CODEN: LUTFD2/(TFRT-5356)/1-143/(1986)

Identifiering av parametrar i delvis kända tidskontinuerliga modeller — En implementationsstudie

Bo Pettersson

Institutionen för Reglerteknik
Lunds Tekniska Högskola
November 1986

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

Document name
Master Thesis

Date of issue
November 1986

Document Number
CODEN:LUTFD2/(TFRT-5356)/1-143/(1986)

Author(s)
Bo Pettersson

Supervisor
Rolf Johansson

Sponsoring organisation

Title and subtitle
Identifiering av parametrar i delvis kända tidskontinuerliga modeller; En implementationsstudie.
(Identification of parameters in partioned continous time models;An implementation study).

Abstract

Presents an implementation of a method for estimating the parameters in partially known continous time models.

The paper uses the fact that it is possible to obtain a realisable differential operator if the signals is filtred by analog low-pass filters of sufficient order.

The problems associated with analog-to-digital translation of filters is solved by interpolating the sampled signal between the sampling points. Filters obtained by the use of first order interpolation is implemented, and formulas is given for the general case of m:th order interpolation.

The implementation was done on IBM-AT in the language Modula-2.

Some features of the implementation is that it is possible to choose the period for the estimation as a multiple of the sampling period, and that the filter coefficients is calculated by the program.

Key words
On-line identification, Partially known systems, analog-to-digital translation, implementation.

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
Swedish

Number of pages
143

Recipient's notes

Security classification

Innehåll

1. Inledning

En mycket viktig del av design av reglersystem är att skaffa sig en modell av den process som man vill reglera. En sådan modell kan man skaffa sig genom att tillämpa fysikaliska principer eller genom att identifiera processen. Identifikationen går till på så sätt man ansätter en modell som man tror beskriver den process man är intresserad av, sedan utnyttjar man statistiska metoder för att bestämma parametrarna i den ansatta modellen.

Det finns finns två principiellt skilda metoder att genomföra identifikationen på. Den första metoden går ut på att man först samlar in all data man tror sig behöva sedan kan man i lugn och ro genom att prova olika modeller söka sig fram till den modell som för det ändamål man är intresserad av beskriver den aktuella processen väl. Denna metod kallar man vanligen för off-line identifikation.

Den andra metoden, den som vi behandlar här, baseras på att man förfinar modellen allt eftersom man får tillgång till nya data. Detta sker vanligen i real tid, varför metoden kallas on-line identifikation.

Man konstaterar direkt att den första metoden ger en mycket större frihet åt den som utför identifikationen. När man utför identifieringen off-line kan man ju prova flera olika modellansatser samt flera olika identifikationsmetoder t ex frekvens- och transient-analys.

Vid on-line identifiering av dynamiska system använder man sig vanligen rekursiva metoder där man antar att den process som ska identifieras är ett tidsdiskret linjärt system, ARMA-modeller.

Det finns å andra sidan klasser av system för vilka det kan vara önskvärt att on-line direkt skatta den tidskontinuerliga processens parametrar.

Antag till exempel att man genom att tillämpa fysikaliska principer kommit fram till att en viss modell beskriver systemet väl, man har då en tidskontinuerlig modell som man vill skatta parametrarna i. Parametrarna i denna modell har då en direkt fysikalisk tolkning. Om man i detta fall använder sig av rekursiva metoder baserade på ARMA-modeller så kommer man att skatta parametrarna i det tidsdiskreta system man får när man samplar den aktuella processen.

Man finner alltså att det finns nackdelar med att använda sig av ARMA-modeller.

En exakt översättning av ett kontinuerligt till ett tidsdiskret system baseras på matrisexponentiering enligt kända resultat. Detta får ogynnsamma konsekvenser, till exempel blir parametrarna i den tidsdiskreta överföringsfunktionen beroende av sampelfrekvensen. En annan nackdel är att det tidsdiskreta systemets parametrar beror på ett olinjärt sätt av det tidskontinuerliga systemets parametrar. Ytterligare en nackdel är att det system man får genem diskretisering mycket väl kan ha fler parametrar än motsvarande kontinuerliga. Det senare innebär att man får fler parametrar att skatta än man egentligen skulle behöva.

Vi antar nu att någon av det tidskontinuerliga systemets parametrar är känd. Eftersom det tidsdiskreta systemets parametrar är olinjära kombinationer av det det tidigares pareametrar så har vi små möjligheter att utnyttja denna kunskap. Man är alltså tvungen att skatta samtliga parametrar i det tidsdiskreta systemet.

Dessa nackdelar med att använda sig av ARMA-modeller har lett till att man undersökt om det är möjligt att hitta någon motsvarighet till dessa modeller som kan användas för kontinuerliga system.

En svårighet med att identifiera det kontinuerliga systemet är att beskrivningen av sådana system baseras på differential-operatorn, medan diskreta system modelleras med skift-operatorn. Differential-operatorn har den nackdelen att den är en obegränsad operator, detta leder till att denna ej är exakt realiserbar. Man har därför sökt finna metoder där man beskriver det kontinuerliga systemet i någon realiserbar operator.

I den metod som används i denna rapport utnyttjas det faktum att man genom att lågpassfiltrera signalerna, med ett filter av tillräcklig ordning, innan differential-operatorn får verka på dessa kan skaffa sig en realiserbar operator.

Den metod som beskrivs är formulerad i det analoga tidsplanet. Den baseras på att identifieringsobjektets in- och utsignaler får passera en viss familj av lågpassfilter. Med tillgång till utsignalerna från dessa filter kan man sedan bilda de lågpassfiltrerade differential-operatorerna. Den estimeringsalgoritm som används är den rekursiva minsta kvadratmetod som framgångsrikt utnyttjats för ARMA-modellerna.

Dessa lågpassfilter är i metoden formulerade som analoga filter. Detta innebär att det blir svårt att ändra egenskaper hos filtren. Man får alltså en onödig begränsning på metoden. För att slippa denna begränsning realiserades filtren digitalt, detta gör det som bekant enkelt ändra filtren.

En exakt översättning av de analoga filtren till digitala filter är endast möjlig när man har information om insignalen till dessa mellan sampelvärdena. Detta faktum följer ur att det finns ett oändligt antal styckvis kontinuerliga funktioner som har samma sampelvärden, dessa utgör vad man brukar kalla en ekvivalensklass.

Den kända övergången som görs mellan kontinuerliga och diskreta system i reglertekniska sammanhang grundar sig på att den regulator som styr processen är implementerad med ZOH-rekonstruktion. Detta betyder att man inskränker sig till att arbeta med insignaler till systemet som tillhör klassen av styckvis h-konstanta funktioner, där h-konstanta betyder konstant mellan samplingspunkterna.

Det visar sig att för att förbättra filter-översättningen är det önskvärt att kunna arbeta med andra klasser av insignaler. I den implementering som gjorts finns tillgängliga översättningar för klassen av h-linjära funktioner. I rapporten beskrivs även det hur en implementering kan göras i det allmänna fallet med insignaler tillhörande klasssen av styckvis h-polynom. Formlerna i det sistnämnda fallet är givna på tillståndsform, som lämpar sig väl för dator-beräkningar.

Metoden implementerades på IBM-AT i Modula-2. Eftersom identifieringen sker i realtid måste man ha tillgång till realtids-primitiver, den vid institutionen utvecklade realtids-kärnan utnyttjades får detta ändamål. Den implementering som gjorts är ett generell i den meningen att användaren enkelt kan, för att nämna några saker,

* välja och ändra ordningen på den modell som ska anpassas till processen som ska identifieras,

* välja estimeringsperioden relativt fritt, den måste vara en heltalsmultipel av samplingsperioden,

* betrakta vissa parametrar som kända, endast de okända parametrarna skattas,

* ändra filtrens egenskaper så de optimala uppnås, programmet räknar ut filterkoefficienter,

* plotta de skattade parametrarnas historia, i y-led finns en zoom-funktion.

För att reglera den process som programmet testades mot implementerades en regulator bestående av två kaskadkopplade PID-regulatorer.

2. Processbeskrivning (Några självklarheter).

Under denna rubrik har jag valt att sammla påpekanden som kan vara till hjälp i den fortsatta diskussionen.

Svarta lådor.

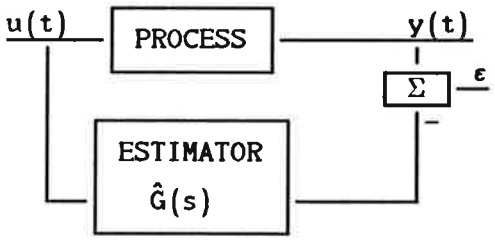
Antag att det någonstans existerar en process, denna är så beskaffad att vi kan mäta dess in- och utsignaler. Detta åskådliggörs på följande sätt

där u(t) är insignalen och y(t) är utsignalen.

Vi antar nu att vi, genom träget arbete eller med hjälp av så kallad himmelsk inspiration, lyckats konstatera att denna process är linjär, tidsinvariant och inte minst kausal.

Då gäller att det är möjligt att till denna process ordna en överföringsfunktion G(s), denna gudabenådade tingest som fullständigt beskriver beteendet hos vår process.

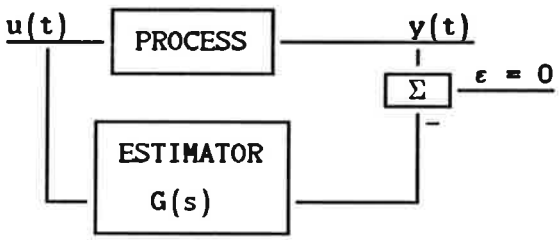
Antag nu att att vi är trötta på att höra berättas om denna fantastiska G(s), vi vill se den med egna ögon. Efter en stunds funderingar konstruerar vi följande apparat



där vi genom att variera $\hat{G}(s)$ kan få $|\varepsilon|$ att bli godtyckligt litet. Vi ska inte närmare gå in på hur denna apparat kan förbättras, det är ju vad resten av rapporten handlar om.

Efter många sena nätters arbete så inträffar det fantastiska, $\varepsilon = 0$.

Vi ritar nu om vårt blockschema på enligt följande figur.

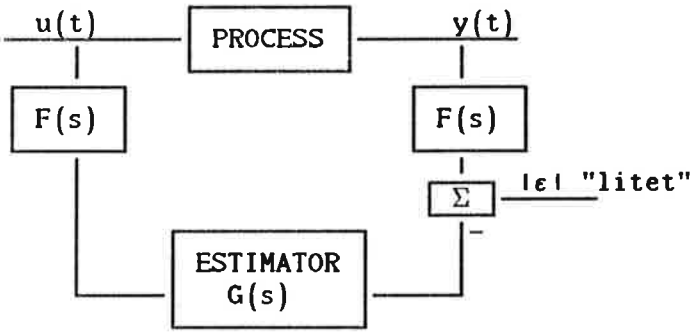


I den verkliga världen visar det sig vara mycket svårt att hitta processer som uppfyller de krav vi hade på vår PROCESS. Detta innebär att vi inte längre kan kräva $\varepsilon = 0$ utan säger att den sista figuren gäller när $|\varepsilon|$ är "litet".

Från och med nu så kommer vi i denna rapport att nästan uteslutande att arbeta med den nedre lådan i den sista figuren ovan.

Svarta lådor och filter.

Följande dag roar vi oss åter med våra lådor. Antag nu att någon hjälpsam person meddelar att det inte är processens verkliga in- och utsignaler som vi använder oss av. Den hjälpsamme har i själva verket sett till att båda signalerna passerar två stycken exakt lika filter, med överföringsfunktion F(s). Nu inträffar något märkligt, vi kan trots det som nyss sagts oss hävda att processen beskrivs av G(s).



Det är till denna figur man ska vända sig vid spörsmål om effekten av den filtrering som görs i kapitlet om den lågpassfiltrerade modellen av systemet.

Ovan hade vi redan innan vår hjälpsamme vän mixtrade med våra lådor ställt in dessa på optimalt sätt, vi fann då att $|\varepsilon|$ blev "litet" även efter att han hade mixtrat med dessa. Vi undrar nu vad som hänt om han även mixtrat med rattarna för $\hat{G}(s)$, skulle vi då fått samma resultat som dagen innan, med kriteriet $|\varepsilon|$ "litet". På denna väsentliga fråga svarar jag: Det beror på.

3. En lågpassfiltrerad modell av systemet

Vi ska i detta avsnitt se hur man genom att lågpassfiltrera in- och utsignaler från ett system kan bilda överföringsfunktionen för systemet.

Resonemanget kan ses som ett specialfall av den metod som används av Young [4].

Några tolkningar till den på detta sätt erhållna representationen görs i termer av icke-observerbara moder. Det är viktigt att påpeka att dessa tolkningar är författarens egna.

Alla resonemang i detta avsnitt genomförs i analog miljö dvs tidskontinuerliga signaler. I senare avsnitt visas hur en analog implementering av dessa filter kan göras.

Den klass av system som behandlas här har en överföringsfunktion av typen

$$G(s) = \frac{B(s)}{A(s)} = \frac{b_1 s^{n-1} + b_2 s^{n-2} + \ldots + b_n}{s^n + a_1 s^{n-1} + \ldots + a_n} \qquad (1)$$

Vi kräver alltså grad B(s) < grad A(s), dvs ingen direktterm.

Denna klass av modeller uppkommer naturligt t ex när man linjäriserar ett system kring någon arbetspunkt. Senare kommer vi att utöka klassen av system som kan beskrivas av denna modell genom att låta koefficienterna i A(s) och B(s) att vara "svagt" tidsberoende.

Vi inför nu operatorn

$$\Lambda = \frac{1}{1 + p\tau} \qquad (2)$$

där p = d/dt. Alternativt får man med Laplace-transformen

$$\Lambda = \frac{1}{1 + s\tau} \qquad (3)$$

På grund av denna likhet mellan representationen i tids- och frekvensdomänen låter jag i det följande framgå av sammanhanget framgå om det är operatorn $\Lambda(p)$ eller dess Laplace-transform $\Lambda(s)$ som avses.

Man lägger märke till att Λ kan tolkas som ett 1:a ordningens lågpassfilter med brytfrekvens $1/\tau$.

Vi kan nu lösa ut s, och får

$$s = \frac{1 - \Lambda}{\Lambda\tau} \qquad (4)$$

Eftersom det gäller att $Y(s) = G(s)U(s)$ har vi $A(s)Y(s) = B(s)U(s)$ eller med polynomen utskrivna

$$(s^n + a_1 s^{n-1} + \ldots + a_n)Y(s) = (b_1 s^{n-1} + b_2 s^{n-2} + \ldots + b_n)U(s) \qquad (5)$$

Sätter vi nu in ekv (4) i ekv (5), får vi

$$\left[\left[\frac{1 - \Lambda}{\Lambda\tau}\right]^n + a_1\left[\frac{1 - \Lambda}{\Lambda\tau}\right]^{n-1} + \ldots + a_n\right]Y(s) =$$

$$= \left[b_1\left[\frac{1 - \Lambda}{\Lambda\tau}\right]^{n-1} + b_2\left[\frac{1 - \Lambda}{\Lambda\tau}\right]^{n-2} + \ldots + b_n\right]U(s) \qquad (6)$$

En multiplikation med $(\Lambda\tau)^n$ på båda sidor ger

$$\left[(1-\Lambda)^n + a_1(\Lambda\tau)(1-\Lambda)^{n-1} + \ldots + a_n(\Lambda\tau)^n\right]Y(s) =$$

$$\left[b_1(\Lambda\tau)(1-\Lambda)^{n-1} + b_2(\Lambda\tau)^2(1-\Lambda)^{n-2} + \ldots + b_n(\Lambda\tau)^n\right]U(s) \qquad (7)$$

Det kan nu vara intressant att se vad ekv (7) motsvaras av i s-planet, enkla algebraiska räkningar ger

$$\left[\frac{s^n}{(1+s\tau)^n} + a_1\frac{s^{n-1}}{(1+s\tau)^n} + \ldots + a_n\frac{1}{(1+s\tau)^n}\right]\tau^n Y(s) =$$

$$= \left[b_1\frac{s^{n-1}}{(1+s\tau)^n} + b_2\frac{s^{n-2}}{(1+s\tau)^n} + \ldots + b_n\frac{1}{(1+s\tau)^n}\right]\tau^n U(s) \qquad (8)$$

Vi ser nu att multiplikationen med $(\Lambda\tau)^n$ gav att det nu är de lågpassfilrerade signalerna

$$Y_f(s) = \frac{1}{(1 + s\tau)^n}\,Y(s) \quad \text{och} \quad U_f(s) = \frac{1}{(1 + s\tau)^n}\,U(s) \qquad (9)$$

som uppfyller ekv (5) , $A(s)Y_f(s) = B(s)U_f(s)$.

Detta är dock ej observerbart i överföringsfunktionen ty

$$G_f(s) = \frac{Y_f(s)}{U_f(s)} = \frac{Y(s)}{U(s)} = G(s) \qquad (10)$$

Det bör också påpekas att det är möjligt att utveckla en teori där man betraktar systemet i Λ-planet. Denna utveckling beskrivs i R. Johansson [1]. I denna

rapport kommer jag att använda mig av ekv (7), men det kan ändå vara intressant att se lite på hur en sådan utveckling kan gå till.

Vi utgår från ekv (7), med hjälp av binomialsatsen kan denna skrivas om till

$$(1 + \alpha_1\Lambda + \alpha_2\Lambda^2 + \ldots + \alpha_n\Lambda^n)Y(s) =$$

$$= (\beta_1\Lambda + \beta_2\Lambda^2 + \ldots + \beta_n\Lambda^n)U(s) \qquad (11)$$

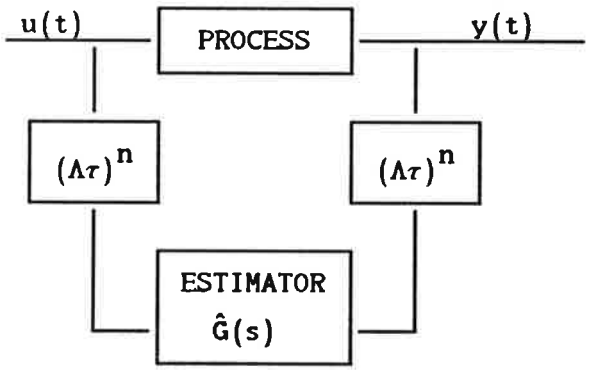
Det är nu möjligt att översätta t ex stabilitetskriterier i s-planet till motsvarande kriterier i Λ-planet. Vi ser då att vänstra halvplanet under avbildningen $\Lambda = 1/(1 + s\tau)$ övergår i cirkelskivan

$$\{ \Lambda : \Lambda = 1/2 + \exp(i\theta)/2, \text{ där } 0 \leq \theta < 2\pi. \}$$

Intressant är att notera den likhet som råder med fallet övergång från kontinuerligt till tidsdiskret system. Då gäller ju, om man använder sig av ZOH-rekonstruktion, att det kontinuerliga systemets poler avbildas innanför enhetscirkeln (naturligtvis under förutsättningen att man har tillräckligt hög sampelfrekvens).

Om vi nu lämnar denna lilla parentes och återvänder till ekv (7), så konstaterade vi att lågpassfiltreringen av in- och utsignal ej var observerbar i överföringsfunktionen. Av detta kan man förledas att tro att man kan bortse från denna filtrering, men det står ju samtidigt klart att filtreringen påverkar signalerna i sammanhanget. Vi ska i detta läge inte gå längre i denna fråga utan nöjer oss med att konstatera att filtreringen är något som görs med signalerna utan att påverka processen, dvs den "arbetar" fortfarande med de ofiltrerade signalerna.

Eftersom de härledda ekvationerna ska användas till att skatta överföringsfunktionen, byter vi ut G(s) ovan mot $\hat{G}(s)$ och kan då rita upp följande blockschema.



Vi återkommer även till frågan om icke-observerbara moder i samband med kompensation för eventuellt församplingsfilters inverkan på observerad överföringsfunktion.

4. En analog realisering av lågpassfiltren

Vi har tidigare sett hur det var möjligt att skriva om systemekvationen i polynom av Λ. Det är nu hög tid att se på hur en möjlig realisering av denna kan ta sig ut.

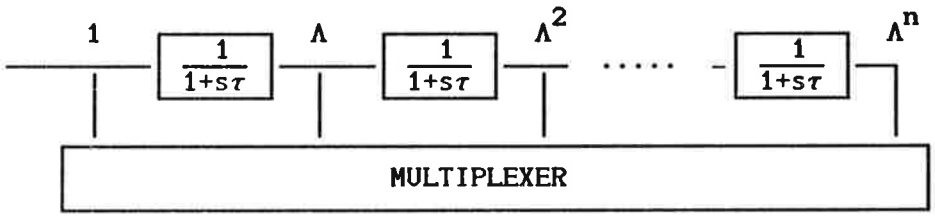
Polynomen har alltså följande utseende

$$P_k(\Lambda) = (\Lambda\tau)^{n-k}(1-\Lambda)^k \qquad (12)$$

där $0 \leq k \leq n$.

För att bilda polynomen $P_k(\Lambda)$ behöver vi ha tillgång till $1, \Lambda, \ldots, \Lambda^n$.

Följande nät ger oss detta



Med signalerna från multiplexern kan vi sedan $P_k(\Lambda)$, eftersom τ är känd.

Man behöver alltså två nät av denna typ, ett för processens insignal (u) och ett för dess utsignal (y).

Villkoret grad A(s) > grad B(s), ger att för insignalen (u) gäller $0 \leq k \leq n-1$. Den ofiltrerade insignalen behövs alltså ej i nätet för densamma.

Som vi tidigare har sett gäller samma relationer i tidsplanet som i s-planet, om vi nu låter $P_k(\Lambda(p)) = P_k$, så kan vi skriva om ekv (7) som ekv (13).

$$(P_n + a_1P_{n-1} + \ldots + a_nP_0)y(t) =$$

$$= (b_1P_{n-1} + b_2P_{n-2} + \ldots + b_nP_0)u(t) \qquad (13)$$

Det är denna ekv som vi senare, i samband med identifieringen, ska arbeta med.

Vi observerar att ekv (13) gäller vid tidpunkten t, detta gör det möjligt för oss att mäta P_ny, ..., P_0y, $P_{n-1}u$, ..., P_0u vid en godtycklig tidpunkt och ekv (13) ska gälla.

Detta är något som alltid gäller för tidskontinuerliga in-utsignal-relationer medan

för tidsdiskreta in-utsignal-relationer motsvarande inte gäller eftersom dessa beror på signalernas tidigare värden.

Denna observation är mycket viktig ty, som vi senare ska se, så är det detta faktum som gör det möjligt att variera estimeringsperioden utan att ändra samplingsfrekvensen.

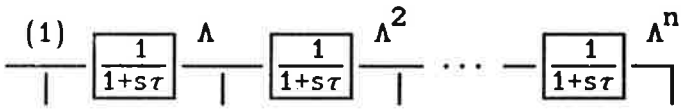
5. Diskreta approximationer av lågpassfiltren

I detta avsnitt ska vi se hur det går till att översätta från analogt till digitalt filter. Vi har tidigare sett hur man i det tidskontinuerliga fallet kunde göra en exakt övergång mellan en representation i termer av filren och överföringsfunktionen. Det visar sig att så ej är fallet när filtren ska realiseras digitalt utan man är tvungen att approximera.

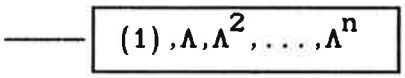
All digital signalbehandling kräver att man har bandbegränsade signaler, i vårt fall rör det sig om signaler med spektra av lågpasstyp. Vad gäller val av samplingsfrekvens har man Shannons samplingsteorem som ger en lägsta teoretisk gräns för denna. De filterapproximationer som tas fram här gör det möjligt att använda en relativt låg samplingsfrekvens, som vi senare ska se är detta ett krav för att den digitala realiseringen ska gå att genomföra.

Nedan kommer jag att förutsätta att processen som ska identifieras styrs av en digital regulator, om man har detta krav visar det sig nämligen att ena halvan av implementeringen är lätt att genomföra.

För den fortsatta diskussionen görs följande förenkling i blockschemat

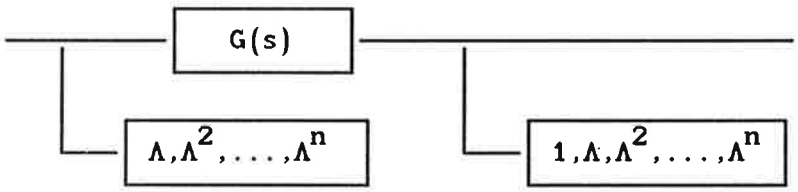


ersätts med



.

Med analog realisering av filtren får man följande blockschema



Fallet med kända insignaler till filtren, ZOH.

Om vi nu skriver filtret $\Lambda^k(p)$ på tillståndsform så får vi

$$\frac{dx(t)}{dt} = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) \qquad (14)$$

där A, B, C är någon lämplig representation av filtret.

Detta system kan lösas, och på känt sätt fås

$$x(t) = e^{A(t-t_0)}x(t_0) + \int_{t_0}^{t} e^{A(t-s)}Bu(s)ds$$
$$y(t) = Cx(t) \; . \qquad (15)$$

Speciellt säger oss denna formel att när formen på u(t) är känd så kan integrationen genomföras och man får en enklare formel.

Denna iakttagelse är grunden till översättningen mellan tidskontinuerligt system och tidsdiskret system, ty formen på insignalen är ju känd och given av den typ av rekonstruktion man använder, vanligen zero-order-hold.

Vi ska nu se hur man i implementeringen av våra lågpassfilter kan utnyttja att processen som ska identifieras styrs av en digital regulator. Vanligen gäller ju att den digitala regulatorn är realiserad med zero-order-hold rekonstruktion, dvs utsgnalen från regulatorn är konstant under sampelperioden. Man har alltså

$$u(t) = u(t_k) \quad , \; t_k \leq t < t_{k+1} \quad \text{där } k = 0,1,2,\ldots \qquad (16)$$

Med denna insignal till vårt filter är det alltså möjligt att utföra integrationen. Man får

$$x(t_{k+1}) = \Phi x(t_k) + \Gamma u(t_k)$$
$$y(t_k) = Cx(t_k) \qquad (17)$$

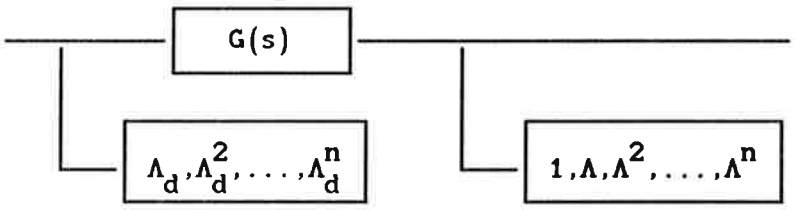
där

$$\Phi = e^{Ah} \; , \qquad \Gamma = \int_0^h e^{As}dsB \; .$$

Vi lägger speciellt märke till att i sampelögonblicken ger det analoga och det digitala filtret samma utsignal.

Vi inför nu följande beteckning Λ_d^k ,vilket betyder att filtret Λ_k är implementerat digitalt

Det är nu möjligt att rita ett nytt blockschema med exakt samma egenskaper som det förra, under förutsättningen att processen styrs med en digital regulator.



Det enda som förutsatts hittills är att insignalen till filtren är bandbegränsad samt att processen som ska identifieras styrs av en digital regulator. Med dessa förutsättningar har vi lyckats lösa halva problemet med implementeringen av filtren. Det svåraste problemet återstår dock.

Fallet med icke kända insignaler.

Med denna rubrik menas att vi känner väldigt lite om insignalen till vårt filter som vi ska implementera digitalt, egentligen vet vi bara att signalen är bandbegränsad. I detta fall är man alltså tvungen att göra approximationer.

Problemet är här lite annorlunda än vad som är vanligt när man ska implementera ett digitalt filter. Då gäller ju t ex att man vill uppfylla en viss kravspecifikation i frekvensplanet, dvs man är inte så intresserad av att implementera en viss överföringsfunktion exakt. I det för oss intressanta fallet är vi dock mycket måna om att göra en så exakt översättning från analogt till diskret som möjligt.

Vi ska först se på en standardmetod som används för att översätta ett analogt filter till ett digitalt dito. Metoden som går tillbaka på trapetsregeln för numerisk integration har många namn Tustin approx., bilinjär transformation och Möbius transformation, jag kommer i fortsättningen att använda mig av den i reglertekniska sammanhang vanligaste nämligen Tustin.

För att se lite på hur Tustins metod fungerar så härleder vi denna.

Vi har som vanligt

p = d/dt = differentialoperatorn

q = (framåt)förskjutningsoperatorn

Enligt Taylors formel gäller

$$f(t+h) = f(t) + \frac{h}{1!}f'(t) + \frac{h^2}{2!}f''(t) + \ldots \qquad (18)$$

man observerar att

$$qf(t) = f(t+h) \qquad \text{och} \qquad p^k f(t) = f^{(k)}(t)$$

sätter man in detta i ekv (18) ovan fås operatorsambandet

$$qf(t) = (1 + \frac{h}{1!}p + \frac{h^2}{2!}p^2 + \ldots)f(t) = e^{ph}f(t) \qquad (19)$$

dvs $q = e^{ph}$.

Formellt leder detta till $p = (\ln q)/h$.

Vi har nu följande utveckling

$$\ln x = 2\{[\frac{x-1}{x+1}] + \frac{1}{3}[\frac{x-1}{x+1}]^3 + \frac{1}{5}[\frac{x-1}{x+1}]^5 + \ldots\} \quad , x > 0. \qquad (20)$$

Gör man en formell utveckling av ln(q) fås alltså sambandet

$$p = \frac{\ln q}{h} = \frac{2}{h}\{[\frac{q-1}{q+1}] + \frac{1}{3}[\frac{q-1}{q+1}]^3 + \frac{1}{5}[\frac{q-1}{q+1}]^5 + \ldots\} \qquad (21)$$

Om man nu approximerar och bara tar med den första termen i utvecklingen, så fås

$$p' = \frac{2}{h}\frac{q-1}{q+1} \qquad (22)$$

detta känner man igen som Tustins approximation. Ett resonemang snarlikt det vi gjorde ovan går naturligtvis att genomföra i frekvensplanet, man får $z = e^{sh}$ och har sedan att göra samma serieutveckling som ovan fast denna gång i det komplexa talplanet. Man får

$$s' = \frac{2}{h}\frac{z-1}{z+1} \qquad (23)$$

När man använder denna approximation gör man på följande sätt:

Man utgår från den kontinuerliga överföringsfunktionen G(s) och sätter för den diskreta överföringsfunktionen H(z)

$$H(z) = G(s') = G[\frac{2}{h}\frac{z-1}{z+1}] \qquad (24)$$

En egenskap hos Tustin-approximationen som är värd att notera är att det vänstra halvplanet Re $s < 0$ avbildas på cirkelskivan $|z| < 1$. Detta betyder att denna approximation alltid ger stabila tidsdiskreta filter om det tidkontinuerliga

filtret är stabilt.

Nackdelen med denna metod är att den ger frekvensdistorsion, det gäller ju på frekvensaxeln att

$$H(e^{i\omega}) = G\left[\frac{2}{h}\,\frac{e^{i\omega}-1}{e^{i\omega}+1}\right] = G\left[\frac{2i}{h}\tan\left[\frac{\omega h}{2}\right]\right] \qquad (25)$$

Detta ska då jämföras med vad vi skulle fått utan frekvensdistorsion : $H(e^{i\omega}) = G(i\omega)$. Det går att modifiera Tustin så att man ej får någon distorsion vid frekvensen ω_1, vi sätter istället

$$s' = \frac{\omega_1}{\tan(\omega_1 h/2)}\,\frac{z-1}{z+1} \qquad (26)$$

man ser direkt att då gäller $H(e^{i\omega}1) = G(i\omega_1)$. Man ska dock komma ihåg att man fortfarande får distorsion vid alla andra frekvenser.

För att få en bra approximation $\Lambda_d^k(z)$ så är man tvungen att låta ωh vara litet för alla tillåtna ω. Detta betyder att man måste ha relativt hög sampelfrekvens.

Detta ledde författaren till att söka efter andra tänkbara approximationer, nedan föreslås en metod som har många fördelar. Metoden är en vidarebyggnad på det resonemang som fördes i samband med ZOH-rekonstruktionen.

Approximation via interpolation.

Till att börja med genomförs resonemangen i tidsplanet, sedan visas hur dessa kan tolkas i frekvensplanet.

Iden är att man bygger vidare på den metod som gjorde det möjligt att implementera filtren exakt i fallet med ZOH. I det fallet var ju insignalen given av sina värden i sampeltidpunkterna. Att insignalen var konstant kan betraktas som att man interpolerar med ett 0:te ordningens polynom. Vi kommer nu att se att det är möjligt att betrakta även interpolation av högre ordning.

Viktig i detta sammanhang är Newtons interpolations-formel som säger att om vi känner signalens värde i (n+1) stycken punkter så kan vi hitta ett polynom $p_n(t)$, där grad $p_n = n$, som går genom dessa (n+1) punkter. Detta polynom p_n är unikt.

Vi låter signalen vara f(t), och $f(t_k)$, $k = 0,1,\ldots,n$ är kända, då ska alltså gälla

$$p_n(t_k) = f(t_k), \qquad t_k = kh, \qquad k = 0,1,\ldots,n. \qquad (27)$$

Det gäller nu att

$$f(t) = p_n(t) + R$$

där

$$p_n(t) = \alpha_0 + \alpha_1(t-t_0) + \alpha_2(t-t_0)(t-t_1) + \ldots +$$

$$+ \alpha_n(t-t_0)(t-t_1)\ldots(t-t_{n-1}) \qquad (28)$$

och

$$R = \frac{f^{(n)}(\xi)}{n!}(t-t_0)(t-t_1)\ldots(t-t_{n-1})(t-t_n), \quad \xi\in[t_0,t_n] \qquad (29)$$

Koefficienterna α_k bestäms enkelt genom att lösa följande ekvationssystem, som fås genom att utnyttja ekv(27).

$$\begin{cases} \alpha_0 & = f(t_0) \\ \alpha_0 + \alpha_1 h & = f(t_1) \\ \alpha_0 + 2\alpha_1 h + 2\alpha_2 h^2 & = f(t_2) \\ \cdot \quad \cdot \quad \cdot & \cdot \\ \alpha_0 + n\alpha_1 h + n(n-1)\alpha_2 h^2 + \ldots + n(n-1)(n-2)\ldots 2\cdot 1\alpha_n h^n & = f(t_n) \end{cases} \qquad (30)$$

Detta ekvationssystem löses med framåtsubstitution.

Vi ska framledes se att det är en fördel att ha $p_n(t)$ på formen

$$p_n(t) = a_0 t^n + a_1 t^{n-1} + \ldots + a_{n-1}t + a_n \qquad (31)$$

För att göra denna översättning kan man utnyttja Stirling-talen av 1:a sorten, det gäller nämligen att

$$t(t-1)(t-2)\ldots(t-n+1) = \Sigma_1^n s_k^n t^k \qquad (32)$$

Stirling-talen av 1:a sorten kan bestämmas ur följande rekursiva formel

$$s_k^{n+1} = s_{k-1}^n - n s_k^n \qquad (33)$$

med

$$s_n^n = 1, \quad s_k^n = 0 \quad \text{om } k \le 0 \text{ eller } k \ge n+1 .$$

I appendix 3 finns några av Stirling-talen tabulerade.

Vi antar i det fortsatta att omskrivningen är gjord och att vi har

$$p_n(t) = a_0 t^n + a_1 t^{n-1} + \ldots + a_{n-1} t + a_n \quad . \tag{31}$$

Dessa kunskaper ska vi nu utnyttja för att göra en filter implementering, tidigare hade vi visat att det var möjligt att ange filtren $\Lambda^k(p)$ på formen

$$\begin{aligned} x(t) &= e^{A(t-t_0)}x(t_0) + \int_{t_0}^{t} e^{A(t-s)}u(s)ds \\ y(t) &= Cx(t) \quad . \end{aligned} \tag{34}$$

Om vi nu antar att insignalen är given av $u(t) = p_n(t)$, så har vi att beräkna integraler av typen $\int \exp(A(t-s)t^k ds$ över ett sampelintervall.

Man ställer sig då frågan om vilket samplingsintervall som är bäst. Eftersom utsignalen från filtren ska användas endast för att estimera överföringsfunktionen finns inget direkt krav på att utsignalen ej får vara fördröjd några sampel.

Man har alltså frihete att välja integrationsintervallet så att feltermen R minimeras. Ett studium av denna ger vid handen att

$$R^*(t) = |\Pi_0^n (t-t_k)| \tag{35}$$

är minst då man centrerar integrationsintervallet i förhållande till intervallet $[t_0,t_n]$. Man ser även att R^* är symmetrisk map $t = (t_0+t_n)/2$. Detta medför att man för att minimera feltermen behöver införa en fördröjning på maximalt int((n-1)/2). Det optimala integrationsintervallet betecknar vi med $[t_m,t_{m+1}]$, vi får då

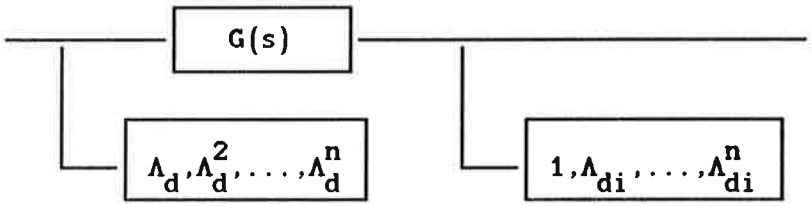
$$\begin{aligned} x(t_{m+1}) &= e^{Ah}x(t_m) + \int_{t_m}^{t_{m+1}} e^{A(t_{m+1}-s)}u_n(s)ds \\ y(t_m) &= Cx(t_m) \quad . \end{aligned} \tag{36}$$

Den ovan angivna realiseringen är optimal i den meningen att man utnyttjar minsta möjliga information om insignalen. Detta innebär att man ej behöver använda sig av så hög samplingsfrekvens som t ex med Tustins metod.

Vi observerar att detta system har samma systemmatris Φ som det system man fick när man använde 0:te ordningens interpolation (ZOH). Detta är en stor fördel med metoden, eftersom vi direkt kan se att stabiliteten bevaras vid övergång från kontinuerligt system till diskret system. Tidigare fann vi att metoden ger en fördröjning på d = int((n-1)/2), det betyder alltså att man får en pol av ordning d i origo.

Den fullständiga beräkningen av överföringsfunktionen med den ovan angivna metoden utförs i det kapitel som följer.

Den på detta sätt gjorda översättningen till diskret av vårt filter Λ^k kommer jag att beteckna Λ_{di}^k. Vi har nu skaffat oss kunskaper som gör det möjligt att göra en helt digital implementering av våra filter, vi får följande blockschema.



Alltså sker reglering och filtrering digitalt. Speciellt ska påpekas att åtminstone Λ_u^k ska beräknas synkront med regleringen, praktiskt sett sker detta även med Λ_{di}, detta krav medför att den enklaste implementeringen fås om filtrering och reglering sköts av samma dator.

6. Interpolering, allmänna fallet

Vi har tidigare sett hur det går till att inplementera ett filter med linjär interpolation. Vi ska nu se hur det går till att implementera ett filter med m:te ordningens interpolation.

Vi antar att det filter som ska översättas är representerat på någon lämplig tillståndsform

$$\begin{aligned} \frac{dx(t)}{dt} &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) \end{aligned} \tag{37}$$

På vanligt sätt kan denna överföras till

$$\begin{aligned} x(t) &= e^{A(t-t_0)}x(t_0) + \int_{t_0}^{t} e^{A(t-\nu)}u(\nu)d\nu B \\ y(t) &= Cx(t) \end{aligned} \tag{38}$$

Vi antar nu att insignalen u(t) är känd i m+1 sampelpunkter, och betecknar dess värden i dessa punkter med $f(t_k)$, $k = 0,1,\ldots,m$.

Det gäller då att det existerar ett (unikt) polynom av m som går genom dessa m+1 punkter, vi betecknar detta polynom med $p_m(t)$.

Newtons interpolationsformel ger $p_m(t)$ som

$$\begin{aligned} p_m(t) = \alpha_0 &+ \alpha_1(t-t_0) + \alpha_2(t-t_0)(t-t_1) + \ldots + \\ &+ \alpha_m(t-t_0)\ldots(t-t_{m-1}) \end{aligned} \tag{39}$$

Vi väljer $t_k = kh$, $k = 0,1,2,\ldots$, och får

$$p_m(t) = \alpha_0 + \alpha_1 t + \alpha_2 t(t-t_1) + \ldots + \alpha_m t(t-t_1)\ldots(t-t_{m-1}) \tag{40}$$

För att bestämma koefficienterna α_k, utnyttjar man att det ska gälla

$$p_m(t_k) = f(t_k), \quad k = 0,1,2,\ldots m \tag{41}$$

vi inför nu följande beteckningar

$$f^* = [f(t_m)\ f(t_{m-1})\ \ldots\ f(t_1)\ f(t_0)]^T \tag{42}$$

$$\alpha^* = [\alpha_m\ \alpha_{m-1}\ \ldots\ \alpha_1\ \alpha_0]^T \tag{43}$$

då gäller

$$f^* = \Psi\alpha^* \tag{44}$$

där Ψ fås ur ekv (40) och (41), man får

$$\Psi = \begin{bmatrix} m!h^m & m(m-1)\cdots 2h^{m-1} & \cdot & \cdot & m(m-1)h^2 & mh & 1 \\ 0 & (m-1)!h^{m-1} & \cdot & \cdot & (m-1)(m-2)h^2 & (m-1)h & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & & \cdot \\ 0 & 0 & \cdot & \cdot & 2!h^2 & 2h & 1 \\ 0 & 0 & \cdot & \cdot & 0 & h & 1 \\ 0 & 0 & \cdot & \cdot & 0 & 0 & 1 \end{bmatrix} \tag{45}$$

Vi observerar speciellt att Ψ är icke-singulär, determinanten fås enkelt eftersom matrisen är triangulär.

För de fortsatta räkningarna är det en fördel att ha $p_m(t)$ på följande form

$$p_m(t) = a_0 t^m + a_1 t^{m-1} + \ldots + a_{m-1} t + a_m \tag{46}$$

Det gäller nu enligt definitionen av Stirling-talen av 1:a sorten att

$$t(t-h)(t-2h)\cdots(t-rh+h) = \sum_{k=1}^{r} s_k^r t^k h^{r-k} \tag{47}$$

Vi får alltså

$$p_m(t) = \alpha_0 + \sum_{k=1}^{m} \alpha_k \sum_{i=1}^{k} s_i^k t^i h^{k-i} \tag{48}$$

som, genom att betrakta koefficienterna framför t_k, kan skrivas om till

$$p_m(t) = \alpha_0 + \sum_{k=1}^{m} t^k \sum_{i=k}^{m} \alpha_i s_k^i h^{i-1} \equiv a_m + \sum_{k=1}^{m} a_{m-k} t^k \tag{49}$$

Med α^* som tidigare och

$$a^* = [a_m \; a_{m-1} \cdots a_1 \; a_0]^T \tag{50}$$

får vi nu

$$a^* = \Xi \alpha^* \tag{51}$$

där

$$\Xi = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdot & \cdot & 0 & 0 \\ 0 & s_1^1 & s_1^2 h & s_1^3 h^2 & \cdot & \cdot & s_1^{m-1} h^{m-2} & s_1^m h^{m-1} \\ 0 & 0 & s_2^2 & s_2^3 h & \cdot & \cdot & s_2^{m-1} h^{m-3} & s_2^m h^{m-2} \\ 0 & 0 & 0 & s_3^3 & \cdot & \cdot & s_3^{m-1} h^{m-4} & s_3^m h^{m-3} \\ \cdot & \cdot & \cdot & \cdot & & & \cdot & \cdot \\ 0 & 0 & 0 & 0 & \cdot & \cdot & s_{m-1}^{m-1} & s_{m-1}^m h \\ 0 & 0 & 0 & 0 & \cdot & \cdot & 0 & s_m^m \end{bmatrix} \qquad (52)$$

Genom de hittills gjorda räkningarna har vi visat hur vi från de observerade värdena f^* kan finna motsvarande a^*. Vi ska nu utnyttja detta för att göra en översättning av ekv (38) till diskret tid.

För att minimera resttermen i interpolationsformeln har vi tidigare sett det intervall i vilket man utnyttjar $p_m(t)$ skall vara centrerat i förhållande till intervallet $[t_0,t_m]$. Detta uppnås genom att införa en fördröjning på int((n-1)/2) sampelintervall.

Vi sätter

$$r = m - d, \text{ där } d = \text{int}\left(\frac{m-1}{2}\right). \qquad (53)$$

Då gäller att det intervall som vi ska integrera över är [rh,rh+h].

Vi har alltså att beräkna integraler av typen

$$J_k = \int_{rh}^{rh+h} e^{A(rh+h-\nu)} \nu^k d\nu \qquad (54)$$

Genom att göra variabelbytet

$$\varsigma = rh+h-\nu \qquad (55)$$

och utnyttja att

$$\frac{d}{d\varsigma} e^{A\varsigma} = Ae^{A\varsigma} \qquad (56)$$

fås genom upprepad partialintegration

$$J_k = \int_0^h e^{A\varsigma} (rh+h-\varsigma)^k d\varsigma =$$

$$= h^k \left[(Ah)^{-1}\Omega_k + (Ah)^{-2}\Omega_{k-1} + \ldots + (Ah)^{-k}\Omega_1 + (Ah)^{-k}\Omega_0 \right] \qquad (57)$$

där

$$\Omega_i = \frac{k!}{i!} \left(e^{Ah} r^i - I(r+1)^i\right), \quad i = 1,\ldots,k \qquad (58)$$

$$\Omega_0 = k! \int_0^h e^{A\xi} d\xi \tag{59}$$

Vi låter nu

$$\Gamma_k^* = J_k B \tag{60}$$

och sätter

$$\Gamma^* = [\Gamma_m^* \; \Gamma_{m-1}^* \; \dots \; \Gamma_1^* \; \Gamma_0^*] \tag{61}$$

då gäller

$$x(rh+h) = \Phi x(rh) + \Gamma^* a^* = \Phi x(rh) + \Gamma^* \Xi \alpha^* =$$

$$= \Phi x(rh) + \Gamma^* \Xi \Psi^{-1} f^* \tag{62}$$

För att få filtrets överföringsfunktion betraktar vi kolonnerna i $\Gamma^* \Xi \Psi^{-1}$ vi inför följande beteckning för dessa

$$\Gamma^* \Xi \Psi^{-1} = [\Gamma_m \; \Gamma_{m-1} \; \dots \; \Gamma_1 \; \Gamma_0] \tag{63}$$

Då gäller att vi får följande överföringsoperator

$$H(q) = q^d C[qI-\Phi]^{-1}(\Gamma_m + q^{-1}\Gamma_{m-1} + \dots + q^{-m+1}\Gamma_1 + q^{-m}\Gamma_0). \tag{64}$$

7. Uppskattning av interpolationsfelet

Vi ska nu se hur man kan bilda sig en uppfattning om interpolationsfelets storlek i det allmänna fallet.

Som tidigare angetts har resttermen följande utseende

$$R_m(t) = \frac{f^{(m)}(\xi)}{m!} \Pi_0^m (t - t_k) \tag{65}$$

där

$$t_k = kh \ , \ k = 0,1,\ldots,m$$
$$\xi \in [t_0, t_m] \ .$$

Med hjälp av Stirlingtalen för man enklare

$$R_m(t) = \frac{f^{(m)}(\xi)}{m!} \sum_{k=1}^{m+1} s_k^{m+1} t^k h^{m+1-k} \ . \tag{66}$$

Tidigare har vi funnit att interpolationspolynomet ska integreras över ett samplingsintervall som är centrerat i förhållande till intervallet $[t_0,t_m]$. Här är vi enbart intresserade av absolutbeloppet av R_m, varför tecknet på resultatet från integrationen saknar betydelse.

Vi valde tidigare att integrera filtrets systemekvation över intervallet $[t_r,t_{r+1}]$.

Man observerar att feltermen går genom systemet på samma sätt som vilken tillåten insignal som helst, och får dess verkan på filtrets utsignal genom att lösa

$$\begin{aligned} x(rh+h) &= \Phi x(rh) + \int_{rh}^{rh+h} e^{A(rh+h-\nu)} B R_m(\nu) d\nu \\ y(rh) &= Cx(rh) \ . \end{aligned} \tag{67}$$

I denna ekvation ska vi rikta vårt intresse mot integraltermen, vi låter $|\cdot|$ beteckna elementvist absolutbelopp och får

$$\left| \int_{rh}^{rh+h} e^{A(rh+h-\nu)} B R_m(\nu) d\nu \right| \le \left| \int_{rh}^{rh+h} e^{A(rh+h-\nu)} B d\nu \right| \left| \int_{rh}^{rh+h} R_m(\nu) d\nu \right| =$$

$$= |\Gamma_0| \left| \frac{f^{(m)}(\xi)}{m!} h^{m+2} \sum_{k=1}^{m+1} s_k^{m+1} \frac{(r+1)^{k+1} - r^{k+1}}{k+1} \right| = |\Gamma_0| R_m^* \ . \tag{68}$$

För att skaffa oss en uppfattning av storleken på den sista faktorn låter vi den signal som interpoleras vara en sinus med frekvens ω.

$$f(t) = F_0 \sin \omega t \tag{69}$$

Enkelt fås

$$\text{Max } |f^{(m)}(t)| = F_0\omega^m, \tag{70}$$

som ger

$$\text{Max } R_m^* = F_0\frac{(\omega h)^m}{m!}h^2\left|\sum_{k=1}^{m+1} s_k^{m+1}\,\frac{(r+1)^{k+1} - r^{k+1}}{k+1}\right| \tag{70}$$

Om vi nu låter

$$f(t) = F_0\sin\omega t, \text{ då } 0 \le \omega \le \omega_B,\ f(t) = 0 \text{ för övrigt,} \tag{71}$$

så ger medelvärdesbildning på frekvens-axeln

$$\text{Max } \hat{R}_m^* = F_0\frac{(\omega_B h)^{m+1}}{(m+1)!}h^2\left|\sum_{k=1}^{m+1} s_k^{m+1}\,\frac{(r+1)^{k+1} - r^{k+1}}{k+1}\right| . \tag{72}$$

Man kan nu tänka sig att förfina denna analys något ty det gällde ju att

$$\xi \in [t_0, t_m], \tag{73}$$

dvs det är ej alltid möjligt att få

$$\text{Max } |f^{(n)}(\xi)| = F_0\omega^n. \tag{74}$$

Efter en stunds funderingar upptäcker man att

$$\underset{\xi\in[t_0,t_m]}{\text{Max}} \ |\sin\omega\xi| \ge \left|\sin\frac{m\omega h}{2}\right|,\ \frac{m\omega h}{2} < \frac{\pi}{2} . \tag{75}$$

Genom att medelvärdesbilda över de möjliga max-värdena fås

$$g(\omega) = \text{Max } |\sin\omega\xi|_{med} = \begin{cases} \dfrac{\sin(\frac{\pi}{2} - \frac{m\omega h}{2})}{\frac{\pi}{2} - \frac{m\omega h}{2}}, & \dfrac{m\omega h}{2} < \dfrac{\pi}{2}, \\ 1, & \dfrac{m\omega h}{2} \ge \dfrac{\pi}{2}. \end{cases} \tag{76}$$

Precis som tidigare kan vi nu medelvärdesbilda över de förekommande frekvenserna, och får

$$\text{Max } \hat{R}_{md}^* = \frac{F_0}{m!}\,\frac{1}{\omega_B}\int_0^{\omega_B}(\omega h)^m g(\omega)\,d\omega\ h^2\left|\sum_{k=1}^{m+1} s_k^{m+1}\,\frac{(r+1)^{k+1} - r^{k+1}}{k+1}\right| . \tag{77}$$

Med dessa maximala värden på interpolationsfelet kan man sedan lätt få en uppfattning av hur felet påverkar filtrets utsignal.

Man betraktar interpolationsfelet som konstant under en samplingsperiod och tar de maximala medelvärdesskattningarna ovan som insignal till filtret. Då får man

osäkerheten i den genom interpolation erhållna överföringsfunktionen uttryckt i andelar av ZOH-översättningen.

8. Identifiering

Vi har tidigare sett hur det var möjligt att göra en översättning av systemekvationen som är realiserbar. Det är nu dags att utnyttja detta för att estimera koefficienterna i överföringsfunktionen.

Metoden som används för att skatta parametrarna i överföringsfunktionen är den välkända rekursiva minstakvadrat-metoden. Metoden är modifierad på så sätt att man infört en dödzon i skattaren, dvs en zon där ingen uppdatering av de skattade parametrarna sker.

Först behandlas det fall där samtliga parametrar i IU-relationen är okända sedan visas att det är lätt att modifiera metoden, så att man även klarar av det fall då överföringsfunktionen är delvis känd.

Tidigare har redogjorts för hur det gick till att översätta relationen

$$(p^n + a_1 p^{n-1} + \ldots + a_n)y(t) = (b_1 p^{n-1} + \ldots + b_n)u(t) \tag{78}$$

till följande realiserbara relation

$$(P_n + a_1 P_{n-1} + \ldots + a_n P_0)y(t) =$$

$$= (b_1 P_{n-1} + b_2 P_{n-2} + \ldots + b_n P_0)u(t) \tag{79}$$

där

$$P_k = (\Lambda\tau)^{n-k}(1-\Lambda)^k \quad , \quad k = 0,1,\ldots,n$$

$$\Lambda = \frac{1}{1 + p\tau}$$

Vi inför nu följande beteckningar

$$\varphi = [P_{n-1}y \ \ldots \ P_0 y \ P_{n-1}u \ \ldots \ P_0 u]^T \tag{80}$$

$$\theta = [-a_1 \ \ldots \ -a_n \ b_1 \ \ldots \ b_n]^T \tag{81}$$

då gäller

$$P_n y = \varphi^T \theta \tag{82}$$

och vi har nu den relation vi behöver för att använda rekursiva MK-metoden för att skatta de okända parametrarna θ.

Den rekursiva uppdateringen av de skattade parametrarna och kovariansmatrisen sker enligt följande ekvationer

$$\varepsilon(k) = P_n y(k) - \hat{\theta}^T(k-1)\varphi(k) \tag{83}$$

$$P(k) = \frac{1}{\lambda}\left[P(k-1) - \frac{P(k-1)\varphi(k)\varphi^T(k)P(k-1)}{\lambda + \varphi^T(k)P(k-1)\varphi(k)}\right] \tag{84}$$

$$\hat{\theta}(k) = \hat{\theta}(k-1) + P(k)\varphi(k)\varepsilon(k) \tag{85}$$

Det är värt att varna för förväxlingar av polynomen P_k och kovariansmatrisen P.

Vi ska nu se hur man enkelt kan generera P_k då Λ^k, k = 0,1,...,n, är kända

Vi inför beteckningarna

$$P_n^* = [P_n \; P_{n-1} \; \cdots \; P_0]^T \tag{86}$$

$$\Lambda_n^* = [1 \; \Lambda \; \ldots \; \Lambda^n]^T \tag{87}$$

då gäller

$$P_n^* = M_n \Lambda_n^* \tag{88}$$

där rader och kolonner i M_n är numrerade 0,1,...,n.

Det gäller då att

$$P_k = (\text{rad}_{n-k} \; M_n)\Lambda_n^* \;, \; k = 0,1,\ldots,n \tag{89}$$

Det är nu möjligt att översätta skattningekvationen i termer av de nyss gjorda definitionerna, vi inför beteckningarna

$$P_{nr}^* = [P_{n-1} \; P_n \; \cdots \; P_0]^T \tag{90}$$

$$\Lambda_{nr}^* = [\Lambda \; \Lambda^2 \; \ldots \; \Lambda^n] \tag{91}$$

samt M_{nr} för den matris man får om man från M_n tar bort rad 0 och kolonn 0. Vi sätter även

$$a^* = [a_1 \; a_2 \; \cdots \; a_n]^T \tag{92}$$

$$b^* = [b_1 \; b_2 \; \cdots \; b_n]^T \tag{93}$$

Man får nu följande uttryck för skattningsekvationen

$$P_n y = -a^{*T} M_{nr} \Lambda^*_{nr} y + b^{*T} M_{nr} \Lambda^*_{nr} u \tag{94}$$

Vi konstaterar att

$$\varphi = [M_{nr} \Lambda^*_{nr} y \; M_{nr} \Lambda^*_{nr} u]^T \tag{95}$$

$$\theta = [-a^* \; b^*]^T \tag{96}$$

Tolkningen till detta är att man till varje element i φ knyter en rad i M_{nr}.

Vi ska nu generalisera resulatateн ovan till det fall då IU-relationen är delvis känd. Man konstaterar att om något a_i elller b_j är känt så ska faktorn

$$a_i P_{n-i} y \quad \text{eller} \quad b_j P_{n-j} u$$

flyttas till vänstersidan av ekv (94).

För att beskriva hur detta kan implementeras fixerar vi n, detta för att förenkla beteckningarna. Ekv (94) skrivs nu på följande sätt

$$P_n y = -a^{*T}_h M_{yh} \Lambda^*_{nr} y + b^{*T}_h M_{uh} \Lambda^*_{nr} u \tag{97}$$

där index h indikerar att termen står på höger sida i ekvationen.

Antag nu att parametern a_i är känd, då kan vi skriva

$$a^{*T}_v M_{yv} \Lambda^*_n y = -a^{*T}_h M_{yh} \Lambda^*_{nr} y + b^{*T}_h M_{uh} \Lambda^*_{nr} u \tag{98}$$

där

$$a^*_v = [1 \; a_i]^T \tag{99}$$

$$a^*_h = [a_1 \; \ldots \; a_n]^T \text{ , elementet } a_i \text{ är eleminerat} \tag{100}$$

$$b^*_h = [b_1 \; \ldots \; b_n]^T \tag{101}$$

$$M_{yv} = \begin{bmatrix} \text{rad}_0 \; M_n \\ \text{rad}_{n-i} \; M_n \end{bmatrix} \tag{102}$$

och M_{yh} är den matris man får när man eleminerar rad n-i från M_{nr}. M_{uh} är samma som M_{nr} ovan.

Antag nu att dessutom b_j är känd, vi kan då skriva

$$a^{*T}_v M_{yv} \Lambda^*_n y - b^{*T}_v M_{uv} \Lambda^*_{nr} u = -a^{*T}_h M_{yh} \Lambda^*_{nr} y + b^{*T}_h M_{uh} \Lambda^*_{nr} u \tag{103}$$

där

$$b_v^* = [b_j] \tag{104}$$

$$b_h^* = [b_1 \ \ldots \ b_n]^T \tag{105}$$

$$M_{uv} = [\text{rad}_{n-j} \ M_{nr}] \tag{106}$$

och M_{uh} är den matris som fås när rad n-j elemineras från M_{nr}.

Det är nu lätt att generalisera de ovan gjorda beräkningarna till det allmänna fallet, vi får

$$a_v^* = [1 \text{ kända a-parametrar}]^T \tag{107}$$

$$a_h^* = [\text{okända a-parametrar}]^T \tag{108}$$

$$b_v^* = [\text{kända b-parametrar}]^T \tag{109}$$

$$b_h^* = [\text{okända b-parametrar}]^T \tag{110}$$

där elementen i dessa vektorer är ordnade efter stigande index. För de fyra matriserna gäller

$$M_{yv} = \begin{bmatrix} \text{rad}_0 \ M_n \\ \text{rader}_{\text{kända a-par.}} \ M_n \end{bmatrix} \tag{111}$$

$$M_{uv} = [\ \text{rader}_{\text{kända b-par.}} \ M_{nr}\ ] \tag{112}$$

$$M_{yh} = [\ \text{rader}_{\text{okända a-par.}} \ M_{nr}\ ] \tag{113}$$

$$M_{uh} = [\ \text{rader}_{\text{okända b-par.}} \ M_{nr}\ ] \tag{114}$$

där raderna tas ur M_n respektive M_{nr} enligt samnma självklara regel som vi använde oss av i fallet med a_i och b_j kända ovan.

Vi är nu klara att ge den generella skattningsekvationen, med vars hjälp vi kan skatta de okända parametrarna i överföringsfunktionen.

Inför nu följande beteckningar

$$\varsigma = a_v^{*T} M_{yv} \Lambda_n^* y - b_v^{*T} M_{uv} \Lambda_{nr}^* u \tag{115}$$

$$\varphi = [M_{yh}\Lambda^*_{nr}y \; M_{uh}\Lambda^*_{nr}u]^T \tag{116}$$

$$\theta = [-a^*_h \; b^*_h]^T \tag{117}$$

då gäller

$$\zeta = \theta^T\varphi \tag{118}$$

Den rekursiva uppdateringen av skattningarna sker enligt

$$\varepsilon(k) = \zeta(k) - \theta^T(k-1)\varphi(k) \tag{119}$$

$$P(k) = \frac{1}{\lambda}\left[P(k-1) - \frac{P(k-1)\varphi(k)\varphi^T(k)P(k-1)}{\lambda + \varphi^T(k)P(k-1)\varphi(k)}\right] \tag{120}$$

$$\hat{\theta}(k) = \hat{\theta}(k-1) + P(k)\varphi(k)\varepsilon(k) \tag{121}$$

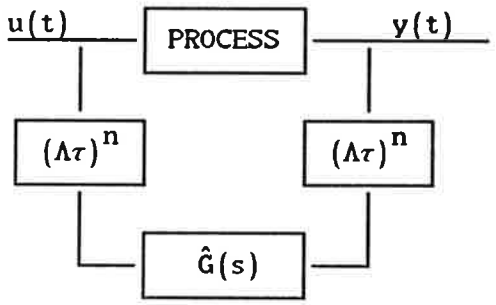
I och med dessa ekvationer har vi alltså allt vi behöver för att implementera den rekursiva skattningen på ett sådant sätt att man enbart skattar de okända parametrarna i överföringsfunktionen.

Att man infört en dödzon i parameterskattaren innebär att ingen uppdatering av P-matrisen sker om

$$|\varepsilon(k)| > \varepsilon_{min} \; .$$

9. Angående valet av filterparametern .

Nedan ska vi göra några iakttagelser angående hur valet av filterparametern τ påverkar skattningarna. Vi utgår från det tidigare använda blockschemat.



Man ser ur detta att modellen exciteras olika mycket beroende på hur man väljer τ. Detta leder till följande iaktagelser:

(i) Om $1/\tau$ är för litet, dvs τ för stort, så kommer det att från parameterskattaren att se ut som om systemet ej vore tillräckligt exciterat. Man kommer här alltså ej att excitera modellen på högfrekvensassymptoten. Detta kan leda till dålig konvergens hos de skattade parametrarna.

(ii) Största värdet på $1/\tau$ begränsas i första hand av samplingsfrekvensen, men det är ju önskvärt att få in så lite brus som möjligt i systemet varför man lämpligen väljer $1/\tau \approx \omega_B$ hos den process som ska identifieras. Dock är ju brytfrekvensen hos identifieringsobjektet okänd. Därför är det behändigt att enkelt kunna variera τ.

Valet av parameterskattarens sampelfrekvens påverkar även den konvergenshastigheten hos skattningarna, för stor estimeringsperiod gör att systemet verkar dåligt exciterat. Om estimeringsperioden däremot är liten innehåller konsekutiva värden ingen ny information.

10. Eliminering av vikningsfiltrets inverkan

Vi ska i det följande se hur man kan eliminera ett eventuellt vikningsfilters inverkan på den skattade modellen av systemet.

Antag att vi har ett system med följande systemekvation

$$A(s)Y(s) = B(s)U(s) \quad , \; G(s) = B(s)/A(s) \tag{122}$$

där

$$A(s) = s^n + a_1 s^{n-1} + \ldots + a_n \tag{123}$$

$$B(s) = b_1 s^{n-1} + b_2 s^{n-2} + \ldots + b_n \tag{124}$$

Antag nu att vi i stället för att mäta y(t) mäter $L^{-1}\{G_{vf}(s)Y(s)\}$, då gäller

$$A(s)A_{vf}(s)Y(s) = B(s)B_{vf}(s)U(s) \tag{125}$$

med

$$G_{vf}(s) = B_{vf}(s)/A_{vf}(s).$$

Om vi nu tolkar G_{vf} ovan som ett vikningsfilter så framgår den inverkan som ett sådant har på den modell av systemet som vi har för avsikt att skatta.

Vår estimeringsalgoritm har ingen möjlighet att skilja poler och nollställen hos den process vi vill identifiera från de som hör till vikningsfiltret. Detta får till följd att det blir svårare att identifiera processen. Antivikningsfiltret bidrar till att öka komplexiteten exempelvis så ökas ordningen på systemet som vi vill identifiera vidare har man ingen garanti för att signalen exciterar moderna i vikningsfiltret. Det senare får till följd att skattningarnas konvergens försämras.

Det finns dock en enkel metod som eliminerar vikningsfiltrets inverkan. Tidigare när vi definierade den lågpassfiltrerade modellen av ett system, så framgick att denna kunde tolkas som att man lågpassfiltrerade in- och utsignal till från den process man ville identifiera. Detta märktes dock ej i överföringsfunktionen, som man bildade med hjälp av dessa filtrerade signaler. Filterpolerna bildade nämligen ett icke-observerbart par av poler och nollställen.

Kontentan av detta resonemang är att om vi låter in- och utsignalen passera exakt lika filter innan vi mäter dessa, så kommer detta filter ej att var observerbart i den överföringsfunktion som ska skattas. För att eliminera vikningsfiltrets inverkan har vi alltså att se till att processens insignal passerar ett filter med samma egenskaper som vikningsfiltret.

Vi har i denna rapport byggt mycket av resonemangen på att den process som ska identifieras styrs av en digital regulator. Detta faktum ska vi nu åter utnyttja.

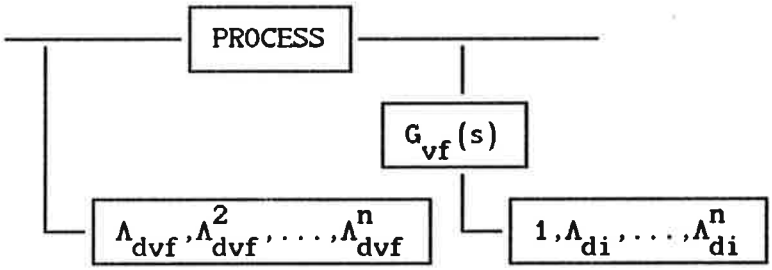
Tidigare gjorde vi översättningen från kontinuerlig tid till diskret tid av $\Lambda^k(s)$, vi ska nu se att om vi istället gör denna översättning av $\Lambda^k(s)G_{vf}(s)$ så har vikningsfiltrets inverkan eliminerats. Vi gör alltså följande modifiering i blockschemat

$$\Lambda_d^k(z) = (1 - z^{-1})Z\{ L^{-1}\{ \Lambda^k(s)/s \} \} \tag{126}$$

byts mot

$$\Lambda_{dvf}^k(z) = (1 - z^{-1})Z\{ L^{-1}\{ \Lambda^k(s)G_{vf}(s)/s \} \} \tag{127}$$

Det gäller nu som vanligt att detta filter i sampeltidpunkterna ger exakt den utsignal man skulle fått om signalen gått genom det kontinuerliga filtret $\Lambda^k(s)G_{vf}(s)$.



11. Tidsvariabel samplingsfrevkvens.

Vi ska nu se hur man kan modifiera den angivna metoden till att klara av identifiering av system där man inte använder sig av konstant samplingsperiod.

Tidigare gjorde vi antagandet att den process som ska identifieras styrs av en digital regulator. Nu när vi har resultatet i fallet med allmän interpolation, så inser vi att detta inte är nödvändigt. Det går lika bra att göra samma filteröversättning för identifieringsobjektets insignal som för dess utsignal.

Vi betecknar nu den analoga signalen med f(t), och antar att denna samplas i tidpunkterna

$$t = t_0, t_1, t_2, \ldots,$$

och inför

$$\Delta t_k = t_k - t_{k-1} \quad , \; k = 1,2,3,\ldots \tag{128}$$

Man måste kräva att max Δt är så liten att man har tillräckligt med information om f(t). k

Antag nu att vi på samma sätt som gjorts tidigare låter ett interpolationspolynom glida utmed f(t). Det är nu enkelt att interpolera f(t) så att vi får dess värden i ekvidistanta tidpunkter

$$t = t_{i0}, t_{i1}, t_{i2}, \ldots.$$

På dessa interpolerade värden av kan vi nu utnyttja den metod som redogjorts för i tidigare kapitel.

Den konstanta samplingsperioden

$$h = t_{ik} - t_{ik-1} \quad , \tag{129}$$

får man nu dimensionera utifrån värdet av max Δt_k och de dimensioneringsregler som man använder i fallet med konstant samplingsfrekvens.

Man frågar sig hur man ska välja det interpolationspolynom som ger f(t) i ekvidistanta tidpunkter. Vi betecknar detta polynom med p_m.

Vi har tidigare sett att interpolationspolynomets felterm har minimum då man väljer den tidpunkt som man interpolerar f(t) i som

$$t_w = \frac{t_m + t_0}{2} \quad . \tag{130}$$

Detta önskvärda val är tyvärr ej möjligt att uppnå för alla Δt_k. Det är då möjligt att arbeta med interpolationspolynom av varierande ordning m_k.

Lämpligt sätt att välja polynomets ordning är t ex att använda sig av ett kriterium av typen

$$rh \approx \sum_{k=k_0+1}^{k_0+m} \Delta t_k \ , \tag{131}$$

där det värde man är intresserad av

$$p_m(\frac{rh}{2}) \tag{132}$$

Detta betyder att man observerar signalen i ett intervall vars längd man försöker hålla konstant.

Om det skulle vara så att detta kriterium leder till ett orimligt högt värde på ordninge m, så kan man välja att kasta bort invärden där dessa är som tätast. Det vill säga man inför ett kriterium till, nämligen följande

$$m \leq m_{max} \tag{133}$$

12. Andra typer av filtrering

I den hittils förda diskussionen har jag använt lågpassfilter av typen

$$\Lambda^k(s) = \frac{1}{(1+s\tau)^k} \quad . \tag{134}$$

Anledningen till detta är först och främst att den mjukvaru-implementering som gjorts, bygger på denna typ av filter. Det visar sig också vara lätt att ange slutna uttryck för koefficienterna i de diskreta överföringsfunktionerna. Vidare var det så att den teori som var upprinnelsen till detta arbete byggde helt på dessa filter, se referens [1].

Min åsikt är att det egentligen inte finns någon anledning att inskränka sig till denna typ.

Den tolkning man kan ge dessa lågpassfilter är att de begränsar signalernas bandbredd, att man i praktiken ser till att ingen energi finns i signalen över någon viss frekvens. Detta medför ju även att energin i derivatorna av signalerna är liten.

En egenskap hos den gjorda systembeskrivningen som är värd att notera är att derivatorna är begränsade för steg och högre ordningens polynom i signalerna, detta följer enkelt ur begynnelsevärdes-satsen för Laplace-transformen.

$$\lim_{s\to\infty} s\,\frac{s^k}{(1+s\tau)^n}\cdot 1 = \begin{cases} 0 & , \quad k < n-1 \\ 1/\tau^n & , \quad k = n-1 \\ \infty & , \quad k > n-1 \end{cases} \tag{135}$$

Detta är en egenskap som är viktig ty steg i insgnalen är vanligt förekommande, processen styrs ju av en digital regulator med ZOH.

Det skulle alltså vara frestande att använda någon annan typ av n:te ordningens lågpassfilter för att få mindre signalförlust i passbandet. Ett n:te ordningens Butterworth har egenskapen att det av alla n:te ordningens lågpassfilter är det som ger minst ripple i passbandet.

Med ett sådant val av filter så får man genomföra översättningen från analogt till digitalt på ett lite annorlunda sätt, detta är ej speciellt mycket mera komplicerat.

Det finns även en intressant möjlighet att göra filtren adaptiva, ty det gäller ju att skattningarna blir bäst om $1/\tau$ är något större än ω_B hos den process man ska identifiera. Det nyss sagda följer ur kravet på att signalerna ska vara tillräckligt intressanta (persistently exciting inputs). För varje modellansats är det ju möjligt, att ur denna modell, beräkna ω_B. Det förefaller dock nödvändigt att att denna modifiering ej sker förrän skattningarna "stabiliserat" sig, man måste åtminstone låta adaptionen ske med en lägre hastighet än skattningen.

Ett vanligt förekommande fall är att den linjära modellen uppkommit sedan man linjäriserat kring någon arbetspunkt (u_0,y_0), om det nu gäller att $(u_0,y_0) \neq (0,0)$ så kommer den angivna metoden att ge en bias i skattningarna. För att undvika

detta skulle det vara önskvärt om man kunde eleminera frekvensen $\omega = 0$ från signalernas spektra. Ett sådant filter är tyvärr ej realiserbart, men ett brant högpassfilter skulle kunna tjäna som en approximation av ett sådant. Det enklaste man kan göra i ett sådant fall är att subtrahera arbetspunkten från de signaler man mäter och sedan filtrera de på så sätt uppkomna signalerna Detta är naturligvis en acceptabel lösning om arbetspunkten ej ändras. Om denna ändras skulle man vilja ha ett annat sätt. Om man förutsätter att arbetspunkten ändras "sakta" så skulle en tänkbar lösning vara att använda ett glidande medelvärde som sedan subtraheras från de aktuella signalerna.

Ett problem, när man gör skattningen i kontinuerlig, tid är att storleken på elementen i φ-vektorn beror på frekvensen, ju högre frekvens man tillåter desto större blir skillnaden i storleksordning mellan elementen. Detta kan ge upphov till numeriska problem. För att minska denna effekt något kan man använda Biermann's implementering av den rekursiva MK-skattningen.

Man kan även tänka sig att arbeta med en filterbank bestående av bandpassfilter, där varje filter tar hand om sin del av spektrat. Genom att sedan utföra estimering med varje filters utsignal kan man välja den estimeringsperiod, som är optimal för respektive filter.

13. Implementering

I det följande kommer den implementering som gjorts på IBM-AT att beskrivas. En kort beskrivning av de realtidsprimitiver samt grafikrutiner som använts ges i de sammanhang där dessa dyker upp.

Programmet består av fyra processer, av vilka en sköter all operatörs--kommunikation.

En mus används av operatören för att göra val av menyer samt val av någon typ av data som ska läsas från tangentbordet. Tangentbordet används alltså enbart för att läsa in tal.

Operatörskommunikationen är gjord på så sätt att de operationer som kan utföras är uppdelade i ett antal menyer. I varje meny kan man sedan arbeta med de data som är relevanta för den del av programmet som den aktuella menyn avser. Till menyerna räknar jag här även plottningen av de skattade parametrarnas historia samt visningen av P-matrisens samtliga element. I dessa finns dock inga data att arbeta med. En utförlig beskrivning av de olika menyerna lämnas nedan.

De fyra processerna kan delas in i två grupper. En som sköter all kommunikation med operatören medan de tre övriga processerna som sköter identifiering och reglering. Till den sista gruppen räknas även den process som sköter synkroniseringen. Man kan om man vill även betrakta identifieringen som en meny, i vilken det bara existerar ett val, nämligen att avsluta identifiering och reglering. De tre processerna som sköter identifieringen befinner sig i "stand-by", tills operatören ger kommandot att en identifiering ska utföras.

Menyerna är upplagda i en trädstruktur, där man med musen tar sig fram till grenarna. För att slippa ett antal onödiga mus-tryckningar är implementeringen gjord så att man från varje gren kan nå huvudmenyn samt stoppa programmet. Det är via denna huvudmeny, som alla övriga menyer nås.

14. Beskrivning av processer

Nedan följer en kortfattad beskrivning av de i programmet förekommande processerna.

För att maximalt utnyttja datorns kapacitet används fyra processer, dessa är, uppräknade efter prioritet:

(i) Klockprocessen (Clock), som ser till att samplingen sker med konstant frekvens, dvs den synkroniserar hela programmet.

(ii) Reglerprocessen (Regulate), denna process sköter filtrering och reglering. Processen håller även reda på när det är dags att estimera.

(iii) Estimeringsprocessen (Estimate), som utför den rekursiva minsta-kvadrat--skattningen.

(iv) Operatörskommunikationen (Main process), som svarar för all betjäning av användaren.

Gemensamma resurser skyddas med semaforer, men det finns undantag. Eftersom alla processer utom MainProcess befinner sig i "stand-by" medan användaren kommunicerar med programmet, så behöver ej de variabler som kan ändras av denna skyddas. Vidare gäller att man ej behöver skydda datan mot operationer som ej modifierar denna, dvs mot läsning.

En anledning till att operatörskommunikationen har lägst prioritet är att denna utnyttjar tangentbordet och musen, de rutiner som läser från dessa använder sig av "busy-wait". Det är alltså nödvändigt att denna process har lägst prioritet för att man ska ta så lite tid som möjligt från de processer som har höga tidskrav.

Det är nu dags att lite närmare precisera vad de olika processerna gör.

Clock.

Det enda denna process gör är att den väntar en viss tid svarande mot sampeltiden, när denna tid har gått så signalerar Clock till regulate att denna tid har gått. När detta är gjort lägger sig åter Clock och väntar, och släpper på så vis in de andra processerna.

Regulate.

Även om namnet på denna process antyder att den sköter regleringen så är dess huvuduppgift att varje sampelperiod filtrera in- och utsignalerna till/från det objekt som ska identifieras.

Detta sker så att när Clock har signalerat att en sampelperiod har gått så utförs i tur och ordning följande:

(1) Sampling av identifieringsobjektets utsignal samt reglering av detsamma.

(2) In- respektive utsignal till respektive från identifieringsobjektet filtreras.

(3) Testar om det är dags för estimering, om så är fallet skickas φ-vektorn till Estimate.

(4) Testar om det maximala antal sampel som identifieringen kan pågå har förflutit. Om detta är fallet så stoppas identifieringen.

(5) Testar om användaren har uttryckt önskan om att identifieringen ska upphöra. Om så har skett betjänas hans önskan.

När identifieringen ska stoppas så sker detta genom att Regulate sätter Clock i "stand-by". När detta är gjort så kommer Regulate och Estimate att köra färdigt det som återstår att göra, sedan stannar även dessa. Först därefter släpps operatörskommunikationen in.

Estimate.

Som framgår av namnet sköter denna process estimeringen. Den estimeringsalgoritm som används är en rekursiv minsta-kvadrat-skattning, som är modifierad genom att man infört en dödzon i densamma.

Av de tre processerna som är aktiva under estimeringen är Estimate den som har lägst prioritet. Detta för att man bättre ska kunna utnyttja datorns kapacitet, genom att processen kommer in när de andra, Clock och Regulate, ej har något att göra.

Estimate arbetar på följande sätt:

(1) När Regulate signalerar att det är dags att estimera, att det finns en ny φ-vektor tillgänglig, så utförs en estimering.

(2) När estimeringen är gjord, så kontrolleras om de skattade parametrarna ska sparas, för att sedan kunna plottas. Hur ofta de skattade parametrarna ska sparas ges av det av användaren specifierade värdet på PlotCount.

Normalt befinner sig alltså Estimate i ett väntetillstånd.

Operatörskommunikationen (Main process).

Denna process svarar alltså för kommunikationen mot användaren. Den består av ett antal menyer, som man enkelt väljer bland med musen. Menyerna beskrivs på andra ställen i rapporten, så för mer information om processen hänvisas till dessa.

15. Beskrivning av menyer.

Nedan följer en beskrivning av de i programmet förekommande menyerna.

Huvudmenyn (MAIN).

Vid uppstart av programmet hamnar man i denna meny. Härifrån är det sedan möjligt att nå de övriga delarna av operatörskommunikationen.

De parametrar som kan editeras här är:

(i) Ordningen (n) på den modell som man antar beskriver systemet som ska identifieras. Denna parameter tillåts anta värden 1-4.

(ii) Samplingsperioden (h), lägsta tillgängliga är 0.01s.

(iii) Filtrens brytfrekvens r: Denna parameter är placerad här av anledningen att filtren för insignal respektive utsignal från identifieringsobjektet ska ha samma värde på parametern.

(iv) Antalet sampel mellan de tidpunkter som estimeringen genomförs. Som nämnts tidigare kan denna parameter väljas relativt fritt, enda kravet är att den ska vara ett naturligt tal.

När man valt ordningen på systemet så visas även överföringsfunktionens koefficienter. Dessa är från början okända. Om någon av dem är känd så kan man genom att trycka med musen göra det möjligt att betrakta koefficienten som känd och ange dess värde.

De menyer som kan nås härifrån är: STOP, FILTER, DISPLAY, ESTIMATE och RUN.

Filtermenyerna (FILTER).

Detta är två menyer: INPUT och OUTPUT. Vilket naturligtvis syftar på in- respektive utsignal från den process som ska identifieras. När man gjort valet FILTER i MAIN så hamnar man i INPUT.

De parametrar som kan editeras här är koefficienterna i respektive filter. Antalet filter för respektive signal är lika med den ordning man har ansatt för identifieringsobjektet.

Programmet räknar om man vill ut dessa koefficienter, användaren kan även ange egna om så är önskvärt. Alla filterberäkningar som görs sker med det i MAIN angivna värdet på r. Det användaren har att göra är att specifiera den typ av filterimplementering som önskas.

De tillgängliga implementeringarna är:

(i) Insignalen till filtret konstant mellan samplen. Detta är det fall som uppkommer i reglertekniska fall, när man samplar ett tidskontinuerligt system. Alternativet är avsett att användas för insignalen till identifieringsobjektet då detta styrs av en

digital regulator implementerad i samma dator som filtren.

(ii) Tustin approximation: Detta är den modifierade Tustin som ger att detta filter ger samma utsignal som ett motsvarande kontinuerligt skulle ha gjort vid frekvensen $1/\tau$.

(iii) Insignalen till filtret linjär mellan samplen. Denna approximation fås genom att integrera den kontinuerliga systemekvationen med den insignal man får geneom att interpolera linjärt mellan två konsekutiva sampelvärden.

(iv) Användarens möjlighet att modifiera koefficienterna.

De menyer som kan nås härifrån är: STOP, MAIN, (INPUT), OUTPUT. Där INPUT kan nås från OUTPUT.

Estimeringsmenyn (ESTIMATE).

I denna meny editerar man det, som har direkt med estimeringen att göra.

De parametrar som kan editeras här är:

(i) Diagonalelementen i P-matrisen: Samtliga diagonalelement får det specifierade värdet.

(ii) Startvärden för de okända koefficienterna i överförningsfunktionen. Kända koefficienter kan ej tilldelas värden här. Detta sker i MAIN.

(iii) Glömskefaktorn λ i den rekursiva MK-skattningen som utnyttjas. Denna parameter ska tilldelas värden i intervallet $]0,1]$.

(iv) Dödzonsparametern ϵ (MinEps). Om de nya insignalerna tillsammans med de gamla koefficientestimaten ger ett vänsterled i skattningsekvationen som ligger närmare det nya vänsterledet än ϵ, så sker ingen ny skattning.

Här visas även de numeriska värdena på de skattade parametrarna i den senaste genomförda skattningen.

De menyer som kan nås härifrån är: STOP, MAIN, P-MATRIX.

Displaymenyn (DISPLAY).

Här finns det som har med plottningen av de skattade parametrarnas historia att göra.

De parametrar som kan editeras här är y-axelns största respektive minsta tillåtna värde. Detta kan göras för att möjliggöra förstoring av graferna i höjdled. Denna finess gör det också möjligt att zooma in en skattning inom ett visst intervall.

Skattade koefficienter som ska plottas markeras genom musnedtryckning på aktuell parameter.

När man är klar med editeringen och vill göra en plottning väljer man GRAPH.

De menyer som kan nås härifrån är: STOP, MAIN, GRAPH.

Identifiering (RUN).

Detta är ingen egentlig meny, men kan ändå ses som en sådan i någon generaliserad bemärkelse.

Ett val av denna meny initierar en identifikations-körning, som genomförs med de data som angetts innan man gjorde detta val. I den implementering som beskrivs här regleras den process som ska identifieras av datorn under den tid, som identifieringen pågår. Detta betyder, att regleringen slutar när identifieringen upphör.

Identifieringen kan sluta av två anledningar, antingen att man har samlat tillräckligt med data för att plotta skärmen full eller att användaren själv stannar denna. En pågående identifiering stoppas genom en musnedtryckning, oberoende av musens placering på skärmen.

När man gjort musnedtryckningen är man tillbaka i MAIN.

P-matrismenyn (P-MATRIX).

Denna meny nås från ESTIMATE. Här visas hela P-matrisen.

De menyer som kan nås härifrån är: STOP, MAIN, ESTIMATE.

Grafmenyn (GRAPH).

Denna meny nås från DISPLAY.

Här visas de skattade parametrar som användaren valt att plotta. Olika parametrar plottas antingen med olika färg eller med olika typ av linjer. Genom att utnyttja de numeriska värden som anges i ESTIMATE, får man en uppfattning om de värden man i DISPLAY ska ange för y-axelns utsträckning.

Efter ett fåtal itereringar mellan DISPLAY och GRAPH kan man få fram den graf som önskas.

De menyer som kan nås härifrån är: STOP, MAIN, DISPLAY.

16. Utvärdering

En metod för att skatta parametrarna i tidskontinuerliga system har presenterats. En implementering av metoden har gjorts på IBM-AT.

I inledningsskedet av arbetet uppstod problemet med att översätta de kontinuerliga överföringsfunktionerna till diskreta. En enkel analys av problemet visade att det var av största vikt att göra denna översättning så exakt som möjligt.

De kända approximationerna som vanligtvis utnyttjas för att göra sådana översättningar har den nackdelen att de ger relativt stort fel, om man inte väljer hög samplingsfrekvens. Den tillämpning som metoden var avsedd att testas mot krävde det mesta av vad IBM-AT hade att erbjuda i termer av samplingsfrekvens, vilket gör det omöjligt att använda den höga samplingsfrekvens som man skulle behöva för att använda de ovan nämnda approximationerna.

En annan anledning till att de vanliga approximationerna är olämpliga är att metoden, som vi tidigare sett, bygger på att man använder en speciell typ av lågpassfilter. Dessa filter har den numeriskt otrevliga egenskapen att ha samtliga poler samlade i en punkt, i s-planet. För att minimera de numeriska problemen är det därför önskvärt att göra en så noggrann översättning som möjligt.

Dessa problem gjorde att jag försökte hitta andra sätt att göra denna översättning. Resultatet av detta sökande blev den i rapporten beskrivna interpolationsmetoden. Resttermsbetraktelser av denna metod visar, att man kan använda relativt låg samplingsfrekvens med denna metod. För de flesta tillämpningar torde man inte behöva högre gradtal än tre på interpolationspolynomet.

Den implementering som gjorts klarar likväl helt okända system som delvis kända system. Fördelarna med metoden visar sig främst vid identifiering av delvis kända system, där man kan minska antalet parametrar, som behöver skattas.

Implementeringen är helt generell, i den meningen att programmet kan anslutas till ett godtyckligt identifieringsobjekt, sånär som på den regulator som används för att styra den process som programmet testades mot.

Programmet räknar själv ut filterkoefficienter, detta gör det mycket enkelt att ändra filterparametern τ. Detta var önskvärt, eftersom denna parameter i högsta grad påverkar konvergensen hos skattningarna. För att kunna implementera högre grad av interpolationsöversättning är man dock tvungen att frångå detta och beräkna filterkoefficienterna på annat sätt.

Eftersom programmet skulle klara höga tidskrav, så lät jag plottningen av de skattade parametrarnas historia ske efter det att identifieringen var gjord. Med lägre tidskrav eller med kraftigare data-resurser hade man kunnat låta denna plottning ske efterhand som nya skattningar blev tillgängliga.

Programmet testades mot analogi-maskin och befanns då fungera tillfredsställande. I de testningarna lät jag en fyrkant-signal excitera identifieringsobjektet, dvs programmet användes enbart för estimering.

Efter detta implementerades den regulator, som används för att styra bommen. Även i detta fall fick man konvergens på de skattade parametrarna. Dock är inte

den fysikaliska tolkningen av dessa parametrar trivial. Anledningarna till detta är flera:

(i) Små utslag: Vinkelutslaget på motoraxeln är litet, sällan mer än 5-10°. Detta gör att man inte fullt utnyttjar upplösningen hos AD-omvandlaren.

(ii) Störningar: Mycket otrevliga saker kan hända om t ex nätfrekvensen viks in i det signalspektra som vi försöker anpassa skattningarna till.

(iii) Kulans vikt: Det visar sig att överföringsfunktionen mellan motorspänning och axelvinkel är mycket känslig för kulans storlek.

(iv) Fel i referensvärdet: Överföringen mellan motor och motoraxel är ej fast. Detta leder till att det är svårt att få kulan exakt över motoraxeln, följden blir att man får bias i skattningarna.

17. Referenser.

Johansson, R. Identification of continous time dynamic systems,Lund institute of technology, Lund, Sweden.

Beskriver den i kap. 3 beskrivna metoden , i vilken man betraktar systemet i Λ-planet, innehåller simuleringar med kontinuerligt realiserade filter.

Åström, J och Wittenmark, B. Computer controlled systems. Prentice-Hall, Englewood Cliffs, USA.

Kapitel 3 och 4 innehåller den utnyttjade härledningen för ZOH-översättningen mellan kontinuerligt och diskret system. Utmärkt lärobok.

För närmare beskrivning av Newtons interpolations-formel,se t ex

Ekman, T. Numeriska metoder på dator och dosa, Sigma-Tryck, Lunds Tekniska Högskola, Lund.

För beskrivning av metoden att bilda realiserbara differentialoperatorer genom att filtrera de aktuella signalerna, se

Young, P.C. Process parameter estimation and self-adaptive adaptive control, Proc. 2nd IFAC Symp. -The theory of self-adaptive control systems, Plenum Press, New York.

App.1. Bestämning av filtrens överföringsfunktioner.

I det följande ska vi ta fram explicita uttryck för överföringsfunktionerna till filtren med hjälp av de tidigare angivna metoderna.

Innan vi gör detta ska vi först se lite på några egenskaper hos Z-transformen.

Z-transformen F(z) av tidsfunktionen f(t), $t \geq 0$, definieras av

$$F(z) = \sum_{k=0}^{\infty} f(kh) z^{-k} \tag{136}$$

En egenskap hos Z-transformen som vi ska utnyttja i det följande är:

Multiplikation med t.

Låt

$$f^*(t) = tf(t) , \qquad t \geq 0. \tag{137}$$

Då får vi

$$F^*(z) = \sum f^*(kh) z^{-k} = \sum f(kh) khz^{-k} = -zh \sum f(kh) kz^{-k-1} =$$

$$= -zh \frac{\partial}{\partial z} \sum f(kh) z^{-k} = -zh \frac{\partial F(z)}{\partial z} \tag{138}$$

där summationsindex är givna av definitionen ovan.

Med hjälp av denna regel kan vi härleda de transformpar som följer.

(1). $f(t) = 1, t \geq 0$

Man utnyttjar kända resultat för geometriska serier och får

$$F(z) = \sum z^{-k} = \frac{1}{1 - z^{-1}} = \frac{z}{z - 1} \tag{139}$$

(2). $f(t) = t, t \geq 0.$

Tillämpning av derivationsregeln på resultatet i (1) ger

$$F(z) = -zh \frac{\partial}{\partial z} \frac{z}{z - 1} = \frac{hz}{(z - 1)^2} \tag{140}$$

(3). $f(t) = e^{-at}, t \geq 0$

$$F(z) = \sum e^{-akh} z^{-k} = \frac{1}{1 - e^{-ah} z^{-1}} = \frac{z}{z - e^{-ah}} \tag{141}$$

(4). $f(t) = te^{-at}$, $t \geq 0$.

Tillämpning av derivationsregeln på resultatet i (3) ger

$$F(z) = -hz \frac{\partial}{\partial z} \frac{z}{z - e^{-ah}} = he^{-ah} \frac{z}{(z - e^{-ah})^2} \tag{142}$$

(5). $f(t) = t^2 e^{-at}$, $t \geq 0$.

Tillämpning av derivationsregeln på resultatet i (4) ger

$$F(z) = -hz \frac{\partial}{\partial z} he^{-ah} \frac{z}{(z - e^{-ah})^2} = h^2 e^{-ah} \frac{z^2 + e^{-ah} z}{(z - e^{-ah})^3} \tag{143}$$

(6). $f(t) = t^3 e^{-at}$, $t \geq 0$.

Tillämpning av derivationregeln på resultatet i (5) ger

$$F(z) = -hz \frac{\partial}{\partial z} h^2 e^{-ah} \frac{z^2 + e^{-ah} z}{(z - e^{-ah})^3} =$$

$$= h^3 e^{-ah} \frac{z^3 + 4e^{-ah} z^2 + e^{-2ah} z}{(z - e^{-ah})^4} \tag{144}$$

App.2. Insignalen konstant mellan samplen (ZOH).

Vi antar att insignalen till filtret är

$$u(t) = u(t_{k-1}) \ , \ t_{k-1} \leq t < t_k \ , \ t_k = kh \ , \ k = 0,1,2,\ldots \tag{145}$$

Laplace-transformering av denna ger

$$U(s) = \int_0^\infty e^{-st}u(t)dt = \sum_{k=0}^{\infty} \int_{kh}^{kh+h} e^{-st}dt \; u(kh) =$$

$$= \sum_{k=0}^{\infty} \left[-\frac{1}{s}e^{-st} \right]_{kh}^{kh+h} u(kh) = \frac{1}{s}(1 - e^{-st}) \sum_{k=0}^{\infty} e^{-skh}u(kh) \tag{146}$$

Om vi nu låter denna signal passera ett filter med överföringsfunktionen G(s), så gäller för utsignalen, Y(s), från filtret

$$Y(s) = G(s)U(s) . \tag{147}$$

Eller fullt utskrivet

$$Y(s) = \frac{G(s)}{s} (1 - e^{-sh}) \sum_{k=0}^{\infty} e^{-skh}u(kh) \tag{148}$$

Om vi nu låter L^{-1} beteckna inversa Laplace-transformen och utnyttjar faltningsteoremet för Laplace-transformen, så får vi

$$y(t) = L^{-1} \left\{ \frac{G(s)}{s} \right\} * L^{-1} \left\{ (1 - e^{-sh}) \sum_{k=0}^{\infty} e^{-skh}u(kh) \right\} \tag{149}$$

Vi beräknar nu Z-transformen Y(z) och utnyttjar faltningteoremet för Z-transformen.

$$Y(z) = Z\left\{ L^{-1}\left\{ \frac{G(s)}{s} \right\} \right\} Z\left\{ L^{-1}\left\{ (1 - e^{-sh}) \sum_{k=0}^{\infty} e^{-skh}u(kh) \right\} \right\} \tag{150}$$

Den sista faktorn kan beräknas, man får

$$Y(z) = Z\left\{ L^{-1}\left\{ \frac{G(s)}{s} \right\} \right\} (1 - z^{-1})U(z) \ . \tag{151}$$

Det tidsdiskreta systemets överföringsfunktion, H(z), blir allså

$$H(z) = (1 - z^{-1})Z\{ L^{-1}\{ \frac{G(s)}{s} \}\} . \quad (152)$$

För att beräkna inversa Laplace-transformen till G(s)/s använder vi oss av följande resultat:

Låt G(s) = L{g(t)} och sätt $G^*(s) = G(s)/s = L\{g^*(t)\}$, då gäller

$$g^*(t) = \int_0^t g(x)dx \quad . \quad (153)$$

De filter som för oss är intressanta har en överföringsfunktion med följande utseende

$$G_n(s) = \frac{1}{(1 + s\tau)^n} = \frac{1/\tau^n}{(s + 1/\tau)^n} = \frac{a^n}{(s + a)^n} \quad (154)$$

där $a = 1/\tau$.

Motsvarande tidsfunktion är

$$g_n(t) = a^n \frac{t^{n-1}}{(n-1)!} e^{-at} \quad , \quad \text{då } t \geq 0. \quad (155)$$

Alltså får vi

$$g_n^*(t) = \frac{a^n}{(n-1)!} \int_0^t x^{n-1} e^{-ax} dx \quad , \quad (156)$$

och genom att partialintegrera, slutligen

$$g_n^*(t) = \left[-e^{-ax}\left[\frac{(ax)^{n-1}}{(n-1)!} + \frac{(ax)^{n-2}}{(n-2)!} + \ldots + ax + 1 \right] \right]_0^t =$$

$$= 1 - \left[1 + at + \ldots + \frac{(at)^{n-2}}{(n-2)!} + \frac{(at)^{n-1}}{(n-1)!} \right] e^{-at} \quad (157)$$

som gäller då $t \geq 0$.

Vi har nu all nödvändig information för att beräkna de tidsdiskreta överföringsfunktionerna $H_n(z)$.

För att minska skrivandet införs beteckningen $\gamma = e^{-ah}$.

<u>n = 1</u>

Ekvation (157) ger

$$g_1^*(t) = 1 - e^{-at} \tag{158}$$

Med hjälp av transformpar (1) och (3) fås

$$G_1^*(z) = \frac{z}{z-1} - \frac{z}{z-\gamma} \tag{159}$$

och vi får filtrets överföringsfunktion

$$H_1(z) = \frac{z-1}{z} G_1^*(z) = \frac{1-\gamma}{z-\gamma} . \tag{160}$$

$\underline{n = 2}$

Ekvation (157) ger

$$g_2^*(t) = 1 - (1 + at)e^{-at} = g_1^*(t) - ate^{-at} \tag{161}$$

Med hjälp av transformpar (4) samt resultatet ovan för n = 1 fås

$$G_2^*(z) = G_1^*(z) - ah\gamma \frac{z}{(z-\gamma)^2} \tag{162}$$

och vi får filtrets överföringsfunktion

$$H_2(z) = \frac{z-1}{z} G_2^*(z) = H_1(z) - ah\gamma \frac{z-1}{(z-\gamma)^2} =$$

$$= \frac{(1-\gamma(1+ah))z + \gamma(\gamma + ah - 1)}{(z-\gamma)^2} \tag{163}$$

$\underline{n = 3}$

Ekvation (157) ger

$$g_3^*(t) = 1 - (1 + at + \frac{(at)^2}{2})e^{-at} = g_2^*(t) - \frac{(at)^2}{2} e^{-at} \tag{164}$$

Med hjälp av transformpar (5) samt resultatet ovan för n = 2 fås

$$G_3^*(z) = G_2^*(z) - \frac{(ah)^2}{2}\gamma \frac{z^2 + \gamma z}{(z-\gamma)^3} \tag{165}$$

och vi får filtrets överföringsfunktion

$$H_3(z) = \frac{z-1}{z} G_3^*(z) = H_2(z) - \frac{(ah)^2}{2} \gamma \frac{(z-1)(z+\gamma)}{(z-\gamma)^3} \equiv$$

$$\equiv \frac{b_1 z^2 + b_2 z + b_3}{(z-\gamma)^3} \tag{166}$$

där

$$b_1 = 1 - \gamma(1 + ah + \frac{(ah)^2}{2}) \tag{167}$$

$$b_2 = \gamma(ah + \frac{(ah)^2}{2} + \gamma(ah - \frac{(ah)^2}{2}) + 2(\gamma - 1)) \tag{168}$$

$$b_3 = \gamma^2(1 - \gamma - ah + \frac{(ah)^2}{2}) \tag{169}$$

n = 4

Ekvation (157) ger

$$g_4^*(t) = 1 - (1 + at + \frac{(at)^2}{2} + \frac{(at)^3}{6}) e^{-at} = g_3^*(t) - \frac{(at)^3}{6} e^{-at} \tag{170}$$

Med hjälp av transformpar (6) samt resultatet ovan för n = 3 fås

$$G_4^*(z) = G_3^*(z) - \gamma \frac{(ah)^3}{6} \frac{z^3 + 4\gamma z^2 + \gamma^2 z}{(z-\gamma)^4} \tag{171}$$

och vi får filtrets överföringsfunktion

$$H_4(z) = \frac{z-1}{z} G_4^*(z) = H_3(z) - \gamma \frac{(ah)^3}{6} \frac{(z-1)(z^2 + 4\gamma z + \gamma^2)}{(z-\gamma)^4} \equiv$$

$$\equiv \frac{b_1 z^3 + b_2 z^2 + b_3 z + b_4}{(z-\gamma)^4} \tag{172}$$

där

$$b_1 = 1 - \gamma(1 + ah + \frac{(ah)^2}{2} + \frac{(ah)^3}{6}) \tag{173}$$

$$b_2 = \gamma\left(ah + \frac{(ah)^2}{2} + \frac{(ah)^3}{6} - 3\right) +$$

$$+ \gamma^2\left(3 + 2ah - \frac{(ah)^3}{2}\right) \qquad (174)$$

$$b_3 = 2\gamma - \gamma^2\left(1 + 2ah - \frac{2(ah)^3}{3}\right) -$$

$$- \gamma^3\left(ah - \frac{(ah)^2}{2} + \frac{(ah)^3}{6}\right) \qquad (175)$$

$$b_4 = \gamma^3\left(-1 + \gamma + ah - \frac{(ah)^2}{2} + \frac{(ah)^3}{6}\right) \qquad (176)$$

App.3. Insignalen linjär mellan samplen (Linjär interpolation).

Vi antar att insignalen till filtret är

$$u(t) = u(t_{k-1}) + \frac{t - t_{k-1}}{t_k - t_{k-1}}(u(t_k) - u(t_{k-1})) =$$

$$= u(t_{k-1}) + \frac{1}{h}(t - t_{k-1})(u(t_k) - u(t_{k-1})) \qquad (177)$$

då $t_{k-1} \leq t \leq t_k$, $t_k = kh$, $k = 0,1,2,...$

För att ta fram den tidsdiskreta överföringsfunktion som svarar mot denna insignal, använder vi oss av samma metod som användes då insignalen var konstant mellan samplen.

Laplace-transformation av insignalen ger

$$U(s) = \int_0^\infty e^{-st}u(t)dt = \sum_{k=0}^{\infty} I_k \qquad (178)$$

där

$$I_k = \int_{kh}^{kh+h} e^{-st}\left[u(kh) + \frac{1}{h}(t - kh)(u(kh+h) - u(kh)) \right]dt =$$

$$= \left[-\frac{1}{s}e^{-st}u(kh) - \left[\frac{t - kh}{sh} + \frac{1}{s^2h} \right]e^{-st}(u(kh+h) - u(kh)) \right]_{kh}^{kh+h} =$$

$$= \left[-\frac{1}{s}e^{-sh} + \frac{1}{s^2h}(1 - e^{-sh}) \right]e^{-skh}u(kh+h) +$$

$$+ \left[\frac{1}{s} - \frac{1}{s^2h}(1 - e^{-sh}) \right]e^{-skh}u(kh) \qquad (179)$$

Innan vi sätter in detta i uttrycket för U(s) lägger vi märke till att

$$\sum_{k=0}^{\infty} s^{-skh}u(kh+h) = e^{sh}\sum_{k=0}^{\infty} e^{-s(kh+h)}u(kh+h) = e^{sh}\sum_{k=0}^{\infty} e^{-skh}u(kh) \qquad (180)$$

Utnyttjar vi detta fås

$$U(s) = \frac{1}{s^2 h}(e^{sh} - 1)(1 - e^{-sh}) \sum_{k=0}^{\infty} e^{-skh} u(kh) =$$

$$= \frac{1}{s^2} \frac{(e^{sh} - 1)^2}{he^{sh}} \sum_{k=0}^{\infty} e^{-skh} u(kh) \qquad (181)$$

Vi låter åter U(s) vara insignalen till ett filter med överföringsfunktionen G(s), och för utsignalens Laplace-transform gäller

$$Y(s) = G(s)U(s)$$

det vill säga

$$Y(s) = \frac{G(s)}{s^2} \frac{(e^{sh} - 1)^2}{he^{sh}} \sum_{k=0}^{\infty} e^{-skh} u(kh) \qquad (182)$$

Vi låter L^{-1} beteckna inversa Laplace-transformen. I tidsplanet får vi, med hjälp av faltningsteoremet för Laplace-transformen

$$y(t) = L^{-1}\left\{ \frac{G(s)}{s^2} \right\} * L^{-1}\left\{ \frac{(e^{sh} - 1)^2}{he^{sh}} \sum_{k=0}^{\infty} e^{-skh} u(kh) \right\} \qquad (183)$$

Z-transformation av y(t) leder, med hjälp av faltningsteoremet för Z-transformen, till

$$Y(z) = Z\left\{ L^{-1}\left\{ \frac{G(s)}{s^2} \right\} \right\} Z\left\{ L^{-1}\left\{ \frac{(e^{sh} - 1)^2}{he^{sh}} \sum_{k=0}^{\infty} e^{-skh} u(kh) \right\} \right\} \qquad (184)$$

Där den sista faktorn kan beräknas, man får

$$Y(z) = Z\left\{ L^{-1}\left\{ \frac{G(s)}{s^2} \right\} \right\} \frac{(z - 1)^2}{hz} U(z) \qquad (185)$$

Det tidsdiskreta systemets överföringsfunktion, H(z), blir alltså

$$H(z) = \frac{(z - 1)^2}{hz} Z\left\{ L^{-1}\left\{ \frac{G(s)}{s^2} \right\} \right\} \qquad (186)$$

För att beräkna inversa Laplace-transformen till $G(s)/s^2$ utnyttjar vi följande resultat:

Låt $G(s) = L\{g(t)\}$ och sätt $G^*(s) = G(s)/s^2 = L\{g^*(t)\}$, då gäller

$$g^*(t) = \int_0^t (t - x)g(x)dx \tag{187}$$

Vårt intresse var ju riktat mot filter med en överföringsfunktion av typen

$$G_n(s) = \frac{1}{(1 + s\tau)^n} = \frac{1/\tau^n}{(s + 1/\tau)^n} = \frac{a^n}{(s + a)^n} \tag{188}$$

där $a = 1/\tau$.

Inverstransformering av denna ger

$$g_n(t) = a^n \frac{t^{n-1}}{(n-1)!} e^{-at} \quad , \quad t \geq 0. \tag{189}$$

Nu använder vi oss av ekvation (*), och får

$$g_n^*(t) = \frac{a^n}{(n-1)!} \int_0^t (t - x)x^{n-1}e^{-ax}dx \tag{190}$$

Genom upprepad partialintegration, eller genom att utnyttja resultatet ovan i fallet med ZOH, fås

$$g_n^*(t) = t - \left[t + at^2 + \ldots + \frac{a^{n-2}t^{n-1}}{(n-2)!} + \frac{a^{n-1}t^n}{(n-1)!}\right]e^{-at}$$

$$- \frac{n}{a} + \left[\frac{n}{a} + nt + \frac{nat^2}{2!} + \ldots + \frac{na^{n-2}t^{n-1}}{(n-1)!} + \frac{a^{n-1}t^n}{(n-1)!}\right]e^{-at} =$$

$$= t - \frac{n}{a} +$$

$$+ \left[\frac{n}{a} + (n-1)t + \left(\frac{n}{2!} - \frac{1}{1!}\right)at^2 + \ldots + \left(\frac{n}{(n-1)!} - \frac{1}{(n-2)!}\right)a^{n-2}t^{n-1}\right]e^{-at}$$

som gäller då $t \geq 0$ och $n \geq 1$.

Med hjälp av denna ekvation och transformparen kan vi nu beräkna de sökta tidsdiskreta överföringsfunktionerna, $H_n(z)$.

Även här nyttjar vi beteckningen $\gamma = e^{-ah}$.

<u>n = 1</u>

Ekvation (191) ger

$$g_1^*(t) = t - \frac{1}{a} + \frac{1}{a}e^{-at} \tag{192}$$

Med hjälp av transformpar (1), (2) och (3) fås

$$G_1^*(z) = \frac{hz}{(z-1)^2} - \frac{1}{a}\frac{z}{z-1} + \frac{1}{a}\frac{z}{z-\gamma} \tag{193}$$

och vi får filtrets överföringsfunktion

$$H_1(z) = \frac{(z-1)^2}{hz} G_1^*(z) = 1 - \frac{1}{ah}(z-1) + \frac{1}{ah}\frac{(z-1)^2}{z-\gamma} \equiv$$

$$\equiv \frac{b_0 z + b_1}{z-\gamma} \tag{194}$$

där

$$b_0 = 1 - \frac{1}{ah}(1-\gamma) \tag{195}$$

$$b_1 = \frac{1}{ah}(1-\gamma) - \gamma \tag{196}$$

$\underline{n = 2}$

Ekvation (191) ger

$$g_2^*(t) = t - \frac{2}{a} + (\frac{2}{a} + t)e^{-at} \tag{197}$$

Med hjälp av transformpar (1)-(4) fås

$$G_2^*(z) = \frac{hz}{(z-1)^2} - \frac{2}{a}\frac{z}{z-1} + \frac{2}{a}\frac{z}{z-\gamma} + \frac{h\gamma z}{(z-\gamma)^2} \tag{198}$$

och vi får filtrets överföringsfunktion

$$H_2(z) = \frac{(z-1)^2}{hz} G_2^*(z) =$$

$$= 1 - \frac{2}{ah}(z-1) + \frac{2}{ah}\frac{(z-1)^2}{z-\gamma} + \gamma\frac{(z-1)^2}{(z-\gamma)^2} \equiv \frac{b_0 z^2 + b_1 z + b_2}{(z-\gamma)^2} \tag{199}$$

där

$$b_0 = 1 + \gamma - \frac{2}{ah}(1-\gamma) \tag{200}$$

$$b_1 = \frac{2}{ah}(1 - \gamma^2) - 4\gamma \tag{201}$$

$$b_2 = \gamma(1 + \gamma - \frac{2}{ah}(1 - \gamma)) \tag{202}$$

n = 3

Ekvation (191) ger

$$g_3^*(t) = t - \frac{3}{a} + (\frac{3}{a} + 2t + \frac{at^2}{2})e^{-at} \tag{203}$$

Med hjälp av transformpar (1)-(5) fås

$$G_3^*(z) = \frac{hz}{(z-1)^2} - \frac{3}{a}\frac{z}{z-1} + \frac{3}{a}\frac{z}{z-\gamma} + \frac{2h\gamma z}{(z-\gamma)^2} +$$

$$+ \frac{ah^2\gamma}{2}\frac{z^2 + \gamma z}{(z-\gamma)^3} \tag{204}$$

och vi får filtrets överföringsfunktion

$$H_3(z) = \frac{(z-1)^2}{hz} G_3^*(z) =$$

$$= 1 - \frac{3}{ah}(z-1) + \frac{3}{ah}\frac{(z-1)^2}{z-\gamma} + 2\gamma\frac{(z-1)^2}{(z-\gamma)^2} +$$

$$+ \frac{ah\gamma}{2}\frac{(z-1)^2(z+\gamma)}{(z-\gamma)^3} \equiv \frac{b_0z^3 + b_1z^2 + b_2z + b_3}{(z-\gamma)^3} \tag{205}$$

där

$$b_0 = 1 + 2\gamma - \frac{3}{ah}(1 - \gamma) + \frac{ah\gamma}{2} \tag{206}$$

$$b_1 = -7\gamma - 2\gamma^2 + \frac{3}{ah}(1 - \gamma)(1 + 2\gamma) - \frac{ah\gamma}{2}(2 - \gamma) \tag{207}$$

$$b_2 = 2\gamma + 7\gamma^2 - \frac{3\gamma}{ah}(1 - \gamma)(1 + 2\gamma) + \frac{ah\gamma}{2}(1 - 2\gamma) \tag{208}$$

$$b_3 = -\gamma^2(2 + \gamma - \frac{3}{ah}(1 - \gamma) - \frac{ah}{2}) \tag{209}$$

n = 4

Ekvation (191) ger

$$g_4^*(t) = t - \frac{4}{a} + (\frac{4}{a} + 3t + at^2 + \frac{a^2t^3}{6})e^{-at} \tag{210}$$

Med hjälp av transformpar (1)-(6) fås

$$G_4^*(z) = \frac{hz}{(z-1)^2} - \frac{4}{a}\frac{z}{z-1} + \frac{4}{a}\frac{z}{z-\gamma} + \frac{3h\gamma z}{(z-\gamma)^2} + ah^2\gamma\frac{z^2+\gamma z}{(z-\gamma)^3} +$$

$$+ \frac{a^2h^3\gamma}{6}\frac{z^3+4\gamma z^2+\gamma^2 z}{(z-\gamma)^4} \tag{211}$$

och vi får filtrets överföringsfunktion

$$H_4(z) = \frac{(z-1)^2}{hz}G_4^*(z) =$$

$$= 1 - \frac{4}{ah}(z-1) + \frac{4}{ah}\frac{(z-1)^2}{z-\gamma} + 3\gamma\frac{(z-1)^2}{(z-\gamma)^2} +$$

$$+ ah\gamma\frac{(z-1)^2(z+\gamma)}{(z-\gamma)^3} + \frac{(ah)^2\gamma}{6}\frac{(z-1)^2(z^2+4\gamma z+\gamma^2)}{(z-\gamma)^4} \equiv$$

$$\equiv \frac{b_0z^4 + b_1z^3 + b_2z^2 + b_3z + b_4}{(z-\gamma)^4} \tag{212}$$

där

$$b_0 = 1 + 3\gamma - \frac{4}{ah}(1-\gamma) + ah\gamma + \frac{(ah)^2\gamma}{6} \tag{213}$$

$$b_1 = -\gamma(10+6\gamma) + \frac{4}{ah}(1-\gamma)(1+3\gamma) - 2ah\gamma - \frac{(ah)^2\gamma}{3}(1-2\gamma) \tag{214}$$

$$b_2 = \gamma\left[3 + 18\gamma + 3\gamma^2 - \frac{12}{ah}(1-\gamma^2) + ah(1-\gamma^2) +\right.$$

$$\left. + \frac{(ah)^2}{6}(1-8\gamma+\gamma^2)\right] \tag{215}$$

$$b_3 = \gamma^2\left[-6 - 10\gamma + \frac{4}{ah}(1-\gamma)(3+\gamma) + 2ah\gamma + \frac{(ah)^2}{3}(2-\gamma)\right] \quad (216)$$

$$b_4 = \gamma^3\left[3 + \gamma - \frac{4}{ah}(1-\gamma) - ah + \frac{(ah)^2}{6}\right] \quad (217)$$

App.4. Tustins metod.

Vi ska nu se hur det går till att översätta filtren Λ^k från kontinuerlig till diskret tid med hjälp av Tustins approximation.

Tidigare fann vi att en sådan översättning går till genom att sätta

$$s = \frac{\omega_1}{\tan(\omega_1 h/2)} \frac{z-1}{z+1} \tag{218}$$

i den kontinuerliga överföringsfunktionen. Av de översättnings-metoder som används här är detta alltså den klart enklaste att utföra, därmed är dock inte sagt att det är den bästa.

De filter som vi är intresserade att översätta har överföringsfunktioner av typen

$$G_n(s) = \frac{1}{(1 + s\tau)^n} = \frac{1/\tau^n}{(s + 1/\tau)^n} = \frac{a^n}{(s + a)^n} \tag{219}$$

där $a = 1/\tau$.

Om vi sätter

$$\kappa = \frac{\omega_1}{\tan(\omega_1 h/2)} , \tag{220}$$

och sedan gör transformationen enligt ovan, så fås

$$H_n(z) = G_n\left(\kappa\frac{z-1}{z+1}\right) = \frac{a^n}{\left[\kappa\frac{z-1}{z+1} + a\right]^n} =$$

$$= \frac{a^n(z + 1)^n}{((\kappa + a)z - (\kappa - a))^n} \tag{221}$$

division med $(\kappa + a)^n$ ger nu

$$H_n(z) = \frac{a^n}{(\kappa + a)^n} \frac{(z + 1)^n}{\left[z - \frac{\kappa - a}{\kappa + a}\right]^n} \tag{222}$$

Vi sätter nu

$$\gamma = \frac{\kappa - a}{\kappa + a} \tag{223}$$

och får slutligen

$$H_n(z) = \left[\frac{1-\gamma}{2}\right]^n \frac{(z+1)^n}{(z-\gamma)^n} \tag{224}$$

För fullständighetens skull utnyttjar vi binomialteoremet och får de sökta överföringsfunktionerna (n = 1,2,3,4)

$$H_1(z) = \frac{1-\gamma}{2}\,\frac{z+1}{z-\gamma} \tag{225}$$

$$H_2(z) = \left[\frac{1-\gamma}{2}\right]^2 \frac{z^2+2z+1}{z^2-2\gamma z+\gamma^2} \tag{226}$$

$$H_3(z) = \left[\frac{1-\gamma}{2}\right]^3 \frac{z^3+3z^2+3z+1}{z^3-3\gamma z^2+3\gamma^2 z-\gamma^3} \tag{227}$$

$$H_4(z) = \left[\frac{1-\gamma}{2}\right]^4 \frac{z^4+4z^3+6z^2+4z+1}{z^4-4\gamma z^3+6\gamma^2 z^2-4\gamma^3 z+\gamma^4} \tag{228}$$

Det återstår nu bara att välja ω_1, i de implementering som gjorts har denna valts så att $\omega_1 = 1/\tau$. Vi får med detta val

$$\gamma = \frac{\kappa - a}{\kappa + a} = \frac{1-\tan(\frac{h}{2\tau})}{1+\tan(\frac{h}{2\tau})} \,. \tag{229}$$

App.5. M-matriserna.

Nedan visas de matriser som används för att generera

$$P_k = (\Lambda\tau)^{n-k}(1-\Lambda)^k \quad , \quad k = 0,1,2,3,4 \tag{230}$$

Tidigare har vi definierat

$$\Lambda_n^* = [1 \ \Lambda \ \Lambda^2 \ \ldots \ \Lambda^n]^T \tag{231}$$

samt att det skulle gälla

$$P_k = (\mathrm{rad}_{n-k} \ M_n)\Lambda_n^* \quad , \quad k = 0,1,2,3,4. \tag{232}$$

Vi får

$$M_1 = \begin{bmatrix} 1 & -1 \\ 0 & \tau \end{bmatrix} \tag{233}$$

$$M_2 = \begin{bmatrix} 1 & -2 & 1 \\ 0 & \tau & -\tau \\ 0 & 0 & \tau^2 \end{bmatrix} \tag{234}$$

$$M_3 = \begin{bmatrix} 1 & -3 & 3 & -1 \\ 0 & \tau & -2\tau & \tau \\ 0 & 0 & \tau^2 & -\tau^2 \\ 0 & 0 & 0 & \tau^3 \end{bmatrix} \tag{235}$$

$$M_4 = \begin{bmatrix} 1 & -4 & 6 & -4 & 1 \\ 0 & \tau & -3\tau & 3\tau & -\tau \\ 0 & 0 & \tau^2 & -2\tau^2 & \tau^2 \\ 0 & 0 & 0 & \tau^3 & -\tau^3 \\ 0 & 0 & 0 & 0 & \tau^4 \end{bmatrix} \tag{236}$$

App.6. Stirling-talen av 1:a sorten.

Stirling-talen av 1:a sorten, s_k^n, är definierade av följande samband

$$\Pi_0^n(t - k + 1) = \Sigma_1^n s_k^n t^k \tag{237}$$

där både produkt- och summationsindex är k, och $n \geq 1$.

Man kan visa att dessa uppfyller följande rekursiva formel

$$s_k^{n+1} = s_{k-1}^n - n s_k^n \tag{238}$$

med

$$s_n^n = 1 \tag{239}$$

$$s_k^n = 0 \ , \ k \leq 0 \text{ eller } k \geq n+1 \ .$$

Nedan följer en tabell med några av dessa tal tabulerade.

n\k	1	2	3	4	5
1	1				
2	-1	1			
3	2	-3	1		
4	-6	11	-6	1	
5	24	-50	35	-10	1

App.7. Definitionsmoduler.

DEFINITION MODULE GlobDef;

```
(* This module contains all the global variables and the data types
   used to define them. There is one exception to this the vector used
   to store the history of the estimated parameters is defined in the
   module PBuff.def. *)

FROM GEMtypes IMPORT handletype, point;

FROM Kernel IMPORT Semaphore;

EXPORT QUALIFIED ScreenWindow, OrderOfSystem, h, tau, NoParToEst,
                 Tick,
                 OptionType, FiltKoeffType, StepType,
                 KnownAPolType, KnownBPolType, FiPolType,
                 KnownArrayType, KnownRefType, TypeOfPar, VariansType,
                 MaxOrderOfSystem, MaxEstimOrder, EstTime,
                 AKoeffRef, BKoeffRef, AKoeffAngle, BKoeffAngle,
                 AKoeffPos, BKoeffPos,
                 P, FiPol,
                 Lambda, MinEps,
                 KnownAPar, KnownBPar,
                 KnownAnglePol, KnownRefPol,
                 RefOption, AngleOption, PosOption,
                 MainPrio, ClockPrio, RegulatePrio, EstPrio,
                 Ref, Angle, Pos,
                 FiltredRef, FiltredAngle, FiltredPos,
                 StartClock, ClockMutex, SyncSem, NewFi,
                 StartRegulate, RegulateMutex, MonitorFi,
                 StopClock, StopRegulate,
                 YMinLimit, YMaxLimit,
                 RefChannel, AngleChannel, PosChannel;


CONST MaxOrderOfSystem = 4;
        (* maximum order of the process under identifikation *)
      MaxEstimOrder = 2*MaxOrderOfSystem;
        (* the largest number of parameters that can be estimated *)
      StringLength = 10;
      MainMem = 10000;   (* memory size for the processes *)
      ClockMem = 2000;
      RegulateMem = 5000;
      EstMem = 5000;
      MainPrio = 4;      (* the priority of the processes *)
      ClockPrio = 1;
      RegulatePrio = 2;
      EstPrio = 3;

TYPE  FiltKoeffType = ARRAY[1..MaxOrderOfSystem],
```

```
                    [0..MaxOrderOfSystem] OF REAL;
    (* koefficients for all filters *)
  KnownAPolType = ARRAY[0..MaxOrderOfSystem] OF REAL;
    (* used for left hand side in the estimation equation *)
  KnownBPolType = RECORD
                    Pol : ARRAY[1..MaxOrderOfSystem] OF REAL;
                    Start : INTEGER;
                  END;
    (* used for left hand side in the estimation equation *)
  TypeOfPar = ( APar, BPar );
  PolElement = RECORD
                 Pol : ARRAY[1..MaxOrderOfSystem] OF REAL;
                 Start : INTEGER;
                 ParType : TypeOfPar;
               END;
  FiPolType = ARRAY[1..MaxEstimOrder] OF PolElement;
    (* used to tell which filter outputs to use *)
  VariansType = ARRAY[1..MaxEstimOrder],
                  [1..MaxEstimOrder] OF REAL;
    (* type for the covariance matrix *)
  StringType = ARRAY[0..StringLength] OF CHAR;
    (* the type used for all strings in the program *)
  KnownRefType = RECORD
                   ParP : point;
                   ParString,
                   PlotString : StringType;
                   ParValue : REAL;
                   ToPlot,
                   Value : BOOLEAN;
                 END;
  KnownArrayType = ARRAY[1..MaxOrderOfSystem] OF KnownRefType;
    (* used to hold the values for the known parameters, if a
    parameter is not known it holds the start value *)
  OptionType = ( Tustin, ConstIn, LinIn, Own );
    (* used to tell whitch filter approximation is selected *)
  StepType = ( Stop, None, MainWindow, ReferensWindow,
               PositionWindow, AngleWindow, DisplayWindow,
               EstWindow, RunWindow, GraphWindow, POutWindow );
    (* specifies the possible menues *)

VAR  ScreenWindow : handletype;
       (* the handle that is used by all procedures *)
    OrderOfSystem, NoParToEst, EstTime, Tick : INTEGER;
    RefChannel, AngleChannel, PosChannel : CARDINAL;
    h, tau, Lambda, MinEps : REAL;
    RefOption, AngleOption, PosOption : OptionType;
    AKoeffRef, BKoeffRef, AKoeffAngle, BKoeffAngle,
    AKoeffPos, BKoeffPos                    : FiltKoeffType;
    P : VariansType;
    FiPol : FiPolType;
    KnownAPar, KnownBPar : KnownArrayType;
    KnownAnglePol : KnownAPolType;
    KnownRefPol : KnownBPolType;
    Ref, Angle, Pos : ARRAY[0..MaxOrderOfSystem] OF REAL;
```

```
        (* holds the values of the input and output of the
           object under identifycation *)
     FiltredRef, FiltredAngle, FiltredPos :
           ARRAY[1..MaxOrderOfSystem],[0..MaxOrderOfSystem] OF REAL;
        (* holds the corresponding filtrated values *)
    StartClock, ClockMutex, SyncSem, NewFi,
    StartRegulate, RegulateMutex, MonitorFi : Semaphore;
    StopClock, StopRegulate : BOOLEAN;

    YMinLimit, YMaxLimit : REAL;
      (* max and min values on the y-axis for the plotted parameters *)
END GlobDef.
```

DEFINITION MODULE PBuff;

```
(* This modula contains the vector used to store the history
   of the estimated parameters, also defined here is the maximum
   allovable number of values that can be stored. *)

FROM GlobDef IMPORT MaxEstimOrder;

EXPORT QUALIFIED PlotPoints, NoPointsToPlot, MaxNoPoints;

CONST MaxNoPoints = 500;

VAR   PlotPoints : ARRAY[1..MaxNoPoints],[1..MaxEstimOrder] OF REAL;
      NoPointsToPlot : INTEGER;


END PBuff.
```

DEFINITION MODULE CalcFilt;

```
(* The corresponding implementation module contains the procedures that
   calculates the coefficients in the transfer function of the filter
   approximations. *)

EXPORT QUALIFIED TustinApprox, ConstInpApprox, LinInpApprox;


PROCEDURE TustinApprox( VAR AKoeff, BKoeff : FiltKoeffType;
                        h, tau : REAL );
(* Calculates the filters using the Tustin approximation. *)

PROCEDURE ConstInpApprox( VAR AKoeff, BKoeff : FiltKoeffType;
                          h, tau : REAL );
(* Calculates the filters assuming that the filter input is constant
   between the samples. *)

PROCEDURE LinInpApprox( VAR AKoeff, BKoeff : FiltKoeffType;
                        h, tau : REAL );
(* Calculates the filters assuming that filter input is linear between
   the samples. That is linearly interpolated. *)
```

```
END CalcFilt.
```

DEFINITION MODULE Running;

```
(* The corresponding implemetation module contains all processes
   that are active when the identification is carried out. Also
   contained here is the procedure that sets up the vectors used
   to make it possible to not estimate the known parameters. *)

FROM GlobDef IMPORT KnownArrayType, KnownAPolType, KnownBPolType,
                    FiPolType, MaxEstimOrder;


EXPORT QUALIFIED SetupForFi, Clock, Regulate, TestEstim, InitForRun,
                 Theta;

TYPE  FiType = ARRAY[0..MaxEstimOrder] OF REAL;

VAR   Theta : FiType;

PROCEDURE SetupForFi( KnownAPar, KnownBPar : KnownArrayType;
                      VAR KnownAnglePol : KnownAPolType;
                      VAR KnownRefPol : KnownBPolType;
                      VAR FiPol : FiPolType;
                      VAR NoParToEst : INTEGER );
(* this procedure sets up the vectors that are used to calculate the
   fi-vector, if some parameter is known it is eliminated from the
   theta-vector thereby making it possible to only estimate the unknown
   parameters of the transfer function. *)

PROCEDURE InitForRun;
(* this procedure initiates all variables used during identification *)

(* process *) PROCEDURE Clock;
(* this process makes sure that the sampling frequency is kept constant *)

(* process *) PROCEDURE Regulate;
(* this process carries out the filtering and regulation, it also sends
   the fi-vector to the process TestEstim *)

(* process *) PROCEDURE TestEstim;
(* this process carries out the estimation, it also saves the old values
   of the estimated parameters so that they can be plotted afterwards *)

END Running.
```

DEFINITION MODULE Filtin;

```
(* The corresponding implementation module contains the procedures that
   handles the filter menues. *)


FROM GlobDef IMPORT StepType;
```

```
EXPORT QUALIFIED  FiltRefMain, FiltAngleMain, FiltPosMain;


PROCEDURE FiltRefMain( VAR NextStep : StepType );
(* Handles the choise of the type of approximation used for the filtering
   of the input of the identification object *)

PROCEDURE FiltAngleMain( VAR NextStep : StepType );
(* Handles the choise of the type of approximation used for the filtering
   of the output of the identification object *)

PROCEDURE FiltPosMain( VAR NextStep : StepType );
(* This procedure is only used when there is two identifycation loops
   running, this can not be handled in the current implementation *)
END Filtin.
```

DEFINITION MODULE EstIn;

```
(* The corresponding implementation module contains the procedures that
   handles the ESTIMATE menue. *)

FROM GlobDef IMPORT  StepType;

EXPORT QUALIFIED EstimateMain;


PROCEDURE EstimateMain( VAR NextStep : StepType );
(* The main procedure of the ESTIMATE menue. *)

END EstIn.
```

DEFINITION MODULE DoRun;

```
(* The corresponding implementation module contains the procedure that
   puts the outer process in a waiting state. *)

EXPORT QUALIFIED RunMain;

PROCEDURE RunMain;
(* Main procedure for the RUN -menue-, read virtual. *)

END DoRun.
```

DEFINITION MODULE Disp;

```
(* The corresponding implementation module containes the routines
   that handles the DISPLAY menue. In this menue it is possible to
   do the following things: set the values for the max and min values
   represented on the y-axis, select the parameters that one whiches
   to plot and select one of the following menues STOP,MAIN and GRAPH *(

FROM GlobDef IMPORT StepType;
```

```
EXPORT QUALIFIED DispMain;

PROCEDURE DispMain( VAR NextStep : StepType);
(* main procedure of the DISPLAY module. *)

END Disp.
```

DEFINITION MODULE Graph;

```
(* The corresponding implementation module contains the used to graph
   the history of the estimated parameters. *)

FROM GlobDef IMPORT StepType;

EXPORT QUALIFIED GraphMain;

PROCEDURE GraphMain( VAR NextStep : StepType );
(* The main procedure that handles the GRAPH module *)

END Graph.
```

DEFINITION MODULE POut;

```
(* The corresponding implementation module contains the procedures used
   for the writing of the covariance matrix P to the screen. *)

FROM GlobDef IMPORT  StepType;

EXPORT QUALIFIED POutMain;

PROCEDURE POutMain( VAR NextStep : StepType );
(* The main procedure for the POUT menue. *)

END POut.
```

DEFINITION MODULE Opcom;

```
EXPORT QUALIFIED OpcomMain;

PROCEDURE OpcomMain;

END Opcom
```

App.8. Implementationsmoduler.

IMPLEMENTATION MODULE GlobDef;

```
END GlobDef.
```

IMPLEMENTATION MODULE PBuff;

```
END PBuff
```

IMPLEMENTATION MODULE CalcFilt;

```
(* This module contaons the prcedures that calculates the coefficients for
   the filter approximations. *)

FROM MathLib IMPORT exp, sin, cos;

FROM GlobDef IMPORT FiltKoeffType;


PROCEDURE TustinApprox( VAR AKoeff, BKoeff : FiltKoeffType;
                        h, tau : REAL );

(* this procedure calculates the filter coefficients using the Tustin *)
(* approximation, the current values of h and tau is used *)
(* upon return AKoeff contains the coefficients of A(q) and *)
(* BKoeff contains the coefficients of B(q) *)

VAR  a, b, Tan, Help : REAL;

BEGIN
   Tan := sin( h/(2.0*tau) )/cos( h/(2.0*tau) );
   b := Tan/( 1.0 + Tan );
   a := ( 1.0 - Tan )/( 1.0 + Tan );
   BKoeff[ 1, 0 ] := b;
   BKoeff[ 1, 1 ] := b;
   Help := b*b;
   BKoeff[ 2, 0 ] := Help;
   BKoeff[ 2, 1 ] := 2.0*Help;
   BKoeff[ 2, 2 ] := Help;
   Help := Help*b;
   BKoeff[ 3, 0 ] := Help;
   BKoeff[ 3, 1 ] := 3.0*Help;
   BKoeff[ 3, 2 ] := 3.0*Help;
   BKoeff[ 3, 3 ] := Help;
   Help := Help*b;
   BKoeff[ 4, 0 ] := Help;
   BKoeff[ 4, 1 ] := 4.0*Help;
   BKoeff[ 4, 2 ] := 6.0*Help;
   BKoeff[ 4, 3 ] := 4.0*Help;
   BKoeff[ 4, 4 ] := Help;
```

```
   AKoeff[ 1, 0 ] := 1.0;
   AKoeff[ 1, 1 ] := -a;

   AKoeff[ 2, 0 ] := 1.0;
   AKoeff[ 2, 1 ] := -2.0*a;
   AKoeff[ 3, 0 ] := 1.0;
   AKoeff[ 3, 1 ] := -3.0*a;
   AKoeff[ 4, 0 ] := 1.0;
   AKoeff[ 4, 1 ] := -4.0*a;
   Help := a*a;
   AKoeff[ 2, 2 ] := Help;
   AKoeff[ 3, 2 ] := 3.0*Help;
   AKoeff[ 4, 2 ] := 6.0*Help;
   Help := Help*a;
   AKoeff[ 3, 3 ] := -Help;
   AKoeff[ 4, 3 ] := -4.0*Help;

   AKoeff[ 4, 4 ] := Help*a;
END TustinApprox;


PROCEDURE ConstInpApprox( VAR AKoeff, BKoeff : FiltKoeffType;
                          h, tau : REAL );

(* this procedure calculates the filter coefficients using the fact *)
(* that if the input filter is assumed constant over the sampling *)
(* period it is possible to calculate the exact discrete transfer *)
(* function, the current values of h and tau is used *)
(* upon return AKoeff contains the coefficients of A(q) and *)
(* BKoeff contains the coefficients of B(q) *)

VAR  Help, Help2, Kvot, p : REAL;

BEGIN
   Kvot := h/tau;
   p := exp( -Kvot );
   BKoeff[ 1, 0 ] := 0.0;
   BKoeff[ 1, 1 ] := 1.0 - p;
   AKoeff[ 1, 0 ] := 1.0;
   AKoeff[ 1, 1 ] := -p;

   BKoeff[ 2, 0 ] := 0.0;
   BKoeff[ 2, 1 ] := 1.0 - p*( 1.0 + Kvot );
   BKoeff[ 2, 2 ] := p*( p + Kvot - 1.0 );

   Help := Kvot*Kvot/2.0;
   Help2 := p*p;
   BKoeff[ 3, 0 ] := 0.0;
   BKoeff[ 3, 1 ] := BKoeff[ 2, 1 ] - p*Help;
   BKoeff[ 3, 2 ] := BKoeff[ 2, 2 ] - p*BKoeff[ 2, 1 ] + Help*p*(1.0-p);
   BKoeff[ 3, 3 ] := Help2*(Help - p - Kvot + 1.0 );

   Help := p*Help*Kvot/3.0;
```

```
   BKoeff[ 4, 0 ] := 0.0;
   BKoeff[ 4, 1 ] := BKoeff[ 3, 1 ] - Help;
   BKoeff[ 4, 2 ] := BKoeff[3,2] - p*BKoeff[3,1] - Help*(4.0*p - 1.0 );
   BKoeff[ 4, 3 ] := BKoeff[3,3] - p*BKoeff[3,2] + Help*p*(4.0 - p );
   BKoeff[ 4, 4 ] := p*( Help*p - BKoeff[3,3] );

   AKoeff[ 1, 0 ] := 1.0;
   AKoeff[ 1, 1 ] := -p;

   AKoeff[ 2, 0 ] := 1.0;
   AKoeff[ 2, 1 ] := -2.0*p;
   AKoeff[ 3, 0 ] := 1.0;
   AKoeff[ 3, 1 ] := -3.0*p;
   AKoeff[ 4, 0 ] := 1.0;
   AKoeff[ 4, 1 ] := -4.0*p;
   Help := p*p;
   AKoeff[ 2, 2 ] := Help;
   AKoeff[ 3, 2 ] := 3.0*Help;
   AKoeff[ 4, 2 ] := 6.0*Help;
   Help := Help*p;
   AKoeff[ 3, 3 ] := -Help;
   AKoeff[ 4, 3 ] := -4.0*Help;

   AKoeff[ 4, 4 ] := Help*p;
END ConstInpApprox;


PROCEDURE LinInpApprox( VAR AKoeff, BKoeff : FiltKoeffType;
                        h, tau : REAL );

(* this procedure calculates the filter coefficients using the fact *)
(* that if the input filter is assumed linear over the sampling *)
(* period it is possible to calculate the exact discrete transfer *)
(* function, the current values of h and tau is used *)
(* upon return AKoeff contains the coefficients of A(q) and *)
(* BKoeff contains the coefficients of B(q) *)

VAR  Help, p2, Kvot, p, Kvot2 : REAL;

BEGIN
   Kvot := h/tau;
   p := exp( -Kvot );

   BKoeff[1,0] := 1.0 - (1.0-p)/Kvot;
   BKoeff[1,1] := (1.0-p)/Kvot - p;

   BKoeff[2,0] := 1.0+p-2.0*(1.0-p)/Kvot;
   BKoeff[2,1] := 2.0*(1.0-p)*(1.0+p)/Kvot-4.0*p;
   BKoeff[2,2] := p*BKoeff[2,0];

   BKoeff[3,0] := 0.5*Kvot*p + 2.0*p + 1.0 - 3.0*(1.0-p)/Kvot;
   BKoeff[3,1] := 3.0*(1.0-p)*(1.0+2.0*p)/Kvot - p*(0.5*Kvot*(2.0-p) +
                  2.0*(2.0+p) + 3.0);
   BKoeff[3,2] := p*(0.5*Kvot*(1.0-2.0*p) + 2.0*(1.0+2.0*p) + 3.0*p -
```

```
                3.0*(1.0-p)*(2.0+p)/Kvot);
   BKoeff[3,3] := p*p*(0.5*Kvot - 2.0 - p + 3.0*(1.0-p)/Kvot);

   Kvot2 := Kvot*Kvot;
   p2 := p*p;

   BKoeff[4,0] := p*(Kvot2/6.0 + Kvot + 3.0) + 1.0 - 4.0*(1.0-p)/Kvot;
   BKoeff[4,1] := p*(Kvot2*(2.0*p-1.0)/3.0 - 2.0*Kvot - 6.0*(1.0+p) - 4.0) +
                  4.0*(1.0-p)*(1.0+3.0*p)/Kvot;
   BKoeff[4,2] := p*(Kvot2*(1.0-8.0*p+p2)/6.0 + Kvot*(1.0-p2) +
                  3.0*(1.0+4.0*p+p2) + 6.0*p - 12.0*(1.0-p2)/Kvot);
   BKoeff[4,3] := p2*(Kvot2*(2.0-p)/3.0 + 2.0*p*Kvot - 6.0*(1.0+p) - 4.0*p +
                  4.0*(1.0-p)*(3.0+p)/Kvot);
   BKoeff[4,4] := p*p2*(Kvot2/6.0 - Kvot + 3.0 + p - 4.0*(1.0-p)/Kvot);


   AKoeff[ 1, 0 ] := 1.0;
   AKoeff[ 1, 1 ] := -p;

   AKoeff[ 2, 0 ] := 1.0;
   AKoeff[ 2, 1 ] := -2.0*p;
   AKoeff[ 3, 0 ] := 1.0;
   AKoeff[ 3, 1 ] := -3.0*p;
   AKoeff[ 4, 0 ] := 1.0;
   AKoeff[ 4, 1 ] := -4.0*p;
   Help := p*p;
   AKoeff[ 2, 2 ] := Help;
   AKoeff[ 3, 2 ] := 3.0*Help;
   AKoeff[ 4, 2 ] := 6.0*Help;
   Help := Help*p;
   AKoeff[ 3, 3 ] := -Help;
   AKoeff[ 4, 3 ] := -4.0*Help;

   AKoeff[ 4, 4 ] := Help*p;
END LinInpApprox;

END CalcFilt.
```

IMPLEMENTATION MODULE Running;

```
FROM Kernel IMPORT Wait, Signal, SetPriority, WaitTime;

FROM MathLib IMPORT round;

FROM PBuff IMPORT NoPointsToPlot, PlotPoints;

FROM GlobDef IMPORT h, tau, OrderOfSystem, MaxOrderOfSystem, EstTime,
                    Tick,
                    MinEps, Lambda, MaxEstimOrder, NoParToEst,
                    ClockPrio, RegulatePrio, EstPrio,
                    KnownArrayType, KnownAPolType, KnownBPolType,
                    FiPolType, TypeOfPar,
                    RefChannel, AngleChannel, PosChannel,
                    StartClock, ClockMutex, SyncSem,
```

```
                    StartRegulate, RegulateMutex, MonitorFi,
                    NewFi, StopClock, StopRegulate,
                    Ref, Angle, Pos,
                    FiltredRef, FiltredAngle, FiltredPos,
                    AKoeffRef, BKoeffRef, AKoeffAngle,
                    BKoeffAngle, AKoeffPos, BKoeffPos,
                    FiPol, P, KnownAnglePol, KnownRefPol,
                    KnownAPar, KnownBPar;


FROM AnalogIO IMPORT ADIn, DAOut;


TYPE  PolMatrixType = ARRAY[0..MaxOrderOfSystem],
                      [0..MaxOrderOfSystem] OF REAL;
 (*   FiType = ARRAY[0..MaxEstimOrder] OF REAL;  *)


VAR   FiMon, PFi : FiType;
      Yoldxpid, Yoldfipid, Yrxpid, Yrfipid,
      Kxpid, Kfipid, Alfaxpid, Alfafipid, Betaxpid, Betafipid,
      Gammaxpid, Gammafipid, Gdxpid, Gdfipid : REAL;
      Ppartxpid, Ppartfipid, Ipartxpid,
      Dpartxpid, Dpartfipid : REAL;


PROCEDURE Abs( Value : REAL ) : REAL;

BEGIN
   IF Value > 0.0 THEN
      RETURN Value;
   ELSE
      RETURN -Value;
   END;
END Abs;

PROCEDURE InitPID;

(* initiates the PID-regulators used to control the ball and beam plant
   that was used for testing the program. There is two kascaded PID's
   the first takes the position of the ball and delivers the setpoint
   for the angle to the second PID whitch compare this with the actual
   value of the angle and delivers the control signal to the motor. *)

VAR  Tixpid, Tdxpid, Tdfipid, Gdxpid, Gdfifid : REAL;

BEGIN
   Kxpid := -0.6;
   Kfipid := 2.0;
   Tixpid := 2.0;
   Tdxpid := 0.65;
   Tdfipid := 0.15;
   Gdxpid := 10.0;
   Gdfipid := 10.0;
```

```
   Alfaxpid := Kxpid*h/Tixpid;
   Betaxpid := Tdxpid/(Gdxpid*h+Tdxpid);
   Betafipid := Tdfipid/(Gdfipid*h+Tdfipid);
   Gammaxpid := Betaxpid*Kxpid*Gdxpid;
   Gammafipid := Betafipid*Kfipid*Gdfipid;
   Ppartxpid := 0.0;
   Ppartfipid := 0.0;
   Ipartxpid := 0.0;
   Dpartxpid := 0.0;
   Dpartfipid := 0.0;
   Yrxpid := 0.0;
   Yrfipid := 0.0;
END InitPID;


PROCEDURE InitForRun;

(* makes the initialisations that is needed before an identification
   run can be made. *)

VAR I, J, Index : INTEGER;

BEGIN
   InitPID;
   FOR I := 0 TO MaxOrderOfSystem DO
      Ref[I] := 0.0;
      Angle[I] := 0.0;
      Pos[I] := 0.0;
   END;
   FOR I := 1 TO MaxOrderOfSystem DO
      FOR J := 0 TO MaxOrderOfSystem DO
         FiltredRef[I,J] := 0.0;
         FiltredAngle[I,J] := 0.0;
         FiltredPos[I,J] := 0.0;
      END;
   END;
   Index := 0;
   FOR I := 1 TO OrderOfSystem DO
      IF NOT KnownAPar[I].Value THEN
         Index := Index + 1;
         Theta[Index] := -KnownAPar[I].ParValue;
      END;
   END;
   FOR I := 1 TO OrderOfSystem DO
      IF NOT KnownBPar[I].Value THEN
         Index := Index + 1;
         Theta[Index] := KnownBPar[I].ParValue;
      END;
   END;
   FOR I := 1 TO MaxEstimOrder DO
      FOR J := 1 TO MaxEstimOrder DO
         IF I <> J THEN
            P[I,J] := 0.0;
         END;
```

```
      END;
   END;
END InitForRun;


PROCEDURE MakePasTri( VAR PasTri : PolMatrixType );

(* upon return the matrix PasTri contains a Pascal triangle *)
(* however the sign of the elements is not allways positive, *)
(* the sign is positive if (I+J) even, and negative if (I+J) odd *)
(* think of the classical adjoint *)

VAR  I,J : INTEGER;

BEGIN
   FOR J := 0 TO 4 DO
      FOR I := J TO 4 DO
         PasTri[I,J] := 0.0;
      END;
   END;
   PasTri[0,0] := 1.0; PasTri[0,1] := -4.0; PasTri[0,2] := 6.0;
   PasTri[0,3] := -4.0; PasTri[0,4] := 1.0;
   PasTri[1,1] := 1.0; PasTri[1,2] := -3.0; PasTri[1,3] := 3.0;
   PasTri[1,4] := -1.0;
   PasTri[2,2] := 1.0; PasTri[2,3] := -2.0; PasTri[2,4] := 1.0;
   PasTri[3,3] := 1.0; PasTri[3,4] := -1.0;
   PasTri[4,4] := 1.0;
END MakePasTri;


PROCEDURE ShiftAll;

(* before the new filter outputs can be calculated it is nessesary *)
(* to shift everything one timestep backwards, that is taken care of *)
(* by this procedure *)

VAR  I, J : INTEGER;

BEGIN
   FOR I := 1 TO OrderOfSystem DO
      FOR J := I TO 1 BY -1 DO
         FiltredRef[I,J] := FiltredRef[I,J-1];
         FiltredAngle[I,J] := FiltredAngle[I,J-1];
  (*     FiltredPos[I,J] := FiltredPos[I,J-1];  *)
      END;
   END;
   FOR I := OrderOfSystem TO 1 BY -1 DO
      Ref[I] := Ref[I-1];
      Angle[I] := Angle[I-1];
 (*   Pos[I] := Pos[I-1];  *)
   END;
END ShiftAll;
```

```
PROCEDURE Filtrate( NewRef, NewAngle : REAL );

(* when this procedure is called the the filter outputs for the *)
(* present time is allmost calculated, however the present value is not *)
(* taken into account, that is done in this procedure *)

VAR  NewPos : REAL;
     I : INTEGER;

BEGIN
   FOR I := 1 TO OrderOfSystem DO
      FiltredRef[I,0] := FiltredRef[I,0] + BKoeffRef[I,0]*NewRef;
      FiltredAngle[I,0] := FiltredAngle[I,0] + BKoeffAngle[I,0]*NewAngle;
 (*   FiltredPos[I,0] := FiltredPos[I,0] + BKoeffPos[I,0]*NewPos; *)
   END;
   Ref[0] := NewRef;
   Angle[0] := NewAngle;
(* Pos[0] := NewPos; *)
END Filtrate;


PROCEDURE AllmostFiltrate;

(* this procedure calculates the filter outputs as they should have *)
(* been if the next input values all was zero, that is it not takes *)
(* the present values into accout *)

VAR  I, J : INTEGER;
     Help1, Help2, Help3 : REAL;

BEGIN
   FOR I := 1 TO OrderOfSystem DO
      Help1 := 0.0;  Help2 := 0.0;  Help3 := 0.0;
      FOR J := 1 TO I DO
         Help1 := BKoeffRef[I,J]*Ref[J] -
                  AKoeffRef[I,J]*FiltredRef[I,J] + Help1;
         Help2 := BKoeffAngle[I,J]*Angle[J] -
                  AKoeffAngle[I,J]*FiltredAngle[I,J] + Help2;
  (*       Help3 := BKoeffPos[I,J]*Pos[J] -
                  AKoeffPos[I,J]*FiltredPos[I,J] + Help3; *)
      END;
      FiltredRef[I,0] := Help1;
      FiltredAngle[I,0] := Help2;
  (*  FiltredPos[I,0] := Help3;  *)
   END;
END AllmostFiltrate;


PROCEDURE SetupForFi( KnownAPar, KnownBPar : KnownArrayType;
                      VAR KnownAnglePol : KnownAPolType;
                      VAR KnownRefPol : KnownBPolType;
                      VAR FiPol : FiPolType;
                      VAR NoParToEst : INTEGER );
```

```
(* to make it possible to calculate Fi, it is nessesary to have a polynomial *)
(* for each element specifying which filter outputs to use, this procedure *)
(* sets up a matrix containing all such polynomials *)
(* It also connects to each element of the fi-vector a vector specifying
   whitch filter outputs that shall be used to calculate the corresponding
   fi element. This vector also contains the coefficients by whitch the
   filter outputs is multiplicated. In the same way there is two vectors
   specifying the left hand side of the estimation equation. *)

VAR  I, J, K, Index, MinStart,
     LineIndex, ColIndex : INTEGER;
     tauHelp : REAL;
     HelpPol, PasTri : PolMatrixType;

BEGIN
   MakePasTri( PasTri );
   LineIndex := 0;
   tauHelp := 1.0;
   FOR I := (MaxOrderOfSystem-OrderOfSystem) TO MaxOrderOfSystem DO
       ColIndex := LineIndex;
       FOR J := I TO MaxOrderOfSystem DO
          HelpPol[LineIndex,ColIndex] := PasTri[I,J]*tauHelp;
          ColIndex := ColIndex + 1;
       END;
       tauHelp := tauHelp*tau;
       LineIndex := LineIndex + 1;
   END;
   FOR I := 0 TO OrderOfSystem DO
      KnownAnglePol[I] := HelpPol[0,I];
   END;
   FOR I := 1 TO OrderOfSystem DO
      KnownRefPol.Pol[I] := 0.0;
   END;
   Index := 0;
   FOR I := 1 TO OrderOfSystem DO
      IF KnownAPar[I].Value THEN
         IF KnownAPar[I].ParValue <> 0.0 THEN
            FOR K := I TO OrderOfSystem DO
               KnownAnglePol[K] := KnownAnglePol[K] +
                                   HelpPol[I,K]*KnownAPar[I].ParValue;
            END;
         END;
      ELSE
         Index := Index + 1;
         FOR K := I TO OrderOfSystem DO
            FiPol[Index].Pol[K] := HelpPol[I,K];
         END;
         FiPol[Index].Start := I;
         FiPol[Index].ParType := APar;
      END;
   END;
   MinStart := OrderOfSystem + 1;
   FOR I := 1 TO OrderOfSystem DO
      IF KnownBPar[I].Value THEN
```

```
        IF KnownBPar[I].ParValue <> 0.0 THEN
          FOR K:= I TO OrderOfSystem DO
            KnownRefPol.Pol[K] := KnownRefPol.Pol[K] -
                                  HelpPol[I,K]*KnownBPar[I].ParValue;
          END;
          IF I < MinStart THEN
            MinStart := I;
          END;
        END;
      ELSE
        Index := Index + 1;
        FOR K := I TO OrderOfSystem DO
          FiPol[Index].Pol[K] := HelpPol[I,K];
        END;
        FiPol[Index].Start := I;
        FiPol[Index].ParType := BPar;
      END;
    END;
    KnownRefPol.Start := MinStart;
    NoParToEst := Index;
END SetupForFi;



PROCEDURE CalcFi( VAR Fi : FiType );

(* using the filter outputs this procedure calculates the vector Fi *)
(* to do is various polynomials is used, matrix FiPol contains these *)
(* polynomials, Fi[0] contains the present left hand side of the equation *)
(* y = Fi*Theta *)

VAR  I, J : INTEGER;
     Help : REAL;

BEGIN
   FOR I := 1 TO NoParToEst DO
      Help := 0.0;
      WITH FiPol[I] DO
        CASE ParType OF
          APar :
            FOR J := Start TO OrderOfSystem DO
              Help := Pol[J]*FiltredAngle[J,0] + Help;
            END|
          BPar :
            FOR J := Start TO OrderOfSystem DO
              Help := Pol[J]*FiltredRef[J,0];
            END;
        END;
      END;
      Fi[I] := Help;
   END;
   Help := 0.0;
   FOR I := 1 TO OrderOfSystem DO
      Help := KnownAnglePol[I]*FiltredAngle[I,0] + Help;
```

```
    END;
    Help := KnownAnglePol[0]*Angle[0] + Help;
    WITH KnownRefPol DO
      FOR I := Start TO OrderOfSystem DO
        Help := Pol[I]*FiltredRef[I,0] + Help;
      END;
    END;
    Fi[0] := Help;
END CalcFi;


PROCEDURE Estimate( VAR Fi : FiType );

(* estimates the coefficients in Theta using recursive least-sqares *)

VAR I, J : CARDINAL;
    Help, Divisor, Epsilon : REAL;

BEGIN
    Help := 0.0;
    FOR I := 1 TO NoParToEst DO
      Help := Theta[ I ]*Fi[ I ] + Help;
    END;
    Epsilon := Fi[ 0 ] - Help;
    IF Abs( Epsilon ) > MinEps THEN
      (* if the new inputs and the old coefficients dont changes the left *)
      (* hand side of the equation more than MinEps, it is not nessesary to *)
      (* update the coefficients *)
      Divisor := Lambda;
      FOR I:= 1 TO NoParToEst DO
        Help := 0.0;
        FOR J := 1 TO NoParToEst DO
          Help := P[ I,J ]*Fi[ J ] + Help;
        END;
        PFi[ I ] := Help;
        Divisor := Fi[ I ]*PFi[ I ] + Divisor;
      END;
      FOR I := 1 TO NoParToEst DO
        FOR J := 1 TO NoParToEst DO
          P[I,J] := (P[I,J] - PFi[I]*PFi[J]/Divisor)/Lambda;
        END;
      END;
      FOR I := 1 TO NoParToEst DO
        Help := 0.0;
        FOR J := 1 TO NoParToEst DO
          Help := P[I,J]*Fi[J] + Help;
        END;
        Theta[I] := Theta[I] + Help*Epsilon;
      END;
    END;
END Estimate;
```

```
(* process *) PROCEDURE Clock;

(* this process make sure that sampling, regulation and filtration is *)
(* done with appropriate timing *)

VAR  Ready : BOOLEAN;

BEGIN
   SetPriority( ClockPrio );
   WHILE TRUE DO
      Wait( StartClock );
      Ready := FALSE;
      WHILE NOT Ready DO
         Wait( ClockMutex );
            Ready := StopClock;
         Signal( ClockMutex );
         WaitTime( Tick );
         Signal( SyncSem );
      END;
   END;
END Clock;


PROCEDURE PIDreg( VAR Ufipid, Fimeasured : REAL );

(* this procedure is an implementation of two PID-controllers coupled *)
(* in series, they are used for controll of the ball and beam *)

VAR  e, Uxpid,
     Xmeasured : REAL;

BEGIN
   Fimeasured := ADIn( AngleChannel );
   Xmeasured := ADIn( PosChannel );
   e := Yrxpid - Xmeasured;
   Ppartxpid := Kxpid*e;
   Ipartxpid := Ipartxpid + Alfaxpid*e;
   Dpartxpid := Betaxpid*Dpartxpid + Gammaxpid*(Yoldxpid-Xmeasured);
   Uxpid := Ppartxpid + Ipartxpid + Dpartxpid;
   IF Uxpid < -1.0 THEN Uxpid := -1.0; END;
   IF Uxpid >  1.0 THEN Uxpid :=  1.0; END;
   e := Uxpid - Fimeasured;
   Ppartfipid := Kfipid*e;
(*   Ipartfipid := Ipartfipid + Alfafipid*e;   *)
   Dpartfipid := Betafipid*Dpartfipid + Gammafipid*(Yoldfipid-Fimeasured);
   Ufipid := Ppartfipid + Dpartfipid;
   IF Ufipid < -1.0 THEN Ufipid := -1.0; END;
   IF Ufipid >  1.0 THEN Ufipid :=  1.0; END;
   DAOut( RefChannel, Ufipid );
   Yoldxpid := Xmeasured;
   Yoldfipid := Fimeasured;
   Ipartxpid := Uxpid - Ppartxpid - Dpartxpid;
END PIDreg;
```

```
(* PROCESS *) PROCEDURE Regulate;

(* this process filtrates, handles the controll and sees to it that *)
(* the estimation is carried out, that is gives the estimating process *)
(* new values *)

VAR  RunCount, EstCount, InitCount : INTEGER;
     Ready : BOOLEAN;
     Fi : FiType;
     NewRef, NewAngle : REAL;

BEGIN
   SetPriority( RegulatePrio );
   WHILE TRUE DO
      Wait( StartRegulate );
      Ready := FALSE;
      InitCount := round( 4.0*tau/h );
      RunCount := 400*EstTime;
(*    EstTime := 0;   *)
      EstCount := EstTime;
      WHILE InitCount > 0 DO
         Wait( SyncSem );
         PIDreg( NewRef, NewAngle );
         Filtrate( NewRef, NewAngle );
         ShiftAll;
         AllmostFiltrate;
         InitCount := InitCount - 1;
      END;
      WHILE NOT Ready DO
         IF RunCount = 0 THEN
            Ready := TRUE;
         ELSE
            Wait( SyncSem );
            PIDreg( NewRef, NewAngle );
            Filtrate( NewRef, NewAngle );
            (*   *)
            IF EstCount = 0 THEN
               CalcFi( Fi );
               Wait( MonitorFi );
                  FiMon := Fi;
               Signal( MonitorFi );
               Signal( NewFi );
               EstCount := EstTime;
            ELSE
               EstCount := EstCount - 1;
               RunCount := RunCount - 1;
            END;
            ShiftAll;
            AllmostFiltrate;
            Wait( RegulateMutex );
               Ready := StopRegulate;
            Signal( RegulateMutex );
         END;
      END;
```

```
      Wait( ClockMutex );
         StopClock := TRUE;     (* stops the clock *)
      Signal( ClockMutex );
   END;
END Regulate;


(* process *) PROCEDURE TestEstim;

VAR  Fi : FiType;
     I : INTEGER;
     ThisPlot : BOOLEAN;

BEGIN
   SetPriority( EstPrio );
   ThisPlot := FALSE;
   WHILE TRUE DO
      Wait( NewFi );
      Wait( MonitorFi );
         Fi := FiMon;
      Signal( MonitorFi );
      Estimate( Fi );
      IF ThisPlot THEN
         IF NoPointsToPlot < 300 THEN    (* MaxNoPointsToPlot *)
            NoPointsToPlot := NoPointsToPlot + 1;
            FOR I := 1 TO NoParToEst DO
               PlotPoints[ NoPointsToPlot, I ] := Theta[I];
            END;
         END;
         ThisPlot := FALSE;
      ELSE
         ThisPlot := TRUE;
      END;
   END;
END TestEstim;


BEGIN
END Running.
```

IMPLEMENTATION MODULE Filtin;

```
FROM GEMin IMPORT RequestMouse, ShowCursor, ReadString;

FROM GEMout IMPORT Text, FillRectangle;

FROM GEMset IMPORT VirtualScreen, Shutdown, SetWindow, SetViewPort,
                   SetTextColor, SetFillColor;

FROM GEMtypes IMPORT handletype, point, colortype, modetype, linetype,
                     markertype, buttontype;
```

```
FROM Strings IMPORT Delete, Insert, CompareStr, Assign, Copy;

FROM ConvReal IMPORT StringToReal, RealToString;

FROM MathLib IMPORT float;

FROM NumberConversion IMPORT StringToInt, IntToString;

FROM CalcFilt IMPORT TustinApprox, ConstInpApprox, LinInpApprox;

FROM GlobDef IMPORT ScreenWindow, h, tau, OrderOfSystem,
                    RefOption, AngleOption, PosOption,
                    AKoeffRef, BKoeffRef, AKoeffAngle,
                    BKoeffAngle, AKoeffPos, BKoeffPos,
                    StepType,
                    RefChannel, AngleChannel, PosChannel,
                    OptionType, FiltKoeffType;


CONST  MaxReferenses = 30;
       MaxOrderOfSystem = 4;
       Blanks = '          ';
       Known = '  known    ';
       NotKnown = ' notknown ';

TYPE  StringType = ARRAY[ 0..10 ] OF CHAR;
      InputDataType = ( Natural, Whole, Fraction );
      ReferensDataType = RECORD
         ParP : point;
         ParString : StringType;
         TypeOfPar : InputDataType;
         ParRef : CARDINAL;
      END;
      RefArrayType = ARRAY[ 1..MaxReferenses ] OF ReferensDataType;
      KindType = ( RefKind, AngleKind, PosKind );


VAR  UpLeft, LoRight,
     StopP, MainP, AngleP,
     ReferensP, PositionP,
     TustinP, ConstInpP, LinInpP, OwnP : point;
     Parameters : RefArrayType;
     NoParameters : CARDINAL;
     NextStep : StepType;

(* for comments of the procedures used in all modules connected with
   menues see the module OPCOM.MOD *)

PROCEDURE ResetBackground;

BEGIN
   UpLeft.h := 0.0;     UpLeft.v := 1.0;
   LoRight.h := 1.5;    LoRight.v := 0.0;
   SetFillColor( ScreenWindow, black );
```

```
   FillRectangle( ScreenWindow, UpLeft, LoRight );
END ResetBackground;


PROCEDURE ConvToPoint( Line, Col : INTEGER; VAR P : point );

BEGIN
   P.h := float( Col )*0.01875;
   P.v := 1.0 - ( float( Line )*0.04 );
END ConvToPoint;


PROCEDURE OutText( Line, Col : INTEGER; String : ARRAY OF CHAR );

VAR  P : point;

BEGIN
   SetTextColor( ScreenWindow, red );
   ConvToPoint( Line, Col, P );
   Text( ScreenWindow, P, String );
END OutText;



PROCEDURE DeletePar( P : point );

VAR  UL, LR : point;

BEGIN
   SetFillColor( ScreenWindow, black );
   UL.h := P.h;
   UL.v := P.v + 0.04;
   LR.h := P.h + 0.1875;
   LR.v := P.v - 0.01;
   FillRectangle( ScreenWindow, UL, LR );
END DeletePar;



PROCEDURE InitParInput( Line, Col : INTEGER;
                        VAR Referens : ReferensDataType;
                        InitString : StringType );


BEGIN
   WITH Referens DO
      ConvToPoint( Line, Col, ParP );
      Delete( ParString, 0, 11 );
      Copy( InitString, 0, 9, ParString );
      Insert( CHR(0), ParString, 10 );
   END;
END InitParInput;
```

```
PROCEDURE InitText;

VAR  P : point;

BEGIN
   OutText( 2, 3, 'THIS IS A TEST FOR THE ONE OF THE FILTER MENUES' );
END InitText;


PROCEDURE InitData( VAR Parameters : RefArrayType;
                    VAR NoParameters : CARDINAL );

BEGIN
   NoParameters := NoParameters + 1;
   RealToString( tau, Parameters[NoParameters].ParString, 10 );
   InitParInput( 3, 30, Parameters[NoParameters],
                 Parameters[NoParameters].ParString );
END InitData;


PROCEDURE InitParameters( VAR Parameters : RefArrayType;
                          VAR NoParameters : CARDINAL;
                          VAR AKoeff, BKoeff : FiltKoeffType );

VAR  I, J, StartLine, StartCol : INTEGER;
     ThisIndex : CARDINAL;

BEGIN
   ThisIndex := 0;
   StartLine := 4;
   StartCol := 3;
   FOR I := 1 TO OrderOfSystem DO
      FOR J := 0 TO I DO
         ThisIndex := ThisIndex + 1;
         RealToString( BKoeff[I,J], Parameters[ThisIndex].ParString, 10 );
         InitParInput( StartLine + 4*(I-1), StartCol + 16*J,
                       Parameters[ThisIndex], Parameters[ThisIndex].ParString);
         ThisIndex := ThisIndex + 1;
         RealToString( AKoeff[I,J], Parameters[ThisIndex].ParString, 10 );
         InitParInput( StartLine + 4*(I-1) + 2, StartCol + 16*J,
                       Parameters[ThisIndex], Parameters[ThisIndex].ParString);
      END;
   END;
   NoParameters := ThisIndex;
END InitParameters;


PROCEDURE InitSelectPar( Line, Col : INTEGER; VAR ParP : point;
                         String : ARRAY OF CHAR );

BEGIN
   ConvToPoint( Line, Col, ParP );
```

```
   OutText( Line, Col, String );
END InitSelectPar;


PROCEDURE InitSelectOpt;

BEGIN
   InitSelectPar( 3, 60, TustinP, '  TUSTIN  ' );
   InitSelectPar( 5, 60, ConstInpP, ' CONST.IN ' );
   InitSelectPar( 7, 60, LinInpP, '  LIN.IN  ' );
   InitSelectPar( 9, 60, OwnP, '   OWN     ' );
END InitSelectOpt;


PROCEDURE ShowOption( Opt : OptionType );

BEGIN
   SetTextColor( ScreenWindow, blue );
   CASE Opt OF
      Tustin :
         InitSelectPar( 3, 60, TustinP, '  TUSTIN  ' )|
      ConstIn :
         InitSelectPar( 5, 60, ConstInpP, ' CONST.IN ' )|
      LinIn :
         InitSelectPar( 7, 60, LinInpP, '  LIN.IN  ' )|
      Own :
         InitSelectPar( 9, 60, OwnP, '   OWN     ' );
   END;
   SetTextColor( ScreenWindow, red );
END ShowOption;



PROCEDURE InitRefSelect;
(* Defines the menues that are possible to choose from the REFERENCE
   menue *)

BEGIN
   InitSelectPar( 24, 3, StopP, '   STOP    ' );
   InitSelectPar( 24, 17, MainP, '   MAIN    ' );
   InitSelectPar( 24, 31, AngleP, '  ANGLE   ' );
(* InitSelectPar( 24, 45, PositionP, ' POSITION ' ); *)
   InitSelectOpt;
   ShowOption( RefOption );
END InitRefSelect;



PROCEDURE InitAngleSelect;
(* Defines the menues that are possible to choose from the ANGLE
   menue *)

BEGIN
   InitSelectPar( 24, 3, StopP, '   STOP    ' );
```

```
   InitSelectPar( 24, 17, MainP, '   MAIN    ' );
   InitSelectPar( 24, 31, ReferensP, ' REFERENS ' );
(* InitSelectPar( 24, 45, PositionP, ' POSITION ' ); *)
   InitSelectOpt;
   ShowOption( AngleOption );
END InitAngleSelect;



PROCEDURE InitPosSelect;
(* This procedure is not used in the current implementation *)
BEGIN
   InitSelectPar( 24, 3, StopP, '   STOP    ' );
   InitSelectPar( 24, 17, MainP, '   MAIN    ' );
   InitSelectPar( 24, 31, ReferensP, ' REFERENS ' );
   InitSelectPar( 24, 45, AngleP, '  ANGLE    ' );
   InitSelectOpt;
   ShowOption( PosOption );
END InitPosSelect;



PROCEDURE DisplayParameter( Referens : ReferensDataType );

BEGIN
   WITH Referens DO
      Text( ScreenWindow, ParP, ParString );
   END;
END DisplayParameter;


PROCEDURE DisplayAllPar( VAR Parameters : RefArrayType;
                         NoParameters : CARDINAL );
VAR  I : CARDINAL;

BEGIN
   SetTextColor( ScreenWindow, red );
   FOR I := 1 TO NoParameters DO
      DisplayParameter( Parameters[ I ] );
   END;
END DisplayAllPar;



PROCEDURE Input( VAR Parameters : RefArrayType; Index : CARDINAL );

VAR   InString, NullString : StringType;

BEGIN
   WITH Parameters[ Index ] DO
      SetTextColor( ScreenWindow, blue );
      Delete( InString, 0, 10 );
      Delete( NullString, 0, 10 );
```

```
      Copy( ' ', 0, 1, InString );
      Copy( ' ', 0, 1, NullString );
(*    WITH Referens DO    *)
        Text( ScreenWindow, ParP, Blanks );
        ReadString( ScreenWindow, ParP, InString );
        OutText( 22, 30, InString );
        IF CompareStr( NullString, InString ) <> 0 THEN
          Insert( CHR(0), InString, 10 );
          SetTextColor( ScreenWindow, red );
          Delete( ParString, 0, 11 );
          Assign( InString, ParString );
          Text( ScreenWindow, ParP, ParString );
        ELSE
          SetTextColor( ScreenWindow, red );
          Text( ScreenWindow, ParP, ParString );
        END;
(*    END;     *)
  END;
END Input;


PROCEDURE Detect( P, InP : point ): BOOLEAN;

BEGIN
  IF (InP.h > P.h) AND (InP.h < P.h + 0.1875) AND
     (InP.v > P.v) AND (InP.v < P.v + 0.04) THEN
     RETURN TRUE;
  ELSE
     RETURN FALSE;
  END;
END Detect;


PROCEDURE DecodeRefPar( VAR Parameters : RefArrayType;
                        Index : CARDINAL );

VAR  IntValue : INTEGER;
     RealValue : REAL;
     Done : BOOLEAN;
     Pos : CARDINAL;

BEGIN
  Pos := 0;
  IntValue := 0;
  Done := FALSE;
  WITH Parameters[ Index ] DO
    CASE Index OF
      1 : StringToReal( ParString, Pos, RealValue );
          tau := RealValue;
          RealToString( tau, ParString, 10 );
          DisplayParameter( Parameters[ Index ] );
          OutText( 13, 7, ParString );
      ELSE ;
```

```
      END;
   END;
END DecodeRefPar;


PROCEDURE DecodeAnglePar( VAR Parameters : RefArrayType;
                          Index : CARDINAL );

VAR  IntValue : INTEGER;
     RealValue : REAL;
     Done : BOOLEAN;
     Pos : CARDINAL;

BEGIN
   Pos := 0;
   IntValue := 0;
   Done := FALSE;
   WITH Parameters[ Index ] DO
      CASE Index OF
         1 : StringToReal( ParString, Pos, RealValue );
             tau := RealValue;
             RealToString( tau, ParString, 10 );
             DisplayParameter( Parameters[ Index ] );
             OutText( 13, 7, ParString );
         ELSE ;
      END;
   END;
END DecodeAnglePar;


PROCEDURE DecodePosPar( VAR Parameters : RefArrayType;
                        Index : CARDINAL );
(* This procedure is not used in the current implementation *)

VAR  IntValue : INTEGER;
     RealValue : REAL;
     Done : BOOLEAN;
     Pos : CARDINAL;

BEGIN
   Pos := 0;
   IntValue := 0;
   Done := FALSE;
   WITH Parameters[ Index ] DO
      CASE Index OF
         1 : StringToReal( ParString, Pos, RealValue );
             tau := RealValue;
             RealToString( tau, ParString, 10 );
             DisplayParameter( Parameters[ Index ] );
             OutText( 13, 7, ParString );
         ELSE ;
      END;
   END;
END DecodePosPar;
```

```
PROCEDURE DetRefSelect( InP : point; VAR Ready : BOOLEAN;
                        VAR NextStep : StepType );
(* Tests if another menue has been choosen from the REFERENCE menue *)

BEGIN
   IF Detect( StopP, InP ) THEN
      Ready := TRUE;
      NextStep := Stop;
   ELSIF Detect( MainP, InP ) THEN
      Ready := TRUE;
      NextStep := MainWindow;
   ELSIF Detect( AngleP, InP ) THEN
      Ready := TRUE;
      NextStep := AngleWindow;
(* ELSIF Detect( PositionP, InP ) THEN
      Ready := TRUE;
      NextStep := PositionWindow; *)
   ELSE
      Ready := FALSE;
      NextStep := None;
   END;
END DetRefSelect;



PROCEDURE DetAngSelect( InP : point; VAR Ready : BOOLEAN;
                        VAR NextStep : StepType );
(* Tests if another menue has been choosen from the ANGLE menue *)

BEGIN
   IF Detect( StopP, InP ) THEN
      Ready := TRUE;
      NextStep := Stop;
   ELSIF Detect( MainP, InP ) THEN
      Ready := TRUE;
      NextStep := MainWindow;
   ELSIF Detect( ReferensP, InP ) THEN
      Ready := TRUE;
      NextStep := ReferensWindow;
(* ELSIF Detect( PositionP, InP ) THEN
      Ready := TRUE;
      NextStep := PositionWindow; *)
   ELSE
      Ready := FALSE;
      NextStep := None;
   END;
END DetAngSelect;



PROCEDURE DetPosSelect( InP : point; VAR Ready : BOOLEAN;
                        VAR NextStep : StepType );
(* This procedure is not used in the current implementation *)
```

```
BEGIN
   IF Detect( StopP, InP ) THEN
      Ready := TRUE;
      NextStep := Stop;
   ELSIF Detect( MainP, InP ) THEN
      Ready := TRUE;
      NextStep := MainWindow;
   ELSIF Detect( AngleP, InP ) THEN
      Ready := TRUE;
      NextStep := AngleWindow;
   ELSIF Detect( ReferensP, InP ) THEN
      Ready := TRUE;
      NextStep := ReferensWindow;
   ELSE
      Ready := FALSE;
      NextStep := None;
   END;
END DetPosSelect;


PROCEDURE DetSelectOpt( InP : point; VAR Pressed : BOOLEAN;
                        Kind : KindType );
(* calculates the the coefficients for the choosen type of filter
   approximation, note that in the current implementation *Kind* can
   never be assigned the value *PosKind*. *)

BEGIN
   IF Detect( TustinP, InP ) THEN
      CASE Kind OF
         RefKind :
            IF RefOption <> Tustin THEN
               RefOption := Tustin;
               ShowOption( RefOption );
               TustinApprox( AKoeffRef, BKoeffRef, h, tau );
               InitParameters( Parameters, NoParameters,
                               AKoeffRef, BKoeffRef );
            END|
         AngleKind :
            IF AngleOption <> Tustin THEN
               AngleOption := Tustin;
               ShowOption( AngleOption );
               TustinApprox( AKoeffAngle, BKoeffAngle, h, tau );
               InitParameters( Parameters, NoParameters,
                               AKoeffAngle, BKoeffAngle );
            END|
         PosKind :
            IF PosOption <> Tustin THEN
               PosOption := Tustin;
               ShowOption( PosOption );
               TustinApprox( AKoeffPos, BKoeffPos, h, tau );
               InitParameters( Parameters, NoParameters,
                               AKoeffPos, BKoeffPos );
            END;
      END;
```

```
      Pressed := TRUE;
   ELSIF Detect( ConstInpP, InP ) THEN
      CASE Kind OF
         RefKind :
            IF RefOption <> ConstIn THEN
               RefOption := ConstIn;
               ShowOption( RefOption );
               ConstInpApprox( AKoeffRef, BKoeffRef, h, tau );
               InitParameters( Parameters, NoParameters,
                               AKoeffRef, BKoeffRef );
            END|
         AngleKind :
            IF AngleOption <> ConstIn THEN
               AngleOption := ConstIn;
               ShowOption( AngleOption );
               ConstInpApprox( AKoeffAngle, BKoeffAngle, h, tau );
               InitParameters( Parameters, NoParameters,
                               AKoeffAngle, BKoeffAngle );
            END|
         PosKind :
            IF PosOption <> ConstIn THEN
               PosOption := ConstIn;
               ShowOption( PosOption );
               ConstInpApprox( AKoeffPos, BKoeffPos, h, tau );
               InitParameters( Parameters, NoParameters,
                               AKoeffPos, BKoeffPos );
            END;
      END;
      Pressed := TRUE;
   ELSIF Detect( LinInpP, InP ) THEN
      CASE Kind OF
         RefKind :
            IF RefOption <> LinIn THEN
               RefOption := LinIn;
               ShowOption( RefOption );
               LinInpApprox( AKoeffRef, BKoeffRef, h, tau );
               InitParameters( Parameters, NoParameters,
                               AKoeffRef, BKoeffRef );
            END|
         AngleKind :
            IF AngleOption <> LinIn THEN
               AngleOption := LinIn;
               ShowOption( AngleOption );
               LinInpApprox( AKoeffAngle, BKoeffAngle, h, tau );
               InitParameters( Parameters, NoParameters,
                               AKoeffAngle, BKoeffAngle );
            END|
         PosKind :
            IF PosOption <> LinIn THEN
               PosOption := LinIn;
               ShowOption( PosOption );
               LinInpApprox( AKoeffPos, BKoeffPos, h, tau );
               InitParameters( Parameters, NoParameters,
                               AKoeffPos, BKoeffPos );
```

```
            END;
        END;
        Pressed := TRUE;
    ELSIF Detect( OwnP, InP ) THEN
        CASE Kind OF (* however the coefficients is never updated *)
            RefKind :
                RefOption := Own|
            AngleKind :
                AngleOption := Own|
            PosKind :
                PosOption := Own;
        END;
        ShowOption( Own );
        Pressed := TRUE;
    ELSE
        Pressed := FALSE;
    END;
END DetSelectOpt;


PROCEDURE FiltRefMain( VAR NextStep : StepType );
(* Main procedure for the REFERENCE menue *)

VAR   InP : point;
      Button : buttontype;
      Ready, Pressed : BOOLEAN;
      CIndex : CARDINAL;
      Index  : INTEGER;

BEGIN
    ResetBackground;
    InitText;
    InitParameters( Parameters, NoParameters, AKoeffRef, BKoeffRef );
    DisplayAllPar( Parameters, NoParameters );
    InitRefSelect;
    SetTextColor( ScreenWindow, red );
    Ready := FALSE;
    WHILE NOT Ready DO
        RequestMouse( ScreenWindow, InP, Button );
        DetRefSelect( InP, Ready, NextStep );
        IF Ready THEN
        ELSE
            DetSelectOpt( InP, Pressed, RefKind );
            IF NOT Pressed THEN
                CIndex := 1;
                WHILE NOT Pressed AND (CIndex <= NoParameters) DO
                    IF Detect( Parameters[CIndex].ParP, InP ) THEN
                        Pressed := TRUE;
                    ELSE
                        CIndex := CIndex + 1;
                    END;
                END;
                IF Pressed THEN
                    Input( Parameters, CIndex );
```

```
            DecodeRefPar( Parameters, CIndex );
          END;
        ELSE
          DisplayAllPar( Parameters, NoParameters );
        END;
      END;
    END;
END FiltRefMain;



PROCEDURE FiltAngleMain( VAR NextStep : StepType );
(* Main procedure for the ANGLE menue *)

VAR   InP : point;
      Button : buttontype;
      Ready, Pressed : BOOLEAN;
      CIndex : CARDINAL;
      Index  : INTEGER;

BEGIN
   ResetBackground;
   InitText;
   InitParameters( Parameters, NoParameters, AKoeffAngle, BKoeffAngle );
   DisplayAllPar( Parameters, NoParameters );
   InitAngleSelect;
   SetTextColor( ScreenWindow, red );
   Ready := FALSE;
   WHILE NOT Ready DO
      RequestMouse( ScreenWindow, InP, Button );
      DetAngSelect( InP, Ready, NextStep );
      IF Ready THEN
      ELSE
         DetSelectOpt( InP, Pressed, AngleKind );
         IF NOT Pressed THEN
            CIndex := 1;
            WHILE NOT Pressed AND (CIndex <= NoParameters) DO
               IF Detect( Parameters[CIndex].ParP, InP ) THEN
                  Pressed := TRUE;
               ELSE
                  CIndex := CIndex + 1;
               END;
            END;
            IF Pressed THEN
               Input( Parameters, CIndex );
               DecodeAnglePar( Parameters, CIndex );
            END;
         ELSE
            DisplayAllPar( Parameters, NoParameters );
         END;
      END;
   END;
END FiltAngleMain;
```

```
PROCEDURE FiltPosMain( VAR NextStep : StepType );
(* This procedure is not used in the current implementation *)
VAR   InP : point;
      Button : buttontype;
      Ready, Pressed : BOOLEAN;
      CIndex : CARDINAL;
      Index  : INTEGER;

BEGIN
   ResetBackground;
   InitText;
   InitParameters( Parameters, NoParameters, AKoeffPos, BKoeffPos );
   DisplayAllPar( Parameters, NoParameters );
   InitPosSelect;
   SetTextColor( ScreenWindow, red );
   Ready := FALSE;
   WHILE NOT Ready DO
      RequestMouse( ScreenWindow, InP, Button );
      DetPosSelect( InP, Ready, NextStep );
      IF Ready THEN
      ELSE
         DetSelectOpt( InP, Pressed, PosKind );
         IF NOT Pressed THEN
            CIndex := 1;
            WHILE NOT Pressed AND (CIndex <= NoParameters) DO
               IF Detect( Parameters[CIndex].ParP, InP ) THEN
                  Pressed := TRUE;
               ELSE
                  CIndex := CIndex + 1;
               END;
            END;
            IF Pressed THEN
               Input( Parameters, CIndex );
               DecodePosPar( Parameters, CIndex );
            END;
         ELSE
            DisplayAllPar( Parameters, NoParameters );
         END;
      END;
   END;
END FiltPosMain;


BEGIN
END Filtin.
```

IMPLEMENTATION MODULE EstIn;

```
(* This module contains the procedures that handles the ESTIMATE menue.*)

FROM GEMin IMPORT RequestMouse, ShowCursor, ReadString;

FROM GEMout IMPORT Text, FillRectangle;
```

```
FROM GEMset IMPORT VirtualScreen, Shutdown, SetWindow, SetViewPort,
                   SetTextColor, SetFillColor;

FROM GEMtypes IMPORT handletype, point, colortype, modetype, linetype,
                     markertype, buttontype;

FROM Strings IMPORT Delete, Insert, CompareStr, Assign, Copy;

FROM ConvReal IMPORT StringToReal, RealToString;

FROM MathLib IMPORT float;

FROM NumberConversion IMPORT StringToInt, IntToString;

FROM CalcFilt IMPORT TustinApprox, ConstInpApprox;

FROM Running IMPORT SetupForFi, Theta;

FROM GlobDef IMPORT ScreenWindow, h, tau, OrderOfSystem,
                    NoParToEst, P, VariansType, FiPol,
                    Lambda, MinEps,
                    KnownAPar, KnownBPar, KnownAnglePol, KnownRefPol,
                    RefOption, AngleOption, PosOption,
                    AKoeffRef, BKoeffRef, AKoeffAngle,
                    BKoeffAngle, AKoeffPos, BKoeffPos,
                    OptionType, FiltKoeffType, StepType;


CONST  MaxReferenses = 30;
       MaxOrderOfSystem = 4;
       Blanks = '          ';
       Known = '  known    ';
       NotKnown = ' notknown ';

TYPE  StringType = ARRAY[ 0..10 ] OF CHAR;
      InputDataType = ( Natural, Whole, Fraction );
      ReferensDataType = RECORD
         ParP : point;
         ParString : StringType;
         TypeOfPar : InputDataType;
         ParRef : CARDINAL;
      END;
      RefArrayType = ARRAY[ 1..MaxReferenses ] OF ReferensDataType;


VAR  UpLeft, LoRight,
     StopP, MainP, POutP : point;
     Parameters : RefArrayType;
     NoParameters : CARDINAL;

(* For info on the procedures used by all menue modules see the module
   OPCOM.MOD *)

PROCEDURE ResetBackground;
```

```
BEGIN
   UpLeft.h := 0.0;     UpLeft.v := 1.0;
   LoRight.h := 1.5;    LoRight.v := 0.0;
   SetFillColor( ScreenWindow, black );
   FillRectangle( ScreenWindow, UpLeft, LoRight );
END ResetBackground;


PROCEDURE ConvToPoint( Line, Col : INTEGER; VAR P : point );

BEGIN
   P.h := float( Col )*0.01875;
   P.v := 1.0 - ( float( Line )*0.04 );
END ConvToPoint;


PROCEDURE OutText( Line, Col : INTEGER; String : ARRAY OF CHAR );

VAR  P : point;

BEGIN
   SetTextColor( ScreenWindow, red );
   ConvToPoint( Line, Col, P );
   Text( ScreenWindow, P, String );
END OutText;



PROCEDURE DeletePar( P : point );

VAR  UL, LR : point;

BEGIN
   SetFillColor( ScreenWindow, black );
   UL.h := P.h;
   UL.v := P.v + 0.04;
   LR.h := P.h + 0.1875;
   LR.v := P.v - 0.01;
   FillRectangle( ScreenWindow, UL, LR );
END DeletePar;



PROCEDURE InitParInput( Line, Col : INTEGER;
                        VAR Referens : ReferensDataType;
                        InitString : StringType );


BEGIN
   WITH Referens DO
      ConvToPoint( Line, Col, ParP );
      Delete( ParString, 0, 11 );
      Copy( InitString, 0, 9, ParString );
      Insert( CHR(0), ParString, 10 );
```

```
   END;
END InitParInput;


PROCEDURE InitText;
(* clear *)

VAR  P : point;

BEGIN
   OutText( 2, 3, 'THIS IS A TEST FOR EST WINDOW' );
   OutText( 3, 3, 'forgetting faktor Lambda' );
   OutText( 3, 53, 'MinEps' );
   OutText( 10, 21 , 'parameter estimates' );
END InitText;


PROCEDURE OutEstValues;
(* Writes to the screen the values of the estimated parameters. *)

VAR  I, J, StartLine, StartCol : INTEGER;
     ThisIndex : CARDINAL;
     HelpString : StringType;
     Help : REAL;

BEGIN
   Help := 0.0;
   StartLine := 12;
   StartCol := 21;
   ThisIndex := 0;
   FOR I := 1 TO OrderOfSystem DO
      IF NOT KnownAPar[I].Value THEN
         ThisIndex := ThisIndex + 1;
         Help := -Theta[ThisIndex];
         RealToString( Help, HelpString, 10 );
         OutText( StartLine + 2, StartCol + 15*(I-1), HelpString );
      ELSE
         OutText( StartLine + 2, StartCol + 15*(I-1), KnownAPar[I].ParString );
      END;
   END;
   FOR I := 1 TO OrderOfSystem DO
      IF NOT KnownBPar[I].Value THEN
         ThisIndex := ThisIndex + 1;
         RealToString( Theta[ThisIndex], HelpString, 10 );
         OutText( StartLine, StartCol + 15*(I-1), HelpString );
      ELSE
         OutText( StartLine, StartCol + 15*(I-1), KnownBPar[I].ParString );
      END;
   END;
END OutEstValues;


PROCEDURE InitData( VAR Parameters : RefArrayType;
```

```
                    VAR NoParameters : CARDINAL );
(* Initiates the parameters Lambda and MinEps. *)

BEGIN
   NoParameters := NoParameters + 1;
   RealToString( Lambda, Parameters[NoParameters].ParString, 10 );
   InitParInput( 3, 30, Parameters[NoParameters],
                 Parameters[NoParameters].ParString );
   NoParameters := NoParameters + 1;
   RealToString( MinEps, Parameters[NoParameters].ParString, 10 );
   InitParInput( 3, 65, Parameters[NoParameters],
                 Parameters[NoParameters].ParString );
END InitData;


PROCEDURE InitParameters( VAR Parameters : RefArrayType;
                          VAR NoParameters : CARDINAL;
                          VAR P : VariansType );
(* Writes to the screen the parameters of this menue, that is:
   diagonal elements of the P-matrix, Lambda, MinEps, Initial values for the
   estimates and the estimated values. *)

VAR  I, J, StartLine, StartCol : INTEGER;
     ThisIndex : CARDINAL;

BEGIN
   ThisIndex := 0;
   StartLine := 5;
   StartCol := 4;
   FOR I := 1 TO NoParToEst DO
      ThisIndex := ThisIndex + 1;
      RealToString( P[I,I], Parameters[ThisIndex].ParString, 10 );
      InitParInput( StartLine + 2*(I-1), StartCol,
                    Parameters[ThisIndex], Parameters[ThisIndex].ParString);
   END;
   StartCol := 21;
   NoParameters := ThisIndex;
   FOR I := 1 TO OrderOfSystem DO
      IF NOT KnownBPar[I].Value THEN
         ThisIndex := ThisIndex + 1;
         RealToString( KnownBPar[I].ParValue,
                       Parameters[ThisIndex].ParString, 10 );
         InitParInput( StartLine, StartCol + 15*(I-1),
                       Parameters[ThisIndex], Parameters[ThisIndex].ParString);
      ELSE
         OutText( StartLine, StartCol + 15*(I-1), KnownBPar[I].ParString );
      END;
   END;
   FOR I := 1 TO OrderOfSystem DO
      IF NOT KnownAPar[I].Value THEN
         ThisIndex := ThisIndex + 1;
         RealToString( KnownAPar[I].ParValue,
                       Parameters[ThisIndex].ParString, 10 );
         InitParInput( StartLine + 2, StartCol + 15*(I-1),
```

```
                    Parameters[ThisIndex], Parameters[ThisIndex].ParString);
      ELSE
         OutText( StartLine + 2, StartCol + 15*(I-1), KnownAPar[I].ParString );
      END;
   END;
   NoParameters := ThisIndex;
END InitParameters;


PROCEDURE InitSelectPar( Line, Col : INTEGER; VAR ParP : point;
                         String : ARRAY OF CHAR );

BEGIN
   ConvToPoint( Line, Col, ParP );
   OutText( Line, Col, String );
END InitSelectPar;


PROCEDURE InitEstSelect;
(* Defines the selectable menues from the ESTIMATE module. *)

BEGIN
   InitSelectPar( 24, 3, StopP, '   STOP   ' );
   InitSelectPar( 24, 17, MainP, '   MAIN   ' );
   InitSelectPar( 24, 31, POutP, ' P-MATRIX ' );
END InitEstSelect;


PROCEDURE DisplayParameter( Referens : ReferensDataType );

BEGIN
   WITH Referens DO
      Text( ScreenWindow, ParP, ParString );
   END;
END DisplayParameter;


PROCEDURE DisplayAllPar( VAR Parameters : RefArrayType;
                         NoParameters : CARDINAL );
VAR  I : CARDINAL;

BEGIN
   SetTextColor( ScreenWindow, red );
   FOR I := 1 TO NoParameters DO
      DisplayParameter( Parameters[ I ] );
   END;
END DisplayAllPar;


PROCEDURE Input( VAR Parameters : RefArrayType; Index : CARDINAL );
```

```
VAR   InString, NullString : StringType;

BEGIN
   WITH Parameters[ Index ] DO
      SetTextColor( ScreenWindow, blue );
      Delete( InString, 0, 10 );
      Delete( NullString, 0, 10 );
      Copy( ' ', 0, 1, InString );
      Copy( ' ', 0, 1, NullString );
(*    WITH Referens DO    *)
         Text( ScreenWindow, ParP, Blanks );
         ReadString( ScreenWindow, ParP, InString );
         OutText( 22, 30, InString );
         IF CompareStr( NullString, InString ) <> 0 THEN
            Insert( CHR(0), InString, 10 );
            SetTextColor( ScreenWindow, red );
            Delete( ParString, 0, 11 );
            Assign( InString, ParString );
            Text( ScreenWindow, ParP, ParString );
         ELSE
            SetTextColor( ScreenWindow, red );
            Text( ScreenWindow, ParP, ParString );
         END;
(*    END;     *)
   END;
END Input;



PROCEDURE Detect( P, InP : point ): BOOLEAN;

BEGIN
   IF (InP.h > P.h) AND (InP.h < P.h + 0.1875) AND
      (InP.v > P.v) AND (InP.v < P.v + 0.04) THEN
      RETURN TRUE;
   ELSE
      RETURN FALSE;
   END;
END Detect;


PROCEDURE DecodeEstPar( VAR Parameters : RefArrayType;
                            Index : INTEGER );
(* It is possible to update the starting values for the parameters that
   is estimated and the diagonal elements of the P-matrix. *)

VAR  IntValue : INTEGER;
     RealValue : REAL;
     Ready : BOOLEAN;
     Pos : CARDINAL;
     ThisIndex, HelpIndex, I : INTEGER;

BEGIN
   Pos := 0;
```

```
    RealValue := 0.0;
    WITH Parameters[ Index ] DO
      IF (Index >= 1) AND (Index <= NoParToEst) THEN
        StringToReal( ParString, Pos, RealValue );
        P[Index, Index] := RealValue;
        RealToString( P[Index, Index], ParString, 10 );
        DisplayParameter( Parameters[Index] );
      ELSIF (Index > NoParToEst) AND (Index <= 2*NoParToEst) THEN
        Ready := FALSE;
        HelpIndex := Index - NoParToEst;
        ThisIndex := 0;
        FOR I := 1 TO OrderOfSystem DO
          IF NOT KnownBPar[I].Value THEN
            ThisIndex := ThisIndex + 1;
            IF ThisIndex = HelpIndex THEN
              Ready := TRUE;
              StringToReal( ParString, Pos, RealValue );
              KnownBPar[I].ParValue := RealValue;
              RealToString( KnownBPar[I].ParValue, ParString, 10 );
              DisplayParameter( Parameters[Index] );
            END;
          END;
        END;
        IF NOT Ready THEN
          FOR I := 1 TO OrderOfSystem DO
            IF NOT KnownAPar[I].Value THEN
              ThisIndex := ThisIndex + 1;
              IF ThisIndex = HelpIndex THEN
                StringToReal( ParString, Pos, RealValue );
                KnownAPar[I].ParValue := RealValue;
                RealToString( KnownAPar[I].ParValue, ParString, 10 );
                DisplayParameter( Parameters[Index] );
              END;
            END;
          END;
        END;
      ELSIF Index = (2*NoParToEst + 1) THEN
        StringToReal( ParString, Pos, RealValue );
        Lambda := RealValue;
        RealToString( Lambda, ParString, 10 );
        DisplayParameter( Parameters[Index] );
      ELSIF Index = 2*(NoParToEst + 1) THEN
        StringToReal( ParString, Pos, RealValue );
        MinEps := RealValue;
        RealToString( MinEps, ParString, 10 );
        DisplayParameter( Parameters[Index] );
      END;
    END;
  END DecodeEstPar;


PROCEDURE DetEstSelect( InP : point; VAR Ready : BOOLEAN;
                        VAR NextStep : StepType );
```

```
(* Has another menue been choosen -question mark-. *)

BEGIN
   IF Detect( StopP, InP ) THEN
      Ready := TRUE;
      NextStep := Stop;
   ELSIF Detect( MainP, InP ) THEN
      Ready := TRUE;
      NextStep := MainWindow;
   ELSIF Detect( POutP, InP ) THEN
      Ready := TRUE;
      NextStep := POutWindow;
   ELSE
      Ready := FALSE;
      NextStep := None;
   END;
END DetEstSelect;



PROCEDURE EstimateMain( VAR NextStep : StepType );
(* Main procedure for the ESTIMATE menue. *)

VAR   InP : point;
      Button : buttontype;
      Ready, Pressed : BOOLEAN;
      CIndex : CARDINAL;
      Index  : INTEGER;

BEGIN
   ResetBackground;
   SetupForFi( KnownAPar, KnownBPar, KnownAnglePol, KnownRefPol,
               FiPol, NoParToEst );
   (* The placing of this procedure call here, makes it nesessary
      to visit this menue before an estimation run should be made,
      if there has been any change in the system order, tau or the
      configuration of known parameters. *)
   InitText;
   InitParameters( Parameters, NoParameters, P );
   InitData( Parameters, NoParameters );
   DisplayAllPar( Parameters, NoParameters );
   InitEstSelect;
   OutEstValues;
   SetTextColor( ScreenWindow, red );
   Ready := FALSE;
   WHILE NOT Ready DO
      RequestMouse( ScreenWindow, InP, Button );
      DetEstSelect( InP, Ready, NextStep );
      IF Ready THEN
      ELSE
         Pressed := FALSE;
         Index := 1;
         CIndex := 1;
```

```
        WHILE NOT Pressed AND (CIndex <= NoParameters) DO
          IF Detect( Parameters[Index].ParP, InP ) THEN
            Pressed := TRUE;
          ELSE
            Index := Index + 1;
            CIndex := CIndex + 1;
          END;
        END;
        IF Pressed THEN
          Input( Parameters, Index );
          DecodeEstPar( Parameters, Index );
        END;
      END;
    END;
END EstimateMain;



BEGIN
END EstIn.
```

IMPLEMENTATION MODULE DoRun;

```
(* This module contains the procedures that puts the outer process in
   a waiting state. *)

FROM Kernel IMPORT InitSem, Wait, Signal;

FROM GEMin IMPORT RequestMouse;

FROM GEMtypes IMPORT point, buttontype;

FROM Running IMPORT InitForRun;

FROM PBuff IMPORT NoPointsToPlot;

FROM GlobDef IMPORT ScreenWindow, SyncSem, ClockMutex, RegulateMutex,
                    EstTime,
                    StartRegulate, StartClock, StopClock, StopRegulate;

PROCEDURE RunMain;

(* this procedure is called when the operator has decided to *)
(* perform an estimation on a process, that is pressed RUN *)

VAR  DummyP : point;
     DummyButton : buttontype;

BEGIN
   NoPointsToPlot := 0;          (* makes sure that proc estimate gets this *)
   InitForRun;
   InitSem( SyncSem, 0 );
   Wait( ClockMutex );
```

```
      StopClock := FALSE;        (* so the clock proc. not stops at this *)
    Signal( ClockMutex );
    Wait( RegulateMutex );
      StopRegulate := FALSE;     (* so the regulate proc. not stops at this *)
    Signal( RegulateMutex );
    Signal( StartRegulate );     (* starts the regulate process *)
    Signal( StartClock );        (* starts the clock process *)
    RequestMouse( ScreenWindow, DummyP, DummyButton );
     (* by this call the main proc. puts itself in standby *)
     (* while the other processes takes care of the regulation and estimation *)
    Wait( RegulateMutex );
      StopRegulate := TRUE; (* makes sure that the regulate proc has stopped *)

    Signal( RegulateMutex );
END RunMain;

END DoRun.
```

IMPLEMENTATION MODULE Disp;

```
(* This module contains the procedures that handles the DISPLAY menue. *)

FROM GEMin IMPORT RequestMouse, ShowCursor, ReadString;

FROM GEMout IMPORT Text, FillRectangle;

FROM GEMset IMPORT VirtualScreen, Shutdown, SetWindow, SetViewPort,
                   SetTextColor, SetFillColor;

FROM GEMtypes IMPORT handletype, point, colortype, modetype, linetype,
                     markertype, buttontype;

FROM Strings IMPORT Delete, Insert, CompareStr, Assign, Copy;

FROM ConvReal IMPORT StringToReal, RealToString;

FROM MathLib IMPORT float;

FROM NumberConversion IMPORT StringToInt, IntToString;

FROM GlobDef IMPORT OrderOfSystem, ScreenWindow, h, tau,
                    Lambda, MinEps, P,
                    YMinLimit, YMaxLimit,
                    AKoeffRef, BKoeffRef, AKoeffAngle, BKoeffAngle,
                    AKoeffPos, BKoeffPos, RefOption, AngleOption,
                    PosOption, OptionType, FiltKoeffType, StepType,
                    KnownAPar, KnownBPar, KnownRefType, KnownArrayType;


CONST  MaxReferenses = 20;
       MaxOrderOfSystem = 4;
       Blanks = '          ';
       Plot = '   plot   ';
       NoPlot = '  noplot  ';
```

```
TYPE  StringType = ARRAY[ 0..10 ] OF CHAR;
      InputDataType = ( Natural, Whole, Fraction );
      ReferensDataType = RECORD
         ParP : point;
         ParString : StringType;
         TypeOfPar : InputDataType;
         ParRef : CARDINAL;
      END;
      RefArrayType = ARRAY[ 1..MaxReferenses ] OF ReferensDataType;


VAR  UpLeft, LoRight,
     StopP, MainP, GraphP : point;
     Parameters : RefArrayType;
     NoParameters : CARDINAL;
     NextStep : StepType;


(* For info on the procedures used by the menue handling modules see
   the module OPCOM.MOD. *)


PROCEDURE ResetBackground;

BEGIN
   UpLeft.h := 0.0;     UpLeft.v := 1.0;
   LoRight.h := 1.5;    LoRight.v := 0.0;
   SetFillColor( ScreenWindow, black );
   FillRectangle( ScreenWindow, UpLeft, LoRight );
END ResetBackground;


PROCEDURE ConvToPoint( Line, Col : INTEGER; VAR P : point );

BEGIN
   P.h := float( Col )*0.01875;
   P.v := 1.0 - ( float( Line )*0.04 );
END ConvToPoint;


PROCEDURE OutText( Line, Col : INTEGER; String : ARRAY OF CHAR );

VAR  P : point;

BEGIN
   SetTextColor( ScreenWindow, red );
   ConvToPoint( Line, Col, P );
   Text( ScreenWindow, P, String );
END OutText;



PROCEDURE DeletePar( P : point );
```

```
VAR  UL, LR : point;

BEGIN
   SetFillColor( ScreenWindow, black );
   UL.h := P.h;
   UL.v := P.v + 0.04;
   LR.h := P.h + 0.1875;
   LR.v := P.v - 0.01;
   FillRectangle( ScreenWindow, UL, LR );
END DeletePar;



PROCEDURE InitParInput( Line, Col : INTEGER;
                        VAR Referens : ReferensDataType;
                        InitString : StringType );


BEGIN
   WITH Referens DO
      ConvToPoint( Line, Col, ParP );
      Delete( ParString, 0, 11 );
      Copy( InitString, 0, 9, ParString );
      Insert( CHR(0), ParString, 10 );
   END;
END InitParInput;



PROCEDURE InitText;

VAR  P : point;

BEGIN
   OutText( 2, 3, 'THIS IS A TEST FOR DISPWINDOW' );
   OutText( 3, 3, 'Max for Y' );
   OutText( 5, 3, 'Min for Y' );
END InitText;



PROCEDURE InitData( VAR Parameters : RefArrayType;
                    VAR NoParameters : CARDINAL );
(* Initiates the parameters YMaxLimit and YMinLimit. *)

BEGIN
   NoParameters := 1;
   RealToString( YMaxLimit, Parameters[NoParameters].ParString, 10 );
   InitParInput( 3, 30, Parameters[NoParameters],
                 Parameters[NoParameters].ParString );
   NoParameters := NoParameters + 1;
   RealToString( YMinLimit, Parameters[NoParameters].ParString, 10 );
   InitParInput( 5, 30, Parameters[NoParameters],
                 Parameters[NoParameters].ParString );
```

```
END InitData;


PROCEDURE InitSelectPar( Line, Col : INTEGER; VAR ParP : point;
                         String : ARRAY OF CHAR );

BEGIN
   ConvToPoint( Line, Col, ParP );
   OutText( Line, Col, String );
END InitSelectPar;


PROCEDURE InitDispSelect;

(* defines the menues that are possible to choose from the
   DISPLAY menue *)
BEGIN
   InitSelectPar( 24, 3, StopP, '   STOP   ' );
   InitSelectPar( 24, 17, MainP, '   MAIN   ' );
   InitSelectPar( 24, 31, GraphP, '  GRAPH   ' );
END InitDispSelect;



PROCEDURE DisplayParameter( Referens : ReferensDataType );

BEGIN
   WITH Referens DO
      Text( ScreenWindow, ParP, ParString );
   END;
END DisplayParameter;


PROCEDURE DisplayAllPar( VAR Parameters : RefArrayType;
                         NoParameters : CARDINAL );
VAR  I : CARDINAL;

BEGIN
   SetTextColor( ScreenWindow, red );
   FOR I := 1 TO NoParameters DO
      DisplayParameter( Parameters[ I ] );
   END;
END DisplayAllPar;


PROCEDURE DisplayPlotPar( VAR KnownRef : KnownRefType;
                          Line, Col : INTEGER );

BEGIN
   WITH KnownRef DO
      ConvToPoint( Line, Col, ParP );
      IF NOT Value THEN
         Text( ScreenWindow, ParP, PlotString );
```

```
      ELSE
         Text( ScreenWindow, ParP, ParString );
      END;
   END;
END DisplayPlotPar;


PROCEDURE InitPlotPar( OrderOfSystem : INTEGER );

(* makes sure that no parameter is selected to be plotted
   upon start *)
VAR  I : INTEGER;

BEGIN
   FOR I := 1 TO MaxOrderOfSystem DO
      WITH KnownBPar[ I ] DO
         Copy( NoPlot, 0, 10, PlotString );
         ToPlot := FALSE;
      END;
      WITH KnownAPar[ I ] DO
         Copy( NoPlot, 0, 10, PlotString );
         ToPlot := FALSE;
      END;
   END;
END InitPlotPar;


PROCEDURE DisplayPlot( OrderOfSystem : INTEGER );

(* displays whitch parameters in the transfer function it
   it is possible to plot *)

VAR  I, StartLine, StartCol : INTEGER;

BEGIN
   SetTextColor( ScreenWindow, red );
   StartLine := 8;
   StartCol := 2;
   FOR I := 0 TO (OrderOfSystem - 1) DO
      DisplayPlotPar( KnownBPar[I+1], StartLine, StartCol + ( 16*I ) );
   END;
   StartLine := 10;
   FOR I := 0 TO (OrderOfSystem - 1) DO
      DisplayPlotPar( KnownAPar[I+1], StartLine, StartCol + ( 16*I ) );
   END;
END DisplayPlot;



PROCEDURE Input( VAR Parameters : RefArrayType; Index : CARDINAL );

(* reads the selected parameterdata from the keyboard *)
```

```
VAR   InString, NullString : StringType;

BEGIN
   WITH Parameters[ Index ] DO
      SetTextColor( ScreenWindow, blue );
      Delete( InString, 0, 10 );
      Delete( NullString, 0, 10 );
      Copy( ' ', 0, 1, InString );
      Copy( ' ', 0, 1, NullString );
(*    WITH Referens DO    *)
         Text( ScreenWindow, ParP, Blanks );
         ReadString( ScreenWindow, ParP, InString );
         OutText( 22, 30, InString );
         IF CompareStr( NullString, InString ) <> 0 THEN
            Insert( CHR(0), InString, 10 );
            SetTextColor( ScreenWindow, red );
            Delete( ParString, 0, 11 );
            Assign( InString, ParString );
            Text( ScreenWindow, ParP, ParString );
         ELSE
            SetTextColor( ScreenWindow, red );
            Text( ScreenWindow, ParP, ParString );
         END;
(*    END;     *)
   END;
END Input;



PROCEDURE Detect( P, InP : point ): BOOLEAN;

(* checks if the point from the mouse, InP, is placed
   within the square defined by the point P *)

BEGIN
   IF (InP.h > P.h) AND (InP.h < P.h + 0.1875) AND
      (InP.v > P.v) AND (InP.v < P.v + 0.04) THEN
      RETURN TRUE;
   ELSE
      RETURN FALSE;
   END;
END Detect;


PROCEDURE DetDispSelect( InP : point; VAR Ready : BOOLEAN;
                         VAR NextStep : StepType );

(* checks if another menue has been choosen *)

BEGIN
   IF Detect( StopP, InP ) THEN
      Ready := TRUE;
      NextStep := Stop;
```

```
   ELSIF Detect( MainP, InP ) THEN
      Ready := TRUE;
      NextStep := MainWindow;
   ELSIF Detect( GraphP, InP ) THEN
      Ready := TRUE;
      NextStep := GraphWindow;
   ELSE
      Ready := FALSE;
      NextStep := None;
   END;
END DetDispSelect;


PROCEDURE DecodeParameter( VAR Parameters : RefArrayType;
                           Index : CARDINAL );

(* uppdates the actual parameter that has been read from the
   keyboard *)

VAR  IntValue : INTEGER;
     RealValue : REAL;
     Done : BOOLEAN;
     Pos : CARDINAL;

BEGIN
   Pos := 0;
   IntValue := 0;
   Done := FALSE;
   WITH Parameters[ Index ] DO
      CASE Index OF
         1 : StringToReal( ParString, Pos, RealValue );
             YMaxLimit := RealValue;
             RealToString( YMaxLimit, ParString, 10 );
             DisplayParameter( Parameters[ Index ] )|

         2 : StringToReal( ParString, Pos, RealValue );
             YMinLimit := RealValue;
             RealToString( YMinLimit, ParString, 10 );
             DisplayParameter( Parameters[ Index ] );
         ELSE ;
      END;
   END;
END DecodeParameter;



PROCEDURE DispMain( VAR NextStep : StepType);
(* Main procedure for the DISPLAY menue. *)

VAR   InP : point;
      Button : buttontype;
      Ready, Pressed : BOOLEAN;
```

```
    CIndex : CARDINAL;
    Index  : INTEGER;

BEGIN
  ResetBackground;
  InitText;
  InitData( Parameters, NoParameters );
  DisplayAllPar( Parameters, NoParameters );
  InitDispSelect;
  InitPlotPar( OrderOfSystem );
  DisplayPlot( OrderOfSystem );
  SetTextColor( ScreenWindow, red );
  Ready := FALSE;
  WHILE NOT Ready DO
    RequestMouse( ScreenWindow, InP, Button );
    DetDispSelect( InP, Ready, NextStep );
    IF Ready THEN
    ELSE
      Pressed := FALSE;
      CIndex := 1;
      WHILE NOT Pressed AND (CIndex <= NoParameters) DO
        IF Detect( Parameters[CIndex].ParP, InP ) THEN
          Pressed := TRUE;
        ELSE
          CIndex := CIndex + 1;
        END;
      END;
      IF Pressed THEN
        Input( Parameters, CIndex );
        DecodeParameter( Parameters, CIndex );
      ELSE
        Pressed := FALSE;
        Index := 1;
        WHILE NOT Pressed AND (Index <= OrderOfSystem) DO
          IF Detect( KnownBPar[Index].ParP, InP ) THEN
            Pressed := TRUE;
          ELSE
            Index := Index + 1;
          END;
        END;
        IF Pressed THEN
          WITH KnownBPar[Index] DO
            IF NOT Value THEN
              IF NOT ToPlot THEN
                ToPlot := TRUE;
                PlotString := Plot;
                Text( ScreenWindow, ParP, PlotString );
              ELSE
                ToPlot := FALSE;
                PlotString := NoPlot;
                Text( ScreenWindow, ParP, PlotString );
              END;
            END;
          END;
```

```
            ELSE
               Index := 1;
               WHILE NOT Pressed AND (Index <= OrderOfSystem) DO
                  IF Detect( KnownAPar[Index].ParP, InP ) THEN
                     Pressed := TRUE;
                  ELSE
                     Index := Index + 1;
                  END;
               END;
               IF Pressed THEN
                  WITH KnownAPar[Index] DO
                     IF NOT Value THEN
                        IF NOT ToPlot THEN
                           ToPlot := TRUE;
                           PlotString := Plot;
                           Text( ScreenWindow, ParP, PlotString );
                        ELSE
                           ToPlot := FALSE;
                           PlotString := NoPlot;
                           Text( ScreenWindow, ParP, PlotString );
                        END;
                     END;
                  END;
               END;
            END;
         END;
      END;
   END;
END DispMain;


BEGIN
END Disp.
```

IMPLEMENTATION MODULE Graph;

```
FROM GEMin IMPORT RequestMouse;

FROM GEMout IMPORT Text, FillRectangle, PolyLine, PolyMarker;

FROM GEMset IMPORT SetWindow, SetViewPort, SetTextColor, SetFillColor,
                   SetLineType, SetLineWidth, SetLineColor, SetMode,
                   SetMarkerType, SetMarkerHeight, SetMarkerColor;

FROM GEMtypes IMPORT handletype, point, colortype, modetype,
                     linetype, markertype, buttontype;

FROM MathLib IMPORT float;

FROM ConvReal IMPORT RealToString;

FROM PBuff IMPORT MaxNoPoints, PlotPoints, NoPointsToPlot;
```

```
FROM GlobDef IMPORT OrderOfSystem, ScreenWindow, StepType, MaxEstimOrder,
                    YMaxLimit, YMinLimit, EstTime, h,
                    KnownAPar, KnownBPar, KnownRefType, KnownArrayType;


CONST GraphWidth = 1;
      LineWidth = 1;
      MarkerHeight = 1;

TYPE  IndexType = [0..MaxEstimOrder];
      TypeOfPar = ( APar, BPar );
      StringType = ARRAY[0..10] OF CHAR;

VAR   Polygon : ARRAY[0..MaxNoPoints-1] OF point;
      OnScreenUL, OnScreenLR, ScaleUL, ScaleLR, UL, LR,
      StopP, MainP, DispP, UpLeft, LoRight : point;
      XAxis, YAxis : ARRAY[0..1] OF point;
      Min, Max : REAL;


(* for info on the procedures used by all menue modules, see the
   module OPCOM.MOD *)

PROCEDURE ResetBackground;

BEGIN
   UpLeft.h := 0.0;       UpLeft.v := 1.0;
   LoRight.h := 1.5;      LoRight.v := 0.0;
   SetFillColor( ScreenWindow, black );
   FillRectangle( ScreenWindow, UpLeft, LoRight );
END ResetBackground;


PROCEDURE ConvToPoint( Line, Col : INTEGER; VAR P : point );

BEGIN
   P.h := float( Col )*0.01875;
   P.v := 1.0 - ( float( Line )*0.04 );
END ConvToPoint;



PROCEDURE OutText( Line, Col : INTEGER; String : ARRAY OF CHAR );

VAR P : point;

BEGIN
   SetTextColor( ScreenWindow, red );
   ConvToPoint( Line, Col, P );
   Text( ScreenWindow, P, String );
END OutText;
```

```
PROCEDURE InitText;

BEGIN
   OutText( 2, 3, 'THIS IS A TEST FOR GRAPHWINDOW' );
END InitText;


PROCEDURE InitSelectPar( Line, Col : INTEGER; VAR ParP : point;
                         String : ARRAY OF CHAR );

BEGIN
   ConvToPoint( Line, Col, ParP );
   OutText( Line, Col, String );
END InitSelectPar;


PROCEDURE InitGraphSelect;
(* Defines the menues that is selectable from the GRAPH menue *)

BEGIN
   InitSelectPar( 24, 3, StopP, '   STOP   ' );
   InitSelectPar( 24, 17, MainP, '   MAIN   ' );
   InitSelectPar( 24, 31, DispP, '  DISPLAY ' );
END InitGraphSelect;


PROCEDURE InitGraph( Min, Max : REAL );
(* Draws the diagram and the axis to the screen. *)

VAR  HelpString : StringType;
     Help : REAL;

BEGIN
   OnScreenUL.h := 0.0;
   OnScreenUL.v := 1.0;
   OnScreenLR.h := 1.5;
   OnScreenLR.v := 0.2;

   ScaleUL.h := 0.0;
   ScaleUL.v := Max;
   ScaleLR.h := float( NoPointsToPlot );
   ScaleLR.v := Min;

   XAxis[0].h := 0.0;
   XAxis[0].v := 0.0;
   XAxis[1].h := ScaleLR.h;
   XAxis[1].v := 0.0;

   YAxis[0].h := 0.0;
   YAxis[0].v := ScaleLR.v;
   YAxis[1].h := 0.0;
   YAxis[1].v := ScaleUL.v;

   SetViewPort( ScreenWindow, OnScreenUL, OnScreenLR );
```

```
   SetWindow( ScreenWindow, ScaleUL, ScaleLR );

   SetMode( ScreenWindow, transparent );
   SetFillColor( ScreenWindow, yellow );
   SetLineType( ScreenWindow, Solid );
   SetLineWidth( ScreenWindow, LineWidth );
   SetLineColor( ScreenWindow, black );

   FillRectangle( ScreenWindow, ScaleUL, ScaleLR );
   IF Min <= 0.0 THEN
      PolyLine( ScreenWindow, XAxis, 2 );
   END;
   PolyLine( ScreenWindow, YAxis, 2 );

   SetViewPort( ScreenWindow, UpLeft, LoRight );
   SetWindow( ScreenWindow, UpLeft, LoRight );
   RealToString( Max, HelpString, 10 );
   OutText( 1, 1, HelpString );
   RealToString( Min, HelpString, 10 );
   OutText( 20, 1, HelpString );
   Help := 2.0*h*float( NoPointsToPlot*(EstTime+1) );    (* PlottCount *)
   RealToString( Help, HelpString, 10 );
   OutText( 20, 70, HelpString );

   SetViewPort( ScreenWindow, OnScreenUL, OnScreenLR );
   SetWindow( ScreenWindow, ScaleUL, ScaleLR );
END InitGraph;


PROCEDURE DispOnePar( Index, ParIndex : IndexType;
                      Min, Max : REAL; PType : TypeOfPar );

(* Displays one parameter, also makes sure that no two parameters is plotted
   with the same type of color or markers. The zooming function is implemented
   via the vector -Polygon-. *)

VAR  I : INTEGER;
     Help : REAL;

BEGIN
   CASE PType OF
      APar :
         FOR I := 1 TO NoPointsToPlot DO
            Help := -PlotPoints[I,Index];
            IF Help > Max THEN
               Polygon[I-1].v := Max;
            ELSIF Help < Min THEN
               Polygon[I-1].v := Min;
            ELSE
               Polygon[I-1].v := Help;
            END;
            Polygon[I-1].h := float( I-1 );
         END|
      BPar :
```

```
        FOR I := 1 TO NoPointsToPlot DO
          Help := PlotPoints[I,Index];
          IF Help > Max THEN
            Polygon[I-1].v := Max;
          ELSIF Help < Min THEN
            Polygon[I-1].v := Min;
          ELSE
            Polygon[I-1].v := Help;
          END;
          Polygon[I-1].h := float( I-1 );
        END;
    END;
    CASE ParIndex OF
      1 : SetLineType( ScreenWindow, Solid );
          SetLineWidth( ScreenWindow, GraphWidth );
          SetLineColor( ScreenWindow, black );
          PolyLine( ScreenWindow, Polygon, NoPointsToPlot )|

      2 : SetLineType( ScreenWindow, Solid );
          SetLineWidth( ScreenWindow, GraphWidth );
          SetLineColor( ScreenWindow, blue );
          PolyLine( ScreenWindow, Polygon, NoPointsToPlot )|

      3 : SetLineType( ScreenWindow, Solid );
          SetLineWidth( ScreenWindow, GraphWidth );
          SetLineColor( ScreenWindow, red );
          PolyLine( ScreenWindow, Polygon, NoPointsToPlot )|

      4 : SetMarkerType( ScreenWindow, Dot );
          SetMarkerHeight( ScreenWindow, MarkerHeight );
          SetMarkerColor( ScreenWindow, black );
          PolyMarker( ScreenWindow, Polygon, NoPointsToPlot )|

    (* 3 : SetMarkerType( ScreenWindow, plus );
          SetMarkerHeight( ScreenWindow, MarkerHeight );
          SetMarkerColor( ScreenWindow, black );
          PolyMarker( ScreenWindow, Polygon, NoPointsToPlot )|

      4 : SetMarkerType( ScreenWindow, asterisk );
          SetMarkerHeight( ScreenWindow, MarkerHeight );
          SetMarkerColor( ScreenWindow, black );
          PolyMarker( ScreenWindow, Polygon, NoPointsToPlot )|  *)

      5 : SetMarkerType( ScreenWindow, square );
          SetMarkerHeight( ScreenWindow, MarkerHeight );
          SetMarkerColor( ScreenWindow, black );
          PolyMarker( ScreenWindow, Polygon, NoPointsToPlot )|

      6 : SetMarkerType( ScreenWindow, X );
          SetMarkerHeight( ScreenWindow, MarkerHeight );
          SetMarkerColor( ScreenWindow, black );
          PolyMarker( ScreenWindow, Polygon, NoPointsToPlot )|

      7 : SetMarkerType( ScreenWindow, diamond );
```

```
            SetMarkerHeight( ScreenWindow, MarkerHeight );
            SetMarkerColor( ScreenWindow, black );
            PolyMarker( ScreenWindow, Polygon, NoPointsToPlot )|

        8 : SetMarkerType( ScreenWindow, diamond );
            SetMarkerHeight( ScreenWindow, 2*MarkerHeight );
            SetMarkerColor( ScreenWindow, black );
            PolyMarker( ScreenWindow, Polygon, NoPointsToPlot );
    END;
END DispOnePar;


PROCEDURE FindMinMax( VAR Min, Max : REAL );
(* Was originally designed for autoscaling function, however there is some
   very trivial error in the algorithm. *)

VAR  I,J : INTEGER;

BEGIN
(*   Min := 0.0;       -Autoscaling function is not in use for the moment-
   Max := 0.0;
   FOR I := 1 TO NoPointsToPlot DO
      FOR J := 1 TO NoParToEst DO
         IF PlotPoints[I,J] > Max THEN
            Max := PlotPoints[I,J];
         ELSIF PlotPoints[I,J] < Min THEN
            Min := PlotPoints[I,J];
         END;
   END;
   IF Max > YMaxLimit THEN
      Max := YMaxLimit;
   END;
   IF Min < YMinLimit THEN
      Min := YMinLimit;
   END;   *)
   Max := YMaxLimit;
   Min := YMinLimit;
END FindMinMax;


PROCEDURE Detect( P, InP : point ):BOOLEAN;

BEGIN
   IF (InP.h > P.h) AND (InP.h < P.h + 0.1875) AND
      (InP.v > P.v) AND (InP.v < P.v + 0.04) THEN
      RETURN TRUE
   ELSE
      RETURN FALSE;
   END;
END Detect;


PROCEDURE DetGraphSelect( InP : point; VAR Ready : BOOLEAN;
                          VAR NextStep : StepType );
```

```
(* Is another menue requested -question mark- *)

BEGIN
   IF Detect( StopP, InP ) THEN
      Ready := TRUE;
      NextStep := Stop;
   ELSIF Detect( MainP, InP ) THEN
      Ready := TRUE;
      NextStep := MainWindow;
   ELSIF Detect( DispP, InP ) THEN
      Ready := TRUE;
      NextStep := DisplayWindow;
   ELSE
      Ready := FALSE;
      NextStep := None;
   END;
END DetGraphSelect;



PROCEDURE GraphIt( Min, Max : REAL );
(* Takes the history of the estimated parameters graphs it. *)

VAR  Index, ParIndex : IndexType;
     I : INTEGER;

BEGIN
   Index := 0;
   ParIndex := 0;
   FOR I := 1 TO OrderOfSystem DO
      WITH KnownAPar[I] DO
         IF NOT Value THEN
            Index := Index + 1;
            IF ToPlot THEN
               ParIndex := ParIndex + 1;
               DispOnePar( Index, ParIndex, Min, Max, APar );
            END;
         END;
      END;
   END;
   FOR I := 1 TO OrderOfSystem DO
      WITH KnownBPar[I] DO
         IF NOT Value THEN
            Index := Index + 1;
            IF ToPlot THEN
               ParIndex := ParIndex + 1;
               DispOnePar( Index, ParIndex, Min, Max, BPar );
            END;
         END;
      END;
   END;
END GraphIt;
```

```
PROCEDURE GraphMain( VAR NextStep : StepType );
(* Main procedure for the GRAPH menue. *)

VAR  Ready : BOOLEAN;
     InP : point;
     Button : buttontype;

BEGIN
   ResetBackground;
   FindMinMax( Min, Max );
   InitGraph( Min, Max );
   GraphIt( Min, Max );
   UL.h := 0.0;
   UL.v := 1.0;
   LR.h := 1.5;
   LR.v := 0.0;
   SetViewPort( ScreenWindow, UL, LR );
   SetWindow( ScreenWindow, UL, LR );
   InitGraphSelect;
   Ready := FALSE;
   WHILE NOT Ready DO
      RequestMouse( ScreenWindow, InP, Button );
      DetGraphSelect( InP, Ready, NextStep );
   END;
END GraphMain;

END Graph.
```

IMPLEMENTATION MODULE POut;

```
(* This module contains the procedures that handles the printing of
   the covarianse matrix to the screen. *)

FROM GEMin IMPORT RequestMouse, ShowCursor, ReadString;

FROM GEMout IMPORT Text, FillRectangle;

FROM GEMset IMPORT VirtualScreen, Shutdown, SetWindow, SetViewPort,
                   SetTextColor, SetFillColor;

FROM GEMtypes IMPORT handletype, point, colortype, modetype, linetype,
                     markertype, buttontype;

FROM Strings IMPORT Delete, Insert, CompareStr, Assign, Copy;

FROM ConvReal IMPORT StringToReal, RealToString;

FROM MathLib IMPORT float;

FROM NumberConversion IMPORT StringToInt, IntToString;

FROM GlobDef IMPORT ScreenWindow, h, tau, OrderOfSystem,
                    NoParToEst, P, VariansType, FiPol,
                    Lambda, MinEps,
```

```
                KnownAPar, KnownBPar, KnownAnglePol, KnownRefPol,
                RefOption, AngleOption, PosOption,
                AKoeffRef, BKoeffRef, AKoeffAngle,
                BKoeffAngle, AKoeffPos, BKoeffPos,
                OptionType, FiltKoeffType, StepType;


CONST  MaxReferenses = 70;
       MaxOrderOfSystem = 4;
       Blanks = '          ';
       Known = '  known    ';
       NotKnown = ' notknown ';

TYPE  StringType = ARRAY[ 0..7 ] OF CHAR;
      InputDataType = ( Natural, Whole, Fraction );
      ReferensDataType = RECORD
         ParP : point;
         ParString : StringType;
         TypeOfPar : InputDataType;
         ParRef : CARDINAL;
      END;
      RefArrayType = ARRAY[ 1..MaxReferenses ] OF ReferensDataType;


VAR  UpLeft, LoRight,
     StopP, MainP, EstP : point;
     Parameters : RefArrayType;
     NoParameters : CARDINAL;

(* For info on the procedures used by all the menue-modules see
   the module OPCOM.MOD *)

PROCEDURE ResetBackground;

BEGIN
   UpLeft.h := 0.0;    UpLeft.v := 1.0;
   LoRight.h := 1.5;   LoRight.v := 0.0;
   SetFillColor( ScreenWindow, black );
   FillRectangle( ScreenWindow, UpLeft, LoRight );
END ResetBackground;


PROCEDURE ConvToPoint( Line, Col : INTEGER; VAR P : point );

BEGIN
   P.h := float( Col )*0.01875;
   P.v := 1.0 - ( float( Line )*0.04 );
END ConvToPoint;


PROCEDURE OutText( Line, Col : INTEGER; String : ARRAY OF CHAR );

VAR  P : point;
```

```
BEGIN
   SetTextColor( ScreenWindow, red );
   ConvToPoint( Line, Col, P );
   Text( ScreenWindow, P, String );
END OutText;



PROCEDURE DeletePar( P : point );

VAR  UL, LR : point;

BEGIN
   SetFillColor( ScreenWindow, black );
   UL.h := P.h;
   UL.v := P.v + 0.04;
   LR.h := P.h + 0.1875;
   LR.v := P.v - 0.01;
   FillRectangle( ScreenWindow, UL, LR );
END DeletePar;



PROCEDURE InitParInput( Line, Col : INTEGER;
                        VAR Referens : ReferensDataType;
                        InitString : StringType );


BEGIN
   WITH Referens DO
      ConvToPoint( Line, Col, ParP );
      Delete( ParString, 0, 8 );
      Copy( InitString, 0, 6, ParString );
      Insert( CHR(0), ParString, 7 );
   END;
END InitParInput;



PROCEDURE InitText;

VAR  P : point;

BEGIN
   OutText( 2, 3, 'THIS IS A TEST FOR POUT WINDOW' );
(*   OutText( 3, 3, 'forgetting faktor Lambda' );
   OutText( 3, 53, 'MinEps' ); *)
END InitText;



PROCEDURE InitData( VAR Parameters : RefArrayType;
                    VAR NoParameters : CARDINAL );
```

```
BEGIN
END InitData;


PROCEDURE InitParameters( VAR Parameters : RefArrayType;
                         VAR NoParameters : CARDINAL;
                         VAR P : VariansType );

VAR  I, J, StartLine, StartCol : INTEGER;
     ThisIndex : CARDINAL;

BEGIN
   ThisIndex := 0;
   StartLine := 5;
   StartCol := 1;
   FOR I := 1 TO NoParToEst DO
      FOR J := 1 TO NoParToEst DO
         ThisIndex := ThisIndex + 1;
         RealToString( P[I,J], Parameters[ThisIndex].ParString, 7 );
         InitParInput( StartLine + 2*(I-1), StartCol + 10*(J-1),
                       Parameters[ThisIndex], Parameters[ThisIndex].ParString);
      END;
   END;
   NoParameters := ThisIndex;
END InitParameters;



PROCEDURE InitSelectPar( Line, Col : INTEGER; VAR ParP : point;
                         String : ARRAY OF CHAR );

BEGIN
   ConvToPoint( Line, Col, ParP );
   OutText( Line, Col, String );
END InitSelectPar;


PROCEDURE InitPOutSelect;
(* Defines the selectable menues from the POUT menue. *)

BEGIN
   InitSelectPar( 24, 3, StopP, '   STOP   ' );
   InitSelectPar( 24, 17, MainP, '   MAIN   ' );
   InitSelectPar( 24, 31, EstP, ' ESTIMATE ' );
END InitPOutSelect;



PROCEDURE DisplayParameter( Referens : ReferensDataType );

BEGIN
   WITH Referens DO
      Text( ScreenWindow, ParP, ParString );
   END;
```

```
END DisplayParameter;


PROCEDURE DisplayAllPar( VAR Parameters : RefArrayType;
                         NoParameters : CARDINAL );
VAR  I : CARDINAL;

BEGIN
   SetTextColor( ScreenWindow, red );
   FOR I := 1 TO NoParameters DO
      DisplayParameter( Parameters[ I ] );
   END;
END DisplayAllPar;



PROCEDURE Input( VAR Parameters : RefArrayType; Index : CARDINAL );

VAR   InString, NullString : StringType;

BEGIN
   WITH Parameters[ Index ] DO
      SetTextColor( ScreenWindow, blue );
      Delete( InString, 0, 7 );
      Delete( NullString, 0, 7 );
      Copy( ' ', 0, 1, InString );
      Copy( ' ', 0, 1, NullString );
(*    WITH Referens DO    *)
         Text( ScreenWindow, ParP, Blanks );
         ReadString( ScreenWindow, ParP, InString );
         OutText( 22, 30, InString );
         IF CompareStr( NullString, InString ) <> 0 THEN
            Insert( CHR(0), InString, 7 );
            SetTextColor( ScreenWindow, red );
            Delete( ParString, 0, 8 );
            Assign( InString, ParString );
            Text( ScreenWindow, ParP, ParString );
         ELSE
            SetTextColor( ScreenWindow, red );
            Text( ScreenWindow, ParP, ParString );
         END;
(*    END;     *)
   END;
END Input;



PROCEDURE Detect( P, InP : point ): BOOLEAN;

BEGIN
   IF (InP.h > P.h) AND (InP.h < P.h + 0.1875) AND
      (InP.v > P.v) AND (InP.v < P.v + 0.04) THEN
      RETURN TRUE;
```

```
   ELSE
      RETURN FALSE;
   END;
END Detect;


PROCEDURE DecodePOutPar( VAR Parameters : RefArrayType;
                         Index : INTEGER );
(* There is no use for making the elements of the P-matrix manipulatable. *)

VAR  RealValue : REAL;
     Found : BOOLEAN;
     Pos : CARDINAL;
     ThisIndex, I, J : INTEGER;

BEGIN
   Pos := 0;
   RealValue := 0.0;
   I := 0;
   J := 0;
   ThisIndex := 0;
   Found := FALSE;
   WITH Parameters[ Index ] DO
      WHILE (I <= NoParToEst) AND NOT Found DO
         I := I + 1;
         WHILE (J <= NoParToEst) AND NOT Found DO
            J := J + 1;
            ThisIndex := ThisIndex + 1;
            IF ThisIndex = Index THEN
               Found := TRUE;
               StringToReal( ParString, Pos, RealValue );
               P[I,J] := RealValue;
               RealToString( P[I,J], ParString, 8 );
               DisplayParameter( Parameters[Index] );
            END;
         END;
      END;
   END;
END DecodePOutPar;



PROCEDURE DetPOutSelect( InP : point; VAR Ready : BOOLEAN;
                         VAR NextStep : StepType );
(* Has another menue been selected -question mark-. *)

BEGIN
   IF Detect( StopP, InP ) THEN
      Ready := TRUE;
      NextStep := Stop;
   ELSIF Detect( MainP, InP ) THEN
      Ready := TRUE;
      NextStep := MainWindow;
   ELSIF Detect( EstP, InP ) THEN
```

```
      Ready := TRUE;
      NextStep := EstWindow;
   ELSE
      Ready := FALSE;
      NextStep := None;
   END;
END DetPOutSelect;



PROCEDURE POutMain( VAR NextStep : StepType );
(* Main procedure for the POUT menue. No hard parts. *)

VAR   InP : point;
      Button : buttontype;
      Ready, Pressed : BOOLEAN;
      CIndex : CARDINAL;
      Index  : INTEGER;

BEGIN
   ResetBackground;
   InitText;
   InitParameters( Parameters, NoParameters, P );
   InitData( Parameters, NoParameters );
   DisplayAllPar( Parameters, NoParameters );
   InitPOutSelect;
   SetTextColor( ScreenWindow, red );
   Ready := FALSE;
   WHILE NOT Ready DO
      RequestMouse( ScreenWindow, InP, Button );
      DetPOutSelect( InP, Ready, NextStep );
      IF Ready THEN
      ELSE
         Pressed := FALSE;
         Index := 1;
         CIndex := 1;
         WHILE NOT Pressed AND (CIndex <= NoParameters) DO
            IF Detect( Parameters[Index].ParP, InP ) THEN
               Pressed := TRUE;
            ELSE
               Index := Index + 1;
               CIndex := CIndex + 1;
            END;
         END;
         IF Pressed THEN
            Input( Parameters, Index );
            DecodePOutPar( Parameters, Index );
         END;
      END;
   END;
END POutMain;
```

```
BEGIN
END POut.
```

IMPLEMENTATION MODULE Opcom;

```
FROM GEMin IMPORT RequestMouse, ShowCursor, ReadString;

FROM GEMout IMPORT Text, FillRectangle;

FROM GEMset IMPORT VirtualScreen, Shutdown, SetWindow, SetViewPort,
                   SetTextColor, SetFillColor;

FROM GEMtypes IMPORT handletype, point, colortype, modetype, linetype,
                     markertype, buttontype;

FROM Strings IMPORT Delete, Insert, CompareStr, Assign, Copy;

FROM ConvReal IMPORT StringToReal, RealToString;

FROM MathLib IMPORT float, round;

FROM NumberConversion IMPORT StringToInt, IntToString;

FROM DoRun IMPORT RunMain;

FROM Filtin IMPORT FiltRefMain, FiltAngleMain, FiltPosMain;

FROM CalcFilt IMPORT TustinApprox, ConstInpApprox;

FROM EstIn IMPORT EstimateMain;

FROM Disp IMPORT DispMain;

FROM Graph IMPORT GraphMain;

FROM PBuff IMPORT PlotPoints, MaxNoPoints, NoPointsToPlot;

FROM POut IMPORT POutMain;

FROM GlobDef IMPORT OrderOfSystem, ScreenWindow, h, tau, Tick,
                    Lambda, MinEps, P, EstTime,
                    RefChannel, AngleChannel, PosChannel,
                    YMinLimit, YMaxLimit, MaxEstimOrder,
                    AKoeffRef, BKoeffRef, AKoeffAngle, BKoeffAngle,
                    AKoeffPos, BKoeffPos, RefOption, AngleOption,
                    PosOption, OptionType, FiltKoeffType, StepType,
                    KnownAPar, KnownBPar, KnownRefType, KnownArrayType;


CONST  MaxReferenses = 20;
       MaxOrderOfSystem = 4;
       Blanks = '          ';
       Known = '  known    ';
       NotKnown = ' notknown ';
```

```
TYPE  StringType = ARRAY[ 0..10 ] OF CHAR;
      InputDataType = ( Natural, Whole, Fraction );
      ReferensDataType = RECORD
         ParP : point;
         ParString : StringType;
         TypeOfPar : InputDataType;
         ParRef : CARDINAL;
      END;
      RefArrayType = ARRAY[ 1..MaxReferenses ] OF ReferensDataType;


VAR  UpLeft, LoRight,
     StopP, FilterP, DisplayP,
     EstimateP, RunP : point;
     Parameters : RefArrayType;
     NoParameters : CARDINAL;
     NextStep : StepType;



PROCEDURE InitBackground;

(* creates the virtual screen that is used in the whole program *)
(* and also sets the background color to black *)

VAR  I,J : INTEGER;

BEGIN
   UpLeft.h := 0.0;    UpLeft.v := 1.0;
   LoRight.h := 1.5;   LoRight.v := 0.0;
   VirtualScreen( ScreenWindow );
   SetViewPort( ScreenWindow, UpLeft, LoRight );
   SetWindow( ScreenWindow, UpLeft, LoRight );
   SetFillColor( ScreenWindow, black );
   FillRectangle( ScreenWindow, UpLeft, LoRight );
   FOR I := 1 TO 8 DO
      FOR J := 1 TO 8 DO
         P[I,J] := 0.0;
      END;
      P[I,I] := 100.0;
   END;
END InitBackground;


PROCEDURE Init;

VAR  I, J : INTEGER;

BEGIN
   RefOption := Tustin;
   AngleOption := Tustin;
   PosOption := Tustin;
   TustinApprox( AKoeffRef, BKoeffRef, h, tau );
   TustinApprox( AKoeffAngle, BKoeffAngle, h, tau );
```

```
   TustinApprox( AKoeffPos, BKoeffPos, h, tau );
   Lambda := 0.99;
   MinEps := 0.01;
   YMinLimit := -1.0;
   YMaxLimit := 10.0;
   NoPointsToPlot := 50;
   FOR I := 1 TO MaxNoPoints DO
      FOR J := 1 TO MaxEstimOrder DO
         PlotPoints[I,J] := 0.0;
      END;
   END;
   RefChannel := 0;
   AngleChannel := 1;
   PosChannel := 2;
END Init;


PROCEDURE ResetBackground;

(* clears the background *)

BEGIN
   UpLeft.h := 0.0;    UpLeft.v := 1.0;
   LoRight.h := 1.5;   LoRight.v := 0.0;
   SetFillColor( ScreenWindow, black );
   FillRectangle( ScreenWindow, UpLeft, LoRight );
END ResetBackground;


PROCEDURE ConvToPoint( Line, Col : INTEGER; VAR P : point );

(* converts a position on screen given in (Line,Col) to a point *)
(* that can be handled by the GEM-routines *)

BEGIN
   P.h := float( Col )*0.01875;
   P.v := 1.0 - ( float( Line )*0.04 );
END ConvToPoint;


PROCEDURE OutText( Line, Col : INTEGER; String : ARRAY OF CHAR );

(* prints the text, String, to the screen at the location given by  *)
(* Line and Col *)

VAR  P : point;

BEGIN
   SetTextColor( ScreenWindow, red );
   ConvToPoint( Line, Col, P );
   Text( ScreenWindow, P, String );
END OutText;
```

```
PROCEDURE OutBlackText( Line, Col : INTEGER; String : ARRAY OF CHAR );

(* prints the text, String, to the screen at the location given by  *)
(* Line and Col *)

VAR  P : point;

BEGIN
   SetTextColor( ScreenWindow, black );
   ConvToPoint( Line, Col, P );
   Text( ScreenWindow, P, String );
END OutBlackText;



PROCEDURE DeletePar( P : point );

(* deletes 10 characters starting at the point P *)

VAR  UL, LR : point;

BEGIN
   SetFillColor( ScreenWindow, black );
   UL.h := P.h;
   UL.v := P.v + 0.04;
   LR.h := P.h + 0.1875;
   LR.v := P.v - 0.01;
   FillRectangle( ScreenWindow, UL, LR );
END DeletePar;



PROCEDURE InitParInput( Line, Col : INTEGER;
                        VAR Referens : ReferensDataType;
                        InitString : StringType );

(* takes a Reference and assign the string, InitString, to Referens.ParString *
)

BEGIN
   WITH Referens DO
      ConvToPoint( Line, Col, ParP );
      Delete( ParString, 0, 11 );
      Copy( InitString, 0, 9, ParString );
      Insert( CHR(0), ParString, 10 );
   END;
END InitParInput;



PROCEDURE InitText;

(* initiates and writes to the screen the text in the main menu *)
```

```
VAR  P : point;

BEGIN
   OutText( 2, 3, 'THIS IS A TEST' );
   OutText( 3, 3, 'order of system' );
   OutText( 5, 3, 'sampling period' );
   OutText( 7, 3, 'parameter tau  ' );
   OutText( 9, 3, 'samples between est' );
END InitText;



PROCEDURE InitData( VAR Parameters : RefArrayType;
                    VAR NoParameters : CARDINAL );

(* initiates the parameters, on the screen, in the main menu *)

BEGIN
   InitParInput( 3, 30, Parameters[ 1 ], '       0  ' );
   OrderOfSystem := 0;
   InitParInput( 5, 30, Parameters[ 2 ], '   0.02   ' );
   h := 0.02;
   tau := h;
   InitParInput( 7, 30, Parameters[ 3 ], '   0.02   ' );
   Tick := 2;
   InitParInput( 9, 30, Parameters[ 4 ], '       0  ' );
   EstTime := 0;
   NoParameters := 4;
END InitData;



PROCEDURE InitSelectPar( Line, Col : INTEGER; VAR ParP : point;
                         String : ARRAY OF CHAR );

(* prints the string, String, to the screen at location (Line,Col) , *)
(* also returns the converted position ParP *)

BEGIN
   ConvToPoint( Line, Col, ParP );
   OutText( Line, Col, String );
END InitSelectPar;


PROCEDURE InitSelect;

(* initiates the text showing selectable menues from the the main menue *)
(* also asigns values to the variables used to determine the selected menue *)

BEGIN
   InitSelectPar( 24, 3, StopP, '   STOP    ' );
   InitSelectPar( 24, 17, FilterP, '  FILTER   ' );
   InitSelectPar( 24, 31, DisplayP, ' DISPLAY   ' );
   InitSelectPar( 24, 45, EstimateP, ' ESTIMATE ' );
```

```
   InitSelectPar( 24, 59, RunP, '   RUN    ' );
END InitSelect;


PROCEDURE DisplayParameter( Referens : ReferensDataType );

(* takes a Reference and displays its ParString on the screen *)

BEGIN
   WITH Referens DO
      Text( ScreenWindow, ParP, ParString );
   END;
END DisplayParameter;


PROCEDURE DisplayAllPar( VAR Parameters : RefArrayType;
                         NoParameters : CARDINAL );

(* displays all parameters defined in this module on the screen *)

VAR  I : CARDINAL;

BEGIN
   SetTextColor( ScreenWindow, red );
   FOR I := 1 TO NoParameters DO
      DisplayParameter( Parameters[ I ] );
   END;
END DisplayAllPar;


PROCEDURE DisplayLogicalPar( VAR KnownRef : KnownRefType;
                             Line, Col : INTEGER );

(* shows on the screen if one coefficient in the transfer function *)
(* is known or not known, ParString *)

BEGIN
   WITH KnownRef DO
      ConvToPoint( Line, Col, ParP );
      Text( ScreenWindow, ParP, ParString );
   END;
END DisplayLogicalPar;


PROCEDURE InitLogical( OrderOfSystem : INTEGER );

(* makes sure that no coefficient in the transfer function is known *)
(* at the startup or when there has been a change in the OrderOfSystem *)

VAR  I : INTEGER;

BEGIN
   FOR I := 1 TO MaxOrderOfSystem DO
```

```
      WITH KnownBPar[ I ] DO
         Value := FALSE;
         Copy( NotKnown, 0, 10, ParString );
         ParValue := 0.0;
      END;
      WITH KnownAPar[ I ] DO
         Value := FALSE;
         Copy( NotKnown, 0, 10, ParString );
         ParValue := 0.0;
      END;
   END;
END InitLogical;


PROCEDURE DisplayLogical( OrderOfSystem : INTEGER );

(* displays the whole transfer function *)

VAR  I, StartLine, StartCol : INTEGER;

BEGIN
   SetTextColor( ScreenWindow, red );
   StartLine := 12;                (* starting point on the screen *)
   StartCol := 2;
   FOR I := 0 TO (OrderOfSystem - 1) DO
      DisplayLogicalPar( KnownBPar[I+1], StartLine, StartCol + ( 16*I ) );
   END;
   StartLine := 14;
   FOR I := 0 TO (OrderOfSystem - 1) DO
      DisplayLogicalPar( KnownAPar[I+1], StartLine, StartCol + ( 16*I ) );
   END;
   OutBlackText( 11, 2, '   b1    ' );
END DisplayLogical;


PROCEDURE DeleteLogical( NumberOf : INTEGER );

(* takes away the old transfer functio from the screen *)

VAR  I, StartLine, StartCol : INTEGER;

BEGIN
   StartLine := 12;
   StartCol := 2;
   FOR I := 0 TO ( NumberOf - 1 ) DO
      DeletePar( KnownBPar[I+1].ParP );
      DeletePar( KnownAPar[I+1].ParP );
      KnownBPar[I+1].Value := FALSE;
      KnownAPar[I+1].Value := FALSE;
   END;
END DeleteLogical;
```

```
PROCEDURE Input( VAR Parameters : RefArrayType; Index : CARDINAL );

(* reads the Parameter that has been selected ( Index ) *)
(* upon return Parameters[Index].Referens.ParString contains the data *)

VAR   InString, NullString : StringType;

BEGIN
   WITH Parameters[ Index ] DO
      SetTextColor( ScreenWindow, blue );
      Delete( InString, 0, 10 );
      Delete( NullString, 0, 10 );
      Copy( ' ', 0, 1, InString );
      Copy( ' ', 0, 1, NullString );
(*    WITH Referens DO    *)
         Text( ScreenWindow, ParP, Blanks );
         ReadString( ScreenWindow, ParP, InString );
         OutText( 22, 30, InString );
         IF CompareStr( NullString, InString ) <> 0 THEN
            Insert( CHR(0), InString, 10 );
            SetTextColor( ScreenWindow, red );
            Delete( ParString, 0, 11 );
            Assign( InString, ParString );
            Text( ScreenWindow, ParP, ParString );
         ELSE
            SetTextColor( ScreenWindow, red );
            Text( ScreenWindow, ParP, ParString );
         END;
(*    END;     *)
   END;
END Input;


PROCEDURE LogInput( VAR KnownPar : KnownArrayType; Index : INTEGER );

(* if a coefficient is selected as known this procedure reads the value *)
(* that is given to the coefficient *)

VAR   InString, NullString, ParValString : StringType;
      Pos : CARDINAL;

BEGIN
   SetTextColor( ScreenWindow, blue );
   Delete( InString, 0, 10 );
   Delete( NullString, 0, 10 );
   Copy( ' ', 0, 1, InString );
   Copy( ' ', 0, 1, NullString );
   Pos := 0;
   WITH KnownPar[Index] DO
      RealToString( ParValue, ParValString, 10 );
      Text( ScreenWindow, ParP, Blanks );
      ReadString( ScreenWindow, ParP, InString );
      OutText( 22, 30, InString );
      IF CompareStr( NullString, InString ) <> 0 THEN
```

```
            Insert( CHR(0), InString, 10 );
            SetTextColor( ScreenWindow, red );
            Delete( ParValString, 0, 11 );
            StringToReal( InString, Pos, ParValue );
            RealToString( ParValue, ParValString, 10 );
            Text( ScreenWindow, ParP, ParValString );
         ELSE
            SetTextColor( ScreenWindow, red );
            Text( ScreenWindow, ParP, ParValString );
         END;
      END;
   END LogInput;



   PROCEDURE Detect( P, InP : point ): BOOLEAN;

   (* detects if the mouse has been pressed within the input box *)
   (* given by P *)

   BEGIN
      IF (InP.h > P.h) AND (InP.h < P.h + 0.1875) AND
         (InP.v > P.v) AND (InP.v < P.v + 0.04) THEN
         RETURN TRUE;
      ELSE
         RETURN FALSE;
      END;
   END Detect;


   PROCEDURE DetMainSelect( InP : point; VAR Ready : BOOLEAN;
                            VAR NextStep : StepType );

   (* checks if another menue has been chosen, if thets the case Ready is *)
   (* TRUE and NextStep contains the selected menue *)

   BEGIN
      IF Detect( StopP, InP ) THEN
         Ready := TRUE;
         NextStep := Stop;
      ELSIF Detect( FilterP, InP ) THEN
         Ready := TRUE;
         NextStep := ReferensWindow;
      ELSIF Detect( DisplayP, InP ) THEN
         Ready := TRUE;
         NextStep := DisplayWindow;
      ELSIF Detect( EstimateP, InP ) THEN
         Ready := TRUE;
         NextStep := EstWindow;
      ELSIF Detect( RunP, InP ) THEN
         Ready := TRUE;
         NextStep := RunWindow;
```

```
   ELSE
      Ready := FALSE;
      NextStep := None;
   END;
END DetMainSelect;


PROCEDURE DecodeParameter( VAR Parameters : RefArrayType;
                           Index : CARDINAL );

(* when a parameter has been read, this procedure updates the value *)
(* of the parameter given by ParString *)

VAR  IntValue : INTEGER;
     RealValue : REAL;
     Done : BOOLEAN;
     Pos : CARDINAL;

BEGIN
   Pos := 0;
   IntValue := 0;
   Done := FALSE;
   WITH Parameters[ Index ] DO
      CASE Index OF
         1 :    StringToInt( ParString, IntValue, Done );
                IF Done THEN
                   IF (IntValue > 0) AND (IntValue <= MaxOrderOfSystem) THEN
                      DeleteLogical( OrderOfSystem );
                      OrderOfSystem := IntValue;
                      IntToString( IntValue, ParString, 10 );
                      DisplayParameter( Parameters[ Index ] );
                      OutText( 13, 30, ParString );
                      InitLogical( OrderOfSystem );
                      DisplayLogical( OrderOfSystem );
                   END;
                ELSE
                   OutText( 23, 30, 'WrongInDec ' );
                END|
        2 : StringToReal( ParString, Pos, RealValue );
            IF RealValue >= 0.01 THEN
               Tick := round( RealValue/0.01 );
               h := 0.01*float( Tick );
            END;
            RealToString( h, ParString, 10 );
            DisplayParameter( Parameters[ Index ] )|
        3 : StringToReal( ParString, Pos, RealValue );
            IF RealValue >= 0.0001 THEN
               tau := RealValue;
            END;
            RealToString( tau, ParString, 10 );
            DisplayParameter( Parameters[ Index ] )|
        4 :    StringToInt( ParString, IntValue, Done );
               IF Done THEN
```

```
                IF IntValue >= 0 THEN
                   EstTime := IntValue;
                   IntToString( IntValue, ParString, 10 );
                   OutText( 13, 30, ParString );
                END;
                DisplayParameter( Parameters[ Index ] );
             ELSE
                OutText( 23, 30, 'WrongInDec ' );
             END;
          ELSE ;
       END;
    END;
END DecodeParameter;


PROCEDURE Restart;

(* this procedure is called after a visit to another menue *)
(* it redisplays the complete menue *)

BEGIN
   ResetBackground;
   InitText;
   DisplayAllPar( Parameters, NoParameters );
   DisplayLogical( OrderOfSystem );
   InitSelect;
END Restart;


PROCEDURE DoNextStep( VAR NextStep : StepType );

(* this recursive procedure is the heart of the operator communication *)
(* it handles the execution of the menues, a call chain is always ended *)
(* by a select of either MAIN or STOP *)

BEGIN
   CASE NextStep OF
      ReferensWindow :
         FiltRefMain( NextStep );
         DoNextStep( NextStep )|
      AngleWindow :
         FiltAngleMain( NextStep );
         DoNextStep( NextStep )|
(*    PositionWindow :
         FiltPosMain( NextStep );
         DoNextStep( NextStep )|   *)
      EstWindow :
         EstimateMain( NextStep );
         DoNextStep( NextStep )|
      RunWindow :
         RunMain|
      DisplayWindow :
         DispMain( NextStep );
```

```
        DoNextStep( NextStep )|
      POutWindow :
        POutMain( NextStep );
        DoNextStep( NextStep )|
      GraphWindow :
        GraphMain( NextStep );
        DoNextStep( NextStep );
      ELSE ;
   END;
END DoNextStep;


PROCEDURE OpcomMain;

(* this procedure drives the main menue, whitch is the outmost menue *)

VAR   InP : point;
      Button : buttontype;
      Ready, Pressed : BOOLEAN;
      CIndex : CARDINAL;
      Index  : INTEGER;

BEGIN
   InitBackground;
   InitText;
   InitData( Parameters, NoParameters );
   DisplayAllPar( Parameters, NoParameters );
   InitSelect;
   Init;
   SetTextColor( ScreenWindow, red );
   Ready := FALSE;
   WHILE NOT Ready DO
      RequestMouse( ScreenWindow, InP, Button );
      DetMainSelect( InP, Ready, NextStep );
      IF Ready THEN
         DoNextStep( NextStep );
         IF NextStep = Stop THEN
            Shutdown;        (* clears the screen before return to DOS *)
         ELSE
            Ready := FALSE;
            Restart;
         END;
      ELSE
         Pressed := FALSE;
         CIndex := 1;
         WHILE NOT Pressed AND (CIndex <= NoParameters) DO
            IF Detect( Parameters[CIndex].ParP, InP ) THEN
               Pressed := TRUE;
            ELSE
               CIndex := CIndex + 1;
            END;
         END;
         IF Pressed THEN
```

```
            Input( Parameters, CIndex );
            DecodeParameter( Parameters, CIndex );
         ELSE
            Pressed := FALSE;
            Index := 1;
            WHILE NOT Pressed AND (Index <= OrderOfSystem) DO
               IF Detect( KnownBPar[Index].ParP, InP ) THEN
                  Pressed := TRUE;
               ELSE
                  Index := Index + 1;
               END;
            END;
            IF Pressed THEN
               WITH KnownBPar[Index] DO
                  IF NOT Value THEN
                     Value := TRUE;
                     ParString := Known;
                  END;
                  LogInput( KnownBPar, Index );
               END;
            ELSE
               Index := 1;
               WHILE NOT Pressed AND (Index <= OrderOfSystem) DO
                  IF Detect( KnownAPar[Index].ParP, InP ) THEN
                     Pressed := TRUE;
                  ELSE
                     Index := Index + 1;
                  END;
               END;
               IF Pressed THEN
                  WITH KnownAPar[Index] DO
                     IF NOT Value THEN
                        Value := TRUE;
                        ParString := Known;
                     END;
                     LogInput( KnownAPar, Index );
                  END;
               END;
            END;
         END;
      END;
   END;
END OpcomMain;


BEGIN
END Opcom.
```

MODULE Main;

```
    (* This procedure starts the processes and initiates the semaphores *)

FROM Kernel IMPORT InitKernel, SetPriority, InitSem, CreateProcess;
```

```
FROM GlobDef IMPORT MainPrio,
                    StartClock, StartRegulate, NewFi, ClockMutex,
                    RegulateMutex, MonitorFi, SyncSem;

FROM Opcom IMPORT OpcomMain;

FROM Running IMPORT Clock, Regulate, TestEstim;

CONST MainMem = 50000;        (* memory size for main process *)
      ClockMem = 5000;        (* memory size for clock process *)
      RegulateMem = 10000;    (* memory size for reg. and filter process *)
      EstMem = 10000;         (* memory size for estimating process *)

BEGIN
   InitKernel( MainMem );                 (* starts main process *)
   SetPriority( MainPrio );
   InitSem( StartClock, 0 );              (* initialisation of all semaphores *
)
   InitSem( SyncSem, 0 );
   InitSem( StartRegulate, 0 );
   InitSem( NewFi, 0 );
   InitSem( ClockMutex, 1 );
   InitSem( RegulateMutex, 1 );
   InitSem( MonitorFi, 1 );
   CreateProcess( Clock, ClockMem );      (* starts the other processes *)
   CreateProcess( Regulate, RegulateMem );
   CreateProcess( TestEstim, EstMem );
   OpcomMain;                             (* this procedure handles all *)
                                          (* operator communication *)
END Main.
```