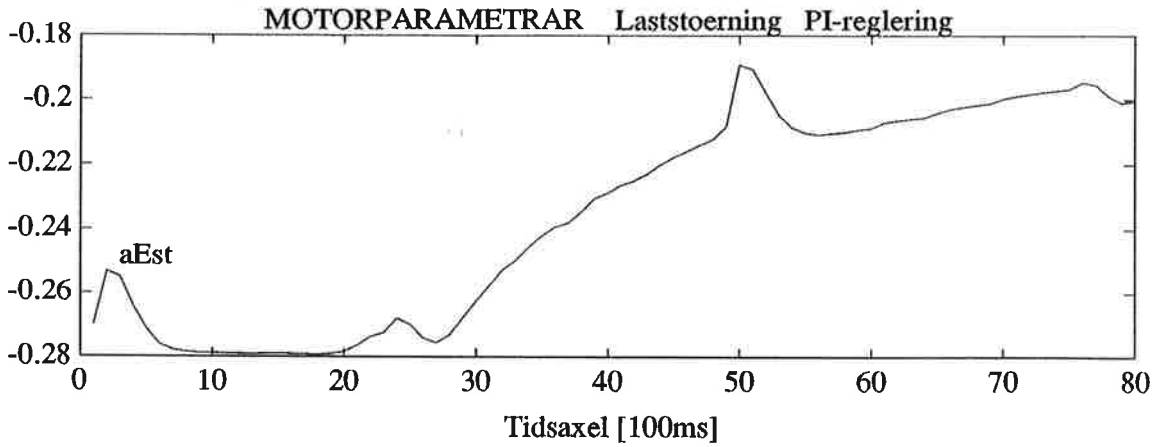

MOTORPARAMETRAR Laststoerning Adaptiv reglering
-0.15
-0.2
-0.25
-0.3
aEst
0
10
20
30
40
50
60
70
80
Tidsaxel [100ms]

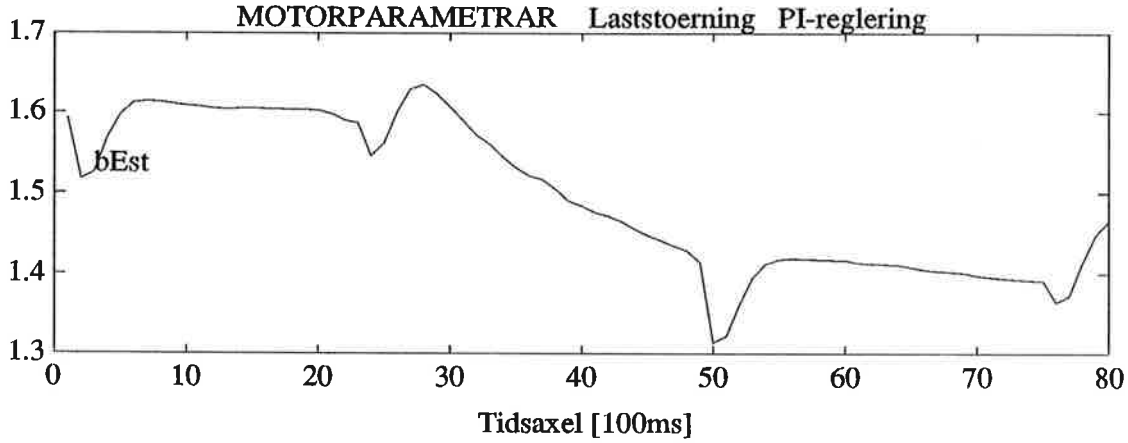

MOTORPARAMETRAR Laststoerning Adaptiv reglering
1.7
1.6
1.5
1.4
1.3
1.2
bEst
0
10
20
30
40
50
60
70
80
Tidsaxel [100ms]


Regulatorsignaler Laststoerning Adaptiv Reglering
Referenssignal
Varvtal
Styrsignal
0.6
0.4
0.2
0
-0.2
-0.4
0
10
20
30
40
50
60
70
80
Tidsaxel [100ms]


REGULATORPARAMETRAR Laststoerning Adaptiv Reglering
K
Ti
4
3
2
1
0
0
10
20
30
40
50
60
70
80
Tidsaxel [100ms]


P - MATRIS Laststoerning PI-reglering
P22
P11
P12,P21
4
3.5
3
2.5
2
1.5
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
Tidsaxel [100ms]


MOTORPARAMETRAR Laststoerning PI-reglering
-0.18
-0.2
-0.22
-0.24
-0.26
-0.28
aEst
0
10
20
30
40
50
60
70
80
Tidsaxel [100ms]


MOTORPARAMETRAR Laststoerning PI-reglering
1.7
1.6
1.5
1.4
1.3
bEst
0
10
20
30
40
50
60
70
80
Tidsaxel [100ms]