Realtidsimplementering av modellbaserad diagnos

Examensarbete utfört i Fordonssystem
vid Tekniska Högskolan i Linköping
av

Lars Andersson

Reg nr: LiTH-ISY-EX-1878

Realtidsimplementering av modellbaserad diagnos

Examensarbete utfört i Fordonssystem
vid Tekniska Högskolan i Linköping
av

Lars Andersson

Reg nr: LiTH-ISY-EX-1878

Handledare: **Mattias Nyberg**

Examinator: **Lars Nielsen**

Linköping, 8 april 1998.

Sammanfattning

Modellbaserad diagnos är en metod som utnyttjar processmodeller för att utföra diagnos. Residualer genereras genom att kombinera modellvärden och sensorvärden för en process. Residualer ska reagera på skillnader mellan modell och verklig process. Residualerna jämförs med tröskelvärden vilket ger felbeslut.

Under arbetet utvecklades ett diagnossystem för modellbaserad diagnos on-line. Diagnossystemet består av ett realtidssystem programmerat i C och ett i Matlab utvecklat GUI, implementerade på två olika datorer. Dom kommunicerar med varandra via en CAN-buss.

Realtidssystemet utför diagnos på en besinmotors luftintagssystem. Det består av två processer. En process avläser sensorer, genererar residualer, beräknar felbeslut och sparar dessa värden. Den andra processen hanterar all kommunikation med GUI exempelvis att meddela fel. I GUI indikeras om fel uppstår. Där kan också värden plottas.

Diagnossystemet validerades genom motorkörningar. Resultatet visar att alla simulerade fel detekteras och isoleras korrekt om motorns arbetsområde hålls väl inom modellernas gränser. Nära modellernas gränser kan felaktig detektering ske.

Nyckelord: residual, felbeslut, tröskelvärden, diagnossystem, realtidssystem, GUI

Tackord

Jag vill tacka alla på Fordonssystem för en trevlig och intressant tid. Min handledare Mattias Nyberg vill jag tacka speciellt. Han har gett värdefull hjälp och goda förslag. Jag vill också tacka Andrej Perkovic för bra ideer och assistans vid motorkörningar. Erik Frisk tackas för sin hjälp med LaTeX. Ett stort tack riktas också till Simon Edlund för all programmeringshjälp vid inledningen av arbetet.

Linköping, 8 april 1998

Lars Andersson

Innehåll

1 Inledning **1**
1.1 Syfte 1
1.2 Metod 2
1.3 Rapportens uppläggning 2

2 Modellbaserad diagnos **3**
2.1 Strukturerade residualer 3
2.2 Diagnossystem 4
2.3 Residualgenerering 5

3 Diagnos av luftintagssystem **6**
3.1 Modell av luftintagssystemet 6
3.2 Residualgenerering 8
3.3 Residualevaluering 10
3.4 Tidsdiskretisering av modellerna 10

4 Uppbyggnad och funktion av diagnossystemet **12**
4.1 Översikt av diagnossystemet 12
4.2 Valet av systemstruktur 13
4.3 Syftet med diagnossystemet 14
4.4 Realtidssystemets funktion 15
4.4.1 Översikt 15
4.4.2 Diagnosprocess 15
4.4.3 Managerprocess 18
4.5 GUIs funktion 21
4.5.1 Översikt 21
4.5.2 Delsystem 21
4.5.3 Diagnosfönster 22
4.5.4 Initieringsfönster 24
4.5.5 Parameterfönster 24
4.5.6 Informationsfönster 25
4.5.7 Plotfönster 26
4.5.8 Väljfönster 27

5 Detaljerad beskrivning av realtidssystem **30**
5.1 Realtidsskal 30
5.2 Flödet i realtidssystemet 31
5.3 Buffertar 32

5.3.1 Buffertar i realtidssystemet 35
5.4 Diagnosprocess 35
5.4.1 Beräkna sensorvärden 36
5.4.2 Residualgenerering 37
5.4.3 Residualevaluering 38
5.5 Managerprocess 39
5.5.1 Starta diagnos 39
5.5.2 Stoppa diagnos 43
5.5.3 Ändra tröskelvärden 43
5.5.4 Ändra status på fel 43
5.5.5 Kontrollera om något fel har inträffat 43
5.5.6 Återställa fel 44
5.5.7 Skicka plotvärden 44
5.5.8 Meddela Diagnosprocessen 45
5.6 Utökning och förändring av diagnossystemet 45
5.7 Initiering av diagnossystemet 46
5.7.1 Skapa processer och initiera realtidssystemet . . . 46
5.7.2 Skicka information till GUI 46
5.8 CAN-kommunikation 46
5.9 Kommentarer om realtidssystemet 47

6 Detaljerad beskrivning av GUI 49
6.1 Grafiska objekt i Matlab 49
6.2 Datatyper i Matlab 50
6.3 Dynamisk dataöverföring, DDE 51
6.4 Initiering av GUI 51
6.5 Beskrivning av vissa objekt och händelser i GUI 52
6.5.1 Parameterfönster 52
6.5.2 Initieringsfönster 53
6.5.3 Diagnosfönster 53
6.5.4 Informationsfönster 53
6.5.5 Plotfönster 54
6.5.6 Väljfönster 54

7 Validering av diagnossystemet 55
7.1 Felsimulering 55
7.2 Resultat 56
7.3 Prestanda och tillförlitlighet hos systemet 57

8 Slutsatser **58**
8.1 Sammanfattning 58
8.2 Resultat 59
8.3 Utvidgningar 59

Bilaga A: Plottar av motorkörningar **63**

Bilaga B: CAN_DRV **76**

Bilaga C: DDE i Matlab **78**

Kapitel 1

Inledning

Diagnos kan förenklat beskrivas som att, i en process avgöra om ett fel har uppstått och i så fall identifiera felet. Diagnos kan ske *on-line* eller *off-line*, dvs under drift eller efter att processen har avslutats.

Diagnos on-line av tekniska system blir allt viktigare. Det beror bland annat på att systemen blir allt komplexare och fel som inte upptäcks i tid kan bli mycket dyrbara. Om ett fel upptäcks tidigt så kan åtgärder vidtas så att felet inte förvärras. Säkerheten kan också förbättras med diagnos. Dessutom blir miljökraven allt strängare, vilket medför att fel som orsakar föroreningar måste upptäckas och åtgärdas.

En vanlig diagnosmetod är att kontrollera att olika mätvärden från sensorer håller sig inom vissa gränser. De allt högre krav som ställs på diagnos medför att denna metod är otillräcklig i vissa fall. Kvaliten på diagnosen kan höjas genom att använda *modellbaserad* diagnos. Då används matematiska modeller av den process som ska diagnostiseras. Då kan diagnosen utföras på ett större arbetsområde och mindre fel kan detekteras och isoleras.

1.1 Syfte

Idag finns det få system som använder modellbaserad diagnos on-line. Syftet med detta arbete är därför att se hur ett sådant diagnossystem kan utvecklas och vilken prestanda det får. Diagnossystemet ska främst användas för att analysera tillförlitligheten hos den modellbaserade diagnosmetoden på olika processer. Det ska också vara ett hjälpmedel för att demonstrera hur modellbaserad diagnos i *realtid* fungerar. Då bör det vara möjligt att plotta värden som beräknas under diagnosen. Det

ska vara enkelt ändra parametrar som är av avgörande betydelse för diagnosen. Det ska också vara enkelt att implementera diagnos av nya processer. Eftersom diagnossystemet bara ska användas som analysverktyg, behöver inga speciella åtgärder vidtas om ett fel uppstår.

1.2 Metod

Diagnosen sker on-line vilket medför att alla beräkningar måste utföras i realtid. Diagnossystemet ska implementeras i ämnesområdet Fordonssystems laborationsutrustning vid Linköpings universitet. Där används ett realtidssystem [9, 11] som är skrivet i C. Realtidssystemet har ett skal som förenklar utvecklandet av program för realtidsberäkningar. Den del av diagnossystemet som utför alla beräkningar implementeras i realtidssystemet och programmeras därför i C. Realtidssystemet utför diagnos på processer och sparar värden som kan plottas.

För att kommunicera med omvärlden utvecklas ett grafiskt användargränssnitt, GUI. Där indikeras om ett fel uppstår och där kan även vissa parametrar som används vid diagnosen ändras. Dessutom kan värden som beräknas under diagnosen plottas. GUI utvecklas i Matlab [6].

Realtidsystemet och GUI implementeras på två olika datorer och kommunicerar med varandra via en CAN[1]-buss.

I diagnossystemet införs diagnos av luftintagssystemet hos en SAAB 2.3 liters motor. Modellerna som används tas från en licentiatavhandling [8] och ett tidigare examensarbete [7].

1.3 Rapportens uppläggning

I kapitel 2 ges en allmän introduktion till området modellbaserad diagnos. I kapitel 3 beskrivs luftintagssystemet och de matematiska modeller som används vid diagnosen av detta. Där visas också hur modellerna tidsdiskretiseras. Kapitel 4 beskriver hur det utvecklade diagnossystemet, både realtidssystemet och GUI, är uppbyggt och vilka funktioner det har. En detaljerad beskrivning av hur diagnossystemet fungerar ges i kapitel 5 för realtidssystemet och i kapitel 6 för GUI. I kapitel 7 valideras diagnossystemet. Kapitel 8 består av slutsatser och förslag till utvidgningar av arbetet. Bilaga A innehåller plottar från motorkörningar, bilaga B beskriver ett gränssnitt för CAN-bussen och bilaga C förklarar DDE-funktioner i Matlab.

[1] CAN = Controller Area Network

Kapitel 2

Modellbaserad diagnos

Det här kapitlet behandlar modellbaserad diagnos på ett allmänt sätt för att lägga en grund till kapitel 3 där diagnos av det implementerade luftintagssystemet beskrivs. Kapitlet bygger helt på en licentiatavhandling [8] om modellbaserad diagnos. För en ingående beskrivning av modellbaserad diagnos hänvisas läsare till denna licentiatavhandling.

Modellbaserad diagnos baseras på *analytisk redundans*. En process innehåller analytisk redundans om en insignal eller utsignal kan beräknas enbart genom att använda andra insignaler eller utsignaler. I det enklaste fallet används analytisk redundans genom att jämföra utsignaler från den verkliga processen med utsignaler från en processmodell, vilken matas med samma insignaler som den verkliga processen. Till diagnos hör både feldetektering och isolering av fel. Med isolering menas att avgöra var felet har inträffat.

2.1 Strukturerade residualer

En *residual* är en signal som reagerar på skillnader mellan en modell och en verklig process. Vid diagnos av en process används ett antal residualer och dom görs känsliga för olika fel på ett sådant sätt att isolering mellan felen uppnås. Detta kallas för *strukturerade residualer*. Om inget fel har inträffat, så bör residualerna vara nära noll. Om ett fel har inträffat så bör residualerna som felet beror på vara väsentligt skilda från noll. En *residualstruktur* tas fram för fel som diagnostiseras i en process och de residualer som används vid diagnosen. Tabell 2.1 visar hur en residualstruktur kan se ut. En viktig egenskap hos residualstrukturen är att den bör vara *starkt isolerande*. Det betyder att två kolumner inte

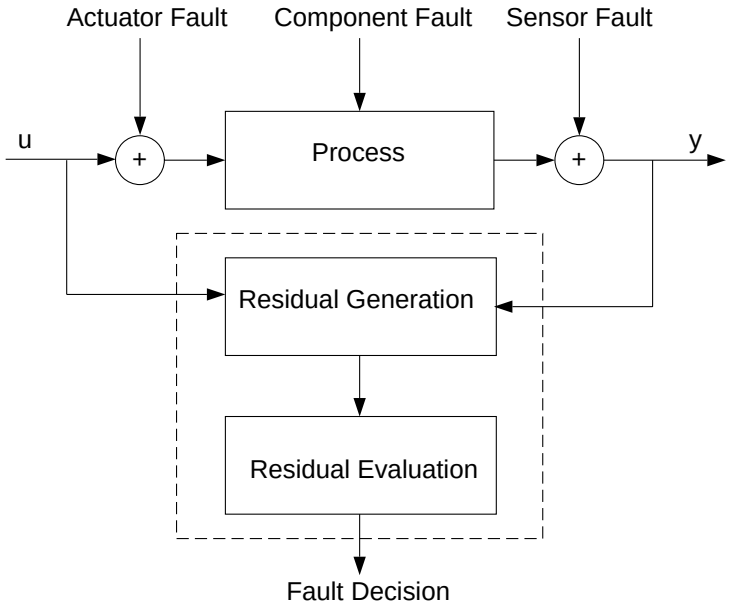
	f_1	f_2	f_3
r_1	1	1	0
r_2	0	1	1
r_3	1	0	1

Tabell 2.1: Ett exempel på en residualstruktur. Varje kolumn representerar en felaktig signal.

kan bli identiska om en 1:a i en kolumn ändras till en 0:a. Då undviks att små fel blir felaktigt isolerade när någon residual inte når upp till en tillräckligt hög nivå.

2.2 Diagnossystem

Ett diagnossystems uppbyggnad kan ses i figur 2.1. Processen kan påverkas



Figur 2.1: En process med dess fel och diagnossystemet.

av tre sorters fel: fel i styrdon, komponentfel och sensorfel. Insignaler till diagnossystemet är processens in- och utsignaler. Diagnosen utförs i två

steg som kallas *residualgenerering* och *residualevaluering*.

Residualgenerering är att beräkna residualernas värden. Nästa steg är residualevaluering. Syftet med det är att avgöra om residualerna är noll eller inte. Ett vanligt sätt att göra detta är att filtrera residualerna och testa dom mot tröskelvärden. Ett *felbeslut* genereras för varje fel genom att kontrollera de residualer felet beror på mot tröskelvärden.

2.3 Residualgenerering

Det finns flera olika sätt att designa residualgeneratorer på. Den mest intuitiva residualgenerator som kan användas är när det existerar statiska relationer mellan insignaler och utsignaler. Detta kallas för *statisk redundans* och kan visas på följande sätt:

$$r_1(t) = y_1(t) - \hat{y}_1(t) = y_1(t) - h(u(t), y_2(t))$$

där $r_1(t)$ är residualen, $u(t)$ är insignalen, $y_1(t)$ och $y_2(t)$ är utsignaler, och h en funktion som definierar den statiska relationen
$y_1(t) = h(u(t), y_2(t))$.

När systemet innehåller dynamik, kan *temporal redundans* användas. Det betyder att residualen också är en funktion av tidigare insignaler och utsignaler, eller innehåller tillstånd. Ett exempel på det är om en observatör skattar utsignalen $y_2(t)$. Då kan residualen formuleras som

$$\begin{aligned} x(t+1) &= g(x(t), u(t)) + K(y - \hat{y}) \\ r_2(t) &= y_2(t) - \hat{y}_2(t) \end{aligned}$$

En observatör som används för diagnos kallas *diagnostisk observatör* och behöver inte nödvändigtvis skatta något tillstånd hos den verkliga processen.

Kapitel 3

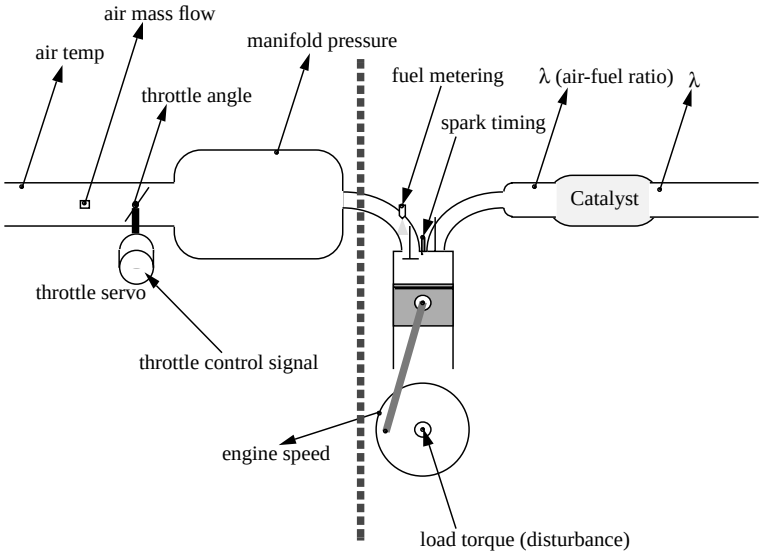
Diagnos av luftintagssystem

Det här kapitlet beskriver luftintagssystemet till en 4-cylindrig SAAB 2.3 liters motor och de residualer som används för att diagnostisera det. Kapitlet bygger till stor del på ett tidigare arbete [8]. Modellen och residualerna som används togs fram i det arbetet. En tidsdiskretisering av residualerna visas också i slutet av kapitlet. Residualerna implementerades i det diagnossystem som utvecklades.

3.1 Modell av luftintagssystemet

I luftintagssystemet behövs ingen extremt snabb feldetektering. Därför används en medelvärdesmodell, vilket betyder att inga variationer inom en cykel tas med. En skiss av motorn kan ses i figur 3.1. Det som kallas för luftintagssystemet är allting till vänster om den streckade linjen. Även motorvarvtalet måste tas med eftersom det påverkar hur mycket luft som sugs in i motorn.

Modellen av luftintagssystemet som beskrivs är kontinuerlig och består av luftdynamiken. Till systemet hör egentligen också en trottelmodell. Diagnos av trotteln har dock inte implementerats och den delen tas därför inte med här. Luftdynamiken härleds ur den ideala gaslagen och har ett tillstånd, som är trycket i insugsröret. Insignaler till systemet är motorvarvtalet n och trottelvinkeln α_s. Utsignaler är trottelvinkelsensorn α_s, luftmassflödessensorn $\dot{m}_{air,s}$ och sensorn för trycket i insugsröret



Figur 3.1: Principskiss av motor.

$p_{man,s}$. Ekvationerna som beskriver den felfria modellen kan skrivas som:

$$p_{man} = \frac{RT_{man}}{V_{man}}(\dot{m}_{air} - \dot{m}_{ac}) \quad (3.1)$$

$$\dot{m}_{air} = f(p_{man}, \alpha) \quad (3.2)$$

$$\dot{m}_{ac} = g(p_{man}, n) \quad (3.3)$$

Variablerna och dess enheter förklaras i tabell 3.1. Modellen består av en fysikalisk del, ekvation (3.1), och en blackbox del, funktionerna (3.2) och (3.3). Funktionerna f och g har tagits fram som mappar i ett tidigare examensarbete [7]. Mappar kan ses som statiska funktionssamband, där funktionsvärdet bestäms av en eller flera insignaler. Dom konstrueras genom att ta fram mätdata vid stationära förlopp vid olika arbetspunkter. Mapparna har ett giltighetsområde som är:

$$1211 \leq n \geq 4016 \text{ rpm}$$
$$35.4 \leq p_{man} \geq 70.0 \text{ kPa}$$

p_{man} (kPa)	tryck i insugsrör
R (J/(kg K))	gaskonstant
T (K)	lufttemperatur i insugsrör vilken antas vara likadan som omgivningstemperaturen
V (m^3)	volymen i insugsrör
$\dot{m}_{air}$ (kg/s)	luftmassflöde in i insugsröret och som är lika som flödet förbi luftmassflödessensorn
$\dot{m}_{ac}$ (kg/s)	luftmassflöde ut från insugsröret
f	statisk funktion som beskriver flödet förbi trotteln
g	statisk funktion som beskriver flödet in i cylindrarna
α (grader)	trottelvinkel
n (rpm)	motorvarvtal

Tabell 3.1: Symboler och enheter.

3.2 Residualgenerering

Insignaler till diagnossystemet är $\dot{m}_{air,s}$, α_s, $p_{man,s}$, och n. Komponenter som diagnostiseras är trottelvinkelsensorn, luftmassflödessensorn och sensorn som mäter trycket i insugsröret. Det antas att bara ett fel kan ske samtidigt. För att diagnostisera felen används fyra residualer. Första residualen är en residual för statisk redundans och har detta utseende:

$$r_1 = \dot{m}_{air,s} - f(p_{man,s}, \alpha_s)$$

Residualen bygger på en statisk relation i modellen och kontrollerar överensstämmelsen med ekvation 3.2.

Dom andra residualerna har tagits fram med temporal redundans. Den andra residualen kontrollerar överensstämmelsen med ekvationerna 3.1 och 3.3:

$$r_2 = \dot{m}_{air,s} - g(p_{man,s}, n) - \frac{V_{man}}{RT_{man}}\hat{\dot{p}}_{man}$$

Det antas att derivatan av $p_{man,s}$ kan skattas med tillräcklig noggrannhet.

De två sista är två observatörsresidualer. En observatörsresidual ska göra en residual känslig för bara ett sensorfel, genom att bara mäta

en utsignal. Det kan ses i r_3 som bara mäter $p_{man,s}$ och r_4 som bara mäter $\dot{m}_{air,s}$, (α_s räknas inte som en utsignal i detta fall). Utseendet på residualerna är:

$$\begin{aligned}\dot{\hat{p}}_{man} &= \frac{RT_{man}}{V_{man}}\big(f(\hat{p}_{man},\alpha_s) - g(\hat{p}_{man},n) + K(p_{man,s} - \hat{p}_{man})\big)\\ r_3 &= p_{man,s} - \hat{p}_{man}\end{aligned}$$

$$\begin{aligned}\dot{\hat{p}}_{man} &= \frac{RT_{man}}{V_{man}}\big(f(\hat{p}_{man},\alpha_s) - g(\hat{p}_{man},n) + K(\dot{m}_{air,s} - \hat{\dot{m}}_{air})\big)\\ \hat{\dot{m}}_{air} &= f(\hat{p}_{man},\alpha_s)\\ r_4 &= \dot{m}_{air,s} - \hat{\dot{m}}_{air}\end{aligned}$$

I r_3 skattas p_{man} med en ickelinjär diagnostisk observatör. Trycket p_{man} mäts och skattningsfelet förs tillbaks till observatören. Residualen r_3 visar skattningfelet för p_{man}. Fjärde residualen r_4, har en liknande konstruktion. I r_4 är det dock skattningsfelet av $\dot{m}_{air}$ som förs tillbaka till observatören. Residualen r_4 visar skattningsfelet för $\dot{m}_{air}$.

Dessa residualer är strukturerade, dvs, olika residualer är känsliga för olika fel. Det kan ses genom att studera tabell 3.2. En etta i tabellen betyder att residualen på samma rad är hög och en nolla att den är låg. Ett X betyder att effekten av ett fel på residualen i vissa fall inte är tillräckligt stark för att garantera ett robust diagnossystem.

Residualerna är starkt isolerande, vilket kan ses i tabellen. Om en etta ändras till en nolla eller tvärtom i någon rad så kan inte den raden bli identisk med någon annan rad.

	α_s	$\dot{m}_{air,s}$	$p_{man,s}$
r_1	1	1	X
r_2	0	1	1
r_3	1	0	1
r_4	1	1	0

Tabell 3.2: Residualstrukturen för diagnossystemet. Varje kolumn representerar en felaktig signal.

3.3 Residualevaluering

De fyra residualerna bör reagera enligt tabell 3.2 om ett fel inträffar och vara noll i det felfria fallet. Syftet med residualevalueringen är att kontrollera detta genom att generera ett felbeslut. För ett specifikt fel, så har residualer markerade med ett X en osäker respons. Det betyder att felbeslutet inte får bero på den residualen.

När residualerna har genererats så lågpassfiltreras dom. Sedan jämförs residualerna mot tröskelvärden. För att ett fel ska inträffa så måste alla residualerna som ska reagera enligt tabell 3.2, vara höga och alla residualerna som inte ska reagera måste vara låga. Ett exempel på detta är $p_{man,s}$-kolumnen som har regeln:

IF r_2 är *hög* AND r_3 är *hög* AND r_4 är *låg*
THEN *trycksensorfel*

Felbesluten beräknas sen genom att exekvera alla regler, en för varje fel.

3.4 Tidsdiskretisering av modellerna

Modellerna är kontinuerliga och måste därför tidsdiskretiseras innan dom kan implementeras i en dator. Eftersom diagnosen ska ske on-line är det viktigt att utföra så få beräkningar som möjligt. Derivatorna approximeras därför med Eulers metod för numerisk lösning av diffrentialekvationer. Den är enkel och medför få beräkningar. Eulers metod har detta utseende:

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

där T_s är samplingstiden. Diskretiseringen av residualerna blir med Eulers metod, på följande vis:
Residual r_1:

$$r_1 = \dot{m}_{air,s} - f(p_{man,s}, \alpha_s)$$

Residual r_2:

$$r_2 = \dot{m}_{air,s} - g(p_{man,s}, n) - \frac{V_{man}}{RT_{man}} \frac{p_{man}(t) - p_{man}(t-1)}{T_s}$$

Residual r_3:

$$\begin{aligned}\hat{p}_{man,3}(t+1) &= \hat{p}_{man,3}(t) + T_s\frac{RT_{man}}{V_{man}}\big(f(\hat{p}_{man,3}(t),\alpha_s) - \\ &\quad g(\hat{p}_{man,3},n) + K_1(p_{man,s} - \hat{p}_{man,3}(t))\big)\\ r_3 &= p_{man,s} - \hat{p}_{man,3}(t)\end{aligned}$$

Residual r_4:

$$\begin{aligned}\hat{p}_{man,4}(t+1) &= \hat{p}_{man,4}(t) + T_s\frac{RT_{man}}{V_{man}}\big(f(\hat{p}_{man,4}(t),\alpha_s) - \\ &\quad g(\hat{p}_{man,4},n) + K_2(\dot{m}_{air,s} - \hat{\dot{m}}_{air,4}(t))\big)\\ \hat{\dot{m}}_{air,4} &= f(\hat{p}_{man,4}(t),\alpha_s)\\ r_4 &= \dot{m}_{air,s} - \hat{\dot{m}}_{air,4}\end{aligned}$$

Kapitel 4

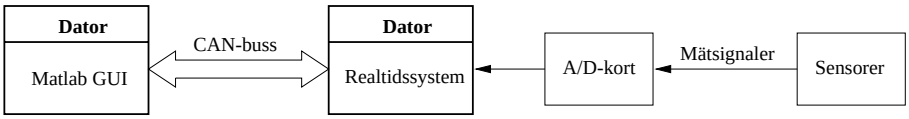
Uppbyggnad och funktion av diagnossystemet

Detta kapitel beskriver på ett övergripande sätt hur diagnossystemet är uppbyggt och vad som utförs av systemets olika delar. Kapitlet inleds med en översiktlig beskrivning av systemets uppbyggnad. Därefter följer en motivering till den valda systemstrukturen och syftet med diagnossystemet. Sedan följer två avsnitt som beskriver systemets delar, realtidssystem och GUI, och vad dessa utför, dock utan att gå ner på detaljnivå. Kapitlet avslutas med några kommentarer om systemet. En detaljerad beskrivning av diagnossystemet ges i kapitel 5 för realtidssystemet och i kapitel 6 för GUI.

4.1 Översikt av diagnossystemet

Diagnossystemet utför diagnos on-line, dvs, systemet utför diagnos på en process under tiden som processen pågår. Det medför att data måste samlas in från processen och analyseras för att avgöra om något fel har inträffat. Om ett fel har inträffat så måste det meddelas på något sätt. Figur 4.1 visar en översiktlig skiss på diagnossystemets uppbyggnad. Figuren visar diagnossystemets två huvuddelar som är implementerade på två olika datorer, nämligen:

Realtidssystem
: I realtidssystemet samlas data in från sensorer som avläser nödvändiga storheter för diagnosen. Mätsignalerna från sensorerna går till ett A/D-kort som avläses av realtidssystemet. Sedan beräknas residualer och felbeslut med hjälp av mätsignalerna. GUI meddelas



Figur 4.1: Översikt av diagnossystemet.

om något diagnostiserat fel har inträffat. Funktionaliteten hos realtidssystemet beskrivs i avsnitt 4.4 och en detaljbeskrivning av systemet finns i kapitel 5.

GUI
: Ett grafiskt användargränssnitt i Matlab [4, 5] används för att ge information till användaren om det som diagnostiseras i realtidssystemet. Namn på önskade fel visas och om ett fel inträffar indikeras detta. Dessutom kan användaren plotta olika värden som beräknas i realtidssystemet och ändra vissa parametrar som används i realtidssystemet. Funktionaliteten hos GUI beskrivs i avsnitt 4.5 och viktiga delar beskrivs i detalj i kapitel 6.

Mellan datorerna finns en CAN-buss som används för kommunikationen mellan diagnossystemets delar. En sammanfattning av diagnossystemets funktioner ges nedan:

- Utföra diagnos på implementerade processer. Diagnos kan ske på generella typer av processer. Under detta arbete implementerades luftintagssystemet för en SAAB 2.3 liters motor.
- Meddela användaren när fel inträffar. Användaren kan välja vilka fel som ska meddelas.
- Visa hur många gånger ett fel har inträffat.
- Plotta önskade värden som beräknas under diagnosen.
- Plotta önskade värden då ett fel har inträffat.
- Under diagnosen ändra tröskelvärden.

4.2 Valet av systemstruktur

Det finns ett antal orsaker till den valda systemstrukturen. De viktigaste anledningarna till att implementera realtidssystemet och GUI på olika

datorer är följande:

Snabbhet
Realtidssystemet är ett DOS-program och GUI är utvecklat i Matlab som är ett Windows-program. Även om det går att använda bägge operativsystemen samtidigt så skulle realtidssystemet bli väldigt långsamt. Realtidssystemet har tidskrav som måste hållas och det kräver ett kompakt operativsystem med realtidsegenskaper.

Robusthet
Realtidssystem konstrueras så att dom blir robusta. Dom använder små och säkra operativsystem. Ett GUI däremot är svårt att få robust. Det kräver ett större och komplexare operativsystem, som blir instabilare. Om realtidssystemet är implementerat på en annan dator än GUI så påverkas det inte om GUI kraschar.

Efterlikna verkliga förhållanden
I t.ex.en bil kommunicerar olika datorer med varandra via CAN-bussar.

Det finns program som klarar att kombinera realtidstillämpningar och samtidigt presentera resultaten grafiskt. Exempel på detta är LabView [2]. Ett annat program är dSpace [1] som består av ett GUI och i samma dator ett A/D-kort med inbyggd processor. Dessa program kräver dock fömodligen en kraftfullare dator än den som används i labbet.

4.3 Syftet med diagnossystemet

Diagnossystemet har inte utvecklats för att bli ett kommersiellt program. Det är ska istället användas för att studera och demonstrera hur modellbaserad diagnos fungerar för verkliga processer. Då är inte viktigt att vidta speciella åtgärder om ett fel uppstår. Det viktiga är istället att kunna studera de värden som beräknas och används vid diagnosen. Då är det nödvändigt att spara nya och gamla värden så att variationerna i tiden kan studeras. Dessutom bör det vara lätt att lägga till eller ta bort delar av programmet. Att implementera diagnos av ett nytt fel ska vara enkelt.

4.4 Realtidssystemets funktion

Detta avsnitt beskriver *vad* realtidssystemet utför, men inte så ingående *hur* det utförs. Avsnittet tar upp vad realtidssystemets olika delar gör och hur delarna samverkar med varandra. En ingående beskrivning av realtidssystemet finns i kapitel 5.

4.4.1 Översikt

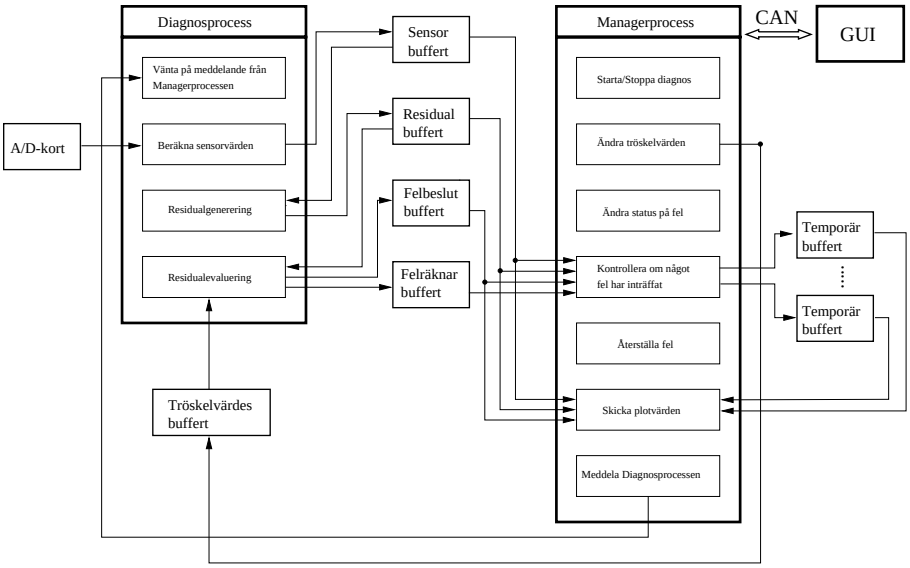
Realtidsystemet startas med ett kommando från GUIs dator. Kommandot skickas när diagnossystemet startas i Matlabs kommandoprompt. Realtidssystemets uppgift är att utföra den modellbaserade diagnosen av implementerade delsystem. Det ska dessutom kunna kommunicera med GUI, bl.a då fel inträffar. Strukturen på realtidssystemet visas i figur 4.2. Systemet består av två processer, kallade *Diagnosprocess* och *Managerprocess*, samt ett antal *buffertar* där värden sparas. Diagnosprocessen utför alla beräkningar som behövs för diagnosen och sparar resultaten i buffertar. Managerprocessen kontrollerar värdena i buffertarna och meddelar GUI om ett fel inträffar, samt sköter övrig kommunikation med GUI. Buffertarna som visas i figuren är *cirkulära*, dvs värden som är ett visst antal sampel gamla sparas, därefter skrivs de äldsta värdena över. Buffertarna beskrivs i avsnitt 5.3.

4.4.2 Diagnosprocess

Nedan följer en beskrivning av vad Diagnosprocessen gör, utgående från figur 4.2. Diagnosprocessen exekveras med en frekvens på 10 Hz. Om realtidssystemet blir hårt belastat kan dock frekvensen minska. Konsekvenserna av det har inte testats men det skulle bland annat innebära att dom filter som används får en gränsfrekvens som inte stämmer och att vissa tidsdiskretiserade ekvationer använder fel samplingstid. Det är svårt att ange var gränsen för hård belastning går eftersom det inte bara beror på antalet implementerade delsystem, utan också på hur ofta fel inträffar eller hur ofta användaren vill ha värden plottade etc. Se avsnitt 5.4 för detaljerad information.

Vänta på meddelande från Managerprocessen

Varje gång Diagnosprocessen börjar exekvera så måste den få ett meddelande från Managerprocessen. Detta görs för att synkronisera processerna, och utförs av en anledning som är lättare att inse när hela



Figur 4.2: Struktur på realtidssystemet. Figuren visar vad Diagnosprocessen och Managerprocessen kan utföra och hur buffertarna används. Figuren visar också exekveringsordningen för processerna, förutom rutan Starta/Stoppa diagnos i Managerprocessen. Om diagnosen har stoppats exekveras inga andra delar av Managerprocessen.

avsnitt 4.4 har lästs igenom. Därför förklaras anledningen till detta sist i avsnitt 4.4.3.

Beräkna sensorvärden

Det är sensorernas mätvärden som ligger till grund för diagnosen. Därför är det viktigt att sensorvärdena som används i diagnosen går att lita på. Mätstörningar måste dämpas så mycket som möjligt samtidigt som ingen viktig information får försvinna. Mätvärdena för diagnosen hämtas från ett A/D-kort. Dessa värden filtreras av ett lågpassfilter med en gränsfrekvens på 5 Hz för att undertrycka störningarna. Mätvärdena är i form av spänningar som därför måste konverteras till rätt storlek. När detta har skett så sparas värdena i en *Sensorbuffert*.

	f_1	f_2	f_3
r_1	t_{11}	t_{12}	X
r_2	t_{21}	t_{22}	t_{23}
r_3	t_{31}	t_{32}	t_{33}
r_4	t_{41}	t_{42}	t_{43}

Tabell 4.1: Tabellen visar de fel och residualer som har implementerats i realtidssystemet. Dessutom visas alla tröskelvärden. f_1 är fel i trottelvinkelsensorn, f_2 är fel luftmassflödessensorn och f_3 är fel i sensorn som mäter trycket i insugsröret.

Residualgenerering

Residualernas värden avgör om något fel har inträffat. Varje gång nya sensorvärden har beräknats och sparats så ska nya residualer genereras. Residualer kan beräknas på olika sätt, men oftast används en eller flera sensorvärden vid genereringen av en residual. Därför måste sensorvärden hämtas från Sensorbufferten. När en residual har beräknats så lågpassfiltreras den. Residualerna sparas sedan i en *Residualbuffert*.

Residualevaluering

I tabell 4.1 visas de fel som har implementerats i realtidssystemet och vilka residualer dom är beroende av. I tabellen visas också tröskelvärden. Tabellen är likadan som tabell 3.2 men där visas residualstrukturen istället. Varje felbeslut beräknas med hjälp av de residualer som felet beror på. Till varje residual som felet beror på finns ett tröskelvärde. Tröskelvärdena för en residual kan vara olika för olika fel. Felet har inträffat om absolutbeloppet för felets residualer ligger på rätt sida av motsvarande tröskelvärde. Ett X betyder att residualen på motsvarande rad inte används vid felbeslutet.

Till exempel så kan regeln för att ett fel i trottelvinkelsensorn ska inträffa beskrivas som:

$$\text{IF } abs(r_1) > t_{11} \text{ AND } abs(r_2) < t_{21} \text{ AND } abs(r_3) > t_{31} \text{ AND } abs(r_4) > t_{41} \text{ THEN } f_1$$

Tröskelvärdena kan ändras från GUI och finns sparade i en *Tröskelvärdesbuffert*. Att dom sparas i en buffert beror på att det var en lämplig datastruktur som fanns tillgänglig då felbesluten implementerades (se avsnitt 5.3).

Felbesluten beräknas genom att hämta de senaste beräknade residualerna ur Residualbufferten och tröskelvärdena ur Tröskelvärdesbufferten och sedan beräkna alla implementerade regler. Resultaten sparas sedan i en *Felbeslutbuffert*. För varje fel finns det dessutom en felräknare som håller reda på hur många gånger felbeslutet har ändrats från 'inget fel' till 'fel' sedan det meddelades första gången till GUI. Dessa felräknare uppdateras också i en *Felräknarbuffert*.

4.4.3 Managerprocess

I detta avsnitt beskrivs vad Managerprocessen gör utgående från figur 4.2. Managerprocessen sköter all kommunikation med GUI och initierar realtidssystemet vid start. Den kontrollerar om fel har inträffat, med hjälp av de värden som beräknas av Diagnosprocessen. Den sykroniserar dessutom de båda processerna. Managerprocessen exekverar med en frekvens på 10 Hz. Avsnitt 5.5 ger en detaljerad beskrivning av Managerprocessen.

Starta/Stoppa diagnos

Om GUI har skickat meddelade om att diagnosen ska starta (se avsnitt 4.5.3) så väntar Managerprocessen på att få information om vilka fel som ska kunna meddelas till GUI. När informationen har kommit så skapas de buffertar som behövs vid diagnosen (se avsnitt 5.3). Vid varje exekvering av processen därefter, så utförs alla uppgifter som Managerprocessen har enligt figur 4.2. Det betyder också att Diagnosprocessen kommer att börja exekveras när den får meddelande från Managerprocessen.

Om GUI har skickat meddelande om att diagnosen ska stoppas, så tas buffertarna bort. Sen väntar Managerprocessen på meddelande från GUI att diagnosen ska starta igen. Övriga delar av Managerprocessen utförs inte. Diagnosprocessen som varje exekvering väntar på meddelande från Mangerprocessen, kommer därför att sluta exekvera.

Ändra tröskelvärden

Om GUI har skickat meddelande om att värden i Tröskelvärdesbufferten ska ändras så väntar Managerprocessen på att få dom nya värdena från GUI. När dom har kommit så sparas dom i Tröskelvärdesbufferten.

Ändra status på fel

Vid starten av diagnosen så fick realtidssystemet information om vilka fel som ska *kunna* meddelas till GUI (se avsnitt 4.5.3). Vilka fel som ska ingå bland dessa fel kan sen inte ändras under diagnosens gång. Däremot så har dessa fel olika status som kan ändras under diagnosen. Ett fel kan ha följande status:

Meddela GUI om felet inträffar
: Det betyder att användaren har markerat felet som visas i GUI. Se avsnitt 4.5.3.

Meddela inte GUI om felet inträffar
: Användaren har inte markerat felet i GUI.

När användaren markerar eller avmarkerar ett fel i GUI så skickas information om detta till realtidssystemet som ändrar statusen.

Kontrollera om något fel har inträffat

Om nya felbeslut har skrivits in i Felbeslutbufferten så kontrolleras om något fel har inträffat som ska meddelas till GUI enligt föregående avsnitt. Managerprocessen kontrollerar alla fel som har statusen *Meddela GUI om felet inträffar*. Om ett sådant fel har inträffat, så ska de värden som felet beror på kopieras till en *Temporär buffert*. Det görs för att dessa värden inte ska skrivas över av Diagnosprocessen i efterföljande exekveringar. Notera att värden både före och efter felets inträffande kopieras. Det innebär att kopierandet inte sker denna exekvering. Se avsnitt 5.5.5 för detaljerad förklaring.

De värden som kopieras till den Temporära bufferten, är dels det felbeslut som beräknas för felet, dels de residualer som används vid felbeslutet, samt de sensorvärden som behövs vid beräkningen av dessa residualer.

Felet meddelas till GUI när värdena kopierats till den Temorära bufferten. Om samma fel inträffar igen, så kopieras inga värden till den Temporära bufferten, om inte det gamla felet är *återställt*. Vad som menas med det beskrivs nedan. Däremot skickas en felräknare till GUI, som talar om hur många gånger som felbeslutet ändrats från 'inget fel' till 'fel', sedan värdena sparats undan.

Återställa fel

Detta sker om GUI har skickat meddelande om att återställa ett fel. Då tas den Temporära buffert som skapades när felet inträffade bort. Om felet inträffar igen så kommer en ny Temporär buffert att skapas och nya värden sparas. Se avsnitt 5.5.6.

Skicka plotvärden

Om värden ska plottas i GUI så måste dom först skickas från realtidssystemet. Managerprocessen får först information om vilka värden som ska skickas. När informationen har kommit så hämtas värdena ur buffertarna. Det kan antingen vara de buffertar som uppdateras kontinuerligt eller också de Temporära buffertar som skapas för att spara undan värden då ett fel inträffar. Sedan skickas dessa värden till GUI.

Meddela Diagnosprocessen

Managerprocessen kontrollerar varje exekvering om nya värden har beräknats och sparats av Diagnosprocessen. Om detta har skett ska Managerprocessen kontrollera om något fel har inträffat och om ett fel har inträffat, kopiera värden till Temporära buffertar. Då får inte detta ske:

Ett fel har inträffat och Managerprocessen ska kopiera värden till en Temporär buffert. Managerprocessen hinner skapa den temporära bufferten och spara värden från Felbeslutbufferten. Sen tar Diagnosprocessen över processorn och hinner beräkna och spara nya värden i Sensorbufferten och Residualbufferten. Därefter tar Managerprocessen över processorn och kopierar sensorvärden och residualer till den Temporära bufferten. Då blir felbeslut förskjutna ett sampel i förhållande till sensorvärden och residualer i den Temorära bufferten.

För att förhindra detta så väntar Diagnosprocessen på ett meddelande från Managerprocessen varje gång den börjar exekvera. Managerprocessen skickar meddelandet sist i sin exekvering då den har utfört eventuella kopieringar av värden.

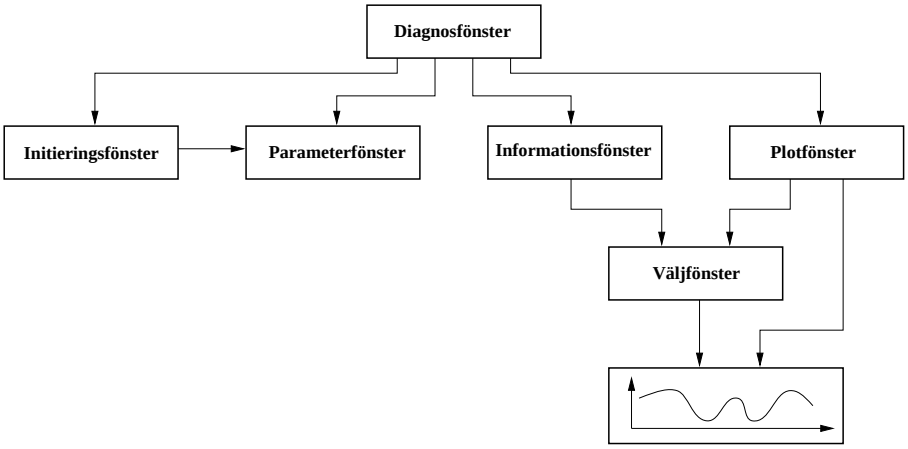
En annan tänkbar lösning skulle kunna vara att ge processerna olika prioritet som kan ändras beroende på var i exekveringen processen befinner sig.

4.5 GUIs funktion

I detta avsnitt beskrivs hur GUI är uppbyggt och vilken funktion varje del har. Grafiska objekt som är centrala vid beskrivningen namnges och är markerade med kursiv stil. En mer ingående förklaring hur funktioner utförs ges i kapitel 6.

4.5.1 Översikt

För användaren består det grafiska användargränssnittet GUI av ett antal fönster, där varje fönster har ett antal objekt. Objekten kan t.ex vara tryckknappar, radiobuttons, menyer eller text. Fönstren används för att göra diagnossystemet mer åskådligt och begränsa den mängd information som användaren behöver ha kontroll över. Figur 4.3 visar GUIs fönster och varifån olika fönster kan skapas.



Figur 4.3: GUIs olika fönster. Figuren visar varifrån olika fönster kan skapas.

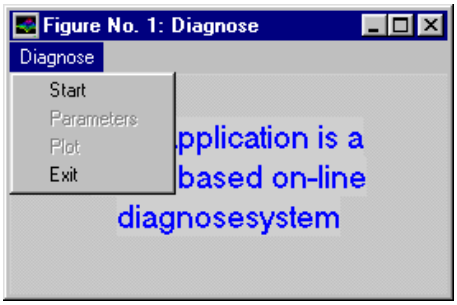
4.5.2 Delsystem

En användare ska kunna välja vilka delar av det implementerade systemet som ska diagnostiseras. Då antalet implementerade fel kan vara stort, grupperas dessa i delsystem. När användaren vid start väljer vad som ska *kunna* diagnostiseras, är det delsystemen som väljs och inte enskilda fel. Sen visas alla fel som hör till valda delsystem, i ett fönster. Där

kan enskilda fel markeras. Är ett fel markerat kommer det att indikeras om det uppstår.

4.5.3 Diagnosfönster

Diagnossystemet startas i Matlab genom att anropa funktionen *diagnos.m*. Då skickas en kommandosträng till realtidssystemet som startar Diagnosprocessen och Managerprocessen som beskrivits i avsnitt 4.4. Dessutom skapas Diagnosfönstret som kan ses i figur 4.4.



Figur 4.4: Diagnosfönstrets utseende innan delsystem har valts och diagnosen börjat.

Från början består fönstret bara av menyn *Diagnos* där undermenyerna *Start* och *Exit* är valbara. Om *Start* väljs så skapas Initieringsfönstret där parametrar och delsystem som ska kunna diagnostiseras, väljs (se avsnitt 4.5.4). När det är gjort stängs Initieringsfönstret och dom fel som hör till valda delsystem visas i Diagnosfönstret. Då ser fönstret ut som i figur 4.5. Bredvid varje fel finns en statustext som visar om felet har inträffat eller inte eller om det inte är diagnostiserat. För varje fel finns det dessutom en knapp som heter *Info*, som när den trycks in skapar ett Informationsfönster (figur 4.8). Det fönstrets funktioner beskrivs i avsnitt 4.5.6.

När delsystemen valdes skickades information till realtidssystemet om vilka fel som ska kunna diagnostiseras. Från början meddelas dock inga fel till GUI. Användaren måste först markera i Diagnosfönstret vilka fel som ska diagnostiseras. Det görs genom att klicka på önskade fel. Då får realtidssystemet informationen att detta fel ska meddelas till GUI om det inträffar (se avsnitt 4.4.3).

När delsystemen valdes blev också dom övriga undermenyerna i me-

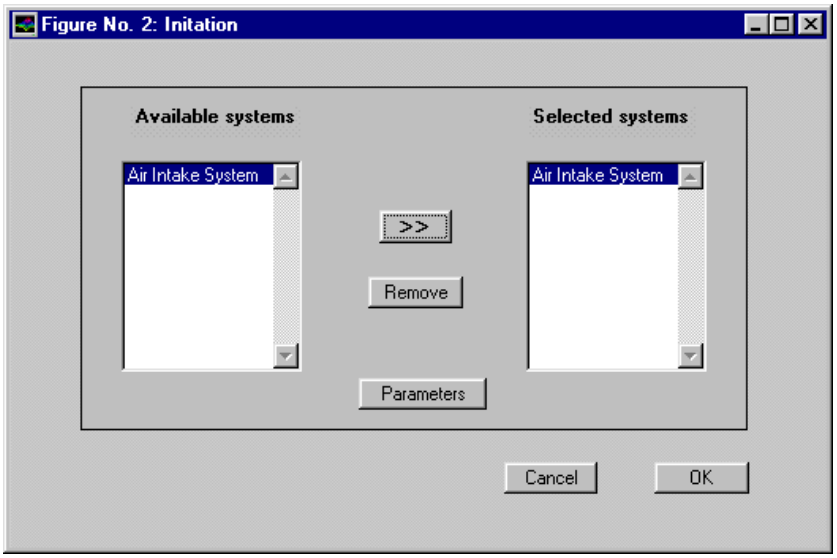
Figur 4.5: Diagnosfönstrets utseende när delsystem har valts och diagnosen har startat.

nyn *Diagnos* valbara . Om *Parameters* väljs så skapas Parameterfönstret som beskrivs i avsnitt 4.5.5. Där kan tröskelvärdena ändras, men däremot kan inte buffertstorleken ändras nu. Det beror på att buffertar redan existerar och realtidssystemet beräknar och sparar värden i buffertarna. Buffertstorleken kan bara väljas när delsystemen väljs (se avsnitt 4.5.5). Om undermenyn *Plot* väljs så skapas Plotfönstret, där önskade värden kan väljas och plottas. Se avsnitt 4.5.7 för mer information.

När diagnosen är igång så har undermenyn *Start* ändrats till *Stop*. Om *Stop* väljs så ändras utseendet på Diagnosfönstret tillbaks till hur det såg ut innan diagnosen började (figur 4.4). Dessutom skickas meddelande till realtidssystemet att det ska stoppa diagnosen och vänta på nya instruktioner.

4.5.4 Initieringsfönster

Initieringsfönstret skapas när menyn *Start* väljs i Diagnosfönstret. Utseendet på fönstret kan ses i figur 4.6.



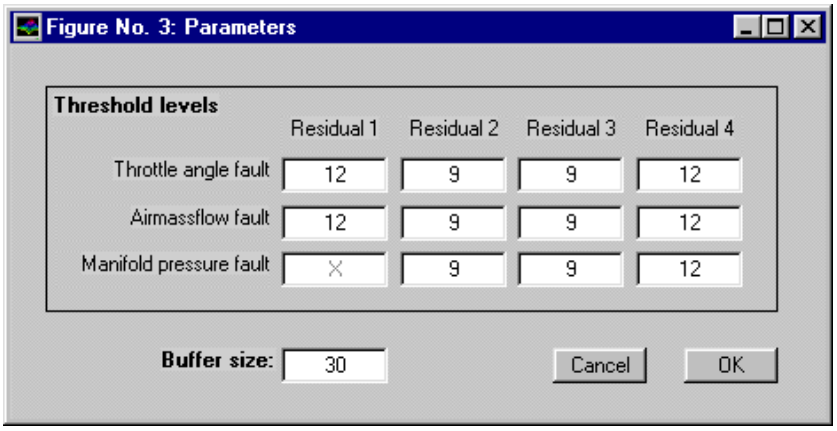
Figur 4.6: Initieringsfönster där inplementerade delsystem visas i den vänstra listboxen. Valda system visas i den högra listboxen.

I fönstret finns en listbox där alla implementerade delsystem visas. Dessutom finns en listbox där valda delsystem visas. Med knapparna >> och *Remove* kan delsystem läggas till och tas bort i listboxen för valda delsystem. Det finns också en knapp som heter *Parameters* som, när den väljs, skapar Parameterfönstret som beskrivs i avsnitt 4.5.5. Där kan tröskelvärden och storlek på buffertarna i realtidssystemet väljas. När delsystem och parametrar är valda så väljs knappen *OK*. Då stängs Initieringsfönstret och i Diagnosfönstret visas dom fel som hör till valda delsystem. Dessutom skickas information till realtidssystemet om att felen i dom valda delsystemen, ska kunna meddelas till GUI. Parametrarna som har valts i Parameterfönstret skickas också till realtidssystemet.

4.5.5 Parameterfönster

Parameterfönstrets utseende kan ses i figur 4.7. Fönstret används för att visa och ändra parametrar i realtidssystemet. Parameterfönstret ska-

pas både från Diagnosfönstret och Initieringsfönstret. I fönstret visas vilka tröskelvärden som residualerna har för de valda felen. Även storleken på buffertarna visas. Om Parameterfönstret öppnas från Initie-



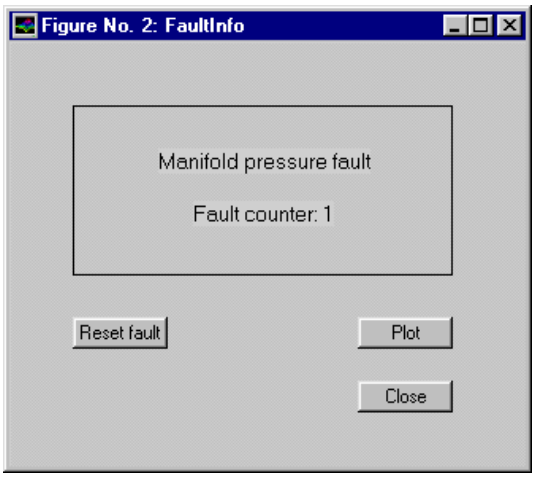
Figur 4.7: I parameterfönstret visas tröskelvärden för residualer hos valda delsystem. Dessutom visas buffertstorleken.

ringsfönstret (se avsnitt 4.5.4) så har diagnosen inte börjat än. Då kan både tröskelvärden *och* buffertstorlek ändras. När *OK*-knappen tryck in så stängs fönstret. Parametrarna skickas till realtidssystemet först när *OK* väljs i Initieringsfönstret.

Om fönstret har öppnats från Diagnosfönstret (se avsnitt 4.5.3) så kan tröskelvärdena men inte buffertstorleken väljas. Det beror på att diagnosen redan är igång och buffertarna existerar. När knappen *OK* väljs så stängs fönstret och parametrarna skickas till realtidssystemet.

4.5.6 Informationsfönster

Fönstret skapas om någon av *Info*-knapparna som finns bredvid varje fel i Diagnosfönstret (se avsnitt 4.5.3) väljs. Fönstret kan ses i figur 4.8. Där syns namnet på felet och en räknare som visar hur många gånger felet har inträffat. Det finns även en knapp som heter *Reset fault* som kan väljas om ett fel har inträffat. Om den knappen väljs så skickas meddelande till realtidssystemet att dom värden som sparades när felet inträffade ska tas bort (se avsnitt 4.4.3). Dessutom ändras statustexten i Diagnosfönstret till *No Fault*. Räknaren som finns i Initieringsfönstret nollställs också. Väljs knappen *Plot*, skapas Väljfönstret som beskrivs i



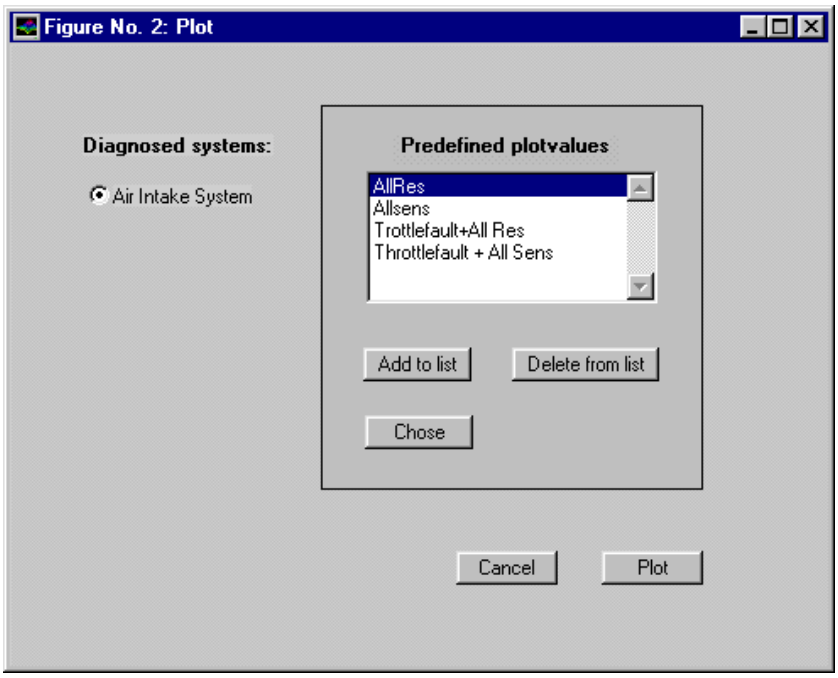
Figur 4.8: Fönster som visar information om ett fel.

avsnitt 4.5.8. Där kan önskade värden som sparades av realtidssystemet när felet inträffade, väljas och plottas.

4.5.7 Plotfönster

När undermenyn *Plot* väljs i Diagnosfönstret skapas detta fönster. Figur 4.9 visar hur fönstret ser ut. I fönstret syns alla delsystem som diagnostiseras som valbara radiobuttons. Det finns också en listbox som är tom från början. När ett delsystem väljs så visas i listboxen ett antal namn. Det är namn på fördefinierade grupper av plotvärden som finns för detta delsystem. För att lägga till nya namn till listboxen så trycks knappen *Add to list* in. Då skapas Väljfönstret som beskrivs i avsnitt 4.5.8. Där kan önskade värden väljas och ges ett namn som sedan läggs till listan av fördefinierade namn när knappen *Add to list* trycks in i Väljfönstret. Det finns även en knapp för att ta bort fördefinierade namn. Den knappen kallas *Delete from list*. Om ett fördefinierat namn väljs och knappen *Plot* trycks in så hämtas de värden som hör till namnet från realtidssystemet och plottas i ett nytt fönster.

Det finns en knapp som heter *Chose* som, när den väljs skapar ett Väljfönster. Det är alltså ett likadant fönster som när knappen *Add to list* valdes, med den skillnaden att nu finns det en knapp som heter *Plot* i Väljfönstret. När *Plot* trycks in i Väljfönstret så plottas värdena på samma sätt som när *Plot* valdes i Plotfönstret.

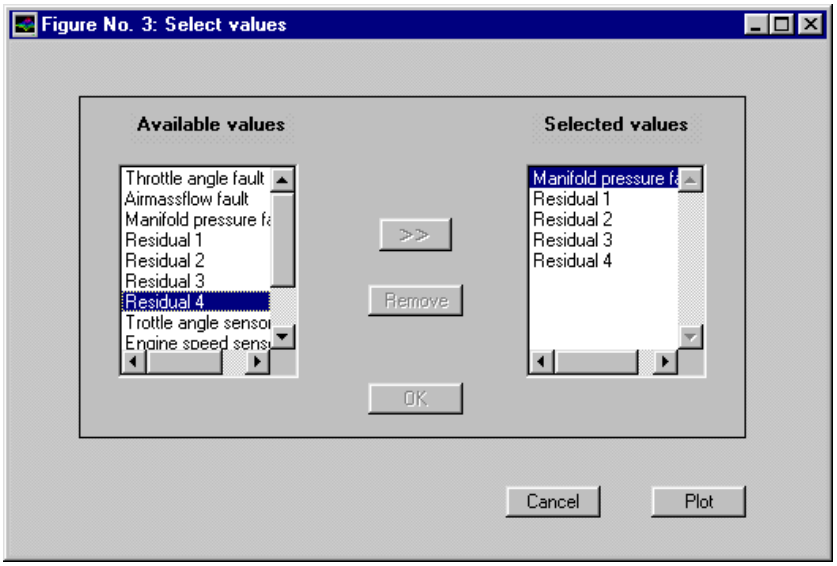


Figur 4.9: Fönster där fördefinierade plotnamn kan plottas eller skapas.

4.5.8 Väljfönster

Detta fönster kan ha tre olika utseenden beroende på varifrån fönstret skapades. Om knappen *Plot* väljs i Informationsfönstret (se avsnitt 4.5.6) så får fönstret ett utseende enligt figur 4.10. I den vänstra listboxen visas dom värden som felet är beroende av. Med det menas dels felbeslut för detta fel, dels dom residualer som felbeslutet är beroende av, samt dom sensorvärden som residualerna beror på. I den högra listboxen visas valda värden. Det finns en knapp >> som lägger ett markerat värde i den vänstra listboxen till den högra listboxen. Det finns också en knapp *Remove* som tar bort ett markerat värde i den högra listboxen. När önskade värden har valts så trycks knappen *OK* in. Då blir knappen *Plot* valbar. När den trycks in så stängs Väljfönstret och information skickas till realtidssystemet om vilka värden GUI vill ha. Sedan tas dessa värden emot av GUI och plottas.

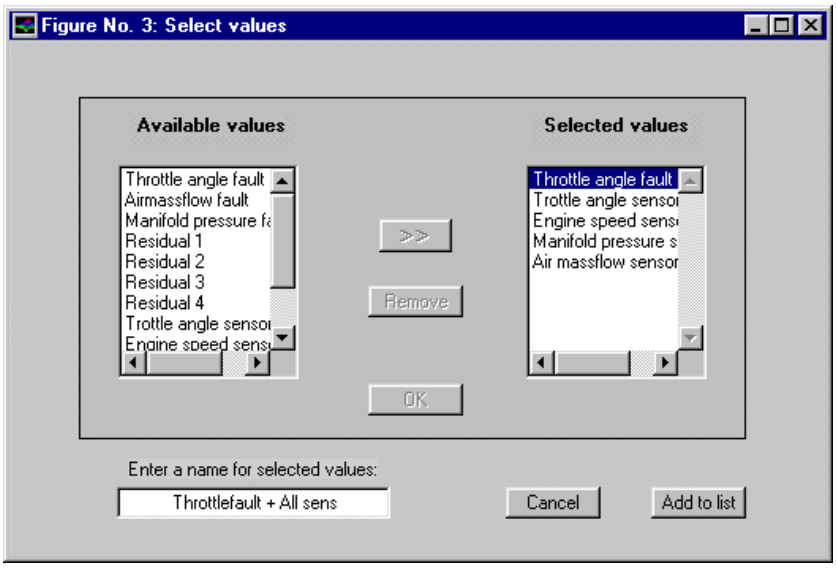
Om knappen *Chose* väljs i Plotfönstret (se avsnitt 4.5.7) så skapas ett likadant Väljfönster som det som beskrevs innan. Skillnaden är dock



Figur 4.10: Fönster där önskade värden kan väljas och sen plottas

att nu visas alla värden som hör till det delsystem som har markerats i Plotfönstret. I övrigt är fönstret likadant.

Om knappen *Add to list* trycks in i Plotfönstret så skapas ett Välj fönster som ser ut som i figur 4.11. Istället för en *Plot*-knapp så finns nu en knapp som heter *Add to list* i Väljfönstret. Nu visas alla värden som hör till det delsystem som är markerat i Plotfönstret. När önskade värden är valda och *OK* trycks in så visas en textruta där ett namn ska anges för plotten. När ett namn har skrivts in och knappen *Add to list* har valts så stängs Väljfönstret och namnet läggs till i listan i Plotfönstret.



Figur 4.11: Fönster där fördefinierade plotnamn kan skapas.

Kapitel 5

Detaljerad beskrivning av realtidssystem

Det här kapitlet beskriver viktiga delar av realtidssystemets uppbyggnad. Det förklarar hur olika funktioner utförs och viktiga datatyper och datastrukturer definieras. Kapitlet är ett komplement till avsnitt 4.4, som beskriver realtidssystemets funktion på en mer övergripande nivå. För att få behållning av detta kapitel bör åtminstonde avsnitt 4.4 ha lästs igenom och helst hela kapitel 4.

Kapitlet börjar med att förklara ett programskal som realtidssystemet bygger på. Sen beskrivs i stora drag hur flödet i realtidssystemet sker. Därefter förklaras en datatyp som kallas *buffert* (avsnitt 5.3) där alla värden sparas. Sen följer en detaljerad förklaring av det som sker i Diagnosprocessen i avsnitt 5.4 och i Managerprocessen i avsnitt 5.5 sedan initieringen av diagnossystemet har skett. En beskrivning av hur diagnossystemet kan utökas eller förändras beskrivs i avsnitt 5.6. Initieringen av diagnossystemet beskrivs i avsnitt 5.7. I avsnitt 5.8 beskrivs hur CAN-kommunikationen går till. Som exempel i hela avsnittet, används det implementerade luftintagssystemet, vilket beskrivs i kapitel 3.

5.1 Realtidsskal

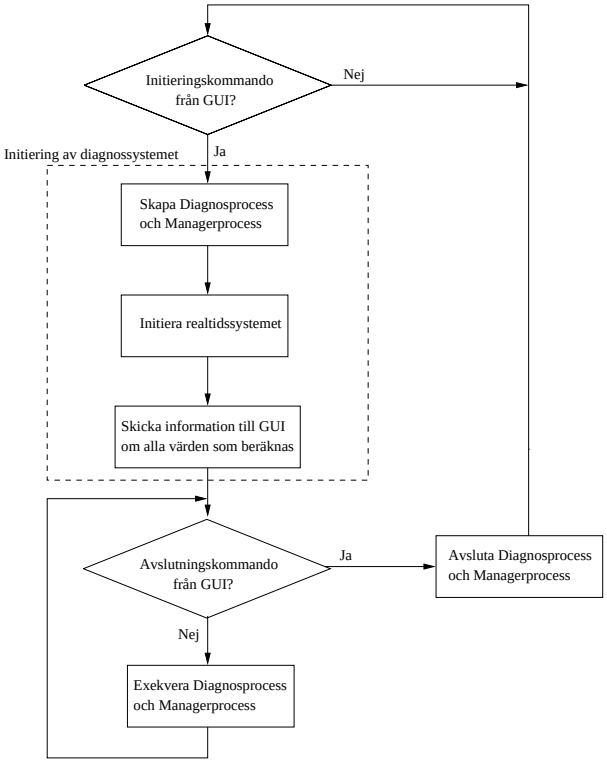
Realtidssystemet består av ett skal som sköter processhantering, IO, CAN-kommunikation etc. Skalet har utvecklats av Fordonssystem [9] och används vid realtidstillämpningar. När en realtidstillämpning ska användas så implementeras de processer som behövs som funktioner i realtidssystemet. Sen skapas instanser av processerna genom att exekve-

ra en initieringsfil. Processerna kan använda alla funktioner som finns färdiga i skalet. Det finns t.ex. funktioner för att skapa, starta och avsluta processer. Andra funktioner kan avläsa A/D-kort, skicka och ta emot CAN-meddelanden.

Skalet är skrivet i C och är ett DOS-program. Det använder sig av RT-Kernel 4.0 [10] för processhanteringen.

5.2 Flödet i realtidssystemet

För att starta diagnosen krävs först att realtidssystemet har startats. Från början utförs ingenting. När GUI startas på den andra datorn, skickas ett initieringskommando till realtidssystemet. Figur 5.1 visar ett översiktligt flödesschema på vad som sker när initeringskommandot har kommit. Kommandot exekverar en fil som skapar och startar Diagnos-



Figur 5.1: Översiktligt flödesschema av realtidssystemet.

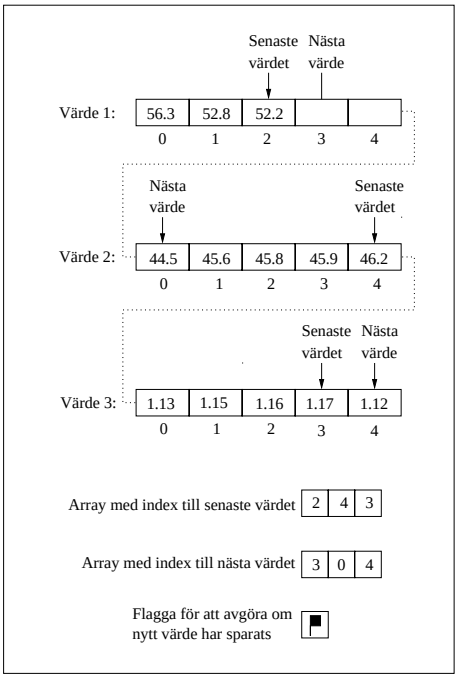
processen och Managerprocessen. Sen initieras realtidssystemet med de datastrukturer som behövs för diagnosen och GUI informeras om vad som kan diagnostiseras i realtidssystemet. Initieringen av diagnossystemet beskrivs utförligare i avsnitt 5.7. Därefter börjar Diagnosprocessen att beräkna värden som behövs för diagnosen och Managerprocessen att kontrollera felbesluten samt kommunikationen med GUI. Processerna existerar tills ett kommando skickas från den andra datorn då GUI avslutas.

5.3 Buffertar

En grundläggande funktion hos realtidsprogrammet är att spara olika värden på ett effektivt sätt. För att kunna studera dessa värdens förändring i tiden så måste gamla värden sparas. För att få struktur på det som sparas, så grupperas och sparas värden av samma typ i en gemensam abstrakt datatyp. Denna datatyp kallas här för en buffert. En buffert består av ett antal rader och kolumner. Figur 5.2 visar hur en buffert är uppbyggd. Varje rad i bufferten används för ett visst värde. Bufferten är cirkulär, vilket betyder att när ett värde har sparats på en rads sista index så kommer nästa värde att sparas på index noll. Bufferten består av följande fält:

- En array där alla värden sparas.
- En array med ett index för varje rad, där nästa värde kommer att sparas.
- En array med ett index för varje rad, där senast inlästa värde finns.
- Radstorlek hos bufferten. Visas ej i figuren.
- Kolumnstorlek hos bufferten. Visas ej i figuren.
- En flagga som kan användas för att avgöra om ett nytt värde har skrivits in i bufferten.
- En semafor för att garantera mutual exlusion i bufferten. Visas ej i figuren.

Enligt ovanstående beskrivning går det inte att välja på vilket index man vill spara ett värde. Värdet sparas på det index som ligger efter det senast inskrivna värdet. Det går dock att skriva till en godtycklig



Figur 5.2: En principskiss av en bufferts uppbyggnad

position i bufferten. Vilket sätt som används beror på det som ska sparas. Om det är värden som kontinuerligt uppdateras så är det enkelt att använda den cirkulära bufferten. Är det tröskelvärden i bufferten så måste alla platser i bufferten vara tillgängliga. Se avsnitt 5.4 för exempel på hur buffertar används.

Semaforen används för att förhindra att fler än en process försöker använda bufferten 'samtidigt'. Enda sättet att använda en buffert är genom ett antal funktioner. Då fås ett gemensamt gränssnitt som förenklar användandet av bufferten. Följande funktioner kan användas på en buffert:

AllocBuf

Funktionen skapar en buffert. Argumenten är radstorlek och kolumnstorlek. Funktionen allokerar nödvändigt minne och initierar variablerna radstorlek och kolumnstorlek till rätt värde och initierar övriga variabler till 0. Dessutom skapas en semafor med en resurs.

DeallocBuf
Tar bort en buffert. Frigör allt minne som bufferten har använt.

WriteBuffer
Skriver ett värde till en buffert. Argumenten är buffertnamn, rad i buffert och värde som ska sparas. Värdet skrivs alltid till positionen efter det senaste värdet.

WriteIndBuffer
Skriver ett värde till godtycklig plats i en buffert. Argument är buffertnamn, rad i buffert, index i raden och värde som ska sparas.

ReadBuffer
Läser ett värde ur en buffert. Funktionen har argumenten buffertnamn, rad i buffert och position från det senaste värdet. Om positionen är 0 så tas det senaste värdet, om index är 1 tas det näst senaste och så vidare.

ReadIndBuffer
Läser ett värde ur en buffert. Argumenten är buffertnamn, rad i buffert och index i raden.

HoldBuffer
Begära tillträde till en buffert. Funktionen har argumentet buffertnamn och väntar på att resursen hos buffertens semafor ska bli ledig och när den blir det så allokeras den.

ReleaseBuffer
Släpper kontrollen över en buffert. Resursen hos buffertens semafor lämnas tillbaka.

SetDirtyBuffer
Sätter flaggan som används för att avgöra om något värde har sparats i bufferten.

ResetDirtyBuffer
Återställer flaggan i bufferten.

DirtyBuffer
Kontrollerar värdet på flaggan.

RowSizeBuffer
Tar fram vilken radstorlek bufferten har.

ColSizeBuffer
Tar fram hur många kolumner bufferten har.

5.3.1 Buffertar i realtidssystemet

I realtidssystemet används buffertar av sex olika typer. Dessa buffertar kallas:

Sensorbuffert
Där sparas sensorvärden som beräknas. Används som cirkulär buffert. Se avsnitt 5.4.1.

Residualbuffert
Där sparas residualer som beräknas. Används som cirkulär buffert. Se avsnitt 5.4.2.

Felbeslutbuffert
Där sparas felbeslut som beräknas . Används som cirkulär buffert. Se avsnitt 5.4.3.

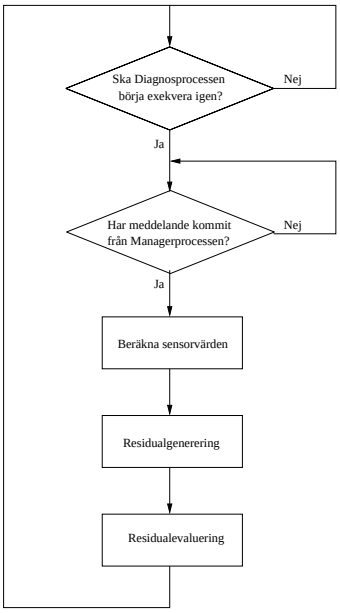
Felräknarbuffert
Där sparas felräknare för varje fel som diagnostiseras. Räknarna anger hur många gånger felen har inträffat. Används som cirkulär buffert. Se avsnitt 5.4.3.

Tröskelvärdesbuffert
Där sparas tröskelvärden för alla fel och alla residualer som felen beror på. Skriver och läser till godtyckliga index i bufferten. Se avsnitt 5.4.3.

Temporär buffert
När ett fel uppstår skapas en Temporär buffert dit värden som beror på felet kopieras. Flera Temporära buffertar kan finnas samtidigt, en för varje fel som har inträffat. Används som cirkulär buffert. Se avsnitt 5.5.5.

5.4 Diagnosprocess

Diagnosprocessen utför alla beräkningar som behövs för diagnosen och sparar dessa i buffertar. Figur 5.3 visar ett flödesschema för processen. Varje gång processen börjar exekvera så kontrolleras om meddelande har kommit från Managerprocessen. Har inget meddelande kommit så väntar



Figur 5.3: Flödesdiagram för Diagnosprocessen.

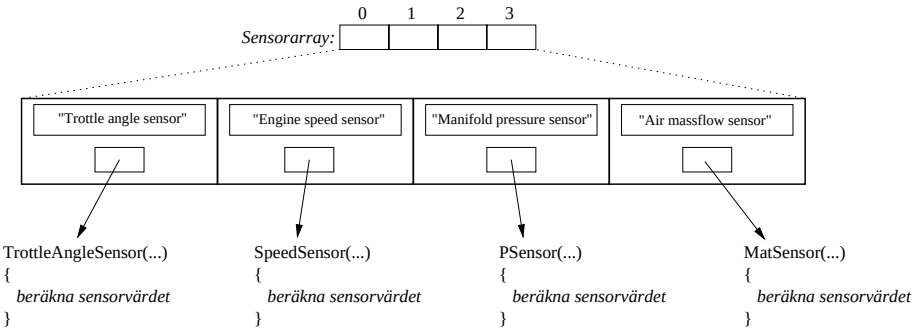
Diagnosprocessen tills det kommer. Det görs för att Managerprocessen ska hinna med sina uppgifter innan nya värden skrivs in i buffertarna. Nedan följer en beskrivning av det som sker när meddelandet har kommit.

5.4.1 Beräkna sensorvärden

En *Sensorarray* som skapades vid initieringen av diagnossystemet används vid beräkningen av sensorvärden. Figur 5.4 visar hur Sensorarrayen är uppbyggd. Varje sensorvärde som ska beräknas är ett element i Sensorarrayen. Varje element i Sensorarrayen är i sin tur en datastruktur som består av två fält. Datastrukturens fält innehåller:

- En sträng med namnet på sensorvärdet.
- En pekare till en funktion som beräknar sensorvärdet.

När nya sensorvärden ska beräknas så genomlöps Sensorarrayen och alla funktioner som pekas på i Sensorarrayen exekveras. Sensorvärdena sparas i en Sensorbuffert. Varje funktion beräknar sensorvärdet genom att

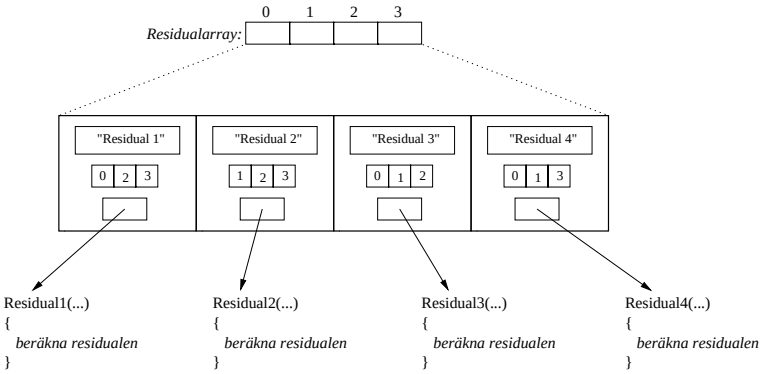


Figur 5.4: Sensorarray som används för att beräkna sensorvärden för det implementerade luftintagssystemet. Arrayen skapas vid initieringen av realtidssystemet.

läsa av en port på A/D-kortet, filtrera värdet och sen konvertera värdet till rätt nivå.

5.4.2 Residualgenerering

Vid initieringen skapades en *Residualarray* för beräkningen av residualer. Dess uppbyggnad visas i figur 5.5. Varje residual som ska beräknas



Figur 5.5: Residualarray som används vid beräkningen av residualer. Skapas vid initiering av realtidssystemet.

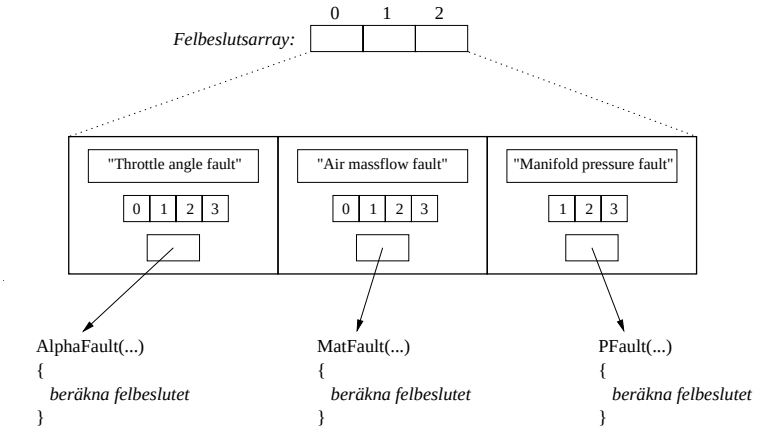
är ett element i Residualarrayen. Varje element i arrayen är en struktur med fyra fält:

- En sträng med namnet på residualen.
- En pekare till en funktion som beräknar residualen.
- En array som anger vilka sensorvärden funktionen är beroende av för att kunna beräkna residualen. Dessa sensorvärden anges som index till den Sensorarray som beskrevs ovan.
- En variabel som talar om hur många sensorvärden residualen är beroende av. Den visas inte i figuren.

Vid residualgenereringen så genomlöps Residualarrayen och funktionerna exekveras. Funktionerna använder sig av de modeller som beskrivs i kapitel 3 för att beräkna residualerna. Residualerna beräknas och lågpassfiltreras. Residualerna sparas sen i en Residualbuffert. Residualberäkningarna är implementerade enligt den tidsdiskretiserade form som visas i avsnitt 3.4.

5.4.3 Residualevaluering

Vid residualevalueringen används en *Felarray* som skapades under initieringen. Felarrayens uppbyggnad kan ses i figur 5.6. Varje fel som kan



Figur 5.6: Felarray som används vid beräkningen av felbeslut. Skapas vid initiering av realtidssystemet.

detekteras är ett element i Felarrayen. Dessa element består i sin tur av en struktur med fyra fält:

- En sträng med namnet på felet.
- En pekare till en funktion som beräknar felbeslutet.
- En array som anger vilka residualer funktionen är beroende av för att kunna beräkna felbeslutet. Anges som index till den Residualarray som beskrevs ovan.
- En variabel som talar om hur många residualer felbeslutet är beroende av. Den visas inte i figuren.

Felarrayen genomlöps och funktionerna exekveras varje gång nya felbeslut ska beräknas. Funktionerna returnerar en etta om ett fel har uppstått och noll annars.

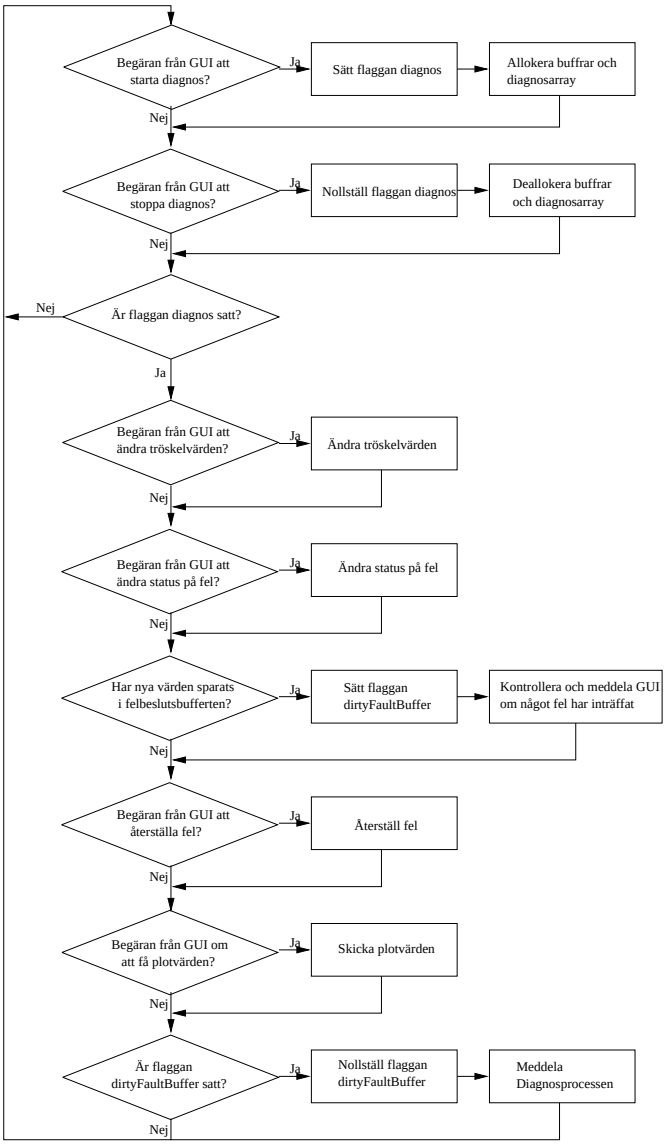
Varje gång nya felbeslut har beräknats så kontrolleras om några *nya* fel har uppstått. Det görs genom att för varje fel, kontrollera om felbeslutet var noll förra exekveringen och ett den här exekveringen. Om så är fallet uppdateras en Felräknarbuffert (se avsnitt 5.3). Den har en felräknare för varje fel. Den har också en flagga som används av Managerprocessen för att avgöra om nya värden har sparats i Felräknarbuffertten. Denna flagga sätts. Felbesluten sparas till sist i en Felbeslutbuffert (se avsnitt 5.3). Även Felbeslutbuffertens flagga sätts. Flaggan används av Mangerprocessen att avgöra när nya felbeslut har beräknats.

5.5 Managerprocess

Managerprocessen sköter all kommunikation med GUI. Den kontrollerar om något fel som ska meddelas till GUI har inträffat. Ett flödesschema för processen visas i figur 5.7. I figuren visas inte vad som sker när Managerprocessen skapas. Detta beskrivs istället i avsnitt 5.7. Flödesschemat tar inte upp allt som sker, utan det förklaras istället i texten. Det har gjorts för att begränsa flödesschemats storlek. I avsnittet beskrivs Managerprocessen utgående från flödesschemat.

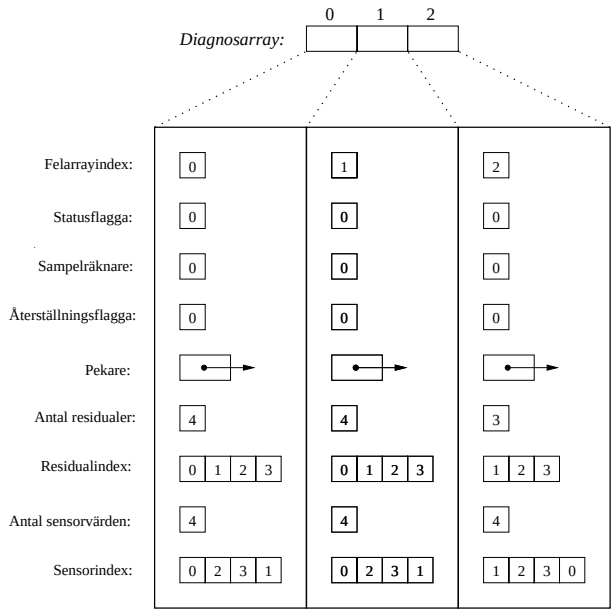
5.5.1 Starta diagnos

GUI skickar först ett meddelande att realtidssystemet ska börja utföra diagnos. Då väntar Managerprocessen på information om vilka fel som ska kunna meddelas till GUI. Felen som skickas, anges som index till den Felarray som visas i figur 5.6. När GUI har skickat dessa index, skapas en *Diagnosarray* enligt figur 5.8. Figuren visar hur Diagnosarrayen



Figur 5.7: Flödesdiagram för Managerprocessen.

initialiseras för felen som kan diagnostiseras i luftintagssystemet. Varje fel som ska kunna diagnostiseras är ett element i Diagnosarrayen. Varje element består i sin tur av en datastruktur med ett antal fält, nämligen:



Figur 5.8: Diagnosarray som skapas när GUI har skickat information om vilka fel som ska kunna meddelas.

Felarrayindex

Index till ett fel i Felarrayen som GUI angett att det vill ha möjlighet att få meddelande om.

Statusflagga

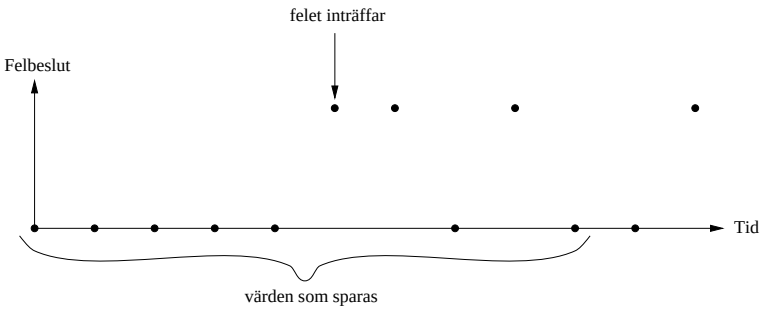
Flaggan anger om felet ska meddelas till GUI om det inträffar. Från början är flaggan noll. Då meddelas inte felet.

Sampelräknare

När felet inträffar börjar Sampelräknaren att räkna upp. Används för att avgöra när värden ska kopieras till en Temporär buffert. Om felet inträffar ska nya värden till ett antal av halva buffertradlängden beräknas innan dom kopieras. Se figur 5.9. Anledningen till det är att det är intressant att se hur värden ändras även *efter* det att felet inträffat. Sampelräknaren initialiseras till noll.

Återställningsflagga

Är flaggan satt har felet inträffat och värden har kopierats. Flaggan initialiseras till noll.



Figur 5.9: Visar när värden kopieras om ett fel inträffar, då buffertarna som används har en radlängd på 10 värden. Värden kopieras till en Temporär buffert när nya värden till ett antal av halva buffertradlängen har sparats.

Pekare
: En pekare till datatypen buffert (se avsnitt 5.3). Den pekar på en Temporär buffert som skapas om felet uppstår och dit värden kopieras.

Antal residualer
: Anger hur många residualer som felet är beroende av, för att beräkna felbeslut.

Residualindex
: En array med index till residualerna i Residualarrayen. Arrayen är likadan som den som finns i Felarrayen för samma fel. Används för att avgöra vilka residualer som ska kopieras om felet uppstår.

Antal sensorvärden
: Anger hur många sensorvärden som felet är beroende av för att beräkna felbeslut.

Sensorindex
: En array med index till sensorvärdena i Sensorarrayen. Arrayen tas fram genom, att för varje residual som felet är beroende av, ta fram dom sensorvärden som residualen är beroende av. Används för att avgöra vilka sensorvärden som ska kopieras om felet uppstår.

När Diagnosarrayen har skapats så sätts också en flagga som kallas *diagnos* som styr flödet i Managerprocessen (se figur 5.7). Buffertarna

som behövs vid diagnosen allokeras också. Buffertstorleken har skickats från GUI. Tröskelvärden som behövs för diagnosen har också skickats och Tröskelvärdesbufferten, som används vid residualevalueringen (se avsnitt 5.4.3), uppdateras med dessa värden. Till sist så sätts flaggan *dirtyFaultBuffer* som kan ses i figur 5.7 för att Diagnosprocessen ska meddelas.

5.5.2 Stoppa diagnos

När GUI skickar meddelande att diagnosen ska stoppas så nollställs flaggan *diagnos*. Det betyder att Managerprocessen bara kommer att vänta på ett meddelande att starta diagnosen igen. Då kommer inte Diagnosprocessen att få något meddelande från Managerprocessen vilket medför att den kommer att sluta exekveras.

Dessutom så avallokeras minnet som Diagnosarrayen och buffertarna använder.

5.5.3 Ändra tröskelvärden

Om GUI meddelat att tröskelvärdena ska ändras så väntar Managerprocessen på dom nya värdena. När dom kommit så uppdateras Tröskelvärdesbufferten.

5.5.4 Ändra status på fel

GUI har skickat ett index till Diagnosarrayen i figur 5.8 och ett värde som statusflaggan på detta index ska ändras till (se avsnitt 4.4.3). Om flaggan ska sättas till noll tas även den Temporära buffert som eventuellt existerar för felet, bort.

5.5.5 Kontrollera om något fel har inträffat

Om nya värden har sparats i Felbeslutsbufferten så sätts flaggan *dirtyFaultBuffer*. Den flaggan används för att avgöra när Diagnosprocessen ska meddelas. Dessutom nollställs flaggan i Felbeslutsbufferten så att Managerprocessen kan avgöra när nya värden har sparats igen. Sen kontrolleras om något fel som ska diagnostiseras har inträffat. Det görs genom att gå igenom Diagnosarrayen (figur 5.8) och kontrollera vilka index som har statusflaggorna satta. För dessa index i Diagnosarrayen kan ett antal olika saker utföras beroende på dess tillstånd:

1. Har felbeslutet ändrats från noll förra samplingen till ett den senaste samplingen och sampelräknaren är noll?

   Sätt sampelräknaren till ett. Det betyder att felet har inträffat senaste samplet.

2. Är sampelräknaren skild från noll?

   Räkna i så fall upp den med ett. Talar om hur många samplingar som skett sedan felet inträffade.

3. Är sampelräknaren lika med halva radlängden hos buffertarna och återställningsflaggan är inte satt?

   Skapa en Temporär buffert och kopiera dom värden som felet beror på till bufferten. Sätt dessutom återställningsflaggan. Meddela GUI vilket fel som har inträffat.

4. Har felbeslutet ändrats från noll förra samplingen till ett den senaste samplingen och återställningsflaggan är satt?

   Skicka information till GUI hur många gånger felet har inträffat. Detta sker alltså om felvärden finns undansparade och felet inträffar igen. Då uppdateras GUI med hur många gånger felet har uppstått sedan värdena sparades undan. Den informationen finns i Felräknarbufferten (se avsnitt 5.4.3) på samma rad som felarray-indexet i Diagnosarrayen (se figur 5.8).

5.5.6 Återställa fel

GUI har skickat ett index till Diagnosarrayen. På detta index nollställs statusflaggan, åtrställningsflaggan och sampelräknaren och den Temporära buffert där felvärden finns sparade avallokeras. Dessutom nollställs felräknaren för felet som finns i Felräknarbufferten (se avsnitt 5.4.3).

5.5.7 Skicka plotvärden

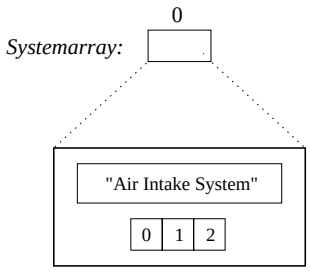
Om GUI har begärt att få plotvärden så väntar Managerprocessen först på information om vilka värden som den ska skicka tillbaka till GUI. Informationen talar om vilken buffert värdet finns i och vilken rad i bufferten det är sparat på. Buffertarna kan antingen vara de som uppdateras kontinuerligt eller en Temporär buffert som skapades när ett fel inträffade. Se avsnitt 5.3. När informationen har kommit så skickas önskade värden till GUI.

5.5.8 Meddela Diagnosprocessen

Om flaggan dirtyFaultBuffer (se avsnitt 5.5.5) är satt så meddelas Diagnosprocessen. Då börjar den att beräkna nya värden. Dessutom nollställs flaggan så att inte Diagnosprocessen meddelas förrän kontroll av senast beräknade felbeslut har utförts av Managerprocessen.

5.6 Utökning och förändring av diagnossystemet

Målet har varit att få ett diagnossystem som är lätt att utöka med diagnos av nya fel. En uppdatering kan innebära att nya sensorvärden, residualer och felbeslut måste beräknas. Att lägga till dessa nya beräkningar bör ske på så få ställen i koden som möjligt. För att uppnå detta används en Sensorarray (figur 5.4), en Residualarray (figur 5.5), en Felarray (figur 5.6) och en *Systemarray* (figur 5.10). Om ett nytt värde ska imple-



Figur 5.10: Systemarray som anger vilka delsystem som är implementerade. Systemarrayen innehåller namn på delsystemen och index till dom fel som ingår i delsystemen. Skapas vid initiering av realtidssystemet.

menteras i realtidssystemet så utökas arrayen för denna typ av värden, med ett element. Sen fylls fälten i med dom rätta uppgifterna. Om det är en array med pekare till funktioner som beräknar värdet, så måste förstås funktionen skapas också. En konstant som anger arrayens storlek måste också uppdateras. Inga andra delar av koden behöver ändras.

Genom att använda arrayer på detta sätt blir det enkelt att ta fram vilka andra värden ett speciellt värde är beroende av. Det används när värden ska sparas då ett fel har uppstått. I avsnitt 5.4 beskrivs de olika funktionerna som pekas på i arrayerna.

5.7 Initiering av diagnossystemet

Detta avsnitt beskriver vad som händer när diagnossystemet startas från kommandoprompten i Matlab och ett initieringskommando skickas till realtidssystemet. Figur 5.1 visar vad som sker vid initieringen av diagnossystemet.

5.7.1 Skapa processer och initiera realtidssystemet

Först skapas och startas Diagnosprocessen och Managerprocessen. Diagnosprocessen väntar från början på ett meddelande från Managerprocessen. Managerprocessen skapar först Sensorarrayen (figur 5.4), Residualarrayen (figur 5.5), Felarrayen (figur 5.6) och en Systemarray (figur 5.10).

5.7.2 Skicka information till GUI

Diagnossystemet är uppbyggt så att all information om det som diagnostiseras finns i realtidssystemet. GUI ska bara visa vad som sker i realtidssystemet. Detta gör att utökningar och förändringar bara behöver införas i realtidssystemet. Vid start av diagnossystemet måste därför information om det som beräknas i realtidssystemet skickas till GUI. Det som skickas till GUI är helt enkelt en teckensträng med innehållet i Sensorarrayen, Residualarrayen, Felarrayen och Systemarrayen, dock utan pekarna till funktionerna. Avsnitt 5.4 beskriver arrayerna. Teckensträngen byggs upp som ett antal delsträngar med olika tecken som avskiljare. Figur 5.11 visar hur teckensträngen är uppbyggd. Strängen delas sen upp på samma sätt i GUI som den skapades i realtidsprogrammet. Avsnitt 6.4 beskriver vad GUI gör med teckensträngen.

5.8 CAN-kommunikation

Meddelanden skickas över CAN-bussen i paket på max åtta byte åt gången. När fler än ett meddelande i följd måste skickas över CAN-bussen så används de två första byten för att indexera paketen. Figur 5.11 visar hur teckensträngen som beskrevs i förra avsnittet indexeras och skickas. Första paketet har det högsta indexet och det sista paketet har index 0. Det gör det möjligt för mottagaren att upptäcka om paketen kommer i rätt ordning. Mottagaren kontrollerar helt enkelt att det senaste paketet har ett lägre index än det förra paketet.

Teckensträng:

System:Air Intake System(0,1,2);Fault:Trottle angle fault(0,1,2,3)Airmassflow fault(0,1,2,3)
Manifold pressure fault(1,2,3);Residual:Residual 1(0,2,3)Residual 2(1,2,3)Residual 3(0,1,2)
Residual 4(0,1,3);Sensor:Trottle angle sensor()Engine speed sensor()Air massflow sensor()
Manifold pressure sensor();

Hur strängen skickas:

Index:	Sträng:
50	System
49	:Air I
48	ntake
⋮	⋮
1	re sen
0	sor();

Figur 5.11: Om luftintagssystemet är det enda implementerade systemet så blir teckensträngen enligt figuren. Den visar också hur strängen delas upp och skickas över CAN-bussen.

Strängen skickas genom att, för varje paket anropa en funktion som finns i realtidsskalet. Funktionen tar som argument en pekare till paketet, längden på paketet och ett identifieringsnummer.

För att ta emot meddelanden används funktioner som finns tillgängliga i realtidsskalet. Först skapas en brevlåda med ett identifieringsnummer och en struktur där meddelandet hamnar tillsammans med längden på meddelandet. För att kontrollera om ett meddelande har kommit anropas en funktion. Argument till funktionen är brevlådan och strukturen där meddelandet hamnar. Det går också att specifiera en tidsperiod som funktionen ska vänta på ett meddelande.

Hur GUI hanterar CAN-kommunikationen beskrivs i avsnitt 6.3.

5.9 Kommentarer om realtidssystemet

Det är värt att upprepa att Diagnosprocessen alltid beräknar alla värden som finns i arrayerna som skapades vid start. Vilka fel som meddelas till GUI är dock en annan sak. När användaren vill starta diagnosen väljs vilka fel som ska vara möjliga att få meddelande om. GUI skickade sen denna information till realtidssystemet. Notera att det står *ska vara möjliga att få meddelande om*. Från början meddelas dock inga fel till GUI. Diagnosarrayen som skapades av Managerprocessen, har för varje index en flagga som avgör om felet ska meddelas till GUI. Från början sätts varje flagga till 0. Användaren markerar sen i GUI vilka fel som han vill ha information om. Varje gång ett fel markeras i GUI, skickas

ett meddelande till Managerprocessen som sätter motsvarande flagga i Diagnosarrayen till 1. Först därefter meddelas GUI om detta fel inträffar.

Anledningen till detta är att användaren, sedan han valt vilka fel som ska visas på skärmen, interaktivt ska kunna välja vilka fel som ska diagnostiseras.

Kapitel 6

Detaljerad beskrivning av GUI

Det här kapitlet beskriver viktiga delar av GUI. Kapitlet är ett komplement till avsnitt 4.5, som beskriver GUIs funktion. För att få behållning av detta kapitel bör kapitel 4 ha lästs igenom först.

Kapitlet börjar med att beskriva grafiska objekt och datatyper i Matlab. Sen förklaras dynamisk dataöverföring, DDE, som har använts mellan GUI och det gränssnitt som används vid CAN-kommunikationen. Sedan beskrivs delar av GUI som har central betydelse. Dessa delar separeras i dom olika fönster som GUI består av. Grafiska objekt och händelser som inte har avgörande betydelse utelämnas för inte göra beskrivningen för omfattande.

Möjligheten att använda Matlabs funktioner för matematiska beräkningar, plottning och dess flexibla datastrukturer har varit en stor fördel vid utvecklandet av GUI. Däremot blir snabbheten lägre jämfört med program skrivet i tex C++.

6.1 Grafiska objekt i Matlab

Det grafiska användargränssnittet i Matlab [4, 5] består av olika fönster och i varje fönster finns ett antal objekt. Objekten kan t.ex vara pushbuttons, radiobuttons, menyer, text eller listboxar. Alla fönster och objekt som skapas har en *handle*. Den används som argument till funktioner som används på objektet. Funktionerna kan antingen ändra eller hämta objektets egenskaper.

Varje objekt har ett antal egenskaper som kan påverkas. Det kan t.ex

vara position, storlek, valbarhet eller namn. En användbar egenskap är *UserData*. Det är en matris som kan kopplas till ett objekt eller fönster. Där kan nödvändiga data sparas som existerar så länge fönstret är öppet. En fördel med UserData är att det inte är en global variabel som kan ändras av andra delar av GUI eller från kommandoprompten. UserData används hos de flesta objekten och fönstren i GUI exempelvis för att hålla reda på delsystem och fel som ska diagnostiseras eller när värden ska plottas.

En viktig egenskap hos objekten kallas för *CallBack*. Den egenskapen definierar vad som ska ske när användaren påverkar objektet. Ett objekts CallBack har en sträng associerad till sig. När objektet påverkas evalueras strängen av Matlabs evalfunktion. Detta GUI är uppbyggt så att varje objekts CallBack är en sträng med namnet på en M-funktion samt funktionens argument. Ett objekts CallBack kan se ut så här:

'dfun("plot")'

När objektet påverkas så exekveras funktionen *dfun*. Argument till funktionen är strängen *"plot"*. Strängen används för att exekvera rätt del av funktionen. Alla olika händelser som kan utföras i GUI ges unika namn och används i en CASE-sats i funktionen. CASE-satsen jämför namnen med det argument som funktionen har, i det här fallet *"plot"* och exekverar den delen av funktionen. På detta sätt samlas alla händelser som kan ske, i en funktion, vilket gör programmet mer överskådligt.

6.2 Datatyper i Matlab

I Matlab 5 [6] har det införts nya datatyper och datastrukturer. Datatyper som har införts är bland annat *integer*, *float* och *cell*. En användbar datastruktur som införts kallas för *cellarray*. En sådan kan t.ex. se ut som i figur 6.1. I en cellarray kan variabler av olika datatyper och storlek samlas i en enda indexerbar array eller matris. Om t.ex.index (2,2) i figuren hämtas, fås hela arrayen som ligger på detta index. Därefter kan ett visst index i arrayen hämtas. Cellarrayer har bland annat använts för att spara arrayerna som skickas från realtidssystemet vid start (se avsnitt 6.4) och dom fördefinierade plotvärden som kan skapas (avsnitt 6.5.5).

6.3 Dynamisk dataöverföring, DDE

Dynamic Data Exchange, DDE, är en kommunikationsteknik för att utbyta data mellan olika applikationer. DDE sker alltid mellan en klient och en server. Klienten initierar datautbytet genom att skapa en konversation med servern. Klienten får då en *handle* (ej att förväxla med *handle* i avsnitt 6.1) som används av klienten för att identifiera konversationen. Klienten kan sen begära datautbyte med servern. När klienten inte längre behöver konversationen så avslutas den.

DDE har använts mellan GUI och ett gränssnitt för CAN-bussen, CAN_DRV [3], som beskrivs i bilaga B. I Matlab finns funktioner för DDE som har använts i GUI. Dessa beskrivs i bilaga C.

6.4 Initiering av GUI

Avsnittet beskriver vad som sker när funktionen *diagnos.m* anropas från Matlabs kommandoprompt, vilket startar och initierar diagnossystemet.

När diagnossystemet startas skickas en kommandosträng till realtidssystemet som startar Diagnosprocessen och Managerprocessen. Managerprocessen skapar arrayerna för sensorvärden, residualer, fel och delsystem, vilket beskrivs i avsnitt 5.7. Processen skapar dessutom en teckensträng av arrayerna och skickar den till GUI.

Teckensträngen skickas på det sätt som beskrivs i avsnitt 5.8. När hela strängen har kommit så jämförs den med en teckensträng som sparats sedan tidigare. Den teckensträngen sparades senaste gången realtidssystemet modifierades. Om strängarna är olika så betyder det att realtidssystemet har modifierats igen. I så fall delas den senaste strängen upp på samma sätt som arrayerna i realtidssystemet. Det vill säga att sensorvärden, residualer, fel och delsystem skiljs åt och sparas i olika cellarrayer. I figur 6.1 visas vilken information som sparas för varje fel. Namnen på varje fel läggs i kolumn ett, residualerna som varje fel beror på läggs i kolumn två. För felen så tillkommer dessutom vilka tröskelnivåer som varje residual jämförs mot. Dessa läggs i kolumn tre. Sensorvärdenas cellarray består enbart av namn. Residualernas cellarray består av namn och vilka sensorvärden dom beror på. Delsystemens cellarray består av namn och vilka fel som hör till varje delsystem. Cellarrayerna samlas sen i en enda stor cellarray och sparas i en fil. Dessutom sparas den senaste teckenstängen i samma fil. Är teckensträngarna som jämfördes med varandra identiska, så används den gamla cellarrayen

{'Throttle angle fault'}	{ [0 1 2 3] }	{ [12 9 12 9] }
{ 'Air massflow fault' }	{ [0 1 2 3] }	{ [12 9 12 9] }
{ 'Manifold pressure fault' }	{ [1 2 3] }	{ [9 12 9] }

Figur 6.1: En cellarray med information om dom fel som kan diagnostiseras. Kolumn ett innehåller namnet på felen, kolumn två anger vilka residualer felen är beroende av och kolumn tre anger tröskelvärden för varje residual.

som togs fram förra gången realtidssystemet modifierades.

Från realtidssystemet skickas också den samplingstid som används och ett standardvärde på buffertstorleken. Samplingstiden sparas i samma fil som cellarrayen. Även buffertstorleken sparas där om det inte redan finns ett sparat värde för den.

Nu har GUI fått all nödvändig information om det som diagnostiseras i realtidssystemet. Nu skapas också Diagnosfönstret.

6.5 Beskrivning av vissa objekt och händelser i GUI

Följande avsnitt kompletterar avsnitt 4.5 med en detaljbeskrivning av viktiga delar av GUI. Beskrivningen är uppdelad i dom olika fönster som GUI består av.

6.5.1 Parameterfönster

Figur 4.7 visar hur fönstret ser ut och avsnitt 4.5.5 beskriver dess funktion.

Buffersize Buffervärdet som visas när fönstret skapas, tas från den fil som beskrevs i avsnitt 6.4.

Threshold levels Tröskelvärdena som visas när fönstret skapas, tas från cellarrayen i samma fil.

OK När parametrarna har satts till önskade värden och *OK* trycks in sparas buffervärde och tröskelvärdena i filen igen.

6.5.2 Initieringsfönster

Fönstret visas i figur 4.6 och avsnitt 4.5.4 beskriver dess funktion.

Delsystem Delsystemen som visas i vänstra listboxen tas från cellarrayen som beskrevs i avsnitt 6.4.

OK När knappen *OK* trycks in sparas valda delsystem i en array. Det som sparas är indexen till delsystemens plats i den cellarray som beskrevs i avsnitt 6.4. I cellarrayen har varje delsystem index till de fel som hör till delsystemet. Dessa index tas fram och sparas i en annan array. Indexen skickas dessutom till realtidssystemet. Där skapas Diagnosarrayen (se avsnitt 5.5) med hjälp av dessa index. En *hot-link* skapas för att fel automatiskt ska kunna meddelas till GUI. Vad en hot-link är beskrivs i bilaga B och C. Arrayerna med delsystemindex och felindex används för att skapa Diagnossystemets nya utseende (se figur 4.5) där delsystemen och felens namn visas. Arrayerna sparas i Diagnosfönstrets UserData (se avsnitt 6.1).

6.5.3 Diagnosfönster

Figur 4.4 visar hur fönstret ser ut innan diagnosen har startat och 4.5 visar hur fönstret ser ut efter att diagnosen har startat. Fönstrets funktion beskrivs i avsnitt 4.5.3.

Status När ett av felen som är markerade i Diagnosfönstret inträffar, så uppdateras en variabel med information om felet. Detta sker via den hot-link som skapades tidigare. Sen exekveras en funktion som tar variabeln med felinformation som argument och meddelar i rätt statusruta att felet har inträffat.

6.5.4 Informationsfönster

Figur 4.8 visar fönstrets utseende. Fönstrets funktion beskrivs i avsnitt 4.5.6.

Fault counter Räknaren uppdateras varje gång felet som hör till det Informationsfönster som visas, inträffar. Detta sker via den hotlink som har beskrivits. Räknaren sätts till noll när ett fel återställs.

Plot Om knappen *Plot* trycks in så tas dom värden som felet är beroende av fram. Värdena tas fram ur cellarrayen som beskrevs i avsnitt 6.4.

6.5.5 Plotfönster

Fönstret visas i figur 4.9 och beskrivs i avsnitt 4.5.7.

Predefined values Fördefinierade värden finns för alla delsystem, sparade i en cellarray. Det är en annan cellarray än den som beskrivs i avsnitt 6.4 om initieringen. Denna cellarray består av delsystemens index, namnen på de fördefinierade värdena samt index till dessa värden. Delsystemens och värdenas index avser den cellarray som sparades vid initieringen.

Plot När ett fördefinierat namn är markerat i listboxen så kan *Plot* väljas. Då tas indexen för de värden som hör till detta namn fram och skickas till realtidssystemet. Sen väntar GUI tills alla plotvärden har skickats tillbaka och plottar sen dessa.

6.5.6 Väljfönster

Fönstret visas i figur 4.10 och 4.11 och beskrivs i avsnitt 4.5.8.

Available values Värden tas fram ur cellarrayen (se avsnitt 6.4) som sparades vid initieringen.

Plot Om fönstret ser ut som i figur 4.10 så kan värden plottas härifrån. De valda värdenas index finns sparade i högra listboxens UserData. När *Plot* väljs så skickas indexen till realtidssystemet. När alla värden har skickats tillbaks plottas dessa.

Add to list Om fönstret ser ut som i figur 4.11 så väljs värden som ges ett fördefinierat namn. När det är gjort så tas indexen som finns i högra listboxens UserData och det namn som har angetts och sparas i cellarrayen för fördefinierade värden.

Kapitel 7

Validering av diagnossystemet

Diagnossystemet valideras genom att diagnostisera den riktiga motorns luftintagssystem.

7.1 Felsimulering

Fel simuleras genom att manipulera med spänningarna som A/D-kortet får från sensorerna. Två typer av fel simuleras, kortslutning och biasfel. Kortslutning i sensorn simuleras genom att kortsluta sensorns ingång på A/D-kortet. Ett biasfel betyder att en sensor ger ett konstant för högt eller för lågt värde. Biasfel simuleras genom att, på sensorns ingång på A/D-kortet, addera en spänning till det mätta värdet. De två typerna av fel simuleras för varje sensor som används vid diagnosen. När ett fel har uppstått har plottar gjorts av felvärden, dels av sensorernas värden och dels av residualernas värden.

Vid valideringen hade Residal 1 och Residual 4 tröskelvärdet 12 för alla fel. Residual 2 och Residual 3 hade tröskelvärdet 9 för alla fel. Dessa nivåer verkade vara rimliga för att inte generera falska fel. Buffertarna hade en radstorlek på 30 värden för att göra det möjligt att visa värden från det att felet inträffat i sensorn tills ett fel genereras av diagnossystemet. Arbetsområdet för motorn hålls klart inom det område som mapparna (se avsnitt 3.1) klarar av. Plottar av motorkörningarna finns i bilaga A.

7.2 Resultat

Resultatet av en kortslutning av luftmassflödessensorn visas i figurerna A.1 och A.2. Figur A.1 visar när felet inträffar samt alla sensorer som felet är beroende av. Figur A.2 visar när felet inträffar samt alla residualer som felet är beroende av. I figur A.1 syns tydligt när luftmassflödessensorns värde ändras. Figur A.2 visar att Residual 1, Residual 2 och Residual 4 hamnar utanför tröskelvärdena när felet har inträffat och att Residual 3 inte påverkas. Enligt tabell 4.1 stämmer detta med ett fel i luftmassflödet. I samma figur syns att Residual 1 och Residual 4 ligger nära tröskelvärdet innan felet uppstår. En höjning av tröskelvärdet kan därför vara nödvändig. Figurerna A.3 och A.4 visar på samma sätt när ett biasfel uppstår. Då läggs en spänning till på det mätta luftmassflödesvärdet vilket kan ses i figur A.3. Att residualerna reagerar på korrekt sätt kan ses i figur A.4. Figur A.3 visar att det är en tidsfördröjning mellan felets inträffande i sensorn och när det genereras i diagnossystemet. Det beror på att residualerna som ska reagera ligger långt ifrån sina tröskelvärden. Snabbare filter kan hjälpa i detta fall.

Fel på grund av kortslutning av trycksensorn visas i figurerna A.5 och A.6 och för biasfel i figurerna A.7 och A.8. Fel hos trottelvinkelsensorn på grund av kortslutning syns i figurerna A.9 och A.10 och för biasfel i figurerna A.11 och A.12.

Försöken visar att alla simulerade fel detekteras och isoleras korrekt vid det arbetsområde som motorn körs vid. Om ett fel i varvtalssensorn simuleras så genereras ingen felsignal. Det är korrekt eftersom detta fel inte diagnostiseras.

Något som kan ge problem är mapparna. Om ett insignalvärde till en mapp ligger utanför tillåtet intervall så returneras resultatet noll från mappen. Om motorn körs för nära mapparnas giltighetsområde så kan det leda till felaktig resultat. I avsnitt 3.4 visas residualgenereringen för det tidsdiskretiserade luftintagssystemet. Antag att motorn körs med ett tryck som ligger nära gränsen för mapparnas giltighetsområde. Då finns det risk att bägge mapparna som används returnerar noll. I så fall kommer r_1, r_2 och r_4 i ekvationerna att få höga värden. Däremot är det inte säkert att r_3 får ett högt värde. Då kan enligt tabell 3.2 ett felaktigt luftmassflödesfel genereras.

En lösning på detta problem är att använda modeller (t.ex.mappar) som täcker ett större område och som returnerar ett lämpligt värde om intervallet överskrids.

7.3 Prestanda och tillförlitlighet hos systemet

Den största bristen hos diagnossystemet är överföringen av plotvärden från realtidssystemet till GUI. Ofta, ibland varannan gång, så kommer inte alla värden fram till GUI. Det inträffar när motorn är igång och data från styrsystemet skickas på CAN-bussen till motorn. Om inte motorn körs så uppstår inte detta problem. Det tyder på någon konflikt mellan signalerna från diagnossystemet och signalerna från motorns styrsystem. Tiden har dock inte räckt till för att fastställa orsaken till detta.

Ett annat problem är att DDE-funktionerna som används vid överföringen av data på CAN-bussen slutar fungera efter ett tag. Då måste GUI startas om för att funktionerna ska fungera igen. Orsaken till problemet har dock inte fastställts.

Det har vid enstaka tillfällen hänt att realtidssystemet har slutat fungera. Det kan bero på att någon avallokering av minnesutrymme har glömts.

I övrigt har inga fel i diagnossystemet upptäckts. Kontroller har gjorts för att visa att realtidssystemets Managerprocess detekterar alla fel som inträffar enligt Diagnosprocessens beräkningar och att räknarna för felen stämmer. Det har också kontrollerats att alla fel som inträffar i realtidssystemet meddelas till GUI.

Kapitel 8

Slutsatser

I detta kapitel kommer först en sammanfattning av diagnossystemet. Sen beskrivs de resultat som uppnåtts med diagnossystemet och sist ges exempel på utvidgningar av arbetet.

8.1 Sammanfattning

Ett diagnossystem har utvecklats för modellbaserad diagnos i realtid. Diagnossystemet består av två delar implementerade på två olika datorer, ett realtidssystem och ett GUI. Realtidssystemet och GUI kommunicerar med varandra via en CAN-buss. I realtidssystemet finns två processer, kallade Diagnosprocess och Managerprocess.

Diagnosprocessen avläser sensordata från ett A/D-kort och med hjälp av dessa data beräknas felbeslut för diagnostiserade komponeneter. Felbesluten beräknas med residualgenerering och residualevaluering. Vid residualgenerering beräknas residualer utifrån sensorvärden och modeller. Vid residualevaluering tas felbeslut fram genom att jämföra residualerna med tröskelvärden. Sensorvärden, residualer och felbeslut sparas i buffertar.

Managerprocessen kontrollerar om något fel har inträffat och meddelar i så fall GUI. Den sköter också övrig kommunikation med GUI. Den håller reda på vilka fel som ska meddelas GUI, sparar undan värden om fel uppstår och skickar plotvärden till GUI.

GUI har konstruerats för att ge information till användaren om diagnosen som sker i realtidssystemet. I GUI väljs vilka delsystem som ska diagnostiseras. Felen som kan detekteras i dessa delsystem, visas i ett fönster och kan markeras så att meddelande fås om felet inträffar i

realtidssystemet. Vid start kan tröskelvärden och storleken på buffertarna i realtidsprogrammet väljas. Under diagnosen kan tröskelvärdena för residualerna ändras. Värden från realtidssystemet kan plottas, dels från den kontinuerliga diagnosen och dels värden som sparats undan då fel inträffat.

I realtidssystemet har diagnos av luftintagssystemet på en SAAB 2.3 liters motor implementerats. Komponenter som kan diagnostiseras är trottelvinkelsensor, luftmassflödessensor och sensor för trycket i insugsröret.

8.2 Resultat

Körningar på riktig motor visar att fel detekteras och isoleras korrekt om arbetsområdet för motorn är bra. Om motorn körs nära gränserna för implementerade modeller så kan felaktig detektering ske.

Managerprocessen fungerar korrekt. Den upptäcker och sparar undan fel vid rätt tidpunkt, den meddelar GUI korrekt. Det har vid enstaka tillfällen hänt att realtidssystemet har slutat fungera. Det kan bero på att någon avallokering av minnesutrymme inte utförs.

De flesta av GUIs funktioner fungerar bra. Ett problem är att GUI ofta inte får alla plotvärden som skickas från realtidssystemet. Dessutom slutar DDE-funktionerna som används vid CAN överföringen att fungera efter ett tag. Orsaken till dessa problem har inte hittats.

CAN-kommunikationen är ganska långsam vilket troligen beror på att funktionerna för CAN-kommunikationen som finns i realtidsskalet skickar CAN-meddelanden med en ganska låg frekvens.

Det finns ytterst få modellbaserade diagnossystem som har implementerats idag. Arbetet har visat att modellbaserad diagnos i realtid kan fungera bra. Förhoppningsvis kan arbetet ge ideer till hur modellbaserade diagnossystem kan utvecklas och användas i verkliga miljöer.

8.3 Utvidgningar

Den naturligaste utvidgningen av arbetet är kanske att ta reda på orsaken till problemen med CAN-överföringen och att DDE-funktionerna slutar fungera i GUI.

Att kunna välja fler parametrar i GUI vore också bra, t.ex. samplingstid och vilka portar på A/D-kortet som ska läsas av.

En annan tänkbar utvidgning av arbetet är att förbättra modellen som används vid residualgenereringen. Mapparna som används bör täcka ett större område och om en insignal till mappen ligger utanför tillåtet intervall så bör det tas om hand på något sätt.

Andra förbättringar kan vara att skicka meddelande till GUI dels i samma sampel som ett fel inträffar, dels när felvärden sparas. Då kommer ett fel att indikeras snabbare i GUI.

I realtidssystemet används en funktion för att hålla en buffert. Om det glöms bort kan bufferten användas ändå. En förbättring kan vara att införa riktig mutual exlusion i buffertarna.

Ett tillägg är att implementera diagnos av fler delsystem, t.ex diagnos av trottel. Då kan det verifieras att diagnossystemet fungerar även om fler delsystem diagnostiseras.

Diagnossystemet kan detektera och isolera fel. Det vore bra om det även gick att klassificera fel, t.ex. om det är ett biasfel, ett avbrott, om felet är intermittent eller stokastiskt.

En bättre hantering av olika fel som kan inträffa i koden är också en tänkbar utvidgning.

Litteraturförteckning

[1] *dSpace GmbH, URL: http//www.dspace.de.*

[2] *LabView, National Instruments Corporation, URL: http//www.natinst.com.*

[3] *Manual zum CAN-DDE-Treiber, CAN_DRV*, Maj 1994.

[4] The MathWorks, Inc., Natick, MA. USA. *MATLAB: Building a Graphical User Interface*, 1993.

[5] The MathWorks, Inc., Natick, MA. USA. *MATLAB: Building GUIs with MATLAB, Version 5*, 1996.

[6] The MathWorks, Inc., Natick, MA. USA. *MATLAB:Using MATLAB Version 5*, 1996.

[7] M.Bergman. Realtidssimulering av modellbaserad lamdareglering för bensinmotorer. Master's thesis LiTH-ISY-EX-1794, Department of Electrical Engineering, Linköping University, Linköping, Sweden, September 1997.

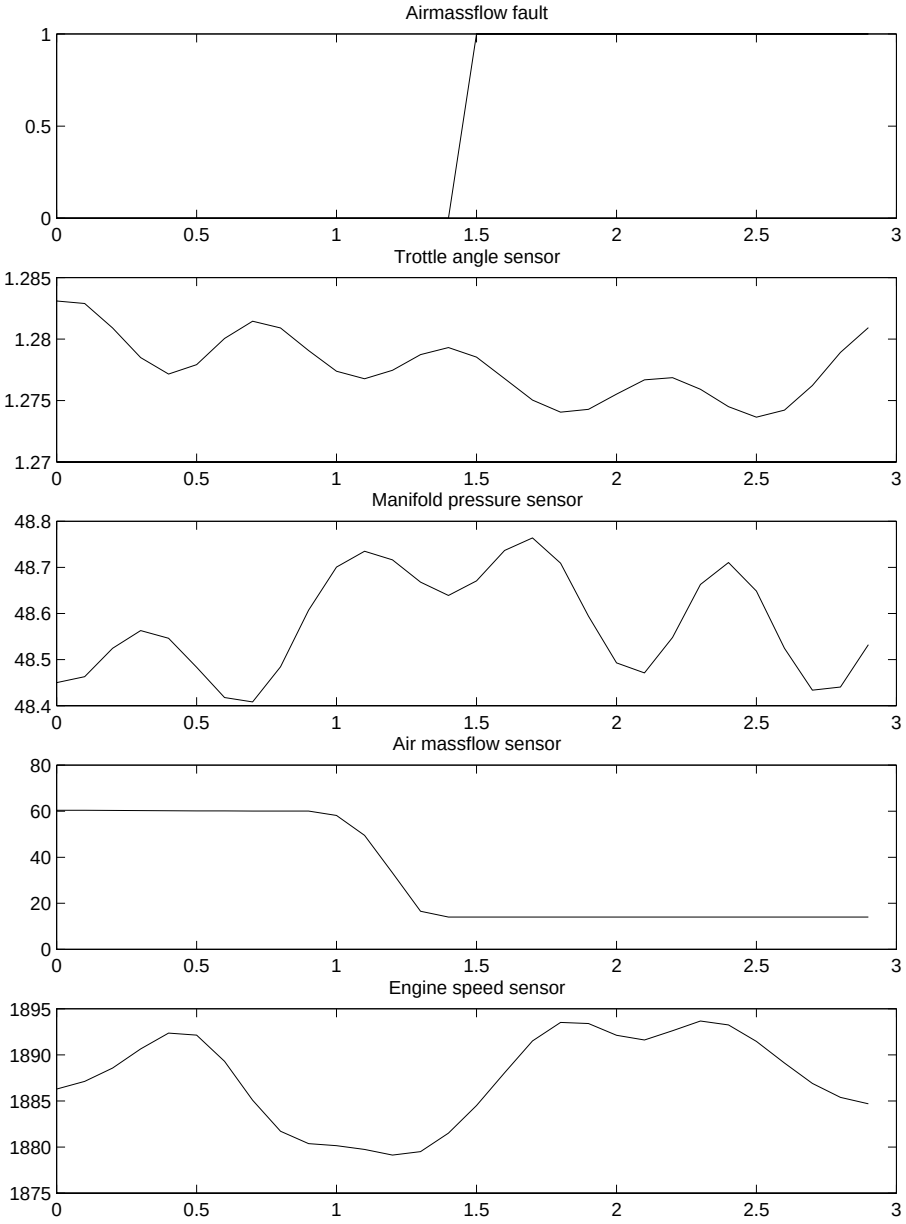
[8] M. Nyberg. *Model Based Diagnosis with Application to Automotive Engines.* Licentiate thesis LIU-TEK-LIC-1997:38, Department of Electrical Engineering, Linköping University, Linköping, Sweden, September 1997. URL: http://www.vehicular.isy.liu.se.

[9] S Edlund och T Henriksson. *User's manual and Technical Documentation of Vehicular System's Realtime System.* Vehicular Systems, ISY, Linköping University, S-581 83 Linköping, Oktober 1996.

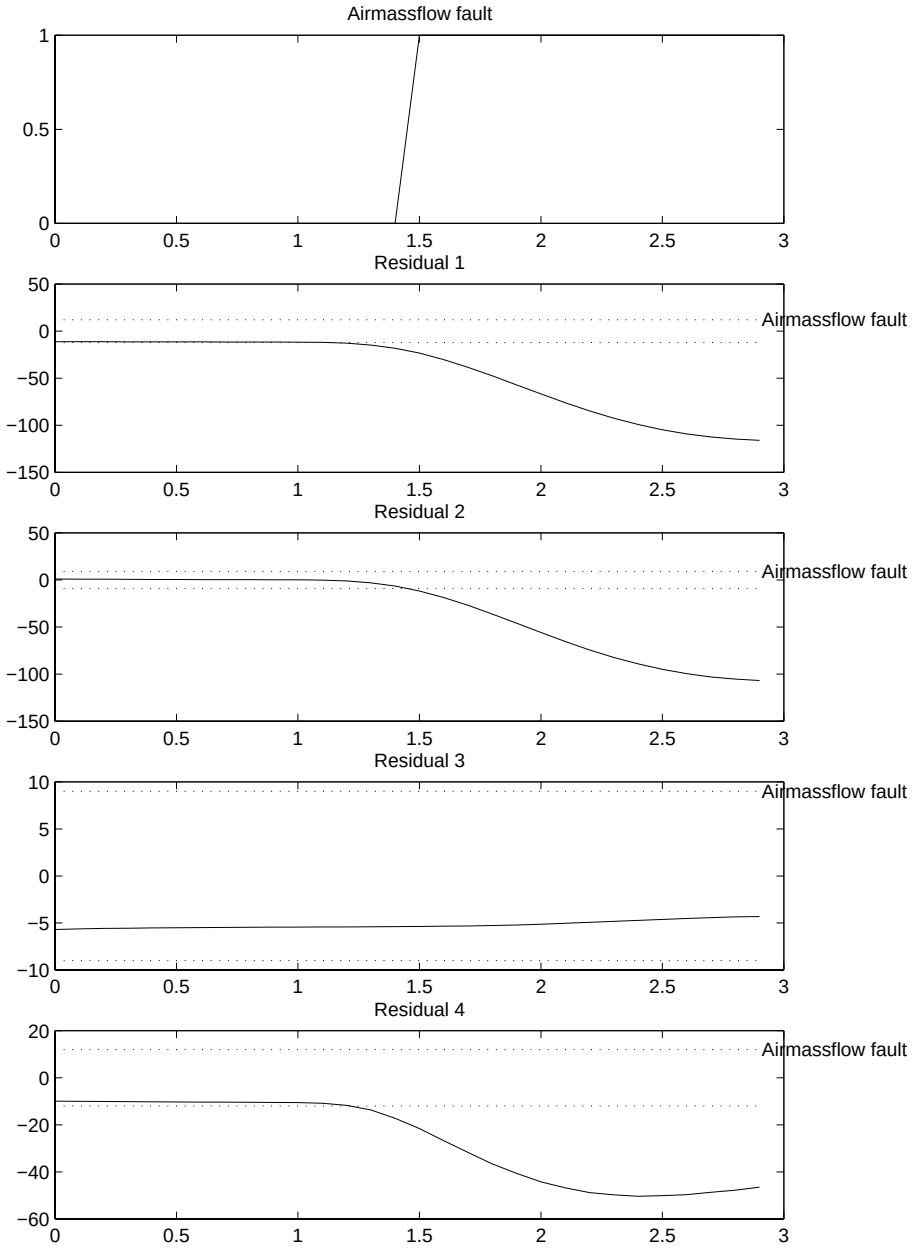
[10] On Time Marketing, Hamburg, Germany. *RTKernel 4.0 Real-Time Multitasking Kernel.*

[11] M. Pettersson. Description of an SI- engine testcell. Technical Report LiTH-ISY-R-1954, Department of Electrical Engineering, Linköping University, Linköping, Sweden.

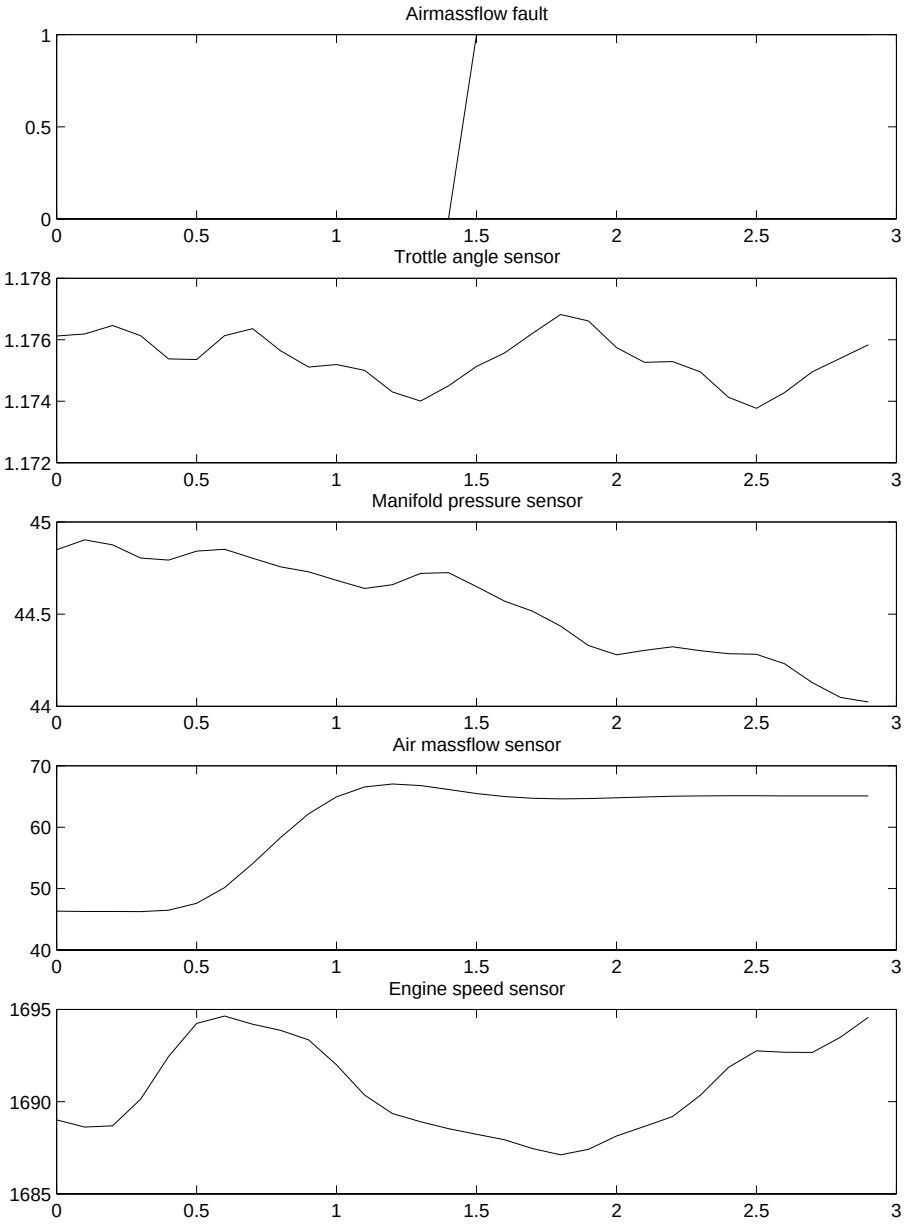
Bilaga A: Plottar av motorkörningar



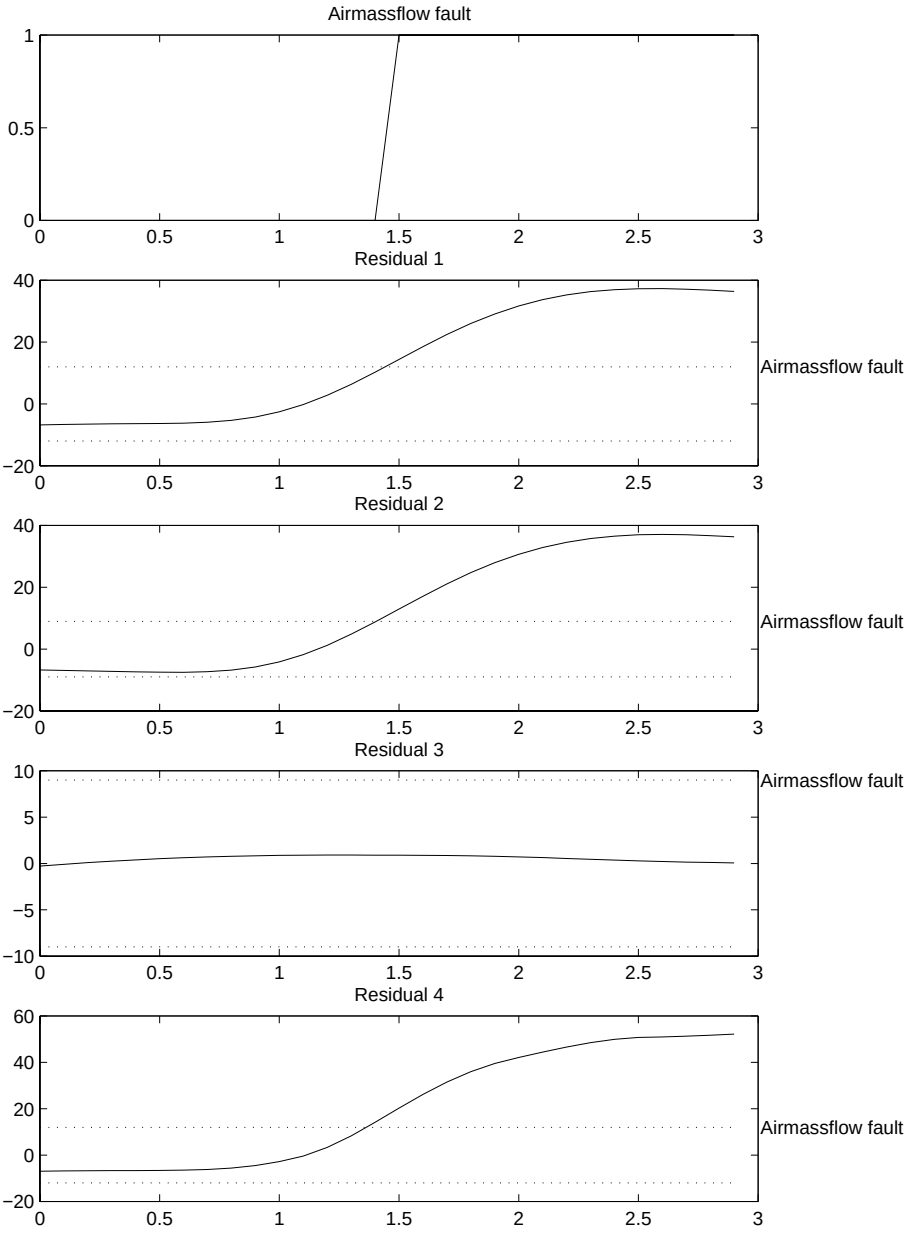
Figur A.1: Kortslutning av luftmassflödessensorn. Felbeslut och sensorvärden



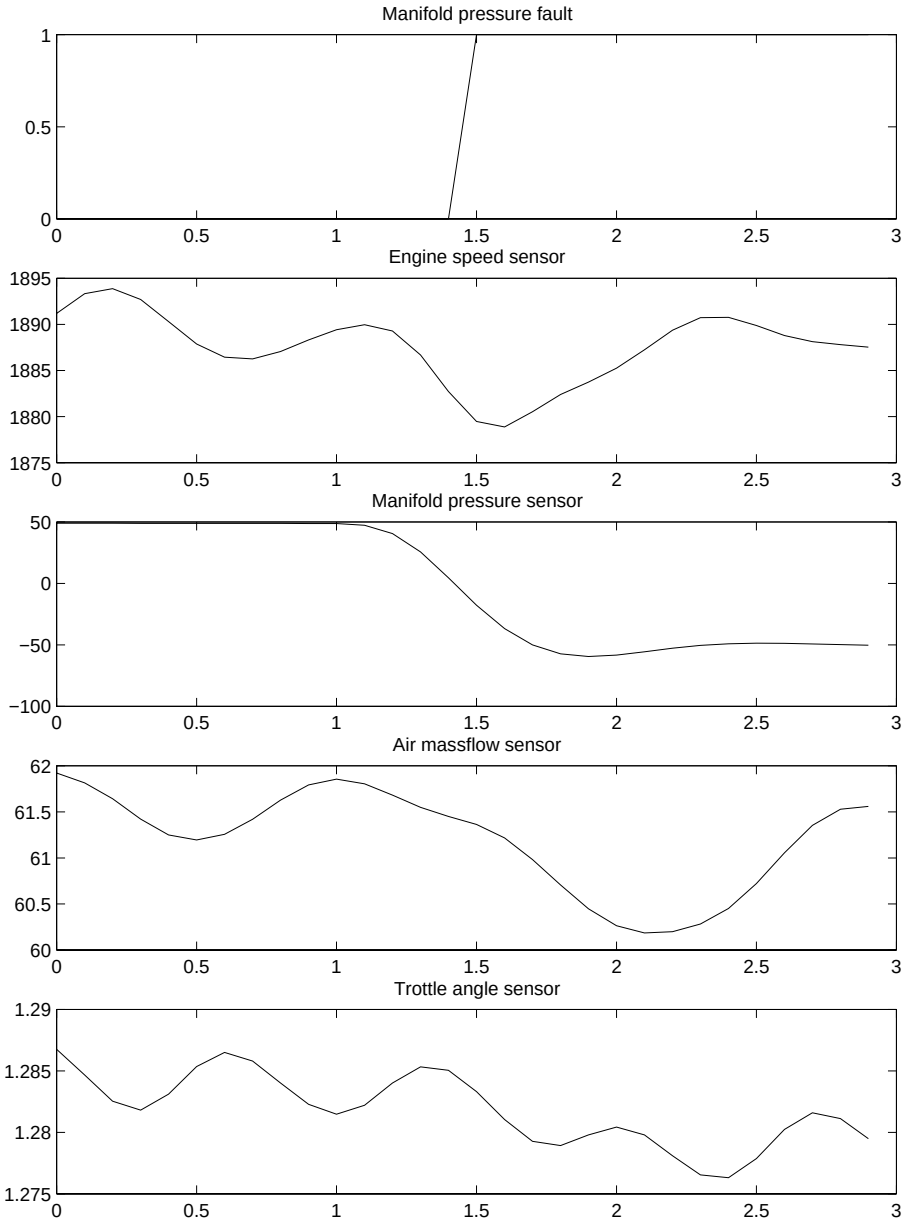
Figur A.2: Kortslutning av luftmassflödessensorn. Felbeslut och residualer



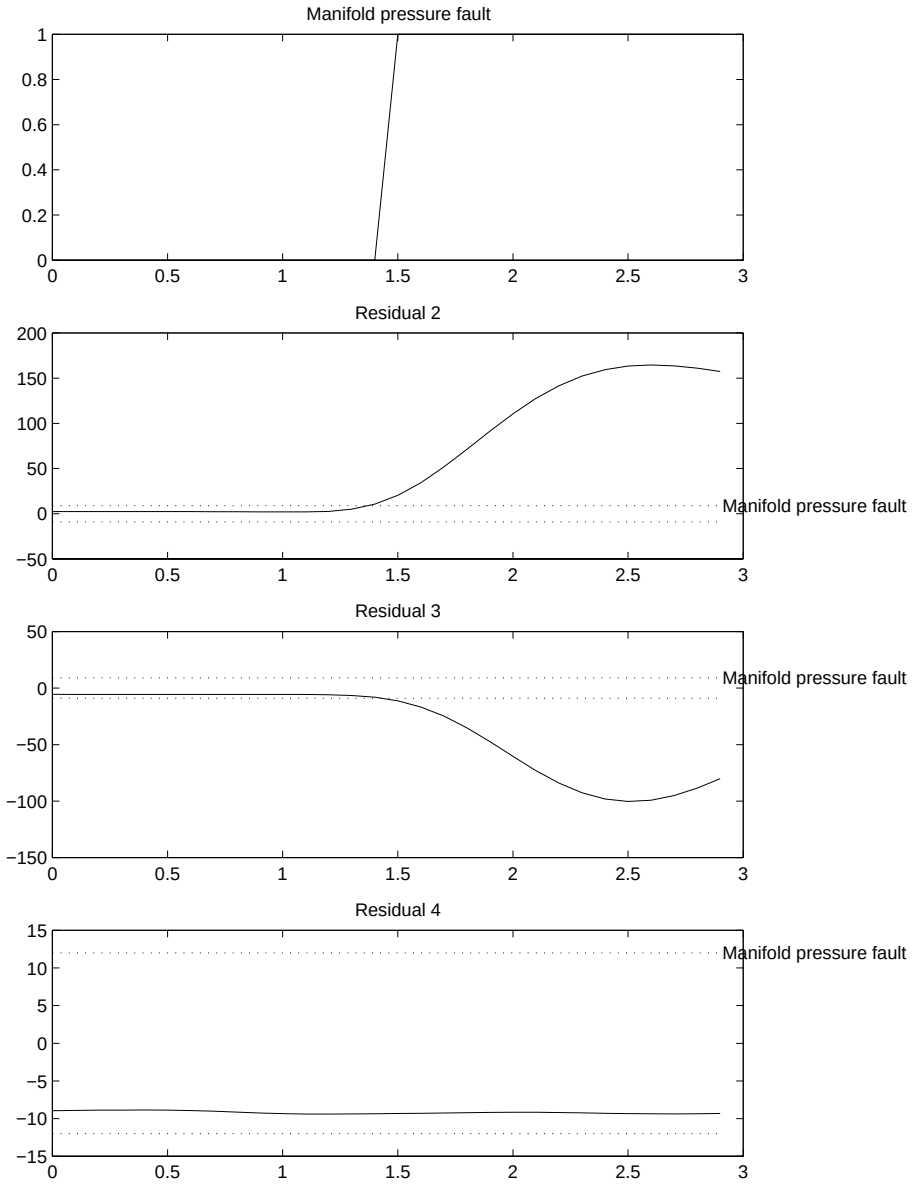
Figur A.3: Biasfel på luftmassflödessensorn. Felbeslut och sensorvärden



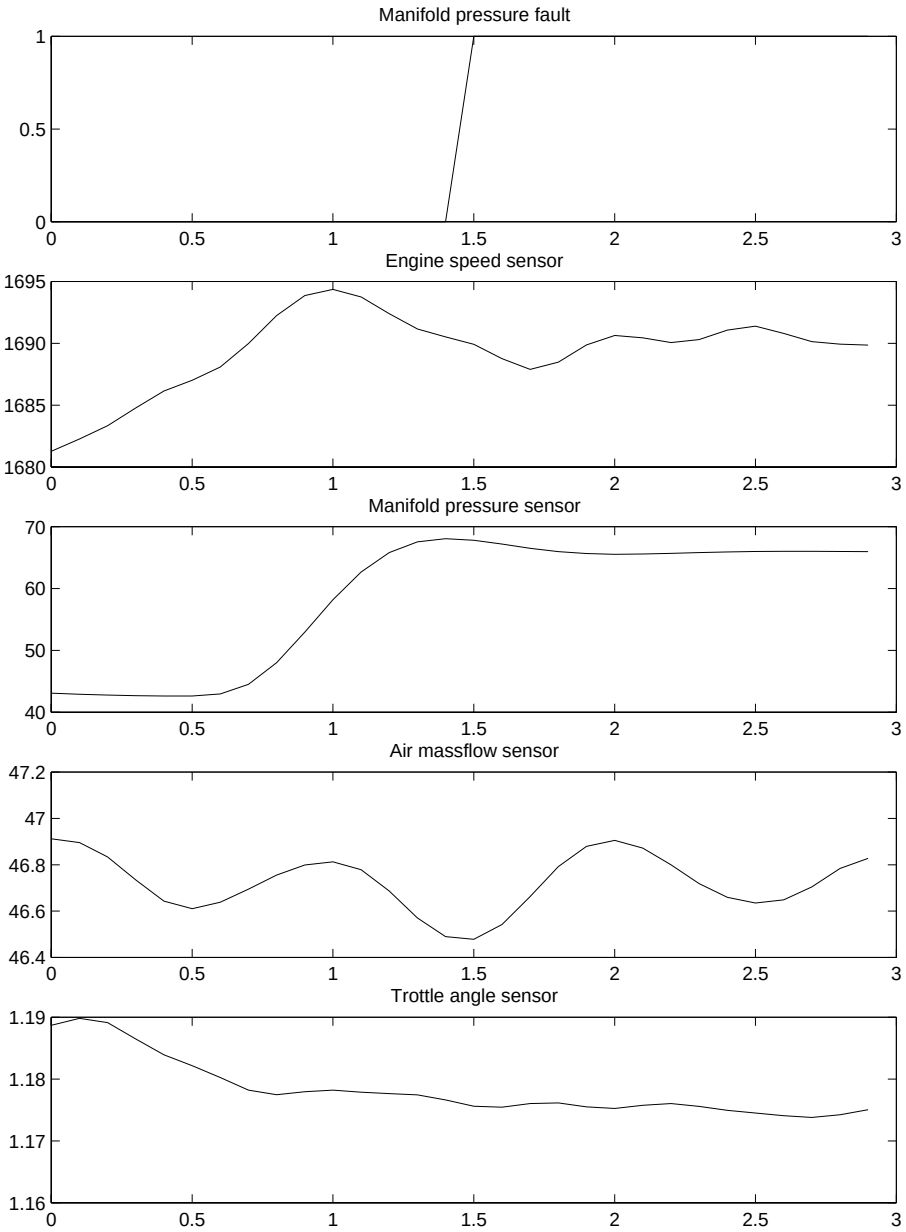
Figur A.4: Biasfel på luftmassflödessensorn. Felbeslut och residualer



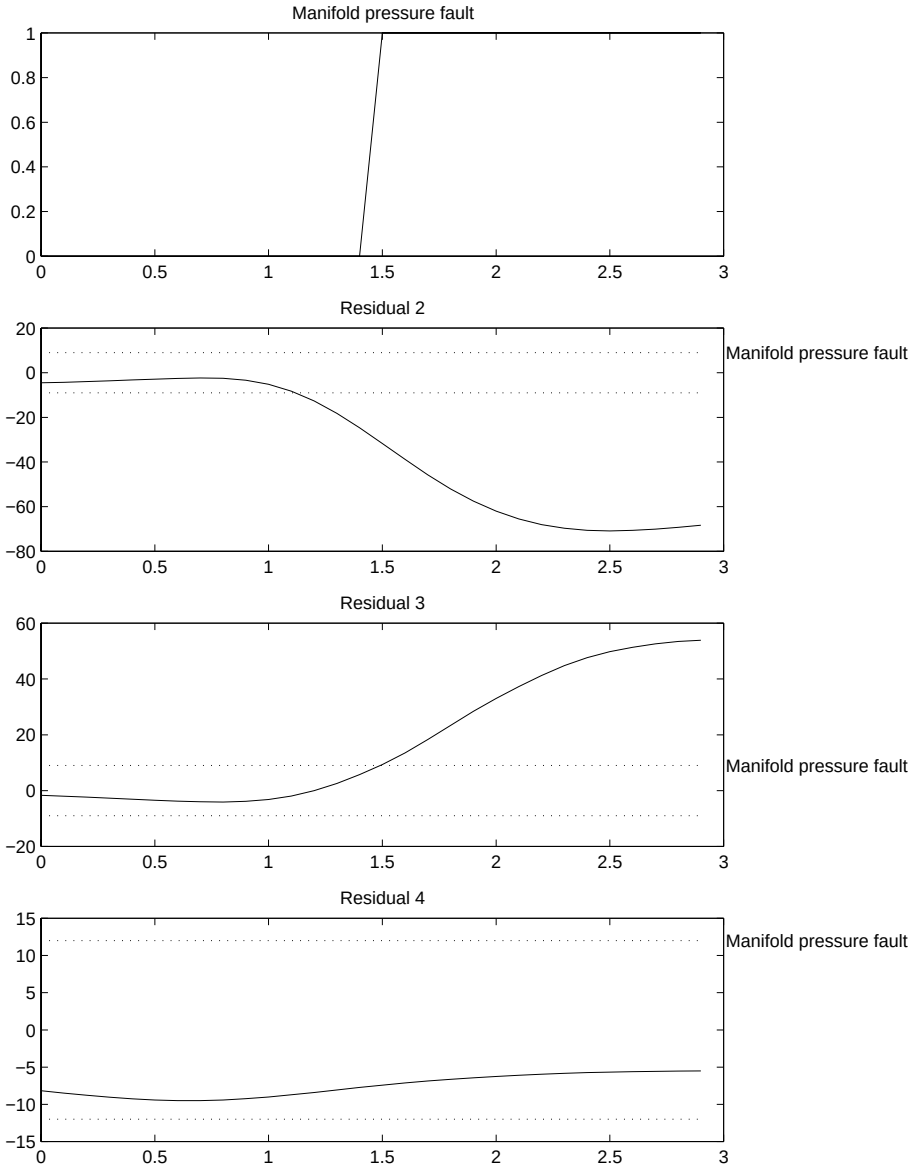
Figur A.5: Kortslutning av trycksensorn. Felbeslut och sensorvärden



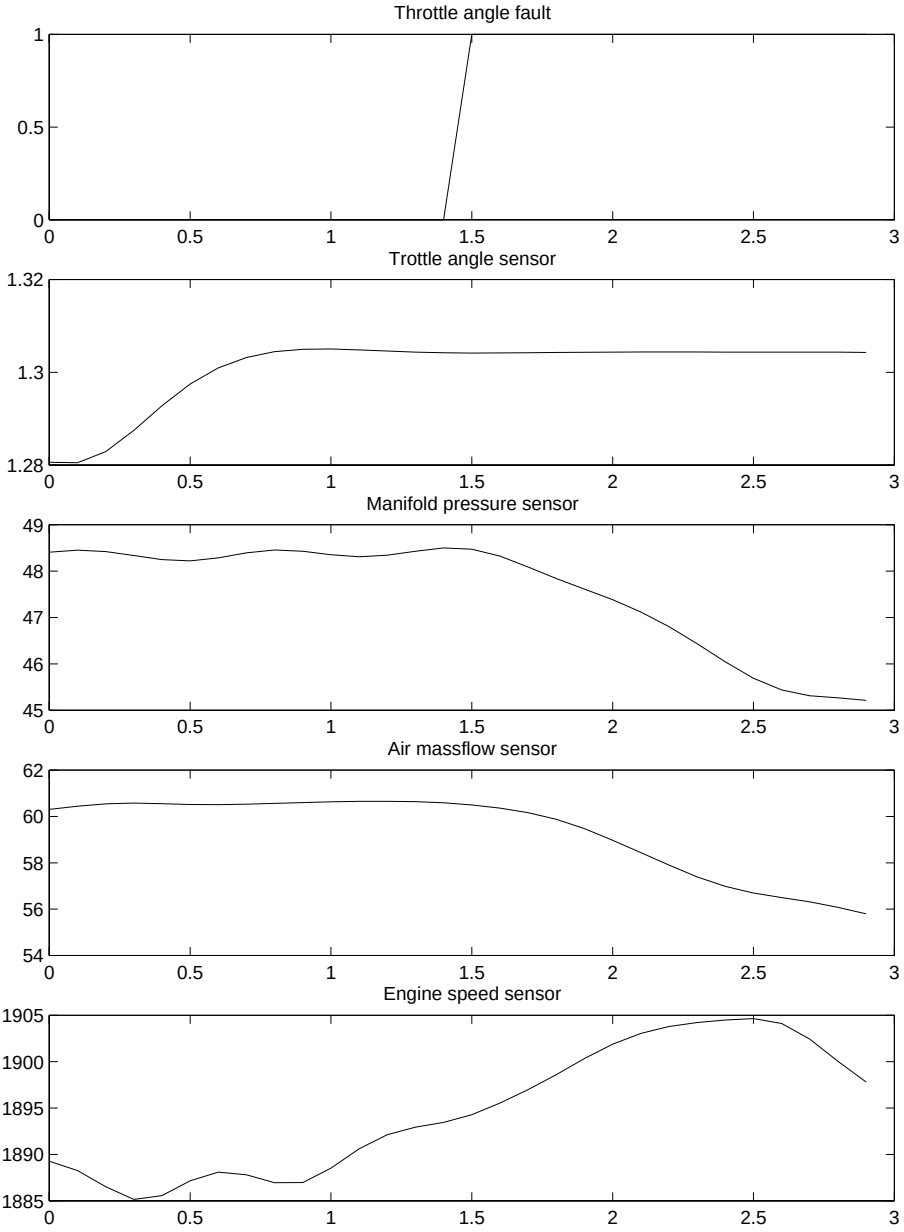
Figur A.6: Kortslutning av trycksensorn. Felbeslut och residualer



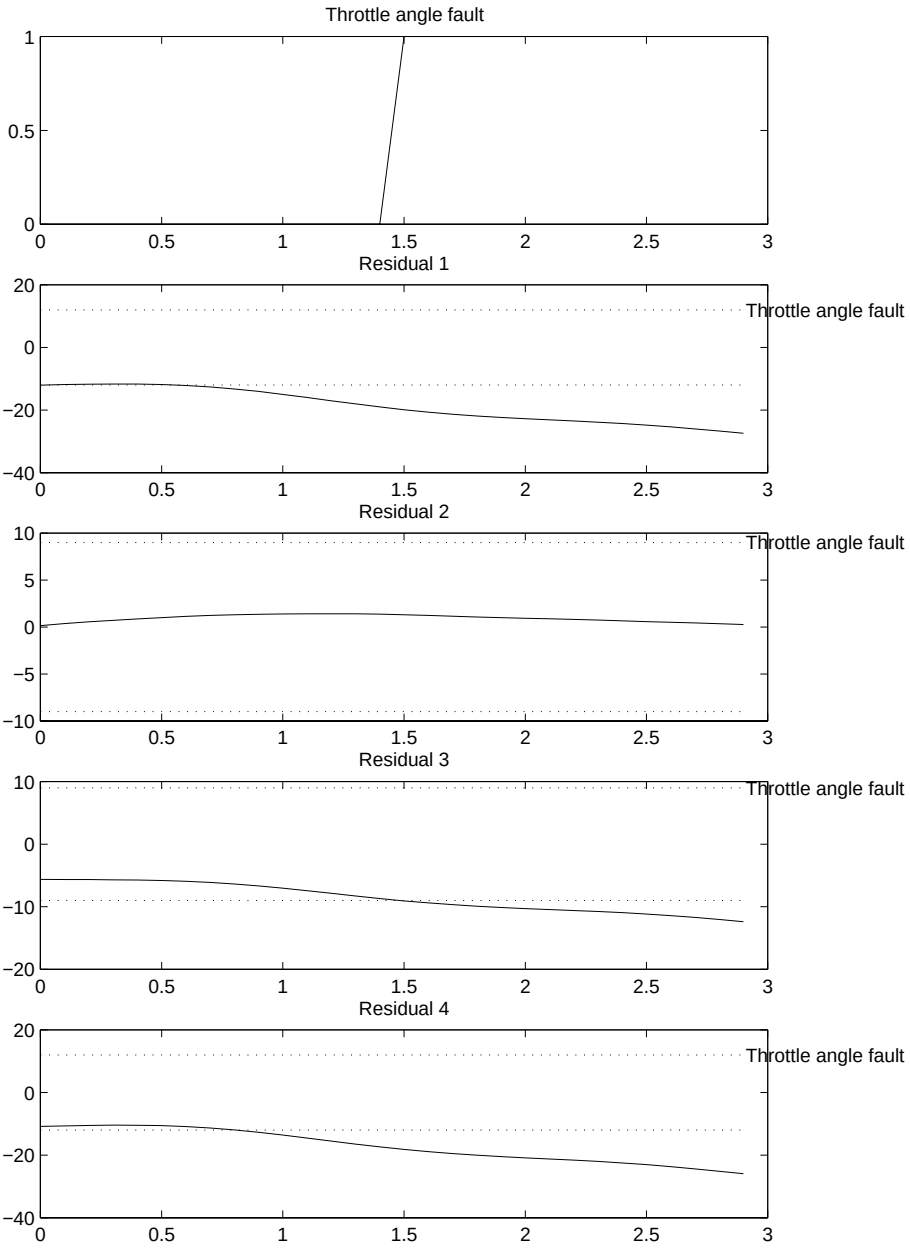
Figur A.7: Biasfel på trycksensorn. Felbeslut och sensorvärden



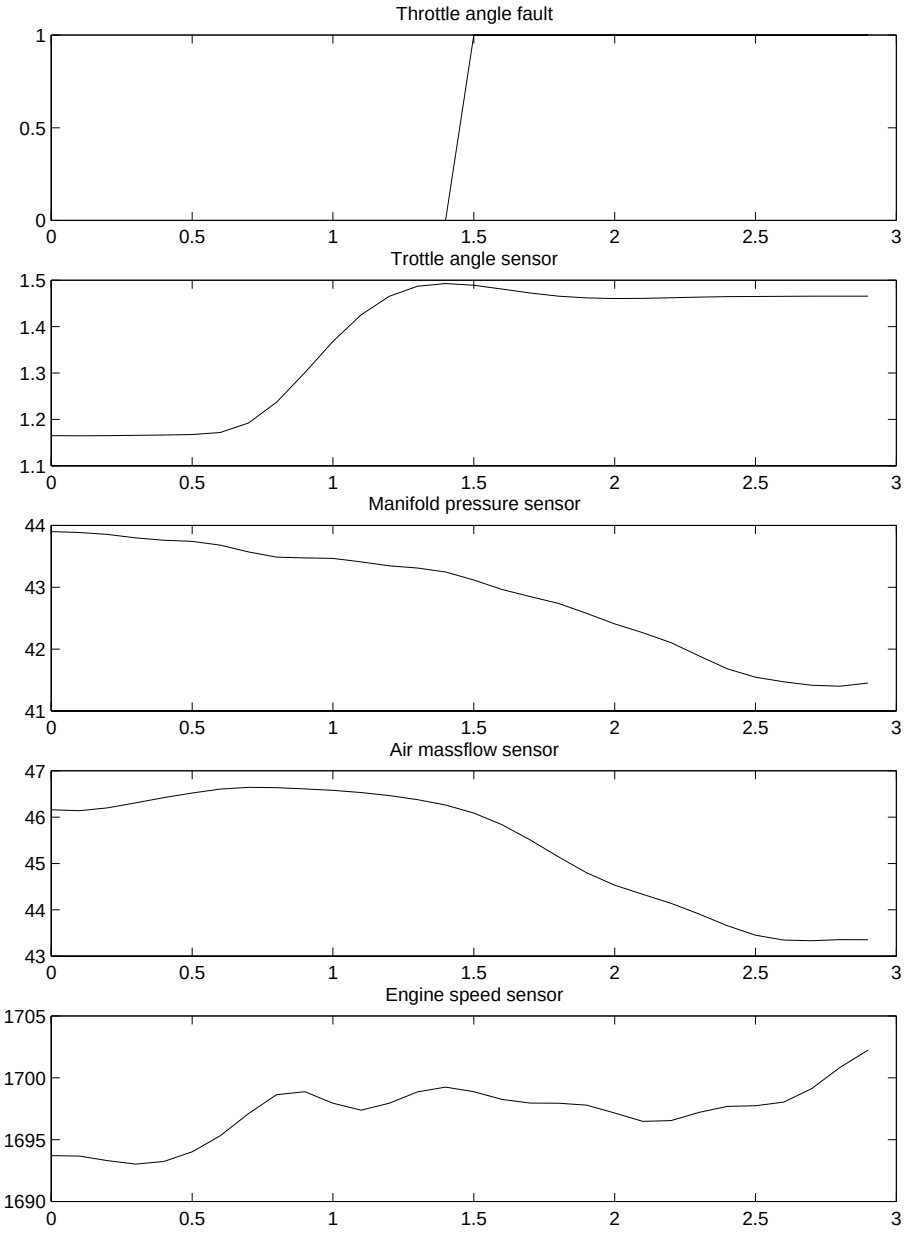
Figur A.8: Biasfel på trycksensorn. Felbeslut och residualer



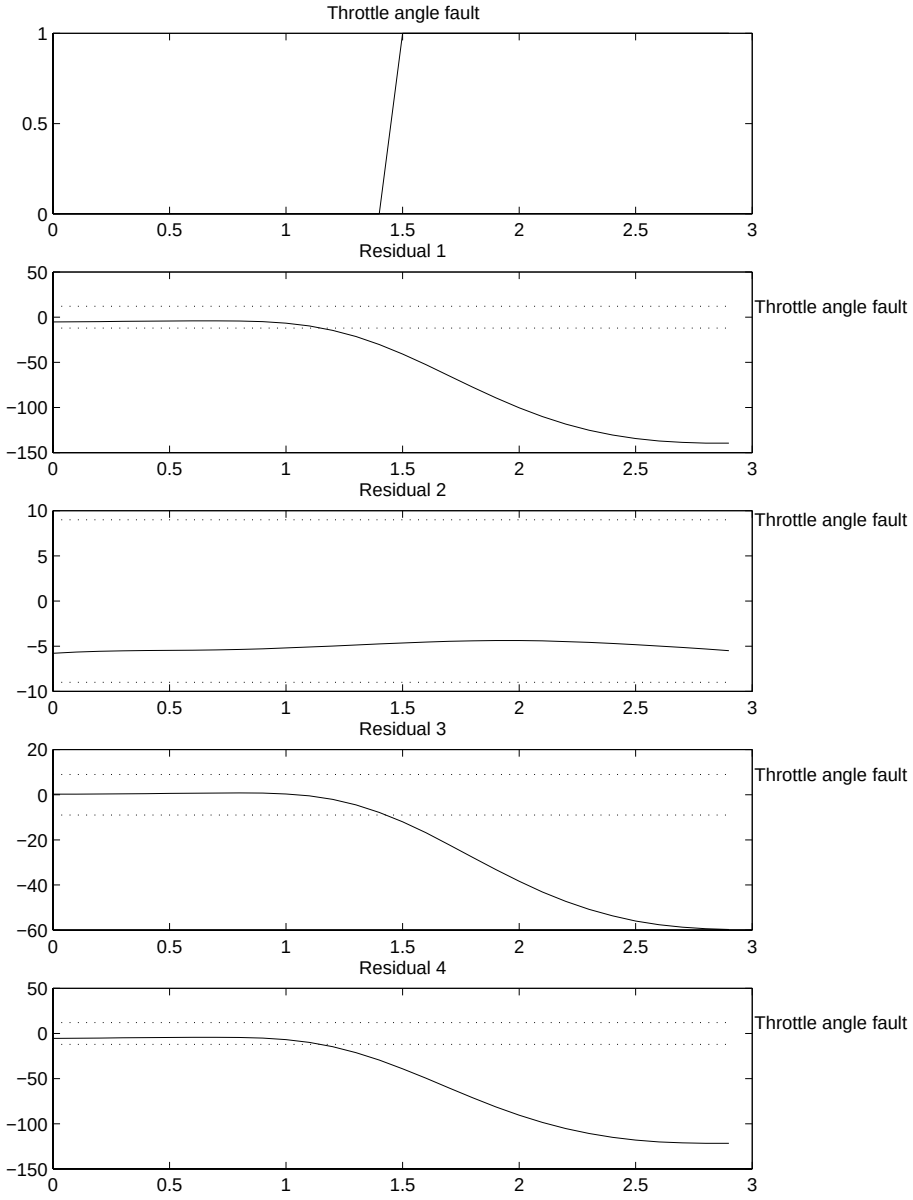
Figur A.9: Kortslutning av trottelvinkelsensorn. Felbeslut och sensorvärden



Figur A.10: Kortslutning av trottelvinkelsensorn. Felbeslut och residualer



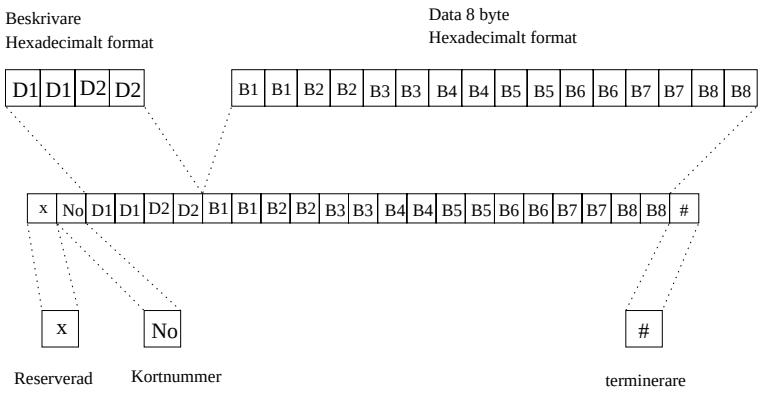
Figur A.11: Biasfel på trottelvinkelsensorn. Felbeslut och sensorvärden



Figur A.12: Biasfel på trottelvinkelsensorn. Felbeslut och residualer

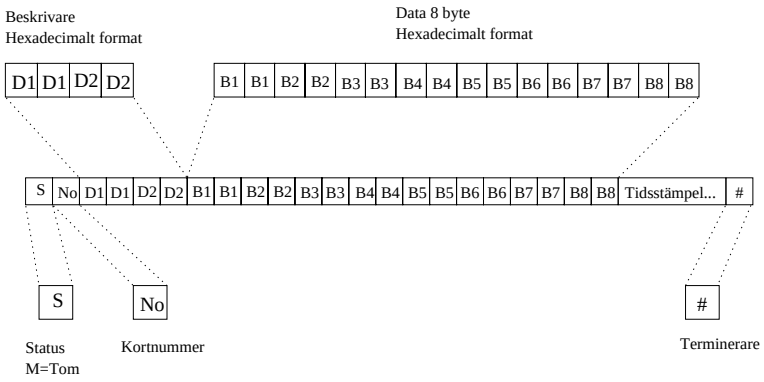
Bilaga B: CAN_DRV

CAN_DRV [3] är ett gränssnitt mellan en CAN-buss och en applikation som använder CAN-bussen. CAN_DRV är en DDE-server (se avsnitt 6.3) som transporterar data mellan CAN-bussen och applikationen, som är klient. CAN_DRV ställer ut data till ett kort. Dessa data går via CAN-bussen till ett annat kort som i sin tur avläses av ett annat gränssnitt. CAN_DRV tar emot data genom att läsa av kortet. CAN_DRV har olika brevlådor där meddelanden hamnar. Meddelanden som ska skickas hamnar i Transmit-Mailbox och data som tas emot hamnar i Recieve-Mailbox. En klient kan komma åt brevlådorna genom att skapa en konversation med servern CAN_DRV. Meddelanden som hamnar i brevlådorna ser i princip ut som i figur B.1 och B.2.



Figur B.1: Utseende på meddelanden som hamnar i Transmit-Mailbox.

Beskrivarna i figurerna består av ett identifieringsnummer och information om hur många byte av meddelandet som består av relevant information. Dessa värden anges av applikationen som skickar meddelandet. Beskrivarna och data är hexadecimalt kodade. Receive-Mailboxen har ett statustecken för att avgöra om det finns några meddelanden i den. Dessutom tidsstämplas meddelandena när dom kommer.
Det finns möjlighet att i en initieringsfil skapa *Channels* för meddelanden. I initieringsfilen anges identifieringsnummer för dessa meddelanden. En Channel är en kanal som går till en *Channel-Mailbox*. Om ett meddelande som hör till en Channel hamnar i



Figur B.2: Utseende på meddelanden som hamnar i Receive-Mailbox.

Receive-mailboxen så förs det vidare till Channel-mailboxen. En klient kan upprätta en *hot-link* till Channel-Mailboxen, vilket gör att meddelanden som hamnar i den, uppdateras automatiskt till klienten.

Bilaga C: DDE i Matlab

I Matlab finns ett antal funktioner för DDE (avsnitt 6.3). Dessa används i diagnossystemet när datautbyte ska ske mellan GUI i Matlab och realtidssystemet via CAN-bussen. Då används Matlab som klient och applikationen CAN_DRV som beskrivs i bilaga B är server. Nedan följer exempel på DDE-funktioner som har använts för diagnossystemet:

handle = ddeinit('CAN_DRV', 'TR_MAILBOX') Skapar en konversation med CAN_DRV för ta emot eller skicka data. Funktionen returnerar en *handle* som identifierar konversationen.

msg = ddereq(handle, 'Recieve') Hämtar ett meddelande från Receive-Mailboxen.

ddepoke(handle, 'Transmit', msg) Skickar *msg* till Transmit-Mailboxen som ställer ut den på CAN-bussen.

handle = ddeinit('CAN_DRV', 'Channels') Skapar en konversation med CAN_DRV till Channel-Mailboxen.

ddeadv(handle, 'id', 'str', 'msg') Upprättar en *hot-link* för identifieringsnummret *'id'*. När ett meddelande med detta identifieringsnummer kommer till CAN_DRV så uppdateras variabeln *'msg'* med meddelandet. Dessutom så evalueras strängen *'str'* av Matlabs evalfunktion.

ddeterm(handle) Avslutar konversationen

Alla meddelanden som skickas eller tas emot av GUI är hexadecimalt kodade eftersom CAN_DRV kräver det (se bilaga B).