Realtidssimulering av modellbaserad lambdareglering för bensinmotorer

Examensarbete utfört i Fordonssystem
vid Tekniska Högskolan i Linköping
av

Magnus Bergman

Reg nr: LiTH-ISY-EX-1794

Realtidssimulering av modellbaserad lambdareglering för bensinmotorer

Examensarbete utfört i Fordonssystem
vid Tekniska Högskolan i Linköping
av

Magnus Bergman

Reg nr: LiTH-ISY-EX-1794

Handledare: **Lars Eriksson**

Examinator: **Lars Nielsen**

Linköping, 3 september 1997.

Sammanfattning

Avgasrening med katalysatorer finns i nästan alla nyproducerade bilar. Den vanligaste katalysatorn, trevägskatalysatorn, kräver en bra lambdareglering (luft/bränslereglering) för att arbeta effektivt. Lambdaregulatorn styr mängden bränsle som sprutas in i cylindrarna, i förhållande till luftmängden som sugs in i cylindrarna.

En dynamisk motormodell med fyra tillstånd byggs med mätdata från en verklig motor för att möjliggöra konstruktion, testning och verifiering av en modellbaserad lambdaregulator. Den konstuerade lambdaregulatorn arbetar i två moder, där den ena moden reglerar vid snabba förändringar och den andra moden vid stationära tillstånd samt vid långsammare förändringar. De snabba förändringar hanteras av en observatör samt en olinjär kompensator, och de långsamma förändringarna sköts av en PI-regulator med parameterstyrning. En adaptiv uppslagstabell (motormapp) används i regulatorn för att eliminera effekterna av fel i en modellbaserad motormapp. Realistiska simuleringar åstadkoms genom att implementera regulatorn och modellen i varsitt realtidssystem och göra gränssnittet mellan dessa identiskt med gränssnittet i motorlaboratoriet mellan regulator och motorcell. Modellen och regulatorn placeras i olika datorer, och kommunikationen mellan dessa sköts via seriebuss (CAN-buss) samt D/A- och A/D-kort. Resultatet av simuleringarna visar att lambdaregulatorn fungerar mycket bra om felen i regulatorns modellbaserade mappar och konstanter är försumbara. Regulatorn fungerar också tämligen väl, då fel påförs mapparna och konstanterna. Felet i den adaptiva mappen elimineras i princip efter några uppdateringscykler, vilket är tillräckligt snabbt då förändringar i luftmassflödesmappen sker långsamt.

Nyckelord: luft/bränslereglering, motorstyrning, adaptiv reglering, motormodellering

Abstract

Exhaust-gas treatment by catalytic converterers is used in almost all new cars, and the most common one, a three-way catalytic converter, demands good lambda control (air/fuel-ratio control) to work effective. The lambda controller calculates the amount of fuel to be injected, in relation to the air inducted into the cylinders.

A four-state dynamic engine model is constructed and model parameters are estimated from measurements on a real engine. The simulation model is used for development and evaluation of a model-based lambda controller. The implemented controller operates in two modes, where one mode controls fast changes, and the other mode takes care of slow changes and stationary state. The fast changes are controlled by an observer and a nonlinear compensator, while the slow changes are controlled by a PI-controller with gain-scheduling. An adaptive look-up table (engine map) is used in the controller to eliminate errors in a model-based engine map. Realistic simulations are made possible by placing the controller and the simulation model in separate realtime systems with an interface between them identical to the physical interface in the engine laboratory between controller and engine cell. The simulations show that the lambda controller works very well if the errors in model-based maps and constants are neglible. The controller is also working quite well if errors are introduced in the maps and constants. The error in the adaptive map is almost completely eliminated after a few update-cycles, which is fast enough because changes in the map are slow over time.

Key Word: air/fuel-ratio control, engine control, adaptive control, engine modelling

Tackord

Jag vill tacka alla på Fordonssystem för en stimulerande och trevlig tid. Speciellt vill jag tacka min handledare Lars Eriksson för värdefull hjälp och idérika förslag. Andrej Perkovic förtjänar också ett omnämnande för sin assistans vid motorkörningarna, sitt arbete med realtidssystem och AD/DA-kort samt för bilaga A. Dessutom vill jag tacka Mattias Nyberg för bra råd och Magnus Petterson för en funktion som omvandlar loggade data till matlabkod.

Innehåll

1 Inledning **1**
1.1 Rapporten i sammandrag 1
1.2 Läsanvisningar 2

2 Medelvärdesmodellering av bensinmotor **3**
2.1 Grundläggande motorbegrepp 3
2.2 Fysikalisk medelvärdesmodell 4
2.3 Tillståndsbeskrivning av modellen 8
2.4 Mätningar påmotorn 9
2.5 Modellidentifiering 11
2.6 Realtidsimplementering av modellen 17
2.7 Modellvalidering 22

3 Lambdaregulator **24**
3.1 Svårigheter med lambdareglering 24
3.2 Realtidsimplementering av lambdaregulatorn 24
3.3 Verifiering av lambdaregulatorn 29
3.4 Lambdaregulatorns robusthet mot fel i mappar och konstanter 30

4 Slutsatser och utvidgningar **35**
4.1 Utvidgningar 35

Referenser **36**

Bilaga A: Fordonssystems motorlaboratorium **37**

Bilaga B: Programkod för modell **39**

Bilaga C: Programkod för regulator **48**

1 Inledning

Stränga miljökrav och en explosionsartad elektronikutveckling har gjort att elektroniska motorstyrsystem blivit mycket vanliga i bilar. Styrsystemen minskar påfrestningen på miljön genom minskad bränsleförbrukning och kan, tillsammans med katalysatorer, reducera utsläpp av giftiga avgaser. Den vanligaste katalysatorn, en trevägskatalysator, kräver en bra bränsle- och luftreglering om en optimal avgasrening skall uppnås. Katalysatorn riskerar även att skadas eller i värsta fall förstöras, ifall inte regleringen fungerar tillfredsställande. Men genom att hålla det normaliserade luft/bränsleförhållandet, lambda, nära ett arbetar katalysatorn effektivt. Lambda definieras som kvoten mellan det verkliga luft/bränsleförhållandet A/F och det stökiometriska luft/bränsleförhållandet $(A/F)_{stök}$ enligt nedan. Målet att hålla lambda kring ett är alltså identiskt med att hålla A/F nära $(A/F)_{stök}$.

$$\lambda = \frac{\text{aktuellt luft/bränsleförhållande}}{\text{stökiometriskt luft/bränsleförhållande}} = \frac{A/F}{(A/F)_{stök}}$$

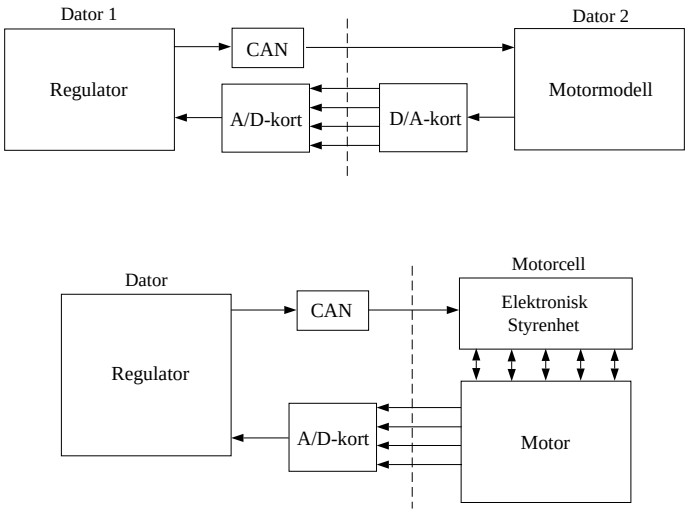
Det stökiometriska luft/bränsleförhållandet definieras som kvoten mellan luftmassflödet och bränslemassflödet, då fullständig förbränning sker. Detta inträffar när mängden syre i luften är precis tillräcklig för att förbränna allt bränsle. Ett typiskt värde på $(A/F)_{stök}$ för bensin är 14,7. Det verkliga luft/bränsleförhållandet anger kvoten mellan luftmassflödet och bränslemassflödet som flödar in i cylindrarna.

Kraven på en effektiv avgasrening motiverar ingående studier av bränsle- och luftreglering, och i rapporten beskrivs och analyseras en lambdaregulator. Inledningsvis byggs en medelvärdesmodell av en motor, som implementeras och verifieras i ett realtidssystem. Modellen används för konstruktion, testning och verifiering av en lambdaregulator. Den framtagna regulatorn arbetar i två moder, där den ena moden reglerar i stationärt tillstånd och vid långsamma förändringar, och den andra vid snabba förändringar. Regleringen vid snabba förändringar sköts av en observatör och en olinjär kompensator, medan den stationära regleringen hanteras av en PI-regulator med parameterstyrning.

Realistiska simuleringar åstadkoms genom att efterlikna ett verkligt gränssnitt mellan regulator och motorcell. Detta gör det möjligt att utföra tester och verifieringar av regulatorer, innan dessa kopplas in på en verklig motor. Figur 1.1 visar en skiss över laborationsuppställningen under simuleringarna, jämfört med den riktiga uppställningen. Kommunikationen med regulatorn sköts via CAN-buss samt D/A- och A/D-kort, och gränssnitten mellan regulator och modell respektive regulator och motorcell är identiska.

1.1 Rapporten i sammandrag

Modelleringen av motorn åskådliggörs i kapitel 2, som inledningsvis behandlar konstruktionen av en fysikalisk modell samt mätningar på den verkliga motorn. Detta följs av en parameteridentifiering samt en realtidsimplementering av modellen, och kapitlet avslutas med en modellvalidering. Kapitel 3 handlar om konstruktion av lambdaregulatorn. Problem vid lambdareglering diskuteras och ett förslag på regulatorstruktur presenteras. Kapitlet beskriver också realtidsimplementeringen av regulatorn och avslutningsvis genomförs en robusthetsanalys. Slutsatser och förslag på utvidgningar ingår i kapitel 4.



Figur 1.1. Skiss över laborationsuppställningen och den verkliga uppställningen. Den övre figuren visar laborationsuppställningen med kommunikationen mellan regulator och modellcell. Den verkliga uppställningen visas i den undre figuren. Observera att gränssnitten mot regulatorn, markerade med streckade linjer, är identiska.

1.2 Läsanvisningar

I Heywood [7] ges en grundläggande beskrivning av motorer och dess funktion. Boken behandlar det mesta inom motordynamik och modellering av motorer. I Almkvist [1] undersöks bränslefilmsdynamiken närmare och det visas hur dess parametrar är beroende av motorns tillstånd.

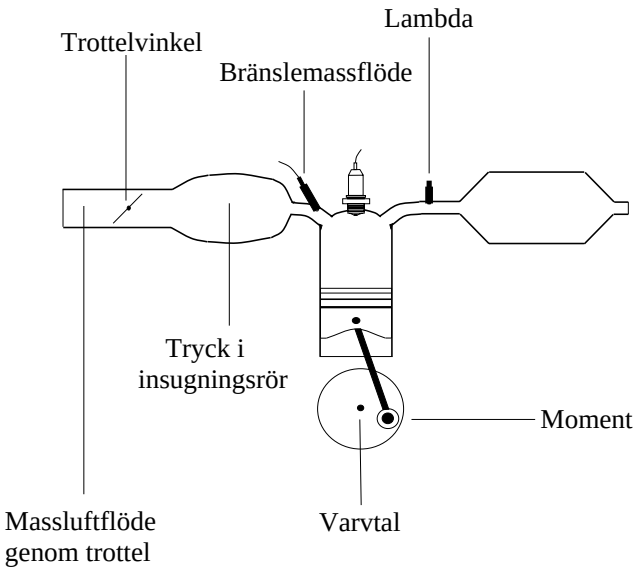
Många intressanta artiklar, rörande lambdareglering och modellering av motorer, publiceras av SAE (Society of Automotive Engineers, Inc). I Aquino [2] konstrueras en MVEM (mean value engine model) för en motor med central bränsleinsprutning. En motormodell på tillståndsform, där luftdynamik och bränslefilmsdynamik ingår, behandlas. I Hendricks [5] byggs en MVEM på tillståndsform som också inkluderar vevaxelns dynamik. Hendricks konstruerar även en lambdaregulator, baserad på en återkopplad olinjär observatör och en MVEM i [6]. I Lenz [8, 9] konstrueras lambdaregulatorer baserade på neuronnät.

2 Medelvärdesmodellering av bensinmotor

Modellbygget syftar till att möjliggöra testning och utvärdering av en lambdaregulator, samt öka förståelsen för motorns dynamik. Modellen implementeras i ett realtidssystem för att kunna simulera motorn som en självständig enhet. Kapitlet inleds med en kortare översikt av grundläggande begrepp i syfte att öka förståelsen för modellbygget. Sedan följer ett avsnitt som behandlar det fysikaliska modellbygget. Huvuddelen av kapitlet behandlar identifiering samt realtidsimplementering och avslutningsvis kommer en modellvalidering.

2.1 Grundläggande motorbegrepp

Materialet som beskriver grundläggande motorbegrepp är hämtat ur Heywood [7] och BOSCH [4]. En översiktlig skiss av motorn visas i figur 2.1. Luftflödet in till insugningsröret bestäms av trottelvinkeln och trycket i insugningssröret. Trottelvinkeln varierar mellan noll och nittio grader, där noll grader motsvarar stängd trottel. Bränsle sprutas på insugsventilerna och när dessa öppnas sugs luft och bränsle in i cylindrarna.



Figur 2.1. Schematisk skiss av bensinmotor. Trycket i insugningsröret, varvtalet samt lambda används som tillstånd i modellen. Trottelvinkeln, bränslemassflödet och lastmomentet är insignaler till modellen av motorn.

Förbränningsprocessen är beroende av luft/bränsleförhållandet i cylindrarna, detsamma gäller katalysatorns effektivitet. För att den skall arbeta effektivt bör, som tidigare nämnts, luft/bränsleförhållandet hållas nära det stökiometriska, d.v.s. lambda nära ett.

Den modellerade motorn, en SAAB 2.3L-motor, finns i Fordonssystems motorlaboratorium. Det är en fyrtaktsmotor, vilket innebär att en motorcykel består av fyra faser: insugningsfasen, kompressionsfasen, expansionsfasen samt utblåsfasen. De fyra faserna förklaras översiktligt nedan och motorns specifikationer beskrivs i bilaga A.

Insugningsfas

Insugsventilen är öppen och utblåsventilen är stängd under insugningsfasen, som inleds när kolven har nått sitt högsta läge. Luft och bränsle strömmar in i cylindern och blandas då kolven dras ned mot sitt lägsta läge. Insugningsfasen upphör, när kolven befinner sig i sitt lägsta läge, och insugsventilen stängs.

Kompressionsfas

Under kompressionsfasen är båda ventilerna stängda, och kolven trycks uppåt mot sitt högsta läge, vilket gör att luft/bränsleblandningen komprimeras. Något innan kolven når upp till sitt högsta läge antänds luft/bränsleblandningen av en gnista från tändstiftet och förbränningen påbörjas.

Expansionsfas

Under expansionsfasen hålls båda ventilerna stängda och det höga trycket i cylindern från förbränningen tvingar kolven neråt, vilket får vevaxeln att rotera. När kolven når sitt lägsta läge upphör expansionsfasen och utblåsventilen öppnas.

Utblåsningsfas

När utblåsventilen öppnas strömmar resterna av de förbrända gaserna ut, eftersom trycket i cylindern är högre än trycket utanför cylindrarna. Samtidigt trycks också gasresterna ut av kolven, när den rör sig uppåt. Utblåsningsfasen avslutas då kolven befinner sig i sitt högsta läge och en ny motorcykel inleds.

2.2 Fysikalisk medelvärdesmodell

Det är viktigt att modellen inte är onödigt komplicerad, men ändå inkluderar alla dynamiska samband som är väsentliga för lambdareglering. För lambdareglering är dynamiska samband med tidskonstanter från ungefär 30 ms och upp till 1-2 s intressanta. Variationer under en motorcykel kan därför ignoreras och långsamma dynamiska förlopp anses vara konstanta.

En s.k. MVEM (mean value engine model), som beskrivs i Aquino [2] och Hendricks [5], passar bäst till dessa specifikationer. Modellen har visat sig ge god överensstämmelse över motorns hela arbetsområde och används allmänt för konstruktion och verifiering av reglersystem.

Utgångspunkten för modellbygget i [2] och [5] är att undersöka var energin finns koncentrerad i motorn. Det ingående energiflödet består av bränsleflödet samt luftflödet, och utmomentet, tillsammans med värme- och friktionsförluster, betraktas som utgående energiflöde. Detta utmynnar i en modell med tre tillståndsekvationer som beskriver luftdynamik, bränsledynamik och vevaxelns dynamik. För att göra modellen fullständig behövs också en tillståndsekvation för dynamiken hos lambdasensorn och transport av avgaser. I tabell 2.1 listas väsentliga variabler som används vid modellbygget. Suffixet *man* kommer från det engelska uttrycket för insugningsrör; *manifold*, och suffix med f kopplas till bränslet; *fuel*.

p_{man}	Tryck i insugningsröret [kPa]
T_{man}	Temperatur i insugninsröret [K]
ρ_{man}	Luftdensitet i insugningsröret $[kg/m^3]$
m_{fp}	Bränslefilmsmassa [kg]
n	Varvtal [rpm]
λ	Normaliserat luft/bränsleförhållande
$\dot{m}_{at}$	Luftmassflöde genom trottel [kg/h]
$\dot{m}_{ac}$	Luftmassflöde in till cylindrarna [kg/h]
V_d	Slagvolym $[m^3]$
η_v	Fyllnadsgrad
c_t	Urladdningskoefficient
D	Trotteldiameter [m]
p_{omg}	Omgivningstryck [kPa]
T_{omg}	Omgivningstemperatur [K]
κ	Specifikt värmeförhållande
$\dot{m}_f$	Bränslemassflöde in i cylindrarna [kg/h]
X	Bränsledepositionsfaktor
τ_{fp}	Tidskonstant för bränslefilm [s]
J	Totalt tröghetsmoment för vevaxel och motor $[kgm^2]$
M_{in}	Moment genererat under expansionsfasen [Nm]
$M_{friktion}$	Friktionsförluster [Nm]
M_{eff}	Effektivt moment med friktionsförluster inräknade [Nm]
M_{last}	Lastmoment [Nm]
η_f	Termisk effektivitet
Q_{HV}	Bränslevärde [J/kg]
$(A/F)_{stök}$	Stökiometriskt luft/bränsleförhållande
t_d	Tidsfördröjning
T_{fi}	Bränsletid [ms]
$T_{död}$	Dödtid [ms]

Tabell 2.1. Variabler och enheter.

Luftdynamik

Luftdynamiken modelleras i [2] och [5] med hjälp av ideala gaslagen:

$$p_{man}V_{man} = m_{man}RT_{man} \tag{2.1}$$

Tryckförändringen i insugningsingsröret är proportionell mot skillnaden mellan luftmassflödet genom trotteln och luftmassflödet in till cylindrarna, vilket tillsammans med ekvation 2.1 ger tillståndsekvationen:

$$\dot{p}_{man} = \frac{RT_{man}}{3600V_{man}}(\dot{m}_{at} - \dot{m}_{ac})$$

De roterande cylindrarna i motorn betraktas som en pump, vilket gör att luftmassflödet in till cylindrarna kan skrivas som:

$$\dot{m}_{ac} = \frac{n}{120} V_d \rho_{man} \eta_v(n, p_{man}) = \frac{n}{120} \frac{p_{man} V_d \eta_v(n, p_{man})}{R T_{man}}$$

Fyllnadsgraden η_v är ett effektivitetsmått för motorn och betraktas som en funktion av varvtal och tryck i insugningsröret. Koefficienten η_v definieras som kvoten mellan luftmassflödet och luftvolymen som cylindrarna pumpar ut per tidsenhet:

$$\eta_v = \frac{2\dot{m}_a}{n V_d \rho_{man}}$$

Luftmassflödet genom trotteln approximeras med uttrycket för flödet genom ett trattliknande munstycke:

$$\dot{m}_{at} = c_t \frac{\pi}{4} D^2 \frac{p_{omg}\sqrt{\frac{2\kappa}{\kappa-1}}}{\sqrt{R T_{omg}}} \beta_1(\alpha)\beta_2(p_{man}) + \dot{m}_{at0}$$

$$\beta_1(\alpha) = 1 - \cos(\alpha - \alpha_0)$$

$$\beta_2(p_{man}) = \begin{cases} \sqrt{(\frac{p_{man}}{p_{omg}})^{\frac{2}{\kappa}} - (\frac{p_{man}}{p_{omg}})^{\frac{\kappa+1}{\kappa}}} & , (\frac{p_{man}}{p_{omg}}) \geq (\frac{2}{\kappa+1})^{\frac{\kappa}{\kappa-1}} \\ \sqrt{\frac{\kappa-1}{\kappa+1}(\frac{2}{\kappa+1})^{\frac{2}{\kappa-1}}} & , \; f.ö. \end{cases}$$

Bränsledynamik

När bränslet sprutas in förångas den största delen och går direkt in i cylindrarna, men en liten del av det insprutade bränslet fastnar på väggarna i ingångsrören och bildar en film. I Almkvist [1] visas att ett tillstånd är tillräckligt för att beskriva bränslefilmsdynamiken, samt att parametrarna i ekvationen är starkt beroende av varvtalet och trycket i insugningsingsröret. Detta leder till nedanstående tillståndsekvation, där m_{fp} betecknar massan bränslefilm och $\dot{m}_{fi}$ insprutat bränslemassflöde. Filmen antas förånga med en tidskonstant $\tau_{fp}(n, p_{man})$ och andelen bränsle som fastnar och bildar filmen definieras som $X(n, p_{man})$.

$$\dot{m}_{fp} = X(n, p_{man})\dot{m}_{fi} - \frac{1}{\tau_{fp}(n, p_{man})} m_{fp}$$

Bränslemassflödet in i cylindrarna betecknas $\dot{m}_f$ och bestäms enligt nedan. Notera att ekvation 2.2 inte är en tillståndsekvation, utan ett statiskt samband.

$$\dot{m}_f = (1 - X(n, p_{man}))\dot{m}_{fi} + \frac{1}{\tau_{fp}(n, p_{man})} m_{fp} \tag{2.2}$$

Vevaxelns dynamik

I [5] används Newtons andra kraftlag för roterande massor till att modellera vevaxelns dynamik. Den säger att vinkelaccelerationen är proportionell mot det moment som verkar på vevaxeln:

$$M = J\dot{\omega} \tag{2.3}$$

Ett moment genereras under expansionsfasen av bränsleflödet, samtidigt som ett motriktat moment genereras av vevaxelns friktionsförluster och lastmomentet. Detta, tillsammans med ekvation 2.3, leder till att tillståndsekvationen för vevaxelns dynamik kan skrivas som:

$$J\dot{\omega} = M_{in} - M_{friktion} - M_{last} \tag{2.4}$$

Energin som genereras under förbränningen av bränslet orsakar ett moment M_{in}, där storleken bestäms av bl.a. bränsleflödet in i cylindrarna $\dot{m}_f$, bränslevärdet Q_{HV} och den termiska effektiviteten $\eta_f(\lambda, n, p_{man})$ enligt nedan.

$$M_{in} = \frac{\eta_f(\lambda, n, p_{man}) Q_{HV} \dot{m}_f(t - t_c)}{2\pi n 60}$$

Uttrycket för M_{in} tillsammans med ekvation 2.4 ger slutligen nedanstående samband.

$$\frac{2\pi J\dot{n}}{60} = \frac{\eta_f(\lambda, n, p_{man}) Q_{HV} \dot{m}_f(t - t_c)}{2\pi n 60} - M_{friktion}(n, p_{man}) - M_{last}$$

$$\dot{n} = \frac{60}{2\pi J}\left(\frac{\eta_f(\lambda, n, p_{man}) Q_{HV} \dot{m}_f(t - t_c)}{2\pi n 60} - M_{friktion}(n, p_{man}) - M_{last}\right)$$

Notera att friktionsförlusterna är beroende av varvtal och tryck i insugningsröret. Den termiska effektiviteten η_f är ett mått på hur stor del av energin i bränslet som genererar effekt till motorn och betraktas som en funktion av lambda, varvtal och tryck i insugningsröret. Koefficienten η_f definieras som kvoten mellan uteffekt och ineffekt enligt:

$$\eta_f = \frac{P_{ut}}{\dot{m}_f Q_{HV}}$$

Lambdasensordynamik

Det är två faktorer som bidrar till lambdadynamiken; dynamiken hos lambdasensorn och en tidsfördröjning, vilken orsakas av att de förbrända avgaserna måste färdas en sträcka innan lambdasensorn nås. Lambdasensorns tidskonstant betecknas τ_λ och är i storleksordningen 80 ms. Tidsfördröjningen betecknas $t_d(n)$ och är beroende av både varvtal och tryck, men tryckberoendet är inte stort, vilket gör att $t_d(n)$ bara antas bero på varvtalet. Tillståndsekvationen för lambdadynamiken får då följande utseende:

$$\dot{\lambda} = \frac{1}{\tau_\lambda}\left(\frac{\dot{m}_{ac}(n, p_{man}, t - t_d(n))}{(A/F)_{stök}\, \dot{m}_f(t - t_d(n))} - \lambda\right)$$

2.3 Tillståndsbeskrivning av modellen

Den fysikaliska modellen får slutligen nedanstående utseende:

$$\dot{p}_{man} = c_t \frac{\pi}{4} D^2 \frac{p_{omg}\sqrt{\frac{2\kappa}{\kappa-1}}}{\sqrt{RT_{omg}}} \beta_1(\alpha)\beta_2(p_{man}) + \dot{m}_{at0} - \frac{n}{120}\frac{p_{man}V_d\eta_v(n,p_{man})}{RT_{man}} \quad (2.5)$$

$$\dot{m}_{fp} = X(n,p_{man})\dot{m}_{fi} - \frac{1}{\tau_{fp}(n,p_{man})} m_{fp} \quad (2.6)$$

$$\dot{n} = \frac{60}{2\pi J}\left(\frac{\eta_f(\lambda,n,p_{man})Q_{HV}\dot{m}_f(t-t_c)}{2\pi n 60} - M_{friktion}(n,p_{man}) - M_{last}\right) \quad (2.7)$$

$$\dot{\lambda} = \frac{1}{\tau_\lambda}\left(\frac{\dot{m}_{ac}(n,p_{man},t-t_d(n))}{(A/F)_{stök}\,\dot{m}_f(t-t_d(n))} - \lambda\right) \quad (2.8)$$

$$\dot{m}_f = (1-X(n,p_{man}))\,\dot{m}_{fi} + \frac{1}{\tau_{fp}(n,p_{man})}\dot{m}_{fp} \quad (2.9)$$

Den fysikaliska modellen är överparametriserad, alltså antalet parametrar är så stort att en identifierng av enskilda parametrar är omöjlig. Många av parametrarna i den fysikaliska modellen är också olinjära, exempelvis friktionsförlusterna $M_{friktion}(n,p_{man})$ och den termiska effektiviteten $\eta_f(\lambda,n,p_{man})$ i ekvation 2.7. Båda uppvisar ett olinjärt beroende av varvtal och tryck i insugningsröret, vilket ställer till bekymmer vid identifieringen. Det olinjära uttrycket för luftmassflödet genom trotteln $\dot{m}_{at}(\alpha,p_{man})$ i ekvation 2.5 kan också ställa till svårigheter vid identifieringen. Detsamma gäller bränslefilmsparametrarna $X(n,p_{man})$ och $\tau_{fp}(n,p_{man})$ i ekvationerna 2.6 och 2.9.

Parameterreduktion

En modell som använder mappar och inte funktionsuttryck eliminerar en stor del av problemen vid modellidentifieringen. Mappar kan ses som flerdimensionella statiska funktionssamband, där funktionsvärdet bestäms av en eller flera insignaler. Genom att integrera olinjäriteter samt okända parametrar och konstanter i mappar förenklas identifieringen avsevärt. Mappar kan förändras lokalt, utan att hela mappens värden förändras, vilket gör det enkelt att införa adaptivitet. Ifall funktionssamband skulle användas kan inte funktionsuttryck ändras lokalt, utan att påverka funktionens värden i hela arbetsområdet.

Olinjära uttryck som är idealiska för mappkonstruktion är bränslefilmsparametrarna $X(n,p_{man})$, $\tau_{fp}(n,p_{man})$ samt luftmassflödena $\dot{m}_{at}(\alpha,p_{man})$ och $\dot{m}_{ac}(n,p_{man})$. Dessutom kan friktionsförlusterna $M_{friktion}(n,p_{man})$ och inmomentet $M_{in}(\lambda,n,p_{man})$ integreras i en mapp, vilket gör att den slutgiltiga tillståndsbeskrivningen får nedanstående utseende. Insignaler är trottelvinkeln α, lastmomentet M_{last} och bränslemassflödet $\dot{m}_{fi}$. Trycket i insugningsröret p_{man}, bränslefilmsmassan m_{fp}, varvtalet n och lambda är modellens tillstånd. Ekvation 2.14 för bränslemassflödet in i cylindrarna $\dot{m}_f$ är ingen tillståndsekvation, utan ett statiskt samband, och ingår i tillståndsbeskrivningen för att öka förståelsen och läsbarheten.

$$\dot{p}_{man} = C_1(\dot{m}_{at}(\alpha, p_{man}) - \dot{m}_{ac}(n, p_{man})) \quad (2.10)$$

$$\dot{m}_{fp} = X(n, p_{man})\dot{m}_{fi} - \frac{1}{\tau_{fp}(n, p_{man})} m_{fp} \quad (2.11)$$

$$\dot{n} = C_2(M_{eff}(\lambda, n, p_{man}) - M_{last}) \quad (2.12)$$

$$\dot{\lambda} = \frac{1}{\tau_\lambda}\left(\frac{\dot{m}_{ac}(n, p_{man}, t - t_d(n))}{(A/F)_{stök}\, \dot{m}_f(t - t_d(n))} - \lambda\right) \quad (2.13)$$

$$\dot{m}_f = (1 - X(n, p_{man}))\dot{m}_{fi} + \frac{1}{\tau_{fp}(n, p_{man})}\dot{m}_{fp} \quad (2.14)$$

2.4 Mätningar på motorn

Den enda konstanten eller parametern som är känd innan identifieringen är tidskonstanten för lambdasensorn τ_λ, som antas vara 83 ms. Övriga konstanter och parametrar identifieras med mätdata från motorkörningar. Innan några mätningar utförs är det viktigt att bestämma samplingsfrekvens och modellens giltighetsområde. Andra beslut som bör fattas är vilka typer av mätningar som skall göras och vilka variabler som skall mätas. Alla motorkörningar är utförda på en SAAB 2.3L-motor och en dynamometer i Fordonssystems laboratorium. Dynamometern fungerar som en motorbroms och används bl.a. för reglering av varvtal och moment, och den bör inte belastas av moment högre än ca. 150 Nm.

Modellens giltighetsområde

En modells giltighetsområde måste väljas med omsorg, eftersom ett för stort område kan ge stora modellfel och ett för litet område kan göra modellen obrukbar. Dessutom måste det verkliga systemets fysikaliska begränsningar beaktas, så att haverier och störningar undviks. Modellens giltighetsområde är här valt till:

$$1211 \leq n \leq 4016 \quad \text{rpm}$$
$$35,4 \leq p_{man} \leq 70,0 \quad \text{kPa}$$

Modellen skall beskriva den verkliga motorn så bra som möjligt och därför är de lägre gränserna för varvtal och tryck i insugningsröret något högt valda. Detta följer av att modellbygget skulle bli mycket svårt vid låga varvtal och låga tryck, eftersom de dynamiska förhållandena varierar stort från körning till körning. Den övre gränsen för trycket i insugningsröret är vald till 70,0 kPa för att inte överbelasta dynamometern.

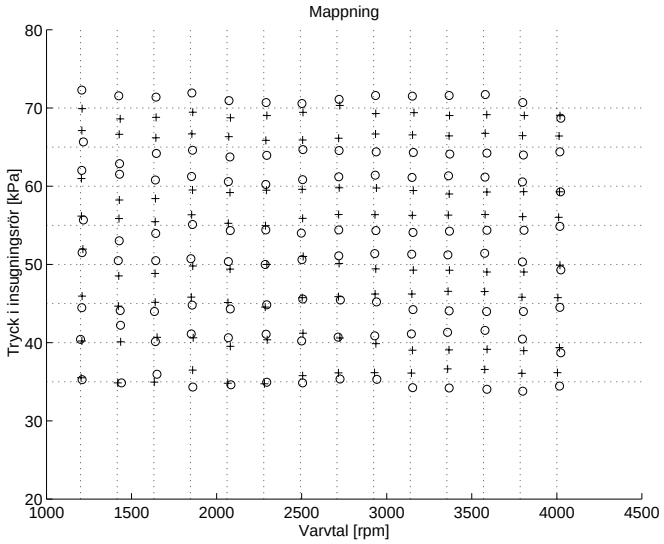
Motorkörningar

Motorkörningarna delas upp i två separata delar, där den ena delen genererar mätdata för mappkonstruktion och den andra delen ger mätdata till identifiering av dynamiken, som t.ex. konstanten C_1 i ekvation 2.10. I den förstnämnda delen skall stationära förlopp åskådliggöras och i den senare skall snabba förändringar beskrivas. Alla mätdata från motorkörningarna genereras med en samplingsfrekvens på 1500 Hz, vilket är tillräckligt snabbt för att ge information om de snabba förloppen. Till mappkonstruktionen är inte valet av samplingsfrekvens viktigt, då mätdata endast skall beskriva stationära förlopp. Samma samplingsfrekvens kan alltså väljas för de bägge typerna av mätningar.

Mätningar för mappar

Mätdata för mappkonstruktionen genereras vid stationära förlopp i olika arbetspunkter. Till det ändamålet finns en varvtals- och tryckregulator som håller motorn konstant på en arbetspunkt under tiden mätningar sker och styr sedan motorn till nästa arbetspunkt. En lambdaregulator används också för att försöka hålla lambda konstant kring ett önskat referensvärde. Genom att mäta varvtalet n, trycket i insugningsröret p_{man} och luftmassflödet genom trotteln $\dot{m}_{at}$ kan en mapp för luftmassflödet in till cylindrarna $\dot{m}_{ac}(n, p_{man})$ konstrueras. Detta beror på att de båda luftmassflödena $\dot{m}_{at}$ och $\dot{m}_{ac}(n, p_{man})$ är identiska under stationära förhållanden ($\dot{p}_{man}$=0 i ekvation 2.10). Även mapparna för det effektiva momentet $M_{eff}(\lambda, n, p_{man})$ kan bestämmas på samma sätt, då det effektiva momentet är identiskt med lastmomentet M_{last} i stationärt tillstånd ($\dot{n}$=0 i ekvation 2.12). Den enda skillnaden är att mappar för olika lambda måste konstrueras, eftersom $M_{eff}(\lambda, n, p_{man})$ beror av lambda. Mappen för luftmassflödet genom trotteln $\dot{m}_{at}(\alpha, p_{man})$ bestäms genom att mäta trottelvinkeln α, trycket i insugningsröret p_{man} samt $\dot{m}_{at}$ i arbetspunkterna.

I figur 2.2 visas de nådda arbetspunkterna i en mätning för mappkonstruktion. En matlabfunktion utför mätningarna genom att ställa ut referensvärden till varvtals- och tryckregulatorn. Efter sju sekunder nås den nya arbetspunkten och mätningar samt medelvärdesbildningar av varvtal, tryck, luftmassflöde genom trottel, trottelvinkel och lambda görs under fyra sekunder, sedan läggs nästa referensvärden ut o.s.v. Motormappningen sker i 112 olika arbetspunkter, för 14 olika varvtal och 8 olika tryck. Vid mappningen arbetar regulatorn sig uppåt tills arbetspunkten i det övre vänstra hörnet nås och följer sedan samma väg tillbaka. Systematiska fel undviks därför att mätningar utförs två gånger i varje arbetspunkt, en gång på uppvägen och en gång på nervägen. Varje mätvariabel representeras slutligen av två matriser med dimensioner 8 x 14.



Figur 2.2. Figur över en mätning för mappkonstruktion då λ=0,90. Varje nod är en önskad arbetspunkt. Ringarna visar de nådda arbetspunkterna för uppvägen och plustecknen för nervägen.

Mätningarna utförs för fem olika lambda ($\lambda = 0,90; 0,95; 1,00; 1,05; 1,10$) för att få med lambdaberoendet i $M_{eff}(\lambda, n, p_{man})$. Vid konstruktionen av luftmassflödesmapparna $\dot{m}_{at}(\alpha, p_{man})$ och $\dot{m}_{ac}(n, p_{man})$ används bara mätdata från mappningen då λ=1,00.

Mätningar för identifiering av dynamik

Modellen innehåller två konstanter, C_1 och C_2, som måste identifieras. Dessa bestäms genom att mäta väsentliga variabler, som t.ex. tryck och varvtal under trottelsteg respektive momentsteg. Trottelstegen för identifieringen av C_1 görs vid tre arbetspunkter och momentstegen för C_2 görs vid två arbetspunkter. Eftersom identifieringen utförs med mätdata från flera arbetspunkter lindras effekterna av fel i någon arbetspunkt. Under mätningarna för identifieringarna av C_1 och C_2 är ingen tryckregulator inkopplad och då C_2 bestäms är också varvtalsregulatorn urkopplad.

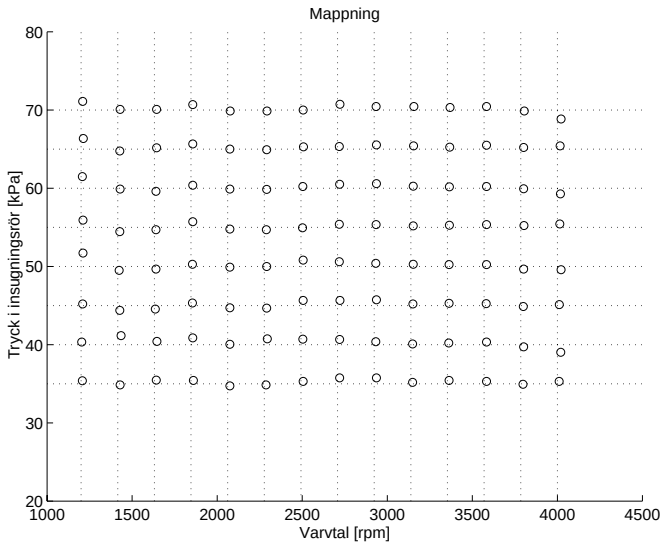
De två bränslefilmsparametrarna $X(n, p_{man})$ och $\tau_{fp}(n, p_{man})$ bestäms genom att mäta lambdasensorsvar på bränsletidssteg. Åtta steg utförs i tolv olika arbetspunkter och varvtal, tryck, lambda och $\dot{m}_{fi}$ mäts. Under mätningarna måste lambdaregulatorn vara urkopplad för att inte förstöra mätdata.

Oförklarliga störningar

Störningar med en frekvens kring 8 Hz observeras på alla mätsignaler. Störningarna, vars ursprung är obekant, modelleras inte, eftersom de varken har sitt ursprung i motorn eller är normala störningar från mätgivare. I exempelvis figur 2.9 framträder dessa oförklarliga störningar tydligt.

2.5 Modellidentifiering

Med mätdata från motorkörningarna kan identifieringen av alla parametrar och konstanter i modellen utföras. Inledningsvis medelvärdesbildas dock mätdata till mappkonstruktionen, för att eliminera systematiska fel. Figur 2.3 visar resultatet från en medelvärdesbildning. Observera att avstånden till de önskade arbetspunkterna har minskat signifikant.

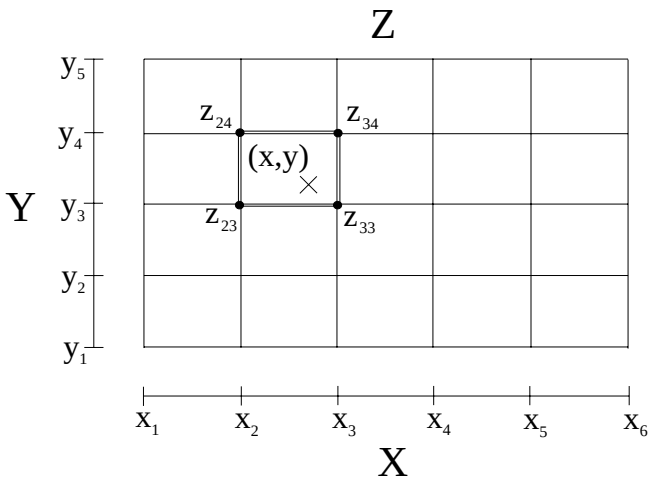


Figur 2.3. Figur över en medelvärdesbildning av en mätning för mappkonstruktion då λ=0,90. Notera att avstånden till de önskade arbetspunkterna i noderna har minskat signifikant.

Konstruktion av mappar

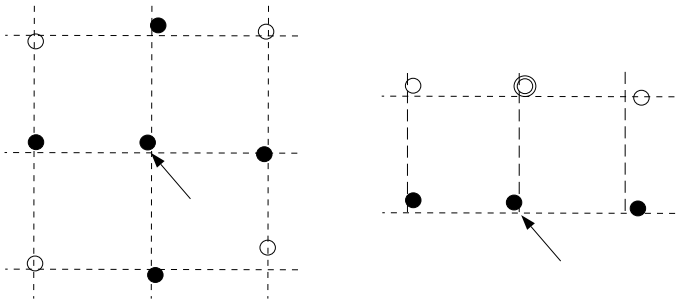
Mappar som implementeras i realtidssystem bör vara rektangulära, eftersom interpolationer och sökningar i mapparna skall gå snabbt. Mätdata från motorkörningarna måste därför anpassas till rektangulära mappar. En rektangulär mapp representeras av en tvådimensionell matris Z med dimension m x n och två vektorer X och Y av längd m respektive n. Elementen i vektorerna motsvarar insignalvärden och elementen i matrisen motsvarar utsignalvärden. Funktionsvärdet i en punkt, som inte representeras av vektorernas element, bestäms genom någon form av interpolation. En mappning består alltså av två vektorsökningar för att finna de fyra hörnpunkterna och en interpolation. Ekvationerna för en mappning visas nedan, där funktionen Ψ anger de två vektorsökningarna och , interpolationen. Skissen i figur 2.4 förklarar variablerna i ekvationerna och visar utseendet av en mapp med dimension 6 x 5.

$$\begin{aligned} z_{\square} &= \Psi(x,y) = (z_{22}, z_{32}, z_{23}, z_{33}) \\ z(x,y) &= , (z_{\square}) \end{aligned}$$



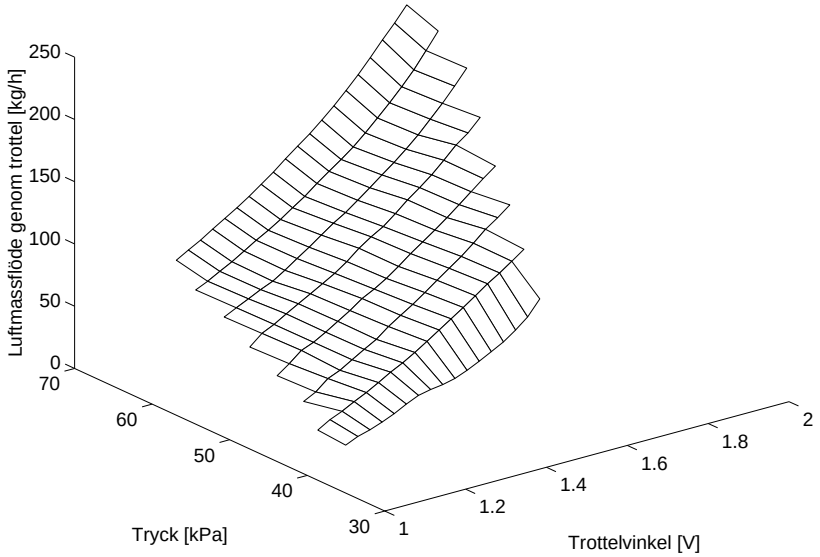
Figur 2.4. Schematisk skiss av en mapp med dimensionen 6 x 5, där ett kryss anger punkten som skall mappas och rektangeln de fyra hörnpunkter som används vid interpolationen.

Varvtalsmatrisens kolumner och tryckmatrisens rader medelvärdesbildas för att ge de optimala elementen i varvtals- respektive tryckvektorn. Avstånden från de mätta punkterna till de önskade arbetspunkterna blir då minimalt, vilket gör att felet vid mappkonstruktionen blir litet. Mapparna bildas genom att vikta det normaliserade viktade avståndet från den önskade arbetspunkten. Den punkt som ligger närmast den önskade arbetspunkten, samt de punkter som bildar ett kors kring den används vid mappkonstruktionen. Om den önskade arbetspunkten ligger i någon kant används endast punkterna längs kanten och hörnelementen i mappen skattas manuellt. En schematisk skiss, figur 2.5, visar hur punkterna som ingår i skattningen väljs. Eftersom bara motsatta punkter används under mappkonstruktionen uppstår ingen obalans i skattningen. En mapp skulle exempelvis bli felaktig om den dubbelt inringade punkten skulle ingå i skattningen, samtidigt som det finns ett linjärt beroende i den riktningen.



Figur 2.5. Skiss över vilka punkter som används i skattningen. Den önskade arbetspunkten som skattas anges med pil och de använda punkterna är fyllda. Den dubbelt inringade punkten orsakar ett fel i mappen om den ingår i skattningen och det finns ett linjärt beroende i den riktningen.

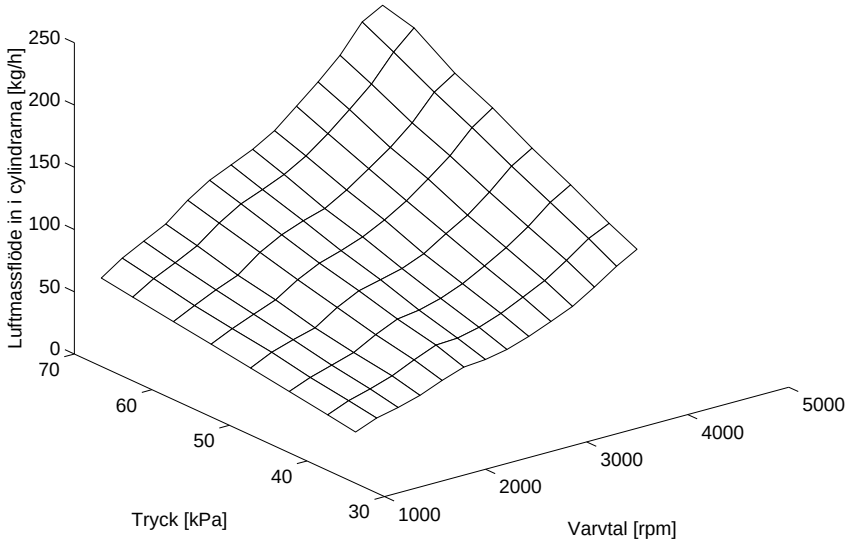
Figur 2.6 visar mappen för luftmassflödet genom trotteln $\dot{m}_{at}(\alpha, p_{man})$. I modellens giltighetsområde är flödet nästan linjärt med avseende på trottelvinkel och approximativt konstant med avseende på trycket i insugningsröret. Observera att mappen är konstruerad för värden från en trottelgivare och trottelvinkeln anges därför i volt, istället för i grader. Mappen skrivs ändå i fortsättningen som en funktion av trottelvinkeln α för att behålla den fysikalisk meningen. Notera också att mappens giltighetsområde inte är konstant, då trottelvinkeln påverkar trycket.



Figur 2.6. Luftmassflödet genom trotteln $\dot{m}_{at}(\alpha, p_{man})$. Linjärt beroende i tryckled och approximativt konstant beroende av trottelvinkel.

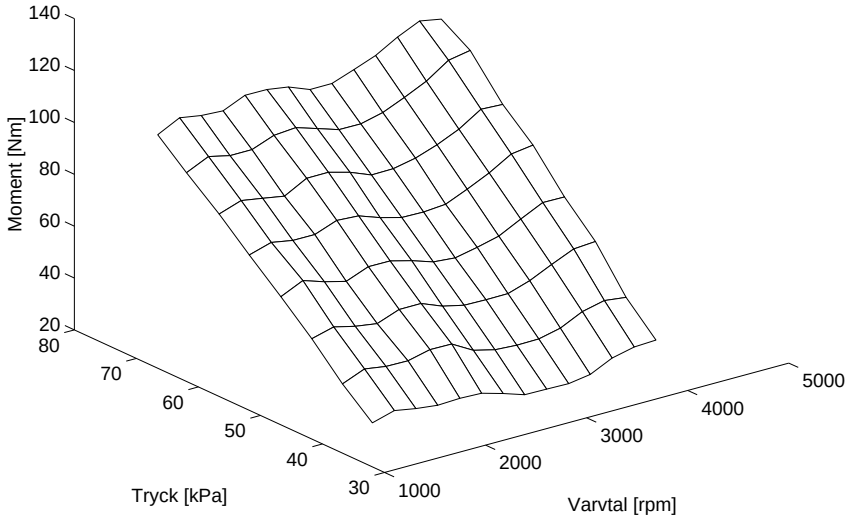
Luftmassflödet in i cylindrarna $\dot{m}_{ac}(n, p_{man})$ visas i figur 2.7. Flödet beror praktiskt taget linjärt på både varvtal och tryck i insugningsröret över modellens giltighetsområde.

I figur 2.8 visas den skattade mappen för det effektiva momentet $M_{eff}(n, p_{man})$ då λ=1,00. Momentet beror linjärt på trycket i insugningsröret och olinjärt på varvtalet. Variationen i varvtalsled orsakas av att stående vågor bildas i insugningsröret vid vissa varvtal, vilket ger toppar i momentet. De fyra andra mapparna, för övriga lambda,



Figur 2.7. Luftmassflödet in till cylindrarna $\dot{m}_{ac}(n, p_{man})$. Linjärt beroende både på tryck och varvtal.

visas inte eftersom de har ungefärligen samma utseende och tillför därför ingenting av intresse. Det bör dock nämnas att momentnivåerna ökar med minskande lambda, eftersom utmomentet är maximalt då λ=0,90.



Figur 2.8. Det effektiva momentet $M_{eff}(n, p_{man})$. Observera variationen i varvtalsled som orsakas av stående vågor i insugningsröret. Tryckberoendet är approximativt linjärt.

Identifiering av konstanter

Konstanten C_1 identifieras genom att mäta svaret på ett trottelsteg. Ekvationen 2.10 integreras då derivering av en brusig mätsignal, trycket i insugningsröret, inte är tillrådligt. Luftmassflödet in till cylindrarna $\dot{m}_{ac}(n, p_{man})$ interpoleras fram och luftmassflödet genom trotteln $\dot{m}_{at}$ mäts. Detta ger ekvationen nedan, där tiden t står för tidpunkten då steget startar och $t + T_{tran}$ anger när transienten ebbar ut. C_2 identifieras på samma sätt och integralerna i uttrycken approximeras med trapetssummor.

$$p_{man}(t + T_{tran}) - p_{man}(t) = C_1 \int_t^{t+T_{tran}} (\dot{m}_{at} - \dot{m}_{ac})\, dt$$

Identifieringen av konstanterna ger:

$$C_1 = 4,360$$
$$C_2 = 46.395$$

Identifiering av bränslemassflödesekvation

Det är omöjligt att mäta bränslemassflödet $\dot{m}_{fi}$, men bränsletiden T_{fi}, som anger hur länge bränsleinsprutningsventilerna är öppna per cykel, kan mätas. Det tar dock tid för ventilerna att öppnas, vilket gör att en dödtid $T_{död}$ måste subtraheras från bränsletiden. Varvtalet ingår också i uttrycket för bränslemassflödet, eftersom varvtalet bestämmer hur många gånger ventilerna öppnas per tidsenhet. En funktion för bränslemassflödet får alltså nedanstående utseende, där n_0 är en normeringskonstant.

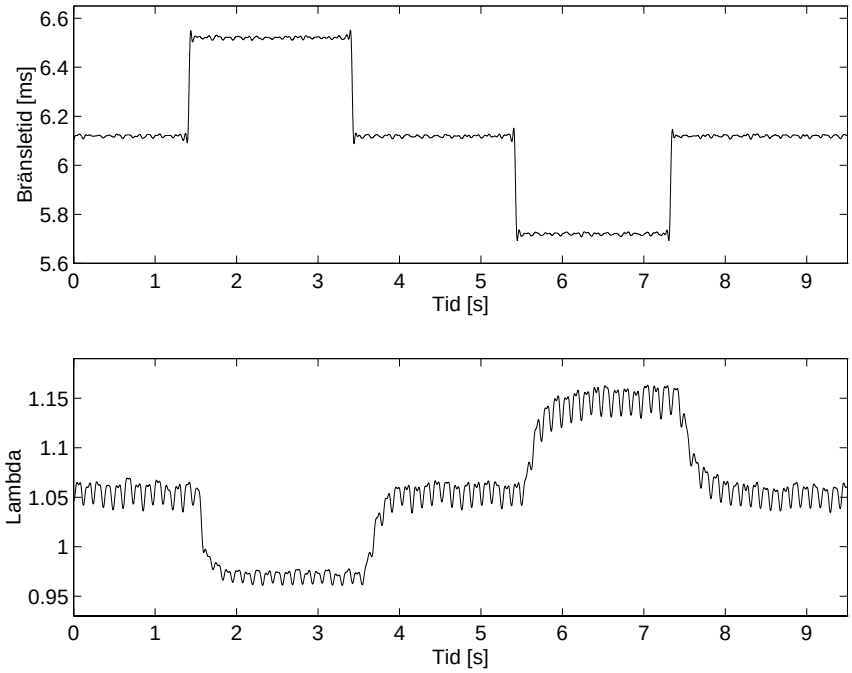
$$\dot{m}_{fi} = \frac{n}{n_0}(T_{fi} - T_{död})$$

Om bränslemassflödet beräknas i några punkter, samtidigt som bränsletiden och varvtalet mäts, kan n_0 och $T_{död}$ bestämmas genom att anpassa en rät linje. Bränslemassflödet skattas genom att mäta lambda och luftmassflödet genom trotteln i stationärt tillstånd, samtidigt som det stökiometriska luft/bränsleförhållandet $(A/F)_{stök}$ antas vara 14,67. Mätningarna som senare används vid skattningarna av bränslefilmparametrar utnyttjas också i den här identifieringen. Resultatet av identifieringen blir:

$$\dot{m}_{fi} = \frac{n}{2401}(T_{fi} - 0.657)$$

Identifiering av bränslefilmsparametrar

Bränslefilmsparametrarna identifieras ur mätdata, där bränslefilmsdynamiken har exciterats genom steg i bränslemassflödet med lambdaregulatorn urkopplad. Dessa steg utförs i tolv arbetspunkter, för fyra olika varvtal och tre olika tryck i insugningsröret. En simulinkmodell som beskriver bränslefilmsdynamiken och lambdasensordynamiken används vid identifieringen. Genom att anpassa simulinkmodellens lambdasensorsvar till det uppmätta svaret för identiska bränsletidssteg kan de båda bränslefilmsparametrarna $X(n, p_{man})$, $\tau_{fp}(n, p_{man})$ och tidsfördröjningen $t_d(n)$ bestämmas. I [1] visas att parametrarna påverkar lambdasensorsvarets utseende på olika sätt, vilket gör det möjligt att



Figur 2.9. Fyra bränsletidssteg och lambdasensorsvar då n=1580 rpm och p_{man}=50,8 kPa. Den övre figuren visar bränsletiden T_{fi} och den undre lambda. Simulinkmodellens lambdasensorsvar anpassas till det uppmätta svaret. Notera de oförklarliga störningarna, som är mest signifikanta på lambdasensorsvaret.

utföra identifieringen. I figur 2.9 visas fyra bränsletidssteg med lambdasensorsvar då n=1580 rpm och p_{man}=50,8 kPa, notera de stora störningarna kring 8 Hz.

Resultatet av identifieringen visas i nedanstående tabeller; 2.2, 2.3 samt 2.4. Depositionsfaktorn $X(n, p_{man})$ minskar med ökande varvtal och tryck, vilket innebär att mindre av bränslet fastnar och bildar bränslefilm. Tidskonstanten minskar också och det betyder att en större del av bränslefilmen förångas, då varvtalet och trycket ökar. Tidsfördröjningen $t_d(n)$ minskar med ökande varvtal, vilket innebär att gasresterna når lambdasensorn tidigare vid ökat varvtal. Resultaten av identifieringen är logiska och stämmer väl överens med de som beskrivs i [1].

$X(n, p_{man})$		Varvtal [rpm]			
		1580	2013	2717	3510
Tryck [kPa]	39,0	0,3	0,3	0,3	0,05
	51,0	0,2	0,1	0,05	0,05
	59,0	0,1	0,05	0,05	0,05

Tabell 2.2. Depositionsfaktorns beroende av varvtal och tryck i insugningsröret. Observera att $X(n, p_{man})$ minskar med ökande varvtal och tryck. Det betyder att mindre och mindre av det insprutade bränslet fastnar och bildar film.

$\tau_{fp}(n, p_{man})$ [s]		Varvtal [rpm]			
		1580	2013	2717	3510
Tryck [kPa]	39,0	0,5	0,25	0,25	0,1
	51,0	0,25	0,125	0,1	0,1
	59,0	0,125	0,1	0,1	0,1

Tabell 2.3. Tidskonstantens beroende av varvtal och tryck i insugningsröret. Notera att $\tau_{fp}(n, p_{man})$ minskar med ökande varvtal och tryck, vilket innebär att en större och större del av bränslefilmen förångas.

	Varvtal [rpm]				
	1200	1580	2013	2717	3510
$t_d(n)$ [s]	0,16	0,14	0,11	0,08	0,06

Tabell 2.4. Tidsfördröjningens beroende av varvtalet. Fördröjningen minskar med ökande varvtal, vilket betyder att gasresterna når lambdasensorn tidigare vid ökat varvtal.

Jämförelse mellan tidskonstanter

Det är intressant med en jämförelse mellan tidskonstanterna för de olika dynamiska effekterna, i syfte att undersöka om skillnaderna är stora, men resultatet får inte överskattas. Tidskonstanten för bränslefilmsdynamiken, $\tau_{fp}(n, p_{man})$, varierar och bara en del av bränslet fastnar, medan merparten av bränslet flödar direkt in i cylindrarna. Tidsfördröjningen för gasresterna, $t_d(n)$, påverkar förmodligen lambdadynamiken mer än tidskonstanten för lambdasensorn. De ungefärliga tidskonstanterna och tidsfördröjningen $t_d(n)$ visas i tabell 2.5, där det visas att luftdynamiken är snabbast och varvtalsdynamiken är långsammast. Lambdasensordynamiken är något långsammare och bränslefilmsdynamiken något snabbare, än vad tidskonstanterna antyder. Detta beror på tidsfördröjningen, $t_d(n)$, respektive att bara en del av bränslet fastnar och bildar bränslefilm.

Tidskonstanter [ms]	
Luftdynamik	60
Lambdasensordynamik	80
Tidsfördröjning $t_d(n)$	60-160
Bränslefilmsdynamik	100-500
Varvtalsdynamik	500

Tabell 2.5. Tidsfördröjning samt ungefärliga tidskonstanter för dynamiken, som visar att luftdynamiken är snabbast och varvtalsdynamiken långsammast.

2.6 Realtidsimplementering av modellen

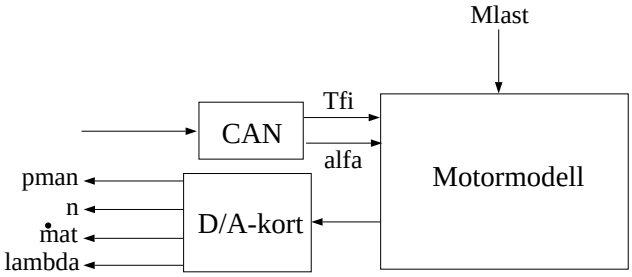
Modellen av motorn implementeras som en process i ett realtidssystem. Realtidskärnan i systemet har konstruerats med RTKernel 4.0 och alla processer programmeras i C. Ett programskal, som är förberett för nya processer, sköter kommunikationen med realtidssystemet. Systemet allokerar processortid till processen och sköter om intern och extern kommunikation. Avläsning respektive skrivning på interna och externa portar

är möjliga och signaler kan mätas genom att logga interna portar. Realtidssystemet är också förberett för seriell kommunikation via CAN-buss.

Algoritmen för realtidsimplementeringen av modellen får följande utseende:

```
Avläsning av insignaler
Konvertering av insignaler
Mappning
Uppdatering av tillstånd
Addering av brus
Konvertering av utsignaler
Utställning av utsignaler
```

Modellen tar emot två insignaler från regulatorn; trottelvinkeln α samt bränsletiden T_{fi}, och dessutom kan lastmomentet M_{load} ändras via en intern port. Modellen genererar fyra utsignaler som används av regulatorn; trycket i insugningsröret p_{man}, varvtalet n, luftmassflödet genom trottel $\dot{m}_{at}$ samt lambda. I figur 2.10 visas en skiss över modellens ut- och insignaler. Utsignalerna ställs ut via ett D/A-kort och insignalerna läses in via CAN-buss.



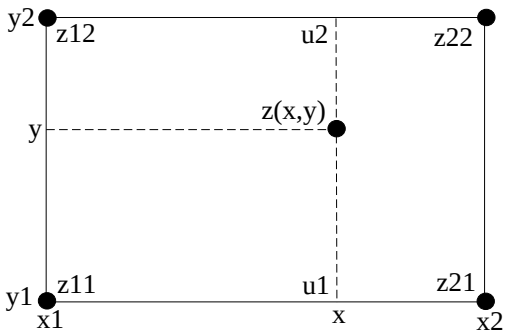
Figur 2.10. Modellen med alla ut- och insignaler.

Konvertering av in- och utsignaler

Av insignalerna är det bara bränsletiden T_{fi}, som behöver konverteras om till bränslemassflödet $\dot{m}_{fi}$, eftersom trottelvinkeln sänds över CAN-bussen ograverad. Utsignalerna konverteras till de spänningar som de verkliga givarna ställer ut.

Mappning

Mappningen av variabler och parametrar kan delas upp i två steg; sökning och interpolation. En rektangel söks, bestående av fyra element i mappens matris. Rektangeln omgärdar punkten som skall mappas och bestäms genom linjärsökningar i mappens två vektorer. Anledningen till att linjärsökning används är att en implementation av algoritmen är enkel och att vektorerna inte är långa. Funktionsvärdet i den sökta punkten bestäms genom linjär interpolation, vilket är en enkel algoritm som också ger bra resultat. Ekvationerna för den linjära interpolationen visas nedan. Skissen i figur 2.11 förklarar variablerna z_{ij}, x_i, y_i och u_i i ekvationerna.



Figur 2.11. Skiss som åskådliggör linjär interpolation. Mappens arbetspunkter betecknas x_1 och x_2 respektive y_1 och y_2. Den sökta arbetspunkten anges med x och y. Det mappade värdet betecknas z och värdena i mappens arbetspunkter anges med z_{11}, z_{21}, z_{12} samt z_{22}.

$$\begin{aligned} k_1 &= \frac{x - x_1}{x_2 - x_1} \\ k_2 &= \frac{y - y_1}{y_2 - y_1} \\ u_1 &= z_{11} + k_1(z_{21} - z_{11}) \\ u_2 &= z_{12} + k_1(z_{22} - z_{12}) \end{aligned}$$

$$z(x, y) = u_1 + k_2(u_2 - u_1) \tag{2.15}$$

Ekvation 2.15 kan skrivas om på en mer lätthanterlig form:

$$z(x, y) = \gamma_{11} z_{11} + \gamma_{21} z_{21} + \gamma_{12} z_{12} + \gamma_{22} z_{22} \tag{2.16}$$

De fyra variablerna γ_{ij} bestäms då enligt:

$$\begin{aligned} \gamma_{11} &= 1 - k_1 - k_2 + k_1 k_2 \\ \gamma_{21} &= k_2 - k_1 k_2 \\ \gamma_{12} &= k_1 - k_1 k_2 \\ \gamma_{22} &= k_1 k_2 \end{aligned}$$

Tidsdiskretisering av modellen

Den kontinuerliga modellen måste först diskretiseras innan den kan implementeras i ett realtidssystem. Genom att approximera derivatorna enligt nedanstående uttryck fås Eulers metod för numeriska lösningar av differentialekvationer, där T_{sm} är modellens samplingstid. Eulers metod har den minsta komplexiteten, vilket leder till snabba beräkningar. De snabba beräkningarna gör att steglängden (samplingstiden) kan minskas och att noggrannheten därigenom ökar. Om en mer komplex numerisk metod, exempelvis Runge-Kutta, används måste beräkningshastigheten minskas. De långsamma

beräkningarna leder till minskade steglängder och en försämrad noggrannhet. Det är alltså inte säkert att en mer komplex metod ger ett bättre resultat vid simuleringarna, trots att det globala trunkeringsfelet är mindre.

$$\begin{aligned} \dot{x} &= f(x,y,u) \\ \dot{x} &\approx \frac{x(t+T_{sm})-x(t)}{T_{sm}} \\ x(t+T_{sm}) &\approx x(t)+T_{sm}f(x,y,u) \end{aligned}$$

Den diskretiserade modellen implementerad enligt Eulers metod får följande utseende:

$$p_{man}(t+T_{sm}) = p_{man}(t)+C_1T_{sm}(\dot{m}_{at}(\alpha,p_{man})-\dot{m}_{ac}(n,p_{man})) \quad (2.17)$$

$$m_{fp}(t+T_{sm}) = (1-\frac{T_{sm}}{\tau_{fp}(n,p_{man})})m_{fp}(t)+X(n,p_{man})T_{sm}\dot{m}_{fi} \quad (2.18)$$

$$n(t+T_{sm}) = n(t)+C_2T_{sm}(M_{eff}(\lambda,n,p_{man})-M_{last}) \quad (2.19)$$

$$\lambda(t+T_{sm}) = (1-\frac{T_{sm}}{\tau_\lambda})\lambda(t)+\frac{T_{sm}}{\tau_\lambda}\frac{\dot{m}_{ac}(n,p_{man},t-t_d(n))}{(A/F)_{stök}\,\dot{m}_f(t-t_d(n))} \quad (2.20)$$

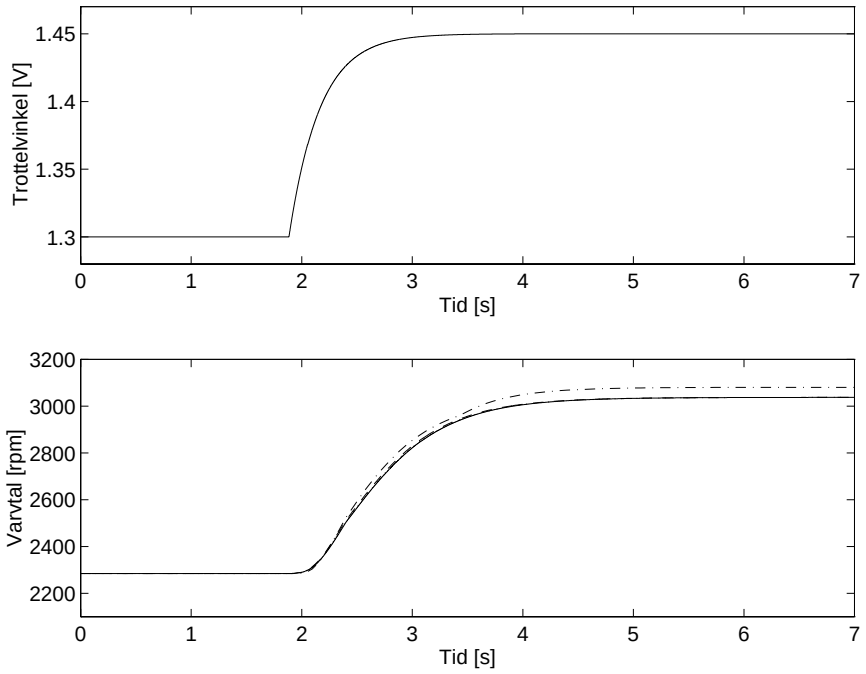
$$\dot{m}_f(t+T_{sm}) = (1-X(n,p_{man}))\dot{m}_{fi}+\frac{1}{\tau_{fp}(n,p_{man})}m_{fp}(t+T_{sm}) \quad (2.21)$$

Addering av brus

En modell som försöker efterlikna verkligheten är inte fullständig utan brus. Därför adderas normalfördelat brus till modellen och amplituderna väljs, så att de efterliknar de uppmätta nivåerna i laboratoriet så mycket som möjligt. De oförklarliga störningarna på 8 Hz ignoreras dock, eftersom de inte orsakas av motorn eller är normala störningar från mätgivare.

Val av samplingstid

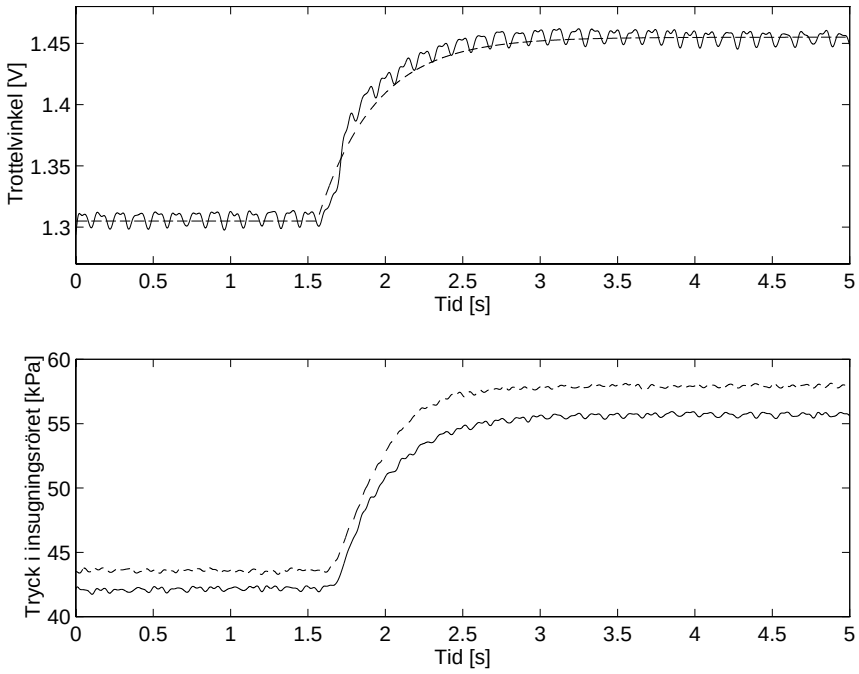
Det är väsentligt att den valda samplingstiden (steglängden) ger små fel vid lösningen av differentialekvationerna. Dessutom är det viktigt att samplingstiden är tillräckligt stor för att inte överbelasta datorn och tillräckligt liten för att modellen skall uppfattas som kontinuerlig av regulatorn. Modellens samplingstid T_s väljs till 0,0005 s, vilket uppfyller alla ovanstående kriterier. I figur 2.12 visas trottelvinkel och varvtal då ett trottelsteg påförs modellen för fyra olika samplingstider; 0,0005 s; 0,002 s; 0,008 s samt 0,06 s. Endast den längsta steglängden 0,06 s, mer än hundra gånger större än den valda samplingstiden, ger ett fel som är märkbart.



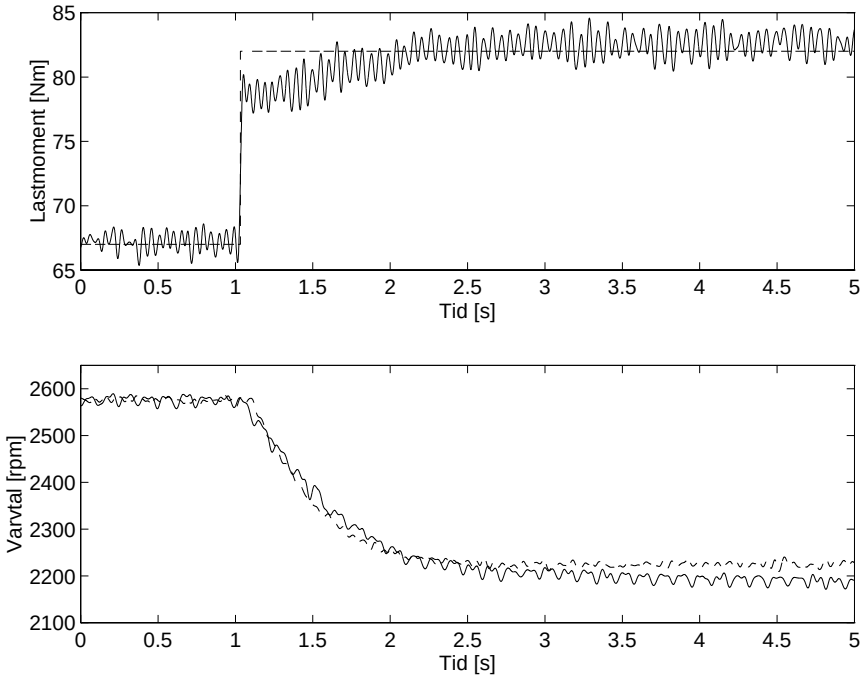
Figur 2.12. Trottelvinkel och varvtal då ett trottelsteg påförs modellen för fyra olika samplingstider; 0,0005 s (heldragen); 0,002 s (streckad linje); 0,008 s (prickad linje) samt 0,06 s (streckprickad linje). Endast den längsta steglängden orsakar ett märkbart fel.

2.7 Modellvalidering

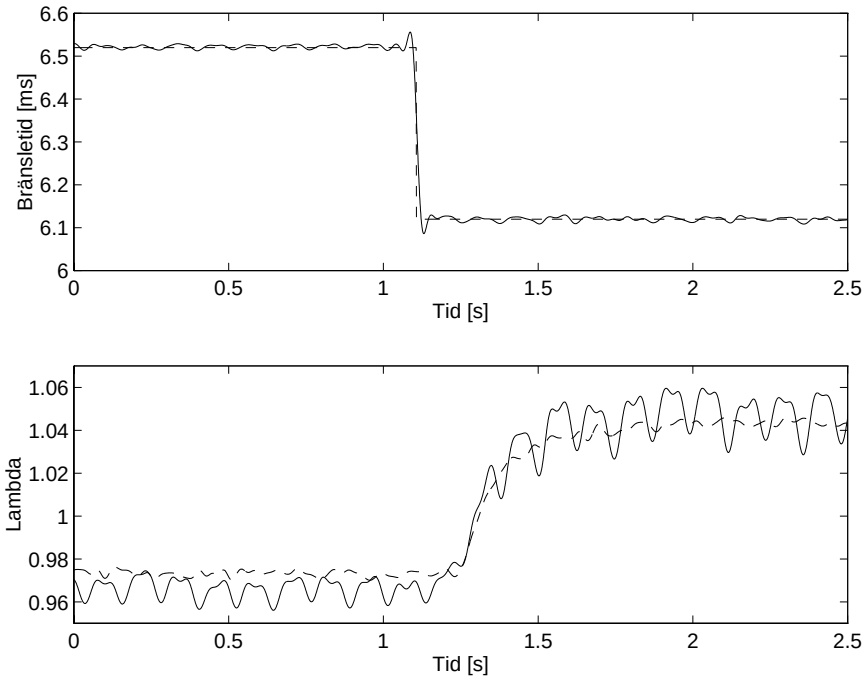
Modellen valideras genom att lägga på steg som exciterar motorns dynamik och jämföra modellens utsignaler med mätdata från motorn. Ett trottelsteg med varvtalsregulator inkopplad används för att validera luftdynamiken och ett momentsteg, utan varvtalsreglering, används för validering av vevaxelns dynamik. I båda fallen hålls lambda nära ett av en lambdaregulator. Bränslefilmsdynamiken samt lambdasensordynamiken valideras med ett bränsletidssteg, då alla regulatorer är urkopplade. Resultaten av valideringen för luftdynamiken visas i figur 2.13. Figur 2.14 visar valideringen av vevaxelns dynamik, och figur 2.15 åskådliggör valideringen av bränslefilmsdynamiken och lambdasensordynamiken. Figurerna visar att dynamiken är väl beskriven, medan nivåerna inte är riktigt korrekta. Detta är dock inte allvarligt, eftersom modellen skall åskådliggöra motorns dynamiska uppförande så bra som möjligt och det är inte lika viktigt att nivåerna är helt identiska med den verkliga motorns nivåer. Skillnaden på störningsamplituden mellan mätningarna på modellen och mätdata från motorn beror på de oförklarliga störningarna kring 8 Hz. Dessa kan inte filtreras bort, eftersom filtreringen skulle påverka motorns dynamik. Störningarna är mest signifikanta i lambdassensorsvaret från motorn, se figur 2.15.



Figur 2.13. Validering av luftdynamiken. De heldragna kurvorna beskriver mätdata från motorn och de streckade modellens utsignaler. Den övre figuren visar trottelsteg och den undre trycksvaren. Figuren visar att dynamiken är väl beskriven, medan modellen inte åskådliggör nivåerna riktigt korrekt.



Figur 2.14. Validering av vevaxelns dynamik. De heldragna kurvorna beskriver mätdata från motorn och de streckade modellens utsignaler. Den övre figuren visar momentsteg och den undre varvtalssvaren. Figuren visar att dynamiken är mycket väl beskriven, medan den understa nivån är något hög.



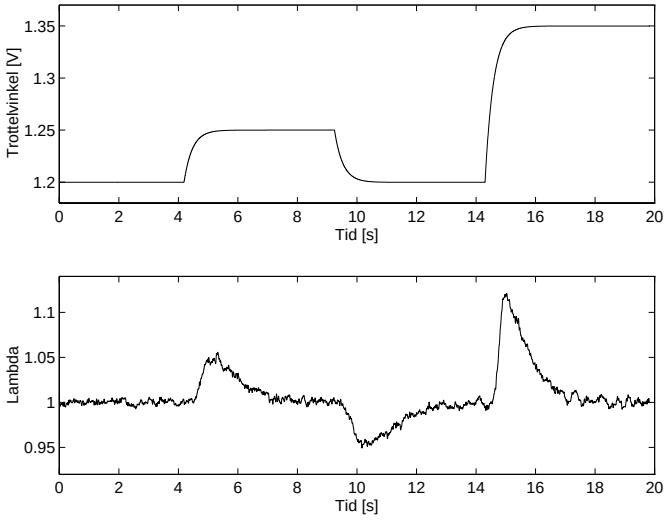
Figur 2.15. Validering av bränslefilmsdynamiken och lambdasensordynamiken. De heldragna kurvorna beskriver mätdata från motorn och de streckade modellens utsignaler. Den övre figuren visar bränsletidssteg och den undre lambdasensorsvaret. Figuren visar att dynamiken är väl beskriven och att även nivåerna i princip är korrekta. Notera de oförklarliga störningarna på ca. 8 Hz.

3 Lambdaregulator

Kapitlet inleds med en diskussion om svårigheter med lambdareglering. Därefter kommer ett avsnitt om realtidsimplementeringen av lambdaregulatorn, där regulatorns olika delar beskrivs i detalj. Avslutningsvis kommer en analys av regulatorns robusthet mot fel i modellbaserade mappar och konstanter.

3.1 Svårigheter med lambdareglering

Det är inte svårt att uppnå en bra lambdareglering i stationärt tillstånd, vid konstant tryck och varvtal, en PI-regulator med parameterstyrning är då ofta tillräcklig. Problemen uppstår när transienter, exempelvis trottelsteg, påförs systemet. Figur 3.16 visar lambda för en återkopplad PI-regulator vid tre snabba trottelöppningar då n=1800 rpm. Det tar tid för de förbrända gaserna att nå lambdasensorn, som inte heller har en försumbar responstid. Detta ger upphov till en lambdaspik, eftersom den återkopplade regulatorn inte hinner med att reglera ut trottelsteget. För att kompensera för de snabba transienterna måste en bra lambdaregulator därför ha någon form av framkoppling. En annan svårighet med lambdareglering är bränslefilmsdynamiken, d.v.s. dynamiken som uppstår då en del av det insprutade bränslet fastnar och bildar en film på insugningsrörens väggar. Lambdaregulatorn måste kunna eliminera eller i alla fall lindra effekterna av bränslefilmen. Motorns egenskaper förändras över tiden, beroende på slitage och åldring, vilket gör att mappar kalibrerade vid produktionen blir felaktiga. Modellosäkerheten i regulatorns mappar minskas kraftigt om adaptivitet införs.



Figur 3.16. Lambda för trottelsteg med PI-reglering då n=1800 rpm. Den övre figuren visar trottelvinkeln och den undre lambda. Observera spikarna som beror på att regulatorn får informationen om stegen för sent.

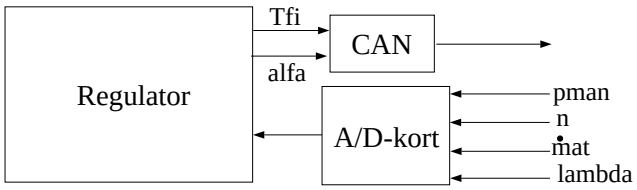
3.2 Realtidsimplementering av lambdaregulatorn

En lambdaregulator som skall klara av att kompensera för snabba transienter kan arbeta i två moder, en mod vid stationärt tillstånd och en annan mod vid transienter. Med hjälp av en transientdetektor kopplas en observatör in för att kompensera för snabba

ändringar, som t.ex. trottelsteg. Efter transienten kopplas en PI-regulator med parameterstyrning in för att reglera lambda i stationärt tillstånd. Fördelen med uppdelningen i moder är att regulatorn konstrueras för optimal funktion i de olika moderna, transientregleringen optimeras för reglering av snabba förändringar och PI-regulatorn för långsamma förändringar och stationära tillstånd. Inverkan av bränslefilmsdynamiken löses med en olinjär bränslekompensator som bara kopplas in vid transienter, eftersom kompensatorn är överflödig i stationärt tillstånd. Uppdatering av en adaptiv mapp införs också för att eliminera fel i den modellbaserade mappen, som beror på åldring och slitage av motorn. Detta leder till att regleralgoritmen får följande utseende:

```
Inläsning av insignaler
Konvertering av insignaler
Filtrering av insignaler
Transientdetektering
PI-reglering eller transientreglering
Bränslekompensering
Konvertering av utsignaler
Utställning av utsignaler
Uppdatering av luftmassflödesmapp
```

Lambdaregulatorn implementeras, liksom modellen, som en process i ett realtidssystem. Realistiska simuleringar åstadkoms genom att efterlikna gränssnittet mellan regulator och motorcell så mycket som möjligt. Lambdaregulatorn tar emot fyra insignaler från modellen via ett A/D-kort; trycket i insugningsröret p_{man}, varvtalet n, luftmassflödet genom trotteln $\dot{m}_{at}$ samt lambda. Regulatorn sänder två styrsignaler till modellen via CAN-bussen; bränsletiden T_{fi} och trottelvinkeln α. Figur 3.17 visar en schematisk skiss över regulatorns ut- och insignaler. Lambdaregulatorns samplingstid T_{sr} väljs till 0,004 s, vilket är tillräckligt långsamt för att inte överbelasta datorn och tillräckligt snabbt för att uppnå en bra lambdareglering.



Figur 3.17. Regulatorn med alla ut- och insignaler.

Filter

Design av filter för undertryckande av mätstörningar är viktigt. Det gäller att konstruera filtren så att ingen viktig information går förlorad, samtidigt som störningar dämpas så mycket som möjligt. I regulatorn används tredje ordningens butterworthfilter med gränsfrekvens 22,5 Hz. Dessa implementeras enligt en direktform II_t-realisering.

Transientdetektor

En regulator med två moder måste ha någon typ av detektor, som kan avgöra om modomkoppling skall ske. Lambdaregulatorns transientdetektor känner av ett snabbt

trottelsteg och kopplar in observatören. När sedan transienten klingar av kopplas PI-regulatorn in. Trottelns referensvärde används för att detektera transienter. Derivatan av det filtrerade referensvärdet beräknas och om det är högre än ett tröskelvärde antas att en transient påbörjats. Den framkopplade observatören kopplas in under 1,6 sekunder, vilket är tillräckligt länge för att alla transienter skall klinga av. Därefter kopplas PI-regulatorn in igen för reglering i stationärt tillstånd och för långsamma förändringar. En fast tid gör att regulatorn inte fastnar i transientreglering, vilket kan få obehagliga konsekvenser för stabiliteten.

Regulator

Regulatorn har två moder; en mod för transientreglering och en mod för reglering av långsamma förändringar och stationära tillstånd. Regleringen i stationärt mod sköts av en PI-regulator med parameterstyrning, där parametrarna bestäms enligt Zieger-Nichols inställningsregler, för två olika tryck och fem olika varvtal. Styrsignalen, bränslemassflödet $\dot{m}_{fi}$, kan inte mättas i modellens giltighetsområde och därför behöver inte algoritmen ta hand om någon integratoruppvridning. Detta gör att algoritmen får nedanstående utseende.

$$\begin{aligned} (K, T_i) &= f(n, p_{man}) \\ I &= I + \frac{KT_{sr}}{T_i}(\lambda - 1) \\ \dot{m}_{fi} &= I + K(\lambda - 1) \end{aligned}$$

Vid transienter kopplas en observatör in som skattar de snabba förändringarna i trycket, genom att mäta varvtalet och trycket i insugningsröret, samt använda mappen för luftmassflödet in till cylindrarna $\dot{m}_{ac}(n, p_{man})$. Mappen uppdateras under långsamma förändringar och stationära tillstånd, vilket eliminerar en stor del av felen, som i den verkliga motorn bl.a. beror på slitage och åldring. Observatören använder sig av Eulers formel för att prediktera trycket, vilket gör att den får följande utseende:

$$\hat{p}_{man} = p_{man} + \frac{C_1 T_{sr}}{2}(\dot{m}_{at} - \dot{m}_{ac}(n, p_{man})) \tag{3.22}$$

Det skattade trycket används sedan till att beräkna det önskade insprutade bränslemassflödet $\dot{m}_{fi,önskad}$, enligt nedanstående ekvation, eftersom det verkliga luft/bränsleförhållandet skall hålla sig nära det stökiometriska luft/bränsleförhållandet.

$$\dot{m}_{fi,önskad} = \frac{\dot{m}_{ac}(n, \hat{p}_{man})}{(A/F)_{stök}} \tag{3.23}$$

Skattning av det stökiometriska luft/bränsleförhållandet

Det stökiometriska luft/bränsleförhållandet används av observatören vid transientregleringen, se ekvation 3.23. $(A/F)_{stök}$ är dock inte känd exakt, utan varierar för bensin med samma oktantal. Detta gör att regulatorn måste skatta $(A/F)_{stök}$ under uppstarten av regulatorn för att uppnå en tillfredsställande reglering i transientmod. Luftmassflödet genom trotteln $\dot{m}_{at}$ mäts och bränslemassflödet $\dot{m}_{fi}$ beräknas, och om lambda hålls

kring ett av PI-regulatorn och stationärt tillstånd upprätthålls kan det stökiometriska luft/bränsleförhållandet bestämmas enligt:

$$(\hat{A/F})_{stök} = \frac{\dot{m}_{at}}{\lambda \dot{m}_{fi}} = \frac{\dot{m}_{at}}{\dot{m}_{fi}}$$

Genom att medelvärdesbilda över 200 värden elimineras eventuella störningar.

Olinjär bränslekompensator

Inverkan av bränslefilmsdynamiken på lambda lindras med en olinjär bränslekompensator. Genom att försöka skatta hur mycket bränsle som fastnar på väggarna och kompensera bort dynamiken dämpas effekterna av bränslefilmen. I Hendricks [6] beskrivs en kompensator som fungerar bra och bygger på en MVEM, och där ekvationerna 2.18 och 2.21 används till att få fram nedanstående uttryck:

$$\dot{m}_{fi} = \frac{1}{1 - X(n, p_{man})}(\dot{m}_f(t) - \frac{1}{\tau_{fp}(n, p_{man})} m_{fp}(t))$$

Bränslemassflödet in till cylindrarna $\dot{m}_f$ är samma sak som $\dot{m}_{fi,önskad}$ och bränslefilmsmassan $m_{fp}(t)$ är samma tillstånd som finns i modellen, vilket ger nedanstående algoritm för den olinjära bränslekompensatorn.

$$\begin{aligned} \dot{m}_{fi} &= \frac{1}{1 - X(n, p_{man})}(\dot{m}_{fi,önskad} - \frac{1}{\tau_{fp}(n, p_{man})} \hat{m}_{fp}(t)) \\ \hat{m}_{fp}(t + T_{sr}) &= (1 - \frac{T_{sr}}{\tau_{fp}(n, p_{man})})\hat{m}_{fp}(t) + T_{sr} X(n, p_{man}) \dot{m}_{fi} \end{aligned}$$

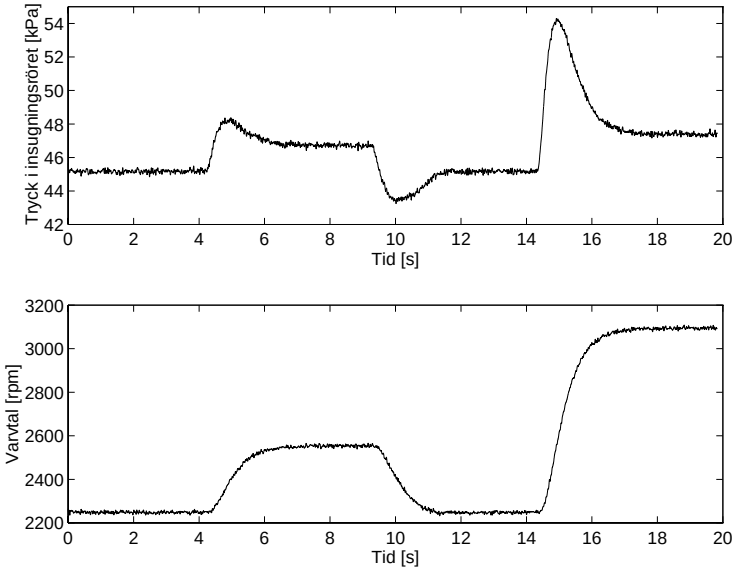
Uppdatering av luftmassflödesmapp

Under PI-regleringen jämförs luftmassflödet genom trotteln $\dot{m}_{at}$ och det skattade luftmassflödet in till cylindrarna $\dot{m}_{ac}(n, p_{man})$. Dessa storheter bör vara identiska i stationärt tillstånd, eftersom det strömmar in lika mycket luft i insugningsrören, som det strömmar in till cylindrarna ($\dot{p}_{man}$=0 i ekvation 2.10). Om de inte är identiska måste mappen för $\dot{m}_{ac}(n, p_{man})$ vara felaktig och bör därför uppdateras. I Bergkvist [3] beskrivs en algoritm för uppdatering av mappar, som är enkel och visar sig fungera bra. Uppdateringsalgoritmen bygger på en avståndsberoende viktning och att de mappade värdena tas fram med linjär interpolation. Ekvation 2.16 används som utgångspunkt för en beskrivning av algoritmen. Algoritmen får då nedanstående utseende, där det visas att koefficienterna som framkommer vid den linjära interpolationen också används vid uppdateringen, vilket är praktiskt då inga nya koefficienter behöver beräknas. δ anger hur stor del av felet i mappen som skall användas vid uppdateringen av mappen, där δ=1 betyder full uppdatering. Ett lägre δ leder till att mappen uppdateras långsammare och att känsligheten för störningar minskar. Förändringar i $\dot{m}_{ac}(n, p_{man})$ sker långsamt, vilket gör att δ kan väljas så lågt som 0,005, eftersom uppdateringen av mappen inte behöver vara snabb.

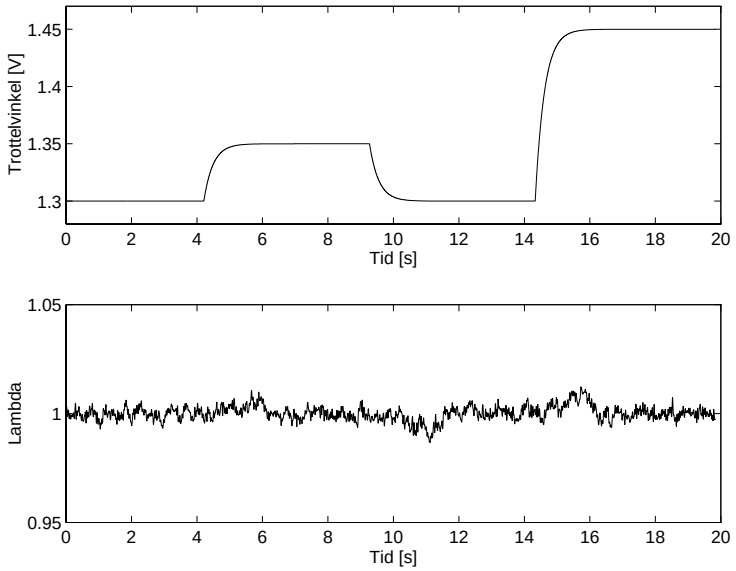
$$
\begin{aligned}
\dot{m}_{ac}(n, p_{man}) &= \gamma_{11}\dot{m}_{ac,11} + \gamma_{21}\dot{m}_{ac,21} + \gamma_{12}\dot{m}_{ac,12} + \gamma_{22}\dot{m}_{ac,22} \\
\hat{e}_m &= \dot{m}_{at} - \dot{m}_{ac}(n, p_{man}) \\
\dot{m}_{ac,ij,ny} &= \dot{m}_{ac,ij} + \frac{\delta\, \gamma_{ij}\, \hat{e}_m}{{\gamma_{11}}^2 + {\gamma_{21}}^2 + {\gamma_{12}}^2 + {\gamma_{22}}^2}
\end{aligned}
$$

3.3 Verifiering av lambdaregulatorn

Lambdaregulatorns funktion verifieras med tre trottelsteg. Figurerna 3.18 och 3.19 visar tryck, varvtal, trottelvinkel samt lambda under stegen. Lambdaregulatorn visar sig klara av de snabba transienterna mycket bra. Det finns inga tendenser till spikar och lambda ligger stadigt kring ett.



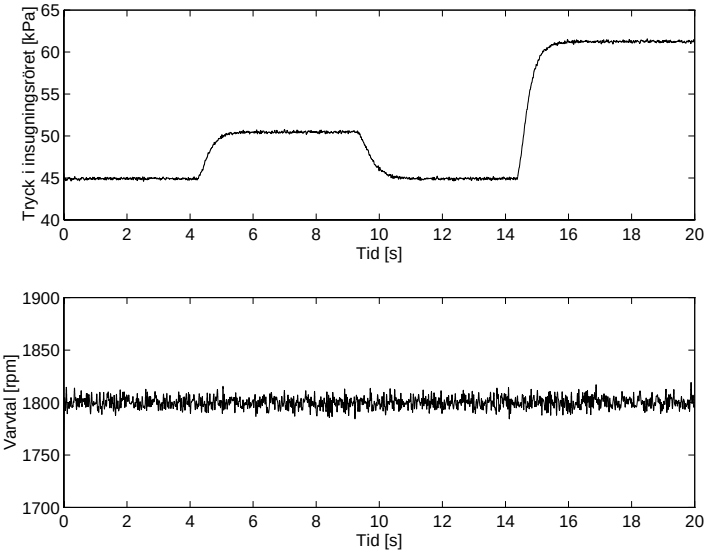
Figur 3.18. Arbetspunktens variation under trottelstegen. Övre figuren visar trycket i insugningsröret och den nedre visar varvtalet. Tryckspikarna i början av varje steg orsakas av att vevaxelns dynamik är långsammare än luftdynamiken. Trycket i insugningsröret minskar successivt, när det ökande varvtalet gör att mer och mer luft sugs in i cylindrarna.



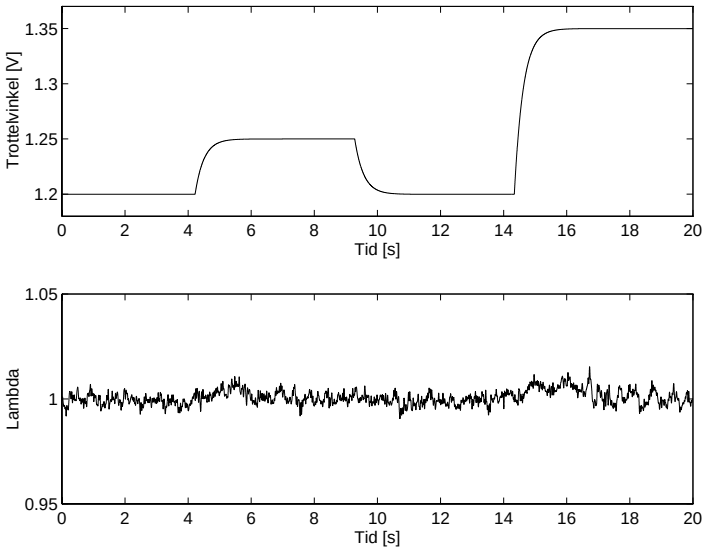
Figur 3.19. Trottelvinkel och lambda under trottelsteg med PI- och transientreglering. Den övre figuren visar trottelvinkeln och den undre lambda. Lambdaspikarna har försvunnit och lambda ligger stadigt kring ett.

3.4 Lambdaregulatorns robusthet mot fel i mappar och konstanter

Det är viktigt att regulatorer har en stor robusthet mot modellosäkerhet och därför är det väsentligt att undersöka robustheten i lambdaregulatorn. Fel i modellbaserade mappar och konstanter införs och lambdasensorsvaret på trottelsteg analyseras. Identiska trottelsteg vid identiska arbetspunkter utförs för att möjliggöra jämförelser. Trottelsteg på en regulator utan fel, se figur 3.20 och 3.21, används som referens. Observera att varvtalet hålls konstant under trottelstegen, vilket gör att uppdateringen av mappar förenklas betydligt. Med ett varierande varvtal skulle uppdateringarna bli otroligt tidsödande, utan att förutsättningarna i grunden förändras.



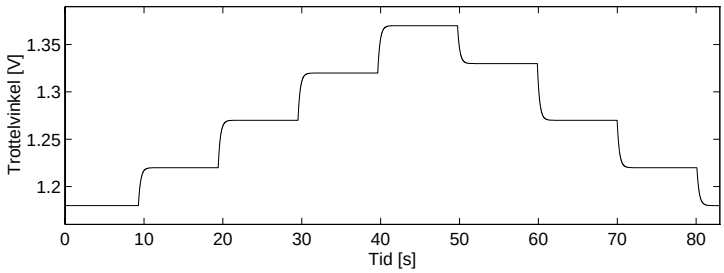
Figur 3.20. Arbetspunktens variation under trottelstegen då n=1800 rpm. Övre figuren visar trycket i insugningsröret och den undre varvtalet. Konstant varvtal leder till stora tryckökningar.



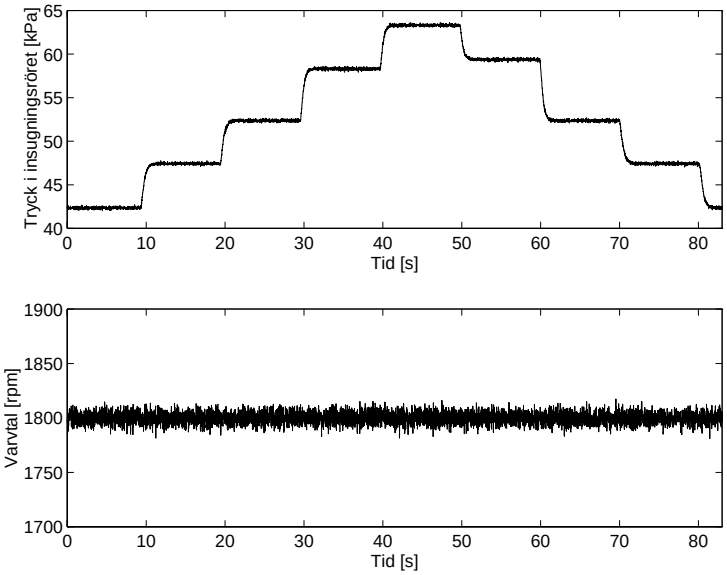
Figur 3.21. PI- samt transientreglering utan fel då n=1800 rpm. Trottelvinkeln ses i den övre figuren och lambda i den undre. Utan fel hålls lambda mycket nära ett.

Uppdateringscykel

Fel, orsakade av bl. a. motorslitage och åldring, elimineras genom att använda en adaptiv $\dot{m}_{ac}(n, p_{man})$-mapp. I robusthetsanalysen görs en jämförelse mellan regleringen före och efter mappuppdateringen. Cykeln som används till uppdateringen av $\dot{m}_{ac}(n, p_{man})$-mappen åskådliggörs i figurerna 3.23 och 3.22. Dessa figurer visar arbetspunktsvariationer respektive trottelsteg under cykeln.



Figur 3.22. Trottelstegen under uppdateringscykeln.

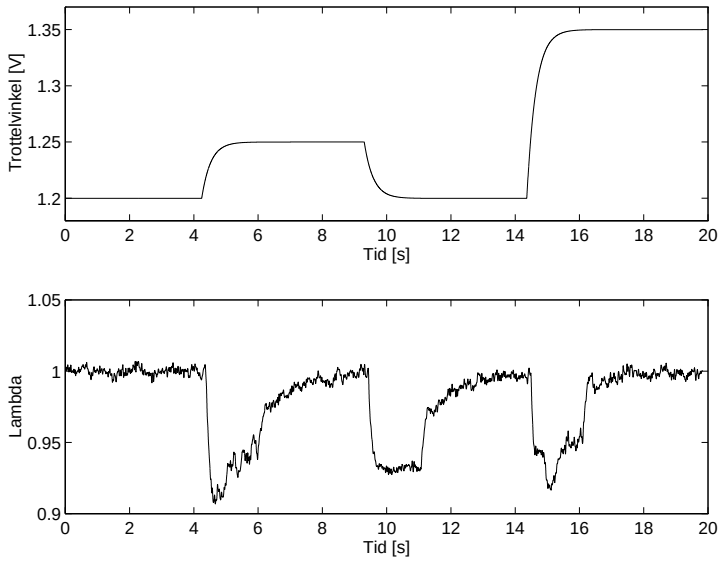


Figur 3.23. Arbetspunktens variation under uppdateringscykeln. Den övre figuren visar trycket i insugningsröret och den undre varvtalet. Trycket ökar vid trottelsteg uppåt, eftersom varvtalet hålls kring n=1800 rpm.

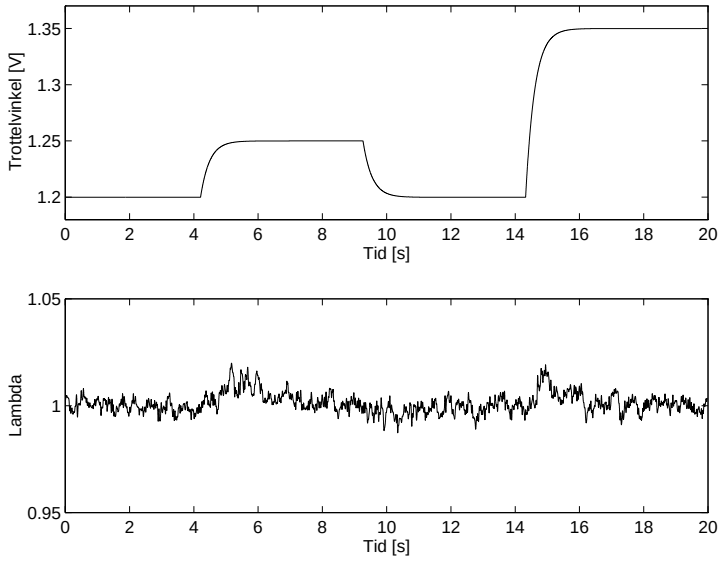
Fel i luftmassflödesmapp

Mappen för luftmassflödet in till cylindrarna, $\dot{m}_{ac}(n, p_{man})$, används i observatören. Genom att öka alla värden i mappen med sju procent påförs ett stort fel, men eftersom det sker en kontinuerlig uppdatering av mappen minskas felet successivt. I figur 3.24 visas lambdasensorsvaret på de tre trottelstegen, då ännu ingen uppdatering av mappen skett. Mappen innehåller för höga värden, vilket gör att observatören överkompenserar och sprutar in för mycket bränsle. Detta leder slutligen till att lambda blir alldeles

för litet. Figur 3.25 visar lambda efter tre uppdateringscykler. Mappen adapteras till korrekta värden, vilket får till följd att regleringen blir mycket bättre.



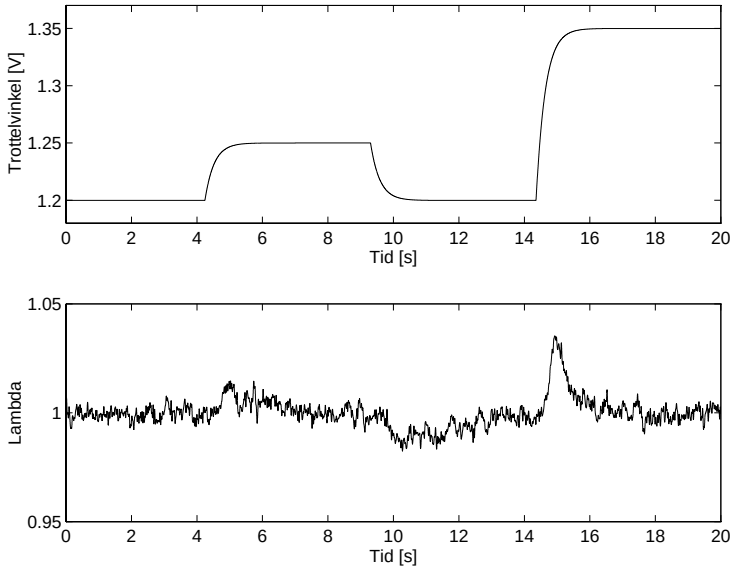
Figur 3.24. PI- samt transientreglering då fel påförs $\dot{m}_{ac}(n, p_{man})$-mappen. Alla värden i mappen ökas med sju procent. Varvtalet hålls kring n=1800 rpm. Figuren visar trottelvinkel och lambda under trottelstegen, innan någon uppdatering skett. Observatören underkompenserar och sprutar in för mycket bränsle, beroende på för höga värden i mappen. Detta får till följd att lambda får mycket låga värden.



Figur 3.25. PI- samt transientreglering då fel påförs $\dot{m}_{ac}(n, p_{man})$-mappen. Alla värden i mappen ökas med sju procent. Varvtalet hålls kring n=1800 rpm. Figuren visar trottelvinkel och lambda under trottelstegen, då mappen adapterats under tre uppdateringscykler. Lambda håller sig nära ett efter uppdateringen och felen har i princip eliminerats.

Fel i bränslefilmsparametrar

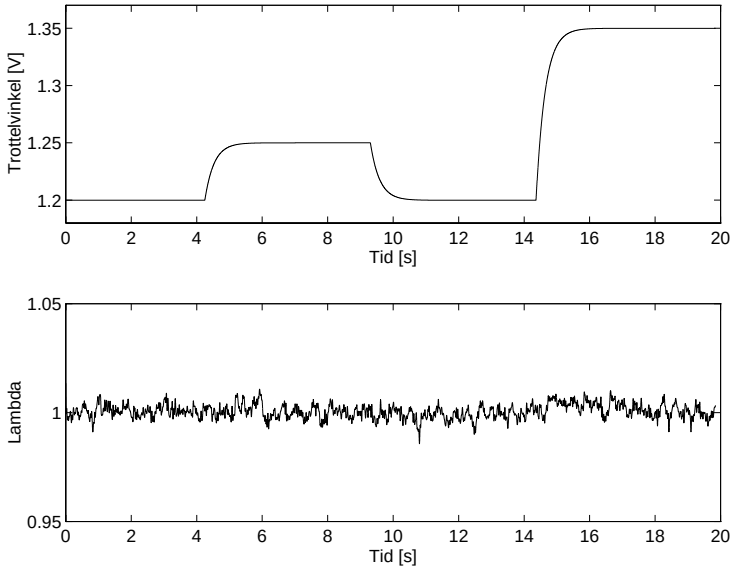
Den olinjära bränslekompensatorn använder mappar av bränslefilmsdepositionsfaktorn $X(n, p_{man})$ och tidskonstanten för bränslefilmen $\tau_{fp}(n, p_{man})$. Fel påförs parametrarna genom att minska alla värden i $X(n, p_{man})$- och $\tau_{fp}(n, p_{man})$-mapparna med tjugo procent. I figur 3.26 visas lambdasensorsvaret på de tre trottelstegen. Den förstnämnda mappen innehåller för låga värden. Detta får till följd att regulatorn underkompenserar och sprutar in för liten mängd bränsle, vilket gör att små lambdaspikar uppträder. De lägre värdena i den sistnämnda mappen leder till att lambda går långsammare mot ett.



Figur 3.26. PI- samt transientreglering då fel påförs bränslefilmsparametrarna i den olinjära kompensatorn. Alla värden i $X(n, p_{man})$- och $\tau_{fp}(n, p_{man})$-mapparna minskas med tjugo procent. Varvtalet hålls kring n=1800 rpm. Den övre figuren visar trottelvinkeln och den undre lambda. Eftersom bränsledepositionsfaktorn är för liten underkompenserar regulatorn och sprutar in för liten mängd bränsle, vilket leder till små lambdaspikar. Minskningen av värdena i $\tau_{fp}(n, p_{man})$-mappen gör att regulatorn kompenserar för en större tidskonstant. Detta får till följd att lambda går långsammare mot ett.

Fel i observatörskonstant

I observatören används konstanten från luftdynamiksekvationen (C_1 i ekvation 3.22). Figur 3.27 visar lambdasensorsvaret efter de tre trottelstegen då ett stort fel införs, genom att öka konstanten med fyrtio procent. Lambda håller sig mycket nära ett och det påförda felet påverkar inte lambdaregleringen.



Figur 3.27. PI- samt transientreglering då fel påförs konstanten C_1 i observatörsalgoritmen. Konstanten ökas med fyrtio procent och varvtalet hålls kring n=1800 rpm. Den övre figuren visar trottelvinkel och den undre lambda, som hålls mycket nära ett och påverkas inte märkbart av felet.

4 Slutsatser och utvidgningar

En motormodell har tagits fram för att validera en lambdaregulator. Den har fyra tillstånd och dess olinjära statiska samband modelleras med mappar. Körningar på en riktig motor utförs i syfte att generera mätdata till modellidentifierngen. Modellvalideringen visar att modellens dynamik överensstämmer väl med den verkliga motorn, däremot är de statiska nivåerna inte riktigt korrekta. Motormodellen används sedan för konstruktion, testning och verifiering av lambdaregulatorn.

En lambdaregulator, baserad på den dynamiska motormodellen, har konstruerats och implementerats. Regulatorn arbetar i två moder, där den ena moden reglerar vid stationära tillstånd och vid långsamma förändringar och den andra vid snabba förändringar. Regleringen vid snabba förändringar sköts av en observatör och en olinjär kompensator, vilka båda innehåller modellbaserade mappar. Regleringen vid stationära tillstånd och vid långsamma förändringar hanteras av en PI-regulator med parameterstyrning. En adaptiv mapp används för att minska modellosäkerheten i lambdaregulatorns observatör. Realistiska simuleringar åstadkoms genom att efterlikna det verkliga gränssnittet mellan regulator och motor. Modellen och regulatorn placeras i olika datorer, och kommunikationen mellan dessa sköts via CAN-buss samt D/A- och A/D-kort. Resultaten av simuleringarna visar att lambdaregulatorn fungerar bra och håller lambda mycket nära ett, om felen i modellbaserade mappar och konstanter är försumbara. Regulatorn fungerar också tämligen väl, då fel påförs mapparna och konstanterna. Felet i den adaptiva mappen elimineras i princip efter tre uppdateringscykler (ca. fyra minuter), vilket är tillräckligt snabbt eftersom förändringar i luftmassflödesmappen sker långsamt.

4.1 Utvidgningar

Det finns många intressanta utvidgningar till arbetet, och en är att förfina modellen, exempelvis genom att införa finare mappar och använda flera mätningar. En sådan förfinad modell kan sedan användas till testning, verifiering och konstruktion av andra reglersystem.

En annan utvidgning kan vara att införa adaptiva mappar för bränslefilmsparametrarna. Robusthetsanalysen visar att modellfel i dessa mappar ger upphov till små lambdaspikar och genom att uppdatera mapparna kan spikarna minskas.

Lambdaregulatorn är konstruerad på ett strukturerat sätt. Varje del i regulatorn, t.ex. transientdetekteringen och bränslefilmskompenseringen, är avgränsad från de övriga, vilket gör det möjligt att utveckla och förfina detaljer i någon del, utan att påverka andra delar av regulatorn. På detta sätt kan regulatorn utvecklas och förfinas ytterligare för optimal funktion.

Gränssnittet under simuleringarna mellan regulator och modell är identiskt med gränssnittet i motorlaboratoriet mellan regulator och motorcell. Detta gör det möjligt att implementera regulatorn på den verkliga motorn, utan att genomföra några större förändringar.

Referenser

[1] Göran Almkvist. *Transient Air To Fuel Ratio Response in a Fuel Injected S.I. Engine.* Phd thesis ISBN 91-7197-097-5, Department of Thermo- and Fluid Dynamics, Chalmers University of Technology, Göteborg, Sweden, 1995.

[2] C. F. Aquino. Transient A/F control characteristics of the 5 liter central fuel injection engine. *SAE-Technical Paper Series*, 1981.

[3] Morgan Bergkvist. Simulering av luft-bränslereglering med adaptiv parameteruppdatering för ottomotorer. Master's thesis LiTH-ISY-EX-1721, Department of Electrical Engineering, Linköping University, Linköping, Sweden, Maj 1997.

[4] BOSCH. *Automotive Electric/Electronic Systems.* Number ISBN 3-18-419121-4. Robert Bosch GmbH, 2nd edition, 1995.

[5] Elbert Hendricks and Spencer C. Sorensen. Mean value modelling of spark ignition engines. *SAE-Technical Paper Series*, 1990.

[6] Elbert Hendricks, Thomas Vesterholm, and Spencer C. Sorensen. Nonlinear, closed loop, SI engine control observers. *SAE-Technical Paper Series*, 1992.

[7] John B. Heywood. *Internal Combustion Engine Fundamentals.* Number ISBN 0-07-100499-8. McGraw-Hill, Singapore, 1989.

[8] Ulrich Lenz and Dierk Schroeder. Artificial intelligence for combustion engine control. *SAE-Technical Paper Series*, 1996.

[9] Ulrich Lenz and Dierk Schroeder. Transient air-fuel ratio control using artificial intelligence. *SAE-Technical Paper Series*, 1997.

Bilaga A: Fordonssystems motorlaboratorium

Bilagan beskriver utrustningen i Fordonssystems motorlaboratorium.

Motorspecifikationer

Motorn som används vid modelleringen är en SAAB 2.3L-motor, utrustad med extra givare för mätning av cylindertryck, jonströmmar, luftmassflöde o.s.v. Den elektroniska styrenheten kallas SELMA och har utvecklats av MECEL AB. SELMA styr bl.a. bränsleinsprutningsventilerna och sköter kommunikationen via en CAN-buss, vilket gör det möjligt att sända och ta emot information under motorkörningar. Motorns specifikationer är:

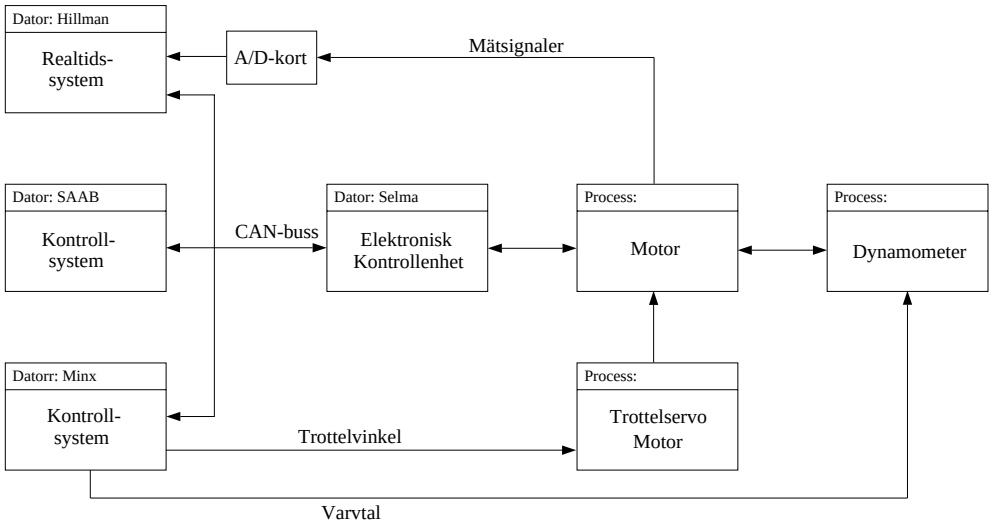
Motortyp:	Fyrtaktsmotor med fyra cylindrar.
Ventiler:	Fyra ventiler per cylinder drivna av dubbla överliggande kamaxlar.
Cylindervolym:	2290 cm^3 (2.3 liter)
Borrning:	90 mm
Slag:	90 mm
Maximal motoreffekt:	150 hk (110 kW)
Maximalt utmoment:	212 Nm
Vikt:	$\approx$ 160 kg
Serienummer:	B2341.4N10M219569

Dynamometer

Körsituationer, varvtal och lastmoment simuleras med en dynamometer (motorbroms). Dynamometern som finns i motorlaboratoriet är en DYNASYN NT 85 servomotor/generator från Schenk. Den bör inte belastas med motormoment som är högre än 150 Nm.

Datorer

Utöver SELMA används tre vanliga PC-datorer under motorkörningarna. Dessa tre datorer kommunicerar också via CAN-bussen. Figur A.1 visar en skiss över datorerna och kommunikationen mellan dessa och motorn samt dynamometern. Hillman, en av datorerna, innehåller ett realtidssystem som använder RTKernel 4.0. Det är i den datorn som varvtals- och tryckregulatorn implementeras under motorkörningarna. I Minx, en annan dator, ändras referensvärden för exempelvis trottelvinkel och varvtal. Dessutom kan intressanta variabler mätas via ett A/D-kort. På SAAB, en annan av de tre datorerna, kan variabler som används av SELMA styras. Som exempel på dessa variabler kan bränsletid och lambda nämnas.



Figur A.1. Konfiguration av styrsystem i Fordonssystems motorlaboratorium..

Bilaga B: Programkod för modell

Nedan visas programkoden för realtidsimplementeringen av modellen. De stora mapparna för det effektiva momentet $M_{eff}(\lambda, n, p_{man})$, samt luftmassflödena $\dot{m}_{ac}(n, p_{man})$ och $\dot{m}_{at}(\alpha, p_{man})$ utelämnas, eftersom de inte tillför beskrivningen något. Detsamma gäller vektorerna för lambda och $\dot{m}_{at}$, som används av funktionen interpol vid konverteringen av utsignaler.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <math.h>

#include "main.h"
#include "dstruct.h"
#include "command.h"
#include "can.h"
#include "display.h"
#include "serial.h"
#include "nidaqcns.h"
#include "nidaq.h"

#include <RTKernel.h>
#include <timer.h>

#define frand() ((double) rand() / (RAND_MAX+1.0))


// Linjärsökning av vektor.
int search_vector(double val, const double* vector, int size)
{
        int found=0, count=0, index=-1;
        while ((!found) && (count < size))
        {
                if ((vector[count] <= val) && (vector[count+1] > val))
                        found=1;
                else
                        count=count+1;
        }

        if (found)
                index=count;

        return index;

}

// Linjär interpolation.
double interpol(double xval, double yval,
double* x, double* y, double z[][2])
{
        double k1, k2, z1, z2, zval;

        k1=(xval-x[0])/(x[1]-x[0]);
        k2=(yval-y[0])/(y[1]-y[0]);

        z1=z[0][0]+k1*(z[0][1]-z[0][0]);
```

```
        z2=z[1][0]+k1*(z[1][1]-z[1][0]);

        zval=z1+k2*(z2-z1);

        return zval;
}


// Mappning av momentet och mac. Funktionen returnerar också arbetspunkts-
// rektangeln som används vid mappningen.
double mapping1(double n_val, double p_val, const double* n_vector,
                const double* p_vector, const double z_vector[][14],
                double* n_help, double* p_help, double z_help[][2])
{
        int i, j, index1, index2, out_of_range=0;
        int length_n=14, length_p=8;
        double zval=0;

        index1=search_vector(n_val,n_vector,length_n);
        index2=search_vector(p_val,p_vector,length_p);

        if ((index1 >=0) && (index2 >=0))
        {
                for (i=0; i<=1; i=i+1)
                {
                        n_help[i]=n_vector[index1+i];
                        p_help[i]=p_vector[index2+i];
                        for (j=0; j<=1; j=j+1)
                        {
                                z_help[j][i]=z_vector[index2+j][index1+i];
                                if (z_vector[index2+j][index1+i]==0)
                                {
                                        out_of_range=1;
                                        break;
                                }
                        }

                }

                if (!out_of_range)
                        zval=interpol(n_val,p_val,n_help,p_help,z_help);

        }
        return zval;

}

// Mappning av mat. Funktionen returnerar också arbetspunktsrektangeln
// som används vid mappningen.
double mapping2(double alpha_val, double p_val, const double* alpha_vector,
                const double* p_vector, const double z_vector[][18],
                double* alpha_help, double* p_help, double z_help[][2])
{
        int i, j, index1, index2, out_of_range=0;
        int length_alpha=18, length_p=8;
        double zval=0;
        index1=search_vector(alpha_val,alpha_vector,length_alpha);
        index2=search_vector(p_val,p_vector,length_p);
        if ((index1 >=0) && (index2 >=0))
```

```
        {
                for (i=0; i<=1; i=i+1)
                {
                        alpha_help[i]=alpha_vector[index1+i];
                        p_help[i]=p_vector[index2+i];
                        for (j=0; j<=1; j=j+1)
                        {
                                z_help[j][i]=z_vector[index2+j][index1+i];
                                if (z_vector[index2+j][index1+i]==0)
                                {
                                        out_of_range=1;
                                        break;
                                }
                        }

                }
                if (!out_of_range)
                        zval=interpol(alpha_val,p_val,alpha_help,
                                        p_help,z_help);
        }
        return zval;

}

// Mappning av bränslefilmsparametrar. Funktionen returnerar också arbets-
// punktsrektangeln som används vid mappningen.
void mapping3(double n_val, double p_val, double* n_help, double* p_help,
              double X_help[][2], double T_help[][2], double* lambda_val)
{
        int i, j, index1, index2, length_n=4, length_p=3;
        const double n_vector[4] = { 1580.0 , 2013.0 , 2717.0 , 3510.0 };
        const double p_vector[3] = { 39.0 , 51.0 , 59.0 };
        const double X_map[3][4] = { { 0.3, 0.3, 0.3, 0.05 },
                                { 0.2, 0.1, 0.05, 0.05 },
                                { 0.1, 0.05, 0.05, 0.05 } };
        const double T_map[3][4] = { { 2.0, 4.0, 4.0, 10.0 },
                                { 4.0, 8.0, 10.0, 10.0 },
                                { 8.0, 10.0, 10.0, 10.0 } };

        if (n_val <= 1580.0)
        {
                if (p_val <= 45.0)
                {
                        lambda_val[0]=0.3;
                        lambda_val[1]=2;
                }
                else
                {
                        if (p_val <= 55.3)
                        {
                                lambda_val[0]=0.2;
                                lambda_val[1]=4;
                        }
                        else
                        {
                                lambda_val[0]=0.1;
                                lambda_val[1]=8;
                        }
                }
```

```
        }
        else
        {
                if ((n_val >= 3510.0) || (p_val >= 59.0))
                {
                        lambda_val[0]=0.05;
                        lambda_val[1]=10;
                }
                else
                {
                        if (p_val <= 39.0)
                        {
                                lambda_val[0]=0.3;
                                lambda_val[1]=4;
                        }
                        else
                        {
                                index1=search_vector(n_val,n_vector,length_n);
                                index2=search_vector(p_val,p_vector,length_p);
                                for ( i=0 ; i <= 1 ; i=i+1 )
                                {
                                        n_help[i]=n_vector[index1+i];
                                        p_help[i]=p_vector[index2+i];
                                        for ( j=0 ; j <= 1 ; j=j+1 )
                                        {
                                                X_help[j][i]=X_map[index2+j]
                                                [index1+i];
                                                T_help[j][i]=T_map[index2+j]
                                                [index1+i];
                                        }
                                }
                                lambda_val[0]=interpol(n_val,p_val,n_help,
                                                        p_help,X_help);
                                lambda_val[1]=interpol(n_val,p_val,n_help,
                                                        p_help,T_help);
                        }
                }
        }
}

// Linjär endimensionell interpolation, används vid konvertering av utsignaler.
double linterpol(double val, const double* x_vector, const double* y_vector,
                 int size)
{
        int found=0, count=0;
        double k, zval;

        if (val <= x_vector[0])
                zval = y_vector[0];
        else
        {
                while ((!found) && (count < (size-1)))
                {
                        if ((x_vector[count] <= val) &&
                           (x_vector[count+1] > val))
                                found=1;
                        else
                                count=count+1;
```

```
                }
                if (!found)
                        zval = y_vector[size-1];
                else
                {
                        k=(y_vector[count+1]-y_vector[count])/
                        (x_vector[count+1]-x_vector[count]);
                        zval=y_vector[count]+k*(val-x_vector[count]);
                }
        }
        return zval;
}

// Mottagning av insignaler via CAN-buss.
int Recieve(int* value, int* recieved)
{
        CAN_FRAME message;
        int new_value=0;

        recieved[0] = (!GetCanMailBoxNow(&message,"Model-CanMailBox1"));
        if (recieved[0])
        {
                value[0]=((message.data[0]<<8)+message.data[1]);
                new_value=1;
        }
        recieved[1] = (!GetCanMailBoxNow(&message,"Model-CanMailBox2"));
        if (recieved[1])
        {
                value[1]=((message.data[0]<<8)+message.data[1]);
                new_value=1;
        }
        return new_value;
}



DECLARE_PROCESS(Model)

void Model()
{
        Semaphore sem;
        double time;
        double n_new, n_old, p_new, p_old, mfp_new, mfp_old;
        double lambda_new, lambda_old;
        double n_volt, p_volt, lambda_volt, mat_volt;
        double alpha_new, alpha_old;
        double M_eff, M_load, mat, mac, alpha, mfi, mf, tau_inv, X, mfp_dot;
        double lambda_real;
        double c1,c2,c3,c4,k1,k2,k3,k4,k5,k6;
        double n_brus=0, p_brus=0, lambda_brus=0, mat_brus=0;
        double n_out, p_out, lambda_out, mat_out;
        double n_help1[2]={0,0},p_help1[2]={0,0},mac_help[2][2]={{0,0},{0,0}};
        double n_help2[2]={0,0},p_help2[2]={0,0},M_help[2][2]={{0,0},{0,0}};
        double alpha_help[2]={0,0},p_help3[2]={0,0};
        double mat_help[2][2]={{0,0},{0,0}};
        double n_help4[2]={0,0},p_help4[2]={0,0},X_help[2][2]={{0,0},{0,0}};
        double T_help[2][2]={{0,0},{0,0}};
        double lambda_val[2]={0,0};
        int buf_size=325, index, t_del, count;
```

```
double lambda_del[325];
long int mfi_time, alpha_in;
int new_value, recieved[2]={0,0}, control_signals[2]={0,0};
int mfi_id=31, alpha_id = 32;
int in_old_square1=0, in_old_square2=0;
int in_old_square3=0, in_old_square4=0;
const int size_lambda=15, size_mat=25, size_t=5;

Port n_port, p_port, lambda_port, mat_port, M_port;

Time nextActivation;

sem = RegisterDouble(time);

// Skapande av "postlådor" för CAN-kommunikation.
CreateCanMailBox((DWORD)mfi_id, "Model-CanMailBox1");
CreateCanMailBox((DWORD)alpha_id, "Model-CanMailBox2");

RegisterPort(n_port);
RegisterPort(p_port);
RegisterPort(lambda_port);
RegisterPort(mat_port);
RegisterPort(M_port);

time = 0.0005;
M_port = 18;
n_port = 0;
p_port = 1;
lambda_port = 2;
mat_port = 3;

// Konstanter i ekvationer för uppdatering av tillstånd.
c1=46.395;
c2=4.360;
c3=12.000;
c4=3.600;
k1=c1*time;
k2=c2*time;
k3=1.000-c3*time;
k4=c3*time;
k5=1-c4*time;
k6=c4*time;

// Inledande arbetspunkt.
n_new=2284.19;
p_new=45.1582;
mfp_new=0.141847;
lambda_new=0.999984;
alpha_in=13070;
M_load=54.8672;
mfi_time=5786;

// Fyllning av lambdabuffer.
for (count=0 ; count <= (buf_size-1) ; count=count+1 )
        lambda_del[count]=0.999984;
count=0;
```

```
/*
 *  Wait for start
 */
RTKReceive(NULL, 0);

nextActivation = RTKGetTime();

RTKWait(sem);
while(True) {
RTKSignal(sem);
RTKDelayUntil(nextActivation);
RTKWait(sem);
nextActivation = RTKGetTime() + Ticks(time);

/* -------- code here ---------------- */
// Inläsning av insignaler.
M_load=Read(M_port);
new_value=Recieve(control_signals,recieved);
if (new_value)
{
        if (recieved[0])
                mfi_time=control_signals[0];
        if (recieved[1])
                alpha_in=control_signals[1];
}

// Konvertering av insignaler.
mfi=4.16454E-4*n_new*(((double)(mfi_time))/1000-0.6570);
alpha=((double)(alpha_in))/10000;

// Trotteldynamik
alpha_new=k5*alpha_old+k6*alpha;

// Test om den nya arbetspunkten ligger i den gamla
// arbetspunktsrektangeln.
in_old_square1=((n_help1[0] <= n_new) && (n_help1[1] > n_new) &&
        (p_help1[0] <= p_new) && (p_help1[1] > p_new));
in_old_square2=((n_help2[0] <= n_new) && (n_help2[1] > n_new) &&
        (p_help2[0] <= p_new) && (p_help2[1] > p_new));
in_old_square3=((alpha_help[0] <= alpha) && (alpha_help[1] > alpha) &&
        (p_help3[0] <= p_new) && (p_help3[1] > p_new));
in_old_square4=(((n_help4[0] <= n_new) && (n_help4[1] > n_new) &&
        (p_help4[0] <= p_new) && (p_help4[1] > p_new)) ||
        (n_new >= 3510.0) || (n_new <= 1580.0) || (p_new >= 59.0) ||
        (p_new <= 39.0));

// Mappning av moment.
if (!in_old_square1)
{
        if (lambda_new <= 1.025)
        {
                if (lambda_new > 0.975)
                        M_eff=mapping1(n_new,p_new,n1_00,p1_00,
                        torque1_00,n_help1,p_help1,M_help);
                else
                {
                        if (lambda_new > 0.925)
                                M_eff=mapping1(n_new,p_new,n0_95,p0_95,
                                torque0_95,n_help1,p_help1,M_help);
```

```
                        else
                                M_eff=mapping1(n_new,p_new,n0_90,p0_90,
                                torque0_90,n_help1,p_help1,M_help);
                }
        }
        else
        {
                if (lambda_new <= 1.075)
                        M_eff=mapping1(n_new,p_new,n1_05,p1_05,
                        torque1_05,n_help1,p_help1,M_help);
                else
                        M_eff=mapping1(n_new,p_new,n1_10,p1_10,
                        torque1_10,n_help1,p_help1,M_help);
        }
}
else
        M_eff=interpol(n_new,p_new,n_help1,p_help1,M_help);

// Mappning av mac.
if (!in_old_square2)
        mac=mapping1(n_new,p_new,n1_00,p1_00,mac_map,
                n_help2,p_help2,mac_help);
else
        mac=interpol(n_new,p_new,n_help2,p_help2,mac_help);

// Mappning av mat.
if (!in_old_square3)
        mat=mapping2(alpha_new,p_new,alpha_vec,p1_00,
                mat_map,alpha_help,p_help3,mat_help);
else
        mat=interpol(alpha_new,p_new,alpha_help,p_help3,mat_help);

// Mappning av bränslefilmskonstanter.
if (!in_old_square4).
        mapping3(n_new,p_new,n_help4,p_help4,X_help,T_help,lambda_val);
else
{
        lambda_val[0]=interpol(n_new,p_new,n_help4,p_help4,X_help);
        lambda_val[1]=interpol(n_new,p_new,n_help4,p_help4,T_help);
}
X=lambda_val[0];
tau_inv=lambda_val[1];

// Uppdatering av tillstånd.
n_old=n_new;
p_old=p_new;
mfp_old=mfp_new;
lambda_old=lambda_new;
alpha_old=alpha_new;

// Eulers metod för de tre första tillstånden.
n_new=n_old+k1*(M_eff-M_load);
p_new=p_old+k2*(mat-mac);
mfp_new=(1-time*tau_inv)*mfp_old+time*X*mfi;

// Tidsfördröjning av lambda.
mfp_dot=-tau_inv*mfp_new+X*mfi;
mf=mfi-mfp_dot;
lambda_real=mac/(14.67*mf);
```

```
        lambda_del[count]=lambda_real;
        t_del=floor(linterpol(n_new,n_table,t_table,size_t)/time);
        if (count > t_del)
                index=count-t_del;
        else
                index=(buf_size+count-t_del)%buf_size;
        count=count+1;
        if (count==buf_size)
                count=0;

        // Eulers metod för lambda.
        lambda_new=k3*lambda_old+k4*lambda_del[index];

        // Addition av brus.
        n_brus=100*(frand()-0.5);
        p_brus=2*(frand()-0.5);
        lambda_brus=0.03*(frand()-0.5);
        mat_brus=3.5*(frand()-0.5);
        n_out=n_new+n_brus;
        p_out=p_new+p_brus;
        lambda_out=lambda_new+lambda_brus;
        mat_out=mat+mat_brus;

        // Konvertering av utsignaler.
        n_volt=n_out/1000;
        lambda_volt=linterpol(lambda_out,lambda_table,
                        lambda_table_volt,size_lambda);
        p_volt=0.025789*p_out+1.303557;
        mat_volt=linterpol(mat_out,mat_table,mat_table_volt,size_mat);

        // Utställning av utsignaler.
        Write(n_port,n_volt);
        Write(p_port,p_volt);
        Write(lambda_port,lambda_volt);
        Write(mat_port,mat_volt);


        /* -------- end code here ------------ */
        }
}
```

Bilaga C: Programkod för regulator

Nedan visas programkoden för realtidsimplementeringen av regulatorn.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <math.h>

#include "nidaqcns.h"
#include "nidaq.h"

#include "main.h"
#include "dstruct.h"
#include "command.h"
#include "can.h"
#include "display.h"
#include "serial.h"

#include <RTKernel.h>
#include <timer.h>

// Mappar för mac och bränslefilmsparametrarna. Dessutom vektorerna för
// lambda och mat, som används av funktionen interpol vid konvertering av
// insignaler.
const double n1_00[14]={ 1210.5, 1427.6, 1640.4, 1857.3, 2075.6, 2291.2,
                         2504.4, 2717.8, 2934.2, 3152.1, 3366.3, 3583.2,
                         3802.0, 4016.9};
const double p1_00[8]={ 35.3901, 40.2004, 45.1575, 50.1320, 55.2191,
                        60.0870, 65.3424, 70.0004 };
const double lambda_table_volt[15]={ 2.50,2.75,2.80,2.85,2.90,2.95,
                       3.00,3.05,3.10,3.15,3.20,3.25,3.30,3.40,3.50 };
const double lambda_table[15]={ 0.835,0.910,0.925,0.942,0.960,0.980,1.000,
                          1.038,1.076,1.117,1.163,1.208,1.267,1.391,1.534 };
const double mat_table_volt[25]={ 1,1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2.0,
                    2.1,2.2,2.3,2.4,2.5,2.6,2.7,2.8,2.9,3.0,3.1,3.2,3.3,3.4 };
const double mat_table[25]={ 14.0545,17.1303,20.6212,24.5355,28.8765,33.6441,
                38.8440,44.4912,50.6113,57.2435,64.4437,72.2503,80.6778,
                89.7364,99.4325,109.7688,120.7440,132.3530,144.5885,
                157.4397,170.8934,184.9339,199.5436,214.7034,230.3929 };
double mac_map[8][14]={
         { 27.1401, 33.4780, 37.4038, 42.8025, 49.1926, 54.9813, 57.0070,
          60.1525, 66.0269, 73.0761, 80.4801, 91.0819,102.6179,111.9639 },
         { 31.3762, 38.4915, 43.5331, 49.4941, 57.5409, 64.4143, 65.8180,
          69.6307, 76.1935, 84.5371, 93.1909,104.7772,117.2829,125.8456 },
         { 36.5816, 43.7063, 49.0834, 54.5279, 64.0735, 71.8202, 74.9825,
          79.8041, 87.1288, 96.7028,106.7984,119.3549,134.2199,144.8055 },
         { 41.0598, 49.7507, 55.6499, 60.7004, 71.6939, 80.0748, 85.2749,
          89.8921, 97.3243,108.4406,119.0551,133.2336,148.9151,159.6585 },
         { 45.5687, 54.1242, 61.2068, 68.8084, 80.1682, 88.8392, 94.0991,
          100.4105,109.3648,119.8345,132.7870,147.0740,165.2814,178.3642 },
         { 49.5799, 59.0484, 67.3137, 74.3973, 87.6511, 97.1026,104.1713,
          109.4608,119.8695,132.2914,144.8106,159.9984,179.9865,191.3365 },
         { 53.9649, 64.9329, 73.4057, 82.6206, 95.3679,106.2232,113.0504,
          120.5076,130.7490,143.7264,158.6442,174.5264,195.3475,211.7156 },
         { 58.6835, 70.5801, 80.0620, 89.4361,103.2296,114.4673,122.4754,
          130.4844,141.0092,155.7108,170.8532,188.4639,209.2648,216.2723 }
         };
```

```
const double n_vector[4] = { 1580.0 , 2013.0 , 2717.0 , 3510.0 };
const double p_vector[3] = { 39.0 , 51.0 , 59.0 };
double X_map[3][4] = { { 0.3, 0.3, 0.3, 0.05 },
                       { 0.2, 0.1, 0.05, 0.05 },
                       { 0.1, 0.05, 0.05, 0.05 } };
double T_map[3][4] = { { 2.0, 4.0, 4.0, 10.0 },
                       { 4.0, 8.0, 10.0, 10.0 },
                       { 8.0, 10.0, 10.0, 10.0 } };

// Uppdatering av macmapp.
void update_macmap(double map_val, double real_val, double* k_vec,
                   int* ind_vec)
{
        double c1, c2, c3, c4;
        double k1, k2;
        double delta=0.005;
        int x_ind, y_ind;
        double sum,e;

        x_ind=ind_vec[0];
        y_ind=ind_vec[1];
        k1=k_vec[0];
        k2=k_vec[1];
        e=real_val-map_val;

        if ((x_ind >= 0) && (y_ind >= 0))
        {
                c1=1-k1-k2+k1*k2;
                c2=k2-k1*k2;
                c3=k1-k1*k2;
                c4=k1*k2;
                sum=(c1*c1)+(c2*c2)+(c3*c3)+(c4*c4);
                mac_map[x_ind][y_ind]=mac_map[x_ind][y_ind]+delta*c1*e/sum;
                mac_map[x_ind+1][y_ind]=mac_map[x_ind+1][y_ind]+
                delta*c2*e/sum;
                mac_map[x_ind][y_ind+1]=mac_map[x_ind][y_ind+1]+
                delta*c3*e/sum;
                mac_map[x_ind+1][y_ind+1]=mac_map[x_ind+1][y_ind+1]+
                delta*c4*e/sum;
        }
}

// Linjärsökning av vektor.
int search_vector(double val, const double* vector, int size)
{
        int found=0, count=0, index=-1;
        while ((!found) && (count < size))
        {
                if ((vector[count] <= val) && (vector[count+1] > val))
                        found=1;
                else
                        count=count+1;
        }

        if (found)
                index=count;

        return index;
}
```

```
// linjär interpolation.
double interpol(double xval, double yval, double* x, double* y, double z[][2],
                double* k_vec)
{
        double k1, k2, z1, z2, zval;

        k1=(xval-x[0])/(x[1]-x[0]);
        k2=(yval-y[0])/(y[1]-y[0]);

        z1=z[0][0]+k1*(z[0][1]-z[0][0]);
        z2=z[1][0]+k1*(z[1][1]-z[1][0]);

        zval=z1+k2*(z2-z1);

        k_vec[0]=k1;
        k_vec[1]=k2;

        return zval;
}

// Mappning av mac. Funktionen returnerar också arbetspunktsrektangeln
// som används vid mappningen, samt index och koefficienterna k1, k2 från
// den linjära interpolationen. Dessa används vid uppdateringen av mappen.
double mapping1(double n_val, double p_val, const double* n_vector,
                const double* p_vector, const double z_vector[][14],
                double* n_help, double* p_help, double z_help[][2],
                int* ind_vec, double* k_vec)
{
        int i, j, index1, index2, out_of_range=0;
        int length_n=14, length_p=8;
        double zval=0;

        index1=search_vector(n_val,n_vector,length_n);
        index2=search_vector(p_val,p_vector,length_p);

        ind_vec[0]=index2;
        ind_vec[1]=index1;

        if ((index1 >=0) && (index2 >=0))
        {
                for (i=0; i<=1; i=i+1)
                {
                        n_help[i]=n_vector[index1+i];
                        p_help[i]=p_vector[index2+i];
                        for (j=0; j<=1; j=j+1)
                        {
                                z_help[j][i]=z_vector[index2+j][index1+i];
                                if (z_vector[index2+j][index1+i]==0)
                                {
                                        out_of_range=1;
                                        break;
                                }
                        }

                }

                if (!out_of_range)
                        zval=interpol(n_val,p_val,n_help,p_help,z_help,k_vec);
```

```
        }
        return zval;

}


// Mappning av bränslefilmsparametrar. Funktionen returnerar också arbets-
// punktsrektangeln som används vid mappningen.
void mapping3(double n_val, double p_val, double* n_help, double* p_help,
              double X_help[][2], double T_help[][2], double* lambda_val)
{
        int i, j, index1, index2, length_n=4, length_p=3;

        if (n_val <= 1580.0)
        {
                if (p_val <= 45.0)
                {
                        lambda_val[0]=0.3;
                        lambda_val[1]=2;
                }
                else
                {
                        if (p_val <= 55.3)
                        {
                                lambda_val[0]=0.2;
                                lambda_val[1]=4;
                        }
                        else
                        {
                                lambda_val[0]=0.1;
                                lambda_val[1]=8;
                        }
                }
        }
        else
        {
                if ((n_val >= 3510.0) || (p_val >= 59.0))
                {
                        lambda_val[0]=0.05;
                        lambda_val[1]=10;
                }
                else
                {
                        if (p_val <= 39.0)
                        {
                                lambda_val[0]=0.3;
                                lambda_val[1]=4;
                        }
                        else
                        {
                                index1=search_vector(n_val,n_vector,length_n);
                                index2=search_vector(p_val,p_vector,length_p);

                                ind_vec2[0]=index2;
                                ind_vec2[1]=index1;

                                for ( i=0 ; i <= 1 ; i=i+1 )
```

```
                                {
                                        n_help[i]=n_vector[index1+i];
                                        p_help[i]=p_vector[index2+i];
                                        for ( j=0 ; j <= 1 ; j=j+1 )
                                        {
                                                X_help[j][i]=X_map[index2+j]
                                                [index1+i];
                                                T_help[j][i]=T_map[index2+j]
                                                [index1+i];
                                        }
                                }
                                lambda_val[0]=interpol(n_val,p_val,n_help,
                                                        p_help,X_help,k_vec2);
                                lambda_val[1]=interpol(n_val,p_val,n_help,
                                                        p_help,T_help,k_vec2);
                        }
                }
        }
}

// Linjär endimensionell interpolation, används vid
// konvertering av insignaler.
double linterpol(double val, const double* x_vector, const double* y_vector,
                                        int size)
{
        int found=0, count=0;
        double k, zval;

        if (val <= x_vector[0])
                zval = y_vector[0];
        else
        {
                while ((!found) && (count < (size-1)))
                {
                        if ((x_vector[count] <= val) &&
                           (x_vector[count+1] > val))
                                found=1;
                        else
                                count=count+1;

                }
                if (!found)
                        zval = y_vector[size-1];
                else
                {
                        k=(y_vector[count+1]-y_vector[count])/
                          (x_vector[count+1]-x_vector[count]);
                        zval=y_vector[count]+k*(val-x_vector[count]);
                }
        }
        return zval;
}

// Filter för filtrering av mätsignaler.
double filter(double* v, double x)
{
        double y;
        const double b[4]={0.0138,0.0415,0.0415,0.0138};
        const double a[3]={-1.8816,1.3096,-0.3174};
```

```
        y=b[0]*x+v[0];
        v[0]=b[1]*x+v[1]-a[0]*y;
        v[1]=b[2]*x+v[2]-a[1]*y;
        v[2]=b[3]*x-a[2]*y;

        return y;
}

// Sändning av styrsignaler via CAN-buss.
void Send(int* val)
{
        BYTE data[2];
        int mfi_id=31, alpha_id=32;

        data[0] = ((255)&(val[0]>>8));
        data[1] = ((255)&(val[0]));
        SendCan((DWORD)mfi_id,(BYTE)2,(BYTE*)data);

        data[0] = ((255)&(val[1]>>8));
        data[1] = ((255)&(val[1]));
        SendCan((DWORD)alpha_id,(BYTE)2,(BYTE*)data);
}

DECLARE_PROCESS(RegMat)

void RegMat()
{
        double n_filt, p_filt, mat_filt, lambda_filt;
        double alpha_filt, mac, mac_old, mfi;
        double n_vect[3]={0,0,0}, p_vect[3]={0,0,0};
        double lambda_vect[3]={0,0,0}, mat_vect[3]={0,0,0};
        double alpha_vect[3]={0,0,0};
        double n, alpha_old, p, p_old, mat, lambda;
        double new_error;
        double k1, c1, p_hat;
        double n_volt, p_volt, lambda_volt, mat_volt, alpha_volt;
        double X, tau_inv, mfp_old, mfp_new, mfi_demanded;
        double diff, alpha_limit=0.04, diff_limit=2.5;
        double n_help1[2]={0,0}, p_help1[2]={0,0};
        double mac_help[2][2]={{0,0},{0,0}};
        double n_help2[2]={0,0}, p_help2[2]={0,0};
        double n_help3[2]={0,0}, p_help3[2]={0,0}, lambda_val[2]={0,0};
        double X_help[2][2]={{0,0},{0,0}}, T_help[2][2]={{0,0},{0,0}};
        double I, K, Ti;
        int control_signals[2]={0,0};
        int in_old_square1, in_old_square2, in_old_square3;
        int trans_stop, step, up, no_down, count, time_count, time_limit;
        int end_start1, start_up1, start_limit1;
        int end_start2, start_up2, start_limit2, start_limit3;
        double fuel_help, fuel_coeff;
        int no_val=0, full=0;
        int ind_vec1[2]={0,0}, ind_vec2[2]={0,0};
        double k_vec1[2]={0,0}, k_vec2[2]={0,0};
        int adapt=0;
        const int size_lambda=15, size_mat=25, bias_size=200, down_limit=6;
        Port alpha_port, n_port, p_port, mat_port, lambda_port;
        Semaphore sem;
        double time;
```

```
Time nextActivation;

sem = RegisterDouble(time);
RegisterPort(n_port);
RegisterPort(p_port);
RegisterPort(mat_port);
RegisterPort(lambda_port);
RegisterPort(alpha_port);
RegisterInt(adapt);

/*
 *  Default values
 */
time = 0.004;
n_port = 2;
p_port = 3;
lambda_port = 4;
mat_port = 5;
alpha_port = 19;

// Konstant i ekvation för uppdatering av bränslefilmstillstånd.
c1=4.360;
k1=c1*time;

// Initialvärden.
mfi=5.7856;
mfp_old=0.141847;
mfp_new=0.141847;
p_old=45.1582;
alpha_old=1.307;
fuel_coeff=14.67;
step=0;
trans_stop=1;
time_limit=floor(1.6/time);
start_limit1=100;
start_limit2=300;
start_limit3=500;
end_start1=0;
end_start2=0;
start_up1=0;
start_up2=0;
fuel_help=0;
no_down=0;
time_count=0;
count=0;
I=4.8785;
K=3.6;
Ti=0.35;

/*
 *  Wait for start
 */
RTKReceive(NULL, 0);

nextActivation = RTKGetTime();

RTKWait(sem);
while(True) {
RTKSignal(sem);
```

```
RTKDelayUntil(nextActivation);
RTKWait(sem);
nextActivation = RTKGetTime() + Ticks(time);

/* -------- code here ---------------- */

// Läsning av insignaler.
n_volt=Read(n_port);
p_volt=Read(p_port);
lambda_volt=Read(lambda_port);
mat_volt=Read(mat_port);
alpha_volt=Read(alpha_port);

// Filtrering.
n_filt=filter(n_vect,n_volt);
p_filt=filter(p_vect,p_volt);
mat_filt=filter(mat_vect,mat_volt);
lambda_filt=filter(lambda_vect,lambda_volt);
alpha_filt=filter(alpha_vect,alpha_volt);

// Insvängningstid för filtren.
if (!end_start1)
{
        start_up1=start_up1+1;
        if (start_up1 == start_limit1)
                end_start1=1;
}

// Konvertering av insignaler.
if (end_start1)
{
        n=1000*n_filt;
        p=12.5*(3.102090*p_filt-4.043751);
        if (cont)
                lambda=linterpol(lambda_filt,lambda_table_volt,
                                 lambda_table,size_lambda);
        else
                lambda=lambda_filt;
        mat=linterpol(mat_filt,mat_table_volt,mat_table,size_mat);
}
else
{
        n=1000*n_volt;
        p=12.5*(3.102090*p_volt-4.043751);
        if (cont)
                lambda=linterpol(lambda_volt,lambda_table_volt,
                                 lambda_table, size_lambda);
        else
                lambda=lambda_filt;
        mat=mat_volt;
        alpha_filt=alpha_volt;
}

// Skattning av stökiometriskt luft/bränsleförhållande.
if (!end_start2)
        if (end_start1)
        {
                start_up2=start_up2+1;
                if (start_up2 >= start_limit2)
```

```
                {
                        fuel_help=fuel_help+mat/mfi;
                        if (start_up2 == start_limit3)
                        {
                                fuel_coeff=fuel_help/
                                (start_limit3-start_limit2+1);
                                end_start2=1;
                        }
                }
        }


// Mappning av mac_old.
in_old_square1=((n_help1[0] <= n) && (n_help1[1] > n) &&
        (p_help1[0] <= p) && (p_help1[1] > p));
if (!in_old_square1)
        mac_old=mapping1(n,p,n1_00,p1_00,mac_map,n_help1,
                         p_help1,mac_help,ind_vec1,k_vec1);
else
        mac_old=interpol(n,p,n_help1,p_help1,mac_help,k_vec1);

// Uppdatering av macmapp.
if ((adapt) && (!step))
        update_macmap(mac_old,mat,k_vec1,ind_vec1);

p_hat=p+(k1/2)*(mat-mac_old);

// Mappning av mac.
in_old_square2=((n_help2[0] <= n) && (n_help2[1] > n) &&
        (p_help2[0] <= p) && (p_help2[1] > p));
if (!in_old_square2)
        mac=mapping1(n,p_hat,n1_00,p1_00,mac_map,n_help1,
                     p_help1,mac_help,ind_vec1,k_vec1);
else
        mac=interpol(n,p_hat,n_help1,p_help1,mac_help,k_vec1);


// Kontroll om ett steg har påbörjats.
diff=(alpha_filt-alpha_old)/time;
if ((alpha_volt-alpha_old >= alpha_limit) || (diff >= diff_limit))
{
        step=1;
        trans_stop=0;
        up=1;
}
else
{
        if ((alpha_volt-alpha_old <= -alpha_limit) ||
            (diff <= -diff_limit))
        {
                step=1;
                trans_stop=0;
                up=0;
        }
}

// Kontroll om transienten är slut.
if (step)
{
```

```
        if (((up) && ((mac - mat) > 0)) ||
                  ((!up) && ((mat - mac) > 0)))
                mac=mat;

        if (time_count != time_limit)
                time_count=time_count+1;

        if (time_count == time_limit)
        {
                if (up)
                {
                        if ((p - p_old) < 0.1)
                        {
                                if (no_down == down_limit)
                                {
                                        step=0;
                                        trans_stop=1;
                                        no_down=0;
                                        time_count=0;
                                }
                                else
                                        no_down=no_down+1;
                        }
                }
                else
                {
                        if ((p_old - p) < 0.1)
                        {
                                if (no_down == down_limit)
                                {
                                        step=0;
                                        trans_stop=1;
                                        no_down=0;
                                        time_count=0;
                                }
                                else
                                        no_down=no_down+1;
                        }
                }
        }
}

// Regulator
if ((!out_of_range) && (step))
{
        mfi_demanded=mac/fuel_coeff;
        I=mfi;
}
else
{
        //PI-regulator
        if (n < 1593)
        {
                if (p < 51.3)
                {
                        K=2;
                        Ti=0.45;
                }
                else
```

```
        {
                K=2.7;
                Ti=0.38;
        }
}
else
        if (n < 2015)
        {
                if (p < 51.3)
                {
                        K=2.5;
                        Ti=0.42;
                }
                else
                {
                        K=3.6;
                        Ti=0.34;
                }
        }
        else
                if (n < 2564)
                {
                        if (p < 51.3)
                        {
                                K=3.6;
                                Ti=0.35;
                        }
                        else
                        {
                                K=5.0;
                                Ti=0.31;
                        }
                }
                else
                        if (n < 3122)
                        {
                                if (p < 51.3)
                                {
                                        K=5.4;
                                        Ti=0.30;
                                }
                                else
                                {
                                        K=6.8;
                                        Ti=0.28;
                                }
                        }
                        else
                                if (p < 51.3)
                                {
                                        K=7.7;
                                        Ti=0.23;
                                }
                                else
                                {
                                        K=9.9;
                                        Ti=0.21;
                                }
```

```
                I=I+K/Ti*time*(lambda-1);
                mfi=K*(lambda-1)+I;
        }

        // Uppdatering av tillstånd.
        mfp_old=mfp_new;
        alpha_old=alpha_filt;
        p_old=p;

        // Mappning av bränslefilmskonstanter.
        in_old_square3=(((n_help3[0] <= n) && (n_help3[1] > n) &&
                (p_help3[0] <= p) && (p_help3[1] > p)) || (n >= 3510.0) ||
                (n <= 1580.0) || (p >= 59.0) || (p <= 39.0));
        if (!in_old_square3)
                mapping3(n,p,n_help3,p_help3,X_help,T_help,lambda_val);
        else
        {
                lambda_val[0]=interpol(n,p,n_help3,p_help3,X_help,k_vec2);
                lambda_val[1]=interpol(n,p,n_help3,p_help3,T_help,k_vec2);
        }
        X=lambda_val[0];
        tau_inv=lambda_val[1];

        // Olinjär bränslekompensator.
        if (step)
                mfi=(mfi_demanded-tau_inv*mfp_new)/(1-X);
        mfp_new=(1-time*tau_inv)*mfp_old+time*X*mfi;

        // Konvertering av utsignaler.
        control_signals[0]=1000*(2401.2*mfi/n+0.6570);
        control_signals[1]=10000*alpha_volt;

        // Sändning av styrsignaler.
        Send(control_signals);

        /* -------- end code here ------------ */
        }
}
```