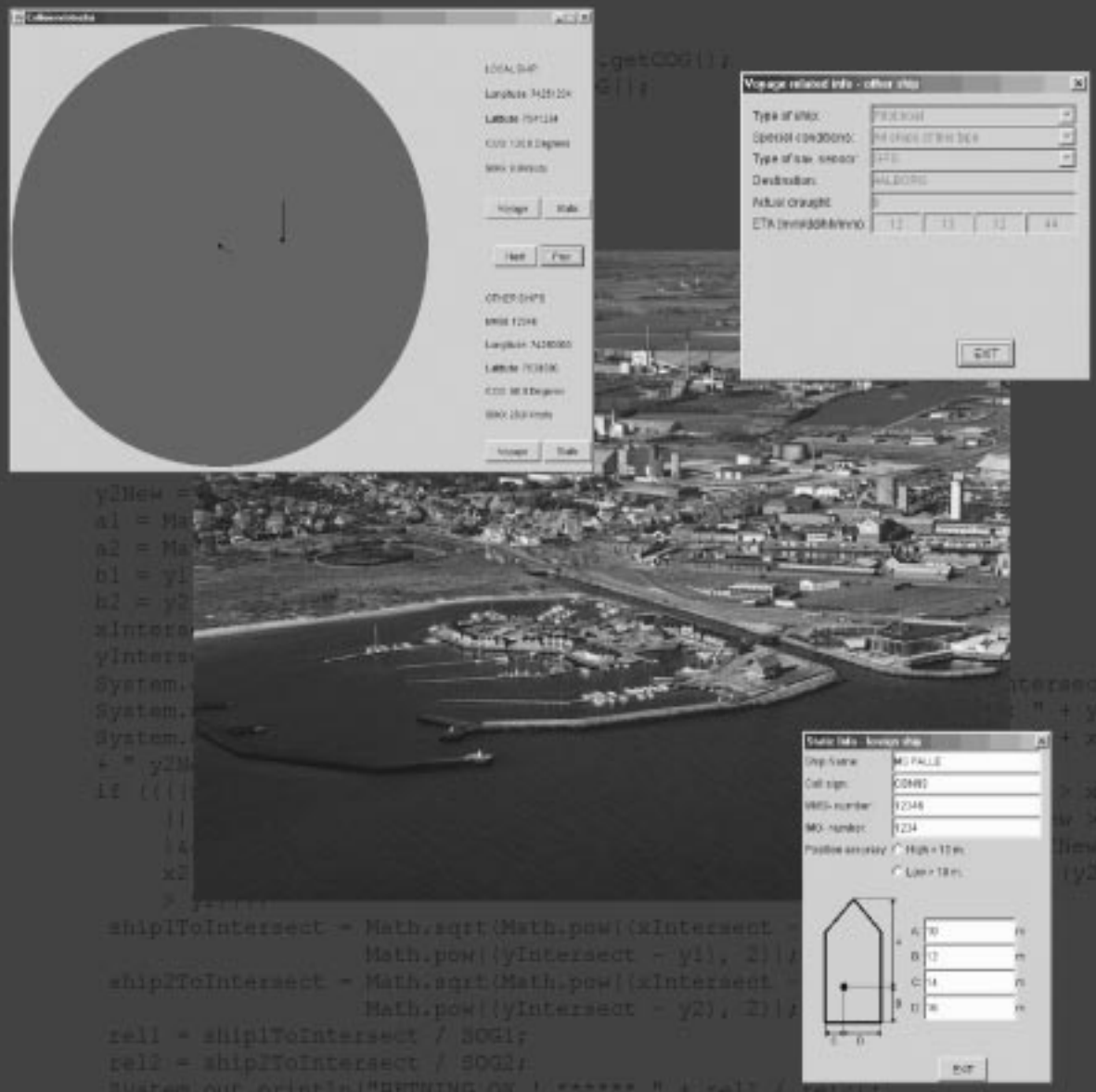
AIS-Receiver



Aalborg Universitet
Institut for Elektroniske Systemer
Gruppe 507

5. semester 2000

AALBORG UNIVERSITY
INSTITUTE OF ELECTRONIC SYSTEMS, I8
DEPARTMENT OF COMMUNICATION TECHNOLOGY
Fredrik Bajers Vej 7 • DK-9220 Aalborg Ø • www.ies.auc.dk

Title:
AIS-Receiver

Topic:
Contruction of devices / systems.

Project period:
5th Semester 2000.09.04 - 2000.12.19

Project group:
E5-507

Group members:
Lars Klitgaard Jakobsen
Christian Corfits Jensen
Kenneth Kristensen
Peter Ilsøe Nielsen
Jan Ozimek
Thomas Søhus

Supervisor:
Hans Ebert

Number printed: 9

Number of pages: 177

Abstract:
This project documents the design and development of a system that uses special radiosignals, emitted by ships, for collision detection.
From the year 2003 it becomes statutory for large ships to send out radiosignals regularly, that indicate the position, speed and several static and voyage related informations. This information is picked up by the system using a VHF radio. Furthermore the system obtains information about the position of own ship from a GPS receiver. All this informaiton is processed in order to determine if there is any risk of collisions. The static and voyage related information is stored, and on request they are displayed on a connected PC.
The system consists of a 8051based microcontroller, that manages the communication between GPS, VHF and PC, through interfaces. The programme for the microcontroller is written in the programming language “C”. The programme that has been written for PC, makes the calculations necessary to determine any risk of collisions. A graphical user interface has been implemented to present the information about the ships. The PC programme is written in the programming language JAVA.

AALBORG UNIVERSITY
INSTITUTE OF ELECTRONIC SYSTEMS, I8
DEPARTMENT OF COMMUNICATION TECHNOLOGY
Fredrik Bajers Vej 7 • DK-9220 Aalborg Ø • www.ies.auc.dk

Titel:
AIS-Receiver

Tema:
Apparat- og / eller systemkonstruktion.

Projektperiode:
5. Semester 04.09.2000 - 19.12.2000

Projektgruppe:
E5-507

Gruppemedlemmer:
Lars Klitgaard Jakobsen
Christian Corfits Jensen
Kenneth Kristensen
Peter Ilsøe Nielsen
Jan Ozimek
Thomas Søhus

Vejleder:
Hans Ebert

Oplag: 9

Sideantal: 177

Synopsis:
Rapporten omhandler konstruktionen af et system, som udnytter specielle radiosignaler udsendt af skibe, til kollisiondetektion.
Fra år 2003 bliver det lovpligtigt for større skibe at udsende radiosignaler, som angiver skibets position, hastighed og en række statiske og rejserelaterde informationer regelmæssigt. Disse informationer opsamles af systemet, vha. en VHF-radio. Desuden får systemet oplysninger om skibets egen position fra en GPS-modtager.
Informationerne om skibenes position sammenlignes med henblik på at bestemme, om der er fare for kollision med andre skibe. Desuden gemmes de statiske og rejserelaterede informationer, og det er muligt for brugeren at få vist disse på en tilhørende PC.
Systemet består af en 8051 baseret microcontroller, som vha. tilhørende interfaces sikrer kommunikationen mellem GPS, VHF og PC. Programmet til denne microcontroller er skrevet i programmeringssproget "C". Til PC'en er der skrevet et program, som foretager kollisionsberegning og præsenterer skibene og informationerne om disse i en grafisk brugerflade. Dette program er skrevet i programmeringssproget JAVA.

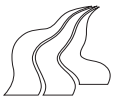
Forord

Denne rapport er udarbejdet på Aalborg Universitet, Institut for Elektroniske Systemer, som projekt på 5. semester i perioden 4. september til 19. december 2000, med temaet "Apparat- og / eller systemkonstruktion".

Rapporten henvender sig til studerende på Aalborg Universitet og øvrige, som måtte have interesse for de, i rapporten, behandlede emner.
Kildehenvisninger er angivet ved forfatterens efternavn, udgivelsesår og evt. side i firkantede parenteser som vist: [efternavn, udg. år, side]. Kilderne er nærmere angivet i litteraturlisten.
Henvisninger til appendiks er angivet ved et stort bogstav, som henviser til appendiksbogstavet.
Figurer, tabeller og grafer er nummereret efter hvilket afsnit, de er placeret i, hvor de første tal angiver, hvilket hovedafsnit de tilhører, og det sidste tal angiver figurens individuelle nummer.

Teksten og layoutet er lavet i Corel WordPerfect. Brødteksten er sat i Times New Roman.
Figurer og lignende er lavet i CorelDRAW, Corel Photo-Paint, OrCAD 9.1 og Microsoft Visio 5.0.
Kompilering af kildekode er fortaget med: µVision fra KEIL Software til microcontrolleren og Java Development Kit fra SUN til PC programmet.
Bagerst i rapporten er der en CD-ROM, som indeholder datablade på de anvendte komponenter, de dokumenterede programmer, samt den samlede rapport i PDF-format. CD'en vil endvidere indeholde øvrige projektrelevante data.

Lars Klitgaard Jakobsen	Peter Ilsøe Nielsen
Christian Corfits Jensen	Jan Ozimek
Kenneth Kristensen	Thomas Søhus



Indholdsfortegnelse

1 Indledning

I dette første afsnit gives en kort beskrivelse af nogle organisationer, hvis arbejde har indflydelse på det konstruerede system. Dernæst resumeres den standard, som systemet er bygget op efter, og selve systemet og testen af det beskrives kort.
Endeligt gives en vurdering af systemets anvendelighed.

Transporten af mennesker og gods er generelt stigende, hvilket blandt andet giver udslag i, at trafikken på havene eskalerer. Dette betyder, at der må træffes en række forholdsregler for at forhindre kollisioner og andre ulykker. Der er opfundet og indført en række systemer, som skal gøre det mere sikkert at færdes til søs. I den nærmeste fremtid indføres endnu et. Det er mere end noget andet maritimt system, et resultat af den stigende trafik. Systemet hedder AIS, og det bygger på kommunikation mellem skibe. I denne rapport beskrives konstruktionen af et apparat, som indgår i dette system.

1.1.1 Organisationer

Søfart er en verdensomspændende industri, og skibe er derfor underlagt en række regler, som er fastsat af internationale organisationer. I det følgende beskrives nogle organisationer, hvis arbejde har betydning for det pågældende system.

IMO:
"International Maritime Organization" er en organisation under UN (United Nations). IMO har til opgave at forbedre den maritime sikkerhed og at forhindre forurening fra skibe. Dette gøres ved at udstede og opdatere regler for søfart, og sikre at de bliver overholdt af så mange lande som muligt. Således gælder disse regler i dag for over 98% af verdens handelsskibstonnage. I dag lægges hovedvægten på at sikre, at de lande, der har underskrevet div. konventioner og traktater, også overholder og håndhæver dem. Det er vigtigt at understrege, at IMO ikke selv håndhæver de regler de udsteder; de enkelte lande indvilliger i at indføre IMO's regler i deres egen lovgivning, når de underskriver div. traktater m.m. Det er således landenes myndigheder, der håndhæver reglerne. I Danmark omsættes IMO's regler til dansk lovgivning af Søfartsstyrelsen.
[URL:www.imo.org]

ITU:
"International Telecommunication Union" er en international organisation, hvor regeringer og den private sektor koordinerer telekommunikation. ITU har næsten 200 medlemslande, og over 600 virksomheder er medlemmer.
ITU administrerer bl.a. brugen og tildelingen af MMSI-numre (Maritime Mobile Service Identity), som er unikke identifikationsnumre for skibe og maritime jordstationer. Disse numre bruges til adressering af radiobeskeder.
Under ITU findes ITU-R, som er ansvarlig for alt ITU's arbejde indenfor radiokommunikation.

Det er blandt andet ITU-R's opgave, at sikre at spektrum'et af radiofrekvenser udnyttes effektivt. [www.itu.int/itu-r]
Det er særdeles relevant at tage standarder og anbefalinger fra ITU-R med i betragtningen, når der designes radiokommunikationsudstyr. ITU-R har bl.a. udstedt en "recommendation" med en række anbefalinger til en protokol, som bør anvendes ved et system til automatisk udveksling af informationer mellem skibe; Et såkaldt "Automatic Identification System" (AIS). Denne recommendation betegnes herefter AIS-standarden.

1.1.2 AIS-standarten

Denne standard beskriver de protokolmæssige, tekniske specifikationer til et "Universal Shipborne Automatic Identification System". Protokollen er udarbejdet iht. OSI reference modellen, og lagene 1 (Physical Layer) til 4 (Transport Layer) er beskrevet.
Formålet med AIS-systemet er, at udveksle en række informationer mellem skibe indbyrdes og mellem skibe og havnemyndigheder.
Informationerne sendes og modtages via en VHF-radio på en reserveret, fast defineret frekvens og med en fast defineret bithastighed.
Til at styre adgangen til mediet benyttes en metode kaldet TDMA (Time Division Multiple Access). Det vil sige, at hver station (skib eller landstation) tildeles et givet tidsrum (slot) til at transmittere i. Afgrænsningen af disse slots defineres i forhold til UTC-tiden; et minut inddeles i et antal slots, hvori forskellige stationer må sende. Tildelingen af slots kan foregå efter forskellige metoder:
FATDMA: Fixed Access Time Division Multiple Access.
Denne metode benyttes, når en autoritet (styrende station) dikterer tildelingen af slots.
SOTDMA (Self Organized Time Division Multiple Access):
Denne metode benyttes når der ikke er en styrende station til at tildele slots. Stationerne finder automatisk ud af at allokere slots efter et givet mønster.
Der findes også andre metoder, som kun benyttes til synkronisering, når en ny station begynder at sende.

Der findes en række fast definerede beskeder (frames), som kan sendes. Informationerne i disse kan opdeles i følgende kategorier:

- Statiske informationer: IMO-nummer, navn, længde, bredde, skibstype og position af positionsangivende antenne.
- Rejserelaterede informationer: Dybdegang, farlig last, destination, forventet tid for ankomst og evt. rute.
- Dynamiske informationer: Position, UTC-tid, beholdende kurs, fart, styrende kurs, vendehastighed og status: ikke under kommando osv.
- Tekstbesked: Det er muligt at sende brugerdefinerede tekstbeskeder. Disse bør dog have lavere prioritet end de tre ovenstående punkter.
- MMSI-nummer: Hver besked udstyres med afsenderens MMSI-nummer, for at identificere denne.

De dynamiske informationer udsendes løbende med et interval på 2 til 12 sekunder, afhængigt af hvor hurtigt det pågældende skib sejler. De statiske og rejserelaterede informaioner udsendes hvert sjette minut eller på forespørgsel.
Fra år 2003 bliver det lovpligtigt, for større, erhvervsdrivende skibe at udsende informationer iht. standarden.

Informationerne kan fx. benyttes af havnemyndigheder til en hurtigere og mere effektiv fordeling

af kajpladser og lignende. De kan også benyttes til at styre trafikken gennem et meget trafikeret farvand. I dette projekt fokuseres, dog på en anden anvendelse: De dynamiske informationer om skibenes kurs, fart m.m. udnyttes til at beregne om der er fare for kollisioner. Dette gøres ved at sammenholde de informationer, et skib modtager fra sin egen GPS, med de, via AIS, modtagne informationer om andre skibe.

1.1.3 Selve systemet

Systemet på det enkelte skib består af følgende fire enheder:

- GPS-modtager
- VHF-radio
- PC
- Sammenkoblingsboks (Controller)

Disse enheder var oprindeligt tiltænkt at skulle udføre følgende funktioner:

- GPS-modtageren leverer de dynamiske informationer om skibet.
- VHF-radioen modtager informationerne fra andre skibe, og udsender tilsvarende informationer om eget skib.
- På PC'en indtastes de statiske og rejserelaterede informationer om skibet, og de informationer, der modtages fra GPS-modtager og VHF-radio, vises. Desuden foretager PC'en beregning af evt. kollisionsfarer på baggrund af de indsamlede data, og advarer brugeren, hvis der opstår kollisionsfare.
- Sammenkoblingsboksen består af en microcontroller samt nogle tilhørende kredsløb. Den har til opgave at fordele data mellem de øvrige enheder i systemet, samt at håndtere tildelingen af slots, og dermed kontrollere, hvornår skibet må sende.

Det system, der dokumenteres i rapporten, er en begrænset udgave af det oprindelige system. Det er kun i stand til at *modtage* informationer fra VHF-radioen, og dermed er de enkelte enheders funktioner ændret lidt:
GPS-modtageren leverer stadig dynamiske informationer til resten af systemet.
VHF-radioen skal, som beskrevet, kun modtage informationer fra andre skibe, og sende disse videre til sammenkoblingsboksen.
PC'en skal stadig vise de modtagne informationer og foretage kollisionsfareberegninger. Det skal være mulgt at indtaste rejserelaterede og statiske informationer om skibet, og disse skal gemmes sammen med informationerne om de andre skibe. Denne funktion beholdes, da det så vil være langt simplere at lave en evt. udvidelse af systemet.
Sammenkoblingsboksens opgave er blevet simplificeret, da den ikke længere skal håndtere allokering af slots; den skal blot opsamle data fra VHF-radioen og sende de relevante informationer videre til PC'en.

1.1.4 Test af apparatet

I rapporten er der ikke dokumenteret en test af systemet i sin helhed. Dette skyldes dels at visse dele af systemet ikke er implementeret til fulde, og dels at det ville kræve omfattende praktiske foranstaltninger at teste det:

- De relevante VHF-signaler skal være til rådighed, og kunne styres til at simulere signalerne fra et skib på kollisionskurs.
- Apparatet skal bevæge sig over større afstande, for at få de rigtige signaler fra GPS-modtageren. Denne bevægelse skal tilpasses informationerne i signalerne fra VHF-radioen.

Da bl.a. sende/modtagedelen af systemet ikke er implementeret, ville det under alle omstændigheder være umuligt at teste hele systemet.
For at teste den del af systemet som *er* implementeret, simuleres de manglende dele. Dette resulterede i et tilfredsstillende testresultat: Med forbehold for de ting som blot er simuleret, opfylder systemet de funktionelle krav.

1.1.5 Eksisterende apparater

Der eksisterer allerede en lang række elektroniske apparater, til at hjælpe med navigation: Radar, ekkolod og GPS. De har alle til formål at indsamle informationer om omgivelserne, og præcentere dem for navigatøren. Dette gælder også AIS-systemet, men der er den store forskel, at AIS-systemet bygger på *kommunikation*, hvor de andre systemer blot indsamler informationer og anvender dem *lokalt* på skibet. Da informationere udsendes direkte fra kilden, er de mere præcise og detaljerede, end dem de andre systemer kan vise. Dette berettiger inførelsen af endnu et system.

1.1.6 Det konstruerede systems anvendelighed

Det begrænsede system overholder ikke AIS-standardens krav om at udsende informationer, men det vurderes, at systemet stadig vil være nyttigt for lystsejlere og mindre skibe, som ikke er underlagt kravet i om at sende. Det er her detektionen af evt. kollisionsfarer, der fokuseres på.
Det kunne fx. være et supplement til en radar; der kan forekomme situationer, hvor en radar ikke er i stand til at lokalisere et skib, fordi det er dækket af en ø eller lignende. Det er mindre sandsynligt, at VHF-signalerne bliver blokeret, og desuden vil de altid give et entydigt billede, hvilket ikke er tilfældet med en radar.
Ydermere ville systemet kunne benyttes i et mindre skib, som ikke er udstyret med radar.

2 Kravsspecifikation

Her specificeres de krav, som på forhånd stilles til apparatet.
Ved accepttesten i kapitel 7, kontrolleres det, om systemet lever op til kravene.

2.1 Indledning

Formål
Formålet er at konstruere et system, som ud over identifikation af andre skibe, skal advare navigatøren, hvis der er fare for kollision med andre skibe. Dette gøres ved, at systemet dels bestemmer sin egen position, ved hjælp af en GPS-modtager, og dels ved at lytte til de signaler, som andre skibe udsender. De modtagne signaler behandles i et datamatsystem, og der udsendes en alarm, hvis der er fare for kollision.
Systemets navn er "AIS-Receiver".
Da der er tale om et indlæringsforløb, er det tilladt at ændre i kravsspecifikationen undervejs. Eventuelle ændringer vedtages på møder, hvor hele projektgruppen deltager.

2.1.1 Referencer

I denne rapport er der anvendt følgende standarder:
NMEA 0183, National Marine Electronics Association, Standard For Interfacing Marine Electronic Devices, Version 2.30, March 1st 1998 [Cassidy, 1998].

ITU-R M.1371, Recommendation ITU-R M.1371, Technical Characteristics For A Universal Shipborne Automatic Identification System Using Time Division Multiple Access In The VHF Maritime Mobile Band, 1998 [AIS, 1998].

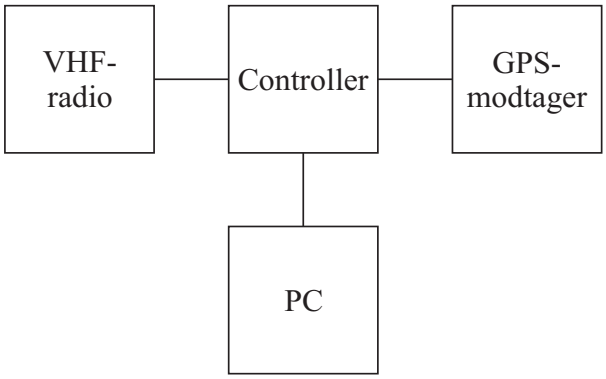
Ovenstående standarder forefindes på medfølgende CD.

2.1.2 Læsevejledning

Denne kravsspecifikation er baseret på den model, som er beskrevet i bogen "Håndbog i Struktureret Program-Udvikling"[SPU,1998]. Idet ikke alle punkter i SPU's kravsspecifikation gør sig gældende i vores system, er enkelte punkter udeladt. Det forudsættes således, at læseren er bekendt med omtalte model.

2.2 Generel beskrivelse

2.2.1 Sytembeskrivelse / systemfunktion



Figur 2.1 Dataflow for systemet.

Systemet opbygges som et datamatsystem, som modtager skibets egen position, fart og kurs fra en GPS, for derefter at udsende disse ved hjælp af en VHF-radio. Systemet modtager ligeledes andre skibes position, fart og kurs. Disse skal sorteres således, at kun de skibe, der er indenfor en nærmere fastsat grænse, behandles med henblik på udregning af kollisionsfare.
Systemet skal opbygges således, at det overholder AIS-standarden (ITU-R M.1371), som er en maritim kommunikationsstandard for skibe og havnemyndigheder [AIS, 1998].

2.2.2 Systemets begrænsninger

I tilfælde af kollisionsfare skal systemet ikke kunne komme med forslag til kurs- eller hastighedsændring. Endvidere skal systemet ikke kunne advare om risiko for grundstødning.

2.2.3 Systemets fremtid

Da AIS-standarden ikke er en endelig standard, og dermed kan indeholde fejl og mangler, skal det være muligt, at ændre og tilpasse den i standarden definerede protokol. Den anvendte protokol bør derfor, så vidt muligt, opdeles i lag, således at det er nemt at udskifte et enkelt lag, hvis dette skulle blive nødvendigt.

2.2.4 Brugerprofil

Systemet vil kun have een type bruger, og denne benævnes navigatør.
Det forventes ikke, at navigatøren har indgående kendskab til betjening af EDB-udstyr, og derfor skal brugerfladen være simpel.

2.2.5 Krav til udviklingsforløbet

Idet der er tale om et undervisningsforløb, stilles der i studieordningen for E5 krav til, hvordan systemet skal udvikles. Studieordningens krav er som følger:

- *"Der tages udgangspunkt i en konkret fysisk problemstilling som kræver anvendelse af et datamatsystem i forbindelse med løsningen.*
- *Det fysiske system beskrives, hvorefter der foretages en analyse og modeldannelse.*
- *De nødvendige grænseflader specificeres.*
- *Der foretages en specifikation af hardware og software i det samlede system.*
- *Der foretages design og konstruktion af udvalgte hardwaredele.*

- *Der foretages udvikling af software mhp. realisering af det samlede systems funktionalitet.*
- *Det samlede system testes mhp. verificering af systemets performance iht. problemspecifikationen."*

2.2.6 Forudsætninger

Det forudsættes, at brugeren har en VHF-radio, der kan modtage de data, som afsendes fra systemet, og udsende dem som defineret i AIS-standarden. Da der skal anvendes en PC til beregning af kollisionsfare, samt til visning og indtastning af data, forudsættes det at brugeren har en sådan til rådighed. For at tilgodese de brugere, som allerede er i besiddelse af en GPS-modtager, udvikles systemet således, at der er et stik til en ekstern GPS-modtager.

2.3 Specifikke krav

2.3.1 Funktionelle krav

Her angives de funktionelle krav til systemet. I de følgende punkter gives en detaljeret beskrivelse af funktionerne fra systembeskrivelsen.

Afsendelse og modtagelse af data

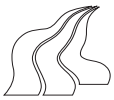
Apparatet skal kunne sende og modtage informationer iht. AIS-standarden: ITU-R M.1371 [AIS, 1998]. Dette medfører, at der skal udsendes og modtages nedenstående informationer:

Dynamiske informationer:
- Skibets position.
- UTC tid.
- Skibets beholdende kurs (COG).
- Skibets fart (SOG).
- Skibets styrende kurs (Heading).
- Skibets kursændring.

Rejserelaterede informationer:
- Skibet dybdegang (Draught).
- Information om farlig last.
- Bestemmelsessted og estimeret tid for ankomst.
- Skibets navigationsstatus.

Statiske informationer:
- Skibets MMSI nummer.
- Skibets kaldetegn- og nummer.
- Skibets type.
- Position af GNSS-antenne på skibet, herunder skibets længde og dæksbredde.
- Angivelse af positionens nøjagtighed.

Apparatet skal kunne modtage information iht. NMEA-standarden: NMEA 0183 [Cassidy, 1998]:
- UTC tid.
- Egen position.
- Egen fart. (SOG).
- Egen kurs (COG).
- Dato.



Kollisionsdetektion
Apparatet skal, ud fra de modtagne data, kunne detektere, om der er risiko for kollision.
- Der skal være realtime detektion, således at ikke er mærkbar forsinkelse i systemet.
- Der skal være både visuel og auditiv indikation, ved risiko for kollision.
- Der skal være mulighed for frakobling af auditiv indikation.
- Detektionen skal kun foregå indenfor en radius af 3 sømil.

Begrundelsen for ovenstående valg af rækkevidde er vurderet ud fra følgende:
Et skib sejler max 30 knob, og hvis to skibe sejler direkte mod hinanden, vil afstanden mellem dem mindskes med 60 sømil pr. time. Det vurderes, at en reaktionstid på 3 min vil være passende, og ud fra ovenstående findes en max radius på:

$$Hastighed \cdot Reaktionstid = \frac{60\ \text{knob} \cdot 3\ \text{min}}{60\ \text{min / time}} = 3\ \text{sømil} \tag{2.1}$$

Indtastning af skibsinformation
Da der skal udsendes en række informationer, som ikke bliver leveret af GPS'en, skal der være mulighed for at indtaste disse i brugerfladen. Det gælder de statiske og rejserelaterede informationer.

2.4 Eksterne grænsefladekrav

De eksterne grænseflader opdeles i brugergrænseflade, hardwaregrænseflade, kommunikationsgrænseflade og softwaregrænseflade:

2.4.1 Brugergrænseflade
Brugergrænsefladen, som skal være grafisk, præsenteres i et program, der installeres på en tilsluttet PC. Brugergrænsefladen skal opfylde nedenstående krav:
- Alle funktioner skal kunne tilgås med mus.
- Sproget skal være engelsk.
- Nærliggene skibe, samt eget skib skal vises grafisk i et vindue.
- Hvis der er risiko for kollision, skal dette markeres i det grafiske vindue.
- Det skal være muligt at få vist et "Popup-vindue" med detaljeret beskrivelse af det valgte skib.

2.4.2 Hardwaregrænseflade
- I/O port til kommunikation med VHF-radio.
- NMEA 0183 interface til GPS-modtager.
- Forsyningsstik til alle enheder.

2.4.3 Kommunikationsgrænseflader
- Kommunikation til VHF-radio vha. ITU-R M.1371 [AIS, 1998].
- Kommunikation til GPS-modtageren vha. NMEA 0183 [Cassidy, 1998].

2.4.4 Softwaregrænseflader
På PC'en er der en softwaregrænseflade mellem applikationen og operativsystemet. Programmet skal udvikles til MS Windows-platformen, men skal kunne omsættes til afvikling på andre platforme.

2.5 Krav til systemets ydelse

Systemet skal være i stand til at behandle de signaler, det får fra VHF-radioen og GPS'en, samt beregne, om der er skibe på kollisionskurs med eget skib. Endvidere skal systemet være i stand til at afsende de dynamiske informationer realtime. Det vil sige, at systemet skal kunne følge med til den hastighed, der kommunikeres med ifølge AIS-standarden.

2.6 Kvalitetsfaktorer

Pålidelighed er vigtigt, og derfor må systemet ikke præsentere data mere præcist, end der er belæg for. Det er endvidere vigtigt, at algoritmen til beregning af kollisionfare udføres på en sådan måde, at der altid gives alarm, hvis der er belæg derfor. Brugervenlighed er også vigtigt, idet systemet skal kunne anvendes af navigatører uden større EDB erfaring.

2.7 Afgrænsning

I kravsspecifikationen beskrives kravene til et mere eller mindre komplet system. Da der er begrænsede resurser til rådighed, er det ikke muligt at implementere et system, som kan varetage alle de, i kravsspecifikationen, stillede krav. Derfor har projektgruppen måtte prioritere visse dele af systemets funktioner og dermed udelade andre. Dette har resulteret i følgende afgrænsning, som er et kompromis mellem det komplette system, og hvad der har været tid til at implementere:

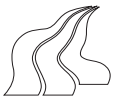
2.7.1 Ekstern kommunikation

Ifølge kravsspecifikationen, skal systemet sende og modtage informationer iht. AIS-standarden. Dette ændres hermed til, at systemet kun skal kunne modtage de informationer, som andre skibe udsender, og foretage den, i kravsspecifikationen, definerede behandling af disse. Dette gøres for at simplificere systemet således, at det er muligt at konstruere det på den givne tid. Desuden vil systemet stadig være brugbart for mindre lystsejlere, som med systemet vil kunne modtage informationer fra større skibe, og dermed blive advaret. Dette er muligt, da det fra 2003 bliver lovpligtigt for større skibe at udsende de informationer, der er defineret i standarden.

2.7.2 Frames til behandling

Systemet skal kun kunne behandle de frames, som er nødvendige for kollisionsdetektionen. Det værende: Dynamiske oplysninger, statiske oplysninger, samt korte tekstbeskeder fra andre skibe. Disse ændrede krav resulterer i, at systemet skal kunne behandle følgende frames fra AIS-standarden [AIS, 1998]:

- *1, Position:*
  Denne frame indeholder informationer om skibets placering, kurs og hastighed samt andre parametre, som er interessante i forbindelse med kollisionsdetektion. De to følgende frames indeholder de samme informationer, og derfor skal systemet også kunne behandle dem.
- *2, Assigned Position:*
  Se frame nr. 1
- *3, Position:*
  Se frame nr. 1
- *5, Static and voyage related data:*
  Denne frame benyttes til at overføre statiske informationer om skibet som fx: Navn, MMSI nummer, Last og Destination. Det skal i brugerfladen være muligt at få vist disse informationer.



- *8, Binary broadcast message:*
  Når man, fra et skib, ønsker at sende en tekstbesked med "valgfrit" indhold, benyttes denne frame. Da systemet skal kunne vise disse beskeder på skærmen, skal det være i stand til at behandle denne frame.

2.7.3 Radio

Der opbygges ikke nogen egentlig sende- / modtagedel (VHF-radio) til systemet, men i stedet benyttes en teststub, som skal generere de signaler, der tænkes leveret af VHF-radioen. Denne teststub implementeres ved hjælp af en PC, som i rapporten benævnes "testPC". Da testPC'en tænkes at kommunikere med systemet via dens parallelport, skal interfacet tilpasses denne teststub. Denne tilpasning beskrives nærmere under afsnittet "systemdesign".

2.7.4 Brugerflade

Brugerfladen på PC'en udvikles således, at den er forberedt for, at resten af systemet senere kan udbygges til også at sende. Det vil sige, at der skal være mulighed for indtastning af statiske og rejserelaterede informationer.

Det vurderes, at denne begrænsede udgave af systemet vil have en praktisk anvendelighed for lystbåde. Da man, ved at kende data fra de større skibe i nærheden, vil kunne begrænse faren for alvorlige kollisioner betragteligt.

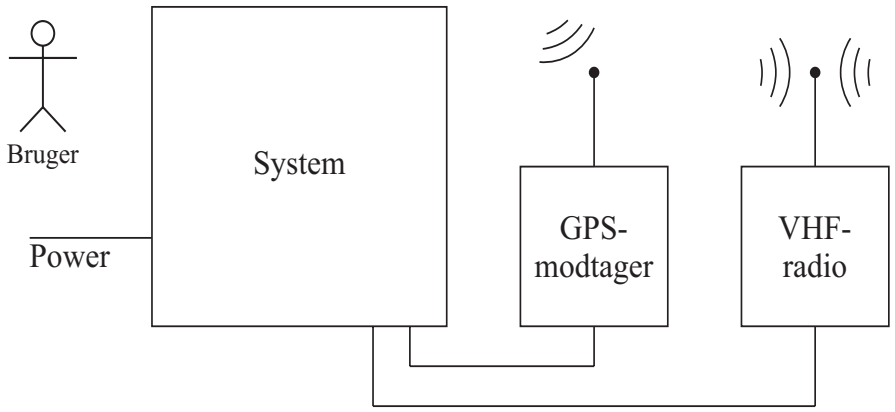
Efter de reviderede krav er opstillet, designes det pågældende system. Designet indledes med afsnittet "Systemdesign".

3 Systemdesign

I dette afsnit beskrives det samlede system, der skal varetage de funktioner, som blev beskrevet i kravsspecifikationen. Systemet vil blive opdelt i forskellige processer for at lette design, implementation og test.
Opdelingen i disse processer vil blive foretaget efter retningslinierne beskrevet i "Håndbog i Struktureret ProgramUdvikling" [SPU, 1998].

3.1 Systemets eksterne grænseflader

Som beskrevet i kravsspecifikationen, har systemet fire eksterne grænseflader. Disse grænseflader kan ses på figur 3.1. Det ses, at grænsefladerne udgøres af en brugergrænseflade, to systemgrænseflader til hhv. GPS-modtageren og VHF-radioen, samt en grænseflade til strømforsyningen. Det efterfølgende vil være en specifikation af disse eksterne grænseflader.



Figur 3.1 Grænseflader til systemet.

3.1.1 Brugergrænseflade

Brugergrænsefladen præsenterer systemet for brugeren, og det er derfor væsentligt, at den formår at levere de nødvendige oplysninger på en nem og overskuelig måde. Da brugergrænsefladen implementeres på en PC, vil det være naturligt at benytte et grafisk udseende, som designes iht. normen for MS Windows baserede programmer.
Der er endvidere følgende krav til brugergrænsefladen, som skal opfyldes:

- Feltet med vektorer skal være nordvendt, hvilket vil sige, at skærmens top altid ligger i nord.
- Eget skib skal visualiseres med en vektor, som er placeret i centrum af skærmen.
- Nærtliggende skibe skal visualiseres med vektorer, således at det er muligt, at vurdere deres position, fart og retning i forhold til eget skib.
- Det skal være muligt at se de skibsdata, som er specificeret i kravsspecifikationen, for alle skibe indenfor det kritiske område, forudsat at det pågældene skib har udsendt de relevante data over VHF-radioen.

3.1.2 Grænseflade til GPS-modtager

Denne grænseflade er bestemt af GPS-modtager kommunikationsgrænseflade. Da en GPS kommunikerer med eksterne enheder vha. en NMEA 0183 standard, kræves det naturligvis, at systemet overholder denne standard [Cassidy, 1998]. Standarden fortæller imidlertid intet om hardware-interfacet, og defor må "stiktypen" vælges på baggrund af den benyttede GPS-modtager. NMEA-standarden giver mulighed for modtagelse af en række forskellige informationer, som ikke har relevans i dette system. Det er valgt at bruge den besked der hedder RMC, da den indeholder informationer om: position, fart, retning, tid og dato.

Input fra GPS-modtager:
Protokollen, der definerer inputtet fra GPS'en, ser ud som følger:

$GPRMC,UTC,status,latitude,longitude,SOG,COG,date,mag. var.,mode*CRC<CR><LF>

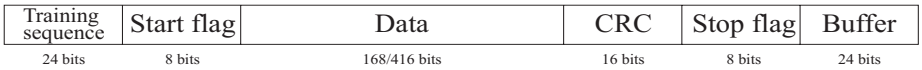
Data mellem "$" og "CRC" optager 63 Byte.

3.1.3 Grænseflade til VHF-radio

VHF-radioen er erstattet af en teststub. Denne teststub udgøres af en parallelport på en testPC samt et tilhørende program, der styrer porten, og derfor skal grænsefladen tilpasses denne port. Grænsefladen skal således være TTL kompatibel, hvilket betyder, at der kan forventes spændinger på 0 V eller 5 V. Endvidere er data aktivt lave (0 V svarer til logisk "høj" og 5 V svarer til logisk "lav"). Dataudvekslingen sker ved en bitstrøm med en baudrate på 9600, som foreskrevet i AIS-standarden. Grænsefladen overholder i øvrigt AIS-protokollen, dog med de forbehold og begrænsninger, der er beskrevet i kravsspecifikationen og afgrænsningen.

I afgrænsningen blev det vedtaget at systemet udelukkende skulle *modtage* data fra andre skibe og havnemyndigheder. Dette medfører, at kommunikationen med VHF-radioen er unidirektionel. Radioen modtager således signalerne fra andre skibe og havnemyndigheder, konverterer dem til en kontinuert bitstrøm, som defineret i [AIS, 1998], og sender dem til microcontrolleren.

Input fra VHF:
Bitstrømmen er opdelt i sekvenser, som vist på figur 3.2.



Figur 3.2 Bitstrømmen fra VHF-radioen opdelt i sekvenser.

Den første sekvens er en training sequence, hvor der skiftevis udsendes "0" og "1". Denne sekvens bruges til at synkronisere modtageren med senderen, for at sikre at data aflæses på det rigtige tidspunkt. Herefter sendes startflag, som bruges til at identificere starten af en datapakke. Efter startflaget følger selve de data, der skal sendes, efterfulgt af CRC (cyclic redundancy code) og stopflag. Både start- og stopflag består af en sekvens, der ser således ud: "01111110". CRC sekvensen består af en checksum, der er udregnet på baggrund af de sendte data, hvormed man kan kontrollere, om der er sket en fejl under transmissionen (se endvidere appendix B). Disse start- og stopflag er defineret i HDLC. For at undgå misfortolkning af start- og stopflag, er der foretaget bitstuffing på data og CRC [Tanenbaum, 1996, 225]. I denne protokol er der foretaget bitstuffing når der har været fem ettaller i træk: Der indsættes et nul efter det femte ettal.
Bitstrømmen afsluttes med en buffer, som bruges til de ekstra bit, der stammer fra bitstuffing, samt til at sikre, at der ikke sker overlap mellem to frames.

Den første del af datasekvensen indeholder message ID, som fortæller hvilken type besked, der er tale om. I denne begrænsede udgave af systemet kan der kun blive tale om beskederne,1, 2, 3, 5 og 8, som kan opdeles i følgende kategorier:

Positionsrapport:

Hvis message ID er enten 1, 2 eller 3, kan man fra de følgende 162 bit aflæse følgende information:
MMSI nummer, skibets status, kursændring i grader pr. minut, skibets hastighed (SOG - speed over ground), positionens nøjagtighed, længde- og breddegrad, beholdende kurs (COG - course over ground), styrende kurs, samt et tidsmærke.

Statiske og rejserelaterede informationer:

Hvis message ID er 5, kan man, fra de følgende 410 bit, aflæse følgende informationer: MMSI nummer, IMO nummer, kaldenavn, skibets navn, skibets type og last, GNSS antennes position, forventet ankomst, dybdegang samt destination.

Tekstbesked:

Hvis message ID er 8, kan man fra de følgende 1186 bit aflæse følgende informationer: MMSI nummer, samt en tekstbesked på maksimalt 121 tegn.

3.1.4 Grænseflade til forsyning

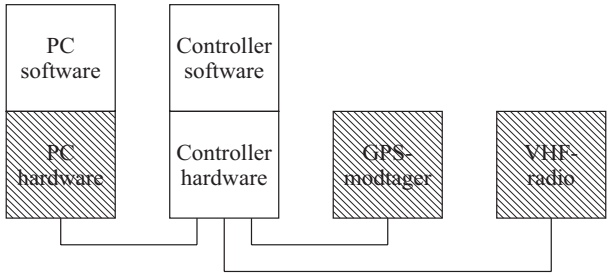
Strømforsyningen udgør ligeledes en grænseflade til systemet. Da der på mange skibe kun er et batteri til rådighed, vælges grænsefladen til 12V DC. Batteriet vurderes at skulle kunne levere en strøm på 300 mA.

3.2 Opdeling i processer

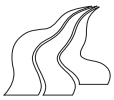
På baggrund af figur 3.1, opdeles systemet i en række delsystemer; processer. Processerne adskilles der, hvor de har den mindste grænseflade, for at gøre dem så uafhængige af hinanden som muligt.
Systemet deles derfor op i følgende processer:

- Controller hardware.
- Controller software.
- PC program.

Systemet er inddelt i disse processer, for at gøre det mere overskueligt, samt for at lette implementationen af systemet. Endvidere vil det være muligt, at designe og implementere de enkelte processer sideløbende, under forudsætning af, at der eksisterer veldefinerede grænseflader mellem dem. I rapporten refereres der til controlleren, som omfatter både controller hardware og controller software. Det strukturerede system ser nu ud, som vist på figur 3.3.



Figur 3.3 Opdelingen i processer. De skraverede dele skal ikke konstrueres og betragtes dermed ikke som en del af det udviklede system.



3.2.1 Controller hardware

Denne proces består af den hardware, som er nødvendig, for at controlleren kan varetage den påkrævede funktion. Controlleren skal kommunikere med en GPS-modtager, en VHF-radio og en PC. Dette betyder, at kernen i controlleren er et microprocessorsystem, som varetager kommunikationen mellem de tre enheder. Endvidere skal der laves et interface til hver af de tre enheder, som opfylder de tidligere beskrevne grænsefladekrav. I controller hardware indgår desuden en strømforsyning, der skal levere de nødvendige strømme og spændinger, som hardwaren har brug for.

3.2.2 Controller software

Denne proces skal styre hardwaren og behandle de indkomne data. Processen styrer de nederste lag i AIS protokollen, dvs. fra "Physical layer" til og med "Transport layer" i OSI reference modellen [Tanenbaum,1996]. Endvidere skal denne proces kommunikere med GPS-modtageren og videresende relevante data til PC'en.

3.2.3 Applikation på PC

Denne proces skal behandle de indkommende data og varetage de øverste lag i AIS-protokollen. Der skal endvidere foretages en kollisionsberegning på baggrund af de data, som modtages fra controlleren. De relevante data skal visualiseres vha. en grafisk brugerflade. Den skal sørge for, at afgive både visuel og auditiv alarm, hvis der er kollisionsfare. Brugerfladen skal baseres på MS Windows platformen og være så simpel som muligt.

3.3 Grænseflader mellem processer

Kommunikationsgrænsefladerne mellem de forskellige processer bestemmes, således at processerne kan implementeres parallelt og uafhængigt.

3.3.1 Controller Hardware / Software

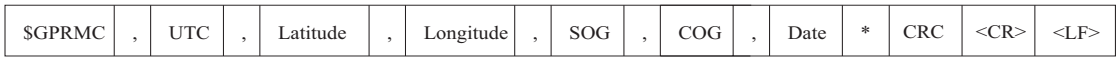
Da denne grænseflader er afhængig af hvilket microcontroller der benyttes, er det ikke muligt at definere den mere præcist på dette sted. Dette betyder at de hardware-nære rutiner skal koordineres med controller hardware.

3.3.2 Controller / PC program

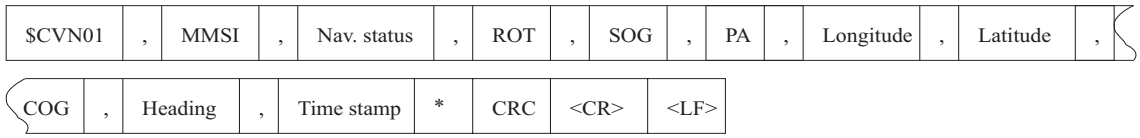
Grænsefladen mellem controller og PC udgøres af en RS232C seriel asynkron forbindelse. Data sendes fra controller til PC i 8 bit ad gangen med en start- og stopbit og ingen paritetsbit. Der sendes med en hastighed på 38.400 baud. Der benyttes ingen form for handshake. Data fejlkontrolleres vha. standarden CRC-ITU-T, som overholdes af programmet på controlleren. De data der sendes til PC'en benytter den protokol, der er defineret i NMEA-standarden. Denne protokol angiver, at der startes med en "$", som efterfølges af en afsender ID, samt at der afsluttes med "*", 16 bit CRC, <CR> og <LF>. Afsender ID er opdelt, således at de 2 første tegn angiver, hvem afsenderen er, og de næste 3 hvilken besked, der er tale om. I NMEA-standarden foreskrives det, at der bruges "GP" som afsender ID for GPS beskeder, og "CV" som afsender ID for VHF beskeder.
Beskedtypen er for GPS'ens vedkommende, den der hedder "RMC" (Recommended Minimum Specifik GNSS Data), og for VHF'ens vedkommende besluttes det at bruge "N0x", hvor x angiver beskednummeret.

Beskederne, der sendes til PC'en, findes således i dataframes af følgende format:

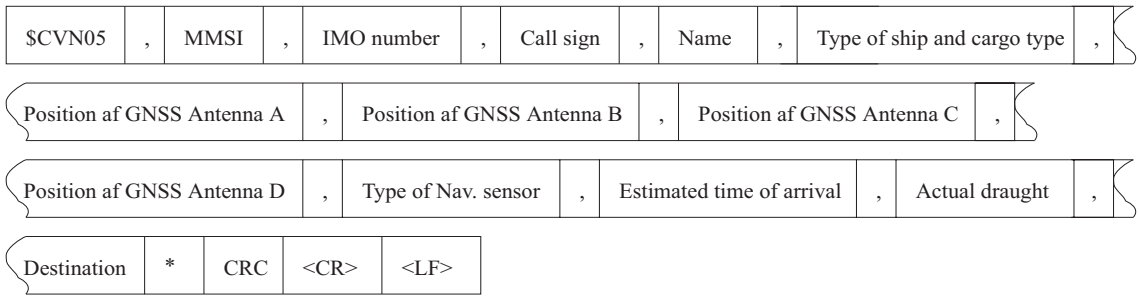
GPS besked



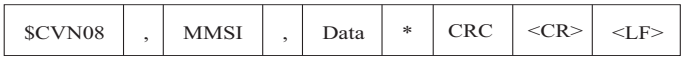
VHF besked 1, 2 og 3



VHF besked 5



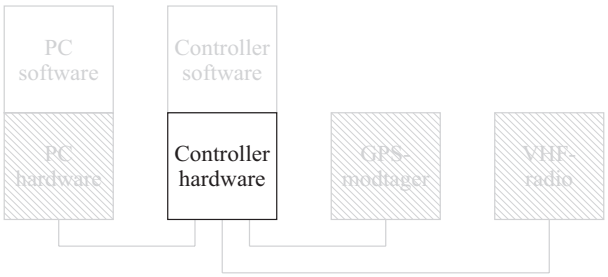
VHF besked 8



Hermed er de overordnede rammer for designet fastlagt, således at det er muligt at designe de enkelte processer.

4 Controller Hardware

Denne del af systemet består af de fysiske kredsløb, såsom digitale kredse, kondensatorer, modstande osv. I dette kapitel foretages en opdeling af processen i moduler, og disse modulers grænseflader udgør processens interne grænseflader. Modulerne designes, implementeres og testes. Desuden defineres de eksterne grænseflader til de andre processer.

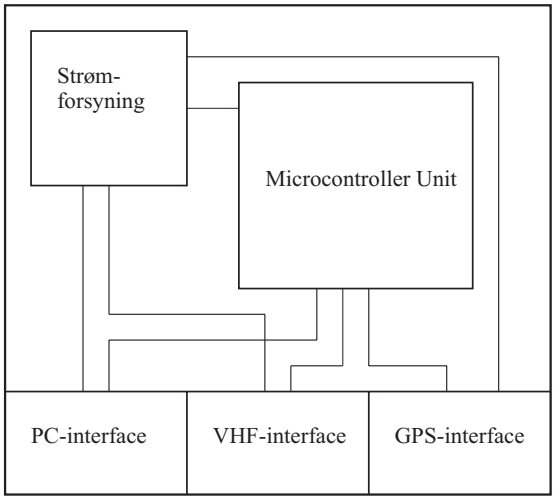


4.1 Procesdesign

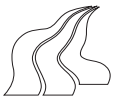
Denne proces udgør den, af projektgruppen, opbyggede hardware. Den skal sikre kommunikationen mellem de andre dele (processer) af systemet. Dette gøres ved at opbygge interfaces til de enheder, der skal kommunikeres med, samt ved at konstruere et microcontroller system, der kontrollerer signalerne og distribuerer dem til de forskellige interfaces.

4.1.1 Opdeling i moduler

Ud fra en betragtning af processens opgaver, opdeles den i følgende moduler, som skal varetage dem: GPS-interface, VHF-interface, PC-interface, Microcontroller Unit og Spændingsforsyning.



Figur 4.1 Modulernes placering i processen.



Disse modulers funktioner og eksterne grænseflader specificeres dermed som følger. Der defineres en forkortelse af navnet på hvert modul, som benyttes ved navngivning af signalerne. Signalets type skrives med stort, og signalets afsender og modtager med lille. Afsender står først og modtager sidst.

4.1.2 GPS-interface (IGPS)

Dette modul består af interfacet mellem GPS-modtageren og selve microcontroller systemet. Modulets opgave er, at omforme signalerne fra GPS-modtageren fra NMEA-0183 standard til TTL-kompatible signaler og omvendt.

4.1.3 VHF-interface (IVHF)

Dette modul udgør interfacet til VHF-radioen. Da systemet kun skal kunne *modtage* fra VHF-radioen, er der tale om envejskommunikation. Grænsefladen til radioen svarer til systemets eksterne grænseflade til denne. Denne grænseflade er defineret under systemdesign afs. 3.1.3.

4.1.4 PC-interface (IPC)

Dette modul skal gøre det muligt for microcontrolleren at kommunikere med PC'en via en RS232C-forbindelse.

4.1.5 Microcontroller Unit (MCU)

Dette modul er det centrale i processen; Det skal kunne kommunikere med samtlige interfaces, og dermed sikre kommunikationen mellem de øvrige processer i systemet. Herunder skal det varetage de nederste lag i AIS-protokollen, dvs. fra og med "link layer" op til og med "transport layer". Desuden skal det foretage CRC-kontrol af signalerne fra VHF-radioen og GPS-modtageren.

4.1.6 Spændingsforsyning (PSU)

Dette modul skal kunne levere de nødvendige strømme og spændinger til de øvrige moduler i processen. Modulet bliver forsynet af en 12 V DC forsyning eller alternativt af lysnettet (hvis skibet har en 230 V AC forsyning).

I det følgende designes, implementeres og testes modulerne.

4.2 GPS-Interface

Til systemet anvendes et Motorola M12 Oncore GPS-modul, som er en 12 kanals GPS modtager [Oncore, 2000]. GPS-modtageren kræver en forsyningsspænding på mellem 2,75 V DC og 3,2 V DC, med en maksimal ripple på 50 mV fra peak til peak. GPS-modtageren har et interface af typen NMEA-0183, hvorfra outputtet er 0 V og 3 V, for henholdsvis logisk "1" og logisk "0". Modulet er endvidere forsynet med et 10 polet stik af typen "Double row vertical header", hvortil interfacet mellem GPS-modtageren og microcontrolleren tilkobles. For at kunne kommunikere med GPS-modulet, skal forbindelsen mellem GPS-modtageren og microcontrolleren være en duplex forbindelse. Der kommunikeres med en hastighed på 4800 baud.
Signalerne fra GPS-modtageren skal tilpasses til TTL-niveau på microcontrollerens side. Logisk "1" skal svare til 5 V og logisk "0" til 0 V.
NMEA-0183 standarden kræver endvidere, at der er en galvanisk adskillelse mellem de to kommunikerende enheder.

Input:

VCC_3V:	3 V DC forsyning
VCC_5V:	5 V DC forsyning
GND:	Stel (Signal Ground)
$GPSREQ_{MCU-IGPS}$:	Kontrolsignaler til GPS-modtageren. TTL-kompatible.
$GPSDATA_{GPS-IGPS}$:	Data fra GPS-modtageren. Overholder NMEA-0183 standarden.

Output:

$GPSDATA_{IGPS-MCU}$:	Data fra GPS-modtageren. Skal være TTL-kompatible.
$GPSREQ_{IGPS-GPS}$:	Forespørgsler til GPS-modtageren. Skal overholde NMEA-0183 standarden.

4.2.1 Design

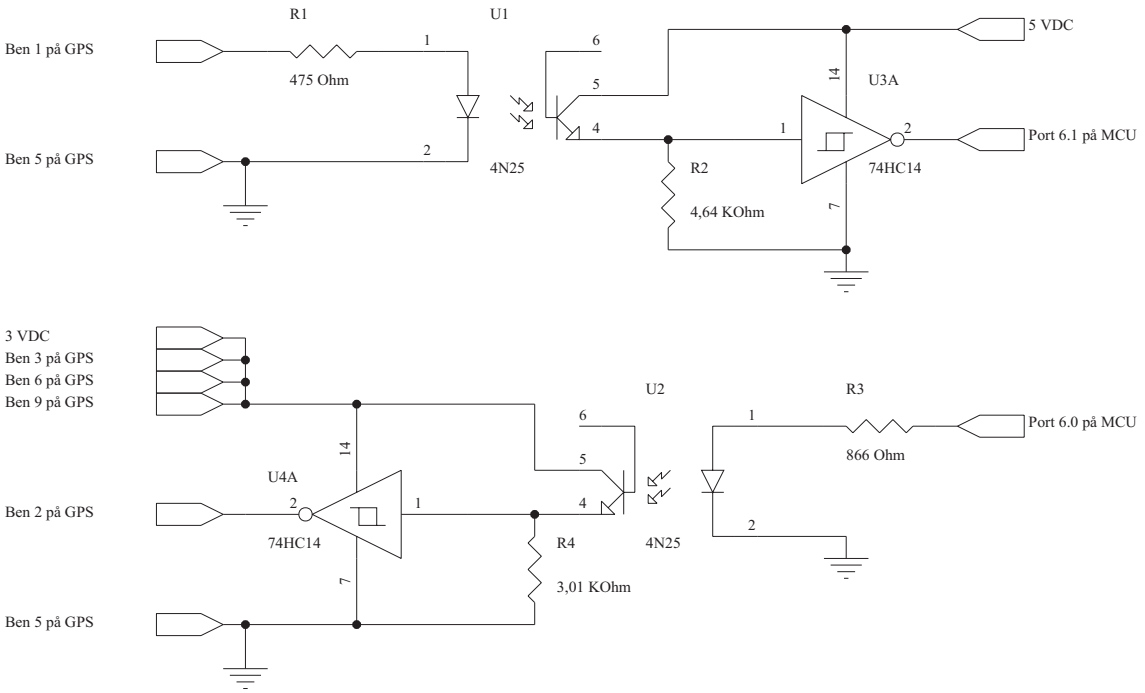
Interfacet skal designes således, at det omsætter spændingsniveauet fra GPS-modtageren til TTL-niveau, samt omsætter spændingsniveauet fra microcontrollerens TTL-niveau til GPS-modtagerens spændingsniveau (0 V og 3 V). Denne omformning af spændinger omfatter således også en invertering af signalet. Interfacet skal endvidere designes således, at der ikke er signalmæssig galvanisk forbindelse mellem GPS-modtageren og microcontrolleren.

4.2.2 Implementering

Implementeringen af interfacet er opdelt i henholdsvis et transmit- og et receivekredsløb.
På databladet for GPS-modtageren ses det, at pin 1 er transmit, pin 2 er receive, pin 3 er +3 V DC og pin 5 er ground.
Transmitkredsløbet opbygges som vist i øverste del af figur 4.2, og virker således, at når GPS-modtageren udsender "0", hvilket svarer til 3 V, vil der løbe en strøm gennem lysdioden i optokobleren. Ifølge databladet for optokobleren (4N25) kan modstanden R1 findes:

$$R1 = \frac{V_{GPS} - V_{Diode}}{I_{Diode}} = \frac{3\ V\ -\ 0.65\ V}{5\ mA} = 470\ \Omega \qquad (4.1)$$

Der vælges således en modstand på 475 Ω. Lyset fra dioden åbner fototransistoren, således at der løber en strøm gennem denne. Ifølge databladet for optokobleren vil der løbe en strøm på 0,6 mA fra collector til emitter, når der løber en strøm på 5 mA gennem dioden. For at undgå, at der sendes udefinerede spændingsniveauer til microcontrolleren, og for at invertere signalet, kobles transistorens emitter til en schmitttrigger.



Figur 4.2 GPS-interface mellem MCU og GPS-modtager.

Ifølge databladet for schmitttriggeren, aktiveres denne ved en spænding på 2,8 V, hvorved modstanden R2 findes ved:

$$\frac{V_{t+}}{I_c} = \frac{2{,}8\ \text{V}}{0{,}6\ \text{mA}} = 4{,}67\ \text{k}\Omega \tag{4.2}$$

Receivekredsløbet opbygges efter samme princip som transmitkredsløbet, blot med den forskel at signalerne løber den modsatte vej. Dette ses på den nederste halvdel af figur 4.2.

4.2.3 Test

Interfacet er testet ved at tilslutte strømforsyninger på henholdsvis 3 V og 5 V, samt en variabel strømforsyning til signalgivning. Herved er det målt, at transmitkredsløbet skifter fra lav til høj ved en spænding på 1,5 V, og skifter tilbage ved en spænding på 1,3 V.
Ligeledes skifter receivekredsløbet, fra lav til høj ved en spænding på 2,1 V, og tilbage ved en spænding på 1,8 V. Det må altså konkluderes, at interfacet virker efter hensigten.
Systemet hastighed er også tilfredsstillende, da Schmidttriggeren og optokobleren opfylder kravet om at kunne skifte med en hastighed på 2400 Hz.

4.3 VHF-interface

VHF-interfacet varetager kommunikationen mellem microcontrolleren og VHF-radioen. Da VHF-radioen erstattes af en teststub, som implementeres vha. parallelporten på en test-PC, er de signaler, der kommer ind til processen TTL-kompatible. De signaler, der skal videresendes til microcontrolleren, skal også være TTL-kompatible, og derfor er interfacets eneste opgave at virke som buffer, for at sikre at der ikke trækkes for meget strøm fra parallelporten på test-PC'en. Der kommunikeres med en hastighed på 9600 baud.

Input:

VCC_5V:	5 V DC forsyning
GND:	Stel (Signal Ground)
$VHFDATA_{VHF-IVHF}$:	Data fra VHF-radioen. TTL-kompatibel.

Output:

$VHFDATA_{IVHF-MCU}$:	Data fra VHF-radioen.

4.3.1 Design

Interfacet består af en buffer, som sender signalerne direkte videre.

4.3.2 Implementering

Der indsættes en buffer af typen 74LS241. Denne buffer er TTL-kompatibel, og den har otte indgange med 8 tilhørende udgange, men det er kun nødvendigt at benytte een indgang, og dennes udgang. “Output enable* ”, samt de indgange, der ikke benyttes, holdes permanent lave, således at udgangen altid følger indgangen. Kredsen kan forsynes med 4,5 V - 5,5 V, så der vælges en forsyning på 5 V.

4.3.3 Test

Interfacet testes ved at tilslutte en tonegenerator, med et 4800 Hz firkantsignal og en spænding på 5 Vpp, på bufferens indgang. Derefter måles med en “digital analyzer” på ind- og udgang for at konstatere, at de svarer til hinanden. Testen er udført, og det viste sig, at outputtet svarede til inputtet.

4.4 PC-Interface

For at kunne kommunikere mellem PC og microcontroller, er det nødvendigt med et interface mellem disse processer. Kommunikationen skal foregå via en seriel RS232C forbindelse.

Input:

VCC_5V:	5 V DC forsyning
GND:	Stel (Signal Ground)
$DATA_{MCU-IPC}$:	Data til PC. TTL-kompatibel.
$DATA_{PC-IPC}$:	Data til microcontroller. RS232C-kompatibel.

Output:

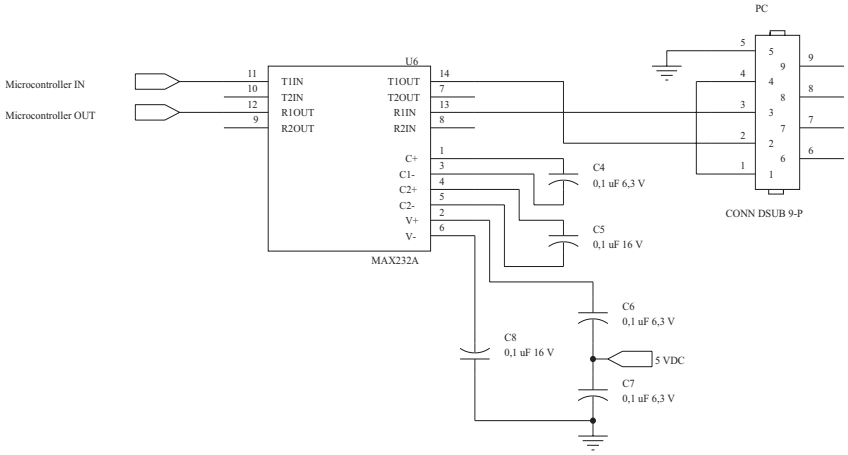
$DATA_{IPC-PC}$:	Data til PC. RS232C-kompatibel.
$DATA_{IPC-MCU}$:	Data til microcontroller. TTL-kompatibel.

4.4.1 Design

Interfacet designes således, at det omsætter spændingsniveauet fra PC’ens serielport til TTL-niveau, samt omsætter spændingsniveauet fra microcontrollerens TTL-niveau til RS232C niveau.

4.4.2 Implementering

Implementeringen foretages med en kreds, som ganske enkelt omsætter spændingsniveauerne til henholdsvis TTL-niveau og RS232C-niveau. Der bruges en MAX232A, som er en RS232C line driver / receiver. Kredsen opkobles som vist på figur 4.3, hvor kondensatorværdierne er valgt i henhold til databladet.



Figur 4.3 PC-interface mellem MCU og PC.

Test
Interfacet er testet ved at sætte en tonegenerator med et firkantsignal på 19.2 kHz og en spænding på 0 - 5 V på ben 11 . Outputtet kontrolleres ved hjælp af en "digital analyzer" på ben 14, hvor der skal være et firkantsignal med en frekvens på 19.2 kHz og en spænding på ±7 V. Testen er gennemført, og outputtet var som forventet.

4.5 Microcontroller Unit (MCU)

Dette modul udgør kernen af hardwaren. Modulet skal koordinere al kommunikation mellem modulerne i processen. Dvs. at det skal kommunikere med GPS-modtageren, VHF- radioen og PC'en gennem de interfaces, som også indgår i hardwaren. Dette modul består hovedsageligt af en microcontroller og noget hukommelse.

Input:

VCC_5V:	5 V DC forsyning
GND:	Stel (Signal Ground)
$GPSDATA_{IGPS-MCU}$:	Data fra GPS-modtageren. TTL-kompatibel.
$VHFDATA_{IVHF-MCU}$:	Data fra VHF-radioen. TTL-kompatibel.

Output:

$DATA_{MCU-IPC}$:	Data til PC. Skal være TTL-kompatible.
$GPSREQ_{MCU-IGPS}$:	Forespørgsler til GPS-modtageren. Skal være TTL-kompatible.

4.5.1 Design

Da modulet består af to hoveddele; en microcontroller og noget hukommelse (til program og data), skal der opstilles nogle krav til disse enheder. Disse krav kan opstilles, dels på baggrund af modulets grænseflader og dels på baggrund af en æstimering. Det efterfølgende er opdelt i to dele; krav til microcontroller og krav til hukommelse.

Krav til microcontroller:

- 2 serielle porte: Der skal være minimum 2 serielle porte på microcontrolleren, som kan køre uafhængigt af hinanden (med forskellige hastigheder). Den ene serielle port skal benyttes til kommunikation med PC'en og skal køre med en hastighed på 38.400 baud. Den anden serielle port benyttes til kommunikation med GPS-modtageren gennem et NMEA 0183 interface. Denne port skal pga. denne standard køre med en hastighed på 4800 baud.

- 1 input port: Der skal være minimum een inputport, som skal benyttes til at modtage en bitstrøm på 9600 baud fra en VHF-radio.

- 1 timer: Der skal være minimum een intern timer på microcontrolleren, som kan benyttes til synkronisering med VHF-radioen.

- Eksternt adresserum: Der skal være mulighed for adressering af ekstern hukommelse. Det vurderes, at der som minimum, skal kunne adresseres 32 kB ekstern hukommelse, hvilket betyder, at microcontrolleren skal have en ekstern adressebus på minimum 15 bit.

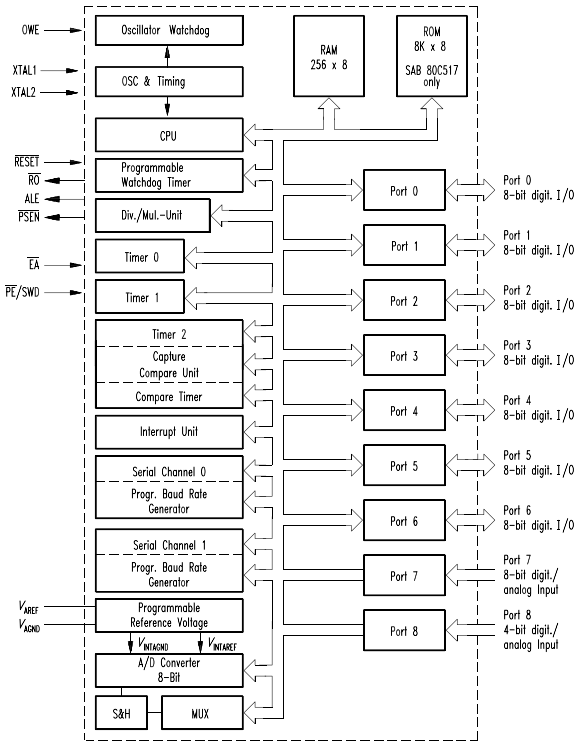
Krav til hukommelse:

- 32 kB RAM: Det vurderes, at der skal være 32 kB hukommelse til rådighed. I praksis vil der være brug for langt mindre, men da der er tale om et udviklingsforløb, vil det være fordelagtigt, hvis der er rigelig plads til rådighed.

- 32 kB ROM: Det vurderes ligeledes, at der skal benyttes 32 kB program hukommelse til hovedprogram, testprogrammer og lignende.

4.5.2 Implementering

På baggrund af de tidligere nævnte krav, er det nu muligt at fastlægge, hvilken microcontroller der skal benyttes. Valget er foretaget dels på baggrund af de nævnte krav, og dels på baggrund af, hvad der var tilgængeligt hos komponentudleveringen. Valget faldt på en Siemens SAB80C537, da den lever op til de stillede krav. SAB80C537 bygger på en 80C51 struktur, og er fuldt ud kompatibel med denne. Der er dog lavet en række forbedringer på SAB80C537 i forhold til 80C51 strukturen. Hovedstrukturen i SAB80C537 ses på figur 4.4.

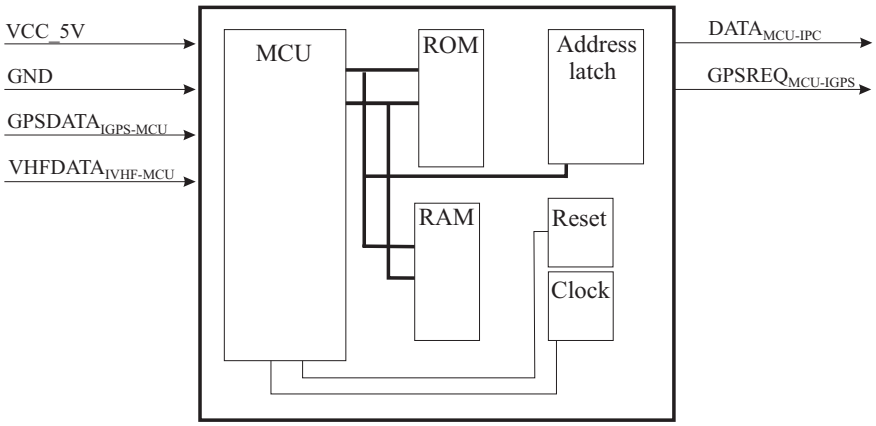


Figur 4.4 Hovedstruktur af SAB80C537.

Udvalgte specifikationer for SAB80C537:

- Op til 16 MHz clockfrekvens.
- 64 ud af 111 instruktioner bliver udført på 1 maskincyklus.
  (1 maskincyklus = 12 clock perioder).
- 256 Byte "on die" RAM.
- Adressering af op til 64 kB eksternt RAM og ROM.
- 8 bit A/D konverter.
- 2 16-bit timere.
- 1 Compare / Capture Unit (CCU).
- Seperat aritmetisk enhed til udførelse af multiplikation og division.
- 8 16 bit datapointere (Udvidelse af 80C51 strukturen).
- 2 full duplex serielle porte med egen baudrate generator.
- 7 bidirektionelle (I/O) porte.

Disse egenskaber for SAB80C537 stemmer godt overens med de stillede krav. SAB80C537 giver endda flere muligheder end krævet.
Microcontrolleren kræver nogle få eksterne kredsløb for at fungere, og opsætningen af disse samt microcontrolleren ses på figur 4.5.



Figur 4.5 Pseudodiagram over MCU modulet

Det ses på figur 4.5, at det kræver forholdsvis få eksterne kredsløb for at drive SAB80C537. Udover spændingsforsyningen er der kun opsat et simpelt kredsløb til at "resette" microcontrolleren, en ekstern oscillator og et RAM/ROM kredsløb.
Den eksterne krystaloscillator er valgt til en værdi på 14,7456 MHz. Valget af denne frekvens er baseret på den måde, hastigheden på de serielle porte bestemmes ud fra oscillator- frekvensen. Ved betragtning af formlen for bitrate kan det ses, at de mest gængse bitrate (4800, 9600, 19.200 og 38.400 baud) kan opnås med denne oscillatorfrekvens. Oscillatorkredsløbet kan ses på figur 4.6.
Den kapacitet, som krystallen ser ind i, skal ifølge databladet være på 18 pF. Kapaciteten kan beregnes ved følgende formel:

$$C_L = \frac{C_1 \cdot C_2}{C_1 + C_2} + C_S \tag{4.3}$$

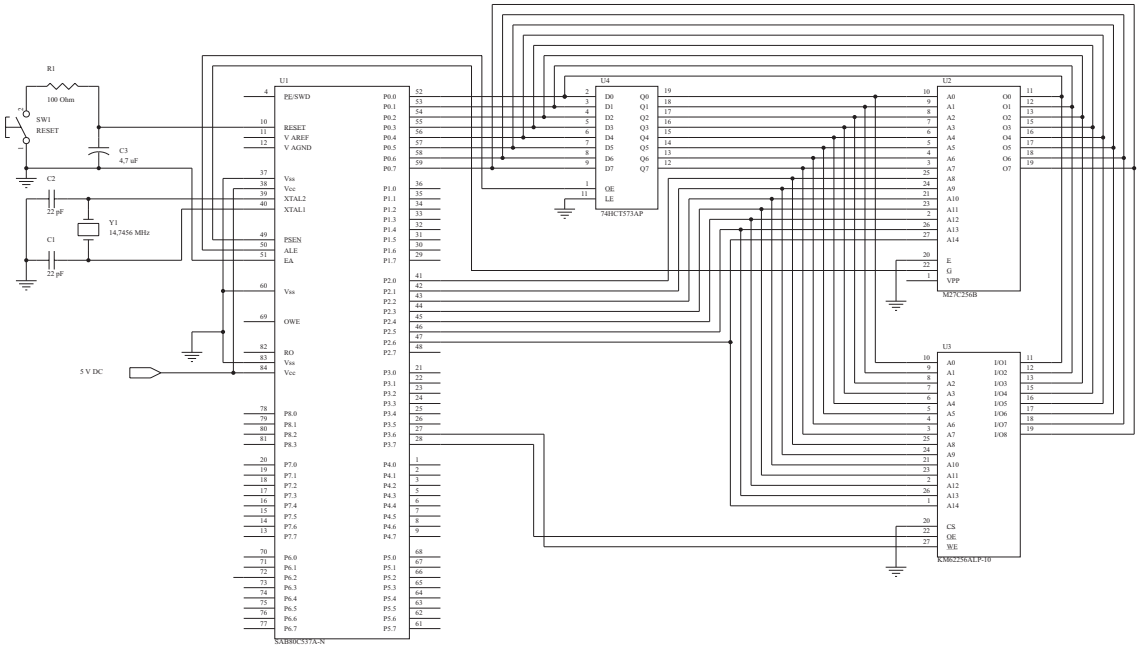
hvor, C_L er den samlede kapacitet.
C_1 og C_2 er de eksterne kondensatorer.
C_S er kapaciteten på microcontrollerens indgang (9 pF).

Hvis C_1 og C_2 vælges til at være lige store, kan de beregnes ved følgende formel:

$$C = 2(C_L - C_S) \tag{4.4}$$

Ved indsættelse af værdierne fra databladene fås C til 18 pF, hvilket medfører, at der vælges en standardværdi på 22 pF.
Microcontrolleren skal forbindes til en ekstern ROM- og RAM-kreds, hvor software kan lagres. I databladet for microcontrolleren oplyses det, at det under en read cycle, ved brug af 12 MHz krystal, maksimalt må tage 233 ns fra ALE, aktiveres til der er data på bussen. Desuden kommer adressevektoren ud på bussen ca. 74 ns efter ALE aktiveres, og holdes der i ca. 101 ns. Både RAM og ROM kredsen skal overholde disse tidskrav. Derfor vælges en RAM-kreds af typen 62256-10 og en EPROM af typen M27C256-10 til ROM-kreds.

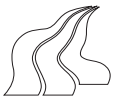
Microcontrolleren er opbygget således, at den bruger samme bus til både data og adresser, hvilket betyder, at der skal implementeres en adressedekoder, som kan holde adressen, mens data sendes og modtages (Harward struktur).
Der er ligesom for RAM og ROM krav til, hvor hurtigt den skal kunne latche data fra den enables. Ud fra disse krav vælges en adressedekoder af typen 74HCT573.
På nedenstående figur ses, hvorledes de forskellige komponenter er forbundet.



Figur 4.6 Det samlede MCU modul.

4.5.3 Test af MCU

For at teste om microcontrolleren lever op til de stillede krav, er der lavet testprogrammer ved hjælp af C-compileren µvision-51. Derefter er de testet i dScope, for at se om de virker efter hensigten. Testprogrammerne er herefter brændt ned i en EPROM og testet på det virkelige system.
For at teste MCU'ens sammenspil med RAM og ROM, er der lavet et testprogram. Programmet opretter en række globale variable, da disse allokerer plads i RAM'en. Variablene tillægges nogle værdier, adderes sammen og sendes ud på port 4. Da den korrekte værdi kan aflæses på port 4, konkluderes det, at RAM-ROM-MCU kredsløbet fungerer efter hensigten. For at kontrollere, om timingen er korrekt, anvendes en "digital analyzer" af typen HP 54620A, som sættes til at trigge



på RD* (Read) signalet. Det konkluderes, at timingen på RAM/ROM overholder de tidskrav, som microcontrolleren stiller.

4.6 Strømforsyning

Som strømforsyning til det samlede system, anvendes enten en 230 V AC til 12 V DC adapter eller en 12 V accumulator. I systemet er der behov for to spændingsniveauer, på hhv. 3 V og 5 V. Spændingen på 3 V skal anvendes til GPS-modtageren og spændingen på 5 V er til forsyning af de øvrige moduler.

Modulet har til formål at transformere spændingen fra 12 V til 3 V og 5 V. Endvidere skal modulet sørge for, at de stillede krav for ripple, afvigelse og strøm bliver opfyldt.

Input:
230 V AC / 12 V DC: Forsyning fra lysnet eller accumulator.

Output:

VCC_3V:	3 V DC forsyning.
VCC_5V:	5 V DC forsyning.
GND:	Stel (Signal Ground)

Design
Modulet deles op i to dele; een for 3 V og een for 5 V.

5 V forsyning:
For at designe 5 V spændingsforsyningen, er det nødvendigt at fastlægge, hvor meget strøm, der skal leveres til systemet, samt hvor meget spændingen må variere. Strømforbruget for hvert modul er vha. datablade beregnet til:

Strømforbrug	
Modul	Strøm(max)
MCU	110 mA
VHF interface	46 mA
GPS interface	15 mA
PC interface	15 mA
Total:	186 mA

Den maksimale afvigelse fra 5 V er fundet til ±0,25 V, da microcontroller modulet, omfatter en latch, som ifølge databladet skal have en spænding på mellem 4,75 og 5,25.

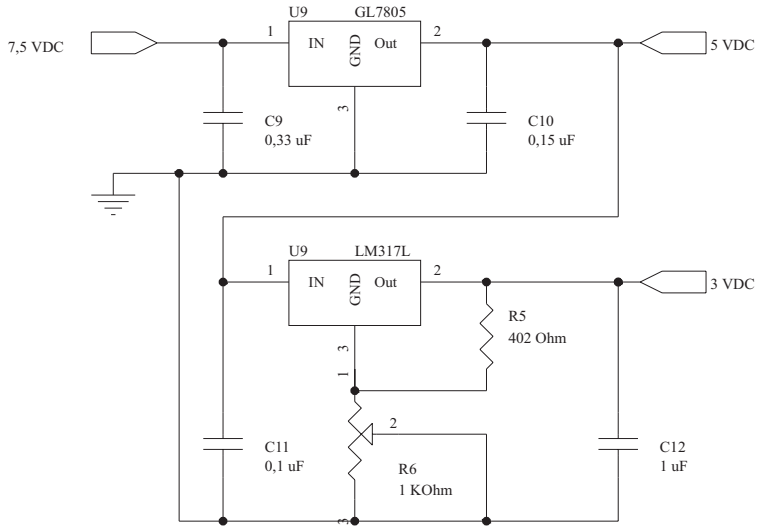
3 V forsyning:
GPS-modtageren skal forsynes med 3 V og har, med antenne, et strømforbrug på max 100 mA. Spændingen må maksimalt variere med ± 0,2 V og have en ripple på 50 mV. Da 3 V forsyningen skal levere strøm og spænding til GPS-modulet, skal den overholde disse krav.

Implementering
Til implementering af 5 V spændingsforsyning, anvendes en spændingsregulator af typen "GL8705BG". Spændingsregulatoren opkobles, som det er foreskrevet i databladet. Den afsatte effekt i spændingsregulatoren bliver omsat til varme, og for at beregne denne varme, opstilles følgende formel:

$$\begin{aligned} T_j &= (U_i - U_o) \cdot I \cdot R_{j-amb} + T_{amb} \\ &= (12\ V - 5\ V) \cdot 186\ mA \cdot 50^\circ C / W + 70^\circ C \\ &= 135.1^\circ C \end{aligned} \tag{4.5}$$

T_j: Temperaturen i spændingsregulatorens junction.
U_i: Inputspænding.
U_o: Outputspænding.
I: Strømforbrug.
$R_{j\text{-}amb}$: Termisk modstand fra junction til omgivelserne.
T_{amb}: Omgivelsestemperaturen(default 70°C).

Som det fremgår af formlen, bliver spændingsregulatorens junction 135.1 °C varm, ved en omgivelsestemperatur på 70°C. Da spændingsregulatoren må blive 150 °C, er det ikke nødvendigt at montere en køleplade.



Figur 4.7 Diagram over strømforsyningen.

3 V spændingsforsyningen er implementeret med en spændingsregulator af typen "LM317" og denne forsynes med 5 V fra 5 V forsyningen, da der derved bliver mindst spændingstab over spændingsregulatoren. Da den afsatte effekt i spændingsregulatoren afhænger af spændingsfaldet, bliver denne tilsvarende mindre.

Test
For at teste, om strømforsyningen kan levere den ønskede strøm, belastes den med en effektmodstand, således at den skal levere den strøm, som systemet maksimalt kommer til at trække. Det er konstateret, at strømforsyningen kan levere den ønskede strøm til systemet uden at blive varm. Ripplespændingen blev målt vha. et oscilloscop til 27 mV.

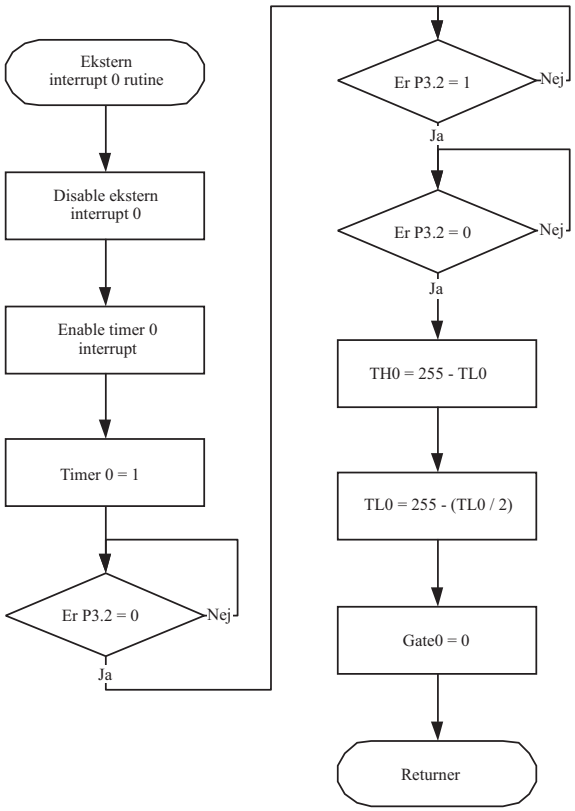
4.7 Procestest

De enkelte moduler er sat sammen, og der er foretaget en procestest på den samlede hardware. Dette er blandt andet gjort ved hjælp af forskellige testprogrammer, som interagerer med det totale system. GPS-interfacet er dog ikke testet sammen med den øvrige hardware, da den udleverede GPS-modtager var defekt. Dette betød, at der istedet blev lavet endnu et PC-interface, således at en PC kunne benyttes til at simulere GPS-modtageren.
Den samlede hardware blev testet ved at gennemgå en række tests som beskrevet herunder.

4.7.1 Test af VHF-radio tilslutning

For at teste om microcontrolleren er i stand til at modtage signaler fra VHF-radioen, som er forbundet til port 3.2 på microcontrolleren gennem et bufferkredsløb, er der udviklet et testprogram. Dette testprogram skal ved brug af interrupts og interne timere, gøre controlleren i stand til at synkronisere efter-, og modtage en bitstrøm på 9600 bps. I det følgende benyttes nogle registernavne fra microcontrolleren. Disse registre er nærmere beskrevet i appendiks A.

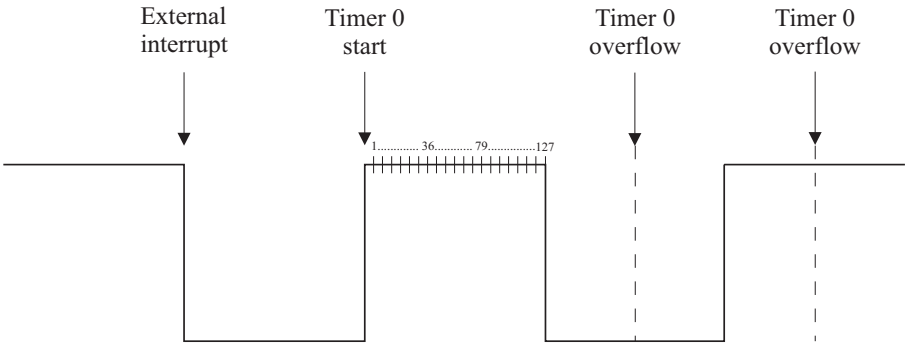
Gennemgang af testprogram:
Flowet i testprogrammet, som synkroniseres med en indkommende bitstrøm på 9600 bps, er vist nedenfor.



Figur 4.8 Flowchart for synkronisering af port 3.2.

Til at styre timingen, anvendes en intern timer (timer 0), som sættes op således, at port 3.2 skal være høj før timeren begynder at tælle. Timeren er en 16 bit timer, der har de to registre TH0 og TL0, som er henholdsvis den høje og den lave byte i timeren. Timeren skal sættes op til at tælle på de nederste 8 bit (TL0), mens de øverste 8 bit (TH0) skal holde en reload værdi. Timeren tæller op på TL0, og når der er overflow i TL0, hentes værdien fra TH0 ind i TL0. Ved at justere værdien i TH0, kan man justere tiden mellem hvert overflow. Ved overflow skal timeren generere et interrupt, som trigger en aflæsning af, hvad der står på port 3.2. Testprogrammet måler ved

hjælp af timer 0 tiden fra bit'en går høj til den går lav. Derved er der fundet den tid, der skal gå mellem hver aflæsning. For ikke at aflæse data på flankerne forskydes den første aflæsning en halv gange en bits tid, så værdien aflæses på midten af bit'en. Dette er illustreret på figur 4.9.



Figur 4.9 Synkronisering med bitstrømmen fra VHF-radioen. Der benyttes 2 bit til synkronisering af hastighed, hvorefter den egentlige sampling af signalet foretages.

For at undersøge, om VHF-radiotilslutningen virker, tilsluttes en tonegenerator (Philips PM5138), som sender firkantsignaler med en frekvens på 4800 Hz, til indgangen på VHF-interfacet. Der tilsluttes en "digital analyzer" HP 54620A på microcontrollerens port 4. Resultatet af samplingen af inputtet udskrives på denne port.
Målingerne viste imidlertid, at timingen ikke var præcis nok, da den ikke kunne aftaste 1192 bit i træk. Den skal kunne aftaste 1192 bit er, da det er den længste besked systemet skal kunne modtage. En af grundene var, at tonegeneratoren ikke var præcis nok. Tonegeneratoren udsendte firkantsignaler med en duty cycle på 104,4 µs, hvilket svarer til 9578,544 bitrate. Ifølge AIS-standarden skal bitrateraten være $9600 \pm 50\times10^{-6}$.
Den teoriske værdi for, hvor mange bit, der kan aftastes ved brug af den anvendte MCU, inden aftastningen bliver forkert, kan findes ud fra følgende formel:

$$Antal\ aftastninger = \frac{1}{2} \cdot \frac{\frac{1}{baud}}{\frac{1}{baud} - \left| \frac{f_{osc}}{12 \cdot baud} \right| \cdot \frac{12}{f_{osc}}} = \frac{1}{2} \cdot \frac{1}{1 - \left| \frac{f_{osc}}{12 \cdot baud} \right| \cdot \frac{12 \cdot baud}{f_{osc}}} \quad (4.6)$$

Baud: Er den aktuelle bitrate.
f_{osc}: Oscillatorfrekvensen.

For $9600 + 50\times10^{-6}$ bitrate og en oscillatorfrekvens på 14,7456 MHz er det maksimale antal aftastninger lig med 64 bit.
For $9600 - 50\times10^{-6}$ bitrate og en oscillatorfrekvens på 14,7456 MHz er det maksimale antal aftastninger lig med 96006144 bit.
Da 64 bit, som er *worst case*, ikke er nok, skal der laves noget andet. Dette kan eventuelt løses ved at anvende en microcontroller med en hurtigere / mere nøjagtig timer. Istedet er det valgt, at løse det softwaremæssigt ved, at der efter 20 aftastninger synkroniseres igen.
Det konkluderes, at der, ved brug af den softwaremæssige løsning, er opnået den ønskede funktion til modtagelse af signaler fra VHF-radioen.

4.7.2 Test af serielle porte

For at teste, om de serielle porte på controlleren virker, er der en del registre, der sættes op for at opnå de ønskede hastigheder på de to serielle porte: 38.400 baud på serielport 0 og 4800 baud på serielport 1. Til serielport 0 anvendes timer / counter 1, til at generere bitraten ud fra følgende formel:

$$baudrate = \frac{2^{SMOD}}{2} \cdot \frac{1}{16} \cdot \frac{f_{osc}}{12 \cdot (256 - (TH1))} \tag{4.7}$$

SMOD: Sidste bit i registeret PCON (se appendiks A). SMOD sættes.
f_{osc}: Oscillatorfrekvens = 14,7456 MHz
TH1: Timer/counter 1 bliver brugt som timer, hvor TH1 er den øverste byte, og TL1 er den nederste byte.

For at kunne anvende ovenstående formel skal timer/counter 1 stå i mode 2. Denne indstilling sættes i registeret TMOD (se appendiks A). C/T* bliver sat til 0, og derefter fungerer timer 1 som en timer, der tæller een op for hver maskincyklus. TH1 bliver sat til en reloadværdi, som overføres til TL1 hver gang, der er overflow i TL1.

For at sætte bitraten på serielport 1, benyttes nedenstående formel:

$$baudrate = \frac{f_{osc}}{32 \cdot (256 - S1REL)} \tag{4.8}$$

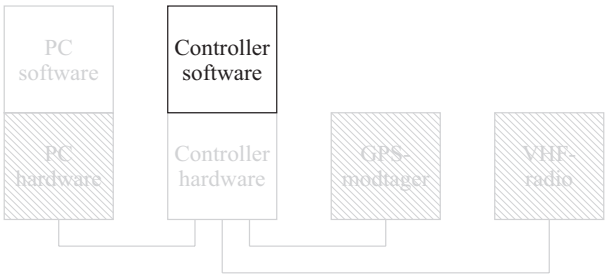
f_{osc}: Oscillatorfrekvensen = 14,7456 MHz
S1REL: 8-bit reload register til baudrategenerator på serielport 1. Sættes til 0xA0, hvilket giver en hastighed på 4800 baud.

For at kontrollere, om de ønskede hastigheder er opnået, laves et program, som skriver 0xAA (10101010b) ud på serielport 0 og 0x55 (01010101b) ud på serielport 1. Derefter sættes en "digital analyzer" (HP 54620A) på disse output og tidsforskellen fra outputtet går fra høj til lav måles. For serielport 0 måles en tidsforskel på 26,04 µs, hvilket svarer til 38.402 bitrate. På serielport 1 måles tidsforskellen til 104,1 µs, hvilket svarer til 9606 bitrate.
Det konkluderes, at de serielle porte virker efter hensigten.

Procestestens formål er at kontrollere, om hardwaren lever op til de krav, der er stillet i procesdesign. Modultesten af microcontrolleren viser at RAM / ROM kredsløbet og adressedekoderen virker efter hensigten. Da der kan kommunikeres med microcontrolleren gennem både serielport 1, serielport 2, og gennem VHF-interfacet, må det konkluderes, at de forskellige interfaces virker efter hensigten. Procestesten viser, at hardwaren virker efter hensigten.

5 Controller SW

I dette kapitel beskrives design, implementering og test af den software, som bruges til at styre den hardware, der blev realiseret i foregående kapitel. Først bestemmes processens eksterne grænseflader, og derefter opdeles processen i moduler. Disse moduler designes, testes og implementeres, og til sidst foretages en test af den samlede proces.



5.1 Procesdesign af software på microcontroller

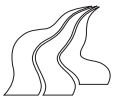
For at kunne designe softwaren til microcontrolleren, er det nødvendigt at kende de softwaremæssige grænseflader mellem systemets forskellige processer. Grænsefladerne, der er defineret i systemdesign afsnit 3.3, tages i betragtning, hvorefter processen opdeles i moduler, som derefter beskrives.

5.1.1 Opdeling i moduler

Processen opdeles i følgende 9 moduler:

1. Initialisering.
2. Ringbuffere.
3. Fjern bit fra bitstuffing.
4. CRC check.
5. Ny protokol.
6. Beregn ny CRC.
7. Datamodtagelse.
8. Dataafsendelse.
9. Hovedprogram.

1. Initialisering:
Dette modul skal foretage initialiseringen af de serielle porte, således at serielport 0 (PC) indstilles til 38.400 baud og serielport 1 (GPS) indstilles til 4800 baud. Endvidere skal det initialisere dataporte (VHF), samt de timere og tællere der evt. skal bruges.



2. Buffere:
Da microcontrolleren skal kommunikere med 3 forskellige enheder, vil det være en fordel at opsætte en ringbuffer til hver af disse. Dette modul skal således foretage håndtering af skrive- og læseoperationer i de 3 ringbuffere (GPSBuffer, VHFBuffer samt PCBuffer), der skal bruges i systemet.

3. Fjern bit fra bitstuffing:
Dette modul skal søge efter de ekstra bit, der evt. er indsat, og fjerne disse.

4. CRC check:
Dette modul skal kontrollere, om CRC på GPS- og VHF-beskederne er i orden.

5. Ny protokol:
Dette modul skal lave data fra GPS-modtageren om, således at den overholder den protokol, som benyttes mellem controller og PC. Det vil sige, at felterne for status og magnetisk variation fjernes.
Data fra VHF-radioen skal laves om fra binært format til tegnbaseret format, således at den overholder den protokol, som benyttes mellem controller og PC. Data skal laves om, således at den starter med "$", afsluttes med "<CR><LF>", og hvert datafelt er adskilt af et komma.

6. Beregn ny CRC:
Dette modul skal foretage beregning af CRC for de data, der sendes til PC'en.

7. Datamodtagelse:
Dette modul skal modtage data fra henholdsvis GPS-modtageren og VHF-radioen. GPS-modtageren er tilsluttet microcontrollerens serielle port 1 (port 6.1 og 6.2), og VHF-radioen er tilsluttet microcontrollerens port 3.2. Modulet skal således læse data fra disse porte og skrive dem til de respektive buffere.

8. Dataafsendelse:
Dette modul skal varetage afsendelsen af data fra controller til PC, herunder læsning af data fra PCBuffer og udskrivning på microcontrollerens serielle port 0 (port 3.0).

9. Hovedprogram.
Dette modul, som skal være en uendelig løkke, skal, ved hjælp af ovenstående funktioner, kontrollere, om der er data i buffere, kontrollere for CRC, fjerne bit fra bitstuffing, lave ny protokol, beregne ny CRC, samt sende data til PC'en. Endvidere er det i dette modul, at ugyldig data bliver frasorteret.

5.2 Moduldesign

I det følgende designes, implementeres og testes modulerne i denne proces.

5.2.1 Initialisering

Modulet har til opgave at initialisere microcontrolleren, således at de porte, den skal benytte til at kommunikere med, bliver initialiseret korrekt.

I appendiks A er det beskrevet, hvorledes microcontrollerens registre skal sættes, så portene er indstillet korrekt. I skemaet nedenfor, er det vist, hvordan portene skal indstilles:

Portindstillinger			
Port	Hastighed	Antal bit	Interrupt nr.
Seriel 0	38.400	8	4
Seriel 1	4.800	8	16
Port 3.2	9.600	1	0 og 1

Input: **Parametre:**
Init() Ingen input.

Output: **Returnerer:**
Init() Ingen output.

Implementering
Modulet implementeres efter flowchartet i appendiks D.1.

Test
For at teste om modulet fungerer efter hensigten, opstilles følgende test:

1 Der skrives et testprogram, der sætter registrene som beskrevet i appendiks A og udskriver indholdet af dem enkeltvis på port 4. Det er så muligt, at måle om registrene er sat som ønsket.
2 Der skrives et testprogram, som sender tegn på seriel port 0. På en PC modtages det afsendte.
3 Der skrives et testprogram, som modtager tegn på seriel port 1 og skriver det på port 4.
4 Der skrives et testprogram, som modtager data på port 3.2 og skriver det på port 4.1.

Testene er udført og resultatet var som følger.

1 Alle registre er sat som forventet.
2 De modtagne data på PC'en stemmer overens med det afsendte data.
3 Det modtagne data er korrekt udskrevet på port 4.
4 De korrekte modtagne data udskrives på port 4.1.

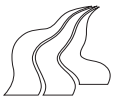
Det konkluderes dermed, at modulet virker efter hensigten.

5.2.2 Ringbuffere

Formålet med modulet er at håndtere læsning fra og skrivning til de ringbuffere, som skal benyttes på microcontrolleren, samt at holde styr på hvor meget data, der er i bufferne. På microcontrolleren skal der benyttes tre ringbuffere: GPS-interface, VHF-interface og PC-interface.

Input: **Parametre:**
WriteToGPSBuffer() Et tegn.
WriteToVHFBuffer() En bit.
WriteToPCBuffer() En streng med tegn, samt længden af denne.

Output: **Returnerer:**
ReadFromGPSBuffer() En streng med tegn, samt længden af denne.
ReadFromVHFBuffer() En streng med bit, samt antallet af disse.
ReadFromPCBuffer() Et tegn.



Design
Til at styre, hvor i bufferen der skal skrives og læses, anvendes to pointere, som peger på hhv. den næste ledige plads (skrivepointer), og næste data der skal læses (læsepointer). Efter skrivning til bufferen, opdateres skrivepointeren, og det kontrolleres, om bufferens skrivepointer er kommet til enden af bufferen. Hvis bufferen er fyldt, skal læsepointeren også opdateres, således at den kommer til at pege på det ældste data i bufferen. Ved læsning fra bufferen kontrolleres først om bufferen er tom, hvorefter data læses. Læsepointeren opdateres og det kontrolleres, om den er nået til enden af bufferen.
I GPS-bufferen modtages der tegn, og ikke hele frames. Det er derfor nødvendig, at kontrollere, om de modtagne tegn udgør en hel besked, inden der gives besked om, at der er modtaget data. I VHF-bufferen modtages der enkelte bit, og det er også her nødvendigt at kontrollere, om der er modtaget hele frames.

Implementering
Funktionerne i GPS-bufferen implementeres efter de flowcharts der findes i appendiks D.2.1.
Funktionerne i VHF-bufferen implementeres efter de flowcharts der findes i appendiks D.2.2.
Funktionerne i PC-bufferen implementeres efter de flowcharts der findes i appendiks D.2.3.

Test
For at teste om modulet fungerer efter hensigten, kontrolleres følgende:
1 Kan bufferen opdatere læsepointeren, hvis bufferen bliver fuld?
2 Kan der læses data, hvis bufferen er tom?
3 Nulstilles pointerne, når de kommer til enden af bufferen?
4 Kan bufferen identificere en hel frame?

De opstillede tests er udført på alle tre ringbuffere, vha. et lille program, som kalder de forskellige buffere, og resultatet er som følger:
1 Bufferen kan overskrive data og opdatere læsepointeren, hvis den er fuld.
2 Det er ikke muligt at læse data, hvis bufferen er tom.
3 Bufferne nulstillede pointerne, og virker derfor som ringbuffere.
4 Flagene sættes høj, når der er modtaget en frame.

5.2.3 Fjern bit fra bitstuffing
Beskederne fra VHF-radioen, startes og stoppes med henholdsvis et start- og et stopflag, for at modtageren kan finde ud af, hvornår en ny besked begynder og slutter. Start og stopflag er flag på 8 bit, hvor de 6 midterste bit er ettaller, og første og sidste bit er nuller. For at undgå at modtageren misfortolker en besked, må der ikke forekomme 6 ettaller lige efter hinanden. Dette kan opfyldes ved bitstuffing på afsendersiden og fjernelse af de ekstra bit på modtagersiden.

Input:	**Parametre:**
BitUnStuffing()	En streng med bit samt antallet af disse.

Output:	**Returnerer:**
BitUnStuffing()	En streng med bit samt antallet af disse.

Implementering
Modulet implementeres efter flowchartet i appendiks D.3.

Test
For at teste om modulet fungerer efter hensigten, opstilles følgende testscenarier:

1 En streng hvori der ikke findes 5 på hinanden følgende ettaller, samt dennes længde sendes ind i modulet, og skal komme uberørt ud.
2 En streng hvori der findes 5 på hinanden følgende ettaller, samt dennes længde sendes ind i modulet, og skal komme ud med nullet efter ettallerne fjernet, samt en længde der er een mindre.
3 En streng med 10 på hinanden følgende ettaller, samt dennes længde sendes ind i modulet, og skal komme ud med det 6. ettal fjernet, samt en længde der er een mindre.
4 En streng med 11 på hinanden følgende ettaller, samt dennes længde sendes ind i modulet, og skal komme ud med det 6. ettal fjernet samt nullet efter de 11 ettaller fjernet. Endvidere skal længden være to mindre.

Scenarierne er udført og resultatet var som følger:

1 Strengen kom uberørt ud.
2 Nullet efter de 5 ettaller var fjernet og længden var blevet een mindre.
3 Det 6. ettal var fjernet og længden var een mindre.
4 Det 6. ettal og nullet efter de 11 ettaller var fjernet, og længden var blevet to mindre.

Det konkluderes dermed, at modulet virker efter hensigten.

5.2.4 CRC-Check

For at kontrollere, om der er sket transmissionsfejl, er der på alle beskeder fra både GPS-modtageren og VHF-radioen vedhæftet en checksum. Beskederne fra GPS-modtageren bruger en checksumsmetode, der er defineret i NMEA-standarden, og beskederne fra VHF-radioen bruger en checksumsmetode, der er defineret i [AIS, 1998], og som kaldes CRC-ITU-T (også kaldet CRC-CCITT). Modulet deles op i to funktioner, hvor den ene kontrollerer checksum for GPS-beskederne og den anden for VHF-beskederne. For GPS-beskederne skal der foretages checksumberegning på den modtagne data, for derefter at sammenligne med den sendte checksum. For VHF-beskederne skal der beregnes checksum på den modtagne data inklusive checksum, og kontrolleres at resultatet bliver 0x1D0F, som er residuet for CRC-ITU-T. Beregningsmetoderne er nærmere beskrevet i appendiks B.

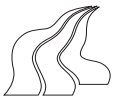
Input:	**Parametre:**
GPSCRCCheck()	En streng med tegn, samt længden af denne.
VHFCRCCheck()	En streng med bit, samt længden af denne.

Output:	**Returnerer:**
GPSCRCCheck()	"1" hvis CRC er OK og "0" ellers.
VHFCRCCheck()	"1" hvis CRC er OK og "0" ellers.

Implementering
Funktionerne i modulet er implementeret efter de flowcharts der findes i appendiks D.4.

Test
For at teste, om modulet fungerer efter hensigten, opstilles følgende testscenarier:

1 En GPS-besked med korrekt CRC, samt dennes længde, sendes ind i modulet, og der skal returneres et ettal.
2 En GPS-besked med forkert CRC, samt dennes længde, sendes ind i modulet, og der skal returneres et nul.



3 En VHF-besked med korrekt CRC, samt dennes længde, sendes ind i modulet, og der skal returneres et ettal.
4 En VHF-besked med forkert CRC, samt dennes længde, sendes ind i modulet, og der skal returneres et nul.

Udfaldet af scenarierne var som følger:
1 Der blev returneret et ettal.
2 Der blev returneret et nul.
3 Der blev returneret et ettal.
4 Der blev returneret et nul.

Det konkluderes dermed, at modulet virker efter hensigten.

5.2.5 Ny protokol

Beskederne fra VHF-radioen modtages bit for bit som defineret i [AIS, 1998]. GPS-modtagerens beskeder kommer fra en UART, hvorved der modtages hele tegn fra denne. Da beskederne skal sendes over en seriel RS232 forbindelse, til en PC, er det mest hensigtsmæssigt at konvertere data fra VHF-radioen til ASCII tegn.

Beskederne, som videresendes til PC'en, skal kun indeholde de informationer, der skal benyttes på denne, og derfor frasorteres de informationer, som ikke benyttes. Protokollerne, som benyttes til at sende beskeder til PC'en, skal ligge tæt op af NMEA-protokollen, hvorfor alle beskeder skal startes med en "$" og afsluttes med *<CRC><CR><LF>. Endvidere skal de forskellige informationer i beskeden være adskilt af et komma.

Input:	**Parametre:**
NewGPSProtokol()	En streng med tegn.
NewVHFProtokol()	En streng med bit, samt antallet af disse.

Output:	**Returnerer:**
NewGPSProtokol()	En streng med tegn.
NewVHFProtokol()	En streng med tegn, samt længden af denne.

Design af GPS-protokol

Data, som modtages fra GPS-modtageren, overholder NMEA-standarden [Cassidy, 1998]. Standarden angiver at beskeden skal starte med en header på 5 tegn, hvor de første 2 angiver afsenderID, og de næste 3 beskedtypen. Ved beskeder fra en GPS-modtager foreskrives det, i standarden, at afsender ID'en er "GP". De forskellige beskedtyper er ligeledes beskrevet i standarden, og her anvendes den beskedtype der hedder RMC (Recommended Minimum specific GPS / Transit Data). Denne besked indeholder information om UTC, status, position, fart, kurs, dato samt misvisning, og ser ud som følger:

$GPRMC,hhmmss.ss,a,ddmm.mmmm,n,dddmm.mmmm,w,z.z,y.y,ddmmyy,d.d,v

og det data som skal sendes videre til PC'en er:

$GPRMC,hhmmss.ss,ddmm.mmmm,n,dddmm.mmmm,w,z.z,y.y,ddmmyy

Som det ses, er det kun statusfeltet "a" og felterne for misvisning "d.d" og "v", som skal fjernes fra den gamle protokol.

Design af VHF-protokol
AIS-standarden benytter fire former for datakodning: Binær, 2's komplement, 6 bit ASCII og 8 bit ASCII. Først beskrives, hvordan de fire datakodninger konverteres til 8 bit ASCII, og derefter gennemgås beskeder for at beskrive, hvordan de er opdelt.

Konverteringen fra binær til decimaltal gøres efter følgende formel

$$N = \sum_{m=0}^{M-1} 2^m \cdot B_m \tag{5.1}$$

N: Naturligt tal i titalssytemet.
B_m: Binært tal på m'te plads, hvor m = 0 er det mindst betydende ciffer.
M: Antal bit.

Konverteringen fra 2's komplement til decimaltal
Hvis mest betydende bit er høj er tallet negativt og der benyttes 2's komplement til at angive værdien. Er mest betydende bit lav benyttes den normale binære konvertering.

$$Z = -2^{M-1} + \sum_{m=0}^{M-2} 2^m \cdot \mathrm{B}_m \tag{5.2}$$

Z: Helt tal.
B_m: Binært tal på m'te plads, hvor m = 0 er det mindst betydende ciffer.
M: Antal bit.

Konverteringen fra binær til 8 bit ASCII
ASCII er en standard, hvor hver tegn er repræsenteret ved en numerisk værdi. Ved 8 bit ASCII er der 256 forskellige tegn. For at konvertere til 8 bit ASCII, er det nødvendigt at opdele beskeden i bytes, for derefter at finde den tilhørende værdi for hvert tegn. For at finde denne værdi, benyttes formlen fra binær til decimaltal, hvor antallet af bit er 8. Denne værdi vil så repræsentere et tegn.

Konvertering fra binær til 6 bit ASCII
Princippet i denne konvertering er det samme som ved konvertering til 8 bit ASCII, blot opdeles beskeden i dele af 6 bit. Efter konverteringen til 6 bit ASCII, skal der konverteres til 8 bit ASCII. Denne konvertering gøres for at have alle informationer på samme form, og for at sende informationerne over en RS232 forbindelse.
Da 6 bit kun indeholder en del af tegnene i 8 bit ASCII, starter 6 bit ASCII ved tegn nummer 32 og slutter ved tegn nummer 95 i 8 bit ASCII.

Typer af beskeder
Der kan modtages tre forskellige typer beskeder fra VHF-radioen: Besked nr. 1, 2 og 3 indeholder information om position og retning, nr. 5 er statiske informationer og nr. 8 er en tekstbesked. Herunder ses, hvordan informationerne i de tre typer beskeder er opdelt:

Besked nr. 1, 2 og 3 (168 bit)		
Parameter	Antal bit	Konvertering
MMSI-nummer	30	Binær
Skibets status	2	Binær
Kursændring i grader pr. minut	8	2's komplement
Hastighed (SOG)	10	Binær
Positionens nøjagtighed	1	Binær
Længdegrad	28	2's komplement
Breddegrad	27	2's komplement
Beholdende kurs (COG)	12	Binær
Styrende kurs	9	Binær

Besked nr. 5 (416 bit)		
Parameter	Antal bit	Konvertering
MMSI-nummer	30	Binær
IMO-nummer	30	Binær (Max 9 decimaltal)
Kaldenavn	36	6 bit ASCII
Skibets navn	120	6 bit ASCII
Skibets type og last	8	Binær
GNSS antennes position	30	Binær
Forventet tid for ankomst	20	Binær
Dybdegang	8	Binær
Destination	120	6 bit ASCII

Besked nr. 8 (1192 bit)		
Parameter	Antal bit	Konvertering
Afsender MMSI-nummer	30	Binær
Besked	968	8 bit ASCII

Implementering
Funktionerne i modulet implementeres efter de flowcharts der findes i appendiks D.5.

Test
For at teste, om modulet fungerer efter hensigten, opstilles følgende testscenarier:
1 En GPS besked sendes ind i modulet og skal komme ud med felterne for status og mag. var. fjernet.
2 En VHF besked 1 sendes ind i modulet, og skal komme ud som defineret i procesdesign for microcontrolleren.
3 En VHF besked 2 sendes ind i modulet, og skal komme ud som defineret i procesdesign for microcontrolleren.
4 En VHF besked 3 sendes ind i modulet, og skal komme ud som defineret i procesdesign for microcontrolleren.
5 En VHF besked 5 sendes ind i modulet, og skal komme ud som defineret i procesdesign for microcontrolleren.
6 En VHF besked 8 sendes ind i modulet, og skal komme ud som defineret i procesdesign for microcontrolleren.

Udfaldet af scenarierne var som følger:
1 Der blev returneret GPS besked, som defineret i procesdesign for microcontrolleren.
2 Der blev returneret en VHF besked 1, som defineret i procesdesign for microcontrolleren.
3 Der blev returneret en VHF besked 1, som defineret i procesdesign for microcontrolleren.
4 Der blev returneret en VHF besked 1, som defineret i procesdesign for microcontrolleren.
5 Der blev returneret en VHF besked 5, som defineret i procesdesign for microcontrolleren.
6 Der blev returneret en VHF besked 8, som defineret i procesdesign for microcontrolleren.

Det konkluderes dermed, at modulet virker efter hensigten.

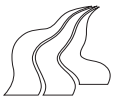
5.2.6 Beregn ny CRC
Idet PC'en ikke sender besked tilbage til controlleren om, at den har modtaget data, og for at undgå at PC'en benytter ugyldige data, forsynes alle beskeder med en checksum.
Der benyttes den samme metode til beregning af checksum, som benyttes i AIS-standarden. Metoden, der hedder CRC-ITU-T, er nærmere beskrevet i appendiks B. Modulet har således til formål at beregne checksum for den data, der står mellem "$" og "*", og tilføje denne efter data.

Input:	**Parametre:**
CalculateCRC()	En streng med tegn, samt længden af denne.

Output:	**Returnerer:**
CalculateCRC()	En streng med tegn, tilføjet checksum, samt længden af denne

Implementering
Modulet implementeres efter flowchartet i appendiks D.6.

Test
For at teste, om modulet fungerer efter hensigten, opstilles følgende testscenarie:
1 En streng data som følger sendes ind i modulet, hvilket skal resultere i en CRC på "IC": $GPRMC,174818.03,A,5831.6341,N,15848.5311,W,7.1,271.9,201100,101.4,W*00<CR><LF>.



Udfaldet af scenariet var som følger:
1 Der blev returneret:
$GPRMC,174818.03,A,5831.6341,N,15848.5311,W,7.1,271.9,201100,101.4,W*IC<CR><LF>.

Det konkluderes dermed, at modulet virker efter hensigten.

5.2.7 Datamodtagelse

Microcontrolleren skal modtage data fra GPS-modtageren på en seriel port, og fra VHF-radioen på en I/O-port. Ved modtagelse af data, skal en interruptroutine kaldes, således at det modtagne data bliver gemt i en buffer.

Input:	**Parametre:**
Seriel 1	Et tegn modtages ved interrupt 16.
Port 3.2	En bit modtages ved interrupt 1.

Output:	**Returnerer:**
WriteToGPSBuffer()	Et tegn.
WriteToVHFBuffer()	En bit.

Design

Ved modtagelse af data fra GPS-modtageren, sættes et interruptflag, som skal bevirke, at microcontrolleren gemmer det modtagne data i GPS-bufferen. Efter at data er gemt, skal interruptflaget deaktiveres for at indikere, at tegnet er blevet gemt i bufferen. Microcontrolleren kan herefter fortsætte hvor den slap, da den blev forstyrret af interruptet.

Data fra VHF-radioen modtages på et databen, hvor det er muligt at detektere en negativ flanke. Ved at lade microcontrolleren bestemme afstanden mellem to flanker i "training sequence", kan det bestemmes, med hvilken frekvens der skal læses data. For at sikre, at det er en "training sequence", der synkroniseres efter, undersøges det, om de efterfølgende dataaflæsninger er skiftevis høje og lave. Hvis det konstateres, at der er synkroniseret efter en "training sequence", aflæses der data fra VHF-radioen, indtil der er modtaget et start- og stopflag. Ved fejlsynkronisering bliver der aldrig modtaget et start- og stopflag og derfor er det nødvendig at have en timeout, som sikrer, at der bliver startet en ny synkronisering.

Implementering

Modulet implementeres efter flowchartet i appendiks D.7.

Test

Da portene på microcontrolleren er en del af den eksterne grænseflade, er de blevet testet i afsnittet "Controller Hardware. Som det fremgår af denne test, er det ikke muligt at læse mere end 64 bit pr. synkronisering. Derfor er der indlagt en resynkronisering, som bevirker at timeren nulstilles efter hver 20. dataaflæsning.

5.2.8 Dataafsendelse

Efter de modtagne beskeder er blevet behandlet af microcontrolleren, skal de sendes videre til PC'en over en RS232 forbindelse. For at styre denne overførsel, er det nødvendigt at benytte interrupts, således at der ikke skrives for hurtigt til UART'en. Interruptflaget fungerer ved, at UART'en sætter flaget højt efter hver afsendelse, og derved bliver interruptroutinen kaldt således, at det næste tegn kan blive sendt.

Input:	**Parametre:**
ReadFromPCBuffer()	Et tegn, samt "1" hvis der er data i bufferen, og "0" ellers.

Output:	**Returnerer:**
S0BUF	Data sendes til bufferen, som afsender det til UART'en.

Implementering
Modulet implementeres efter flowchartet i appendiks D.8.

Test
Dette modul testes ved at sende forskellige beskeder fra microcontrolleren til PC'en. Ved at sammenligne den afsendte og den modtagne besked, kan det kontrolleres, om der er opstået fejl under transmissionen. De afsendte frames modtages som ønsket.

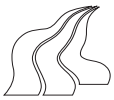
5.2.9 Hovedprogram

Dette modul er det centrale i processen, og det har til formål at samle de andre moduler ved at kalde funktionerne i dem. Modulet skal således kontrollere om der er data i bufferne, hente data, kontrollere checksummer, fjerne ekstra bit, lave nye protokoller, beregne nye checksummer, samt sende data til PC. Selve modulet skal ikke udføre alle disse funktioner, men blot kalde de respektive moduler, der udfører dem.

Input:	**Parametre:**
DataInGPSBuffer()	"0" eller "1" afhængig af, om der er data i GPSBuffer.
ReadFromGPSBuffer()	En streng med tegn, samt længden af denne.
GPSCRCCheck()	En streng med tegn, længden af denne, samt "0" hvis CRC ikke er OK og "1" hvis CRC er OK.
NewGPSProtokol()	En streng med tegn.
DataInVHFBuffer()	"0" eller "1" afhængig af, om der er data i VHFBuffer.
ReadFromVHFBuffer()	En streng med bit, samt antallet af disse.
BitUnStuffing()	En streng med bit, samt antallet af disse.
VHFCRCCheck()	En streng med bit, antallet af disse, samt "0" hvis CRC ikke er OK og "1" hvis CRC er OK.
NewVHFProtokol()	En streng med tegn, samt længden af denne.
CalculateCRC()	En streng med tegn.
WriteToPCBuffer()	Ingen input.

Output:	**Returnerer:**
ReadFromGPSBuffer()	Ingen Output.
GPSCRCCheck()	En streng med tegn, samt længden af denne.
NewGPSProtokol()	En streng med tegn.
ReadFromVHFBuffer()	Ingen Output.
BitUnStuffing()	En streng med bit, samt antallet af disse.
VHFCRCCheck()	En streng med bit, samt antallet af disse.
NewVHFProtokol()	En streng med bit, samt antallet af disse.
CalculateCRC()	En streng med tegn, samt længden af denne.
WriteToPCBuffer()	En streng med tegn, samt længden af denne.

Implementering
Modulet implementeres efter flowchartet i appendiks D.9.



Test
For at teste, om modulet fungerer efter hensigten, opstilles følgende testscenarier:

1 Der fyldes en datastreng, som overholder GPS-protokollen, i GPSBuffer. Dette skulle medføre, at der bliver afsendt en GPS-besked til PC'en. Strengen skal opfylde kravene stillet i systemdesign.
2 Der fyldes en datastreng, som overholder VHF-protokollens besked 1, i VHFBuffer. Dette skulle medføre, at der bliver afsendt en VHF-besked 1 til PC'en. Strengen skal opfylde kravene stillet i procesdesign.
3 Der fyldes en datastreng, som overholder VHF-protokollens besked 2, i VHFBuffer. Dette skulle medføre, at der bliver afsendt en VHF-besked 1 til PC'en. Strengen skal opfylde kravene stillet i systemdesign.
4 Der fyldes en datastreng, som overholder VHF-protokollens besked 3, i VHFBuffer. Dette skulle medføre, at der bliver afsendt en VHF-besked 1 til PC'en. Strengen skal opfylde kravene stillet i systemdesign.
5 Der fyldes en datastreng, som overholder VHF-protokollens besked 5, i VHFBuffer. Dette skulle medføre, at der bliver afsendt en VHF-besked 5 til PC'en. Strengen skal opfylde kravene stillet i systemdesign.
6 Der fyldes en datastreng, som overholder VHF-protokollens besked 8, i VHFBuffer. Dette skulle medføre, at der bliver afsendt en VHF-besked 8 til PC'en. Strengen skal opfylde kravene stillet i systemdesign.

Udfaldet af scenarierne var som følger:

1 Der blev sendt en GPS besked til PC'en.
2 Der blev sendt en VHF besked 1 til PC'en.
3 Der blev sendt en VHF besked 1 til PC'en.
4 Der blev sendt en VHF besked 1 til PC'en.
5 Der blev sendt en VHF besked 5 til PC'en.
6 Der blev sendt en VHF besked 8 til PC'en.

Det konkluderes dermed, at modulet virker efter hensigten.

5.3 Procestest

For at teste om controllernes software lever op til kravene, stillet i procesdesign for softwaren, opstilles følgende testscenarier.

5.3.1 Test af GPS-delen

Der tilsluttes en PC til serielport 0, dens serielport stilles til at modtage med 38400 baud, og der startes et terminalvindue, hvorved man kan aflæse hvad microcontrolleren sender.
Der tilsluttes en PC til serielport 1, og dens serielport stilles til at sende med 4800 baud, og der startes et terminalvindue.

1 Der sendes en korrekt GPS-besked via terminalvinduet, og det kontrolleres at den modtages korrekt på den anden PC.
2 Der sendes en GPS-besked med forkert CRC, og det kontrolleres at den ikke sendes videre.
3 Der sendes en GPS-besked med forkert længde, dvs. manglende datafelter, og det kontrolleres at den ikke sendes videre.

Udfaldet af scenarierne var som følger:

1 Den sendte besked blev korrekt modtaget, med felterne for status og magnesisk variation fjernet. Endvidere var der tilføjet ny CRC, som var korrekt beregnet.

2 Der blev ikke modtaget noget.
3 Der blev ikke modtaget noget.

Det konstateres dermed, at GPS-delen virker efter hensigten.

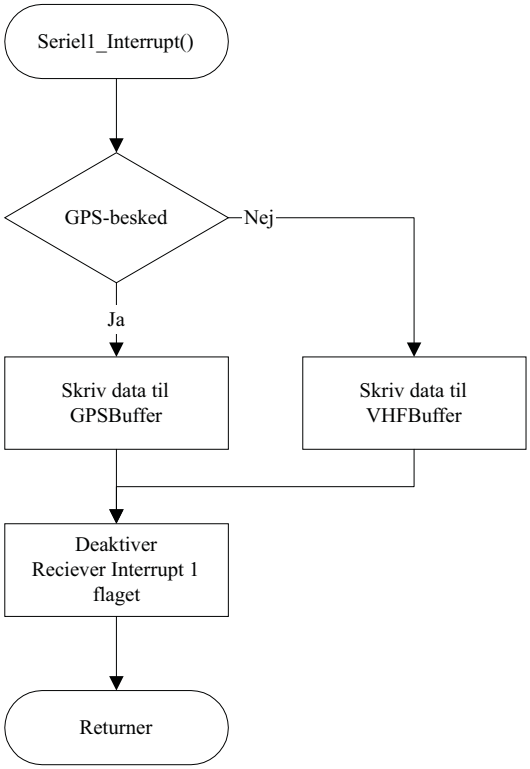
5.3.2 Test af VHF-delen

Der tilsluttes en PC til serielport 0, dens serielport stilles til at modtage med 38.400 baud, og der startes et terminalvindue, hvorved man kan aflæse, hvad microcontrolleren sender.
Endvidere tilsluttes en testPC's parallelport (ben 11) til microcontrollerens port 3.2, og der skrives et testprogram, som sender VHF-beskeder ud på testPC'ens parallelport.

1 Der sendes en korrekt VHF-besked 1 via testprogrammet, og det kontrolleres, at den modtages korrekt på den anden PC.
2 Der sendes en korrekt VHF-besked 2 via testprogrammet, og det kontrolleres, at den modtages korrekt på den anden PC.
3 Der sendes en korrekt VHF-besked 3 via testprogrammet, og det kontrolleres, at den modtages korrekt på den anden PC.
4 Der sendes en korrekt VHF-besked 5 via testprogrammet, og det kontrolleres, at den modtages korrekt på den anden PC.
5 Der sendes en korrekt VHF-besked 8 via testprogrammet, og det kontrolleres, at den modtages korrekt på den anden PC.
6 Der sendes en VHF-besked 1 med forkert CRC via testprogrammet, og det kontrolleres, at den ikke sendes videre til den anden PC.
7 Der sendes en VHF-besked 1 med forkert længde via testprogrammet, og det kontrolleres, at den ikke sendes videre til den anden PC.

Udfaldet af scenarierne var som følger:
1 Der blev ikke modtaget noget.
2 Der blev ikke modtaget noget.
3 Der blev ikke modtaget noget.
4 Der blev ikke modtaget noget.
5 Der blev ikke modtaget noget.
6 Der blev ikke modtaget noget.
7 Der blev ikke modtaget noget.

Som det ses af testresultatet, blev der ikke modtaget de forventede beskeder. Dette kunne tyde på, at den interruptroutine, der skulle lægge data i bufferen, ikke fungerede korrekt. For at finde ud af, om dette var tilfældet, blev microcontrollerprogrammet modificeret således, at det, når det havde aflæst en bit på port 3.2, udskrev værdien på port 4.0. Hermed kunne det, ved hjælp af en logic analyzer, kontrolleres om microcontrolleren aflæste data korrekt. Det viste sig, at det var her fejlen lå, og at microcontrolleren til tider aflæste en forkert værdi.
For at kontrollere, om der var andre fejl i VHF-delen af softwaren, blev microcontrollerprogrammet omprogrammeret, således at det modtog både GPS- og VHF-data via serielport 1. Interruptroutinen for serielport 1 blev modificeret, således at den fyldte GPS-data i GPS-bufferen, og VHF-data i VHF-bufferen. Modifikationen ses på nedenstående flowchart.



Figur 5.1 Flowchart for det modificerede modul Datamodtagelse.

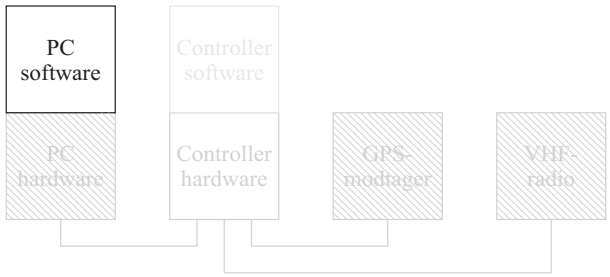
Herefter blev testscenariet gennemført endnu en gang, og denne gang var resultatet mere positivt. Resultatet var som følger:

1. Der blev modtaget en VHF besked 1, som defineret i procesdesign.
2. Der blev modtaget en VHF besked 1, som defineret i procesdesign.
3. Der blev modtaget en VHF besked 1, som defineret i procesdesign.
4. Der blev modtaget en VHF besked 5, som defineret i procesdesign.
5. Der blev modtaget en VHF besked 8, som defineret i procesdesign.
6. Der blev ikke modtaget noget.
7. Der blev ikke modtaget noget.

Det kan dermed konkluderes at den resterende VHF-del af microcontrollersoftwaren virker korrekt.
Det blev valgt at nedprioritere at finde fejlen i interruptroutinen, hvorfor microcontrolleren modtager både GPS og VHF- data på serielport 1.

6 PC-program

Det program, som er skrevet til PC'en, er skrevet i programmeringssproget JAVA. JAVA er et udpræget objektorienteret programmeringssprog, og derfor er der valgt at bruge UML (Unified Modelling Language) under udviklingsfasen. Således er dokumentationen af programmet skrevet i henhold til UML standarden. [Eriksson,1998]



6.1 Use case modellering

For at fastlægge, hvordan programmet skal agere med brugeren og controlleren, opstilles en række "use cases", som definerer de mulige situationer, der kan opstå ved programmets grænseflader. Controlleren opfattes i denne sammenhæng som en aktør, således at kommunikationen kan beskrives med use cases.

Figur 6.1 Use case diagram for PC-program.

6.1.1 User

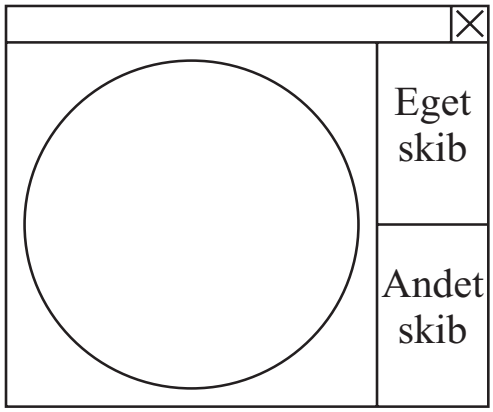
Der er følgende funktioner, som udgør "use cases" på brugerens side:

- Overvågning (Default mode).
- Redigere / Se statiske informationer om eget skib.
- Redigere / Se rejserelaterede informationer om eget skib.
- Se statiske informationer om det valgte skib.
- Se rejserelaterede informationer om det valgte skib.
- Vælge skib til visning.
- Få vist dialogboks med besked fra andet skib.
- Afslutte programmet.

I det følgende beskrives disse use cases i detaljer:

Overvågning

Programmet starter op i denne mode, som er default mode. Her vises et felt med alle nærtliggende skibes placering og hastighed i symboliseret af vektorer. Desuden vises tekstfelter med dynamiske oplysninger om ens eget skib og om et valgt, andet skib.
Der skal vises knapper, som giver mulighed for at aktivere funktionerne, som beskrevet i de øvrige "use cases".
Både grafen og oplysningerne om skibene skal opdateres, når der kommer nye informationer fra controlleren. Skærm-layoutet er skitseret på fig. 6.2.



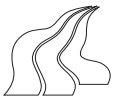
Figur 6.2 Skitse over skærmbilledet. I højre side er der regioner til knapper m.m.

Redigere / Se statiske informationer om eget skib
For at redigere eller se statiske informationer om eget skib, skal brugeren trykke på knappen "Static", som er placeret i regionen "Eget skib".
Når der trykkes på knappen, skal der komme en dialogboks frem, som giver mulighed for at indtaste og redigere følgende informationer om skibet.

MMSI number:	Skrives / Vises i et tekstfelt
Position accuracy:	Kan være høj eller lav. Vælges / Vises vha. "radio buttons"
IMO number:	Skrives / Vises i et tekstfelt
Call sign:	Skrives / Vises i et tekstfelt
Name:	Skrives / Vises i et tekstfelt
Position of GNSS antenna:	Vælges / Skrives i fire tekstfelter, som giver mulighed for indtastning af værdierne A, B, C og D.
Type of navigation sensor:	Vælges / Vises ud fra en "combo-box", med muligheder for at vælge de otte, i AIS-standarden, definerede værdier.

Redigere / Se rejserelaterede informationer om eget skib
Der skal ligeledes være en knap, som bringer en dialogboks op, der giver mulighed for at se eller redigere de rejserelaterede informationer om eget skib (se kravsspecifikation). Knappen hedder "Voyage", og er ligeledes placeret i regionen "Eget skib".
I dialogboksen skal følgende informationer vises, og der skal være mulighed for at redigere dem:

Destination:	Rejsens destination, som skrives / vises i et tekstfelt.
ETA:	Expected Time of Arrival angiver den forventede ankomsttid til destinationen. Skrives / Vises i fire tekstfelter til hhv. måned, dag, time og minut.
Actual Draught:	Skibets dybdegang skrives / vises i et tekstfelt.
Type of Ship / Cargo:	Dette skal vælges / vises vha. to "combo-boxe", hvoraf nummer to kun aktiveres, hvis den første har visse værdier. Den første angiver "type of ship", og hvis denne er et passager-, fragtskib eller lignende aktiveres den anden combo-box, og giver mulighed for at definere denne last. [AIS, 1998, pkt. 3.3.8.2.3.1]
Navigational Status:	Skibets status vælges / skrives vha. en combo-box med følgende fire valgmuligheder: "under way", "at anchor", "not under command" og "restricted manoeuvrability".



Se statiske informationer om det valgte skib
Disse informationer præsenteres i en dialogboks, når der trykkes på en knap kaldet "Static", i regionen "Andet skib". Dialogboksen skal være magen til den der benyttes ved visning af statiske informationer om eget skib, blot med den forskel, at det ikke skal være muligt at ændre informationerne. Felterne skal således være fadede (grå).

Se rejserelaterede informationer om det valgte skib
I regionen "Andet skib" skal der ligeledes være en knap; "Voyage", som får en dialogboks med informationerne til at komme frem på skærmen. Der skal vises de samme informationer, som ved eget skib, men det gælder også her, at det ikke skal være muligt at ændre informationerne. Derfor skal felterne også her være fadede.

Vælge næste skib til visning af dynamiske informationer
I regionen "Andet skib" skal der være en "Next"-knap, som giver mulighed for at bladre fremad i de skibe, der måtte være indenfor den kritiske radius. Dermed vælges det næste skib til visning.

Vælge forrige skib til visning af dynamiske informationer
Det forrige skib skal kunne vælges på samme måde som næste med en knap, "Previous".

Få vist dialogboks med besked fra andet skib
Hvis der modtages en tekstbesked fra et andet skib, skal der "poppe" en dialogboks op, som viser beskeden samt MMSI-nummeret på dens afsender.

Afslutte programmet
Programmet skal kunne afsluttes med krydset i hovedvinduets øverste højre hjørne.

6.1.2 Controller

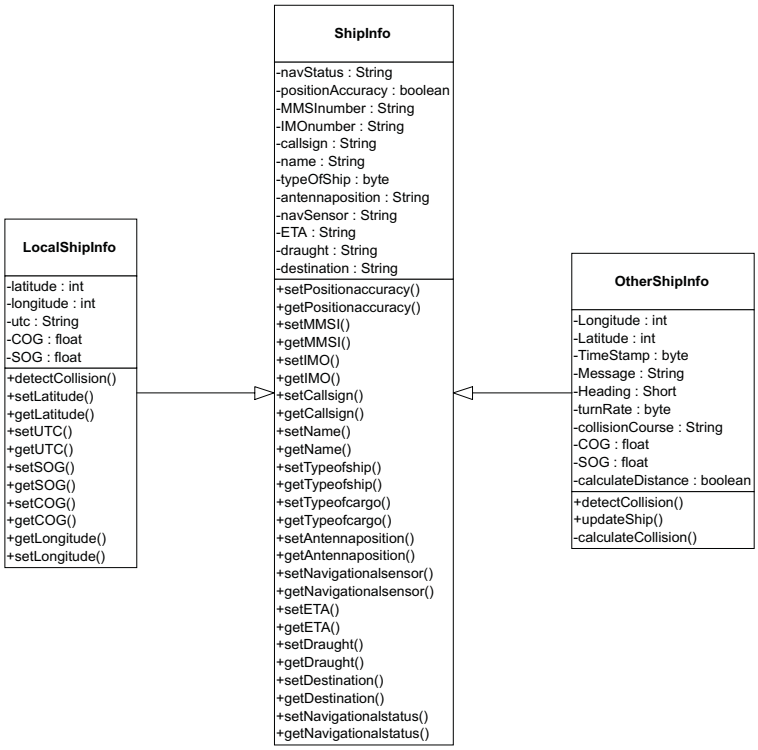
Controlleren kommunikerer med PC-programmet i en række scenarier, og disse udgør følgende "use cases" på controllerens side:

- Levere GPS-data med egen position m.m.
- Levere positionsrapport fra et andet skib.
- Levere statiske og rejserelaterede informationer fra et andet skib.
- Levere tekstbesked fra andet skib.

Disse scenarier er meget ens, og kan derfor beskrives under et:
RS232-forbindelsen sørger for, at data leveres og ender i bufferen på PC'ens UART. Det er PC-programmets opgave at tømme denne buffer, hver gang der er et tegn i den. Controlleren leverer en CRC-checksum, og det forventes, at PC-programmet udfører CRC-check, og ser bort fra data, hvis de er fejlbehæftede.

6.2 Domæne Analyse

I analyse af processen: "PC-program", er det essentielt at komme frem til dennes domæneklasser. For at nå frem til disse domæneklasser, betragtes de, i kravsspecifikationen, opstillede krav samt det "use case diagram", der er opstillet for processen. Ud fra dette ses det, at det primære funktionskrav til programmet består i at lagre informationer om forskellige skibe. Der skal lagres informationer om "eget" skib og andre skibe. Da mange af disse informationer er ens, vil man med fordel kunne benytte nedarvning. Derudfra vurderes det, at processens domæneklasser er: ShipInfo, LocalShipInfo og OtherShipInfo.
På figur 6.4 ses et domæneklassediagram, hvor disse klassers forhold til hinanden er illustreret.



Figur 6.3 De tre domæneklasser. LocalShipInfo og OtherShipInfo nedarver egenskaber fra ShipInfo.

Det understreges, at der på dette stadie i udviklingsfasen er tale om en skitse af disse klasser. De metoder og attributter, som defineres i de forskellige klasser er ikke endegyldige, men blot elementer, som på dette niveau i udviklingsfasen, vurderes at skulle bruges.

6.2.1 Shipinfo

Denne klasse er tiltænkt at skulle indeholde de metoder, som de to domæneklasser OtherShipInfo og LocalShipInfo begge skal bruge. Derfor opbygges denne klasse således, at metoder og attributter nedarves til LocalShipInfo og OtherShipInfo.
I denne klasse findes metoder til at lagre informationerne som den enten får fra CI eller GUI. Det skal være muligt at tilgå og opdatere disse objekters indhold fra CI og GUI.

6.2.2 LocalShipInfo

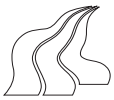
Der laves en domæneklasse med det ene formål at oprette et objekt, hvori informationer om eget skib opbevares. I denne domæneklasse skal position, kurs og fart for eget skib gemmes, da disse informationer ikke sendes i samme format fra VHF-radioen og GPS-modtageren.
Desuden skal der checkes for kollision med alle andre skibe, når der modtages nye data fra GPS-modtageren.

6.2.3 OtherShipInfo

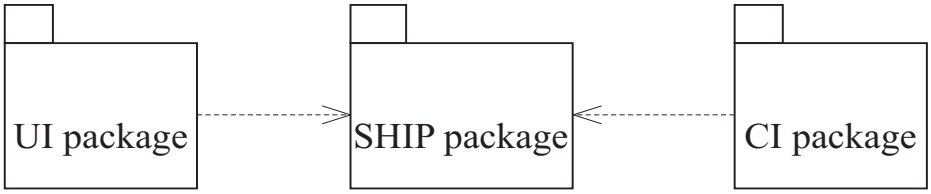
I denne domæneklasse skal findes en metode til oprettelse af de objekter, der repræsenterer andre skibe. Der checkes om skibene er indenfor en radius af 3 sømil. Hvis de er udenfor denne radius, slettes objektet. Er skibet indenfor de 3 sømil, checkes der for kollision med "eget" skib.

6.3 Arkitektur design

Efter at domæneklasserne for programmet er fastlagt, kan de øvrige "support klasser" tilføjes, således at programmet får en fuldstændig struktur. Disse klasser er dem, som gør det muligt for aktørerne, at interagere med domæneklasserne. For at holde de applikationsafhængige



domæneklasser adskilt fra de tekniske "support klasser", inddeles disse i forskellige pakker. Dette sikrer en nemmere vedligeholdelse, som ikke får indflydelse på de øvrige pakker.
Programmet struktureres derfor i tre pakker som vist på nedenstående figur.



Figur 6.4 De tre pakker som programmet består af. Pilenes retning indikerer, hvorfra data hentes / gemmes.

UI package: (UserInterface package)
Denne pakke består af de klasser, som brugeren direkte interagerer med. Denne samling af klasser skal opfylde de krav, som blev stillet, dels i kravsspecifikationen og dels i "Use case modellering". Endvidere skal klasserne have en simpel grænseflade til "SHIP package". Dette betyder, at alt kommunikation, om muligt, skal foregå fra brugerfladen og til "SHIP package" og ikke omvendt.

SHIP package:
Denne pakke indeholder alle de applikationsafhængige klasser (domæneklasser). Klasserne i denne pakke indeholder de informationer og data, som de øvrige pakker leverer.
SHIP package skal om muligt være et passivt element, som blot modtager og afleverer data, når de øvrige pakker efterspørger den. Dette skal sikre en enklere grænseflade til de to øvrige pakker.

CI package: (ControllerInterface package)
Pakken modtager informationer fra systemets anden aktør, Controlleren. CI package skal sørge for at aflevere de data, den modtager fra controlleren de rigtige steder i SHIP package.

6.4 Detaljeret design af SHIP package

Som beskrevet under afsnittet "Domæne analyse", har klasserne i denne pakke til formål at lagre data om eget skib, samt data fra de skibe, der er modtaget data fra.
Da alle data kommer fra CI-package i string-format, skal visse informationer konverteres til andre datatyper. Der er en række metoder og attributter, som er fælles for alle skibe, og disse er placeret i klassen ShipInfo. Klasserne LocalShipInfo og OtherShipInfo arver derfor metoder og attributter fra ShipInfo.

Algoritme til kollisionsberegning
Det er nødvendigt at undersøge, om der er kollisionsfare, hver gang et skib sender en positionsrapport (frame 1), og hver gang der kommer nye data fra GPS'en.
I objektet til hvert skib findes en metode, detectCollision(), til at kontrollere, om det er på kollisionskurs med "eget" skib. Denne metode kaldes, hver gang det pågældende skib sender en positionsrapport. For en forklaring af, hvordan selve kontrollen udføres, se afsnit 6.4.3 OtherShipInfo.
I objektet "eget" skib skal der også være en detectCollision(). Denne skal kaldes, når skibets position, kurs og fart opdateres med data fra GPS-modtager. Den skal ikke selv beregne om der er kollisionsfare, men sørge for, at alle andre skibe under overvågning gør dette.

6.4.1 ShipInfo

Denne klasse indeholder en række attributter, som har samme format for alle skibe. Det vil sige, for eget skib og andre skibe.
Værdierne gemmes i attributter med "private" synlighed, og det er derfor nødvendigt at implementere metoder til at hente og sætte dem med. Disse metoder navngives med hhv. getDataNavn() og setDataNavn().
De data, som kommer til denne pakke, er alle i string format, men ikke alle skal gemmes i dette format, og må derfor konverteres i deres respektive "set-metoder".
De attributter, der blot skal gemmes uden konvertering, er følgende:

- MMSI-number
- IMO-number
- Call sign
- Name
- Position of GNSS Antenna A
- Position of GNSS Antenna B
- Position of GNSS Antenna C
- Position of GNSS Antenna D
- ETA
- Actual draught
- Destination

Der er tre værdier, der skal konverteres:

- Position accuracy: Denne værdi gemmes i en "boolean", som bliver sat efter om der står "0" eller "1" i den modtagne string.
- Navigational Status: Gemmes i en "byte", som får samme værdi, som det tal, der står i den modtagne string.
- Type of navigation sensor: Gemmes i en "byte", som får samme værdi, som det tal, der står i den modtagne string.

Den sidste værdi, "Type of ship" kan komme ind i denne pakke i forskellige formater, alt afhængigt af, hvor den kommer fra; Hvis den kommer fra CI, er den, som de andre derfra, en string, men hvis den kommer fra UI, er den gemt i en byte. Dette er håndteret ved at lave to forskellige metoder med det samme navn, hvilket kun er muligt, fordi de to funktioner tager parametre af forskellig type. Det kaldes at "overloade" metoden. Hvis metodenavnet kaldes med en string, aktiveres den version af metoden, som konverterer en string til en byte før den gemmes, og hvis metodenavnet kaldes med en byte, sørger den anden version blot for at gemme den.

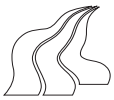
6.4.2 LocalShipInfo

Der oprettes kun eet objekt af denne klasse. Det har til opgave at gemme og, om nødvendigt, konvertere samtlige informationer om eget skib.
Klassen arver metoderne i ShipInfo, hvilket betyder, at der i LocalShipInfo skal implementeres metoder til at håndtere de resterende attributter med:

latitude: Metoden til at sætte denne attribut med er speciel, idet den tager to strings som parametre. Den første angiver positionen i grader og minutter, og den anden angiver retningen (nordlig eller sydlig). Disse skal omsættes til en integer, hvor fortegnet angiver retningen. (sydlig = negativ)

longitude: Set-metoden til denne variabel implementeres efter samme princip som ovenstående, hvor vest er negativ.

UTC: UTC-tiden, som GPS-modtageren leverer, gemmes i en string. Den bruges dog ikke i programmet, men skal alligevel gemmes, for at give mulighed for at udvide



programmet til eksempelvis at vise tiden på skærmen. Desuden skal den præcise UTC-tid angives, når man sender en positionsrapport, hvilket kunne blive aktuelt ved en udbyggelse af systemet.

COG: Course Over Ground konverteres til et tal af typen float, før det gemmes.

SOG: Speed Over Ground konverteres til et tal af typen float, før det gemmes.

Dataflowet gennem metoden kan ses i appendiks F.1.1.

detectCollision()

Denne metode kaldes, hver gang der har været data fra GPS-modtageren. Den indeholder en forløkke, der løber gennem et array af skibe, og sikrer at alle eksisterende skibe foretager en ny beregning af kollisionsfare. Dataflowet gennem metoden kan ses i appendiks F.1.1.

6.4.3 OtherShipInfo

Der oprettes et objekt af denne klasse for hvert skib, der har sendt informationer over VHF-radioen. Disse objekter gemmes i et array af skibe, for at lette tilgangen til de enkelte objekter. Det er således muligt, at referere til objektets plads i arrayet.

På samme måde som LocalShipInfo, nedarver OtherShipInfo også metoderne og attributterne fra ShipInfo, og der er også brug for en række ekstra metoder og attributter.

En række af disse attributter skal blot konverteres til forskellige typer af tal og gemmes:

- latitude
- longitude
- timeStamp
- COG
- SOG
- heading
- turnRate

Der er oprettet en string, som er beregnet til at indeholde tekstbeskeder fra andre skibe, men i programmets nuværende udgave benyttes denne ikke. Den er inkluderet af hensyn til udvidelsesmulighederne. Der kunne fx. oprettes et register over modtagne beskeder, således at man senere kunne bladre i dem.

Attributten collisionCourse er speciel, idet den ikke sættes udefra; den sættes fra detectCollision() metoden, hvis der er fare for kollision med det pågældende skib.

detectCollision()

Denne metode kaldes med en parameter, som angiver, hvilket skib i arrayet der er tale om. detectCollision() kalder calculateDistance() (se nedenfor), som bestemmer om skibet er indenfor en radius af 3 sømil af "eget" skib.

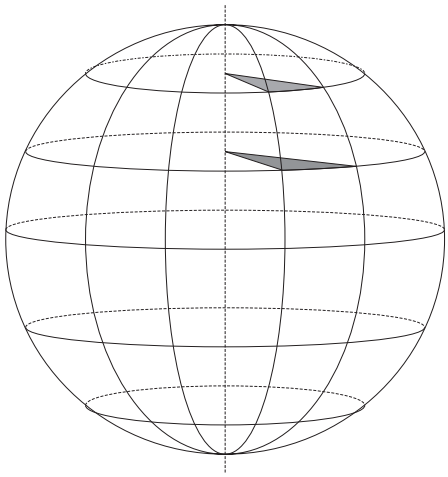
Hvis skibet er indenfor grænsen, kaldes calculateCollision(), og skærmen genoptegnes.

Hvis skibet er udenfor grænsen, slettes det fra arrayet.

calculateDistance()

Parametrene til denne metode angiver koordinaterne til "eget" skib og det andet skib.

Det første, metoden gør, er at beregne afstanden mellem længdegraderne, da denne er afhængig af, hvilken breddegrad man befinder sig på. Se figur 6.5.



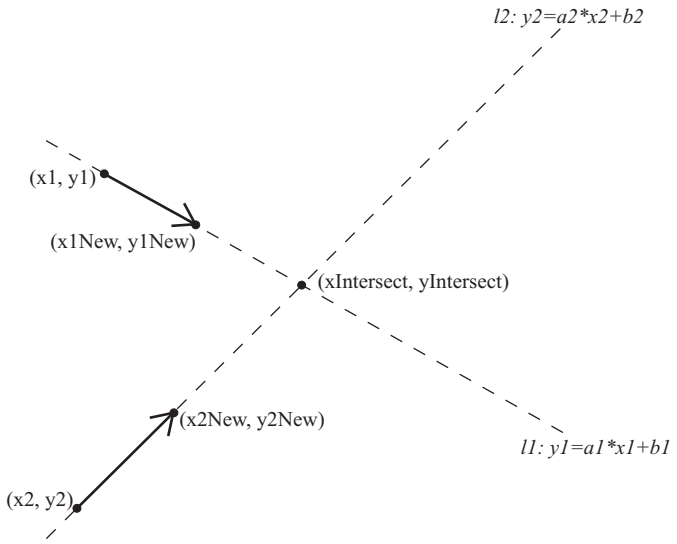
Figur 6.5 Variation af afstanden mellem længdegrader.

Dernæst betemmes den direkte afstand mellem punkterne, idet kuglekoordinaterne tilnærmes med retvinklede koordinater. Dette anses for rimeligt, da der er tale om så kort en afstand, at variationen i afstanden mellem længdegraderne er ubetydelig.
Metoden returnerer "true", hvis afstanden er mindre end 3 sømil og "false" ellers.

calculateCollision()
I denne metode beregnes det, om skibet er på kollisionskurs med eget skib.
Grundprincippet i algoritmen er, at lave en linie for hvert skib, og beregne om skibene mødes i disse liniers skæringspunkt. Dataflowet gennem metoden kan ses i appendiks F.1.1.

I det følgende gennemgås algoritmen trin for trin. For yderligere beskrivelse af algoritmen se appendiks C. I parenteserne er angivet de attributter, som er benyttet i kildekoden (se appendiks H). Se figur 6.6 for at følge forklaringen af udregningsmetoden.

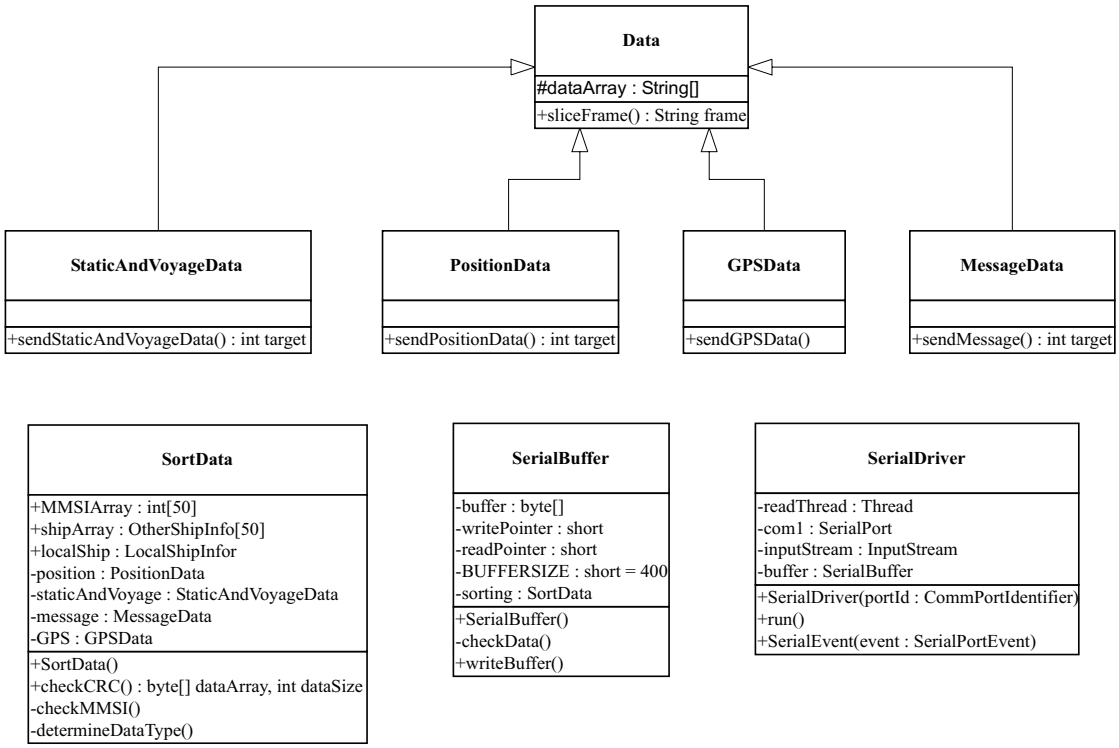
- Graderne i COG for begge skibe omregnes fra kompasgrader til grader der passer til det retvinklede koordinatsystem, som beregningerne foretages i. De omregnede grader svarer til 450 minus kompasgraderne.
- Koordinaterne for skibene omregnes fra minutter til grader (x1 , y1) og (x2 , y2).
- Informationerne i SOG og COG betragtes som en vektor, og slutpunktet for denne vektor beregnes (x1New, y1New og x2New, y2New).
- Skibene tænkes at bevæge sig langs hver sin linie, og der opstilles ligninger for disse linier. *(l1 og l2)*
- Skæringspunktet mellem linierne bestemmes. (xIntersect, yIntersect)
- Det undersøges, om skibene begge har kurs mod dette skæringspunkt. Hvis dette ikke er tilfældet, konkluderes det, at der ikke er kollisionsfare. Har de begge kurs mod skæringspunktet, fortsættes beregningerne.
- Skibenes afstand til skæringspunktet bestemmes. (ship1ToIntersect og ship2ToIntersect)
- Disse afstande divideres med skibenes respektive SOG, for at undersøge "hvornår" de når punktet. (rel1 og rel2)
- For at sammenligne om skibene er ved punktet på det samme tidspunkt, divideres rel1 med rel2, og hvis resultatet ligger tæt på 1, betragtes det som en kollisionsfare.
- Hvis der har været kollisionsfare, sættes boolean-variablen "collisionCourse" til "true".



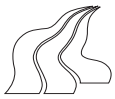
Figur 6.6 Punktet hvor to sejlende skibe krydser hinanden.

6.5 Detaljeret design af CI package

Denne pakke skal varetage kommunikationen mellem controlleren og SHIP package. Det vil altså sige, at den skal sørge for at modtage de informationer, som controlleren sender (VHF data og GPS data). Det skal undersøges om de modtagne data er fejlbehæftede vha. et CRC-check. Dette CRC-check skal laves efter CRC-ITU-T standarden. Endvidere skal de modtagne data undersøges for "afsenderadresse" og derefter fordeles til de rette skibe.
På baggrund af dette, er der opstillet et klassediagram, som vist på nedenstående figur. Klassediagrammet indeholder en række klasser, metoder og attributter. Det er muligt, at der under selve implementeringen af de enkelte klasser, vil forekomme flere metoder og attributter som benyttes internt i klassen. De ekterne grænseflader, som er de metoder og attributter, der har "public" eller "protected" synlighed (indikeret med hhv. + og #), må dog ikke ændres.



Figur 6.7 Klassediagram for CI package. Dataflowet gennem disse klasser er: SerialDriver, SerialBuffer, SortData og til sidst en af *Data klasserne. Alle *Data klassernes egenskaber nedarves fra klassen Data.



Det efterfølgende er en beskrivelse af, hvad de enkelte klasser skal udføre, hvorefter der vil følge et egentligt design og en implementering af de enkelte klasser.

SerialDriver
Denne klasse skal stå for modtagelsen af data fra den serielle port. Driveren skal konfigureres, som beskrevet i systemdesign (dvs. 38.400 baud, 8 data, 0 paritet og 1 stop). Envidere benyttes COM port 1. SerielDriveren skal være interruptstyret, hvilket vil sige, at der skal benyttes en "eventlistener" på serielporten.

SerialBuffer
Bufferen skal løbende kunne modtage data fra den serielle driver, og skal sørge for at sende data videre, når der er modtaget en hel frame. Der stilles ikke nogle videre krav til bufferens størrelse, bortset fra, at den skal kunne rumme en frame, idet den skal virke som et midlertidigt opholdssted for de indkomne data, indtil slutningen af framen er nået.

SortData
Denne klasses hovedformål er at opretholde og vedligeholde en "database" over de skibe, som sender data til systemet. Den skal sørge for at instantiere nye skibe, opdatere allerede instantierede skibe og fordele data til de rigtige skibsobjekter. Endvidere skal denne klasse undersøge, om de modtagne data er fejlbehæftede.

Data
Denne klasses egenskaber nedarves af alle de øvrige *data klasser. Klassen skal sørge for at opdele de modtagne dataframes i de enkelte datafelter. Denne nedarvning laves da disse metoder er fælles for alle *data klasserne.

***Data klasserne**
Disse klassers eneste formål er, at overføre den nu opdelte frame til de respektive pladser i det aktuelle skib.

6.5.1 SerialDriver

Serieldriveren er en enhed, der skal køre parallelt med afviklingen af det samlede program. Driveren programmeres derfor som en selvstændig tråd, hvor der er implementeret en eventlistener på serielporten. Tråden skal blot sættes til at "sove" af hovedprogrammet, og forblive i denne tilstand til den bliver "vækket" af en interrupt på serielporten.
Driveren er bygget op omkring tre metoder, som vist på figur 6.7. En constructor "SerialDriver()" der kaldes, når hovedprogrammet instantierer et objekt af SerialDriver. Denne constructor skal således initialisere serielporten til den rigtige hastighed og lignende. Den anden metode "run()" kaldes internt fra constructoren. Denne metode sørger for at starte en tråd, der kører parallelt med hovedprogrammet. Den sidste metode "serialEvent()" er den eventlistener, som køres, når der har været et interrupt på serielporten. Denne metode skal derfor undersøge, om det interrupt, den fik, var forårsaget af, at der var data i UART'ens Rx register. Hvis der var data i UART'en, hentes de ind i en lokal variabel, og denne variabel skrives over i den serielle buffer (Klassen SerialBuffer). Dataflowet gennem serieldriveren kan ses i appendiks F.2.1.

6.5.2 SerialBuffer

Bufferen består af to "public" metoder til initialisering og skrivning i bufferen, samt en "private" metode til at undersøge, hvorvidt der kommet en frame. Da data videresendes så snart der er modtaget en hel frame, er bufferen blot en lineær buffer, hvor writePointer nulstilles så snart en frame er modtaget. Hvorvidt der er modtaget en frame eller ej, afgøres af den private metode checkData(). Denne metode kaldes hver gang der skrives en byte i bufferen. Metoden undersøger,

om den modtagne byte er et <LF> og om den forrige byte var et <CR> (Da en frame afsluttes med disse tegn). Hvis dette er tilfældet, overføres data til et midlertidigt bytearray som sendes til klassen SortData. Dataflowet gennem bufferen kan ses i appendiks F.2.2.

6.5.3 SortData

Denne klasse administrerer de skibsobjekter, der eksisterer i programmet. Der oprettes nye skibe, og der slettes skibe fra arrayet, når de ikke længere ligger inden for den kritiske grænse på 3 sømil. Ligeledes undersøges det, hvilken form for data, der er modtaget, og dette sendes forskellige steder hen alt efter resultatet. Denne klasse udgøres af en række attributter og metoder, hvoraf de væsentligste beskrives nedenfor. Dataflowet gennem bufferen kan ses i appendiks F.2.3.

Attributter:

MMSIArray[50]: Hver gang der kommer en ny frame fra VHF-radioen, undersøges det, om skibet findes i dette array (Der refereres til skibets MMSI- nummer). Hvis det ikke eksisterer, tilføjes det til arrayet. På den måde kan man holde styr på, om der skal oprettes et nyt skibsobjekt, eller der blot er tale om en opdatering af et af de eksisterende skibsobjekter. Arrayet er på 50 pladser, da det er valgt at der maksimalt kan overvåges 50 skibe samtidigt. Dette er valgt da det vurderes at der maksimalt vil være 50 skibe indenfor det kritiske område.

shipArray[50]: Et array af skibsobjekter (OtherShipInfo). Dette array indeholder alle de instantierede skibsobjekter.

localShip: Der oprettes et objekt af eget skib (LocalShipInfo), hvor data fra GPS-modtageren gemmes.

SortData():

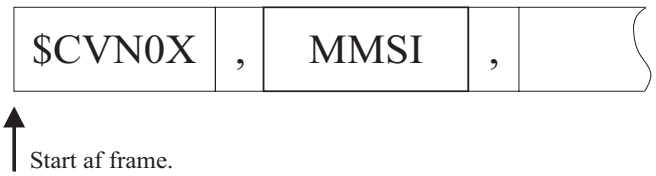
Denne metode er en constructor, som initialiserer MMSIArray, så der står "-1" på alle pladser. På den måde kan der senere søges i arrayet efter en specifik værdi (MMSI-nummer).

checkCRC():

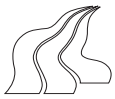
Metoden undersøger den modtagne frame for fejl. Dette gøres ved at beregne residuet på den modtagne frame vha. CRC-ITU-T (minus $ og * som er start og stopflag). Hvis den beregnede værdi ikke giver 0x1D0F (residuet) betyder det, at den modtagne frame er fejlbehæftet, og framen bliver ignoreret. Er data derimod iorden kaldes, metoden checkMMSI(). Flowdiagrammet for denne metode kan ses i appendiks F.2.3.

checkMMSI():

Her skal undersøges for MMSI-nummeret i den modtagne frame. Det skal undersøges, hvorvidt der eksisterer et skib i systemet med det modtagne MMSI-nummer. Dette gøres på baggrund af et kendskab til den protokol, som benyttes mellem controlleren og PC'en. Et udsnit af en frame er vist på nedenstående figur.



Figur 6.8 Et udsnit af en VHF dataframe, som sendes til PC'en. MMSI nummeret står altid på plads 2.



Det ses på figur 6.8, at MMSI nummeret står på plads 2 i framen, adskilt med kommaer til de øvrige datafelter. MMSI-nummeret uddrages fra dataframen ved at søge efter de to første kommaer, og tage det mellemliggende data. Da der ikke er taget højde for, at der kan forekomme kommaer i datafelterne, undersøges der ikke for "escape sequences". Der må med andre ord ikke forekomme et komma i datafelterne. Dette kunne der have været kompenseret for før, hvis Controlleren indsatte en "^" før hvert komma i datafelterne. Tegnet "^" er det der foreskrives af NMEA standarden.
Hvis det udtagne MMSI-nummer ikke eksisterer i systemet, oprettes et nyt skibsobjekt på den første ledige plads i shipArray[]. Derefter kaldes metoden determineDataType().
Flowdiagrammet for metoden checkMMSI() kan ses i appendiks F.2.3.

determineDataType():
Denne metode benyttes til at afgøre, hvilken type data der er modtaget. Dette er væsentligt, da en dataframe skal behandles forskelligt, alt efter hvilken type den er. Der benyttes, som beskrevet under "Controller Software", en protokol, som består af fire forskellige typer beskeder. Disse er:

- En positionsbesked fra VHF-radioen.
- En rejserelateret besked fra VHF-radioen.
- En tekstbesked fra VHF-radioen.
- En positionsbesked fra GPS-modtageren.

Det skal altså blot undersøges, hvilken af disse beskeder, der er modtaget. Dette gøres med en simpel "Switch" sætning.
Flowdiagrammet for metoden determineDataType() kan ses i appendiks F.2.3.

6.5.4 Data

Denne klasses egenskaber nedarves af alle de øvrige Dataklasser (PositionData, MessageData osv.). Grunden til, at denne klasse eksisterer, er, at samtlige Dataklasser har brug for at bryde den modtagne dataframe op i de enkelte datafelter, før de kan sendes over til skibsobjektet. Denne klasse skal altså dele den modtagne frame ind i selvstændige datafelter på baggrund af de kommaer, der adskiller datafelterne. Dette gøres af metoden sliceFrame().

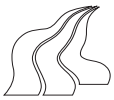
6.5.5 *Data

Disse klasser indeholder en metode (send*Data()), der lægger de enkelte datafelter ind på deres respektive pladser i det enkelte skibsobjekt. *Data klasserne kaldes med et "targetindex" der angiver, hvilket skibsobjekt der skal opdateres.

6.6 Detaljeret design af UI package

"User Interface" udgør aktørens brugerflade til PC-programmet. Disse klasser skal indeholde de fornødne metoder til at oprette en grafisk brugerflade, således at brugeren af systemet (aktøren) kan kommunikere med systemet. Dette betyder at der skal være nogle klasser, som kan oprette følgende:

- Et hovedvindue, hvor dynamiske informationer om eget skib, samt et, af brugeren, valgt skib vises. I dette vindue skal endvidere være et felt, hvor de forskellige skibe, indenfor en radius af 3 sømil, visualiseres med hensyn til hastighed, position og retning. Dialogboksene, hvor statiske og rejserelaterede informationer for andre skibe kan aflæses og egne informationer bestemmes, skal kunne aktiveres med knapper. Programmet afsluttes ved at trykke på krydset øverste i højre hjørne.
- En dialogboks hvori eget skibs statiske informationer fastsættes. Det vil sige indtastning eller på anden måde valg af: "Call sign", "Ship name", "MMSI number", "IMO number",



"Position accuracy" samt "Postion of GNSS antenna"
- En tilsvarende dialogboks, hvori et andet valgt skibs statiske informationer kan aflæses.
- En dialogboks, hvori eget skibs rejserelaterede informationer fastsættes. Dvs. indtastning / valg af: "Type of ship", "Special conditions", "Type of nav. sensor", "Destination", "Actual draught" samt "Expected time of arrival" (ETA).
- En dialogboks, hvor samme rejserelaterede informationer kan aflæses om et andet, valgt skib.
- En dialogboks, som kommer frem, hvis en teksbesked modtages fra et af de andre skibe.

Det ses, at det er det første krav, som er mest omfattende, og at de øvrige krav til dialogboksene for henholdsvis statiske informationer og rejserelaterede informationer om eget og andet skib har mange fælles træk. I det følgende beskrives, hvilke klasser, der bruges til realisering af de ovenstående punkter.

Selve hovedvinduet realiseres med 3 klasser:
En klasse til afslutning af programmet, hvis der trykkes på krydset i øverste højre hjørne, en klasse, som skal optegne selve vinduet og en klasse til optegning af vektorerne. Disse klasser får følgende navne:

- GenericWindowListener.
- MainWindow.
- Vectorfield.

De forskellige dialogbokse opbygges alle på samme skabelon, hvad angår størrelse og udseende, og derfor laves en klasse, som optegner disse fælles træk. Denne klasse får navnet:

- InfoDialog.

Til visning af statiske informationer, er der brug for to forskellige dialogbokse (en til eget skib, og en til andet skib), men med stort set samme udseende.
Derfor vælges der at oprette en klasse, som optegner de fælles træk for dialogboksene for statiske informationer, og dertil tilføje to klasser; en for eget skib, hvor informationerne kan modificeres, og en for andet skib, hvor data ikke kan modificeres.
De dialogbokse, som skal indeholde statiske informationer om skibe, implementeres med klasserne med følgende navne:

- StaticDialog.
- LocalStaticDialog.
- OtherStaticDialog.

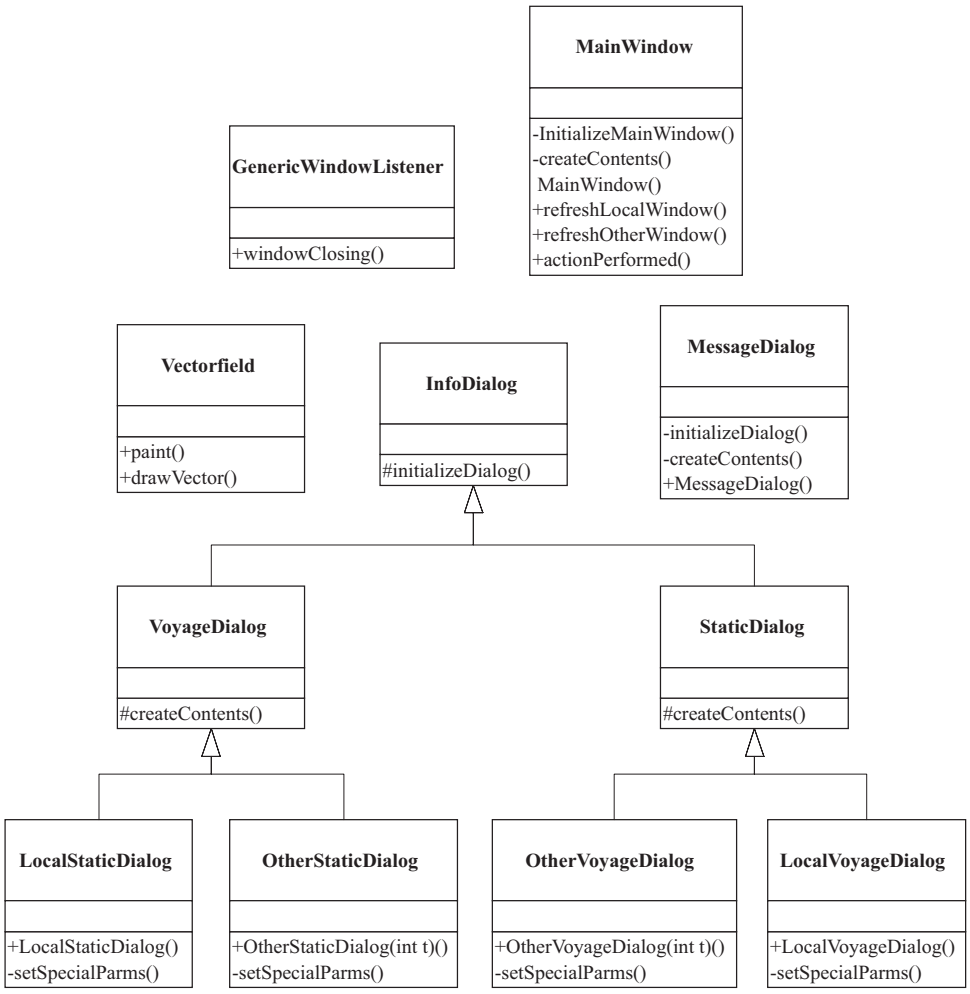
Ligesom dialogboksene for statiske informationer, er der i de rejserelaterede, behov for en dialogboks med samme udseende for både eget og andet skib. Der skal ligeledes kunne redigeres i informationerne om eget skib. Dette implementeres på samme måde som dialogboksene for statiske informationer. En klasse optegner "grundskærmen" for dialogboksen til rejserelaterede informationer, og de to andre specificerer, om det er for eget eller andet skib. Klasserne får følgende navne:

- VoyageDialog
- LocalVoyageDialog
- OtherVoyageDialog

Den sidste klasse i UI package er den, som skal varetage tekstbeskeder, som skal komme op på skærmen, når de videresendes fra CI. Denne funktion varetages af en klasse med følgende navn:

- MessageDialog

På figur 6.9 er de forskellige klassers indbyrdes forhold i pakken illustreret i henhold til UML standarden.



Figur 6.9 De forskellige klasser i UI package, og deres indbyrdes forhold.

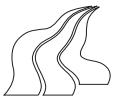
I det følgende vil de forskellige klasser blive beskrevet. Selve implementeringen er ikke dokumenteret her. I stedet henvises til kildekoden til de forskellige klasser, hvor klassernes opbygning, de forskellige, anvendte metoder, attributter og udtryk er beskrevet. Der er inkluderet flowdiagrammer til de væsentligste metoder / klasser i appendiks F. De klasser, der står i parentes efter klassens navn, er klasser, som ligger indeni klassen.

6.6.1 GenericWindowListener

Dette modul skal varetage lukning af programmet. I MainWindow findes en actionlistener, som aktiveres, hver gang brugeren foretager sig noget med knapper. GenericWindowListener aktiveres når der trykkes på krydset i øverste højre hjørne, og lukker derefter programmet.

6.6.2 MainWindow (MainListener)

Dette er den største klasse i denne pakke. I denne klasse optegnes hovedskærmen, med vektorfelt, knapper og tekst. Selve layoutet designes ved brug af paneler, der indeholder objekter, som f.eks



knapper eller tekstfelter. Disse paneler placeres indeni andre paneler. Placeringen af disse paneler, i forhold til hinanden varetages af forskellige layout metoder, som findes i javax.Swing klassen. MainWindow designes således, at de ting der skal opdateres løbende, såsom dynamiske informationer, vectorfield og lignende placeres i selvstændig paneler. Det vil sige at tekstfelterne og Vectorfield opdateres, hver gang der kommer nye dynamiske informationer. Tekstfeltet, for andre skibe, opdateres yderligere hver gang der skiftes mellem forskellige skibe. Tekstfeltet for eget skib opdateres ikke, når der kommer informationer om andre skibe.
Der implementeres en actionlistener i MainWindow, til at kalde metoder i andre klasser, når der trykkes på knapperne. Actionlistener implementeres som en klasse indeni MainWindow, og får navnet MainListener. I appendiks F.3.4 ses et flowdiagram for, hvordan modulet afvikles.

6.6.3 Vectorfield

Denne klasse indeholder en metode, som optegner den blå cirkel, hvor vektorerne er placeres indenfor, samt en metode, der beregner vektorernes placering på skærmen. Den først omtalte metode kaldes fra MainWindow, mens den anden metode, drawVector() kaldes hver gang et af skibenes koordinater, opdateres.
Metoden drawVector() optegner alle skibe med sort indenfor denne radius. Hvis der er risiko for, at et skib kolliderer med eget skib, optegnes det med rødt. Informationen om kollisionsrisiko for de forskellige skibe, fås fra metoden detectcollision fra modulet OtherShipInfo i SHIP package. I appendiks F.3.3, ses dataflowet gennem metoden drawVector().

6.6.4 InfoDialog

I denne klasse ligger en metode, som optegner dialogboksens størrelse, dens placering på skærmen samt sørger for, at dens størrelse er fastholdt. Denne klasses egenskaber nedarves af, VoyageDialog og StaticDialog.

6.6.5 VoyageDialog

I denne klasse optegnes en dialogboks for rejserelateret information. Dens tekstlabels, tekstfields og combo-boxe optegnes. Denne klasses egenskaber nedarves til de to nedenstående dialogbokse.

6.6.6 LocalVoyageDialog (LocalVoyageListener)

Denne klasse muliggør indtastning af information, som er rejserelateret. Der oprettes to knapper: en “OK”-knap og en “Cancel”-knap. Klassen udformes således, at de indtastede / valgte informationer gemmes, hvis man trykker på “OK”-knappen, og ignoreres, hvis man trykker på “Cancel”-knappen. Dialogboksen lukkes også, hvis der trykkes på en af disse knapper. Der implementeres en actionlistener ved navn LocalVoyageListener, som gemmer de indtastede / valgte data i arrayet localShip eller afslutter dialogboksen uden at gemme, afhængigt af hvilken knap der er trykket på. I appendiks F.3.1, ses et flowdiagram for denne klasse.

6.6.7 OtherVoyageDialog (OtherVoyageListener)

Denne dialogboks nedarver egenskaber fra klassen VoyageDialog og optegner selv en knap, hvor der står “Exit”. Dernæst opdateres indholdet af dialogboksen, ved at hente informationer fra klassen, SortData i CI package. I appendiks F.3.2, ses et flowdiagram for denne klasse.

6.6.8 StaticDialog

Denne klasse har en tilsvarende funktion som VoyageDialog. Den optegner dialogboksen for statiske informationer.

6.6.9 LocalStaticDialog (LocalStaticListener)

Dette modul muliggør indtastning af statisk information om eget skib. Der oprettes to knapper: en "OK"-knap og en "Cancel"-knap. Klassen udformes ligesom LocalVoyageDialog, således at data gemmes, hvis der trykkes på "OK" og ignoreres hvis der trykkes på "Cancel". Der implementeres en actionlistener ved navn LocalStaticListener, som gemmer de indtastede / valgte data i arrayet localShip, hvis der er trykket på knappen "OK". I appendiks F.3.1 ses et flowdiagram for denne klasse.

6.6.10 OtherStaticDialog (OtherStaticListener)

Denne klasse implementers på samme måde som OtherVoyageDialog. Det vil sige, at klassen nedarver egenskaberne fra LocalStaticDialog og ændrer "OK"- og "Cancel"-knappen til en "EXIT"-knap. Indholdet af dialogboksen hentes fra et skibsobjekt. I appendiks F.3.2, ses et flowdiagram for denne klasse.

6.6.11 MessageDialog

Denne klasse består af en metode, som optegner en dialogboks med et tekstfelt og en knap. Indholdet af tekstfeltet hentes fra MessageData i Ship-package, som instantierer en MessageDialog, når der kommer en tekstbesked. Derved kommer der en dialogboks op på skærmen, når der modtages en besked. I appendiks F.3.5, ses et flowdiagram for denne klasse.

6.7 Use case test

Til test af PC-programmet er der taget udgangspunkt i de opstillede "use cases". Derudover opstilles der en række testscenarier, som kontrollerer, om systemet virker efter hensigten. For at kunne teste systemet, er der lavet et testprogram til en PC, som simulerer en GPS-modtager og en VHF-radio. Dette testprogram sender frames til JAVA-programmet i henhold til AIS- og NMEA-standarden. På den måde kontrolleres det, at programmet er i stand til at modtage de ønskede frames.

6.7.1 Use cases for Controller

Her kontrolleres om de data, der sendes fra testprogrammet modtages af systemet, og visualiseres korrekt på skærmen. De enkelte use cases fra figur 6.1 gennemgås nedenfor.

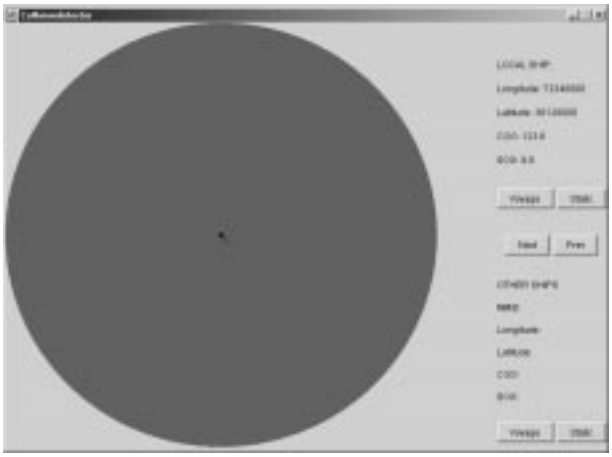
- Levere GPS-data med egen position m.m:

Der afsendes en frame fra testprogrammet, som indeholder information om egen position m.m.:

$GPRMC,174818.03,6012.0000,N,12034.0000,E,9.9,123.6,201100*<CRC>

De understregede data er dem, som skal blive visualiseret på skærmen og skal have følgende værdier:

Input / Output		
Data	Input	Ønsket output
Latitude	60°12'00" N	36120000 min·10^{-4}
Longitude	120°34'00" E	72340000 min·10^{-4}
SOG	9.9 knots	9.9 knots
COG	123.6°	123.6°



Figur 6.10 "Screen shot" af hovedvindue, når der er kommet en GPS frame

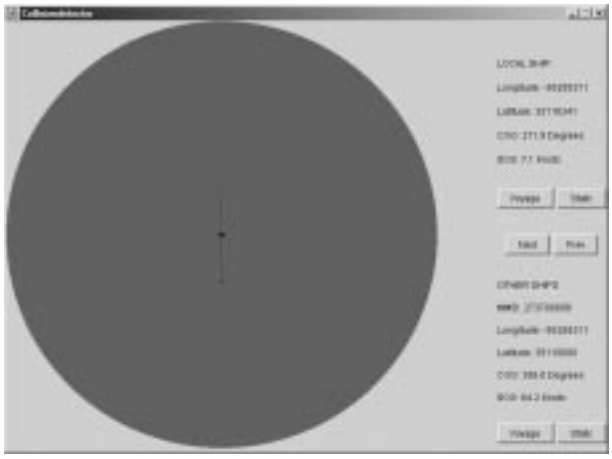
Det ses på figur 6.10 at brugeren kan aflæse longitude, latitude, COG og SOG i hovedvinduet, og at de stemmer overens med de tidligere beskrevne værdier.

- Levere positionsrapport fra andet skib:
Der afsendes en frame fra testprogrammet, som indeholder information om et andet skibs position m.m.:

$CVN01,273700808,0,-96,642,1,-95285311,35110000,3580,20

De understregede data er dem, som skal blive visualiseret på skærmen og skal have følgende værdier:

Input / Output		
Data	Input	Ønsket output
MMSI nr.	273700808	273700808
SOG	64.2 knots	64.2 knots
Latitude	-95285311 min·10^{-4}	-95285311 min·10^{-4}
Longitude	35110000 min·10^{-4}	35110000 min·10^{-4}
COG	358.0°	358.0°



Figur 6.11 "Screen shot" af hovedvindue, når der er kommet en VHF frame.

Det ses på figur 6.11 at brugeren kan aflæse MMSI, longitude, latitude, COG og SOG i hovedvinduet, og at de stemmer overens med de tidligere beskrevne værdier.

- Levere statiske og rejserelaterede informationer fra et andet skib:
Der afsendes en frame fra testprogrammet, som indeholder statisk og rejserelateret information om et andet skib:

$CVN05,123456789,9999,ABC,LARS,63,20,30,40,50,5,12312359,20,ASDF

De understregede data er dem, som skal blive visualiseret på skærmen, og de skal have følgende værdier:

Input / Output		
Data	Input	Ønsket output
MMSI nr.	123456789	123456789
IMO nr.	9999	9999
Call sign	ABC	ABC
Name	LARS	LARS
Type of ship	63	63
Postion of GNSS	20,30,40,50	20,30,40,50
Type of nav. sensor	5	5
Expected time of arrival	12312359	12312359
Actula draught	20	20
Destination	ASDF	ASDF

Figur 6.12 "Screen shot" af statiske og rejserelaterede informationer for et andet skib.

Det ses på figur 6.12, at brugeren kan aflæse MMSI nr., IMO nr., Call sign, Name, Type of ship, Postion of GNSS, Type of nav. sensor, Expected time of arrival, Actual draught og Destination i hovedvinduet, og at de stemmer overens med de tidligere beskrevne værdier.

- Levere tekstbesked fra andet skib:
Der afsendes en frame fra testprogrammet, som indeholder en tekstbesked fra et andet skib:

$CVN08,1234,Dette er en test dette er en test.

De understregede data er dem, som skal blive visualiseret på skærmen og de skal have følgende værdier:

Input / Output		
Data	Input	Ønsket output
MMSI nr.	1234	1234
Besked	Dette er en test dette er en test	Dette er en test dette er en test

Figur 6.13 "Screen shot" af dialogboks med besked.

På figur 6.13 ses det, at der kommer en dialogboks op på skærmen, når PC-programmet modtager en besked fra controlleren. MMSI-nummer vises i den øverste bjælke, mens beskeden vises i tekstfeltet.

6.7.2 Use cases for bruger

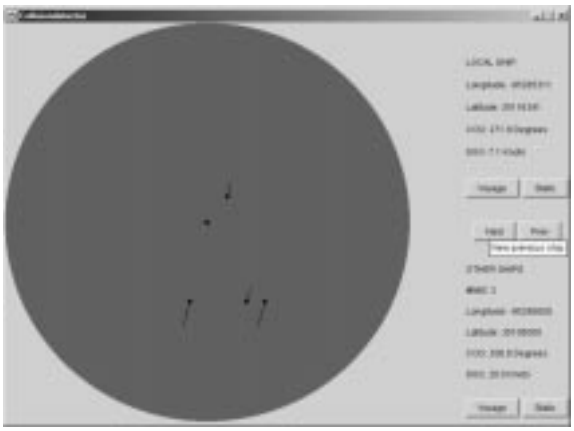
Det kontrolleres her, om de enkelte "use cases" er opfyldt. For at teste dette, kontrolleres og testes den grafiske brugerflade for hver enkelt "use case".

- Få vist dialogboks med besked fra andet skib:
Se leveret tekstbesked fra andet skib.

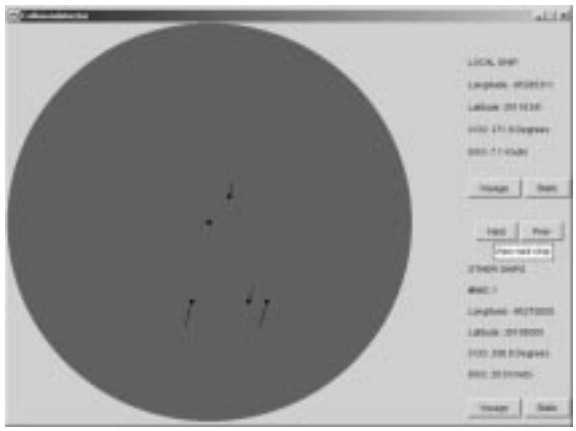
- Se statiske informationer om det valgte skib:
Se leverede statiske informationer om et andet skib.

- Se rejserelaterede informationer om det valgte skib:
Se leverede rejserelaterede informationer om et andet skib.

- Vælge næste skib til visning / vælge forrige skib til visning:
Til denne test sendes der positions- og rejserelaterede informationer fra 4 skibe, der er inden for tre sømil i forhold til eget skib. Derefter skiftes der til næste skib med "Next" knappen og der trykkes på "Prev" knappen for at se det forrige skib.



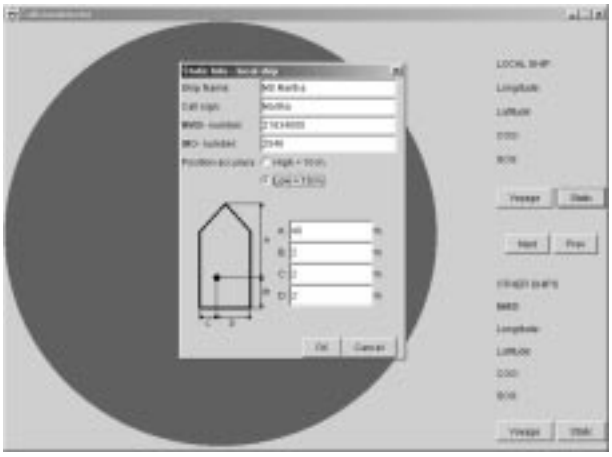
Figur 6.14 "Screen shot" af tryk på Prev.



Figur 6.15 "Screen shot" af tryk på Next.

Det ses på figur 6.14 og 6.15 at de dynamiske informationer skifter når der trykkes på "Next" eller "Prev" knappen. De viste informationer stemmer endvidere overens med de sendte data.

- Redigere / Se statiske informationer om eget skib:
For at teste denne "use case", er der trykket på "Static" knappen, indskrevet data og trykket på "OK". Derefter er der trykket på "Static" knappen igen, for at undersøge om de rigtige oplysninger er blevet gemt.



Figur 6.16 "Screen shot" af statisk information for eget skib.

På figur 6.16 ses det, at de rigtige oplysninger er blevet gemt og kan vises igen.

- Redigere / Se rejserelaterede informationer om eget skib:
Til denne test er der trykket på "Voyage" knappen for eget skib, indskrevet data og trykket på "OK". Derefter er der trykket på "Voyage" igen, for at kontrollere, om de rigtige oplysninger er blevet gemt.

Figur 6.17 "Screen shot" af rejserelateret data for eget skib.

På figur 6.17 ses det, at de indtastede oplysninger er blevet gemt og vises korrekt på skærmen.

- Afslutte programmet:
Testen af denne "use case" er udført ved at trykke på krydset i øverste højre hjørne af hovedvinduet. Programmet lukkes ned som forventet.

6.7.3 Afsendelse af ugyldige data:

For at kontrollere om systemet kan hånterere data, som ikke overholder, den i systemdesignet, definerede protokol, sendes der nogle ugyldige frames til PC'en.
Der afsendes følgende frames fra controlleren:

- En frame med ugyldig checksum.
- En frame med et forkert format.

Ingen af de afsendte frames blev vist, og systemet ignorerede de modtagne data.
Idet programmet kan modtage de definerede frames, behandle dem og vise data i de korrekte felter, må det konkluderes at programmet virker efter hensigten.

7 Accepttest og konklusion

Dette kapitel indeholder en accepttest af det samlede system, en diskussion af, hvilke ændringer / forbedringer, der kunne laves, og hvordan disse kunne realiseres. Til sidst er der en konklusion, hvor de, i kravsspecifikationen opstillede krav og forventninger til systemet, betragtes. Her vurderes, hvorvidt disse krav er opfyldt, og om projektforløbet har indfriet gruppens forventninger til dette.

7.1 Accepttest

I dette afsnit gennemgås de enkelte dele for at undersøge, om de opfylder de krav, som er fastsat i kravsspecifikationen. De følgende underpunkter er de krav, der blev stillet i kravsspecifikationen, hvortil der er knyttet kommentarer om, hvorvidt de er opfyldt.

7.1.1 Afsendelse og modtagelse af data fra VHF-radioen

Ifølge kravsspecifikationen skal det være muligt både at afsende og modtage data fra VHF-radioen. Det er dog senere blevet vedtaget, at der ikke skal afsendes data. Det skal være muligt at modtage dynamiske, rejserelaterede og statiske informationer fra VHF-radioen. Det er ved hjælp af et testprogram testet, om det samlede system kan modtage de ønskede informationer. Testen viste, at det var muligt at modtage alle de ønskede informationer.

7.1.2 Modtagelse af data fra GPS-modtageren

Da der ikke er benyttet en rigtig GPS-modtager, men istedet en PC, som simulerer en GPS-modtager, er dette ikke endeligt testet. Det kan dog nævnes at testprogrammet på PC'en, som simulerer en GPS-modtager, er udviklet i overensstemmelse med NMEA 0183 standarden, hvorfor systemet også burde virke sammen med en rigtig GPS-modtager. De afsendte data fra testprogrammet blev korrekt modtaget af systemet, hvorfor det konkluderes, at kravet til modtagelse af data fra en GPS-modtager, er opfyldt.

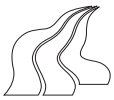
7.1.3 Kollisionsdektion

Kravene til kollisionsdektionen er som følger:

1 Der skal være realtime detektion, således at der ikke er mærkbar forsinkelse i systemet.
2 Der skal være både visuel og auditiv indikation, ved risiko for kollision.
3 Der skal være mulighed for frakobling af auditiv indikation.
4 Detektionen skal foregå indenfor en radius af 3 sømil.

Systemet opfylder kravene således:

1 Det vurderes, at systemet kan detekte i realtime, da der ikke er mærkbar forsinkelse i systemet.
2 Det er kun den visuelle indikation, der er opfyldt. Det skib, som er på kollisionskurs med eget skib, indikeres på skærmen med en rød pil. Den auditive alarm er ikke blevet implementeret.
3 Da der ikke er realiseret en auditiv indikation ved risiko for kollision, er der heller ikke realiseret en frakobling af denne.



4 Hvis et skib kommer inden for en radius af 3 sømil og skibet ligger på kollisionskurs med eget skib, vil systemet detektere det. På baggrund heraf konkluderes det, at kravet er opfyldt. Der er dog visse forbehold, som bliver nærmere beskrevet i diskussionen.

7.1.4 Indtastning af skibsinformation

Det er muligt, at indtaste, de i AIS standarden definerede, skibsinformationer om eget skib. Disse oplysninger bliver gemt i objektet "LocalShipInfo", men sendes ikke til controlleren, da det i afgrænsningen blev besluttet, at systemet kun skulle *modtage* data.

7.2 Diskussion

I dette afsnit diskuteres eventuelle forbedringer og ændringer, der kunne laves i systemet, for at forbedre dets ydeevne og funktionalitet.
Afsnittet opdeles i tre dele; een for hver proces.

7.2.1 Microcontroller Hardware

Her nævnes de forbedringer og ændringer, som kunne foretages på den konstruerede hardware.

Ekstra microcontroller til modtagelse fra VHF-radio
Da der kan forekomme en konstant bitstrøm fra VHF-radioen, er det en tidskrævende proces at synkronisere og modtage data. Derfor kunne det være fordelagtigt med en dedikeret microcontroller, som står for modtagelse af data fra VHF-radioen. Disse data kunne overføres til en master, som håndterer de øvre lag i AIS protokollen. Derved ville systemet også lettere kunne udvides til at sende.

Indstikskort
For at undgå at sende data over en RS232C forbindelse, som kræver datatilpasning til en ny protokol, kunne man istedet lave et indstikskort til en PC, således at modtagne data sendes direkte til PC'en via dens ISA- eller PCI-bus. På den måde kan alle beregningerne foretages på PC'en, som har flere beregningsresourcer til rådighed.

CRC-check med gates
Det er meget almindeligt at foretage et CRC-check ved brug af hardware. Dette kunne eksempelvis implementeres med nogle få XOR gates samt nogle FlipFlops. Der findes også en række integrerede kredsløb, som kan foretage dette. Der blev istedet lavet et softwaremæssigt CRC-check, som bruger ressourcer fra microcontrolleren. Dette kunne med fordel ændres, hvis der var tidsmæssige problemer. Softwareløsningen er til gengæld billigere.

7.2.2 Microcontroller Software

Herunder er en diskussion af de valg og metoder, som er benyttet i softwaren til controlleren.

Problem med læsning fra porte
Som tidligere nævnt var der en række problemer med at synkronisere og modtage data fra VHF-radioen. Dette var bla. på grund af den benyttede microcontroller, hvis timere ikke var nøjagtige nok. Hvis dette skulle løses, kunne der enten benyttes en anden microcontroller, eller der kunne laves et eksternt timerkredsløb. En anden mulighed, som blev afprøvet, var at lave en softwaremæssig løsning, hvor der resynkroniseres hver gang, der er modtaget 20 bit. Det er dog ikke nogen god løsning, at rette hardwaremæssige problemer ved hjælp af software.

Overholdelse af AIS protokol
AIS-protokollen foreskriver, at alle skibe skal få oplyst deres egen position, hastighed, UTC-tid og kurs fra en GPS-modtager, modtage informationer om andre skibe på VHF-radioen, og udsende tilsvarende informationer om eget skib. Hvis det udviklede system også skulle udsende informationer, skulle der foretages en del ændringer. For det første skal det overholde de foreskrevne regler om kun at sende i forskellige timeslots. Desuden skal UTC-tiden tages med i betragtningerne, da der er regler for, hvornår disse timeslots starter i forhold til UTC minuttet. Der skulle endvidere laves en modtagerenhed, som kan varetage de informationer, der kommer fra PC om eget skib. Disse informationer skal inddeles i frames og udsendes til VHF-radioen.

Assembler kode
Hvis programmet skulle optimeres, ville det være en god idé, at skrive dele af det i assembler. Da dette er mere hardwarenært end C, ville det formentlig reducere mængden af kode og forøge eksekveringshastigheden. Til gengæld ville det blive mere omfattende og tidskrævende at skrive koden. [Tanenbaum, 1999, side 487]

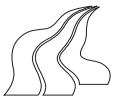
7.2.3 PC-program

Forbedringer af brugerflade
Hvis der var mulighed for at se, hvilke skibe, der hører sammen med hvilke informationer, ville informationerne være mere anvendelige. Dette kunne eventuelt realiseres ved at den vektor, hvis informationer vises på skærmen, blev markeret med en anden farve.
Endvidere ville det være en fordel, hvis der var en zoom funktion, således at brugeren selv kunne bestemme detaljeringsgraden af billedet.
Disse funktioner kunne forholdsvis nemt implementeres på baggrund af den nuværende kodestruktur.

Kollisionsberegningsalgoritme
Den benyttede kollisionsalgoritme benytter sig af retvinklede koordinater, hvilket vil medføre en vis usikkerhed i forhold til de virkelige omstændigheder. Til gengæld vil en hensyntagen til disse forhold medføre en væsentlig større kompleksitet af kollisionsberegningen. Det vurderes, at denne usikkerhed er negligerbar i forhold til de øvrige usikkerheder, der forekommer i systemet. (usikkerhed på kurs, fart, position osv.)

Database over overvågede skibe
I PC-programmet er der oprettet en database over de overvågede skibe ved at benytte et array. Problemet ved et array er, at det skal have en fast størrelse, hvorfor der optages en fast mængde hukommelse uanset, hvor meget der ligger i arrayet. I stedet kunne der benyttes linkede lister, hvorved dette problem ville være løst.

Overholdelse af AIS-protokol
AIS-protokollen stiller ingen krav til brugerfladen, men fastsætter at man blandt andet skal udsende eget skibs statiske og dynamiske informationer. I dette henseende er PC-programmet forberedt for udvidelse, idet det er muligt at gemme de informationer, som er indeholdt i frame 1, 2, 3 og 5. Systemet er dog ikke forberedt til at håndtere de øvrige frames, som er specificeret i AIS standarden.



7.3 Konklusion

Det kan på baggrund af den foregående accepttest og diskussion konkluderes, at de stillede krav stort set er opfyldt. Der er dog nogle enkelte punkter, som ikke er opfyldt til fulde. Blandt de væsentligste kritikpunkter er synkroniseringen med den bitstrøm, som kommer fra VHF-radioen. Til dette, må det konkluderes, at der under dette projektforløb, ikke blev fundet en brugbar løsning. En af grundene til dette var, at der blev foretaget en prioritering. Dette resulterede i en endelig beslutning om, at der ikke skulle bruges mere tid på at finde en løsning på dette problem, men istedet lægges fokus på de øvrige emner i rapporten. Der er dog stillet en række forslag til, hvordan problemet kunne løses i den forgående diskussion.
Der er i rapporten benyttet en, for os, ny udviklings- og design model: UML, som er en standardiseret objektorienteret designmodel. Dette har medført en række problemer, idet denne designmodel ligger langt fra den strukturerede designmodel (SPU), som vi har været vant til fra forrige semestre. Til gengæld, har vi gennem arbejdet med denne model, fået en række værktøjer, som kan benyttes ved fremtidige objektorienterede design.
Det vurderes, at det samlede system som en helhed, vil kunne have en interesse for mindre fartøjer, der ikke bliver underlagt kravet om, at udsende de, i AIS-standarden, nævnte informationer. De vil kunne benytte systemet, som en ekstra sikkerhed for ikke at sejle sammen med andre større fartøjer. Dette forudsætter, at der anvendes en VHF-radio med de, i rapporten, definerede funktioner.

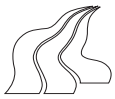
Der følger her en gennemgang af de projektenhedskurser projektgruppen har fulgt, og hvordan de er blevet inddraget i projektet.
Projektgruppen har på dette semester fulgt følgende PE kurser: Objektorienteret analyse og design, Objektorienteret programmering med JAVA, Elektromagnetisk forligelighed, sandtidssystemer, multiprogrammering samt netværk og datakommunikation.
De fulgte PE kurser har været dækket af projektarbejdet på følgende måde:

Objektorienteret analyse og design
Dette er blevet benyttet i vid udstrækning under design af PC programmet, da det er skrevet i JAVA.

Objektorienteret programmering med JAVA
Dette kursus er benyttet under udviklingen af PC programmet, som er programmeret udelukkende i JAVA.

Sandtidssystemer og multiprogrammering
Disse kurser har ikke været benyttet lige så direkte som de to forrige. Da det konstruerede AIS system er et realtime system, er der dog paralleller mellem kurset sandtidssystemer og systemet. Multiprogrammeringkurset har lettet forståelsen af de "actionListeners" som er benyttet i JAVA programmet. De forskellige GUI komponenter er selvstændige programtråde, som kan påvirkes af eksterne stimuli (eksempelvis tastetryk). Endvidere er den serielle driver programmeret, som en selvstændig tråd.
Der er også blevet foretaget nogle overvejelser med hensyn til multiprogrammering af softwaren på controlleren. Der blev dog valgt at benytte almindelig interruptstyring, da vi i det pågældende tilfælde mente, at dette ville være den bedste løsning.

Netværk og datakommunikation
Kurset har haft direkte tilknytning til dette projekt, da der er blevet arbejdet med en AIS standard / protokol. Endvidere er der blevet lavet en ny protokol, som benytter sig af de gængse protokolregler, der er blevet undervist i gennem kurset.



Elektromagnetisk forligelighed
De grundregler for printudlæg, der er blevet undervist i gennem kurset, er i en vis udstrækning blevet benyttet. Der er eksempelvis benyttet et hulprint med stelplan, afkoblingskondensatorer og lignende til hardwaren. Da der ikke er lavet et egentligt printudlæg, er dette ikke dokumenteret i rapporten.

Det konkluderes at de fulgte kurser er faldet inden for rammerne af dette projekt og projektgruppens arbejde.

8 Litteraturliste

[Forkortelse, År, (Side)] Titel, udgivelsesår, forfatter navn, forlag, udgave nr., ISBN

[AIS, 1998] Technical characteristics for a universal shipborne automatic identification system using time division multiple access in the VHF maritime mobile band, 1998, *Ikke oplyst*, ITU, M.1371, *Ikke oplyst*.

[Cassidy, 1998] Standard For Interfacing Marine Electronic Devices, 1998, Frank Cassidy, National Marine Electronics Association, 2.30, *Ikke oplyst*.

[Duffy, 1999] Duelighedbogen, Jørn Duffy Villumsen & Jørgen Marcussen, Iver Weilbach & CO. A/S, 4. udgave 1. Oplæg, 87-7790-100-2

[Ebert, 1996] Grundlæggende teori for fejlkontrol kodning, Hans Ebert, AUC, 4. udgave.

[Eriksson, 1998] UML Toolkit, 1998, Hans-Erik Eriksson & Magnus Penker, John Wiley & Sons Inc, 1. udgave, 0-471-19161-2.

[Keil, 1997] C51 Compiler, Keil software, Keil software.

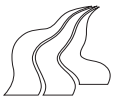
[Kelley, 1997] A book on C, Al Kelley & Ira Pohl, Addison-Wesley, 4. udgave, 0-201-18399-4

[Lewis, 2000] JAVA software solutions, John Lewis & William Loftus, Addison-Wesley, 2. udgave, 0-201-61271-2.

[Oncore, 2000] M12 Oncore User's Guide Supplement, *Ikke oplyst*, Motorola, Udgave 1.0, *Ikke oplyst*. Forefindes på vedlagte CD.

[Sedra, 1998] Microelectronic Circuits, 1998, Adel S. Sedra & Kenneth C. Smith, Oxford University Press, 4 udgave, 0-19-511690-9.

[Siemens, 1995] Microcomputer Components SAB 80C517 / 80C537, Siemens Semiconductor Group



[SPU, 1998] Håndbog i Struktureret ProgramUdvikling, 1998, Stephen Biering-Sørensen, Finn Overgaard Hansen, Susanne Klim og Preben Thalund Madsen, Teknisk Forlag A/S, 1. udgave 8. oplag, 87-571-1046-8

[Tanenbaum, 1996] Computer Networks, Andrew S. Tanenbaum, Prentice hall, 3. udgave, 0-13-394248-1

[Tanenbaum, 1999] Structured Computer Organization, Andrew S. Tanenbaum, Prentice Hall, 4. udgave, 0-13-020435-8

Appendiks

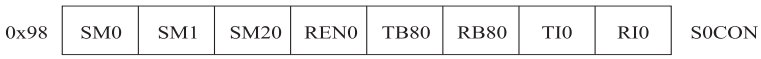
A Registre på microcontroller

I dette appendiks beskrives, hvordan en række af de registre, der er benyttet på microcontrolleren, er indstillet.
Skemaerne, som angiver de enkelte bit's betydning, er hentet fra bogen "Microcomputer Components SAB 80C517 / 80C537" [Siemens, 1995], hvorfor de er skrevet på engelsk.

A.1 Introduktion til registre

Microcontrolleren har nogle registre, som benyttes til at styre, hvorledes de serielle porte og timere fungerer, samt et register, der bestemmer hvilke interrupts, der skal aktiveres. Disse registre stilles dels af software og dels af eksterne begivenheder i hardware, som fx interrupts.
For at give et overblik over registrene, og hvordan de skal indstilles for at opnå de ønskede hastigheder og virkemåder, er hvert af de benyttede registre forklaret her.

A.1.1 S0CON



Figur A.1 Placering af- og navn på de enkelte bit i S0CON

Betydning af bit i S0CON		
Bit		**Function**
SM0 0 0 1 1	SM1 0 1 0 1	Serial mode 0: Shift register mode, fixed baud rate. Serial mode 1: 8-bit UART, variable baud rate. Serial mode 2: 9-bit UART, fixed baud rate. Serial mode 3: 9-bit UART, variable baud rate.
SM20		Enables the multiprocessor communication feature in modes 2 and 3. In mode 2 or 3 and SM20 being set to 1, RI0 will not be activated if the received 9th data bit (RB80) is 0. In mode 1 and SM20 = 1, RI0 will not be activated if a valid stop bit has not been received. In mode 0, SM20 should be 0.
REN0		Receiver enable. Enables serial reception. Set by software to enable reception. Cleared by software to disable reception.
TB80		Transmitter bit 8. Is the 9th bit that will be transmitted in modes 2 and 3. Set or cleared by software as desired.
RB80		Receiver bit 8. Is the 9th data bit that was received. In mode 1, if SM20 = 0, RB80 is the stop bit that was received. In mode 0, RB80 is not used.

TI0	Transmitter interrupt. Is the interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or at the beginning of the stop bit in the other modes, in any serial transmission. Must be cleared by software.
RI0	Receiver interrupt. Is the receive interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or during the stop bit time in the other modes, in any serial reception. Must be cleared by software.

Dette register bruges til at indstille hvorledes serielport 0 skal fungere. Da UART'en på serielport 0, skal fungere som en 8 bit UART, med en baudrate på 38.400, skal registret sættes således at UART'en kommer i mode 1, dvs. SM0 = 0 og SM1 = 1. SM20 er sat høj, således at microcontrolleren kan modtage data via interrupt. For at aktivere serielporten således, at der kan modtages data er REN0 sat høj. De resterende bit i dette register, som blandt andet indeholder serielportens interruptflag, skal ikke stilles af software, men blot læses og sættes af microcontrolleren under brug.
På grundlag af ovenstående sættes registeret til 0x70.

A.1.2 PCON

Figur A.2 Placering af- og navnene på de enkelte bit i PCON

Betydning af bit i PCON	
Bit	**Function**
SMOD	When set, the baud rate of serial interface 0 in modes 1, 2, 3 is doubled.

Dette register benyttes, sammen med registeret TH1, til at indstille serielport 0's baudrate.
I manualen til microcontrolleren findes en formel, der fortæller, hvorledes disse registre skal indstilles, for at opnå den ønskede baudrate:

$$baud\ rate = \frac{2^{SMOD} \cdot f_{osc}}{32 \cdot 12 \cdot (256 - TH1)} \qquad \text{(A.1)}$$

Da serielporten skal indstilles til en baudrate på 38.400, og microcontrolleren er opkoblet med et krystal på 14,7456 MHz, kan registrenes indstillinger udregnes ved hjælp af formel A.1.
SMOD benyttes til at fordoble baudraten, og da der skal benyttes en baudrate på 38.400, sættes SMOD høj. Herefter kan registret TH1's værdi beregnes:

$$TH1 = 256 - \frac{2^{SMOD} \cdot f_{osc}}{Baud\ rate \cdot 12 \cdot 32} = 256 - \frac{2 \cdot 14.7456\ \text{MHz}}{38.400 \cdot 12 \cdot 32} = 254 \qquad \text{(A.2)}$$

På grundlag af ovenstående sættes registret SMOD = 1 og registeret TH1 til 0xFE.
Registeret TH1 og dets virkemåde er nærmere beskrevet under registeret TMOD.

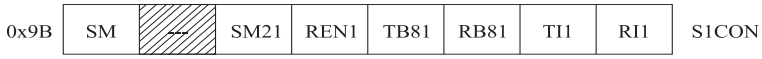
A.1.3 ADCON

Figur A.3 Placering af- og navnene på de enkelte bit i ADCON

Betydning af bit i ADCON	
Bit	**Function**
BD	Baud rate enable. When set, the baud rate in modes 1 and 3 of serial interface 0 is taken from a dedicated prescaler. Standard baud rates 4800 and 9600 baud at 12 MHz oscillator frequency can be achieved.

Dette register benyttes til at bestemme om serielport 0, skal benytte en indbygget skaleringsenhed, til at sætte baudraten. Da der benyttes en timer til at indstille serielportens baudrate, skal skaleringsenheden være deaktiveret, hvorfor BD sættes lav.

A.1.4 S1CON

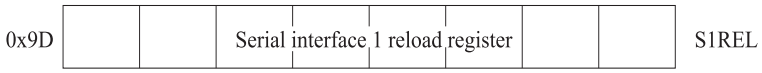


Figur A.4 Placering af- og navnene på de enkelte bit i S1CON

Betydning af bit i S1CON	
Bit	**Function**
SM	SM = 0: serial mode A; 9-bit UART. SM = 1: serial mode B; 8-bit UART.
SM21	Enables the multiprocessor communication feature in mode A. If SM21 is set to 1, RI1 will not be activated if the received 9th data bit (RB81) is 0. In mode B, if SM21 = 1, RI1 will not be activated if a valid stop bit was not received.
REN1	Receiver enable of interface 1. Enables serial reception. Set by software to enable reception. Cleared by software to disable reception.
TB81	Transmitter bit 8 of interface 1. Is the 9th data bit that will be transmitted in mode A. Set or cleared by software as desired.
RB81	Receiver bit 8 of interface 1. Is the 9th data bit that was received in mode A. In mode B, if SM21 = 0, RB81 is the stop bit that was received.
TI1	Transmitter interrupt of interface 1. Is the transmit interrupt flag. Set by hardware at the beginning of the stop bit in any serial transmission. Must be cleared by software.
RI1	Receiver interrupt of interface 1. Is the receive interrupt flag. Set by hardware at the halfway through the stop bit time in any serial transmission. Must be cleared by software.

Dette register bruges til at indstille, hvorledes serielport 1 skal fungere. Da UART'en på serielport 1, skal fungere som en 8 bit UART, med en baudrate på 4800, skal registret sættes således, at UART'en kommer i mode 1, dvs. SM = 1. SM21 er sat høj, således at microcontrolleren kan modtage data via interrupt. For at aktivere serielporten, således at der kan modtages data, er REN1 sat høj. De resterende bit i dette register, som blandt andet indeholder serielportens interruptflag, skal ikke stilles af software, men blot læses og sættes af microcontrolleren under brug.
På grundlag af ovenstående, sættes registeret til 0xB0.

A.1.5 S1REL



Figur A.5 Tallet i S1REL angiver startværdien for tælleren.

Dette register bruges til at styre baudraten på serielport 1. I manualen til microcontrolleren findes en formel, der fortæller, hvorledes dette register skal indstilles, for at opnå den ønskede baudrate:

$$baud\ rate = \frac{f_{osc}}{32 \cdot (256 - S1REL)} \tag{A.3}$$

Da serielporten skal indstilles til en baudrate på 4800, og microcontrolleren er opkoblet med et krystal på 14,7456 MHz, kan registrets indstilling udregnes ved hjælp af formlen:

$$S1REL = 256 - \frac{f_{osc}}{Baud\ rate \cdot 32} = 256 - \frac{14.7456\ \text{MHz}}{4800 \cdot 32} = 160 \tag{A.4}$$

På grundlag af ovenstående, sættes registret S1REL til 0xA0.
Serielporten virker ved at reloadværdien i S1REL indlæses i en tæller som tæller op, og når den når 255, læser den data på porten, og starter forfra.

A.1.6 TCON

Figur A.6 Placeringen af- og navnene på de enkelte bit i TCON

Betydning af bit i TCON	
Bit	**Function**
TR0	Timer 0 run control bit. Set / cleared by software to turn timer / counter 0 ON / OFF.
TF0	Timer 0 overflow flag. Set by hardware on timer / counter overflow. Cleared by hardware when processor vectors to interrupt routine.
TR1	Timer 1 run control bit. Set / cleared by software to turn timer / counter 1 ON / OFF.
TF1	Timer 1 overflow flag. Set by hardware on timer / counter overflow. Cleared by hardware when processor vectors to interrupt routine.

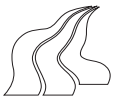
Dette register bruges til at styre, hvilke af timerne, der skal være aktive. Da begge timere skal benyttes, skal både TR0 og TR1 sættes høj. TF0 og TF1 er interruptflag, som sættes og resettes af hardware. De resterende 4 bit i dette register benyttes ikke.
På grundlag af ovenstående, sættes registeret til 0x50.

A.1.7 TMOD

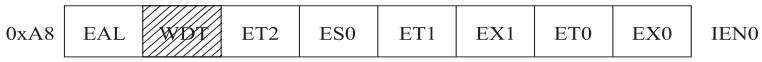
Figur A.7 Placering af- og navnene på de forskellige bit i TMOD

Betydning af bit i TMOD		
Bit		**Function**
Gate		Gating control. When set, timer / counter "x" is enabled only while "INT x" pin is high and "TRx" control bit is set. When cleared timer "x" is enabled whenever "TRx" control bit is set.
C / T		Counter or timer select bit. Set for counter operation (input from "Tx" input pin). Cleared for timer operation (input from internal system clock).
M1 0	M0 0	8-bit timer / counter. "THx" operates as 8-bit timer / counter. "TLx" serves as 5-bit prescaler.
0	1	16-bit timer / counter. "THx" and "TLx" are cascaded; there is no prescaler.
1	0	8-bit auto-reload timer / counter. "THx" holds a value which is to be reloaded into "TLx" each time it overflows.
1	1	Timer 0: TL0 is an 8-bit timer / counter controlled by the standard timer 0 control bits. TH00 is an 8-bit timer only controlled by timer 1 control bits.
1	1	Timer 1: Timer / counter 1 stops.

Dette register benyttes til at bestemme, hvorledes de to timere / tællere skal fungere. Da de begge skal fungere som timere, er de to bit C/T sat lav. Timer 0, som benyttes til timing af modtagelse fra VHF-radioen, skal indstilles som 8 bit auto reload timer, hvorfor felterne M1 og M0 til timer 0 sættes til henholdsvis 1 og 0. "Gating Control" i timer 0 skal fra start sættes, således at timeren tæller, når pin 3.2 er høj. "Gating Control" sættes og resettes i programmet, afhængig af, om tælleren udelukkende skal tælle på høje pulser eller tælle hele tiden.
Timer 1, som benyttes af serielport 0, skal indstilles som 8 bit timer med "auto-reload, hvorfor felterne M1 og M0 til timer 1 sættes til henholdsvis 1 og 0. Det betyder, at timeren indlæser reloadværdien fra registret TH1, tæller op, og når den får overflow, genereres et interrupt, som i dette tilfælde får serielporten til at sende data. Da timer 1 skal tælle hele tiden, er "Gating Control" sat lav.
På grundlag af ovenstående sættes registret til 0x2A.



A.1.8 IEN0



Figur A.8 Placering af- og navnene på de enkelte bit i IEN0

Betydning af bit i IEN0	
Bit	**Function**
EX0	Enables or disables external interrupt 0. If EX0 = 0, external interrupt 0 is disabled.
ET0	Enables or disables the timer 0 overflow interrupt. If ET0 = 0, the timer 0 interrupt is disabled.
EX1	Enables or disables external interrupt 1. If EX1 = 0, external interrupt 1 is disabled.
ET1	Enables or disables the timer 1 overflow interrupt. If ET1 = 0, the timer 1 interrupt is disabled.
ES0	Enables or disables the serial channel 0 interrupt. If ES0 = 0, the serial channel 0 interrupt is disabled.
ET2	Enables or disables the timer 2 overflow or external reload interrupt. If ET2 = 0, the timer 2 interrupt is disabled.
EAL	Enables or disables all interrupts. If EAL = 0, no interrupts will be acknowledged. If EAL = 1, each interrupt source is individualy enabled or disabled by setting or clearing its enable bit.

Dette register benyttes til at bestemme hvilke interrupts, der er udmasket.
Efter initialiseringen ophæves udmaskningen af EX0 for extern interrupt, ES0 for interrupt på serielport 0 og til sidst aktiveres EAL, som sætter ovenstående interrupts i kraft.

A.1.9 IEN2

Figur A.9 Placering af- og navnene på de enkelte bit i IEN2

Betydning af bit i IEN2	
Bit	**Function**
ES1	Enable serial interrupt of interface 1. Enables or disables the interrupt of serial interface 1. If ES1 = 0, the interrupt is disabled.
ECT	Enable compare timer interrupt. Enables or disables the interrupt at compare timer overflow. If ECT = 0, the interrupt is disabled.

Dette register benyttes til at aktivere ES1, som aktiverer interrupt på serielport 1.

B Fejlkontrol af data

Dette appendiks omhandler forskellige metoder til fejlkontrol af data. Det er medtaget for at forklare teorien bag de metoder, der er anvendt i projektet. Afsnittet er baseret på bogen "Grundlæggende Teori for Fejlkontrol Kodning" [Ebert, 1996].

B.1 Introduktion til fejlkontrol

For at undgå, at der arbejdes med ugyldige data, benyttes fejlkontrol. Inden data sendes, beregnes der en checksum, som sendes sammen med data. Når data modtages, kan modtageren ud fra checksummen beregne, om data er gyldige.
Der findes to grundlæggende typer koder til beregning af checksum: blokkoder og trækoder. Under blokkoderne findes der Hammingkoder, cykliske koder og CRC koder. Under trækoderne findes der forskellige former for foldningskoder. I AIS protokollen benyttes en fejlkontrolmetode der hedder CRC-ITU-T (også kaldet CRC-CCITT), som benytter den kodetype, der hedder CRC koder under blokkoderne. NMEA standarden benytter en anden form for checksum. Det er derfor kun disse to, der vil blive yderligere beskrevet. For bedre at give et indblik i, hvordan koderne er opbygget og fungerer, vil nogle af de matematiske værktøjer, der skal bruges, blive beskrevet.

B.2 Matematiske værktøjer

Nøgleordet i fejlkontrol er polynomieregning. Et polynomie kan opskrives ved hjælp af binære tal, således at hver bit repræsenterer antallet af x, x^2,x^3 osv.

X^{16}	X^{15}	X^{14}	X^{13}	X^{12}	X^{11}	X^{10}	X^9	X^8	X^7	X^6	X^5	X^4	X^3	X^2	X^1	X^0
1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1

Figur B.1 Eksempel på polynomie.

Da en bit kun kan have to tilstande, kan polynomiet således også kun angive, om det tilsvarende led er tilstede eller ej. På grundlag af ovenstående vil de matematiske værktøjer "modulo 2 regning" og "polynomieregning" blive beskrevet.

B.2.1 Modulo 2 regning

Da en computer kun kan arbejde med binære tal, er det vigtigt at se på, hvorledes der udføres aritmetik med disse. Figur B.2 viser, hvorledes der regnes modulo 2 med to tal A og B.

Regneregler for modulo 2				
A	B	A+B	A-B	A*B
0	0	0	0	0
0	1	1	1	0
1	0	1	1	0
1	1	0	0	1

Det ses, at sum og differens er det samme, og at de kan realiseres ved hjælp af XOR. Endvidere ses det, at multiplikation kan realiseres ved hjælp af AND.

B.2.2 Polynomieregning

Polynomieregning, herunder multiplikation og division af og med polynomier, er metoder, der benyttes ved udregning af checksum. Der vil derfor blive gennemgået, hvorledes disse metoder udføres.
Multiplikation sker på sædvanlig måde ved at hvert led multipliceres med de andre led. Da antallet af hvert led repræsenteres af en bit, kan man kun angive om leddet er tilstede eller ej. På grundlag af dette, bruges modulo 2 regning til polynomieregningen.
Dette betyder at $x^2 + x^2 = 0$.
Et eksempel på multiplikation med polynomier og modulo 2 regning er vist herunder.

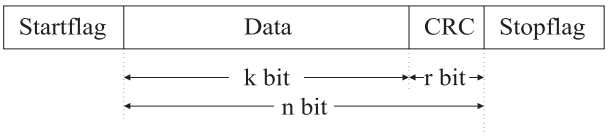
$$(X^3 + X + 1) \cdot (X^2 + 1) = X^5 + X^2 + X + 1 = [100111] \qquad \text{(B.1)}$$

Division foregår også på sædvanlig måde, blot man tager højde for at $x^2 = -x^2$.
Et eksempel på dette er vist herunder.

$$\begin{array}{rcl}
X^7 + 1 & : & X^3 + X + 1 = X^4 + X^2 + X + 1 \\
\underline{X^7 + X^5 + X^4} & & (X^4) \\
X^5 + X^4 + 1 & & \\
\underline{X^5 + X^3 + X^2} & & (X^2) \\
X^4 + X^3 + X^2 + 1 & & \\
\underline{X^4 + X^2 + X} & & (X) \\
X^3 + X + 1 & & \\
\underline{X^3 + X + 1} & & (1) \\
\text{Rest} = 0 & &
\end{array} \qquad \text{(B.2)}$$

B.3 CRC-ITU-T (CRC-CCITT)

CRC-koder benyttes meget i datanetprotokoller, og i næsten alle protokoller findes et checkfelt med CRC-kode. Feltet kaldes enten CRC (Cyclic Redundancy Check), LRC (Longitudinal Redundancy Check) eller FCS (Frame Check Sequence). Metoden benytter en systematisk cyklisk kode, som modificeres, således at der som resultat fås en acyklisk og ulineær, men systematisk kode. Opbygningen af den datablok, som koden beskytter ser ud som følger:



Figur B.2 Opdeling af frame i data og CRC felt.

For at beregne checksummen benyttes et kodepolynomie (også kaldet et generatorpolynomie). Dette kodepolynomie skal kendes af både sender og modtager. I CRC-ITU-T bruges generatorpolynomiet $X^{15}+X^{12}+X^5+1$ = [10001000000100001]. I modsætning til almindelige cykliske koder, hvor checksummen på modtagersiden skal give nul for en fejlfri transmission, skal checksummen ved CRC give et tal forskelligt fra nul. Dette tal kaldes et "residue" og er i CRC-ITU-T lig med 0x1D0F.

B.3.1 Algoritme for beregning af CRC

På senderens side foretages følgende:

1 Data ganges med 2^r, hvor r er antallet af checkbit (i dette tilfælde 16 bit).
2 De første r bit i data inverteres.
3 Resultatet af 2 divideres med generatorpolynomiet, og divisionsresten gemmes.
4 Divisionsresten inverteres og adderes til data, og dermed fås en datablok med data og CRC.

På modtagesiden foretages følgende:

1 Data uddrages af datablokken og gemmes for brug uden CRC check.
2 De første r bit i datablokken inverteres.
3 Resulatet af 2 ganges med 2^r og divideres med generatorpolynomiet.
4 Hvis divisionsresten fra 3. er lig med residuet (i dette tilfælde 0x1D0F), har transmissionen været fejlfri.

B.3.2 Eksempel på beregning af CRC.

For at illustrere denne metode, gennemgås et eksempel på afsendelse og modtagelse af data, svarende til "ABC" ("01000001010000101000011" binært).

```
01000001010000101000011                          Data der skal kodes
01000001010000101000011000000000000000000       Der ganges med 2^16
10111110101111010100001100000000000000000       De første 16 bit invert.
10001000000100001                                Generatorpolynomie
00110110101011011100001100000000000000000       XOR
  10001000000100001                              Generatorpolynomie
  01010010101001111000110000000000000000        XOR
   10001000000100001                             Generatorpolynomie
   00101101010111111001100000000000000000       XOR
     10001000000100001                           Generatorpolynomie
     00111101011011101110000000000000000        XOR
       10001000000100001                         Generatorpolynomie
       01111101101010110000000000000000          XOR
        10001000000100001                        Generatorpolynomie
        01110011010001101000000000000000          XOR
         10001000000100001                       Generatorpolynomie
         01101110100111011000000000000000        XOR
          10001000000100001                      Generatorpolynomie
          01010101000101011100000000000000       XOR
           10001000000100001                     Generatorpolynomie
           00100010001000111100000000000000      XOR
             10001000000100001                   Generatorpolynomie
             00000001000011101000000000000       XOR
                    10001000000100001            Generatorpolynomie
                    00001111010100001000         XOR
                        0000101011110111         Rest inverteres
                           0   A   F   7         Rest i hexadecimal
```

Den beregnede checksum adderes til data og bitstrengen afsendes.

```
01000001010000100100001100001010111110111                                  Data der modtages
01000001010000100100001100001010111101110000000000000000                   Der ganges med 2^16
10111110101111010100001100001010111101110000000000000000                   De første 16 bit invert.
10001000000100001                                                           Generatorpolynomie
00110110101011011100001100001010111101110000000000000000                   XOR
  10001000000100001                                                         Generatorpolynomie
  01010010101001111000110000101011110111000000000000000                    XOR
   10001000000100001                                                        Generatorpolynomie
   00101101010111111001100001010111101110000000000000000                   XOR
     10001000000100001                                                      Generatorpolynomie
     00111101011011101100001010111101110000000000000000                    XOR
       10001000000100001                                                    Generatorpolynomie
       01111101101010110000010101111011100000000000000000                  XOR
        10001000000100001                                                   Generatorpolynomie
        01110011010001101000101011110111000000000000000000                  XOR
         10001000000100001                                                  Generatorpolynomie
         01101110100111011001010111101110000000000000000000                 XOR
          10001000000100001                                                 Generatorpolynomie
          01010101001010111010101111011100000000000000000000                XOR
           10001000000100001                                                Generatorpolynomie
           00100010010001111101011110111000000000000000000000               XOR
             10001000000100001                                              Generatorpolynomie
             00000010000111111011110111000000000000000000000000             XOR
                    10001000000100001                                       Generatorpolynomie
                    00001111111111111111000000000000000000                  XOR
                        10001000000100001                                   Generatorpolynomie
                        01110111111011111000000000000000                    XOR
                         10001000000100001                                  Generatorpolynomie
                         01100111110011111000000000000000                   XOR
                          10001000000100001                                 Generatorpolynomie
                          01000111100011111000000000000000                  XOR
                           10001000000100001                                Generatorpolynomie
                           00000111000011111000000000000000                 XOR
                                10001000000100001                           Generatorpolynomie
                                01101001111000001000000                     XOR
                                 10001000000100001                          Generatorpolynomie
                                 01011011110100011000000                    XOR
                                  10001000000100001                         Generatorpolynomie
                                    1111111011001100000                     XOR
                                    10001000000100001                       Generatorpolynomie
                                    01110110110111101000                    XOR
                                     10001000000100001                      Generatorpolynomie
                                     01100101101101100                      XOR
                                      10001000000100001                     Generatorpolynomie
                                      01000011010010111                     XOR
                                       10001000000100001                    Generatorpolynomie
                                       00001110100001111                    Rest
                                           1   D   0   F                    Rest i hexadecimal
```

Det ses at modtageren får checksumen 0x1D0F, og da det svarer til residuet betragtes transmissionen som fejlfri.

B.4 NMEA checksum

NMEA standarden benytter en anden form for checksum. Checksummen fremkommer ved at XOR'e alle tegnene bitvis, og konvertere checksummen til to tegn. Dette medfører, at der opnås 16 bit checksum.

B.4.1 Algoritme for beregning af checksum.

På senderens side foretages følgende:

1 Det første tegn indlæses og gemmes i variablen CRC.
2 Der indlæses et tegn af gangen, foretages XOR med det foregående resultat (CRC), og resultatet gemmes i variablen CRC.
3 Når al data er XOR'et, haves en 8 bit checksum.
4 Checksummens første fire bit konverteres til hexadecimal, og den tilsvarende karakter er den første byte af den checksum, der skal sendes.
5 Checksummens sidste fire bit konverteres til hexadecimal, og den tilsvarende karakter er den sidste byte af den checksum, der skal sendes.

På modtagesiden foretages følgende:

1 Det første tegn indlæses og gemmes i variablen CRC.
2 Der indlæses et tegn af gangen, foretages XOR med det foregående resultat (CRC), og resultatet gemmes i variablen CRC.
3 Når al data er XOR'et, haves en 8 bit checksum.
4 Checksummens første fire bit konverteres til hexadecimal, og sammenlignes med den første byte af den sendte checksum.
5 Checksummens sidste fire bit konverteres til hexadecimal, og sammenlignes med den sidste byte af den sendte checksum.
6 Hvis de to cheksumme er ens er data gyldig.

B.4.2 Eksempel på beregning af checksum.

For at illustrere metoden, gennemgås et eksempel på afsendelse og modtagelse af data, svarende til "ABC" ("010000010100001001000011" binært).

01000001	"A" binært XOR'es med
01000010	"B" binært
00000011	Resultatet XOR'es med
01000011	"C" binært
01000000	Resultat
4 0	Checksum i hexadecimal

Checksummen konverteres til karaktererne "4" ("00110100" binær) og "0" ("00110000" binær), og sendes efter den sidste byte data.

Der modtages således "010000010100001001000011001101000011000" binært og modtageren begynder at dekode.

01000001	"A" binært XOR'es med
01000010	"B" binært
00000011	Resultatet XOR'es med
01000011	"C" binært
01000000	Resultat
4 0	Checksum i hexadecimal

Da den beregnede checksum på modtagersiden svarer til den modtagne checksum, antages det, at der ikke er opstået transmissionsfejl.

C Kollisionsberegning

I dette appendiks gives der en fyldestgørende redegørelse af, hvordan kollisionsberegningsalgoritmen er lavet. Dette appendiks skal ses som et supplement til den mere overordnede forklaring i hovedrapporten, hvor den matematiske teori er beskrevet med formler.

C.1 Kollisionsberegning

Baggrunden for den foretagne kollisionsberegning er følgende:

- Der er opgivet to vektorer plus deres startpunkt.

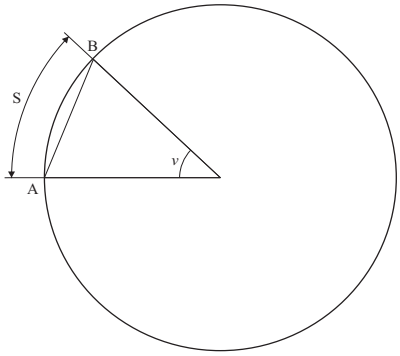
Ud fra disse oplysninger skal det beregnes, om der vil forekomme kollision. Dette gøres ved at finde de linier, som vektorerne følger. Derefter findes deres skæringspunkt, og til slut beregnes det, om skibene vil være i skæringspunktet til samme tidspunkt, inden for en defineret margin. Der er altså tale om en skæring af 2 vektorer i et tredimensionelt rum.

Tilnærmelser

Der er i forbindelse med denne kollisionsberegning foretaget nogle idealiseringer, for at gøre beregningerne simple. Disse er:

- Jorden er flad inden for den kritiske radius.
- Koordinatsystemet er retvinklet inden for den kritiske radius.

For det første er der lavet en tilnærmelse om, at jorden inden for det kritiske område (radius på 3 sømil) er flad. Det vil sige, at afstanden mellem skibenes positioner regnes for lineær og ikke langs en bue, som det i virkeligheden er. Dette er illustreret på figur C.1.



Figur C.1 Liniestykkets størrelse AB set i forhold til buestykket S.

Hvor stor en fejl, dette maksimalt giver udslag i, kan beregnes på følgende måde:

Først beregnes jordens radius ud fra kendskab til definitionen af længdegrader:

$$r = \frac{360° \cdot 1\ \text{sømil} \cdot 1{,}8522\ \frac{\text{km}}{\text{sømil}} \cdot 60\ \text{min}}{2\pi} = 6367{,}40\text{km} \tag{C.1}$$

Dernæst findes den vinkel, som, fra jordens centrum og ud til overfladen, udspænder 3 sømil på jordens overflade (se figur C.1):

$$v = \frac{s \cdot 180°}{\pi \cdot r} = \frac{3\ \text{sømil} \cdot 1{,}8522\ \frac{\text{km}}{\text{sømil}} \cdot 180°}{\pi \cdot 6367{,}40\text{km}} = 0{,}05° \tag{C.2}$$

hvor, *s:* Buelængden i sømil.
r: Jordens radius i km.

Den direkte afstand kan nu beregnes:

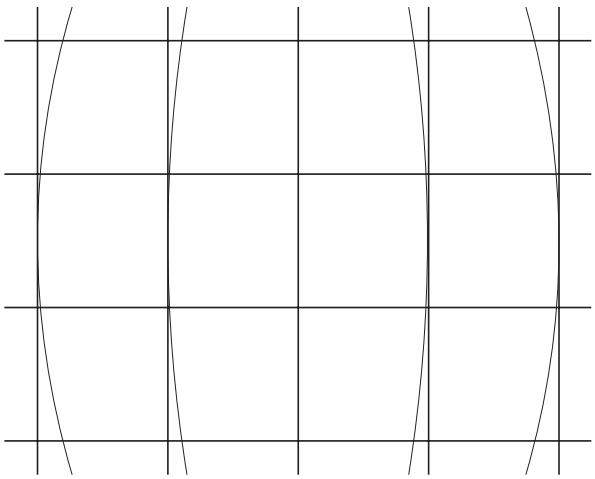
$$\begin{aligned} \text{AB} &= 2 \cdot r \cdot \sin(\tfrac{v}{2}) \\ &= 2 \cdot 6367{,}40\text{km} \cdot \sin(\tfrac{0{,}05°}{2}) \\ &\approx 5{,}556\ \text{km} \end{aligned} \tag{C.3}$$

Dette medfører, at den maksimale fejl bliver:

$$\begin{aligned} E &= 3\text{sømil} \cdot 1{,}8522\ \tfrac{\text{km}}{\text{sømil}} - 5{,}556\text{km} \\ &\approx \underline{0{,}2\text{mm}} \end{aligned} \tag{C.4}$$

Det ses, på baggrund af den beregnede afstand, at fejlmarginen er negligibel.

Den anden idealisering, der foretages er, at koordinatsystemet er retvinklet inden for det kritiske område. Dette er illustreret på figur C.2.



Figur C.2 Afvigelsen fra de virkelige koordinater og det tilnærmede, retvinklede koordinatsystem.

Denne fejl kan beregnes vha. følgende formel:

$$E = a \cdot \cos(\mathrm{BG}) - a \cdot \cos(\mathrm{BG} + v) \tag{C.5}$$

hvor, *a:* Den maximale afstand mellem de to skibe i sømil.
BG: Breddegraden hvor det ene skib er placeret.

Ved at indsætte den maksimale afstand mellem skibene, og sætte breddegraden til den værdi, hvorved den største afvigelse forekommer, findes *worst case* fejlen. Den maksimale afstand mellem eget skib og et andet skib, inden for den kritiske radius, er 3 sømil. Den største afvigelse af breddegrader i sømil, fås ved en vinkel på +/- 90°. Dette gør sig gældende tæt ved Nord- og Sydpolen.Derved fås den maksimale afvigelse:

$$\begin{aligned} E &= 3 \text{ sømil} \cdot \cos(90^\circ) - 3 \text{ sømil} \cdot \cos(90^\circ + 0{,}05^\circ) \\ &\approx 2{,}62 \cdot 10^{-3} \text{ sømil} \\ &\approx \underline{4{,}85 \text{ m}} \end{aligned}$$

Denne afvigelse er betragteligt større end den tidligere beskrevne usikkerhed, hvorfor den ikke umiddelbart kan negligeres. Der forekommer imidlertid en række andre usikkerheder i systemet. Disse er eksempelvis usikkerhed på egen GPS-modtager, samt usikkerheder på de positioner og hastigheder som udsendes fra andre skibe. Det kan nævnes at usikkerheden på positionsangivelsen fra den benyttede GPS-modtager, M12-Oncore, er 25 meter [Oncore, 2000]. På baggrund af de øvrige afvigelser, der forekommer i systemet, samt den øgede beregningskompleksitet der vil forekomme, hvis der skulle tages højde for ovenstående, vælges det ligeledes at se bort fra denne fejl.
Det konkluderes altså, at de tidligere nævnte antagelser / idealiseringer er rimelige.

Udgangspunkt for kollisionsberegningerne
Udgangspunktet for beregningerne er, at der er opgivet egen position, fart og kurs samt, at der er opgivet et andet skibs position, fart og retning. Der er med andre ord opgivet to vektorer samt deres begyndelsespunkt. Vektorerne er opgivet på polær form (med modulus og argument). Det er imidlertid nødvendigt at finde et punkt der angiver, i hvilken retning skibene sejler.

Her omregnes der først fra kompasgrader til almindelige grader

$$\mathrm{COG}_{\mathrm{New}} = 450 - \mathrm{COG} \tag{C.6}$$

Herefter kan slutpunktet findes ved:

$$x_{\mathrm{New}} = x + \mathrm{SOG} \cdot \cos(\mathrm{COG}_{\mathrm{New}}) \tag{C.7}$$
$$y_{\mathrm{New}} = y + SOG \cdot \sin(\mathrm{COG}_{\mathrm{New}}) \tag{C.8}$$

hvor, *x:* Positionen på x-aksen.
25: Positionen på y-aksen.
SOG: Speed Over Ground i knob.
COG: Course Over Ground i kompasgrader.
$\mathrm{COG}_{\mathrm{New}}$: Course Over Ground i grader.
x_{New}: Ny position på x-aksen.
y_{New}: Ny position på y-aksen.

Herefter kan der opskrives et udtryk for liniens ligning:

Først findes hældningen af linien:

$$a = \tan(\mathrm{COG}_{\mathrm{New}}) \tag{C.9}$$

hvor, *a:* Liniens hældning.

Dernæst findes liniens skæringspunkt med y-aksen:

$$b_n = y_n - a_n \cdot x_n \tag{C.10}$$

hvor, *b*: Liniens skæringspunkt med y-aksen.

Ved at sammensætte formel C.9 og C.10, kan to liniers skæringspunkt findes:

$$x_{\mathrm{Intersect}} = \frac{b_2 - b_1}{a_1 - a_2} = \frac{(y_2 - a_2 \cdot x_2) - (y_1 - a_1 \cdot x_1)}{\tan(\mathrm{COG}_1) - \tan(\mathrm{COG}_2)} \tag{C.11}$$

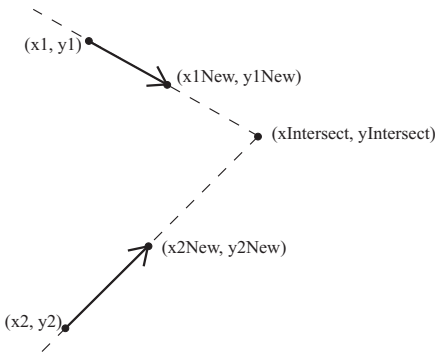
$$y_{\mathrm{Intersect}} = a_1 \cdot x_{\mathrm{Intersect}} + b_1 \tag{C.12}$$

hvor, $x_{\mathrm{Intersect}}$: X-koordinaten hvor linierne skærer hinanden.
$y_{\mathrm{Intersect}}$: Y-koordinaten hvor linierne skærer hinanden.

Det er imidlertid ikke nok, at de to vektorers udspændte linier skærer hinanden, men de skal også være der til samme tidspunkt. Tiden kan betragtes som den tredje dimension. Den første betingelse for kollisionsfare er, at begge vektorer peger ind mod skæringspunktet. Hvorvidt dette er tilfældet, undersøges vha. nedenstående pseudokode. Der henvises iøvrigt til figur C.4.

Start
- ***Hvis*** *det beregnede slutpunkt for vektor1 ligger på samme side af startpunktet som skæringspunktet.*
- ***Så*** *sejler skib1 mod skæringspunktet.*
- ***Hvis*** *det beregnede slutpunkt for vektor2 ligger på samme side af startpunktet som skæringspunktet.*
- ***Så*** *sejler skib2 mod skæringspunktet.*

Slut



Figur C.3 To skibe, hvis linier, skærer hinanden.

Hvis ovenstående er opfyldt, er den eneste betingelse, der mangler at blive opyldt, at de befinder sig i skæringspunktet til samme tid. Dette kan beregnes ved at måle afstanden fra startpunktet til skæringspunktet og dividere med skibets fart. Hvis dette resultat er identisk for de to skibes vektorer, er der kollision. Beregningerne er som følger:

Afstanden fra startpunkt til skæringspunkt beregnes:

$$D_n = \sqrt{(x_{\text{Intersect}} - x_n)^2 + (y_{\text{Intersect}} - y_n)^2} \tag{C.13}$$

hvor, D_n: Afstanden mellem skib n og skæringspunktet.

Forholdet mellem ovenstående afstand og farten beregnes:

$$\text{Rel} = \frac{D_n}{\text{COG}_n} \tag{C.14}$$

hvor, Rel: Forholdet mellem afstanden til skæringspunktet og hvor hurtigt skibet sejler.

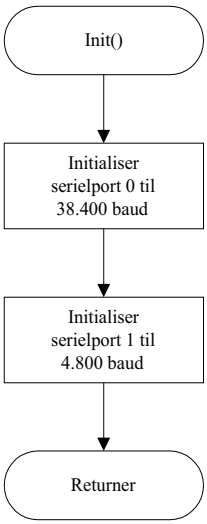
Hvis det beregnede forhold er ens for de to skibe, indenfor en defineret margin, betragtes det som en kollisionsfare.

D Flowcharts MCU software

I dette appendiks findes flowcharts for modulerne i microcontrollersoftwaren.

D.1 Initialisering

Modulet implementeres efter nedenstående flowchart.



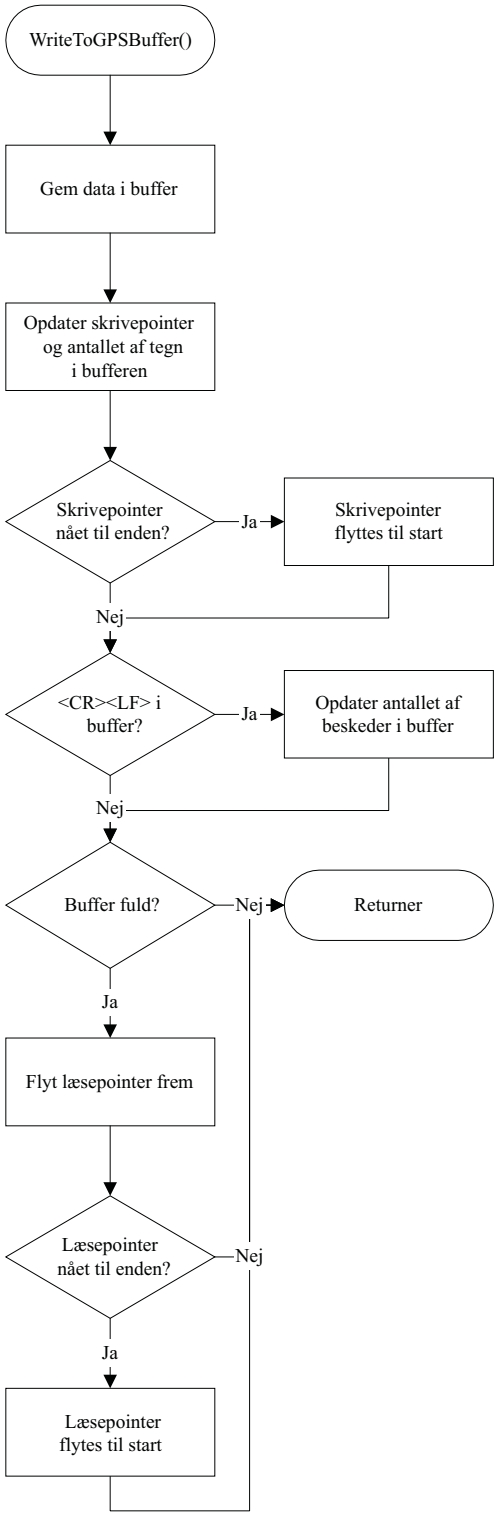
Figur D.1 Flowchart for modulet Initialisering.

D.2 Buffere

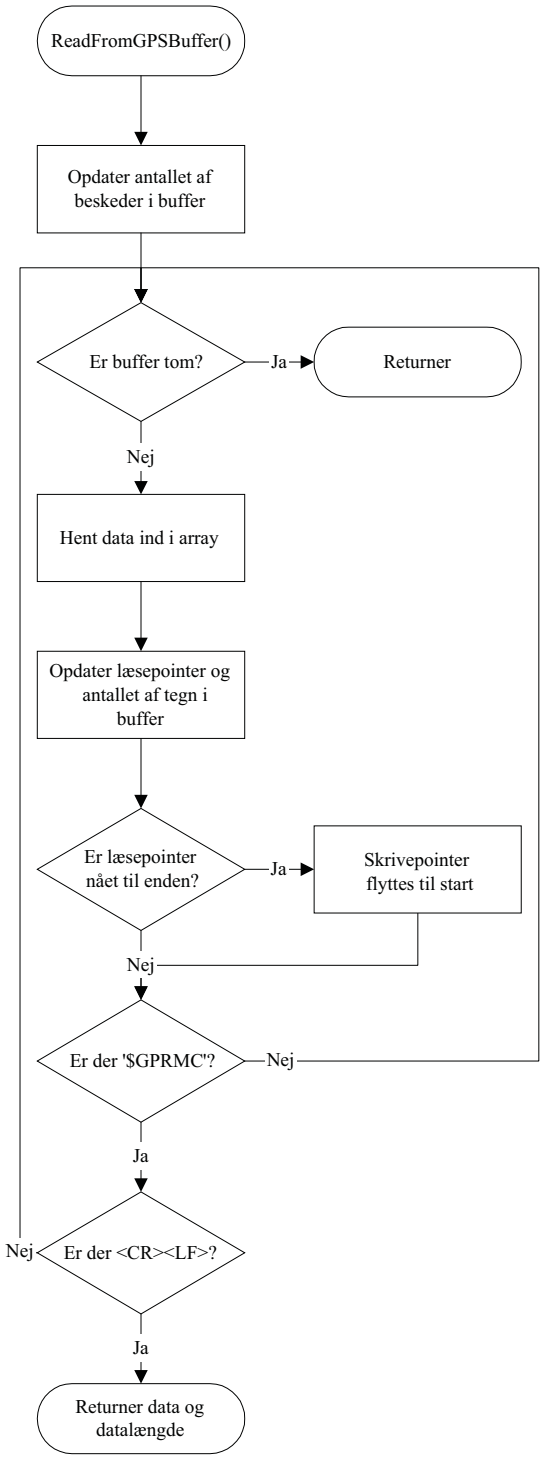
Funktionerne i modulet implementeres efter nedenstående flowcharts.

D.2.1 GPSBuffer

Funktioerne i GPSBuffer implementeres efter nedenstående flowcharts.



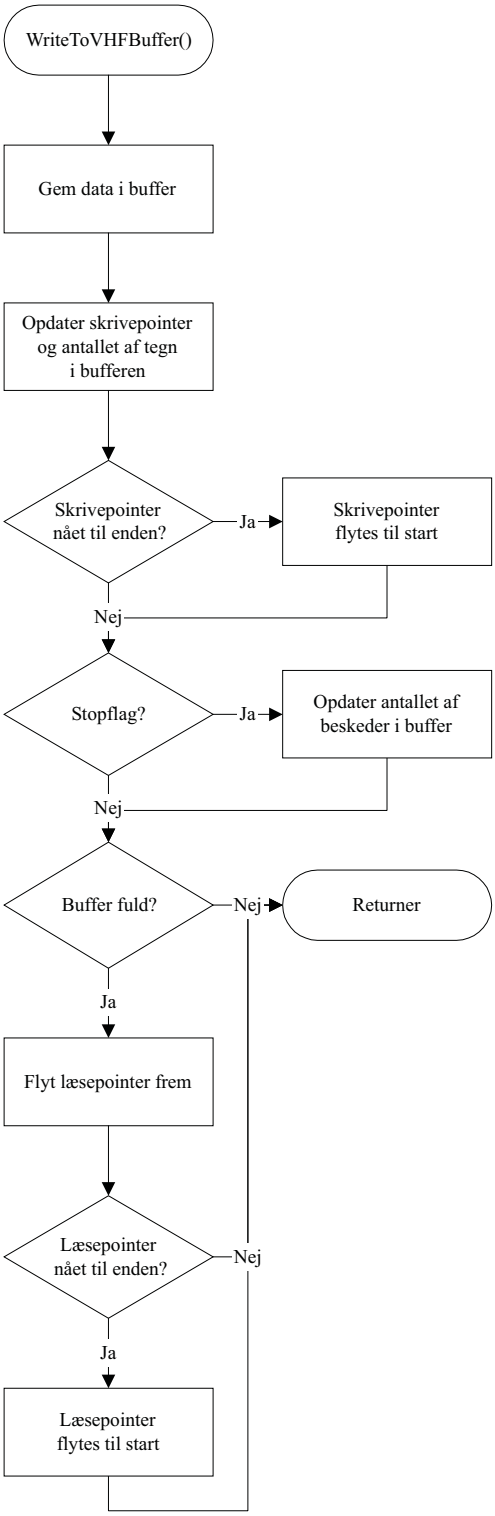
Figur D.2 Flowchart for funktionen WriteToGPSBuffer() i modulet Buffere.



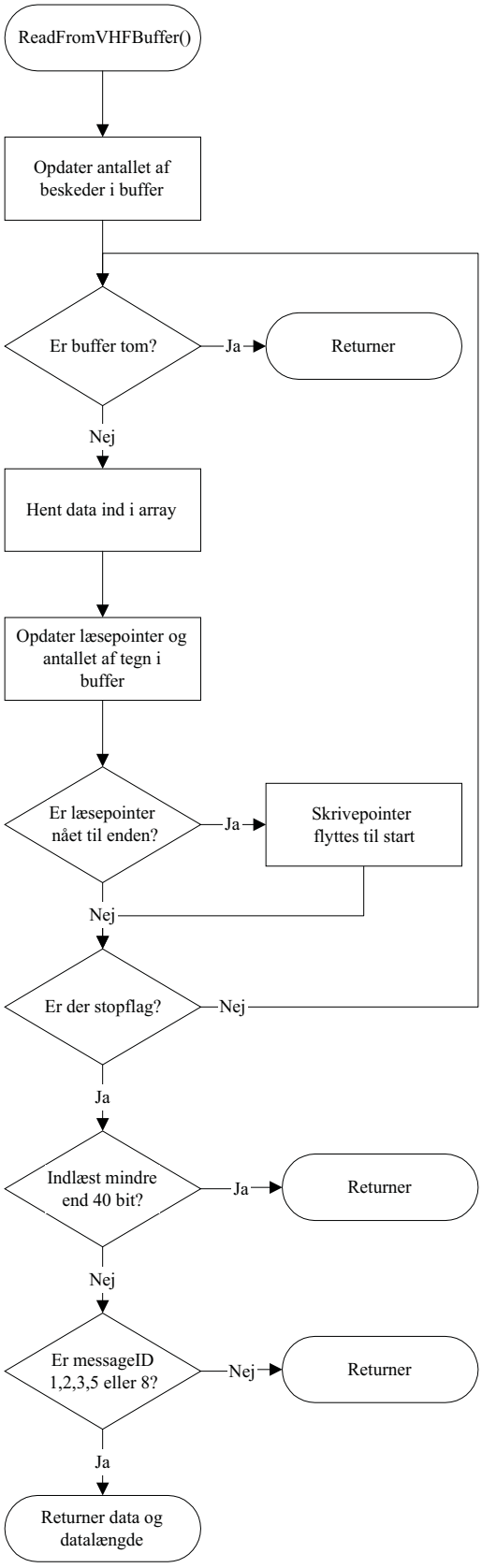
Figur D.3 Flowchart for funktionen ReadFromGPSBuffer() i modulet Buffere.

D.2.2 VHFBuffer

Funktionerne i VHFBuffer implementeres efter nedenstående flowcharts.



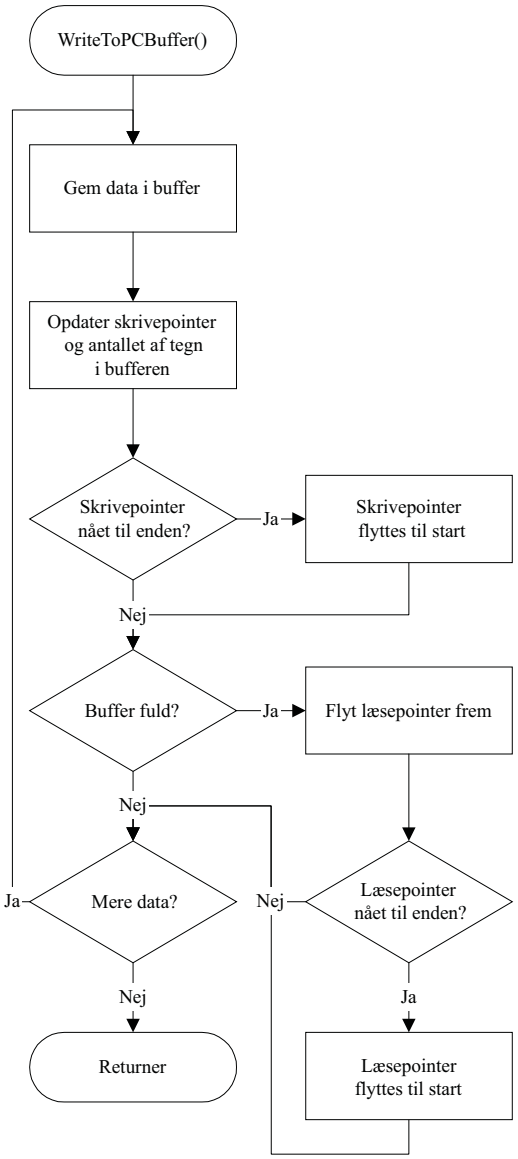
Figur D.4 Flowchart for funktionen WriteToVHFBuffer() i modulet Buffere.



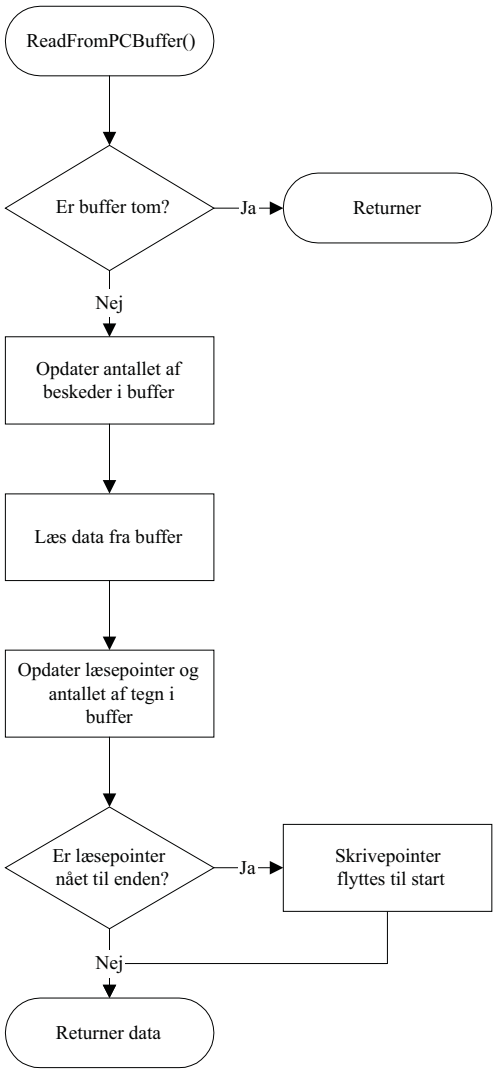
Figur D.5 Flowchart for funktionen ReadFromVHFBuffer() i modulet Buffere.

D.2.3 PCBuffer

Funktionerne i PCBuffer implementeres efter nedenstående flowcharts.



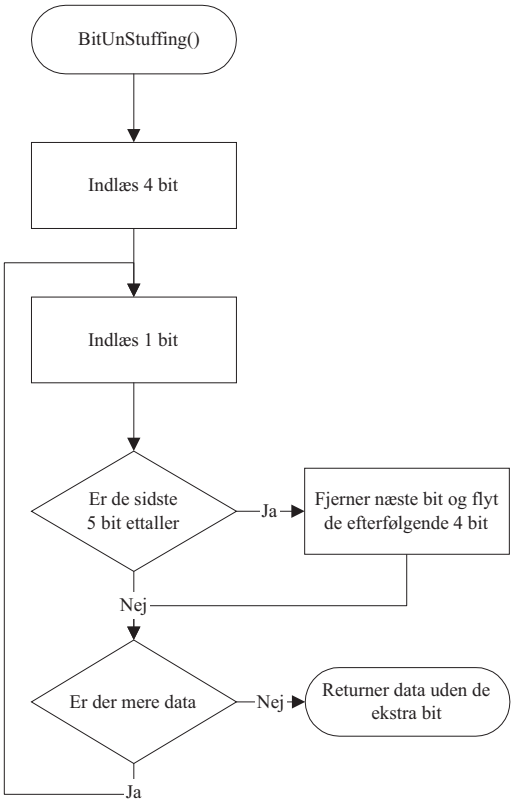
Figur D.6 Flowchart for funktionen WriteToPCBuffer() i modulet Buffere.



Figur D.7 Flowchart for funktionen ReadFromPCBuffer() i modulet Buffere.

D.3 Fjern bit fra bitstuffing

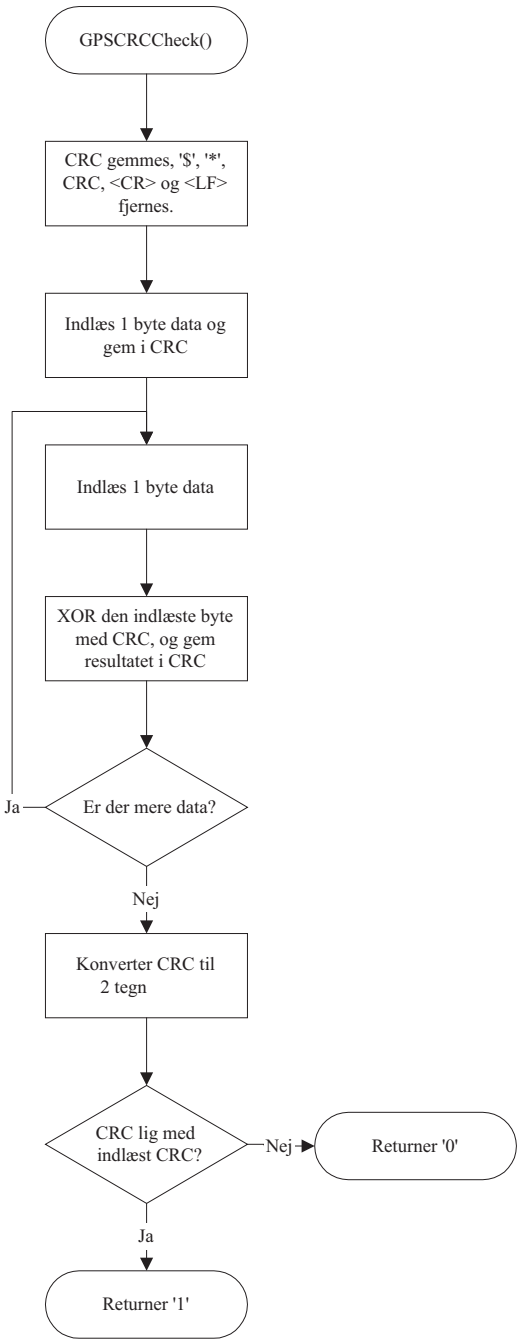
Modulet implementeres efter nedenstående flowchart.



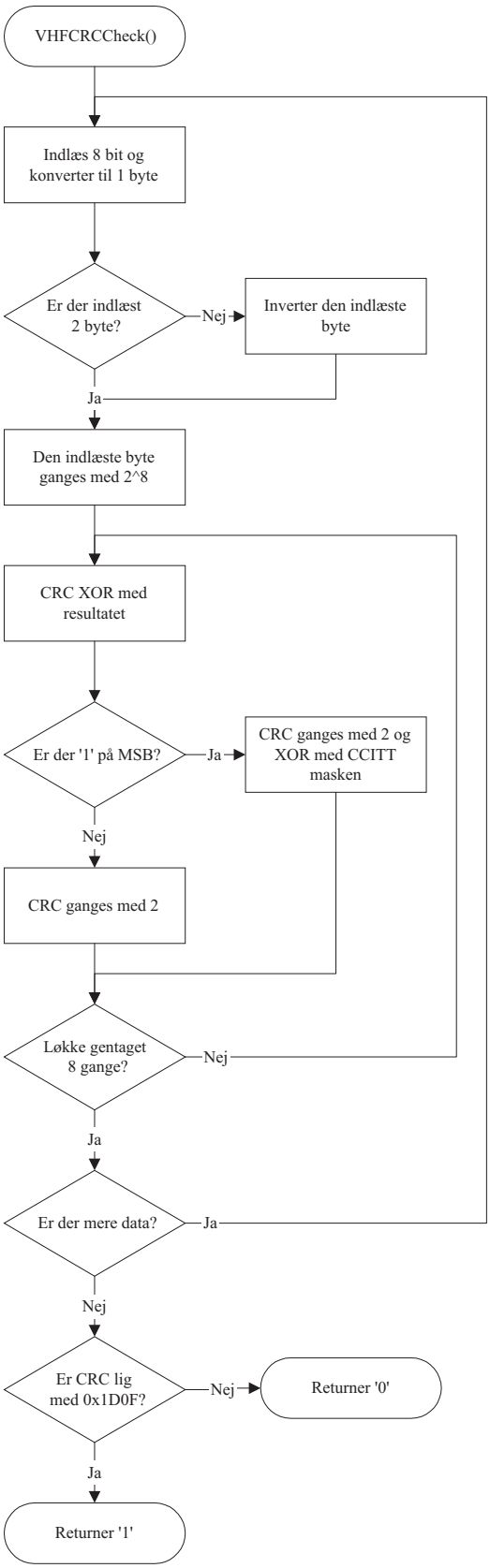
Figur D.8 Flowchart for modulet Fjern bit fra bitstuffing.

D.4 CRCCheck

Funktionerne i modulet implementeres efter nedenstående flowcharts.



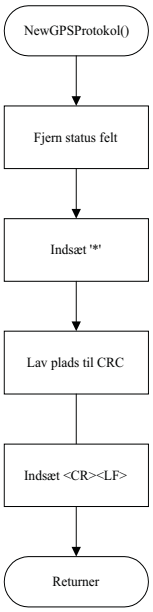
Figur D.9 Flowchart for funktionen GPSCRCCheck() i modulet CRCCheck.



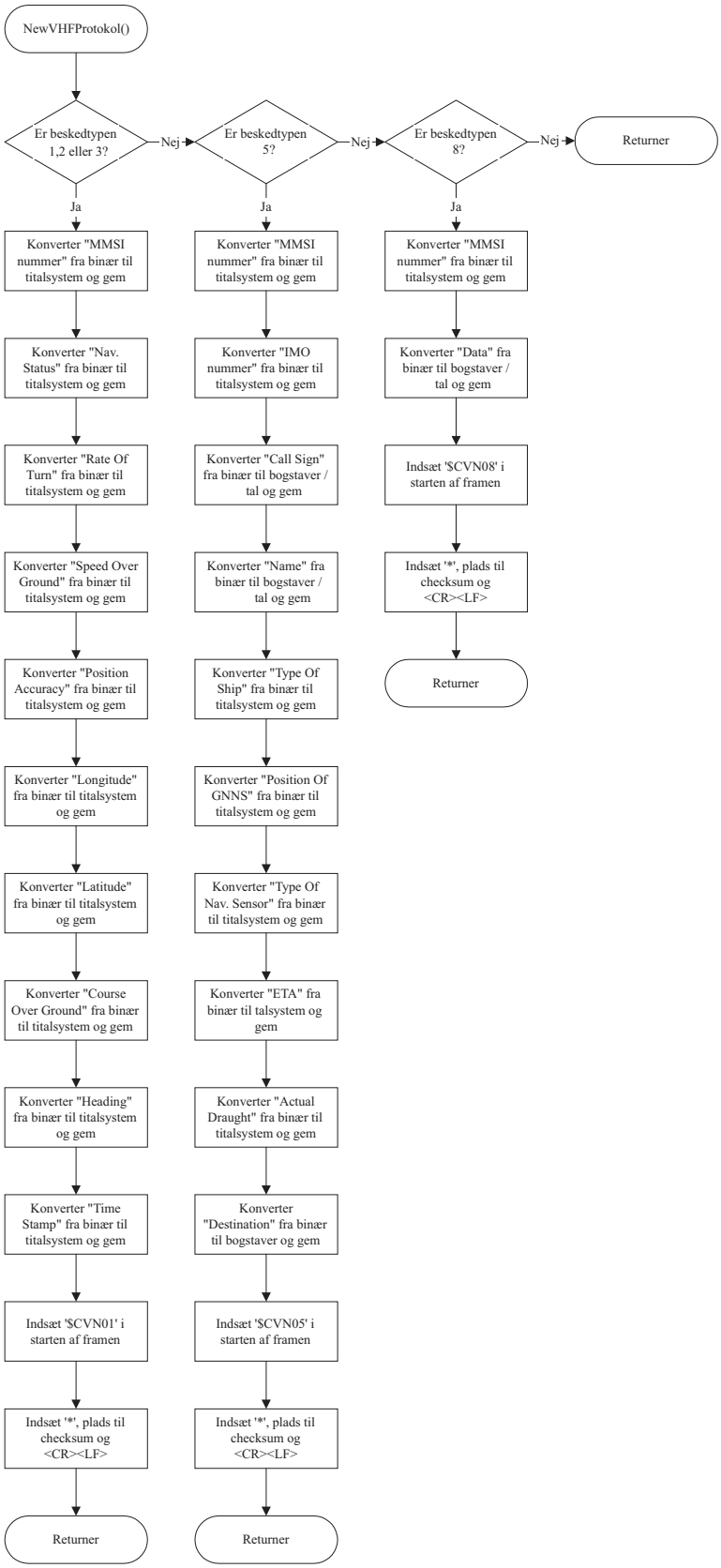
Figur D.10 Flowchart for funktionen VHFCRCCheck() i modulet CRCCheck.

D.5 Ny Protokol

Funktionerne i modulet implementeres efter nedenstående flowcharts.



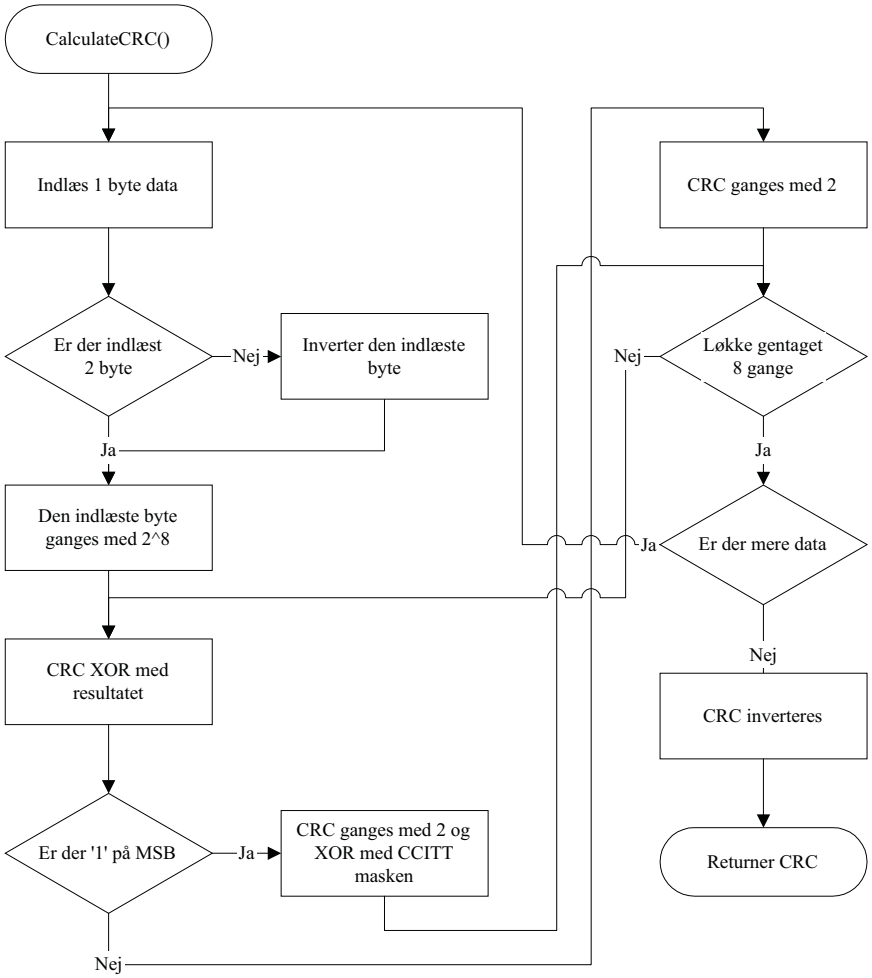
Figur D.11 Flowchart for funktionen NewGPSProtokol() i modulet Ny Protokol.



Figur D.12 Flowchart for funktionen NewVHFProtokol() i modulet Ny protokol

D.6 Beregn ny CRC

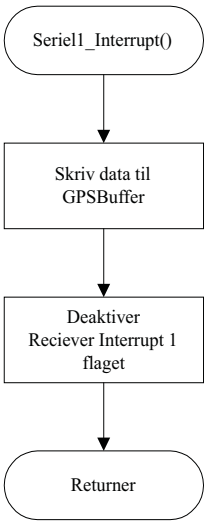
Modulet implementeres efter nedenstående flowchart.



Figur D.13 Flowchart for modulet Beregn ny CRC.

D.7 Datamodtagelse

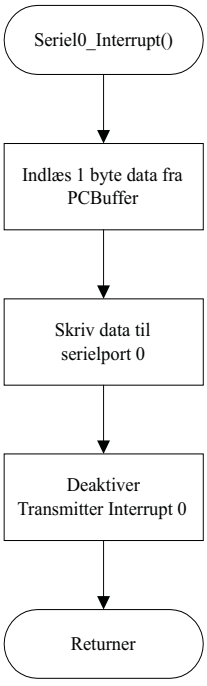
Modulet implementeres efter nedenstående flowchart.



Figur D.14 Flowchart for modulet Datamodtagelse.

D.8 Dataafsendelse

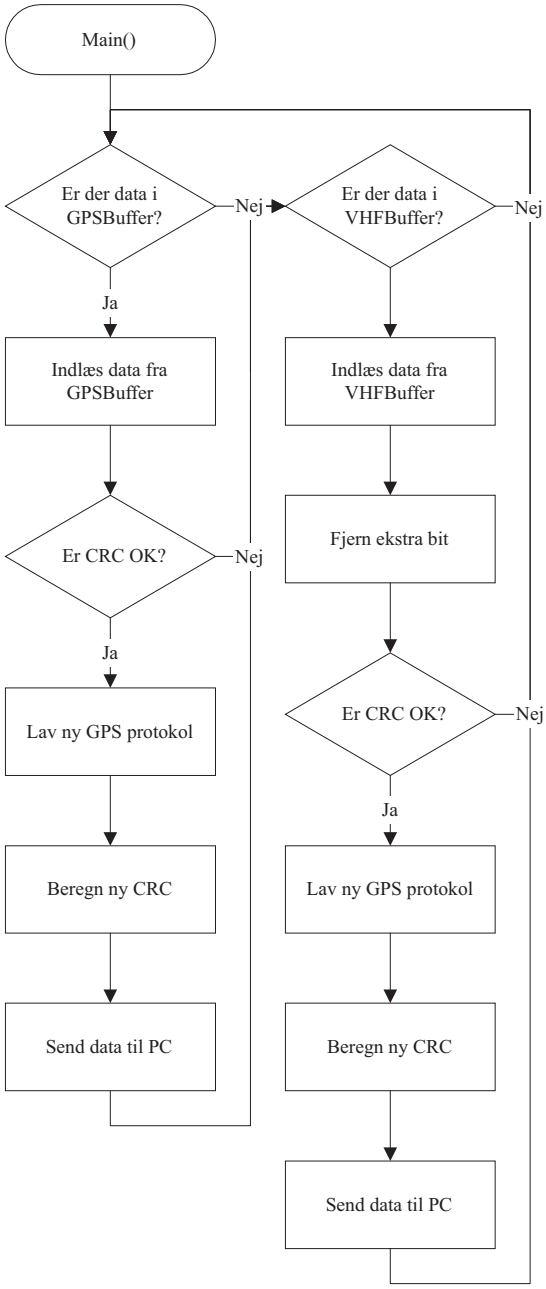
Modulet implementeres efter nedenstående flowchart.



Figur D.15 Flowchart for modulet Dataafsendelse.

D.9 Hovedprogram

Modulet implementeres efter nedenstående flowchart.



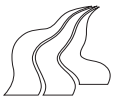
Figur D.16 Flowchart for modulet Hovedprogram.

E Kildekode til microcontroller

I dette appendiks forefindes kildekoden til programmet for microcontrolleren.

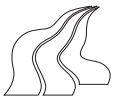
E.1 Kildekode til microcontroller

```
#include <reg517.h>
#include <stdio.h>

#define GPSBufSize 200                  // Definerer størrelsen på GPSBufferen.
#define VHFBufSize 2400                 // Definerer størrelsen på VHFBufferen.
#define PCBufSize 4000                  // Definerer størrelsen på PCBufferen.
#define CCITT 0x1021                    // Definerer CRC-CCITT masken.
int GPSBuffer[GPSBufSize];              // Opretter GPSBuffer med Read- og Write pointer
int GPSBufferRP=0;                      // samt variabler, der angiver antallet af
int GPSBufferWP=0;                      // elementer og antallet af beskeder (frames)
int ElementsInGPSBuffer=0;              // i bufferen.
int DataInGPSBuffer=0;
int VHFBuffer[VHFBufSize];              // Opretter VHFBuffer med Read- og Write pointer
int VHFBufferRP=0;                      // samt variabler, der angiver antallet af
int VHFBufferWP=0;                      // elementer og antallet af beskeder (frames)
int ElementsInVHFBuffer=0;              // i bufferen.
int DataInVHFBuffer=0;
int PCBuffer[PCBufSize];                // Opretter PCBuffer med Read- og Write pointer
int PCBufferRP=0;                       // samt variabler, der angiver antallet af
int PCBufferWP=0;                       // elementer og antallet af beskeder (frames)
int ElementsInPCBuffer=0;               // i bufferen.
int DataInPCBuffer=0;
int Sync=0,SyncCounter=0;
int TrainSequenceTest=0;

//*****************************************************************************//
// Denne funktion initialiserer de serielle porte og timere.                   //
//*****************************************************************************//
void Init(void)
{
   //**************************************************************************//
   // Initialiserer Seriel 0 til at køre 38.400 Baud @ 14,7456MHz vha. timer 1.//
   //**************************************************************************//
   EAL=0;                               // Deaktiver alle interrupt.
   S0CON=0x70;                          // Mode på seriel 0 (Mode 1, 8 bit UART).
   BD=0;                                // Muliggør variabel hastighed på seriel 0.
   PCON=0x80;                           // Hastighed på seriel 0 (SMOD=1).
   TMOD=0x2A;                           // Initialiserer timer 0 og 1.
   TCON=0x50;                           // Aktiver timer 0 og timer 1.
   TH1=0xFE;                            // Reloadværdi til timer 1, som bruges af seriel 0.
   //**************************************************************************//
   // Initialiserer Seriel 1 til at køre 4.800 Baud @ 14,7456MHz.              //
   //**************************************************************************//
   S1CON=0xB0;                          // Mode på seriel 1 (Mode B, 8 bit UART).
   S1REL=0xA0;                          // Reloadværdi for seriel 1.
   EX0=0;   // Sat til 0; Just in case  // Enabler extern interrupt fra VHF.
   ES0=1;                               // Enabler seriel 0 interrupt.
   IEN2=1;                              // Enabler seriel 1 interrupt.
   EAL=1;                               // Aktiver alle interrupt.
}


//*****************************************************************************//
// Denne funktion håndterer styring og indskrivning af data i ringbufferen GPSBuffer.  //
//*****************************************************************************//
void WriteToGPSBuffer(int *Data)
```



```
{
   GPSBuffer[GPSBufferWP]=*Data;       // Data indskrives i buffer.
   GPSBufferWP++;                      // Writepointer opdateres.
   ElementsInGPSBuffer++;              // Antallet af tegn i bufferen opdateres.
   if (GPSBufferWP>=GPSBufSize) GPSBufferWP=0;  // Start forfra.
   if (GPSBufferWP==0)                 // Kontrollerer for <CR><LF> i bufferen ved WP=0.
   {
      if ((GPSBuffer[GPSBufSize-2]==13) && (GPSBuffer[GPSBufSize-1]==10))
         DataInGPSBuffer++;
   }
   else if (GPSBufferWP==1)            // Kontrollerer for <CR><LF> i bufferen ved WP=1.
   {
      if ((GPSBuffer[GPSBufSize-1]==13) && (GPSBuffer[0]==10))
         DataInGPSBuffer++;
   }
   else                                // Kontrollerer for <CR><LF> i bufferen
   {                                   // ved WP!=0 & WP!=1.
      if ((GPSBuffer[GPSBufferWP-2]==13) && (GPSBuffer[GPSBufferWP-1]==10))
         DataInGPSBuffer++;
   }
   if ((GPSBufferWP+1==GPSBufferRP) || ((GPSBufferWP==GPSBufSize-1) && GPSBufferRP==0))
   {
      GPSBufferRP++;                   // Hvis bufferen er fuld, skal RP flyttes frem.
      if (GPSBufferRP>=GPSBufSize) GPSBufferRP=0;
   }
}

//*****************************************************************************//
// Denne funktion håndterer styring og læsning af data i ringbufferen GPSBuffer.       //
//*****************************************************************************//
int ReadFromGPSBuffer(int Data[],int *DataLength)
{
   int *k,*l;
   int FrameOK=0,NotFinish=1;
   k=l=Data;
   DataInGPSBuffer--;                  // Antallet af <CR><LF> i buffer tælles ned.
   while (GPSBufferWP!=GPSBufferRP)    // Der må ikke læses forbi WP.
   {
      *l=GPSBuffer[GPSBufferRP];       // Data læses ind fra buffer.
      GPSBufferRP++;                   // Readpointer opdateres.
      ElementsInGPSBuffer--;           // Antallet af tegn i bufferen opdateres.
      if (GPSBufferRP>=GPSBufSize) GPSBufferRP=0;  // Start forfra i buffer.
      if((*l==36)&&(NotFinish))        // Hvis '$' og ikke <CR><LF>.
      {
         k=l;
         *(k+1)=0;                     // For en sikkerheds skyld afsluttes med 0.
         NotFinish=0;                  // Framen er ikke færdig.
      }
      if(((*k)==36)&&((*(k+1))==71)&&((*(k+2))==80)&&((*(k+3))==82)&&((*(k+4))==77)
      &&((*(k+5))==67)&&(!FrameOK))    // Hvis framen indeholder '$GPRMC' er den OK
      {
         FrameOK=1;
      }
      if ((FrameOK)&&(*(l-1)==13)&&(*l==10)) // Hvis framen er OK og afsluttet
      {                                      // med <CR><LF>.
         *(l+1)=0;                     // For en sikkerheds skyld afsluttes med 0.
         *DataLength=72;               // Fast længde på GPS protokol.
         return 1;                     // Der returneres 1 når framen er OK.
      }
      if (*k==36) l++;                 // Hvis '$' i framen indlæses næste tegn.
   }
   return 0;                           // Der returneres 0 når framen ikke er OK.
}

//*****************************************************************************//
// Denne funktion håndterer styring og indskrivning af data i ringbufferen VHFBuffer.  //
//*****************************************************************************//
void WriteToVHFBuffer(int *Data)
{
   VHFBuffer[VHFBufferWP]=*Data;       // Data indskrives i buffer.
   VHFBufferWP++;                      // Writepointer opdateres.
   ElementsInVHFBuffer++;              // Antallet af bit i buffer opdateres.
   if (VHFBufferWP>=VHFBufSize) VHFBufferWP=0;  // Start forfra i buffer.
   if (VHFBufferWP==0)                 // Kontrollerer for stopflag i bufferen ved WP=0.
   {
      if ((VHFBuffer[VHFBufSize-8]==0)&&(VHFBuffer[VHFBufSize-7]==1)
         &&(VHFBuffer[VHFBufSize-6]==1)&&(VHFBuffer[VHFBufSize-5]==1)
         &&(VHFBuffer[VHFBufSize-4]==1)&&(VHFBuffer[VHFBufSize-3]==1)
        &&(VHFBuffer[VHFBufSize-2]==1)&&(VHFBuffer[VHFBufSize-1]==0)) DataInVHFBuffer++;
   }
   else if (VHFBufferWP==1)            // Kontrollerer for stopflag i bufferen ved WP=1.
   {
      if ((VHFBuffer[VHFBufSize-7]==0)&&(VHFBuffer[VHFBufSize-6]==1)
         &&(VHFBuffer[VHFBufSize-5]==1)&&(VHFBuffer[VHFBufSize-4]==1)
         &&(VHFBuffer[VHFBufSize-3]==1)&&(VHFBuffer[VHFBufSize-2]==1)
         &&(VHFBuffer[VHFBufSize-1]==1)&&(VHFBuffer[0]==0)) DataInVHFBuffer++;
```

```
   }
   else if (VHFBufferWP==2)           // Kontrollerer for stopflag i bufferen ved WP=2.
   {
      if ((VHFBuffer[VHFBufSize-6]==0)&&(VHFBuffer[VHFBufSize-5]==1)
         &&(VHFBuffer[VHFBufSize-4]==1)&&(VHFBuffer[VHFBufSize-3]==1)
         &&(VHFBuffer[VHFBufSize-2]==1)&&(VHFBuffer[VHFBufSize-1]==1)
         &&(VHFBuffer[0]==1)&&(VHFBuffer[1]==0)) DataInVHFBuffer++;
   }
   else if (VHFBufferWP==3)           // Kontrollerer for stopflag i bufferen ved WP=3.
   {
      if ((VHFBuffer[VHFBufSize-5]==0)&&(VHFBuffer[VHFBufSize-4]==1)
         &&(VHFBuffer[VHFBufSize-3]==1)&&(VHFBuffer[VHFBufSize-2]==1)
         &&(VHFBuffer[VHFBufSize-1]==1)&&(VHFBuffer[0]==1)&&(VHFBuffer[1]==1)
         &&(VHFBuffer[2]==0)) DataInVHFBuffer++;
   }
   else if (VHFBufferWP==4)           // Kontrollerer for stopflag i bufferen ved WP=4.
   {
      if ((VHFBuffer[VHFBufSize-4]==0)&&(VHFBuffer[VHFBufSize-3]==1)
         &&(VHFBuffer[VHFBufSize-2]==1)&&(VHFBuffer[VHFBufSize-1]==1)
         &&(VHFBuffer[0]==1)&&(VHFBuffer[1]==1)&& (VHFBuffer[2]==1)&&(VHFBuffer[3]==0))
            DataInVHFBuffer++;
   }
   else if (VHFBufferWP==5)           // Kontrollerer for stopflag i bufferen ved WP=5.
   {
      if ((VHFBuffer[VHFBufSize-3]==0)&&(VHFBuffer[VHFBufSize-2]==1)
         &&(VHFBuffer[VHFBufSize-1]==1)&&(VHFBuffer[0]==1)&&(VHFBuffer[1]==1)
         &&(VHFBuffer[2]==1)&&(VHFBuffer[3]==1)&&(VHFBuffer[4]==0)) DataInVHFBuffer++;
   }
   else if (VHFBufferWP==6)           // Kontrollerer for stopflag i bufferen ved WP=6.
   {
      if ((VHFBuffer[VHFBufSize-2]==0)&&(VHFBuffer[VHFBufSize-1]==1)&&(VHFBuffer[0]==1)
       &&(VHFBuffer[1]==1)&&(VHFBuffer[2]==1)&&(VHFBuffer[3]==1)&&(VHFBuffer[4]==1)
       &&(VHFBuffer[5]==0)) DataInVHFBuffer++;
   }
   else if (VHFBufferWP==7)           // Kontrollerer for stopflag i bufferen ved WP=7.
   {
      if ((VHFBuffer[VHFBufSize-1]==0)&&(VHFBuffer[0]==1)&&(VHFBuffer[1]==1)
         &&(VHFBuffer[2]==1)&&(VHFBuffer[3]==1)&&(VHFBuffer[4]==1)&&(VHFBuffer[5]==1)
         &&(VHFBuffer[6]==0)) DataInVHFBuffer++;
   }
   else                               // Kontrollerer for stopflag i bufferen ellers.
   {
      if ((VHFBuffer[VHFBufferWP-8]==0)&&(VHFBuffer[VHFBufferWP-7]==1)
        &&(VHFBuffer[VHFBufferWP-6]==1)&&(VHFBuffer[VHFBufferWP-5]==1)
        &&(VHFBuffer[VHFBufferWP-4]==1)&&(VHFBuffer[VHFBufferWP-3]==1)
        &&(VHFBuffer[VHFBufferWP-2]==1)&&(VHFBuffer[VHFBufferWP-1]==0))
         DataInVHFBuffer++;
   }
   if ((VHFBufferWP+1==VHFBufferRP)||((VHFBufferWP==VHFBufSize-1)&&(VHFBufferRP==0)))
   {
      VHFBufferRP++;                  // Hvis bufferen er fuld skal RP flyttes frem.
      if (VHFBufferRP>=VHFBufSize) VHFBufferRP=0;  // Start forfra i buffer.
   }
}

//*****************************************************************************//
// Denne funktion håndterer styring og læsning af data i ringbufferen VHFBuffer.       //
//*****************************************************************************//
int ReadFromVHFBuffer(int Data[])
{
   int Length=0;
   DataInVHFBuffer--;                 // Antallet af bit i buffer opdateres.
   while (VHFBufferWP!=VHFBufferRP)   // Der må ikke læses forbi WP.
   {
      Data[Length]=VHFBuffer[VHFBufferRP];   // Data læses ind fra buffer.
      VHFBufferRP++;                  // Readpointer opdateres.
      ElementsInVHFBuffer--;          // Antallet af bit i bufferen opdateres.
      if (VHFBufferRP>=VHFBufSize) VHFBufferRP=0;  // Start forfra i buffer.
      if (Length>=7)                  // Hvis der er indlæst 8 bit kontrolleres
      {                               // for stopflag.
         if ((Data[Length]==0)&&(Data[Length-1]==1)&&(Data[Length-2]==1)
            &&(Data[Length-3]==1)&&(Data[Length-4]==1)&&(Data[Length-5]==1)
            &&(Data[Length-6]==1)&&(Data[Length-7]==0))
         {
            if (Length<40)            // Hvis stopflag er fundet inden for 40 bit
            {                         // er framen ikke gyldig, og der returneres et 0.
               return 0;
            }                         // Kontrollerer for gyldig frame; dvs frame
            else                      // der starter med '000001','000010','000011',
            {                         // '000101' eller '001000'.
               if (((Data[0]==0)&&(Data[1]==0)&&(Data[2]==0)&&(Data[3]==0)&&(Data[4]==0)
                  &&(Data[5]==1))||((Data[0]==0)&&(Data[1]==0)&&(Data[2]==0)
                  &&(Data[3]==0)&&(Data[4]==1)&&(Data[5]==0))||((Data[0]==0)
                  &&(Data[1]==0)&&(Data[2]==0)&&(Data[3]==0)&&(Data[4]==1)
                  &&(Data[5]==1))||((Data[0]==0)&&(Data[1]==0)&&(Data[2]==0)
                  &&(Data[3]==1)&&(Data[4]==0)&&(Data[5]==1))||((Data[0]==0)
```

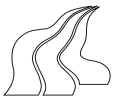


```
                  &&(Data[1]==0)&&(Data[2]==1)&&(Data[3]==0)&&(Data[4]==0)
                  &&(Data[5]==0)))
                  {
//                   EX0=1;            // Enable extern interrupt til VHF.
                     return Length-8;  // Hvis OK returneres længden af framen minus
                  }                    // længden af stopflag ellers returneres 0.
               else
                  {
//                   EX0=1;            // Enable extern interrupt til VHF.
                     return 0;
                  }
            }
         }
      }
      Length++;
   }
   return 0;
}

//*****************************************************************************//
// Denne funktion håndterer styring og indskrivning af data i ringbufferen PCBuffer.   //
//*****************************************************************************//
void WriteToPCBuffer(int Data[],int *Length)
{
   int x;
   for (x=0;x<=*Length;x++)            // Hele framen skal flyttes over i bufferen.
   {
      PCBuffer[PCBufferWP]=Data[x];    // Data indskrives i bufferen.
      PCBufferWP++;                    // Writepointer opdateres.
      ElementsInPCBuffer++;            // Antallet af tegn i bufferen opdateres.
      if (PCBufferWP>=PCBufSize) PCBufferWP=0;  // Start forfra i buffer WP.
      if ((PCBufferWP+1==PCBufferRP) || ((PCBufferWP==PCBufSize-1) && PCBufferRP==0))
      {
         PCBufferRP++;                 // Hvis bufferen er fuld skal RP flyttes frem.
         if (PCBufferRP>=PCBufSize) PCBufferRP=0;  // Start forfra i buffer RP.
      }
   }
}

//*****************************************************************************//
// Denne funktion håndterer styring og læsning af data i ringbufferen PCBuffer.        //
//*****************************************************************************//
int ReadFromPCBuffer(int *Data)
{
   if (PCBufferWP!=PCBufferRP)         // Der må ikke læses forbi WP.
   {
      *Data=PCBuffer[PCBufferRP];      // Data læses ind fra buffer.
      PCBufferRP++;                    // Readpointer opdateres.
      ElementsInPCBuffer--;            // Antallet af tegn i bufferen opdateres.
      if (PCBufferRP>=PCBufSize) PCBufferRP=0;  // Start forfra i buffer.
      return 1;                        // Der returneres 1 hvis der er data med ud
   }                                   // fra funktionen.
   else return 0;                      // Der returneres 0 når bufferen er tom.
}

//*****************************************************************************//
// Denne funktion fjerner de ekstra bit der er indsat for at undgå misfortolkning af   //
// start- og stopflag.                                                                 //
// Hvis der findes 5 efterfølgende '1' springes over den efterfølgende bit.            //
//*****************************************************************************//
void BitUnStuffing(int Data[],int *DataLength)
{
   int k=0,l=4,m=0,BitCounter=0;
   int Data1[1200];
   for (m=0;m<=3;m++) Data1[m]=Data[m];// De første 4 bit flyttes uden check,
   for (k=4;k<=*DataLength;k++)        // da der ikke kan  være indsat ekstra bit her.
   {                                   // Data løbes gennem for 5 efterfølgende '1'.
      Data1[l]=Data[k];                // Data flyttes over i lokal array.
      if ((Data[k-4]==1)&&(Data[k-3]==1)&&(Data[k-2]==1)&&(Data[k-1]==1)&&(Data[k]==1))
      {                                // Hvis 5 efterfølgende '1'
         k++;                          // springes over den efterfølgende bit og de
         for (m=0;m<=3;m++)            // efterfølgende 4 bit flyttes direkte
         {                             // over i det nye array.
            k++;l++;
            Data1[l]=Data[k];
         }
         BitCounter++;                 // Antallet af ekstra bit opdateres.
      }
      l++;
   }
   *DataLength=*DataLength-BitCounter; // Den nye datalængde returneres.
   for (m=0;m<=*DataLength;m++) Data[m]=Data1[m];  // Data flyttes tilbage i
}                                                  // oprindeligt array.

//*****************************************************************************//
// Denne funktion kontrollerer om CRC på beskeden fra GPS'en er OK                     //
```

```
//*****************************************************************************//
int GPSCRCCheck(int Data[],int *DataLength)
{
   int x,CRCHIGH,CRCLOW,CRC=0;
   CRC=Data[1];                        // Der startes i 1 og sluttes i datalængde-5
   for (x=2;x<=(*DataLength-5);x++)    // for at undgå at '$', '*' og <CR><LF> tages
   {                                   // med i CRC beregningen.
   CRC=CRC^Data[x];                    // Bitvis XOR.
   }
   CRCHIGH=CRC>>4;                     // 4 bit MSB gemmes i CRCHIGH.
   CRCLOW=CRC&0x0F;                    // 4 bit LSB gemmes i CRCLOW.
   if ((CRCHIGH>=0) && (CRCHIGH<=9)) CRCHIGH=CRCHIGH+48;
   else CRCHIGH=CRCHIGH+55;            // Konvertering fra hexadecimal til decimal.
   if ((CRCLOW>=0) && (CRCLOW<=9)) CRCLOW=CRCLOW+48;
   else CRCLOW=CRCLOW+55;              // Konvertering fra hexadecimal til decimal.
   if ((CRCHIGH==Data[*DataLength-3])&&(CRCLOW==Data[*DataLength-2]))
      return 1;                        // Kontroller om CRC er OK og returner 1 hvis OK
   else return 0;                      // og 0 ellers.
}

//*****************************************************************************//
// Denne funktion kontrollerer om CRC på beskeden fra VHF'en er OK             //
//*****************************************************************************//
int VHFCRCCheck(int Data[],int *DataLength)
{
   int x,i,CRCData,CRC=0,Length=0;
   int DataTemp[200];
   for (x=0;x<=(*DataLength/8);x++)    // Der indlæses en byte ad gangen.
   {                                   // Omregn fra binær til titalsystem.
      DataTemp[x]=(Data[x*8]*128+Data[(x*8)+1]*64+Data[(x*8)+2]*32+Data[(x*8)+3]*16+
                  Data[(x*8)+4]*8+Data[(x*8)+5]*4+Data[(x*8)+6]*2+Data[(x*8)+7]*1);
      Length++;
   }
   for (x=0;x<=(Length-1);x++)         // Der sluttes i datalængde-1 for at undgå at
   {                                   // stopflag tages med i CRC beregningen.
      CRCData=DataTemp[x];             // Data flyttes fra array over i CRCData.
      if (x<=1) CRCData=255-CRCData;   // De første 16 bit inverteres.
      CRCData<<=8;                     // Data Leftshiftes 8 gange.
      for(i=0;i<8;i++)                 // CRC beregnes bit for bit.
      {
         if ((CRC^CRCData) & 0x8000)   // CRC XOR med CRCData og hvis '1' på første plads,
            CRC=(CRC<<1)^CCITT;        // leftshiftes CRC 1 gang og XOR med CRC masken
         else CRC<<=1;                 // ellers Leftshiftes CRC 1 gang
         CRCData<<=1;                  // CRCData Leftshiftes 1 gang.
      }
   }
   if (CRC==0x1D0F) return 1;          // Hvis CRC er OK returneres 1
   else return 0;                      // ellers returneres 0.
}

//*****************************************************************************//
// Denne funktion laver GPS-beskeden om, således at det kun indeholder de      //
// informationer, der skal sendes videre til PC'en.                            //
// Beskeden indledes med $GPRMC, og hver delbesked er adskilt med et ','       //
//*****************************************************************************//
void NewGPSProtokol(int Data[])
{
   int k;
   for (k=17;k<=57;k++)
   {
      Data[k]=Data[k+2];               // Fjern status.
   }
   Data[58]=42;                        // Indsæt '*'.
   Data[59]=Data[60]='0';              // Lav plads til checksum.
   Data[61]=13;                        // Afslut med <CR><LF>.
   Data[62]=10;
}

//*****************************************************************************//
// Denne funktion laver VHF-beskeden om, således at det kun indeholder de      //
// informationer, der skal sendes videre til PC'en.                            //
// Endvidere laves beskeden om fra binær repræsentation til tegnbaseret        //
// repræsentation.                                                             //
// Beskeden indledes med $CVN0 + besked nummer, og hver delbesked er adskilt med et ','//
//*****************************************************************************//
void NewVHFProtokol(int Data[],int *DataLength)
{
   long DataTemp=0;
   char DataString[12];
   int DataCounter=7,Counter=0;
   if ((Data[2]==0)&&(Data[3]==0))     // Kontroller beskedtype '00' = 1,2 og 3.
   {
      Data[5]='1';                     // Besked nummer 1,2 eller 3 modtaget.
      Data[6]=',';                     // MMSI NUMMER.
      DataTemp=(Data[8]*536870912+Data[9]*268435456+Data[10]*134217728+Data[11]*67108864
               +Data[12]*33554432+Data[13]*16777216+Data[14]*8388608+Data[15]*4194304
```

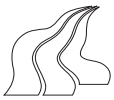
```
         +Data[16]*2097152+Data[17]*1048576+Data[18]*524288+Data[19]*262144
         +Data[20]*131072+Data[21]*65536+Data[22]*32768+Data[23]*16384
         +Data[24]*8192+Data[25]*4096+Data[26]*2048+Data[27]*1024+Data[28]*512
         +Data[29]*256+Data[30]*128+Data[31]*64+Data[32]*32+Data[33]*16+Data[34]*8
         +Data[35]*4+Data[36]*2+Data[37]*1);
sprintf(DataString,"%lu",DataTemp); // Konverter fra integer til streng.
while (DataString[Counter]!=0)   // Skriv ny data ind i array.
{
   Data[DataCounter]=DataString[Counter];
   DataCounter++;
   Counter++;
}
Data[DataCounter]=',';           // NAVIGATIONAL STATUS.
DataCounter++;                   // Omregn fra binær til titalsystem.
if ((Data[38]==0)&&(Data[39]==0)) Data[DataCounter]='0';
else if ((Data[38]==0)&&(Data[39]==1)) Data[DataCounter]='1';
else if ((Data[38]==1)&&(Data[39]==0)) Data[DataCounter]='2';
else if ((Data[38]==1)&&(Data[39]==1)) Data[DataCounter]='3';
DataCounter++;
Data[DataCounter]=',';           // RATE OF TURN.
DataCounter++;                   // Omregn fra binær til titalsystem.
DataTemp=(Data[41]*64+Data[42]*32+Data[43]*16+Data[44]*8+Data[45]*4+Data[46]*2
         +Data[47]*1);
if (Data[40]==1)                 // Positiv eller negativ tal ?
{
   Data[DataCounter]='-';
   DataCounter++;
   DataTemp=128-DataTemp;        // Omregn til negativ tal.
}
sprintf(DataString,"%lu",DataTemp); // Konverter fra integer til streng.
Counter=0;
while (DataString[Counter]!=0)   // Skriv ny data ind i array.
{
   Data[DataCounter]=DataString[Counter];
   DataCounter++;
   Counter++;
}
Data[DataCounter]=',';           // SPEED OVER GROUND.
DataCounter++;                   // Omregn fra binær til titalsystem.
DataTemp=(Data[48]*512+Data[49]*256+Data[50]*128+Data[51]*64+Data[52]*32
         +Data[53]*16+Data[54]*8+Data[55]*4+Data[56]*2+Data[57]*1);
sprintf(DataString,"%lu",DataTemp); // Konverter fra integer til streng.
Counter=0;
while (DataString[Counter]!=0)   // Skriv ny data ind i array.
{
   Data[DataCounter]=DataString[Counter];
   DataCounter++;
   Counter++;
}
Data[DataCounter]=',';           // POSITION ACCURACY.
DataCounter++;
if (Data[58]==0) Data[DataCounter]='0';// '0' = mindre end 10 m.
else Data[DataCounter]='1';            // '1' = større end 10 m.
DataCounter++;
Data[DataCounter]=',';           // LONGITUDE.
DataCounter++;                   // Omregn fra binær til titalsystem.
DataTemp=(Data[60]*67108864+Data[61]*33554432+Data[62]*16777216+Data[63]*8388608
         +Data[64]*4194304+Data[65]*2097152+Data[66]*1048576+Data[67]*524288
         +Data[68]*262144+Data[69]*131072+Data[70]*65536+Data[71]*32768
         +Data[72]*16384+Data[73]*8192+Data[74]*4096+Data[75]*2048+Data[76]*1024
         +Data[77]*512+Data[78]*256+Data[79]*128+Data[80]*64+Data[81]*32
         +Data[82]*16+Data[83]*8+Data[84]*4+Data[85]*2+Data[86]*1);
if (Data[59]==1)                 // Positiv eller negativ tal ?
{
   Data[DataCounter]='-';        // Negativ tal.
   DataCounter++;
   DataTemp=134217728-DataTemp;
}
sprintf(DataString,"%lu",DataTemp); // Konverter fra integer til streng.
Counter=0;
while (DataString[Counter]!=0)   // Skriv ny data ind i array.
{
   Data[DataCounter]=DataString[Counter];
   DataCounter++;
   Counter++;
}
Data[DataCounter]=',';           // LATITUDE.
DataCounter++;                   // Omregn fra binær til titalsystem.
DataTemp=(Data[88]*33554432+Data[89]*16777216+Data[90]*8388608+Data[91]*4194304
         +Data[92]*2097152+Data[93]*1048576+Data[94]*524288+Data[95]*262144
         +Data[96]*131072+Data[97]*65536+Data[98]*32768+Data[99]*16384
         +Data[100]*8192+Data[101]*4096+Data[102]*2048+Data[103]*1024+Data[104]*512
         +Data[105]*256+Data[106]*128+Data[107]*64+Data[108]*32+Data[109]*16
         +Data[110]*8+Data[111]*4+Data[112]*2+Data[113]*1);
if (Data[87]==1)                 // Positiv eller negativ tal ?
{
```

```
      Data[DataCounter]='-';        // Negativ tal.
      DataCounter++;
      DataTemp=67108864-DataTemp;
   }
   sprintf(DataString,"%lu",DataTemp); // Konverter fra integer til streng.
   Counter=0;
   while (DataString[Counter]!=0)   // Skriv ny data ind i array.
   {
      Data[DataCounter]=DataString[Counter];
      DataCounter++;
      Counter++;
   }
   Data[DataCounter]=',';           // COURSE OVER GROUND.
   DataCounter++;                   // Omregn fra binær til titalsystem.
   DataTemp=(Data[114]*2048+Data[115]*1024+Data[116]*512+Data[117]*256+Data[118]*128
            +Data[119]*64+Data[120]*32+Data[121]*16+Data[122]*8+Data[123]*4
            +Data[124]*2+Data[125]*1);
   sprintf(DataString,"%lu",DataTemp); // Konverter fra integer til streng.
   Counter=0;
   while (DataString[Counter]!=0)   // Skriv ny data ind i array.
   {
      Data[DataCounter]=DataString[Counter];
      DataCounter++;
      Counter++;
   }
   Data[DataCounter]=',';           // HEADING.
   DataCounter++;                   // Omregn fra binær til titalsystem.
   DataTemp=(Data[126]*256+Data[127]*128+Data[128]*64+Data[129]*32+Data[130]*16
            +Data[131]*8+Data[132]*4+Data[133]*2+Data[134]*1);
   sprintf(DataString,"%lu",DataTemp); // Konverter fra integer til streng
   Counter=0;
   while (DataString[Counter]!=0)   // Skriv ny data ind i array.
   {
      Data[DataCounter]=DataString[Counter];
      DataCounter++;
      Counter++;
   }
   Data[DataCounter]=',';           // TIME STAMP.
   DataCounter++;                   // Omregn fra binær til titalsystem.
   DataTemp=(Data[135]*32+Data[136]*16+Data[137]*8+Data[138]*4+Data[139]*2
            +Data[140]*1);
   sprintf(DataString,"%lu",DataTemp); // Konverter fra integer til streng.
   Counter=0;
   while (DataString[Counter]!=0)   // Skriv ny data ind i array.
   {
      Data[DataCounter]=DataString[Counter];
      DataCounter++;
      Counter++;
   }
}
if ((Data[2]==0)&&(Data[3]==1))     // Kontroller beskedtype '01' = 5.
{
   Data[5]='5';                     // Besked nummer 5 modtaget.
   Data[6]=',';                     // MMSI NUMMER.
   DataTemp=(Data[8]*536870912+Data[9]*268435456+Data[10]*134217728+Data[11]*67108864
            +Data[12]*33554432+Data[13]*16777216+Data[14]*8388608+Data[15]*4194304
            +Data[16]*2097152+Data[17]*1048576+Data[18]*524288+Data[19]*262144
            +Data[20]*131072+Data[21]*65536+Data[22]*32768+Data[23]*16384
            +Data[24]*8192+Data[25]*4096+Data[26]*2048+Data[27]*1024+Data[28]*512
            +Data[29]*256+Data[30]*128+Data[31]*64+Data[32]*32+Data[33]*16+Data[34]*8
            +Data[35]*4+Data[36]*2+Data[37]*1);
   sprintf(DataString,"%lu",DataTemp); // Konverter fra integer til streng.
   while (DataString[Counter]!=0)   // Skriv ny data ind i array.
   {
      Data[DataCounter]=DataString[Counter];
      DataCounter++;
      Counter++;
   }
   Data[DataCounter]=',';           // IMO NUMBER.
   DataCounter++;                   // Omregn fra binær til titalsystem.
   DataTemp=(Data[40]*536870912+Data[41]*268435456+Data[42]*134217728
            +Data[43]*67108864+Data[44]*33554432+Data[45]*16777216+Data[46]*8388608
            +Data[47]*4194304+Data[48]*2097152+Data[49]*1048576+Data[50]*524288
            +Data[51]*262144+Data[52]*131072+Data[53]*65536+Data[54]*32768
            +Data[55]*16384+Data[56]*8192+Data[57]*4096+Data[58]*2048+Data[59]*1024
            +Data[60]*512+Data[61]*256+Data[62]*128+Data[63]*64+Data[64]*32
            +Data[65]*16+Data[66]*8+Data[67]*4+Data[68]*2+Data[69]*1);
   sprintf(DataString,"%lu",DataTemp); // Konverter fra integer til streng.
   Counter=0;
   while (DataString[Counter]!=0)   // Skriv ny data ind i array.
   {
      Data[DataCounter]=DataString[Counter];
      DataCounter++;
      Counter++;
   }
   Data[DataCounter]=',';           // CALL SIGN.
```



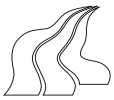
```
DataCounter++;                  // Læs en 6 bit char og konverter til 7 bit char.
for (Counter=0;Counter<=5;Counter++)
{
   DataTemp=(Data[70+(6*Counter)]*32+Data[71+(6*Counter)]*16+Data[72+(6*Counter)]*8
            +Data[73+(6*Counter)]*4+Data[74+(6*Counter)]*2+Data[75+(6*Counter)]*1);
   Data[DataCounter]=DataTemp+32;   // Skriv ny data ind i array
   DataCounter++;
}
Data[DataCounter]=',';          // NAME.
DataCounter++;                  // Læs en 6 bit char og konverter til 7 bit char.
for (Counter=0;Counter<=19;Counter++)
{
   DataTemp=(Data[106+(6*Counter)]*32+Data[107+(6*Counter)]*16
            +Data[108+(6*Counter)]*8+Data[109+(6*Counter)]*4
            +Data[110+(6*Counter)]*2+Data[111+(6*Counter)]*1);
   Data[DataCounter]=DataTemp+32;   // Skriv ny data ind i array.
   DataCounter++;
}
Data[DataCounter]=',';          // TYPE OF SHIP.
DataCounter++;                  // Omregn fra binær til titalsystem.
DataTemp=(Data[226]*128+Data[227]*64+Data[228]*32+Data[229]*16+Data[230]*8
        +Data[231]*4+Data[232]*2+Data[233]*1);
sprintf(DataString,"%lu",DataTemp); // Konverter fra integer til streng.
Counter=0;
while (DataString[Counter]!=0)   // Skriv ny data ind i array.
{
   Data[DataCounter]=DataString[Counter];
   DataCounter++;
   Counter++;
}
Data[DataCounter]=',';          // POSITION OF GNSS A.
DataCounter++;                  // Omregn fra binær til titalsystem.
DataTemp=(Data[234]*256+Data[235]*128+Data[236]*64+Data[237]*32+Data[238]*16
        +Data[239]*8+Data[240]*4+Data[241]*2+Data[242]*1);
sprintf(DataString,"%lu",DataTemp); // Konverter fra integer til streng.
Counter=0;
while (DataString[Counter]!=0)   // Skriv ny data ind i array.
{
   Data[DataCounter]=DataString[Counter];
   DataCounter++;
   Counter++;
}
Data[DataCounter]=',';          // POSITION OF GNSS B.
DataCounter++;                  // Omregn fra binær til titalsystem.
DataTemp=(Data[243]*256+Data[244]*128+Data[245]*64+Data[246]*32+Data[247]*16
        +Data[248]*8+Data[249]*4+Data[250]*2+Data[251]*1);
sprintf(DataString,"%lu",DataTemp); // Konverter fra integer til streng.
Counter=0;
while (DataString[Counter]!=0)   // Skriv ny data ind i array.
{
   Data[DataCounter]=DataString[Counter];
   DataCounter++;
   Counter++;
}
Data[DataCounter]=',';          // POSITION OF GNSS C.
DataCounter++;                  // Omregn fra binær til titalsystem.
DataTemp=(Data[252]*32+Data[253]*16+Data[254]*8+Data[255]*4+Data[256]*2
        +Data[257]*1);
sprintf(DataString,"%lu",DataTemp);
Counter=0;
while (DataString[Counter]!=0)   // Skriv ny data ind i array.
{
   Data[DataCounter]=DataString[Counter];
   DataCounter++;
   Counter++;
}
Data[DataCounter]=',';          // POSITION OF GNSS D.
DataCounter++;                  // Omregn fra binær til titalsystem.
DataTemp=(Data[258]*32+Data[259]*16+Data[260]*8+Data[261]*4+Data[262]*2
        +Data[263]*1);
sprintf(DataString,"%lu",DataTemp); // Konverter fra integer til streng.
Counter=0;
while (DataString[Counter]!=0)   // Skriv ny data ind i array.
{
   Data[DataCounter]=DataString[Counter];
   DataCounter++;
   Counter++;
}
Data[DataCounter]=',';          // TYPE OF NAVIGATION SENSOR.
DataCounter++;                  // Omregn fra binær til titalsystem.
DataTemp=(Data[264]*8+Data[265]*4+Data[266]*2+Data[267]*1);
sprintf(DataString,"%lu",DataTemp); // Konverter fra integer til streng.
Counter=0;
while (DataString[Counter]!=0)   // Skriv ny data ind i array.
{
   Data[DataCounter]=DataString[Counter];
```

```
      DataCounter++;
      Counter++;
   }
   Data[DataCounter]=',';           // ESTIMATED TIME OF ARRIVAL MONTH.
   DataCounter++;                   // Omregn fra binær til titalsystem.
   DataTemp=(Data[268]*8+Data[269]*4+Data[270]*2+Data[271]*1);
   sprintf(DataString,"%2lu",DataTemp);   // Konverter fra binær til titalsystem.
   Counter=0;
   while (DataString[Counter]!=0)   // Skriv ny data ind i array.
   {
      Data[DataCounter]=DataString[Counter];
      DataCounter++;
      Counter++;                    // ESTIMATED TIME OF ARRIVAL DAY.
   }                                // Omregn fra binær til titalsystem.
   DataTemp=(Data[272]*16+Data[273]*8+Data[274]*4+Data[275]*2+Data[276]*1);
   sprintf(DataString,"%2lu",DataTemp);   // Konverter fra integer til streng.
   Counter=0;
   while (DataString[Counter]!=0)   // Skriv ny data ind i array.
   {
      Data[DataCounter]=DataString[Counter];
      DataCounter++;
      Counter++;                    // ESTIMATED TIME OF ARRIVAL HOUR.
   }                                // Omregn fra binær til titalsystem.
   DataTemp=(Data[277]*16+Data[278]*8+Data[279]*4+Data[280]*2+Data[281]*1);
   sprintf(DataString,"%2lu",DataTemp);   // Konverter fra integer til streng.
   Counter=0;
   while (DataString[Counter]!=0)   // Skriv ny data ind i streng.
   {
      Data[DataCounter]=DataString[Counter];
      DataCounter++;
      Counter++;                    // ESTIMATED TIME OF ARRIVAL MINUTE.
   }                                // Omregn fra binær til titalsystem.
   DataTemp=(Data[282]*32+Data[283]*16+Data[284]*8+Data[285]*4+Data[286]*2
            +Data[287]*1);
   sprintf(DataString,"%2lu",DataTemp);   // Konverter fra integer til streng.
   Counter=0;
   while (DataString[Counter]!=0)   // Skriv ny data ind i array.
   {
      Data[DataCounter]=DataString[Counter];
      DataCounter++;
      Counter++;
   }
   Data[DataCounter]=',';           // ACTUAL DRAUGHT.
   DataCounter++;                   // Omregn fra binær til titalsystem.
   DataTemp=(Data[288]*128+Data[289]*64+Data[290]*32+Data[291]*16+Data[292]*8
            +Data[293]*4+Data[294]*2+Data[295]*1);
   sprintf(DataString,"%lu",DataTemp); // Konverter fra integer til streng.
   Counter=0;
   while (DataString[Counter]!=0)   // Skriv ny data ind i array.
   {
      Data[DataCounter]=DataString[Counter];
      DataCounter++;
      Counter++;
   }
   Data[DataCounter]=',';           // DESTINATION.
   DataCounter++;                   // Læs en 6 bit char og konverter til 7 bit char.
   for (Counter=0;Counter<=19;Counter++)
   {
      DataTemp=(Data[296+(6*Counter)]*32+Data[297+(6*Counter)]*16
               +Data[298+(6*Counter)]*8+Data[299+(6*Counter)]*4
               +Data[300+(6*Counter)]*2+Data[301+(6*Counter)]*1);
      Data[DataCounter]=DataTemp+32;   // Skriv ny data ind i array.
      DataCounter++;
   }
}
if ((Data[2]==1)&&(Data[3]==0))     // Kontroller beskedtype '10' = 8.
{
   Data[5]='8';                     // Besked nummer 8 modtaget.
   Data[6]=',';                     // MMSI NUMMER.
   DataTemp=(Data[8]*536870912+Data[9]*268435456+Data[10]*134217728+Data[11]*67108864
            +Data[12]*33554432+Data[13]*16777216+Data[14]*8388608+Data[15]*4194304
            +Data[16]*2097152+Data[17]*1048576+Data[18]*524288+Data[19]*262144
            +Data[20]*131072+Data[21]*65536+Data[22]*32768+Data[23]*16384
            +Data[24]*8192+Data[25]*4096+Data[26]*2048+Data[27]*1024+Data[28]*512
            +Data[29]*256+Data[30]*128+Data[31]*64+Data[32]*32+Data[33]*16+Data[34]*8
            +Data[35]*4+Data[36]*2+Data[37]*1);
   sprintf(DataString,"%lu",DataTemp); // Konverter fra integer til streng.
   while (DataString[Counter]!=0)   // Skriv ny data ind i array.
   {
      Data[DataCounter]=DataString[Counter];
      DataCounter++;
      Counter++;
   }
   Data[DataCounter]=',';           // DATA.
   DataCounter++;                   // Læs en 8 bit char og konverter til 7 bit char.
   for (Counter=0;Counter<=120;Counter++)
```



```
      {
         DataTemp=(Data[40+(8*Counter)]*128+Data[41+(8*Counter)]*64
                  +Data[42+(8*Counter)]*32+Data[43+(8*Counter)]*16+Data[44+(8*Counter)]*8
                  +Data[45+(8*Counter)]*4)+Data[46+(8*Counter)]*2+Data[47+(8*Counter)]*1;
         if (DataTemp)                 // Hvis der er modtaget en tegn skrives
         {                             // den ind i array.
            Data[DataCounter]=DataTemp;
            DataCounter++;
         }
      }
   }
   Data[0]='$';                        // Den faste startsekvens tilføjes til sidst
   Data[1]='C';                        // for at undgå at de originale bit overskrives.
   Data[2]='V';
   Data[3]='N';
   Data[4]='0';
   Data[DataCounter]='*';              // Der afsluttes med stjerne, plads til
   DataCounter++;                      // 16 bit CRC og <CR><LF>.
   Data[DataCounter]='0';
   DataCounter++;
   Data[DataCounter]='0';
   DataCounter++;
   Data[DataCounter]=13;
   DataCounter++;
   Data[DataCounter]=10;
   *DataLength=DataCounter;
}

//*****************************************************************************//
// Denne funktion beregner CRC.                                                //
//*****************************************************************************//
void CalculateCRC(int Data[],int *DataLength)
{
   int x,i,CRCData,CRC=0;              // Der startes i 1 og sluttes i datalængde-5
   for (x=1;x<=(*DataLength-5);x++)    // for at sikre at kun gyldig data
   {                                   // tages med i CRC beregningen.
      CRCData=Data[x];                 // Data flyttes fra array over i CRCData.
      if (x<=2) CRCData=255-CRCData;   // De første 16 bit inverteres.
      CRCData<<=8;                     // Data Leftshiftes 8 gange.
      for(i=0;i<8;i++)                 // CRC beregnes bit for bit.
      {
         if((CRC^CRCData) & 0x8000)    // CRC XOR med CRCData og hvis '1' på første plads
            CRC=(CRC<<1)^CCITT;        // Leftshiftes CRC 1 gang og XOR med CRC masken
         else CRC<<=1;                 // ellers Leftshiftes CRC 1 gang.
         CRCData<<=1;                  // CRCData Leftshiftes 1 gang.
      }
   }
   CRC=65535-CRC;                      // CRC inverteres.
   Data[*DataLength-3]=(CRC>>8)&0xff;  // Første CRC byte beregnes.
   Data[*DataLength-2]=CRC&0xff;       // Anden CRC byte beregnes.
}

//*****************************************************************************//
// Denne funktion læser tegn fra PCBuffer og sender dem via serielport 0 til PC.//
// Denne funktion kaldes via interrupt, hver gang der er afsendt en tegn,      //
// indtil TIO flaget resetes.                                                  //
// Denne interruptvektor ligger på adresse 0x0023 svarende til "Serial 0 interrupt".//
// Der benyttes interruptnummer 4, samt register-bank 3 som stak.              //
//*****************************************************************************//
void seriel0_Interrupt(void) interrupt 4 using 3
{
   int Data;
   if (ReadFromPCBuffer(&Data))        // Der læses en tegn fra PCBuffer.
      S0BUF=Data;                      // Tegnet skrives til serielport 0.
   TI0=0;                              // Interruptflaget resetes.
}

//*****************************************************************************//
// Denne funktion er interruptroutinen for modtagelse af data på seriel 1 (GPS).//
// Denne interruptvektor ligger på adresse 0x0083 svarende til "Serial 1 interrupt".//
// der benyttes interruptnummer 16, samt register-bank 2 som stak.             //
//*****************************************************************************//
// Denne funktion er modificeret således at både GPS- og VHF-beskeder modtages på//
// serielport 1.                                                               //
//*****************************************************************************//
void seriel1_Interrupt(void) interrupt 16 using 2
{
   int SerData;
   SerData=S1BUF;                      // Hent data fra UART.
   if (SerData > 2) WriteToGPSBuffer(&SerData); // Hvis GPS-besked skriv i GPSBuffer.
   else
   {
      SerData=SerData-1;
      WriteToVHFBuffer(&SerData);      // Hvis VHF-besked skriv i VHFBuffer.
   }
   S1CON&=0xFE;                        // Receive interrupt flaget i S1CON lægges ned.
```

```
}

//*****************************************************************************//
// Denne funktion er interruptroutinen for modtagelse af data på Port 3 (VHF).  //
// Denne interruptvektor ligger på adresse 0x0003 svarende til "External interrupt 0". //
// Der benyttes interruptnummer 0, samt register-bank 1 som stak.               //
//*****************************************************************************//
/*void VHF_Interrupt(void) interrupt 0 using 1
{
   EX0=0;                           // Disabler externt interrupt.
   if (Sync==0)
   {
      ET0=1;                        // Enable timer0 interrupt.
      TR0=1;
      while(!INT0)                  // Looper så længe  port 3.2 = 0.
      {
      }
      while(INT0)                   // Hvis port 3.2 = 1, aktiveres timeren
      {}                            // som tæller '1'.
      TH0 = 256-TL0;
      TL0 = 231;
      TMOD &= 0xf0;
      TMOD |= 0x02;
      Sync=1;
   }
   else
   {
      TL0 = 240;
   }
}*/


//*****************************************************************************//
// Denne funktion er interruptroutinen for timer 0.                             //
// Denne interruptvektor ligger på adresse 0x000B svarende til "Timer 0 Overflow".  //
// Der benyttes interruptnummer 1, samt register-bank 2 som stak.               //
//*****************************************************************************//
/*void Timer0_interrupt(void) interrupt 1 using 2
{
   int ParData;
   if (TrainSequenceTest>6)                 // Der undersøges om der har været
   {                                        // seks trainingssekvenser.
      ParData=INT0;                         // Hvis det er tilfældet gemmes
      WriteToVHFBuffer(&ParData);                   // data i VHFBuffer.
      SyncCounter++;
      if (SyncCounter>20)                           // Hvis der er aflæst 20 bit
      {                                     // synkroniseres igen.
         SyncCounter=0;
         EX0=1;                                     // Ekstern interrupt enables.
      }
   }

// I det nedenstående undersøges om det er en trainingssekvens, der synkroniseres på.
// Dette gøres ved at konstatere, om der har været følgende sekvens “010101".

   else if (((TrainSequenceTest==0)||(TrainSequenceTest==2)||(TrainSequenceTest==4)
            ||(TrainSequenceTest==6))&&(!INT0))
      {
         TrainSequenceTest++;
      }
   else if (((TrainSequenceTest==1)||(TrainSequenceTest==3)||(TrainSequenceTest==5))
             &&(INT0))
      {
         TrainSequenceTest++;
      }
   else                                     // Hvis der ikke er sekvensen
      {                                     // “010101" synkroniseres igen.
         TrainSequenceTest=0;
         Sync=0;
         EX0=1;
      }
}*/

//*****************************************************************************//
// Denne funktion, som er en uendelig løkke, skal kalde alle de ovenstående funktioner.//
//*****************************************************************************//
void main (void)
{
   int GPSData[100],VHFData[2000],DataLength=0;
   Init();
   while (1)
   {
      if (DataInGPSBuffer)                          // Hvis der er data i GPSBuffer.
      {
         if (ReadFromGPSBuffer(GPSData,&DataLength))  // Læs fra GPSBuffer.
         {
```



```
            if (GPSCRCCheck(GPSData,&DataLength))     // Kontroller om CRC er OK.
            {
               NewGPSProtokol(GPSData);               // Lav ny protokol.
               DataLength=62;                         // Fast længde på ny protokol.
               CalculateCRC(GPSData,&DataLength);     // Beregn ny CRC.
               WriteToPCBuffer(GPSData,&DataLength);  // Data skrives til PCBuffer
               TI0=1;                                 // og interrupt flaget sættes.
            }
         }
      }
      if (DataInVHFBuffer)                            // Hvis der er data i VHFBuffer.
      {
         DataLength=ReadFromVHFBuffer(VHFData);       // Læs fra VHFBuffer.
         if (DataLength)                              // Hvis længde > 0.
         {
            BitUnStuffing(VHFData,&DataLength);       // Fjern ekstra bit.
            if (VHFCRCCheck(VHFData,&DataLength))     // Kontroller om CRC er OK.
            {
               NewVHFProtokol(VHFData,&DataLength);   // Lav ny protokol.
               CalculateCRC(VHFData,&DataLength);     // Beregn ny CRC.
               WriteToPCBuffer(VHFData,&DataLength);  // Data skrives til PCBuffer
               TI0=1;                                 // og interrupt flaget sættes.
            }
         }
      }
   }
}
```

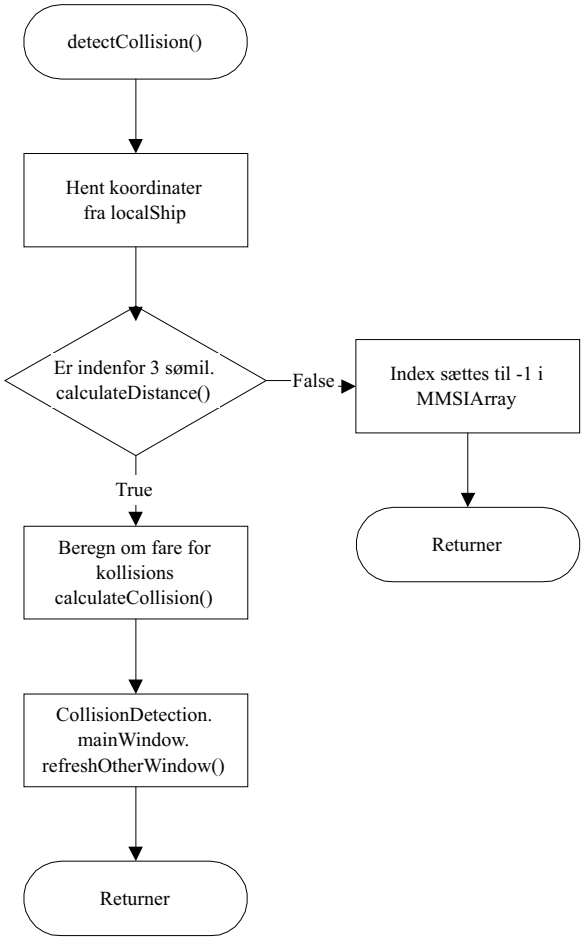
F Flowcharts til PC-program

Dette afsnit indeholder flowcharts over en række af de klasser og metoder som er benyttet i PC programmet. De forskellige flowcharts er delt ind efter hvilken pakke og klasse de tilhører.

F.1 SHIP package

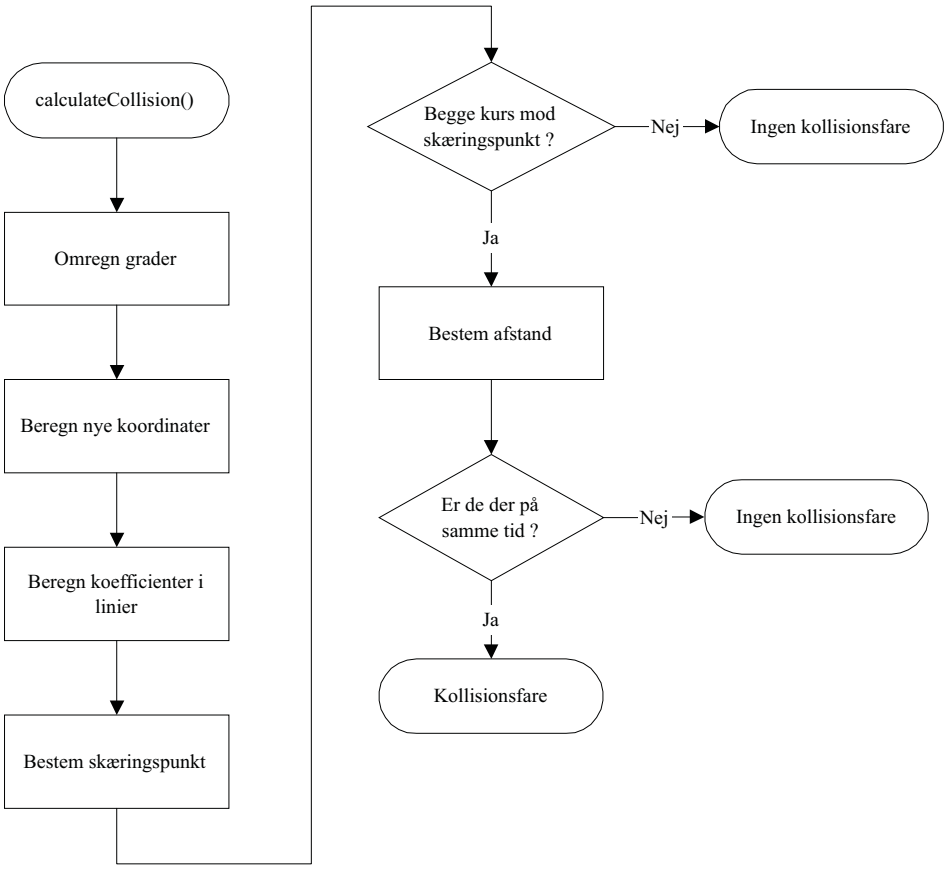
F.1.1 LocalShipInfo:

Kollisionsdetektion:



Figur F.1 Undersøger, på baggrund af den modtagne positionsrapport, om skibet er inden for den kritiske afstand . Hvis dette er tilfældet, kaldes calculateDistance() og skærmen genoptegnes med de nye oplysninger.

Kollisionsberegning:

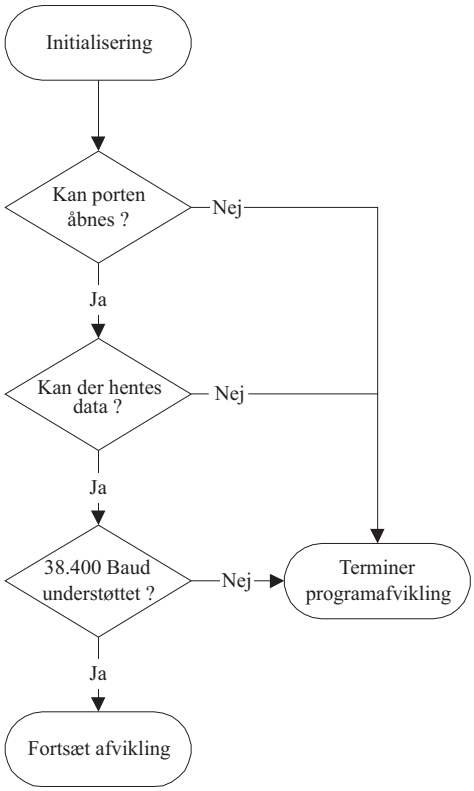


Figur F.2 Her undersøges om der er kollisionskurs for eget skib og andet skib. Dette er opfyldt, hvis skibenes kurs krydser hinandens rute, de sejler mod hinanden og er der til det samme tidspunkt.

F.2 CI package (ControllerInterface)

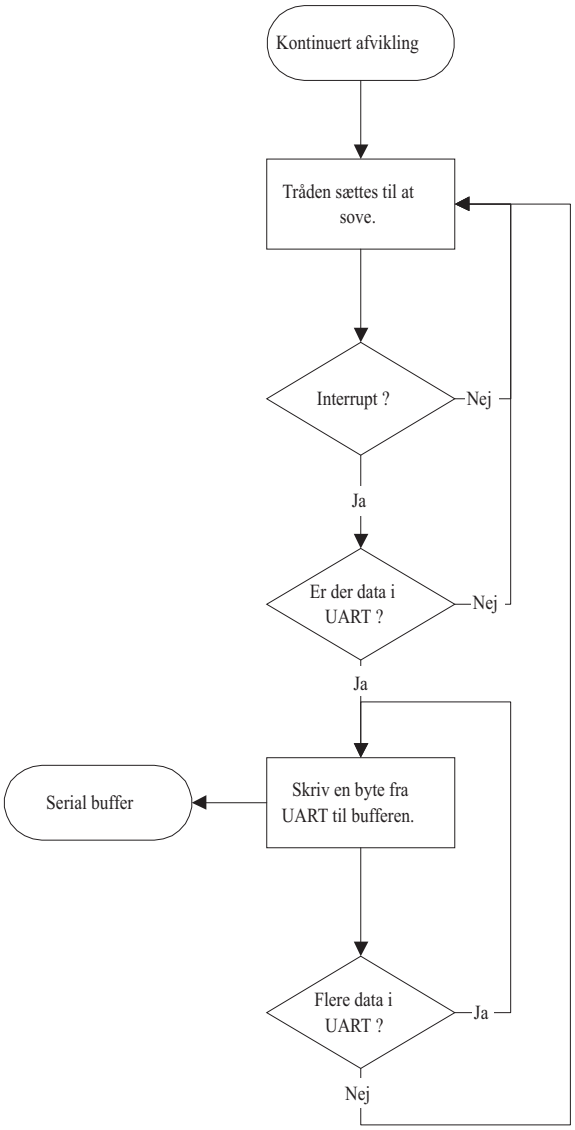
F.2.1 SerialDriver

Initialisering:



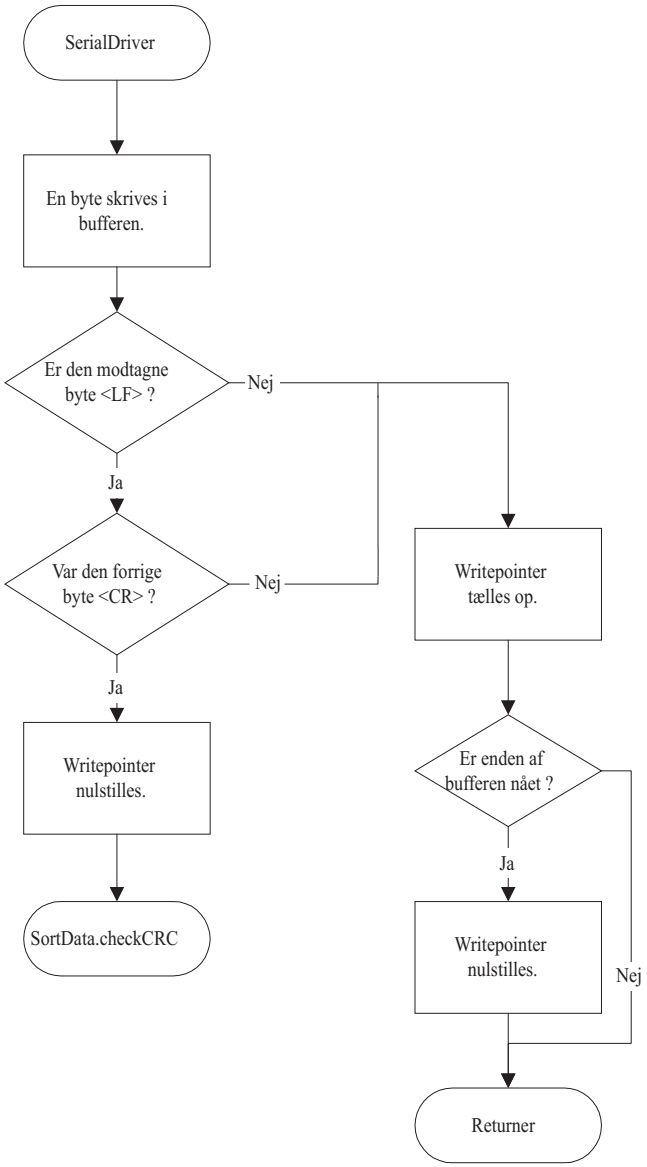
Figur F.3 Initialisering af den serielle port. Dette gøres fra mainløkken.

Kontinuert afvikling:



Figur F.4 Tråden hvor den serielle driver kører. Tråden sættes til at "sove" og vækkes kun, når der er interrupt.

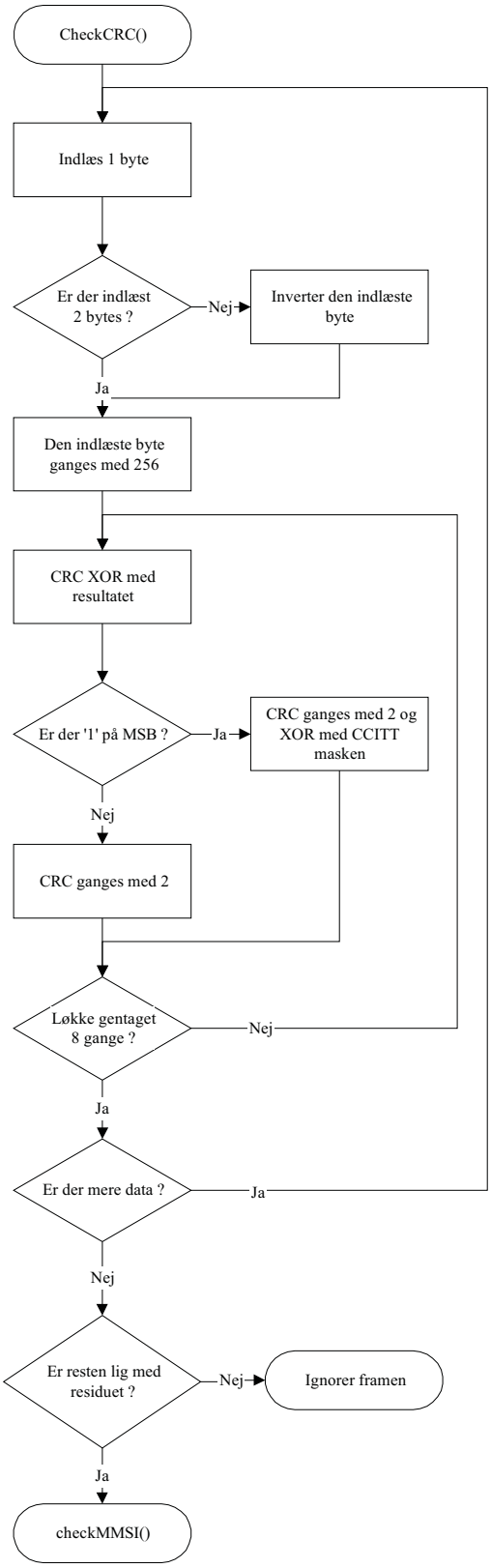
F.2.2 SerialBuffer



Figur F.5 Den serielle buffer, som driveren skriver bytes ind i, når der er data fra controlleren.

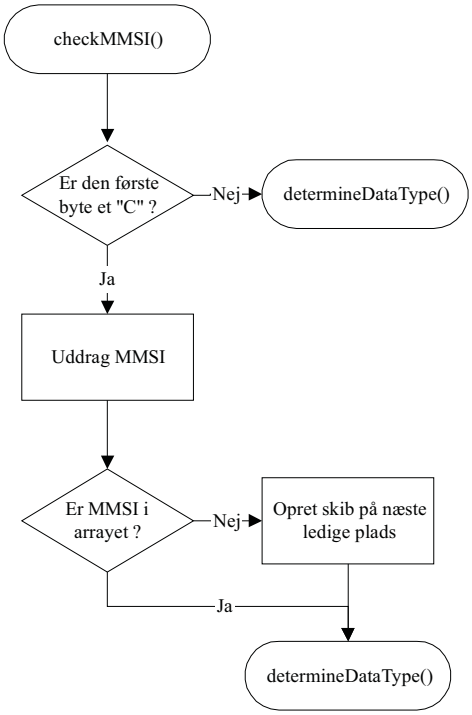
F.2.3 SortData

Check CRC:



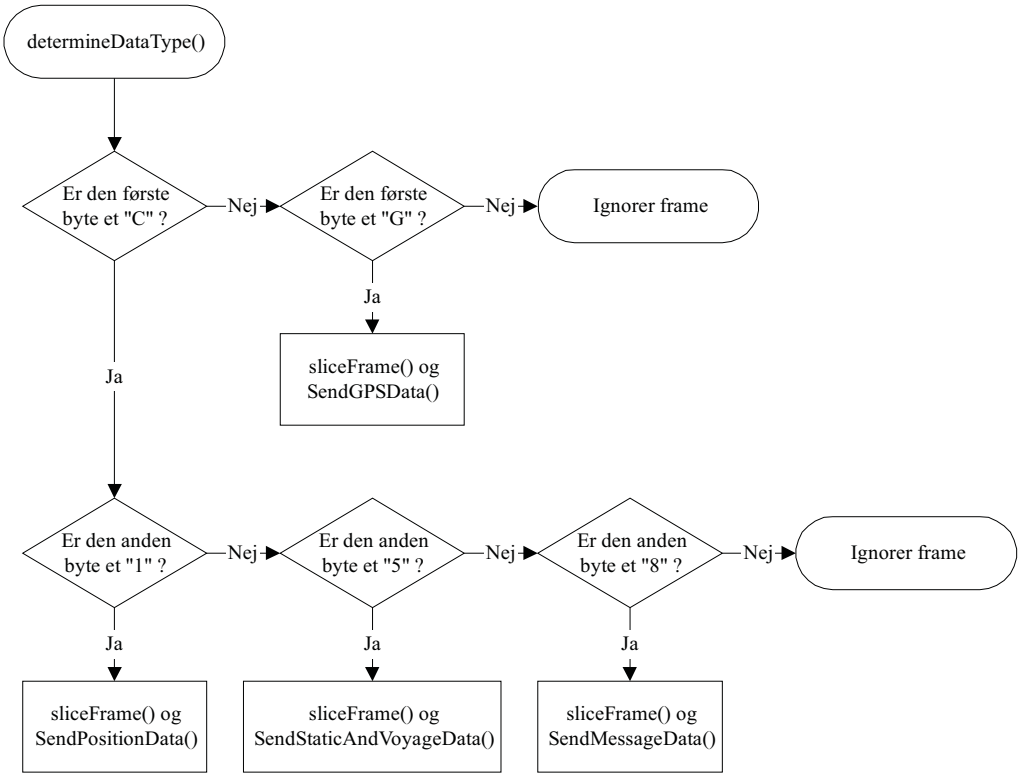
Figur F.6 CheckCRC() undersøger en dataframe fra controlleren for CRC fejl.

Check MMSI nummer:



Figur F.7 Undersøger om der skal oprettes et nyt skibsobjekt på baggrund af den modtagne frame.

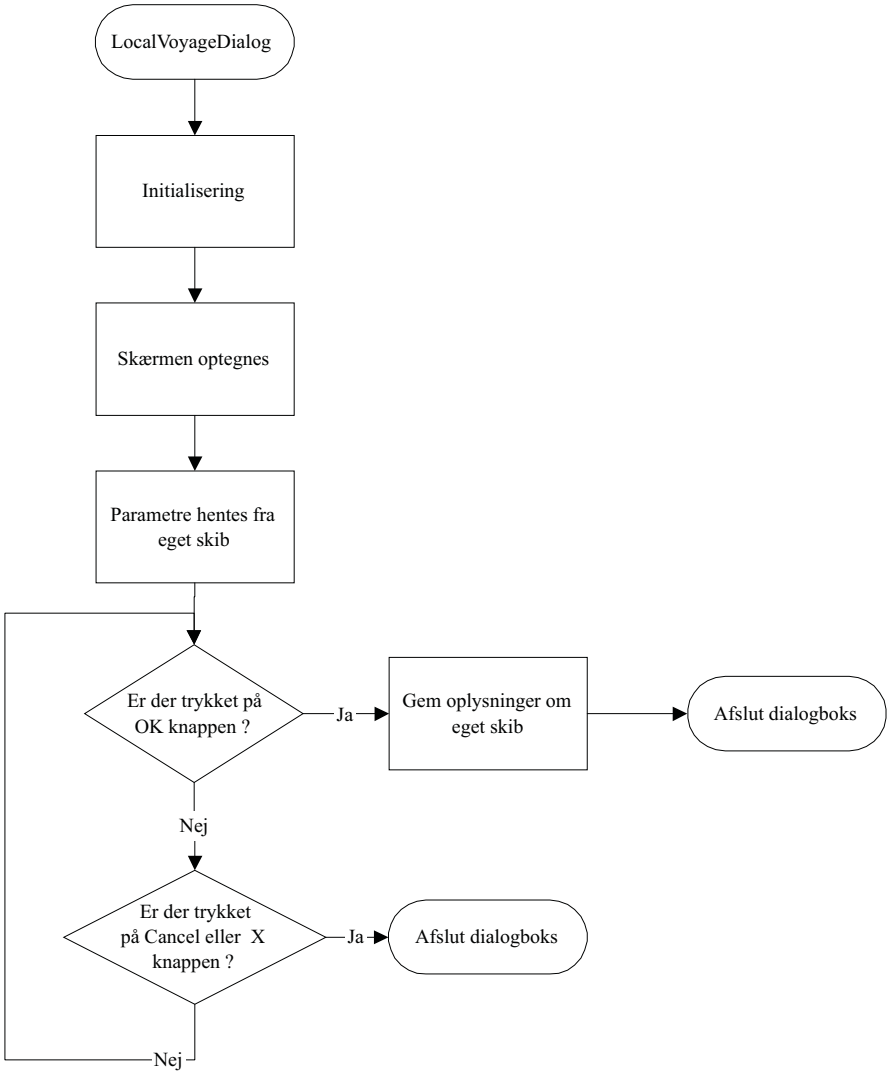
Bestem datatype:



Figur F.8 Bestemmer hvilken dataframe, der er modtaget, og kalder den rette metode.

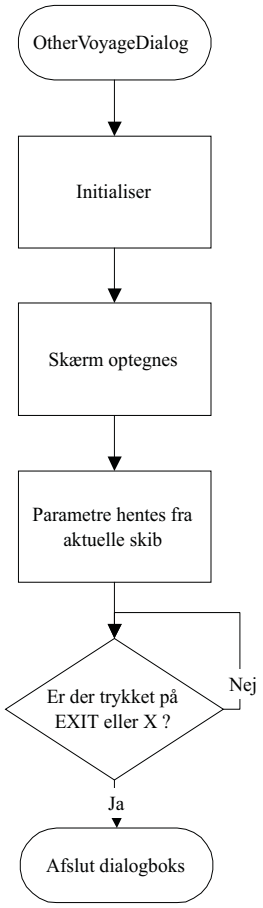
F.3 UI package (UserInterface)

F.3.1 LocalStaticDialog og LocalVoyageDialog



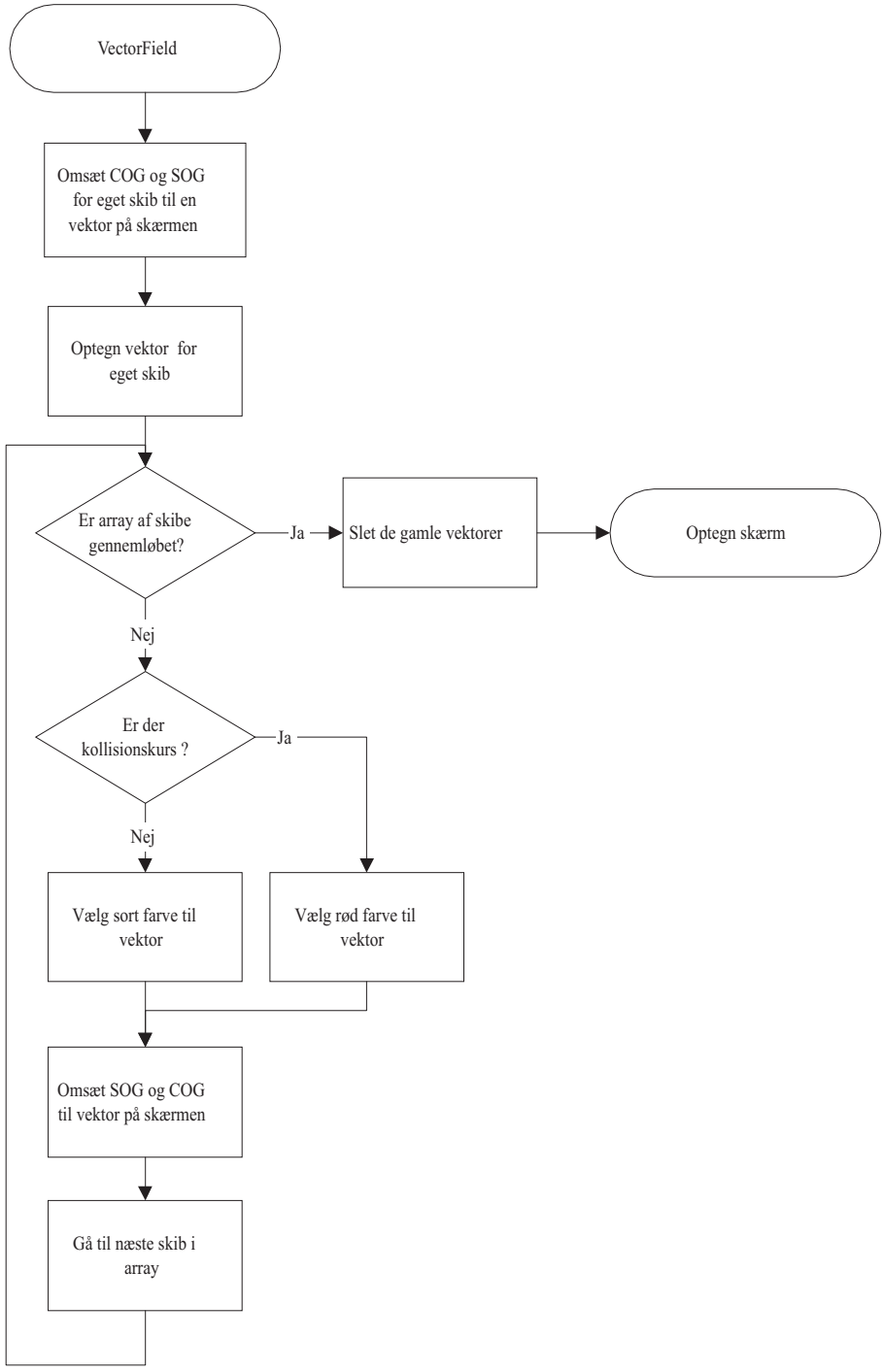
Figur F.9 Dialogbokse for eget skib. De indtastede data gemmes, hvis der trykkes på knappen, OK og ignoreres, hvis der trykkes cancel. Under opstart af dialogboksen hentes de aktuelle data fra eget skib.
Den sidste del af flowchartet, hvor der forekommer nogle valg, aktiveres kun, hvis der trykkes på en knap.

F.3.2 OtherStaticDialog og OtherVoyageDialog



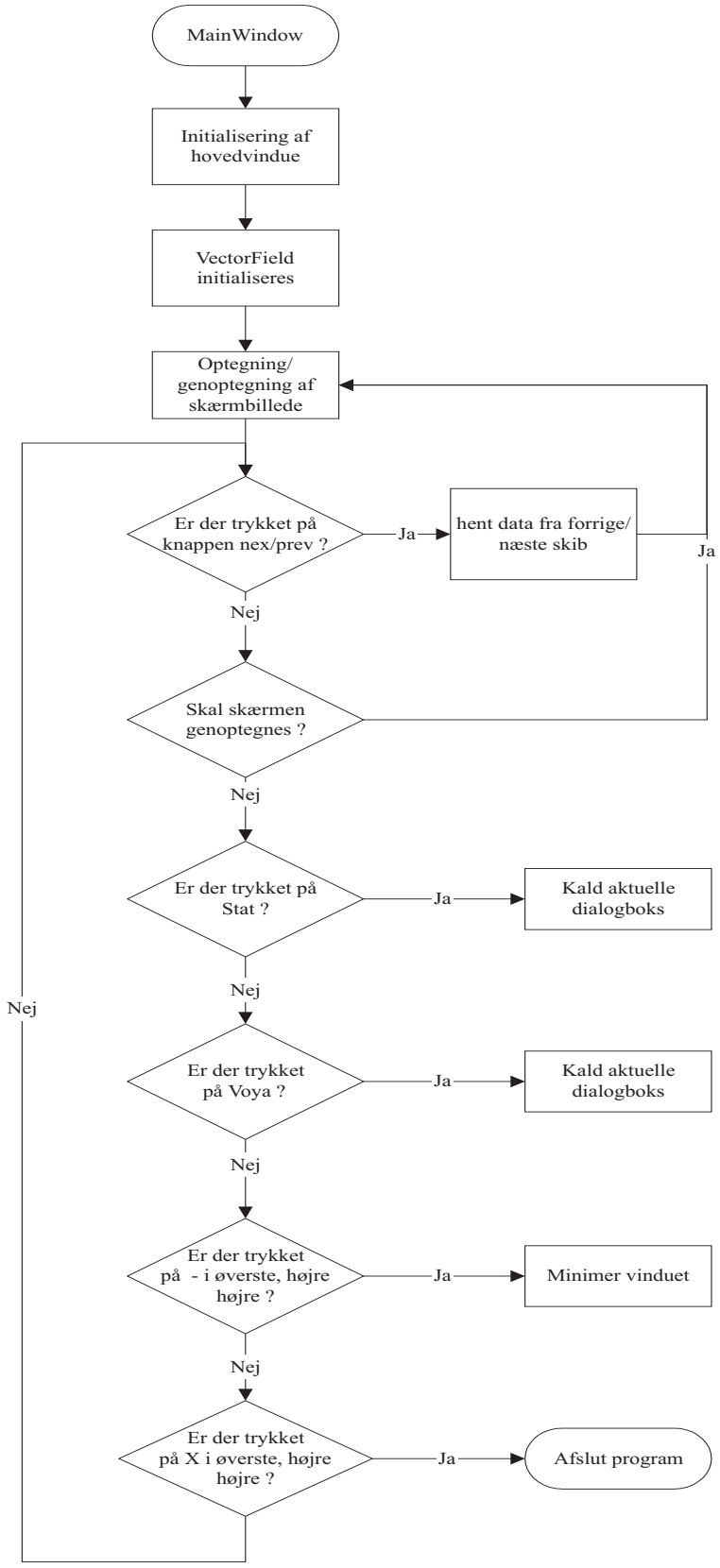
Figur F.10 Dialogbokse for andre skibe. Denne dialogboks initialiseres med data fra det aktuelle skib. Den sidste del af flowchartet, hvor der forekommer et valg, aktiveres kun, hvis der trykkes på en knap.

F.3.3 Vectorfield



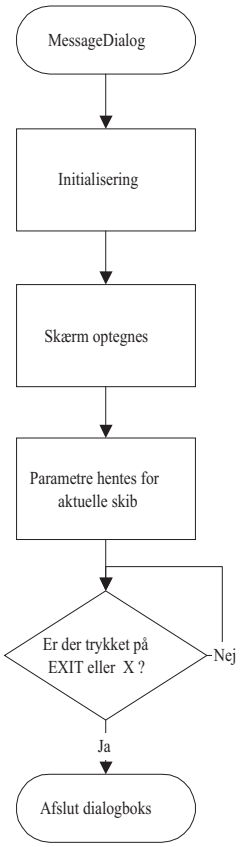
Figur F.11 Her omregnes position, kurs og fart for de enkelte skibe til koordinater på skærmen. Til slut genoptegnes skærmen med de nye værdier.

F.3.4 MainWindow



Figur F.12 Hovedvinduet viser oplysninger om eget og andre skibe og kalder de forskellige dialogbokse. Endvidere er det muligt at bladre i en liste, som indeholder alle skibe inden for den kritiske afstand.

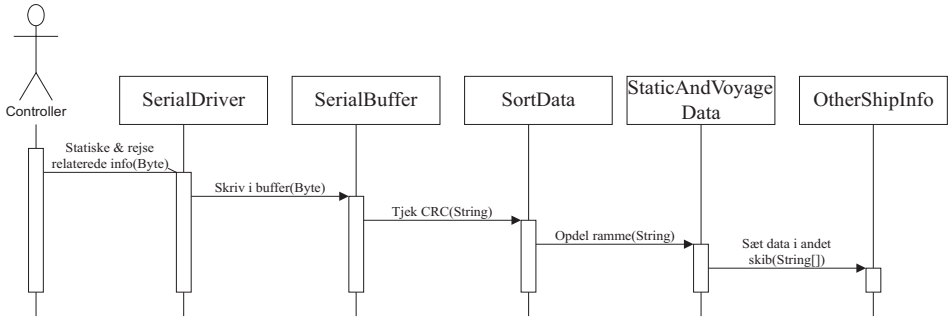
F.3.5 MessageDialog



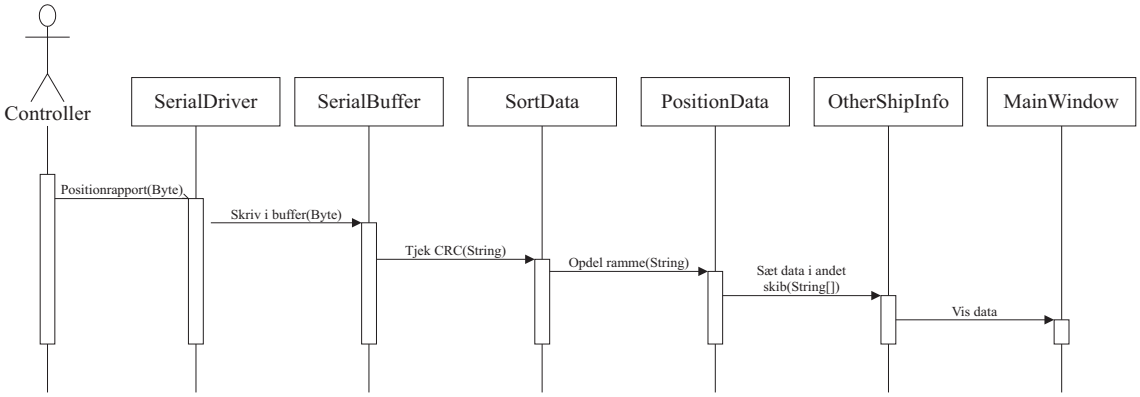
Figur F.13 Dialogboksen for message. Aktiveres når der kommer en besked og MessageData kalder den. Den sidste del af flowchartet, hvor der forekommer et valg, aktiveres kun, hvis der trykkes på en knap.

G Sekvensdiagrammer

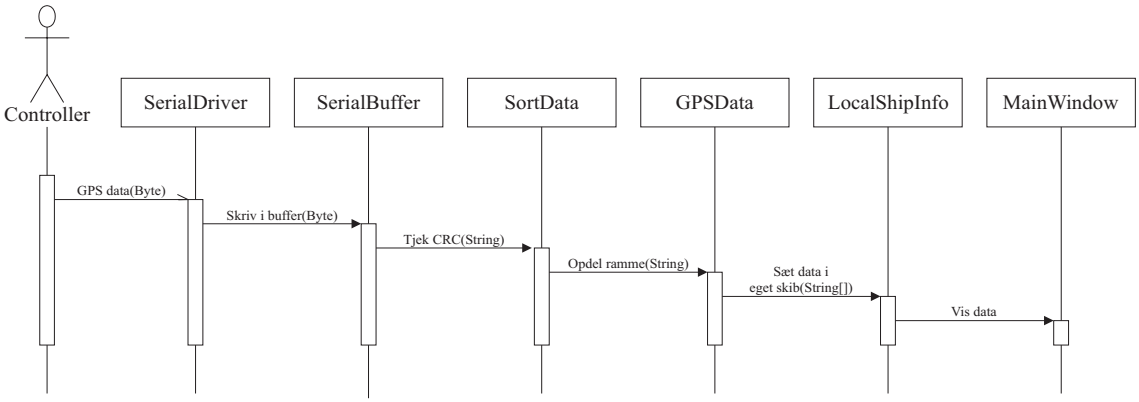
Dette appendiks indeholder sekvensdiagrammer, som giver indblik i. hvordan tid forløbet er igennem de enkelte use cases.
Disse sekvensdiagrammer angiver, hvilke klasser, der anvendes i forskellige use cases. Det er skrevet med pseudokode, hvad der skal foregå i de forskellige klasser på den pil, som går hen til den næste klasse i hvert af diagrammerne.



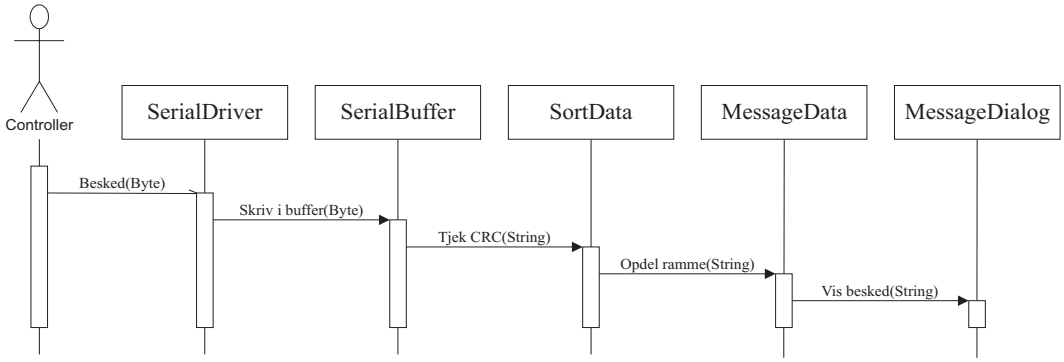
Figur G.1 Modtagelse af statiske / rejserelaterede informationer fra Controller.



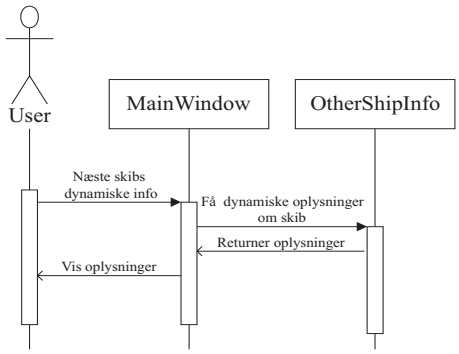
Figur G.2 Modtagelse af dynamiske informationer, såsom kurs, fart andet skib



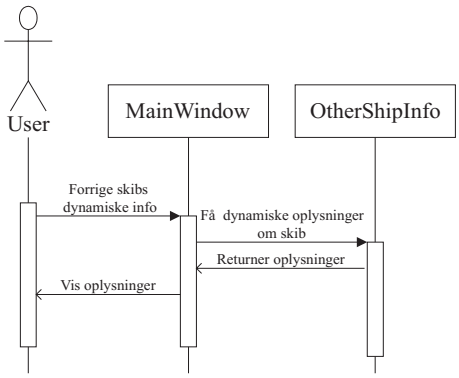
Figur G.3 Modtagelse af dynamiske informationer om eget skib.



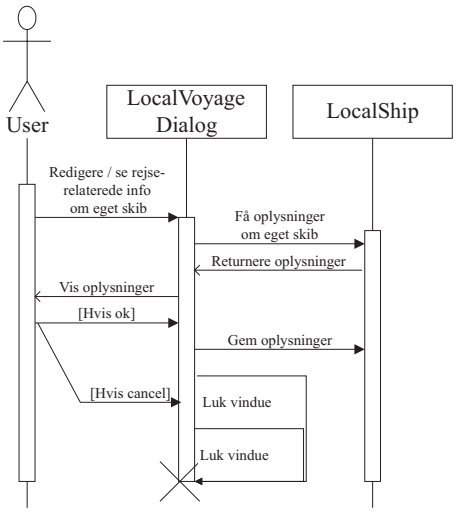
Figur G.4 Modtagelse besked fra andet skib.



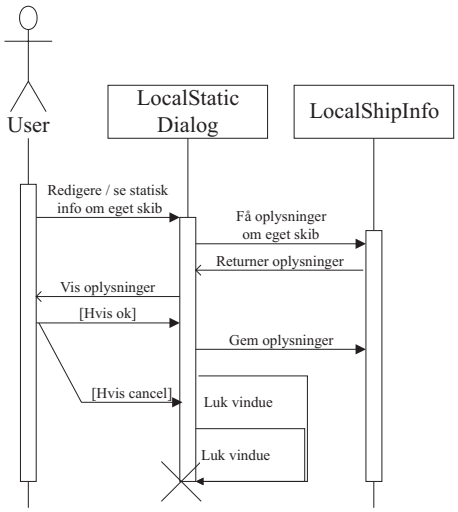
Figur G.5 Der vælges at se informationer om næste skib



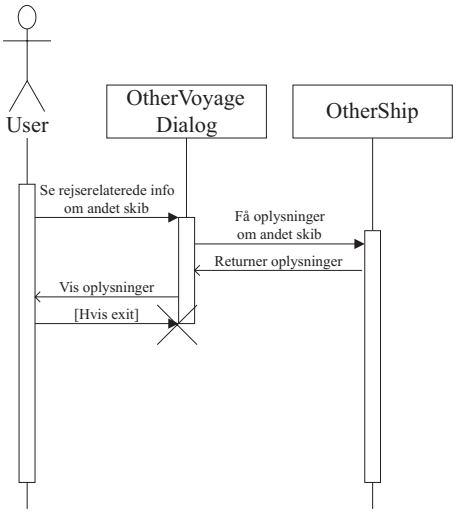
Figur G.6 Der vælges at se informationer om forrige skib.



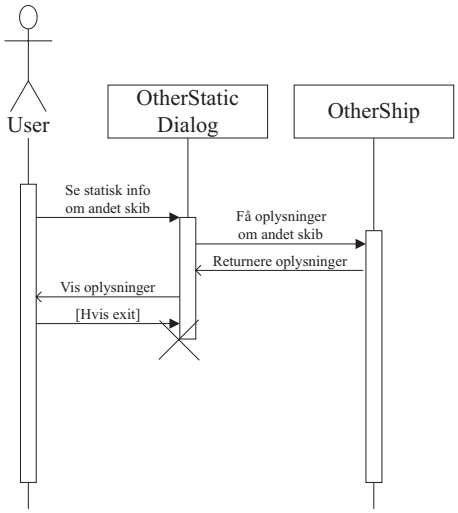
Figur G.7 Der vælges at se / redigere rejserelaterede informationer om eget skib.



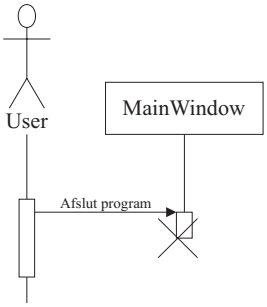
Figur G.8 Der vælges at se / redigere statiske informationer om eget skib.



Figur G.9 Der vælges at se/redigere rejserelaterede informationer om andet skib.



Figur G.10 Der vælges at se/redigere statiske informationer om andet skib.



Figur G.11 Der vælges at afslutte program

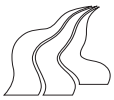
H Kildekode til PC-program

Dette appendiks indeholder kildekode til PC-programmet.
Appendikset er opdelt i afsnit, således at kildekoden til de forskellige filer ligger inde under den pakke, som de hører under.
Først ligger kildekoden til filerne under CI package, derefter UI package og til sidst SHIP package.

H.1 CI package

H.1.1 SortData

```
import OtherShipInfo;         // Importerer klassen OtherShipInfo.
import StaticAndVoyageData;   // Importerer klassen StaticAndVoyageData.
import PositionData;          // Importerer klassen PositionData.
import GPSData;               // Importerer klassen GPSData.
import MessageData;           // Importerer klassen MessageData.

//*****************************************************************************//
// Denne klasse undersøger de data, der kommer fra serielporten for CRC fejl,          //
// opretter og vedligeholder skibsobjekter samt fordeler data til den rigtige Data-    //
// klasse.                                                                     //
//*****************************************************************************//
public class SortData
{
   public static int[] MMSIArray = new int[20]; // Array hvor skibenes MMSI nummer står.
   private String data;
   private int targetIndex =0;   // Nummer på det skib der skal behandles.
   public static OtherShipInfo[] shipArray = new OtherShipInfo[20];  // Array af skibe.
   public static LocalShipInfo localShip = new LocalShipInfo();      // Eget skib.

   PositionData position = new PositionData();
   StaticAndVoyageData staticAndVoyage = new StaticAndVoyageData();
   MessageData message = new MessageData();
   GPSData GPS = new GPSData();

   //*****************************************************************************//
   // Metode, som udgør hovedmetoden i klassen. Der skrives “-1" på alle pladser i    //
   // arrayet med MMSI numre.                                                     //
   //*****************************************************************************//
   public SortData()
   {
      for (byte x = 0; x < 20; x++) //Initialisering af MMSIarrayet.
      {
         MMSIArray[x] = -1;
      }
   }

   //*****************************************************************************//
   // Denne metode undersøger data for CRC fejl.                                  //
   //*****************************************************************************//
   public void checkCRC(byte[] dataArray,int dataSize)
   {
      short CRC=0;
      short CRCData;
      short CCITT = 0x1021;   // Definerer CRC-CCITT masken
      dataArray[dataSize-3] = dataArray[dataSize-2];  // Den første byte i checksum
                                                      // flyttes en plads til venstre.
      dataArray[dataSize-2] = dataArray[dataSize-1];  // Den anden byte i checksum
                                                      // flyttes en plads til venstre.

      for (int x=1;x<=(dataSize-2);x++) //Der startes på plads 1 for at undgå startflag
      {                                 // og der sluttes før stopflaget.
         CRCData=dataArray[x];          // Data flyttes fra arrayet over i CRCData.
```

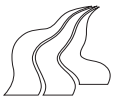


```
         if (x<=2)
         CRCData=(short) (255-CRCData);               // De første 16 bit inverteres
         CRCData<<=8;                                 // Data LeftShiftes 8 gange
         for(int i=0;i<8;i++)                         // CRC beregnes bit for bit
         {
            if (((CRC^CRCData) & 0x8000) == 32768)    // CRC XOR med CRCData og hvis '1'
            {                                         // på første plads,
               CRC=(short)((CRC<<=1)^CCITT);          // leftShiftes CRC 1 gang og
            }                                         // XOR'es med CRC masken.
            else CRC<<=1;                             // ellers LeftShiftes CRC 1 gang.
            CRCData<<=1;                              // CRCData LeftShiftes 1 gang.
         }
      }
      if (CRC==0x1D0F)                    // Hvis residuet er 0x1D0F er data OK.
      {
         String dataCRC;
         dataCRC = new String(dataArray);          // Dataarrayet laves til en streng.
         data = dataCRC.substring(0,dataSize-3);   // CRC fjernes fra data.
         checkMMSI();                              // metoden checkMMSI() kaldes
      }
      else {}  // Hvis CRC ikke var korrekt, ignoreres dataframen.
   }

//*****************************************************************************//
// Denne metode undersøger, om skibet findes, eller om det skal oprettes.      //
//*****************************************************************************//
   private void checkMMSI()
   {
      byte commaFound = 0;
      byte i;
      byte start = 0;
      byte end = 0;
      String mmsiNumber = ("");
      int mmsiInt;
      boolean isInBuffer = false;
      boolean MMSIWritten = false;
      int z = 0;

      if (data.charAt(1) == 'C') // Hvis der er et 'C' på første plads i data.
      {
         i = 2;
         while (commaFound < 2 && i < 30) // Der ledes efter et komma fra plads 2 til 30.
         {
            if (data.charAt(i) == ',') // Hvis der er fundet et komma.
            {
               commaFound++;           // Antallet af fundne kommaer tælles op.
               if (commaFound < 2)     // Hvis der kun er fundet et komma sættes
               {
                  start = ++i;         // start til at pege på næste plads i data.
               }
               end = --i;              // slut peger på sidste plads før det andet komma.
            }
         i++;
         }
         for (byte j = start; j <= end; j++) // Der laves en streng med MMSI-nummeret.
         {
            mmsiNumber = (mmsiNumber + data.charAt(j));  // mmsiNumber concateneres.
         }
         System.out.println(mmsiNumber);     // For testing.
         mmsiInt = Integer.parseInt(mmsiNumber);   // Strengen laves til en Integer.

         for (byte y = 0; y < 20; y++) // Undersøger om MMSI-nummeret findes i arrayet.
         {
            if (MMSIArray[y] == mmsiInt)  // Hvis MMSI-nummeret findes i arrayet.
            {
               targetIndex = y;     // targetIndex sættes.
               isInBuffer = true;   // MMSI-nummeret er i bufferen.
            }
         }

         if (!isInBuffer)  // Hvis MMSI-nummeret ikke er i bufferen.
         {
            while (!MMSIWritten) // Så længe MMSI-nummeret ikke er skrevet i arrayet.
            {
               if (MMSIArray[z] == -1) // Hvis der står "-1" i arrayet overtages pladsen.
               {
                  MMSIArray[z] = mmsiInt; // Nummeret gemmes i MMSIArray.
                  targetIndex = z;        // targetIndex sættes.
                  shipArray[z] = new OtherShipInfo(); // Der oprettes et nyt skibsobjekt.
                  MMSIWritten = true;     // MMSI-nummeret er skrevet i arrayet.
               }
            z++;
            }
            MMSIWritten = false;    // MMSIWritten resettes.
            z = 0;   // tælleren resettes.
         }
```

```
      }
      determineDataType();
   }

//*********************************************************************************//
// Denne metode undersøger, hvilken type data, der er modtaget.                    //
//*********************************************************************************//
   private void determineDataType()
    {
      if (data.charAt(1) == 'C') // Hvis der står 'C' på plads 1 er det data fra VHF.
      {
         if (data.charAt(5) == '1') // Hvis der står '1' på plads 5, er det Positiondata.
         {
            position.sliceFrame(data); // Opdeler dataFramen i enkelte dataelementer.
            position.sendPositionData(targetIndex);   // Sender data til targetskibet.
            System.out.println("Position: " + data);  // For testing.
         }
         else if (data.charAt(5) == '5')
         {
            staticAndVoyage.sliceFrame(data);
            staticAndVoyage.sendStaticAndVoyageData(targetIndex);
            System.out.println("Static&Voyage: " + data);   // For testing.
         }
         else if (data.charAt(5) == '8')
         {
            message.sliceFrame(data);
            message.sendMessage(targetIndex);
            System.out.println("Message: " + data);   // For testing.
         }
         else {}
      }
      if (data.charAt(1) == 'G') // Hvis der står 'G' på plads 1 er det GPS data.
      {
         GPS.sliceFrame(data);
         GPS.sendGPSData();      // Data overføres til eget skib.
         System.out.println("GPS: " + data); // For testing.
      }
      else {}
    }
}
```

H.1.2 SerialDriver

```
import java.io.*;
import java.util.*;
import javax.comm.*; // En særlig klasse til håndtering af porte.

import SerialBuffer; // Så der kan skrives i den serielle buffer.

//*********************************************************************************//
// Denne klasse implementerer en tråd der modtager data fra serielporten.          //
//*********************************************************************************//
public class SerialDriver implements Runnable, SerialPortEventListener
{
   InputStream inputStream;         // En ny inputstream oprettes.
   SerialPort  com1;                // Her oprettes et serielport objekt.
   Thread readThread;               // En ny tråd.
   SerialBuffer buffer;             // En serielbuffer oprettes.

   //*********************************************************************************//
   // Denne klasse initialiserer den serielport den kaldes med.                       //
   //*********************************************************************************//
   public SerialDriver(CommPortIdentifier portId)
   {
      System.out.println("\n*********************************");
      System.out.println("* Seriel driver med 38.400 baud *");
      System.out.println("*********************************\n");

      buffer = new SerialBuffer();

      // Forsøger at overtage COM1 porten i 2000 ms.
      try   {
               com1 = (SerialPort) portId.open("Serial driver", 2000);
               System.out.println("COM1 er overtaget!");
            }
      catch (PortInUseException e)  // Hvis porten er i brug sendes en exception.
            {
               System.out.println("Porten er optaget eller benyttes af andre programmer !");
            }
      // Tester om der kan hentes data fra COM1.
      try   {
               inputStream = com1.getInputStream();      // Inputstream fra COM1 .
               System.out.println("COM1 test passed!");  // For testing.
            }
      catch (IOException   e) // Hvis der ikke kan hentes data sendes en exception.
            {
```



```
                System.out.println("Der er ikke hul igennem til porten !");
            }
        // Tester om der kan tilføjes en eventlistner til COM1.
        try   {
                com1.addEventListener(this);
            }
        catch (TooManyListenersException e) // Hvis der er for mange der lytter på porten.
            {
                System.out.println("Der er andre der \"lytter\" på porten !");
            }
            com1.notifyOnDataAvailable(true);   // Aktiverer "events" på COM1.
        // Tester om de ønskede instillinger er understøttet.
        try   {
                com1.setSerialPortParams(38400, SerialPort.DATABITS_8, SerialPort.STOPBITS_1,
                SerialPort.PARITY_NONE);
            }
        catch (UnsupportedCommOperationException e) // Hvis indstillingerne ikke er tilladte
            {
                System.out.println("Den indstillede bitrate er ikke understøttet !");
            }

        readThread =   new Thread(this);    // Opretter en ny tråd.
        readThread.start();                 // Starter eksekveringen af denne tråd.
    }

    //*************************************************************************//
    // Denne metode kaldes, når tråden startes                                 //
    //*************************************************************************//
    public void run()                       // Denne køres vha. readThread.start()
    {
        try   {
                Thread.sleep(20000);          // Sætter tråden til at sove i 2 sek.
            }
        catch (InterruptedException   e)    // Hvis der er interrupt kaldes "serialEvent".
            {
            }
     }

    //*************************************************************************//
    // Denne metode kaldes når der kommer en Interruptexception på COM1        //
    //*************************************************************************//
    public void serialEvent(SerialPortEvent event)
    {
        switch(event.getEventType())                // Hvilken event var der på COM1.
        {
            case SerialPortEvent.BI:                // Hvis det er "Break Interrupt".
            case SerialPortEvent.OE:                // Hvis det er "Overrun Error".
            case SerialPortEvent.FE:                // Hvis der er "Framing Error".
            case SerialPortEvent.PE:                // Hvis der er "Parity Error".
            case SerialPortEvent.CD:                // Hvis der er "Carrier Detect".
            case SerialPortEvent.CTS:               // Hvis der er "Clear To Send".
            case SerialPortEvent.DSR:               // Hvis der er "Data Set Ready".
            case SerialPortEvent.RI:                // Hvis der er "Ring Indicator".
            case SerialPortEvent.OUTPUT_BUFFER_EMPTY: // Hvis der er "Output Buffer Empty".
            break;
            case SerialPortEvent.DATA_AVAILABLE:    // Hvis der er "Data Available".

            byte[] oneCharacter = new byte[1];       // Et bytearray på et tegn.
            try   {
                    while (inputStream.available() > 0) // Så længe der er bytes i UART'en.
                    {
                        int numBytes   = inputStream.read(oneCharacter);
                        buffer.writeBuffer(oneCharacter);
                    }
                }
            catch (IOException e) {}
            break;
        }
    }
}
```

H.1.3 SerialBuffer

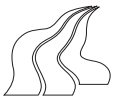
```
import SortData;  // Importerer SortData så data kan sendes til denne klasse.

//*************************************************************************//
// Denne klasse er en lineær serielbuffer med plads til een dataframe      //
//*************************************************************************//
public class SerialBuffer
{

    private byte[] buffer;
    private short writePointer = 1;
    private short readPointer = 1;
    private final short BUFFERSIZE = 400;
    private String dataString;
```

```
   private byte[] tempBuffer;
   SortData sorting = new SortData();
   //*************************************************************************************//
   // Denne metode er constructoren, som initialiserer bufferen                           //
   //*************************************************************************************//
   public SerialBuffer()
   {
      buffer = new byte[BUFFERSIZE];   // Opretter en buffer af bytes med str BUFFERSIZE.
      dataString = ("");
   }

   //*************************************************************************************//
   // Denne metode undersøger, om der kommet en hel frame                                 //
   //*************************************************************************************//
   private void checkData()
   {
      if (buffer[writePointer-1] == 13 && buffer[writePointer] == 10)
      {                                        // Hvis der er kommet et <CR> og en <LF>.
         tempBuffer = new byte[writePointer-2]; // En midlertidig buffer oprettes.
         for (short i = 0; i < writePointer-2; i++)
         {
           tempBuffer[i] = buffer[i+1];
         }
         sorting.checkCRC(tempBuffer, writePointer-2);   // checkCRC med buffer og længde
         System.out.println("Der er modtaget en frame.");   // For testing.
         dataString = ("");
         writePointer = 0;
      }
      else {}
   }
   //*************************************************************************************//
   // Denne metode kaldes med en byte, som skrives i bufferen.                            //
   //*************************************************************************************//
   public void writeBuffer(byte[] c)
   {
      buffer[writePointer] = c[0];  // En byte skrives i bufferen.
      checkData();                  // Check om der er kommet en frame.
      writePointer++;               // writePointer tælles op.

      if (writePointer == BUFFERSIZE)  // Hvis enden af bufferen er nået
      {
         writePointer=1;   // Wrap around.
      }
      else {}
   }
}
```

H.2 UI package

H.2.1 MainWindow

```
import java.awt.event.*;
import javax.swing.JFrame;
import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;
import Vectorfield;
import LocalStaticDialog;
import LocalVoyageDialog;
import OtherStaticDialog;
import OtherVoyageDialog;
import SortData;


//*************************************************************************************//
// Denne klasse indeholder metoderne, som optegner opstartsskærmen, opdaterer den og   //
// tilknytter knapperne en metode. Det er den centrale klasse i UI package.            //
//*************************************************************************************//
public class MainWindow extends JFrame
{
   private JLabel Longitude, Latitude, COG, SOG, Ownship, Longitude1, Latitude1, COG1,
            SOG1, ship, MMSI, OtherShip;
   private JPanel panel1, panel2, panel3, panel4, OWNship, OTHERship, edit, view, Ship,
            Vectorfield;
   private EmptyBorder border1, border2, border3, border4, border5, border6, border7,
            border8;
   private JButton editSTAT, editVOYA, viewVOYA, viewSTAT, next, prev;
   private Vectorfield kanvas;
   private LocalStaticDialog local1;
   private LocalVoyageDialog local2;
   private OtherStaticDialog Other1;
   private OtherVoyageDialog Other2;
   private int target = 0;
```



```
   private boolean nextFound = false;

   //*************************************************************************//
   // Denne metode bruges til at initialisere hovedvinduet. Der optegnes paneler med   //
   // knapper, som aktiverer de forskellige dialogbokse, og til disse tilknyttes       //
   // actionlistener.                                                                  //
   //*************************************************************************//
   private void InitializeMainWindow()
   {
      this.setLocation(0,0);                          //Sætter placeringen af dette objekt
      this.setSize(800,600);                           //Sætter størrelsen af dette objekt
      this.setTitle("Collisiondetector");                  //Sætter titlen på dette objekt
      this.setResizable(false);                         //Fastsætter størrelsen på Jframen

      //De to edit knapper til eget skib oprettes og sættes ind i et panel.
      editSTAT = new JButton("Static");
      editVOYA = new JButton("Voyage");
      editSTAT.setToolTipText ("Edit static related info");
      editVOYA.setToolTipText ("Edit voyage related info");
      edit = new JPanel(false);
      edit.setLayout (new BoxLayout(edit, BoxLayout.X_AXIS));
      edit.add (editVOYA);
      edit.add (Box.createRigidArea (new Dimension(5,0)));
      edit.add (editSTAT);

      //Panel med to view knapper oprettes.
      viewVOYA = new JButton("Voyage");
      viewSTAT = new JButton("Static");
      viewSTAT.setToolTipText ("View static related info");
      viewVOYA.setToolTipText ("View voyage related info");
      view = new JPanel(false);
      view.setLayout (new BoxLayout(view, BoxLayout.X_AXIS));
      view.add (viewVOYA);
      view.add (Box.createRigidArea (new Dimension(5,0)));
      view.add (viewSTAT);

      //Panel med to knapper for next og previous oprettes.
      prev = new JButton("Prev");
      next = new JButton("Next");
      prev.setToolTipText ("View previous ship");
      next.setToolTipText ("View next ship");
      Ship = new JPanel(false);
      Ship.add (next);
      Ship.add (prev);

      //Actionlistener tilknyttes de forskellige knapper.
      prev.addActionListener(new MainListener());
      next.addActionListener(new MainListener());
      viewVOYA.addActionListener(new MainListener());
      viewSTAT.addActionListener(new MainListener());
      editVOYA.addActionListener(new MainListener());
      editSTAT.addActionListener(new MainListener());
   }


   //*************************************************************************//
   // Metode, hvor tekstfelter opdateres i hovedvinduet. Her opdateres cirklen med     //
   // vektorer også.                                                                   //
   //*************************************************************************//
   private void createContents()
   {
      //Forskellige statiske info om eget skib indskrives i labels
      Ownship = new JLabel("LOCAL SHIP:");
      Longitude = new JLabel("Longitude: ");
      Latitude = new JLabel("Latitude: ");
      COG = new JLabel("COG: ");
      SOG = new JLabel("SOG: ");
      panel1 = new JPanel(false);
      panel1.setLayout(new GridLayout(0,1));
      border6 = new EmptyBorder(20, 0, 20, 0);
      panel1.setBorder(border6);
      panel1.add(Ownship);
      panel1.add(Longitude);
      panel1.add(Latitude);
      panel1.add(COG);
      panel1.add(SOG);

      //Forskellige dynamiske info om andre skibe indskrives i labels
      OtherShip = new JLabel("OTHER SHIPS");
      Longitude1 = new JLabel("Longitude: ");
      Latitude1 = new JLabel("Latitude: ");
      COG1 = new JLabel("COG: ");
      SOG1 = new JLabel("SOG: ");
      MMSI = new JLabel("MMSI: ");
      panel2 = new JPanel(false);
      panel2.setLayout(new GridLayout(0,1));
```

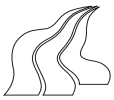
```
    border7 = new EmptyBorder(20, 0, 20, 0);
    panel2.setBorder(border7);
    panel2.add(OtherShip);
    panel2.add(MMSI);
    panel2.add(Longitude1);
    panel2.add(Latitude1);
    panel2.add(COG1);
    panel2.add(SOG1);

    //Panel som indeholder info om andet skib samt to knapper, som gør, at man
    //kan se statisk/voyage related info
    OTHERship = new JPanel();
    OTHERship.setLayout (new BorderLayout(0,0));
    border5 = new EmptyBorder(10, 0, 10, 0);
    OTHERship.setBorder(border5);
    OTHERship.add(Ship, BorderLayout.NORTH);
    OTHERship.add (panel2, BorderLayout.CENTER);
    OTHERship.add (view, BorderLayout.SOUTH);

    //Panel som indeholder info om eget skib samt to knapper til at editere
    //statisk/voyage related info.
    OWNship = new JPanel();
    OWNship.setLayout (new BorderLayout(0,0));
    border4 = new EmptyBorder(20, 0, 20, 0);
    OWNship.setBorder(border4);
    OWNship.add (panel1, BorderLayout.WEST);
    OWNship.add (edit, BorderLayout.SOUTH);

    //De to paneler med info om henholdsvis eget skib og andet skib adskilles
    //her ved y-aksen ( panel Ownship og OtherShip sammensættes)
    panel3 = new JPanel(false);
    panel3.setLayout(new BoxLayout(panel3, BoxLayout.Y_AXIS));
    panel3.add(OWNship);
    panel3.add(OTHERship);

    //Her indlæses class'en Vectorfield og optegner vektorer og blå cirkel
    kanvas = new Vectorfield();
    Vectorfield = new JPanel(true);
    Vectorfield.setLayout(new BoxLayout(Vectorfield, BoxLayout.X_AXIS));
    Vectorfield.add(kanvas);

    //Her opdeles panel3 og Vectorfield ved x-aksen
    panel4 = new JPanel(false);
    panel4.setLayout (new BorderLayout(0,0));
    border4 = new EmptyBorder(0, 0, 0, 0);
    panel4.setBorder(border4);
    panel4.add(Vectorfield);
    panel4.add(panel3, BorderLayout.EAST);
    this.setContentPane(panel4);

}

//*****************************************************************************//
// Hovedmetode i denne klasse. Gennemløber metoderne til oprettelse/opdatering af   //
// hovedvinduet.                                                               //
//*****************************************************************************//
    public MainWindow()
    {
        InitializeMainWindow();
        createContents();
    }

    //*****************************************************************************//
    // Metode til at opdatere de dynamiske informationer om eget skib.            //
    //*****************************************************************************//
    public void refreshLocalWindow()
    {
        Longitude.setText("Longitude: " + SortData.localShip.getLongitude());
        Latitude.setText("Latitude: " + SortData.localShip.getLatitude());
        COG.setText("COG: "+ SortData.localShip.getCOG() + " Degrees");
        SOG.setText("SOG: " + SortData.localShip.getSOG() + " Knots");
        this.setContentPane(panel4);
    }

    //*****************************************************************************//
    // Metode til at opdatere de dynamiske informationer om eget skib.            //
    //*****************************************************************************//
    public void refreshOtherWindow()
    {
        Longitude1.setText("Longitude: " + SortData.shipArray[target].getLongitude());
        Latitude1.setText("Latitude: " + SortData.shipArray[target].getLatitude());
        COG1.setText("COG: " + SortData.shipArray[target].getCOG() + " Degrees");
        SOG1.setText("SOG: " + SortData.shipArray[target].getSOG() + " Knots");
        MMSI.setText("MMSI: " + SortData.shipArray[target].getMMSInumber());
        this.setContentPane(panel4);
    }
```

```
//*****************************************************************************//
// Actionlistener, hvori de forskellige knappers tilhørende metode udføres.    //
//*****************************************************************************//
public class MainListener implements ActionListener
{
   //**************************************************************************//
   // Denne metode aktiveres, når actionlistener aktiveres                     //
   //**************************************************************************//
   public void actionPerformed(ActionEvent e)
   {
      Object source = e.getSource();
      //hvis der trykkes på viewVOYA knappen aktiveres dialogboksen for
      //rejserelaterede informationer. Indholdet af dialogboksen skal være
      //informationer om det skib target(attribut) peger på.
      if (source == viewVOYA)
      {
         if (SortData.MMSIArray[target] != -1)
         {
            Other2 = new OtherVoyageDialog(target);
            Other2.show();
         }
         else {}
      }
      //Hvis der trykkes på viewSTAT aktiveres dialogboksen for statiske informationer.
      //Indholdet af dialogboksen skal være
      //informationer om det skib target(attribut) peger på.
      else if (source == viewSTAT)
      {
         if (SortData.MMSIArray[target] != -1)
         {
            Other1 = new OtherStaticDialog(target);
            Other1.show();
         }
         else {}
      }
      //hvis der trykkes på knappen editVOYA instantieres dialogboksen til egne
      //rejserelaterede informationer.
      else if (source == editVOYA)
      {
         local2 = new LocalVoyageDialog();
         local2.show();
      }
      //hvis der trykkes på knappen editSTAT instantieres dialogboksen til egne
      //statiske informationer.
      else if (source == editSTAT)
      {
         local1 = new LocalStaticDialog();
         local1.show();
      }
      //hvis der trykkes på prev knappen, skal der bladres til næste skib i arrayet.
      else if (source == prev)
      {
         target--;       //Attribut til at holde styr på, hvor i arrayet man
                  // er tælles ned.
         int counter = 0;  //Attributten counter nustilles
         if (target == -1) //hvis pladsen i arryet er tomt flyttes til enden af arrayet
         {
            target = 19;
         }
         //indholdet af objektet target peger på skrives ind i label
         while (!nextFound && counter < 20)
         {
            if (SortData.MMSIArray[target] != -1)     //hvis der ikke står -1 skrives
            {                                         //indholdet ind
               Longitude1.setText("Longitude: "
               + SortData.shipArray[target].getLongitude());
                 Latitude1.setText("Latitude: " +SortData.shipArray[target].getLatitude());
               COG1.setText("COG: " + SortData.shipArray[target].getCOG() + " Degrees");
               SOG1.setText("SOG: " + SortData.shipArray[target].getSOG() + " Knots");
               MMSI.setText("MMSI: " + SortData.shipArray[target].getMMSInumber());
               setContentPane(panel4);
               nextFound = true;
            }
            else     //hvis der står -1 flyttes der een plads ned i arrayet og
                  //der checkes hvad der
            {        //står i objektet der og target flyttes til slutning af arrayet.
               target--;
               counter++;
               if (target == -1)
               {
                  target = 19;
               }
            }
         }
         nextFound = false;
```

```
            }
            //Hvis der trykkes på next, skal der flyttes en plads op i arrayet skibe
            else if (source == next)
            {
               target++;
               int counter = 0;
               if (target == 20)
               {
                  target = 0;
               }
               while (!nextFound && counter < 20) //er lavet på samme måde som ovenstående
               {                                  //med den forskel at der her tælles op i arrayet
                  if (SortData.MMSIArray[target] != -1)
                  {
                     Longitude1.setText("Longitude: " +
                     SortData.shipArray[target].getLongitude());
                     Latitude1.setText("Latitude: " + SortData.shipArray[target].getLatitude());
                     COG1.setText("COG: " + SortData.shipArray[target].getCOG() + " Degrees");
                     SOG1.setText("SOG: " + SortData.shipArray[target].getSOG() + " Knots");
                     MMSI.setText("MMSI: " + SortData.shipArray[target].getMMSInumber());
                     setContentPane(panel4);
                     nextFound = true;
                  }
                  else
                  {
                     target++;
                     counter++;
                     if (target == 20)
                     {
                        target = 0;
                     }
                  }
               }
               nextFound = false;
            }
            else {}
         }
      }
   }
```

H.2.2 VoyageDialog

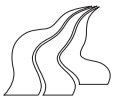
```
import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
import InfoDialog;

//*****************************************************************************//
// I denne klasse oprettes dialogboksen for rejserelaterede informationer.     //
//*****************************************************************************//
public class VoyageDialog extends InfoDialog
{
   protected JLabel typeOfShip, specialConditions;
   protected JLabel actualDraught, destination, eta;
   protected JLabel typeOfSensor;
   protected JButton okButton, cancelButton;
   protected JTextField draughtField, destinationField;
   protected JTextField etaField1, etaField2, etaField3, etaField4;
   protected JComboBox shipBox, specialBox, sensorBox;
   protected JPanel labelPanel, boxPanel, etaPanel, boxAndlabelPanel;
   protected JPanel buttonPanel, mainPanel;


   //*****************************************************************************//
   // I denne metode optegnes dialogboksen med alle knapper osv.                  //
   //*****************************************************************************//
   protected void createContents()
   {

      //Her defineres alle text labels
      typeOfShip = new JLabel("Type of ship: ", JLabel.LEFT);
      specialConditions = new JLabel("Special conditions: ", JLabel.LEFT);
      typeOfSensor = new JLabel("Type of nav. sensor: ", JLabel.LEFT);
      destination = new JLabel("Destination: ", JLabel.LEFT);
      actualDraught = new JLabel("Actual draught: ", JLabel.LEFT);
      eta = new JLabel("ETA (mm/dd/hh/mm): ", JLabel.LEFT);

      //Her defineres alle text fields
      destinationField = new JTextField(1);
      draughtField = new JTextField(1);
      etaField1 = new JTextField(1);
      etaField1.setHorizontalAlignment(JTextField.CENTER);
      etaField2 = new JTextField(1);
      etaField2.setHorizontalAlignment(JTextField.CENTER);
      etaField3 = new JTextField(1);
      etaField3.setHorizontalAlignment(JTextField.CENTER);
```



```
        etaField4 = new JTextField(1);
        etaField4.setHorizontalAlignment(JTextField.CENTER);

        //Her defineres alle knapper samt panelet de sidder i
        okButton = new JButton("OK");
        cancelButton = new JButton("Cancel");
        buttonPanel = new JPanel(false);
        buttonPanel.setLayout(new BoxLayout(buttonPanel, BoxLayout.X_AXIS));
        buttonPanel.setBorder(new EmptyBorder(0,210,0,0));
        buttonPanel.add(okButton);
        buttonPanel.add(cancelButton);

        //Her defineres indholdet af comboboxes
        String[] ships = {"<Undefined>","Pilot boat","Search & rescue vessel","Tugs",
                        "Port tender", "Vessel with anti pollution facilities",
                        "Law enforcement vessel", "Medical transport",
                        "Ships according to Res. 18","Passenger ship *",
                        "Cargo ship *","Tanker *","Other type of ship *"};
        String[] special = {"All ships of this type","DG, HS or MP Cat. A",
                        "DG, HS or MP Cat. B", "DG, HS or MP Cat. C","DG, HS or MP Cat. D",
                        "Not under command", "Restricted abillity to manoeuvre",
                        "Constrained by draught", "No additional information"};
        String[] sensors = {"<Undefined>","GPS","GLONASS","GPS/GLONASS","Loran-C","Chayka",
                        "Integrated Nav. System", "Surveyed (base station)"};

        //Her defineres alle comboboxes
        shipBox = new JComboBox(ships);
        specialBox = new JComboBox(special);
        sensorBox = new JComboBox(sensors);

        //Her defineres et panel sammensat af JLabels
        labelPanel = new JPanel(false);
        labelPanel.setLayout(new GridLayout(0,1));
        labelPanel.add(typeOfShip);
        labelPanel.add(specialConditions);
        labelPanel.add(typeOfSensor);
        labelPanel.add(destination);
        labelPanel.add(actualDraught);
        labelPanel.add(eta);

        //Her defineres et panel med ETA tekstfelter
        etaPanel = new JPanel(false);
        etaPanel.setLayout(new GridLayout(1,0));
        etaPanel.add(etaField1);
        etaPanel.add(etaField2);
        etaPanel.add(etaField3);
        etaPanel.add(etaField4);

        //*************************************************************************//
        // Her defineres et panel med comboboxes, som sammensættes med panelet med ETA   //
        // tekstfelter, og panelet hvor der står/indskrives info om destination          //
        // og draught.                                                                   //
        //*************************************************************************//
        boxPanel = new JPanel(false);
        boxPanel.setLayout(new GridLayout(0,1));
        boxPanel.add(shipBox);
        boxPanel.add(specialBox);
        boxPanel.add(sensorBox);
        boxPanel.add(destinationField);
        boxPanel.add(draughtField);
        boxPanel.add(etaPanel);

        //Her sammensættes panelerne boxpanel og labelpanel
        boxAndlabelPanel = new JPanel(false);
        boxAndlabelPanel.setLayout(new BoxLayout(boxAndlabelPanel, BoxLayout.X_AXIS));
        boxAndlabelPanel.add(labelPanel);
        boxAndlabelPanel.add(boxPanel);

        //*************************************************************************//
        // Her sammensættes de sidste paneler, nemlig boxAndlabelPanel og                //
        // buttonpanel. Dette er det panel som sammensætter de to store paneler          //
        //*************************************************************************//
        mainPanel = new JPanel(false);
        mainPanel.setLayout(new BorderLayout(0,0));
        mainPanel.setBorder(new EmptyBorder(10,10,10,10));
        mainPanel.add(boxAndlabelPanel, BorderLayout.NORTH);
        mainPanel.add(buttonPanel, BorderLayout.SOUTH);

        //Her opdateres indholdet af dialogboksen med alle paneler
        this.setSize(360,300);
        this.setTitle("Voyage related info");
        this.setContentPane(mainPanel);
    }
}
```

H.2.3 StaticDialog

```
import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
import InfoDialog;

//*****************************************************************************//
// I denne klasse oprettes de objekter som går igen i alle dialogbokse for statiske    //
// informationer.                                                              //
//*****************************************************************************//
public class StaticDialog extends InfoDialog
{
   protected ImageIcon ship;                  //Billede af antennens placering på skibet.
   protected JLabel shipName, callSign, MMSI, IMO, ship1;  //Labels som tilknyttes tekst.
   protected JLabel posAcc, empty1, empty2;
   protected JLabel a, b, c, d, m1, m2, m3, m4;
   protected JButton okButton, cancelButton;               //To knapper
   protected JPanel panel1, panel2, panel3, panel4, panel5; //Nogle paneler der kan inde-
   protected JPanel panel6, panel7, panel8, panel9;        //deholde knapper osv.
   protected JTextField shipName1, callSign1, MMSI1, IMO1, navSensor1;  //Nogle tekstfel-
   protected JTextField aFelt, bFelt, cFelt, dFelt;  //ter, hvor brugeren kan indtaste i.
   protected JRadioButton high, low;     //En Radiobutton er en form for knap som bruges.
   protected ButtonGroup toggle;                      //En ButtonGroup med navnet toggle.
   protected EmptyBorder border1, border2, border3; //Emptyborders som bruges til layout.

   //*****************************************************************************//
   // Her oprettes indholdet af selve dialogboksen                                //
   //*****************************************************************************//
   protected void createContents()
   {
      shipName = new JLabel("Ship Name: ", JLabel.LEFT);
      //Her oprettes labels som hører sammen med nedenstående tekstfelter
      callSign = new JLabel("Call sign: ", JLabel.LEFT);
      MMSI = new JLabel("MMSI- number: ", JLabel.LEFT);
      IMO = new JLabel("IMO- number:", JLabel.LEFT);
      posAcc = new JLabel("Position accuracy:  ", JLabel.LEFT);
      empty1 = new JLabel("", JLabel.LEFT);

      //Her oprettes tekstfelterne, til indtastning/visning af data
      shipName1 = new JTextField(100);
      callSign1 = new JTextField(100);
      MMSI1 = new JTextField(100);
      IMO1 = new JTextField(100);

      //Her oprettes to radiobuttons
      high = new JRadioButton("High < 10 m.");
      low = new JRadioButton("Low > 10 m.");

      //Her oprettes en buttongroup til de to radiobuttons
      toggle = new ButtonGroup();
      toggle.add(high);
      toggle.add(low);

      //Her sættes et billede ind af antennens position
      ship = new ImageIcon("ship.gif");
      ship1 = new JLabel(ship);

      //Her oprettes panel med forskellige labels
      panel1 = new JPanel(false);
      panel1.setLayout(new GridLayout(0,1));
      panel1.add(shipName);
      panel1.add(callSign);
      panel1.add(MMSI);
      panel1.add(IMO);
      panel1.add(posAcc);
      panel1.add(empty1);

      //Her oprettes panel med tekstfelterne
      panel2 = new JPanel(false);
      panel2.setLayout(new GridLayout(0,1));
      panel2.add(shipName1);
      panel2.add(callSign1);
      panel2.add(MMSI1);
      panel2.add(IMO1);
      panel2.add(high);
      panel2.add(low);

      // Her samles panel 1 og panel 2 til et samlet panel.
      panel3 = new JPanel();
      panel3.setLayout(new BoxLayout(panel3, BoxLayout.X_AXIS));
      panel3.add(panel1);
      panel3.add(panel2);

      //Der oprettes fire labels til info om antennens placering
```



```
      a = new JLabel("A: ", JLabel.LEFT);
      b = new JLabel("B: ", JLabel.LEFT);
      c = new JLabel("C: ", JLabel.LEFT);
      d = new JLabel("D:", JLabel.LEFT);

      //Her oprettes tekstfelter til indtastning/visning af antennens placering
      aFelt = new JTextField(10);
      bFelt = new JTextField(10);
      cFelt = new JTextField(10);
      dFelt = new JTextField(10);

      //Her oprettes labels til info om ankomsttid
      m1 = new JLabel("m", JLabel.LEFT);
      m2 = new JLabel("m", JLabel.LEFT);
      m3 = new JLabel("m", JLabel.LEFT);
      m4 = new JLabel("m", JLabel.LEFT);

      //Her oprettes panel med labels om antennens placering
      panel5 = new JPanel(false);
      panel5.setLayout(new GridLayout(0,1));
      panel5.add(a);
      panel5.add(b);
      panel5.add(c);
      panel5.add(d);

      //Her oprettes panel med tekstfelter info om antennens placering
      panel6 = new JPanel(false);
      panel6.setLayout(new GridLayout(0,1));
      panel6.add(aFelt);
      panel6.add(bFelt);
      panel6.add(cFelt);
      panel6.add(dFelt);

      //Her oprettes panel med ETA felterne
      panel7 = new JPanel(false);
      panel7.setLayout(new GridLayout(0,1));
      panel7.add(m1);
      panel7.add(m2);
      panel7.add(m3);
      panel7.add(m4);

      //Panel 5, panel 6 og panel 7 sammensættes til et panel.
      panel8 = new JPanel(false);
      panel8.setLayout(new BoxLayout(panel8, BoxLayout.X_AXIS));
      border2 = new EmptyBorder(40,0,40,20);
      panel8.setBorder(border2);
      panel8.add(panel5);
      panel8.add(panel6);
      panel8.add(panel7);

      //Her oprettes en OK- og en Cancel-knap
      okButton = new JButton("OK");
      cancelButton = new JButton("Cancel");

      //Knapperne samles i et panel
      panel9 = new JPanel(false);
      panel9.setLayout(new BoxLayout(panel9, BoxLayout.X_AXIS));
      border3 = new EmptyBorder(0,160,0,0);
      panel9.setBorder(border3);
      panel9.add(okButton);
      panel9.add(cancelButton);

      //Alle paneler samlet til et stort panel.
      panel4 = new JPanel(new BorderLayout(0,0));
      border1 = new EmptyBorder(0,0,0,10);
      panel4.setBorder(border1);
      panel4.add(panel3, BorderLayout.NORTH);
      panel4.add(ship1, BorderLayout.CENTER);
      panel4.add(panel8, BorderLayout.EAST);
      panel4.add(panel9, BorderLayout.SOUTH);

           this.setContentPane(panel4); //Indholdet af Dialogboksen opdateres.
   }
}
```

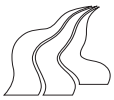
H.2.4 GenericWindowListener

```
import java.awt.event.*;

//*****************************************************************************//
// Applikation der lytter på mainvinduet.                                      //
//*****************************************************************************//

public class GenericWindowListener extends WindowAdapter
{
```

```
   //*********************************************************************************//
   // Kaldes når der trykkes på krydset. Programmet afsluttes når vinduet lukkes      //
   //*********************************************************************************//
   public void windowClosing (WindowEvent event)
   {
     System.exit(0);   // Programmet lukkes.
   }
}
```

H.2.5 LocalStaticDialog

```
import StaticDialog;
import java.awt.event.*;
import SortData;


//*************************************************************************************//
// Denne klasse specificerer StaticDialog til at være for eget skib og opretter en    //
// dialogboks. Den opbygges således, at de data, som indtastes gemmes, hvis der trykkes//
// på OK knappen og ignoreres, hvis der trykkes på cancel.                             //
//*************************************************************************************//
public class LocalStaticDialog extends StaticDialog
{

    //*********************************************************************************//
    // Hovedmetoden i klassen, hvor der bestemmes, hvornår de forskellige metoder      //
    // skal køres.                                                                     //
    //*********************************************************************************//
    public LocalStaticDialog()
    {
       this.initializeDialog();      //først intitialiseres dialogboksen mht. udseende
       this.createContents();        //derefter optegnes den.
       this.setSpecialParms();       //de pre-indtastede statiske informationer hentes fra
    }                                //et array.

    //*********************************************************************************//
    // Tekstfelterne opdateres med de data, som står i localShip, og actionlistener    //
    // knapperne tilføjes en actionlistener.                                           //
    //*********************************************************************************//
    private void setSpecialParms()
    {
       this.setTitle("Static Info - local ship");
       //shipName1.setText(mmsiString);       // ONLY FOR TESTING.

       // Her sættes indholdet af tekstfelterne så de passer til det lokale skib.
       shipName1.setText(SortData.localShip.getName());
       callSign1.setText(SortData.localShip.getCallsign());
       MMSI1.setText(SortData.localShip.getMMSInumber());
       IMO1.setText(SortData.localShip.getIMOnumber());
       aFelt.setText(SortData.localShip.getAntennaPositionA());
       bFelt.setText(SortData.localShip.getAntennaPositionB());
       cFelt.setText(SortData.localShip.getAntennaPositionC());
       dFelt.setText(SortData.localShip.getAntennaPositionD());

       // Her oprettes de specifikke actionlisteners.
       okButton.addActionListener(new LocalStaticListener());
       cancelButton.addActionListener(new LocalStaticListener());
       high.addActionListener(new LocalStaticListener());
       low.addActionListener(new LocalStaticListener());
    }

    //*********************************************************************************//
    // Her implementeres actionlistener. Hvis der trykkes på OK-knappen, gemmes de     //
    // indtastede data i localShip. Hvis der trykkes på Cancel afsluttes uden at gemme.//
    // Hvis radiobutton high er sat til høj, gemmes det som "1" ellers "0".            //
    //*********************************************************************************//
    private class LocalStaticListener implements ActionListener
    {
       private String var;

       //******************************************************************************//
       // Her tilknyttes de specifikke metoder de forskellige knapper.                //
       //******************************************************************************//
       public void actionPerformed(ActionEvent e)
       {
          Object source = e.getSource();

          if (source == okButton)     // Hvis der trykkes på OK, gemmes info i localShip
          {                           // i attributter for hvert af de forskellige info.
             var = shipName1.getText();
             SortData.localShip.setName(var);
             var = callSign1.getText();
             SortData.localShip.setCallsign(var);
             var = MMSI1.getText();
```

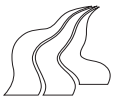


```
            SortData.localShip.setMMSInumber(var);
            var = IMO1.getText();
            SortData.localShip.setIMOnumber(var);
            var = aFelt.getText();
            SortData.localShip.setAntennaPositionA(var);
            var = bFelt.getText();
            SortData.localShip.setAntennaPositionB(var);
            var = cFelt.getText();
            SortData.localShip.setAntennaPositionC(var);
            var = dFelt.getText();
            SortData.localShip.setAntennaPositionD(var);
            dispose();
         }
         else if (source == cancelButton)    //Hvis der trykkes på cancel, lukkes
         {                                   //dialogboksen.
            dispose();
         }
         else if (source == high)
         {
            SortData.localShip.setPositionAccuracy("1");
         }
         else if (source == low)
         {
            SortData.localShip.setPositionAccuracy("0");
         }
         else {}
      }
   }
}
```

H.2.6 LocalVoyageDialog

```
import javax.swing.*;
import javax.swing.border.*;
import java.awt.event.*;
import SortData;
import VoyageDialog;

//*****************************************************************************//
// Denne klasse optegner resten af dialogboksen for eget skibs rejserelaterede data.   //
// Hvis der trykkes på OK-knappen gemmes data og menuen afsluttes.                 //
//*****************************************************************************//
public class LocalVoyageDialog extends VoyageDialog
{
   //*****************************************************************************//
   // Hovedmetodeen i denne klasse.                                               //
   //*****************************************************************************//
   public LocalVoyageDialog()
   {
      this.initializeDialog();                     //her initialiseres dialogboksen
      this.createContents();  //her opdateres indholdet af tekstfelter og valg.
      this.setSpecialParms();               //her optegnes resten af dialogboksen.
   }

   //*****************************************************************************//
   // Denne metode tilføjer metoder fra actionlistener som følge af at der trykkes på  //
   // nogle knapper eller foretages nogle valg.                                   //
   //*****************************************************************************//
   private void setSpecialParms()
   {
      this.setTitle("Voyage related info - local ship");
      if (SortData.localShip.getTypeOfShip() < 60)
      {
         specialBox.setEnabled(false);
      }
      shipBox.addActionListener(new LocalVoyageListener());
      okButton.addActionListener(new LocalVoyageListener());
      cancelButton.addActionListener(new LocalVoyageListener());
   }

   //*****************************************************************************//
   // Actionlistener implementeres, således at valgte info gemmes, hvis der trykkes på //
   // OK knappen og negligeres, hvis der trykkes på Cancel knappen. Begge instancer    //
   // lukker dialogboksen.                                                        //
   //*****************************************************************************//
   private class LocalVoyageListener implements ActionListener
   {
      private String text;
      private byte var, secondDigit;
      private final byte PASSENGER=60, CARGO=70, TANKER=80, OTHER=90;

      //*****************************************************************************//
      // De ovenstående, ønskede metodeer implementeres.                             //
      //*****************************************************************************//
      public void actionPerformed(ActionEvent e)
```

```
{
   Object source = e.getSource();

  if (source == okButton) //Hvis der trykkes på OK gemmes valget fra combobox i
  {                       //localShip.
     switch (shipBox.getSelectedIndex())
     {                    //Herunder gemmes indholdet af comboboxes med skibstype.
         case 0:
           break;
       case 1:
          var = (byte) 50;
          SortData.localShip.setTypeOfShip(var);
          break;
       case 2:
          var = (byte) 51;
          SortData.localShip.setTypeOfShip(var);
          break;
       case 3:
          var = (byte) 52;
          SortData.localShip.setTypeOfShip(var);
          break;
       case 4:
          var = (byte) 53;
          SortData.localShip.setTypeOfShip(var);
          break;
       case 5:
          var = (byte) 54;
          SortData.localShip.setTypeOfShip(var);
          break;
       case 6:
          var = (byte) 55;
          SortData.localShip.setTypeOfShip(var);
          break;
       case 7:
          var = (byte) 58;
          SortData.localShip.setTypeOfShip(var);
          break;
       case 8:
          var = (byte) 59;
          SortData.localShip.setTypeOfShip(var);
          break;
       case 9:
          secondDigit = (byte) specialBox.getSelectedIndex();
          SortData.localShip.setTypeOfShip((byte) (PASSENGER + secondDigit));
          break;
       case 10:
          secondDigit = (byte) specialBox.getSelectedIndex();
          SortData.localShip.setTypeOfShip((byte) (CARGO + secondDigit));
          break;
       case 11:
          secondDigit = (byte) specialBox.getSelectedIndex();
          SortData.localShip.setTypeOfShip((byte) (TANKER + secondDigit));
          break;
       case 12:
          secondDigit = (byte) specialBox.getSelectedIndex();
          SortData.localShip.setTypeOfShip((byte) (OTHER + secondDigit));
          break;
          default:
          break;
       }
    switch (sensorBox.getSelectedIndex())         //Indholdet af comboboxes om
    {                                             // navigation sensor gemmes.
       case 0:
          break;
       case 1:
          SortData.localShip.setNavSensor("1");
          break;
       case 2:
          SortData.localShip.setNavSensor("2");
          break;
       case 3:
          SortData.localShip.setNavSensor("3");
          break;
       case 4:
          SortData.localShip.setNavSensor("4");
          break;
       case 5:
          SortData.localShip.setNavSensor("5");
          break;
       case 6:
          SortData.localShip.setNavSensor("6");
          break;
       case 7:
          SortData.localShip.setNavSensor("7");
          break;
       default:
```



```
            break;
         }

        text = destinationField.getText();              // Indholdet af tekstfelt med
        SortData.localShip.setDestination(text);        // destination gemmes.

        text = draughtField.getText();                  // Info om draught gemmes.
        SortData.localShip.setDraught(text);

        text = etaField1.getText();                     // ETA info gemmes.
        text = (text + etaField2.getText());
        text = (text + etaField3.getText());
        text = (text + etaField4.getText());
        SortData.localShip.setETA(text);
        dispose();
     }
        else if (source == cancelButton)                // Hvis der trykkes på cancel
        {                                               //lukkes dialogboks uden at gemme.
          dispose();
        }
        else if (source == shipBox)
        {
          if (shipBox.getSelectedIndex() > 8)
          {
              specialBox.setEnabled(true);
          }
          else
          {
             specialBox.setEnabled(false);
          }
        }
        else {}
      }
   }
}
```

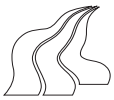
H.2.7 MessageDialog

```
import java.awt.event.ActionListener;
import javax.swing.JFrame;
import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;
import java.awt.event.*;

//*********************************************************************************//
// Denne klasse indeholder de fornødne metoder til oprettelse af de sidste modifice-   //
// ringer til dialogboksen med "Short binary message"                                  //
//*********************************************************************************//
public class MessageDialog extends JDialog
{
   private JTextArea Message;
   private JTextField mmsi;
   private JButton Exit;
   private JPanel Text, exitButton, mtext, app;
   private String message;
   private String MMSI;
   private String menuText;

   //*********************************************************************************//
   // I denne metode initialiseres dialogboksen mht. størrelse                        //
   //*********************************************************************************//
   private void initializeDialog()
   {
      this.setLocation(0,0);                       //Sætter placeringen af dette objekt.
      this.setSize(400,300);                        //Sætter størrelsen af dette objekt.
      this.setTitle(MMSI);   //Skriver MMSI-nummeret på skibet, der modtages besked fra.
      this.setResizable(false);                   //Sætter dialogboksens størrelse fast.
   }

   //*********************************************************************************//
   // Denne metode optegner dialogboksens indhold, en knap og et tekstfelt.           //
   // Tekstfeltets indhold hentes fra                                                 //
   //*********************************************************************************//
   private void createContents()
   {
      JTextArea Message = new JTextArea(message, 10, 55);          //Opretter textareal.
      Message.setEditable(false); //Fryser indholdet af boksen så det ikke kan editeres.
      Message.setLineWrap(true);               //Sørger for linieskift, hvis nødvendigt.
      Message.setWrapStyleWord(true);            //Ord deles ikke af tvungen linieskift.
      Text = new JPanel(false);        //Opretter et panel til tekstfeltet med beskeden.
      Text.setBorder(new EmptyBorder(10,10,10,10));
      Text.add(Message);
```

```
      Exit = new JButton("Exit");   //Opretter Exit knap til afslutning af message boks.
      exitButton = new JPanel();
      exitButton.add (Exit);

      app = new JPanel();                          //Panel som placerer textbox og knap.
      app.setLayout (new BorderLayout(0,0));
      app.add (Exit, BorderLayout.SOUTH);
      app.add (Text, BorderLayout.NORTH);
      setContentPane(app);

      Exit.addActionListener(new MessageListener());//Tilknytter knappen actionlistener.
   }
   //*************************************************************************************//
   // I denne metode kaldes metoderne til oprettelse af dialogboksen.                     //
   // Metoden kaldes med parametrene mmsi1 og data.                                       //
   //*************************************************************************************//
   public MessageDialog(String mmsi1, String data)
   {
      MMSI = mmsi1;
      message = data;
      initializeDialog();
      createContents();
   }

   //*************************************************************************************//
   // Actionlistener, som kaldes, hvis der trykkes på exit knappen                        //
   //*************************************************************************************//
   private class MessageListener implements ActionListener
   {


      //*************************************************************************************//
      // Metode, som udføres, hvis actionlistener kaldes. Lukker dialogboksen.            //
      //*************************************************************************************//
      public void actionPerformed (ActionEvent event)
      {
         Object source = event.getSource();
         if (source == Exit)
         {
            dispose();
         }
      }
   }
}
```

H.2.8 OtherStaticDialog

```
import StaticDialog;
import java.awt.event.*;
import SortData;

//*************************************************************************************//
// Denne klasse indeholder de fornødne metoder til oprettelse af de sidste modifice-   //
// ringer til dialogboksen for andre skibes statiske informationer.                    //
//*************************************************************************************//
public class OtherStaticDialog extends StaticDialog
{
   private int target;              //en attribut, som peger på en plads i skibsarrayet

   //*************************************************************************************//
   // I denne metode kaldes metoderne til oprettelse af dialogboksen.                     //
   //*************************************************************************************//
   public OtherStaticDialog(int t)
   {
      target = t;                    //en attribut, som peger på en plads i skibsarrayet
      this.initializeDialog();
      this.createContents();
      this.setSpecialParms();
   }

   //*************************************************************************************//
   // Metoden herunder optegner den specifikke dialogboks og fylder dem med              //
   // informationer om det specifikke skib.                                               //
   //*************************************************************************************//
   private void setSpecialParms()
   {
      this.setTitle("Static Info - foreign ship");       //skriver titlen på dialogboksen

      // Tekstfelter tilpasses til denne klasse så de ikke kan editeres
      shipName1.setEditable(false);
      callSign1.setEditable(false);
      MMSI1.setEditable(false);
      IMO1.setEditable(false);
      aFelt.setEditable(false);
      bFelt.setEditable(false);
      cFelt.setEditable(false);
```



```
      dFelt.setEditable(false);
      panel9.remove(okButton);      // OK knappen fjernes.
      cancelButton.setText("EXIT"); // Cancel knappen ændres til en EXIT knap.

      // Her sættes indholdet af tekstfelterne så de passer til det pågældende skib.
      // informationerne fås fra klassen Sortdata, som indeholder shipArray.
      shipName1.setText(SortData.shipArray[target].getName());
      callSign1.setText(SortData.shipArray[target].getCallsign());
      MMSI1.setText(SortData.shipArray[target].getMMSInumber());
      IMO1.setText(SortData.shipArray[target].getIMOnumber());
      aFelt.setText(SortData.shipArray[target].getAntennaPositionA());
      bFelt.setText(SortData.shipArray[target].getAntennaPositionB());
      cFelt.setText(SortData.shipArray[target].getAntennaPositionC());
      dFelt.setText(SortData.shipArray[target].getAntennaPositionD());

      cancelButton.addActionListener(new OtherStaticListener());     //tilknytter knappen
                                                                      //en actionlistener
   }

   //*************************************************************************//
   // Actionlistener, som kaldes, hvis der trykkes på exit knappen            //
   //*************************************************************************//
   private class OtherStaticListener implements ActionListener
   {

      //*************************************************************************//
      // Metode, som udføres, hvis actionlistener kaldes. Lukker dialogboksen.   //
      //*************************************************************************//
      public void actionPerformed(ActionEvent e)
      {
         Object source = e.getSource();

         if (source == cancelButton)
         {
            dispose();
         }
         else {}
      }
   }
}
```

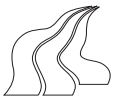
H.2.9 InfoDialog

```
import javax.swing.*;

//*****************************************************************************//
// Klasse som nedarves til alle dialogboksene, for at sikre samme størrelse m.m.//
//*****************************************************************************//
public class InfoDialog extends JDialog
{
   public final short WIDTH = 300;
   public final short HEIGHT = 400;

   //*************************************************************************//
   // Her optegnes dialogboksens skabelon                                     //
   //*************************************************************************//
   protected void initializeDialog()
   {
      this.setLocation(400,50);                  // Sætter placeringen af dialogboxen
      this.setSize(WIDTH,HEIGHT);                // Sætter størrelsen af dialogboxen.
      this.setResizable(false);                  // Gør at brugeren ikke kan ændre på
                                                 // størrelsen af dialogboxen.
   }
}
```

H.2.10 OtherVoyageDialog

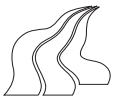
```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import SortData;
import VoyageDialog;

//*****************************************************************************//
// Denne klasse indeholder de fornødne metoder til oprettelse af de sidste modifice-  //
// ringer til dialogboksen for andre skibes rejserelaterede informationer      //
//*****************************************************************************//
public class OtherVoyageDialog extends VoyageDialog
{
   private int target;              //en attribut, som peger på en plads i skibsarrayet

   //*************************************************************************//
   // I denne metode kaldes metoderne til oprettelse af dialogboksen fra andre klasser.//
```

```
//*****************************************************************************//
public OtherVoyageDialog(int t)
{
   target = t;                  // Peger på en plads i skibsarrayet
   this.initializeDialog();
   this.createContents();
   this.setSpecialParms();
}

//*****************************************************************************//
// Metoden herunder optegner den specifikke dialogboks og fylder den med       //
// informationer om det specifikke skib.                                       //
//*****************************************************************************//
private void setSpecialParms()
{
   String eta;

   this.setTitle("Voyage related info - other ship");       //Navnet på dialogboksen.
   buttonPanel.remove(okButton);                              //Fjerner "OK"-knappen.
   cancelButton.setText("EXIT");     //Ændrer teksten på "cancel"-knappen til "EXIT".

   //her fades tekstfelterne i dialogboksen og fastsætter den indeholdende tekst
   destinationField.setEnabled(false);
   destinationField.setBackground(new Color(212,208,200));
   draughtField.setEnabled(false);
   draughtField.setBackground(new Color(212,208,200));
   etaField1.setEnabled(false);
   etaField1.setBackground(new Color(212,208,200));
   etaField2.setEnabled(false);
   etaField2.setBackground(new Color(212,208,200));
   etaField3.setEnabled(false);
   etaField3.setBackground(new Color(212,208,200));
   etaField4.setEnabled(false);
   etaField4.setBackground(new Color(212,208,200));

   sensorBox.setEnabled(false);    //her fastsættes indholdet af sensorbox panelet

   //Informationer om navigational status for pågældende skib, som target peger på,
   //hentes ind og combobox'ens indhold fastsættes. Er realiseret ved en switch.
   switch(SortData.shipArray[target].getNavSensor())
      {
         case 0:
            sensorBox.setSelectedIndex(0);
            break;
         case 1:
            sensorBox.setSelectedIndex(1);
            break;
         case 2:
            sensorBox.setSelectedIndex(2);
            break;
         case 3:
            sensorBox.setSelectedIndex(3);
            break;
         case 4:
            sensorBox.setSelectedIndex(4);
            break;
         case 5:
            sensorBox.setSelectedIndex(5);
            break;
         case 6:
            sensorBox.setSelectedIndex(6);
            break;
         case 7:
            sensorBox.setSelectedIndex(7);
            break;
         default:
            sensorBox.setSelectedIndex(0);
      }

   shipBox.setEnabled(false);
   specialBox.setEnabled(false);

   //Informationer om skibstype hentes ind og skrives på rette plads
   if ((SortData.shipArray[target].getTypeOfShip() >= 50) &&
   (SortData.shipArray[target].getTypeOfShip() < 60))
   {
      switch(SortData.shipArray[target].getTypeOfShip())
      {
         case 50:
            shipBox.setSelectedIndex(1);
            break;
         case 51:
            shipBox.setSelectedIndex(2);
            break;
         case 52:
            shipBox.setSelectedIndex(3);
```



```
            break;
         case 53:
            shipBox.setSelectedIndex(4);
            break;
         case 54:
            shipBox.setSelectedIndex(5);
            break;
         case 55:
            shipBox.setSelectedIndex(6);
            break;
         case 56:
            shipBox.setSelectedIndex(0);
            break;
         case 57:
            shipBox.setSelectedIndex(0);
            break;
         case 58:
            shipBox.setSelectedIndex(7);
            break;
         case 59:
            shipBox.setSelectedIndex(8);
            break;
      }
   }

   //Informationer om skibstype hentes ind og skrives på rette plads
   else if ((SortData.shipArray[target].getTypeOfShip() >= 60) &&
   (SortData.shipArray[target].getTypeOfShip() < 70))
   {
      shipBox.setSelectedIndex(9);
      switch(SortData.shipArray[target].getTypeOfShip())
      {
         case 60:
            specialBox.setSelectedIndex(0);
            break;
         case 61:
            specialBox.setSelectedIndex(1);
            break;
         case 62:
            specialBox.setSelectedIndex(2);
            break;
         case 63:
            specialBox.setSelectedIndex(3);
            break;
         case 64:
            specialBox.setSelectedIndex(4);
            break;
         case 65:
            specialBox.setSelectedIndex(5);
            break;
         case 66:
            specialBox.setSelectedIndex(6);
            break;
         case 67:
            specialBox.setSelectedIndex(7);
            break;
         case 68:
            specialBox.setSelectedIndex(0);
            break;
         case 69:
            specialBox.setSelectedIndex(8);
            break;
      }
   }

   //Informationer om skibstype hentes ind og skrives på rette plads
   else if ((SortData.shipArray[target].getTypeOfShip() >= 70) &&
   (SortData.shipArray[target].getTypeOfShip() < 80))
   {
      shipBox.setSelectedIndex(10);
      switch(SortData.shipArray[target].getTypeOfShip())
      {
         case 70:
            specialBox.setSelectedIndex(0);
            break;
         case 71:
            specialBox.setSelectedIndex(1);
            break;
         case 72:
            specialBox.setSelectedIndex(2);
            break;
         case 73:
            specialBox.setSelectedIndex(3);
            break;
         case 74:
            specialBox.setSelectedIndex(4);
```

```
            break;
         case 75:
            specialBox.setSelectedIndex(5);
            break;
         case 76:
            specialBox.setSelectedIndex(6);
            break;
         case 77:
            specialBox.setSelectedIndex(7);
            break;
         case 78:
            specialBox.setSelectedIndex(0);
            break;
         case 79:
            specialBox.setSelectedIndex(8);
            break;
      }

   }

   //Informationer om skibstype hentes ind og skrives på rette plads
   else if ((SortData.shipArray[target].getTypeOfShip() >= 80) &&
   (SortData.shipArray[target].getTypeOfShip() < 90))
   {
   shipBox.setSelectedIndex(11);
               switch(SortData.shipArray[target].getTypeOfShip())
      {
         case 80:
            specialBox.setSelectedIndex(0);
            break;
         case 81:
            specialBox.setSelectedIndex(1);
            break;
         case 82:
            specialBox.setSelectedIndex(2);
            break;
         case 83:
            specialBox.setSelectedIndex(3);
            break;
         case 84:
            specialBox.setSelectedIndex(4);
            break;
         case 85:
            specialBox.setSelectedIndex(5);
            break;
         case 86:
            specialBox.setSelectedIndex(6);
            break;
         case 87:
            specialBox.setSelectedIndex(7);
            break;
         case 88:
            specialBox.setSelectedIndex(0);
            break;
         case 89:
            specialBox.setSelectedIndex(8);
            break;
      }
   }

   //Informationer om skibstype hentes ind og skrives på rette plads
   else if ((SortData.shipArray[target].getTypeOfShip() >= 90) &&
   (SortData.shipArray[target].getTypeOfShip() < 100))
   {
      shipBox.setSelectedIndex(12);
      switch(SortData.shipArray[target].getTypeOfShip())
      {
         case 90:
            specialBox.setSelectedIndex(0);
            break;
         case 91:
            specialBox.setSelectedIndex(1);
            break;
         case 92:
            specialBox.setSelectedIndex(2);
            break;
         case 93:
            specialBox.setSelectedIndex(3);
            break;
         case 94:
            specialBox.setSelectedIndex(4);
            break;
         case 95:
            specialBox.setSelectedIndex(5);
            break;
         case 96:
```



```
               specialBox.setSelectedIndex(6);
               break;
            case 97:
               specialBox.setSelectedIndex(7);
               break;
            case 98:
               specialBox.setSelectedIndex(0);
               break;
            case 99:
               specialBox.setSelectedIndex(8);
               break;
         }
      }
      else
      {
         shipBox.setSelectedIndex(0);
         specialBox.setSelectedIndex(0);
      }
      //Information, som skal stå om destination og draught, skrives ind på rette plads.

      destinationField.setText(SortData.shipArray[target].getDestination());
      draughtField.setText(SortData.shipArray[target].getDraught());

      //tekstfeltet om ETA fyldes ud med de oplyste informationer fra pågældende skib
      eta = SortData.shipArray[target].getETA();
      etaField1.setText(eta.substring(0,2));             //værdi for antal dage indsættes
      etaField2.setText(eta.substring(2,4));            //værdi for antal timer indsættes
      etaField3.setText(eta.substring(4,6));         //værdi for antal minutter indsættes
      etaField4.setText(eta.substring(6,8));         //værdi for antal sekunder indsættes

      // Her opsættes actionlisteners til denne dialogboks.
      cancelButton.addActionListener(new OtherVoyageListener());
   }
   //*************************************************************************//
   // Actionlistener, som kaldes, hvis der trykkes på exit knappen            //
   //*************************************************************************//
   private class OtherVoyageListener implements ActionListener
   {

      //*************************************************************************//
      // Metode, som udføres, hvis actionlistener kaldes. Lukker dialogboksen.   //
      //*************************************************************************//
      public void actionPerformed(ActionEvent e)
      {
         Object source = e.getSource();

         if (source == cancelButton)
         {
            dispose();
         }
      }
   }
}
```

H.2.11 Vectorfield

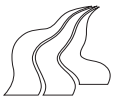
```
import java.awt.*;
import MainWindow;

//*****************************************************************************//
// Klasse, der kaldes fra MainWindow. I denne klasse instantieres vektorfeltet og      //
// genoptegner det hver gang der modtages noget på serielporten.                //
//*****************************************************************************//
public class Vectorfield extends Canvas
{
   public static Graphics circle;

   public void paint(Graphics page)
   {
      Color blue = new Color(64,104,159);    // En farve laves (blå).
      circle = page;
      setSize(570,570);                      // Størrelsen af kanvaset.
      page.setColor(blue);
      page.fillOval(0,0,570,570);
      page.setColor(Color.black);
      drawVector();
   }

   public void drawVector()
   {
      long localLatitude, localLongitude;
      long otherLatitude, otherLongitude;
      double localSpeed, otherSpeed;
      double localCourse, otherCourse;
      long SOGpix, xLocalNew, yLocalNew;
```

```
      double scale, localLatDeg;

      localLatitude = SortData.localShip.getLatitude();
      localLongitude = SortData.localShip.getLongitude();
      localSpeed = SortData.localShip.getSOG();
      localCourse = SortData.localShip.getCOG();

      localLatDeg = (localLatitude/(60 * 10000));  // Eget skibs breddegrad i grader.
      scale = Math.cos(localLatDeg*(3.1416/180));  // Skaleringsfaktor til longitude.

      SOGpix = 2 * Math.round(localSpeed);  //Længden af egen vektor i pixels.
      xLocalNew = -Math.round(SOGpix * Math.cos((localCourse+90)*3.1416/180)*scale) + 285;
      yLocalNew = -Math.round(SOGpix * Math.sin((localCourse+90)*3.1416/180)) + 285;
      //Tegn en vektor fra pixel midten til de nye x,y koordinater.
      circle.drawLine(285,285,(int) xLocalNew,(int) yLocalNew);
      circle.fillOval(285-3,285-3,6,6); //Tegner en cirkel i centrum.

      for (byte x = 0; x < 20; x++) // Alle skibe i MMSIarrayet gennemløbes.
      {
         if (SortData.MMSIArray[x] != -1) // Hvis skibet eksisterer.
         {
            double deltaX, deltaY;
            long xOtherStart, yOtherStart;
            double xOtherEnd, yOtherEnd;

            if (SortData.shipArray[x].getCollisionCourse())
            {
               circle.setColor(Color.red);   // Tegn med rødt hvis der er kollisionskurs.
            }
            otherLatitude = SortData.shipArray[x].getLatitude();
            otherLongitude = SortData.shipArray[x].getLongitude();
            otherSpeed = SortData.shipArray[x].getSOG();
            otherCourse = SortData.shipArray[x].getCOG();
            // Beregner afstanden fra eget til andet skib:
            deltaY = (otherLatitude - localLatitude);
            deltaX = (otherLongitude - localLongitude)*scale;
            // Placering af andet skib på skærmen:
            xOtherStart = Math.round(deltaX*0.01) + 285;
            yOtherStart = -Math.round(deltaY*0.01) + 285;
            // Slutpunkt for vektoren på skærmen beregnes:
            SOGpix = 2 * Math.round(otherSpeed);
            xOtherEnd = -Math.round(SOGpix * Math.cos((otherCourse+90)*3.1416/180)*scale)
                        + xOtherStart;
            yOtherEnd = -Math.round(SOGpix * Math.sin((otherCourse+90)*3.1416/180))
                        + yOtherStart;
            // Her tegnes vektoren på skærmen:
            circle.drawLine((int) xOtherStart,(int) yOtherStart,
                            (int) xOtherEnd,(int) yOtherEnd);
            circle.fillOval((int) xOtherStart-3,(int) yOtherStart-3,6,6);
            circle.setColor(Color.black); // Farven sættes til sort igen.
         }
      }
   }
}
```

H.3 SHIP package

H.3.1 Data

```
//*****************************************************************************//
// Denne klasse danner grundlaget for behandlingen af data fra "SortData". Når data    //
// kommer fra "SortData" indsættes de i "dataArray", således at de er forberedt for    //
// senere sortering og lagring.                                                //
//*****************************************************************************//
public class Data
{
    protected String[] dataArray= new String[14]; //Opretter "dataArray" af 14 strings.
    protected short charCount=0;            //Angiver den nuværende char i "frame".
    protected short x=0;                   //Angiver den nuværende string i "dataArray".
    protected int lengthOfFrame=0;          //Antal chars i "frame".

   //*****************************************************************************//
   // Opdeler den indkommende string/frame i dataklynger, som indsættes på pladser i   //
   // dataArray.                                                                  //
   //*****************************************************************************//
   public void sliceFrame(String frame)
   {
      for (short count=0 ; count<14 ; count++) // Tømmer dataArray
      {
         dataArray[count]=("");
      }
      lengthOfFrame = frame.length()-1;      // Bestemmer længden af "frame".

      while (charCount<=lengthOfFrame)       // Data fra "frame" indsættes i "dataArray".
```



```
      {
         if (frame.charAt(charCount)!=',')   // Data der ikke er et komma indsættes i
         {                                   // dataArray.
            dataArray[x] = (dataArray[x] + frame.charAt(charCount));
            charCount++;
         }
         else                                // Hvis der er tale om et komma, flyttes
         {                                   // til næste plads i "dataArray".
            x++;
            charCount++;
         }
      }
      charCount=0;                           // Løkken nulstilles til næste gang.
      x=0;
      lengthOfFrame=0;
   }
}
```

H.3.2 GPSData

```
import Data;

//*****************************************************************************//
// Varetager den endelige sortering og lagring af data fra GPS'en.             //
//*****************************************************************************//

public class GPSData extends Data
{
   //**************************************************************************//
   // De forskellige data hentes ud fra "dataArray", og gemmes på deres respektive     //
   // pladser i "LocalShip".                                                   //
   //**************************************************************************//
   public void sendGPSData()
   {
      SortData.localShip.setUTC(dataArray[1]);
      SortData.localShip.setLatitude(dataArray[2], dataArray[3]);
      SortData.localShip.setLongitude(dataArray[4], dataArray[5]);
      SortData.localShip.setSOG(dataArray[6]);
      SortData.localShip.setCOG(dataArray[7]);
      SortData.localShip.detectCollision();
   }
}
```

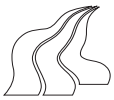
H.3.3 LocalShipInfo

```
import ShipInfo;
import CollisionDetection;

//*****************************************************************************//
// Objektet af denne klasse har til opgave at konvertere og lagre div. informationer   //
// om eget skib.                                                               //
//*****************************************************************************//
public class LocalShipInfo extends ShipInfo
{
   //**************************************************************************//
   // Kalder detectCollision i alle andre eksisterende skibe                   //
   //**************************************************************************//
   public void detectCollision()
   {
      CollisionDetection.mainWindow.refreshLocalWindow();
      for (int k=0; k < 20; k++)
      {
         if (SortData.MMSIArray[k] != -1)
         {
            SortData.shipArray[k].detectCollision(k);
         }
      }
   }

   //**************************************************************************//
   // Latitude                                                                 //
   //**************************************************************************//
   private int latitude;

   public void setLatitude(String inlatitude, String indirection)
   {
      int latitude1;                          // Minutter.
      String latitude2;                       // Grader.
      String latitude3;                       // Minutter.

      latitude2 = inlatitude.substring(0,2);       // Graderne trækkes ud af string.
      this.latitude = Integer.parseInt(latitude2); // Der konverteres fra string til int.
      latitude = 60*latitude*10000;                // Graderne omregnes til minutter.
      latitude3 = (inlatitude.substring(2,4) + inlatitude.substring(5,9)); //Ny string.
```

```
      latitude1 = Integer.parseInt(latitude3);     // Minutterne trækkes ud af stringen.
      latitude = latitude + latitude1;             // Minutter i alt beregnes.
      if (indirection.charAt(0) == 'S') latitude = latitude*(-1); // Retning angives.
   }
   public int getLatitude(){ return latitude; }

   //*********************************************************************************//
   // Longitude                                                                       //
   //*********************************************************************************//

   private int longitude;

   public void setLongitude(String inlongitude,String indirection)
   {
      int longitude1;                              // Minutter.
      String longitude2;                           // Grader.
      String longitude3;                           // Minuter.

      longitude2 = inlongitude.substring(0,3);     // Graderne trækkes ud af string.
      this.longitude = Integer.parseInt(longitude2);  // Der konverteres til int.
      longitude = 60*longitude*10000;              // Der omregnes til minutter.
      longitude3 = (inlongitude.substring(3,5) + inlongitude.substring(6,10));
                                   // Der fjernes et komma, og tallene lægges sammen,
                                   // hvilket svarer til at gange med 10.000.
      longitude1 = Integer.parseInt(longitude3);   // Minutterne trækkes ud af string.
      longitude = longitude + longitude1;          // Minutter i alt beregnes.
      if (indirection.charAt(0) == 'W')            // Retning angives vha. fortegn.
      {
         longitude = longitude*(-1);
      }
   }
   public int getLongitude(){ return longitude; }


   //*********************************************************************************//
   // UTC                                                                             //
   //*********************************************************************************//
   private String utc;

   public void setUTC(String UTC)
   {
      this.utc = UTC;
   }
   public String getUTC(){ return utc; }


   //*********************************************************************************//
   // Course Over Ground                                                              //
   //*********************************************************************************//
    protected float COG;

    public void setCOG(String COG)
    {
      this.COG = Float.parseFloat(COG);
    }
    public float getCOG(){ return COG; }

   //*********************************************************************************//
   // Speed Over Ground                                                               //
   //*********************************************************************************//

    protected float SOG;

    public void setSOG(String SOG)
    {
      this.SOG = Float.parseFloat(SOG);
    }

    public float getSOG(){ return SOG; }

}
```

H.3.4 MessageData

```
import javax.swing.*;
import Data;
import MessageDialog;

//*********************************************************************************//
// Håndterer beskeder fra andre skibe, ved at udskrive dem på skærmen.             //
//*********************************************************************************//
public class MessageData extends Data
{

   //*********************************************************************************//
```

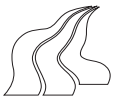


```
   // Udvælger beskeden fra "dataArray", og udskriver den i en dialogbox på skærmen.   //
   //*************************************************************************************//
   public void sendMessage(int target)
   {
      MessageDialog message = new MessageDialog(dataArray[1], dataArray[2]);

      // Sætter et Windows Look and feel.
      try
      {
         UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
         SwingUtilities.updateComponentTreeUI(message);
      }
      catch (Exception exc)
      {
         System.err.println("Could not load LookAndFeel: ");   // Hvis ikke understøttet.
      }
      message.show();   // Viser dialogboxen på skærmen.
   }
}
```

H.3.5 OtherShipInfo

```
import ShipInfo;
import CollisionDetection;

//*************************************************************************************//
// Denne klasse nedarver fra ShipInfo, og overtager dermed dennes metoder. Desuden     //
// er der tilføjet en række metoder, som kun benyttes til andre skibe.                 //
//*************************************************************************************//
public class OtherShipInfo extends ShipInfo
{

   //*************************************************************************************//
   // Kollisionsdetektionen er samlet i denne metode, som kalder de egentlige          //
   // beregningsmetoder.                                                                //
   //*************************************************************************************//
   public void detectCollision(int target)
   {
      int x1= SortData.localShip.getLongitude();            // Længdegraden af eget skib.
      int y1= SortData.localShip.getLatitude();             // Breddegraden af eget skib.
      if (calculateDistance(x1, y1, longitude, latitude))   // Kontrollerer afstanden.
         {
            calculateCollision(x1, y1, longitude, latitude);// Beregner kollisionsfare.
            CollisionDetection.mainWindow.refreshOtherWindow();// Vinduet gentegnes.
         }
         else
         {
            SortData.MMSIArray[target] = -1; // Hvis det er udenfor radius, slettes skib.
         }
   }

   //*************************************************************************************//
   // Beregner om det pågældende skib er inden for den kritiske radius på 3 sømil.     //
   //*************************************************************************************//
   private boolean calculateDistance(int x1, int y1, int x2, int y2)
   {
      int lattInDegrees;
      double cosBredde, deltaX, deltaY, xk, yk, distance;
      lattInDegrees = (y1/(60 * 10000));                // Minutter omregnes til grader.
      cosBredde = Math.cos(lattInDegrees*(3.1416/180)); // Afstand mellem længdegraderne.
      deltaX = Math.abs(x1-x2);
      deltaY = Math.abs(y1-y2);
      xk = Math.pow((cosBredde*deltaX), 2);
      yk = Math.pow(deltaY, 2);
      distance = Math.sqrt(xk + yk)/10000;              // Afstand i sømil.
      if (distance < 3)
         return true;
      else
         return false;
   }

   //*************************************************************************************//
   // Foretager den egentlige kollisionsberegning                                       //
   //*************************************************************************************//
   private void calculateCollision(float x1, float y1, float x2, float y2)
   {
      float COG1 = 450 - SortData.localShip.getCOG();    //
      float SOG1 = SortData.localShip.getSOG();
      float COG2 = 450 - COG;
      float SOG2 = SOG;
      double x1New, x2New, y1New, y2New;                  // Koordinater efter flytning.
      double xIntersect, yIntersect;                     // Skæringpunktets koordinater.
      double a1, a2, b1, b2;                 // Koefficienter i ligningerne for linierne.
      double mag1X, mag1Y, mag2X, mag2Y;           // Bruges under udregningen af vektor.
```

```
   //Bruges under udregningen af vektor.
   double ship1ToIntersect, ship2ToIntersect, rel1, rel2;
   // Skibenes startpunkter:
   x1 = x1/10000; //Der omregnes fra 1/100000 dele.
   y1 = y1/10000;
   x2 = x2/10000;
   y2 = y2/10000;

   x1New = x1 + SOG1 * Math.cos((COG1) * 3.14159265 / 180); // egen ny x-placering.
   y1New = y1 + SOG1 * Math.sin((COG1) * 3.14159265 / 180); // egen ny y-placering.

   x2New = x2 + SOG2 * Math.cos((COG2) * 3.14159265 / 180); // ny x-placering.
   y2New = y2 + SOG2 * Math.sin((COG2) * 3.14159265 / 180); // ny y-placering.

   a1 = Math.tan(COG1 * 3.14159265 / 180); // Hældningskoefficienten bestemmes.
   a2 = Math.tan(COG2 * 3.14159265 / 180); // Hældningskoefficienten bestemmes.

   b1 = y1 - a1 * x1; // b1 bestemmes ved indsættelse af et punkt.
   b2 = y2 - a2 * x2; // b2 bestemmes ved indsættelse af et punkt.

   xIntersect = (b2 - b1) / (a1 -a2); // Skæringspunktets x-koordinat beregnes.
   yIntersect = a1 * xIntersect + b1; // Skæringspunktets y-koordinat beregnes.

   // Det undersøges om begge skibe har retning mod skæringspunktet:

   if (
         (
           ((xIntersect < x1) && (x1New < x1) || (xIntersect > x1) && (x1New > x1)) ||
           ((yIntersect < y1) && (y1New < y1) || (yIntersect > y1) && (y1New > y1))
         )
       &&
         (
           ((xIntersect < x2) && (x2New < x2) || (xIntersect > x2) && (x2New > x2)) ||
           ((yIntersect < y2) && (y2New < y2) || (yIntersect > y2) && (y2New > y2))
         )
      )

   {
      ship1ToIntersect =      // Afstanden fra skib1 til skæringspunktet.
      Math.sqrt(Math.pow((xIntersect - x1), 2) + Math.pow((yIntersect - y1), 2));

      ship2ToIntersect =      // Afstanden fra skib2 til skæringspunktet.
      Math.sqrt(Math.pow((xIntersect - x2), 2) + Math.pow((yIntersect - y2), 2));

      rel1 = ship1ToIntersect / SOG1;     // Hvor hurtigt skib1 er ved punktet.
      rel2 = ship2ToIntersect / SOG2;     // Hvor hurtigt skib2 er ved punktet.

      // Bestemmelse af om de er der på "samme" tid:
      if (0.1 < (rel1 / rel2) && (rel1 / rel2) < 1.5)
      {
         collisionCourse = true; // Hvis de er for tæt på hinanden i skæringspunktet.
      }
   }
}


//*****************************************************************************//
// Collision Course                                                            //
//*****************************************************************************//
private boolean collisionCourse;

public void setCollisionCourse(boolean collisionCourse)
{
   this.collisionCourse = collisionCourse;
}

public boolean getCollisionCourse(){ return collisionCourse; }


//*****************************************************************************//
// Latitude                                                                    //
//*****************************************************************************//
private int latitude;

public void setLatitude(String latitude)
{
   this.latitude = Integer.parseInt(latitude);
}

public int getLatitude(){ return latitude; }


//*****************************************************************************//
// Longitude                                                                   //
//*****************************************************************************//
private int longitude;
```



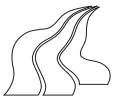
```
    public void setLongitude(String longitude)
    {
    this.longitude = Integer.parseInt(longitude);
    }

    public int getLongitude(){ return longitude; }


    //*****************************************************************************//
    // TimeStamp                                                                   //
    //*****************************************************************************//
    private byte timeStamp;

    public void setTimeStamp(String timeStamp)
    {
       this.timeStamp = Byte.parseByte(timeStamp);
    }

    public byte getTimeStamp(){ return timeStamp; }


    //*****************************************************************************//
    // Course Over Ground                                                          //
    //*****************************************************************************//
    protected float COG;

    public void setCOG(String COG)
    {
       this.COG = (Float.parseFloat(COG))/10;
    }
    public float getCOG(){ return COG; }


    //*****************************************************************************//
    // Speed Over Ground                                                           //
    //*****************************************************************************//
    protected float SOG;

    public void setSOG(String SOG)
    {
       this.SOG = Float.parseFloat(SOG)/10;
    }

    public float getSOG(){ return SOG; }


    //*****************************************************************************//
    // Message                                                                     //
    //*****************************************************************************//
    private String message;

    public void setMessage(String message)
    {
       this.message = message;
    }

    public String getMessage(){ return message; }


    //*****************************************************************************//
    // Heading                                                                     //
    //*****************************************************************************//
    private short heading;

    public void setHeading(String heading)
    {
       this.heading = Short.parseShort(heading);
    }

    public short getHeading(){ return heading; }


    //*****************************************************************************//
    // TurnRate                                                                    //
    //*****************************************************************************//
    private byte turnRate;

    public void setTurnRate(String turnRate)
    {
       this.turnRate = Byte.parseByte(turnRate);
    }

    public byte getTurnRate(){ return turnRate; }

}
```

H.3.6 PositionData

```
import Data;

//*********************************************************************************//
// Denne klasse skriver positionsinformation til skibe                             //
//*********************************************************************************//
public class PositionData extends Data
{
   //*********************************************************************************//
   // Metode der sender data til det specifikke skibsobjekt.                          //
   //*********************************************************************************//
   public void sendPositionData(int target)
   {
      System.out.println("Target ship at place: " + target);
      SortData.shipArray[target].setMMSInumber(dataArray[1]);
      SortData.shipArray[target].setNavStatus(dataArray[2]);
      SortData.shipArray[target].setTurnRate(dataArray[3]);
      SortData.shipArray[target].setSOG(dataArray[4]);
      SortData.shipArray[target].setPositionAccuracy(dataArray[5]);
      SortData.shipArray[target].setLongitude(dataArray[6]);
      SortData.shipArray[target].setLatitude(dataArray[7]);
      SortData.shipArray[target].setCOG(dataArray[8]);
      SortData.shipArray[target].setHeading(dataArray[9]);
      SortData.shipArray[target].detectCollision(target);

      System.out.println("MMSI: " + SortData.shipArray[target].getMMSInumber());
      System.out.println("Navstatus: " + SortData.shipArray[target].getNavStatus());
      System.out.println("Turnrate: " + SortData.shipArray[target].getTurnRate());
      System.out.println("SOG: " + SortData.shipArray[target].getSOG());
      System.out.println("PosAcc: " + SortData.shipArray[target].getPositionAccuracy());
      System.out.println("Longitude: " + SortData.shipArray[target].getLongitude());
      System.out.println("Latitude: " + SortData.shipArray[target].getLatitude());
      System.out.println("COG: " + SortData.shipArray[target].getCOG());
      System.out.println("Heading: " + SortData.shipArray[target].getHeading());
   }
}
```

H.3.7 ShipInfo

```
//*********************************************************************************//
// Denne klasse nedarves til LocalShipInfo og OtherShipInfo                        //
//*********************************************************************************//
public class ShipInfo
{
   //*********************************************************************************//
   // Navigational Status                                                             //
   //*********************************************************************************//
   private byte navStatus;

   public void setNavStatus(String innavStatus)
   {
      navStatus = Byte.parseByte(innavStatus);
   }

   public byte getNavStatus() { return navStatus; }

   //*********************************************************************************//
   // Position Accuracy                                                               //
   //*********************************************************************************//
   private boolean positionAccuracy;
   private char chr =1;

   public void setPositionAccuracy(String inpositionAccuracy)
   {
      chr = inpositionAccuracy.charAt(0);
      if (chr == '0') positionAccuracy = false;
      else positionAccuracy = true;
   }

   public boolean getPositionAccuracy(){ return positionAccuracy; }

   //*********************************************************************************//
   // MMSI number                                                                     //
   //*********************************************************************************//
    private String MMSInumber;

    public void setMMSInumber(String MMSInumber)
    {
      this.MMSInumber = MMSInumber;
    }

    public String getMMSInumber() { return MMSInumber; }
```

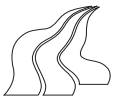


```
//*********************************************************************************//
// IMO number                                                                      //
//*********************************************************************************//
 private String IMOnumber;

 public void setIMOnumber(String IMOnumber)
 {
   this.IMOnumber = IMOnumber;
 }

 public String getIMOnumber(){ return IMOnumber; }

//*********************************************************************************//
// Call Sign                                                                       //
//*********************************************************************************//
 private String callsign;

 public void setCallsign(String callsign)
 {
   this.callsign = callsign;
 }

 public String getCallsign(){ return callsign; }

//*********************************************************************************//
// Name                                                                            //
//*********************************************************************************//
 private String name;

 public void setName(String name)
 {
   this.name = name;
 }

 public String getName(){ return name; }

//*********************************************************************************//
// Type of Ship and Cargo Type                                                     //
//*********************************************************************************//
private byte typeOfShip=0;

public void setTypeOfShip(String typeOfShip)             // Fra CI
{
   this.typeOfShip = Byte.parseByte(typeOfShip);
}

public void setTypeOfShip(byte type)                     // Fra GUI
{
   this.typeOfShip = type;
}

public byte getTypeOfShip(){ return typeOfShip; }

//*********************************************************************************//
// Antenna Position A                                                              //
//*********************************************************************************//
private String antennapositionA;

public void setAntennaPositionA(String antennapositionA)
{
   this.antennapositionA = antennapositionA;
}

public String getAntennaPositionA(){ return antennapositionA; }

//*********************************************************************************//
// Antenna Position B                                                              //
//*********************************************************************************//
private String antennapositionB;

public void setAntennaPositionB(String antennapositionB)
{
   this.antennapositionB = antennapositionB;
}

public String getAntennaPositionB(){ return antennapositionB; }

//*********************************************************************************//
// Antenna Position C                                                              //
//*********************************************************************************//
private String antennapositionC;

public void setAntennaPositionC(String antennapositionC)
{
   this.antennapositionC = antennapositionC;
```

```
   }

   public String getAntennaPositionC(){ return antennapositionC; }

   //*****************************************************************************//
   // Antenna Position D                                                          //
   //*****************************************************************************//
   private String antennapositionD;

   public void setAntennaPositionD(String antennapositionD)
   {
      this.antennapositionD = antennapositionD;
   }

   public String getAntennaPositionD(){ return antennapositionD; }

   //*****************************************************************************//
   // Navigational Sensor                                                         //
   //*****************************************************************************//
   private byte navSensor=0;

   public void setNavSensor(String navSensor)
   {
      this.navSensor = Byte.parseByte(navSensor);
   }

   public byte getNavSensor(){ return navSensor; }

   //*****************************************************************************//
   // ETA                                                                         //
   //*****************************************************************************//
   private String ETA="00000000";

   public void setETA(String ETA)
   {
      this.ETA = ETA;
   }

   public String getETA(){ return ETA; }

   //*****************************************************************************//
   // Draught                                                                     //
   //*****************************************************************************//
   private String draught="0";

   public void setDraught(String draught)
   {
      this.draught = draught;
   }

   public String getDraught(){ return draught; }

   //*****************************************************************************//
   // Destination                                                                 //
   //*****************************************************************************//
   private String destination="0";

   public void setDestination(String destination)
   {
      this.destination = destination;
   }

   public String getDestination(){ return destination; }

}
```

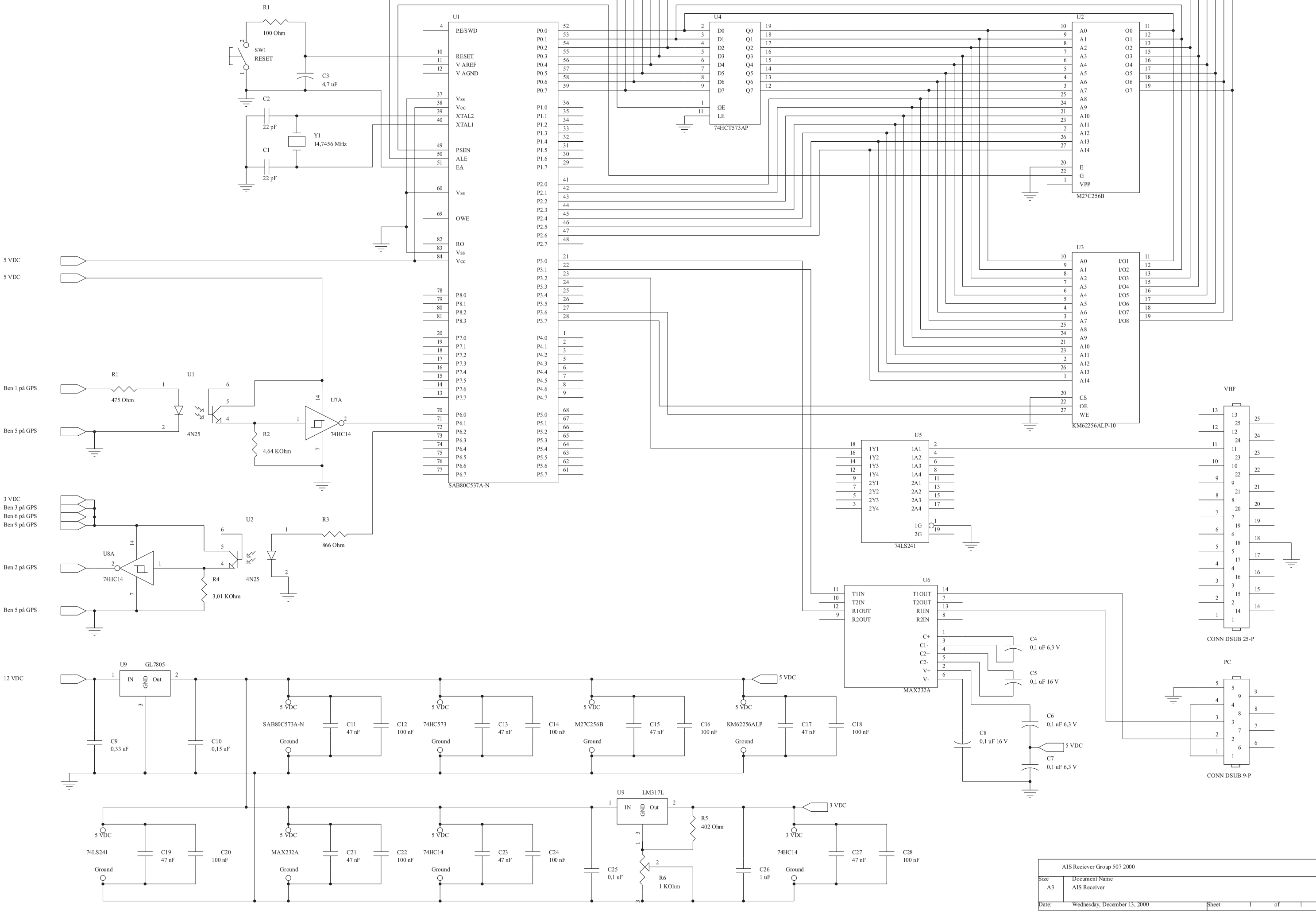
H.3.8 CollisionDetection

```
import java.io.*;
import java.util.*;
import javax.comm.*;
import javax.swing.*;

import SerialDriver;
import MainWindow;
import GenericWindowListener;

public class CollisionDetection
{
   static CommPortIdentifier portId;   // Et objekt der kan kontrollere en serielport.
   static Enumeration portList;  // En abstrakt datatype der indholder forskellig typer.

   public static MainWindow mainWindow;
```



```
//*********************************************************************************//
// Mainløkken der sørger for at afvikle programmet                                 //
//*********************************************************************************//
   public static void main(String[] args)
   {
      System.out.println("\n***************************************");
      System.out.println("* Ship Collision detection:             *");
      System.out.println("* By group 507:                         *");
      System.out.println("* Thomas Søhus, Kenneth Kristensen      *");
      System.out.println("* Christian Jensen, Peter Nielsen       *");
      System.out.println("* Lars Jackobsen & Jan Ozimek           *");
      System.out.println("***************************************");

      portList = CommPortIdentifier.getPortIdentifiers();   // En liste med porte hentes.

      while (portList.hasMoreElements())  // Hvis der er porte i systemet.
      {
         portId = (CommPortIdentifier) portList.nextElement(); // Næste port.
         if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) // Hvis serielport.
         {
            if (portId.getName().equals("COM1"))   // Hvis serielporten er COM1.
            {
               SerialDriver reader = new SerialDriver(portId); // Objekt der læser COM1.
            }
            else {}
         }
         else {}
      }

      mainWindow = new MainWindow();   // Opretter et nyt mainvindue.
      mainWindow.addWindowListener(new GenericWindowListener());  // En WindowListner.

      try   // Sætter et Window look and feel.
      {
         UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
         SwingUtilities.updateComponentTreeUI(mainWindow);
      }
      catch (Exception exc)
      {
         System.err.println("Could not load LookAndFeel: ");
      }
      mainWindow.show();
   }
}
```

I Diagram



AIS Reciever Group 507 2000		
Size: A3	Document Name: AIS Receiver	Rev: 1.0
Date: Wednesday, December 13, 2000	Sheet 1 of 1	