CODEN: LUTFD2/(TFRT-5264)/1-088/(1981)

SJÄLVINSTÄLLANDE REGULATORER PÅ MIKRODATOR I HÖGNIVÅSPRÅK

LARS BÅÅTH
PER-OLOF MALMQVIST

INSTITUTIONEN FÖR REGLERTEKNIK
LUNDS TEKNISKA HÖGSKOLA

NOVEMBER 1981

LUND INSTITUTE OF TECHNOLOGY
DEPARTMENT OF AUTOMATIC CONTROL
Box 725
S 220 07 Lund 7 Sweden

Document name
Master thesis
Date of issue
November 1981
Document number
CODEN: LUTFD/(TFRT-5264)/1-088/(1981)

Author(s)
Lars Bååth
Per-Olof Malmqvist

Supervisor
Carl Fredrik Mannerfelt
Sponsoring organization

Title and subtitle
Självinställande regulatorer på mikrodator i högnivåspråk.
(Self-tuning regulators on a micro-computer in high-level language.)

Abstract
This master thesis is concerned with the problem of implementation of a self-tuning regulator on a micro-computer. The computer programs for the self-tuner are organized as a control-package, which may contain ten different control loops. The package further gives the operator the possibility to interactively change the values of design parameters, etc. All the programming code was written in the high-level language Pascal.
The self-tuning regulator is of a modified minimum-variance type, which proved to behave satisfactory in the practical tests we performed. The regulator was also applied to an air heat-exchanger in a large company in Malmö. Here the self-tuner proved to be superior to the existing commercial regulator.

Key words
Classification system and/or index terms (if any)
Supplementary bibliographical information
ISSN and key title
ISBN
Language
Swedish
Number of pages
90
Recipient's notes
Security classification

SJÄLVINSTÄLLANDE REGULATORER

PÅ MIKRODATOR I HÖGNIVÅSPRÅK

LARS BÅATH
PER-OLOF MALMQVIST

INSTITUTIONEN FÖR REGLERTEKNIK
LUNDS TEKNISKA HÖGSKOLA
1981

INNEHÅLLSFÖRTECKNING

1. INLEDNING

Detta examensarbete initierades efter kontakter med avdelningen för styr- och reglerteknik vid ÅF Energikonsult, Malmö. Vid dessa samtal framkom önskemål om att få en självinställande regulator implementerad på avdelningens mikrodatorsystem. Detta system är baserat på en Z80 med ett realtidsoperativsystem utvecklat vid ÅF. Efter diskussion med vår handledare Carl Fredrik Mannerfelt vid Institutionen för Reglerteknik,LTH , kunde vi formulera vår målsättning enligt följande

-Implementering och utprovning av en självinställande regulator på en mikrodator,Z80. Speciellt ska programspråket vara Pascal samt regulatorn ha möjlighet till ett flertal reglerloopar.

Emellertid uppstod en del problem ty det befintliga realtidsoperativsystemet var ej anpassat för program skrivna i Z80-Pascal. Efter samråd med ÅF och institutionen beslöt vi tillsammans med vår handledare att istället utföra examensarbetet på institutionens LSI-11 dator, men i övrigt ej ändra på vår målsättning. Sålunda kan vår nuvarande målsättning formuleras

-Implementering och utprovning av en självinställande regulator på LSI-11. Speciellt ska programspråket vara Pascal samt regulatorn ha möjlighet till ett flertal reglerloopar.

Vi har utfört praktiska experiment på två av institutionens labprocesser samt en vid ÅF befintlig anläggning för luftvärmeåtervinning , en sk entalpiväxlare.
Vi gjorde även förberedande försök på en värmeväxlare för förbrukningsvatten hos ÅF. Emellertid visade det sig att processens ställdon hade en inställningstid som var avsevärt längre än varaktigheten hos de störningar vi kunde observera. Detta plus ställdonets relativt stora läckflöde gjorde att vi bedömde denna process som mindre lämplig för uttestning av programpaketet.

Erfarenheterna av programpaketet har varit mycket goda. Speciellt kan nämnas några praktiska hänsynstaganden som har förbättrat funktionen avsevärt
- de införda restriktionerna på styrsignalen
- förhindrande av en ev explosion av P-matrisen
- användandet av fördröjda parameterestimat i styrsignalen

Vidare bör påpekas vikten av att utnyttja AD-omvandlarens hela utstyrningsområde för att få säkrare estimat och därmed en bättre styrning.

På laborationsprocesserna har vi provat två olika regulatorer. Avgörande skillnaden mellan dessa båda, Regul 1 och Regul 2, har varit sättet att behandla bias-komponenten. I Regul 1 skattas en biasparameter , modellen tillåter flera B-parametrar men endast en tidsfördröjning. I Regul 2 differensbildar vi signalerna och bias uppfattas därmed som ett tillstånd hos en extra integrator i processen. Denna modell har endast en B-parameter men däremot flera tidsfördröjningar.

De olika avsnitten har vi disponerat så att i kap 2 definieras målet för vårt examensarbete.
Kap 3 innehåller teorin för de båda modellerna utgående ifrån ARMA-modellen. I slutet på kapitlet kommenteras en del beräkningstekniska synpunkter på de olika regulatorerna samt de införda olinjäriteterna på styrsignalen, dödzon och derivatabegränsing.
En beskrivning av programpaketet, dess minnesbehov samt övriga prestanda finns i kap 4.
Försök gjordes på institutionens värmefläkt och saltprocess, samt den hos AF i drift varande entalpiväxlaren. De olika experimentella försöken och resultaten av dessa beskrives i kap 5.
Slutligen finns i kap 6 en avslutning och sammanfattning av våra erfarenheter från detta examensarbete.
Appendix innehåller programlistningar i tur och ordning av Regul 1, Regul 2 samt de SIMNON-program vår handledare skrev vid den första utprovningen av algoritmerna.

Vi vill här också passa på att tacka vår handledare Carl Fredrik Mannerfelt för allt det han har lärt oss, samt sist men inte minst det stora tålamod han har visat vid våra "intelligenta" frågor.

2. PROBLEMDEFINITION

Målet med detta examensarbete var att realisera en självinställande regulator av minimalvarianstyp, skriven i högnivåspråket Pascal på en mikrodator LSI-11.

Programpaketet skall klara 10 st. separata reglerloopar samt ha möjlighet till "on-line" förändringar av varje regulators designparametrar.

3. TEORI

3.1 Inledande teori

För en beskrivning av processen som skall styras har vi utgått från en ARMA-modell dvs

$$A(q^{-1})y(t)=B(q^{-1})u(t-k)+D(t)$$

där B-polynomet har alla nollställen innanför enhetscirkeln. D(t) består ofta av en statisk komponent, "bias", vilken betecknas med d, samt en komponent av stokastisk karaktär. Vi får då följande modell

$$A(q^{-1})y(t)=B(q^{-1})u(t-k)+C(q^{-1})v(t)+d$$

där v(t) är vitt brus. I fortsättningen antas $C(q^{-1})=1$.

Bias-komponenten, som ofta är försummad i litteraturen, har här behandlats på två olika sätt. I den första algoritmen Regul 1, skattas en separat bias-parameter d, som införs direkt i styrlagen.

I den andra algoritmen, Regul 2, differensbildas processens styr- och utsignal vilket innebär att en extra integrator beskriver den aktuella bias-nivån.

För vidare information och härledning av minimalvariansstyrlag med rekursiv LS-skattning av parametrarna, se referens {1}.

3.2 Regul_1

Målet med regleringen är en algoritm som samtidigt löser regulator och servoproblemet.

Modell

Antag att systemet kan beskrivas som

$$A(q^{-1})y(t)=B(q^{-1})u(t-1)+v(t)+d$$

där v(t) är vitt brus.
För att förtydliga härledningen av regulatorn utförs denna endast för en modell av 1:a ordningen med en B-parameter. Införs referensvärden som extra variabler kan modellen skrivas som

$$y(t+1)-y_r(t)+\alpha_1(y(t)-y_r(t-1))=\beta_o u(t)-y_r(t)-\gamma_1 y_r(t-1)+d+v(t+1)$$

Om felet $e(t)=y(t)-y_r(t-1)$ införs, kan modellen skrivas på formen

$$e(t+1)+\alpha_1 e(t)=\beta_o u(t)-y_r(t)-\gamma_1 y_r(t-1)+d+v(t+1)$$

Styrlag

Minimalvarians-styrlagen, enligt [1], baserad på ovanstående modell ger

$$u(t)=1/\beta_o(\alpha_1 e(t)+y_r(t)+\gamma_1 y_r(t-1)-d)$$

vilket resulterar i det slutna systemet $e(t+1)=v(t+1)$.

Estimering

Om parametrarna i ovanstående styrlag ersätts med sina minsta-kvadrat estimat, fås ett prediktionsfel för felet e(t)

$$\varepsilon(t)=e(t)-(-\hat{\alpha}_1 e(t-1)+\hat{\beta}_o u(t-1)-y_r(t-1)-\hat{\gamma}_1 y_r(t-2)+\hat{d})$$

Inför $\hat{\theta}^T(t)=[\hat{\alpha}_1 \;\; \hat{\beta}_o \;\; \hat{\gamma}_1 \;\; \hat{d}]$

vilka är de vid tidpunkten t estimerade parametrarna. Inför dessutom en regressionsvektor

$$\varphi^T(t)=[-e(t-1) \;\; u(t-1) \;\; -y_r(t-2) \;\; 1]$$

Nu kan det vid tidpunkten t-1 predikterade värdet av e(t) skrivas

$$\hat{e}(t)=\hat{\theta}^T(t-1)\varphi(t)-y_r(t-1)$$

$\hat{\theta}(t)$ kan rekursivt beräknas enligt de i {1} beskrivna formlerna:

$$\hat{\theta}(t)=\hat{\theta}(t-1)+P(t)\varphi(t)\varepsilon(t)$$

$$\varepsilon(t)=e(t)-\hat{\theta}^T(t-1)\varphi(t)$$

$$P(t)=\left[P(t-1)-\frac{P(t-1)\varphi(t)\varphi^T(t)P(t-1)}{\lambda(t)+\varphi^T(t)P(t-1)\varphi(t)}\right]/\lambda(t)$$

Beräkningstekniska aspekter

För att erhålla en snabbare insvängning av estimaten och kunna följa tidsvariabla parametrar införs en glömskefaktor, λ.

Eftersom bias-störningen skattas som en parameter, kan det vara lämpligt att använ[illegible] lambdaD, för denna. Detta för att en eventuell förändring av bias-nivån snabbt ska "plockas upp" av estimatorn.

Om processinformation endast fås i en riktning dvs φ-vektorn förändras ej nämvärt, kan P-matrisen växa till orimliga värden så att numeriska problem uppstår. Detta inses genom att betrakta den rekursiva ekvationen för P-matrisens invers.

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

Ur denna framgår också att man kan begränsa P-matrisens tillväxt genom att sätta λ=1.

Denna begränsning har vi infört då Tr P överstiger antalet diagonalelement multiplicerat med 2*PO. Även här behandlas bias-delen separat, såtillvida att lambdaD sätts lika med 1 då motsvarande diagonalelement överstiger 2*PO.

Om man kan antaga att estimatens förändringar sker långsamt, kan föregående estimat användas vid beräkningen av styrsignalen. Detta medför en mer tidsexakt utställning av styrsignalen, och man får ett avsevärt mindre beroende av antalet parametrar som estimeras. Huvuddelen av beräkningstiden i datorn åtgår för estimering, vars tidsåtgång är starkt beroende av antalet parametrar.

I många tillämpningar är det ur praktisk synpunkt nödvändigt att begränsa styrsignalen på olika sätt. Vi har här infört begränsningar i form av dödzonsbegränsning, derivatabegränsning samt största och minsta styrsignal.

Dödzonsbegränsning innebär att om skillnaden mellan den beräknade och föregående styrsignal är mindre än begränsningen förändras ej styrsignalen. Detta medför dock att ett litet stationärt fel kan uppkomma.

Deadbeat-styrning samt minimalvarians-styrning innebär ofta stora styrsignaler, vilka kan medföra påfrestningar för processens styrdon. Derivata begränsningen är ett försök att förhindra detta . Styrsignalen skall följa en motsvarande analog derivata, dvs styrsignal-förändringen får ej överskrida begränsningen, i programmet kallat uderv, multiplicerad med samplingsintervallet. Denna begränsning

bör dock inte inträda för normala störningar.

Största och minsta styrsignal är en naturlig begränsning då olika styrdon ej har samma insignalsområde. Det är viktigt att begränsningarna på styrsignalen inträder före ställdonens begränsningar. Detta först och främst för att slippa "wind-up" av styrsignalen, och dessutom för att få vettiga parameter-estimat.

Det bör observeras att speciellt derivatabegränsningen kan innebära minskad processinformation för parameter-estimeringen. Samtliga dessa begränsningar är starkt processbundna och bör användas med viss försiktighet.

3.3 Regul_2

Målet är att kunna styra processer med en tidsfördröjning som ej är försumbar i jämförelse med processens tidkonstant.

Modell

Antag att vi har systemet

$$A(q^{-1})y(t)=bu(t-k)+d'+v(t) \quad ; \quad \text{grad } A = n$$

där k är antalet tidsfördröjningar, d' svarar mot bias-nivån i systemet och v(t) är vitt brus.

Inför polynomidentiteten

$$1=AR'+q^{-k}S \quad ; \quad \text{grad } R'=k-1, \quad \text{grad } S = n-1$$

samt $R=bR'$ och $d=R'd'$

Nu kan processen beskrivas enligt

$$y(t)=Ru(t-k)+Sy(t-k)+d+R'v(t)$$

Styrlag

Minimalvarians styrlagen med kända parametrar och känd bias samt referenssignalen $y_r(t)$ införd

$$Ru(t-k)=-Sy(t-k)-d+y_r(t)$$

Insätt skattningarna av R och S samt uttryck bias-nivån i termer av estimaten

$$\hat{d}=y(t)-\hat{R}u(t-k)-\hat{S}y(t-k)$$

så kan styrlagen skrivas

$$\hat{R}(u(t)-u(t-k))=-\hat{S}(y(t)-y(t-k))-y(t)+y_r(t)$$

Inför de differentierade variablerna

$$\Delta y(t)=y(t)-y(t-k)$$
$$\Delta u(t)=u(t)-u(t-k)$$

då fås $\hat{R}\Delta u(t)=-\hat{S}\Delta y(t)-y(t)+y_r(t)$

vilket ger styrlagen

$$u(t)=u(t-k)-[\hat{r}_1\Delta u(t-1)+...+\hat{r}_{k-1}\Delta u(t-k+1)+$$
$$+\hat{s}_o\Delta y(t)+...+\hat{s}_{n-1}\Delta y(t-n+1)+y(t)-y_r(t)]/\hat{r}_o$$

Estimering

Estimeringen baseras på LS-metoden och regressorerna utgörs här av differensbildade in- och utsignaler. På detta sätt påverkas ej parameter-estimaten av eventuella nivåer på signalerna.

Utgå ifrån följande modell

$$\nabla y(t)=R\nabla u(t-k)+S\nabla y(t-k)$$

där ℓ-differenserna

$$\nabla y(t)=y(t)-y(t-\ell)$$
$$\nabla u(t)=u(t)-u(t-\ell)$$

kan väljas godtyckligt.

Om de estimerade parametrarna införs så kan prediktionsfelet beräknas enligt

$$\varepsilon(t)=\nabla y(t)-[\hat{R}_{t-1}\nabla u(t-k)+\hat{S}_{t-1}\nabla y(t-k)]$$

Inför nu regressionsvektorn

$$\varphi^T(t)=[\nabla u(t-k)...\nabla u(t-2k+1)\ \nabla y(t-k)...\nabla y(t-k-n+1)]$$

samt parametervektorn

$$\hat{\theta}^T(t)=[\hat{r}_o\ ...\ \hat{r}_{k-1}\ \hat{s}_o\ ...\ \hat{s}_{n-1}]$$

vilken består av de vid tidpunkten t estimerade parametrarna.

Nu kan prediktionsfelet skrivas som

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

$\hat{\theta}^T(t)$ beräknas med de rekursiva formlerna för LS-skattning, jfr {1}.

Beräkningstekniska aspekter

Vi har antagit att endast långsamma förändringar av bias-nivån uppträder. Detta medför en frihet att välja olika ℓ-stegs differenser i estimatorns processmodell, och därmed kan olika filtreringar av regressorerna i estimeringen uppnås. I programmet har vi kallat antalet steg i denna differensbildning för estk. För exempelvis $\ell=2$ fås

$$\nabla u(t)=u(t)-u(t-2) \quad \text{eller} \quad \nabla u(t)=(1-q^{-2})u(t)$$

dvs bias-nivå samt signaler med Nyqvist-frekvensen, påverkar ej estimeringen. Det har vid praktiska försök visat sig vara lämpligt att välja estk lika med antalet fördröjningar i processmodellen.

I övrigt sker parameter-estimeringen på samma sätt som i Regul 1.

4. BESKRIVNING AV PROGRAMPAKET

Programpaketet är skrivet i högnivåspråket Pascal samt använder det av Institutionen för Reglerteknik, LTH utvecklade realtidsoperativsystemet Kernel, se referens [2]. Programpaketet kan maximalt hantera 10 st reglerloopar. Minnesbehovet för Regul 1 är 32 kbyte och för Regul 2 30 kbyte. Varje reglernod kräver av detta ca 300 byte vardera. Större delen av minnesutrymmet består av operatörskommunikation och listhantering.
Den kortaste samplingsperioden om man samtidigt önskar en väl fungerande operatörskommunikation bör ej understiga 100 ms. Både Regul 1 och Regul 2 kan maximalt estimera 9 st parametrar. Detta betyder för Regul 1 att 2*na+nb=<8 och för Regul 2 att max(k+estk+na,2*k+estk)=<9.
Programmet kan i stort sett uppdelas i fem delar, vilka visas i figur 4.1 .

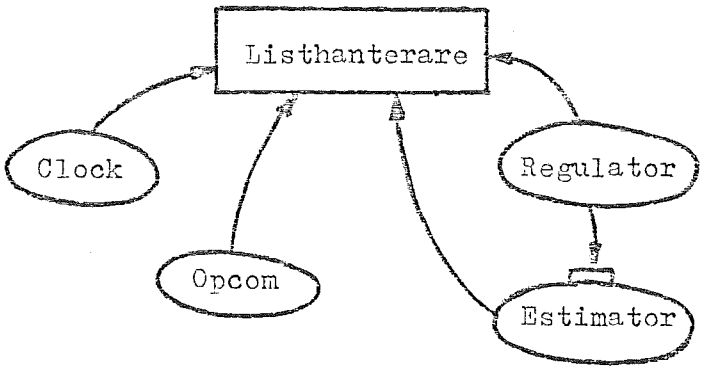


Fig. 4.1 Programpaketets struktur

Varje reglernods parametrar är lagrade i en record av meddelandetyp. Dessa meddelande administreras av listhanteraren vilken sorterar in dem i länkade listor.
Varje nod kan arbeta i en av följande moder, selftuning, tuning eller fixregulator.
I selftuning mode arbetar regulatorn med de aktuella parameter-estimaten.
Detta är däremot ej fallet i tuning mode. Där arbetar regulatorn med manuellt inmatade parametrar eller med de parametrar som fanns vid övergången till tuning mode.
I Regul 1 ställer detta till en del problem ty man kan ej

låsa fast bias-parametern då detta skulle medföra ett stationärt reglerfel vid en ev förändring av bias-nivån. Vi har löst detta genom att fortsätta estimeringen av bias-parametern enligt

$$\hat{d}(t+1) = \hat{d}(t) + freezek * \varepsilon(t)$$

Freezek är det senaste innan övergång till tuning mode beräknade elementet i $P(t)*\varphi(t)$, som svarar mot biasparametern, jmf kap 3.1. I övrigt estimeras alla parametrar i bakgrunden av estimatorn på samma vis som i selftuning mode.
I fixreg_mode stängs estimeringen helt av utom för biasparametern, vilken beräknas enligt samma mönster som i tuning mode.

Operatörskommunikationen är utformad så att "on-line" förändring av designparametrar för varje regulatormod är möjlig. Dessa parametrar är för Regul 1:

tsamp - samplingsperiod i sekunder

uhi, ulo, uderv, deadzon - de i föregående kapitel nämnda begränsningarna

na, nb - gradtalet på A- repektive B-polynomet

yrchan, ychan, uchan - kanaler för in- och utgångar

lam, lamD - glömskefaktorer där lamD användes enbart för biasparametern

PO - initialvärdet för kovariansmatrisen, PO*I

För Regul 2 gäller samma designparametrar frånsett nb och lamD. Istället kommer här in k samt estk där k betecknar antalet fördröjningar räknat i samplingsperioder och estk är estimatorns tidsdifferens.

I övrigt skriver operatörskommunikationen ut regulatormodellen , i tuning mode även estimatormodellen, samt P-matrisen.

Den hanterar även ett antal olika kommandon vilka opererar på [illegible] i ett av tillstånden enligt figur 4.2 . Kommmandon som överför noden mellan olika tillstånd står angivna vid pilarna.

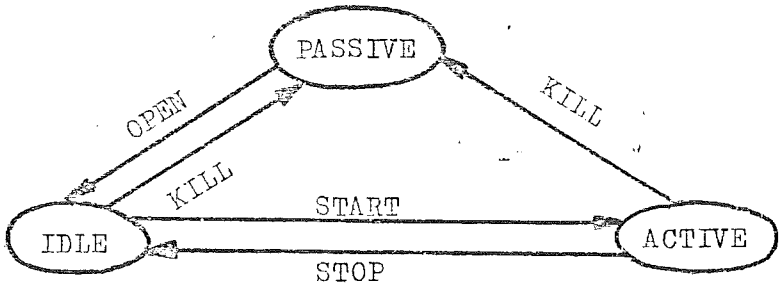


Fig. 4.2 Regulatornodernas olika tillstånd

Befinner sig noden i tillstånden IDLE eller ACTIVE finnes även ett antal andra kommandon att tillgå. Tillgängliheten hos dessa kommandon samt op.kommunikationens svar visas i figur 4.3 .

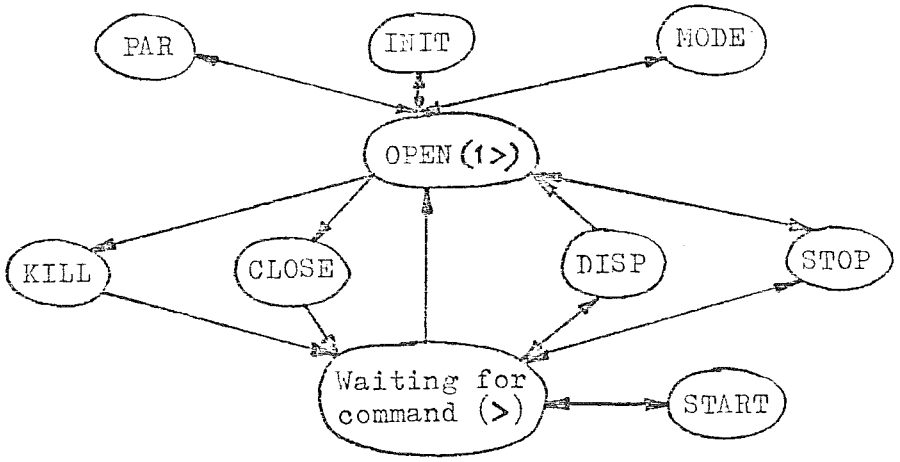


Fig. 4.3 Tillgängligheten hos olika kommandon

OPEN - öppnar en nod så att nästföljande fyra kommandon blir tillgängliga

PAR - ger möjlighet att ändra de ovan nämnda regulatorparametrarna.

INIT - initierar kovariansmatrisen samt ger möjlighet att ändra startvärden på modellens parametrar.

MODE - välja fixreg, tuning eller selftuning mode för regulatornoden.

KILL - överför regulatornoden från IDLE- eller ACTIVE- till PASSIVE-tillstånd.

CLOSE - inför gjorda ändringar av parametrar till regulatorn.

DISP - skriver ut parametrar för angiven nod eller tillståndena för alla noder.

STOP - överför noden från ACTIVE- till IDLE-tillstånd.

START - överför noden från IDLE- till ACTIVE-tillstånd.

Om man i PAR-kommandot ändrat någon av de för modellen väsentliga parametrarna dvs gradtal av polynom, fördröjning eller samplingstid blir automatiskt kovariansmatrisen samt θ- och φ-vektorerna återinitierade. Detta sker även vid START kommandot, dock med undantag av θ-vektorn som behåller sina gamla värden. Vid övergång från PASSIVE- till IDLE-tillstånd sätts alla parametrar till default-värden.

5. PRAKTISKA FÖRSÖK

I samarbete med vår handledare, Carl Fredrik Mannerfelt, diskuterade vi fram olika regleralgoritmer, Regul 1 och Regul 2, vilka senare utprovades av honom genom simuleringar i programpaketet SIMNON.Listning av SIMNON-programmen finns i appendix C.

5.1 Simuleringar på analogimaskin

För att enkelt kunna utprova implementeringen av regleralgoritmerna Regul 1 och Regul 2 på LSI 11 utfördes försök på analogimaskin där processer av 1:a och 2:a ordningen simulerades.Vid simuleringarna provades olika val av samplingstid, gradtal, begränsningar av styrsignalen etc.

Nedanstående kurvor demonstrerar vilken inverkan derivatabegränsning av styrsignalen har. Vi har även simulerat en förändring av biasnivån i båda försöken.
Försöket är utfört på 1:a ordningens system med överföringsfunktionen G(s) = 1/(s+1) vilken har reglerats med Regul 2. Då processens tidskonstant T=1 s bör en samplingsperiod på 0.5 s vara ett lämpligt val.
Designparametrarna valdes till:

tsamp=0.5 s , na=1 , k=1 , estk=1

Periodtiden för referenssignalen yr är i Försök 1 t=25 s och i Försök 2 t=20 s.

Försök_1

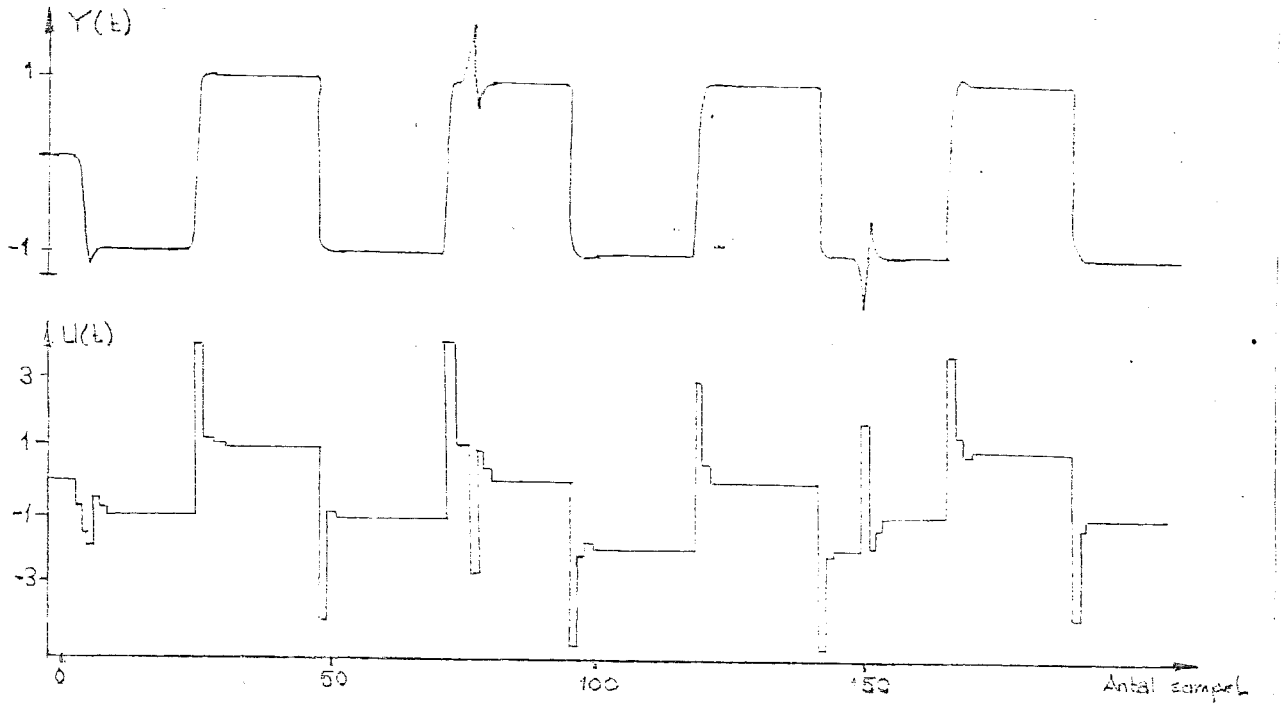


Fig. 5.1.1 Uppstart och reglering utan begränsning av styrsignalen.

Regulatorn ger här ett acceptabelt stegsvar efter 20-30 sampel. Efter 80 sampel införs en biasnivå vilken snabbt regleras ut. Biasnivån tas bort då 150 sampel har förflutit och man ser att regulatorn snabbt adapterar sig.

Försök_2

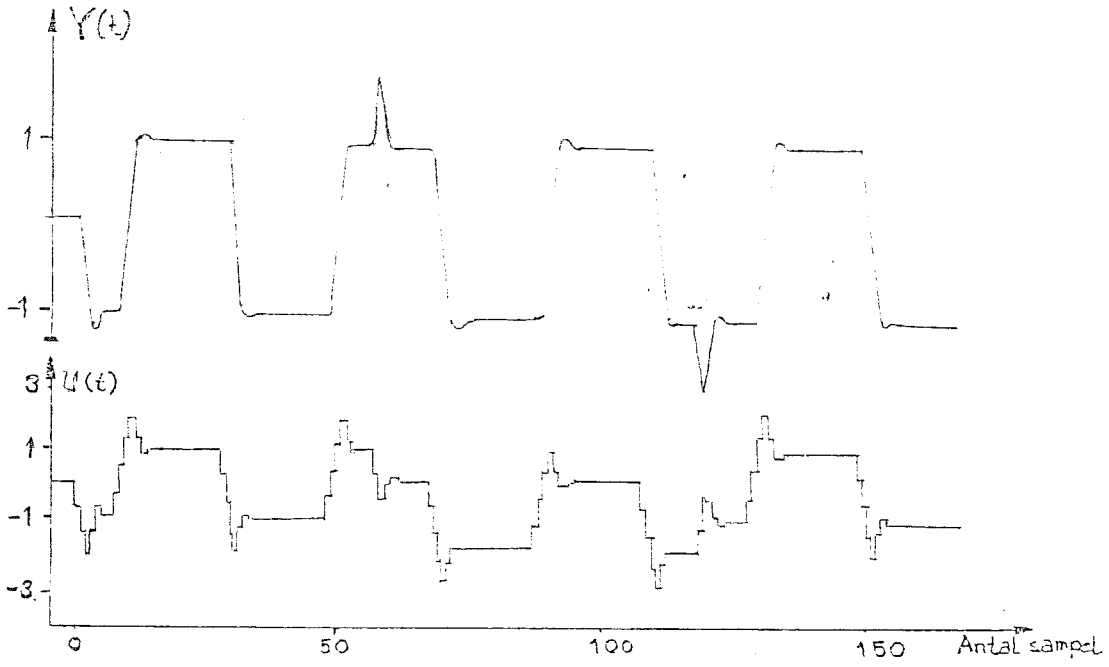


Fig._5.1.2 Uppstart och reglering med styrsignalen begränsad till udery=2 V/s.

Även här har regulatorn ställt in sig på 20-30 sampel. Stegsvaret är dock långsammare än tidigare beroende på derivatabegränsningen. Styrsignalen har här ett betydligt lugnare beteende än i föregående försök.

Försöken visar att man med fördel kan införa en derivatabegränsning av styrsignalen om man samtidigt har måttliga krav på systemets snabbhet. Detta är användbart i industriella tillämpningar där det är viktigt att inte utsätta ställdonen för alltför stora påkänningar.

5.2 Värmefläkt

Processbeskrivning

Processen består väsentligen av inloppsspjäll(A), fläkt(B), värmespiral(C) och temperaturgivare(D). Temperaturgivaren kan flyttas i förhållande till värmespiralen, så att mätning sker på olika avstånd från denna.

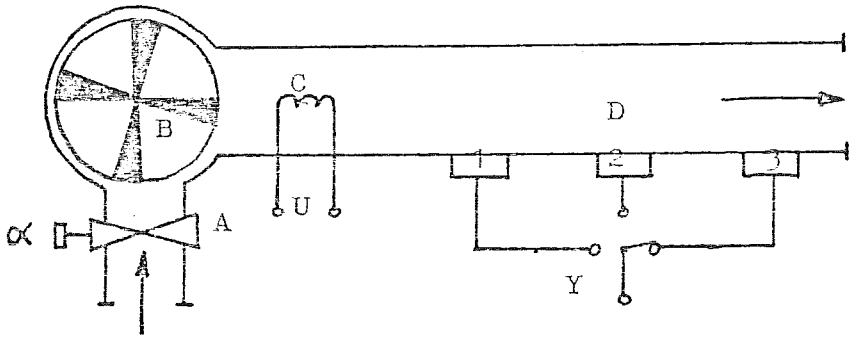


Fig. 5.2.1 Värmefläkt

öppningsvinkeln α, för spjället kan regleras mellan 10-160 grader. Stort α ger ett stort luftflöde vilket medför låg förstärkning samt kort fördröjning medan ett litet α ger den omvända effekten.

För att få en uppfattning om processens fördröjning,τ,samt tidskonstant,T,togs stegsvar upp för ett antal olika öppningsvinklar på spjället.

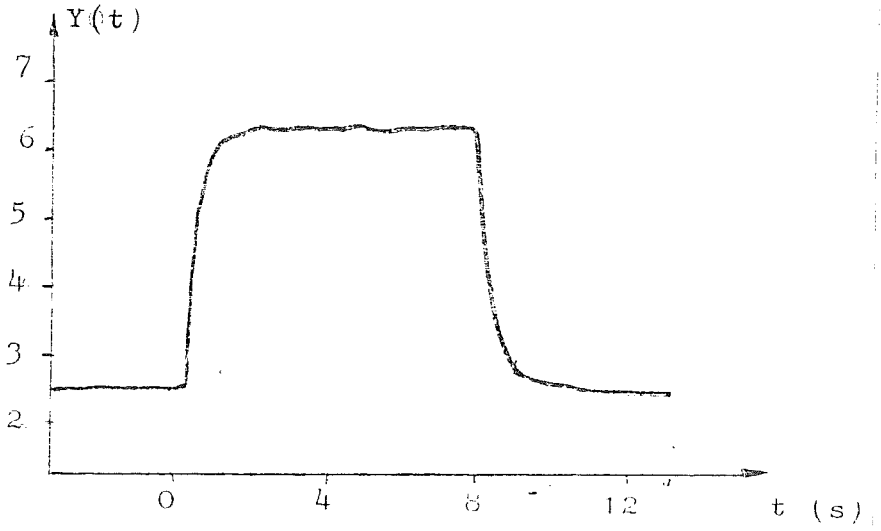


Fig. 5.2.2 öppningsvinkel α=10 ger K=0.4, τ=0.4 s, T=0.5 s

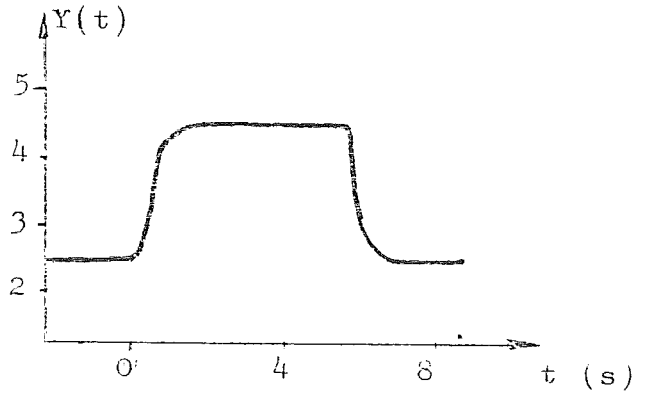


Fig. 5.2.3 öppningsvinkel α=35 ger K=0.2, τ=0.25 s, T=0.6 s

Stegsvaren uppför sig väsentligen som om det svarade mot ett första ordningens system med dödtid.En enkel approximation ger processmodellen

$$Y(s)=K\,\frac{e^{-s\tau}}{1+sT}\,U(s)$$

där parametrarna K,τ,T samtliga beror på spjällets öppningsvinkel.
Utsignalen, y, ligger inom området 2.5-6.5 V då styrsignalen u, varieras mellan 0-10 V. Utsignalen y=2.5 V, dvs biasnivån, svarar mot rumstemperead luft, 20 grader C. Notera att här är biasnivån relativt stor i förhållande till utstyrningsområdet för processen.

De störningar som uppkommer vid körning av processen är kortvariga och beror i huvudsak på svaga temperaturvariationer i den inkommande luften.

Försök

I utprovningen av Regul 1 och Regul 2 gjordes körningar där samplingstid, gradtalet för modellen, antal tidsfördröjningar etc ändrades. Samtliga försök utfördes med spjällöppningsvinkeln α=35 samt temperaturgivaren i position nr 3. Detta ger processmodellen

$$G(s)=0.2\,\frac{e^{-0.25s}}{1+0.6s}$$

Samtliga försök utfördes utan församplingsfilter.

Regul_1

Med utgångspunkt från ovanstående processmodell samt krav på ett snabbt stegsvar borde tsamp=0.6 s vara ett lämpligt val av samplingsperiod. Körningen nedan startades med följande designparametrar:

tsamp=0.6 s
na=nb=1
lam=lamD=0.98

Referenssignalen yr består här av en fyrkantvåg med periodtiden T=12 s .

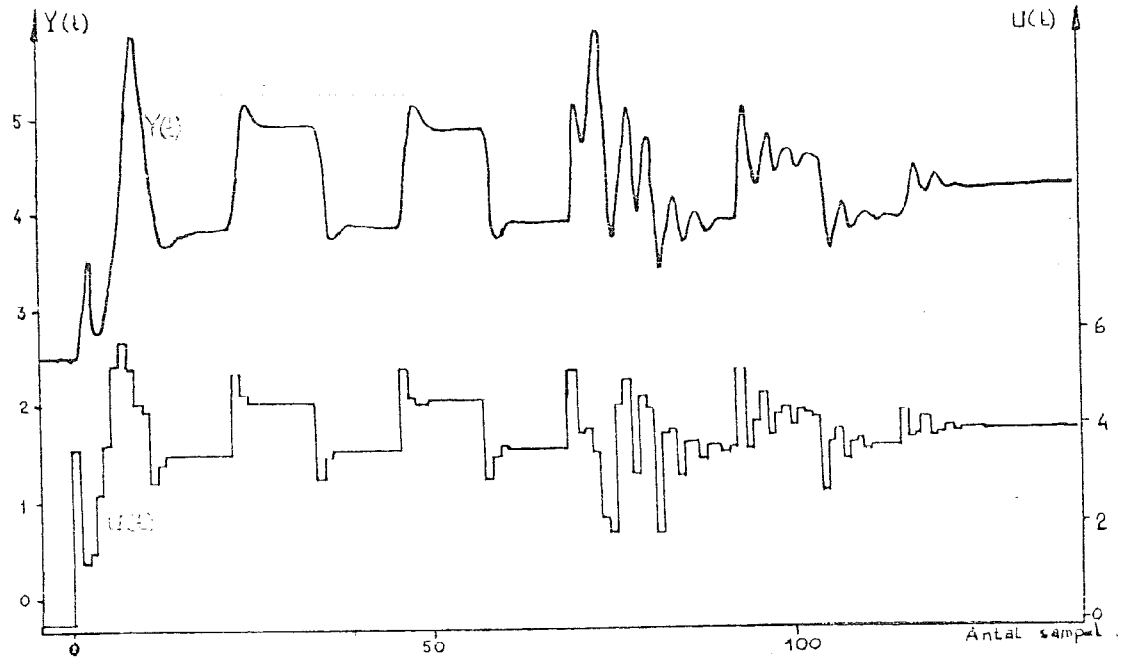


Fig. 5.2.4 Uppstart och reglering med Regul 1

Regulatorn har efter 20-30 sampel ställt in sig så att ett acceptabelt stegsvar erhålls. Vid denna tidpunkt har även styrsignalen ett acceptabelt utseende. Svängningen minskar väsentligt efter en längre tids körning.
Efter ca 70 sampel störs processen genom att spjällöppningen minskas kortvarigt. Efterföljande stegsvar är oscillativt men svängningarnas amplitud dämpas succesivt.

Regul_2

I Regul 2 finns möjlighet att variera antalet tidsfördröjningar. Genom att införa en extra tidsfördröjning samt välja samplingsperioden tsamp=0.3 s kan stegsvaret ytterligare snabbas upp.
Designparametrar:

tsamp=0.3 s
na=1
k=estk=2
lam=0.98
uderv=2 V/s

Även i detta fall består referenssignalen av en fyrkantvåg med periodtiden 12 s.

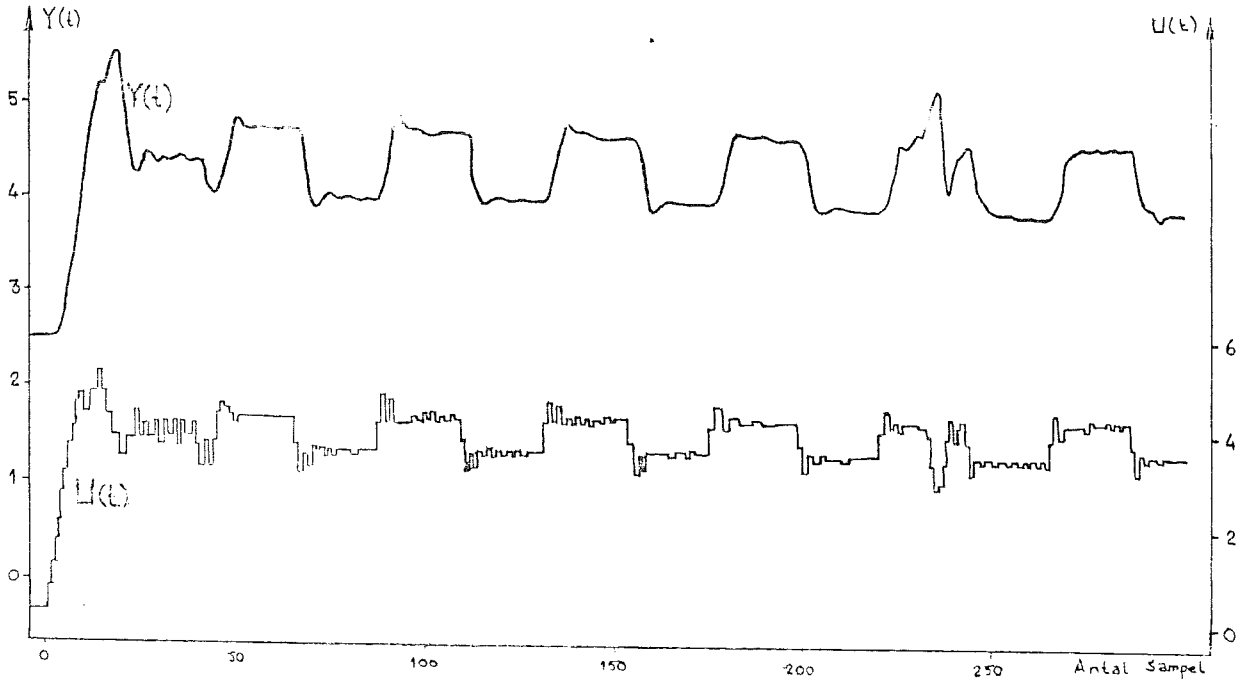


Fig. 5.2.5 Uppstart och reglering med Regul 2

Efter ca 40 sampel har regulatorn ställt in sig så att ett acceptabelt stegsvar erhålles. Även här har stegsvaret en översläng som dock kraftigt minskas efter ca 180 sampel då

även styrsignalen får ett bättre utseende.
På processen införs efter ca 230 sampel samma typ av störning som i försöket med Regul 1. Regulatorn klarar som synes denna störning bra och efterföljande stegsvar [illegible] acceptabla.

Erfarenheter

I ovanstående körning med Regul 1 har lamD valts till samma värde som lam, dvs 0.98. Vanligtvis väljes ett lågt värde på lamD, 0.5-0.6, för att regulatorn snabbt ska kunna ta hand om biasförändringar. Vid ett lågt värde på lamD förhindrades dock ibland α,β,γ-parametrarna att konvergera mot relevanta värden. Det blir då väsentligen biasparametern som tar hand om reglerfelen, dvs vi får mycket långsamma stegsvar.
Vid körning med Regul 2 valdes tsamp=0.35 s. Med utgångspunkt från vår modell skulle tsamp teoretiskt kunna väljas till 0.25 s. Det har dock visat sig ge ett betydligt lugnare uppträdande av både ut- och styrsignal om modellens fördröjning väljes något längre än den verkliga i processen.

5.3 Saltprocessen

Processbeskrivning

Systemet som skall [illegible] är en kontinuerlig process för inblandning av salt i vatten till önskad koncentration. En schematisk bild av processen visas i figur 5.3.1

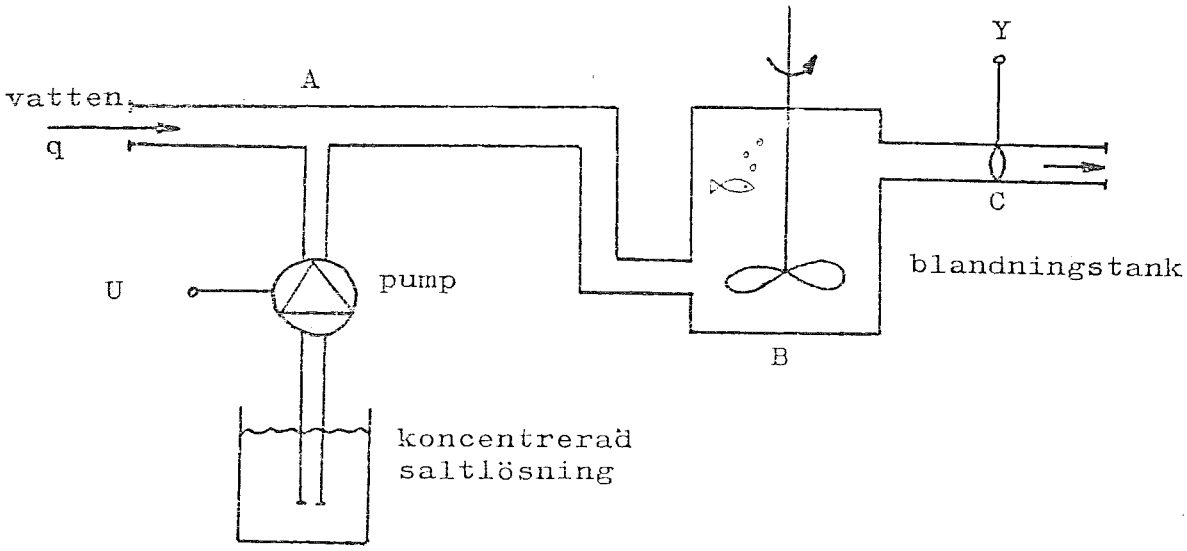


Fig. 5.3.1 Schematisk bild av saltprocessen

Vid A injiceras koncentrerad saltlösning i vattnet. Detta blandas till en homogen lösning i tanken B. Vattnets konduktivitet, som ger ett mått på saltkoncentrationen, mäts vid C.

Styrsignalen till processen är insignalen till en elektrisk dialyspump, vilken injicerar saltlösning i vattnet. Pumpens karakteristik är något olinjär.

En approximativ modell för systemet kan vara av första ordningen med dödtid, dvs

$$Y(s)=K\,\frac{e^{-s\tau}}{1+sT}\,U(s)$$

Stegsvaret i figur 5.3.2 är upptaget då vattenflödet håller nominell nivå, 100%.

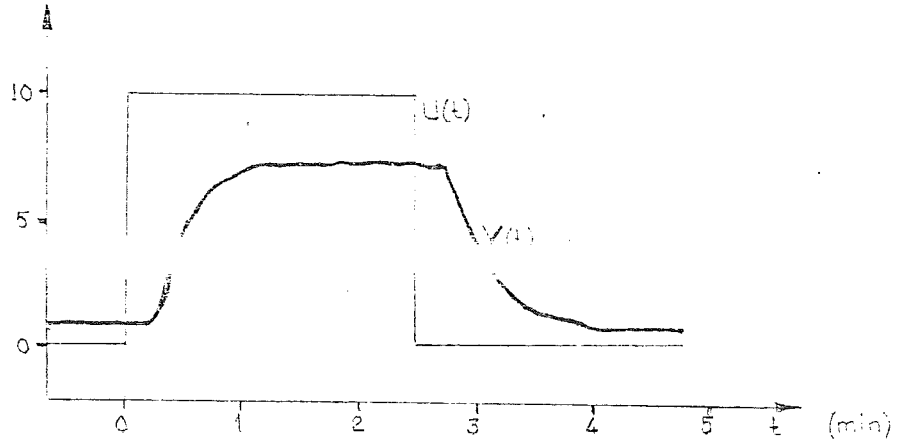


Fig. 5.3.2 Stegsvar, saltprocess K=0.6 , τ=13 s , T=20 s

Vid reglering av processen är just vattenflödet en processförändring som långsamt varierar omkring det nominella värdet. Detta påverkar processparametrarna så att T,τ och K ökar vid minskande flöde.
Utsignalen, y, kan ibland störas av luftbubblor som passerar konduktivitetsmätaren. Likaledes är blandningen i tanken ej helt fullständig, vilket kan medföra oväntade förändringar i utsignalen.

Försök

Högfrekventa mätstörningar från processen orsakade genom vikningseffekten att styrsignalen oscillerade. Detta försökte vi undvika genom att införa ett församplingsfilter. Enligt samplingsteoremet skall frekvenser högre än halva samplingsfrekvensen skäras bort. I detta fallet är samplingsfrekvensen låg och för att ej ta bort för mycket information ur signalen eller få för mycket invikta rester valde vi att använda ett andra ordningens Butterworthfilter. Med Regul 1 använde vi oss enbart av ett första ordningens lågpassfilter vilket visade sig vara tillräckligt.

Regul_1

Med utgångspunkt från stegsvaret i fig. 5.3.2 kan en lämplig samplingsperiod vara 45-50 s.
Fig. 5.3.3 visar ett försök med

na=nb=1
tsamp=50 s
lam=lamD=0.98

Att välja glömskefaktorerna lika stora i uppstarten visade sig vara bra, ty om biasparameterns glömskefaktor, lamD, valdes mindre än lam, konvergerade inte alltid de övriga parametrarna till vettiga värden.
Efter ca 20 sampel ändrades lam till 0.8. Motivet bakom detta är att då vi antar att biasförändringar, dvs det rena vattnets konduktivitet, sker betydligt långsammare än andra processförändringar som tex vattenflödet, bör denna parameters historia kommas ihåg bättre än för de andra parametrarna.
Efter ytterligare ca 5 sampel ökas vattenflödet med ca 10%. Då utsignalen åter ställt in sig minskas vattenflödet till 10% under nominell nivå.

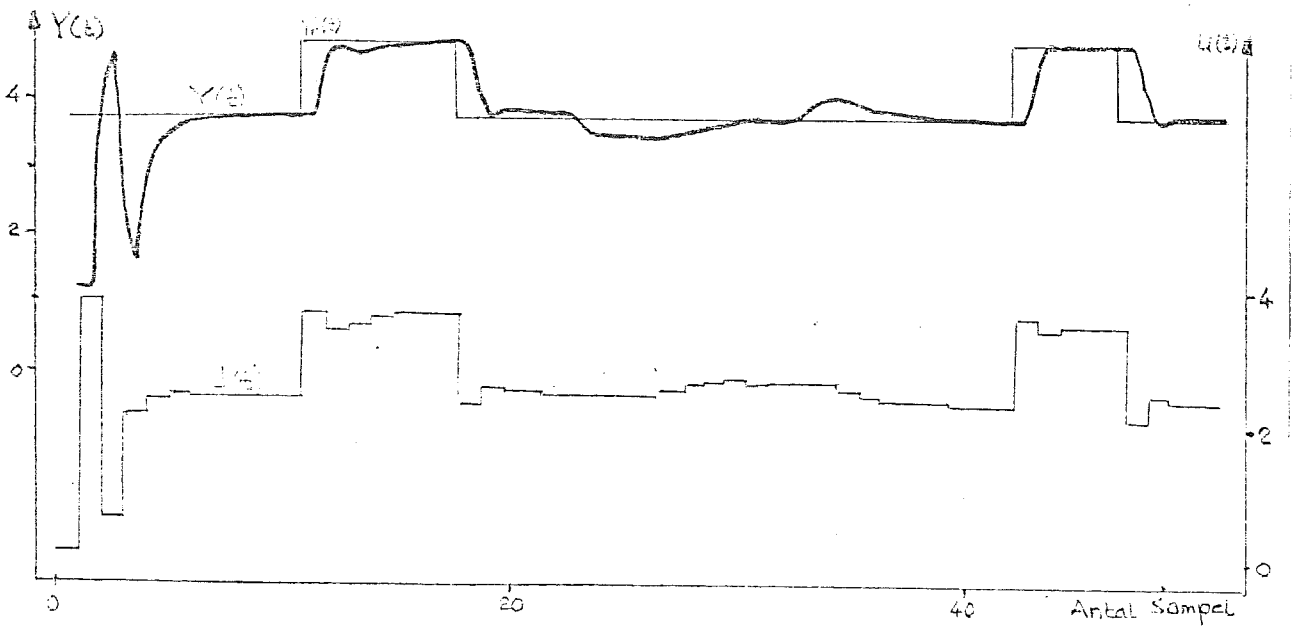


Fig. 5.3.3 Regul 1, lam<lamD efter uppstart.

I fig. 5.3.4 är ett liknande försök gjord fast då med lam>lamD. Man ser här att inställningstiden efter en flödesförändring är längre samt att uppförandet efter denna är betydligt sämre.

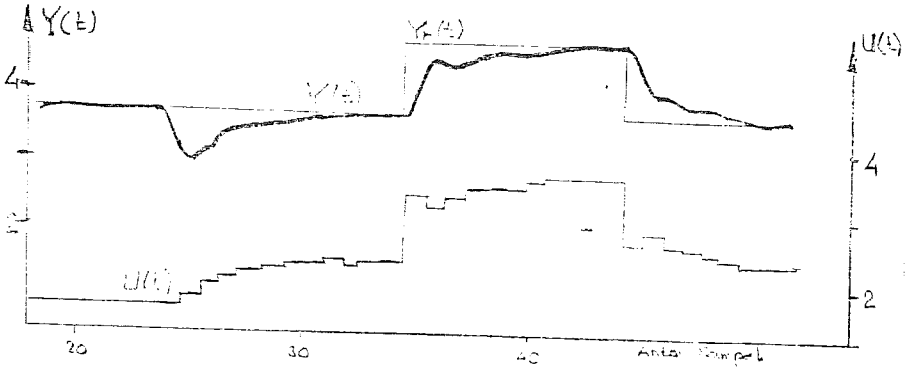


Fig. 5.3.4 Regul 1, lam>lamD efter uppstart.

Regul 2

Eftersom fördröjningen i processen är ca 13 s vid nominellt vattenflöde, skall Regul 2 kunna klara en samplingsperiod på ca 15 s med en extra tidsfördröjning införd.
Fig. 5.3.5 visar ett försök med

na=1 , tsamp=15 s
k=estk=2
lam=0.98

Församplingsfiltret är här ett 2:a ordningens Butterworthfilter.

Efter 110 samplingsperioder ökades vattenflödet med 35%, vilket regulatorn som synes klarade utmärkt. Detsamma gäller sänkningen till nominell nivå, vilket skedde ytterligare 25 sampel senare. När 170 sampel har förflutit minskas vattenflödet till 80% av nominell nivå. Detta medför att fördröjningen i processen blir större än samplingsperioden, vilket regulatorn ej är designad för och den börjar helt följdriktigt att självsvänga. En mindre sänkning tex 90% av nominell nivå, vilken utförs efter drygt 230 sampel, regleras däremot helt bort.
En störning simulerades efter 290 sampel. I detta fall injicerades extra saltlösning under 10 sek. Svårigheterna regulatorn här uppvisar beror med största sannolikhet på att informationen till estimatorn under de senaste 200 samplingsperioderna varit tämligen liten. En exakt likadan

störning genereras några tiotal sampel senare. Här ser man att modellen är betydligt bättre ty störningen regleras relativt snabbt bort.
Slutligen görs ett antal referensvärdesändringar och det syns både på utsignal och styrsignal att estimaten blir allt bättre.

Fig. 5.3.6 uppvisar en sekvens liknande den ovan beskrivna. Skillnaden består i att tsamp=20 s och församplingsfiltret är ett ordinärt 1:a ordningens lågpassfilter.

Vad som här bör observeras är den ökade möjligheten att klara processförändringar, tex flödesändringen till 80% vilken förorsakade instabilitet i föregående fall. Vidare utförs efter ca 260 sampel en förändring av referensvärdet då flödet är 20% under nominell nivå, vilket ger bevis för den goda adapteringsförmågan.

Erfarenheter

För Regul 1 har vi noterat att det i uppstarten är lämpligt att välja lam=lamD. Vidare är det i en process som den här, där biasnivån förväntas ändra sig betydligt långsammare än andra processparametrar som förstärkning, tidskonstant eller dödtid, lämpligt att välja lam mindre än lamD efter uppstart. Görs ej detta utan tex lamD väljes mindre än lam finns risk för att en ev processförändring "sugs upp" av biasparametern.
När det gäller Regul 2 har antagandet om att samplingsperioden skall kunna väljas ungefär lika lång som dödtiden, om en extra tidsfördröjning införes, visat sig vara riktigt. Dock måste man konstatera att regulatorn vid detta val blir känslig för högfrekventa störningar och därför kräver ett brant församplingsfilter. Likaledes blir förmågan att klara av en ökning av dödtiden mindre.
Ett bättre val är då att välja tsamp ung. 1.3 -1.5 ggr längre än dödtiden. Fördelarna med detta är då

-större adaptiv förmåga tex vid en ökning av dödtiden.

-mindre känslighet för högfrekventa störningar, tillräckligt med ordinärt 1:a ordningens församplingsfilter.

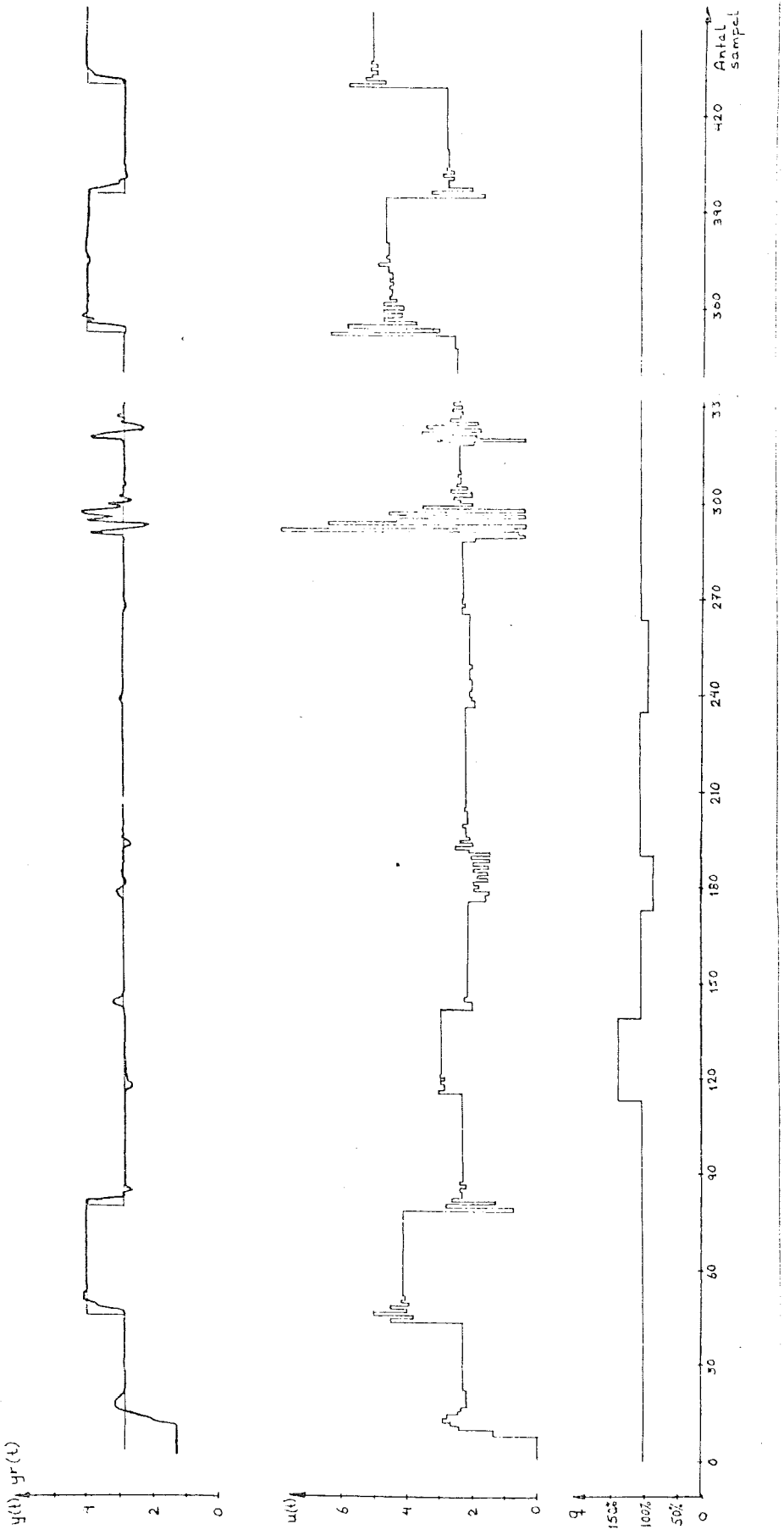


Fig. 5.3.5 Regul 2, saltprocess, samplingsperiod 15 s.

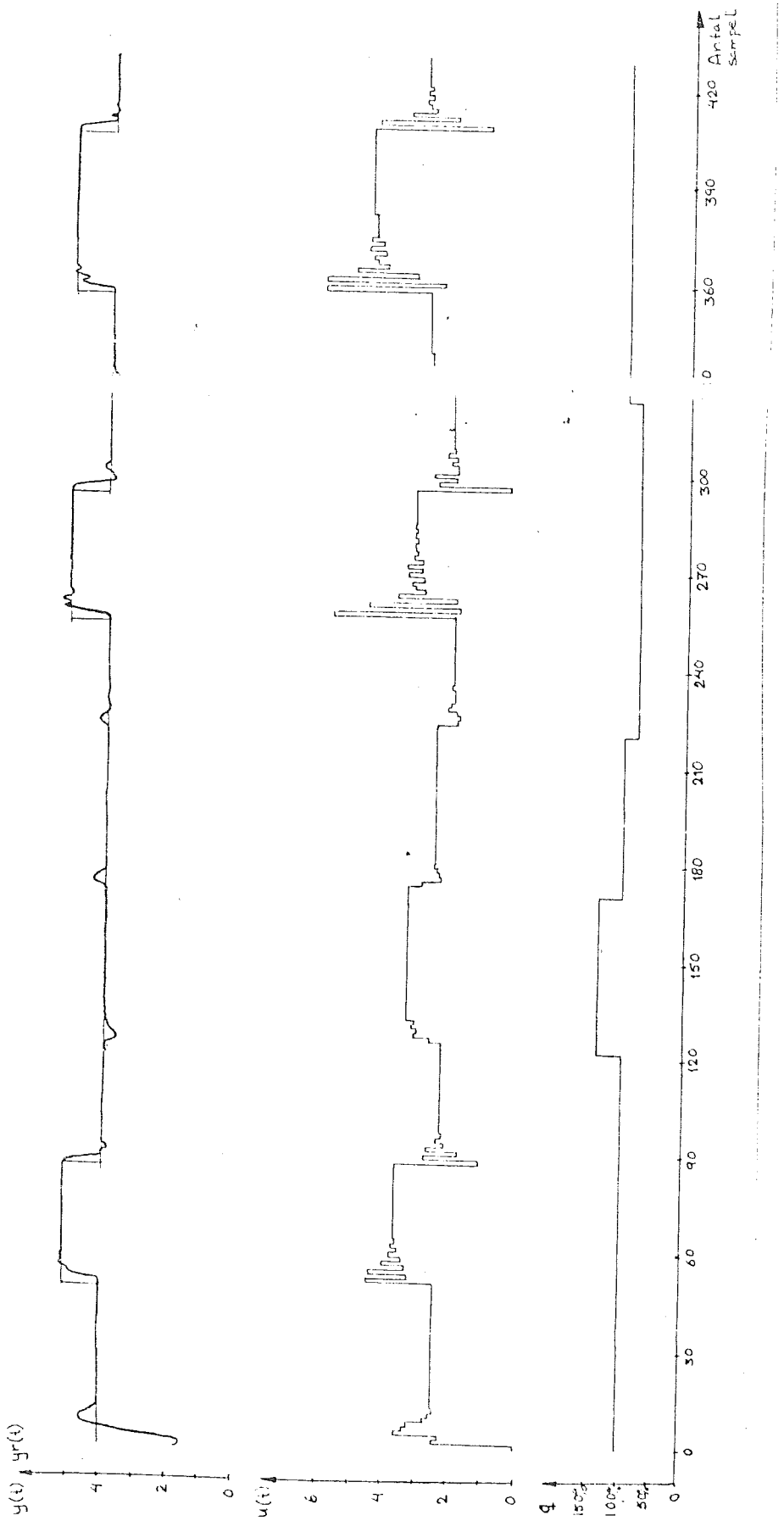


Fig. 5.3.6 Regul 2, saltprocess, samplingsperiod 20 s.

5.4 Entalpiväxlare

Detta försök har utförts på en sk entalpiväxlare i en anläggning för luftvärmeåtervinning, vid AF Energikonsult i Malmö.

Entalpiväxlaren är en luftvärmeväxlare med uppgift att överföra värme mellan två luftkanaler. Värmeväxlaren består i huvudsak av en skiva som roterar i axiella till- och frånluftskanaler.

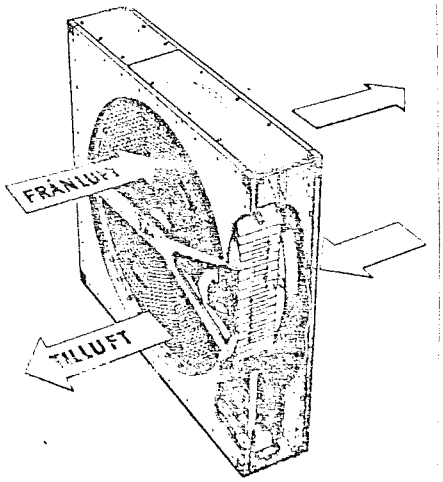


Fig. 5.4.1 Schematisk bild av entalpiväxlaren.

Värmeutbytet sker genom en kontinuerlig uppvärmning och avkylning av den roterande skivan, dvs skivans segment befinner sig omväxlande i till- och frånluftskanalen. Temperaturen i till-luften styrs därför med skivans varvtal, vilket kan varieras mellan 0-10 varv/min. Den normaliserade statiska förstärkningen vid olika arbetspunkter visas i fig. 5.4.2.

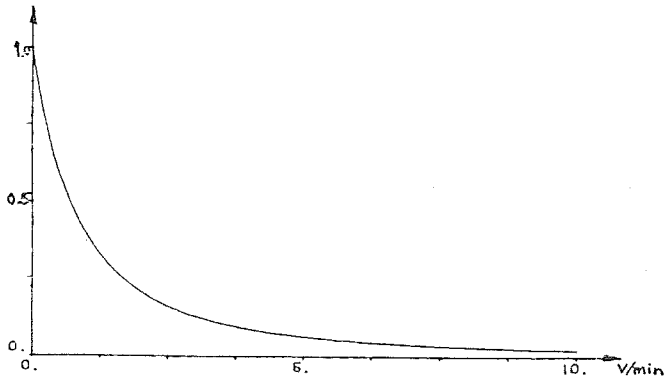


Fig. 5.4.2 Statiska förstärkningen hos entalpiväxlaren.

Uppenbarligen är processen mycket olinjär.
Entalpiväxlaren hade vid tidpunkten för vårt experiment varit i drift under ca 10 års tid vilket medförde vissa defekter i skivan.
Den uppvisade ett onormalt stort och dessutom ojämnt fördelat tryckfall för olika sektorer.Detta ger upphov till svängningar i luftflödet då skivan roterar,vilket i sin tur medför förstärkningsvariationer i takt med svängningarna i luftflödet.
Processen är också olinjär och tidsvariabel i den meningen att svängningens period beror av styrsignalens storlek, dvs skivans varvtal.

Processens tidsfördröjning och tidskonstant uppskattades ur stegsvaret i fig. 5.4.3.

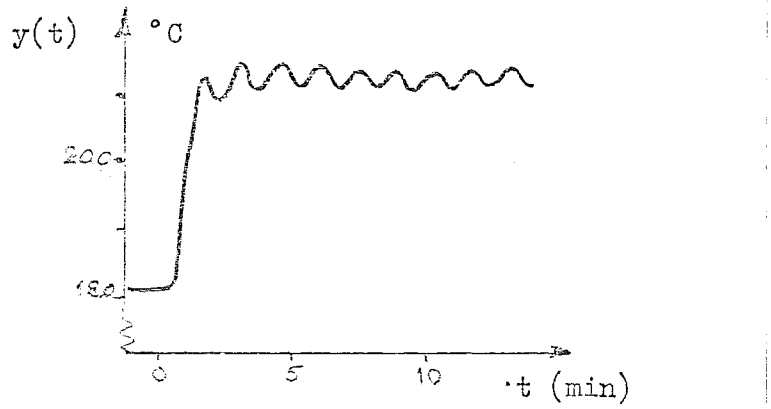


Fig. 5.4.3 Stegsvar för entalpiväxlaren.

En grov approximation av processen kan vara ett 1:a ordningens system med tidsfördröjning, $\tau \approx 20$ s.
Utstyrningsområdet, som bestämmes av temperaturen i till- respektive från-luften, var vid experimentet begränsat till 18-21 grader C.

Befintligt reglersystem

För att reglera temperaturen i till-luften används ett reglersysytem som i huvudsak består av följande komponenter:

-PI-regulator med insticksgivare

-Ställmotor med potentiometer

-Tyristorstyrning

-Likströmsmotor

Reglersystemets uppbyggnad är visat i fig. 5.4.4

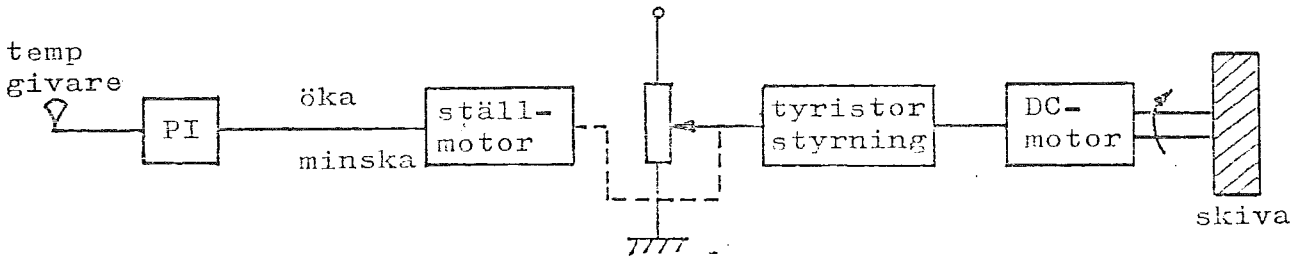


Fig. 5.4.4 Befintligt reglersystem.

PI-regulatorn ger vid börvärdesavvikelse en öka/minska signal till ställmotorn. Denna ställmotor är kopplad till en vridpotentiometer för likströmsmotorns tyristorstyrning. Likströmsmotorn driver skivan via en remtransmission som växlar ner motorns varvtal.
För att prova den befintliga regulatorn registrerades temperaturen under en stegändring av börvärdet.

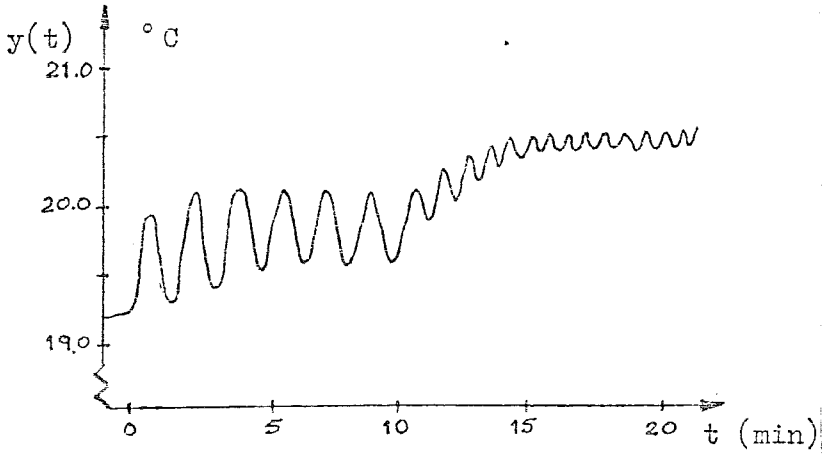


Fig. 5.4.5 Reglering med befintligt reglersystem.

Regulatorn reglerar på medelvärdet av temperaturen. Vid börvärdesinställningen 19.8 grader C regleras temperaturen till 19.8 ± 0.3 grader C.
Börvärdet ändras från 19.8 till 20.4 grader C varvid temperaturen regleras till 20.4 ± 0.1 grader C.

Försöksuppställning

För att möjliggöra styrning med LSI-11 behövdes ett interface mellan datorns DA-utgång och ställmotorn. Detta kostruerades och byggdes av Lennart Svensson vid AF Enegikonsult, som också konstruerade en förstärkarenhet till temperaturgivaren. Denna utgjordes av ett halvledarelement. Uppställningen kompletterades med ett församplingsfilter.

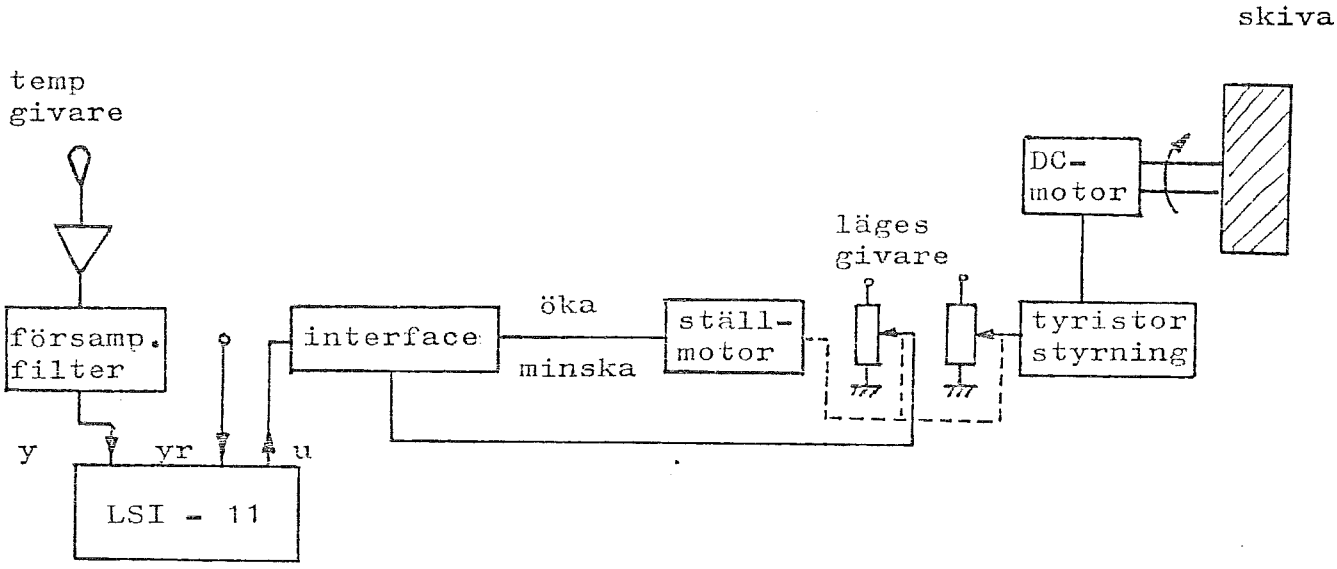


Fig. 5.4.6 Försöksuppställningen vid entalpiväxlaren.

Systemet har till följd av interfacet en liten dödzon, uppmätt till 0.05 V.
Då utstyrningsområdet var relativt litet ca 18-21 grader C gjordes en uppskalning av utsignalen för att bättre utnyttja AD-omvandlarens utstyrningsområde.

Försök

Experimenten utfördes endast med algoritmen Regul 2 då denna bedömdes vara lämpligast för denna process.
På grundval av processens stegsvar (jfr fig. 5.4.2) valdes en samplingsperiod på 20 s samt en extra tidsfördröjning.
För att undvika invikning av frekvenser högre än halva samplingsfrekvensen användes ett 2:a ordningens Butterworthfilter.
Fig. 5.4.7 visar ett försök med följande regulatorparametrar:

k=2 , estk=2
uderv=0.02 V/s , deadzon=0.05
lam=0.6

Då processen är starkt olinjär har vi valt ett litet λ-värde för att erhålla en snabbare adaptering av regulatorn vid olika arbetspunkter. Med andra ord tillåter vi regulatorn att glömma tidigare information för att den snabbt ska kunna skatta nya parametrar.
Derivatabegräsningen, uderv, är satt till 0.02 V/s vilket medför att maximala ändringen av styrsignalen under ett samplingsintervall är 0.4 V. Detta ger en mjuk uppstart.
Regulatorn har ställt in sig efter ca 30 sampel och håller då referensvärdet, 19.6 grader C, med en största avvikelse på 0.15 grader C.
Efter ca 50 sampel inkommer en störning vilken regleras ut på ca 10 sampel.
50 sampel senare görs en referensvärdesändring från 19.6 till 20.0 grader C. Detta medför att skivan roterar snabbare varför regleravvikelsen minskar till ± 0.10 grader C.
Efter ytterligare 40 sampel görs ännu en referensvärdesändring till 20.4 grader C vilket ger avvikelsen ± 0.08 grader C.

Det bör här observeras att svängningarna i utsignalen är helt beroende av styrsignalens storlek dvs skivans rotationshastighet.
Tidigare försök på entalpiväxlaren har gjorts av L.Jensen, se ref.{5}.

Erfarenheter

Ovan beskrivna försök visar att det ibland är praktiskt möjligt att med självinställare reglera en starkt olinjär process genom att välja ett lågt värde på λ. Det låga värdet på λ gör att parametrarna snabbt adapterar sig vilket också medför att utsignalen snabbt följer en ändring av referensvärdet.
Begränsningen av styrsignalens derivata, uderv, har här visat sig fungera bra och regulatorns inställningstid, från uppstart, påverkas inte nämnvärt.

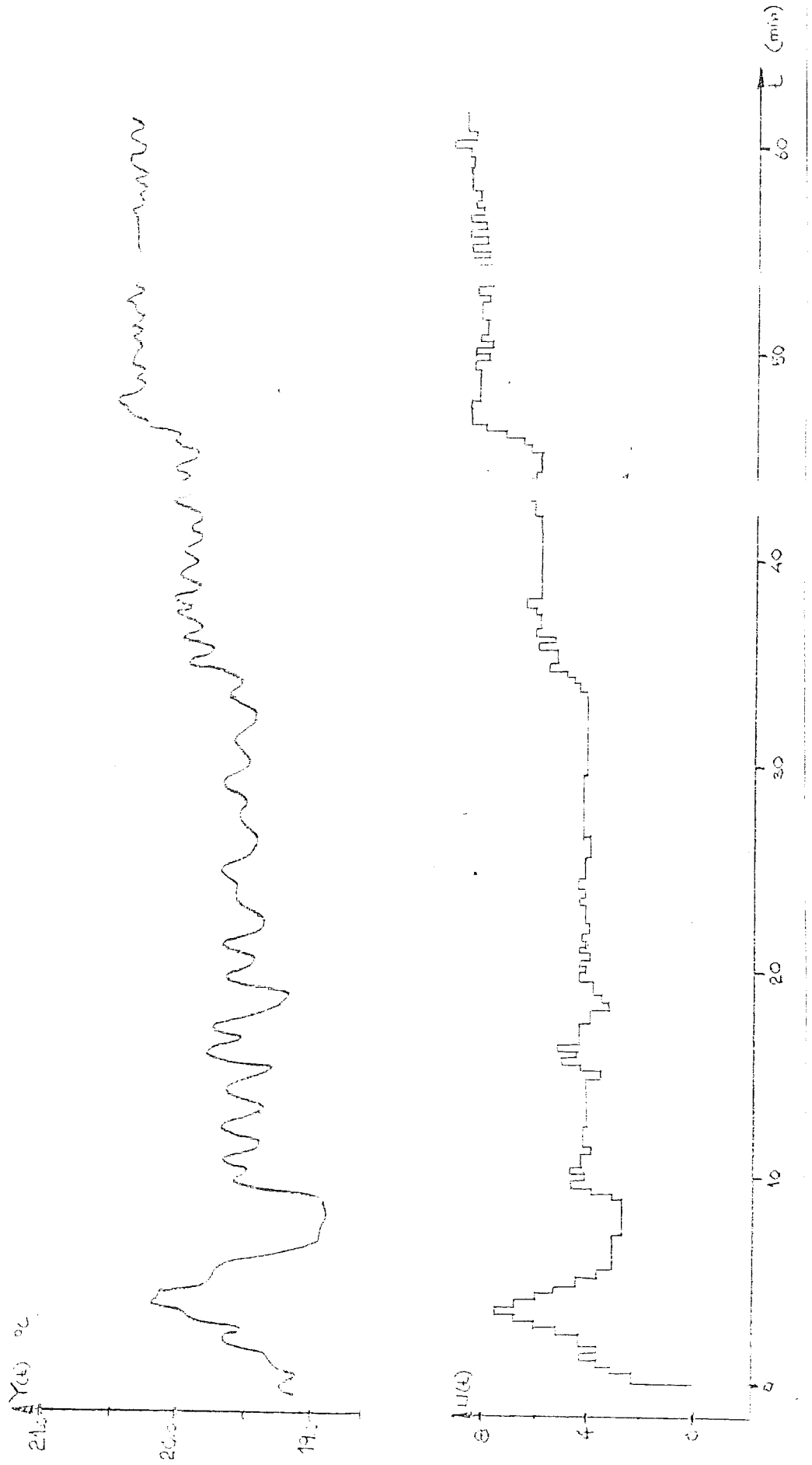


Fig. 5.4.7 Regul 2, entalpiväxlare.

6. AVSLUTNING

Regleralgoritmerna Regul 1 och Regul 2 har utprovats vid försök utförda på två av institutionens labprocesser samt med en luftvärmeväxlare för värmeåtervinning, en så kallad entalpiväxlare vid AF Energikonsult, Malmö.

Regulatorn Regul 1 är designad för att samtidigt lösa regulator- och servoproblemet. Regulatorn tillåter ingen extra tidsfördröjning varför valet av samplingsperiod måste inkludera processens eventuella tidsfördröjning. Är processens tidsfördröjning, τ, liten i förhållande till tidskonstanten, T, bestäms samplingsperioden väsentligen av T och systemets snabbhet påverkas ej nämnvärt. Problem uppstår då processer som har en tidsfördröjning i samma storleksordning som tidskonstanten, ska styras. Så är fallet i den ovan beskrivna saltprocessen där τ =13s och T=20s. Detta ger en relativt lång samplingstid, vilket kan accepteras då måttliga krav på systemets snabbhet finns.

Gradtalen på A- och B-polynomen jämte val av samplingsperiod och glömskefaktorer utgör de viktigaste designparametrarna. Skattningen av bias-parametern, d, görs med en separat glömskefaktor, lambdaD.
Erfarenheterna från försöken visar att lambda och lambdaD bör väljas lika i uppstarten för att parametrarna ska konvergera mot relevanta värden. Är processen av sådan typ att bias-nivån förändras långsammare än övriga processparametrar visar det sig fördelaktigt att efter uppstart använda ett lägre värde på lambda än på lambdaD.

Regulatorn Regul 2 realiserar en algoritm som tillåter ett antal extra tidsfördröjningar i processen och samtidigt tar hand om bias-nivå. Algoritmen är baserad på en differensbildad processmodell, och bias-störningar kan som tidigare nämnts, ses som ett extra tillstånd i en integrator i processen.

Gradtalet på A-polynomet kan, liksom i Regul 1, väljas fritt medan B-polynomet alltid består av endast en parameter. Då vi här ej skattar någon speciell bias-parameter används endast en glömskefaktor, lambda, vilket är en fördel då man eftersträvar att ha så få designparametrar som möjligt.

Processens tidsfördröjning anges som ett antal multiplar av samplingsperioden. Detta ger möjligheten att välja en liten samplingsperiod och därmed öka systemets snabbhet. Det har i försöken visat sig fördelaktigt att välja samplingstid och antalet fördröjningar så att samplingen ej sker precis i början av stegsvaret utan en bit upp på detsamma.

Processer där varken störningar eller referensvärdesändringar är ofta förekommande, lämnar en begränsad mängd information till estimeringen. Man bör därför vara observant på att antalet parametrar som skattas av algoritmen inte blir för stort. T.ex. för ett system av 1:a ordningen utan tidsfördröjning skattar Regul 1, 4 st parametrar medan Regul 2 endast skattar 2 st.

Då estimeringen av parametrarna i Regul 2 görs med differensbildade processvariabler kan känsligheten för högfrekventa störningar öka. Detta gör att kraven på församplingsfiltret också ökar. Ett 2:a ordningens filter visade sig här nödvändigt medan algoritmen Regul 1 endast behövde ett filter av 1:a ordningen.

Är processens utstyrningsområde litet, utnyttjas AD-omvandlarnas upplösning dåligt vilket i sin tur kan medföra en dålig reglering. I förekommande fall bör därför en uppskalning av processens utsignal göras.

Experimenten visar att det är praktiskt möjligt att använda självinställande regulatorer, ibland också på olinjära processer. Detta kan möjliggöras genom användning av en liten glömskefaktor vilken medger en snabb adaptering av regulatorns parametrar vid en ändring av arbetspunkt. Regulatorn kommer således att arbeta med en linjäriserad modell kring sin aktuella arbetspunkt.

7. REFERENSER

(1) Aström and Wittenmark 'On Selftuning Regulators' (1973), Automatica 9, 185-199.

(2) H.Elmqvist and S.E.Mattsson 'A realtime Kernel for Pascal' (1981), Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

(3) T.Söderström 'Kompendium i processidentifiering' Institute of Technology, Uppsala.

(4) K.J.Aström 'Simpel Self-Tuners I' (1979), Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.
CODEN:LUTFD2/(TFRT-7184)/1-063/(1979).

(5) L.Jensen 'Digital Reglering av Klimatprocesser' (1978), Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.
CODEN: LUTFD2/(TFRT-1064)/1-263/(1978).

(6) Aström, Borisson, Ljung and Wittenmark 'Theory and Applications of Self-Tuning Regulators', (1977), Automatica 13, 457-476.

APPENDIX A

Programpaketet Regul 1 består av följande filer:

HEAD1 - externdeklarerade procedurer från Kernel samt globala variabler och procedurer

MONIT - gemensam listhanterare för Regul 1 och Regul 2

REGUL1 - regleralgoritm av minimalvarians-typ, samt intern avbrottsklocka

ESTIM1 - estimatordel, LS-metoden

OPCOM1 - operatörskommunikation

MAIN - huvudprogram

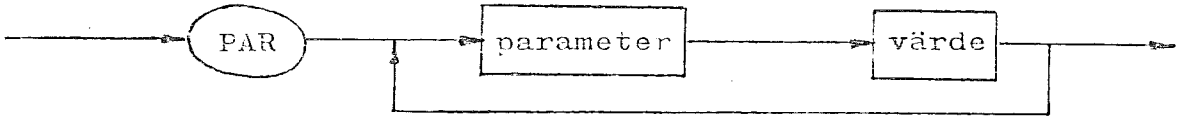
Regul 1 och Regul 2 använder sig av samma listhanterare varför den finns med enbart i appendix A.

Nedan följer syntaxerna för operatörskommunikationens kommandon och en kort förklaring till dessa. Operatörskommunikationen svarar med en hake, >, när den är redo att ta emot ett nytt kommando.

OPEN[1-10]
Anger att operatören vill arbeta med den nod som anges. Skulle noden befinna sig i tillståndet killed, överförs den till idle-listan och parametrarna ges default-värden. Datorn svarar med aktuell nods nummer följt av en hake.

Följande kommandon, fram till CLOSE, kräver att OPEN-kommando har givits.

PAR

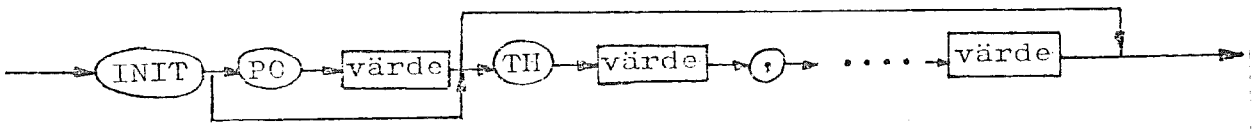


Anger att parametrarna skall uppdateras med angivna värden. De nya parametrarna används ej förrän CLOSE-kommando ges. Förteckning över parametrarna finns nedan.

MODE[1-3]
Anger vilken mode regulatorn skall arbeta i.

av parametrarr avstängd
2 - Tuning, reglering med fixa parametrar men estimering fortgår i bakgrunden.
3 - Selftuning, reglering med de aktuella estimerade parametrarna.

INIT



Anger att kovariansmatrisen skall återinitieras med angivet PO*I eller med redan befintligt PO samt att parametrarna får angivna värden.

KILL[1-10]
Anger att angiven nod ej längre ska finnas med i systemet.

CLOSE
Anger att bearbetning av noden är avslutad. De uppdaterade parametrarna används i regleringen.

START[1-10]
Angiven nod överförs från tillståndet idle till active och börjar reglera.

STOP[1-10]
Angiven nod överförs från tillståndet active till idle och upphör därmed med sin reglering.

DISP[0-10]
Angiven nods parametrar och modell skrivs ut . Anges 0 som argument skrivs en lista ut där varje nods aktuella tillstånd anges. Har OPEN-kommando givits kommer den nod, som är under bearbetning, att få sina mest aktuella parametrar utskrivna oavsett om något argument anges eller ej.

Observera att STOP och DISP-kommandona kan ges oavsett om någon nod är under bearbetning eller ej.

Nedan visas hur utskriften efter kommandot DISP 1 ser ut. De parametrar vilka står under rubriken, PARAMETERS, går alla att ändra efter kommandot PAR. Speciellt ska nämnas att vid en ändring av NA, NB eller TSAMP, vilket förändrar hela modellen, sker i princip en återstart.

```
            REGULATOR 1
            -----------

tsamp=  1.00  uhi=    10.00  ulo=  -10.00
uderv= 20.00  deadzon= 0.10
na=     2     nb=      1
yrchan= 0     ychan=   1     uchan=  1
lam=    0.980 lamD=    0.980 Po=    100.0


REGULATORMODEL       mode - selftuning
Alfa   :  0.0000   0.0000
Beta   :  1.0000
Gamma  :  0.0000   0.0000
D      :  0.0000


DO YOU WANT THE P-MATRIX? (Y/N)
>
```

Fig. 1 Utskrift efter DISP-kommando i Regul 1.

Parametrar

TSAMP - samplingsintervall, anges i sekunder

UHI,ULO - högsta respektive lägsta gräns på styrsignalen

UDERV - derivatabegränsning på styrsignalen, anges i V/s

DEADZON - dödzonsbegränsning på styrsignalen, anges i V

NA,NB - gradtal på modellens A och B-polynom, NA≤3, NB≤2

YRCHAN,YCHAN,UCHAN - in- och utgångar för respektive signal

LAM,LAMD - glömskefaktorer där LAMD hänför sig till bias-parametern

PO - kovariansmatrisen initieras med PO*I

```
program Regul1;

{*****************************************************************
*                                                                *
*     Selftuning Regulator with minimum-variance                 *
*     control.                                                   *
*     The estimator uses the LS-method.                          *
*                                                                *
*                                                                *
*                                                                *
*     Author: Lars Baath                                         *
*             Per-Olof Malmqvist                                 *
*                                                                *
*                                                                *
*                                                                *
*     Date:  November 5, 1981                                    *
*                                                                *
*                                                                *
*****************************************************************}


const sizekerneldata = 25 ;
      sizeterminaldata = 262;
      maxpriority = 1000;
      tick = 1; sec = 50; min = 3000;

type unsignedinteger = 0..65535;
     semaphore = unsignedinteger;
     event = unsignedinteger;
     deviceregister =
       record
         status: unsignedinteger;
         buff: char;
       end;
     messageref=^message;
     mailbox = unsignedinteger;

var  kerneldata: array[1..sizekerneldata] of unsignedinteger;
     terminaldata: array[1..sizeterminaldata] of integer;

procedure initkernel(memreq: unsignedinteger); external;
procedure createprocess(procedure proced;
          memreq: unsignedinteger); external;
procedure setpriority(priority: integer); external;
procedure initsem(var sem: semaphore; initval: integer); external;
procedure wait(sem: semaphore); external;
procedure signal(sem: semaphore); external;
procedure initevent(var e: event; sem: semaphore); external;
procedure await(e: event); external;
procedure cause(e: event); external;
procedure waitio(vecaddr: unsignedinteger;
          var statusreg: integer); external;
procedure waittime(t: integer); external;
procedure daout(chan:integer; value:real); external;
```

```
Function  adin(chan:integer):real; external;
procedure initmailbox(var box: mailbox); external;
procedure sendmessage(box: mailbox;
                      var mess: messageref); external;
procedure receivemessage(box: mailbox;
                         var mess: messageref); external;
procedure attachterminal(term: integer); external;
procedure initio; external;

{-------------GLOBAL DECLARATIONS-------------------------}

const     n=9;            { n-1 >= 2na+nb }
          maxnoofreg=10;


type      modetype=(fixreg,tuning,selftuning)    ;
            regspectype=record
              tsamp,uhi,ulo,uderv,deadzon,
              lam,lam0,po                    :  real;
              na,nb,npar,ychan,
              yrchan,uchan                   :  integer;
              mode                           :  modetype;
              mark                           :  boolean;
            end;

            estpartype=record
              e,yr,u            : array[0..n] of real;
              th,regth          : array[1..n] of real;
              p                 : array[1..n,1..n] of real;
              freezek           : real;
            end;

            message=record
              nextmess              :  messageref;
              idno,timer            :  integer;
              regspec               :  regspectype;
              estpar                :  estpartype;
              newpar,newstate       :  boolean;
            end;

            regstate=(killed,idle,active,running);
            statusarray=array[1..maxnoofreg] of regstate;


var
            timesync                :  semaphore;
            poolbox,estbox          :  mailbox;
```

```
{-----------------GLOBAL PROCEDURES-------------------------}


procedure initp(reg:messageref);
{ Initiates the P-matrix }

var i,j : integer;


  with reg^.regspec,reg^.estpar do
  begin
    npar:=2*na+nb+1 ;
    for i:=1 to npar do
      for j:=1 to npar do
        if i=j then p[i,j]:=po else p[i,j]:=0.0;
  end;
end;



procedure initth(reg:messageref);
{ Initiates the th-vector }

var i : integer;

begin
  with reg^.regspec,reg^.estpar do
  begin
    for i:=1 to npar do th[i]:=0;
    for i:=1 to nb do th[i+na]:=1;
    if mode = selftuning then regth:=th;
  end;
end;



procedure inituy(reg:messageref);
{ Initiates the regressors }

var i : integer;

begin
  with reg^.estpar,reg^.regspec do
  begin
    for i:=1 to na do begin e[i]:=0;yr[i]:=0 ; end ;
    for i:=1 to nb do u[i]:=0;
  end;
end;
```

```
procedure initspecif(reg:messageref);
{ Gives a new node default values }

var   i     :   integer;

begin
  with reg^.estpar;reg^.regspec do
  begin
    ts:=1.0 ; ubi:=10.0; ulo:=-10.0; uprev:=20;
    deadzon:=0.1 ; na:=2     ; nb:=1;
    yrchan:=0  ; ychan:=1 ; uchan:=1 ;
    lam:=0.98  ; lamD:=0.98; po:=100 ;
    mode:=selftuning ;

    initp(reg);
    initth(reg);
    inituy(reg);
    regth:=th ;
  end;
end;


{-------------------------------------------------------------}
```

```
{------------------  monitor  ----------------------------}

var
       regulatorlist:record
         idlepointer;firstreg     :  messageref;
         runtime          :   array[1..maxnoofreg] of integer;
         status           :   statusarray;
         mutex            :   semaphore;
         put              :   event;
       end;

{$E+}

procedure  putactivelist(var reg:messageref);
{ Puts the node into the regulatorlist, the value of the
  timer is calculated relative the preceding nodes.}

var          ptr1,ptr2     :   messageref;

begin
  with regulatorlist,reg^ do
  begin
    wait(mutex);
    timer:=round(50*regspec.tsamp)-runtime[idno];
    status[idno]:=active;
    runtime[idno]:=0;
    ptr1:=firstreg;
    if  (firstreg=nil) or (timer<firstreg^.timer)   then
    begin
      reg^.nextmess:=firstreg;
      firstreg:=reg;
    end
     else
     begin
       ptr1:=firstreg;
       while (ptr1<>nil) and (timer>=ptr1^.timer) do
       begin
         timer:=timer-ptr1^.timer;
         ptr2:=ptr1;
         ptr1:=ptr1^.nextmess;
       end;  { while }
       ptr2^.nextmess:=reg;
       nextmess:=ptr1;
      end;   { else }
      if ptr1<>nil then ptr1^.timer:=ptr1^.timer-timer;
  cause(put);
  signal(mutex);
 end;   { with }
end;    { putactivelist }
```

```
procedure getlist(var reg:messageref);
{ Decrement the first nodes timer and removes it from the
  active list if the timer =< 0.}

var     i    :  integer;

begin
  with regulatorlist do

    wait(mutex);
    reg:=nil;
    for i:=1 to maxnoofreg do
    if status[i]=running then runtime[i]:=runtime[i]+1;
    if firstreg<>nil then
    begin
      firstreg^.timer:=firstreg^.timer-1;
      if firstreg^.timer<=0 then
      begin
        reg:=firstreg;
        firstreg:=firstreg^.nextmess;
        status[reg^.idno]:=running;
      end;    {  if  }
    end;    { if nil  }
    signal(mutex);
  end;  {   with   }
end;    {   getlist  }

{$E-}

procedure putpar(reg:messageref);
{ Updates the parameters of the node called 'reg^.idno' with
  the values of 'reg'.}

var   ptr     :  messageref;

begin
  with regulatorlist,reg^ do
  begin
    wait(mutex);
    while status[idno]=running do await(put);
    case status[idno] of
    active :  ptr:=firstreg;
    idle   :  ptr:=idlepointer;
    end; { case  }
    while ptr^.idno<>idno do ptr:=ptr^.nextmess;
    if newpar then ptr^.regspec:=regspec;
    if newstate then ptr^.estpar:=estpar;
    signal(mutex);
  end;  {  with  }
end;    {  putpar  }
```

```
procedure listopen(var copyref:messageref);
{ Copies the parameters of the node called 'copyref^.idno'.
  If it's killed  the node will be placed in the idle list. }

var     ptr     :    messageref;
        id      :    integer;

begin
  with regulatorlist do
  begin
    wait(mutex);
    id:=copyref^.idno;
    while status[id]=running do await(put);
    case status[id] of
    killed : begin
               receivemessage(poolbox,ptr);
               initspecif(ptr);
               ptr^.idno:=id;
               ptr^.nextmess:=idlepointer;
               idlepointer:=ptr;
               status[id]:=idle;
             end;

    idle : begin
             ptr:=idlepointer;
             while ptr^.idno<>id do
             ptr:=ptr^.nextmess;
           end;

    active : begin
               ptr:=firstreg;
               while ptr^.idno<>id do
               ptr:=ptr^.nextmess;
             end;
    end;  {   case  }
    copyref^:=ptr^;
    signal(mutex);
  end;     {  with  }
end;   {  listopen  }



procedure getptr(nodid:integer;var firstptr,ptr1:messageref);
{ Removes referred nodes ptr either from active- or idle-list.}

var    ptr2        :        messageref;

begin
with regulatorlist do
begin
  ptr1:=firstptr;
  while (status[nodid]=running) do await(put);
  if (ptr1^.idno=nodid) then firstptr:=ptr1^.nextmess
```

```
    else
    begin
      while ptr1^.idno<>nodid do
      begin
        ptr2:=ptr1;
        ptr1:=ptr1^.nextmess;
      end; { while }
      ptr2^.nextmess:=ptr1^.nextmess;
    end;    { else }
  end;
  end;      { getptr }


  procedure getidle(nodid:integer;var reg:messageref);
  { Removes the node called 'nodid' from the idle list.}

  begin
    with regulatorlist do
    begin
      wait(mutex);
      if (status[nodid]=active) or (status[nodid]=killed)
      then reg:=nil
      else   getptr(nodid,idlepointer,reg);
      signal(mutex);
    end;  { with  }
  end;  { getidle  }


  procedure   putidle(nodid:integer);
  { Places the node called 'nodid' in the idle list.}

  var  ptr1    :   messageref;

  begin
    with regulatorlist do
    begin
      wait(mutex);
      if (status[nodid]=active) or (status[nodid]=running)
      then
      begin
        getptr(nodid,firstreg,ptr1);
        daout(ptr1^.regspec.uchan,0);
        ptr1^.nextmess:=idlepointer;
        idlepointer:=ptr1;
        status[nodid]:=idle;
      end;
      signal(mutex);
    end;    { with  }
  end;    { putidle  }
```

```
procedure killreg(nodid:integer);
{ Places the node called 'nodid' in the pool.}

var  ptr1,ptr2        :         messageref;

begin
  with regulatorlist do
  begin
    wait(mutex);
    while status[nodid]=running do await(put);
    case status[nodid] of
    active :  getptr(nodid,firstreg,ptr1);
    idle   :  getptr(nodid,idlepointer,ptr1);
    end;   {  case  }
    sendmessage(poolbox,ptr1);
    status[nodid]:=killed;
    signal(mutex);
  end;    {  with  }
end;     { killreg  }


procedure listdisp(var reg:messageref;
                   var regstatus:statusarray);
{ Copies the parameters of the wanted node and the status
  of all nodes }

var     ptr    :    messageref;
        id,i     :    integer;

begin
  with regulatorlist do
  begin
    wait(mutex);
    id:=reg^.idno;
    regstatus:=status;
    if id<>0 then
    if status[id]=killed then reg^.idno:=0
    else
    begin
      while status[id]=running do await(put);
      case status[id] of
      active :  ptr:=firstreg;
      idle   :  ptr:=idlepointer;
      end;  {  case  }
      while (ptr^.idno<>id) do ptr:=ptr^.nextmess;
      reg^:=ptr^;
    end;  {  else  }
    signal(mutex);
  end;    {  with  }
end;    {  listdisp    }
```

```
{ initialize the monitor }

var       reg                 :   messageref;
          i,maxnoofreg2       :   integer;

begin
  with regulatorlist do
  begin
    initsem(mutex,1);
    initevent(put,mutex);
    firstreg:=nil;
    idlepointer:=nil;
    for i:=1 to maxnoofreg do
    begin
      status[i]:=killed;
      runtime[i]:=0;
    end;
    maxnoofreg2:=maxnoofreg+2;
    for i:=1 to maxnoofreg2 do
    begin
      new(reg);
      sendmessage(poolbox,reg);
    end;
  end;    { with }
end;     { initregulatorlist }



procedure getlist(var reg:messageref);  external;

{-------------------------------------------------------clock-}

{$E+}

{process}
procedure clock;
{Internal interrupt to update the next sampel}
begin
  setpriority(2);
  while true do
  begin
    waittime(1);
    signal(timesync);
  end; {while}
end;    {clock}
```

```
{-----------------------------------------minvarregulator-}

{process}
procedure minvarregulator;
{ Performs minimum variance control.}

var reg1 : messageref ;
     u0  : real;
      i  : integer;

begin
  setpriority(3);
  while true do
  begin
    wait(timesync);
    getlist(reg1);
    if reg1<>nil then
    with reg1^.regspec,reg1^.estpar do
    begin
      e[0]:=adin(ychan)*10-yr[1];


      yr[0]:=adin(yrchan)*10;
      u0:=yr[0];

      if na>0 then for i:=1 to na do
           u0:=u0+regth[i]*e[i-1]+regth[na+nb+i]*yr[i];
      if nb>1 then for i:=2 to nb do
                   u0:=u0-regth[na+i]*u[i-1] ;
      u0:=(u0-regth[npar])/regth[na+1];

      { Control limitation }

      if abs(regth[na+1]*(u0-u[1]))<deadzon then u0:=u[1];

      if abs(u0-u[1])>uderv*tsamp  then
        if u0>u[1] then u0:=u[1]+ uderv*tsamp else
        u0:=u[1]-uderv*tsamp;

       if u0>uhi then u0:=uhi ;
       if u0<ulo then u0:=ulo ;


      u[0]:=u0;

      daout(uchan,u0/10);

      sendmessage(estbox,reg1);
    end; {with}
  end; {while}
end; {minvarregulator}



{-------------------------------------------------------------}
```

```
procedure putactivelist(var reg:messageref); external ;
{-------------------------------------------------estimator-}
{ process }
procedure estimator;
{ Estimates the parameters by using the LS-method.}

const  delta = 1E-6 ;

var  k,fi    :  array[1..n] of real;
     lamR,
     lambda,
     lambdaD,
     tracep,
     eps,r   :  real;
     reg     :  messageref;
     i,j     :  integer;

procedure newth(reg:messageref);
{ Updates the parameters }

begin
with reg^.regspec,reg^.estpar do
begin
  { Form the k=p*fi vector }
  r:=1  ;
  for i:=1 to npar do
  begin
    k[i]:=0;
    for j:=1 to npar do k[i]:=k[i]+p[i,j]*fi[j];
    r:=r+fi[i]*k[i];
  end;

  { Update  the parameters }
  for i:=1 to npar do th[i]:=th[i]+k[i]*eps/r;

  { Control the limitation of the P-matrix }
  tracep:=0;
  for i:=1 to npar do tracep:=tracep+p[i,i];
  tracep:=tracep-p[npar,npar];
  if tracep>((npar-1)*2*po) then lambda:=1.0 else lambda:=lam;
  if p[npar,npar]>(2*po) then lambdaD:=1.0 else lambdaD:=lamD;

  { Update the P-matrix }
  for i:=1 to npar do
    for j:=i to npar do
    begin
      lamR:=lambda;
      if j=npar then
       if i=j then lamR:=lambdaD else lamR:=sqrt(lambda*lambdaD);
      p[i,j]:=(p[i,j]-k[i]*k[j]/r)/lamR +delta ;
      p[j,i]:=p[i,j];
    end;
end;
end;
```

```
procedure update(reg:messageref);
{ Update the regression }

begin
with reg^.regspec,reg^.estpar do
 begin
  for i:=na downto 0 do yr[i+1]:=yr[i] ;
  for i:=nb downto 1 do u[i]:=u[i-1];
  for i:=n  downto 1 do e[i]:=e[i-1];
 end;
end; {update}

begin
  setpriority(4);
  while true do
  begin
   receivemessage(estbox,reg);
   with reg^.regspec,reg^.estpar do
   begin

    { Create the fi-vector }
    for i:=1 to na  do begin
                         fi[na+nb+i]:=-yr[i+1];
                         fi[i]:=-e[i];
                       end;
    for i:=1 to nb do fi[na+i]:=u[i];
    fi[npar]:=1;

    { Form the prediction error }
    eps:=e[0]+yr[1];
    for i:=1 to npar do eps:=eps-regth[i]*fi[i];

    case mode of

    fixreg    : begin
                 regth[npar]:=regth[npar]+freezek*eps;
                 th[npar]:=regth[npar];
                end;
    tuning    : begin
                 regth[npar]:=regth[npar]+freezek*eps;
                 newth(reg);
                end;
   selftuning: begin
                 newth(reg);
                 freezek:=k[npar]/r;
                 regth:=th;
                end;
    end; {case}

    update(reg);
    putactivelist(reg);
  end; {with}
 end; { while}
end;   {estimator}
```

```
{-------------------------------------------------------opcom-}

{process}
procedure opcom;
{ Handles the communication with the operator.}

label 999;

const idlength = 8;
      blanks   = '        ';

                 xlix,startx,stopx,lastopx);
      pars = (tsampx,uhix,ulox,udervx,deadzonx,nax,nbx,
              yrchanx,uchanx,ychanx,lamx,lamDx,pox,lastparx);
      identtype = array[1..idlength] of char;

      errors = (syntax,toolongid,noname,alreadyopen,notopen,
                maxpar,nostart,noclose,wrongid);

var   opname     :   array[ops] of identtype ;
      parname    :   array[pars] of identtype ;
      opx        :   ops;
      ch         :   char;
      identifier:    identtype ;
      opened     :   boolean;
      nodid,i,j  :   integer;
      dispref,
      copyref    :   messageref;


procedure error(err:errors);
{ Writes an errormessage and jumps to the end of the loop
  of opcom main.}

begin
  case err of
  syntax       : writeln('syntaxerror,write : OPEN 2,NA 1,TH 5,7,2');
  toolongid    : writeln('command to long,max 8 letters');
  noname       : writeln('illegal command');
  alreadyopen  : writeln('regulator is already opened');
  notopen      : writeln('regulator is not opened');
  noclose      : writeln('the regulator must be closed');
  wrongid      : writeln('must be a number between 1 and 10');
  nostart      : writeln('already started or not created');
  maxpar       : writeln('na+nb out of limits ' );
  end;
  goto 999;
end;
```

```
procedure skipblanks;
begin
  while (not eoln) and (ch=' ') do read(ch);
end;


procedure getident;
{ Reads an identifier in to the identifier-vector }
begin
  identifier:=blanks;
  skipblanks;
  i:=1;
  while (ch>='A')and(ch<='Z') do
  begin
    if i>idlength then error(toolongid);
    identifier[i]:=ch;
    i:=i+1;
    if eoln then ch:=' ' else read(ch);
  end; {while}
end;


procedure getint(var int:integer);
{ Reads an integer }

begin
  if (ch<>' ') and (ch<>',') then error(syntax);
  read(int);
  skipblanks;
end;


procedure getreal(var  r:real);
{ Reads a real }

begin
  if (ch<>' ') and (ch<>',') then error(syntax);
  read(r);
  skipblanks;
end;


procedure open;
{ Handles the command:OPEN no }

begin
  if opened then error(alreadyopen);
  getint(nodid);
  if (nodid>10) or (nodid<1) then error(wrongid);
  copyref^.idno:=nodid;
  listopen(copyref);
  opened:=true;
  copyref^.newpar:=false;
  copyref^.newstate:=false;
end;
```

```
 procedure par;
{ Handles the command:PAR parameter value ... }
var     param   : pars ;
        int     : integer ;

begin
  if not opened then error(notopen);
  repeat
  getident;
  parname[lastparx]:=identifier;
  param:=tsampx;
  while parname[param]<>identifier do param:=succ(param);
  with copyref^.regspec,copyref^.estpar do
  case param of
  tsampx      : begin getreal(tsamp);
                      initp(copyref);
                      initth(copyref);
                      inituy(copyref);
                      copyref^.newstate:=true;
                end;
  uhix        : getreal(uhi);
  ulox        : getreal(ulo);
  udervx      : getreal(uderv);
  deadzonx    : getreal(deadzon);
  nax         : begin getint(int);
                      if (2*int+nb)>(n-1) then error(maxpar)
                         else na:=int ;
                      initp(copyref);
                      initth(copyref);
                      inituy(copyref);
                      copyref^.newstate:=true;
                end;
   nbx        : begin getint(int);
                      if (2*na+int)>(n-1) then error(maxpar)
                         else nb:=int ;
                      initp(copyref);
                      initth(copyref);
                      inituy(copyref);
                      copyref^.newstate:=true;
                end;
  yrchanx     : getint(yrchan);
  ychanx      : getint(ychan);
  uchanx      : getint(uchan);
  lamx        : getreal(lam);
  lamDx       : getreal(lamD);
  pox         : getreal(po);
  lastparx    : error(noname);
  end;
  write(identifier,',')
  until eoln ;
  writeln('has been read');
  copyref^.newpar:=true;
end;
```

```
procedure mode;
{ Handles the command: MODE }

var imode : integer;

begin
  if not opened then error(notopen);
  writeln('(fixreg=1,tuning=2,selftuning=3)');
  getint(imode);
  with copyref^.regspec do
  case imode of
  1  :  mode:=fixreg;
  2  :  mode:=tuning;
  3  :  mode:=selftuning;
  end;
  copyref^.newpar:=true;
end;



procedure init;
{ Handles the command:INIT PO 25,TH 1,2,3,..,npar and
  initiates the P-matrix }

var     k  :  integer;

begin
  if not opened then error(notopen);
  with copyref^ do
  begin
    k:=1;getident;
    while (not eoln) and (k<=regspec.npar) do
    begin
     if identifier='PO      '
      then begin getreal(regspec.po);getident;end
      else
       if identifier='TH      ' then
       begin
         getreal(estpar.th[k]);
         k:=k+1;
       end;
     end;{while}
     estpar.regth:=estpar.th;
     newstate:=true;
     newpar:=true;
     initp(copyref);
   end;
end;
```

```
procedure close;
{ Handles the command:CLOSE, and incorporates the changed
  parameters to the right node }

begin
  if not opened then error(notopen);
  putpar(copyref);
  opened:=false;
  nodid:=0;
end;


procedure kill;
{ Handles the command:KILL,the node will be placed
  in the pool.}

begin
  if not opened then error(notopen);
  killreg(nodid);
  opened:=false;
  nodid:=0;
end;


procedure start;
{ Handles the command:START no,the .node will be placed
  in the active list.}

var reg  :  messageref;

begin
  if opened then error(noclose);
  getint(nodid);
  getidle(nodid,reg);
  if reg=nil then error(nostart);
  initp(reg);
  inituy(reg);
  if reg^.regspec.mode<>fixreg then initth(reg);
  putactivelist(reg);
  nodid:=0;
end;


procedure stop;
{ Handles the command:STOP no,the node willl be placed
  in the idle list.}

var nodid1:integer;

begin
  getint(nodid1);
  putidle(nodid1);
end;
```

```
procedure disp;
{ Handles the command DISP no }

var  nod         : integer;
     regstatus   : statusarray ;
     ch1         : char;

begin
  if opened then nod:=nodid else getint(nod);
  if (nod>10) or (nod<0) then error(wrongid);
  dispref^.idno:=nod;
  listdisp(dispref,regstatus);
  if dispref^.idno=0 then
  begin
    writeln('REGULATORS:');
    for i:=1 to maxnoofreg do
    case regstatus[i] of
    killed  : writeln(i:4,' -  killed');
    idle    : writeln(i:4,' -  idle ');
    active  : writeln(i:4,' -  active ');
    running : writeln(i:4,' -  running');
   end; {case}
  end
  else
  begin
    writeln(' ':19,'REGULATOR',nod:3);
    writeln(' ':19,'============');
    writeln('PARAMETERS');
    with dispref^.regspec,dispref^.estpar do
    begin
      write('tsamp=  '); write(tsamp:6:2,' ':5);
      write('uhi=    '); write(uhi:6:2,' ':5);
      write('ulo=    '); writeln(ulo:6:2);
      write('uderv=  '); write(uderv:6:2,' ':5);
      write('deadzon='); writeln(deadzon:6:2,' ':5);
      write('na=     '); write(na:3,' ':8);
      write('nb=     '); writeln(nb:3,' ':8);
      write('yrchan= '); write(yrchan:3,' ':8);
      write('ychan=  '); write(ychan:3,' ':8);
      write('uchan=  '); writeln(uchan:3);
      write('lam=    '); write(lam:7:3,' ':4);
      write('lamD=   '); write(lamD:7:3,' ':4);
      write('po=    ');  writeln(po:6:1);writeln;
```

```
        write('REGULATORMODELL');
        case mode of
         fixreg      : writeln(' ':3,'mode - fixreg');
         tuning      : writeln(' ':3,'mode - tuning');
         selftuning  : writeln(' ':3,'mode - selftuning');
        end; {case}
        write('Alfa : ');
        for i:=1 to na do write(regth[i]:10:4);writeln;
        write('Beta : ');
        for i:=1 to nb do write(regth[na+i]:10:4);writeln;
        write('Gamma: ');
        for i:=1 to na do write(regth[na+nb+i]:10:4);writeln;
        writeln('D    : ',regth[npar]:10:4);writeln;


        if mode=tuning then
        begin
          writeln('ESTIMATORMODELL');
          write('Alfa : ');
          for i:=1 to na do write(th[i]:10:4);writeln;
          write('Beta : ');
          for i:=1 to nb do write(th[na+i]:10:4);writeln;
          write('Gamma: ');
          for i:=1 to na do write(th[na+nb+i]:10:4);writeln;
          writeln('D    : ',th[npar]:10:4);
        end; {tuning} writeln;


        writeln('DO YOU WANT THE P-MATRIX?(Y/N) ');
        readln;
        read(ch1);
        if ch1='Y' then
        begin
          writeln('P-MATRIX');
          for i:=1 to npar do
          begin
            for j:=1 to i do write(p[i,j]:6:1,' ');writeln;
          end;
        end; {p-matrix}
      end;  {with}
    end; {regulatordisp}
  end; {disp}
```

```
procedure initnames;
{ Initiates the name of the commends and parameters.}

begin
  opname[openx]:=    'OPEN     ' ;
  opname[dispx]:=    'DISP     ';
  opname[parx] :=    'PAR      ';
  opname[modex]:=    'MODE     ';
  opname[closex]:=   'CLOSE    ';
  opname[killx]:=    'KILL     ';
  opname[initx]:=    'INIT     ';
  opname[startx]:=   'START    ';
  opname[stopx]:=    'STOP     ';
  parname[tsampx]:= 'TSAMP    ';
  parname[uhix]:=    'UHI      ';
  parname[ulox]:=    'ULO      ';
  parname[udervx]:= 'UDERV    ';
  parname[deadzonx]:='DEADZON ';
  parname[nax]:=     'NA       ';
  parname[nbx]:=     'NB       ';
  parname[yrchanx]:='YRCHAN   ';
  parname[ychanx]:= 'YCHAN    ';
  parname[uchanx]:= 'UCHAN    ';
  parname[lamx]:=    'LAM      ';
  parname[lamDx]:=   'LAMD     ';
  parname[pox]:=     'PO       ';
end;


begin  {opcom}
  setpriority(10);
  initnames;
  nodid:=0;
  receivemessage(poolbox,copyref);
  receivemessage(poolbox,dispref);
  opened:=false;
  while true do
  begin
    ch:=' ';
    if opened then write(nodid:2);
    write('>');
    getident;
    opname[lastopx]:=identifier;
    opx:=openx;
    while  opname[opx]<>identifier   do opx:=succ(opx) ;
    case opx of
     openx  :  open;
     dispx  :  disp;
     parx   :  par ;
     modex  :  mode;
     initx  :  init;
     closex :  close;
```

```
        killx   :  kill;
        startx  :  start;
        stopx   :  stop;
        lastopx:   error(noname);
      end;
 999: readln;
   end;
end;
```

```
{--------------------    main    ----------------------------}

procedure clock; external;
procedure minvarregulator; external;
procedure estimator; external;

begin
  initkernel(8000);
  initio;
  initmailbox(estbox);
  initmailbox(poolbox);
  initsem(timesync,0);
  initregulatorlist;
  createprocess(clock,200);
  createprocess(minvarregulator,500);
  createprocess(estimator,500);
  createprocess(opcom,1000);
  setpriority(maxpriority);
end.   { main }
```

APPENDIX B

Programpaketet Regul 2 består av följande filer:

HEAD2 - externdeklarerade procedurer från Kernel samt globala variabler och procedurer

MONIT - gemensam listhanterare för Regul 1 och Regul 2

REGUL2 - regleralgoritm av minimalvarians-typ samt intern avbrottsklocka

ESTIM2 - estimatordel, LS-metoden

OPCOM2 - operatörskommunikation

MAIN - huvudprogram

Regul 2 använder sig av samma listhantering som Regul 1 varför den ej finns utskriven här utan enbart i appendix A. Operatörskommunikationen skiljer sig enbart ifråga om ett antal parametrar. Vi hänvisar därför till appendix A för en beskrivning av de olika kommandon som existerar.

Vid DISP 1 i Regul 2 fås en utskrift enligt nedan.

```
          REGULATOR 1
          -----------
PARAMETERS
tsamp=   1.00  uhi=     10.00  ulo=  -10.00
uderv=  20.00  deadzon=  0.10
na=         2  k=           1  estk=     1
yrchan=     0  ychan=       1  uchan=    1
lam=    0.980  Po=      100.0

REGULATORMODELL      mode - selftuning
R :   1.0000
S :   0.0000    0.0000

DO YOU WANT THE P-MATRIX? (Y/N)
>
```

Fig. 2 Utskrift vid DISP-kommando i Regul 2

Skillnaden mellan Regul 1 och Regul 2 består av att NB ersätts med följande parametrar:

K - antalet fördröjningar i modellen, anges i antal hela samplingsperioder

ESTK - antalet differenssteg i estimeringsmodellen

```
program Regul2;

{*************************************************************
*                                                            *
*                                                            *
*       Selftuning Regulator with minimum                    *
*       variance control.                                    *
*       The estimator uses the LS-method.                    *
*                                                            *
*                                                            *
*       Author: Lars Baath                                   *
*               Per-Olof Malmqvist                           *
*                                                            *
*                                                            *
*       Date:   November 10, 1981                            *
*                                                            *
*                                                            *
*                                                            *
*                                                            *
*************************************************************}


const sizekerneldata = 25;
      sizeterminaldata = 262;
      maxpriority = 1000;
      tick = 1; sec = 50; min = 3000;


type unsignedinteger = 0..65535;
     semaphore = unsignedinteger;
     event = unsignedinteger;
     deviceregister =
       record
         status: unsignedinteger;
         buff: char;
       end;
     messageref=^message;
     mailbox = unsignedinteger;

var  kerneldata: array[1..sizekerneldata] of unsignedinteger;
     terminaldata: array[1..sizeterminaldata] of integer;

procedure initkernel(memreq: unsignedinteger); external;
procedure createprocess(procedure proced;
          memreq: unsignedinteger); external;
procedure setpriority(priority: integer); external;
procedure initsem(var sem: semaphore; initval: integer); external;
procedure wait(sem: semaphore); external;
procedure signal(sem: semaphore); external;
procedure initevent(var e: event; sem: semaphore); external;
procedure await(e: event); external;
procedure cause(e: event); external;
procedure waitio(vecaddr: unsignedinteger;
          var statusreg: integer); external;
```

```
procedure waittime(t: integer); external;
procedure daout(chan:integer; value:real); external;
Function  adin(chan:integer):real; external;
procedure initmailbox(var box: mailbox); external;
procedure sendmessage(box: mailbox;
                      var mess: messageref); external;
procedure receivemessage(box: mailbox;
                         var mess: messageref); external;
procedure attachterminal(term: integer); external;
procedure initio; external;

{------------GLOBAL DECLARARTIONS-------------------------}


const     n=9;              { max(k+estk+na,2k+estk) <= n }
          maxnoofreg=10;

type    modetype=(fixreg,tuning,selftuning)     ;
          regspectype=record
            tsamp,uhi,ulo,uderv,deadzon,
            lam,po                          :  real;
            na,npar,ychan,k,estk,
            yrchan,uchan                    :  integer;
            mode                            :  modetype;
          end;

          estpartype=record
            y,u             : array[0..n] of real;
            th,regth        : array[1..n] of real;
            p               : array[1..n,1..n] of real;
            yr              : real;
          end;

          message=record
            nextmess            :  messageref;
            idno,timer          :  integer;
            regspec             :  regspectype;
            estpar              :  estpartype;
            newpar,newstate     :  boolean;
          end;

          regstate=(killed,idle,active,running);
          statusarray=array[1..maxnoofreg] of regstate;


var
          timesync              :  semaphore;
          poolbox,estbox        :  mailbox;
```

```
{-----------------GLOBAL PROCEDURES-------------------------}

procedure initp(reg:messageref);
{ Initiates the P-matrix }

var i,j : integer;

begin
  with reg^.regspec,reg^.estpar do
  begin
    npar:=k+na;
    for i:=1 to npar do
      for j:=1 to npar do
        if i=j then p[i,j]:=po else p[i,j]:=0.0;
  end;
end;



procedure initth(reg:messageref);
{ Initiates the th-vector }

var i : integer;

begin
  with reg^.regspec,reg^.estpar do
  begin
    th[1]:=1;
    for i:=2 to npar do th[i]:=0;
    if mode = selftuning then regth:=th;
  end;
end;



procedure inituy(reg:messageref);
{ Initiates the regressors }

var i : integer;

begin
  with reg^.estpar,reg^.regspec do
    for i:=1 to n do begin y[i]:=0;u[i]:=0;yr:=0;end;
end;
```

```
  procedure initspecif(reg:messageref);
  { Gives a new node default values }
  var   i     :   integer;

  begin
    with reg^.estpar,reg^.regspec do
    begin
      tsamp:=1.0 ; uhi:=10.0; ulo:=-10.0; uderv:=20;
      deadzon:=0.1 ; na:=2     ; k:=1   ; estk:=1;
      yrchan:=0  ; ychan:=1 ; uchan:=1 ;
      lam:=0.98  ;  po:=100 ;
      mode:=selftuning ;

      initp(reg);
      initth(reg);
      inituy(reg);
      regth:=th ;
    end;
  end;


  {-------------------------------------------------------------}


  procedure getlist(var reg:messageref);  external;
  procedure putactivelist(var reg:messageref); external ;

  {-------------------------------------------------------clock-}

  {$E+}

  {process}
  procedure clock;
  { Internal interrupt to update the next sampel }

  begin
    setpriority(2);
    while true do
    begin
      waittime(1);
      signal(timesync);
    end; {while}
  end;   {clock}
```

```
{-----------------------------------------minvarregulator-}

{process}
procedure minvarregulator;
{ Performs minimum variance control.}

var reg1 : messageref ;
    u0   : real;
     i   : integer;

begin
  setpriority(3);
  while true do
  begin
    wait(timesync);
    getlist(reg1);
    if reg1<>nil then
    with reg1^.regspec,reg1^.estpar do
    begin
      y[0]:=adin(ychan)*10 ;
      yr:=adin(yrchan)*10 ;
      u0:=0;
      if k>1 then
         for i:=2 to k do u0:=u0+regth[i]*(u[i-1]-u[i+k-1]);
      for i:=1 to na do u0:=u0+regth[k+i]*(y[i-1]-y[i+k-1]);
      u0:=u[k]-(u0+y[0]-yr)/regth[1];

      { Control limitation }

      if abs(regth[1]*(u0-u[1]))<deadzon then u0:=u[1];

      if abs(u0-u[1])>uderv*tsamp  then
        if u0>u[1] then u0:=u[1]+ uderv*tsamp else
        u0:=u[1]-uderv*tsamp;

      if u0>uhi then u0:=uhi ;
      if u0<ulo then u0:=ulo ;

      u[0]:=u0 ;

      daout(uchan,u0/10);

      if mode=fixreg then putactivelist(reg1)
      else sendmessage(estbox,reg1);
    end; {with}
  end; {while}
end; {minvarregulator}


{-------------------------------------------------------------}
```

```
{-------------------------------------------------estimator-}

{$E+}

{ process }

procedure estimator;
{ Estimates the parameters by using the LS-method.}

const  delta = 1E-6 ;

var  kk,fi    :  array[1..n] of real;
     lambda,
     tracep,
     eps,r    :  real;
     reg      :  messageref;
     i,j      :  integer;


procedure newth(reg:messageref);
{ Update the parameters }

begin
with reg^.regspec,reg^.estpar do
begin
  { Form the k=P*fi vector }
  r:=1  ;
  for i:=1 to npar do
  begin
    kk[i]:=0;
    for j:=1 to npar do kk[i]:=kk[i]+p[i,j]*fi[j];
    r:=r+fi[i]*kk[i];
  end;

  { Update the parameters }
  for i:=1 to npar do th[i]:=th[i]+kk[i]*eps/r;

  { Control the limitation of the P-matrix }
  tracep:=0;
  for i:=1 to npar do tracep:=tracep+p[i,i];
  if tracep>((npar)*2*po) then lambda:=1.0 else lambda:=lam;

  { Update the P-matrix }
  for i:=1 to npar do
    for j:=i to npar do
    begin
      p[i,j]:=(p[i,j]-kk[i]*kk[j]/r)/lambda +delta ;
      p[j,i]:=p[i,j];
    end;
end;
end;
```

```
procedure update(reg:messageref);
{ Update the regressors }

begin
with reg^.estpar do
 for i:=n  downto 1 do
  begin
    y[i]:=y[i-1];
    u[i]:=u[i-1];
  end;
end; {update}



begin
  setpriority(4);
  while true do
  begin
   receivemessage(estbox,reg);
   with reg^.regspec,reg^.estpar do
   begin

    { Create the fi-vector }
    for i:=1 to k  do
        fi[i]:=u[i+k-1]-u[i+k-1+estk];
    for i:=1 to na do
        fi[i+k]:=y[i+k-1]-y[i+k-1+estk];


    { Form the prediction error }
    eps:=y[0]-y[estk] ;
    for i:=1 to npar do eps:=eps-regth[i]*fi[i];


    newth(reg);
    if mode=selftuning then regth:=th ;


    update(reg);
    putactivelist(reg);
  end; {with}
 end; { while}
end;   {estimator}
```

```
{-------------------------------------------------------opcom->

{process}
procedure opcom;
{ Handles the communication with the operator.}

label 999;

const idlength = 8;
      blanks   = '        ';

type  ops  = (openx,dispx,parx,modex,initx,closex,
              killx,startx,stopx,lastopx);
      pars = (tsampx,uhix,ulox,udervx,deadzonx,nax,kx,estkx,
              yrchanx,uchanx,ychanx,lamx,pox,lastparx);
      identtype = array[1..idlength] of char;

      errors = (syntax,toolongid,noname,alreadyopen,notopen,
                maxpar,nostart,noclose,wrongid);

var   opname     :   array[ops] of identtype ;
      parname    :   array[pars] of identtype ;
      opx        :   ops;
      ch         :   char;
      identifier:    identtype ;
      opened     :   boolean;
      nodid;i...  :  integer;
      dispref;
      copyref    :   messageref;


procedure error(err:errors);
{ Writes an errormessage and jumps to the end of the loop
  of opcom main.}

begin
  case err of
  syntax       : writeln('syntaxerror,write : OPEN 2,NA 1,TH 5,7,2');
  toolongid    : writeln('command to long,max 8 letters');
  noname       : writeln('illegal command');
  alreadyopen  : writeln('regulator is already opened');
  notopen      : writeln('regulator is not opened');
  noclose      : writeln('the regulator must be closed');
  wrongid      : writeln('must be a number between 1 and 10');
  nostart      : writeln('already started or not created');
  maxpar       : writeln('na+k+estk out of limits ' );
  end;
  goto 999;
end;
```

```
procedure skipblanks;
begin
  while (not eoln) and (ch=' ') do read(ch);
end;


procedure getident;
{ Reads an identifier into the identifier-vector.}

begin
  identifier:=blanks;
  skipblanks;
  i:=1;
  while (ch>='A')and(ch<='Z') do
  begin
    if i>idleng : then error(toolong);
    identifier[i]:=ch;
    i:=i+1;
    if eoln then ch:=' ' else read(ch);
  end; {while}
end;



procedure getint(var int:integer);
{ Reads an integer.}

begin
  if (ch<>' ') and (ch<>',') then error(syntax);
  read(int);
  skipblanks;
end;



procedure getreal(var  r:real);
{ Reads a real.}

begin
  if (ch<>' ') and (ch<>',') then error(syntax);
  read(r);
  skipblanks;
end;
```

```
procedure open;
{ Handles the command:OPEN no.}

begin
  if opened then error(alreadyopen);
  getint(nodid);
  if (nodid>10) or (nodid<1) then error(wrongid);
  copyref^.idno:=nodid;
  listopen(copyref);
  opened:=true;
  copyref^.newpar:=false;
  copyref^.newstate:=false;
end;


 procedure par;
{ Handles the command: PAR parameter value ... }

var     param   : pars ;
        int     : integer ;

begin
  if not opened then error(notopen);
  repeat
  getident;
  parname[lastparx]:=identifier;
  param:=tsampx;
  while parname[param]<>identifier do param:=succ(param);
  with copyref^.regspec,copyref^.estpar do
  case param of
  tsampx      : begin getreal(tsamp);
                      initp(copyref);
                      initth(copyref);
                      inituy(copyref);
                      copyref^.newstate:=true;
                end;
  uhix        : getreal(uhi);
  ulox        : getreal(ulo);
  udervx      : getreal(uderv);
  deadzonx    : getreal(deadzon);
  nax         : begin getint(int);
                      if (int+k+estk)>n then error(maxpar)
                         else na:=int ;
                      initp(copyref);
                      initth(copyref);
                      inituy(copyref);
                      copyref^.newstate:=true;
                end;
```

```
    kx          : begin getint(int);
                        if (na+int+estk)>n then error(maxpar)
                           else k:=int;
                        initp(copyref);
                        initth(copyref);
                        inituy(copyref);
                        copyref^.newstate:=true;
                  end;
   estkx        : begin getint(int);
                        if (na+k+int)>n then error(maxpar)
                           else estk:=int;
                  end;
   yrchanx      : getint(yrchan);
   ychanx       : getint(ychan);
   uchanx       : getint(uchan);
   lamx         : getreal(lam);
   pox          : getreal(po);
   lastparx     : error(noname);
   end;
   write(identifier,',')
   until eoln ;
   writeln('has been read');
   copyref^.newpar:=true;
end;


procedure mode;
{ Handles the command:MODE.}

var imode : integer;

begin
   if not opened then error(notopen);
   writeln('(fixreg=1,tuning=2,selftuning=3)');
   getint(imode);
   with copyref^.regspec do
   case imode of
   1   :  mode:=fixreg;
   2   :  mode:=tuning;
   3   :  mode:=selftuning;
   end;
   copyref^.newpar:=true;
end;
```

```
procedure init;
{ Handles the command:INIT PO 25,TH 1,2,3,...,npar, and
  initiates the P-matrix.}

var     l  :  integer;

begin
  if not opened then error(notopen);
  with copyref^ do
  begin
    l:=1;getident;
    while (not eoln) and (l<=regspec.npar) do
    begin
     if identifier='PO      '
      then begin getreal(regspec.po);getident;end
      else
       if identifier='TH      ' then
       begin
         getreal(estpar.th[l]);
         l:=l+1;
       end;
    end;{while}
    estpar.regth:=estpar.th;
    newstate:=true;
    newpar:=true;
    initp(copyref);
  end;
end;


procedure close;
{ Handles the command:CLOSE, and incorporates the changed
  parameters into the right node.}

begin
  if not opened then error(notopen);
  putpar(copyref);
  opened:=false;
  nodid:=0;
end;



procedure kill;
{ Handles the command:KILL, the node will be placed
  in the pool.}

begin
  if not opened then error(notopen);
  killreg(nodid);
  opened:=false;
  nodid:=0;
end;
```

```
procedure start;
{ Handles the command:START no, the node will be placed
  in the active list.}

var reg  :  messageref;

begin
  if opened then error(noclose);
  getint(nodid);
  getidle(nodid,reg);
  if reg=nil then error(nostart);
  initp(reg);
  inituy(reg);
  if reg^.regspec.mode<>fixreg then initth(reg);
  putactivelist(reg);
  nodid:=0;
end;


procedure stop;
{ Handles the command STOP no, the node will be placed
  in the idle list.}

var nodid1:integer;

begin
  getint(nodid1);
  putidle(nodid1);
end;


procedure disp;
{ Handles the command:DISP no.}

var  nod        : integer;
     regstatus  : statusarray ;
     ch1        : char;

begin
  if opened then nod:=nodid else getint(nod);
  if (nod>10) or (nod<0) then error(wrongid);
  dispref^.idno:=nod;
  listdisp(dispref,regstatus);
  if dispref^.idno=0 then
  begin
    writeln('REGULATORS:');
    for i:=1 to maxnoofreg do
    case regstatus[i] of
    killed  : writeln(i:4,' -  killed');
    idle    : writeln(i:4,' -  idle ');
    active  : writeln(i:4,' -  active ');
    running : writeln(i:4,' -  running');
   end; {case}
  end
  else
```

```
    begin
      writeln(' ':19,'REGULATOR',mod:3);
      writeln(' ':19,'============');
      writeln('PARAMETERS');
      with dispref^.regspec,dispref^.estpar do
      begin
        write('tsamp=  '); write(tsamp:6:2,' ':5);
        write('uhi=    '); write(uhi:6:2,' ':5);
        write('ulo=    '); writeln(ulo:6:2);
        write('uderv=  '); write(uderv:6:2,' ':5);
        write('deadzon='); writeln(deadzon:6:2);
        write('na=     '); write(na:3,' ':8);
        write('k=      '); write(k:3,' ':8);
        write('estk=   '); writeln(estk:3);
        write('yrchan= '); write(yrchan:3,' ':8);
        write('ychan=  '); write(ychan:3,' ':8);
        write('uchan=  '); writeln(uchan:3);
        write('lam=    '); write(lam:7:3,' ':4);
        write('po=    ');  writeln(po:6:1);writeln;

        write('REGULATORMODELL');
        case mode of
         fixreg     : writeln(' ':3,'mode - fixreg');
         tuning     : writeln(' ':3,'mode - tuning');
         selftuning : writeln(' ':3,'mode - selftuning');
        end; {case}
        write('R : ');
        for i:=1 to k do write(regth[i]:10:4);writeln;
        write('S : ');
        for i:=1 to na do write(regth[k+i]:10:4);writeln;
        writeln;

        if mode=tuning then
        begin
          writeln('ESTIMATORMODELL');
          write('R : ');
          for i:=1 to k do write(th[i]:10:4);writeln;
          write('S : ');
          for i:=1 to na do write(th[k+i]:10:4);writeln;
        end; {tuning} writeln;

        writeln('DO YOU WANT THE P-MATRIX?(Y/N) ');
        readln;
        read(ch1);
        if ch1='Y' then
        begin
          writeln('P-MATRIX');
          for i:=1 to npar do
          begin
            for j:=1 to i do write(p[i,j]:6:1,' ');writeln;
          end;
        end; {p-matrix}
      end;  {with}
    end; {regulatordisp}
end; {disp}
```

```
procedure initnames;
{ Initiates the name of the commands and parameters.}

begin
  opname[openx]:=    'OPEN     ' ;
  opname[dispx]:=    'DISP     ';
  opname[parx] :=    'PAR      ';
  opname[modex]:=    'MODE     ';
  opname[closex]:=   'CLOSE    ';
  opname[killx]:=    'KILL     ';
  opname[initx]:=    'INIT     ';
  opname[startx]:=   'START    ';
  opname[stopx]:=    'STOP     ';
  parname[tsampx]:=  'TSAMP    ';
  parname[uhix]:=    'UHI      ';
  parname[ulox]:=    'ULO      ';
  parname[udervx]:=  'UDERV    ';
  parname[deadzonx]:='DEADZON  ';
  parname[nax]:=     'NA       ';
  parname[kx]:=      'K        ';
  parname[estkx]:=   'ESTK     ';
  parname[yrchanx]:='YRCHAN   ';
  parname[ychanx]:=  'YCHAN    ';
  parname[uchanx]:=  'UCHAN    ';
  parname[lamx]:=    'LAM      ';
  parname[pox]:=     'PO       ';
end;


begin  {opcom}
  setpriority(10);
  initnames;
  nodid:=0;
  receivemessage(poolbox,copyref);
  receivemessage(poolbox,dispref);
  opened:=false;
  while true do
  begin
    ch:=' ';
    if opened then write(nodid:2);
    write('>');
    getident;
    opname[lastopx]:=identifier;
    opx:=openx;
    While  opname[opx]<>identifier   do opx:=succ(opx) ;
    case opx of
     openx   :  open;
     dispx   :  disp;
     parx    :  par ;
```

```
      modex   :  mode;
      initx   :  init;
      closex  :  close;
      killx   :  kill;
      startx  :  start;
      stopx   :  stop;
      lastopx:   error(noname);
    end;
999: readln;
  end;
```

```
{---------------------   main   ----------------------------}

procedure clock; external;
procedure minvarregulator; external;
procedure estimator; external;

begin
  initkernel(8000);
  initio;
  initmailbox(estbox);
  initmailbox(poolbox);
  initsem(timesync,0);
  initregulatorlist;
  createprocess(clock,200);
  createprocess(minvarregulator,1000);
  createprocess(estimator,1000);
  createprocess(opcom,1500);
  setpriority(maxpriority);
end.   { main }
```

APPENDIX C

```
Discrete system Regul1

" Discrete time self-tuning regulator. The
" estimator in the controller estimates one
" a-parameter, one b-parameter and one gamma-
" parameter. In addition a constant bias is
" estimated. This implies that steady-state
" errors are eliminated.
"
"
Time t
Tsamp ts
Input y yr
Output u
State a b g d p11 p12 p13 p14 p22 p23 p24 p33 p34 p44
New na nb ng nd np11 np12 np13 np14 np22 np23 np24 np33 np34 np44
State y1 yr1 u1 yr2
New ny1 nyr1 nu1 nyr2

" Initial values to the estimator

b:1
p11:100
p22:100
p33:100
p44:100

" Computation of the control signal and the
" fi-vector

v=(a*(y-yr1)+yr+g*yr1-d)/b
u=if abs(v)>ulim then sign(v)*ulim else v

f1=-(y1-yr2)
f2=u1
f3=-yr2
f4=1

" Computation of the prediction error and
" the P*fi-vector

eps=(y-yr1)-(a*f1+b*f2-yr1+g*f3+d*f4)

pf1=p11*f1+p12*f2+p13*f3+p14*f4
pf2=p12*f1+p22*f2+p23*f3+p24*f4
pf3=p13*f1+p23*f2+p33*f3+p34*f4
pf4=p14*f1+p24*f2+p34*f3+p44*f4

n=f1*pf1+f2*pf2+f3*pf3+f4*pf4

" Computation of the forgetting factors

lam1=if (p11+p22+p33)>300 then 1 else lamb1
lam2=if p44>100 then 1 else lamb2
lam=sqrt(lam1*lam2)
```

```
" Computation of the estimates and the P-matrix

k1=pf1/(n+1)
k2=pf2/(n+1)
k3=pf3/(n+1)
k4=pf4/(n+1)

na=a+k1*eps
nb=b+k2*eps
ng=g+k3*eps
nd=d+k4*eps

n11=(p11-pf1*pf1/(n+1))/lam1+del
n12=(p12-pf1*pf2/(n+1))/lam
n13=(p13-pf1*pf3/(n+1))/lam1
n14=(p14-pf1*pf4/(n+1))/lam
n22=(p22-pf2*pf2/(n+1))/lam1+del
n23=(p23-pf2*pf3/(n+1))/lam1
n24=(p24-pf2*pf4/(n+1))/lam
n33=(p33-pf3*pf3/(n+1))/lam1+del
n34=(p34-pf3*pf4/(n+1))/lam
n44=(p44-pf4*pf4/(n+1))/lam2+del

" Updating of the regressors

ny1=y
nyr1=yr
nyr2=yr1
nu1=u

ts=t+dt

dt:1
del:1e-5
lambi:0.98
lambi2:0.4
ulim:10

end
```

```
Discrete system
" A 1:st order system with time delay. To be
" used in connection with the controller Regul2.
"
" Author C F Mannerfelt 1981-09-15
"
Time t
Tsamp ts
Input u e
Output y
State y1 u1 u2
New ny1 nu1 nu2

y=-a*y1+b*u2+bias+e

ny1=y
nu1=u
nu2=u1

ts=t+h

a:-0.9
b:0.1
bias:0

h:1

End
```

- -

```
Discrete system Regul2
" Minimum variance regulator for a 1:st order system
" with time delay.
"
" Author C F Mannerfelt 1981-09-15
"
Time t
Tsamp ts
Input y yref
Output u
State r0 r1 s0 p11 p12 p13 p22 p23 p33
New nr0 nr1 ns0 n11 n12 n13 n22 n23 n33
State u1 u2 du1 du2 du3 y1 y2 dy1 dy2
New nu1 nu2 ndu1 ndu2 ndu3 ny1 ny2 ndy1 ndy2

" Initial values to the estimator

p11:100
p22:100
p33:100
r0:1

" Computation of the prediction error and the
" vector P*fi

dy=y-y2

eps=dy-r0*du2-r1*du3-s0*dy2
```

```
f1=du2
f2=du3
f3=dy2

pf1=p11*f1+p12*f2+p13*f3
pf2=p12*f1+p22*f2+p23*f3
pf3=p13*f1+p23*f2+p33*f3

fpf=f1*pf1+f2*pf2+f3*pf3

lam=if (p11+p22+p33)>300 then 1 else lam

r=la+fpf

uu=u2-(nr1*du1+ns0*dy+y-yref)/nr0
u=if abs(uu)>ulim then sign(uu)*ulim else uu

nr0=r0+pf1*eps/r
nr1=r1+pf2*eps/r
ns0=s0+pf3*eps/r

n11=(p11-pf1*pf1/r)/la+alfa
n12=(p12-pf1*pf2/r)/la
n13=(p13-pf1*pf3/r)/la
n22=(p22-pf2*pf2/r)/la+alfa
n23=(p23-pf2*pf3/r)/la
n33=(p33-pf3*pf3/r)/la+alfa

nu1=u
nu2=u1
ndu1=u-u2
ndu2=du1
ndu3=du2
ny1=y
ny2=y1
ndy1=dy
ndy2=dy1

ts=t+h

lam:0.98
h:1
alfa:1e-4
ulim:10

End
```