ISSN 0280–5316
ISRN LUTFD2/TFRT--5461--SE

Snabb självinställande regulator

Torbjörn Olsson
Jörgen Svensson

Institutionen för Reglerteknik
Lunds Tekniska Högskola
Augusti 1992

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

Document name
MASTER THESIS

Date of issue
August 1992

Document Number
ISRN LUTFD2/TFRT--5461--SE

Author(s)
Torbjörn Olsson
Jörgen Svensson

Supervisor
Anders Blomdell, Karl Johan Åström

Sponsoring organisation
NUTEK 726-91-01709
Apple Computer

Title and subtitle
Snabb självinställande regulator
(A fast adaptive controller)

Abstract

This thesis describes an implementation of an indirect adaptive controller that can operate at sampling rates of several kHz. The algorithm is based on a square root implementation of a recursive estimator and polynomial reduction method for the control design. The algorithm can handle common factors when solving the Diophantine equation. The hardware used is a Macintosh with a signal processor and AD and DA cards from National Instruments. The controller is implemented as a virtual instrument in LabView. A man–machine interface has also been implemented in LabView. The system has been tested to control servomotors and simulated drive systems. For a simple servo application, where the plant can be described by a second order model, the algorithm runs at 6 kHz.

Key words
Adaptive control. Signal processors. Real-time control. LabView. Macintosh. Digital control.

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280–5316

ISBN

Language
Swedish

Number of pages
108

Recipient's notes

Security classification

Examensarbete i reglerteknik

Snabb självinställande regulator

920814

Torbjörn Olsson
Jörgen Svensson

Handledare:

Anders Blomdell
Karl Johan Åström

INNEHÅLLSFÖRTECKNING

Förord

Detta examensarbete är gjort på institutionen för reglerteknik vid Lunds Tekniska Högskola. Handledare på institutionen har varit Karl Johan Åström och Anders Blomdell.

Vi vill härmed rikta ett stort tack till dessa handledare.

Tobbe och Jörgen

Lund 920814

1 Inledning

Vårt examensarbete gick ut på att implementera en snabb indirekt självinställande regulator (STR) med hjälp av signalprocessorkortet NB-DSP-2300, som är monterat i en Macintosh II. Vi använde oss av en likströmsmotor som vi positionsreglerade, för att efterhand testa koden som skrevs. Vi utgick från en lösning av professor K.Jagannathan. Hans lösning var emellertid specialiserad för ett visst problem och mycket ostrukturerat skriven. Vi ville göra en generell lösning och strukturera programmet ordentligt, så att det blev mera lättläst och lättare att förstå. Ett mål var också att optimera kritiska bitar av koden så att samplingshastigheten kunde ökas.
Eftersom vår regulator skulle vara **generell**, gick den till en början inte att köra lika snabbt som professor K.Jagannathans gjorde. I början var flaskhalsen i systemet lösningen av den Diophantiska ekvationen. Vi använde en metod som med hjälp av Gausselimination löste Sylvestermatrisen. Vi fann senare en metod, polynomreduceringsmetoden, som var ca tre gånger snabbare. Den klarade även av gemensamma faktorer på vänster- respektive högersida av den Diophantiska ekvationen. För att ytterligare snabba upp systemet, lade vi denna algoritm utanför avbrottsrutinerna. Algoritmen körs under tiden datorn väntar på en ny samplingspuls eller A/D-omvandlaren hämtar in ett mätvärde. Detta är möjligt eftersom nya regulatorparametrar inte behöver beräknas efter varje sampel.
För estimering av processparametrarna använde vi en variant av minstakvadrat-metoden, som heter kvadratrotsmetoden. Denna metod är bättre konditionerad p.g.a. att den inte kvadrerar kovariansmatrisen. I estimatorn har vi infört skydd mot estimatoruppvridning samt en hysteres som avgör när estimatorn ska estimera.
Efter en del strukturering och optimering av algoritmer, fick vi upp samplingsfrekvensen till ca **6 kHz** för ett andra ordningens system. Med översampling minskade dock den maximala samplingsfrekvensen.
För presentation av signaler och inställning av parametrar använde vi LabView. Med detta program är det lätt att få ett bra gränssnitt till användaren. I appendix D finns en utförlig beskrivning för användning av programmet.

Våra erfarenheter av LabView och signalprocessorn :
-Grafiska program är trevliga att arbeta med.
-Programmen blir snabbt stora och dåligt strukturerade.
-LabView är långsamt. Det är därför inte möjligt att ha snabba moduler i signalprocessorn, som kopplas ihop med LabView.
-Signalprocessorn skulle behöva en realtidskärna

Under arbetets gång stötte vi på många problem t.ex.:
-Kompilatorn översatte inte alltid programmet rätt.
-Debuggern var svår att använda tillsammans med vårt program. Detta berodde på att våra avbrott hade högre prioritet än debuggern.
-Debuggern översatte ibland flyttal på ett felaktigt sätt.
-Vid uppstart av datorn ettställs alltid en kontrollbit för avbrotten, vilket medförde att avbrottssignalen alltid var aktiv.
-I/O-kortet bytte plötsligt kanal för mätsignalen.
-Ett färdigt program för konvertering av flyttal fungerade inte som det skulle.
-Vid länkning av LabView-programmet skapades ingen ny utfil, om inte den förra versionen först kastades.
-Filterpaketet vi tänkte använda fungerade inte.

Till slut lyckades vi dock lösa alla problem. Tiden räckte tyvärr inte till för att finna ett väl fungerande filterpaket.

2. Adaptiv reglering

I adaptiva system förmodas det alltid att regulatorparametrarna justeras efter hand som processen förändrar sig. En regulator med dessa egenskaper kallas *självinställande*, eftersom den själv ställer in regulatorn för bästa möjliga reglering av processen.

2.1 Självinställande regulator

Den självinställande regulatorn bygger på iden att separera **estimatorn** från **designen** av regulatorn. De okända **process**-parametrarna estimeras "on-line", med hjälp av en rekursiv metod. Processparametrarna skickas till en **design**-algoritm som beräknar **regulatorns** parametrar. På så vis påverkas inte reglerprestanda av processvariationer. Grundiden för en självinställande regulator (STR, Self Tuning Regulator) är illustrerad i Fig. 2.1.

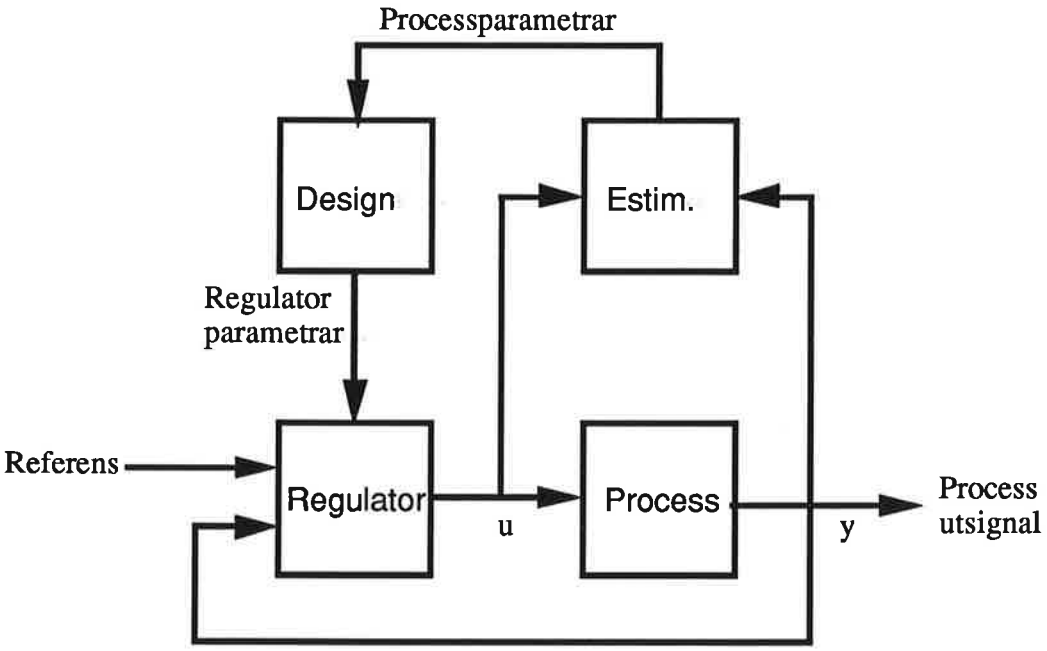

Fig. 2.1 Principskiss, STR

2.2 Direkt och indirekt STR

Den metod som beskrivs i avsnitt 2.1, kallas en *indirekt självinställande regulator* . Detta kallas den p.g.a. att uppdatereringen av regulatorns parametrar inte sker direkt, utan indirekt via estimatorn som skattar processparametrarna. Det är ibland möjligt att omparametrisera modellen för regulatorparametrarna, så att regulatorparametrarna kan estimeras direkt. På så vis fås en *direkt självinställande regulator* . Här kommer en kort beskrivning på en direkt STR.

Vi utgår från den Diophantiska ekvationen

$$A_o(q)A_m(q) = A(q)R_1(q) + B^-(q)S(q)$$

där

$$R_1(q) = R(q)/B^+(q)$$

och beskriver processen som

$$A(q)y(t) = B(q)u(t) + C(q)e(t).$$

Multiplicerar vi nu y(t) med vänster respektive högerled fås

$$\begin{aligned} A_o(q)A_m(q)y(t) &= A(q)R_1(q)y(t) + B^-(q)S(q)y(t) \\ &= R_1(q)(B(q)u(t) + C(q)e(t)) + B^-(q)S(q)y(t) \\ &= R_1(q)B(q)u(t) + B^-(q)S(q)y(t) + R_1(q)C(q)e(t) \\ &= B^-(q)(R(q)u(t) + S(q)y(t)) + R_1(q)C(q)e(t) \end{aligned}$$

Detta kan ses som en processmodell som är parametriserad i B^-, R och S. Estimering av dessa parametrar ger polynomen, R och S, direkt till regulatorn. Ett problem med denna modell är att den blir olinjär, såvida inte B^- är en konstant. Om $B^- = b_0$ blir processmodellen enligt nedan

$$y(t) = R'(q)u_f(t) + S'(q)y_f(t) + R'_1(q)C(q)e_f(t)$$

där

$$R'(q) = b_0R(q)$$

$$R'_1(q) = b_0R_1(q)$$

$$S'(q) = b_0S(q)$$

$$u_f(t) = u(t)/(A_o(q)A_m(q))$$

$$y_f(t) = y(t)/(A_o(q)A_m(q))$$

$$e_f(t) = e(t)/(A_o(q)A_m(q))$$

Fördelen med direkt STR är att design- och estimator-blocken i Fig. 2.1 kan slås samman till ett block. På detta sätt minskas algoritmberäkningarna drastiskt, vilket medför att samplingsfrekvensen kan pressas ytterligare.
Vårt problem var att göra en generell lösning, och i en sådan kan man knappast anta att alla nollställen går att förkorta bort. Tvärtom händer det ofta att nollställena är dåligt dämpade, vilket skulle medföra ringningar på styrsignalen, eller ännu värre att systemet blir instabilt. Detta är orsaken till varför, vi valt (eller snarare blivit rekommenderade) att använda den indirekta metoden. På så vis förlorar vi i snabbhet men vinner i stabilitet.

2.3 Stabilitet

Ett stort problem inom adaptiv reglering är beräkningar av algoritmer för stabilitet. Orsaken till detta ligger i att dessa system är olinjära. Förutom den vanliga reglerloopen med tillståndsåterkoppling, finns den långsammare loopen (via estimatorn och designen) som uppdaterar regulatorparametrarna. Processparametrarna, som identifieras av estimatorn, varierar oftast med tiden (för adaptiva system), vilken är den bidragande orsaken till varför dessa reglersystem blir olinjära. Det grundläggande konceptet för stabilitet av olinjära system refereras till stabilitet av en speciell lösning, typ Lyapunov stabilitet. Det kan hända att en lösning är stabil, medan en annan inte är det. Det kan därför inte finnas någon generell lösning för stabilitet av olinjära system.

Vad kan man då göra för att förebygga dessa problem på ett så bra sätt som möjligt. Modellen skall ha en sådan struktur att processens dynamik och störningen kan representeras väl. Om processmodellen inte stämmer väl överens med processen, har man redan från början ett mer känsligt system än nödvändigt, oavsett hur bra estimatorn och designen är. Estimatorn, som har till uppgift att skatta processparametrarna, är den viktigaste delen i ett adaptivt system. Det är t.ex. mycket viktigt att styrsignalen exciterar systemet tillräckligt och att vissa variabler i algoritmerna begränsas eller stängs av helt tills systemet nästa gång exciteras ordentligt. Orsaken till detta är att estimatorn kan ses uppbyggd av ett antal integratorer, vilket medför att efter en lång tid av väntan på ny excitation, har dessa växt så kraftigt att en processvariation ej blir märkbar. Man inför därför också ibland en så kallad glömskefaktor. Bättre stabilitet uppnås även då signalerna filtreras på rätt sätt. Vikning av signaler undviks genom att, innan ingångskortet till datorn, ansluta ett analogt lågpassfilter och sedan även använda ett digitalt filter. Till estimatorn filtreras de signaler som används, så att endast intressanta delar av den används. Ett annat känsligt område, i den klassiska reglertekniken, är förstärkningen vid 180 graders fasförskjutning. Då systemet är olinjärt är det svårt att beräkna denna förstärkning, vilket är anledningen till att vi, i designen, tvingar in ett nollställe i (z+1). Eftersom vårt examensarbete inte gick ut på att undersöka stabiliteten, vilket säkert skulle täcka flera examensarbeten, lämnar vi det därhän.

3. Teoretisk bakgrund

I detta kapitel förklaras de olika blocken i Fig. 2.1. Avsnitt 3.1 tar upp det matematiska sätt på vilket vi valt att beskriva processen. Avsnitt 3.2 behandlar den metod vi använt oss av för estimering. Designen finns i avsnitt 3.3 och regulatorn i avsnitt 3.4. De mätvärden estimeringen använder sig av är av stor vikt och därför är det viktigt att dessa filtreras. Detta behandlas i avsnitt 3.5.

3.1 Processen

Det finns en hel del olika sätt att beskriva en process. Den metod vi valt att använda, beskrivs av ett system med enkel insignal och utsignal (SISO), enligt fig. 3.1

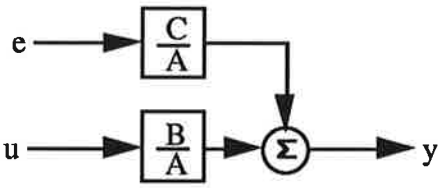

Fig. 3.1

$$A(q)y(t) = B(q)u(t) + C(q)e(t)$$

där y är utsignalen och u är insignalen till processen, och där e(t) är vitt brus. A, B och C är polynom i framåtskiftoperatorn q. A-polynomet är moniskt. Det antages att deg A = deg C = n och att deg A - deg B = d_0. Det är ibland fördelaktigt att skriva processmodellen i bakåtskiftoperatorn q^{-1}. Detta kan göras genom att införa det reciproka polynomet

$$A^*(z) = z^n A(z^{-1})$$

eller

$$A^*(z^{-1}) = z^{-n} A(z)$$

där n = deg A. Processmodellen kan då skrivas som

$$A^*(q^{-1})y(t) = B^*(q^{-1})u(t-d_0) + C^*(q^{-1})e(t)$$

där d_0 = deg A - deg B.

I kommande beräkningar tar vi ej hänsyn till det vita bruset , e(t), i processmodellen. Detta för att förenkla algoritmer i andra problemställningar. Den förenklade processmodellen blir nu enligt fig. 3.2

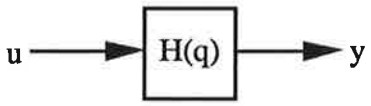

Fig. 3.2

där

$$H(q) = \frac{B(q)}{A(q)}$$

eller

$$A^*(q^{-1})y(t) = B^*(q^{-1})u(t-d_0).$$

3.2 Estimatorn

Estimatorn är hjärtat i ett adaptivt system. noggrannheten på de skattade processparametrarna bestämmer hur bra regulatorn ska bli. Det finns många olika typer av estimatoralgoritmer t.ex. stokastisk approximation, minstakvadraten (RLS), kvadratrots-RLS, instrumentvariabel och maximum likelyhood. Vi provade först RLS-metoden, men övergick senare till kvadratrots-RLS-metoden, vilken är en bättre konditionerad algoritm.

Minsta kvadratmetoden kan beskrivas på ett antal sätt. Vi har valt följande modell:

$$y(t) = \varphi_1(t)\,\theta_1 + \varphi_2(t)\,\theta_2 + \ldots + \varphi_n(t)\,\theta_n = \varphi(t)^T\,\theta$$

där y(t) är den observerade variabeln, $\theta_1, \theta_2, \ldots, \theta_n$ är okända parametrar,
och $\varphi_1(t), \varphi_2(t), \ldots, \varphi_n(t)$ är kända funktioner som även kan bero på andra kända variabler.

Variablerna $\varphi_i(t)$ kallas regressionsvariabler Regressorvektorer är i sin tur uppbyggda av regressionsvariabler :

$$\varphi(t)^T = [\ \varphi_1(t)\ \ \varphi_2(t) \ldots \varphi_n(t)\ ]$$

Vi inför även en vektor för de okända parametrarna q_i :

$$\theta^T = [\ \theta_1\ \ \theta_2 \ldots \theta_n\ ]$$

Par av observationer och regressorer fås från experiment :

$$y(1) = [\ \varphi_1(1)\ \ \varphi_2(1) \ldots \varphi_n(1)\ ]\ \theta$$

$$y(2) = [\ \varphi_1(2)\ \ \varphi_2(2) \ldots \varphi_n(2)\ ]\ \theta$$
$$\vdots \qquad \vdots \qquad \vdots \qquad \vdots \qquad \vdots$$
$$y(t) = [\ \varphi_1(t)\ \ \varphi_2(t) \ldots \varphi_n(t)\ ]\ \theta$$

Vi kan här införa en ny vektor :

$Y(t) = [\, y(1)\ \ y(2) \ldots y(t)\,]^T$

och en matris :

$$\Phi = \begin{pmatrix} \varphi^T(1) \\ \varphi^T(2) \\ \vdots \\ \varphi^T(t) \end{pmatrix} = \begin{pmatrix} [\ \varphi_1(1) & \varphi_2(1) \ldots & \varphi_n(1)\ ] \\ [\ \varphi_1(2) & \varphi_2(2) \ldots & \varphi_n(2)\ ] \\ \vdots & \vdots & \vdots \\ [\ \varphi_1(t) & \varphi_2(t) \ldots & \varphi_n(t)\ ] \end{pmatrix}$$

Vilket medför att

$$\mathbf{Y} = \Phi\theta$$

Problemet är nu att lösa de okända parametrarna på ett sådant sätt, att de beräknade utsignalerna från modellen ligger så nära som möjligt de uppmätta variablerna y(t).
Eftersom de uppmätta variablerna y(t) är linjära i parametrarna θ och minsta kvadrat kriteriet är kvadratiskt, finns där en analytisk lösning.

Om vi inför residualerna $\varepsilon(t)$, definierade som

$$\varepsilon(i) = y(i) - \hat{y}(i) = y(i) - \varphi^T(i)\hat{\theta}$$

kan minsta kvadrat felet skrivas enligt :

$$V(\hat{\theta},t) = \frac{1}{2}\sum_{i=1}^{t} \varepsilon(i)^2 = \frac{1}{2}\sum_{i=1}^{t} \left(y(i) - \varphi^T(t)\hat{\theta}\right) = \frac{1}{2}\, E^T E = \frac{1}{2}||E||^2$$

där

$$E = Y - \hat{Y} = Y - \Phi\hat{\theta}$$

Lösningen av minsta kvadrat problemet blir då :

$$\hat{\theta} = (\Phi^T\Phi)^{-1}\Phi^T Y$$

Bevis :

Om vi som innan, inför minsta kvadraten av felet

$$V(\hat{\theta},t) = \frac{1}{2}\sum_{i=1}^{t} \varepsilon(i)^2 = \frac{1}{2}\, E^T E$$

så blir derivatan av detta

$$\frac{\partial V(\hat{\theta},t)}{\partial \hat{\theta}} = \frac{1}{2}\,\frac{\partial (E^T E)}{\partial \hat{\theta}} = -E^T\Phi = -\,(\Phi^T E)^T = -\,(\Phi^T (Y - \Phi\hat{\theta}))^T = 0$$

sättes detta till noll kan vi lösa ut $\hat{\theta}$.

$$\rightarrow\ \Phi^T\Phi\hat{\theta} = \Phi^T Y \ \rightarrow\ \hat{\theta} = (\Phi^T\Phi)^{-1}\Phi^T Y$$

Uttrycket ovan

$$\frac{1}{2}\,\frac{\partial (E^T E)}{\partial \theta} = -E^T\Phi$$

kan beräknas på följande sätt. Vi vet att :

$$E = Y - \hat{Y} = Y - \Phi\hat{\theta} \rightarrow \frac{\partial E}{\partial \theta} = 0 - \Phi = -\,\Phi$$

vilket medför :

$$\frac{1}{2}\,\frac{\partial (E_1^T E_2)}{\partial \hat{\theta}} = \frac{1}{2}\,E_1^T\frac{\partial E_2}{\partial \hat{\theta}} + \frac{1}{2}\,E_2^T\frac{\partial E_1}{\partial \hat{\theta}} = -\frac{1}{2}\,E_1^T\Phi - \frac{1}{2}\,E_2^T\Phi = (E_1 = E_2) = -E^T\Phi$$

För att inte förlora för mycket tid i datorimplementeringen av algoritmerna, så måste en rekursiv lösning beräknas. Beräkningarna kan göras på ett sådant sätt att resultaten som uppnåtts vid tiden t-1 används för att beräkna parametrarna vid tiden t.
Lösningen till detta minstakvadratproblem kan nu skrivas om till en rekursiv algoritm.
Låt
$\hat{\theta}(t-1)$
representera minsta kvadrat estimatet baserat på mätningen vid tiden t-1.
Uttrycket
$(\Phi^T\Phi)^{-1}$
ersätts med P(t), vilket medför

$$P(t) = (\Phi^T(t)\Phi(t))^{-1} = \left(\sum_{i=1}^{t} \varphi(i)\varphi^T(i)\right)^{-1}$$

eller

$$P(t)^{-1} = \Phi^T(t)\Phi(t) = \sum_{i=1}^{t} \varphi(i)\varphi^T(t) = \left(\sum_{i=1}^{t-1} \varphi(i)\varphi^T(t) \right) + \varphi(t)\varphi^T(t) = P(t-1)^{-1} + \varphi(t)\varphi^T(t)$$

Ställer vi nu upp normalekvationen får vi

$$\hat{\theta} = (\Phi^T\Phi)^{-1}\Phi^T Y = \left(\sum_{i=1}^{t} \varphi(i)\varphi^T(i)\right)^{-1}\left(\sum_{i=1}^{t} \varphi(i)y(i)\right) = P(t)\left(\sum_{i=1}^{t} \varphi(i)y(i)\right)$$

som med en del algebraiska beräkningar leder fram till nedanstående resultat.
Rekursiv minstakvadratestimering:

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

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

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

Bevis:

Om vi utgår från normalekvationen:

$$\hat{\theta}(t) = P(t)\left(\sum_{i=1}^{t} \varphi(i)y(i)\right) = P(t)\left(\sum_{i=1}^{t-1} \varphi(i)y(i) + \varphi(t)y(t)\right)$$

blir

$$\hat{\theta}(t-1) = P(t-1)\left(\sum_{i=1}^{t-1} \varphi(i)y(i)\right)$$

vilket medför att

$$P(t-1)^{-1}\,\hat{\theta}(t-1) = \left(\sum_{i=1}^{t-1} \varphi(i)y(i)\right)$$

Använder vi oss nu av att

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

blir

$$\left(\sum_{i=1}^{t-1} \varphi(i)y(i)\right) = P(t-1)^{-1}\,\hat{\theta}(t-1) = (P(t)^{-1} - \varphi(t)\varphi^T(t))\,\hat{\theta}(t-1) = P(t)^{-1}\hat{\theta}(t-1) - \varphi(t)\varphi^T(t)\hat{\theta}(t-1)$$

sätts detta sedan in i normalekvationen får vi följande uttryck:

$$\hat{\theta}(t) = P(t)\left(\sum_{i=1}^{t-1}\varphi(i)y(i) + \varphi(t)y(t)\right) = P(t)\left((P(t)^{-1}\hat{\theta}(t-1) - \varphi(t)\varphi^T(t)\hat{\theta}(t-1)) + \varphi(t)y(t)\right)$$

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

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

där

$K(t) = P(t)\varphi(t)$

och

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

Residualen $\varepsilon(t)$ kan ses som ett prediktionsfel (ett steg framåt i tiden) av $y(t)$ baserat på estimatet $\theta(t-1)$. För att nå fram till den slutgiltiga lösningen måste även en rekursiv algoritm för $P(t)$ beräknas.
I den lösningen använder vi oss av "Matrisinversions lemma" (Appendix A) som ger att

$$(A + BCD)^{-1} = A^{-1} - A^{-1}B(C^{-1} + DA^{-1}B)^{-1}DA^{-1}$$

Använder vi nu detta lemma på $P(t)$ får vi:

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

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

Detta medför att

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

En matrisinversion är alltså nödvändigt för att beräkna $P(t)$.

Ett annat problem som inträffar är att parametrarna θ inte alltid är konstanta. I många adaptiva reglerproblem får man ta hänsyn till att parametrarna varierar med tiden. För att i en rekursiv lösning ta hand om detta problem införs en glömskefaktor λ. Genom att ersätta det gamla uttrycket för minsta kvadraten av felet till

$$V(\hat{\theta},t) = \frac{1}{2}\sum_{i=1}^{t}\lambda^{t-i}(y(i) - \varphi^T(i)\hat{\theta})^2$$

där $0 < \lambda < 1$. Parametern λ kallas för *glömskefaktorn*. Om vi nu inför detta i den rekursiva lösningen kommer aktuell data vara viktat med ett, medan data som är n tidsenheter gammal viktas med λ^n. Utför vi nu samma beräkningar för rekursiv minsta kvadrat metoden sedan tidigare, får vi följande resultat.

Rekursiv minstakvadratmetod med glömskefaktor:

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

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

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

En nackdel med denna metod är att även om det inte finns information i ny data, så ökar P(t) exponentiellt med λ. Detta är en typ av estimatoruppvridning. För att förhindra detta begränsar vi P(t).
I numerisk analys är det väl känt att noggrannheten kan bli betydligt försämrad vid användning av minstakvadratmetoden för att lösa normalekvationerna. Orsaken till detta är att de uppmätta värdena kvadreras onödigt. Därför har vi valt att beräkna estimatet med en kvadratrotsmetod, som är en variant av RLS. Denna metod är mycket bättre numeriskt konditionerad.

Kvadratrotsalgoritmen

Om vi utgår från att

$$E = Y - \Phi\theta$$

så förändrar inte en ortogonal transformation Q den Euclidiska normen av felet.

$$\tilde{E} = QE = QY - Q\Phi\theta = \tilde{Y} - \tilde{\Phi}\theta$$

Väljs transformationen Q så att $Q\Phi$ blir en övre triangulär matris, blir ovanstående ekvation

$$\begin{bmatrix} \tilde{e}^1 \\ \tilde{e}^2 \end{bmatrix} = \begin{bmatrix} \tilde{y}^1 \\ \tilde{y}^2 \end{bmatrix} - \begin{bmatrix} \tilde{\Phi}_1 \\ 0 \end{bmatrix} \theta$$

där $\tilde{\Phi}_1$ är övre triangulär. Minstakvadratestimatet ges då som

$$\tilde{\Phi}_1\theta = \tilde{y}^1$$

där felet blir

$$(\tilde{e}^2)^T\tilde{e}^2$$

För att uppnå en rekursiv lösning till denna metod, utgår man från att beräkna väntevärdet av θ givet det senast uppmätta y(t)

$$E(\theta/y)$$

där

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

θ är i detta fallet $N(\theta^0,P)$ och e är $N(0,\sigma^2)$. Kovariansmatrisen för y och θ, som denna metod bygger på (detta gäller även RLS), blir

$$R = \mathrm{cov}\begin{bmatrix} y \\ \theta \end{bmatrix} = \begin{bmatrix} R_y & R_{y\theta} \\ R_{\theta y} & R_\theta \end{bmatrix} = \begin{bmatrix} \varphi^T P\varphi & \varphi^T P \\ P\varphi & P \end{bmatrix} + \begin{bmatrix} \sigma^2 & 0 \\ 0 & 0 \end{bmatrix}$$

Bevis:

Vi utgår från , enligt tidigare, att

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

där

$E\{\theta\} = \theta^0$ och $E\{\theta\theta^T\} = P$

$E\{e\} = 0$ och $E\{ee^T\} = \sigma^2$

Kovariansmatrisen R beräknas då enligt följande

$$E\left\{\left(\begin{bmatrix} y \\ \theta \end{bmatrix} - \begin{bmatrix} m_y \\ \theta^0 \end{bmatrix}\right)\left(\begin{bmatrix} y \\ \theta \end{bmatrix} - \begin{bmatrix} m_y \\ \theta^0 \end{bmatrix}\right)^T\right\} = E\left\{\left(\begin{bmatrix} y \\ \theta \end{bmatrix} - \begin{bmatrix} m_y \\ \theta^0 \end{bmatrix}\right)\left(\begin{bmatrix} y^T & \theta^T \end{bmatrix} - \begin{bmatrix} m_y^T & (\theta^0)^T \end{bmatrix}\right)\right\}$$

$$= E\begin{bmatrix} yy^T & y\theta^T \\ \theta y^T & \theta\theta^T \end{bmatrix} - E\begin{bmatrix} m_y m_y^T & m_y(\theta^0)^T \\ \theta^0 m_y^T & \theta^0(\theta^0)^T \end{bmatrix} = E\begin{bmatrix} yy^T & y\theta^T \\ \theta y^T & \theta\theta^T \end{bmatrix}$$

där

$$R_y = E\{yy^T\} = E\{(\varphi^T\theta + e)(\varphi^T\theta + e)^T\} = E\{\varphi^T\theta\theta^T\varphi + \varphi^T\theta e^T + e\theta^T\varphi + ee^T\}$$

$$= \varphi^T E\{\theta\theta^T\}\varphi + \varphi^T\theta E\{e^T\} + E\{e\}\theta^T\varphi + E\{ee^T\}$$

$$= \varphi^T P\varphi + 0 + 0 + \sigma^2 = \varphi^T P\varphi + \sigma^2$$

$$R_{y\theta} = E\{y\theta^T\} = E\{(\varphi^T\theta + e)\theta^T\} = E\{\varphi^T\theta\theta^T + e\theta^T\} = \varphi^T E\{\theta\theta^T\} + E\{e\theta^T\}$$

$$= \varphi^T P + 0 = \varphi^T P$$

$$R_{\theta y} = E\{\theta y^T\} = E\{\theta(\varphi^T\theta + e)^T\} = E\{\theta\theta^T\varphi + \theta e^T\} = E\{\theta\theta^T\}\varphi + E\{\theta e^T\}$$

$$= P\varphi + 0 = P\varphi$$

$$R_\theta = E\{\theta\theta^T\} = P$$

Detta leder till resultatet

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

Lösningen för $E(\theta/y)$ blir då (se Appendix B)

$$E(\theta/y) = \theta^0 + R_{\theta y}R_y^{-1}\,(y - m_y)$$

där kovariansen är

$$\text{cov}\,(\theta/y) = R_{\theta/y} = R_\theta - R_{\theta y}R_y^{-1}R_{y\theta}$$

Vi ser här att det fortfarande rör sig om minstakvadratmetoden ty ersätts
$R_y^{-1} = (\varphi^T P\varphi)^T$, $R_{y\theta} = \varphi^T P$, $R_{\theta y} = P\varphi$ och $R_\theta = P\varphi^T$ blir

$$\hat{\theta} = E(\theta/y) = \theta^0 + (P\varphi)(\varphi^T P\varphi)^{-1}\,(y - m_y) \qquad \text{där } m_y = \varphi^T\theta^0$$

och

$$\text{cov}\,(\hat{\theta}) = \text{cov}\,(\theta/y) = R_{\theta/y} = P - (P\varphi)(\varphi^T P\varphi)^{-1}(\varphi^T P)$$

Återgår vi nu till problemet med att undvika kvadreringen av Φ måste vi utnyttja någon typ av QR-faktorisering, där vi formar matriserna enligt tycke. Den symmetriskt positiva matrisen P har dekompositionen $P = LDL^T$, där L är en undre triangulär matris med enhetsdiagonal och D är en positiv diagonalmatris. Matrisen R kan nu skrivas om som

$$R = \begin{bmatrix} \varphi^T P\varphi+\sigma^2 & \varphi^T P \\ P\varphi & P \end{bmatrix} = \begin{bmatrix} \varphi^T LDL^T\varphi+\sigma^2 & \varphi^T LDL^T \\ LDL^T\varphi & LDL^T \end{bmatrix}$$

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

Om R transformeras till

$$R = \begin{bmatrix} 1 & 0 \\ K & \tilde{L} \end{bmatrix}\begin{bmatrix} \tilde{\sigma}^2 & 0 \\ 0 & \tilde{D} \end{bmatrix}\begin{bmatrix} 1 & K^T \\ 0 & \tilde{L}^T \end{bmatrix}$$

enligt en algoritm kallad Dyadic decomposition (se Appendix C), ges det rekursiva estimatet som

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

med kovariansen

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

På detta sett har vi nu lyckats undvika kvadrering av Φ, vilket som tidigare nämnt ger en bättre konditionerad algoritm.

3.3 Design

Designblocket har till uppgift att beräkna en regleralgoritm för processen. Detta sker med de estimerade processparametrarna från estimatorn. Det finns flera olika designmetoder t.ex.: minsta varians-, linjär kvadratisk-, polplacering-, modellföljande-metoden. Vi har använt oss av polplaceringsmetoden. I denna metod utgår vi från den Diophantiska ekvationen.

Om vi, som tidigare nämnt, låter processen beskrivas på följande vis

$$A(q)y(t) = B(q)u(t) + C(q)e(t)$$

och det önskade återkopplade systemet enligt

$$A_m(q)y(t) = B_m(q)u_c(t)$$

får vi en regulator

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

där R(q) och S(q) är beräknade ur den Diophantiska ekvationen

$$AR + BS = A_oA_mB^+$$

där

$$B = B^+B^-$$

$$B_m = B^-B_m'$$

A_o är observerarpolynomet, B^+ innehåller de nollställen som går att förkorta och är moniskt. Resterande del i B-polynomet finns i B^-. B_m' är den del i B_m som ser till att det stationära tillståndet i det återkopplade systemet blir ett. I vårt fall blir detta

$$B_m' = B(1)/A_m(1)$$

Eftersom vår lösning ska vara så generell som möjligt , vill vi ej förkorta några nollställen. Dåligt dämpade nollställen ger tråkiga ringningar på styrsignalen, vilket kan ge tidiga förslitningar på styrdon och det vill vi undvika. Därför sättes

$$B^+ = 1$$

$$B^- = B$$

detta ger följande resultat

$AR + BS = A_o A_m$

$B_m = B\, B_m^{'}$

$T = A_o\, B_m^{'}$

Vi tvingar även in ett nollställe i S-polynomet, för att försäkra oss om att förstärkningen är noll vid 180 graders fasvridning. På liknande sätt gör vi med R-polynomet, för att få en integrerande verkan. R- och S-polynomen får då dessa utseende:

$R = (q - 1)R^{'}$

$S = (q + 1)S^{'}$

Detta kan få till följd att observerarpolynomet måste ökas i gradtal för att kausaliteten ska bibehållas. Det finns en hel del villkor som måste vara uppfyllda för att uppnå kausalitet.

$\deg A_o >= 2 \deg A - \deg A_m - \deg B^+ + f - 1$

där f avser antalet integratorer i R-polynomet, som i vårt fall är ett. Deg $B^+ = 0$ p.g.a. att vi ej förkortar några nollställen. Detta ger att

$\deg A_o >= 2 \deg A - \deg A_m$

Summan av gradantalet på den Diophantiska ekvationens högersida, ska vara lika stort som gradtalet av AR. Dessutom måste R vara moniskt. Detta leder fram till följande villkor:

$\deg R = \deg A_o + \deg A_m + \deg B^+ - \deg A$

men $\deg B^+ = 0$, i vårt fall, medför att

$\deg R = \deg A_o + \deg A_m - \deg A$

Det påtvingade nollstället ger

$\deg R^{'} = \deg R - 1$

Gradtalet av BS måste vara åtminstone ett mindre än gradtalet av AR, för att (AR+BS) ska vara moniskt. Samtidigt måste

$\deg A > \deg B$

och

$\deg R >= \deg S$

för att kausalitet ska uppnås. Detta medför att

$\deg S^{'} = \deg S - 1$

p.g.a. det påtvingade nollstället (q-1).

Lösning av den Diophantiska ekvationen:

För att lösa den Diophantiska ekvationen har vi använt oss av polynomreduktionsmetoden. Vi provade först med en metod där vi ställde upp Sylvestermatrisen och sedan löste den med Gausselimination. Denna metod visade sig dock ha sina brister. Den klarar inte av om det finns gemensamma faktorer i den Diophantiska ekvationens vänsterled respektive högerled. Även tidsmässigt är den långsammare än polynomreduktionsmetoden. Det skiljer en faktor tre i snabbhet. Detta är också av stor vikt inom adaptiv reglering, eftersom det finns många stora beräkningar som begränsar samplingstiden. Den mest tidsödande algoritmen är just lösningen av den Diophantiska ekvationen, vilket gör den till systemets flaskhals vad gäller tiden.

Polynomreduktionsmetoden:

Metoden går ut på att lösa den Diophantiska ekvationen

$$A(z)X(z) + B(z)Y(z) = C(z)$$

med villkoret att deg Y(z) < deg A(z). Den ger en minsta ordningens lösning med avseende på Y(z) och klarar även av om det finns gemensamma faktorer i vänster respektive högerled. En förutsättning för att metoden ska fungera, är att ordning och index på polynomen är enligt följande

$$A(z) = a_0 + a_1 z + a_2 z^2 + \ldots + a_p z^p$$

$$B(z) = b_0 + b_1 z + b_2 z^2 + \ldots + b_q z^q$$

$$C(z) = c_0 + c_1 z + c_2 z^2 + \ldots + c_r z^r$$

$$Y(z) = y_0 + y_1 z + y_2 z^2 + \ldots + y_{p-1} z^{p-1}$$

om $r < p + q$ så blir

$$X(z) = x_0 + x_1 z + x_2 z^2 + \ldots + x_{q-1} z^{q-1}$$

annars

$$X(z) = x_0 + x_1 z + x_2 z^2 + \ldots + x_{r-p} z^{r-p}$$

om $r >= p + q$.

där deg A = p, deg B = q och deg C = r.

Polynomreduktionsmetoden går ut på att fyra olika villkor genomgående testas, ända tills två av polynomen, A(z), B(z) och C(z), har blivit lika med noll . Villkoren bygger på polynomens gradtalsförhållanden.

* villkor 1 * Om (r >= p och C'<> 0 och A'<> 0) så ersätt

$AX + BY = C$ med $AX' + BY = C'$ där $C' = C - A(c_r/a_p)z^{r-p}$

* villkor 2 * Om istället (r >= q och C'<> 0 och B'<> 0) så ersätt

$AX + BY = C$ med $AX + BY' = C'$ där $C' = C - B(c_r/b_q)z^{r-q}$

* villkor 3 * Om istället (q >= p och B'<> 0 och A'<> 0) så ersätt

$AX + BY = C$ med $AX' + B'Y = C$ där $B' = B - A(b_q/a_p)z^{q-p}$

* villkor 4 * Om istället (p >= q och A'<> 0 och B'<> 0) så ersätt

$AX + BY = C$ med $A'X + BY' = C$ där $A' = A - B(a_p/b_q)z^{p-q}$

Varje gång ett villkor har blivit uppfyllt minskar gradtalet, på antingen A, B eller C, med ett. Dessa reduktioner pågår tills två av polynomen har blivit lika med noll. Detta resulterar i att X'=0 och Y'=0. För att få lösningarna till X och Y används bakåtsubstitution. Det är viktigt att hålla reda på i vilken ordning villkoren har använts. Bakåtsubstitution sker i omvänd ordning mot vad polynomreduktionen gjort.

Bakåtsubstitution

* villkor 1 * $X = X' + (c_r/a_p)z^{r-p}$

* villkor 2 * $Y = Y' + (c_r/b_q)z^{r-q}$

* villkor 3 * $X = X' + Y(b_q/a_p)z^{q-p}$

* villkor 4 * $Y = Y' + X(a_p/b_q)z^{p-q}$

För att förenkla hanteringen av R och S polynomen låter vi nollstället (z-1) bakas ihop med A- istället för R-polynomet. På liknade sätt har vi gjort med nollstället (z+1), som är ihopbakat med B- istället för S-polynomet. Efter det skickas A-, B- och C-polynomen samt dess gradtal p, q och r till en procedur som beräknar polynomreduktionen. Resultatet som kommer från denna procedur, X och Y eller rättare sagt R' och S', multipliceras med nollstället (eller egentligen polen) (z-1) respektive nollstället (z+1). Detta leder till följande resultat:

$$R(q) = (q-1)R'(q) = (q-1)(r'_0 q^{n-1} + r'_1 q^{n-2} + \ldots + r'_{n-1})$$

$$= r'_0 q^n + (r'_1 - r'_0) q^{n-1} + (r'_2 - r'_1) q^{n-2} + \ldots + (r'_{n-1} - r'_{n-2}) q + (-r'_{n-1})$$

$$= r_0 q^n + r_1 q^{n-1} + r_2 q^{n-2} + \ldots + r_{n-1} q + r_n$$

där $r_0 = 1$ ty R(q) är moniskt.

På liknande sätt blir S-polynomet

$$S(q) = (q+1)S'(q) = (q+1)(s'_0 q^{n-1} + s'_1 q^{n-2} + \ldots + s'_{n-1})$$

$$= s'_0 q^n + (s'_1 + s'_0) q^{n-1} + (s'_2 + s'_1) q^{n-2} + \ldots + (s'_{n-1} + s'_{n-2}) q + s'_{n-1}$$

$$= s_0 q^n + s_1 q^{n-1} + s_2 q^{n-2} + \ldots + s_{n-1} q + s_n$$

T(q)-polynomet beräknas sedan som

$$T(q) = A_o(q)\, B'_m$$

där B'_m i vårt fall är en skalfaktor för att få det återkopplade systemets förstärkning vid stationaritet lika med ett. Detta fås som

$$T(q) = A_o(q)\, B'_m = A_o(q) \frac{A_m(1)}{B(1)} = A_o(q)\left(\frac{a_{m0} + a_{m1} + \ldots + a_{mn}}{b_0 + b_1 + \ldots + b_k} \right)$$

därefter skickas R(q)-, S(q)- och T(q)-polynomen vidare till regulatorn.

3.4 Regulator

Om processen beskrivs som

$A(q)y(t) = B(q)u(t)$

där u(t) är styrsignalen och y(t) är den uppmätta utsignalen kan man finna en regulatorlösning med en framkopplingsdel och en återkopplingsdel enligt polplaceringsmetoden. P.g.a. kausalitet måste deg A >= deg B. A(q) måste även vara moniskt. Fig. 3.3 visar det återkopplade systemet.

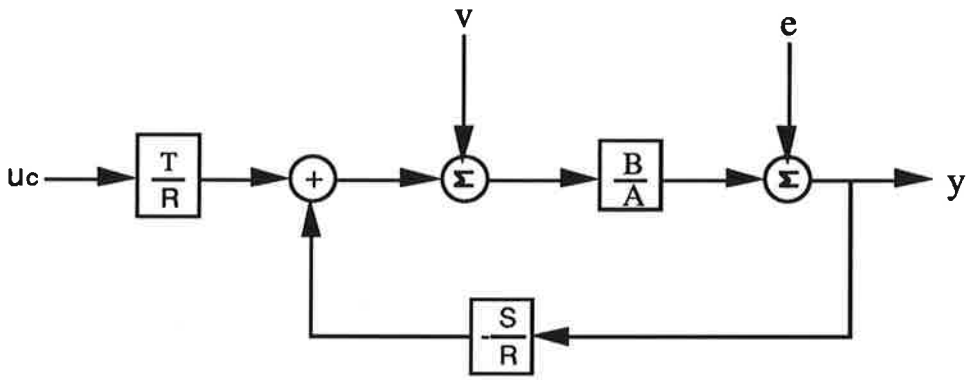

Fig. 3.3

där u_c är börvärdet, y är den uppmätta processutsignalen, v är en laststörning och e är mätbrus. T/R är den framkopplade regulatorn och S/R är den återkopplade regulatorn. Blockschemat kan även visas enligt fig. 3.4 , där det tydligt framgår att regulatorn blir enligt

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

där u är styrsignalen, u_c är referensvärdet och y är processens utsignal.

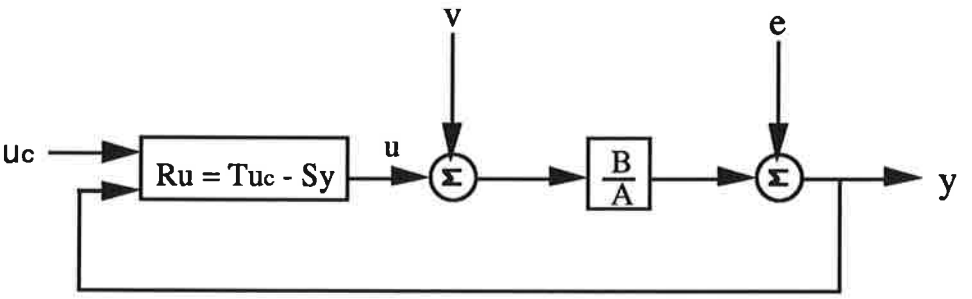

Fig. 3.4

I fig. 3.4 syns det att om R-polynomet är moniskt så kan styrsignalen beräknas enligt

$$u(t) = - r_1u(t-1) - r_2u(t-2) \ldots - r_nu(t-n) + t_0u_c(t) + t_1u_c(t-1) \ldots + t_mu_c(t-m)$$
$$- s_0y(t) - s_1y(t-1) \ldots - s_vy(t-v)$$

r_0 är alltså i detta fall lika med ett. Beräkning av det återkopplade systemet ger att

$$y = \frac{BT}{AR + BS} u_c + \frac{BR}{AR + BS} \upsilon + \frac{AR}{AR + BS} e$$

där vi i fortsättningen inte tar hänsyn till laststörningar och mätbrus. Eftersom vi har tvingat in en integrator i R-polynomet, kommer laststörningar inte att ge något stationärt fel. Den reducerade ekvationen ger oss möjlighet att placera polerna där vi vill. Sätter vi nu upp en önskad modell för det återkopplade systemet

$$y = \frac{B_m}{A_m} u_c$$

får vi

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

som leder till den Diophantiska ekvationen som beräknas i designdelen.

Integratoruppvridning

Ett problem som dyker upp med integratorn, är om styrsignalen ger större värde än vad som behövs, för att bottna styrdonet. Det sker då en integratoruppvridning, dvs att inte förrän processens utsignal nått upp till börvärdet slutar integratordelen att växa och först efter att utsignalen blivit större än börvärdet minskar integratordelen. Detta leder till väldiga insvängningsförlopp och ibland även till att systemet blir instabilt. För att förhindra detta beteende måste kunskap om de olinjära styrdon som styrsignalen ska styra utnyttjas. Antingen sätts en modell upp, eller så får man mäta signalen både före och efter styrdonet. I vårt fall vet vi att styrsignalen ligger mellan -10V och +10V. Om vi nu först ser på regulatorn som är

$$R^*(q^{-1})u(t) = T^*(q^{-1})u_c(t) - S^*(q^{-1})y(t)$$

där

$$R^*(q^{-1}) = (1 + r_1q^{-1} + r_2q^{-2} + \ldots + r_nq^{-n})$$

$$S^*(q^{-1}) = (s_0 + s_1q^{-1} + s_2q^{-2} + \ldots + s_vq^{-v})$$

$$T^*(q^{-1}) = (t_0 + t_1q^{-1} + t_2q^{-2} + \ldots + t_mq^{-m})$$

Om nu u(t) adderas på båda sidor, får vi

$$R^*(q^{-1})u(t) + u(t) = T^*(q^{-1})u_c(t) - S^*(q^{-1})y(t)\ \ R^*(q^{-1}) + u(t)$$

vilket medför att

$$u(t) = T^*(q^{-1})u_c(t) - S^*(q^{-1})y(t)\ \ R^*(q^{-1}) + (1 - R^*(q^{-1}))u(t)$$

där

$$(1 - R^*(q^{-1}))u(t) = (- r_1q^{-1} - r_2q^{-2} - \ldots - r_nq^{-n})u(t) = - r_1u(t-1) - r_2u(t-2) - \ldots - r_nu(t-n)$$

En regulator med skydd mot integratoruppvridning blir då

$$v(t) = T^*(q^{-1})u_c(t) - S^*(q^{-1})y(t)\ \ R^*(q^{-1}) + (1 - R^*(q^{-1}))u(t)$$

$$u(t) = \text{sat}\,(v(t))$$

I vårt fall blir

$$u(t) = \begin{cases} -10V & \text{då} \quad v(t) < -10V \\ v(t) & \text{då} \quad -10V < v(t) < +10V \\ +10V & \text{då} \quad v(t) > +10V \end{cases}$$

Det skydd mot integratoruppvridning vi använder oss av, ger en "deadbeat"-observerare. Det går även att använda ett polynom $A^*_0(q^{-1})$. I detta fall adderas $A^*_0(q^{-1})u(t)$ på båda sidor enligt

$$R^*(q^{-1})u(t) + A^*(q^{-1})u(t) = T^*(q^{-1})u_c(t) - S^*(q^{-1})y(t) + A^*(q^{-1})u(t)$$

vilket medför

$$A^*(q^{-1})u(t) = T^*(q^{-1})u_c(t) - S^*(q^{-1})y(t) + (A^*(q^{-1}) - R^*(q^{-1}))u(t)$$

Denna regulator, med skydd mot integratoruppvridning, blir då

$$A^*(q^{-1})v(t) = T^*(q^{-1})u_c(t) - S^*(q^{-1})y(t) + (A^*(q^{-1}) - R^*(q^{-1}))u(t)\ u(t)$$

$$u(t) = \text{sat}\,(v(t))$$

där $A^*_0(q^{-1})$ är ett observerarpolynom.

3.5 Filter

Filter är viktigt. Inget tidsdiskret system fungerar säkert utan filtrering av signalerna. I den här sortens system inför man filter på ingången till regulatorn, filter för insignalerna till estimatorn, samt ett rekonstruktionsfilter på utgången. Tiden medgav ej en ordentlig studie av filter.
För att strukturera dessa filter har vi implementerat två klasser av filter, tillsammans med metoder för att mata in signaler till dem, hämta signaler från dem samt för att skapa och förstöra dem.
Den ena klassen av filter implementerar en "Direkt form 2" struktur för ett generellt linjärt tidsdiskret filter. Denna klass lämpar sig bäst för filter med få poler (helst färre än tre) och FIR-filter, dvs helt pollösa filter. Dessutom har funktioner införts i klassen så att man kan använda filtren av denna klass för omvandling av sampelfrekvens (Decimering, interpolering och konvertering med rationellt förhållande mellan sampelfrekvenserna). Dessa funktioner utnyttjas i ingångs- och utgångsfilterna.
Den andra klassen implementerar filter bestående av kaskadkopplade första eller andra ordningens filter, s.k Biquads. Denna realisering har betydligt bättre numeriska egenskaper för filter med många poler, vilket är fördelaktigt när man skall åstadkomma IIR-filter med skarp avskärning vid gränsfrekvensen. Även i denna klass har införts funktioner för att möjliggöra skilda samplingsfrekvenser på in- och utgångarna.
Denna strukturering, tillsammans med integrerade filterberäkningsrutiner, har gjort det möjligt att enkelt kunna prova egenskaperna för regulatorsystemet med olika typer av filtrering på olika platser.

4 Exempel

I detta kapitel belyser vi teorin i kapitel 3, genom att beräkna ett detaljerat exempel. Organisationen är helt parallell med presentationen i kapitel 3.

4.1 Processen

Vi utgår från en processmodell av första ordningen, d.v.s en tidskonstant, som endast har två parametrar, för estimatorn, att skatta. Detta problem kan tyckas trivialt, men kommande beräkningar lär visa motsatsen.

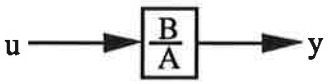

Fig. 4.1

enligt fig. 4.1, blir

$A(q)y(t) = B(q)u(t)$

där

$A(q) = q + a_1$

$B(q) = b_0$

eller om vi använder reciproka polynom

$A^*(q^{-1})y(t) = B^*(q^{-1})u(t-d_0)$

där

$A^*(q^{-1}) = 1 + a_1q^{-1}$

$B^*(q^{-1}) = b_0$

I vårt fall blir ovanstående ekvation

$(1 + a_1q^{-1})y(t) = b_0u(t-1)$

där a_1 och b_0 är okända, vilket brukar vara fallet vid adaptiv reglering och där $d_0 = \deg A - \deg B = 1$.

4.2 Estimatorn

Ställer vi upp processmodellen enligt

$y(t) = - a_1y(t-1) + b_0u(t-1)$

får vi att

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

där

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

$$\theta^T = (\, a_1 \;\; b_0 \,)$$

Om vi nu tar par av observationer (y(t) och u(t)) blir

$$y(1) = (\, -\, y(0) \quad u(0) \quad)\theta$$

$$\begin{array}{llll} y(2) = (\, -\, y(1) & u(1) &) & \theta \\ \vdots & \vdots & \vdots & \vdots \\ y(t) = (-\, y(t-1) & u(t-1) &) & \theta \end{array}$$

detta i sin tur medför att

$$Y(t) = [\; y(1) \;\; y(2) \ldots y(t) \;]^T$$

och

$$\Phi = \begin{pmatrix} \varphi^T(1) \\ \varphi^T(2) \\ \vdots \\ \varphi^T(t) \end{pmatrix} = \begin{pmatrix} -y(0) & u(0) \\ -y(1) & u(1) \\ \vdots & \vdots \\ -y(t-1) & u(t-1) \end{pmatrix}$$

Om vi nu för in dessa värden till lösningen av minsta kvadrat metoden

$$\hat{\theta} = (\Phi^T\Phi)^{-1}\Phi^T Y = \left(\sum_{i=1}^{t} \varphi(i)\varphi^T(i)\right)^{-1}\left(\sum_{i=1}^{t} \varphi(i)y(i)\right) = P(t)\left(\sum_{i=1}^{t} \varphi(i)y(i)\right)$$

får vi följande lösning

$$\hat{\theta} = (\Phi^T\Phi)^{-1}\Phi^T Y = \begin{bmatrix} \sum_{k=1}^{t} y^2(k-1) & \sum_{k=1}^{t} -y(k-1)u(k-1) \\ \sum_{k=1}^{t} -u(k-1)y(k-1) & \sum_{k=1}^{t} u^2(k-1) \end{bmatrix}^{-1} \begin{bmatrix} \sum_{k=1}^{t} -y(k-1)y(k) \\ \sum_{k=1}^{t} u(k-1)y(k) \end{bmatrix}$$

$$= \frac{1}{Q}\begin{bmatrix} \sum_{k=1}^{t} u^2(k-1) & \sum_{k=1}^{t} y(k-1)u(k-1) \\ \sum_{k=1}^{t} u(k-1)y(k-1) & \sum_{k=1}^{t} y^2(k-1) \end{bmatrix}\begin{bmatrix} \sum_{k=1}^{t} -y(k-1)y(k) \\ \sum_{k=1}^{t} u(k-1)y(k) \end{bmatrix}$$

där

$$Q = \sum_{k=1}^{t} y^2(k-1)\sum_{k=1}^{t} u^2(k-1) - (\sum_{k=1}^{t} y(k-1)u(k-1))^2$$

vilket medför att

$$\hat{\theta} = \begin{bmatrix} \hat{a}_1 \\ \hat{b}_0 \end{bmatrix} = \frac{1}{Q}\begin{bmatrix} \sum_{k=1}^{t} y(k-1)u(k-1)\sum_{k=1}^{t} u(k-1)y(k) - \sum_{k=1}^{t} u^2(k-1)\sum_{k=1}^{t} y(k-1)y(k) \\ \sum_{k=1}^{t} y^2(k-1)\sum_{k=1}^{t} u(k-1)y(k) - \sum_{k=1}^{t} u(k-1)y(k-1)\sum_{k=1}^{t} y(k-1)y(k) \end{bmatrix}$$

Dessa processparametrar skickas sedan till designen.

4.3 Design

Om vi nu sätter upp det önskade återkopplade systemet enligt

$A_m(q)y(t) = B_m(q)u_c(t)$

där

$A_m(q) = q + a_{m1}$

$B_m(q) = B^- B'_m = b_0 B'_m$

blir den diophantiska ekvationen

$(q + \hat{a}_1)R(q) + \hat{b}_0 S(q) = A_o(q)(q + a_{m1})$

tvingar vi sedan in nollstället i R och S polynomen får vi

$(q + \hat{a}_1)(q - 1)R'(q) + \hat{b}_0(q + 1)S'(q) = A_o(q)(q + a_{m1})$

Om vi nu istället bakar in (q-1) i A-polynomet kommer gradtalet på det att öka till två.

$A'(q) = (q + \hat{a}_1)(q - 1) = q^2 + (\hat{a}_1 - 1)q + (-\hat{a}_1) = a'_0 q^2 + a'_1 q + a'_2$

Detta leder till att även observerarpolynomet blir av gradtal två.

deg A_o >= 2 deg A' - deg A_m - deg B^+ - 1 = 2*2 - 1 - 0 - 1 = 2

Vi får nu följande ekvation

$$(q + \hat{a}_1)(q - 1)(r_0'q + r_1') + \hat{b}_0(q + 1)(s_0'q + s_1') = (q^2 + a_{o1}q + a_{o2})(q + a_{m1})$$

där

$$C(q) = (q^2 + a_{o1}q + a_{o2})(q + a_{m1}) = q^3 + (a_{o1}+a_{m1})q^2 + (a_{o1}a_{m1}+a_{o2})q + a_{m1}a_{o2}$$

$$= c_0q^3 + c_1q^2 + c_2q + c_3$$

$$R'(q) = (r_0'q + r_1')$$

$$S'(q) = (s_0'q + s_1')$$

och om vi bakar in (q+1) i B istället för R-polynomet fås

$$B'(q) = \hat{b}_0(q + 1) = b_0'q + b_1'$$

där i just detta fall

$$\hat{b}_0 = b_0' = b_1'$$

Det finns nu ett antal sätt att lösa den Diophantiska ekvationen på. Det naturligaste är kanske att sätta upp Sylvestermatrisen och med hjälp av Gausselimination lösa den. Vi har provat den nämnda metoden, men upptäckte senare att metoden med polynomreduktion var en metod som tidsmässigt var cirka tre gånger snabbare.

Polynomreduktionsmetoden:

Om vi utgår från nedanstående ekvationer

$$A'(z) = a_0' + a_1'z + a_2'z^2$$

$$B'(z) = b_0' + b_1'z$$

$$C(z) = c_0 + c_1z + c_2z^2 + c_3z^3$$

där ordningen på gradtalen i förhållande till koefficienterna har ändrats, p.g.a. att lösning av den Diophantiska ekvationen, med hjälp av polynomreduktionsmetoden, kräver det. Vi kan nu, enligt beskrivningen av metoden, testa de fyra villkoren. För att lättare följa vad som händer i de olika stegen, och för att hålla ordning på i vilken följd de utförs, inför vi tre vektorer:

val[]	där villkor 1-4 lagras beroende på i vilken ordning de används
exponent[]	där gradtalet på den tillhörande koefficienten lagras
koefficient[]	där koefficienten till det specifika villkoret lagras

START av algoritm

gradtalen är från början (r = 3 p = 2 q = 1) = (deg C deg A' deg B')

steg 0

villkor 1 (r >= p och C'<> 0 och A"<> 0) vilket ger

val[steg0] = [villkor1]
exponent[steg0] = [(c_3/a'_2)]
koefficient[steg0] = [1]

$$C'(z) = c_0 + c_1 z + c_2 z^2 + c_3 z^3 - (a_0' + a_1' z + a_2' z^2)\frac{c_3}{a_2'} z$$

$$= c_0 + (c_1 - \frac{a_0'}{a_2'} c_3)z + (c_2 - \frac{a_1'}{a_2'} c_3)z^2 = c_0 + c_1' z + c_2' z^2$$

(r = 2 p = 2 q = 1)

steg 1

villkor 1 (r >= p och C'<> 0 och A"<> 0) vilket ger

val[steg0 steg1] = [villkor1 villkor1]
exponent[steg0 steg1] = [(c_3/a'_2) (c'_2/a'_2)]
koefficent[steg0 steg1] = [1 0]

$$C'(z) = c_0 + c_1' z + c_2' z^2 - (a_0' + a_1' z + a_2' z^2)\frac{c_2'}{a_2'}$$

$$= (c_0 - \frac{a_0'}{a_2'} c_2') + (c_1' - \frac{a_1'}{a_2'} c_2')z = c_0' + c_1'' z$$

(r =1 p = 2 q = 1)

steg 2

villkor 2 (r >= q och C'<> 0 och B"<> 0) vilket ger

val[steg0 steg1 steg2] = [villkor1 villkor1 villkor2]
exponent[steg0 steg1 steg2] = [(c_3/a'_2) (c'_2/a'_2) (c''_1/b'_1)]
koefficent[steg0 steg1 steg2] = [1 0 0]

$$C'(z) = c_0' + c_1'' z - (b_0' + b_1' z)\frac{c_1''}{b_1'} = (c_0' - \frac{b_0'}{b_1'} c_1'') = c_0''$$

(r = 0 p = 2 q = 1)

steg 3

villkor 4 (p >= q och A"<> 0 och B"<> 0) vilket ger

val[steg0 steg1 steg2 steg3] = [villkor1 villkor1 villkor2 villkor4]
exponent[steg0 steg1 steg2 steg3] = [(c_3/a'_2) (c'_2/a'_2) (c''_1/b'_1) (a'_2/b'_1)]
koefficent[steg0 steg1 steg2 steg3] = [1 0 0 1]

$$A''(z) = a'_0 + a'_1 z + a'_2 z^2 - (b'_0 + b'_1 z)\frac{a'_2}{b'_1} z$$

$$= a'_0 + (a'_1 - \frac{b'_0}{b'_1} a'_2)z = a'_0 + a''_1 z$$

(r = 0 p = 1 q = 1)

steg4

villkor 3 (q >= p och B''<> 0 och A''<> 0) vilket ger

val[steg0 steg1 steg2 steg3 steg4] = [villk1 villk1 villk2 villk4 villk3]
exponent[st0 st1 st2 st3 st4] = [(c_3/a'_2) (c'_2/a'_2) (c''_1/b'_1) (a'_2/b'_1) (b'_1/a''_2)]
koefficent[steg0 steg1 steg2 steg3 steg4] = [1 0 0 1 0]

$$B''(z) = b'_0 + b'_1 z - (a'_0 + a''_1 z)\frac{b'_1}{a''_1} = (b'_0 - \frac{a'_0}{a''_1} b'_1) = b''_0$$

(r = 0 p = 1 q = 0)

steg5

villkor 2 (r >= q och C'<> 0 och B''<> 0) vilket ger

val[st0 st1 st2 st3 st4 st5] = [villk1 villk1 villk2 villk4 villk3 villk2]
exponent[st0 st1 st2 st3 st4 st5] = [(c_3/a'_2) (c'_2/a'_2) (c''_1/b'_1) (a'_2/b'_1) (b'_1/a''_2) (c''_0/b''_0)]
koefficent[steg0 steg1 steg2 steg3 steg4 steg5] = [1 0 0 1 0 0]

$$C'(z) = c''_0 - \frac{c''_0}{b''_0} b''_0 = 0$$

(r = 0 p = 1 q = 0)

steg6

villkor 4 (p >= q och A''<> 0 och B''<> 0) vilket ger

val[st0 st1 st2 st3 st4 st5 st6] = [villk1 villk1 villk2 villk4 villk3 villk2 villk4]
exponent[st0 st1 st2 st3 st4 st5 st6] = [(c_3/a'_2) (c'_2/a'_2) (c''_1/b'_1) (a'_2/b'_1) (b'_1/a''_2) (c''_0/b''_0) (a''_1/b''_0)]
koefficent[steg0 steg1 steg2 steg3 steg4 steg5 steg6] = [1 0 0 1 0 0 1]

$$C'(z) = c''_0 - \frac{c''_0}{b''_0} b''_0 A''(z) = a'_0 + a''_1 z - \frac{a''_1}{b''_0} b''_0 z = a'_0$$

(r = 0 p = 0 q = 0)

steg 7

villkor 3 (q >= p och B"<> 0 och A"<> 0) vilket ger

val[st0 st1 st2 st3 st4 st5 st6 st7] = [v1 v1 v2 v4 v3 v2 v4 v3]
exponent[st0 st1 st2 st3 st4 st5 st6 st7] = [(c_3/a'_2) (c'_2/a'_2) (c''_1/b'_1) (a'_2/b'_1) (b'_1/a''_2) (c''_0/b''_0) (a''_1/b''_0) (b''_0/a'_0)]
koefficent[steg0 steg1 steg2 steg3 steg4 steg5 steg6 steg7] = [1 0 0 1 0 0 1 0]

$$B''(z) = b''_0 - \frac{b''_0}{a'_0} a'_0 = 0$$

(r = 0 p = 0 q = 0)

Nu har två av polynomen blivit noll, i detta fall C' och B" , vilket medför att villkorstestet är klart. Det som återstår nu är att med hjälp av bakåtsubstitution beräkna R' och S'.

Bakåtsubstitution:

steg 7

villkor 3 ger att

$$X(z) = X' - Y\frac{b''_0}{a'_0} = 0 - 0\frac{b''_0}{a'_0} = 0$$

steg 6

villkor 4 ger att

$$Y(z) = Y' - X\frac{a''_1}{b''_0} z = 0 - 0\frac{a''_1}{b''_0} z = 0$$

steg 5

villkor 2 ger att

$$Y(z) = Y' - \frac{c''_0}{b''_0} = 0 - \frac{c''_0}{b''_0} = -\frac{c''_0}{b''_0}$$

steg 4

villkor 3 ger att

$$X(z) = X' - Y\frac{b_1''}{a_2''} = 0 - (-\frac{c_0''}{b_0''})\frac{b_0''}{a_0'} = \frac{c_0''}{a_0'}$$

steg 3

villkor 4 ger att

$$Y(z) = Y' - X\frac{a_2'}{b_1'}z = -\frac{c_0''}{b_0''} - \frac{c_0''}{a_0'}\frac{a_2'}{b_1'}z$$

steg 2

villkor 2 ger att

$$Y(z) = Y' + \frac{c_1''}{b_1'} = (-\frac{c_0''}{b_0''} - \frac{c_0''}{a_0'}\frac{a_2'}{b_1'}z) + \frac{c_1''}{b_1'} = (\frac{c_1''}{b_1'} - \frac{c_0''}{b_0''}) - \frac{c_0''}{a_0'}\frac{a_2'}{b_1'}z$$

steg1

villkor 1 ger att

$$X(z) = X' - \frac{c_2'}{a_2'} = \frac{c_0''}{a_0'} - \frac{c_2'}{a_2'}$$

steg 0

villkor 1 ger att

$$X(z) = X' - \frac{c_3}{a_2'}z = (\frac{c_0''}{a_0'} - \frac{c_2'}{a_2'}) - \frac{c_3}{a_2'}z$$

Här slutar algoritmen för metoden med polynomreduktion.

De resultat vi nu har fått fram

$$X(z) = (\frac{c_0''}{a_0'} - \frac{c_2'}{a_2'}) - \frac{c_3}{a_2'}z$$

$$Y(z) = (\frac{c_1''}{b_1'} - \frac{c_0''}{b_0''}) - \frac{c_0''}{a_0'}\frac{a_2'}{b_1'}z$$

resulterar i att

$$R'(q) = (\frac{c_0''}{a_0'} - \frac{c_2'}{a_2'})q + (-\frac{c_3}{a_2'}) = r_0'q + r_1'$$

och

$$S'(q) = \left(\frac{c_1''}{b_1'} - \frac{c_0''}{b_0''}\right)q - \frac{c_0''}{a_0'}\frac{a_2'}{b_1'} = s_0'q + s_1'$$

Vi vet, från beskrivningen av designen, att

$$R(q) = (q-1)R'(q) = (q-1)(r_0'q + r_1') = r_0'q^2 + (r_1' - r_0')q + (-r_1') = r_0q^2 + r_1q + r_2$$

$$S(q) = (q+1)S'(q) = (q+1)(s_0'q + s_1') = s_0'q^2 + (s_1' + s_0')q + s_1' = s_0q^2 + s_1q + s_2$$

där $r_0 = 1$. Polynomet T(q) beräknas sedan som

$$T(q) = B_m' A_o(q) = \frac{A_m(1)}{B(1)} A_o(q) = \left(\frac{1 + a_{m1}}{b_0 + b_1}\right)(q^2 + a_{o1}q + a_{o2})$$

$$= (t_0q^2 + t_1q + t_2)$$

där

$$t_0 = B_m' \quad , \quad t_1 = B_m' a_{o1} \quad \text{och} \quad t_2 = B_m' a_{o2}$$

R(q), S(q) och T(q) polynomen skickas nu från designalgoritmen till regulatorn.

4.4 Regulator

Eftersom vårt val att av regulator är av polynomtyp

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

blir regulatorn för detta exempel

$$(q^2 + r_1q + r_2)u(t) = (t_0q^2 + t_1q + t_2)u_c(t) - (s_0q^2 + s_1q + s_2)y(t)$$

eller

$$(1 + r_1q^{-1} + r_2q^{-2})u(t) = (t_0 + t_1q^{-1} + t_2q^{-2})u_c(t) - (s_0 + s_1q^{-1} + s_2q^{-2})y(t)$$

Styrsignalen får då detta utseende

$$u(t) = -r_1u(t-1) - r_2u(t-2) + t_0u_c(t) + t_1u_c(t-1) + t_2u_c(t-2) - s_0y(t) + s_1y(t) + s_2y(t)$$

För att även ha ett skydd mot integratoruppvridning får vi införa en begränsning

$$v(t) = -r_1u(t-1) - r_2u(t-2) + t_0u_c(t) + t_1u_c(t-1) + t_2u_c(t-2) - s_0y(t) + s_1y(t) + s_2y(t)$$

$$u(t) = \text{sat}\,(v(t))$$

där u(t) = sat(v) förklaras i regulatorbeskrivningen (avsnitt 3.4).

4.5 Sammanfattning

Vi hoppas att detta exempel belyser arbetsgången för de beräkningar som krävs i en adaptiv regulator och även ger en bättre förståelse för hur en regulator skall implementeras. Beräkningarna för algoritmer av detta slag, blir lätt väldigt stora. Exemplet är endast av första ordningen. Tänk er samma genomgång av problem med en processmodell av t.ex. sjunde ordningen. Det är då inte svårt att förstå, att dessa beräkningar är tidsödande även för en signalprocessor. Observera också att i flera situationer måste många olika fall beaktas.

5 Maskinvara

5.1 Uppkoppling

Regulatorn är implementerad med en MacIntosh som försetts med två kort från National Instruments: NB-DSP-2300 (Signalprocessor) och NB-MIO16 (A/D-, D/A-omvandlare). Se fig 5.1.

Signalprocessorkortet innehåller flyttalsprocessorn TMS320C30. Den har en klockfrekvens på 33 MHz och kan t.ex. utföra en multiplikation **och** en addition under samma klockcykel. Detta är speciellt användbart vid användning av långa filter. Vi använder denna finess i vårt antialiasfilter, som är 200 tappar långt. I/O-kortet innehåller förstärkare, A/D-D/A-omvandlare och timingkretsar.

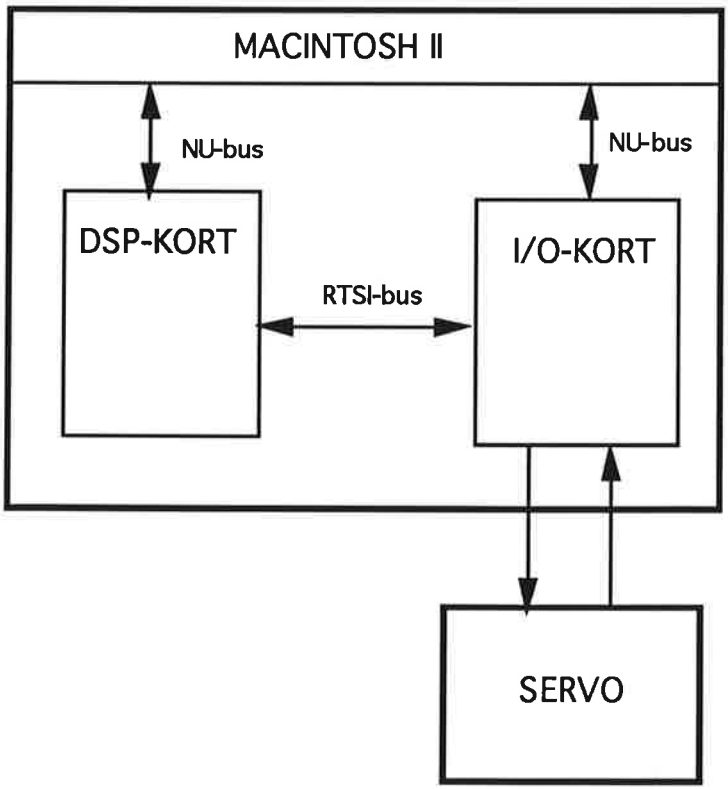

Fig 5.1 Hårdvara

Vi använder inkanal 1 till mätsignalen och utkanal 1 till styrsignalen.
Maximal spänning är +/- 10 V för båda kanalerna.
DSP-kortet sitter på kortplats 3 och I/O-kortet på plats 5.

5.2 Avbrott

Signalprocessorns inbyggda timer skickar ett timeravbrott med ett mellanrum som bestäms av samplingshastigheten. I timer-avbrottsrutinen startas en A/D-omvandling genom att skriva i ett "Start-convert"-register via NuBussen. När en A/D-omvandling är klar skickas ett avbrott via RTSI-bussen till DSP-kortet. Detta görs för att kunna utföra beräkningar i huvudprogrammet under tiden omvandling sker. Figur 5.2 visar översiktligt beräkningarnas tidssekvens.

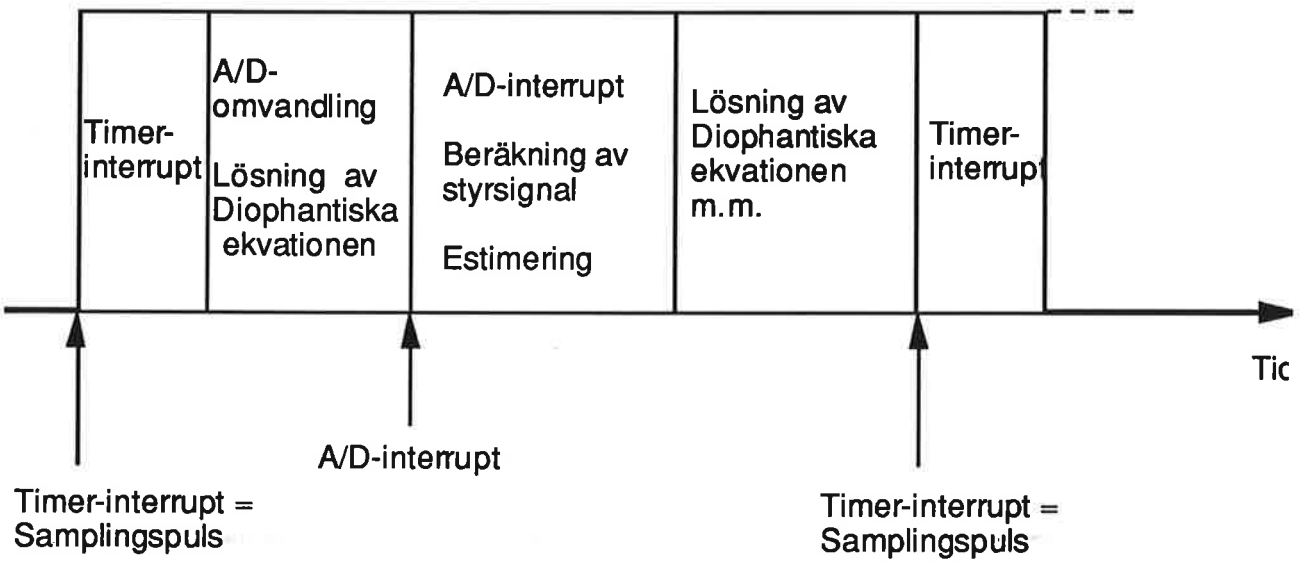

Fig 5.2 Timingdiagram

6 Programvara

Mjukvaran för DSP'n är uppdelad i ett antal moduler. Det finns en programmodul för varje block i blockschemat (Regulator, Design, Estimator, Filter), samt några för hårdvaran (I/O, Hardware, LabViewVariables). Dessutom använder vi oss av ett par färdiga moduler från National Instruments, där bl.a. typer, kortspecifika konstanter samt en omvandlingsrutin för flyttal finns deklarerade(Analog, DSP, Conversion).

Alla programmoduler är separatkompilerade och använder sig av header-filer där gemensamma variabler, konstanter och funktioner finns deklarerade. Detta har sparat mycket tid vid kompileringen och gett ett strukturerat program. Det är därför förhållandevis lätt att byta ut en modul, om man skulle önska det.

Mjukvaran för MacIntosh'en består endast av en rutin som sköter om överföring av parametrar och är mycket enkel.

6.1 DSP

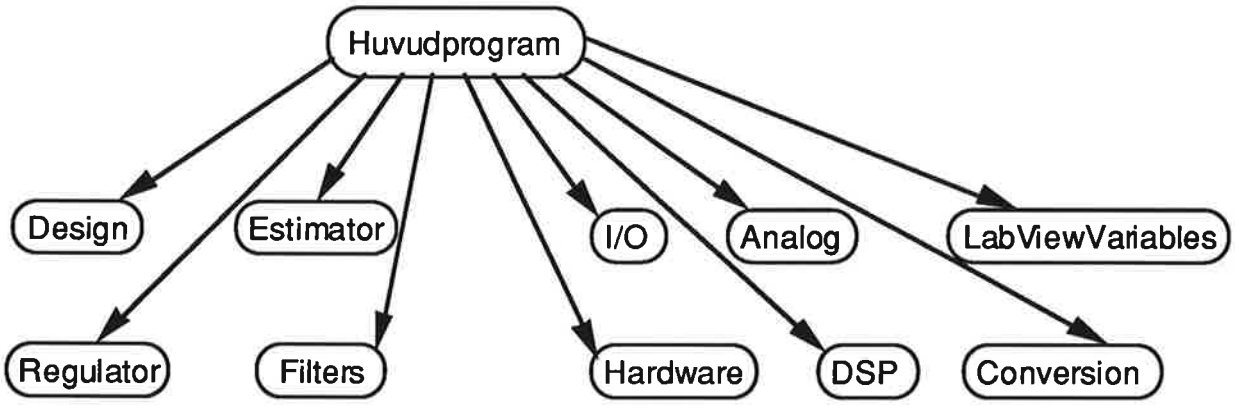

Fig 6.1 Mjukvara

För att lättare kunna ändra, och även få en överblick se fugur 6.1, förklaras här modulerna samt dess in- och ut-parametrar. Modulerna är helt oberoende av varandra.

Modul:	**innehåller:**
Estimator	init_estimator (a_deg, b_deg, lambda, theta)
	där a_deg och b_deg är gradtalen på processen, lambda är glömskefaktorn, och där theta, som är de skattade processparametrarna, knyts till en pekare.
	estimate (y, u)
	där y är den uppmätta processutsignalen och u är styrsignalen. Theta uppdateras i denna procedur, men skickas inte in och ut (se init_estimator).

Design

init_design (am, ao, am_deg, ao_deg, a_deg, b_deg)

där am är modellreferenspolynomet, ao är observerar-polynomet, och där de sista fyra inparametrarna är gradtal.

new_design (theta, regulator)

där theta är det senaste estimatet från estimeraren och regulator är den regulatorstruktur som kommer ut från new_design.

Regulator

init_regulator (initial_PID, K, T_i, T_D, h,AntiWindUp)

där initial_PID anger om en initieringsregulator ska användas. I så fall beräknas en diskret PID-regulator med hjälp av K,Ti,Td och sampeltiden h. AntiWindUp anger om skydd mot integratoruppvridning ska användas.

compute_u_new (y_ref, y)

där y_ref, referensvärdet, och y, processutsignalen, används för att beräkna styrsignalen u_new.

change_regulator (new_regulator)

där new_regulator är den senast uträknade regulatorn från new_design. I denna procedur sker utbyte av gamla till nya regulatorparametrar genom att stänga av avbrotten.

Hardware

init_interrupt_vectors (c_int02, c_int01)

där avbrottsadresserna läggs in på c_int02 resp. c_int01.

init_timer (sample_time, oversampling)

där sample_time är samplingstiden för systemet och oversampling anger vilken översampling som skall användas av filtren.

reset_RTSI_EN_bit ()

nollställer en bit i kontrollregistert p.g.a. att datorn efter uppstart alltid ettställer denna bit som orsakar konstant avbrott, vilket leder till att systemet låser sig.

enable_A_D_interrupt ()

möjliggör påverkan av A/D-avbrott.

enable_timer_interrupt ()

möjliggör påverkan av timer-avbrott.

disable_interrupts ()

stänger av avbrotten för att kunna avsluta programmet.

I_O

init_I_O ()

initierar MIO16 kortet

start_conversion ()

startar en A/D översättning av processutsignalen.

read_A_D ()

läser in och skalar om ett mätvärde från en kö på MIO16.

Filters

FltCreateAB (a, a_deg, b, b_deg)

returnerar ett filter på direkt form där a och b är polynom till filtret.

FltDCreatePFoH (up, down, alfa)

skapar ett "first order and hold" filter med olinjär fas men utan fördröjning. Derivatan upp till nästa sampel är estimerad med alfa, som är en viktad faktor.

FltDCreateFIRLPS (längd, nyc, upp, ner, window)

FIR-filter med fönster. Returnerar ett lågpassfilter med nyc som den önskade brytfrekvensen. Stegsvaret utnyttjar ett fönster window, där window är ett av BOXCAR, BARTLETT HANNING, HAMMING eller BLACKMAN

FltDFree (filter)

frigör allokerat utrymme knutet till filtret filter.

FltDFIRGetSample (filter)

tar ett nytt sampel ur filtret filter, vilket antas vara av typen FIR. Använder sig bara av nämnarpolynomet B.

FltDFIRPutSample (filter, new)

lägger in ett nytt sampel new till filtret filter. Utnyttjar endast B-polynomet (FIR).

FltDGetSample (filter)

tar ett nytt sampel ur filtret filter. Denna funktion är långsammare än de för FIR-fiter.

FltDPutSample (filter, new)

lägger in ett nytt sampel new till filtret filter.

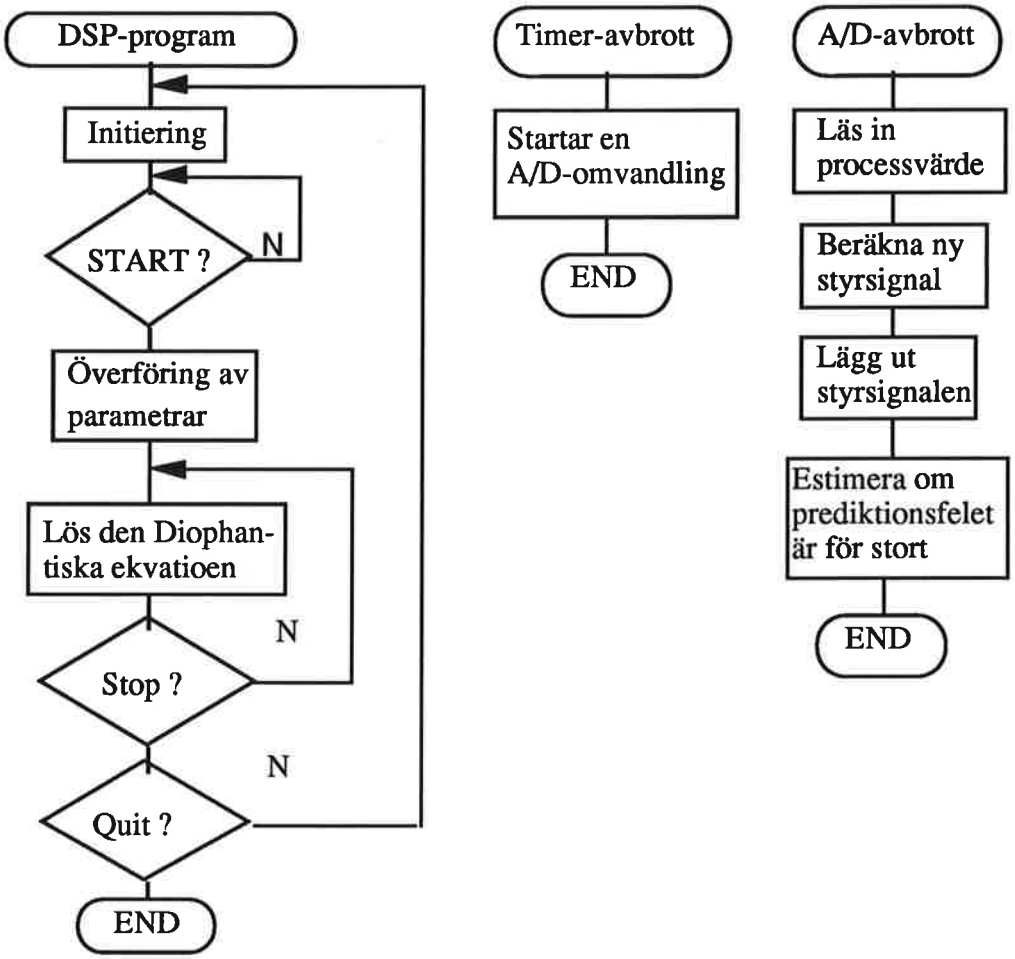

Fig 6.2

Programmet börjar med att vänta på att "START"-knappen ska tryckas in, se figur 6.2. Parametrarna läses då in och flyttalen omvandlas från IEEE-format till det format som DSP'n använder. Regulatorn reglerar processen tills "STOP"-knappen trycks in, och parametrarna kan ändras. I början använder regulatorn en användarspecificerad PID-regulator, så att estimatorn hinner estimera processen ordentligt. Efter en inställd tid övergår regulatorn till att bli självinställande.

6.2 LabView, allmänt.

Programmet som körs i signalprocessorn styrs från MacIntosh'en och programmet "LabView". LabView är ett program som man snabbt, snyggt och enkelt kan bygga upp t.ex. datainsamlingsprogram, styrprogram, beräkningsprogram mm. Programmens grafiska gränssnitt till användaren kan fås mycket snygga och pedagogiska, då det finns många valmöjligheter när det gäller presentation och inhämtning av data. Programspråket, i LabView, heter "G" och består av en stor mängd grafiska symboler. Det finns symboler för alla tänkbara saker som man vill göra i ett program : Aritmetiska och logiska operationer, omvandlingar, styrning av programflödet, matriser, konstanter, filhantering och mycket, mycket mera. Symbolerna kopplas ihop med hjälp av trådar och bestämmer på så vis hur de olika delarna skall fungera tillsammans.
När utveckling pågår i ett G-program, finns två fönster att arbeta med. I ett fönster syns utsidan på "apparaten", med alla knappar, rattar, visare och annat som behövs för att använda den. I det andra fönstret finns "innanmätet" i apparaten, d.v.s. programmet med dess grafiska symboler samt trådarna som kopplar ihop dem.
För att inte programmen ska bli alldeles för ostrukturerade, kan även underprogram göras. I LabView kallas dessa underprogram för virtuella instrument (VI). De virtuella instrumenten är uppbyggda på samma sätt som huvudprogrammet (huvudinstrumentet), har en egendefinierad symbol samt ut- och ingångar. I huvudprogrammet används VI'ns symbol precis som de andra symbolerna.
Om det behövs snabbare, mer komplicerade eller mer hårdvarunära program, kan G-programet knytas till ett C-program. Symbolen som då används kallas Code Interface Node (CIN), och består av ett huvud och valfritt antal in- eller utgångar. C-programmet organiseras i ett antal rutiner : CINRun, CINAbort, CINInit, CINDispose, CINLoad och CINSave. Koden i CINInit exekveras i början på varje programkörning. I CINRun läggs den kod som utförs varje gång noden anropas. Denna rutin tar emot insignalerna, utför de önskade operationerna och skickar tillbaka utsignalerna. Om programmet avbryts, exekveras koden i CINAbort. Om ett VI stängs eller på annat sätt tas bort ur minnet, körs CINDispose. När ett VI sparas eller laddas in körs CINSave resp. CINLoad.
LabView's styrka är dess möjligheter att få ett snyggt gränssnitt mot användaren och möjligheten att snabbt få igång ett enkelt system. Nackdelarna är att det snabbt blir stora och ostrukturerade program som är svåra att läsa. Symbolerna exekveras dessutom i "fel" ordning. Datorn börjar i slutet av kedjan av symboler med att kontrollera vilka insignaler den sista symbolen behöver. Den går sen baklänges och kontrollerar symbol för symbol, ända tills den hittar ett av användaren givet värde eller en konstant. Först då kan datorn lösa hela kedjan av operationer. Detta verkar ju logiskt, men när även programflödet styrs med hjälp av symboler uppstår lätt förvirring.

6.3 LabView, vårt program

Det första programmet skall göra, är att ladda ner DSP-programmet i DSP'n. Till detta finns ett färdigt VI : "Download DSP code". Inparametrar till detta VI är namnet på programmet, vilken slot som DSP-kortet sitter i, samt om programmet skall köras igång eller ej. Om nedladdningen blev lyckad visas felmeddelandet "0", dvs "inget fel". Därefter går datorn in i en oändlig slinga, som inte avbryts förrän man trycker in "QUIT"-knappen och därmed avslutar både DSP- och LabViewprogrammen.
Till en så speciell uppgift som att lägga in parametrarna på bestämda positioner i det gemensamma minnet finns inga färdiga symboler. I den oändliga slingan ingår därför en CIN som, när "START"-knappen trycks in, hämtar de inställda parametrarna från respektive virtuella instrument och lägger de på rätt plats i minnet. Rutinen hämtar även parametrar från minnet och lägger de till ett skiftregister. Skiftregistret behåller värdena till nästa varv i slingan kör och skickar dem då till respektive virtuella instrument för visning.

Knapparna "START", "STOP" och "QUIT" skickas också över till minnet, men bara om de är intryckta, d.v.s. de booleska variablerna sanna. DSP-programmet sköter om uppgiften att nollställa variablerna innan de används.
För att de virtuella instrumentens frontpaneler ska öppnas när dess knapp på huvudfrontpanelen är intryckt, men inte annars, måste en specialmanöver göras. En sub-VI kan konfigureras från huvud-VI'n, med avseende på hur dess frontpanel ska uppföra sig. Panelen kan visa sig om sub-VI'n blir *inladdad*, om den blir *anropad* eller *inte alls*. VI'n tas med i diagrammet två gånger, med två olika konfigurationer. En där panelen inte alls öppnas och en där panelen öppnar sig när den blir anropad. Genom att först testa om knappen är intryckt eller inte väljs vilken av de två virtuella instrumenten som skall anropas. När parametrarna ändras i det ena instrumentet, slår emellertid ändringen igenom till det andra eftersom det egentligen är samma instrument. Parameterändringarna kvarstår därför även när sub-VI'ns panel är stängd.

6.4 LabView, införande av nya parametrar

Om någon modul i programmet skall bytas ut eller på annat sätt ändras, måste kanske nya parametrar överföras från LabView. Då detta är en något omständlig procedur, och det är lätt att glömma något led, lämnar vi här en fullständig beskrivning av förfarandet. Beskrivningen förutsätter att en ny parameter skall överföras från LabView till DSP-programmet, men kan lätt ändras till att beskriva borttagning av en parameter eller införandet av en helt ny modul. Skall parametern överföras från DSP_programmet till LabView för presentation är det en del extra saker att tänka på.

Införande av ny parameter (LabView till DSP)

1. Skapa en ny ratt,knapp eller motsvarande på panelen i rätt sub-VI.
2. Kopiera knappen till panelens utgångscluster.(Krympt och gömt långt ner eller till höger)
3. Gör vid behov om knappen till en indikator. (Görs automatiskt om det redan fanns andra knappar i clustret)
4. Äppleklicka på kanten av clustret och välj "cluster order".
5. Kontrollera att den nya knappen ligger på rätt plats (sist).
6. Tag fram instrumentets diagram. (apple-F)
7. Lägg till ett extra element i den gula bundle-symbolen.(äppleklicka på sista elementet och välj "add element")
8. Dra en tråd från knappen till det nya elementet i bundle-symbolen.
9. Knyt utgångsclustret till rätt utgång genom att:
    9a. Ändra verktyget till trådrulle.
    9b. Äppleklicka på panelens ikon och välj "Show connector pane" (Om instrumentet har en utgång men ingen ingång syns nu endast en vit fyrkant)
    9c. Klicka med trådrullen först på utgången och sen på utgångsclustret.
10. Spara.
11. Hämta in "LabView_Variables.h"
12. Lägg till knappen i "parameter_type". (typen long för booleska variabler)
13. Spara.
14. Hämta in "STR.c"
15. Lägg till knappen i rätt moduls utparameterlista (typen short för booleska variabler)
13. I "CINRun" ska parametern överföras från LabView till det gemensamma minnet. (typecast för booleska variabler)
14. Spara.
15. Kasta STRc.cdvi i papperskorgen.
16. Kompilera och länka STR.c.
17. Hämta in huvudinstrumentets diagram.
18. Äppleklicka på den gula CIN-symbolens huvud och välj "Load code resource...".
19. Dubbelklicka på "STRc.cdvi".

20. Spara huvudinstrumentet.
21. Hämta huvudprogrammet "RST.c".
22. Deklarera en ny variabel under "Variabler som kommer från LabView".
23. Hämta in "LabView_Variables.c"
24. I "Get_Parameters" ska parametern överföras från det gemensamma minnet till den nya variabeln i huvudprogrammet. (flyttal skall först gå igenom konvertering i "to_float")
25. Nu kan den nya variabeln användas på önskat sätt i huvudprogrammet.
26. Spara.
27. Kompilera och länka med kommandot "emake"

Införande av ny variabel (DSP till LabView)

1. Skapa en ny indikator på panelen i rätt sub-VI.
2. Kopiera indikatorn till panelens ingångscluster.
3. Gör vid behov om knappen till en "Control".
4. Äppleklicka på kanten av clustret och välj "cluster order".
5. Kontrollera att den nya indikatorn ligger på rätt plats.
6. Tag fram instrumentets diagram.
7. Lägg till ett extra element i den gula unbundle-symbolen.
8. Dra en tråd från indikatorn till det nya elementet i unbundle-symbolen.
9. Knyt ingångsclustret till rätt ingång.
10. Klipp ut ingångsclustret från panelen för senare användning.
11. Spara.
12. Hämta in huvudinstrumentet "STR"
13. Klistra in ingångsclustret från sub-VI'n någonstans nära den gula CIN-noden.
14. Tag bort alla trådar som har med ingången till sub-VI'n att göra. (4 st : utgång från CIN,utgång från skiftregister, ingång till sub-VI'n TRUE resp. FALSE)
15. Dra en tråd från det just inklistrade ingångsclustret till typingången på CIN-noden (Ingången med en punkt och en cirkel)
16. Dra trådarna som togs bort i punkt 14 på nytt.
17. Radera ingångsclustret och dess tråd.
18. Hämta in "LabView_Variables.h"
19. Lägg till indikatorn i "value_type". (typen long för booleska variabler)
20. Spara.
21. Hämta in "STR.c"
22. Lägg till indikatorn i rätt moduls inparameterlista (typen short för booleska variabler)
23. I "CINRun" ska variabeln överföras till LabView från det gemensamma minnet varje gång CINRun körs (typecast för booleska variabler)
24. Spara.
25. Kasta STRc.cdvi i papperskorgen.
26. Kompilera och länka STR.c.
27. Hämta in huvudinstrumentets diagram.
28. Äppleklicka på den gula CIN-symbolens huvud och välj "Load code resource...".
29. Dubbelklicka på "STRc.cdvi".
30. Spara huvudinstrumentet.
31. Hämta huvudprogrammet "RST.c".
32. Deklarera en ny variabel och en pekare under "Variabler som kommer från LabView".
33. Hämta in "LabView_Variables.c"
34. I "Get_pointers" tilldelas pekaren adressen till variabeln i det gemensamma minnet.
35. Spara.
36. Nu kan den nya variabeln användas i huvudprogrammet.
37. Överföring sker genom "New_pointer* = New_Variable". (flyttal skall genomgå "to_IEEE")
38. Spara.
39. Kompilera och länka med kommandot "emake"

6.5 LabView, användning

LabView-programmet laddas in genom att dubbelklicka på "STR" i Finder. När frontpanelen efter en stund kommer upp på skärmen, startas programmet genom att trycka på symbolen =>, välja "Run VI" i Operate-menyn eller trycka "apple"-R.
Genom att trycka på en av knapparna på frontpanelen bredvid modulerna kommer respektive moduls frontpanel upp på skärmen och dess parametrar kan ställas till önskat värde. Om parameterns rattgradering inte räcker till, kan den ändras genom klicka på max- eller minvärdet och skriva in ett nytt värde. Fönstret stängs på vanligt sätt, med ett klick i fyrkanten i fönstrets övre vänstra hörn.
När alla parametrar är satta till önskade värden kan "START"-knappen tryckas in. Parametrarna förs då över till DSP-programmet och regulatorn körs. Parametrarna kan ändras endast då regulatorn är stoppad med "STOPP"-knappen. Programmet avslutas med "QUIT"-knappen. Om parametrarna ska sparas till nästa programkörning, väljs "Make current values default" i Operatemenyn. När fönstret stängs frågar datorn om ändringarna skall sparas.
En utförlig manual finns i appendix D.

7 Experiment

Parallellt med programutvecklingen, har vi utfört experiment på en process av andra ordningen. Processen bestod av en likströmsmotor, en metallcylinder som ökar tröghetsmomentet och en vinkelgivare. Förstärkningen på processen har varierats dels med en potentiometer, dels med en omkopplare som halverar förstärkningen.
För att följa programflödet har vi utnyttjat en extra utgång kopplat till oscilloskopet. Genom att ändra utspänningen vid bestämda punkter i programmet, har vi kunnat kontrollera avbrott, synkronisering och tidsåtgång för varje rutin.
Mycket tid har lags ner på förbättring av estimatorn. Oscilloskopet har använts för att studera och jämföra insvängningsförloppen för de olika förändringarna i estimatorn.
De olika programmodulerna har först testats var för sig för att sedan bit för bit plockas in i det färdiga programmet.

Referenser

Johansson R.(1990): *Processidentifiering*, Kurslitteratur, Institutionen för reglerteknik, Lunds Tekniska Högskola.

Ljung L och Söderström T (1983): *Theory and Practice Recursive of identification,* MIT press, Cambridge MA

Salomonsson G. m.f (1987): *Tidsdiskreta Kretsar och signaler*, Institutionen för Teletransmissionsteori, Lunds Tekniska Högskola.

Åström K.J. och Wittenmark B.(1989): *Computer Controlled systems*, Prentice Hall, New York

Åström K.J. och Wittenmark B.(1989): *Adaptive Control* , Addison-Wesley,Massachusetts

Bengt Schmidtbauer (1988): *Analog och digital reglerteknik*, Student-litteratur, Lund

Appendix A. Matrisinversions lemma

Låt A, C, och $C^{-1}+DA^{-1}B$ vara ickesingulära kvadratmatriser. Då är

$$(A + BCD)^{-1} = A^{-1} - A^{-1}B(C^{-1} + DA^{-1}B)^{-1}DA^{-1}$$

Bevis:

Genom direkt substitution finner vi att

$$\begin{aligned}(A + BCD)(A^{-1} - A^{-1}B(C^{-1} + DA^{-1}B)^{-1}DA^{-1}) \\ &= I + BCDA^{-1} - B(C^{-1} + DA^{-1}B)^{-1}DA^{-1} - BCDA^{-1}B(C^{-1} + DA^{-1}B)^{-1}DA^{-1} \\ &= I + BCDA^{-1} - BC(C^{-1} + DA^{-1}B)(C^{-1} + DA^{-1}B)^{-1}DA^{-1} \\ &= I + BCDA^{-1} - BCDA^{-1} \\ &= I\end{aligned}$$

Appendix B. Beräkning av väntevärde och kovarians

Låt θ och y vektorerna vara Gaussiska variabler med väntevärde

$$E\begin{bmatrix} y \\ \theta \end{bmatrix} = \begin{bmatrix} m_y \\ \theta^0 \end{bmatrix}$$

och kovariansen

$$\mathrm{cov}\begin{bmatrix} y \\ \theta \end{bmatrix} = \begin{bmatrix} R_y & R_{y\theta} \\ R_{\theta y} & R_\theta \end{bmatrix} = R$$

där $R_{\theta y} = R^T_{y\theta}$. Väntevärdet av θ givet y ger

$$E(\theta/y) = \theta^0 + R_{\theta y}R_y^{-1}\,(y - m_y)$$

och kovariansen

$$\mathrm{cov}\,(\theta/y) = R_{\theta/y} = R_\theta - R_{\theta y}R_y^{-1}R_{y\theta}$$

En positiv matris R kan skrivas som

$$R = \rho\begin{bmatrix} 1 & 0 \\ K & L_\theta \end{bmatrix}\begin{bmatrix} D_y & 0 \\ 0 & D_\theta \end{bmatrix}\begin{bmatrix} 1 & 0 \\ K & L_\theta \end{bmatrix}^T$$

där D_θ och D_y är diagonalmatriser och L_θ är en lägre triangulär matris. Då blir

$$R_{\theta y}R_y^{-1} = K$$

och

$$R_{\theta/y} = \rho L_\theta D_\theta L_\theta^T$$

Bevis:

Vi visar först att vektorn z definierad som

$$z = \theta - \theta^0 - R_{\theta y}R_y^{-1}(y - m_y)$$

har ett väntevärde lika med noll, är oberoende av y, och har kovariansen

$$R_z = R_\theta - R_{\theta y}R_y^{-1}R_{y\theta}$$

Väntevärdet blir noll enligt

$$E\{z(y - m_y)^T\} = E\{(\theta - \theta^0)(y - m_y)^T - R_{\theta y}R_y^{-1}\,(y - m_y)(y - m_y)^T\}$$

$= R_{\theta y} - R_{\theta y}R_y^{-1}R_y = 0$

Detta visar att variablerna z och y är okorrelerade och eftersom de är Gaussiska även oberoende. Kovariansen R_z blir

$$E\{zz^T\} = E\{((\theta - \theta^0) - R_{\theta y}R_y^{-1}(y - m_y))((\theta - \theta^0) - R_{\theta y}R_y^{-1}(y - m_y))^T\}$$

$$= E\{(\theta - \theta^0)(\theta - \theta^0)^T - R_{\theta y}R_y^{-1}(\theta - \theta^0)(y - m_y)^T - R_{\theta y}R_y^{-1}(y - m_y)(\theta - \theta^0)^T +$$

$$+ R_{\theta y}R_y^{-1}R_{\theta y}R_y^{-1}(y - m_y)(y - m_y)^T\}$$

$$= R_\theta - R_{\theta y}R_y^{-1}R_{\theta y} - R_{\theta y}R_y^{-1}R_{y\theta} + R_{\theta y}R_y^{-1}R_{\theta y}R_y^{-1}R_y$$

$$= R_\theta - R_{\theta y}R_y^{-1}R_{y\theta}$$

Eftersom $z = \text{cov}(\theta/y)$ innebär detta att

$$\text{cov}(\theta/y) = R_\theta - R_{\theta y}R_y^{-1}R_{y\theta}$$

vilket skulle bevisas. Kovariansmatrisen R kan skrivas som

$$R = \text{cov}\begin{bmatrix} y \\ \theta \end{bmatrix} = \begin{bmatrix} R_y & R_{y\theta} \\ R_{\theta y} & R_\theta \end{bmatrix} = \rho \begin{bmatrix} 1 & 0 \\ K & L_\theta \end{bmatrix} \begin{bmatrix} D_y & 0 \\ 0 & D_\theta \end{bmatrix} \begin{bmatrix} 1 & 0 \\ K & L_\theta \end{bmatrix}^T$$

$$= \rho \begin{bmatrix} D_y & D_y K^T \\ KD_y & L_\theta D_\theta L_\theta^T + KD_y K^T \end{bmatrix}$$

Identifikation ger

$R_y = \rho D_y$

$R_{\theta y} = \rho K D_y$

$R_\theta = \rho(L_\theta D_\theta L_\theta^T + KD_y K^T)$

vilket i sin tur innebär att

$R_{\theta y}R_y^{-1} = K$

och att

$$\text{cov}(\theta/y) = R_{\theta/y} = R_\theta - R_{\theta y}R_y^{-1}R_{y\theta} = \rho(L_\theta D_\theta L_\theta^T + KD_y K^T) - \rho KD_y K^T$$

$$= \rho L_\theta D_\theta L_\theta^T = \rho P$$

Appendix C. Dyadisk Dekomposition

Givet vektorerna

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

$b = [1\ b_1 \ldots b_n]^T$

och skalärerna α och β, ska vi hitta nya vektorer

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

$\tilde{b} = [0\ \tilde{b}_1 \ldots \tilde{b}_n]^T$

så att

$$\alpha aa^T + \beta bb^T = \tilde{\alpha}\tilde{a}\tilde{a}^T + \tilde{\beta}\tilde{b}\tilde{b}^T$$

Om detta problem kan lösas, kan vi utföra transformationen av

$$R = \begin{bmatrix} 1 & \varphi^T L \\ 0 & L \end{bmatrix} \begin{bmatrix} \sigma^2 & 0 \\ 0 & D \end{bmatrix} \begin{bmatrix} 1 & 0 \\ L^T\varphi & L^T \end{bmatrix}$$

till

$$R = \begin{bmatrix} 1 & 0 \\ K & \tilde{L} \end{bmatrix} \begin{bmatrix} \tilde{\sigma}^2 & 0 \\ 0 & \tilde{D} \end{bmatrix} \begin{bmatrix} 1 & K^T \\ 0 & \tilde{L}^T \end{bmatrix}$$

genom att upprepa förfarandet av metoden.

Ekvationen

$$\alpha aa^T + \beta bb^T = \tilde{\alpha}\tilde{a}\tilde{a}^T + \tilde{\beta}\tilde{b}\tilde{b}^T$$

kan skrivas som

$$\alpha \begin{bmatrix} 1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \begin{bmatrix} 1 & a_2 & \cdots a_n \end{bmatrix} + \beta \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \begin{bmatrix} b_1 & b_2 & \cdots b_n \end{bmatrix}$$

$$= \tilde{\alpha} \begin{bmatrix} 1 \\ \tilde{a}_2 \\ \vdots \\ \tilde{a}_n \end{bmatrix} \begin{bmatrix} 1 & \tilde{a}_2 & \cdots \tilde{a}_n \end{bmatrix} + \tilde{\beta} \begin{bmatrix} \tilde{b}_1 \\ \tilde{b}_2 \\ \vdots \\ \tilde{b}_n \end{bmatrix} \begin{bmatrix} \tilde{b}_1 & \tilde{b}_2 & \cdots \tilde{b}_n \end{bmatrix}$$

Element (1,1) ger ekvationen

$$\alpha + \beta b_1^2 = \tilde{\alpha}$$

Element (1,k) ger, för k>1, ekvationen

$$\alpha a_k + \beta b_1 b_k = \tilde{\alpha}\tilde{a}_k$$

Addera och subtrahera

$$\beta b_1^2 a_k$$

till ovanstående ekvation ger

$$\tilde{\alpha}\tilde{a}_k = \alpha a_k + \beta b_1 b_k + \beta b_1^2 a_k - \beta b_1^2 a_k = (\alpha + \beta b_1^2)a_k + \beta b_1 b_k - \beta b_1^2 a_k$$

vilket i sin tur ger

$$\tilde{a}_k = a_k + \frac{\beta b_1}{\tilde{\alpha}} (b_k - b_1 a_k)$$

Vi har nu sett till att

$$\tilde{\alpha} \text{ och } \tilde{a}_k$$

går att lösa. Det återstår att beräkna

$$\tilde{\beta} \text{ och } \tilde{b}_k$$

Elementet (k,l) ger ekvationen

$$\alpha a_k a_l + \beta b_k b_l = \tilde{\alpha}\tilde{a}_k\tilde{a}_l + \tilde{\beta}\tilde{b}_k\tilde{b}_l = \frac{(\alpha a_k + \beta b_1 b_k)(\alpha a_l + \beta b_1 b_l)}{\tilde{\alpha}} + \tilde{\beta}\tilde{b}_k\tilde{b}_l$$

eftersom

$$\tilde{\alpha}\tilde{a}_k = \alpha a_k + \beta b_1 b_k$$

$$\tilde{a}_l = \frac{1}{\tilde{\alpha}} (\alpha a_l + \beta b_1 b_l)$$

Multiplicerar vi båda sidor med $\tilde{\alpha}$ fås

$$\tilde{\alpha}(\alpha a_k a_l + \beta b_k b_l) = (\alpha a_k + \beta b_1 b_k)(\alpha a_l + \beta b_1 b_l) + \tilde{\alpha}\tilde{\beta}\tilde{b}_k\tilde{b}_l$$

$$\rightarrow \quad (\alpha + \beta b_1^2)(\alpha a_k a_l + \beta b_k b_l) = (\alpha a_k + \beta b_1 b_k)(\alpha a_l + \beta b_1 b_l) + \tilde{\alpha}\tilde{\beta}\tilde{b}_k\tilde{b}_l$$

Utvecklar vi detta blir

$$\alpha^2 a_k a_l + \alpha\beta b_k b_l + \alpha\beta b_k b_l b_1^2 + \beta^2 b_k b_l b_1^2 = \alpha^2 a_k a_l + \alpha\beta a_k b_1 b_l + \alpha\beta a_l b_1 b_k + \beta^2 b_k b_l b_1^2 + \tilde{\alpha}\tilde{\beta}\tilde{b}_k\tilde{b}_l$$

$$\rightarrow \quad \alpha\beta b_k b_l + \alpha\beta b_k b_l b_1^2 = \alpha\beta a_k b_1 b_l + \alpha\beta a_l b_1 b_k + \tilde{\alpha}\tilde{\beta}\tilde{b}_k\tilde{b}_l$$

$$\rightarrow \quad \alpha\beta(b_k b_l + b_k b_l b_1^2 - a_k b_1 b_l - a_l b_1 b_k) = \tilde{\alpha}\tilde{\beta}\tilde{b}_k\tilde{b}_l$$

$$\rightarrow \quad \alpha\beta(b_k - b_l a_k)(b_l - b\, a_l) = \tilde{\alpha}\tilde{\beta}\tilde{b}_k\tilde{b}_l$$

$$\rightarrow \quad (b_k - b_l a_k)(b_l - b\, a_l) = \frac{\tilde{\alpha}\tilde{\beta}}{\alpha\beta}\tilde{b}_k\tilde{b}_l$$

En lösning till det dyadiska dekompositionsproblemet är

$$\tilde{\alpha} = \alpha + \beta b_1^2$$

$$\tilde{\beta} = \frac{\alpha\beta}{\tilde{\alpha}}$$

$$\gamma = \frac{\beta b_1}{\tilde{\alpha}}$$

$$\tilde{b}_k = b_k - b_1 a_k \qquad k = 2, \ldots, n$$

$$\tilde{a}_k = a_k - \gamma\tilde{b}_k \qquad k = 2, \ldots, n$$

Appendix D. Manual till LabView

Denna manual ska i detalj beskriva användningen av de virtuella instrumenten i Labview.

För att starta Labviewprogrammet, så dubbelklickar man på STR, som ligger under Archimedes-1/User/Tjos/Labview. Det fönster som då kommer upp ser ut enligt figur 1.

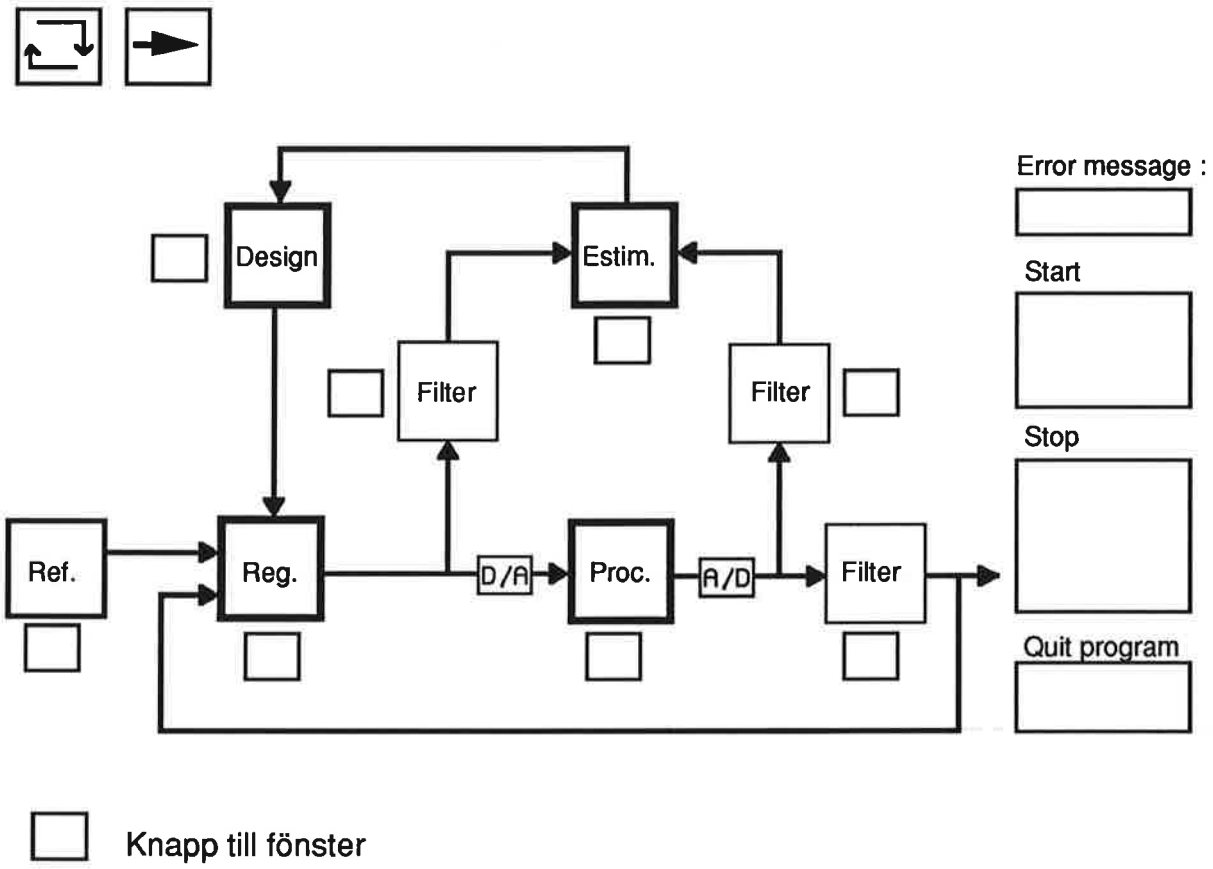

Fig. 1 STR-fönster

Programmet startas genom att trycka på den högra knappen längst upp till vänster (den med en pil i). Då denna knapp ändrar utseende, har DSP-koden laddats ner. Nerladdningen har varit problemfri om "Error message"-rutan visar en nolla. Det är nu möjligt att, med hjälp av knapparna vid sidan om varje fönster, förinställa alla parametrar. När förvalen är klara startas regleringen med "Start"-knappen. När systemet nu är igång kan man gå in i vart och ett av fönstren för att se på variabelförändringar. Önskas ändring av förinställt parametervärde måste systemet stoppas med "Stop"-knappen. Det går nu att ändra alla förval för att sedan starta igen ("Start"-knappen). För att lämna Labviewprogrammet används "Quit program"-knappen.

I fönstret för referensvärden, enligt figur 2, finns det möjlighet att ändra på amplitud, medelvärde och period. Som synes är referens-signalen begränsad till en fyrkantsvåg. Dessa parametrar är de enda som går att ändra på fastän systemet är igång. Det behöver alltså inte stoppas ("Stop"-knappen) för att sedan startas med "start"-knappen.

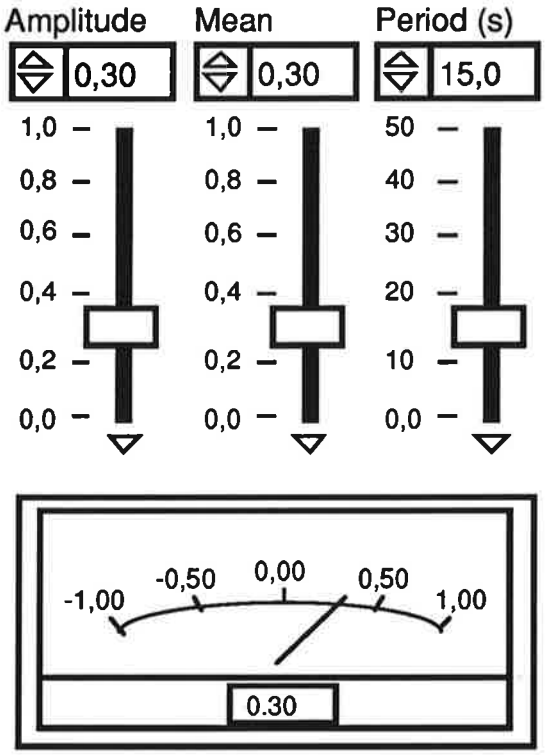

Fig. 2 Ref. - fönster.

Amplituden och medelvärdet är begränsade till +/- 1.0 beroende på att maximal utsignal från utgångskortet är 10 volt , vilket motsvarar 1.0 i programmet. Det går alltså inte att sätta medelvärdet till 0.6 och amplituden till 0.5, ty då ligger referensen mellan 0.1 och 1.1, vilket medför att signalen klipps vid 1.0. Om inte reglagets område räcker till, kan detta enkelt ändras genom att klicka på max- eller minvärdet, och sedan skriva in det önskade värdet . För att återgå från Ref.-fönstret till STR-fönstret enkelklickar man på STR-fönstret.För att stänga ett fönster används den lilla rutan längst upp till vänster i respektive fönster.

Fönstret för regulatorn ser ut enligt figur 3. Här finns möjlighet att förhindra mättning av styrsignalen genom att ha "Antiwindup"-knappen på. Eftersom insvängningsförloppet är extra känsligt för adaptiva regulatorer, har vi infört en PID-regulator om så önskas. Om "Initial PID"-knappen är av, så har resterande inställnings-knappar inte någon betydelse. Om däremot "Initial PID"-knappen är på, så ställs initial-tiden (den tid i början då PID-regulatorn ska verka) in genom att trycka på någon av trianglarna i rektangeln. PID-regulatorn justeras in på vanligt sätt med hjälp av P,I och D parametrarna.

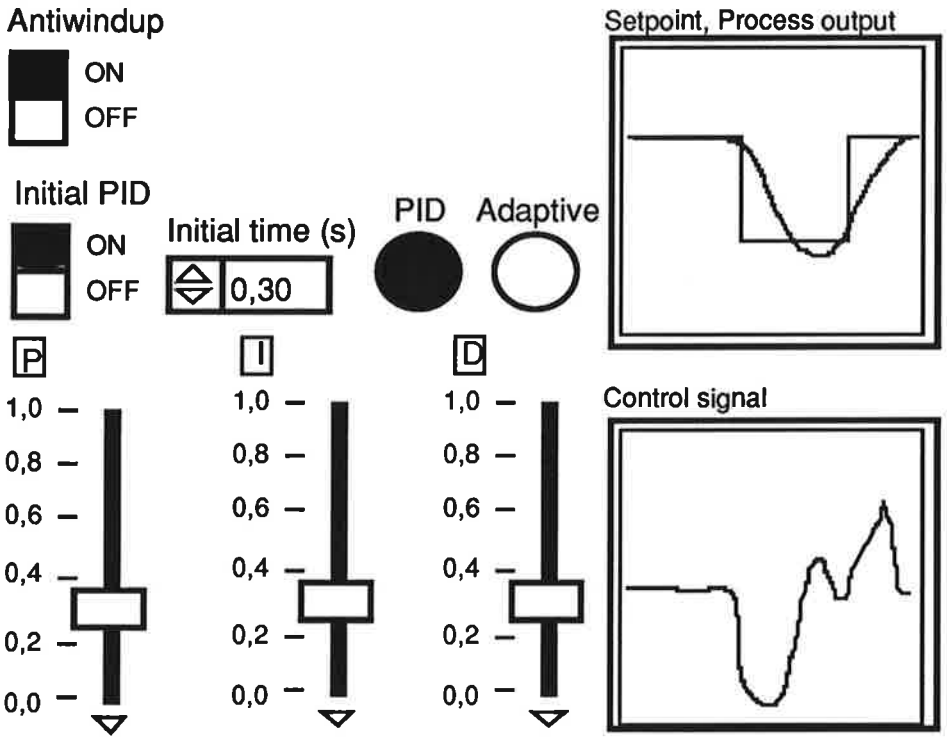

Fig. 3 Reg. - fönster.

Cirklarna med överskriften PID och Adaptive är lampor som visar vilken regulatortyp som är aktiv. Om ändringar av parametrar görs medan systemet är igång slår det inte igenom förrän, som tidigare nämnts, "stop"-knappen och sedan "Start"-knappen används i det virtuella fönstret STR. När systemet är igång visas i övre högra fönstret börvärde och processens utsignal, och i det nedre fönstret visas styrsignalen.

I fönstret till estimatorn förinställs glömskefaktorn lambda, gradtalen A och B för den modell av processen som ska skattas och estimatorfelets lägre gräns (prediktionsfelet). Om estimator-antiwindup önskas, får även den övre gränsen för denna anges. Esimator-antiwindup:en begränsar d-elementen i diagonalmatrisen, som finns beskriven under estimatorn.

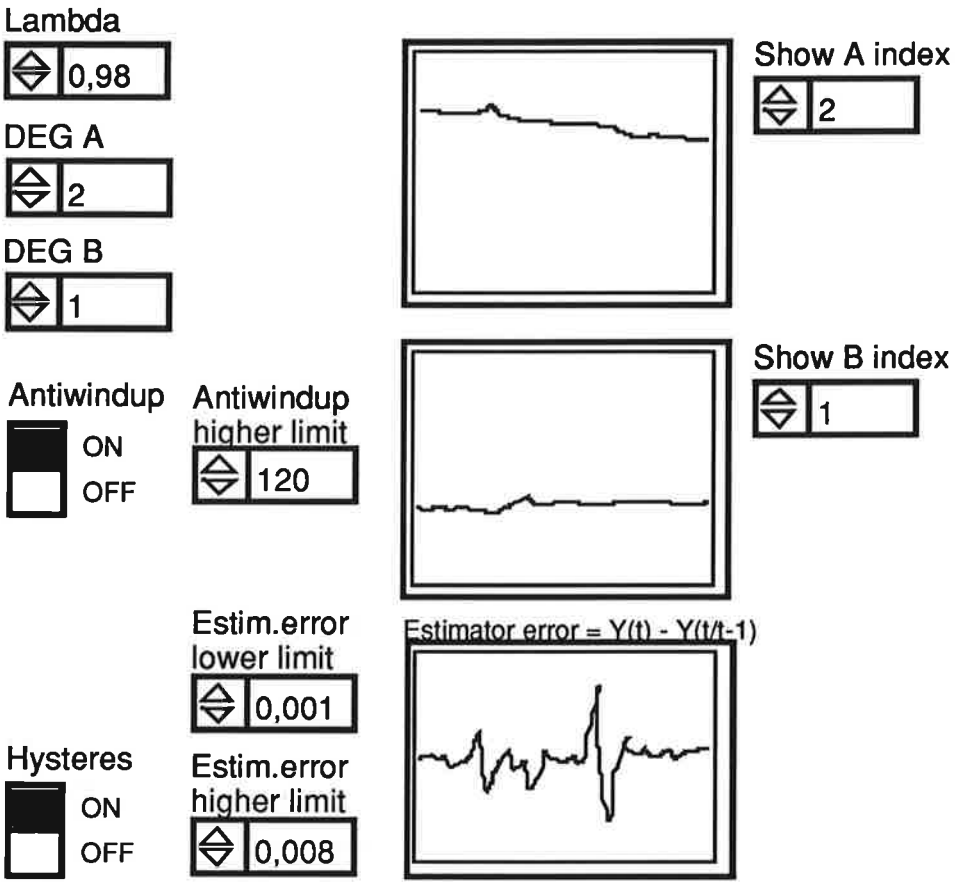

Fig. 4 Estim. - fönster.

Parametern "Estim.error lower limit" finns till för att estimatorn inte ska estimera längre då prediktionsfelet gått ner under denna gräns. Orsaken till detta är att det inte längre är nödvändigt, eller ibland inte går, att minska felet mera. Vid referens-signaler med långa perioder kan viktig information från senaste excitationen gå förlorad, trots att en begränsning är införd. Orsaken till detta är att denna begränsning oftast är mycket låg, vilket medför att efter en lång tid utan excitation börjar prediktionsfelet att flukturera. Detta medför i sin tur att begränsningen överstigs och att estimatorn börjar estimerar och då går viktig data förlorad p.g.a. glömskefaktorn. Detta är också anledningen till estimator -uppvridning, men detta finns det skydd mot.
För att till viss del förhindra detta har vi infört en hysteres om så önskas. Denna är endast inkopplad då "Hysteres"-knappen är till, vilket även innebär att "Estim.error higher limit" måste anges. Denna ska vara större än "Estim.error lower limit". Detta förklaras bättre med hjälp av figur 5.

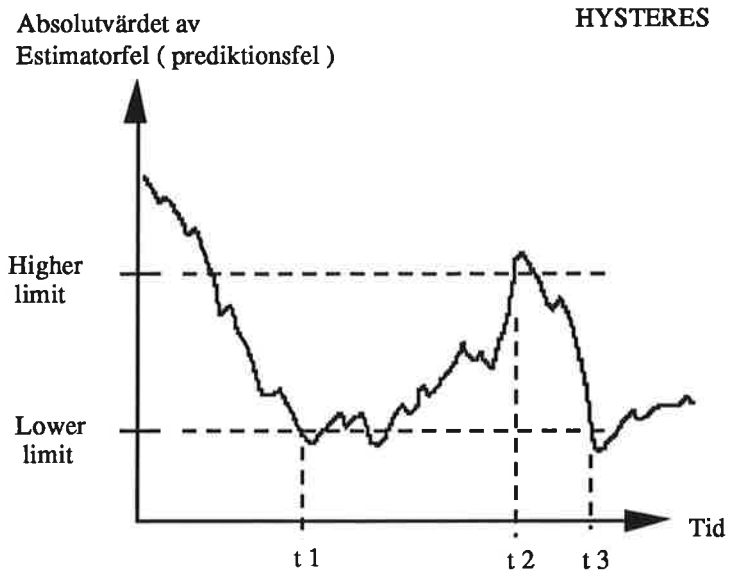

Fig. 5 Hysteres

I insvängningsförloppet är felet (prediktionsfelet) stort, men redan efter någon sekund, beroende på samplingsfrekvens och hur bra excitationen är, så når felet den lägre gränsen. I detta ögonblick, vid tiden t1, slutar estimatorn att estimera. Först vid tiden t2, då den övre gränsen är uppnådd, börjar estimatorn att estimera igen för att sedan, vid tiden t3, sluta igen.

Hur ska då dessa gränser ställas in ?

Om hysteresen inte används, så börja med en mycket låg gräns. Detta medför att estimatorn estimerar hela tiden. Gränsen ska då sättas lite större än vad det genomsnittliga minsta felet visar i det virtuella fönstret för estimatorn. På så sätt estimerar estimatorn med jämna mellanrum.

Om hysteresen används, så gör på samma sätt som utan men med den skillnaden att lägsta gränsen kan sättas ännu lägre. Den övre gränsen sätts lite lägre än det genomsnittliga högsta felet (prediktionsfelet), enligt figur 6.

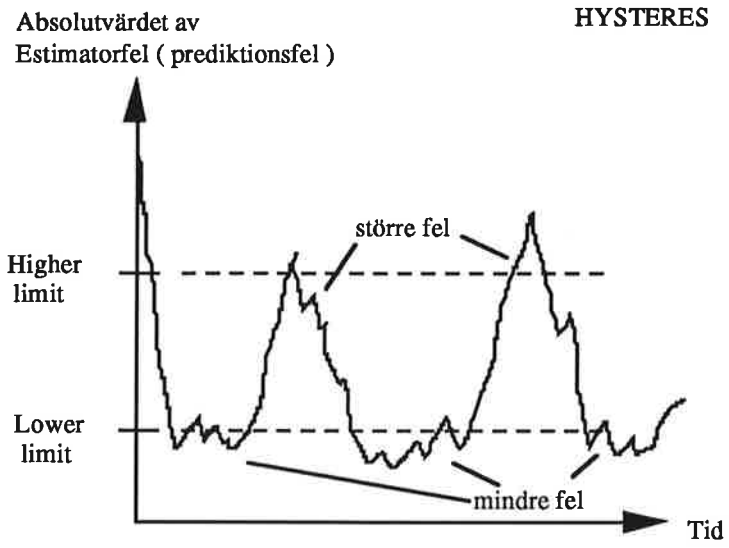

Fig. 6 Exempel

När systemet är igång, kan A och B parametrarna för den skattade processen visas på skärmen. Genom att ändra index fås t.ex. a_1, a_2, ... a_n. Detsamma gäller för B-polynomet.

Det virtuella fönstert för designen visas i figur 7. Här förinställs modellpolynomets (det önskade återkopplade systemets nämnarpolynom) gradtal respektive parametervärden. Detsamma gäller för observerarpolynomet. Då systemet är igång kan R-, S- och T-koefficienterna ses, genom att ändra på indexen för vart och ett av polynomen.

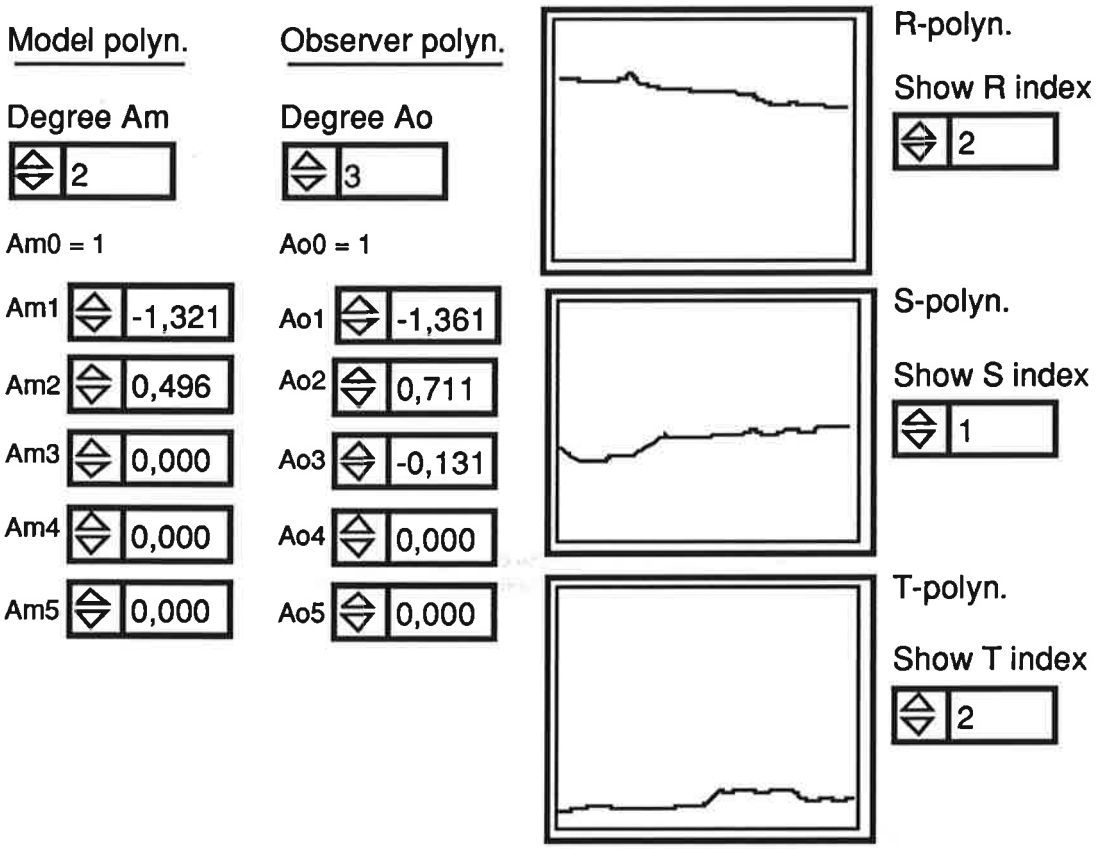

Fig. 7 Design - fönster.

Kontrollera noggrant gradtalet på observerarpolynomet, ty som tidigare nämnts, läggs det till ett nollställe och en pol för att förbättra stabiliteten i vår algoritm.

Fönstret för processen ser ut enligt figur 8. Här förinställs samplingstiden, översamplingen och om så önskas en pålagd sinusstörning på processen. Översamplingen anger hur många mätvärden ett filter får in i förhållande till vad dom tas ut. Inställning av störningen sker på liknande sätt som nämnts i referensdelen.

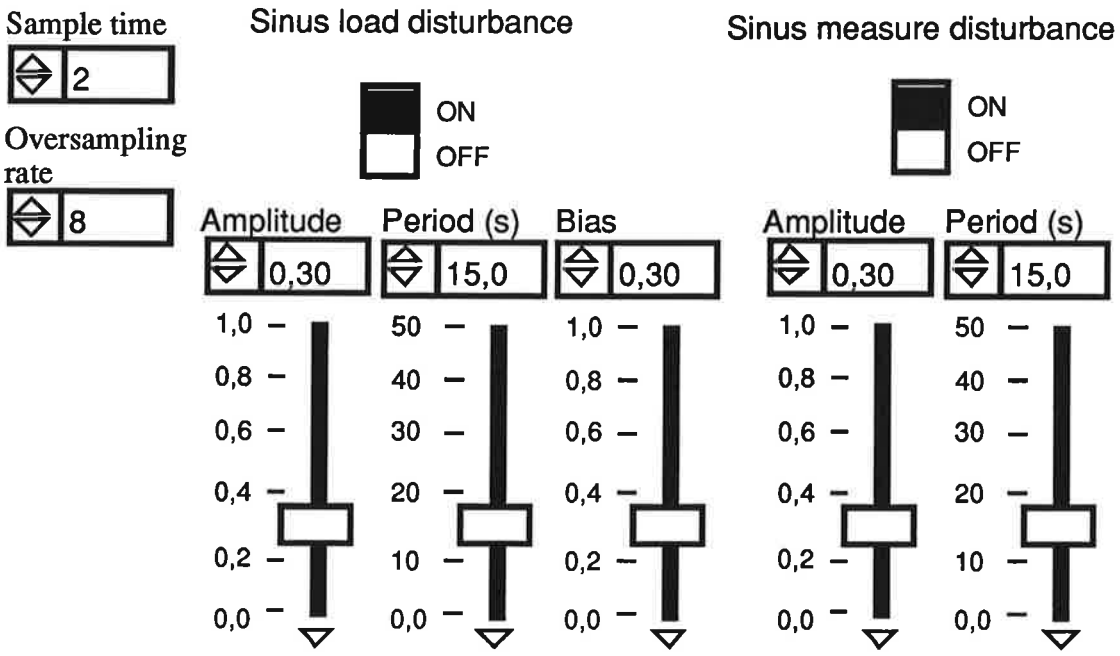

Fig. 8 Proc. - fönster.

Det enda som skiljer dessa åt är att störningen är av sinusform och att referensen är av fyrkantsform.

För inställning av filter finns två virtuella fönster för bandpassfilter, enligt figur 9.

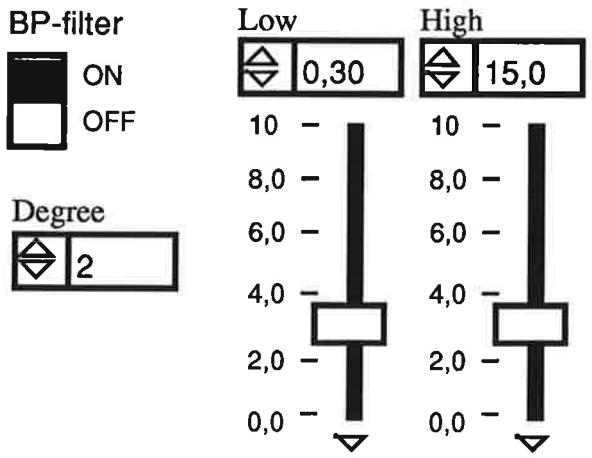

Fig. 9 BP-Filter - fönster

Parametern "degree" anger det gradtal som önskas på filterpolynomet. "Low" och "High" anger brytfrekvenserna för filtret.

Det finns även ett fönster för ett lågpassfilter, som har till uppgift att förhindra vikningsfenomen. Förutom gradtal finns här endast ett "Cut off frequency"-reglage, som anger brytfrekvensen för filtret, enligt figur 10.

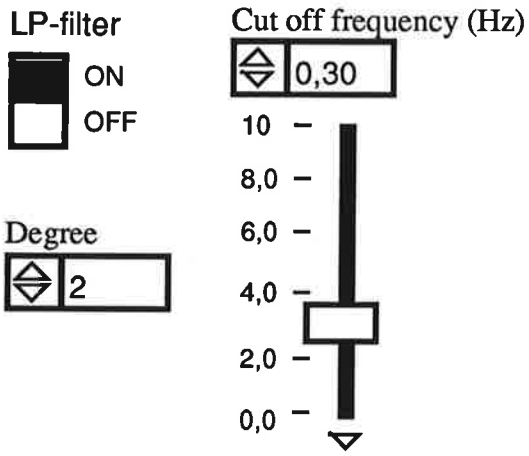

Fig. 10 LP-Filter - fönster.

För att ställa in filterna på bästa möjliga sätt, bör en frekvensanalys av processen göras. Om filtren inte önskas användas, stängs dessa av med "ON/OFF"-knappen i fönstret. Filtrets A- och B-polynom sätts då till ett.

"LabView-program"


```
#define MAX_DEG 10
#include <archimedes-1:User:TJOS:DSP:LabView_Variables.h>
#define  parameter_addr 0x00000100
#define  value_addr     0x00000200
#define DSP_SLOT 3

typedef struct {
    float sample_time;
    long oversampling_rate;
    short load_disturbance;
    float load_amplitude;
    float load_period;
    float load_bias;
    short measure_disturbance;
    float measure_amplitude;
    float measure_period;
} proc_in_type;

typedef struct {
    short   anti_windup;
    short   initial_pid;
    long    initial_pid_time;
    float   p,i,d;
} reg_in_type;

typedef struct {
    float setpoint,process_output,control_signal;
    short pid;
} reg_out_type;

typedef struct {
    short   anti_windup;
    float   estd_max;
    float   est_min;
    long    deg_a,deg_b;
    float   lambda;
    short   hysteresis;
    float   est_max;
} est_in_type;

typedef struct {
    float b_0,b_1,b_2,b_3,b_4,b_5,b_6,b_7,b_8,b_9;
    float a_1,a_2,a_3,a_4,a_5,a_6,a_7,a_8,a_9;
    float error;
} est_out_type;

typedef struct {
    float Am1,Am2,Am3,Am4,Am5;
    float Ao1,Ao2,Ao3,Ao4,Ao5;
    long  Am_deg,Ao_deg;
} des_in_type;

typedef struct {
    float r_1,r_2,r_3,r_4,r_5,r_6,r_7,r_8,r_9;
    float s_0,s_1,s_2,s_3,s_4,s_5,s_6,s_7,s_8,s_9;
    float t_0,t_1,t_2,t_3,t_4,t_5,t_6,t_7,t_8,t_9;
} des_out_type;

typedef struct {
    float ref_value;
} ref_in_type;

typedef struct {
    short lp;
    long deg;
    float cut_off_freq;
} lp_in_type;
```

```
typedef struct {
    short bp;
    long deg;
    float low_cut_off_freq,high_cut_off_freq;
} bp_in_type;

pascal void CINInit(void){}
pascal void CINDispose(void){}
pascal void CINAbort(void){}

pascal void CINRun(proc_in_type *proc_in,reg_in_type *reg_in,reg_out_type *reg_
{

    parameter_type  *p;
    value_type      *v;

    long        *slotBaseAddr;

    /***********************************/
    /*  Put variables in right places  */
    /***********************************/

    slotBaseAddr = (long*) ((long)(DSP_SLOT + 8) << 20);

    p = (parameter_type*)( slotBaseAddr + parameter_addr );
    v = (value_type*)    ( slotBaseAddr + value_addr );

    if (*start)
    {
        /********************/
        /*  Transfer values */
        /********************/

        p -> sample_time =                      proc_in -> sample_time;
        p -> oversampling_rate =                proc_in -> oversampling_rate;
        p -> load_disturbance =         (long)  proc_in -> load_disturbance;
        p -> load_amplitude =                   proc_in -> load_amplitude;
        p -> load_period =                      proc_in -> load_period;
        p -> load_bias =                        proc_in -> load_bias;
        p -> measure_disturbance =      (long)  proc_in -> measure_disturbance;
        p -> measure_amplitude =                proc_in -> measure_amplitude;
        p -> measure_period =                   proc_in -> measure_period;
        p -> reg_anti_windup =          (long)  reg_in -> anti_windup;
        p -> initial_pid =              (long)  reg_in -> initial_pid;
        p -> initial_pid_time =                 reg_in -> initial_pid_time;
        p -> p =                                reg_in -> p;
        p -> i =                                reg_in -> i;
        p -> d =                                reg_in -> d;

        p -> est_anti_windup =          (long)  est_in -> anti_windup;
        p -> estd_max =                         est_in -> estd_max;
        p -> est_min =                          est_in -> est_min;
        p -> deg_a =                            est_in -> deg_a;
        p -> deg_b =                            est_in -> deg_b;
        p -> lambda =                           est_in -> lambda;
        p -> hysteresis =               (long)  est_in -> hysteresis;
        p -> est_max =                          est_in -> est_max;

        p -> Am1 =                              des_in -> Am1;
        p -> Am2 =                              des_in -> Am2;
        p -> Am3 =                              des_in -> Am3;
        p -> Am4 =                              des_in -> Am4;
        p -> Am5 =                              des_in -> Am5;
        p -> Ao1 =                              des_in -> Ao1;
        p -> Ao2 =                              des_in -> Ao2;
        p -> Ao3 =                              des_in -> Ao3;
        p -> Ao4 =                              des_in -> Ao4;
        p -> Ao5 =                              des_in -> Ao5;
```

```
        p -> Ao_deg =                       des_in -> Ao_deg;
        p -> Am_deg =                       des_in -> Am_deg;

        p -> lp =                   (long)  lp_in  -> lp;
        p -> deg_lp =                       lp_in  -> deg;
        p -> low_lp =                       lp_in  -> cut_off_freq;

        p -> bp1 =                  (long)  bp1_in -> bp;
        p -> deg_bp1 =                      bp1_in -> deg;
        p -> low_bp1 =                      bp1_in -> low_cut_off_freq;
        p -> high_bp1 =                     bp1_in -> high_cut_off_freq;

        p -> bp2 =                  (long)  bp2_in -> bp;
        p -> deg_bp2 =                      bp2_in -> deg;
        p -> low_bp2 =                      bp2_in -> low_cut_off_freq;
        p -> high_bp2 =                     bp2_in -> high_cut_off_freq;

        v -> start =                (long)*start;
    }

    v -> ref_value =                        ref_in -> ref_value;

    reg_out -> setpoint =                   v -> ref_value;
    reg_out -> process_output =             v -> process_output;
    reg_out -> control_signal =             v -> control_signal;
    reg_out -> pid =                (short) v -> pid;

    est_out -> a_1 =                        v -> theta[1];
    est_out -> a_2 =                        v -> theta[2];
    est_out -> a_3 =                        v -> theta[3];
    est_out -> a_4 =                        v -> theta[4];
    est_out -> a_5 =                        v -> theta[5];
    est_out -> a_6 =                        v -> theta[6];
    est_out -> a_7 =                        v -> theta[7];
    est_out -> a_8 =                        v -> theta[8];
    est_out -> a_9 =                        v -> theta[9];

    est_out -> b_0 =                        v -> theta[est_in -> deg_a+1];
    est_out -> b_1 =                        v -> theta[est_in -> deg_a+2];
    est_out -> b_2 =                        v -> theta[est_in -> deg_a+3];
    est_out -> b_3 =                        v -> theta[est_in -> deg_a+4];
    est_out -> b_4 =                        v -> theta[est_in -> deg_a+5];
    est_out -> b_5 =                        v -> theta[est_in -> deg_a+6];
    est_out -> b_6 =                        v -> theta[est_in -> deg_a+7];
    est_out -> b_7 =                        v -> theta[est_in -> deg_a+8];
    est_out -> b_8 =                        v -> theta[est_in -> deg_a+9];
    est_out -> b_9 =                        v -> theta[est_in -> deg_a+10];
    est_out -> error =                      v -> error;

    des_out -> r_1 =                        v -> r[1];
    des_out -> r_2 =                        v -> r[2];
    des_out -> r_3 =                        v -> r[3];
    des_out -> r_4 =                        v -> r[4];
    des_out -> r_5 =                        v -> r[5];
    des_out -> r_6 =                        v -> r[6];
    des_out -> r_7 =                        v -> r[7];
    des_out -> r_8 =                        v -> r[8];
    des_out -> r_9 =                        v -> r[9];

    des_out -> s_0 =                        v -> s[0];
    des_out -> s_1 =                        v -> s[1];
    des_out -> s_2 =                        v -> s[2];
    des_out -> s_3 =                        v -> s[3];
    des_out -> s_4 =                        v -> s[4];
    des_out -> s_5 =                        v -> s[5];
    des_out -> s_6 =                        v -> s[6];
    des_out -> s_7 =                        v -> s[7];
    des_out -> s_8 =                        v -> s[8];
```

```
    des_out -> s_9 =                                v -> s[9];

    des_out -> t_0 =                                v -> t[0];
    des_out -> t_1 =                                v -> t[1];
    des_out -> t_2 =                                v -> t[2];
    des_out -> t_3 =                                v -> t[3];
    des_out -> t_4 =                                v -> t[4];
    des_out -> t_5 =                                v -> t[5];
    des_out -> t_6 =                                v -> t[6];
    des_out -> t_7 =                                v -> t[7];
    des_out -> t_8 =                                v -> t[8];
    des_out -> t_9 =                                v -> t[9];

    if (*stop) v -> stop =              (long)  *stop;
    if (*quit) v -> quit =              (long)  *quit;

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

"Huvudprogram"


```
#define   MAX_DEG 10
#include "DSP.h"                        /* int32,int16,MemCodes,Errorcodes and c
#include "Analog.h"                     /* include int_DAQ,In/OutChannel,Gain,An

#include "I_O.h"
#include "Hardware.h"
#include "Estimator.h"
#include "Design.h"
#include "Regulator.h"
#include "Filters.h"
#include "Conversion.h"
#include "butt_filt.h" /* Skall bytas mot fungerande filter */
#include "filtering.h" /* Skall bytas mot fungerande filter */
#include "math.h"

/************************************/
/* Variabler som kommer från LabView */
/************************************/

float    sample_time;
int      oversampling_rate;
int      load_disturbance;
float    load_amplitude;
float    load_period;
float    load_bias;
int      measure_disturbance;
float    measure_amplitude;
float    measure_period;

int   reg_anti_windup;
int   init_pid;
int   pid_time;
float Td,kp,Ti;

float    lambda;
float    est_min;
float    estd_max;
int      est_anti_windup;
int      a_deg,b_deg;
float    est_error;
int      hysteresis;
float    est_max;

int      ao_deg,am_deg;
float    am[MAX_DEG];
float    ao[MAX_DEG];

int      lp;
int      deg_lp;
float    low_lp;

int      bp1;
int      deg_bp1;
float    low_bp1,high_bp1;

int      bp2;
int      deg_bp2;
float    low_bp2,high_bp2;

int      *stop,*quit,*start,*pid;
float    *y_ref_IEEE,*y_IEEE,*u_IEEE,*est_error_IEEE;
float    *r_IEEE,*s_IEEE,*t_IEEE,*theta_IEEE;

#include "LabView_Variables.c"   /* Adresser,typer,variabler från LabView */

/*******************/
/* main- variabler */
/*******************/
```

```
DFilter  outfoh;
int      estimate_ready = 0,i;
float    y_new,y_filt,u_new,y_ref,y_bp,u_bp;
float    theta[MAX_DEG*2];
void     *new_regulator;
struct   regulator_type    regul;
int      time_is_up,stop_pid_time;
float    load_const,measure_const,konst;

float    A_lp[100],B_lp[100],A_bp1[100],B_bp1[100],A_bp2[100],B_bp2[100];
/* Skall bytas mot fungerande filter */


/**********************/
/* Timer-0 interrupt */
/**********************/

void c_int01 ()
{
   start_conversion();
   enable_A_D_interrupt();
   time_is_up++;
}

/******************/
/* A/D interrupt */
/******************/

void c_int02 ()
{
   static int int_count = 0;

   asm(" TSTB  2,IF     ;test INT1 bit");
   asm(" BZ    EXIT_INT1   ;if not set, exit without resetting INT2 level");

   y_new = read_A_D();

   if (measure_disturbance)
      y_new += measure_amplitude * sin (measure_const*int_count);

   filtering(&A_lp[0],&B_lp[0],deg_lp,deg_lp,y_new,&y_filt); /* Skall bytas mot

   if (!((int_count++)%oversampling_rate))
   {
      u_new = compute_u_new(y_ref,y_new); /* Här ska stå (y_ref,y_filt) */

      if (load_disturbance)
         FltDPutSample(outfoh,u_new+load_amplitude*sin(load_const*int_count)+loa
      else
         FltDPutSample(outfoh,u_new);

      Analog_Out(OutChannel_1,FltDGetSample(outfoh));

      estimate(y_new,u_new);  /* Här skall stå (y_bp,u_bp) */
      estimate_ready = 1;
   }
   else
      Analog_Out(OutChannel_1,FltDGetSample(outfoh));

   asm("EXIT_INT1:");
}

/****************/
/* Main program */
/****************/

main()
```

```
{
   /* Initiation that only has to be done once */

   init_interrupt_vectors(c_int02,c_int01);
   init_I_O();
   Analog_Out(OutChannel_1,0.0);
   get_pointers();

   *quit = 0;
   while (!*quit)
   {
      disable_interrupts();
      *pid = 0xffffffff;
      *start = 0;
      while ((! *start ) && (!*quit)){} /* Wait for start or quit */
      if (*start)
      {
         get_parameters();

         /* Create filters */ /* Skall bytas mot fungerande filter */

         konst = (sample_time/oversampling_rate)*0.001;

         if(lp)
            create_butter(deg_lp,low_lp*konst,low_lp*konst,A_lp,B_lp,LP);
         else
         {
            deg_lp = 0;
            A_lp[0] = B_lp[0] = 1.0;
         }

         if(bp1)
            create_butter(deg_bp1,low_bp1*konst,high_bp1*konst,A_bp1,B_bp1,BP);
         else
         {
            deg_bp1 = 0;
            A_bp1[0] = B_bp1[0] = 1.0;
         }

         if(bp2)
            create_butter(deg_bp2,low_bp2*konst,high_bp2*konst,A_bp2,B_bp2,BP);
         else
         {
            deg_bp2 = 0;
            A_bp2[0] = B_bp2[0] = 1.0;
         }

         outfoh = FltDCreatePFoH(oversampling_rate,1,0.8);

         /* Init modules */
         init_design(am, ao, am_deg, ao_deg, a_deg, b_deg);
         init_estimator(a_deg,b_deg,lambda,theta,est_min,est_max,estd_max,est_ar
         init_regulator(pid,kp,Ti,Td,sample_time,reg_anti_windup);
         init_timer(sample_time,oversampling_rate);

         /* Compute constants */
         measure_const = 2.0*0.00314159265*sample_time/(measure_period*oversamp
         load_const = 2.0*0.00314159265*sample_time/(load_period*oversampling_r
         stop_pid_time = (init_pid) ?
         (pid_time*1000*oversampling_rate)/sample_time:0;

         time_is_up = 0;
         reset_RTSI_EN_bit();
         enable_timer_interrupt();

         *stop = 0;
         while ((! *stop) && (!*quit))
         {
```

```
            /* Transfer values */

            y_ref = to_float(*y_ref_IEEE);

            *u_IEEE = to_IEEE(u_new);
            *y_IEEE = to_IEEE(y_new);
            *est_error_IEEE = to_IEEE(est_error);

            for (i=0 ; i<=MAX_DEG-1 ;i++){
               r_IEEE[i] = to_IEEE(regul.r[i]);
               s_IEEE[i] = to_IEEE(regul.s[i]);
               t_IEEE[i] = to_IEEE(regul.t[i]);}

            for(i=0 ; i<= a_deg+b_deg+1 ;i++)
               theta_IEEE[i] = to_IEEE(theta[i]);

            /*******************/

            if ((estimate_ready) && (time_is_up>stop_pid_time))
            {
               *pid = 0;                    /* Change indicator lamp */
               estimate_ready = 0;          /* Reset flag */
               new_design(theta,&regul);
               change_regulator(&regul);
            }
         }
         FltDFree(outfoh);
      }
      disable_interrupts();
      Analog_Out(OutChannel_1,0.0); /* Reset ouytput */
   }
   asm(" BR 1000F3h");              /* Terminate */
}
```

```
#ifndef _ANALOG_H_
#define _ANALOG_H_

#include "int_DAQ.h"     /*  STAT,Command1,Command2,StartDAQ,ADClear,DAC0,DAC1*/
                         /*  Mux_Count,Mux_Gain,AD_FIFO,AMC,AMD,MIO_R_SH,MIO_R_S
                         /*  intCal,tmrIntClr,digOut*/


enum InChannel { InChannel_0 = 0x00000, InChannel_1 = 0x10000,
                 InChannel_2 = 0x20000, InChannel_3 = 0x30000 };
enum OutChannel { OutChannel_0 = DAC0, OutChannel_1 = DAC1 };
enum Gain { gain_1   = 0x000000, gain_10  = 0x400000,
            gain_100 = 0x800000, gain_500 = 0xc00000 };

void Analog_In(enum InChannel channel, enum Gain gain, float* value);
void Analog_Out(enum OutChannel channel,float j);

#endif
```

```
#include "DSP.h"
#include "int_DAQ.h"
#include "Analog.h"

int32 slot_base = 0xfd000000;

float_and_scale(int32 *i,float *j)
{
    *j=*i;
    *j=*j/(2048*0x10000);

}

Read_One_Sample(board_firstadd,sample)
int32 board_firstadd,*sample;
{
register int32 status, StartConvert;
int32 temp;
StartConvert = board_firstadd + 0x10;

Send_NuBus(0, StartConvert);
while (((status = Get_NuBus(board_firstadd)) & 0x20000000) == 0);
if ((status & 0x03000000)!= 0) {
    temp = ((status & 0x01000000)!= 0) ? overRunErr : overFlowErr;
    Send_NuBus(0x0,board_firstadd + ADClear);
    return(temp);
    }
*sample = Get_NuBus(board_firstadd + AD_FIFO);
return(0);
}

void Analog_In(enum InChannel channel, enum Gain gain, float* value)
{
  int32 intvalue;
  Send_NuBus ((gain | channel), slot_base + Mux_Gain);
  Read_One_Sample(slot_base, &intvalue);
  float_and_scale(&intvalue, value);    /* convert to floating point */
}

void Analog_Out(enum OutChannel channel,float j)
{
  int32 i,Dac;
  Dac = slot_base + channel;
  if (j >= 1.0) {
    i = (4095*0x10000);
  } else if (j <= -1.0) {
    i = (0*0x10000);
  } else {
    j=j*(2048*0x010000);    /* scale it back */
    i=j;                    /* convert back to integer */
    i += (2048*0x10000);    /* add the offset to suit DAC */
  };
  Send_NuBus(i, Dac);
}
```

```
extern int32 slot_base;

void init_I_O(void);

void start_conversion(void);

float read_A_D(void);
```

```
#include "int_DAQ.h"
#include "DSP.h"
#include "Analog.h"
#include "I_O.h"

int32 cmd2Reg, cmd1Reg;

Init_MIO16(int32 board_firstadd)

{

int32   AMCommand,AMData,NBadd,MIOstatusReg;
int32   Status,ADFifo,i;
int32   MIOstatus;

asm("       AND     0FFFDh,IE   ");     /* disable interrupt  */
AMCommand = board_firstadd + AMC;
AMData = board_firstadd + AMD;
ADFifo = board_firstadd + AD_FIFO;
MIOstatusReg = board_firstadd + STAT;

cmd1Reg = 0x0;
Send_NuBus(cmd1Reg,board_firstadd + Command1);
cmd2Reg = 0x0;
Send_NuBus(cmd2Reg,board_firstadd + Command2);
Send_NuBus(0x0,board_firstadd + Mux_Gain);

/* 4- initialize RTSI bus switch */
for(i=0;i<56;i++) Send_NuBus(0x0,board_firstadd + MIO_R_SH);
Send_NuBus(0x0,board_firstadd + MIO_R_ST);

/* 5- initialize Am9513A */
Send_NuBus(0xffff0000,AMCommand);
Send_NuBus(0xffef0000,AMCommand);
Send_NuBus(0xff170000,AMCommand);
Send_NuBus(0xf0000000,AMData);

for(i=1;i<=5;i++){
    Send_NuBus(0xff000000+i<<16,AMCommand);
    Send_NuBus(0x00040000,AMData);
    Send_NuBus(0xff080000+i<<16,AMCommand);
    Send_NuBus(0x00030000,AMData);
    }
Send_NuBus(0xff5f0000,AMCommand);
for(i=0;i<1000;i++);
Send_NuBus(0x0,board_firstadd + ADClear);

/* 6- initialize analog output circuitry */
Send_NuBus(0x0, board_firstadd + DAC0);
Send_NuBus(0x0, board_firstadd + DAC1);

/* 7- initialize digital output register */
Send_NuBus(0x0, board_firstadd + digOut);
}


Reset_MIO16(int32 board_firstadd)

{
int32   init,AMCommand,AMData,NBadd;
int32   Status,ADFifo,i;
asm("       AND     0FFFDh,IE   ");     /* disable interrupt  */
AMCommand = board_firstadd + AMC;
AMData = board_firstadd + AMD;
ADFifo = board_firstadd + AD_FIFO;

/* Remove source of interrupt */
Send_NuBus(0x0,board_firstadd + ADClear);
```

```
/* Resetting counter 2 */
Send_NuBus (0xffc20000,AMCommand);          /* disarm counter 2 */
Send_NuBus (0xff020000,AMCommand);          /* select counter 2 mode register *,
Send_NuBus (0x00040000,AMData);             /* high impedance */
Send_NuBus (0xff0a0000,AMCommand);      /* select counter 2 load register */
Send_NuBus (0x00030000,AMData);         /* high impedance */
Send_NuBus (0xff420000,AMCommand);      /* load counter 2 */
Send_NuBus (0xff420000,AMCommand);      /* twice */

/* Resetting counter 3 */
Send_NuBus (0xffc40000,AMCommand);          /* disarm counter 3 */
Send_NuBus (0xff030000,AMCommand);          /* select counter 3 mode register *,
Send_NuBus (0x00040000,AMData);             /* high impedance */
Send_NuBus (0xff0b0000,AMCommand);      /* select counter 3 load register */
Send_NuBus (0x00030000,AMData);         /* high impedance */
Send_NuBus (0xff440000,AMCommand);      /* load counter 3 */
Send_NuBus (0xff440000,AMCommand);      /* twice */


/* Resetting counter 4 */
Send_NuBus (0xffc80000,AMCommand);          /* disarm counter 4 */
Send_NuBus (0xff040000,AMCommand);          /* select counter 4 mode register *,
Send_NuBus (0x00040000,AMData);             /* high impedance */
Send_NuBus (0xff0a0000,AMCommand);      /* select counter 4 load register */
Send_NuBus (0x00030000,AMData);         /* high impedance */
Send_NuBus (0xff480000,AMCommand);      /* load counter 4 */
Send_NuBus (0xff480000,AMCommand);      /* twice */

/* Resetting counter 5 */
Send_NuBus (0xffd00000,AMCommand);          /* disarm counter 5 */
Send_NuBus (0xff050000,AMCommand);          /* select counter 5 mode register *,
Send_NuBus (0x00040000,AMData);             /* high impedance */
Send_NuBus (0xff0d0000,AMCommand);      /* select counter 5 load register */
Send_NuBus (0x00030000,AMData);         /* high impedance */
Send_NuBus (0xff500000,AMCommand);      /* load counter 5 */
Send_NuBus (0xff500000,AMCommand);      /* twice */

/* Clear all error conditions and empty A/D FIFO */
Send_NuBus (0x0,board_firstadd + ADClear);
return(0);
}

/*
 * Setup_MIO16(board_slot,gain,timebase,interval,channel,samples,RTSInt)
 *
 * Set up the MIO for data acquisition. Init_MIO16 and Reset_MIO16 must
 * be run previously (Init defines globals)
 * Continuous sampling: the MIO 16 is set up for continuous
 * sampling if samples <=0 or samples >65535
 * (see NB-MIO-16 User Manual p 3-40):
 * Gain: 0 to 3 in the bit 6 and 7 of Gain_Mux
 * Timebase: 5 (100Hz) to 1 (1MHz) in Counter 3
 *
 * Interrupt service (NB-MIO_16X p 3-79):
 * set CONVINTEN in Command1 if RTSInt != 0
 * set NBINTDIS  in Command2 : MIO16 doesn't assert interrupt on NuBus
 *
 */

extern int32 cmd1Reg,cmd2Reg;
int32 Alloc_Mem();

Setup_MIO16(board_slot,gain,timebase,interval,channel,samples,RTSInt)
 int32  board_slot,gain,timebase,interval,channel,samples,RTSInt;
 {
 int32  err,n,m,AMCommand,AMData,NBadd,timeout;
 int32  i,delay,board_firstadd,temp;
```

```
board_firstadd = (board_slot * 0x01100000) + 0xF8800000;
if(gain >3 || gain < 0) return(outOfRangeErr);
if(channel>15 || channel<0) return(outOfRangeErr);
if (interval<2 || interval>65535) return(outOfRangeErr);
if (samples == 1) return(outOfRangeErr);

/**** 1- Select channel and gain ****/
Send_NuBus (0x0,board_firstadd + Mux_Count);
Send_NuBus ((gain<<22) | (channel<<16),board_firstadd + Mux_Gain);

/**** 2- Program Sample-Interval Counter ****/
AMCommand = board_firstadd + AMC;
AMData = board_firstadd + AMD;

Send_NuBus (0xFF030000,AMCommand);      /* select Counter 3 Mode Register */

switch(timebase){
    case 1 : temp = 0x8b250000;break;       /*   1 MHz */
    case 2 : temp = 0x8c250000;break;       /* 100 kHz */
    case 3 : temp = 0x8d250000;break;       /*  10 kHz */
    case 4 : temp = 0x8e250000;break;       /*   1 kHz */
    case 5 : temp = 0x8f250000;break;       /* 100 Hz  */
    default: return(outOfRangeErr);break;
    }
Send_NuBus (temp,AMData);               /* select timebase */

Send_NuBus (0xFF0B0000,AMCommand);      /* select Counter 3 Load Register */
Send_NuBus (0x00020000,AMData);         /* Counter 3 Load value */
Send_NuBus (0xFF440000,AMCommand);      /* load Counter 3 */
Send_NuBus (0xFFF30000,AMCommand);      /* step Counter 3 to 1 */
Send_NuBus (interval<<16,AMData);       /* select sample interval */
Send_NuBus (0xFF240000,AMCommand);      /* arm Counter 3 */

/**** 3-  Program the Sample Counter if non continuous ****/
if (samples >0 && samples < 65536){
    Send_NuBus (0xFF040000,AMCommand);      /* select Counter 4 Mode Register *,
    Send_NuBus (0x10010000,AMData);         /* store Counter 4 Mode value */
    Send_NuBus (0xFF0C0000,AMCommand);      /* select Counter 4 Load Register *,
    Send_NuBus (samples<<16,AMData);
    Send_NuBus (0xFF680000,AMCommand);      /* arm Counter 4 */
    }
else{   /* set output CTR4 to low */
    Send_NuBus(0xff040000,AMCommand);
    Send_NuBus(0x0,AMData);
    Send_NuBus(0xff0c0000,AMCommand);
    Send_NuBus(0x00030000,AMData);
    }

/*
 * 4- Clear the A/D circuitry and enable continuous data acquisition.
 *    MIO-16 User Manual p3-43.
 */
Send_NuBus (0x0,board_firstadd + ADClear);                  /* clear ADFIFO

cmd1Reg = (RTSInt) ? 0x00500000 : 0x00100000 ;
Send_NuBus (cmd1Reg,board_firstadd + Command1);

/* Command1:
     x x x x x x x DAQ INT  CONV INT  DMA  DAQ  SCAN SCAN 16*-32  2SC*
                     EN       EN     EN   EN    EN  DIV    CNT   ADC*
     0 0 0 0 0 0 0   0        1      0    1     0   0      0     0      */

cmd2Reg =   0x00800000;
Send_NuBus (cmd2Reg,board_firstadd + Command2);
/* Command2:
     x x x x x DOUTB DOUTA  NBINT   DMA  DMA  DMA   A4   A4     A2    A2
               EN   EN      DIS     A2   A1   A0    RCV  DRV    RCV   DRV
```

```
     0 0 0 0 0   0   0      1       0     0    0     0    0      0     0    */

return(0);
}

void start_conversion()
{
    Send_NuBus((gain_1|InChannel_1),slot_base+0x0c);        /* Choose channel a
    Send_NuBus(0,slot_base + 0x18);                         /* Clear A/D */
    Send_NuBus(0,slot_base + 0x10);                         /* Start A/D conver
}

float read_A_D()
{
    int intvalue;
    float float_value;

    intvalue = Get_NuBus(slot_base + AD_FIFO);              /* Get measured valu
    Send_NuBus (0,slot_base + 0x18);                        /* Clear A/D */
    float_and_scale( &intvalue , &float_value );            /* Convert from int
    return(float_value);
}

/************************************************/
/*              Init I/O                        */
/************************************************/

void init_I_O()
{
    Init_MIO16(slot_base);
    Reset_MIO16(slot_base);
    Setup_MIO16(5,0,5,2,1,0,1);
}
```

```
void init_interrupt_vectors(void (*f)(),void (*g)());

void init_timer(float sample_time,int oversampling);

void reset_RTSI_EN_bit(void);

void enable_A_D_interrupt(void);

void enable_timer_interrupt(void);

void disable_interrupts(void);
```

```
#include "Hardware.h"

typedef void (*interrupt_handler)();   /* interrupthandler is a pointer to a fu

void init_interrupt_vectors(void (*f)(),void (*g)())


{
    interrupt_handler *Extint1vector = (interrupt_handler*) 0x001000f1;
    interrupt_handler *Timeint0vector = (interrupt_handler*) 0x001000f8;

    *Extint1vector = f;                        /* Interrupt adress for c_int02 in
    *(int*)Extint1vector |= 0x60000000;            /* Must be jump instruction */

    *Timeint0vector = g;                           /* Interrupt adress for c_int01
    *(int*)Timeint0vector |= 0x60000000;               /* Must be jump instruction
}


void init_timer(float sample_time,int oversampling)
    {
    Setup_Timer(0,0x000003c3,(long)(sample_time*(4167/oversampling)),0);
    }

void reset_RTSI_EN_bit()
    {

    /*
    * The boot-prom sets the RTSI_EN bit in the DSP2300 control register.
    * Since we are going to use this interrupt for Data aquistion timing, we
    * must shut it off first.
    */
    asm("   PUSH    AR4");
    asm(" LDI       @MaskAddr,AR4");
    asm("   PUSH    DP");
    asm(" PUSH  R0");
    asm(" LDI       080h,DP"); /* Point to I/O Section */
    asm(" LDI       @5004h,R0");
    asm(" AND       *AR4++,R0");
    asm(" OR        *AR4,R0");
    asm(" STI       R0,@5004h");
    asm("   BUD     Resume"); /* Delayed jump over the data below */
    asm(" POP       R0");
    asm("   POP     DP");
    asm(" POP       AR4");

    asm("MaskAddr: .word    Mask");
    asm("Mask: .word    0FFFE3FFFh");
    asm("      .word       024000h");

    asm("Resume:");
    }

void enable_A_D_interrupt()
    {
    asm(" OR 0002h,IE");
    }

void enable_timer_interrupt()
    {
    asm(" OR 0100h,IE");
    }


void disable_interrupts()
    {
    asm(" AND 0fefdh,IE");
    }
```

```
#define mzero 1.0e-10
#define initial_covar 2000.0
#define MAX_DEG 10
void estimate(float y,float u);
void init_estimator(int a_deg,int b_deg,float lambda,float in_theta[],float est_
```

```
#include "Estimator.h"
#include "Analog.h"

int estim_a_deg,estim_b_deg,nr_of_parameters ;
float estim_lambda ;
float *estim_theta;
float estim_estd_max;
float estim_est_min;
float estim_est_max;
float *estim_error;
int estim_anti_windup,estim_start;
float estimphi[MAX_DEG*2+1] ;
float estimd[MAX_DEG*2+1] ;
float estiml[MAX_DEG*2+1][MAX_DEG*2+1];
float estimu[MAX_DEG];
int k;

/* Initialize phi- and theta vectors */

init_phi_theta()
{
    int i ;

    for (i=0 ; i<=nr_of_parameters ; i++)
    {
        estimphi[i] = 0.0 ;
        estim_theta[i] = 0.001 ;
    }
}

/* Initialize l- and d vectors */

init_l_d()
{
    int i,j ;

    for (i=0 ; i<=nr_of_parameters ; i++) {
        estimd[i]=initial_covar ;
        for (j=0 ; j<=nr_of_parameters ; j++ ) {
            if (i==j)
                estiml[i][j]=1.0 ;
            else
                estiml[i][j]=0.0 ;
        }
    }
}


/* Update the regressorvector phi with y_vector and u_vector */

/* phi = [ y(t) -y(t-1)  -y(t-2) .... -y(t-(na-1))  u(t)  u(t-1)  u(t-2)  ....

update_phi(float y,float u)
{
    int i,d;


    d = estim_a_deg - estim_b_deg;
    for (i = nr_of_parameters ; i>=1 ; i--)
        estimphi[i] = estimphi[i-1];
    estimphi[1] = -estimphi[0];

    estimphi[0] = y;

    for (i = d; i>=1 ;i--)
        estimu[i]=estimu[i-1];
    estimu[0] = u;
    estimphi[estim_a_deg+1] = estimu[d];
```

```
}


dyadic_reduction()
{
    int     j ;
    float   w1,w2,b1,gamma ;

    if (estimd[k] < mzero)
        estimd[k] = 0.0 ;
    b1 = estiml[k][0] ;
    w1 = estimd[0] ;
    w2 = estimd[k] * b1 ;
    estimd[0] += w2 * b1 ;
    if (estimd[0] > 1.0e-10) {
        estimd[k] *= w1 / estimd[0] ;
        gamma = w2 / estimd[0] ;
        for (j=k ; j<=nr_of_parameters ; j++) {
            estiml[k][j] -= b1 * estiml[0][j] ;
            estiml[0][j] += gamma * estiml[k][j] ;
        }
    }
}

LD_filter()
{
    static int time,estimation;
    int     i,j;
    float   e,w ;

    if (estim_start)
    {
        estimation = 1;
        estim_start = 0;
        time = 0;
    }
    time++;
    e = estimphi[0] ;
    estimd[0]= estim_lambda ;
    for (i=1 ; i<=nr_of_parameters ; i++)
        e -= estim_theta[i] * estimphi[i] ;

    *estim_error = e;

    if ((((e>0.0)?e:-e)<estim_est_max) && (!estimation))
        return;

    if (((e>0.0)?e:-e)>estim_est_max)
        estimation = 1;

    if (((e>0.0)?e:-e)<estim_est_min)
    {
        estimation = 0;
        return;
    }

    for (i=1 ; i<=nr_of_parameters ; i++) {
        w = estimphi[i] ;
        for (j=i+1 ; j<=nr_of_parameters ; j++)
            w += estimphi[j]*estiml[i][j] ;
        estiml[0][i] = 0.0 ;
        estiml[i][0] = w ;
    }

    for (k=nr_of_parameters ; k>=1 ; k--)
          dyadic_reduction() ;
```

```
    for (i=1 ; i<=nr_of_parameters ; i++) {
        estim_theta[i] += estiml[0][i] * e;
        estimd[i] /= estim_lambda;
        if ((estimd[i]>estim_estd_max) && (time>100) && (estim_anti_windup))
            estimd[i] = estim_estd_max;
    }
}

void estimate(float y,float u)
{
    update_phi(y,u);
    LD_filter();
}

void init_estimator(int a_deg,int b_deg,float lambda,float in_theta[],float est_
{
    estim_a_deg = a_deg ;
    estim_b_deg = b_deg ;
    estim_lambda = lambda ;
    estim_theta = &in_theta[0];
    estim_estd_max = estd_max;
    estim_est_min = est_min;
    estim_est_max = est_max;
    estim_anti_windup = anti_windup;
    estim_start = 1;
    estim_error=error;

    nr_of_parameters = a_deg + (b_deg + 1) ;
    init_phi_theta();
    init_l_d();

    if (!hysteresis)
        estim_est_max = estim_est_min;
}
```

```
#define MAX_DEG 10
void new_design(float *theta_from_estimator,void* regulator);
void init_design(float amx[], float aox[], float amx_deg, float aox_deg, float a
```

```
#include <stdlib.h>
#include <math.h>
#include "Design.h"

#define MAX(A,B) ((A)>(B) ? (A) : (B))
int des_ao_deg,des_am_deg,des_c_deg,des_a_deg,des_b_deg;
float *des_am,*des_ao,des_c[MAX_DEG*2];

/*****************************************************/
/*  NEW_DEG                                          */
/*  new_deg returnerar gradtalet ( position for      */
/*  högsta koeff != 0 ) för polynomet poly           */
/*  av maximalt gradtal old_deg, returnerar -1       */
/*  om poly = 0.                                     */
/*                                                   */
/*  poly[0] <--> konstantterm  o s v                 */
/*****************************************************/

int new_deg(poly,old_deg)
    float *poly;
    int old_deg;

{
    int i;
    float limit=1e-10;
    i=old_deg;

    while(fabs(poly[i]) < limit && i>=0 )
        i--;
    if ( i==0 && (fabs(poly[i]) < limit))
        return -1;
    else
        return i;
}


/*****************************************************/
/*  POLY_RED                                         */
/*                                                   */
/*  Löser den 'Diofantiska ekvationen'               */
/*              AX + BY = C                          */
/*  med villkoret deg Y < deg A.                     */
/*                                                   */
/*  Returnerar X,Y,X_deg,Y_deg samt OK=1             */
/*  om lösning funnen.                               */
/*                                                   */
/*  Klarar av om HL och VL har gemensamma faktorer   */
/*                                                   */
/*  OBS! Ordning och index på koefficienter !        */
/*      A[] = { a_0 a_1 ... a_p }                    */
/*       betecknar                                   */
/*           A= a_0 + a_1 z + ... + a_p (z**p)       */
/*           B= b_0 + b_1 z + ... + b_p (z**q)       */
/*           C= c_0 + c_1 z + ... + c_p (z**r)       */
/*****************************************************/

void polyred(A,B,C,p,q,r,X,Y,X_deg,Y_deg,OK)

float *A,*B,*C,*X,*Y;
int p,q,r ;
int *X_deg,*Y_deg,*OK;
{

/* Borde samlas under static (?) struct */

    int choise[30]  ;   /* typ av reduktion [1..4] */
    int exp[30] ;       /* exponent i reduktionsterm */
    float koeff[30];    /* koeff.      - " -         */
```

```
    int i,j;
    int step = 0;
    int Forward_ready = 0;
    int maxXY=15;
    int degX,degY;

    i=MAX(p,q);          /* kolla detta villkor på X o Y*/
    j=MAX(i,r);
    for(i=0;i<=j;i++){
       X[i]=0.0;
       Y[i]=0.0;
    }

    *OK=1;

    while (Forward_ready == 0) {

    if (r >= p && r>-1 && p>-1){
        choise[step]=1;
        koeff[step]=C[r]/A[p];
        exp[step] = r-p ;
        for(i=exp[step];i<=r;i++)
            C[i] -= koeff[step]*A[i-exp[step]];      /* C' = C - k*A*(z**exp)
        r=new_deg(C,r);
    }

    else if (r >= q && r>-1 && q>-1){
        choise[step]=2;
        koeff[step]=C[r]/B[q];
        exp[step] = r-q ;
        for(i=exp[step];i<=r;i++)
            C[i] -= koeff[step]*B[i-exp[step]];      /* C' = C - k*B(z**exp) */
        r=new_deg(C,r);
    }

    else if (q >= p && p>-1 && q>-1){
        choise[step]=3;
        koeff[step]=B[q]/A[p];
        exp[step] = q-p ;
        for(i=exp[step];i<=q;i++)
            B[i] -= koeff[step]*A[i-exp[step]];      /*B' = B - k*A(z**exp)  */
        q=new_deg(B,q);
    }

    else if (p >= q && p>-1 && q>-1){
        choise[step]=4;
        koeff[step]=A[p]/B[q];
        exp[step] = p-q ;
        for(i=exp[step];i<=p;i++)
            A[i] -= koeff[step]*B[i-exp[step]];      /*A' = A - k*B(z**dd)   */
        p=new_deg(A,p);

    }
    else {
           *OK=0;
    /* ej lösbart eller fel uppkommit */
    }

    if ( (p+r)== -2 || (q+r)== -2 || (p+q)== -2 )
        Forward_ready =1;
    else {
        /* printf(" k=%f  diff=%i /n",koeff[step],exp[step]); */
        /* printf(" choise[%i]=%i /n",step,choise[step]);     */
        step +=1;
    }

    } /*while*/
```

```
/* ------------------------ bakåtsubstitution ----------------------*/

    degX=-1;
    degY=-1;
    for(i=step;i>=0;i--) {

        switch ( choise[i] )  {

            case 1 :
                      X[exp[i]] += koeff[i] ;
                      if ( degX < exp[i] )
                         degX = exp[i];
                      break;
            case 2 :  Y[exp[i]] += koeff[i] ;
                      if ( degY < exp[i] )
                         degY = exp[i];
                      break;
            case 3 :
                      for(j=0;j<=degY;j++)
                        X[j+exp[i]] -= koeff[i]*Y[j];
                      if( degY != -1 )
                        if( degX < degY+exp[i] )
                            degX = degY+exp[i];
                      break;
            case 4 :
                      for(j=0;j<=degX;j++)
                        Y[j+exp[i]] -= koeff[i]*X[j];
                      if( degX != -1 )
                          if( degY < degX+exp[i] )
                            degY = degX+exp[i];
                      break;
            default :
                       *OK = 0;
                    /*   printf("DEF! choise[%i]=%i",i,choise[i]);  */
                       break;
        }

    }
    *X_deg=degX;
    *Y_deg=degY;

}


/*****************************************/
/*  PRE_POLY_DAB                         */
/*                                       */
/*  Anropar polyred ( lsg av Diof.ekv)   */
/*  efter att lagt till                  */
/*  pol i 1  (integrator)                */
/*  samt ett nollställe i ( -1 )         */
/*                                       */
/*  justerar också a och c polynomen     */
/*  na = #elt i a-vektor inkl a0  osv.   */
/*                                       */
/*  normerar r,s och t så r0 =1          */
/*****************************************/

void prepolydab(float* th,int a_deg,int b_deg,float* am,float* ao,int am_deg,int


/* theta = [ ?? +a(1) .. + a(a_deg) + b(0) .. + b(b_deg) ]  */

/*  dr = deg r  = length r -1                    */
```

```
{

/*int *ok;                          x1*/
int i,j,xz ;
int dr,ds;
int nA,nB,nC,nao,nam;

float A[20],B[20],C[20] ;
float R[20],S[20];
float tk,sum1,sum2;

nA = a_deg + 1;      /* OBS : Antalet par. inkl. a0  */
nB = b_deg + 1;      /* OBS : Antalet par. inkl. b0  */
nC = c_deg + 1;      /* OBS : Antalet par. inkl. c0  */
nao = ao_deg + 1;    /* OBS : Antalet par. inkl. ao0 */
nam = am_deg + 1;    /* OBS : Antalet par. inkl. am0  */

/* ---------------------------- */


    sum1=0.0;
    for ( i=0 ; i<=nam-1;i++)
        sum1 +=am[i];

    sum2=0.0;
    for ( i=nA ; i<=(nA+nB-1);i++)
        sum2 +=th[i];

    if (fabs(sum2)<1e-4)
        tk=1.0;
    else
        tk=sum1/sum2;



/* ------------ Lägg till integrator och nollställe ---- */

    nA = nA +1;
    nB = nB +1;


    A[nA-1] = 1.0 ;
    A[nA-2] =  th[1]-1.0 ;                    /*   a[2] = a_1 - a_0
                                              /*   OBS! Ger kompilatorfel
                                              /*   om -th[1] -1.0          !!!
    for(i=2; i<=nA-2; i++)
        A[nA-1-i]= th[i] - th[i-1]   ;
    A[0] = -th[nA-2];                         /*   a[na] = - a_(na-1)  */

    B[nB-1] = th[ nA-1 ];                        /*  ... */
       for(i=1; i<=nB-2; i++)                 /*  b[nb]= b_(nb-1)
        B[nB-1-i]= th[i-1+nA]  + th[i+nA-2] ;
    B[0] = th[ nA+nB-3 ];


    for(i=0; i<=nC-1; i++)
        C[nC-1-i]= amao[i] ;



/* --------Summor till skalfaktorn--------- */


    /*sum1=0.0;
    for ( i=0 ; i<=nam-1;i++)
        sum1 +=am[i];
```

```
    sum2=0.0;
    for ( i=nA ; i<=(nA+nB-1);i++)
        sum2 +=th[i];

    if (fabs(sum2)<1e-4)
        tk=1.0;
    else
        tk=sum1/sum2;*/



    polyred(A,B,C,nA-1,nB-1,nC-1,R,S,&dr,&ds,ok)  ;



/***********   skalfaktor för T-polynom  **************/


    for( i=1 ; i<= nao ; i++)           /*                T-poly   */
        t[i-1]=tk*ao[i-1] / R[dr];      /*  0.68 = Am(1)/B(1)   */

/***********   r = ( z-1 ) * r´  **************/

    r[0] = 1.0  ;                   /*    normerar r,s och t map R[0]  */
    for(i=0; i<=dr-1; i++)
        r[i+1]= ( R[dr-i-1]-R[dr-i])/R[dr];
    r[dr+1] =  -R[0]/R[dr];
    dr += 1;

/***********   s = ( z+1 ) * s´  **************/

    s[0] = S[ds]/R[dr-1];
    for(i=0; i<=ds-1; i++)
        s[i+1]= (S[ds-i]  + S[ds-i-1] )/R[dr-1];
    s[ds+1] = S[0]/R[dr-1];
    ds += 1;


    *r_deg = dr    ;
    *s_deg = ds    ;
    *t_deg = nao -1  ;


}

/*************************************************************************

void new_design(float *theta_from_estimator,void* regulator)
{
    int ok=0;
    typedef struct regulator_type
    {
       int r_deg,s_deg,t_deg;
       float r[MAX_DEG],s[MAX_DEG],t[MAX_DEG];
    };

    struct regulator_type *n;
    n = (struct regulator_type*) regulator;
    prepolydab(theta_from_estimator,des_a_deg,des_b_deg,des_am,des_ao,des_am_de



}

void init_design(float amx[], float aox[], float amx_deg, float aox_deg, float
```

```
{
    int i,j;

    des_am = &amx[0];
    des_ao = &aox[0];
    des_am_deg = amx_deg;
    des_ao_deg = aox_deg;
    des_a_deg = ax_deg;
    des_b_deg = bx_deg;
    des_c_deg = des_ao_deg + des_am_deg;
    for( i=0 ; i<= des_c_deg ; i++)
        des_c[i] = 0;
    for( i=0 ; i<= des_am_deg ; i++)
        for( j=0 ; j<= des_ao_deg ; j++)
            des_c[i+j] += des_am[i] * des_ao[j];
}
```

```
#define MAX_DEG 10

typedef struct regulator_type
{
   int r_deg,s_deg,t_deg;
   float r[MAX_DEG],s[MAX_DEG],t[MAX_DEG];
};

void init_regulator(int *init_pid,float k,float Ti,float Td,float sample_time,i

float compute_u_new(float y_ref,float y);

void change_regulator(void* new_regulator);
```

```
#include "Regulator.h"

int   r_deg,s_deg,t_deg;
float t[MAX_DEG],r[MAX_DEG],s[MAX_DEG];
float yref_vector[MAX_DEG],y_vector[MAX_DEG],u_vector[MAX_DEG];
float regulator_k,regulator_Ti,regulator_Tt,regulator_Td;
float regulator_b,regulator_h,regulator_N,P,I,D;
float regulator_bi,regulator_ar,regulator_bd,regulator_ad;
int regulator_anti_windup,*regulator_init_pid;

/************************************************/
/*       Update u-, yref- and y- vectors        */
/************************************************/

void update_uy_vector(float y,float y_ref,float u)
{
    int i;


    for (i = r_deg ; i >= 2 ; i--)                    /* Shift all vectors one st
        u_vector[i]=u_vector[i-1];

    for (i = s_deg ; i >= 1 ; i--)
        y_vector[i]=y_vector[i-1];

    for (i = t_deg ; i >= 1 ; i--)
        yref_vector[i]=yref_vector[i-1];

    yref_vector[0]=y_ref;                            /* [1] is equal to (t-1) */
    y_vector[0]=y;
    u_vector[1]=u;                                   /* u is from last sample */
}

/************************************************/
/*         Reset rst- and uy- vectors           */
/************************************************/

void init_regulator(int *init_pid,float k,float Ti,float Td,float sample_time,i
{
    int i;

    for (i=0 ; i<=MAX_DEG-1 ; i++)
    {
        yref_vector[i]=0.0;
        y_vector[i]=0.0;
        u_vector[i]=0.0;

        r[i]=0.0;
        s[i]=0.0;
        t[i]=0.0;
    }

    r_deg = s_deg = t_deg = MAX_DEG-1;   /* to initiat "update vectors" */

    regulator_anti_windup = reg_anti_windup;
    regulator_init_pid = init_pid;

    regulator_k = k;
    regulator_Ti = Ti;
    regulator_Td = Td;
    regulator_Tt = Ti*0.1;
    regulator_b = 1;
    regulator_N = 10;
    regulator_h = sample_time / 1000.0;

    regulator_bi = k*regulator_h/Ti;
    regulator_ar = regulator_h/regulator_Tt;
    regulator_bd = k*regulator_N*Td/(Td+regulator_N*regulator_h);
```

```
    regulator_ad = Td/(Td+regulator_N*regulator_h);
    I = 0;
    D = 0;
}

/************************************************/
/*                PID_regulator                 */
/************************************************/

float pid_regulator(float y,float yref)
{
    static float yold;
    float v,u;

    P = regulator_k*(regulator_b*yref - y);
    D = regulator_ad*D - regulator_bd*(y - yold);

    v = P + I + D;

    u = v;

    if (regulator_anti_windup)
    {
        if (v >  0.99) u =  0.99;
        if (v < -0.99) u = -0.99;
    }

    I = I + regulator_bi*(yref - y) + regulator_ar*(u - v);

    yold = y;

    return u;
}


/************************************************/
/*                RST_regulator                 */
/************************************************/

float compute_u_new(float y_ref,float y)
{
    int i;
    static float u;

    update_uy_vector(y,y_ref,u);

    if (*regulator_init_pid)
        u = pid_regulator(y,y_ref);
    else
    {
        u = 0.0;

        for (i = 0 ; i <= t_deg ; i++)
            u += t[i] * yref_vector[i];

        for (i = 0 ; i <= s_deg ; i++)
            u -= s[i] * y_vector[i];

        for (i = 1 ; i <= r_deg ; i++)
            u -= r[i] * u_vector[i];

        if (regulator_anti_windup)
        {
            if(u > 0.99) u = 0.99;
            if(u < -0.99) u = -0.99;

        }
    }
```

```
    return u;
}

void change_regulator(void* new)

{
    struct regulator_type *n;
    int i;

    n = (struct regulator_type*) new;

    asm(" PUSH  IE");
    asm(" AND   0fefdh,IE");

    for (i=0;i<=MAX_DEG;i++)
    {
        r[i]=n->r[i];
        s[i]=n->s[i];
        t[i]=n->t[i];
    }
    r_deg = n->r_deg;
    s_deg = n->s_deg;
    t_deg = n->t_deg;

    asm(" POP   IE ");                                  /* ReEnable interrupts
}
```

```
float to_float(float invalue);

float to_IEEE(float invalue);
```

```
float to_float(float invalue)

{
    float temp;
    temp = invalue;
    IEEE_to_C30(&temp,1);
    return(temp);
}

float to_IEEE(float invalue)

{
    float temp;
    temp = invalue;
    C30_to_IEEE(&temp,1);
    return(temp);
}
```

```
typedef struct {

    /******** Process ******************************/
    float sample_time;
    long  oversampling_rate;
    long  load_disturbance;
    float load_amplitude;
    float load_period;
    float load_bias;
    long  measure_disturbance;
    float measure_amplitude;
    float measure_period;

    /******** Regulator ****************************/
    long  reg_anti_windup;
    long  initial_pid,initial_pid_time;
    float p,i,d;

    /******** Estimator ****************************/
    long    est_anti_windup;
    float   estd_max;
    float   est_min;
    long    deg_a,deg_b;
    float   lambda;
    long    hysteresis;
    float   est_max;

    /********* Design ******************************/
    float Am1,Am2,Am3,Am4,Am5;
    float Ao1,Ao2,Ao3,Ao4,Ao5;
    long  Am_deg,Ao_deg;

    /********* Lowpassfilter ***********************/
    long  lp;
    long  deg_lp;
    float low_lp;

    /********* Bandpassfilter 1 ********************/
    long  bp1;
    long  deg_bp1;
    float low_bp1,high_bp1;

    /********* Bandpassfilter 2 ********************/
    long  bp2;
    long  deg_bp2;
    float low_bp2,high_bp2;

} parameter_type;

typedef struct {

    /* variables that are changed on-line */

    /* FROM LabView */
    float ref_value;

    /* TO and FROM LabView */
    long stop,start,quit;

    /* TO LabView */
    float setpoint,process_output,control_signal;

    float r[MAX_DEG];
    float s[MAX_DEG];
    float t[MAX_DEG];

    float theta[MAX_DEG];
    float error;
```

```
    long pid; /* PID or Adaptive regulator */
} value_type;
void get_parameters(void);
void get_pointers(void);
```

```
#define MAX_DEG 10
#define parameter_addr 0x100100
#define value_addr     0x100200
#include "LabView_Variables.h"

parameter_type  *p;
value_type      *v;

void get_parameters()
{
    p = (parameter_type*) ( parameter_addr );

    sample_time =   to_float(p->sample_time);
    oversampling_rate =      p->oversampling_rate;
    load_disturbance =       p->load_disturbance;
    load_amplitude =        to_float(p->load_amplitude);
    load_period =           to_float(p->load_period);
    load_bias =             to_float(p->load_bias);
    measure_disturbance =    p->measure_disturbance;
    measure_amplitude =     to_float(p->measure_amplitude);
    measure_period =        to_float(p->measure_period);
    a_deg =                  p->deg_a;
    b_deg  =                 p->deg_b;
    lambda =        to_float(p->lambda);
    est_min =       to_float(p->est_min);
    estd_max =      to_float(p->estd_max);
    est_anti_windup =        p->est_anti_windup;
    hysteresis =             p->hysteresis;
    est_max =       to_float(p->est_max);

    am[0]  =                 1.0;
    am[1]  =        to_float(p->Am1);
    am[2]  =        to_float(p->Am2);
    am[3]  =        to_float(p->Am3);
    am[4]  =        to_float(p->Am4);
    am[5]  =        to_float(p->Am5);
    ao[0]  =                 1.0;
    ao[1]  =        to_float(p->Ao1);
    ao[2]  =        to_float(p->Ao2);
    ao[3]  =        to_float(p->Ao3);
    ao[4]  =        to_float(p->Ao4);
    ao[5]  =        to_float(p->Ao5);
    am_deg =                 p->Am_deg;
    ao_deg =                 p->Ao_deg;

    kp =            to_float(p->p);
    Ti =            to_float(p->i);
    Td =            to_float(p->d);
    init_pid =               p->initial_pid;
    pid_time =               p->initial_pid_time;
    reg_anti_windup =        p->reg_anti_windup;

    lp =                     p -> lp;
    deg_lp =                 p -> deg_lp;
    low_lp =        to_float(p -> low_lp);

    bp1 =                    p -> bp1;
    deg_bp1 =                p -> deg_bp1;
    low_bp1 =       to_float(p -> low_bp1);
    high_bp1 =      to_float(p -> high_bp1);

    bp2 =                    p -> bp2;
    deg_bp2 =                p -> deg_bp2;
    low_bp2 =       to_float(p -> low_bp2);
    high_bp2 =      to_float(p -> high_bp2);
}

void get_pointers()
```

```
{
    v = (value_type*) ( value_addr );

    y_ref_IEEE =     (float *)    (&(v -> ref_value));

    pid =            (int*)       (&(v -> pid));
    stop =           (int*)       (&(v -> stop));
    start =          (int*)       (&(v -> start));
    quit =           (int*)       (&(v -> quit));
    y_IEEE =         (float*)     (&(v -> process_output));
    u_IEEE =         (float*)     (&(v -> control_signal));
    r_IEEE =                      (&(v -> r[0]));
    s_IEEE =                      (&(v -> s[0]));
    t_IEEE =                      (&(v -> t[0]));
    theta_IEEE =                  (&(v -> theta[0]));
    est_error_IEEE =              (&(v -> error));
}
```

```
/* $ FltDCreate.c v 1.0 1992-04-28 Anders Carlsson $ */
/*
 * Routines for creating Direct form 2 filters of the DFilter class.
 *
 * Version: 1.0
 * Date:    1992-04-28
 *
 * Copyright © 1992 Anders Carlsson
 */

#include <stdarg.h>
#include <float.h>
#include <math.h>
#include "FiltersP.h"
#define MAX(A,B) (A) > (B) ? (A) : (B)

#ifndef PI
#define PI 3.14159265359
#endif

float hamming(int k,int n)
{
    return (0.54 - 0.46*cos(2*PI*k/(n-1)));
}

float hanning(int k,int n)
{
    return ( 1-cos(2*PI*k/(n-1)) );
}

float blackman(int k,int n)
{
    return (0.42 - 0.5*cos(2*PI*k/(n-1)) + 0.08*cos(4*PI*k/(n-1)));
}

float bartlett(int k,int n)
{
    float num,den;
    num = (float) k * 2.00;
    den = (float) n - 1.00;

    return ( (k<n/2) ? (num/den) : (2.00 - num/den) );
}

float boxcar(int k, int n)
{
    return (float) 1.0;
}

static float (*windows[])() = {
    boxcar,
    bartlett,
    hanning,
    hamming,
    blackman
};

float *new_workspace(int nr_samples,float **ws)
{
    int i,space=nr_samples;
    /* first : find out the smallest power of two greater than nr_samples */
    for( i=1 ; (space >>= 1) ; i++);

    /*
     * Then, the needed space is 2^i + nr_samples. This cludge is nessecary
     * because of the circular buffer feature used in then filter
     * implementation, this datastructure has to be aligned with nearest
     * 2^n address space, and malloc allocates space anywhere...
```

```
     * There should be an aalloc "aligned mallaoc" to avoid this problem.
     */
    space = (1 << i);
    *ws = (float*) malloc((space+nr_samples)*sizeof(float));

    /*
     * The returned pointer is increased with space just to guarantee
     * that the buffer is in the middle of the allocated area.
     */
    return (*ws)+space;
}

DFilter _FltDCreate(float *a,int deg_a,
                    float *b,int deg_b,
                    int length_w,int up,int down)
/*
 * Allocates all space needed for a Direct form II filter. returns NULL if
 * any allocation failed.
 */
{
    DFilter f;
    float *w_origin,*w;

    if (!(NULL == (f = (DFilter) malloc(sizeof(_DirectForm2))))) {
        f->deg_a = deg_a;
        f->deg_b = deg_b;
        f->length_w = length_w;
        f->up = up;
        f->down = down;
        f->m = 0;
        f->y_used = 0;
        f->w = new_workspace(f->length_w,&w_origin);
        f->w_origin = w_origin;
        f->a = (a != NULL) ? a : (float*) malloc(f->deg_a*sizeof(float));
        f->b = (b != NULL) ? b : (float*) malloc((f->deg_b+down+up)*sizeof(float

        if ((w_origin == NULL) ||
            ((f->a == NULL) && deg_a) ||
            (f->b == NULL)) {
            free(w_origin);
            free(f->a);
            free(f->b);
            free(f);
            f = NULL;
        }
    }
    return f;
}

void FltDFree(DFilter f)
/* Frees all allocated space associated with a filter. */
{
    free(f->a);
    free(f->b);
    free(f->w_origin);
    free(f);
    f = NULL;
}

DFilter FltDCreateAB(float *a,int deg_a,float *b,int deg_b)
{
    return _FltDCreate(a,deg_a,b,deg_b,MAX(deg_a,deg_b),1,1);
}

DFilter FltDCreateMA(int no_samples,int up,int down)
{
    DFilter f;
    int i;
```

```
    float n = (float) no_samples;

    if (NULL != (f = _FltDCreate(NULL,0,NULL,no_samples,no_samples,up,down))) {
        /* initialize a,b and w */

        for (i = 0; i<(f->deg_b+down+up) ; f->b[i++] = 1.0/n);
        for (i = 0; i<no_samples ; f->w[i++] = 0.0);
    }
    return f;
}

DFilter FltDCreateFoH(int up,int down)
{
    DFilter f;
    int i;

    if (NULL != (f = _FltDCreate(NULL,0,NULL,2*up,2,up,down))) {
        /* initialize a,b and w */

        for (i = 0; i<f->deg_b ; i++)
            f->b[i] = bartlett(i,f->deg_b);
        for (i = 0; i<f->length_w ; f->w[i++] = 0.0);
    }
    return f;
}

DFilter FltDCreatePFoH(int up,int down,float alfa)
{
    DFilter f;
    int i;
    float slope;

    if (NULL != (f = _FltDCreate(NULL,0,NULL,2*up,2,up,down))) {
        /* initialize a,b and w */

        slope = 2.00*alfa/((float) f->deg_b);
        for (i = 0; i<f->deg_b ; i++)
            f->b[i] = (i<up) ? 1.00 + slope*i : slope*(up - i);
        for (i = 0; i<f->length_w ; f->w[i++] = 0.0);
    }
    return f;
}

float sinc(float alfa)
{
    return (abs(alfa) < FLT_EPSILON) ? 1.00 : sin(alfa)/alfa;
}

DFilter FltDCreateFIRLP(int length,float nyc,int up,int down,int window)
{
    DFilter f;
    float alfa,center,omega2,sum,sf;
    int i;

    if (NULL != (f = _FltDCreate(NULL,0,NULL,(length/up)*up,length/up,up,down))) {
        /* Windowed FIR filter */

        omega2 = 2*PI*nyc/ (float) up; /* cutoff freq relative input fs */
        center = ( (float) f->deg_b - 1.00)/2.00;
        sum = 0;
        for (i = 0 ; i < f->deg_b ; i++) {
            alfa = omega2*((float) i-center);
            f->b[i] = sinc(alfa) * windows[window](i,f->deg_b);
            sum += f->b[i];
        }

        /* Normalize the DC response to 1. */
```

```
        sf = (float) up/sum;
        for (i = 0 ; i < f->deg_b ; i++) f->b[i] *= sf;

        /* Prolong the B-vector for interpolators and such alike */
        for (i=0 ; i < up ; i++)
            f->b[i+f->deg_b] = f->b[i];
    }
    return f;
}

DFilter FltDCreateFIRLPS(int length,float nyc,int up,int down)
{
    float newnyc = nyc - 2/length;
    newnyc = newnyc<0.0 ? 0 : newnyc;
    return FltDCreateFIRLP(length,newnyc,up,down,HAMMING);
}

DFilter FltDCreateFIRBP(int length,float ny1,float ny2,int up,int down,int windo
{
    DFilter f;
    float alfa,beta,center,omega1,omega2,omegac,sum,sf;
    int i;

    if (NULL != (f = _FltDCreate(NULL,0,NULL,(length/up)*up,length/up,up,down) )
        /* Windowed FIR filter */
        ny1 /= (float) up;
        ny2 /= (float) up;
        omega1 = PI*fabs(ny2 - ny1);
        omega2 = PI*(ny1+ny2);
        omegac = 2*PI*sqrt(ny1*ny2);
        center = ( (float) f->deg_b - 1.00)/2.00;
        sum = 0;
        for (i = 0 ; i < f->deg_b ; i++) {
            alfa = omega1*((float) i-center);
            beta = omega2*((float) i-center);
            f->b[i] = sinc(alfa) * cos(beta) * windows[window](i,f->deg_b);
            sum += f->b[i] * cos(omegac * ((float) i-center));
        }

        /* Normalize the response at omegac to 1. */

        sf = (float) up/sum;
        for (i = 0 ; i < f->deg_b ; i++) f->b[i] *= sf;

        /* Prolong the B-vector for interpolators and such alike */
        for (i=0 ; i < up ; i++)
            f->b[i+f->deg_b] = f->b[i];
    }
    return f;
}
```

```
/* $ FltDFIRGetSample.c v 0.1 1992-04-23 Anders Carlsson $ */
/*
 * Less complicated variant of FltDGetSample
 * Assumes f to be a FIR filter and thus can forget about
 * the A-polynomial.
 *
 * Version: 0.1
 * Date:    1992-04-23
 *
 * Copyright © 1992 Anders Carlsson
 *
 */

#include "FiltersP.h"

float FltDFIRGetSample(DFilter f)
{
    int i;
    float y = 0;

    for (i = 0; i < f->deg_b/f->up ; i++)
        y += f->w[i] * f->b[f->m + f->up*i];
    f->m = (f->m + f->down)%f->up;
    return y;
}
```

```
/* $ FltDFIRPutSample.c v 0.1 1992-04-23 Anders Carlsson $ */
/*
 * Simplified (And thus faster) version of FltDPutSample
 * Assumes a zero-degree A polynomial
 *
 * Version: 0.1
 * Date:    1992-04-23
 *
 * Copyright © 1992 Anders Carlsson
 */

#include "FiltersP.h"

void FltDFIRPutSample(DFilter f, float new)
{
    int i;

    for (i=f->length_w-1; i>0 ; i--)
        f->w[i] = f->w[i-1];
    f->w[0] = new;
}
```

```
/* $ FltDGetSample.c v 0.2 1992-04-23 Anders Carlsson $ */
/*
 * FltDGetSample
 * General oputput method for the DFilter class.
 *
 * Version: 0.2
 * Date:    1992-04-23
 *
 * Copyright © 1992 Anders Carlsson
 */

#include "FiltersP.h"

float FltDGetSample(DFilter f)
{
    int i;
    float y = 0.0,new = 0.0;
    /*
     * If We have already calculated an output with the
     * most recent input, insert a new, zero, input.
     */
    if (f->y_used) {
        for (i=f->length_w-1; i>0 ; i--)
            f->w[i] = f->w[i-1];
        f->w[0] = 0.0;
    }
    /*
     * Calculate the A polynomial
     */
    for (i=0 ; i<f->deg_a ; i++)
        new += f->w[i+1]*f->a[i];
    f->w[0] += new;
    /*
     * And the B-polynomial, yelding the output.
     */
    for (i = 0; i < f->deg_b/f->up ; i++)
        y += f->w[i] * f->b[f->m + f->up*i];
    f->m = (f->m + f->down)%f->up;
    f->y_used = 1;
    return y;
}
```

```
/* $ FltDPutSample.c v 0.2 1992-04-23 Anders Carlsson $ */
/*
 * FltDPutSample
 * General input method for the DFilter class.
 *
 * Version: 0.2
 * Date:    1992-04-23
 *
 * Copyright © 1992 Anders Carlsson
 */

#include "FiltersP.h"

void FltDPutSample(DFilter f, float new)
{
    int i;
    /*
     * Make shure the A-polynomial is computed
     * with last sample.
     */
    if (!(f->y_used)) {
        float onew = 0.0;
        for (i=0 ; i<f->deg_a ; i++)
            onew += f->w[i+1]*f->a[i];
        f-w[0] += onew;
    }
    /*
     * Shift sample vector
     */
    for (i=f->length_w-1; i>0 ; i--)
        f->w[i] = f->w[i-1];
    /* Insert new sample */
    f->w[0] = new;
    f->y_used = 0;
}
```

```
/* $ FltDPutSample.c v 0.2 1992-04-23 Anders Carlsson $ */
/*
 * FltDPutSample
 * General input method for the DFilter class.
 *
 * Version: 0.2
 * Date:    1992-04-23
 *
 * Copyright © 1992 Anders Carlsson
 */

#include "FiltersP.h"

void FltDPutSample(DFilter f, float new)
{
    int i;
    /*
     * Make shure the A-polynomial is computed
     * with last sample.
     */
    if (!(f->y_used)) {
        float onew = 0.0;
        for (i=0 ; i<f->deg_a ; i++)
            onew += f->w[i+1]*f->a[i];
        f->w[0] += onew;
    }
    /*
     * Shift sample vector
     */
    for (i=f->length_w-1; i>0 ; i--)
        f->w[i] = f->w[i-1];
    /* Insert new sample */
    f->w[0] = new;
    f->y_used = 0;
}
```