NTNU
Det skapende universitet

Tilstandsovervåking av thrustersystemer for skip

Fjerndiagnostikk ved analyse og satellittoverføring av prosessdata

Espen Løkseth

Master i teknisk kybernetikk (2-årig)
Innlevert: juni 2013
Hovedveileder: Tor Arne Johansen, ITK

Norges teknisk-naturvitenskapelige universitet
Institutt for teknisk kybernetikk

Sammendrag

Den teknologiske utviklingen de siste tiårene har gjort automatisk tilstandsovervåking av mekaniske systemer stadig vanligere, samtidig som konkurranse i markedet hever kravene til pålitelighet og langsiktig stabilitet. Derfor er det viktig å nøyaktig kunne tidsfeste neste servicestopp, for å unngå systemhavari. Uventet nedetid kan representere en signifikant økonomisk belastning.

Marine thrustere, eller sidepropeller, brukes til manøvrering av skip. Brunvoll AS designer, produserer og leverer komplette thrustersystemer internasjonalt. I tett samarbeid med bedriften er et funksjonelt tilstandsovervåkningssystem utviklet, bygd og demonstrert i praksis.

Tilstandsovervåkningssystemet består av en industriell datamaskin og en dedikert satellittkommunikasjonsenhet, samt programvare for prosessering, analysering, komprimering, sending, mottak og dekomprimering av data. Prosesserings- og analysemetoder er programmert som moduler, eller *blokker*, som enkelt kan settes sammen ved hjelp av en tekstbasert konfigurasjonsfil. Systemet er derfor meget fleksibelt; det kan tilpasses alle thrustertyper levert av Brunvoll AS. To sett med blokker er implementert: prosesserings- og analyseblokker. Prosesseringsblokkene kan filtrere, derivere og beregne effektiv- og absoluttverdier. Analyseblokkene kan generere histogrammer, utføre FFT-analyser og beregne statistiske parametre. Nye blokker kan relativt enkelt legges til. I konfigurasjonsfilen spesifiseres hvilke prosessvariable som skal behandles og hvor ofte analyseresultatene skal sendes til land. Aktuelle prosessvariable er f.eks. thrusterens pitchvinkel eller motorens turtall. Et mottaksprogram dekoder og lagrer data i en database på land.

Praktiske tester av tilstandsovervåkningssystemet er utført ved å prosessere og analysere eksempeldata, både fra reelle målinger og fra en thrustersimulator. Resultatene viser at alle delene av systemet fungerer slik de skal, selv om FFT-algoritmen som ble implementert ikke er optimal. Komprimering før sending av data er gjort ved nedskalering til mindre datatyper. Påvirkningen nedskaleringen har på overført data er analysert og drøftet i rapporten. Beregninger viser at avvikene som oppstår er så små at de kan neglisjeres i de aller fleste tilfeller.

En positiv ringvirkning av å ha datainnsamling fra mange ulike skip og thrustertyper, er at det over tid vil bygge seg opp en stor samling av analyseresultater. Disse vil danne et sterkt statistisk grunnlag ved videreutvikling av thrustersystemene. Et eksempel på dette kan være å optimalisere thrusterdesignet slik at vibrasjoner rundt de mest brukte turtallsområdene dempes.

Abstract

Automated condition monitoring of mechanical machinery has become increasingly popular, due to the technological development over the past few decades. Condition monitoring uses process data to estimate when the next service stop is required. Thus, the number of unexpected system failures is reduced.

Through this master thesis a functional condition monitoring system for marine thrusters has been developed, in close cooperation with Brunvoll AS. Brunvoll AS designs, manufactures, and delivers complete thruster systems worldwide.

The condition monitoring system consists of an industrial computer, a satellite communication unit, and necessary software. The main software component acquires, processes, and analyzes data from the thruster system. It is composed of two kinds of modules, one for signal processing, and the other for signal analysis. The number and order of the modules are selectable through a text-based configuration file, which also determines the signals to acquire. This makes the system highly flexible. Developed processing modules can filter, differentiate, and compute RMS values. From the processed signals, the analyzing modules extract various information, which includes histograms, Fourier analysis, and statistical parameters. New modules can easily be added if necessary.

Another software component periodically transmits analysis results to the receiving part onshore through the satellite communication unit. Before transmitting, the results are scaled to smaller data types. By doing this, the amount of data sent over the limited satellite link is minimized. Errors introduced by the scaling are examined in the report, and found to be neglectable for the typical use case. At the receiving end, a third software component is developed to unpack and store data in a database.

Through a series of tests, the complete system's performance is characterized. Data from real measurements and simulators are fed through the system. Results show that most of the implemented modules work adequately. One exception is the Fourier analysis algorithm which generates some noise in the resulting spectrum.

Performing large-scale data acquisition and analysis from different types of ships and thrusters lead to a comprehensive data material. This can form a statistically significant basis for further development of the thruster systems. An example is to reduce vibration levels at the most used thruster rotational speeds.

Forord

Denne masteroppgaven er utført ved Institutt for teknisk kybernetikk hos Norges teknisk-naturvitenskapelige universitet (NTNU), i tett samarbeid med thrusterprodusenten Brunvoll AS. I mitt studium har jeg valgt hovedprofilen «Navigasjon og fartøystyring» som blant annet omfatter metoder for styring av skip, modellering og simulering av skip i bevegelse i seks frihetsgrader, tilstandsestimatorer basert på satelittnavigasjonssystemer, gyroer og akselerometre. Oppgaven avslutter det toårige løpet til «Master i teknologi / sivilingeniør» for studenter som har bachelorgrad fra før.

Brunvoll AS brakte fram problemstillingen om tilstandsovervåking av thrustersystemer vha. målinger, analyse og komprimering ombord på skip, overføring via satellitt og presentasjon på landsiden. Oppgaveteksten finnes i vedlegg A bak i rapporten.

Jeg takke Mads Lønsethagen, Ole Magnus Hjellset og Cato Orset ved utviklingsavdelingen hos Brunvoll AS for diskusjoner, tips og god veiledning ved gjennomføringen av prosjektet. Ved NTNU vil jeg takke min veileder professor Tor Arne Johansen for god hjelp underveis i prosjektet.

Denne rapporten skal ikke offentliggjøres før tre år etter innleveringsfrist, fordi den kan inneholde forretningsmessige sensitive opplysninger. Bestemmelsen om utsatt offentliggjøring er i tråd med NTNUs standardavtale om utføring av masteroppgave i samarbeid med ekstern bedrift.

Kjennskap til programmering (C++), signalbehandling, signalanalyse, diskret matematikk (numeriske metoder og transferfunksjoner) og algoritmeanalyse kreves for å få fullt utbytte av denne rapporten.

Jendem,

Dato	Espen Løkseth

Nomenklaturliste

ACID	–	*Atomic, Consistent, Isolated, and Durable* – Egenskaper ved et databasesystem som sikrer at en transaksjon blir utført sikkert og pålitelig. Se 5.1 for mer.
BIBO	–	*Bounded-Input Bounded-Output* – et diskret system er BIBO-stabilt hvis et begrenset inngangssignal x gir et begrenset utgangssignal y: Hvis $\|x(k)\| \leq B \ \forall \ k$ fins det en A slik at $\|y(k)\| \leq A \ \forall \ k$.
CBM	–	*Condition-Based Maintenance* – vedlikeholdsstrategi basert på tilstandsovervåking.
cRIO	–	*Compact Reconfigurable I/O* – datainnsamlingsenhet fra National Instruments.
DFT	–	*Discrete Fourier Transform* – metode brukt for å beregne frekvensspekteret til et diskret signal.
DP	–	*Dynamisk posisjonering* – benyttes til å holde skip i ro (eller kontrollerte forflytninger) uten bruk av anker.
DTAC	–	*Device Tracking and Control* – enhet for dataoverføring via Iridium, med GPS-posisjonering. Designet og produsert av Powex AS.
FFT	–	*Fast Fourier Transform* – algoritme(r) for effektiv beregning av den diskrete fouriertransformasjonen, se DFT.
FIR	–	*Finite Impulse Response* – et dynamisk system som har en endelig impulsrespons, oftest brukt som filter. Se også IIR.
FPGA	–	*Field-programmable Gate Array* – brikke som består av mange logiske blokker som kobles sammen vha. programmering.
HPU	–	*Hydraulic Power Unit* – hydraulikkenhet som leverer kraft til endring av pitchvinkel, asimutvinkel og til heving/senking av thrustere.
IEEE 754	–	*IEEE Standard for Floating-Point Arithmetic* – standard som definerer flyttallsdatatyper og tilhørende aritmetiske operasjoner.
IIR	–	*Infinite Impulse Response* – et dynamisk system som kan ha en uendelig impulsrespons, oftest brukt som filter. Se også FIR.

`int`	–	*Integer* – 32 biters datatype i C++. Positivt eller negativt tall med heltallsdel på 31 bit.
IPC	–	*Industriell PC* – datamaskin beregnet for industriell bruk.
ISO	–	*International Organization for Standardization* – en internasjonal standardiseringsorganisasjon.
KiB	–	*Kibibyte* – binærprefiks vedtatt av IEC i 1998. Tilsvarer 1024 byte, ofte feilaktig kalt kilobyte (som tilsvarer 1000 byte).
MIT	–	*Massachusetts Institute of Technology* – verdensledende universitet på forskning innen teknologi og vitenskap.
NAS	–	*National Aerospace standard* – amerikansk standard for forurensing i fluider.
NMEA	–	*National Marine Electronics Association* – kommunikasjonsstandard som definerer et elektrisk grensesnitt og en dataprotokoll.
PLS	–	*Programmerbar logisk styring* – programmerbar elektronisk enhet brukt til automatisering.
PSD	–	*Power Spectral Density* – Effektspektrum. Ligner på DFT, men angir effektfordelingen til et signal.
PSU	–	*Power Supply Unit* – strømforsyning.
RDT	–	*Rim Driven Thruster* – en akselløs thruster. Består av en elektromotor der propellen er rotoren, og statoren ligger som en ring rundt.
RMS	–	*Root Mean Square* – det kvadratiske gjennomsnittet eller effektivverdien.
RS-232	–	Seriell kommunikasjonsport.
TSM	–	*Thruster Service Management* – system som Brunvoll AS bruker for registrere service-informasjon på sine thrustere.
`uint`	–	*Unsigned Integer* – 32 biters datatype i C++. Positivt heltall.
XML	–	*Extensible Markup Language* – språk som brukes til strukturering av data. Tekstformat leselig for både mennesker og maskiner.
XSD	–	*XML Schema Definition* – definisjon på et XML-dokument. Inneholder regler for elementer, attributter, datatyper osv.

Innholdsfortegnelse

Figurer

Tabeller

Elektroniske vedlegg

Hovedprogrammet

`Mainprogram/`

Hovedprogrammet som prosesserer og analyserer data ombord. Her ligger alle prosesserings- og analyseklasser, som også blir brukt av senderprogrammet.

`xml/mainschema/mainschema.xsd`

XSD-spesifikasjon for oppsett av XML-filer til hovedprogrammet.

`xml/mainschema/main_example.xml`

Eksempel på XML-konfigurasjon for hovedprogrammet.

Senderprogrammet

`Txprogram/`

Program for pakking og sending av analyseresultater fra skip til land. Benytter mange av klassene i hovedprogramkatalogen.

Mottaksprogrammet

`Rxprogram/`

Program for mottak og utpakking av data på landsiden. Dekoder data og lagrer resultatet på strukturert form i en database.

`xml/rxschema/rxschema.xsd`

XSD-spesifikasjon for oppsett av XML-filer til mottaksprogrammet.

`xml/rxschema/rx_example.xml`

Eksempel på XML-konfigurasjon for mottaksprogrammet.

Kapittel 1

Innledning

Tilstandsovervåking av mekaniske systemer er et verktøy som har blitt stadig mer utbredt de siste årene grunnet den teknologiske utviklingen, kostnadsfall og høyere krav til stabilitet og oppetid. Det blir vanligere å utstyre mekaniske systemer med mange ulike sensorer med tilhørende signalprosessering for å kunne overvåke systemenes tilstand [12, 17].

Før 1950 var preventivt vedlikehold lite utbredt, det var vanlig at maskineri gikk helt til det havarerte og dermed *måtte* repareres. På 50-tallet ble det vanligere å utføre periodiske inspeksjoner og vedlikehold, selv om maskineriet tilsynelatende var i orden. For å øke lønnsomheten, var det viktig å velge optimale serviceintervaller, noe det er forsket en god del på. Likevel vil et slikt vedlikeholdsprogram skape unødvendige servicestopp og ekstraarbeid, som kunne ha vært unngått. En annen ulempe er at de mest overraskende og katastrofale feilene sjeldent kan unngås, for å få til det måtte inspeksjoner uføres mye oftere. [12]

Løsningen på dette er *Condition-Based Maintenance* (CBM), det vil si at tilstandsovervåkningsdata brukes til å styre serviceintervallene. Dette kan sammenlignes med en tilnærmet kontinuerlig inspeksjon av systemet, som fører til at også de bråeste og mest katastrofale feilene oftere kan forutsees. CBM økte gradvis i popularitet fra det oppstod på 50-tallet, men det var først utover 1980- og 90-årene at det virkelig tok av, i takt med utviklingen av stadig kraftigere datamaskiner og bedre sensorer [35].

Jardine, Lin og Banjevic [16] deler opp tilstandsbasert vedlikehold i tre steg:

1. Datainnsamling, herunder valg av parametre som gir mest mulig og relevant informasjon om tilstanden til systemet.
2. Prosessering, analyse og presentasjon for bedre å kunne håndtere og tolke data fra første punkt.
3. Vurdering av resultatene og enten anslå når neste service trengs (prognose) eller avgjøre om systemet allerede har feilet og hvordan (diagnose).

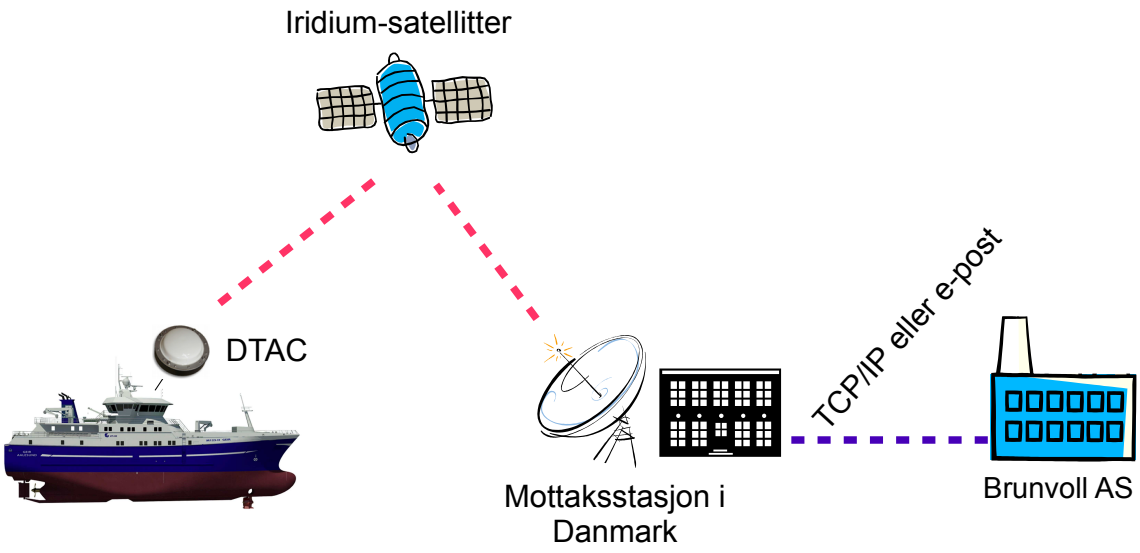
Hovedfokuset i denne oppgaven er på de to første punktene. Valg av parametre, prosesseringsalgoritmer, analyser og presentasjon av data blir beskrevet og drøftet i rapporten. Når det gjelder det siste punktet, vil det alltid være behov for menneskelige vurderinger og avgjørelser basert på erfaring, slik at prognosene blir mest mulig nøyaktige. Erfaringen kan skaffes ved å bruke innsamlet data over lengre tid fra mange ulike systemer, og på den måten danne et stort statistisk grunnlag.

En annen viktig positiv ringvirkning av storskala datainnsamling er å kunne kartlegge bruksmønster og feilsituasjoner, og ta dette i betraktning ved videre utvikling av produktet.

Gjennom denne masteroppgaven er det thrustersystemer fra Brunvoll AS det skal fokuseres på. Thrustersystemene er nærmere beskrevet i neste kapittel. Hovedmålet med prosjektet er å designe og programmere et komplett tilstandsovervåkningssystem med satellittoverføring fra skip til land. Splittet i mindre delmål får vi følgende liste:

- Valg av fornuftige parametre for å få mest mulig informasjon ut av systemet. Tilsvarer første punkt i CBM som beskrevet tidligere.
- Studere, implementere og diskutere ulike algoritmer for signalprosessering og analyse, f.eks. filtrering, beregning av effektivverdier, statistiske egenskaper og frekvensanalyser med FFT.
- Komprimering og seleksjon av data før overføring via satellitt. Dette er nødvendig grunnet overføringskapasitet og -kostnader.
- Mottak, lagring og presentasjon av data på landsiden. Her inngår post-prosessering, f.eks. sammensetting av datapakker fra satellitt og sammenligning av ny data og historisk data.

Figur 1.1 viser hvordan data skal sendes til Brunvoll sentralt ved hjelp av satellittsystemet Iridium. For å få til dette skal det benyttes en DTAC-enhet (*Device Tracking and Control*) fra Powex AS, som antydet i oppgaveteksten. Beskrivelse og spesifikasjoner for DTAC-enheten finnes i henholdsvis vedlegg C og D. Et mer detaljert blokkdiagram presenteres i kapittel 5.



Figur 1.1: Blokkdiagram over tilstandsovervåkningssystemet. Data sendes fra skipet via satellitt til en mottakstasjon i Danmark. Derifra sendes data til Brunvoll AS over TCP/IP eller på e-post.

1.1 Rapportens oppbygning

Denne rapporten er oppbygd på følgende måte:

Kapittel 2 – System- og problembeskrivelse
Dette kapitlet gir en overordnet fremstilling av thrustersystemer og prosjektoppgaven. Viktige motivasjonsfaktorer for tilstandsovervåking trekkes frem. Dagens situasjon med hensyn til datalogging og tilstandsovervåking beskrives.

Kapittel 3 – Analyse- og prosesseringsmetoder i tidsplanet
Teorien bak ulike metoder for signalanalyse og -prosessering beskrives. Den første metoden viser hvordan et signals statistiske egenskaper kan beregnes «online» ved hjelp av robuste algoritmer. Deretter defineres IIR-filteret, som er mye brukt i digital signalbehandling. Filteret er basert på en dynamisk modell med koeffisienter som bestemmer frekvensresponsen. Koeffisientene gjør filteret svært allsidig. Så defineres effektivverdiberegninger, som er et kvadratisk gjennomsnitt av datapunkter. Effektivverdien brukes ofte ved måling av vibrasjon og elektrisk strøm og spenning. Til slutt blir det vist hvordan et histogram kan brukes til å få oversikt over, og komprimere, data. Formler for beregning av statistiske egenskaper til gruppeinndelt data presenteres.

Kapittel 4 – Analysemetoder i frekvensplanet
I dette kapitlet beskrives fouriertransformasjonen, og hvordan denne brukes til å analysere innholdet av ulike frekvenskomponenter i et signal. Både digital fouriertransformasjon (DFT) og en algoritme for å beregne denne, kalt *Fast Fourier Transform* (FFT), beskrives. FFT er en av de aller viktigste algoritmene for signalbehandling (f.eks. komprimering av audio/video) i dag.

Kapittel 5 – Systemdesign
Her beskrives de tre programmene som er utviklet i prosjektet. «Hovedprogrammet» prosesserer og analyserer data fra thrustersystemene ombord. Det vises hvordan teknikkene som ble introdusert i kapittel 3 og 4 blir brukt i praksis. «Senderprogrammet» henter analyseresultater fra hovedprogrammet, pakker og sender de avgårde over satellitt. Til slutt beskrives «mottaksprogrammet» på landsiden. Her pakkes data ut og lagres på strukturert form i en database for enkelt oppslag senere (fremtidig presentasjonsverktøy). Konfigurasjon av programmene ved hjelp av XML-filer spesifiseres.

Kapittel 6 – Valg av parametre og analyser
I dette kapitlet presenteres anbefalinger til valg av måleparametre og analyser av disse. Riktig valg av parametre er viktig for å få mest mulig informasjon om thrustersystemenes tilstand. Både signaler som kan måles direkte fra PLS-grensesnittet og signaler som krever dedikert måleutstyr og videreutvikling av systemet drøftes.

Kapittel 7 – Praktiske resultater
En rekke datasett er kjørt gjennom tilstandsovervåkningssystemet for å dokumentere og verifisere at det fungerer i praksis. Datasettene er både reelle målinger og simulerte. De fleste av blokkene for signalprosessering og -analysering blir prøvd ut, f.eks. IIR-filteret, histogram- og FFT-blokken. Etterhvert som resultatene presenteres drøftes eventuelle avvik i forhold til resultater fra et «ideelt» system uten datakomprimering (Matlab).

Kapittel 8 – Drøfting og forslag til videre arbeid
Dette er et allsidig kapittel som drøfter de ulike delene av prosjektet og kommer med forslag til forbedringer og videre utvikling. Først drøftes ulike kommersielle løsninger for et praktisk salgbart tilstandsovervåkningssystem. Løsningene skilles fra hverandre ved grad av kompleksitet, kostnader, muligheter for høyhastighetsmålinger og satellittkommunikasjon.

Deretter presenteres et forslag til krav et fremtidig presentasjonsverktøy må tilfredsstille. Ulike metoder for presentasjon drøftes, i hovedsak enten et webbasert grensesnitt eller et frittstående program.

Fordeler ved å integrere tilstandsovervåkningssystemet opp mot eksisterende programvare som i dag benyttes i servicearbeid presenteres. Utnytting av data til analyser på tvers av ordre og prosjekter, og ikke bare til tilstandsovervåking, kan være et viktig moment med hensyn til videre utvikling av thrustersystemene.

Til slutt presenteres fordeler og ulemper med de tre programmene som er utviklet. Mange forslag til forbedringer er inkludert.

Kapittel 9 – Konklusjon
Som en avslutning på masteroppgaven, oppsummeres prosjektet i dette kapitlet. De viktigste hovedmomentene fra hvert enkelt kapittel i rapporten trekkes frem, slik at et helhetlig bilde av arbeidet og resultatene dannes. Grad av måloppnåelse med hensyn til punktene listet i innledningen blir presentert.

Kapittel 2

System- og problembeskrivelse

Dette kapitlet gir først en overordnet systembeskrivelse av Brunvolls thrustersystemer, som denne masteroppgaven er rettet mot. Deretter presenteres motivasjonsfaktorer for tilstandsovervåking.

2.1 Thrustersystemer

Thrustere (sidepropeller) øker manøvrerbarheten til skip, og er ofte helt nødvendig når det f.eks. legges til kai eller ved bruk av dynamisk posisjonering (DP). Brunvoll AS produserer thrustere og tilhørende styresystemer til de fleste typer skip, alt fra ferger til cruisebåter.

Tre hoveddtyper thrustere levert av Brunvoll er

- tunnelthrustere som plasseres på tvers i skipsskroget. Brukes for sideveis manøvrering uten behov for bevegelse i lengderetningen.
- asimutthrustere som plasseres under skipet. Med en asimuttruster kan thrusteren, og dermed kraften, roteres i en vilkårlig retning. Dette gir økt manøveringsevne sammenliknet med en tunnelthruster. Asimutthrustere kan også benyttes til framdrift, se f.eks. figur 2.3.
- kombinerte tunnel- og asimutthrustere for å utnytte posistive egenskaper fra begge typene. Kombithrusteren fungerer som en vanlig tunnelthruster, men kan senkes ned under skroget slik at den blir en asimutthruster når det er ønskelig. Ved å heve thrusteren når skipet er i transit, reduseres friksjonen og dermed drivstofforbruket.

Thrusterne drives enten av elektromotorer, dieselmotorer eller hydraulikk. Av disse er førstnevnte vanligst.

Figur 2.1 viser et detaljert tverrsnitt av en tunnelthruster. Et vinkelgir må til for å overføre kraften fra den vertikalmonterte motoren (ikke vist) til den horisontale propellen. Som det går fram av figuren, tar giret relativt stor plass og er med på å hindre vanngjennomstrømningen. Dette fører igjen til redusert virkningsgrad. Giret inneholder også et hydraulisk system for vridning av propellbladene i tillegg til selve kraftoverføringen. Denne vridningen blir heretter omtalt som «pitchvinkel». Som det framgår av tegningen, benyttes hydraulikk også til å beskytte systemet mot inntrenging av vann og til smøring av thrusteren.

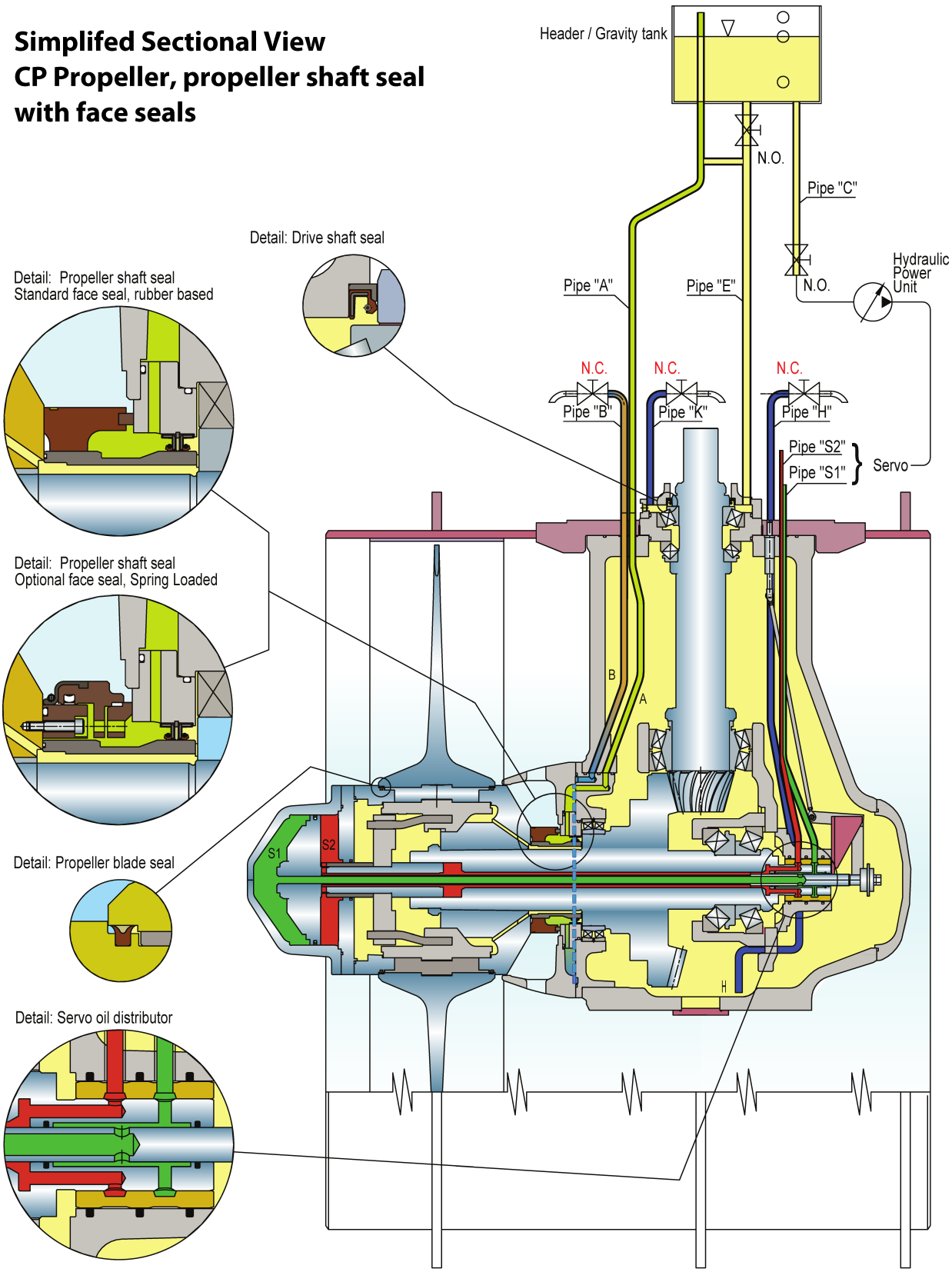
En ny type motor/thrustersystem kalt *Rim Driven Thruster* (RDT) har ikke gir, og dermed øker virkningsgraden vesentlig. En RDT er en permanentmagnet elektromotor der propellen er rotoren, og statoren ligger som en ring rundt. Systemet er sjøvannssmurt og behøver ingen kjøling. Figur 2.3 viser en RDT brukt som framdriftspropell.

2.1.1 Styresystemer

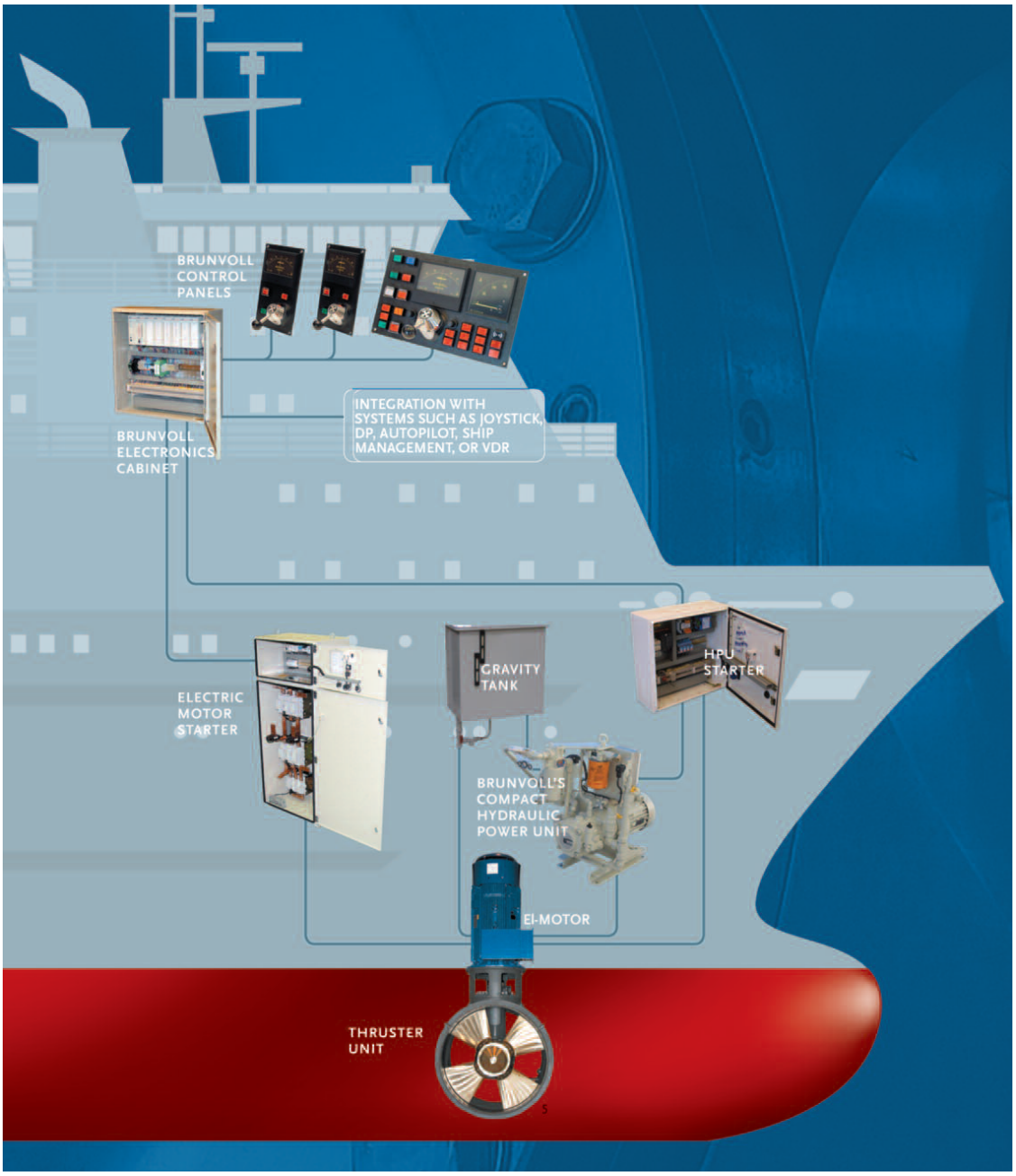
Som tidligere nevnt, leveres thrusterne med komplett styresystem; dette er vist på figur 2.2. En *Hydraulic Power Unit* (HPU) genererer oljetrykk for endring av pitchvinkel, asimutvinkel og heving/senking av kombithrustere. Gravitasjonstanken på figuren fungerer som et reservoar for hydraulikk- og smøreolje.

For å redusere og kontrollere oppstartsstrømmene ved elektromotordrift benyttes en motorstarter. Dette er viktig for å unngå brå overbelastninger på skipets elektriske system som ofte har begrenset kapasitet i utgangspunktet. I tillegg reduseres slitasjen ved å ha en kontrollert oppstartsfase. Servomotoren(e) i HPU-en har også sine egne startere av de samme grunnene. Motorstarterne er i praksis stjerne-trekant-vendere, autotrafostartere, elektroniske «softstartere» eller frekvensomformere. Sistnevnte muliggjør trinnløs regulering av thrusterens turtall.

Alle disse systemene styres av et elektronisk system av PLS-er og/eller industrielle datamaskiner (IPC). Styresystemet kan motta settpunktsdata fra flere ulike kilder: Brunvolls egne styrepaneler på broa, skipets DP-system, joystick, autopilot eller andre eksterne systemer. Signalene behandles og sendes til regulatorsløyfene for pitchvinkel, asimutvinkel, turtall osv. I tilfeller hvor både pitchvinkel og turtall kan endres, beregner PLS-programmet en optimal kombinasjon av disse for å oppnå ønsket skyvkraft. Optimaliseringskriteriene er energiforbruk, slitasje, støy og vibrasjon.



Figur 2.1: Tverrsnitt av en thruster fra Brunvoll. Vinkelgiret overfører kraften fra den vertikalmontert motoren til propellen. Rør «S1» og «S2» brukes til å endre pitchvinkelen.



Figur 2.2: Oppbygningen av et thrustersystem. Styresystemet regulerer kraften og retningen på kraften produsert av thrusteren. Et hydraulisk system brukes til ending av pitch og eventuell asimut og elevasjon. Kraftelektronikk starter og stopper drivmotoren og regulerer turtallet.

Figur 2.3: Asimut RDT for fremdrift på MF «Eiksund». Bildet er hentet fra [14].

2.2 Motivasjon for tilstandsovervåking

Som nevnt i innledningen, settes det stadig strengere krav til stabilitet og oppetid til mekanisk utstyr; her er Brunvolls thrustersystemer intet unntak. Slik situasjonen er nå, har de fleste leverte thrustersystemer begrensede muligheter for datalogging og tilstandsovervåking. Skal data samles inn, må servicepersonell sendes avgårde med temporært måle- og loggeutstyr. Befinner skipet seg tilfeldigvis på andre siden av kloden, blir også kostnadene deretter. Enkelte skip fått installert dataloggere som tar vare på thrusterdata en viss periode, men det fins ingen automatisk overføring eller analysering av data. Minnekort eller disker må enten hentes av servicepersonell eller sendes i posten til Brunvoll.

Gjennom denne masteroppgaven skal det lages et system som tar seg av automatisk logging, analyse, overføring og mottak. Mye tyder på at i fremtiden vil et slikt tilstandsovervåkningssystem måtte leveres med alle nye thrusteranlegg. Forespørsler fra kunder viser også at det er positivitet i markedet for en slik løsning.

I den følgende listen er de viktigste motivasjonsfaktorene for tilstandsovervåking oppsummert.

- Automatisere arbeidet med datainnsamling, som igjen fører til reduserte kostnader for bedriften.
- God og tidlig planlegging av neste servicestopp. Deler og servicepersonell kan bestilles på forhånd. De unødvendige driftsstansene unngås i større grad.
- Over tid vil innsamlet data kunne benyttes til analyse av bruksmønster og feilsituasjoner, som igjen kan brukes til videreutvikling av thrustersystemene. Dette gir en generell økning i kvaliteten på nye anlegg.

- Av forrige punkt følger at bedriftens omtale blir styrket.
- Innsamlet data blir er slagkraftig verktøy i en salgssituasjon. Det kan vises til presise statistikker over f.eks. oppetid.
- Brunvoll viser sine kunder at bedriften tar vedlikehold og tilstandsovervåking på alvor. Automatisk datainnsamling og analyse kan lette arbeidet til maskinistene ombord på skipet.

Kapittel 3

Analyse- og prosesseringsmetoder i tidsplanet

Dette kapitlet vil gi en innføring i signalprosessering og signalanalyse i tidsplanet, relevant for prosjektet. Praktisk implementasjon og robusthet blir beskrevet.

3.1 Deskriptiv statistikk

Ofte er det behov for et raskt overblikk over en stor datamengde. Til dette brukes de kjente nøkkeltallene fra deskriptiv statistikk: middelverdi, varians (og standardavvik), median og typetall. Siden beregning av nøkkeltallene skal implementeres på en datamaskin som regner med flyttall, er det viktig at algoritmene som brukes er numerisk stabile og robuste.

I denne rapporten blir det fokusert på aritmetisk middelverdi og varians, fordi beregninger av disse kan programmeres uten tilnærminger (annet enn avrundingsfeil) og med små minnekrav. Algoritmene som blir beskrevet er «online» eller «inkrementelle», dvs. at de kan kjøres på en del av det endelige datasettet etterhvert som ny data hentes.

Ved beregning av medianen, som er det «midterste» tallet i observasjonene, må det lages en sortert liste over all innputt-data. For hvert nye datapunkt må listen vedlikeholdes, noe som krever et stort arbeidsminne ettersom antall observasjoner øker. En passende online soteringsalgoritme er *Tree sort* med en kjøretid på $O(n^2)$ i verste tilfelle. [25] Det fins raskere og mindre minnekrevende alternativer, men disse er ikke eksakte algoritmer/metoder. Et eksempel på en slik metode er å gruppere data i klasser som for et histogram, dette blir beskrevet i underkapittel 3.3.

Typetallet er det tallet som forekommer flest ganger i observasjonene, noe som ikke gir mening når datatypen er flyttall. Også her er løsningen å gruppere data, se underkapittel 3.3.

De neste underkapitlene er hovedsaklig basert på Knuth [18] og Welford [34], og vil fokusere på statistikkparametrene aritmetisk middelverdi og varians.

3.1.1 Aritmetisk middelverdi

Definisjonen på den aritmetiske middelverdien $\bar{x}$ til et endelig datasett er

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{3.1}$$

hvor n er antallet observasjoner x_i. Dersom definisjonen brukes direkte, må antallet punkter n og summen av dem hele tiden lagres og oppdateres ettersom nye datapunkter legges inn. Etterhvert som n øker, kan summen $\sum_{i=1}^{n} x_i$ vokse seg uforholdsmessig stor, noe som gir tap av presisjon og, i verste tilfelle, fører til overflyt. Dette resulterer i et gjennomsnitt $\bar{x}$ med lav presisjon, på grunn av implementasjonen av et flyttall på en datamaskin. Mange av bytene som er tilgjengelig for f.eks. en IEEE 754 `double`-variabel «brukes opp» til å representere det store tallet slik at desimaltallene får lavere oppløsning.

For å unngå disse problemene, utledes en inkrementell oppdateringlov fra (3.1). Nå brukes betegnelsen $\bar{x}_n$ som gjennomsnittet frem til og med datapunkt n. Først trekker vi ut x_n fra summen i (3.1), og får

$$\bar{x}_n = \frac{1}{n} \left(x_n + \sum_{i=1}^{n-1} x_i \right)$$

Deretter multipliseres dette med $\frac{n-1}{n-1}$, og $\bar{x}_{n-1}$ identifiseres som følger.

$$\begin{aligned}
\bar{x}_n &= \frac{n-1}{n} \left(\frac{1}{n-1} x_n + \frac{1}{n-1} \sum_{i=1}^{n-1} x_i \right) \\
&= \frac{n-1}{n} \left(\frac{1}{n-1} x_n + \bar{x}_{n-1} \right) \\
&= \frac{1}{n} (n\bar{x}_{n-1} + x_n - \bar{x}_{n-1}) \\
&= \bar{x}_{n-1} + \frac{1}{n} (x_n - \bar{x}_{n-1})
\end{aligned} \tag{3.2}$$

Nå har vi fått en rekursiv formel (3.2) for oppdatering av gjennomsnittet hvor vi ikke behøver å lagre den problematiske summen $\sum_{i=1}^{n} x_i$. Dermed får vi økt numerisk presisjon og noe redusert minneforbruk, i tillegg til at vi unngår overflyt. Det er verdt å merke seg at antall datapunkter n uansett må telles.

3.1.2 Empiriske varians og standardavvik

Den empiriske variansen s_n^2 til et datasett med n observasjoner er definert som

$$s_n^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x}_n)^2 \tag{3.3}$$

Denne definisjonen gir tilsvarende problemer som gjennomsnittet. Siden vi nå summerer kvadrerte tall, vil vi faktisk tape presisjon raskere sammenlignet med (3.1). Welford utleder en rekursiv formel for variansen i [34]. Resultatet er

$$s_n^2 = \frac{1}{n-1}\left[(n-2)s_{n-1}^2 + (x_n - \bar{x}_n)(x_n - \bar{x}_{n-1})\right] \tag{3.4}$$

Utledningen av (3.4) er utelatt, fordi den er relativt lang. Sammenligner vi (3.2) og (3.4), ser vi at de har en del likhetstrekk. Utnytter vi dette og kombinerer likningene, får vi følgende «online»-formler som oppdaterer tre variabler, Δ, $\bar{x}_n$ og M_n, etterhvert som ny data x_n legges inn:

$$\Delta = x_n - \bar{x}_{n-1} \tag{3.5}$$

$$\bar{x}_n = \bar{x}_{n-1} + \frac{1}{n}\Delta \tag{3.6}$$

$$M_n = M_{n-1} + \Delta(x_n - \bar{x}_n) \tag{3.7}$$

For å beregne varians brukes

$$s_n^2 = \frac{1}{n-1}M_n \tag{3.8}$$

som ikke er en del av oppdateringsalgoritmen, men kjøres bare når s_n^2 blir etterspurt. Det empiriske standardavviket er $s_n = \sqrt{s_n^2}$. Alle summerings- og subtraheringsoperasjoner i (3.5)–(3.7) blir utført på tall som er relativt små, slik at den numeriske presisjonen blir ivaretatt. Fortsatt er det nødvendig å holde rede på antall observasjoner n, noe som ikke byr på store utfordringer siden dette er et heltall.

3.2 IIR-filter

Filtrering av data er et viktig verktøy i signalbehandling. Målet med filtrering er å trekke ut mest mulig brukbar informasjon fra et signal, f.eks. ved å dempe støy, undertrykke visse frekvensbånd, eller dempe andre fenomener som virker forstyrrende på det vi er interessert i [13]. Filtrering kommer ikke uten kostnad; ønsket data vil alltid, i større eller mindre grad, bli påvirket.

Et enkelt, men likevel allsidig, digitalt filter kalt *Infinite Impulse Response*-filter (IIR) er gitt av følgende transferfunksjon i z-planet:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{i=0}^{P} b_i z^{-i}}{\sum_{j=0}^{Q} a_j z^{-j}}, \quad a_0 \neq 0 \tag{3.9}$$

I tidsplanet tilsvarer dette

$$\begin{aligned} a_0 y(k) = {} & b_0 x(k) + b_1 x(k-1) + \cdots + b_P x(k-P) \\ & - a_1 y(k-1) - a_2 y(k-2) - \cdots - a_Q y(k-Q), \quad a_0 \neq 0 \end{aligned} \tag{3.10}$$

der $x(k)$ er ufiltrert data inn og $y(k)$ er filtrert data ut av IIR-filteret ved punktprøve k. Koeffisientene a_j og b_i bestemmer henholdsvis filterets poler og nullpunkter, disse velges slik at ønsket frekvens- og amplituderespons oppnås. Det er dette som gjør et IIR-filter så allsidig. Andre fordeler er at det er beregningsmessig effektivt sammenliknet med det enklere FIR-filteret (dvs. et IIR-filter med $a_j = 0$ for alle $j \neq 0$) og det er mulig å få til relativt skarpe og smale frekvensresponser. Eksempler på kjente IIR-filtre er Butterworth, Bessel og Chebyshev, i tillegg til det som kalles *moving average* eller flytende gjennomsnitt. Filteret kan også benyttes til mer enn ren filtrering, for eksempel integrasjon og derivasjon.[1]

En ulempe er at filteret kan bli ustabilt. Det finnes flere årsaker til dette [30]:

- Filterkoeffisientene er valgt slik at det dynamiske systemet (3.9) er ustabilt. For å sikre at filteret er BIBO-stabilt, må alle poler ligge innenfor enhetssirkelen i z-planet. Dette betyr at vi må ha $|b_i| < 1$ for alle i. Et FIR-filter er alltid stabilt.
- Siden IIR-filteret er rekursivt, vil numeriske avrundingsfeil kunne akkumuleres under drift og føre til ustabilitet.

Mange metoder for å unngå disse problemene eksisterer, f.eks. konisk-kvadratisk programmering fra optimaliseringsfaget, se Lu og Hinamoto [21].

3.2.1 Implementasjon

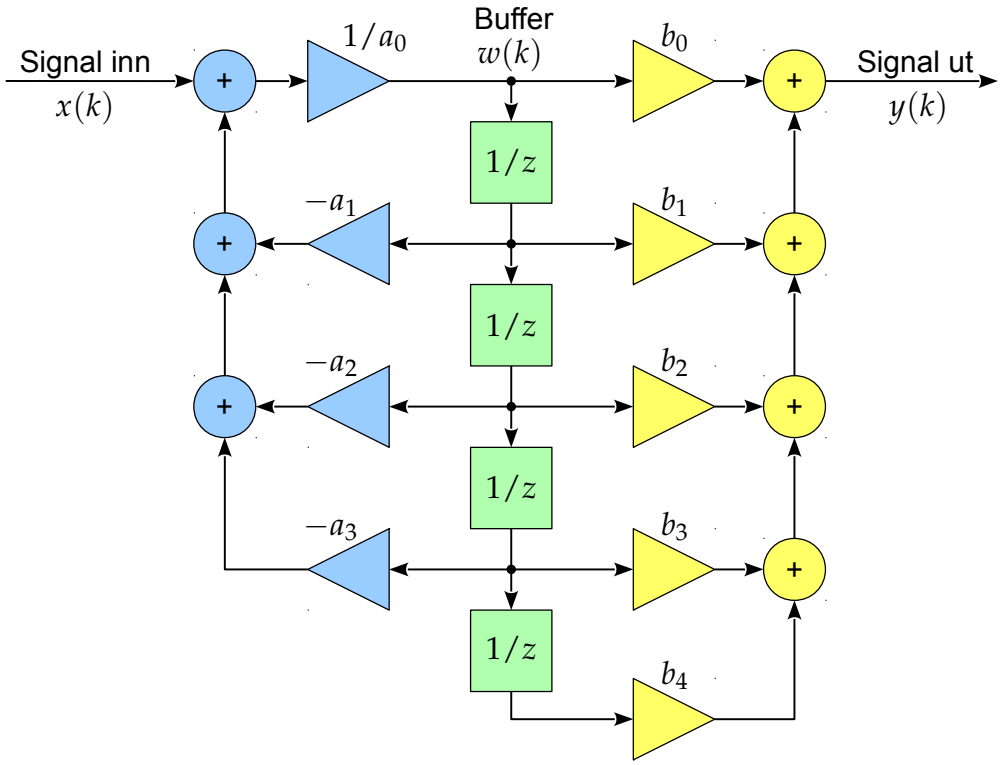
Når IIR-filteret skal implementeres lønner det seg å skrive om og dele opp filterligningen (3.10) og innføre en ny variabel w. Denne variabelen blir i praksis et array som bufrer data. Ved å dele opp (3.10) er det nemlig ikke lengre nødvendig å lagre $x(i)$ for alle $i = k, \ldots, k-P$ og $y(j)$ for alle $j = k, \ldots, k-Q$.

[1] Integrasjon og derivasjon er spesialtilfeller av filtre, henholdsvis lav- og høypass.

En løsning for å få til dette er

$$\begin{aligned} w(k) &= \frac{1}{a_0}\left[x(k) - a_1 w(k-1) - \cdots - a_Q w(k-Q)\right], \quad a_0 \neq 0 \\ y(k) &= b_0 w(k) + b_1 w(k-1) + \cdots + b_P w(k-P) \end{aligned} \tag{3.11}$$

som kalles *Direct Form II* [31]. Et blokkskjema for IIR-filteret på denne formen er vist på figur 3.1. Den største fordelen er at antall elementer i w som må lagres er $N = \max\{P, Q\}$, en betydelig reduksjon sammenlignet med $P + Q$ for (3.10). Dette er tilsvarende[2] implementasjon som Matlab-funksjonen `filter(b,a)` benytter [23].



Figur 3.1: Eksempel på IIR-filter på *Direct Form II* med $P = 4$ foroverkoblinger og $Q = 3$ tilbakekoplinger. Vi behøver bare én buffer w med $N = \max\{P, Q\} = 4$ elementer for mellomlagring.

[2]Matlabfunksjonen `filter(b,a)` benytter egentlig *Direct Form II Transposed*, altså en transponert versjon av (3.11).

3.2.2 Brukseksempel: enkel førsteordensderivator

For å illustrere et bruksområde for IIR-filteret, kan vi utlede en enkel derivator (høypassfilter). Den tidsderiverte til en funksjon $f \in \mathbb{R}$ er definert som

$$\frac{\mathrm{d}f(x)}{\mathrm{d}t} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{3.12}$$

slik at $\frac{\mathrm{d}f(x)}{\mathrm{d}t}$ blir stigningstallet til tangenten i punktet $(x, f(x))$. I et diskret system (eller et diskretisert system) kan den deriverte tilnærmes med en førsteordensapproksimasjon som følger

$$\dot{x}(k) \approx \frac{x(k) - x(k+1)}{T_s} \tag{3.13}$$

der T_s er punktprøveperioden og k er punktprøven ved tidspunkt $t = kT_s$. Transformert til z-planet får vi

$$Y(z) = \frac{1 - z}{T_s} X(z) \tag{3.14}$$

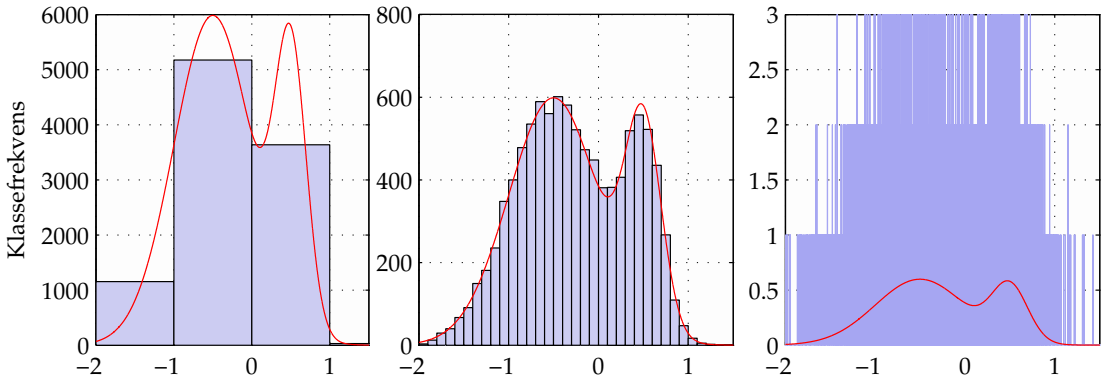
som passer rett inn i IIR-filterets definisjon med koeffisientene $b_0 = 1, b_1 = -1$ og $a_0 = T_s$. Approksimasjoner av høyere orden finnes, f.eks. en som er utledet fra Simpsons integrasjonsregel, se Al-Alaoui [1].

3.3 Histogram og gruppeinndeling av data

Dette underkapitlet er i all hovedsak basert på Kristensen [20].

I situasjoner hvor store datamengder (mange observasjoner) skal analyseres, kan et histogram være et godt hjelpemiddel til å skaffe seg oversikt. Observasjonene deles inn i sammenhengende klasser som dekker hele, eller mesteparten av, verdiområdet til dataene. Antall observasjoner som faller innenfor den enkelte klasse, er klassens *frekvens*. Skulle en observasjon falle på en klassegrense, er den vanlige konvensjonen at observasjonen tilhører den øverste klassen. Dette vil si at en klasse karakteriseres med en inklusiv nedre grense a og en opp til-grense b, altså på formen $[a, b\rangle$.

Antall klasser avhenger av om en god oversikt (få og breie klasser) eller mye informasjon (mange og smale klasser) er ønskelig. Figur 3.2 viser hvordan antall klasser, og dermed klassebreddene, påvirker et histogram. I dette tilfellet har alle samme bredde, men dette er ikke et krav. Det finnes formler og algoritmer som kan velge den «beste» klassebredden, men disse er ofte bare brukt som tommelfingerregler, se f.eks. [7, 28].



(a) For breie. **(b)** Fornuftig bredde. **(c)** For smale.

Figur 3.2: Eksempler på valg av klassebredder i et histogram. For breie klasser gir tap av presisjon, mens for smale gir tap av oversikt. Et kompromiss mellom de to ytterpunktene er det beste valget.

Når data visualiseres i et histogram, er det den enkelte søyles *areal* som er proporsjonalt med antall observasjoner i klassen (frekvensen). Dette vil si at en smal klasse med få observasjoner kan ha en «høyere» søyle enn en brei klasse med mange observasjoner. Høyden på hver enkelt søyle er relativ frekvens delt på klassebredde. Figur 3.2 viser histogrammer laget på grunnlag av 10 000 tilfeldige tall generert ut ifra en stokastisk variabel med en sannsynlighetstetthet

$$g(x) = \frac{3}{4} f_1(x) + \frac{1}{4} f_2(x) \tag{3.15}$$

der f_i for $i = 1, 2$ er tetthetsfunksjonene

$$f_i(x) = \frac{1}{\sigma_i \sqrt{2\pi}} e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}} \tag{3.16}$$

som er tettheten til normalfordelingsfunksjonen. Forventningsverdiene μ_i og standardavvikene σ_i var satt til

$$\begin{aligned} \mu_1 &= -0{,}5 & \sigma_1 &= 0{,}5 \\ \mu_2 &= 0{,}5 & \sigma_2 &= 0{,}2 \end{aligned}$$

Den kontinuerlige kurven tegnet over histogrammene er $g(x)$ skalert opp slik at det totale arealet under kurven blir likt histogrammenes areal. Dermed kan tetthetsfunksjonen og histogrammene sammenlignes direkte.

3.3.1 Deskriptiv statistikk for gruppert datamateriale

Dersom bare klasseinndelt datamateriale er tilgjengelig, og ikke selve rådataene, kan estimater av de vanlige beliggenhets- og spredningsmål beregnes etter metoder som følger i dette underkapitlet. Fellesnevneren for alle beliggenhets- og spredningsmål som beregnes ut ifra gruppeinndelt datamateriale, er tap av presisjon i forhold til om de var beregnet direkte fra rådata. Som tidligere nevnt, gir smalere klasser høyere presisjon. Figur 3.2 viser hvordan informasjon går tapt hvis klassene er for breie (a) og at det er vanskelig å danne seg en god oversikt hvis klassene er for smale (c). Histogram (b) har et «fornuftig» antall klasser. Som det blir vist senere, vil valg av klassebredder i dette prosjektet også påvirke datamengden som må overføres fra skip til land.

Gruppert empirisk middelverdi

Gjennomsnittet til gruppert datamateriale beregnes som om alle observasjonene innenfor hver enkelt klasse faller på klassens midtpunkt m_i. Det er dette som fører til tap av informasjon hvis vi har for breie klasser. Gruppert empirisk middelverdi er gitt ved

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} f_i m_i \tag{3.17}$$

der f_i er frekvensen (antall observasjoner) i klasse i og n er antall klasser.

Gruppert empirisk varians og standardavvik

Ved å benytte den grupperte middelverdien $\bar{x}$ fra (3.17), kan gruppert empirisk varians s^2 beregnes ved hjelp av

$$s^2 = \frac{1}{1-n} \sum_{i=1}^{n} f_i (m_i - \bar{x})^2 \tag{3.18}$$

Standardavviket er som vanlig $s = \sqrt{s^2}$. Også her baserer vi oss på at observasjonene faller på klassenes midtpunkter. Ofte er det så få klasser at både (3.17) og (3.18) kan implementeres direkte, uten at det får vesentlige konsekvenser for den numeriske robustheten.

Kvartiler, kvantiler og median

Kvartiler deler opp et datamateriale i fire like deler, slik at 25 % av observasjonene ligger til venstre for første kvartil Q_1, 50 % til venstre for andre kvartil Q_2 og 75 % til venstre for tredje kvartil Q_3. Alle observasjoner ligger til venstre for fjerde kvartil, som dermed blir maksimumsverdien til observasjonene.

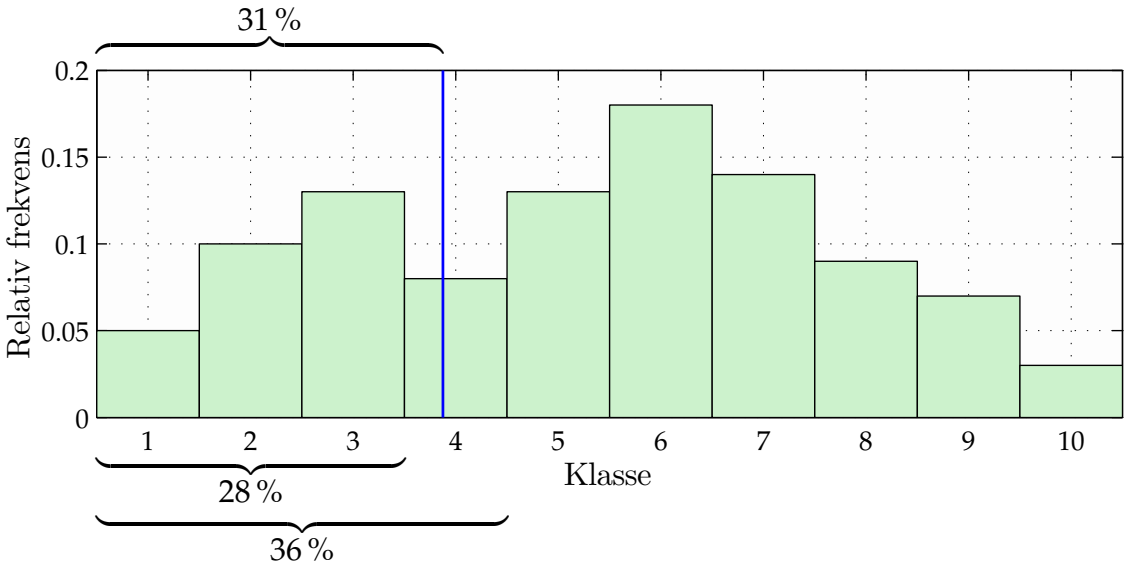
En kvantil (også kalt prosentil) er definert på samme måte, men med valgfri prosentgrense. Dette betyr at kvartilene også er kvantiler.

Medianen $\tilde{x}$ til et datasett er slik at det er like mange observasjoner over $\tilde{x}$ og under $\tilde{x}$ når observasjonene er ordnet i stigende rekkefølge. Som nevnt, ligger 50 % av observasjonene til venstre for Q_2, som er den nøyaktig samme definisjonen som for medianen. Derfor er $\tilde{x} = Q_2$.

Beregning av kvantiler illustreres best med et eksempel. Anta at vi vil finne 31 %-kvantilet $k_{31\,\%}$ til et gruppeinndelt datamateriale vist som et histogram med relative frekvenser på figur 3.3. Ved å summere relativ frekvens fra venstre mot høyre, får vi akkumulert relativ frekvens for hver enkelt klasse. Anta videre at de tre første klassene summeres opp til 28 % mens de fire første summeres til 36 %. Da er det klart at 31 %-kvantilet vi er på jakt etter ligger i fjerde klasse. Siden fjerde klasse inneholder 8 % av observasjonene og vi «mangler» 3 %, må vi gå 3/8 inn i klassen for å komme opp til 31 %. I dette tilfellet beregnes $k_{31\,\%}$ derfor slik:

$$k_{31\,\%} = [\text{fjerde klasses nedre grense}] + \frac{3}{8} \cdot [\text{fjerde klasses bredde}]$$

I et datasett hvor enkelte verdier skiller seg markant fra resten, det vil si at avstanden $|\bar{x} - x_i|$ er stor for enkelte i, vil det aritmetiske gjennomsnittet kunne bli misvisende. I slike tilfeller kan medianen være et mer fornuftig måltall på «gjennomsnittet». Kvantiler brukes ofte for å si noe mer om spredningen av observasjonene enn bare variansen.



Figur 3.3: Eksempel på beregning av kvantiler i et histogram. 28 % av observasjonene ligger i klasse 1–3 og 36 % ligger i 1–4. Den blå streken indikerer 31 %-kvantilet $k_{31\,\%}$ i klasse 4.

3.4 Effektivverdi (RMS)

Ved analyse av tidsvarierende signaler som ofte veksler mellom positiv og negativ verdi, er det greit å vite noe om utsvinget; amplituden eller energien i signalet. Et nyttig måltall har vist seg å være det kvadratiske gjennomsnittet, kalt effektivverdi eller *Root Mean Square* (RMS). Effektivverdien til et kontinuerlig signal $f(t)$ i tidsrommet fra T_1 til T_2 er gitt ved

$$f_{\text{RMS}} = \sqrt{\frac{1}{T_2 - T_1} \int_{T_1}^{T_2} (f(t))^2 \, \mathrm{d}t} \tag{3.19}$$

En viktig egenskap som skiller effektivverdien fra det aritmetiske gjennomsnittet til f, er at effektivverdien til et signal som er symmetrisk om tidsaksen er positiv, mens gjennomsnittet er null. [33]

Tilsvarende definisjon for et sett med diskrete punkter $\{x_i\}_{i=0}^{n}$ er

$$x_{\text{RMS}} = \sqrt{\frac{1}{n} \sum_{i=0}^{n} x_i^2} \tag{3.20}$$

som også forklarer det engelske navnet. Navnet «effektivverdi» stammer fra elektroteknikken. Påtrykkes en rent resistiv motstand en likespenning U, utvikles en effekt P i motstanden. For å få utviklet den samme effekten ved hjelp av en vekselspenning, kan det vises at vekselspenningens RMS-verdi må være U.

I praksis benyttes et RMS-vindu når effektivverdiberegninger implementeres, det vil si bare de m siste datapunktene benyttes:

$$x_{n,\text{RMS}} = \sqrt{\frac{1}{m} \sum_{i=n-m}^{n} x_i^2} \quad \text{der} \quad n \geq m \tag{3.21}$$

Med denne implementasjonen dannes et *RMS-signal*. Valget av m blir en avveining mellom et tregt og stabilt eller et raskt og støyete RMS-signal (henholdsvis stor m og liten m).

Kapittel 4

Analysemetoder i frekvensplanet

Ofte er en ren tidsanalyse ikke tilstrekkelig for å identifisere visse egenskaper ved et signal. Dette kapitlet vil derfor beskrive analysemetoder i frekvensplanet, hovedsaklig spektrumsanalyse med fouriertransformasjon.

4.1 Fouriertransformasjon

Fouriertransformasjonen benyttes når frekvensspekteret til et signal skal identifiseres. Fouriertransformasjonen $\mathcal{F}\{\cdot\}$ til en funksjon $f(t)$ er definert som det komplekse integralet

$$F(j\omega) = \mathcal{F}\{f(t)\} = \int_{-\infty}^{\infty} f(t)e^{-j\omega t}\mathrm{d}t \tag{4.1}$$

der t er tid målt i sekunder og ω er frekvens målt i rad/sekund. Den tilsvarende inverse transformasjonen er

$$f(t) = \mathcal{F}^{-1}\{F(j\omega)\} = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(j\omega)e^{j\omega t}\mathrm{d}t \tag{4.2}$$

Det eksisterer flere alternative definisjoner på fouriertransformasjonen etter bruksområde. For en grei oversikt, se f.eks. Craig [6]. Formel (4.1) og (4.2), som er basert på Brown og Hwang [3], blir benyttet i denne rapporten.

Fouriertransformasjonen er en alternativ representasjon av $f(t)$ som sier hvilke frekvenser $f(t)$ inneholder. Denne representasjonen inneholder både frekvens- og faseinformasjon, og siden tidsfunksjonen $f(t)$ kan rekonstrueres ved hjelp av inverstransformasjonen, er informasjonsinnholdet i $F(j\omega)$ og $f(t)$ likt.

4.1.1 Diskret fouriertransform (DFT)

Dette underkapitlet er basert på Proakis [27] og Brown og Hwang [3]. I praktisk anvendt signalanalyse samples reelle signaler og mates inn i en datamaskin. For å kunne beregne frekvensinnholdet til sekvensen av punktprøver, brukes den diskrete fouriertransformasjonen (DFT), som er definert som følger:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi nk}{N}} \quad \text{der} \quad k = 0, 1, \ldots, N-1 \tag{4.3}$$

Her er $x(n)$ er en sekvens med N punktprøver og $X(k)$ er den tilhørende sekvensen med frekvens- og faseinformasjon av samme størrelse. I praksis kan vi anta at $x(n)$ er reell, mens $X(k)$ er kompleks. Som for den kontinuerlige transformasjonen, eksisterer det også en diskret inverstransformasjon:

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{j\frac{2\pi nk}{N}} \quad \text{der} \quad n = 0, 1, \ldots, N-1 \tag{4.4}$$

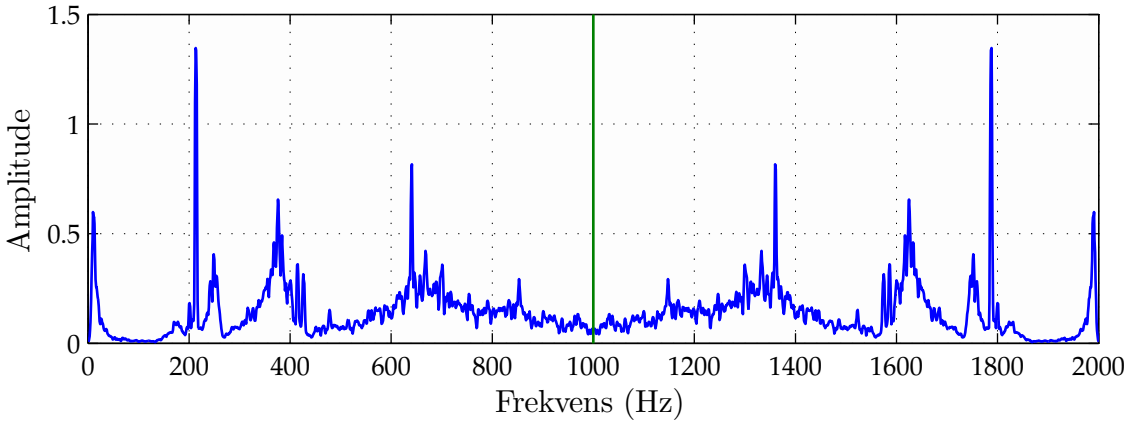
slik at opprinnelig data $x(n)$ kan gjenskapes ved hjelp av $X(k)$. En spesiell egenskap $X(k)$ har, er at den er symmetrisk om sekvensens midtpunkt, det vil si

$$\begin{aligned} X(N-1) &= \overline{X}(1) \\ X(N-2) &= \overline{X}(2) \\ &\vdots \\ X(N-N/2) &= \overline{X}(N/2) \end{aligned}$$

der $\overline{X}$ betyr kompleks konjugert. Konsekvensen av dette er at halvparten av $X(k)$, som også kalles «linjer», er redundant og kan fjernes (se figur 4.1). Dette er intuitivt riktig siden hvert element i X inneholder to verdier, en reell og en imaginær, slik at vi tilsammen har like mange verdier som i det opprinnelige signalet $x(n)$. Ingen informasjon går tapt, det opprinnelige signalet kan gjenskapes ved hjelp av inverstransformasjonen (4.4). Bevis for dette er utledet av Proakis i [27].

Betegnelsene *frekvensspekter* eller *fourierspekter* viser (oftest) til plottet av absoluttverdien til den komplekse $X(k)$. Et slikt spekter er vist i figur 4.1 med $N = 2048$. Her er alle linjene tatt med, og redundansen kommer klart til syne. Videre i rapporten benyttes alltid $|X(k)|$ for $k = 0, \ldots, N/2 - 1$.

Den høyeste frekvensen i spekteret, ved $k = N/2 - 1$, er gitt av $f_{\max} = f_{\text{Nyquist}} = f_s/2$ der f_s er punktprøvefrekvensen. Oppløsningen horisontalt, det vil si avstanden mellom linjene i spekteret, er gitt av $\mathrm{d}f = f_s/N$. Dette betyr at et stort antall punktprøver N gir bedre oppløsning. [24] Ulempen med stor N er at datainnsamlingen tar lengre tid.



Figur 4.1: Fourierspekter med alle $N = 2048$ linjer. Det er tydelig at halvparten er redundante og kan fjernes. Det brukbare frekvensområdet er dermed 0–1023 Hz i dette eksempelet.

4.1.2 Fast Fourier Transform (FFT)

Når den diskrete fouriertransformen skal implementeres i praksis, brukes algoritmer som går under fellesbetegnelsen *Fast Fourier Transform* (FFT). Dersom DFT-formlene (4.3) og (4.4) ble brukt direkte, ville det innebære en kvadratisk kjøretid $O(N^2)$, i motsetning til FFT som oppnår det samme med bare $O(N \log_2 N)$ der N er antall datapunkter. FFT-algoritmer er en helt fundamental byggekloss i mange sammenhenger: kompresjon av audio/video og stillbilder, signalprosessering og dataanalyse.

Videre beskrives en spesiell og populær FFT-algoritme kalt Cooley–Tukey [5] som ble (gjen)oppdaget i 1965. Det er antatt at antall punkter N er en toerpotens i resten av rapporten. Dersom dette ikke er tilfelle, er en vanlig taktikk å legge til nullelementer på slutten av $x(n)$ inntil N blir en toerpotens (*padding*). Padding brukes også for å øke den *tilsynelatende* oppløsningen til spekteret, fordi det fører til flere linjer. De nye linjene er bare et resultat av interpolering, den egentlige oppløsningen forblir uforandret.

Med utgangspunkt i DFT fra (4.3), kan vi dele opp summen i to ledd, ett for partalls-n og ett for oddetall, henholdsvis $n = 2m$ og $n = 2m + 1$. [29] Oppdelingen gir

$$X(k) = \sum_{m=0}^{N/2-1} x(2m) e^{-j\frac{2\pi 2mk}{N}} + \sum_{m=0}^{N/2-1} x(2m+1) e^{-j\frac{2\pi(2m+1)k}{N}} \tag{4.5}$$

Faktoriserer vi ut en konstant fra det siste leddet, sitter vi igjen med

$$X(k) = \sum_{m=0}^{N/2-1} x(2m) e^{-j\frac{2\pi mk}{N/2}} + e^{-j\frac{2\pi k}{N}} \sum_{m=0}^{N/2-1} x(2m+1) e^{-j\frac{2\pi mk}{N/2}} \tag{4.6}$$

Hvis vi gransker (4.6) viser det seg at vi finner igjen definisjonen på DFT for et datasett med $N/2$ punkter i begge leddene, den første for alle partalls-x, den andre for oddetall. Hvis vi setter

$$W_N^k = e^{-j2\pi k/N} \tag{4.7}$$

kan (4.6) omskrives til

$$X(k) = \sum_{m=0}^{N/2-1} x(2m) W_{N/2}^{km} + W_N^k \sum_{m=0}^{N/2-1} x(2m+1) W_{N/2}^{km} \tag{4.8}$$

$$= F_1(k) + W_N^k F_2(k) \tag{4.9}$$

Videre utnytter vi at de to summene F_1 og F_2 har en periode på $N/2$, slik at $F_i(k + N/2) = F_i(k)$ for $i = 1, 2$. Det viser seg også at $W_N^{k+N/2} = -W_N^k$. Kombinerer vi alt dette, får vi

$$X(k) = \begin{cases} F_1(k) + W_N^k F_2(k), & 0 \leq k \leq \dfrac{N}{2} - 1 \\ F_1(k) - W_N^k F_2(k), & \dfrac{N}{2} \leq k \leq N - 1 \end{cases} \tag{4.10}$$

Cooley–Tukey-algoritmen kjører (4.10) rekursivt med stadig oppdeling i partalls- og oddetallselementer. Dette vil si at den er en splitt-og-hersk-algoritme. Kjøretiden er $O(N \log_2 N)$ og den kan utføre alle beregninger *in-place*. En grundigere analyse er gitt av Proakis i [27].

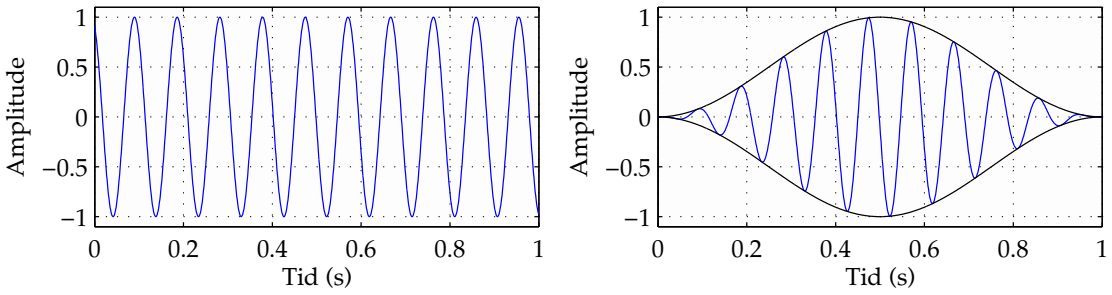
Det eksisterer også FFT-algoritmer for flere dimensjoner. I senere tid (2012) har Hassanieh mfl. ved MIT funnet en metode som potensielt kan redusere kjøretiden til FFT med omlag 10 %, se [11]. Denne metoden er basert på at et fourierspekter ofte domineres av enkelte peaker, mens resten av $X(k)$ er tilnærmet null, såkalt *sparse* data.

4.1.3 Vinduer og lekkasje

I likhet med f.eks. effektivverdiberegninger, kjøres også fouriertransformasjonen på data oppdelt i vinduer. Et vindu kan være de m siste punktprøvene fra et virkelig signal. Anta at vi måler på et rent sinussignal med frekvens 10 Hz og amplitude 1, som vist på figur 4.2 (a).[3] Legg merke til at signalet verken starter eller slutter på null. En FFT-algoritme vil anta at signalvinduet er periodisk, det vil si at det gjentas i det uendelige. Dette er illustrert på figur 4.3 med tre gjentakelser. Her er det lett å se at frekvensinnholdet har blitt forandret på grunn

[3] I eksemplet er det benyttet en punktprøvefrekvens på 1 kHz og et vindu på 1024 punkter. Egentlig burde FFT-spekteret vise én enslig peak på 10 Hz.

av diskontinuitetene mellom vinduene. I frekvensspekteret på figur 4.4 gir dette seg utslag i en stor «lekkasje» av energi rundt den egentlige frekvensen 10 Hz. Lekkasjen skjer også under denne frekvensen, selv om dette ikke kommer godt fram av figuren.



(a) Uten vindusfunksjon. **(b)** Med vindusfunksjon.

Figur 4.2: Eksempel på et sinussignal $x(n)$, både med og uten vindusfunksjon. Vindusfunksjonen tvinger ytterpunktene i signalvinduet mot null for å hindre høyfrekvente sprang ved periodisk utvidelse.

For å motvirke lekkasjeproblemet, benyttes en *vindusfunksjon* som tvinger punktprøvesekvensen til eller nær null i begge endepunktene til vinduet (gjelder ikke alle vindusfunksjoner). Dette minimaliserer diskontinuitetene mellom vinduene og dermed også de høyfrekvente komponentene i spekteret.

En vindusfunksjon $w(n)$ er definert som

$$w(n) = \begin{cases} h(n) & n = 0, 1, \ldots, N-1 \\ 0 & \text{ellers} \end{cases} \tag{4.11}$$

der $h(n)$ bestemmer formen på vinduet. Dersom padding med nullelementer er nødvendig for å gjøre N til en toerpotens, er det viktig at dette gjøres *etter* at vindusfunksjonen er brukt, ellers vil diskontinuitetene fortsatt være der, ikke i overgangen mellom vinduene, men i overgangen mellom reell data og nullelementene. Det finnes et stort utvalg av vindusfunksjoner, alle med sine spesielle egenskaper og bruksområder. Noen av de vanligste er:

Hann (hanning) Brukes til måling på sinuskurver eller kombinasjoner av sinuskurver, smalbåndede signaler og generelt når signalkarektarstikken er ukjent. Dette gjør hannvinduet til et av de mest brukte. Det gir også veldig lite folding (aliasing), som betyr at signalkomponenter med frekvens utenfor FFT-området blir dempet, istedenfor å speile seg ned i området. Et hannvindu er definert ved

$$h(n) = 0{,}5 - 0{,}5 \cos\left(\frac{2\pi n}{N-1}\right) \tag{4.12}$$

Figur 4.7 viser formen på (4.12) i tidsplanet, og dets tilhørende frekvensrespons. Hannvinduet kalles ofte «hanningvindu» pga. (den fonetiske) likheten med hammingvinduet (beskrevet nedenfor).

Hamming Dette vinduet er optimalisert for å fjerne de første (og største) sidelobene i frekvensplanet. Hammingvinduet er gitt av

$$h(n) = 0{,}53836 - 0{,}46164 \cos\left(\frac{2\pi n}{N-1}\right) \tag{4.13}$$

som er plottet på figur 4.8 sammen med frekvensresponsen. Opprinnelig ble hammingvinduet designet som en forbedring av hannvinduet. Til forskjell fra hannvinduet, går ikke amplituden til null ved vinduets start og slutt.

Flat topp er designet for å ha en flat frekvenskarakteristikk rundt senterfrekvensen. I dette smale området blir amplituden helt riktig dersom man måler på et rent sinussignal. Ulempen er at flat topp-vinduet har dårlig frekvensoppløsning utenom senterfrekvensen, se figur 4.9. Definisjon:

$$\begin{aligned} h(n) = 1 - 1{,}93 \cos\left(\frac{2\pi n}{N-1}\right) + 1{,}29 \cos\left(\frac{4\pi n}{N-1}\right) \\ -0{,}338 \cos\left(\frac{6\pi n}{N-1}\right) + 0{,}028 \cos\left(\frac{8\pi n}{N-1}\right) \end{aligned} \tag{4.14}$$

Den korrekte amplituden gjør at flat topp-vinduet ofte benyttes i kalibreringssammenheng der dette er viktig.

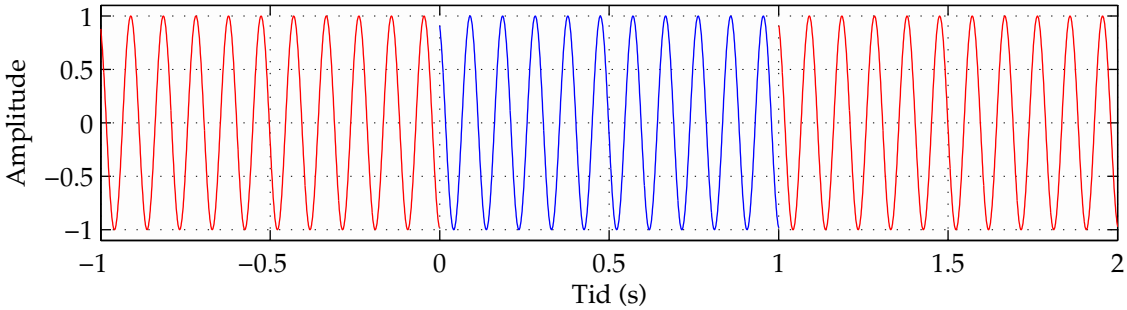
Rektangulært eller uniformt vindu Et rektangulært eller uniformt vindu er det samme som å ikke benytte vindusfunksjon, data sendes direkte til FFT-algoritmen. Definisjonen

$$h(n) = 1 \tag{4.15}$$

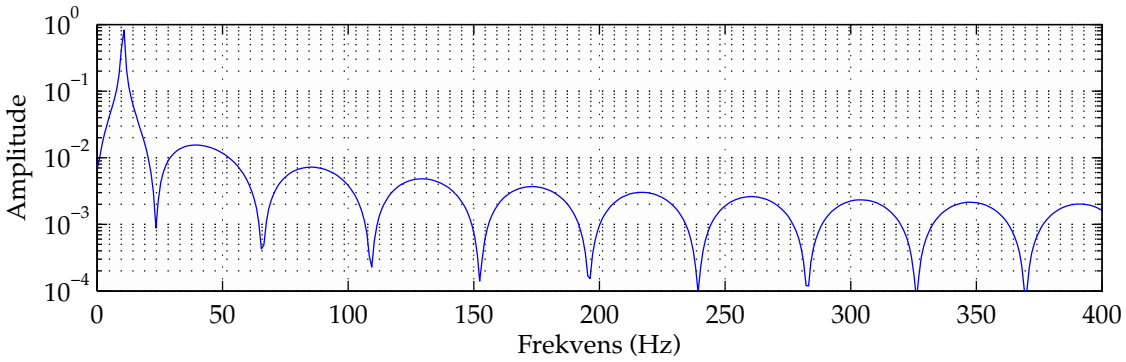
slipper signalet uforandret igjennom. Bruksområdene til dette «vinduet» er i de tilfeller det er kjent at signalet starter og slutter på eller nært null. Eksempler på dette er måling av impulsresponser eller andre transienter.

Generelt blir valg av vindusfunksjon en avveining mellom oppløsning vertikalt (amplitude) og horisontalt (frekvens).

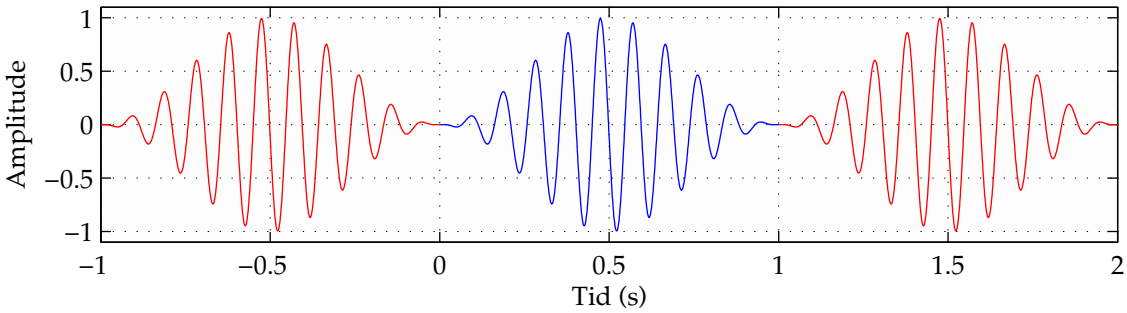
Vi returnerer nå til eksempelet fra begynnelsen av dette underkapitlet. Figur 4.2 (b) viser hvordan amplituden til signalet fra figur 4.2 (a) forvrenges med, i dette tilfellet, et hannvindu. Resultatet er at den periodiske utvidelsen vil bli kontinuerlig, slik det kommer frem på figur 4.5. Dermed er vi kvitt de høyfrekvente sprangene, og vi får et frekvensspekter som vist på figur 4.6. Det er tydelig at vi har fått et mer realistisk spekter når det gjelder frekvensinnholdet i signalet. Men, som det går frem av figurene, vil en vindusfunksjon forvrenge signalets amplitude.



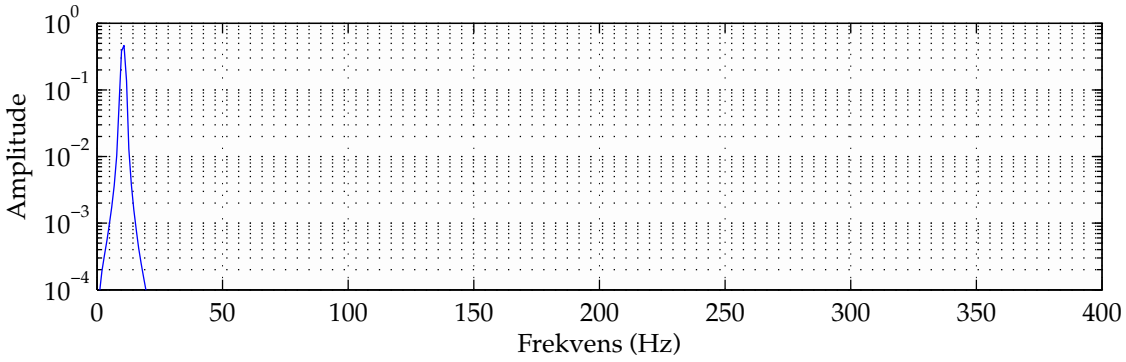
Figur 4.3: Periodisk utvidelse av signalet *uten* vindusfunksjon. Legg merke til diskontinuitetene mellom vinduene som skaper høyfrekvenskomponenter.



Figur 4.4: Frekvensspekter til signalet *uten* vindusfunksjon. På grunn av diskontinuitetene i tidssignalet får vi lekkasje av energi rundt signalfrekvensen 10 Hz.



Figur 4.5: Periodisk utvidelse av signalet *med* vindusfunksjon. Diskontinuitetene mellom vinduene er minimalisert samtidig som amplituden til signalet er forvrengt.



Figur 4.6: Frekvensspekter til signalet *med* vindusfunksjon. Lekkasjen er redusert, fordi vindusfunksjonen fjerner sprangene mellom vinduene.

4.1.4 Overlapping

Hvis vi ser nærmere på figur 4.2 (b) og 4.5, er det klart at i ytterpunktene på vinduene vil en del data rett og slett gå tapt, siden vindusfunksjonen tvinger signalet til null. *Overlapping* av vinduer er en løsning som utnytter mest mulig av innhentet data. Ved å la et visst antall datapunkter fra det forrige vinduet inngå i det neste, vil de fleste punktene havne nær senteret av et vindu i løpet av prosesseringen. På denne måten unngår vi at data blir nullet ut og ikke brukt. Overlappingen oppgis i prosent eller antall punkter.

4.1.5 Gjennomsnitt av frekvensspektre

For å stabilisere frekvensspekteret, redusere støynivået og dermed øke signal-støy-forholdet, brukes en *averaging*-teknikk. Averaging går ut på at det beregnes et visst antall ferdige fourierspektere, og deretter beregnes gjennomsnittet av dem. Antall spektre velges på bakgrunn av kjennskap til signalet som skal analyseres. Er signalet lite påvirket av støy, kan 4–6 spektre (også kalt *averages*) være nok. I tilfeller med mye støy, kan det være aktuelt å benytte f.eks. 20. [15]

Det finnes to hovedmetoder å gjøre gjennomsnittsberegningen på; i det komplekse eller det reelle plan. Dette er forklart detaljert av Lyons i [22], de viktigste poengene herfra presenteres i de neste underkapitlene.

Koherent gjennomsnitt

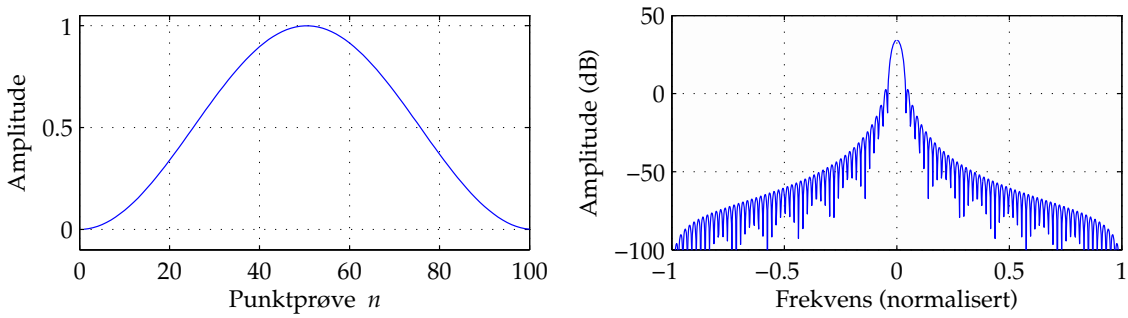
Gjennomsnittsberegning i det komplekse plan kalles *coherent averaging*. Beregning gjøres som følger. Anta at vi vil finne det koherente gjennomsnittet av en vektor G med komplekse elementer, $G = \{a_1 + b_1 j, a_2 + b_2 j\}$, slik frekvensspekteret er. Ved å behandle de reelle og komplekse delene hver for seg, får vi

$$\tilde{G}_{\text{coh}} = \frac{a_1 + a_2}{2} + j\frac{b_1 + b_2}{2} \tag{4.16}$$

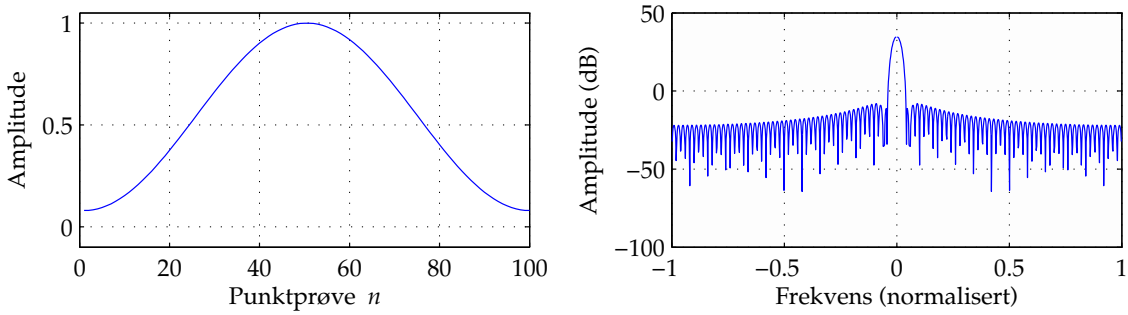
slik at absoluttverdien av $\tilde{G}_{\text{coh}}$ blir

$$\left|\tilde{G}_{\text{coh}}\right| = \frac{1}{2}\sqrt{(a_1 + a_2)^2 + (b_1 + b_2)^2} \tag{4.17}$$

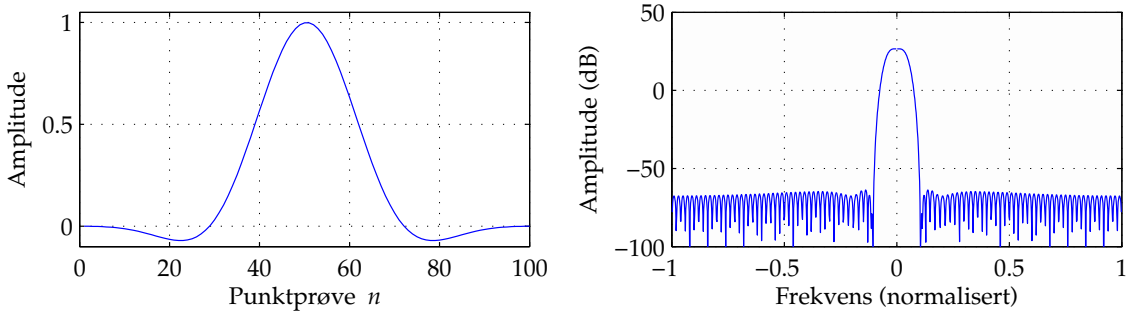
Fordelene med det koherente gjennomsnittet er at vi bevarer både fase- og amplitudeinformasjon samtidig som det *gjennomsnittlige støynivået* i det resulterende FFT-spekteret blir redusert, og i det lange løp fjernet. Årsaken til dette er at vi tar snittet av tall som har lov til å være negative, noe som gjør det mulig å få et støygjennomsnitt lik null. En ulempe er at dersom en frekvenspeak i signalet har tilfeldig fase i hvert FFT-vindu, kan koherent gjennomsnittsberegning redusere signal-støy-forholdet. Det er herfra betegnelsen koherent kommer, signalene i hvert vindu med data må ha like faser.



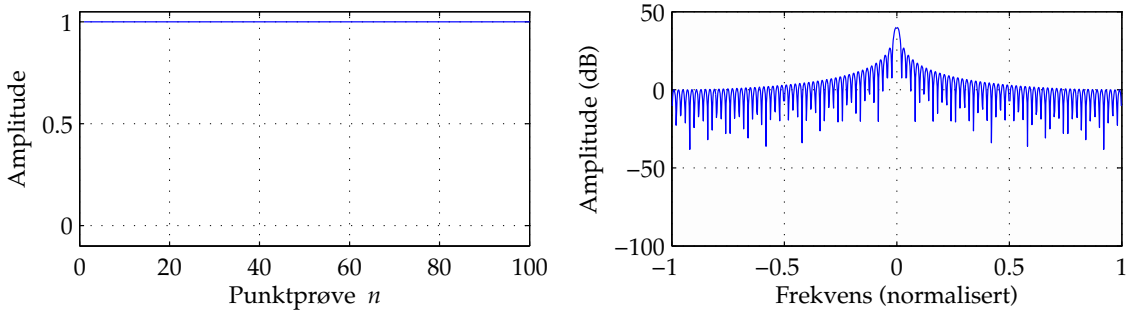
Figur 4.7: Hannvindu.



Figur 4.8: Hammingvindu.



Figur 4.9: Flat topp-vindu.



Figur 4.10: Rektangulært vindu.

Inkoherent gjennomsnitt

Dersom vi i stedet beregner gjennomsnitt i det reelle plan, det vil si gjennomsnittet av absoluttverdiene, kalles dette *incoherent averaging*. Beregning gjøres som følger. Anta at G er som før. Absoluttverdiene til elementene i G blir henholdsvis $\sqrt{a_1^2 + b_1^2}$ og $\sqrt{a_2^2 + b_2^2}$. Det inkoherente gjennomsnittet blir summen av disse delt på to:

$$|\tilde{G}_{\text{incoh}}| = \frac{1}{2}\left(\sqrt{a_1^2 + b_1^2} + \sqrt{a_2^2 + b_2^2}\right) \tag{4.18}$$

Det første vi legger merke til er at vi mister faseinformasjon i det vi kun behandler absoluttverdier. Et annet viktig poeng som skyldes nettopp bruken av absoluttverdier, er at gjennomsnittet alltid blir større enn null. Dette fører til at *variasjonen i støynivået* blir redusert, mens det gjennomsnittlige støynivået ikke forandres. Signal-støy-forholdet vil likevel øke.

I motsetning til koherent gjennomsnittsberegning, vil en frekvenspeak med tilfeldig fase ikke ha noe negativ innvirkning på resultatet. Ved måling på fysiske signaler brukes derfor det inkoherent gjennomsnittet (så lenge faseinformasjon er uviktig). Metoden kalles også «RMS-averaging», på grunn av likheten med (3.20).

Overlapping

Også her kan vi snakke om overlapping av data, men nå i form av vinduer. Hvis vi f.eks. har bestemt at vi beregner snittet over fire vinduer, hvor mange av disse fire skal være med i den neste beregningen? Hvis vi beregner et nytt gjennomsnitt for hvert nye FFT-spekter, vil vi ha en overlapping på tre vinduer. Hvis vi forkaster alle fire, og venter til vi har fire helt nye spektre, har vi ingen overlapping. Siden vi bare får et nytt gjennomsnittsspekter for hvert fjerde spekter inn, er dette også en måte å komprimere data på (på bekostning av oppdateringsfrekvens).

4.2 Effektspektrum (PSD)

Et effektspektrum eller *Power Spectral Density* (PSD) forteller hvordan et signals effekt er fordelt utover ulike frekvenser. Dette underkapitlet er basert på [27]. Effektspektrumet kan beregnes ut ifra DFT (og dermed FFT) som følger:

$$\begin{aligned} P_{xx}\left(\frac{k}{N}\right) &= \frac{1}{N}\left|\mathcal{F}\left\{x(n)\right\}\right|^2 \\ &= \frac{1}{N}\left|\sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi nk}{N}}\right|^2 \quad \text{der} \quad k = 0, 1, \ldots, N-1 \end{aligned} \tag{4.19}$$

Effektspekteret er med på å understreke viktigheten av fouriertransformasjonen og effektive FFT-algoritmer.

Kapittel 5

Systemdesign

Dette kapitlet vil beskrive hvordan tilstandsovervåkningssystemet er designet og hvordan det fungerer. Det er utviklet tre separate programmer i dette prosjektet. En detaljert blokkskjematisk fremstilling av de ulike programmene er vist på figur 5.1.

Ombord i skipet hvor tilstandsovervåkningssystemet er installert, er det behov for en rutine som henter inn data fra thrusternes PLS-systemer, prosesserer, analyserer og sender resultatene over satellitt. Dette er implementert som to adskilte dataprogrammer. Det første, «hovedprogrammet», henter inn, analyserer og lagrer resultatene, mens det andre, «senderprogrammet», kjøres når det er behov for å pakke og sende data til land. Programmene er utviklet i og for Linux, men skal i teorien også kunne kompileres for Windows etter små tilpasninger.

Hovedprogrammet og senderprogrammet kjøres på en dedikert industriell datamaskin (IPC) ombord i skipet, oppkoblingen er vist på figur 5.2. Ved installasjon termineres ethernetkabler fra thrustersystemenes PLS-er på de grønne termineringsblokkene merket «Thrustertilkoblinger». Videre benyttes vanlige patche-kabler (RJ45) mellom termineringsblokkene og svitsjene, som igjen er koblet til datamaskinen. Mellom PLS og IPC brukes en Modbus-protokoll med relativt begrenset hastighet for henting av data. Dette fører til at det i praksis vil bli benyttet en punktprøvefrekvens på 10 Hz eller lavere ved analysering av PLS-data. Hovedprogrammet takler likevel langt høyere frekvenser, noe som er viktig dersom annet datainnsamlingutstyr kobles til senere (les om forslag til kommersielle løsninger i underkapittel 8.1).

For overføring av data via satellitt er det benyttet en DTAC med RS-232-grensesnitt koblet direkte til IPC (ikke vist på figuren). Det er ønskelig å ha et CAN-grensesnitt i det endelige tilstandsovervåkningssystemet, som beskrevet i oppgaveteksten, derfor må det settes inn en adapter for dette.

På landsiden er det utviklet et program som tar imot, setter sammen pakkedeler, dekoder og lagrer data fra satellittoverføringen.

De tre programmene beskrives mer detaljert i de neste underkapitlene. Kildekoden finnes i de elektroniske vedleggene, se oversiktslisten på side XII.

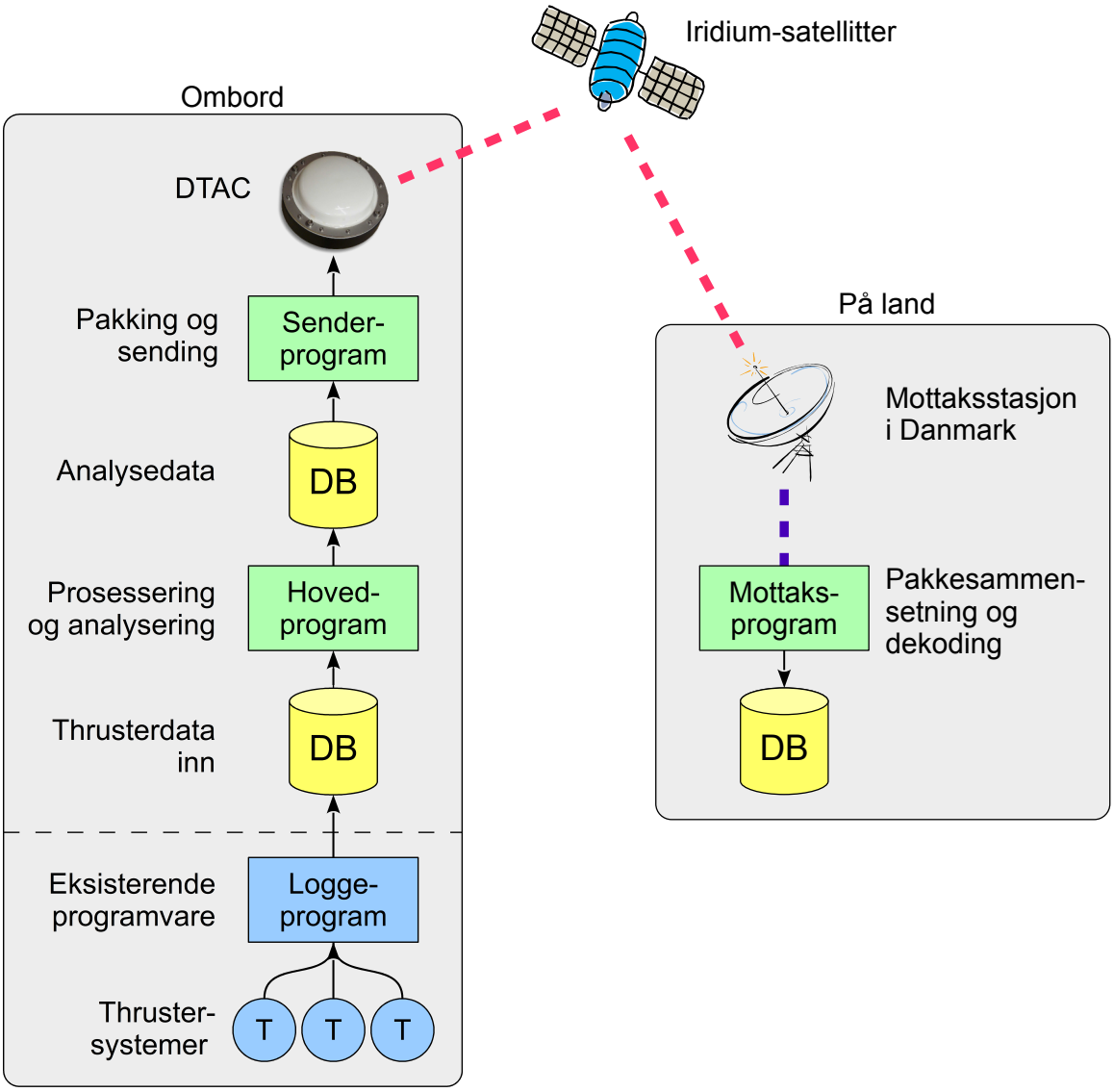
5.1 Dataprosessering og -analyse ombord

Programmet som prosesserer og analyserer data ombord er den viktigste delen av tilstandsovervåkningssystemet, derfor omtales dette som «hovedprogrammet». Oppbygningen til hovedprogrammet er illustrert i figur 5.3. Fra Brunvoll er det satt opp et program som kontinuerlig fyller en database, «thrusterdatabasen», med ønskede måleparametre så lenge thrusteren går. Valg av måleparametre og punktprøvefrekvenser er drøftet i kapittel 8.

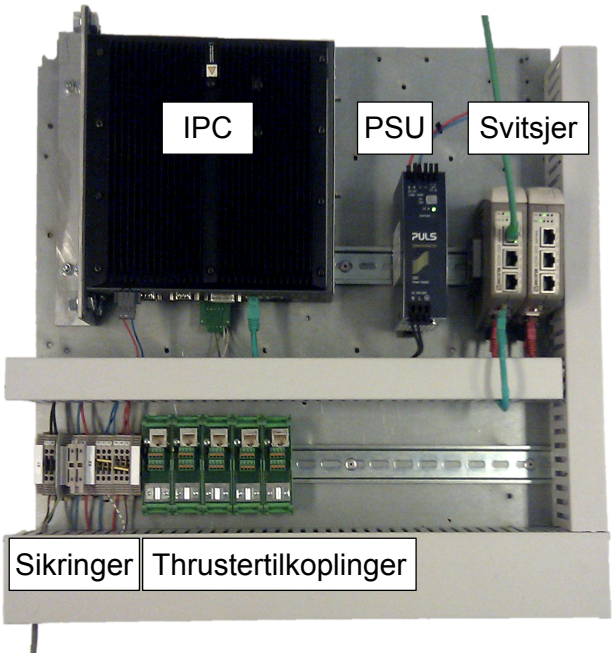
«Signalstier» behandler data fra thrusterdatabasen og lagrer resultatene i en analysedatabase. En signalsti består av én eller flere prosesseringsblokker og én eller flere analyseblokker. På denne måten gjøres programmet modulært og fleksibelt. De neste underkapitlene tar for seg signalstier og blokkene mer detaljert.

Felles for signalstier, prosessering- og analyseblokkene er at de konfigureres ved hjelp av en XML-fil. Antall blokker, blokktyper og antall signalstier er fullstendig valgfritt. Fordelene med å benytte en slik fil er mange: For det første er det ikke nødvendig å kompilere programmene på nytt for å endre enkle ting som f.eks. databasenavn, analysetyper eller antall signaler. For det andre blir alle innstillinger strukturert og samlet på ett sted slik at endringer kan gjøres raskt og oversiktlig. For det tredje er en XML-fil tekstbasert og dermed leselig for mennesker, den er portabel og kan lett utvides eller endres dersom programmene videreutvikles. Oppsettet (regler for elementer, attributter osv.) til en XML-fil defineres ved hjelp av et såkalt XML-skjema, det vil si en XSD-fil. Denne fins også blant de elektroniske vedleggene.

For at programmene skal lese inn nye innstillinger fra XML-filen kreves en omstart. En omstart har ingenting å si for de endelige analyseresultatene og satellittoverføringene så lenge programmene blir avsluttet på riktig måte og databasen med analyseresultater (og mellomregninger) tas vare på.



Figur 5.1: Detaljert blokkdiagram over tilstandsovervåkningssystemet. Eksisterende programvare logger data fra thrusternes PLS-system til en database som hovedprogrammet leser. Videre blir data behandlet, mellomlagret og sendt over satellitt. Hos Brunvoll brukes mottaksprogrammet til å sette sammen, dekode og lagre unna datapakker.

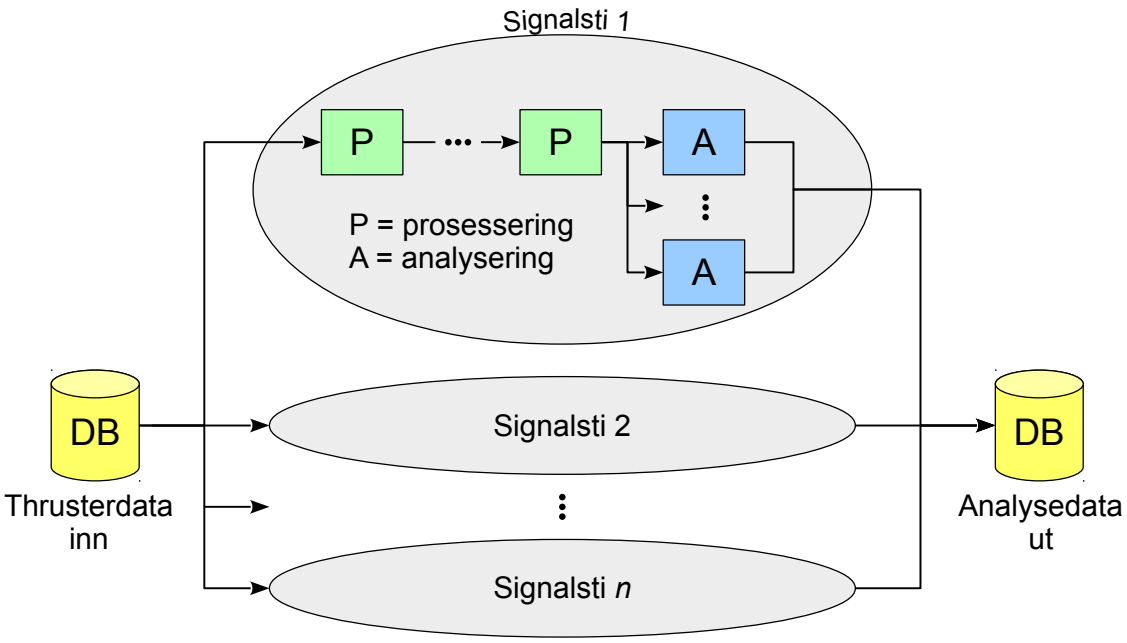


Figur 5.2: Prototype av tilstandsovervåkningssystemet. En industriell datamaskin (IPC) kjører hoved- og senderprogrammet. IPC-en kobles til DTAC-enheten vha. RS-232 eller en CAN-adapter (ikke vist), og til thrustersystemet vha. termineringsblokker og svitsjer.

Databasetypen SQLite er benyttet, denne gir god sikkerhet med hensyn til datatap ved en eventuell programkrasj, operativsystemkrasj eller andre feil. SQLite er basert på transaksjoner med ACID-egenskaper [32], som står for

- *Atomic* – alle eller ingen endringer blir utført. Enhver endring av databasen kan og vil rulles tilbake dersom noe går galt før transaksjonen er ferdig.
- *Consistent* – enhver transaksjon bringer databasen fra en gyldig tilstand til en annen. Dette innebærer også at eventuelle egendefinerte regler må tilfredsstilles (f.eks. at en kolonne med tall skal ha en øvre begrensning).
- *Isolated* – sørger for at flere tråder kan aksessere databasen som om de kjørte etter hverandre.
- *Durable* – tilbakemelding om at en transaksjon var vellykket blir først gitt når systemet er helt sikker på at transaksjonen er ferdig og at endringene er lagret på et ikke-flyktig minne.

I tillegg til prosessering og analysering, vil hovedprogrammet også sette opp såkalte «cronjobber» som bestemmer når senderprogrammet skal kjøre. Cron er en innebygget jobbplanlegger i Linux som kan starte applikasjoner på visse klokkeslett eller ved faste intervaller. Sistnevnte benyttes til å starte senderprogrammet. Oppsett av senderfrekvens er omtalt i 5.2.



Figur 5.3: Signalstier med prosesserings- og analyseblokker i hovedprogrammet. Hver signalsti henter data fra thrusterdatabasen, prosesserer, analyserer og lagrer resultatet i en ny database. Hver sti kjører asynkront, dvs. i sin egen tråd.

5.1.1 Signalstier

Signalstiobjekter, som vist på figur 5.3, henter data fra thrusterdatabasen, prosesserer og analyserer. Analyseresultatene lagres i en databasefil som senderprogrammet senere kan lese fra. Signalstiobjektet er definert med følgende egenskaper:

- Et sett med innstillinger som styrer datainnhenting:
  - Ett inngangssignal gitt ved databasenavn, tabell- og kolonnenavn for oppslag i thrusterdatabasen. Ulike signalstier kan ha samme inngangssignal.
  - En prosesseringsfrekvens, det vil si hvor ofte signalstiobjektet skal hente ny data fra thrusterdatabasen. Prosesseringsfrekvensen må ikke forveksles med signalets punktprøvefrekvens.
- Én eller flere seriekoplede prosesseringsblokker.
- Én eller flere analyseblokker. Alle analyseblokkene er parallellkoplet (felles innputt), etter den siste prosesseringsblokken.

Syntaksen for konfigurasjon av en signalsti i XML-filen følger punktene i foregående liste. Nedenfor er et utdrag av XML-filen som viser dette.

```
<Signal Index="0">
    <SamplingInfo>
        <DbAlias>Thruster 1</DbAlias>
        <TableName>Thruster1Log</TableName>
        <ColumnName>i_ThrustCommand</ColumnName>
        <Period>60</Period>
    </SamplingInfo>
    <Processing>
        ...
    </Processing>
    <Analysis>
        ...
    </Analysis>
</Signal>
```

Elementet <Period>, som definerer prosesseringsfrekvensen, angis i minutter (som et desimaltall). Databasenavnet er implisitt gitt ved et alias for å gjøre XML-filen mer oversiktlig. Alle databasealias defineres i en egen, global <SqlSettings>-blokk som følger.

```
<SqlSettings>
    <DbAlias Alias="Thruster 1" Path ="/home/datalog/↩
       thruster_1_log.db" />
</SqlSettings>
```

Dette kan virke noe tungvint, men det vil lette videre utviklingsarbeid betraktelig dersom det blir nødvendig å legge inn andre elementer eller attributter tilhørende et databasealias. Dette kan f.eks. være en attributt som sier om programmet skal ha skrive- eller bare lesetilgang til databasen.

Hvert signalstiobjekt er programmert slik at det kjører i en egen tråd. Dette gjør det enklere å hanskes med ulike punktprøve-/prosesseringsfrekvenser siden det muliggjør parallellitet. I tillegg fører det til bedre utnyttelse av ressurser på flerkjernede prosessorer.

Elementene <Processing> og <Analysis> inneholder de ulike blokkene som vist på figur 5.3. Disse er beskrevet nærmere i de neste underkapitlene.

5.1.2 Prosesseringsblokker

Prosesseringsblokkene behandler data før analyse. De har alltid én inngang og én utgang hvor det sendes en datavektor, og en tidsvektor hvis nødvendig. Blokken behøver ikke å returnere det samme antall datapunkter som den får inn.

Videre beskrives de fire blokktypene som er implementert: IIR-filter, derivasjon, RMS og absoluttverdi. Syntaksen for konfigurasjonsparametre i XML-filen blir også spesifisert. Felles for alle er at hver enkelt har en indeks-parameter som gir rekkefølgen når signalstiobjektet syr sammen blokkene. Indeks-parametrene må være en kontinuerlig følge $0, 1, \ldots, r$.

IIR-filter

Filtreringsblokken er implementert som et IIR-filter på *Direct Form II*, slik det er beskrevet i teoridelen, underkapittel 3.2.1. Filteret konfigureres i XML med koeffisientvektorene $\boldsymbol{a}$ og $\boldsymbol{b}$ på følgende måte:

```
<Filter Index="0">
    <Coeff_b Index="0">0.3333</Coeff_b>
    <Coeff_a Index="0">1.2</Coeff_a>
    <Coeff_a Index="1">-0.5</Coeff_a>
</Filter>
```

Antall elementer `<Coeff_a>` og `<Coeff_b>` er valgfritt, etter hvor mange poler a_j og nullpunkter b_i det er ønskelig at filteret skal ha. Indeks-attributtene for de to arrayene må være kontinuerlige sekvenser av naturlige tall som starter på null. Som det går frem av filterlikningen (3.11), må koeffisient $a_0 \neq 0$. Dersom a_0 settes lik null, vil programmet sette $a_0 = 1$. Det er ingen slike begrensninger på `<Coeff_b>`. I programmet lagres koeffisientene som 64 bits `double`-verdier for å minimere problemer med kvantifisering.

Design av IIR-filterkoeffisienter kan gjøres med en rekke elektroniske verktøy, f.eks. Matlab, Maple og LabVIEW. Dessuten finnes det nettsider som [9] der filtre kan designes online.

Dervasjon

For å lette konfigureringsarbeidet, er det implementert en førstegrads derivasjonsblokk som et spesialtilfelle av IIR-filteret. Koeffisientene til IIR-filteret er $b_0 = 1$, $b_1 = -1$ og $a_0 = T_s$, som utledningen i underkapittel 3.2.2 viste. Punktprøveperioden T_s og en eventuell initialisering av derivatoren oppgis i konfigurasjonsfilen som følger på neste side.

```
<Derivator Index="1">
    <SampleTime>0.1</SampleTime>
    <InitState>100</InitState>
</Derivator>
```

Initialiseringen i XML-eksemplet over fører til at derivatoren returnerer 100 når det første datapunktet behandles. Dersom initaliseringelementet sløyfes, vil derivatoren returnere null etter første datapunkt.

Effektivverdi (RMS)

Effektivverdiberegninger er implementert med formel (3.21) som beskrevet i underkapittel 3.4, litt modifisert. Vindusstørrelse m (antall punkter) konfigureres med XML-filen slik:

```
<RMS Index="2">
    <WindowSize>30</WindowSize>
</RMS>
```

Initielt vil RMS-blokken returnere et n-punkts RMS-signal helt til antall punkter n blir større enn vindusstørrelsen m, altså

$$x_{n,\mathrm{RMS}} = \begin{cases} \sqrt{\frac{1}{n}\sum_{i=0}^{n} x_i^2} & n < m \\ \sqrt{\frac{1}{m}\sum_{i=n-m}^{n} x_i^2} & n \geq m \end{cases} \tag{5.1}$$

Siden m er antatt relativt liten, er disse formlene implementert direkte uten noen av teknikkene beskrevet i statistiskkapitlet 3.1. Fra (5.1) er det klart at RMS-blokken returnerer like mange datapunkter som den får inn.

Absoluttverdi

Noen ganger kan det være nødvendig å finne absoluttverdien til parametre som skal analyseres. Til dette er det implementert en egen blokk, som konfigureres ganske enkelt med

```
<AbsVal Index="3" />
```

Alt denne blokken gjør er å returnere absoluttverdien til data som sendes til den. Antall datapunkter som returneres er alltid likt antall datapunkter som kommer inn.

5.1.3 Analyseblokker

Det er laget tre typer analyseblokker: statistikk, histogram og FFT. Felles for disse er at de alle blir parallellkoblet etter den siste prosesseringsblokken og de lagrer sine resultater i en egen database. Som i forrige underkapittel, vil XML-syntaksen spesifiseres for hver enkelt blokk.

Statistikk

Beregning av de statistiske egenskapene minimum, maksimum, gjennomsnitt og varians er implementert med formlene og metodene som ble presentert i underkapittel 3.1. Denne blokken behøver ingen konfigurasjon i XML-filen annet enn `<Statistics />`, men likevel er følgende elementer definert i XSD-spesifikasjonen:

```
<Statistics TxSchemeIndex="0">
    <Min>true</Min>
    <Max>false</Max>
    <Mean>true</Mean>
    <Variance>true</Variance>
    <Count>true</Count>
</Statistics>
```

Dette er gjort for i fremtiden å kunne bestemme hvilke statistiske parametre som er ønskelig å sende, og på den måte kunne redusere datamengden som overføres. Foreløpig er denne funksjonaliteten ikke implementert. Slik programmet er nå, sendes de fire nevnte parametrene i tillegg til antall datapunkter (`<Count>`).

Attributtet `TxSchemeIndex="0"`, som alle analyser skal ha, betyr at denne analysen hører til senderskjema 0. Et senderskjema definerer hvilke analyseresultater som skal sendes samtidig over satellitt. Dette er beskrevet nærmere i underkapittel 5.2.

Histogram

Histogramblokken generer et histogram over innputt-data som beskrevet i underkapittel 3.3. Klassebredder og klassegrenser er konfigurerbare i XML. To ulike konfigurasjonsalternativer kan benyttes. Den første brukes når alle klassene skal være like brie, og er vist på neste side.

```
<Histogram TxSchemeIndex="1">
    <BoundLower>-100.0</BoundLower>
    <BoundUpper>100.0</BoundUpper>
    <BinWidth>20</BinWidth>
    <EqualBinWidths>true</EqualBinWidths>
</Histogram>
```

Med denne konfigurasjonen vil programmet opprette et histogramobjekt som dekker området fra -100 til 100 med klasser som alle er 20 enheter breie. Antall klasser n blir beregnet etter

$$n = \left\lfloor \frac{\texttt{BoundUpper} - \texttt{BoundLower}}{\texttt{BinWidth}} \right\rfloor$$

der $\lfloor \cdot \rfloor$ betyr avrunding nedover[4] til nærmeste heltall. Følger vi konfigurasjonseksempelet, vil antall klasser bli $n = 10$. Etter at n er beregnet, redefineres øvre grense slik at

$$\texttt{BoundUpper} = \texttt{BoundLower} + n \cdot \texttt{BinWidth}$$

Dette gjør at programmet takler en «slurvete» XML-konfigurasjon der øvre grense ikke stemmer overens med et helt antall klasser med lik bredde. I eksemplet forblir øvre grense uforandret.

Den andre måten å konfigurere histogramblokken på brukes dersom egendefinerte klassegrenser (og -bredder) ønskes. Dette gjøres som vist nedenfor.

```
<Histogram TxSchemeIndex="1">
    <BoundLower>-100.0</BoundLower>
    <EqualBinWidths>false</EqualBinWidths>
    <binLimitsUpper>-80.0</binLimitsUpper>
    <binLimitsUpper>-50.0</binLimitsUpper>
    <binLimitsUpper>0.0</binLimitsUpper>
    <binLimitsUpper>50.0</binLimitsUpper>
</Histogram>
```

Programmet vil automatisk sorterer klassegrensene etter verdi og bruke den høyeste som øvre grense for histogrammet. I dette tilfellet vil histogrammet dekke området fra -100 til 50 og bestå av $n = 4$ klasser med forskjellige bredder.

[4]For å få det matematisk korrekt burde det ha stått $\text{sgn}(\cdot)\lfloor|\cdot|\rfloor$ som betyr «kutt alle desimaler» eller avrunding mot null, men siden $\texttt{BoundUpper} > \texttt{Boundlower}$ og antall klasser $n > 0$ går avrunding nedover for det samme.

Konvensjonen fra-og-med nedre grense og opp-til øvre grense $[a, b\rangle$ er benyttet uansett konfigurasjon, slik det ble nevnt i 3.3. Metoder for automatisk beregning av alle statistikkparametre definert i underkapittel 3.3.1 er også implementert. Disse er ikke benyttet ombord, men kan være greie å ha på land når histogrammet skal tolkes.

FFT

FFT-blokken returner absoluttverdien til det komplekse fourierspekteret $X(k)$ som tidligere beskrevet i 4.1.2, i tillegg til gjennomsnittspektre (inkoherent beregningsmetode) etter flere kjøringer. Den underliggende algoritmen som er benyttet er en Cooley–Tukey-variant for C++, hentet fra [4]. Algoritmen er listet nedenfor. Her er det enkelt å kjenne igjen den rekursive strukturen og formel (4.10) som ble utledet og forklart i 4.1.2.

```
// Cooley-Tukey FFT (in-place)
void FFTAnalyzer::fft(CArray& X)
{
    const size_t N = X.size();
    if (N <= 1) return;

    // divide
    CArray even = X[std::slice(0, N/2, 2)];
    CArray  odd = X[std::slice(1, N/2, 2)];

    // conquer
    fft(even);
    fft(odd);

    // combine
    for (size_t k = 0; k < N/2; ++k)
    {
        Complex W = std::polar(1, -2*PI*k/N) * odd[k];
        X[k    ] = even[k] + W;
        X[k+N/2] = even[k] - W;
    }
}
```

Konfigurasjon av en FFT-blokk gjøres slik:

```
<FFT TxSchemeIndex="2">
    <Samples>1024</Samples>
    <Window>blackman-harris</Window>
    <Overlap>0</Overlap>
    <Averages>4</Averages>
    <TxLower>20</TxLower>
    <TxUpper>150</TxUpper>
</FFT>
```

Antall punkter N per vindu, gitt ved <Samples>-elementet, behøver ikke å være en toerpotens. Padding av nullelementer blir brukt internt i programmet dersom det er nødvendig, etter vindusfunksjonen.

Vinduselementet kan være en av tekststrengene `rect`, `none`, `hann`, `hamming`, `blackman`, `gaussian`, `welch`, `triangular`, `barlett`, `flattop` eller `blackman-harris`. Dersom elementet utelates, blir det brukt et rektangulært vindu. Overlapping av data mellom vinduer er foreløpig ikke implementert, men <Overlap> er definert i XSD for senere utvidelse.

Overlapping av FFT-spektere for gjennomsnittsberegning er hardkodet til full overlapping, det vil si at hvert nytt FFT-spekter inn vil produsere et nytt gjennomsnitsspekter ut. Antall spektre som brukes i gjennomsnittsberegningen bestemmes av <Averages>-elementet.

Et FFT-spekter blir ofte dominert av enkelte peaker, som nevnt i teoridelen. Ved overføring av spekteret til land kan derfor en unødvendig stor datamengde brukes på å overføre verdier som er null eller tilnærmet lik null. Dette gjelder særlig hvis antall punktprøver per vindu er stort, slik at det korresponderende spekteret også blir stort. Hvis det er ønskelig å redusere datamengden som overføres, brukes de valgfrie elementene <TxLower> og <TxUpper>. Før sending lukes uønskede frekvensområder ut, slik at bare data $X(k)$ for $\texttt{TxLower} \leq k \leq \texttt{TxUpper}$ komprimeres og sendes, medregnet eventuell padding. På denne måten kan det *zoomes* inn på interessante områder i spekteret uten at det går på bekostning av oppløsningen. Hele spekteret sendes dersom de to nevnte elementene utelates.

5.2 Dataoveføring via satellitt

For å sende data fra skip til land, er det benyttet en DTAC-enhet, som tidligere nevnt. Enheten kobles til IPC-en over RS-232 og styres ved hjelp av tekstkommandoer basert på NMEA.

DTAC-enheten gjør det mulig å kommunisere over satellittnettverket Iridium. Dette nettverket består av 66 satellitter som går i lav jordbane, noe som gir fullstendig dekning over hele kloden. Abonnementet på Iridiums tjenester tillater overføring av en datamengde på 12 KiB per måned for omlag 20 USD. En så liten datamengde understreker viktigheten av å gjøre mest mulig analyse ombord og bare sende de mest nødvendige resultatene til land. Det er også mulig å sende data i motsatt retning, fra land til DTAC, men denne funksjonen er ikke benyttet i prosjektet.

Et eget program som startes av den innebygde jobbplanleggeren «Cron» i Linux, sender data til land via DTAC. Hvilke analyseresultater som skal sendes og når de skal sendes, bestemmes av såkalte *senderskjema*, som tidligere nevnt. Konfigurasjon av senderskjema gjøres i den samme XML-filen som hovedprogrammet. Et eksempel på konfigurasjon er:

```
<TxScheme Index="0">
    <Frequency>hourly</Frequency>
</TxScheme>
<TxScheme Index="1">
    <Frequency>daily</Frequency>
</TxScheme>
<TxScheme Index="2">
    <Frequency>hourly</Frequency>
</TxScheme>
```

Frekvenselementene bestemmer hvor ofte sendingen skal skje, og må være en av tekststrengene `hourly`, `daily`, `weekly`, `monthly` eller `yearly`. Det er viktig å merke seg at når programmet sender, vil alle analyseobjektene i det aktuelle senderskjemaet starte sin analyse på nytt. Dette betyr at frekvenselementene også bestemmer hvor lang tid det går mellom hver gang analyseresultatene nulles ut, uavhengig av hvor mye, eller lite, thrusteren har vært brukt i løpet av perioden.

For kommunikasjon mellom IPC og DTAC brukes en NMEA-protokoll definert av Powex i [8] (ikke offentlig). I dette prosjektet benyttes en kommando for overføring av binær informasjon. Denne kommandoen setter ingen krav til strukturering av data eller innhold, alt blir videresendt over Iridium-nettverket. En begrensning er imidlertid at bare 52 byte[5] kan overføres i gangen, noe som gjør at oppdeling av data i pakker er nødvendig.

5.2.1 Pakking av data

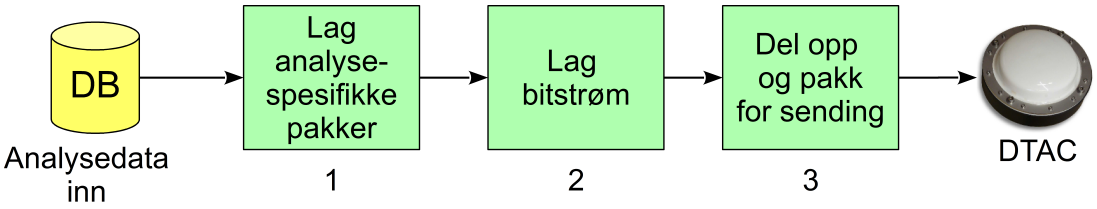
Vedlegg B definerer protokollen som er utviklet for pakking og identifisering av data for sending. Vedlegget er skrevet på engelsk for å enklere samsvare med kildekoden. Pakkingen skjer i tre steg som er illustrert i figur 5.4, og forklart i den følgende listen.

1. Data hentes fra databasen, skaleres (se neste underkapittel) og pakkes i *datapakker*. Hver datapakketype er spesifikk til hvilke analysedata den inneholder: statistikk, histogram eller FFT. Alle datapakkene har en identifikasjonsbyte som sier hvilken analysetype dataene tilhører og en lengdebyte som forteller hvor lang pakken er.
2. Datapakkene settes sammen til et kontinuerlig array med byteverdier.
3. Arrayet splittes opp i nye *overføringspakker* etter 52-bytegrensen. Den første pakken (*startpakken*) inneholder to ekstra headerbyte: en generell pakkeidentifiaksjonsbyte `0xAA` som forteller at dette er starten på en ny overføring av data og én byte som sier hvor mange pakkdeler overføringen består av. Resten av pakkene (*delpakkene* eller *subpakkene*) starter med `0xBB`, pakkenummer, skjemanummer og datalengde. Se vedlegg B.

Ved å pakke data med denne protokollen er det ikke mulighet for feiltolkning på landsiden når data skal pakkes ut igjen. Det er antatt at det eksisterer en eksakt kopi av XML-filene til hvert enkelt skip hos mottakeren. Når overføringspakkene settes sammen, kan de opprinnelige analysespesifikke datapakkene fra første punkt i listen hentes ut.

Dersom flere senderskjema sender «samtidig», kan mottakeren sortere pakkene ved hjelp av byten som forteller hvilket skjema de tilhører. Om ulike skip sender data samtidig, vil mottakeren fortsatt kunne skille pakkene fra hverandre. Dette er enkelt, fordi Iridium identifiserer hvilken DTAC-enhet, og dermed hvilket skip, pakkene er sendt fra.

[5] Powex har varslet en oppdatering av programvaren i DTAC som skal doble antall byte per binærmelding, slik at 104 byte kan sendes på en gang.



Figur 5.4: Blokkdiagram for sending av data. Analysedata hentes fra databasen som hovedprogrammet skriver til. Data pakkes som vist på figuren, før den sendes til DTAC og videre over satellitt til land.

5.2.2 Komprimering ved skalering av data

Når vi har en overføringsbegrensning på 12 KiB per måned, er det klart at komprimering er nødvendig. En typisk sendingsfrekvens antas å være én gang per uke. Da blir maksimal overført datamengde 2,7 KiB per sending. Det viser seg at de fleste av de kjente komprimeringsalgoritmene ikke nytter på binærfiler under ca. 4 KiB med tilfeldig data.

På grunn av dette er det lagt vekt på skalering og riktig seleksjon av data for å minimere overført datamengde i stedet. Skalering av analyseresultater blir beskrevet i de neste underkapitlene. Dette går under første steg i figur 5.4 og den tilhørende listen på forrige side. For hver analyse blir det presentert hvor mange byte nyttedata som sendes per pakke, dette er også oppsummert i tabell 5.1, for å gi en bedre oversikt.

Tabell 5.1: Oppsummering av antall byte med nyttedata i hver analysedatapakke.

Analyse	Antall byte nyttedata	Forklaring
Statistikk	$4+4+4+4+4=20$	Fire flyttal og ett heltall som alle er 4 byte: varians, snitt, minimum, maksimum og antall punkter.
Histogram	$1+k+n$	1: lengden til $\max\{f_i\}$. k: verdien til $\max\{f_i\}$. n: antall klasser.
FFT	$4+$ `TxUpper` $-$ `TxLower` eller $4+N$	4: verdien til $\max\{\lvert X(k)\rvert\}$. `TxUpper` $-$ `TxLower`: antall linjer som skal overføres, spesifisert i XML. N: antall datapunkter per vindu.

Statistikk

Statistikkparametrene er fire flyttall, i tillegg til et positivt heltall som teller antall datapunkter. Til utregninger internt i programmet er det benyttet henholdsvis `double`-verdier og en `uint`.

I C++, som følger IEEE 754-standarden, består en `double`-verdi av 64 bit. Dette gir en presisjon på 15–17 desimaler og en varierende oppløsning. Oppløsningen er best når tallet er nær null, som nevnt i kapitlet for deskriptiv statistikk. Før sending blir `double`-verdiene kastet til `float`, slik at de bare tar 32 bit hver. Dette reduserer presisjonen til 6–9 desimaler, noe som er akseptabelt for tilstandsovervåkningssystemet.

Pakken med statistikkdata består derfor (alltid) av 20 databyte: fire flyttall á fire byte og et heltall som også er fire byte. Nøyaktig spesifikasjon med nødvendige header-byte finnes i B.2.1, se vedlegg.

Histogram

For klasseinndeling av data benyttes et array med 32 bits `uint`-verdier, ett element per klasse. Hvert element teller antall datapunkter som havner innenfor klassen (det vil si frekvensen).

Siden det kan være relativt mange klasser i et histogram, er det besluttet å skalere data ned til byteverdier før sending. For å utnytte verdiområdet til en byte fullt ut, skaleres alle frekvensene i forhold til den høyeste frekvensen. Frekvenesene f_i skaleres til byteverdier b_i som følger.

$$b_i = \left\lfloor \frac{f_i}{\max\{f_i\}} \cdot 255 \right\rceil \quad \text{der} \quad i = 1, 2, \ldots, n \tag{5.2}$$

Her betyr $\lfloor \cdot \rceil$ avrunding til nærmeste heltall på vanlig måte og n er antall klasser. En åpenbar ulempe med dette er at vi mister en absolutt referanse, slik at data ikke kan skaleres opp igjen på mottakersiden. Dette er ikke noe problem dersom et rent prosentvis histogram ønskes, det vil si uten noen benevnelse på andreaksen.

Ofte er et prosentvis histogram ikke godt nok. For å unngå dette problemet overføres derfor $\max\{f_i\}$ uskalert i tillegg til alle b_i. Mottakersiden kan nå gjenskape estimater $\hat{f}_i$ av de opprinnelige frekvensene ved

$$\hat{f}_i = \frac{b_i \max\{f_i\}}{255} \quad \text{der} \quad i = 1, 2, \ldots, n \tag{5.3}$$

Estimatene har fortsatt en oppløsning på bare 256 nivåer, men de har en absolutt referanse.

Siden $\max\{f_i\}$ har et relativt stort verdiområde (fra 0 til $2^{32}-1$), er det stor sannsynlighet for at dette tallet kan representeres med mindre enn fire byte. Dette er utnyttet ved å sende en lengdebyte $k \in [1,4]$ fulgt av k byte med selve verdien til $\max\{f_i\}$. I beste tilfelle sendes derfor to byte, mens i verste tilfelle sendes fem.

Oppsummert betyr dette at histogrampakkene består av n byte med skalert data, én byte k som sier hvor mange byte $\max\{f_i\}$ opptar og k byte for $\max\{f_i\}$ selv. Totalt blir dette $n+1+k$ byte med data i tillegg til headerinformasjon. Histogrampakkene er fullstendig definert i B.2.2.

FFT

Til å komprimere FFT-data før overføring er nesten samme metode som for histogramdata brukt, forskjellen er at vi nå jobber med flyttall. Alle operasjoner internt i programmet er gjort med 64 bits `double`-verdier. Det er valgt å redusere antall bit til en fjerdedel og samtidig gjøre om verdiene til heltall før sending. Hver linje i spekteret representeres dermed med 16 bit. De nedskalerte verdiene $B(k)$ beregnes etter

$$B(k) = \left\lfloor \frac{|X(k)|}{\max\{|X(k)|\}} \cdot (2^{16}-1) \right\rceil \quad \text{der} \quad k = 0, 1, \ldots, \frac{N}{2} - 1 \tag{5.4}$$

Når $B(k)$ overføres, får mottakeren det samme problemet som ved histogramdata; en absolutt referanse mangler. For å løse dette overføres $|X(k)|$ uskalert i tillegg til $B(k)$. På samme måte som for statistikkpakkene, er en oppløsning på fire byte godt nok, derfor kastes $|X(k)|$ til en `float`-variabel (som har konstant lengde, slik at en lengdebyte ikke er nødvendig). Nå kan mottakeren gjenskape estimater av $X(k)$ slik:

$$\left|\hat{X}(k)\right| = \frac{B(k)\max\{|X(k)|\}}{2^{16}-1} \quad \text{der} \quad k = 0, 1, \ldots, \frac{N}{2} - 1 \tag{5.5}$$

Husk at bare data $B(k)$ for $\texttt{TxLower} \leq k \leq \texttt{TxUpper}$ sendes dersom dette er spesifisert i XML-filen. Tilsammen består FFT-pakkene i så tilfelle av $(\texttt{TxUpper} - \texttt{TxLower})/2$ à to byte med absoluttverdier av FFT-spekteret, pluss fire byte for $\max\{|X(k)|\}$. Totalt $\texttt{TxUpper} - \texttt{TxLower} + 4$ eller $N+4$ databyte dersom grensene er utelatt. Se underkapittel B.2.3 i vedleggene.

5.3 Programvare på land

Lokalt hos Brunvoll må datapakkene tas imot, pakkes ut, presenteres og lagres på et søkbart format. Vedlegg D viser at Powex leverer et eget nettbasert presentasjonsverktøy for DTAC-data. Det er valgt å ikke ta dette i bruk, grunnet

relativt begrensede muligheter med hensyn til analyser, søkbarhet og lagring. Systemet ville også kreve en god del tilpasning til Brunvolls behov, fordi det i utgangspunktet er laget for presentasjon av data fra dumpere. Protokollen for overføring (vedlegg B) er ikke kompatibel med nettløsningen fra Powex.

Ved å utvikle egne verktøy for utpakking, presentasjon og lagring internt i Brunvoll, oppnås større grad av fleksibilitet, f.eks. til videreutvikling uten å måtte involvere eksterne aktører (som Powex).

5.3.1 Utpakking og lagring

For dekoding/utpakking og lagring av data er det utviklet et «mottaksprogram». Programmet kjøres hver gang en ny pakke er mottatt. Utpakkingsprosessen er det samme som å utføre trinnene i figur 5.4 baklengs.

Når en ny pakke kommer inn, henter programmet ut mest mulig informasjon og lagrer dette i en database for enkelt oppslag senere. Dersom det er en startpakke (begynner med `0xAA`), identifiseres følgende parametre: senderskjema, antall delpakker, pakkenummer (0) og datalengde. Hvis det er en delpakke (begynner med `0xBB`), identifiseres senderskjema, pakkenummer og datalengde.

Hver gang en ny pakke legges inn av programmet, vil det bli undersøkt (ved hjelp av databasen) om alle delpakker for det aktuelle senderskjemaet er ankommet. Om så er tilfelle, starter sammenslåingen og videre utpakking av alle delpakkene. Først fjernes headerbytene for overføringspakkene. Deretter blir resterende data slått sammen til et sammenhengende array. Vi er nå tilbake i steg to i figur 5.4 og den tilhørende listen.

Nå er det bruk for lengdeparametrene i datapakkene. Ved hjelp av disse deles arrayet opp på nytt, slik at vi sitter igjen med de opprinnelige datapakkene. Til slutt lagres datapakkene i en ny tabell, og delpakkene fra den første tabellen merkes som dekodet.

5.3.2 Presentasjonsverktøy

Innenfor tidsrammene til prosjektet ble det ikke rom for utvikling av et presentasjonsverktøy. Men som forklart i forrige avsnitt, er all data dekodet og lagret på en søkbar måte, slik at det skal være relativt enkelt å implementere en presentasjonsfunksjon i fremtiden.

Det som gjenstår er å lage metoder i analyseklassene som kan opprette objekter ved hjelp av datapakkene. Deretter må metoder for presentasjon utvikles. Mer om dette finnes under forslag til videre arbeid i underkapittel 8.2.

Kapittel 6

Valg av parametre og analyser

Riktig valg av måleparametre og analyser kan være avgjørende for å tolke thrustersystemets tilstand riktig. Her presenteres det derfor forslag til kombinasjoner av parametre, prosesserings- og analyseblokker for deteksjon av ulike feilsitusajoner.

6.1 Realiserbart slik systemet er nå

Analyseoppsettene listet i dette underkapitlet er realiserbare slik tilstandsovervåkningssystemet er på nåværende tidspunkt. Det eneste som kreves er oppsett av logging av de foreslåtte parametrene i programmet som fyller thrusterdatabasen.

1. **Gjennomsnittlig pitchvinkelhastighet.** Ved å derivere tilbakemeldingssignalet for pitchvinkel og deretter kjøre det gjennom en statistikkblokk, beregnes den gjennomsnittlige pitchvinkelhastigheten. Dette er en viktig størrelse som blir direkte påvirket av slitasje i mekanikken for vridning av propellbladene.
2. **Histogram over thrusterens turtall.** Rotasjonshastigheten på histogramform gir et godt bilde av det typiske bruksmønsteret til thrustersystemet. Her kan det f.eks. avdekkes om thrusteren ligger på et høyt turtall ofte, noe som kan føre til større slitasje. Dette kan tyde på enten feil bruk, eller at thrusteren er for liten. Turtallshistogrammet har også andre bruksområder enn tilstandsovervåking: Ved videreutvikling av thrustersystemene kan det tas hensyn til bruksmønsteret, f.eks. ved å optimalisere og beskytte mot vibrasjon rundt de mest brukte turtallsområdene.

   Dårlig lastfordeling mellom thrustere kan avdekkes dersom det settes opp histogrammer for thrustere montert ved siden av hverandre. Det er f.eks. ofte montert flere tunnelthrustere i baugen på mange skip.

3. **Histogram over thrusterens belastning.** Denne analysen kan stort sett avdekke det samme som i det forrige punktet. Forskjellen er at motorbelastningen (målt i watt) også kan benyttes på thrustere med konstant rotasjonshastighet, hvor bare pitch- og eventuell asimutvinkel varierer.
4. **Temperaturendring i motor.** Statistiske parametre for *endring* i temperatur i en motor kan fortelle noe om slitasje i lagre, groe på propellblad eller andre forhold som øker temperaturraten. Spesielt viktig er maksimumsverdien til raten – hvor fort øker temperaturen? Prosesseringen gjøres vha. en derivasjonsblokk, lavpassfilter og en statistikkblokk. Trending over lang tid gjøres på landsiden, hvor det er «ubegrenset» med lagringskapasitet og prosesseringsressurser. En maksimumsverdi som over tid stiger, kan tyde på at et serviceopphold er nødvendig.
5. **Spektrumsanalyse av pitch- og/eller asimuthvinkel.** Spesielt når skipet styres av DP-systemet i skipet, endres pitch- og eventuell asimutvinkel raskt. Brunvoll leverer ikke DP-systemer selv, men lar andre produsenter styre thrusterene. Dersom DP-systemet er dårlig justert, kan det prøve å regulere på bølgefrekvens, noe som øker slitasjen på thrusterene kraftig. Ved å utføre en FFT-analyse av pitch- og/eller asimutvinkelens settpunkt kan eventuelle bølgefrekvenser detekteres. Dermed er en viktig årsak til økt slitasje identifisert, og det kan være grunn til å ta saken videre til leverandøren av DP-systemet.
6. **Trending av gjennomsnittlig giroljetemperatur.** Ved å benytte en statistikkblokk kan den gjennomsnittlige giroljetemperaturen identifiseres og sendes til land. På samme måte som i punkt fire, trendes temperaturen over tid slik at trege endringer kan oppdages. Økt oljetemperatur kan bety at giret eller lagre er slitt, og skipet må inn til et serviceopphold.
7. **Trending av hydraulikkoljenivå i gravitasjonstank.** De fleste thrusteranlegg levert av Brunvoll til nå har hatt digitale (av/på-) givere som kontrollerer om nivået i gravitasjonstanken er for lavt eller ikke. Neste generasjon thrustere skal utstyres med analoge givere som kan anslå det nøyaktige nivået, dette kan utnyttes i tilstandsovervåkningssystemet. Et brått fallende oljenivå kan indikere at en lekkasje har oppstått. Nivået kan overvåkes ved å beregne og overføre statistikkparametre, spesielt gjennomsnittet. Siden olje benyttes til tetning av thrusteren, er det normalt med en viss lekkasje. Dette må tas i betraktning når resultatene skal tolkes.

6.2 Realiserbart etter videreutvikling

Forslagene i den følgende listen krever videreutvikling av tilstandsovervåkningssystemet på ulike måter, f.eks. henting av data fra andre kilder enn PLS.

8. **Spektrumsanalyse eller treding av RMS-nivå for vibrasjonsmålinger.** Ved å ha fastmonterte akselerometer på ulike posisjoner og i ulike retninger på thruseren, kan vibrasjonsnivået overvåkes. Dette krever i de fleste tilfeller høyhastighetsmålinger, som i praksis betyr at systemet må ha dedikert maskinvare for datainnsamling. Et eksempel på egnet maskinvare er National Instruments' CompactRIO-serie med tilhørende vibrasjonsmålingsmoduler, mer om disse finnes i underkapittel 8.1.3. Aktuelle analyser er FFT, som eventuelt kan regnes om til PSD ved mottak, eller statistikk over vibrasjonsnivået uttrykt ved effektivverdien. En økning i vibrasjonsnivået kan skyldes slitte lager, som igjen kan føre til unødvendig slitasje på andre komponenter i thrusteren.
9. **Trending av relativ fuktighet i hydraulikkolje.** Skulle det oppstå lekkasje i thrusteren, kan det observeres med et instrument som måler relativ fuktighet i hydraulikkoljen. En hurtig økning i vanninnholdet tyder på at en lekkasje har oppstått. Lekkasjer vil i de aller fleste tilfeller kreve service. Ved å bruke loggesystemet kan slike hendelser fanges opp før det er «for sent». Målingen krever en oljeanalysator, som ikke er levert som standard i dag.
10. **Partikkelinnhold i girolje.** Økt antall partikler (f.eks. jern og kobber) tyder på høy friksjon grunnet slitte lagre eller gir. En partikkelteller kan analysere giroljen og resultatene kan klassifiseres i henhold til f.eks. NAS eller ulike ISO-standarder. Standardene definerer klassene ut fra antall og størrelsen på partiklene i et visst volum med olje. Ofte er også oljens temperatur korrelert med partikkelinnholdet.
11. **Trending av hydraulikkoljetrykk.** Dersom oljetrykket ved f.eks. pitchvinkelendring øker over tid, betyr det at stadig mer kraft trengs for å vri propellbladene. Dette betyr igjen slitasje, groe, manglende smøring eller andre forhold.
12. **Parametre i matematiske modeller.** Systemet vil kunne overvåke mye mer kompliserte parametre ved å implementere matematiske modeller som beskriver thrusterdynamikken. Eksempelvis kan en modell av thrusteren brukes til å måle dødtid og tidskonstant ved endring av pitchvinkel.

Kapittel 7

Praktiske resultater

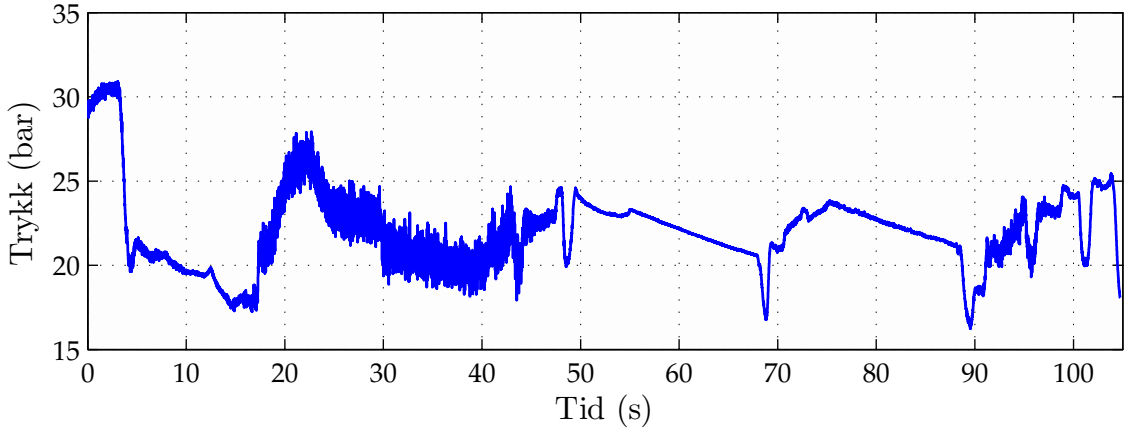
Dette kapitlet presenterer resultater etter praktisk utprøving av tilstandsovervåkningssystemet. Resultatene sammenliknes med teoretiske «ideelle» svar fra Matlab og drøftes fortløpende. Ulike måleserier er matet inn i programmene for å verifisere at de fungerer riktig. Måleseriene kommer fra reelle thrustersystemer eller en simulator. De reelle målingene er eksisterende, statiske databaser med data fra ulike skip og thruster-typer. Simulatormålingene genereres samtidig som tilstandsovervåkningssystemet kjører, slik den normale situasjonen blir når systemet er installert på et skip.

Alle delene av tilstandsovervåkningssystemet er utprøvd: innsamling, prosessering, analysering, pakking, sending, mottak og til slutt dekoding og lagring. Matlab er benyttet til å generere plottene i kapitlet, siden et eget presentasjonsverktøy ikke er utviklet.

7.1 Oljetrykkmålinger

Figur 7.1 viser en reell måleserie over trykket i hydraulikkolje for endring av pitchvinkel, målt i rør «S1» (se figur 2.1), i en thruster ombord i cruiseskipet MV «Norwegian Breakaway» fra 18.03.2013. Thrusteren er av typen FU-115-LTC-3000-3000kW, dvs. en tunnelthruster med vertikalmontert motor, regulerbar pitchvinkel, propelldiameter 3 meter og en effekt på 3 MW. Målingene ble gjort med en HBM Spider 8-enhet med egnede trykkgivere. Punktprøvefrekvensen er $f_s = 400\,\text{Hz}$.

Denne tidsserien blir videre brukt til å vise hvordan IIR-filteret og histogramblokken fungerer i praksis.



Figur 7.1: Oljetrykk i regulegingssløyfen for pitchvinkel (rør «S1» i figur 2.1) i en tunnelthruster i baugen på MV «Norwegian Breakaway».

7.1.1 Filtrering

Måleserien på figur 7.1 inneholder en del høyfrekvent støy som ønskes filtrert bort. Dette er gjort med et lavpass IIR-filter laget med Matlabs filterdesign- og analyseverktøy. Filteret er av 5. orden med en stoppfrekvens[6] $f_c = 10\,\text{Hz}$. Matlab beregner koeffisientvektorene til

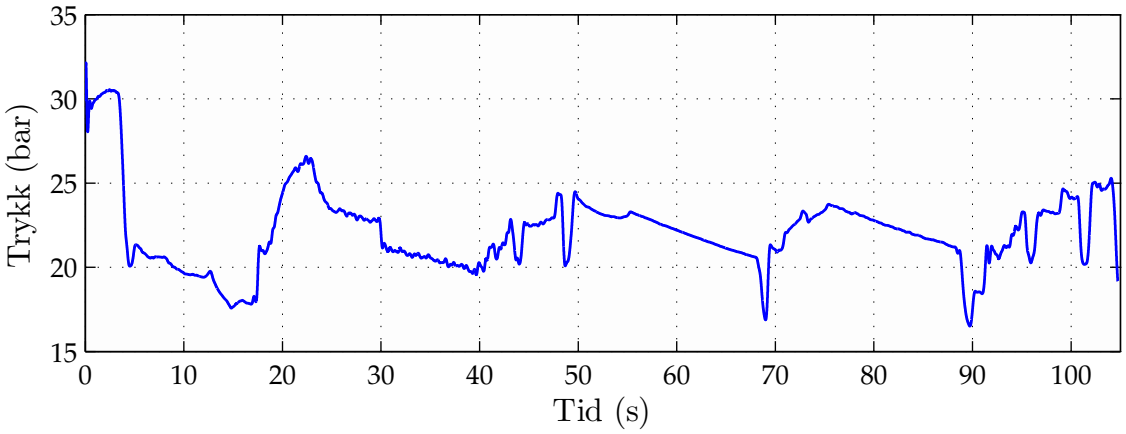
$$\boldsymbol{b} = \begin{bmatrix} 3{,}7612621682594101 \cdot 10^{-5} \\ -1{,}0916484071362099 \cdot 10^{-4} \\ 7{,}1624306415768996 \cdot 10^{-5} \\ 7{,}1624306415768996 \cdot 10^{-5} \\ -1{,}0916484071362099 \cdot 10^{-4} \\ 3{,}7612621682594101 \cdot 10^{-5} \end{bmatrix} \tag{7.1}$$

og

$$\boldsymbol{a} = \begin{bmatrix} 1 \\ -4{,}8604727815912900 \\ 9{,}4515766747452794 \\ -9{,}1914795700471501 \\ 4{,}4701311397723797 \\ -8{,}6975531870444500 \cdot 10^{-1} \end{bmatrix} \tag{7.2}$$

Disse er lagt inn i XML-filen som spesifisert i underkapitlet om prosesseringsblokker, se 5.1.2. Siden et IIR-filter i seg selv ikke er definert som en analyse i

[6] Stoppfrekvensen er definert som punktet der filteret demper signalet med 3 dB.



Figur 7.2: Lavpassfiltrert oljetrykkdata. Den høyfrekvente støyen som tidligere dominerte deler av signalet er filtrert bort med et IIR-filter i tilstandsovervåkningssystemet.

dette prosjektet, er data hentet ut fra hovedprogrammet «manuelt», dvs. ikke skalert, pakket og komprimert, men direkte ut fra filteret med full `double`-presisjon. Dette er gjort for å kontrollere og dokumentere at filterblokken fungerer. Resultatet av filtreringen er vist i figur 7.2. Her ser vi en betydelig reduksjon av høyfrekvente komponenter i signalet. Ved å benytte Matlabs `filter(b,a,x)`-funksjon genereres et identisk plott, noe som bekrefter at IIR-filterblokken fungerer slik den skal.

7.1.2 Histogram

For å danne et bilde av hvordan oljetrykket fordeler seg på ulike nivåer, er data gruppert og vist i et histogram på figur 7.3. Histogramblokken er konfigurert med nedre grense 15 bar og øvre grense 35 bar, valgt ut ifra utsvinget til rådatasignalet. Alle klassene har en bredde på 0,5 bar. Siden et histogram er definert som en analyse, har dataene gått gjennom alle tre programmene: hovedprogrammet, senderprogrammet og mottakerprogrammet. Dette vil si at histogramdata er komprimert (skalert ned) i senderprogrammet før overføring via satellitt. For å dekomprimere (skalere opp) mottatt data er det laget et lite Matlab-skript. I tillegg til den nødvendige oppskaleringen gitt av (5.3), er histogrammet også skalert ved å multiplisere alle frekvensene f_i med punktprøveperioden $\mathrm{d}t = 1/f_\mathrm{s} = 2{,}5\,\mathrm{ms}$, for å få sekunder på andreaksen i stedet for antall punktprøver.

Sammenligner vi histogrammet som tilstandsovervåkningssystemet har generert (figur 7.3) med et histogram generert direkte fra filtrert rådata i Matlab, vist på figur 7.4, er avviket ikke merkbart, til tross for at overført data er skalert ned til bare én byte per klasse. For å se differansen mellom de to histogrammene er det laget et nytt histogram vist på figur 7.5. Den største differansen observert er

$\approx$ 25,53 millisekunder i den 19. klassen, som dekker 24,0–24,5 bar. Denne differansen tilsvarer 0,195 % av den høyeste søylen i diagrammet, som er 13,08 sekunder mellom 23–23,5 bar i klasse 17 (denne søylen er lik i begge histogrammene, fordi frekvensen er overført ukomprimert).

Generelt kan feilen e_i for søyle i i et histogram utledes som følger. La p_i være gitt av formel (5.2) uten avrunding, altså

$$p_i = \frac{f_i}{\max\{f_i\}} \cdot 255 \quad \text{der} \quad i = 1, 2, \ldots, n \tag{7.3}$$

Dette gir $b_i = \lfloor p_i \rceil$. Feilen e_i blir

$$\begin{aligned} e_i = f_i - \hat{f}_i &= \frac{p_i \max\{f_i\}}{255} - \frac{\lfloor p_i \rceil \max\{f_i\}}{255} \\ &= \frac{\max\{f_i\}}{255} \left(p_i - \lfloor p_i \rceil\right) \end{aligned} \tag{7.4}$$

Figur 7.6 viser hvordan feilen (7.4) varierer ved økende frekvenser f_i. Eksempelet er laget med høyeste frekvens $\max\{f_i\} = 510$, plottet viser bare frekvenser fra 0 til 10. Feilen når sine maksimum- og minimumsutslag med perioden $m/255$, i eksempelet $510/255 = 2$. Som det går frem av figuren, er feilen begrenset. Begrensningen skyldes at $-0{,}5 \leq p_i - \lfloor p_i \rceil \leq 0{,}5$, som fører til

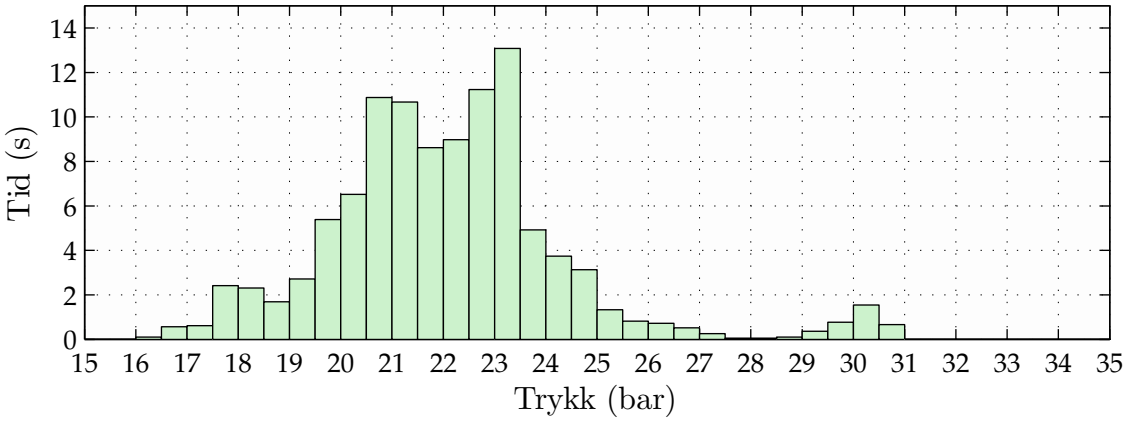
$$|e_i| \leq \frac{\max\{f_i\}}{2 \cdot 255} \tag{7.5}$$

Eller, dersom vi lar q være antall bit vi skalerer ned til (for histogram $q = 8$ bit), får vi den enda mer generelle begrensningen

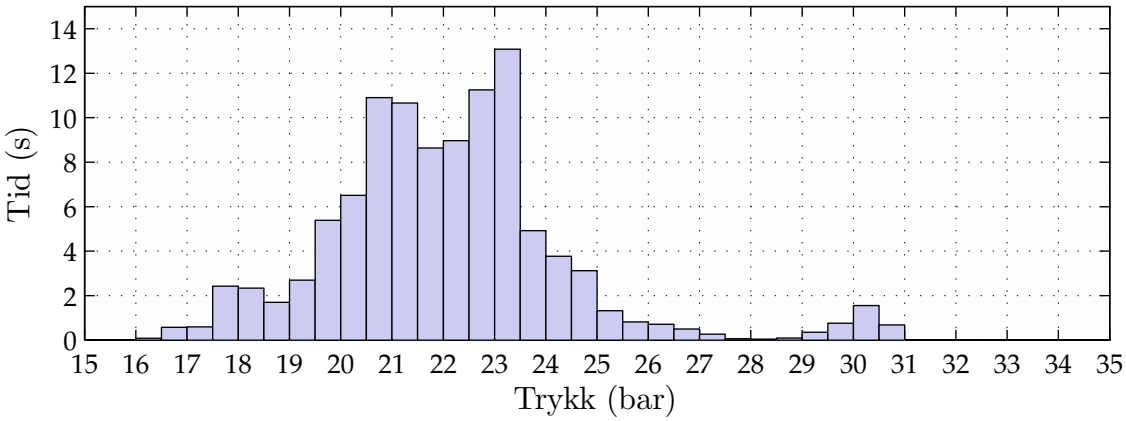
$$|e_i| \leq \frac{\max\{f_i\}}{2^{q+1} - 2} \tag{7.6}$$

Som forventet, er e_i avhengig av maksimumsverdien i histogrammet og antall bit q. I eksempelet på figur 7.6 blir $|e_i| \leq 510/(2^9 - 2) = 1$. Feilmarginen anses som liten nok for med $q = 8$ bit for et histogram.

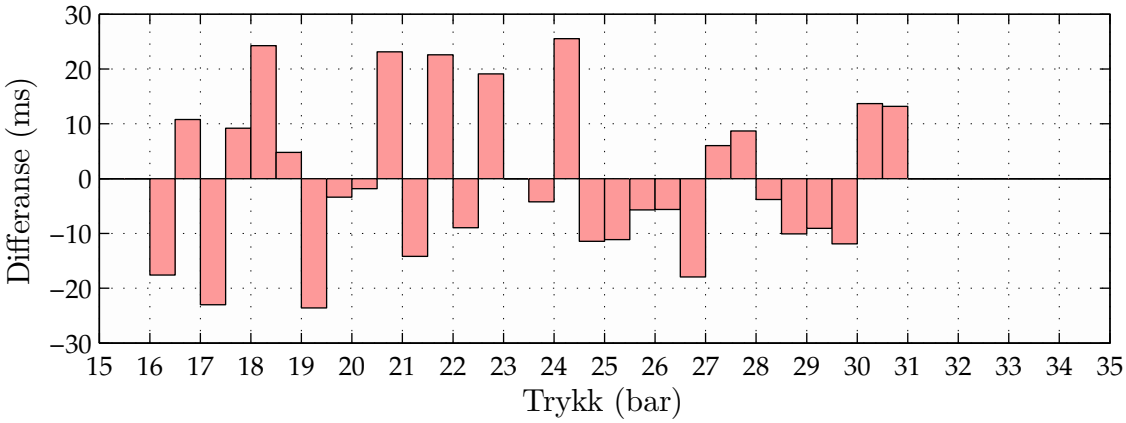
For det overførte histogrammet, som har høyeste søyle 5232 før multiplisering med $\mathrm{d}t = 2{,}5$ ms, blir frekvensfeilen begrenset av $|e_i| \leq 5232/(2^9 - 2) = 10{,}26$. Omregnet ved å multiplisere med $\mathrm{d}t$ tilsvarer dette en maksimal tidsfeil på $\pm$25,65 millisekunder, som stemmer godt overens med figur 7.5.



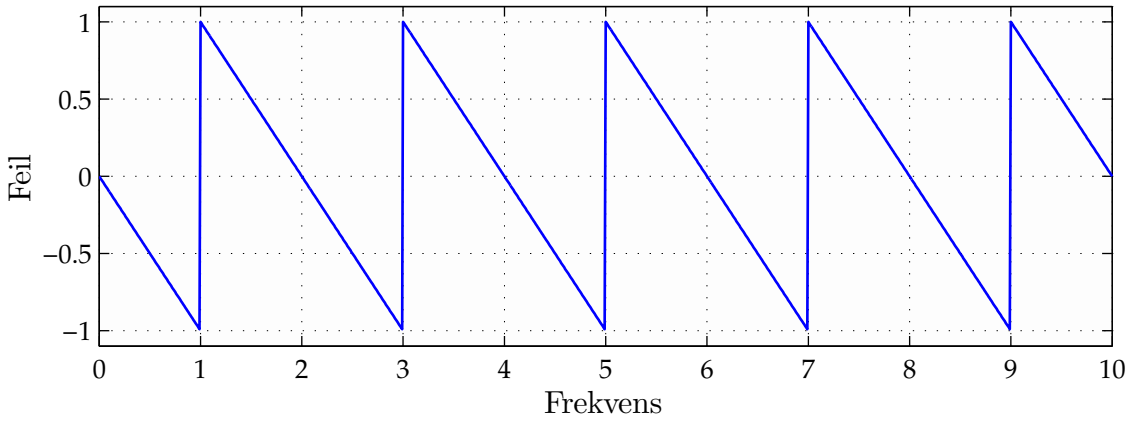
Figur 7.3: Histogram over oljetrykk sendt fra tilstandsovervåkningssystemet. Frekvensene er skalert ned til én byte, overført og skalert opp igjen på mottakersiden.



Figur 7.4: Histogram over oljetrykk generert direkte fra rådata i Matlab. Dette er «fasiten» som tilstandsovervåkningssystemet i figur 7.3 forsøker å tilnærme.



Figur 7.5: Histogram over avrundingsfeilen målt i millisekunder for sendt data. Beregnet som reell data fra Matlab (figur 7.4) minus tilnærmet data fra tilstandsovervåkningssystemet (figur 7.3). Den teoretiske maksimumsfeilen er $\pm$ 25,65 ms.



Figur 7.6: Avrundingsfeil som funksjon av frekvens i et histogram med en maksfrekvens på 510. Feilen når sitt maksimale utslag med en periode på $510/255 = 2$ når data komprimeres til én byte.

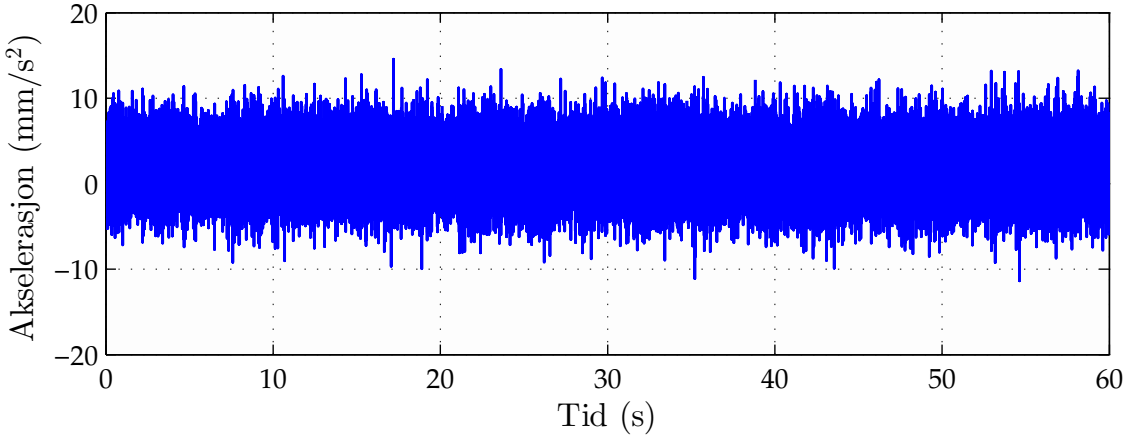
7.2 Vibrasjonsmålinger

Den 16.06.2012 ble det foretatt vibrasjonsmålinger på alle thrustere i baugen på supplybåten MV «Evita». Båten har installert tre thrustere med en samlet effekt på 2,88 MV fordelt på to tunnelthrustere (1 MW hver) og én opptrekkbar asimutthruster (880 kW). Akselerasjonsdata brukt videre i dette underkapitlet er hentet fra den ene tunnelthrusteren, FU-74-LTC-2000-1000kW, målt vertikalt på motorakslingen ved $-100\,\%$ thrustpådrag. Målingene ble gjort med sensorer fra Brüel & Kjær og datainnsamlingsenheten NI-9234 fra National Instruments. En punktprøvefrekvens på $f_s = 2\,\text{kHz}$ ble brukt. Tidsserien på omlag ett minutt (124 000 datapunkter) er vist på figur 7.7.

Akselerasjonsdata er gitt i mm/s^2, mens det er ønskelig å lage et fourierspekter over hastigheten i mm/s. Til dette trengs et høypassfilter, en integrator og en FFT-analyseblokk.

7.2.1 Filtrering og integrering

Ved integrasjon av ren akselerasjonsdata viser det seg at akselerometeret har en bias (nær DC-komponent) som gjør at den teoretiske hastigheten stiger lineært som vist på figur 7.8. Ved å høypassfiltrere akselerasjonsdata før integrasjon, får vi hastigheten som vist på figur 7.9. Filteret som er brukt har koeffisienter som vist i (7.7).



Figur 7.7: Rå akselerasjonsdata målt vertikalt på akslingen til motoren til en tunnelthruster i baugen på MV «Evita». Figuren viser at signalet har et likevektspunkt forskjellig fra null (bias som skyldes drift i måleprobe).

$$\boldsymbol{b} = \begin{bmatrix} 0{,}9905733233651450 \\ -4{,}9528666168257303 \\ 9{,}9057332336514499 \\ -9{,}9057332336514499 \\ 4{,}9528666168257303 \\ -0{,}9905733233651450 \end{bmatrix} \quad \text{og} \quad \boldsymbol{a} = \begin{bmatrix} 1 \\ -4{,}9810583106964597 \\ 9{,}9244104491342409 \\ -9{,}8868804127065495 \\ 4{,}9247627247132000 \\ -0{,}9812344504341760 \end{bmatrix} \tag{7.7}$$

Dette filteret blokkerer bare de aller laveste frekvensene, stoppfrekvensen er $f_c = 2\,\text{Hz}$. Dette betyr at komponenter under f_c vil forsvinne fra spekteret.

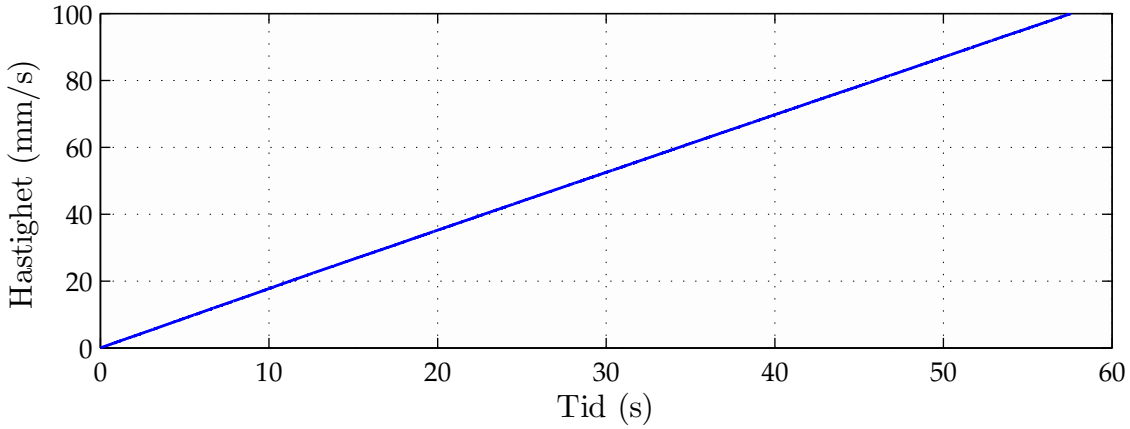
Vanligvis burde data også lavpassfiltreres for å fjerne alle frekvenser over den høyeste frekvensen i fourierspekteret, slik at minst mulig folding oppstår. Dette er allerede gjort med de gitte akselerasjonsdataene.

Et IIR-filter designet etter trapesregelen brukes til integrasjon. Trapesregelen tilnærmer integralet i et diskret datasett. Hvis $y(k)$ er integralet av $x(k)$ for $k = 0, 1, \ldots, N-1$, kan trapesregelen formuleres som

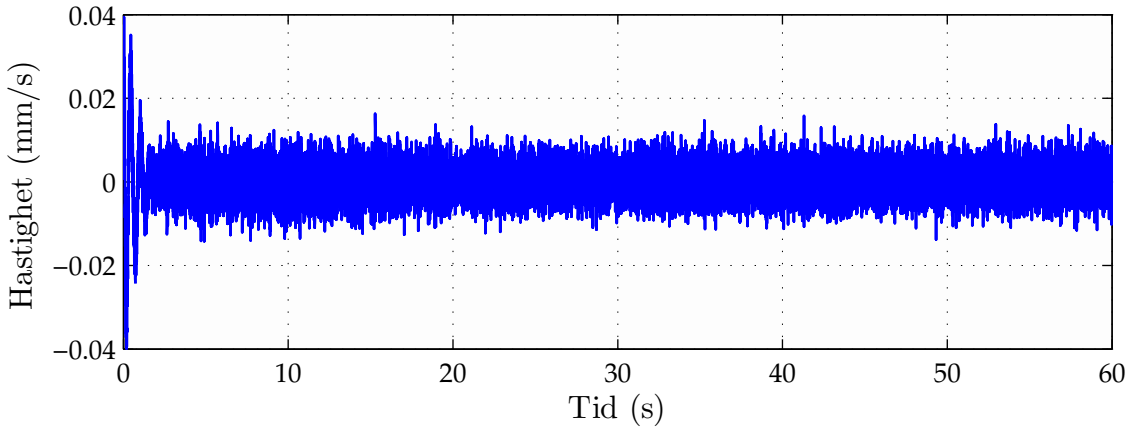
$$y(k) = y(k-1) + \frac{x(k) + x(k-1)}{2}\,\mathrm{d}t, \quad y(0) = 0 \tag{7.8}$$

der $\mathrm{d}t$ er punktprøveperioden $1/f_s = 500\,\mu\text{s}$. Integrasjonsfilteret får dermed koeffisientene

$$\boldsymbol{b} = \begin{bmatrix} 250 \cdot 10^{-6} \\ 250 \cdot 10^{-6} \end{bmatrix} \quad \text{og} \quad \boldsymbol{a} = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \tag{7.9}$$



Figur 7.8: Hastighet fra integrert rå akselerasjonsdata. Det er tydelig at akselerometeret har en bias som fører til en lineær hastighetsøkning når den blir integrert.



Figur 7.9: Hastighet fra integrert høypassfiltrert akselerasjonsdata. De aller laveste frekvensene i akselerasjonsdataene er fjernet, slik at hastigheten får et likevektspunkt nær null.

7.2.2 Spektrumsanalyse

Det er ønskelig å generere et FFT-spekter over hastighetsdata fra figur 7.9. For å få god nok oppløsning benyttes et gjennomsnitt av seks spektere, hvert bestående av 20 000 punkter per vindu. Siden dette ikke er en toerpotens, legges det til nuller slik at $N = 2^{15} = 32\,768$. Et flat topp-vindu benyttes for god amplitudenøyaktighet i spekteret. Videre er det interessante frekvensområdet satt til 0–100 Hz, som gir `TxLower` $= 0$. For å finne verdien til `TxUpper` må vi først beregne den tilsynelatende frekvensoppløsningen

$$\mathrm{d}f = \frac{f_\mathrm{s}}{N} = \frac{2\,\mathrm{kHz}}{2^{15}} \approx 61{,}04 \cdot 10^{-3}\,\mathrm{Hz} \tag{7.10}$$

Den egentlige frekvensoppløsningen er gitt av $f_\mathrm{s}/20\,000 = 0{,}1$ Hz, men siden vinduene er paddet med null, vil det resulterende spekteret ha linjer med avstanden beregnet i (7.10). For å overføre FFT-data opp til og med 100 Hz, blir den siste linjen som overføres

$$\texttt{TxUpper} = \left\lceil \frac{f_\mathrm{max}}{\mathrm{d}f} \right\rceil = \left\lceil \frac{100\,\mathrm{Hz}}{61{,}04 \cdot 10^{-3}\,\mathrm{Hz}} \right\rceil = 1\,639 \tag{7.11}$$

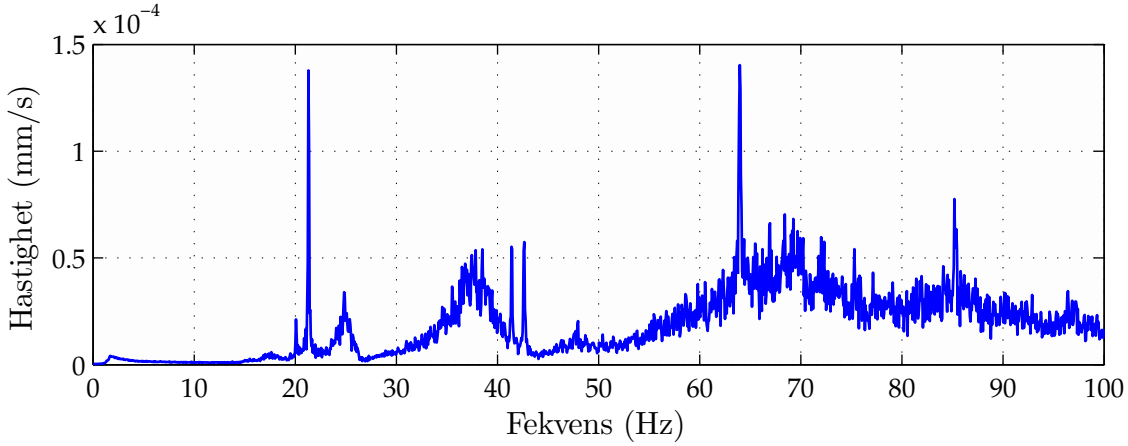
Med denne konfigurasjonen beregner tilstandsovervåkningssystemet spekteret som vist på figur 7.10. Matlab genererer et spekter med merkbart mindre støy, vist på figur 7.11. Samme fremgangsmåte ble brukt både i tilstandsovervåkningssystemet og Matlab: høypassfilter, integrasjon og FFT beregnet som det inkoherente gjennomsnittet av seks spektere. Avviket skyldes ikke skalering av overført data, fordi begrensningen gitt av (7.6) gjelder også for FFT-data, med $q = 16$ bit. Går vi ut ifra Matlab-spekteret, er den største toppen $\approx 1{,}25 \cdot 10^{-4}$ mm/s. Dette gir en feilmargin på

$$|e_i| \leq \frac{1{,}25 \cdot 10^{-4}}{2^{17} - 2} \approx 9{,}51 \cdot 10^{-10}\,\mathrm{mm/s} \tag{7.12}$$

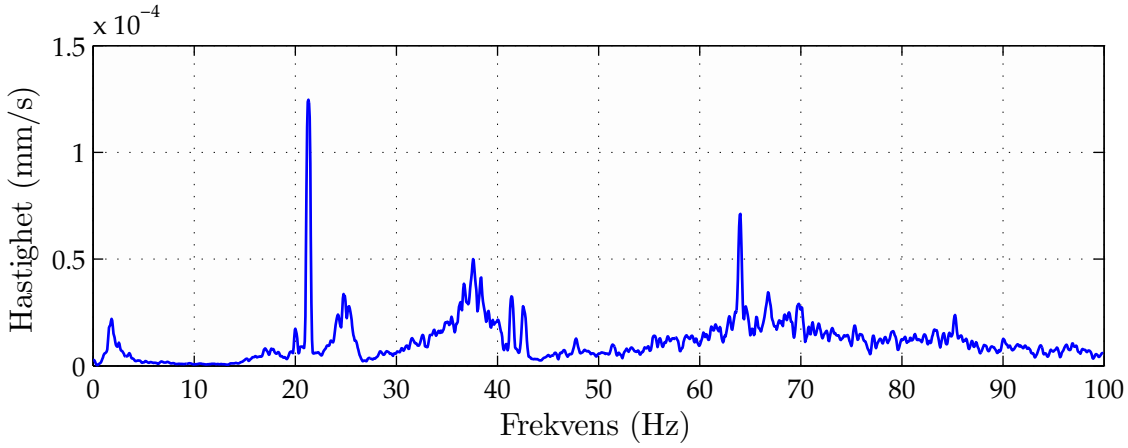
som er flere størrelsesordener lavere enn støyen på figur 7.10. Støykilden kan være FFT-algoritmen i seg selv, fordi små avrundingsfeil akkumuleres fort ved rekursive kall.

7.3 Pitchvinkelmålinger

Pitchvinkelhastigheten er en viktig parameter å analysere. Nedsatt hastighet kan tyde på slitasje eller groe på propellblad. Figur 7.12 viser en tidsserie for pitchvinkelen generert med Brunvolls thrustersimulator. Punktprøvefrekvensen er $f_\mathrm{s} = 10$ Hz. Den mest interessante verdien er den gjennomsnittlige hastigheten. Tilstandsovervåkningssystemet er derfor konfigurert til å derivere, lavpassfiltrere og beregne de statistiske egenskapene til absoluttverdien av resultatet.



Figur 7.10: Frekvensspekter over hastigheten sendt fra tilstandsovervåkningssystemet. Støyen i spekteret kan skyldes en dårlig FFT-algoritme.



Figur 7.11: Frekvensspekter over hastigheten beregnet i Matlab. Det er brukt et flat topp-vindu og seks *averages* med 20 000 punkter i hver, som for figur 7.10. Likevel har dette spekteret mindre støy.

7.3.1 Derivasjon, filtrering og absoluttverdi

Derivasjonsblokken er konfigurert med $\mathrm{d}t = 0{,}1\,\mathrm{s}$. Simulatoren fungerer ikke optimalt, noen ganger gir den identiske pitchvinkelverdier 2–3 punktprøver etter hverandre. Dette fører til at det deriverte signalet inneholder ekstra mye støy, som vist på 7.13. Reduksjon av støy gjøres med et 2.-ordens lavpassfilter med stoppfrekvens $f_c = 4\,\mathrm{Hz}$. Matlab beregner koeffisientene til

$$\boldsymbol{b} = \begin{bmatrix} 0{,}858122741522684 \\ 0{,}858122741522684 \end{bmatrix} \quad \text{og} \quad \boldsymbol{a} = \begin{bmatrix} 1 \\ 0{,}716245483045369 \end{bmatrix} \tag{7.13}$$

Siden vi er interessert i den gjennomsnittlige hastigheten, benyttes en absoluttverdiblokk på det filtrerte signalet. Resultatet y er vist i figur 7.14. Filtreringen er ikke perfekt, men den gjenværende støyen vil ha liten innvirkning på gjennomsnittsberegningen, som i seg selv er et lavpassfilter.

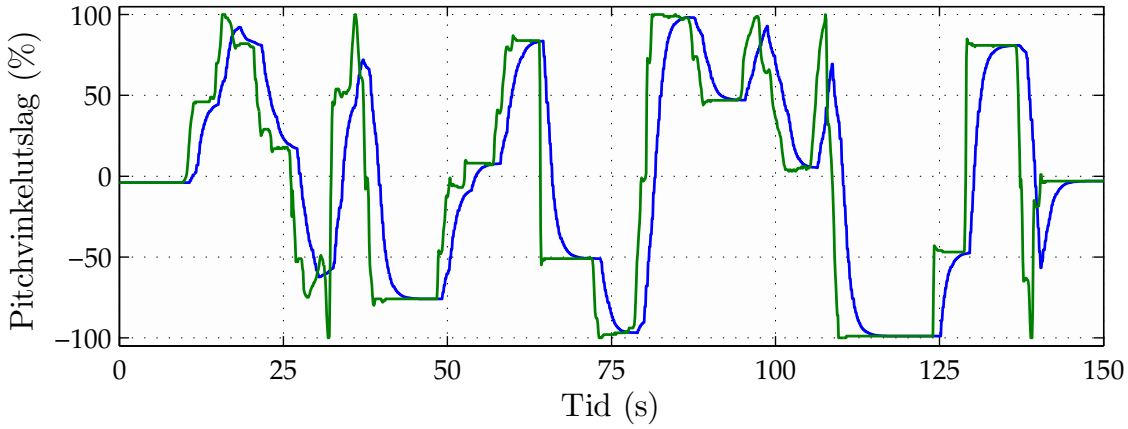
7.3.2 Statistiske egenskaper

Analyseblokken beregner de statistiske egenskapene til signalet y på figur 7.14 til å være

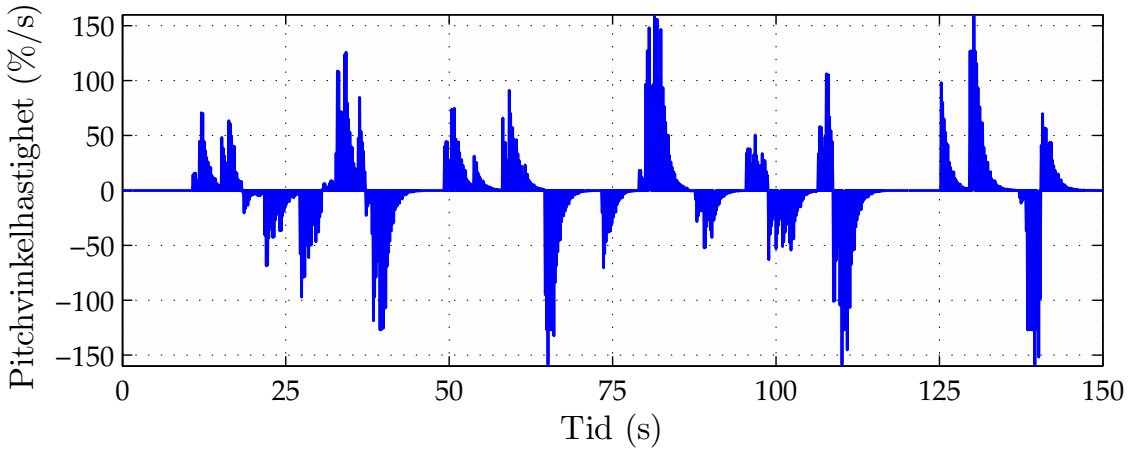
$$\mathrm{Var}(y) = 446{,}46390, \quad \bar{y} = 13{,}391925 \quad \text{og} \quad \max\{y\} = 155{,}81375 \tag{7.14}$$

etter overføring via satellitt. Parametrene er praktisk talt identisk med resultater beregnet i Matlab, ikke overraskende siden de overføres ukomprimert som 32 bits `float`-variabler (og internberegninger utføres med full `double`-presisjon). Den eneste forskjellen er at Matlab presenterer resultatet som `double`-verdier. Dersom Matlab-verdiene kastes til `float`, får vi verdiene i (7.14) eksakt, noe som viser at statistikkblokken og dens algoritmer fungerer.

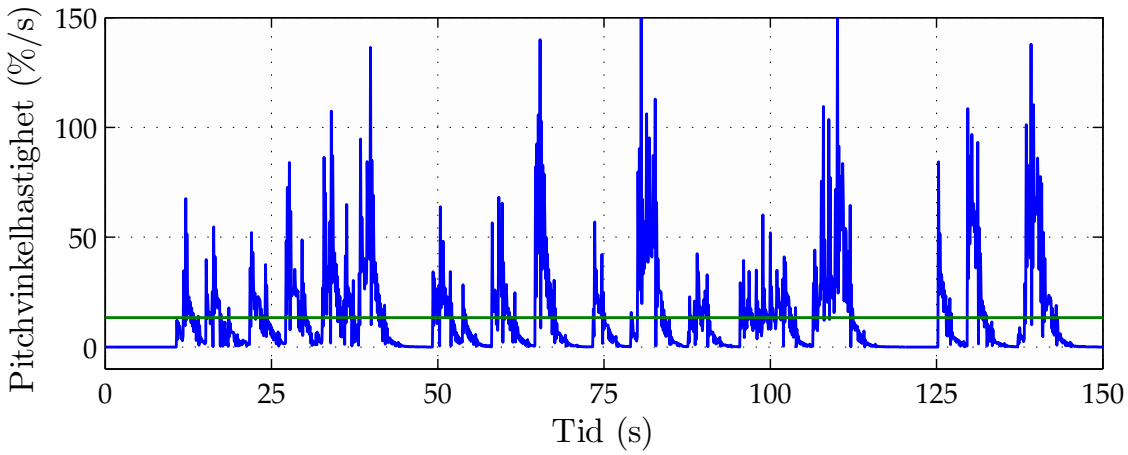
Gyldigheten til denne analysen implementert i praksis kan diskuteres, siden det ikke blir tatt hensyn til at det er perioder hvor pitchvinkelen ikke endres.



Figur 7.12: Pitchvinkel fra Brunvolls thrustersimulator. Den grønne kurven er settpunktet og den blå er simulert «målt» vinkelutslag i prosent.



Figur 7.13: Derivert pitchvinkel fra tilstandsovervåkningssystemet. Simulatoren gir noen ganger 2–3 identiske verdier etter hverandre, som fører til mye støy i den deriverte.



Figur 7.14: Filtrert derivert pitchvinkel y (blå kurve) og gjennomsnittet $\bar{y} \approx 13{,}9$ (grønn linje) fra tilstandsovervåkningssystemet.

Kapittel 8

Drøfting og forslag til videre arbeid

Først i dette kapitlet drøftes ulike løsninger for kommersialisering av tilstandsovervåkningssystemet. Deretter drøftes metodene som er valgt for implementasjon og resultatene. Både fordeler og ulemper blir belyst, samt forslag til forbedringer og videre utvikling.

8.1 Kommersiell løsninger

For i fremtiden å kunne tilby kunder et automatisk tilstandsovervåkningssystem, er det sett på tre ulike løsninger med forskjellig grad av automasjon og kompleksitet. Løsningene er lagt opp slik at de to første er mulig å «oppgradere», med f.eks. satellittkommunikasjon eller høyere punktprøvefrekvens og andre måleprober.

8.1.1 Lavhastighetsmålinger uten kommunikasjon

Velges denne løsningen installeres bare prosesserings- og analyseprogrammet ombord på skipet. Kommunikasjon over satellitt er utelatt, slik at data må hentes manuelt. Fordelen med et slikt system er at en egen satellittkommunikasjonsenhet ikke må installeres. Løsningen egner seg derfor godt til ettermontering.

«Lavhastighetsmålinger» vil si målinger som hentes direkte fra PLS-ene som styrer thrusterene, slik at ingen eksterne sensorer eller datainnsamlingsenheter behøves. PLS-ene med Modbus-tilkobling har en teoretisk øvre punktprøvefrekvens på 1000 Hz, men i praksis benyttes 10 Hz eller lavere.

8.1.2 Lavhastighetsmålinger med kommunikasjon

Denne løsningen er lik den forrige, men har i tillegg mulighet til kommunikasjon. Kommunikasjonen kan skje ved hjelp av en dedikert enhet som DTAC eller gjennom skipets kommunikasjons-/Internett-system. Fordelene med dette er åpenbare: reduserte utgifter til servicepersonell, raskere og enklere tilgang til data. Masteroppgaven har hatt denne løsningen som mål for programmene og systemet som er utviklet.

8.1.3 Høyhastighetsmålinger med kommunikasjon

Høyhastighetsmålinger ($>100\,\text{Hz}$) kan gi mer informasjon om et skips driftskondisjon enn ved bruk av bare lavhastighetsmålinger. Aktuelle målinger kan være strøm, spenning og vibrasjon.

Brunvoll benytter i dag et stort utvalg av enheter fra National Instruments (NI), spesielt moduler fra C-serien [2]. Ved å benytte en NI CompactRIO kan loggesystemet utvides til å takle datainnsamling med mye høyere punktprøvefrekvenser. Dette er ikke den eneste fordelen, ved å benytte en ekstern datainnsamlingsenhet, kan signaler som PLS-systemene ikke har tilgang til også måles. Eksempler slike signaler kan være strøm og spenning, men også lavhastighetsmålinger som omgivelsestemperatur. Tabell 8.1 presenterer et forslag til ulike NI-enheter.

Tabell 8.1: Forslag til komponenter for høyhastighetsmålinger.

Enhet	Beskrivelse
CompactRIO-9076	Sjassis hvor innsamlingsmodulene plugges inn. Kjører LabVIEW Real-Time og har i tillegg en programmerbar FPGA som kan uføre tunge regneoperasjoner.
NI-9234	Analog inngangsmodul for akselerometre. Fire kanaler for IEPE-standarden med opp til 51,2 kHz punktprøvefrekvens.
NI-9215	Analog inngangsmodul for spenningssignaler, $\pm 10\,\text{V}$. Fire kanaler med maksimum 100 kHz punktprøvefrekvens.

Fastemonterte akselerometere kobles til IEPE-modulen. Ved å programmere FPGA-brikken kan signalbehandlingen (f.eks. FFT) gjøres svært effektivt. På denne måten behøver ikke all prosesseringen å gjøres av IPC-en, den trenger bare å lese av resultatene.

8.2 Presentasjonsverktøy

Som tidligere nevnt, ble det ikke tid til å utvikle et presentasjonsverktøy i løpet av masteroppgaven. Dette underkapitlet vil drøfte hvordan et slikt verktøy kan designes og hva som må tas hensyn til.

Et presentasjonsverktøy kan realiseres på flere måter. De to mest aktuelle er et frittstående program eller en web-løsning.

Velges det et frittstående program kan utviklingen bli relativ enkel. De samme klassene som er benyttet for analyser (statistikk, histogram og FFT) kan utvides med presentasjonsmetoder og brukes direkte på dekodet data fra mottaks-programmet. En annen fordel med denne løsningen er at programmet kan brukes uten tilgang til Internett (så lenge databasen med mottatt data er tilgjengelig). Dette gjør det ideelt til bruk ved serviceoppdrag der nettilgang kan være et problem. Skal data presenteres til mange innen bedriften eller til eksterne kunder, vil denne løsningen by på flere utfordringer. Det blir vanskelig å kreve at hver enkelt må installere programmet for å se resultatene.

En web-løsning derimot, vil egne seg godt til presentasjon for mange brukere, både interne og eksterne. Utviklingen kan bli litt mer utfordrende, siden et nytt programmeringsspråk som egner seg for web må tas i bruk. Det må vurderes om data skal hentes direkte fra databasen, eller om det skal opprettes egne metoder til dette i analyseklassene (som for den andre løsningen).

8.2.1 Krav til funksjonalitet

Uansett hvilken måte presentasjonsverktøyet blir realisert på, bør det som et minstekrav inneholde funksjonene i følgende liste.

- Identifikasjon av skip, thrustertype, ordrenummer o.l.
- Mulighet for plotting av data, både vanlige lineære plott og histogrammer.
- Funksjon for å bla i historikk og generere trendlinjer.
- Felt for å vise tallverdien til enkeltparametre, f.eks. statistiske egenskaper.
- Visning av skipets posisjon på et kart (krever at DTAC-enhetens posisjonsfunksjon tas i bruk).
- Mulighet for å sende forespørsler fra land til skip (krever videreutvikling av hovedprogrammet), se 8.5.7.

Det må opprettes «brukere» som har ulik tilgang/rettigheter til de forskjellige funksjonene. For eksempel bør ikke hvem som helst få sende data til DTAC-enheten(e), og hver kunde må kun ha tilgang på data tilhørende sitt/sine skip.

En annen viktig funksjon, i tillegg til de som ble nevnt i listen, er visning av data sortert etter thrustertype eller thrusterserie i stedet for skip. Dermed flyttes fokuset fra enkeltsystemer over til en thrusterfamilie som en helhet. Dette bør ses i sammenheng med eksisterende systemer for servicearbeid, drøftet i neste underkapittel.

8.3 Integrering opp mot eksisterende systemer

Brunvoll benytter i dag et system som kalles *Thruster Service Management* (TSM) med brukergrensesnitt gjennom Lotus Notes til å registrere alle opplysninger som har med service og vedlikehold å gjøre. Opplysningene er sortert etter ordrenummer, der hver ordre inneholder flere thrustere eller prosjekter (f.eks. ombygging av styresystem).

Ved hjelp av TSM-systemet kan det genereres statistikk over thrusterserier, på tvers av ordre. Dette gir mye informasjon om vanlige feilsituasjoner, omsetning på reservedeler o.l, men det sier lite eller ingenting om thrustersystemenes driftskondisjon før feilen oppstod.

Integreres tilstandsovervåkningssystemet opp mot TSM, gir dette et mye større utvalg av parametre til å analysere årsaken til feilsituasjonen. Et hypotetisk eksempel kan være som følger: I en gitt thrusterserie oppstår en spesifikk feil på enkelte enheter, mens andre er helt feilfrie. Noen åpenbar årsak til dette er det ikke mulig å finne i statistikken fra TSM. Ved å benytte data fra tilstandsovervåkningssystemet i tillegg til TSM, viser det seg at enhetene med feil er svært mye brukt og kjøres ofte med tilnærmet maksimalt pådrag. Dette tyder på at feilen skyldes slitasje på grunn av bruksmønsteret.

Prosessen rundt integrasjon kan gjøres gradvis. I første omgang kan det opprettes et felt i TSM som linker til det aktuelle skipet i tilstandsovervåkningssystemets presentasjonsverktøy (web-løsning). På lengre sikt bør det vurderes å integrere systemene enda tettere, slik at TSM får tilgang til all tilstandsovervåkningsdata. Først da kan det hypotetiske eksemplet bli reelt.

8.4 Trigging

Muligheten til å trigge på f.eks. signalnivåer vil kunne gjøre systemet mer «intelligent». Med trigging menes en hendelse eller en *trigger* som fører til at en spesifikk handling blir utført i programmet. Et eksempel kan være å starte en analyse (handling) når et gitt signal er innenfor forhåndsdefinerte grenser (trigger). På denne måten kan mer relevant data analyseres for parametre som varierer sterkt med driftskondisjon; data hentes under mest mulig like forhold.

Mulige triggerkilder kan være

- signalnivåer eller andre egenskaper ved et signal (stabilt nivå, pulsdeteksjon, hurtige endringer o.l.).
- tidspunkter eller tidsintervaller.
- digitale signaler.
- alarmer.
- forespørsler fra land.
- analyseresultater som ligger innenfor et forhåndsdefinert område.
- signaler fra andre systemer, f.eks. skipshastighet eller GPS-posisjon.
- manuell innputt.
- **en kombinasjon av punktene ovenfor.**

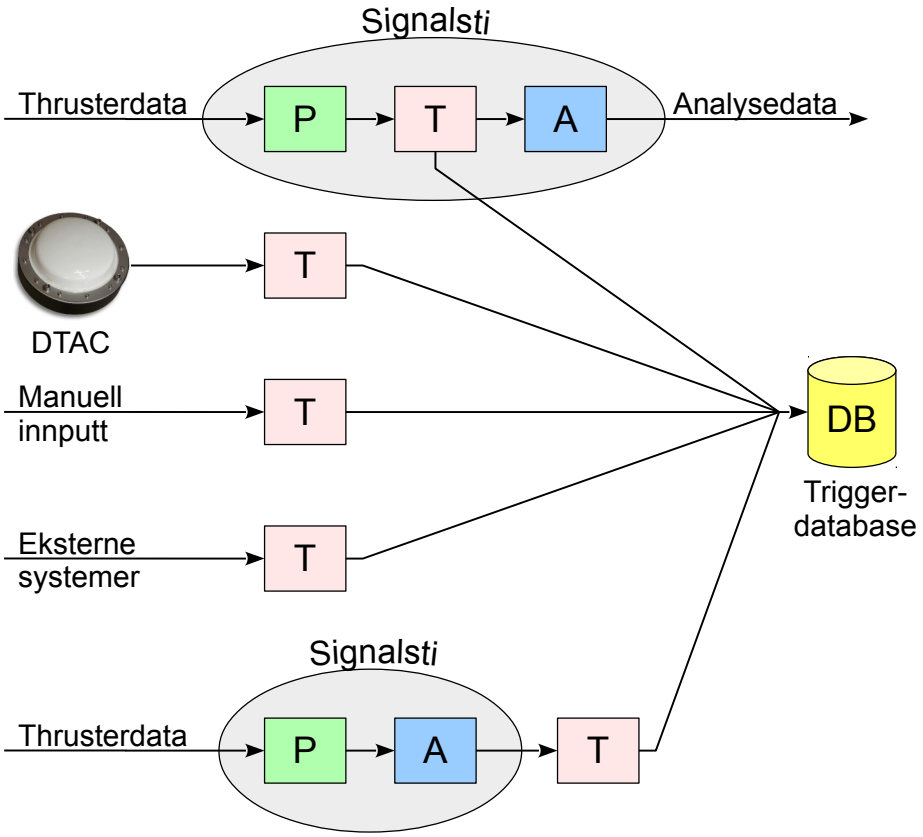
Når systemet blir trigget må en spesifisert handling utføres. Mulige handlinger kan være

- samle inn data fra et spesifikt signal og gjøre en analyse.
- restarte en analyse.
- starte overføringen av et senderskjema eller analyseresultat via satellitt.
- hente og analysere historiske data fra et gitt tidsrom fra databasen.
- **en kombinasjon av punktene ovenfor.**

Slik programmet er på det nåværende tidspunkt, begynner alle signalstier å behandle data når nye punktprøver legges inn i thrusterdatabasen. Det eksisterende loggeprogramet skriver til databasen så lenge thrusteren går, noe som betyr at hovedprogrammet trigges av start/stopp-signalet til thrusteren. Senderprogrammet er på sin side trigget periodisk, i henhold til tidsintervallene spesifisert i senderskjemainnstillingene i XML. Handlingene som utføres er overføring av et senderskjema og restarting av de tilhørende analysene.

Implementasjon av triggere kan gjøres på ulike måter. Den enkleste formen styrer bare start/stopp av analyser trigget av signalegenskaper. Deteksjon av signalegenskaper kan gjøres ved å ha triggerblokker som fungerer på samme måte som prosesseringsblokkene. Triggerblokkene sender data videre til neste blokk bare når kravene for trigging er oppfylt. Denne løsningen muliggjør bare trigging på signalnivåer eller -egenskaper. Handlingen er å starte en analyse.

En annen mulighet er å ha triggerblokker som ovenfor, men som slipper gjennom data hele tiden, og på den måten bare overvåker signalnivået. Hver gang kriteriene for trigging blir oppfylt, skrives triggerinformasjon til en database. Hvilken informasjon dette skal være, varierer etter triggerkilde. Som et minimumskrav må en trigger-ID og et tidsstempel inkluderes. Andre typer triggere kan også implementeres, slik at de tilsammen dekker punktene i listen over forslag til kilder. Alle skriver til den samme triggerdatabasen. Dette konseptet er illustrert på figur 8.1.



Figur 8.1: Blokkskjema over mulige triggerkilder. Hver gang kriteriene for trigging er oppfylt, skrives det til en triggerdatabase. Tegnforklaring: P = prosessering, A = analysering, T = trigging.

Trigging er verdiløst om det ikke fører til handling, databasen må også leses. Signalstiene bør kontrollere om det har oppstått nye triggersituasjoner og avgjøre hvilken handling som eventuelt skal utføres, f.eks. starte, stoppe eller restarte behandlingen av data. Her har vi en meget stor fordel ved å samle triggerne i én database; den kan aksesseres på tvers av tråder i programmet. Ønskes enda mer detaljstyring ved trigging, kan hver enkelt analyseblokk også kontrollere databasen og utføre handlinger deretter. Det blir isåfall viktig å lage en god rutine for når en analyseblokk får lov til å lese fra databasen, ellers kan systemet bli tregt på grunn av stadige spørringer.

Sending av data etter trigging gjøres ved å ha en egen tråd som overvåker databasen og starter senderprogrammet ved behov.

Triggingen kan også «heves et nivå opp» ved å la prosessen (f.eks. signalstien) som leser databasen ha et sett med regler for *kombinasjoner* av triggere. Eksempelvis kan det være ønskelig å analysere motorlast når *både* pitchvinkelen er innenfor et gitt vindu *og* motorens rotasjonshastighet er innenfor et annet, gitt vindu. Triggerblokkene i signalstiene som behandler pitchvinkel og rotasjonshastigheten behøver bare å kontrollere at signalene er innenfor deres respektive vinduer, uten at de «vet om hverandre». Signalstien som leser av databasen starter ikke analysen før begge triggerne er tilstede. (Her er det også behov for triggere som forteller når et av signalene forlater sitt vindu, slik at analysen kan stanses.)

Før implementasjon er det viktig å indentifisere og få en nøyaktig spesifikasjon på alle triggerkilder og -handlinger. Deretter kan et entydig sett med regler for konfigurasjon i XML utvikles.

8.5 Programmenes oppbygning

I dette underkapitlet diskuteres programmenes oppbygning, det vil si valg av bibliotek, hvordan videreutvikle systemet med flere prosesserings- eller analyseblokker og fordeler og ulemper med de løsningene som er valgt. Andre mulige forbedringer eller videreutviklingsforslag presenteres også.

8.5.1 Programvarebibliotek

Det er fokusert på å velge anerkjente og mye brukte biblioteker i utviklingen av programmene. Dette øker robustheten, siden feil i bibliotekene er mindre sannsynlig. Samtidig blir det lettere å vedlikeholde og videreutvikle koden. Den følgende listen på neste side beskriver bibliotekene som er brukt.

SQLite3 brukes til alle databaseoperasjoner. Programvarelisensen er *Public Domain*, som betyr at biblioteket fritt kan benyttes i proprietære produkt uten åpen kildekode.

Libxml2 ligger bak lesing og dekoding av XML-filer. Med *MIT*-lisens kan også dette biblioteket brukes fritt i proprietære produkt uten åpen kildekode. For å få Libxml2, som er et rent C-bibliotek, til å fungere i C++, benyttes *wrapper*-biblioteket **libxml++** sammen med **glibmm-2.4**.

Boost er et anerkjent såkalt *peer reviewed* bibliotek som utvider funksjonaliteten til C++ på mange forskjellige områder. Flere underbiblioteker av Boost har tidligere blitt implementert som en del av C++-standarden, noe som bevitner god kvalitet. I dette prosjektet er Boost benyttet til

- konvertering av tekst til tall eller til arrayer med tall, ved hjelp av `boost::regex`.
- trimming av tekststrenger med `boost::algorithm`.
- generell typekonvertering med `boost::lexical_cast<>`.
- bufring av data i både FFT-, IIR- og RMS-klassen ved hjelp av en ringbuffer fra `boost::circular_buffer<>`.
- filsystemoperasjoner, f.eks. kontrollere om en fil eksisterer, ved hjelp av `boost::filesystem`.

Siden hele Boost er stort og omfattende, er det lagt vekt på å inkludere bare de underbibliotekene som er nevnt i listen ovenfor. *Boost Software License* gir de samme rettighetene som de to tidligere nevnte lisensene.

For FFT-analyse er det laget en egen klasse med en ganske enkel algoritme (listet i 5.1.3) som generer en del støy i spekteret, slik den praktiske utprøvingen av systemet viste. Implementasjon av et bedre FFT-bibliotek bør vurderes dersom mer robust og optimalisert kode eller flere muligheter behøves. Et av de mest anerkjente bibliotekene er **FFTW** [10], som blant annet brukes av Matlab. FFTW er i utgangspunktet gratis og har åpen kildekode, men lisensen er av en såkalt *copyleft*-type, som betyr at alle som benytter biblioteket må ha åpen kildekode og samme lisens på sine produkter. Løsningen på dette er å kjøpe en kommersiell lisens som tilbys av MIT (som har kopirettighetene til FFTW) til 7 500 USD. Et vidt spekter av andre FFT-biblioteker eksisterer også, ofte innen audio-programvare.

8.5.2 Dynamiske modeller og nye blokker

En god forbedring og videreutvikling av programmene vil være å kunne bruke dynamiske modeller for parameterestimering og tilstandsovervåking. Ofte vil slike modeller kreve mer enn ett inngangssignal. Et eksempel på en relevant modell fins i [26], hvor skyvkraften (thrusten) til en propell estimeres ut ifra bare to målte parametre; propellens rotasjonshastighet og motorens dreiemoment.

Thrustpådrag og -tilbakemelding er tilgjengelig i thrusternes PLS-systemer, slik at modellen kan benyttes til parameterestimering. Større endringer i de estimerte verdiene viser at thrustersystemets dynamikk forandrer seg, og det kan igjen indikere slitasje eller at en feil har oppstått.

Analyse- og prosesseringsprogrammet er, som tidligere nevnt, laget slik at hvert signalstiobjekt kjører i sin egen tråd. Implementasjon av modeller med flere innganger krever derfor datautveksling på tvers av tråder, noe som kan bli en utfordring. Teknikken med bruk av databaser (som forslag til triggerløsning beskriver) kan vurderes. Et alternativ til dette er å skrive om signalstiklassen slik at den kan håndtere flere signalkilder innenfor samme objekt (og dermed samme tråd).

Modeller med én inngang kan implementeres som enten en prosesserings- eller analyseblokk, avhengig av om resultatene skal sendes direkte til land, eller om de må gjennom en annen analyseblokk først. Nye blokker må følge visse regler, spesifisert i de neste avsnittene. Så lenge prosesserings- og analyseblokkene tilfredsstiller disse reglene, er det relativt enkelt å inkludere dem i programmene.

Krav til nye prosesseringsblokker

Ny prosesseringsblokker må arve klassen `SignalProcessor` og implementere følgende metoder hentet fra headerfilen til klassen:

```
class SignalProcessor
{
public:
    virtual double processData(double data, double    ↩
       time);
    virtual vector<double> processData(vector<double> ↩
       data, vector<double> time);
};
```

Prosesseringsmetodene må kunne ta inn data fra thrusterdatabasen eller fra andre prosesseringsblokker og returnere ferdigbehandlet data. Et godt utgangspunkt for en ny prosesseringsblokk er å kopiere den enkle absoluttverdiklassen.

Krav til nye analyseblokker

Alle analyseblokker må arve klassen `Analysis`. Et forenklet utdrag av headerfilen som definerer denne klassen viser hvilke virtuelle metoder en arvet klasse må implementere:

```
class Analysis
{
public:
    virtual void addData(double data, double time);
    virtual void addData(vector<double> data,          ↩
       vector<double> time);
    virtual void saveToDb();
    virtual void restartAnalysis();
};
```

Metodene brukes som følger. Funksjonene `addData(...)` tar imot data (med tilhørende tidsstempel) som skal analyseres, enten det er en vektor eller en skalar. Ingenting blir returnert, all mellomlagring må håndteres av det arvede objektet selv. Når programmet har hentet ut all ny thrusterdata og analysen er ferdig, kalles `saveToDb()`. Denne metoden må kunne opprette en egen resultattabell hvor data lagres for sending. Når data blir sendt, kalles `restartAnalysis()`, som gjør at objektet glemmer all innputt- og resultatdata og starter analysen på nytt.

I tillegg til disse må en statisk metode definert ved

```
static vector<uint8_t> getBinaryData(string tableName);
```

implementeres. Denne brukes til å generere en analysespesifikk pakke fra resultatene som ble lagret i databasen. Pakkingen må følge de samme reglene som de eksisterende analysene gjør, se B.2 i vedlegg.

8.5.3 Strukturering av databaser

Slik situasjonene er nå, vil det opprettes nye analysetabeller hver gang et resultat er sendt eller forsøkt sendt. Dette er ønskelig for å kunne ha historikk lagret ombord i tilfelle sendingen mislykkes.

I løpet av prosjektperioden er det blitt klart at måten databasene er *strukturert* på ikke er optimal. Ta for eksempel tabellen en statistikkanalyse oppretter, vist i tabell 8.2. Her lagres hver variabel i en egen rad istedenfor i en egen kolonne. Dette fører til at for hver omstart av statistikkanalysen (etter hver sending), vil det opprettes en ny tabell, alltid med sju rader. Et bedre alternativ til dette er å bruke variablene som kolonnenavn – da blir det brukt bare én rad per analyse, og dermed kan alle analysene fra samme blokk ligge i én tabell.

Det samme gjelder også for histogram- og FFT-klassene. Forskjellen er at disse ikke har et fast antall rader, omgjøring av disse til kolonner kan derfor bli en utfordring. En mulig løsning er å skille ut data til egne tabeller og ha en kolonne som peker til disse i en tabell hvor alle de «faste» feltene lagres. Eksempler på faste felt for et histogram er antall klasser og nedre og øvre grense, for et FFT-spekter er det f.eks. antall punktprøver, vindusfunksjon o.l. Innenfor tidsrammene til dette prosjektet ble en omskrivning av programmene ikke mulig.

Alle analyseobjektene lagres fullstendig i databasen, inkludert mellomregningsvariabler som f.eks. Δ, M_n og n i statistikkbokken.

Tabell 8.2: Eksempel på databasetabell med statistikkparametre. Siden det alltid er sju verdier som lagres per statistikkanalyse, burde tabellen vært omgjort til én rad.

rowid	VarName	Value
0	`_min`	0
1	`_max`	508.653382239392
2	`_mean`	0.9978220790582224
3	`_count`	113994
4	`_M2`	31250843.24930035
5	`StandardDeviation`	16.557386192610686
6	`Variance`	274.14703753125497

8.5.4 Automatisk seleksjon av FFT-data

En endring som bør vurderes innført i FFT-blokken er automatisk fjerning av data for de høyeste frekvensene i spekteret. Dette bør gjøres fordi filtre eller måleutstyr som benyttes før FFT-beregningen ikke alltid greier å fjerne alle frekvenser over nyquistfrekvensen $f_{\text{Nyquist}} = f_s/2$, og dermed får vi folding. I praksis anbefaler blant andre National Instruments [24] å sette den effektive båndbredden til $1/2{,}56$ slik at antall linjer i spekteret N_{FFT} er gitt av

$$N_{\text{FFT}} = \frac{1}{2{,}56} N \tag{8.1}$$

istedenfor $N_{\text{FFT}} = N/2$ slik det er nå. [15] Den høyeste frekvensen i spekteret blir redusert tilsvarende:

$$f_{\max} = \frac{f_s}{2{,}56} \tag{8.2}$$

Begrensingen kan innføres ved å bruke elementet `<TxUpper>` slik programmet er nå, men verdien må beregnes manuelt med `TxUpper` $= N/2{,}56$.

8.5.5 Skalering og pakking av data

Skaleringsteknikken brukt for komprimering av data før sending (se 5.2.2) vil redusere nøyaktigheten på analyseresultatene. Det er nå definert at histogram-data skaleres til én byte, FFT-data til to byte og statistikkparametre til fire byte hver. Skulle det oppstå en situasjon hvor høyere (eller lavere) presisjon og oppløsning ønskes, kreves det en endring av programkoden og tilhørende rekompilering. Et forslag til videre forbedring er derfor å opprette nye elementer i konfigurasjonsfilen som bestemmer databredden for hver enkelt analyseobjekt.

I praksis viste det seg at lengdefeltet i datapakkene burde utvides til to byte. Siden feltet bare er én byte, er maksimal lengde 255 byte. Dette er en begrensning som fort kan komme til hinder, spesielt dersom et FFT-spekter med høy oppløsning ønskes overført.

8.5.6 Kontroll av overført datamengde

Senderprogrammet teller hele tiden antall byte hvert enkelt senderskjema overfører vha. DTAC. Antallene lagres i en databasetabell som beskriver senderskjemaene. Tabellen inneholder fem kolonner: skjemaindeks, frekvens, tidspunkt for første sending, tidspunkt for siste sending og antall byte sendt mellom de to tidspunktene. Programmet oppdaterer alle kolonnene for hver sending, men informasjonen blir ikke benyttet videre til kontroll av overført datamengde i forhold til abonnementsbegrensningene (12 KiB/md.). Kontrollen er overlatt til den som konfigurerer systemet (lager XML-filene for hver enkelt båt). Overført datamengde per måned må beregnes manuelt ut ifra senderfrekvens, opplysningene i 5.1.3 og vedlegg B som forteller hvor mange byte hver data- og overføringspakke er.

Videreutvikling av senderprogrammet bør implementere en funksjon der overført datamengde blir redusert, eventuelt stanset helt, dersom abonnementsgrensen overskrides. En praktisk løsning på dette er å innføre *prioritet*, enten på analysenivå eller senderskjemanivå. Dette gjøres ved å ha egne `<Priority>`-elementer i konfigurasjonsfilen. Prioriteten kan angis med et heltall, der en høy verdi angir høy prioritet. Når antall byte nærmer seg grensen, faller de lavest prioriterte analysene/senderskjemaene gradvis ut.

I begynnelsen av hver måned må bytetellerne og tidspunktene for første og siste sending nulles ut, dette er foreløpig ikke implementert.

8.5.7 Forespørsler fra land til skip

DTAC-enheten kan både sende og motta data, slik det går fram i vedlegg C og D. Sending av data fra land til skip kan utnyttes for å få et enda mer fleksibelt tilstandsovervåkningssystem.

Det første bruksområdet er forespørsler om å få tilsendt et spesifikt analyseresultat eller senderskjema. I situasjoner hvor data som sendes ofte, f.eks. daglig, brått viser store forandringer eller feiltilstander, kan det være interessant å innhente mer data enn vanlig. En forespørsel fra land kan derfor trigge sending av alle senderskjema, visse senderskjema eller visse analyseresultater. På denne måten kan senderfrekvensene overstyres ved behov. Forespørselen kan også brukes til å starte en analyse på nytt, siden en sending fører til dette.

Det andre bruksområdet kan være oppdatering av konfigurasjonsfilen til programmene ombord. Det fins mange grunner til at dette kan være ønskelig, f.eks. hvis det etter en tid blir klart at data som overføres har liten verdi, for lite eller for mye data overføres, eller ved fysiske endringer på thrustersystemet. Ved oppdatering av XML-filen kreves en omstart av programmet, noe som enkelt kan implementeres.

Det tredje bruksområdet er henting av skipets posisjon. Ved å sende en kommando til DTAC kan den returnere sin egen posisjon (gitt at den har GPS-dekning). Posisjonsinformasjon kan brukes til tracking av skip, noe som kan være nyttig med hensyn til tilstandsovervåking. Eksempelvis kan det raskt avgjøres om skipet opererer i tropisk klima (områder med høy havtemperatur), som kan føre til at thrusteren raskere blir full av groe. Et alternativ til å hente posisjonen manuelt med en forespørsel fra land, er at dette legges inn som en egen analyse eller et eget senderskjema på lik linje med de som allerede eksisterer. Da kan vi oppnå en periodisk posisjonsoppdatering.

Praktisk implementasjon av mottak av data fra DTAC-enheten må enten gjøres i hovedprogrammet, siden dette kjører kontinuerlig, eller i et nytt, dedikert program. I hovedprogrammet kan det opprettes en egen tråd, i tillegg til signalstitrådene, som lytter på meldinger fra DTAC. Utfordringen med å ha en lyttetråd er at senderprogrammet må få tilgang til å sende data til DTAC (og leses status tilbake) ved behov, samtidig som lyttertråden kjører. Fordelen er at meldinger som går rett til hovedprogrammet kan styre signalstier direkte, i motsetning til et dedikert program som må gjøre dette indirekte, f.eks. ved å oppdatere XML-filen.

Kapittel 9

Konklusjon

Gjennom arbeidet med denne masteroppgaven er det utviklet et funksjonelt tilstandsovervåkningssystem for thrustersystemer. Tilstandsovervåkningssystemet består i praksis av en industriell datamaskin som samler inn, prosesserer og analyserer data, og en dedikert satellittkommunikasjonsenhet, DTAC, som overfører analyseresultater fra skipet hvor systemet er installert til land. Det er utviklet to programmer som kjører på den industrielle datamaskinen, et «hovedprogram» og et «senderprogram»:

Hovedprogrammet henter inn, og behandler data. Det er oppbygd av moduler eller *blokker* med ulik funksjonalitet som enten prosesserer eller analyserer data. Blokkene kan settes sammen i vilkårlig rekkefølge og antall ved hjelp av en konfigurasjonsfil som hovedprogrammet leser. Dette gjør systemet meget fleksibelt, og det kan brukes på alle thrustertyper og -anlegg. Rapporten presenterer teorien som ligger bak de forskjellige blokkene, med fokus på robuste teknikker for implementasjon. Implementerte prosesseringsblokker er IIR-filter, derivasjon, effektiv- og absoluttverdi. Disse blokkene tar inn data, behandler og sender den videre til neste blokk. Implementerte analyseblokker er statistikkberegninger, histogram og FFT-analyse. Analyseblokkene avslutter hver enkelt «signalsti», det vil si kjeden av blokker for hver signalkilde, og lagrer sine analyseresultater i en database.

Senderprogrammet startes periodisk og sender spesifiserte analyseresultater til DTAC, som igjen videresender disse til land over satellittnettverket Iridium. Analyseresultatene blir komprimert ved skalering ned til mindre datatyper og deretter pakket etter en spesifisert protokoll. Dette reduserer datamengden som må sendes over satellitt. Protokollen som er utviklet sikrer riktig sammensetning og dekoding av data ved utpakking. På mottakersiden er det utviklet et tredje program, som gjør nettopp dette.

Mottaksprogrammet lagrer dekodet data på strukturert og søkbar form i en database. Innenfor tidsrammen til prosjektet ble det ikke rom for utvikling av et eget presentasjonsverktøy, men krav og forslag til et slikt verktøy er drøftet.

Rapporten beskriver valg av måleparametre og analyser som gir mest mulig informasjon om tilstanden til systemet. Eksempler på dette er beregning av gjennomsnittlig pitchvinkelhastighet og generering av histogrammer over motorens turtall eller belastning. Parametre og analyser som krever videreutvikling av systemet eller ekstra måleutstyr drøftes også, f.eks. høyhastighetsinnsamling av vibrasjonsdata og analyse av partikkelinnhold i hydraulikk- eller girolje.

Tre forskjellige løsninger for kommersialisering av tilstandsovervåkningssystemet er foreslått:

- Lavhastighets datainnsamling *uten* satellittkommunikasjon. Dette er det rimeligste alternativet og det som er enklest å installere, siden montering av DTAC ikke er nødvendig.
- Lavhastighets datainnsamling *med* satellittkommunikasjon. Denne løsningen har vært i fokus for tilstandsovervåkningssystemet som er utviklet i dette prosjektet.
- Høyhastighets datainnsamling med satellittkommunikasjon. Ved å ha eksternt måleutstyr, f.eks. enheter fra National Instruments som Brunvoll AS benytter fra før, muliggjør dette høyere punktprøvefrekvenser. Dette gir mer nøyaktige analyser og gjør det mulig å måle f.eks. vibrasjon.

Systemet er prøvd ut i praksis, med både målt data fra fysiske thrusteranlegg og data fra bedriftens thrustersimulator. Programmene fungerer like godt med statiske databaser med tidligere innsamlet data, som med databaser som fylles dynamisk av simulatoren. Resultatene viser at de ulike prosesserings- og analyseblokkene fungerer som de skal, avvikene på grunn av komprimering er så små at de kan neglisjeres. Unntaket er FFT-analysen hvor en del støy er observert i det genererte frekvensspekteret. Dette skyldes at en lite optimalisert algoritme ble implementert.

Data fra ulike skip og thrustersystemer vil samles opp på mottakersiden etter at tilstandsovervåkningssystemet har vært i bruk en periode. Databasen over analyseresulter danner etterhvert et sterkt statistisk grunnlag ved f.eks. videreutvikling og optimalisering av thrustersystemene. Integreres tilstandsovervåkningssystemet opp mot eksisterende programvare som brukes ved serviceoppdrag, kan driftskondisjonsdata sammenholdes med statistikk over feilsituasjoner og omsetning på reservedeler. På denne måten dannes et enda sterkere statistisk grunnlag.

Referanser

[1] Mohamad Adnan Al-Alaoui. «Novel IIR differentiator from the Simpson integration rule». I: *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on* 41.2 (1994), s. 186–187. ISSN: 1057-7122. DOI: 10.1109/81.269060.

[2] André Skare Berg, Espen Løkseth og Kristian Torsvik. «Loggekoffert». Bacheloroppgave. Kjølnes ring 56, 3918 Porsgrunn: Høgskolen i Telemark, 2011.

[3] Robert Grover Brown og Patrick Y. C. Hwang. *Introduction to Random Signals and Applied Kalman Filtering with MATLAB exercises and solutions*. 3. utg. John Wiley og Sons, 1997.

[4] Rosetta Code. *Fast Fourier transform – C++*. 2013. URL: http://rosettacode.org/wiki/Fast_Fourier_transform#C.2B.2B (besøkt 13.03.2013).

[5] James W. Cooley og John W. Tukey. «An algorithm for the machine calculation of complex Fourier series». I: *Mathematics of computation* 19.90 (1965), s. 297–301. DOI: 10.2307/2003354.

[6] David Craig. *Three most common conventions for the Fourier transform*. 2007. URL: http://www.wtamu.edu/~dcraig/PHYS4340/070413_FTconventions.pdf (besøkt 13.04.2013).

[7] Lorraine Denby og Colin Mallows. «Variations on the Histogram». I: *Journal of Computational and Graphical Statistics* 18.1 (2012), s. 21–31. DOI: 10.1198/jcgs.2009.0002. URL: http://pubs.research.avayalabs.com/pdfs/ALR-2007-003-paper.pdf (besøkt 24.02.2013).

[8] Leif Gunar Ellingsen. *DTAC PROTOCOL rev. 2*. Powex AS. 2013.

[9] Tony Fisher. *Interactive Digital Filter Design*. 1999. URL: http://www-users.cs.york.ac.uk/~fisher/mkfilter/ (besøkt 23.04.2013).

[10] Matteo Frigo og Steven G. Johnson. «The Design and Implementation of FFTW3». I: *Proceedings of the IEEE* 93.2 (2005). Spesialutgave om «Program Generation, Optimization, and Platform Adaptation», s. 216–231. DOI: 10.1109/JPROC.2004.840301.

[11] Haitham Hassanieh mfl. «Simple and practical algorithm for sparse Fourier transform». I: *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '12. Kyoto, Japan: SIAM, 2012, s. 1183–1194. URL: http://dl.acm.org/citation.cfm?id=2095116.2095209 (besøkt 14.04.2013).

[12] Aiwina Heng mfl. «Rotating machinery prognostics: State of the art, challenges and opportunities». I: *Mechanical Systems and Signal Processing* 23.3 (2009), s. 724–739. ISSN: 0888-3270. DOI: 10.1016/j.ymssp.2008.06.009. URL: http://www.sciencedirect.com/science/article/pii/S0888327008001489 (besøkt 14.05.2013).

[13] Evans M. Harrell II og James V. Herod. *Linear Methods of Applied Mathematic – Orthogonal series, boundary-value problems, and integral operators*. 1997. URL: http://www.mathphysics.com/pde/filtering.html (besøkt 24.02.2013).

[14] Inpower AS. *INPOWER demonstrerer nyskaping i samarbeid med Brunvoll og Fjord1 MRF*. 2011. URL: http://www.inpower.no/nyheter/inpower-demonstrerer-nyskaping-i-samarbeid-med-brunvoll-og-fjord1-mrf (besøkt 06.04.2013).

[15] Modbius Institute. *Vibration Training Course Book*. 280 Myers Road, Merricks North, Victoria, 3926, Australia: iLearn interactive, 2005.

[16] Andrew K. S. Jardine, Daming Lin og Dragan Banjevic. «A review on machinery diagnostics and prognostics implementing condition-based maintenance». I: *Mechanical Systems and Signal Processing* 20.7 (2006), s. 1483–1510. ISSN: 0888-3270. DOI: 10.1016/j.ymssp.2005.09.012. URL: http://www.sciencedirect.com/science/article/pii/S0888327005001512 (besøkt 14.05.2013).

[17] Krzysztof Jemielniak. «Commercial Tool Condition Monitoring Systems». I: *The International Journal of Advanced Manufacturing Technology* 15 (10 1999), s. 711–721. ISSN: 0268-3768. DOI: 10.1007/s001700050123.

[18] Donald E. Knuth. *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms*. 3. utg. Bd. 2. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1998, s. 232. ISBN: 0-201-89684-2.

[19] Donald E. Knuth, Tracy Larrabee og Paul M. Roberts. *Mathematical Writing*. 14. Mathematical Association of America, 1989. URL: http://tex.loria.fr/typographie/mathwriting.pdf (besøkt 05.03.2013).

[20] Kai Forsberg Kristensen. *Statistikk HiT-TF, høsten 2010. Forelesningsnotater med Maple-kommandoer*. Kjølnes ring 56, 3918 Porsgrunn: Høgskolen i Telemark, 2010.

[21] Wu-Sheng Lu og T. Hinamoto. «Optimal design of IIR digital filters with robust stability using conic-quadratic-programming updates». I: *Signal Processing, IEEE Transactions on* 51.6 (2003), s. 1581–1592. ISSN: 1053-587X. DOI: 10.1109/TSP.2003.811229.

[22] Richard G. Lyons. *Understanding Digital Signal Processing*. 2. utg. Upper Saddle River, New Jersey 07458: Pearson Education, 2004. URL: http://flylib.com/books/en/2.729.1.109/1/ (besøkt 14.05.2013).

[23] MATLAB R2012b. *filter – 1-D digital filter*. 2012. URL: http://www.mathworks.se/help/matlab/ref/filter.html#f83-1015962 (besøkt 28.01.2013).

[24] National Instruments. *FFT Fundamentals (Sound and Vibration Measurement Suite)*. 2008. URL: http://zone.ni.com/reference/en-XX/help/372416B-01/svtconcepts/fft_funda/ (besøkt 15.05.2013).

[25] Nick Parlante. *Binary Trees*. 2001. URL: http://cslibrary.stanford.edu/110/BinaryTrees.html (besøkt 25.03.2013).

[26] Luca Pivano, Tor Arne Johansen og Øyvind N. Smogeli. «A Four-Quadrant Thrust Estimation Scheme for Marine Propellers: Theory and Experiments». I: *Control Systems Technology, IEEE Transactions on* 17.1 (2009), s. 215–226. ISSN: 1063-6536. DOI: 10.1109/TCST.2008.922602.

[27] John G. Proakis. *Digital Signal Processing: Principles Algorithms and Applications*. 4. utg. Upper Saddle River, New Jersey 07458: Pearson Prentice Hall, 2007.

[28] Hideaki Shimazaki og Shigeru Shinomoto. «A method for selecting the bin size of a time histogram». I: *Neural Computation* 19.6 (2007), s. 1503–1527. DOI: 10.1162/neco.2007.19.6.1503. URL: http://toyoizumilab.brain.riken.jp/hideaki/res/histogram.html (besøkt 24.02.2013).

[29] George William Slade. «The Fast Fourier Transform in Hardware: A Tutorial Based on an FPGA Implementation». I: *ResearchGate* (2013). URL: http://www.southernfriedsilicon.com/FFTTutorial121102.pdf (besøkt 14.05.2013).

[30] Julius Orion Smith. *Introduction to Digital Filters with Audio Applications*. Stanford, California 94305 USA: Center for Computer Research in Music og Acoustics (CCRMA), 2007. URL: https://ccrma.stanford.edu/~jos/filters/filters.html (besøkt 18.02.2013).

[31] Julius Orion Smith. *Introduction to Digital Filters with Audio Applications*. Stanford, California 94305 USA: Center for Computer Research in Music og Acoustics (CCRMA), 2007. URL: https://ccrma.stanford.edu/~jos/fp/Direct_Form_II.html (besøkt 01.03.2013).

[32] SQLite. *Atomic Commit In SQLite*. 2013. URL: http://www.sqlite.org/atomiccommit.html (besøkt 25.04.2013).

[33] Eric W. Weisstein. *Root-Mean-Square*. 2013. URL: http://mathworld.wolfram.com/Root-Mean-Square.html (besøkt 05.03.2013).

[34] B. P. Welford. «Note on a Method for Calculating Corrected Sums of Squares and Products». I: *Technometrics* 4.3 (1962), s. 419–420. ISSN: 00401706. DOI: 10.2307/1266577. URL: http://www.jstor.org/stable/1266577 (besøkt 05.01.2013).

[35] Murray Wiseman. *A History of CBM (condition based maintenance)*. 2009. URL: http://www.omdec.com/moxie/Technical/Reliability/a-history-of-cbm.shtml (besøkt 23.03.2013).

Vedlegg A

Oppgavetekst

A.1 Bakgrunn

Tilstandsovervåking av utstyr ombord på skip kan gjøres på mange ulike måter. Systemer finnes kommersielt tilgjengelig, og ulike aktører tilbyr tjenester for analyse av tilstanden på maskineri. Brunvoll ønsker å videreutvikle sine verktøy for å gjøre dette på selvstendig basis. Vi ønsker å forbedre metoder for innsamling av data, gjøre komprimering og analyse og innføre nye rutiner rundt overføring, lagring og presentasjon av data knyttet til spesifikke thrustersystemer. Brunvoll AS vil gjøre dette i et autonomt system, altså at det kan operere uavhengig av systemkonfigurasjonen og infrastruktur om bord i ulike båter.

A.2 Oppgavebeskrivelse

Oppgaven vil bestå i å konstruere et komplett tilstandsovervåkningssystem som skal overføre thrusterdata fra en aktuell båt til Brunvoll sentralt. For å oppnå dette må det utvikles effektive metoder for å overføre data via et eget dedikert satellittkommunikasjonssystem. Tilbydere av satellittkommunikasjon har ofte en fastavgift per måned på en gitt datamengde som det skal tilstrebes å ha kontroll på. Dette innebærer utfordringer rundt seleksjon, komprimering, preprosessering, og analyse av viktige thrusterdata. Tilstandskontrolldata vil inkludere målte størrelser som vil variere med driftskondisjon, og andre som i mindre eller liten grad avhenger av driftskondisjon. Parametre som varierer med driftskondisjon er f.eks. data fra drivsystemet som last, turtall, propellerstigning, rorvinkel, kommando og tilbakemeldingssignal, oljetrykk og vibrasjonsdata. Parametre som i liten grad varierer med driftskondisjon er f.eks. data fra smøre- og hydraulikkoljesystemene som nivå, temperatur, vann- og partikkelinnhold. Besvarelsen bør inneholde en drøfting av fornuftige tilstandskontrollparametre for thrustere som operere under ikke-stasjonære forhold. Dette vil Brunvoll

hjelpe til med å identifisere. Analysemetoder for ulike signaltyper bør også drøftes (f.eks. statistiske egenskaper, spektralanalyse og metoder for trending).

Når dataene er overført, skal disse håndteres hos Brunvoll AS sentralt. Dataene skal lagres forsvarlig og på et lesbart format, det vil si at de skal kunne være søkbare med tanke på båt, thrusternummer, datatyper osv. Videre bør dataene kunne presenteres vha. et egnet verktøy som kan vise nylige, historiske og statistiske diagrammer. Løsningen skal bygges fysisk og kobles opp mot et faktisk satelittsystem. Utstyret skal testes og justeres ved hjelp av en datalogger, simulatorer og en satellittkommunikasjonsenhet. Dataloggeren kobles til flere thrustersimulatorer, og samler data herfra. På landsiden skal verktøyet for lagring og presentasjon av data demonstreres. De mest sentrale verktøyene å jobbe i vil være C++, Linux, SQL som databaseserver og en egen automasjonsprogramvare.

Ute på båt vil vi bruke eksisterende infrastruktur mellom thrustere og kontrollsystemet for å logge operasjonsdata og parametre. Data vil bli logget til en eller flere egne loggeenheter, og være tilgjengelig som SQL-database(r) over ethernet. Satellittkommunikasjonsenheten benytter CAN-buss, slik at tilstandskontrollsystemet må fungere som et bindeledd mellom SQL-databasene (ethernet) og CAN-bussen. Det må bestemmes hvilke data som skal overføres, hvor ofte og på hvilket format, samt etableres en god komprimeringsrutine. Vi burde også se på muligheten for å trigge overføring av data fra sentralt hold. Dersom det blir praktisk mulig, bør systemet som bygges til slutt kunne testes på en faktisk båt.

Vedlegg B

Transmission Protocol Specification

This appendix describes the protocol developed for transmitting data through the DTAC and the Iridium network. The protocol consists of packets which is to be defined in this appendix. After the packets are defined, an example is presented to clarify the protocol (see B.3).

Each packet is illustrated by a diagram similar to:

`Field 1`	`Field 2`	`Field 3`	`Field 4`	`...`	`Field` n

where each field represents a byte or a multiple of bytes. A description follows each diagram to explain the fields. There are two packet types for encoding data in two levels: Transmission Packets and Data Packets. The latter are low-level packets that contain analysis-specific data described in B.2. Before transmitting the Data Packets, they are re-packed into Transmission Packets compatible with the DTAC Protocol specifications and limitations. If the data is to long to be fitted in one Transmission Packet, multiple packets are created. These are called Start-of-Transmission Packets and Packet Parts (Subpackets), respectively. All Transmission Packets have the two first fields: `Start` and `NoOfParts` in common:

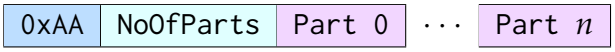
- The `Start` field is always 1 byte long and identifies the packet overall type: A value of `0xAA` indicates a Start-of-Transmission packet, while a value of `0xBB` means that the packet is a Packet Part (Subpacket).
- The `NoOfParts` field is always 1 byte long and defines the total number of Transmission Packets, i.e. the Start-of-Transmission Packet + the number of Subpackets.

Important encoding rules used when converting various datatypes to a stream of bytes are as follows:

- All values have big-endian encoding, so the most significant bytes comes first and least significant bytes comes last.
- All parameters are unsigned integer values unless otherwise specified.

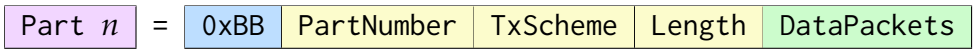
B.1 Transmission Packets

B.1.1 Start-of-Transmission Packet



Field	Size in bytes	Description
`0xAA`	1	A value of `0xAA` means that this is a Start-of-Transmission Packet.
`NoOfParts`	1	Total number of parts needed to transmit all data.
`Part` $0 \ldots n$	x	The «`Part` n» packets where $n > 0$ are transmitted separatly, only the first is transmittet together with the start-of-transmission information. The size is given in each Subpacket, as described below.

B.1.2 Packet Part (Subpacket)



Field	Size in bytes	Description
`0xBB`	1	A value of `0xBB` means that this is a Packet Part (also called a Subpacket).
`PartNumber`	1	Identifies the packet number. Valid data range is from 0 to `NoOfParts` $-$ 1, where `NoOfParts` is defined by the Start-of-Transmission Packet.
`TxScheme`	1	Identifies which transfer scheme the packet belongs to.
`Length`	1	Total number of bytes from `Length` to the end of the Subpacket, not counting `PartNumber`, `TxScheme` and `Length`.
`DataPackets`	`Length`	Data bytes in the form of (possibly) splitted Data Packets as defined in Subsection B.2.

B.2 Data Packets

B.2.1 Statistical Properties

ID = 0x02	Length = 0x14	Variance	Mean	Min	Max	Count

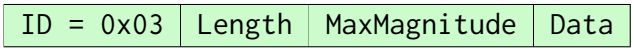
Field	Size in bytes	Description
ID	1	A value of 0x02 means this is Statistical data.
Length	1	Number of bytes from Length to end, not counting ID and Length. Always 0x14 (20 bytes) for this packet.
Variance	4	Statistical properties as IEEE 754 floating point values.
Mean	4	
Min	4	
Max	4	
Count	4	Number of datapoints used to generate the statistics. Unsigned integer value.

B.2.2 Histogram Data

ID = 0x01	Length	MaxBinLength	MaxBinValue	Data

Field	Size in bytes	Description
ID	1	A value of 0x01 means this is Histogram data.
Length	1	Number of bytes from Length to end, not counting ID and Length.
MaxBinLength	1	Number of bytes in MaxBinValue.
MaxBinValue	MaxBinLength	Largest frequency in the histogram. Unsigned integer value.
Data	1 for each bin	Bin frequencies scaled to 1 byte each.

B.2.3 FFT Spectrum

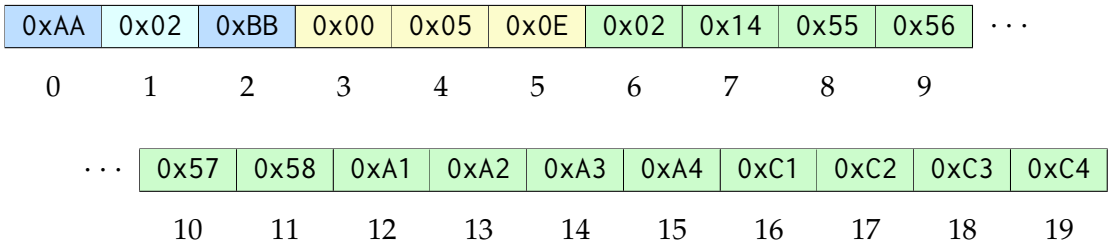


Field	Size in bytes	Description
`ID`	1	A value of `0x03` means this is FFT data.
`Length`	1	Number of bytes from `Length` to end, not counting `ID` and `Length`.
`MaxMagnitude`	4	Larges value in the FFT spectrum, 32 bits IEEE 754 floating point value.
`Data`	2 for each freq.	Scaled magnitude data for each line in the spectrum.

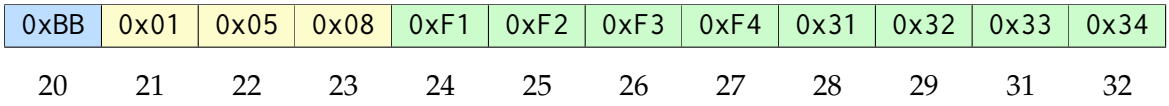
B.3 Example Data

This subsection presents example data to clarify how the protocol is used in practice. Concider the following 2-part data packet:

Part 0



Part 1



which consists of a single statistical data packet (not particulary realistic, but it serves as a good example). In total there is a payload of 22 bytes (green colored) to be transmitted. Because of the DTAC protocol only a certain amount of data is allowed in each packet. **I the example, a maximum size of 20 bytes is used to illustrate the functionality.** In reality the limit is 52 bytes, as defined by the DTAC Protocol document from Powex [8]. Each of the bytes are explained in Table B.1.

Table B.1: Description of each byte in the transmission example.

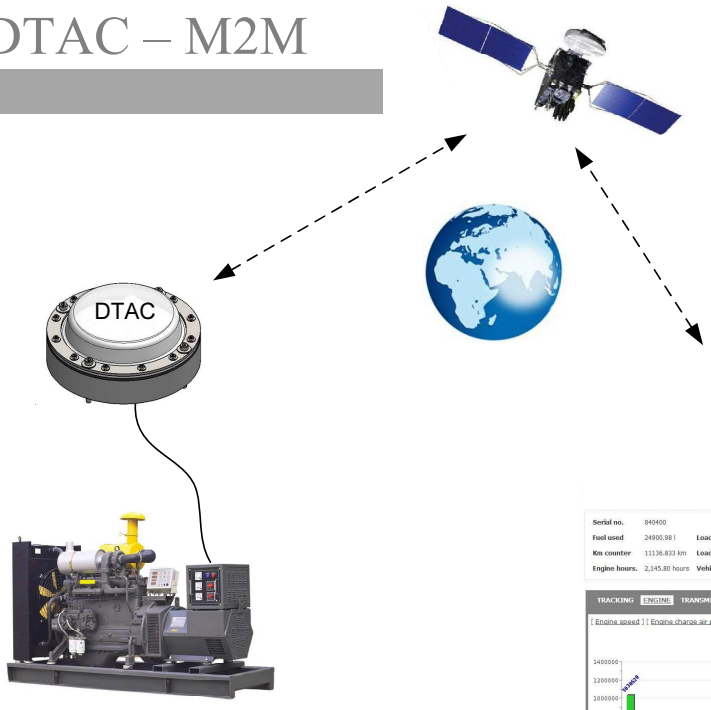
Byte	Value	Description
0	`0xAA`	This is the Start-of-Transmission Packet.
1	`0x02`	The data transaction consists of 2 parts.
2	`0xBB`	Start of Packet Part.
3	`0x00`	Part number 0. Always transmitted together with the three start-of-transmission bytes.
4	`0x05`	Transmission scheme 5.
5	`0x0E`	Part 0 carries 14 bytes of data.
6	`0x02`	This is the ID field of a data packet, `0x02` means statistical data.
7	`0x14`	The statistical data packet is 20 bytes long.
8–11	`--`	The variance as 4-byte floating point.
12–15	`--`	The mean value as 4-byte floating point.
16–19	`--`	The the minimum value as 4-byte floating point.
		We have now reached the last byte of the first packet, even if the statistic subpacket is not completed.
20	`0xBB`	Start of Packet Part.
21	`0x01`	Part number 1. All parts ≥ 1 are transmitted as separate packets over the Iridium network.
22	`0x05`	Transmission scheme 5.
23	`0x09`	Part 1 carries 9 databytes.
24–27	`--`	The 4-byte floating point maximum value.
28–32	`--`	Number of datapoints used to generate the statistics. Unsigned integer value.

Vedlegg C

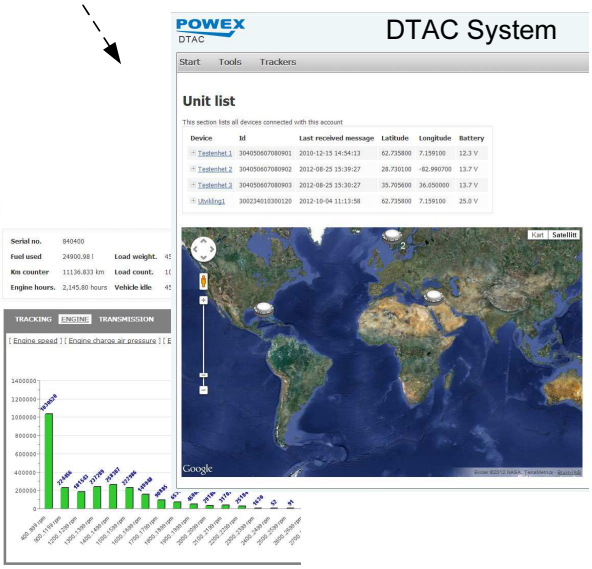
DTAC-info

Antall sider: 1

DTAC – M2M



OEM equipment



DTAC
"Device Tracking and Control" system provides communication by Iridium satellites from your desktop for tracking and control of mobile and remote assets all around the world

Applications:
- Remote control and monitoring
- Diagnostics
- Tracking
- Geo Fencing
- Event trigging and syncronization
- Buoys

Engineering:
POWEX provides engineering and automation services to interface our customers remote applications to the DTAC system.

DTAC can be customized to communicate with existing CRM/ERP systems, automating the customer logistic processes.

Product No.:

25119201	DTAC M2M IRIDIUM
25119202	DTAC M2M WEB APPLICATION
25119203	DTAC M2M GSM
25119204	DTAC M2M IRIDIUM w/ Signal light

Powex AS, Sofus Jørgensensv 5, 6415 Molde, Norway, T +47 71 24 78 00

Web: www.powex.no E.mail: office@powex.no

Vedlegg D

DTAC-spesifikasjoner

Antall sider: 3

Device Tracking and Control - DTAC

The DTAC system provides communication by Iridium satellites from your desktop for tracking and control purposes of mobile and remote assets all around the world.

Applications:

- Remote control and monitoring
- Diagnostics
- Tracking
- Geo Fencing
- Event trigging and synchronization
- Buoys

Product No.:

25119201	DTAC M2M IRIDIUM	Hardware
25119202	DTAC M2M WEB APPLICATION	Software

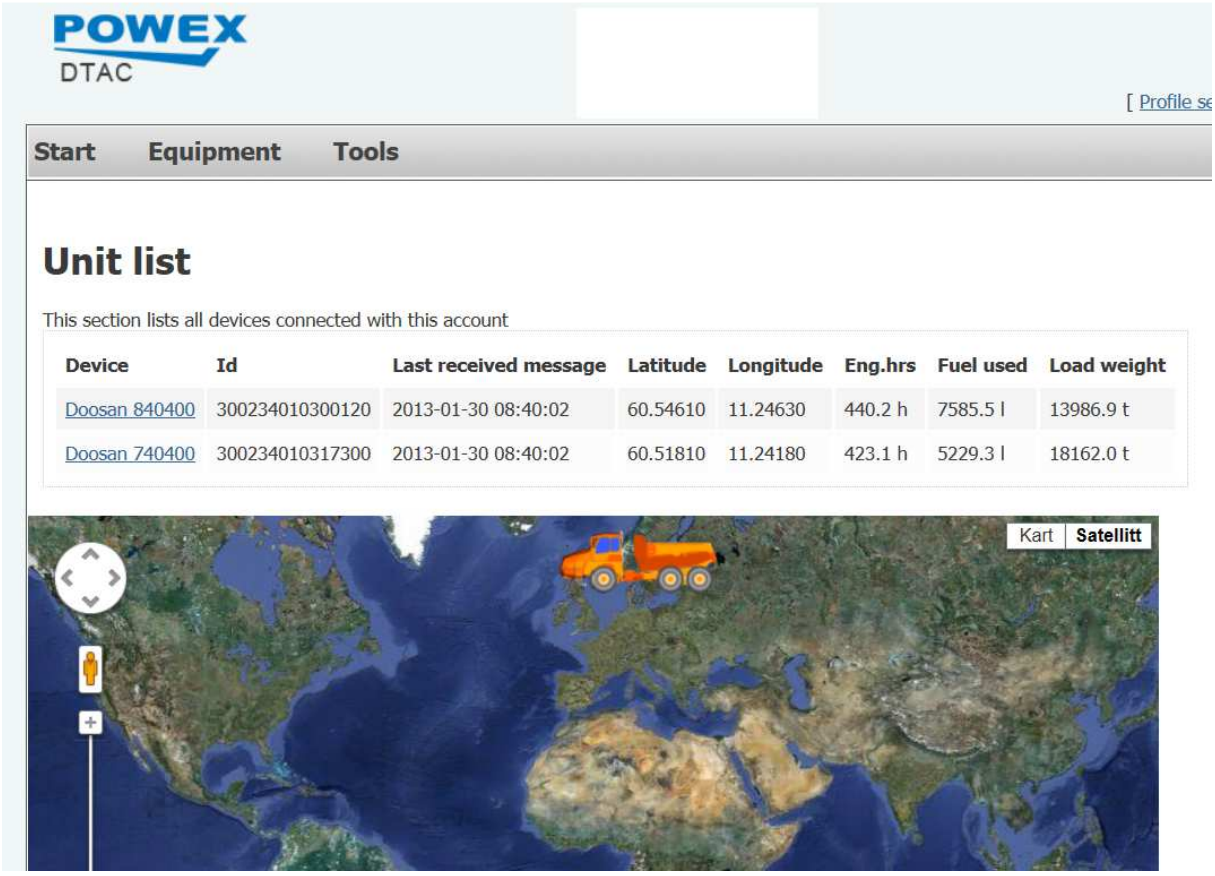
TECHNICAL DATA: 25119201 DTAC M2M IRIDIUM Hardware

Supply voltage:		9-32V	
Data communication		RS232	
		CAN (J1939)	
Current consumption		< 0.1Amp @ 12V	(operation)
		< 100uA @ 24V	(sleep)
Dimension:	diameter	164mm	
	height	75mm	
IP grade		IP67 (occasionally submerged to 10m)	
Temperature range		–30°C to +50°C	
GPS receiver		GPS L1 C/A	
		SBAS: WAAS, EGNOS, MSAS	
		Accuracy	Position 2.5 m CEP
			SBAS 2.0 m CEP
		Acquisition	Cold start: 34 s
			Warm start: 34 s
			Hot start: 1 s

Description: 25119202 DTAC M2M WEB APPLICATION Software

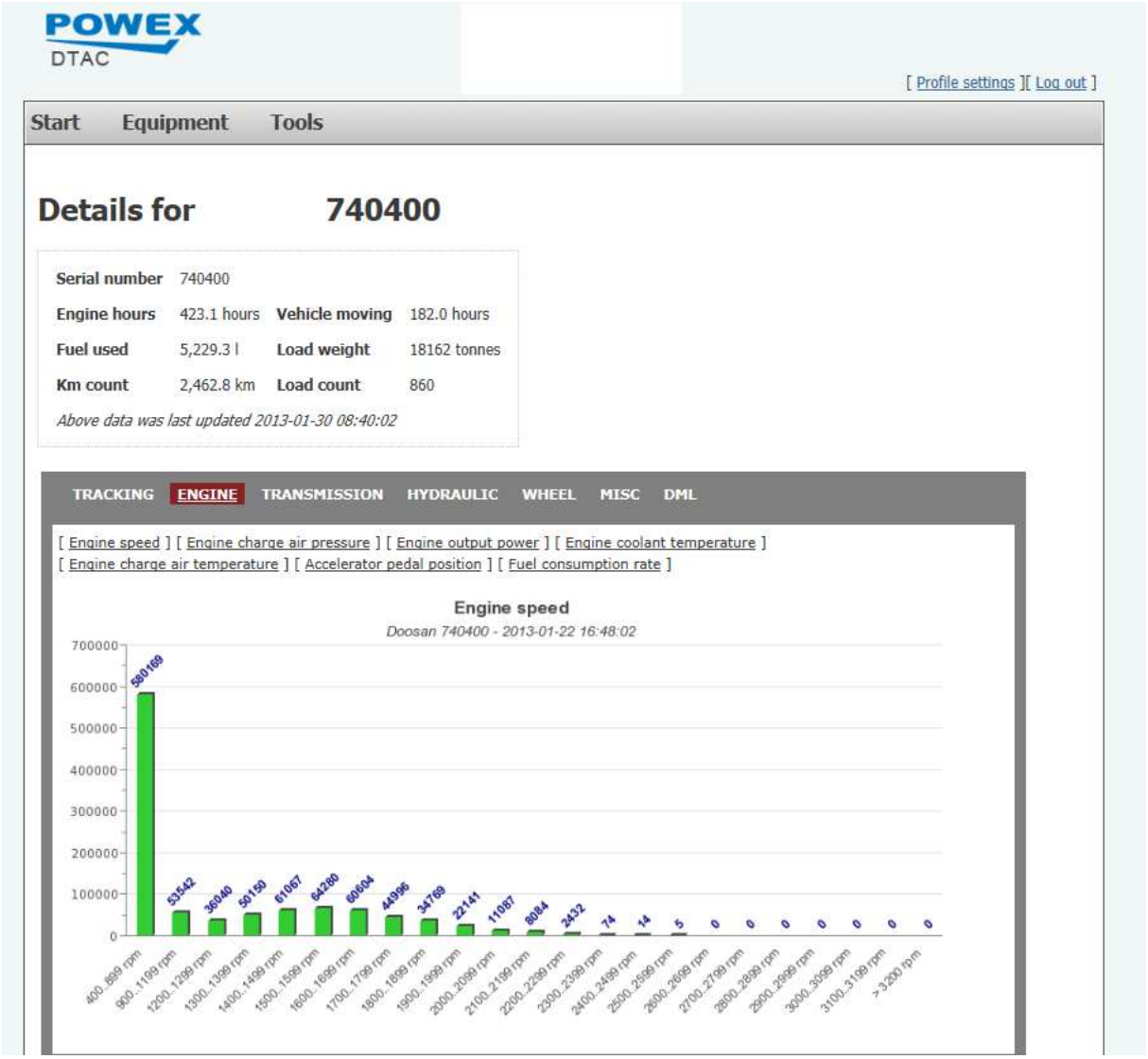
POWEX provides engineering and automation services to interface our customers remote applications to the DTAC system.

DTAC M2M WEB APPLICATION is a web based system capable of locating and retrieving information from remote located installations. This system is accessible worldwide only demanding an active internet connection at the clients end.
Using a given authorization credential, our customers can log into DTAC and in a few seconds get an overview of all registered units including their position lon/lat location. The integrated map can be zoomed and browsed freely. All registered units within the current map region are displayed at all time.



The unit list view is configurable regarding columns to display. Normally there will be only a condensed amount of information shown for all units while the detailed view is used to retrieve further information for a unit.
The monitored units send their location and position statuses in a predefined schedule (which can be reconfigured remotely). Any message will normally be accessible to the WEB system user within 1-2 minutes after delivery from a unit. Given that the unit is under open sky it will always get its messages sent, worldwide. Confirmation or error status' are always returned to the unit for each sent message allowing the unit to retry any bad transmission.

In addition to the unit list view one can also access detailed information from a given unit. By selecting a unit from the unit list an information view opens. The map will automatically center its position to the chosen unit's last reported location.



While the unit list view shows only the last status for each unit, the detailed view offers a complete historical view of virtually all data sent from the selected unit. The amount of different data available for a unit is decided by its configuration. The unit detailed view will automatically present all available information received, thus adding information headers as needed.

While the unit detailed view is configured automatically, the unit list view column configuration is made through an easy-to-use configuration menu in the WEB system interface. This enables the user to customize the unit list view showing only the most relevant information for all registered units.