Titelblad

Titel: Kollisionsdetektor for lystbåde
Tema: Apparat- og/eller maskinkonstruktion
Projektperiode: E5 (04.09.2002 til 19.12.2002)

Projektgruppe
E501

Deltagere
Michael Cidlik
Jesper Dueholm
Michael Eriksen
Tommy Jensen
Mads Kelter-Wesenberg
Jacob Larsen

Vejleder
Hans Ebert

Censor
Finn Hedehus

Oplagstal
9

Sidetal
141

Bilag
CD, manual og hæfte
med kildekode

Afsluttet den
19-12-2002

Synopsis

Denne rapport har til formål, at designe, konstruere og teste et system til lystbåde, der kan alarmere i tilfælde af en mulig kollision med et andet skib.
For at muliggøre dette tages der udgangspunkt i AIS (Automatic Identification System).

Projektet opdeles i tre hoveddele, en AIS-simulator, en GPS (Global positioning System)-simulator og en brugerterminal.
AIS-simulatoren realiseres på en uafhængig pc og er udelukkende softwarebaseret. AIS-informationerne indeholder, foruden oplysninger om omkringliggende skibes kurs og fart, en række statiske data som eksempelvis skibsnavne og identifikationsdata. Disse informationer afsendes til brugerterminalen med bestemte intervaller.
GPS-simulatoren er i lighed med AIS-simulatoren udelukkende softwarebaseret, på en uafhængig pc. Denne simulator skal udelukkende simulere og sende informationer om eget skibs position, kurs og fart m.m. til brugerterminalen.
For både AIS- og GPS-simulatorerne gælder det, at simulationerne tager udgangspunkt i informationer fra simulatorernes respektive databaser.
Brugerterminalen modtager henholdsvis simuleret AIS- og GPS-information. På baggrund af dette beregner brugerterminalen, hvorvidt der er risiko for kollision med andre skibe. Er dette tilfældet alarmeres brugeren vha. brugerfladen.
Forbindelsen mellem simulatorerne og brugerterminalen foregår over en seriel RS232 forbindelse.
Hele systemet er testet og sammenlignet med de, i kravspecifikationen, opstillede krav.

Konklusionen på rapporten er, at det er lykkedes, at konstruere en kollisionsdetektor der lever op til kravene.

Summary

The purpose of this report is, to design, construct and test a system for a pleasure boat, which is capable of raising an alarm, in case of collision with another ship.
To achieve this purpose the AIS (Automatic Identification System) is introduced.

The project is divided into three main sections, an AIS-simulator, a GPS (Global positioning System) simulator and an user terminal.
The AIS-simulator is realized on a independent computer and is exclusively software based. The AIS-information contains information regarding the surrounding ships, their course and speed, and a number of static data such as name and identification number. This information is transmitted to the user terminal in certain intervals.
Similar to the AIS-simulator, the GPS-simulator is exclusively software based, on a independent computer. This simulator simulates and transmits information regarding own ship's position, course, speed etc. to the user terminal.
Both the AIS- and the GPS-simulator receives this information from their databases.
The user terminal receives the simulated AIS- and GPS information. With this information the user terminal calculates whether there is a risk of collision with other ships. In case of collision the user is alarmed through the user interface.
The connection between the simulators and the user terminal consists of a RS-232 serial connection.
The entire system is tested and compared to the demands in the specification.
The conclusion of the report is, that it was possible to construct a collision detector that performs within the specification parameters.

Forord

Denne rapport er udarbejdet på Aalborg universitet af gruppe E501 i perioden fra den 04.09.2002 til 19.12.2002.

Læsevejledning

Rapporten er opbygget i en hovedrapport samt appendiks i nævnte rækkefølge.
Kildehenvisningerne er anført i parentes med en forkortelse, der refererer
til litteraturlisten, og sidehenvisning til den angivne kilde.
Eks: …er på 1575,42 MHz.[GPS s.91-100].
Kilderne, i litteraturlisten, er opstillet på følgende måde: Forkortelse, Titel, ISBN, Forlag, Udgave, Udgivelse år, Forfatter og evt. Link.
Endvidere er nummereringen af figurer og tabeller sket fortløbende gennem de enkelte kapitler i rapporten.
Bilag er vedlagt på medfølgende cd-rom i pdf-format, placeret på bagsiden af rapporten.
Henvisning til CD'en gøres ved henvisning til den specifikke placering på CD'en, i litteraturlisten.
Endelig er de enkelte testkoder placeret på samme CD. Henvisningen til disse er foretaget i forbindelse med de enkelte testafsnit.
Kildekoden er placeret i separat hæfte.

Aalborg Universitet d. 19. december 2002

Michael Cidlik _____

Jesper Dueholm _____

Michael Eriksen _____

Tommy Jensen _____

Mads Kelter-Wesenberg _____

Jacob Larsen _____

Indholdsfortegnelse

1 Indledning

I lighed med trafik på land, er også al trafik til søs pålagt at følge visse regler, gældende alle farvande, uanset fartøjets størrelse og nationalitet.
Disse regler er udarbejdet af "International Maritime Organization" (IMO) under navnet "Convention on the international regulations for preventing collisions at sea" eller i daglig tale "Søvejsreglerne" (se appendiks 0).
Til trods for dette regelsæt sker det stadig, at fartøjer kolliderer, hvorfor der løbende udvikles nye metoder og teknologier, med netop det formål at afværge kollisioner mellem fartøjer. Af sådanne kan eksempelvis nævnes radio og radarsystemer.
En ny metode er et automatisk identifikationssystem "Automatic Identification System" (AIS). Dette system virker ved at sende en lang række informationer ud i æteren og ligeledes modtage tilsvarende informationer fra omkringliggende AIS-kompatible fartøjer. Disse informationer indeholder bl.a. oplysninger vedrørende det enkelte skibs position, kurs, hastighed, type, last og destinationshavn. Sammenholdes omkringliggende fartøjers AIS-informationer med egne oplysninger kan en eventuel kollisionsrisiko udregnes. I tilfælde af kollisionsrisiko skal systemet alarmere fartøjets mandskab.
Oplysninger vedrørende egen position indhentes vha. en GPS modtager, medens AIS-informationerne sendes og modtages vha. en VHF-radio.
Fra januar 2003 skal alle IMO-skibe (Erhvervsskibe) være udstyret med AIS.
Foruden at kunne afværge kollisioner fartøjer imellem, giver AIS også andre fordele. Herunder mulighed for at spore miljøsyndere samt muligheder for automatisk kommunikation mellem fartøj og instanser såsom havne og "Vessel Traffic Service"-anlæg (VTS).
Dette projekt har til formål at designe og konstruere en AIS-baseret kollisionsdetektor til lystbåde. Til dokumentation af projektet vil blive benyttet "Struktureret Program Udvikling"- modellen (SPU), hvorimod "Unified Modeling Language"-modellen (UML) vil lægge grundlaget for design af softwaren.

2 Kommunikation

Dette afsnit vil indeholde beskrivelser af anvendte kommunikationsmetoder, herunder standarden ITU-R M.1371[AIS], der specificerer AIS. Endvidere beskrives NMEA 0183[NMEA], der er en ofte anvendt standard ved dataoverførsel maritimt udstyr imellem.
Der tages udgangspunkt i ovennævnte to standarder, i det disse tilsammen lægger grundlaget for, hvorledes behandling og overførelse af data skal foregå.
Disse analyser baseres på Open Systems Interconnection (OSI) modellen, der er en generel model til beskrivelse af datakommunikation.

2.1 OSI modellen

OSI modellen er opbygget af 7 lag, der beskriver de forskellige abstraktionsniveauer der anvendes til beskrivelse af kommunikationen mellem 2 systemer. OSI modellens 7 lag er illustreret i tabel 2.1, og er beskrevet i appendiks B.

7	Applikation
6	Præsentation
5	Session
4	Transport
3	Netværk
2	Data link
1	Fysisk lag

Tabel 2.1: OSI modellen

2.2 ITU-R M.1371 (AIS)

AIS standarden beskriver et system til automatisk identifikation af skibe. Dette system skal køre autonomt. Endvidere skal alle skibe i systemet sende data indeholdende informationer omkring position og identitet, på foruddefinerede radiokanaler i VHF-båndet. De forskellige skibe arrangerer selv hvilke skibe der sender på hvilke tidspunkter efter en metode kaldet ”Self organised time division multiple access”(SOTDMA).
AIS standarden dækker lag 1-4 i OSI modellen. Disse vil efterfølgende blive gennemgået med udgangspunkt i standarden ITU-R M.1371.

2.2.1 Fysisk lag

Det fysiske lag er ansvarlig for overførsel af rå bit over et fysisk medie. De vigtigste data og krav for dette lag er opsummeret i tabel 2.2.

Beskrivelse	Minimum	Maksimum
Afstand mellem kanaler i VHF-båndet (kHz)	12,5	25
AIS kanal 1 (MHz)	161,975	161,975
AIS kanal 2 (MHz)	162,025	162,025
Kanal båndbredde (kHz)	12,5	25
Bithastighed (bit/s)	9600	9600
Træningssekvens (bit)	24	32
Sendeeffekt (W)	1	25

Tabel 2.2: De vigtigste parametre for det fysiske lag

2.2.2 Link lag

Det materiale i link laget der har relevans for projektet, beskrives i det følgende. For yderligere information henvises til [AIS].
Link laget deles op i følgende 3 underlag:

Underlag 1: Medium access control (MAC)

MAC underlaget beskriver hvorledes der opnås adgang til transmissions mediet, hvilket i det aktuelle tilfælde består af VHF kanalen. Metoden er bygget op omkring TDMA (time division multiple access) med en fælles tidsreference. Som reference bruges UTC(Coordinated universal time).
Systemet arbejder med en ramme (frame), der spænder over 1 minut, og deles op i 2250 slots. En sendestation får adgang til kanalen i et antal slots, hvor der ikke er andre sendere, der sender. En ramme starter samtidig med UTC minuttet.

Underlag 2: Data link service (DLS)

DLS sørger for:

- Aktivering og deaktivering af linket.
- Data overførsel
- Fejldetektion og kontrol

Aktivering og deaktivering foregår ved hjælp af MAC underlaget. Systemet undersøger hvorvidt hvert enkelt slot er internt allokeret, eksternt allokeret eller frit. Et internt allokeret slot betyder, at dette slot benyttes til at sende data. Et frit eller eksternt allokeret slot betyder derimod, at dette slot har modtaget data eller er klar til at modtage data, hvorfor systemet skal lytte efter data fra andre data link brugere.
Dataoverførsel foregår normalvis efter en protokol baseret på HDLC(high-level data link control), og specificerer tre forskellige pakketyper. Information packet er den pakketype, der benyttes i det aktuelle tilfælde, dog med den undtagelse, at kontrolfeltet udelades. Adressefeltet udelades ligeledes, da der er tale om broadcast. Strukturen for typen information packet er skitseret i figur 2.1.

Træningssekvens	Startflag	Data	FCS	Slutflag	Buffer

Figur 2.1: Opbygningen af en HDLC pakke af typen information.

Transmissionen i et slot starter med, at der gøres plads til opstart af senderen. En VHF-sender har brug for opstart svarende til 8 bit, før denne er oppe på fuld styrke.
Selve pakken starter med en træningssekvens bestående af 24 bit vekslende mellem 1 og 0, der har til formål at synkronisere sender og modtager.
Startflaget er et standard HDLC flag af 8 bits længde, der består af en standard bitsekvens, 01111110(7Eh), der ikke kan forekomme i resten af pakken. For at sikre, at denne bitsekvens ikke kan forekomme i resten af datapakken, udføres ”bitstuffing”. Dette fungerer på en sådan måde, at der efter 5 på hinanden følgende 1-taller automatisk vil blive indsat et 0. Startflaget kan således ikke forekomme i selve datapakken.
Datadelen er som standard 168 bit lang, og indholdet er udefineret i DLS. Det er dog muligt at sende data der fylder mere end 168 bit. I dette tilfælde kan der allokeres op til 5 på hinanden følgende slots til formålet. Data med tilhørende headers må dog under ingen omstændigheder fylde ud over disse 5 slots, da dette kan forstyrre andre sendinger.
Frame check sequence (FCS) delen består af et 16 bit cyclic redundancy check (CRC), der har til formål at verificere gyldigheden af data.

Endvidere er slutflaget identisk med startflaget.
Bufferen indsættes for at gøre plads til bitstuffing, forsinkelser i forbindelse med afstand, repeatere og unøjagtigheder i forbindelse med synkronisering. Bufferen fylder 24 bit.
Pakkens opbygning er opsummeret i tabel 2.3.

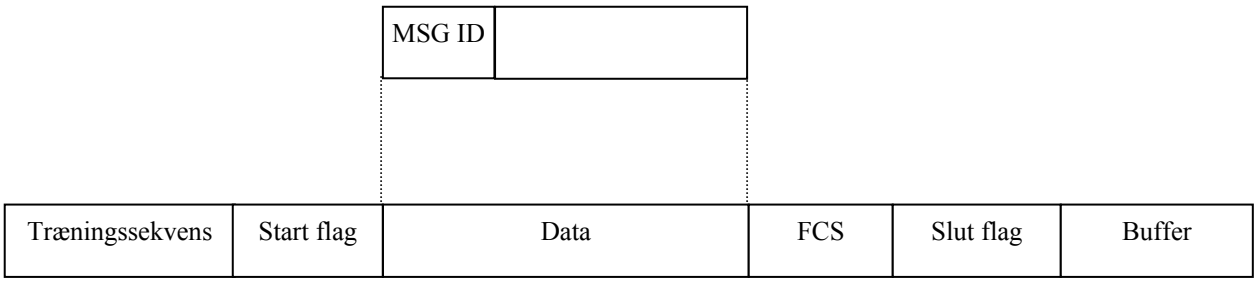
Senderopstart	8 bit
Træningssekvens	24 bit
Start flag	8 bit
Data	168 bit
CRC	16 bit
Slut flag	8 bit
Buffer	24 bit
Total	256 bit

Tabel 2.3: Opsummering af en pakke der fylder en slot

Underlag 3: Link management entity (LME)

LME underlaget kontrollerer DLS, MAC og det fysiske lag. Det er her det defineres hvilke parametre, de data der skal afsendes skal overholde, og det er også her det defineres hvordan modtaget data er opbygget. Da parametrene for afsendelse af data ikke er relevante her, vil disse ikke blive behandlet. For yderligere information henvises til ITU-R M.1371 standarden [AIS]. Det er således kun parametrene for modtaget data, der efterfølgende vil blive behandlet.

Datadelen af en pakke starter med et ”Message ID”, som vist på figur 2.2.



Figur 2.2: Fast struktur for alle typer af datapakker

Dette ”Message ID” består af 6 bit, og spænder over værdier fra 0-63. Strukturen for de efterfølgende data er afhængige af dette ID.
De pakkertyper, der har relevans i projektet, har MSG ID 1,2,3 og 5, hvor 1, 2 og 3 alle beskriver en positionsrapport, og har samme opbygning, hvorimod pakke type 5 beskriver statiske data om afsenderskibet.
Opbygningen af pakke 1, 2 og 3 kan ses i tabel 2.4.

Parameter	Længde i bits	Beskrivelse
Message ID	6	Pakke type for denne pakke. Kan være 1, 2 eller 3
Data terminal udstyr	1	0=tilgængelig, 1= ikke tilgængelig
Data indikator	1	Indikerer om data er klar til afsendelse(0=ikke klar, 1=klar)
MMSI	30	Skibets MMSI nummer
Navigations status	2	0=undervejs, 1=for anker, 2=ikke under kommando, 3=begrænset bevægelighed
Drejningshastighed	8	±127 grader/min(-128 indikerer ikke tilgængelig)
Fart	10	Fart over land i 1/10 knobs intervaller (0-102,3 knob)
Positions nøjagtighed	1	1=høj (<10m), 0=lav(>10m)
Længde	28	1/10000 min (±180º, øst = positiv, vest = negativ)
Bredde	27	1/10000 min (±90º, nord = positiv, syd = negativ)
Kurs	12	Kurs over grunden i 1/10º (0-3599)
Retning	9	Grader (0-359) (511 indikerer ikke tilgængelig)
Tid	6	UTC minut for generering af rapporten (0-59, eller 63, hvis positionssystemet ikke kører)
Rest	9	Ikke brugt
Kommunikations status	18	Ikke relevant her
Total antal bit	168	

Tabel 2.4: Oversigt over pakke 1, 2 og 3

Opbygningen af pakke 5 kan ses i tabel 2.5.

Parameter	Længde i bits	Beskrivelse
Message ID	6	Pakke type for denne pakke.
Rest	2	Ikke brugt. Skal være 0
MMSI	30	Skibets MMSI nummer
Rest	2	Ikke brugt. Skal være 0
IMO nummer	30	Max 9 numeriske karakterer
Kalde signal	36	6×6 bit ASCII karakterer
Navn	120	Max 20 6 bit ASCII karakterer
Skibs- og lasttype	8	Defineret i [AIS]
GNSS antenne position	30	Defineret i [AIS] (global navigation satellite system)
Navigations sensor type	4	
Forventet ankomsttidspunkt	20	MMDDTTMM
Dybgang	8	1/10 m, max 25,5
Destination	120	Max 20 6 bit ASCII karakterer
Total antal bit	416	

Tabel 2.5: Oversigt over pakke 5

2.2.3 Netværkslag

Netværkslaget sørger blandt andet for at håndtere trængsel på link'et, hvilket sker efter den såkaldte Robin Hood algoritme. Denne algoritme sørger for, at hvis belastningen på link'et overstiger 90 % for en enkelt station, vælges den station der ligger længst væk fra senderen, hvorefter dennes slots

overtages af den førstnævnte station. Det sker på den betingelse, at afstanden stationerne imellem er over 12 sømil, og den samme station kun bruges 1 gang pr. ramme.

2.2.4 Transportlag

Transportlaget er ansvarligt for omdannelse af datapakker til brugerflade, og hver pakke svarer til én besked til brugerfladen.
Til at overføre sætninger til brugerfladen benyttes protokollen NMEA 0183 med de tilføjelser, der findes i [AIS 2001]. Der benyttes endvidere RS232 til fysisk protokol i det aktuelle tilfælde.
Endelig foreskriver ITU-R M.1371 standarden, at interfacet til brugerfladen skal kunne køre 38400bps.

2.3 NMEA 0183

NMEA 0183 er en standard, der bruges til at overføre data maritimt udstyr imellem. Der anvendes en struktur, der er baseret på ASCII karakterer arrangeret i sætninger. Det overordnede format for disse sætninger følger den følgende model:

$aabbb,c—c,d—d*hh<CR><LF>

Forklaring til denne sætning ses i tabel 2.6.

Tegn	Beskrivelse
$	Start indikator
Aa	Identifikation af afsender
Bbb	Sætningstype
,	Feltadskiller
c—c,d—d	Dataindhold i sætning opdelt i felter.
*	Checksumadskiller. Indikerer start på checksum
Hh	Checksum for sætningen
<CR><LF>	Afslutning af sætning

Tabel 2.6: Oversigt over en NMEA 0183 sætning.

Maksimal længde af sætning inklusiv start- og slutindikator er 82 tegn.
I forbindelse med AIS beskeder, der overføres over NMEA, modificeres formatet. I dette tilfælde bruges ikke ”$” som start indikator, men i stedet ”!”. Som afsenderidentifikation bruges AI, der indikerer, at sætningen kommer fra et AIS signal. Som sætningstype bruges VDM, der angiver en sætning med indkapslede data fra et VHF data link. En sætning fra et AIS signal vil således have følgende format:

!AIVDM,x,y,z,a,s—s,f*hh<CR><LF>

Data indholdet i sætningen er defineret i tabel 2.7.

Tegn	Beskrivelse
x	Antal sekvenser til denne sætning.
y	Sekvenssnummer i aktuel sætning.
z	Sekventielt ID. Identificerer hvilken sætning sekvensen hører til.
a	AIS kanal.
s—s	Indkapslede data.
f	Fyldbits til indkapsling.

Tabel 2.7: Oversigt over VDM sætningen.

2.3.1 Indkapsling af AIS data

Datadelen af en AIS sætning indkapsles ved, at dataene deles op i dele af 6 bits størrelse. ASCII tegnene med værdier i området 30h-57h og 60h-77h defineres efterfølgende til at svare til hver sin 6 bit kombination. Dette er nødvendigt idet AIS-informationer ellers ikke kan overføres vha. en tegnbaseret kommunikationsprotokol, så som RS-232.

3 Global Positioning System

Kollisionsdetektoren kræver, at kende egen position. Til dette formål kan GPS anvendes. Dette system kan med en fejlmargin på max 15 meter, fastslå modtagerens position ethvert sted på jorden.

3.1 Virkemåde

Virkemåden for GPS-systemet fungerer ved, at der i kredsløb om Jorden er 24 satellitter, der konstant sender data mod Jorden. Disse data opfanges af GPS-modtageren, og ud fra de modtagne data fra satellitterne, kan GPS-modtageren beregne sin egen position.
Måden hvorpå dette kan lade sig gøre, er ved at GPS-modtageren altid kender de forskellige satellitters position i rummet. Satellitternes bane om Jorden, kan beregnes ud fra en model. Der kan selvfølgelig ikke tages højde for uregelmæssigheder i en sådan model, derfor opdateres afvigelser fra satellittens forventede bane til den reelle gennem data sendt fra satellitterne til modtagerne konstant. Modtageren vil således altid kende hver satellits position i rummet.
Det er ud fra kendskabet satellitternes position, at modtagerens position kan bestemmes, med en nøjagtighed på ned til15 meter [Tri]. Når GPS-modtageren får data fra en GPS-satellit kan forsinkelsestiden fra afsendelse af signalet, til modtagelsen bestemmes. Og ud fra kendskabet til denne tid kan den aktuelle afstand til satellitten, der afsendte signalet bestemmes.

Metoden, der anvendes til afstandsberegninger, illustreres her v.h.a. et eksempel:

En GPS-satellit, udsender følgende signal til tiden 17:04:59:

H K W E E G J W G Q J D J U W T X U…

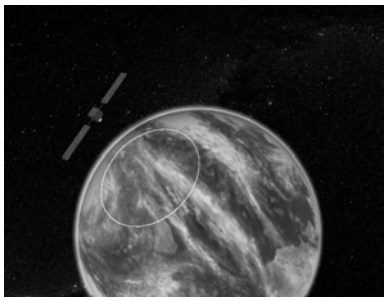
Ovenstående data genereres ud fra en ligning, der gør at dataene der sendes/modtages altid er unik i forhold til tiden og afsendelses-satellitten.
Modtageren vil nu sammenligne denne sekvens, med dens egen kopi af den. Modtagerens egen kopi, startede i tiden 17:04:59, mens sekvensen modtaget fra satellitten vil være forsinket i forhold til denne, på grund af tiden det har taget signalet at tilbagelægge afstanden mellem satellit og modtager. Nedenfor er illustreret hvordan de to signaler vil se ud for modtageren:

	↓tiden 17:04:59	↓12/100 sekund efter tiden 17:04:59
Modtager:	H K W E E G J W G Q J D J U W T X U…	
Sender:		H K W E E G J W G Q J D J U W T X U…

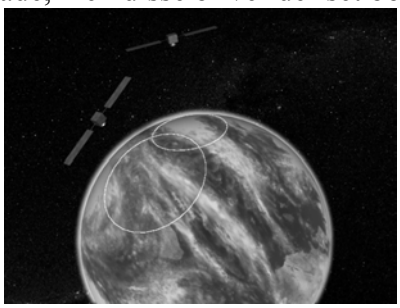
Modtageren bestemmer, at det har taget signalet 12/100 sekunder at tilbagelægge afstanden fra satellitten til modtageren.
Da radiobølger har en hastighed på 300.000 km/sek. viser tidsforskellen dermed, at den pågældende satellit er $\frac{12}{100} s \cdot 300000 \, {}^{km}\!/_{s}$ = 36.000 km væk fra modtageren.

Nu kender GPS-modtageren afstanden til satellitten, samt dennes position i rummet. Med afstanden og positionen kendt, vil modtageren nu kunne indskrænke mulige positioner til at omfatte en kugle, rundt om satellitten, da modtagerens position kan ligge hvor som helst i afstanden fra satellitten.

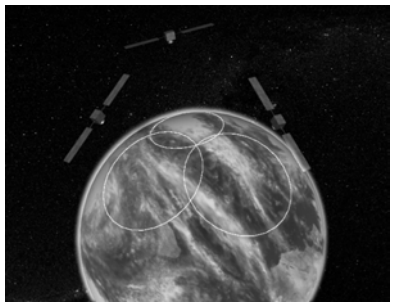
Jorden repræsenterer samtidig en kugle i rummet. Holdes kuglen, satellitten repræsenterer, op mod jorden, vil disse skære hinanden i en cirkel. Dette er illustreret på figur 3.1, og GPS-modtagerens position, vil kunne ligge hvor som helst på denne cirkel.
Modtagerens position kan derfor ikke fastslås ud fra 1 satellit. Der søges derfor efter endnu en GPS-satellit, og samme procedure som i eksemplet ovenfor gennemgås for denne satellit. Dette vil tegne endnu en cirkel på jordens overflade, som det ses på figur 3.2. Nu kan antallet af mulige positioner indskrænkes til 2 skæringspunkterne mellem de 2 cirkler fremkommet fra satellit-positionerne.
Men for at bestemme hvilken af de 2 mulige positioner, der er tale om, skal endnu en satellit bestemmes, hvilket er illustreret på figur 3.3. Positionen for modtageren kan nu bestemmes til skæringspunktet mellem disse tre cirkler. Der vil naturligvis fremkomme andre skæringspunkter udenfor og indenfor jordens overflade, men disse bliver der set bort fra.



Figur 3.1: 1 GPS-satellit



Figur 3.2: 2 GPS-satellitter



Figur 3.3: 3 GPS-satellitter

For at kunne fastslå selve positionen af modtageren er der altså brug for signaler fra 3 satellitter. Kravet for en GPS-modtager er dog oppe på signaler fra 4 satellitter. Grunden til dette er, at tiden betegnes som den 4. dimension, og denne skal også positioneres.

Hver satellit afsender signaler på 2 bære-frekvenser, der begge er såkaldt Pseudo-Random-Noise kode (PRN kode).

Den første type kaldes Precise kode (P kode), og er fortrinsvis beregnet for militært brug, og er som navnet antyder, en præcis kode. Kommercielle GPS-modtagere, har ikke adgang til at anvende denne del af systemet, af hensyn til risikoen for terrorisme. Bærefrekvensen for dette signal er på 1227.6 MHz.

Den anden type er Coarse/Acquisition kode (C/A kode), der er en 1023 bit sekvens, der gentages hvert sekund. C/A koden er beregnet til civilt brug, og dermed den del af GPS-systemet der skal anvendes til kollisionsdetektoren. Til dette signal er der tilføjet et lavfrekvent status signal, der oplyser modtageren om satellittens bane, klok-korrektioner til modtageren og system-status. Bærefrekvensen for dette signal er på 1575,42 MHz.[GPS s.91-100]

3.2 GPS og NMEA0183

En meget anvendt standard til kommunikation med en GPS er NMEA 0183. Standarden kører 4800 baud. NMEA definerer en række sætninger, der har en struktur som beskrevet i afsnit 2.3
En GPS kan typisk levere en række forskellige NMEA sætninger, der hver især egner sig til forskellige formål. I forbindelse med beregninger af afstande og kollisionsfarer er sætningstypen RMC velegnet, da den indeholder alle relevante data.

3.2.1 RMC

Sætningstypen RMC fra en GPS har følgende format, idet en GPS har afsenderidentifikation GP:

$GPRMC,hhmmss.ss,A,llll.ll,a,yyyyy.yy,b,S.S,C.C,ddmmyy,m.m,c,M*hh<CR><LF>

Data indholdet i sætningen er defineret i tabel 3.1.

1	hhmmss.ss	UTC tid for position
2	A	Status
3	llll.ll	Bredde
4	a	Nordlig eller sydlig bredde
5	yyyyy.yy	Længde
6	b	Østlig eller vestlig længde
7	S.S	Fart
8	C.C	Kurs over grunden
9	ddmmyy	Dato
10	m.m	Misvisning
11	c	Østlig eller vestlig misvisning
12	M	GPS tilstand

Tabel 3.1: Oversigt over RMC sætningen

Dataformat i felter

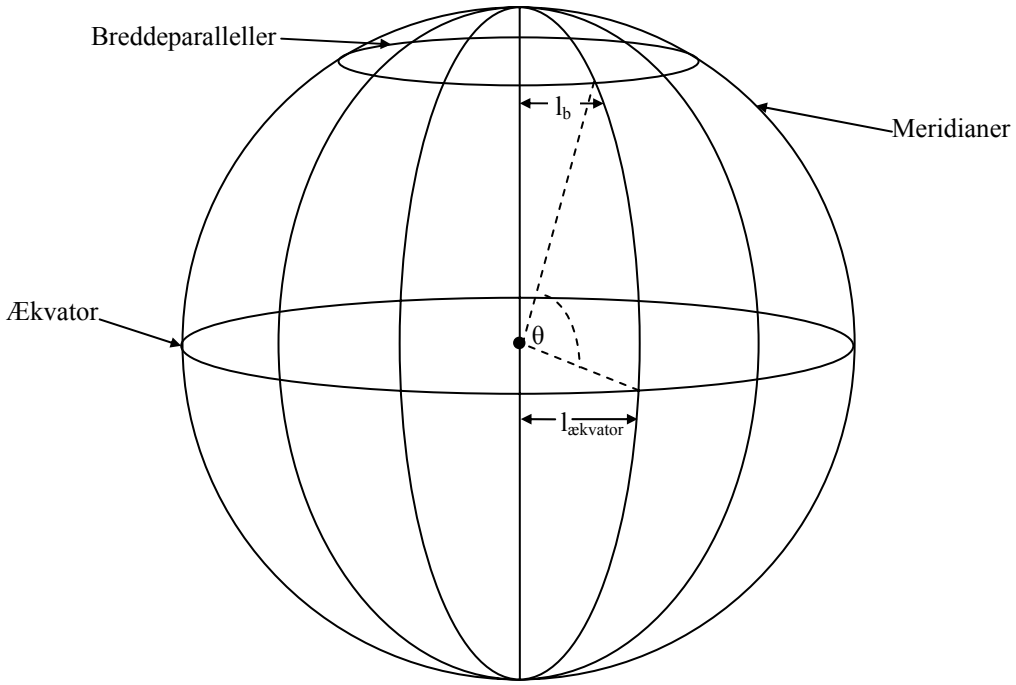
1. UTC tid angives med 2 cifre til timer, 2 cifre til minutter, og 2 cifre til sekunder samt et variabelt antal cifre til decimaler for sekunder.
2. Status kan være A eller V. A betyder at data er gyldige, medens V betyder, at der er problemer med modtageren.
3. Bredden angives med 2 cifre til grader, 2 cifre til minutter og et variabelt antal cifre til decimaler på minutter.
4. Angiver om der er tale om nordlig bredde eller sydlig bredde. Kan være N eller S.
5. Længden angives med 3 cifre til grader, 2 cifre til minutter og et variabelt antal cifre til decimaler på minutter.
6. Angiver om der er tale om østlig eller vestlig længde. Kan være E eller W.
7. Farten angives i knob. Der kan være et variabelt antal cifre på hver side af kommaet.
8. Kursen angives retvisende i grader. Der kan være et variabelt antal cifre på begge sider af kommaet.
9. Dato angives med 2 cifre til dagen, 2 cifre til måneden og 2 cifre til året.
10. Angiver misvisningens størrelse i grader. Der kan være et variabelt antal cifre på begge sider af kommaet
11. Angiver fortegnet på misvisningen. Kan være E eller W.
12. Indikerer hvilken tilstand GPS'en opererer i. Ved normal operation skal den være A eller D, da data ellers er ubrugelige.

4 Positionsberegning

Når brugerterminalen modtager AIS- og GPS-informationer omhandlende positioner for henholdsvis andre og eget skib, vil disse være opgivet i længde og bredde. Det er valgt at konvertere disse positionsangivelser til rektangulære koordinater, idet brugerfladen vil være baseret på et rektangulært koordinatsystem.
Endvidere er det valgt at tilegne det rektangulære koordinatsystem på en sådan måde, at eget skib altid vil være placeret i origo med retning identisk med den positive y-akse. Dette gøres for at øge brugervenligheden.

4.1 Koordinatsystem

Koordinatsystemet for jordens overflade kan tænkes som et gradnet, bestående af breddeparalleller og meridianer, nedlagt over jordens overflade, hvilket er illustreret på figur 4.1.



Figur 4.1: Jordens breddeparalleller og meridianer

Figur 4.1 viser hvorledes koordinatsystemet er opbygget af meridianer gående mellem geografisk nordpol og geografisk sydpol. Disse meridianer er inddelt i en vestlig og en østlig halvkugle, der går fra 0° til 180°. Meridianen der passere Greenwich definerer nul-meridianen.
Storcirklen er omkredsen af den cirkelflade, hvis flade ved gennemskæring af en kugle har det største areal, og hermed også den største omkreds. Jordens ækvator er dermed en storcirkel, der har den specielle egenskab at overalt på cirklen er afstanden til polerne 90º. Ækvator opdeler dermed jorden i to halvkugler, en nordlig halvkugle og en sydlig halvkugle.
Breddeparallellerne er derved lillecirkler, der ligger parallelt med ækvator. Dermed er der uendelig mange breddeparaller, der bliver mindre og mindre i areal og omkreds, desto nærmere de er placeret polerne. Breddeparallellerne er opdelt i en nordlig og en sydlig halvkugle, gående fra 0° ved ækvator til 90° ved polerne. Dette gør, at breddeparallelerne kan opdeles i sydlige og nordlige bredder der går fra 0 til 90°. Disse grader kan endvidere opdeles i bueminutter, der er defineret ved

$\frac{1}{60}$ del af en grad. Bueminutter er opdelt i et decimaltal af bueminutterne. Et eksempel på dette kan være 48° 15′20N, hvor 15 angiver bueminutter og 20 angiver 0,20 bueminut.
Meridianerne er alle storcirkler, da de alle står vinkelret på ækvator, hvilket er ensbetydende med, at de går fra pol til pol, og tilsvarende breddeparallellerne er der uendelig mange meridianer.
Meridianerne kan deles op i 180°, henholdsvis i østlige længder og vestlige længder. Længderne har den egenskab, at afstanden mellem disse varierer alt efter hvilken breddeparallel der er tale om. Dette er illustreret i figur 4.1, hvor den varierende afstand er som længden l_b.
Opdelingen af meridianerne er tilsvarende parallellerne, men med 180° i stedet for 90°. Et eksempel på en angivelse af længde kunne være 130° 27′10Ø. Stedangivelse på jorden kan således angives ved 48° 15′20N og 130° 27′10Ø.
Udfra denne opdeling kan længden af et bueminut på storcirklen, også kaldet et storcirkelminut, beregnes hvis jordens omkreds sættes til 40.000 km [OMK]:

$$\frac{40.000\text{km}}{360° \cdot 60\text{bueminutter}} = 1,852\text{km} \qquad (4.1)$$

Dette svarer til 1 sømil [NAV s.25]. Som førnævnt er afstanden mellem meridianerne ikke konstant, og derved er der ikke lige mange sømil mellem disse. Udfra figur 4.1 kan der således opsættes en ligning der beskriver, hvor mange sømil et længdeminut svarer til på en given bredde.

$$1' \text{ ved } \theta = 1\text{sm} \cdot \cos\theta \qquad (4.2)$$

4.2 Omregning af positioner

Med udgangspunkt i figur 4.1, er det muligt at omregne koordinaterne af eget og andre skibe, til rektangulære koordinater.
Y-koordinaterne i det rektangulære koordinatsystem findes simpelt, idet afstanden mellem breddeparallellerne er konstant. Denne afstand er identisk med $l_{ækvator}$, der angiver afstanden mellem to meridianer ved ækvator.
Koordinaterne for eget skib er givet ved (λ_1, γ_1) og for et andet skib (λ_2, γ_2).
Eget skib lægges i koordinatet (0,0), hvorved alle andre skibe således skal plottes i forhold til eget skib.
Ligningen til beregning af et andet skibs y-koordinat bliver således:

$$y = (\gamma_2 - \gamma_1) l_{ækvator} \qquad (4.3)$$

Ved beregning af et andet skibs x-koordinat, skal der tages højde for ændringen af afstanden mellem meridianerne. Afstanden mellem to meridianer, ved en vilkårlig breddeparallel l_b, er givet ved følgende ligning, idet det antages at jorden er kugleformet:

$$l_b = \cos\theta \cdot l_{ækvator} \qquad (4.4)$$

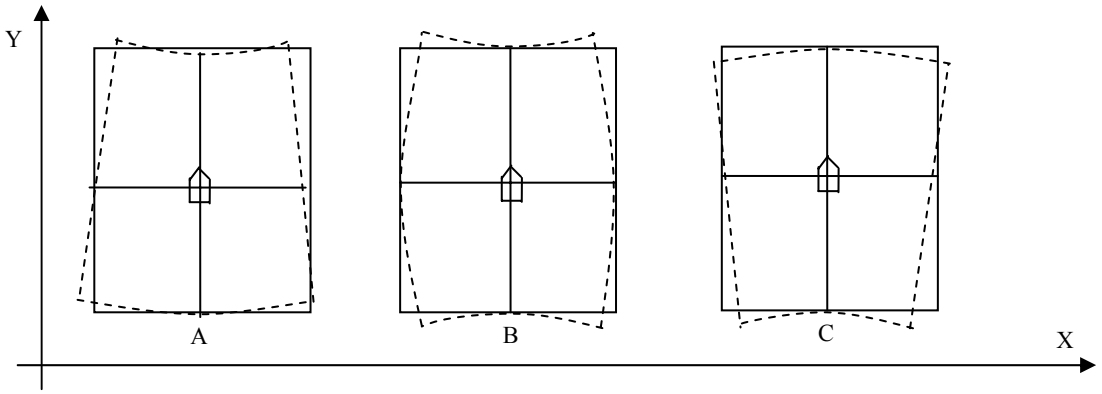
Ligningen til beregning af et andet skibs x-koordinat bliver derfor:

$$x = (\lambda_2 - \lambda_1) l_b \qquad (4.5)$$

4.3 Placering af andre skibe

Efter at have beregnet koordinaterne for omkringliggende skibe, skal disse koordinater føres over i et kartesisk koordinatsystem. Dette gøres til trods for en afvigelse vil forekomme, da jorden er

kugleformet. Dette er illistreret på figur 4.2, hvor A viser situationen hvor skibet befinder sig på den nordlige halvkugle, B ved ækvator og C på den sydlige halvkugle.



Figur 4.2:Approksimation til kartetisk koordinatsystem

For at beregne denne afvigelse regnes et eksempel igennem. Først omregnes de 25 sømil til et antal grader:

$$25\,\text{sømil} = \frac{25\text{minutter}}{60°} = \frac{5}{12}° \qquad (4.6)$$

Herefter beregnes længden ud for henholdsvis 20° og for 20° + 5/12°:

$$\begin{aligned} &20° : \cos(20°)=0{,}9397 \\ &20+5/12\,°:\cos(20+5/12°)=0{,}9372 \end{aligned} \qquad (4.7)$$

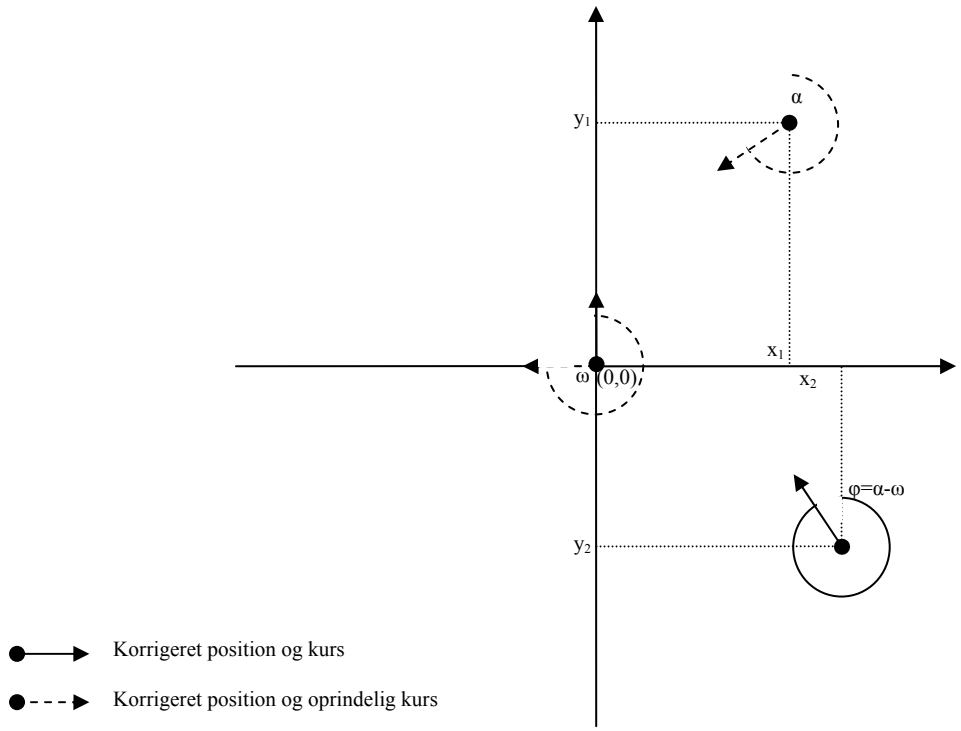
Tilsidst beregnes den procentvise forskel:

$$1-\frac{1}{0{,}9397}\cdot 0{,}9372\cdot 100\% = 0{,}266\% \qquad (4.8)$$

Da den største afvigelse, der vises på den grafiske brugerflade, således er 0,266%, ses der bort fra denne afvigelse.
Ved indsættelse af skibene i det kartesiske koordinatsystem, vælges det, at eget skib skal være i origo, således eget skib altid vil være i centrum. Dette betyder, at de koordinater der blev beregnet i afsnit 4.2 for eget skib skal trækkes fra henholdsvis eget og andre skibe. Dette bevirker, at de andre skibe bliver lagt ind i koordinatsystemet i forhold til eget skib.

4.3.1 Beregningsmetode

Det nye korrigerede koordinatsystem bestemmes som vist på figur 4.3:



Figur 4.3: Korrigeret koordinatsystem med eget skib i origo

Efter at have placeret eget skib i origo, og omkringliggende skibes koordinater i forhold til dette, skal eget skibs kurs ændres, således denne får samme retning som y-aksen. Dette gøres af to årsager. Dels for at øge brugervenligheden, idet eget skib altid vil have retning opad på skærmen, og dels for at simplificere de nødvendige kollisionsberegninger.
På baggrund af dette, skal alle andre skibes kurser korrigeres på en sådan måde, at den oprindelige vinkel imellem eget skibs kurs og omkringliggende skibes kurser bibeholdes.
Denne korrektion er illustreret på figur 4.3. Der tages udgangspunkt i, at samtlige kurser er givet ved et gradtal i forhold til nord jvf. Afsnit 3.2.1.
Da eget skibs kurs altid vil blive ændret til kurs 0°, findes omkringliggende skibes kurser ved at trække eget skibs oprindelige kurs ω fra det aktuelle skibs egen kurs α. Et omkringliggende skib vil således have fået en ny kurs φ, der repræsenterer vinkelforskellen imellem kurserne på eget og omkringliggende skib.
Dette giver mulighed for en negativ φ-værdi. I sådanne tilfælde lægges 360° til φ, hvilket vil resultere i den korrekte positive φ-værdi.

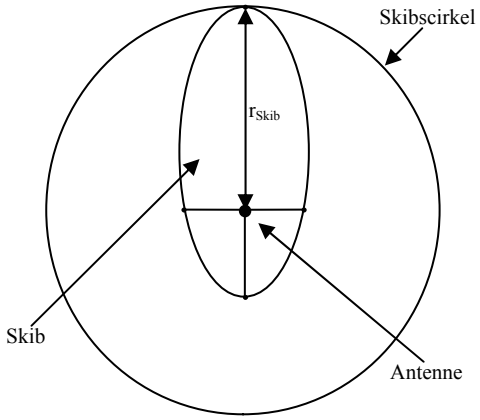
5 Kollisionsberegning

Inden designfasen af kollisionsdetektoren er det nødvendigt at klarlægge hvorledes kollisionsberegningen skal foregå. Herunder skal bl.a. indgå hvilke parametre brugeren skal have mulighed for at bestemme, desuden er det nødvendigt at definere hvad systemet skal opfatte som en kollision. Endeligt er det en nødvendighed at beregningerne foregår på en sådan måde, at omfanget af disse minimeres, idet systemet skal belastes mindst muligt.

5.1 Definition af skibsparametre

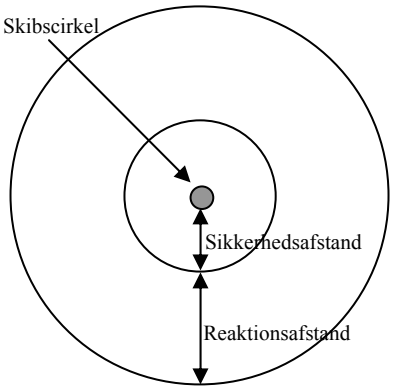
Inden kollisionsberegningerne kan foretages er det nødvendigt først at definere hvordan et skib opfattes, i projektet, samt hvilke parametre beregningerne tager udgangspunkt i.

Idet AIS-informationerne fra de omkringliggende skibe indeholder information om antennens placering på det enkelte skib, samt afstanden fra skibets yderste punkt til antennen, er det valgt at opfatte disse skibe som cirkler, hvilket ses i figur 5.1



Figur 5.1: Skibscirkel

Radius for disse cirkler er identisk med den længste afstand mellem skibets yderste punkt og antennen og er således et worst case tilfælde. Denne cirkel kaldes i projektet for skibscirkel. Foruden denne cirkel er det valgt yderligere at give brugeren mulighed for at indlægge en sikkerhedsafstand og en reaktionstid, symboliseret ved to cirkler omkring de øvrige skibe. Sikkerhedsafstanden er den mindste afstand der ønskes fra egen position til de øvrige skibes skibscirkler. Brugeren skal derfor tage hensyn til eget skibs længde når sikkerhedsafstanden indtastes. Sikkerhedsafstanden samt reaktionsafstanden er illustreret i figur 5.2



Figur 5.2: Sikkerheds- og reaktionsafstand

Reaktionsafstandscirklen, tilkendegiver hvor lang tid der går fra reaktionstiden er brudt til sikkerhedsafstanden brydes, i det tilfælde at de to skibe har kurs lige mod hinanden. Længden af reaktionstidsradiusen er således afhængig af såvel egen som de øvrige skibes hastighed og er et worst case tilfælde.
Reaktionsafstanden tilkendegiver blot hvornår omfanget af kollisionsberegninger skal øges.
I det øjeblik reaktionsafstanden brydes skal alarmen kun gå, såfremt skibene har kurs direkte mod hinanden. Hvorledes kollisionsberegningerne foretages vil blive gennemgået i det følgende.

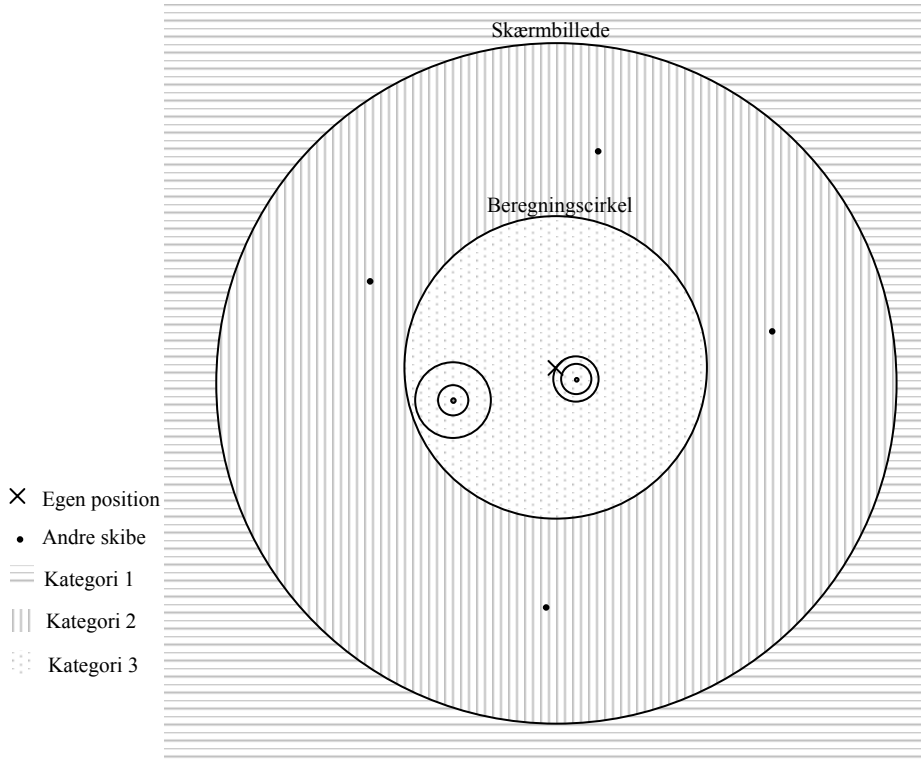
5.2 Kollisionsberegning

Til kollisionsberegningen er det valgt at inddele de omkringliggende skibe i fire kategorier der hver især tilkendegiver en bestemt afstand til eget skib. Begrundelsen for denne kategorisering er et ønske om at minimere det samlede antal beregninger.
De enkelte kategoriseringer er illustreret i figur 5.3 og er defineret således:

- *Kategori 1*: Skibe udenfor skærmbilledet, der således ikke skal illustreres.
- *Kategori 2*: Skibe indenfor skærmbilledet, men uden for en fastsat beregningscirkel.
- *Kategori 3*: Skibe indenfor beregningscirklen, men hvor reaktionsafstandscirklen ikke er brudt.
- *Kategori 4*: Skibe hvor reaktionsafstandscirklen er brudt.

Det er valgt at visualisere skærmbilledet i et interval mellem to målestoksforhold, således at brugeren opnår en bedre brugervenlighed. I det største målestoksforhold er radius af skærmbilledet 25 sømil ca. svarende til rækkevidden for en VHF-radio. I det mindste målestoksforhold er radius af skærmbilledet valgt til 1 sømil.
Beregningscirklens radius er lig den afstand to skibe skal have fra hinanden, således at disse ikke støder sammen inden for den maksimale reaktionstid, i det tilfælde de har retning direkte mod hinanden med maksimal fart. Den maksimale fart de to skibe sejler med er valgt til henholdsvis 44 og 15 knob. De 44 knob er maksimalhastigheden for en hurtigfærge, medens de 15 knob er den maksimalhastighed det antages at brugeren maksimalt sejler med. Den maksimale reaktionstid er valgt til 5 minutter, idet denne tid antages at være tilstrækkelig, til at afværge en kollision.
Radius af beregningscirklen bliver derfor:

$$B_r = \left((15+44)\,knob\right)\cdot 5\,\text{min} = 4,92sm \qquad (5.1)$$



Figur 5.3: Oversigt over kategoriinddeling i målestoksforhold 1

Omfanget af beregninger stiger i takt med kategorinummer, således at kategori 1 og 2 skibe udelukkende behandles på baggrund af afstand til eget skib. Er afstanden til et skib større end skærmbillederadius, er dette et kategori 1 skib, og der skal ikke gøres yderligere ved dette skib. Er afstanden derimod mindre skal skibet som et minimum plottes på skærmen. Til beregning af afstanden benyttes ligning 5.2

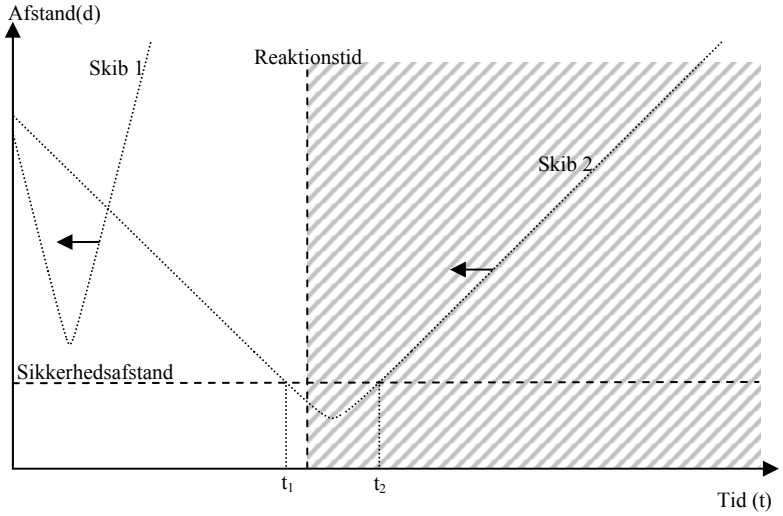
$$d = \sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2} \qquad (5.2)$$

Er afstanden endvidere mindre end radius på beregningscirklen skal det afgøres, hvorvidt der er tale om et kategori 3 eller 4 skib. Til dette benyttes ligning 5.3

$$d = \left((v_1 + v_2) \cdot t_r + \text{skibsradius+sikkerhedsafstand}\right) \qquad (5.3)$$

,hvor v_1 og v_2 er farten for henholdsvis eget og andet skib mens t_r er en reaktionstid bestemt af brugeren.
Er denne afstand mindre end afstanden i ligning 5.2 er skibet et kategori 3 skib. I modsat fald er der tale om et kategori 4 skib.
Er et kategori 4 skib først konstateret, er det nødvendigt at beregne, hvorvidt sikkerhedsafstanden brydes og om en kollision evt. kan forekomme.
I modsætning til beregninger på skibe i de øvrige kategorier, er det for kategori 4 skibe nødvendigt at inddrage fart og kurs, for såvel eget skib, som for de omkringliggende kategori 4 skibe. Dette medfører, at beregningerne for kategori 4 skibe er betydeligt mere omfattende end for skibe i de øvrige kategorier.
Formålet med beregningerne er at bestemme, hvorvidt sikkerhedsafstanden brydes indenfor den af brugeren fastsatte reaktionstid.

Dette gøres ved at beregne afstanden mellem det aktuelle kategori 4 skib og eget skib til forskellige tider. Figur 5.4 illustrerer teorien bag denne beregning.



Figur 5.4: Afstand til andet skib

Skib 1 og skib 2's kurver illustrerer afstanden til eget skib over en tidsperiode, og er beregnet ud fra de øjebliksoplysninger AIS- og GPS-modtageren har modtaget. Figur 5.4 er således en approksimation af fremtiden i det tilfælde, at øjebliksoplysningerne ikke ændrer sig.
Skib 1 og skib 2's kurver består af en række afstandspunkter, der er beregnet ved t gående fra nul mod reaktionstiden i intervaller af 1 sekund. Der vil således ikke blive foretaget nogen beregninger for t større end reaktionstiden, hvilket i figur 5.4 er illustreret som et skraveret område. Eksemplet på figur 5.4 viser, at skib 1 ikke udløser nogen alarm, idet kurven ikke krydser sikkerhedsafstanden. Skib 2 udløser derimod en alarm, da kurven for afstanden til eget skib krydser sikkerhedsafstanden, indenfor reaktionstiden.
Som tiden går, vil begge kurver rykke mod venstre, og først når kurven for skib 2 ikke længere krydser sikkerhedsafstanden slår alarmen fra. Det antages igen, at øjebliksoplysninger fra AIS- og GPS-modtageren ikke ændrer sig for det aktuelle skib. I det tilfælde, at øjebliksoplysningerne ændrer sig, f.eks. ved en kursændring, vil kurverne ændre sig tilsvarende i takt med hver opdatering.
Beregningerne er for kategori 4 skibe er udledt og yderligere forklaret i appendiks C. I dette appendiks er endvidere foretaget beregningseksempler for skibe i samtlige kategorier.
Begrundelsen for at inddele skibene i fire kategorier er at minimere omfanget af beregningerne, og således spare tid. Tidsforbruget ved de forskellige kategorier ses i tabel 5.1.

Kategori	1	2	3	4
Varighed	0,240µs	0,240µs	0,241µs	1,522µs

Tabel 5.1: Gennemsnitstid for beregningerne på ét skib i de enkelte kategorier.

Tiderne i tabel 5.1 er udregnet som gennemnittet af 10 mil. beregninger jf. [CD\Test\Tidsberegninger] og foretaget på en pc med en Intel Pentium 4 1,8 GHz processor.
Da omfanget af tidsforbruget er fundet tilstrækkeligt lavt, sammenlignet med et minimal opdateringsinterval på 2 sekunderer jf. [AIS], er det ikke fundet nødvendigt at benytte en mikrocomputer til grovsortering af data.

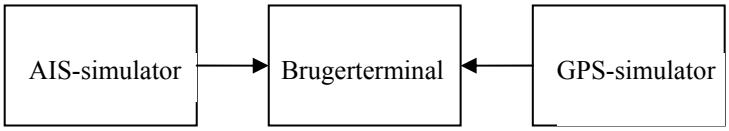
6 Kravspecifikation

Formålet med dette projekt er at designe og konstruere et system, der advarer lystbåde om risiko for kollision.
Kravspecifikationen har til formål at beskrive dette system, herunder bl.a. hvilke elementer der skal indgå i systemet, samt disses grænseflader.
Systemet vil blive baseret på AIS-standarden [AIS], dog er det i projektet valgt ikke at udsende AIS-informationer. Systemet skal således designes til at modtage GPS- og AIS-informationer, og på baggrund af disse afgøre, hvorvidt der er risiko for kollision.
Det samledes system vil i den resterende del af rapporten blive omtalt som "kollisionsdetektoren".
Kravspecifikationen er baseret på SPU-modellen, som den er beskrevet i "Håndbog i struktureret programudvikling"[SPU s.85-96].

6.1 Generel beskrivelse

6.1.1 Systembeskrivelse

Kollisionsdetektoren skal opbygges på en sådan måde, at denne er i stand til at modtage GPS-informationer omhandlende egen kurs og position, samt AIS-informationer indeholdende andre fartøjers kurs, position m.m. Det er valgt at simulere disse AIS-informationer, idet AIS er et endnu ikke særligt udbredt system, hvorfor modtagelse af AIS-informationer ikke kan påregnes. Det er ligeledes valgt at simulere informationerne fra en GPS-modtager, idet kollisionsdetektoren under opbygning og test vil være stationær.
Kollisionsdetektoren vil således bestå af blokkene i figur 6.1:



Figur 6.1: Blokdiagram over kollisionsdetektoren

AIS-simulator
AIS-simulatoren skal simulere AIS-informationer indeholdende andre fartøjers kurs, position m.m. Simulatoren skal endvidere være software baseret og køre på en PC, uafhængig af det øvrige system.

GPS-simulator
GPS-simulatoren skal simulere GPS-informationer omhandlende egen kurs og position, og ligeledes være software baseret og køre på en PC, uafhængig af det øvrige system.

Brugerterminal
Brugerterminalen modtager data fra henholdsvis AIS- og GPS-simulator, hvorefter ikke relevant data sorteres fra og relevant data omsættes til en grafisk visualisering af den omkringliggende skibstrafik. Brugerterminalen skal endvidere kunne advare brugeren i tilfælde af risiko for kollision. Endelig giver Brugerterminalen brugeren mulighed for at indhente en række informationer vedrørende de omkringliggende skibe.
Brugerterminalen skal være softwarebaseret og køre på en PC, uafhængig af det øvrige system, med et Windows styresystem.

6.1.2 Kollisionsdetektorens funktioner

Det er valgt at kollisionsdetektoren skal have følgende funktioner:

- Skal kunne modtage informationer fra en GPS-modtager i henhold til NMEA 0183 standarden.
- Skal kunne modtage AIS-informationer i henhold til NMEA 0183 standarden.
- Skal kunne give en grafisk visualisering af omkringliggende AIS kompatible skibe, vha. en PC.
- Skal kunne give brugeren mulighed for at indhente informationer omkring disse skibe.
- Skal kunne afgøre hvorvidt der er risiko for kollision, og i tilfælde af dette advare brugeren visuelt og auditivt.
- Skal kunne foretage samtlige beregninger i så tæt på realtid som muligt.
- Skal kunne give brugeren mulighed for at indstille sikkerhedsafstanden og reaktionstiden til omkringliggende skibe.
- Skal kunne give brugeren mulighed for at afbryde audioalarmen.

6.1.3 Kollisionsdetektorens begrænsninger

Det er valgt at kollisionsdetektoren skal have følgende begrænsninger:

- Skal ikke kunne afsende AIS-informationer, idet der ikke er lagt op til en sådan afsendelse i projektforslaget.
- Skal i projektet ikke fungere i direkte forbindelse med VHF-modtager, i stedet skal AIS-informationer simuleres vha. software. Begrundelsen for dette er, at der endnu ikke findes en standard for øverførelse af AIS-informationer, fra VHF-modtager til den egentlige kollisionsdetektor.

6.1.4 Kollisionsdetektorens fremtid

Da de fleste mindre skibe allerede er i besiddelse af såvel GPS-modtager som PC og VHF-radio, vil det være forholdsvis simpelt at implementere en kollisionsdetektor i sådanne skibe. Vælges det desuden, at kollisionsdetektoren skal være i stand til at afsende AIS-informationer, vil dette give endnu flere fordele, bl.a. i forbindelse med indsejling til havne, hvor havnefoged og skib vil kunne kommunikere automatisk. Sammenholdes dette med den øgede sikkerhed, vil en kollisionsdetektor være ønskeligt på ethvert skib.

6.1.5 Brugerprofil

Kollisionsdetektoren henvender sig primært til ejerne af lystbåde, der ønsker at forbedre sikkerheden ombord.
Det antages at brugerne selv er i stand til at opsætte systemet, idet det i den endelige udgave blot er et spørgsmål om at forbinde nogle få stik korrekt. Til gengæld gøres brugerterminalen brugerflade så enkel og funktionel som muligt, idet det ikke kan forventes, at brugeren har et større kendskab til software programmel. Det er på baggrund af dette, at det er valgt, at brugerterminalen skal være Windows kompatibel, og brugerterminalens brugergrænseflade skal ligeledes grafisk ligne en Windows-brugerflade.

6.1.6 Krav til udviklingsforløbet

Kravene til projektets udviklingsforløb er detaljeret beskrevet i E-studienævnets E5 projektenhedsbeskrivelse.

Til at modellere og udvikle projektets softwaredel er det valgt at gøre brug af UML. Selve programmeringssproget et valgt til Borleand Delphi 7, der implementerer UML via programmet Modelmaker.

6.1.7 Forudsætninger

Til opbygning og realisering af kollisionsdetektoren forudsættes følgende:

- De tre PC'ers (AIS simulator, GPS simulator og brugerterminal) serielporte skal være RS232-kompatible.
- PC-softwaren skal skrives til at begå sig i et Windows miljø.
- Brugerterminalens PC skal have tilstrækkelig processorkraft til sortere og behandle data indenfor realtid. Realtid defineres således at evnen til at opdatere AIS-informationer i henhold til AIS standarden [AIS s.3] skal overholdes.

6.2 De specifikke krav

6.2.1 AIS simulator

- Skal kunne simulere op til 450 skibe, af forskellig fart og kurs indefor et afgrænset område på op til 25 sømil i radius. Dette gøres da en realistisk situation ikke vil kunne overstige dette scenarie
- Skal simulere AIS-informationer i en uafhængig PC og videresende disse via NMEA 0183 protokollen
- Skal udstyres med en pausefunktion
- Skal afsende AIS-informationer med det i AIS standarden angivne tidsinterval
- Skal kunne hente AIS-scenarier fra en bestemt database

6.2.2 GPS simulator

- Skal hente GPS scenarier fra en bestemt database
- Skal simulere GPS-informationer, for eget skib, i en uafhængig PC og videresende disse via NMEA 0183 protokollen
- Skal udstyres med en pausefunktion

6.2.3 Brugerterminalen

- Skal kunne modtage AIS-informationer
- Skal kunne modtage GPS-informationer
- Skal kunne modtage og behandle AIS-informationer svarende til 450 skibe, indenfor realtiden
- Skal give en grafisk visualisering, herunder en zoomfunktion af den omkringliggende skibstrafik
- Skal give en visuel og auditiv alarm ved kollisionsrisiko
- Auditiv alarm skal kunne deaktiveres af brugeren
- Skal kunne give en række AIS-informationer vedrørende de omkringliggende skibe
- Skal give brugeren mulighed for at indstille sikkerhedsafstand og reaktionstid

6.3 Eksterne grænsefladekrav

De eksterne grænsefladekrav beskriver hvorledes de tre blokke i figur 6.1 kommunikerer.

6.3.1 Brugergrænseflade

Brugergrænsefladen er den grænseflade der eksisterer mellem brugeren og brugerterminalen. Brugerterminalen kommunikerer til brugeren via et skærmbillede, medens kommunikation den modsatte vej foregår vha. en PC-mus.
På skærmbilledet skal vises en grafisk visualisering af placering af omkringliggende skibe, i lighed med en radarskærm. Endvidere skal brugeren kunne indhente informationer om de enkelte skibe, der optræder på skærmbilledet, og ligeledes være i stand til at indhente informationer vedrørende eget skib.
Brugergrænsefladen skal udformes på dansk og fremstå så simpel og forståelig som mulig.

6.3.2 Hardwaregrænseflader

Kollisionsdetektoren indeholder to hardwaregrænseflader:

- AIS-simulator → Brugerterminalen
- GPS-simulator → Brugerterminalen

Begge grænseflader består af RS232-kompatible serielforbindelser, sammenkoblet med 9 bens DSUB stik.

6.3.3 Softwaregrænseflader

Kollisionsdetektoren indeholder fire softwaregrænseflader:

- Database → AIS-simulator
- Database → GPS-simulator
- AIS-simulator → Brugerterminalen
- GPS-simulator → Brugerterminalen

Det er valgt at placere de to uafhængige databaser på samme PC som henholdsvis AIS- og GPS-simulator.

6.3.4 Kommunikationsgrænseflader

Kollisionsdetektoren indeholder fire kommunikationsgrænseflader:

- Database → AIS-simulator
- Database → GPS-simulator
- AIS-simulator → Brugerterminalen
- GPS-simulator → Brugerterminalen

Det gælder for de to sidstnævnte af disse grænseflader, at kommunikationen skal ske i henhold til NMEA 0183 standarden. Det er dog nødvendigt, i kommunikationsgrænsefladen mellem AIS-simulatoren og brugerterminalen, at tilpasse NMEA 0183 standarden til AIS standarden, hvilket betyder at dataoverførselshastigheden skal øges til 38400bps.

6.4 Kvalitetsfaktorer

Da det grundet projektets omfang er fundet nødvendigt at differentiere imellem kollisionsdetektorens forskellige kvalitetsfaktorer, følger en prioritering af disse på en skala fra 1 til 5, hvor 1 er ukritisk og 5 er særdeles vigtigt. Kvalitetsfaktorerne er gældende for projektets prototype af en kollisionsdetektor med dennes begrænsninger.

Pålidelighed
Kollisionsdetektorens pålidelighed er af særdeles vigtighed, da det er pålideligheden, der er grundlaget for dennes berettigelse. Det er bedre at være foruden en kollisionsdetektor end at benytte en upålidelig kollisionsdetektor, da denne i så fald vil give en falsk tryghed, der i yderste tilfælde vil kunne resultere i en kollision.
Prioritering: 5, særdeles vigtig.

Vedligeholdelsesvenlighed
Da vedligeholdelsesvenligheden angiver den tid det tager at finde og rette en fejl, efter produktet er taget i brug, er denne faktor prioriteret meget lavt. Dette gøres på baggrund af, at evt. vedligeholdelse skyldes en fejl, der allerede er konstateret og således ikke udgør nogen sikkerhedsrisiko, idet brugeren er bevidst om fejlen.
Prioritering: 1, ukritisk.

Udvidelsesvenlighed
Da der er tale om en prototype vælges det at lægge vægt på de allerede opstillede krav, på bekostning af evt. udvidelsesmuligheder.
Prioritering: 2, ikke særlig vigtig.

Brugervenlighed
Selvom det ikke forventes at brugeren af kollisionsdetektoren har særlig kendskab til edb, må der ikke opstå misforståelser i forbindelse med brugergrænsefladen. Denne skal således gøres enkel og utvetydig, hvilket er ensbetydende med en høj prioritering af brugervenligheden.
Prioritet: 4, meget vigtig.

Genbrugbarhed
Da kollisionsdetektoren ikke skal udvides eller forbedres efter færdiggørelse, har det ingen særlig interesse at kunne genbruge dele af programmellet.
Prioritet: 1, ukritisk.

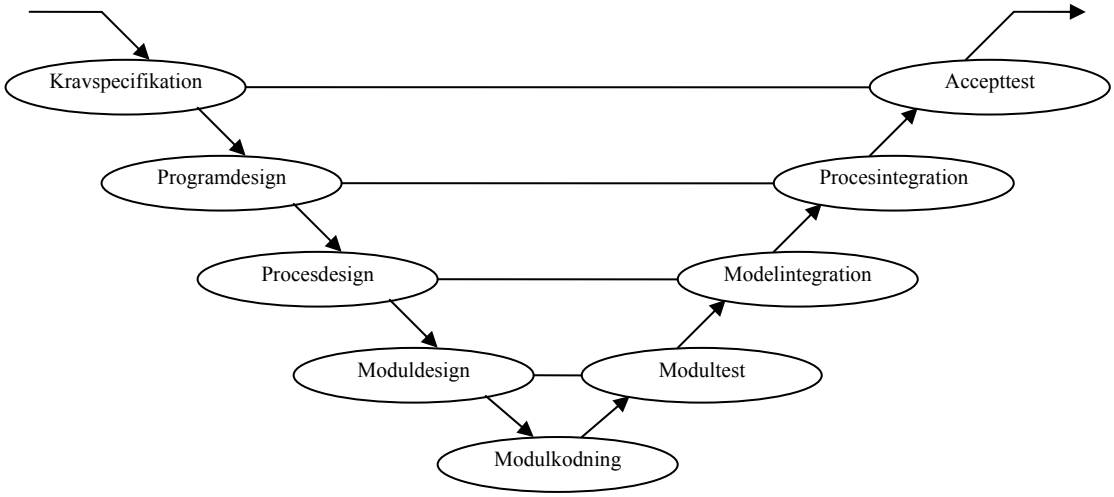
Integritet
Med integritet menes produktets evne til at beskytte sine data. Denne faktor prioriteres højt, idet den er en del af produktets pålidelighed. Kollisionsdetektoren må under drift eksempelvis ikke kunne påvirkes af andet elektrisk udstyr. Desuden skal integriteten være så høj, at alle beregninger kan udføres korrekt jf. pålideligheden.
Kollisionsdetektorens følsomhed over for korte driftsforstyrrelser er dog mindre væsentlig, idet der, med AIS, foregår en løbende opdatering af data.
Prioritering: 4, meget vigtig.

Effektivitet
I forbindelse med kollisionsdetektorens effektivitet fokuseres særligt på evnen til at foretage alle beregninger i tilnærmelsesvis realtid. Kollisionsdetektorens effektivitet skal således være så høj, at brugeren kan regne med, at brugergrænsefladen giver et korrekt billede af den omkringliggende skibstraffik.
Prioritering: 5, særdeles vigtig.

7 Design-/testopbygning

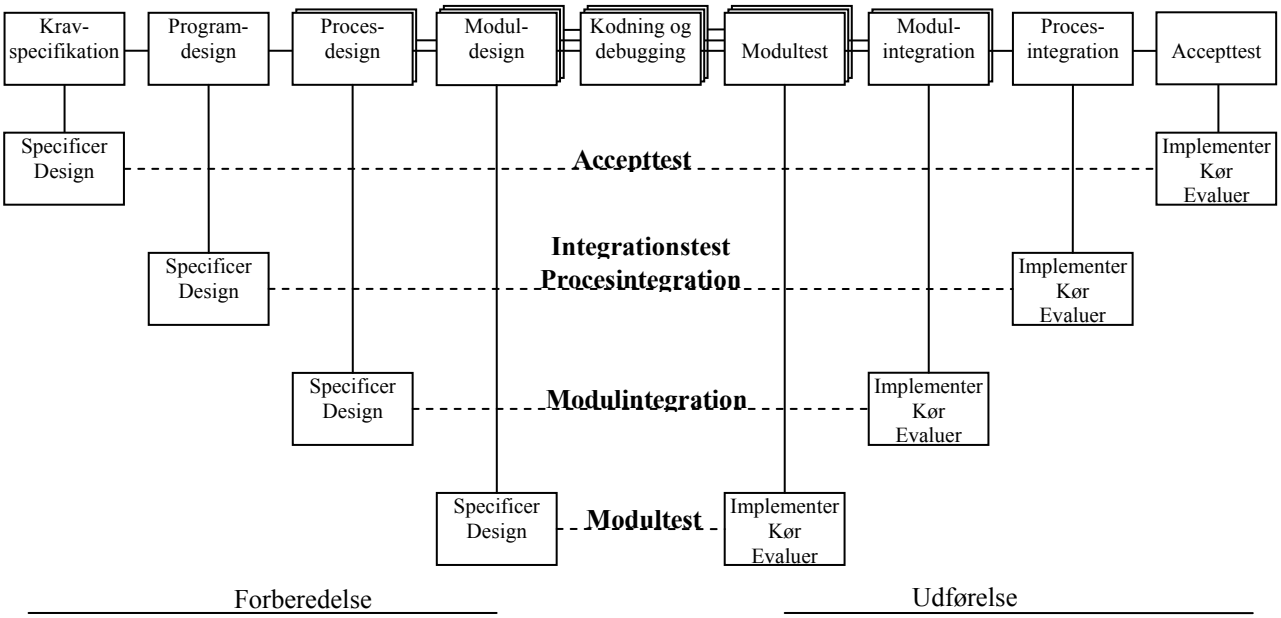
En vigtig del af projektplanlægning er at forberede og fastlægge, hvorledes projektets produkt skal testes. Disse tests skal som udgangspunkt finde sted så tidligt i projektforløbet som muligt, idet fundne fejl nemmere og hurtigere kan rettes. Det er således nødvendigt fra projektets start at afgøre, hvilke tests der skal finde sted, og hvorledes disse tests skal effektueres. Ydermere har fastlæggelse af testvektorer fra projektstart den fordel, at produktet designes til at opfylde testvektorerne og ikke omvendt.
Til rapportopbygning er det valgt at tage udgangspunkt i SPU-modelens vejledning i softwaretest.

I forbindelse med testplanlægning og rapportopbygning tages udgangspunkt i figur 7.1, der skitserer, hvorledes de enkelte testfaser er knyttet sammen med de egentlige udviklingsfaser.



Figur 7.1: SPU's V-model for testaktiviteter.

Figuren viser hvorledes kravspecifikationen ligger til grund for accepttesten, programdesignet ligger til grund for procesintegrationen osv. Ved et designlags afslutning kan en tilhørende test specificeres, og når modulkodningen således er overstået, kan de allerede specificerede tests udføres.
En mere udførlig oversigt over arbejdsgangen for de enkelte testaktiviteter ses i figur 7.2, der viser sammenhængen mellem forberedelse og udførelse.



Figur 7.2: Testaktiviteter i projektets faser

Som det ses af figuren forberedes de enkelte designdele af de samme to testaktiviteter: Specificer og design. Det er således nødvendigt at specificere *hvad* der skal testes, og *hvordan* denne test skal designes.
Tilsvarende udføres testene ud fra tre punkter: Implementer, kør og evaluer. Implementeringen finder sted ved at skrive et testprogram og sætte testomgivelser op, hvorefter testen køres og resultatet sammenlignes med det forventede. Endeligt evalueres testforløbet.
Af figur 7.1 og 7.2 kan udledes, at det er muligt at opstille testvektorer til accepttesten på baggrund af kravspecifikationen. Endvidere ses modul- og integrationstestenes placering i projektforløbet. Disse tre tests vil efterfølgende blive gennemgået, og accepttesten vil blive forberedt.

7.1 Testformer

7.1.1 Modultest

Formålet med modultest er at teste hvert enkelt modul for således at sikre, at modulet fungerer som beskrevet i modulspecifikationen. En modultest skal endvidere teste det enkelte moduls grænseflade samt den interne logiske struktur af modulet. Dette medfører at samtlige kalde- og returparametre skal anvendes minimum én gang, og alle grene i programmellet skal gennemgås.
Da en modultest af samtlige moduler kan være meget tidskrævende, kan det vælges at udelade test af enkelte moduler, hvis disse vurderes at være så simple at fejl er usandsynlige.

7.1.2 Integrationstest

En integrationstest består af henholdsvis en modulintegrationstest og en procesintegrationstest. Modulintegrationstesten har til formål, at teste sammenkoblede modulers funktionalitet. Fungerer dette som forventet sammenkobles endnu et modul, og der testes igen. Dette gentages indtil de sammenkoblede moduler udgør en proces.
Procesintegrationstesten udføres efter modulintegrationstesten er afviklet, idet denne test tester funktionalitet af de sammenkoblede processor. Når alle processor er sammenkoblet og testede er det samlede system, og dermed det egentlige produkt, færdigt, hvorefter accepttesten kan påbegyndes.

7.1.3 Accepttest

Accepttesten har til formål at verificere, at samtlige funktionelle krav opstillet i kravspecifikationen er opfyldt. Desuden undersøges hvorvidt programmellet opfylder den ønskede prioritering af kvalitetsfaktorerne opstillet i kravspecifikationen. Afslutningsvis testes systemet ved diverse spidsbelastninger og ekstreme situationer, således eventuelle fejl eller begrænsninger findes og rettes, inden systemet tages i brug.
Accepttesten markerer således afslutningen af udviklingsforløbet.

Specificering af accepttesten

Foruden tests af de funktionelle krav opstillet i kravspecifikationen, er det valgt, at accepttesten skal inkludere følgende tests:

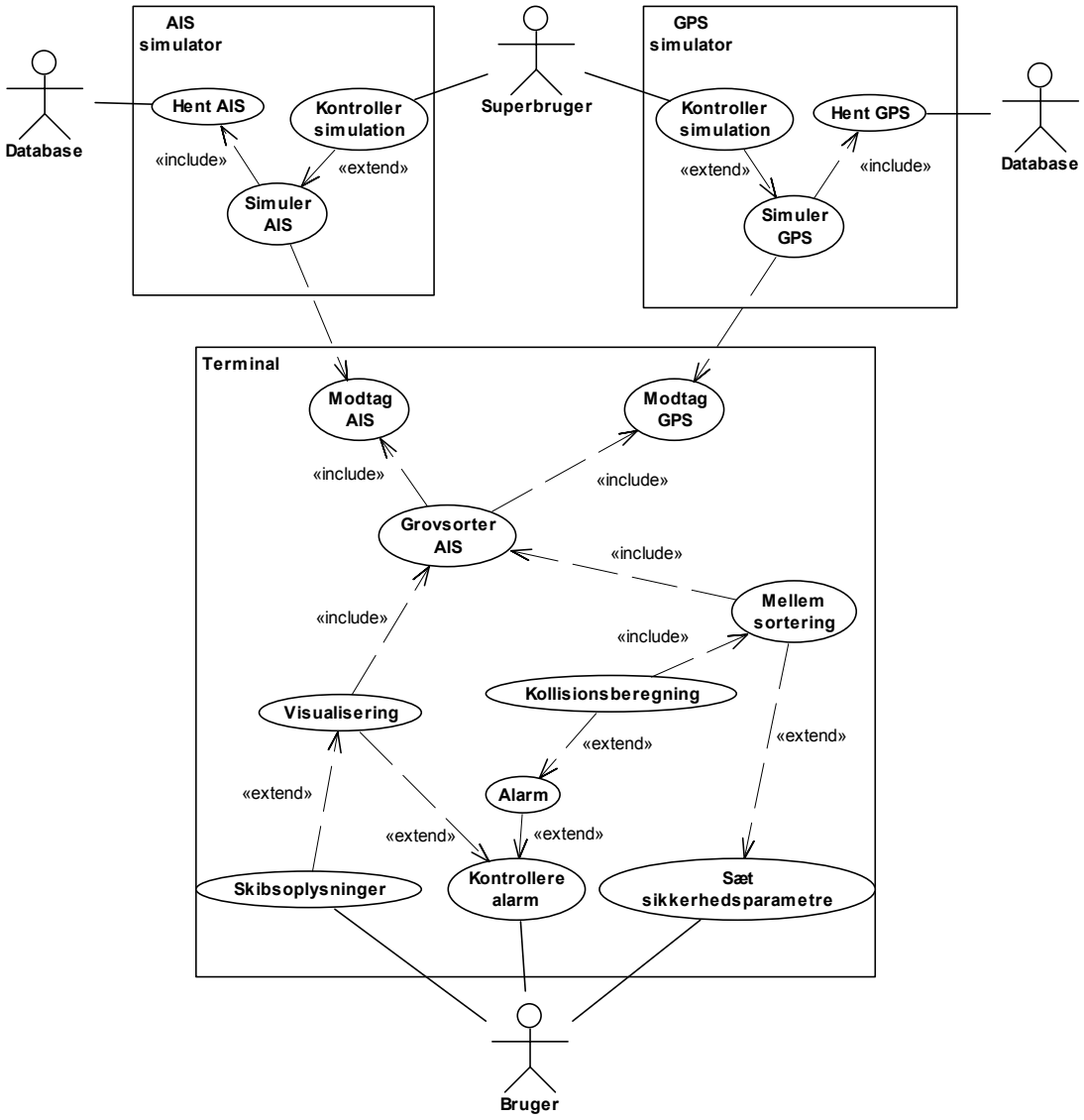
- Stress-test
  - Kollisionsdetektoren skal testes for det tilfælde, at et skib krydser datolinien.
- Konfigurationstest
  - I konfigurationstesten skal det undersøges, hvorvidt kollisionsdetektoren fungerer korrekt ved tilslutning til en GPS-modtager.
- Test af ydeevne
  - I test af ydeevnen undersøges kollisionsdetektorens evne til at opdatere AIS-informationer i henhold til AIS standarden [AIS s.3]. Endvidere undersøges det om kollisionsdetektoren kan behandle og plotte AIS-informationer fra 450 skibe.
- Pålidelighedstest
  - Kollisionsdetektoren skal testes ved kontinuert normaldrift i 48 timer.
- Fejlbehandlingstest
  - Kollisionsdetektoren skal testes for det tilfælde, at der ikke længere modtages AIS-informationer fra et bestemt skib. I dette tilfælde skal informationerne for dette skib slettes fra brugerterminalens hukommelse vha. en time-out. Formålet med denne funktion skal være at undgå, at skibe der bevæger sig uden for sendeafstand stadig optager hukommelse. Endvidere er formålet, at undgå muligheden for spøgelsesskibe. Med dette menes AIS-informationer for skibe, der ikke længere befinder sig på den sidst registrerede position, men har bevæget sig uden for sendeafstand.

Design af accepttesten

Design af accepttesten indebærer, som tidligere nævnt, en beskrivelse af, hvorledes de tidligere specificerede tests skal gennemføres i praksis.
Det er valgt at gennemføre samtlige tests vha. en række scenarier, der tilsammen indeholder samtlige testcases, der ønskes behandlet.
Da disse scenarier på nuværende tidspunkt i rapporten vil være vanskelige at gennemskue, er det valgt først at beskrive disse i forbindelse med udførelsen af accepttesten.

8 Programdesign

Efter færdiggørelse af kravspecifikationen designes programmet, hvorved grundlaget for det endelige programmel lægges.
Dette afsnit indeholder design og gennemgang af de 3 forskellige hoveddele, GPS-simulator, AIS-simulator og terminal. Desuden gør programmelet brug af databaser til de to simulatorer, hvilket ligeledes beskrives og designes.
De forskellige hovedafsnit vil blive opdelt i henholdsvis grænseflader, use cases og klasser.
Det overordnede programdesign ses i figur 8.1.



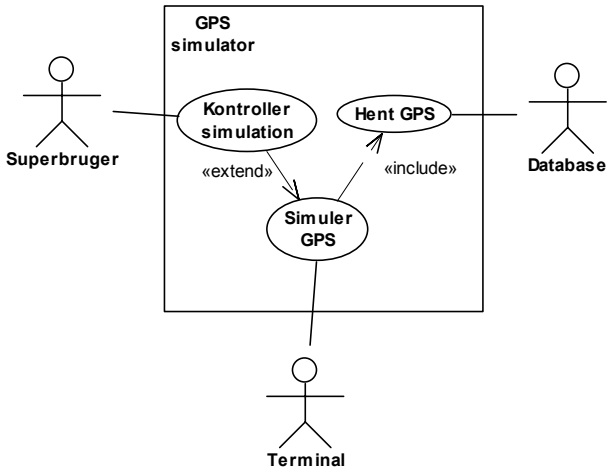
Figur 8.1: Use case diagram over hele systemet

I figur 8.1 angiver ”extend” og ”include” afhængigheden mellem de enkelte use cases. Disse afhængigheder er defineret således:

- *Include:* Inkluderer en anden use case således at "mål-use casen" ikke er afhængig af "kilde-use casen", men omvendt.
- *Extend:* Udvider en use case ved at dele den op i en central og en mindre central del.

8.1 GPS-simulator

I designet af softwaren til GPS-simulatoren tages der udgangspunkt i det opstillede use case diagram i figur 8.2.



Figur 8.2: Use case diagram over GPS-simulatoren

GPS-simulatoren skal opfylde de opstillede krav fra kravspecifikationen, hvilket er opsummeret nedenfor:

- Skal hente GPS-scenarier fra en database
- Skal simulere GPS-informationer, for eget skib, i en uafhængig PC og videresende disse via NMEA 0183 protokollen
- Skal udstyres med en pausefunktion

8.1.1 Grænseflader

Simulatorens eksterne grænseflader (aktører) er fastlagt til henholdsvis Superbruger, Database og Terminal.

Superbrugeren

Superbrugeren har til opgave at kontrollere simulationen. Kontrol af simulationen indeholder tre funktioner "start", "stop" og "pause". Funktionen "start" skal først og fremmest starte det scenarie superbrugeren har valgt, dernæst skal funktionen "start" starte timeren til opdatering af GPS simulatoren. Funktionen "stop" skal stoppe simulationen, medens funktionen "pause" skal kunne stoppe simulationen midlertidig, og starte igen fra samme punkt. Disse funktioner bruges i forbindelse med test af programmet. Desuden har superbrugeren mulighed for at korrigere data i databasen.

Database

Databasen skal indeholde information om de forskellig ruter der skal simuleres, databasens indhold er defineret i afsnit 10.

Terminal

Terminalen skal modtage det genererede GPS-signal i henhold til NMEA 0183.

8.1.2 Beskrivelse af use cases

Use case containeren i GPS-simulatoren indeholder tre use cases, ”Kontroller simulatoren”, ”Hent GPS” og ”Simuler GPS.

Kontroller simulatoren

Dette use case har til formål at servicere superbrugeren, den skal ved opstart sørge for, at det er det rigtige scenarie der bliver kørt i GPS-simulatoren. Den skal desuden sørge for at stoppe, starte eller pause GPS-simulatoren, når superbrugeren har valgt en af de tre funktioner.

Hent GPS

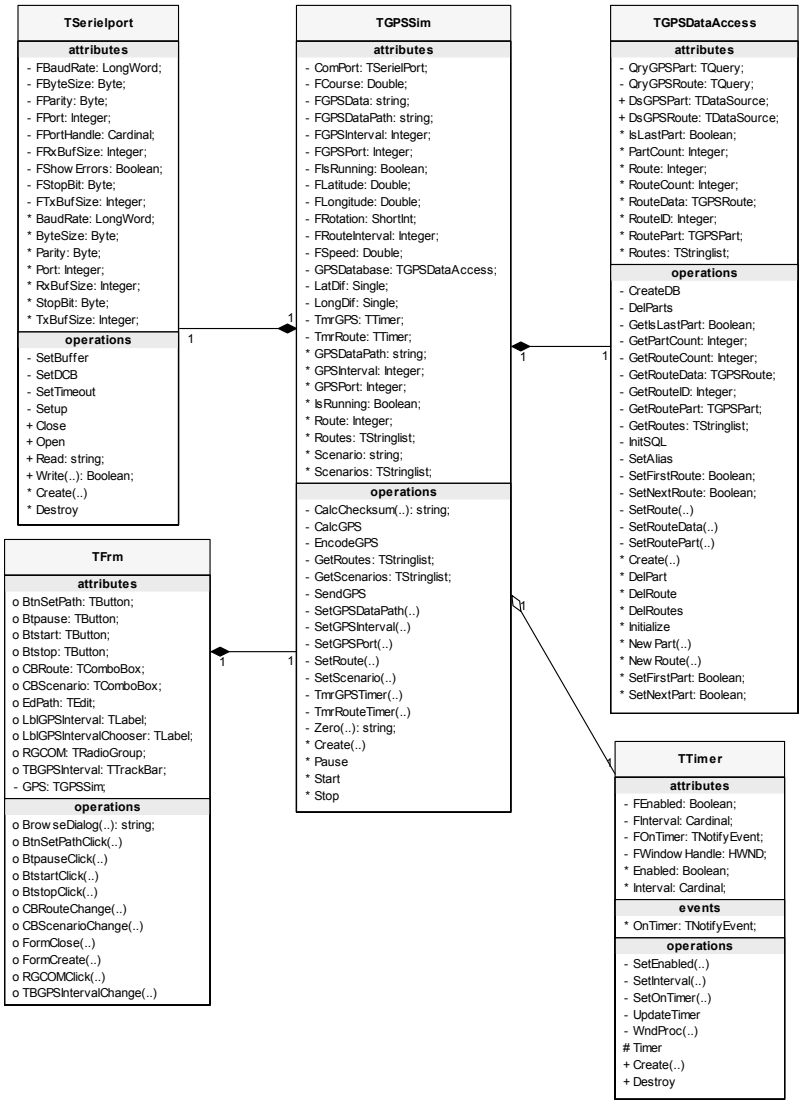
Use casen ”hent GPS” skal fra databasen hente det valgte scenarie og indlæse det i ”simuler GPS”.

Simuler GPS

Dette use case skal ud fra det valgte scenarie generere GPS-signaler, der sendes videre til terminalen. Samtidig skal ”simuler GPS” kunne reagere på ”start”, ”stop” og ”pause”.

8.1.3 Klasser

Efter fastsættelse af use cases for GPS-simulatoren, er det valgt at konstruere programdesignet med udgangspunkt i den på figur 8.3 viste inddeling af GPS-simulatoren i klasser med tilhørende metoder (operations).



Figur 8.3: Oversigt af klasser i GPS-simulatoren

Af figur 8.3 fremgår det at GPS-simulatoren er opbygget af fem klasser med tilhørende metoder. Klassen TSerielport gør det muligt for klassen TGPSSim at videresende simulerede GPS-informationer til terminalen. Denne klasse bliver desuden benyttet af terminalen og AIS-simulatoren.
Klassen TGPSSim simulerer GPS-informationer på baggrund af data fra databasen, der tilgås vha. klassen TGPSDataAccess.
Endvidere benyttes klassen TTimer af klassen TGPSSim som timer.
TTimer er en, i Delphi's VCL (Visual Component Library) bibliotek, indbygget klasse der giver mulighed for at eksekvere kode i bestemte intervaller. Det er således denne timer, der bestemmer hastigheden hvormed data bliver simuleret og afsendt. Metoderne i klassen TTimer vil ikke blive behandlet nærmere i dette projekt.
Klassen TGPSForm benyttes til at styre simuleringerne af de enkelte scenarier. Superbrugeren har således mulighed for bl.a. at starte, stoppe og pause simuleringerne, samt vælge scenarier via klassen TGPSForm. Endvidere giver denne klasse mulighed for at indstille hastigheden, hvormed GPS-informationer skal afsendes til terminalen.

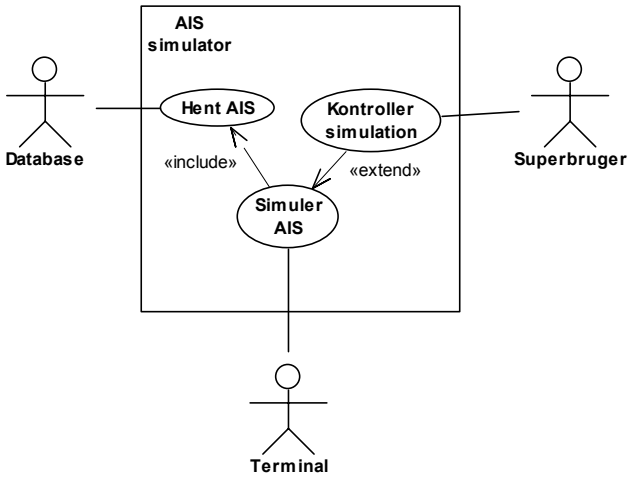
Attributterne i en klasse beskriver de data, der er indeholdt i et objekt af denne klasse. Events markerer en ændring i et objekts tilstand, hvilket initialiserer en handling ud fra den ændrede tilstand i form af et stykke kode. Operationerne definerer på hvilken måde objekterne kan samarbejde.
De enkelte attributter, events og metoder i klasserne kan have forskellig grad af tilgængelighed. Graden af tilgængelighed er givet i fire forskelle grader, og er defineret på følgende vis:

- *Private (-):* Kan kun tilgås fra egen klasse og klasserne i egen unit.
- *Protected (#):*Kan tilgås fra egen og andre klasser i samme unit samt disses subklasser.
- *Public(+):*Kan tilgås fra egen og alle andre klasser.
- *Published (*):*Kan tilgås fra egen og alle andre klasser samt Delphis Object Inspector.

Endvidere findes i Delphi en default indstilling, hvor Delphi selv vælger tilgængeligheden. Denne mulighed er markeret med (o) og er som standard published.

8.2 AIS-simulator

I designet af softwaren til AIS-simulatoren tages der udgangspunkt i det opstillede use case diagram i figur 8.4



Figur 8.4: Use case diagram over AIS-simulatoren

AIS-simulatoren skal opfylde de opstillede krav fra kravspecifikationen, hvilket er opsumeret nedenfor:

- Skal kunne simulere op til 450 skibe, af forskellig fart og kurs indenfor et afgrænset område på op til 25 sømil i radius. Dette gøres da en realistisk situation ikke vil kunne overstige dette
- Skal simulere AIS-informationer i en uafhængig PC og videresende disse via den udvidede version af NMEA 0183 standarden
- Skal udstyres med en pausefunktion
- Skal afsende AIS-informationer med det i AIS standarden angivne tidsinterval
- Skal kunne hente AIS-scenarier fra en database

8.2.1 Grænseflader

AIS-simulatorens eksterne grænseflader er fastlagt til Superbruger, Database og Terminal.

Superbrugeren

Superbrugeren har samme formål som det er tilfældet i forbindelse med GPS-simulatoren.

Database
Databasen skal indeholde informationer om de enkelte skibe som det ønskes at simulere AIS-informationer for. Indhold og opbygning af databassen er beskrevet i afsnit 10.

Terminal
Terminalens opgave, er at modtage de genererede AIS-informationer i henhold til NMEA 0183.

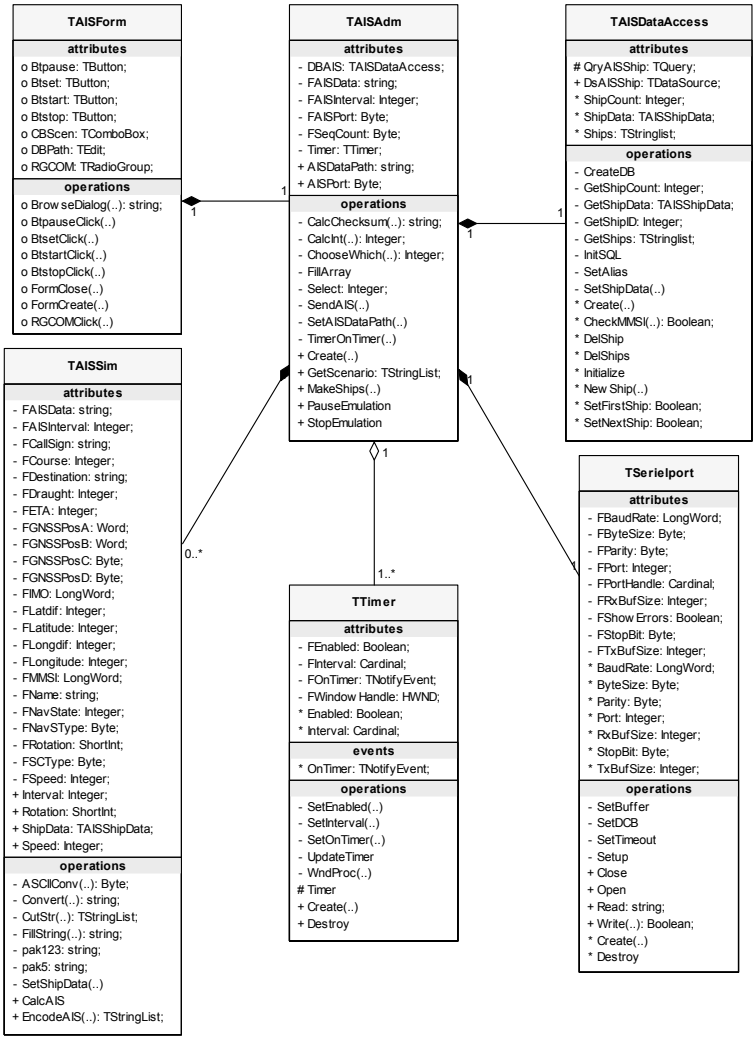
8.2.2 Beskrivelse af use cases
Use case containeren i AIS-simulatoren indeholder tre use cases, ”Kontroller simulatoren”, ”Hent AIS” og ”Simuler AIS.

Kontroller simulatoren
Denne use case har til formål at servicere superbrugeren, den skal ved opstart sørge for, at det er det rigtige scenarie der bliver kørt i AIS-simulatoren. Den skal desuden sørge for at stoppe, starte eller pause AIS-simulatoren, når superbrugeren har valgt en af de tre funktioner.

Hent AIS
Use casen ”hent AIS” skal fra databasen hente det valgte scenarie og indlæse det i ”simuler AIS”.

Simuler AIS
Denne use case skal ud fra det valgte scenarie generere AIS-signaler, der sendes videre til terminalen. Samtidig skal ”simuler AIS” kunne reagere på ”start”, ”stop” og ”pause”.

8.2.3 Klasser
Efter fastsættelse af use cases for AIS-simulatoren videreudvikles programdesignet. Dette gøres med udgangspunkt i inddeling af AIS-simulatoren i klasser og metoder, hvilket er illustreret i figur 8.5.

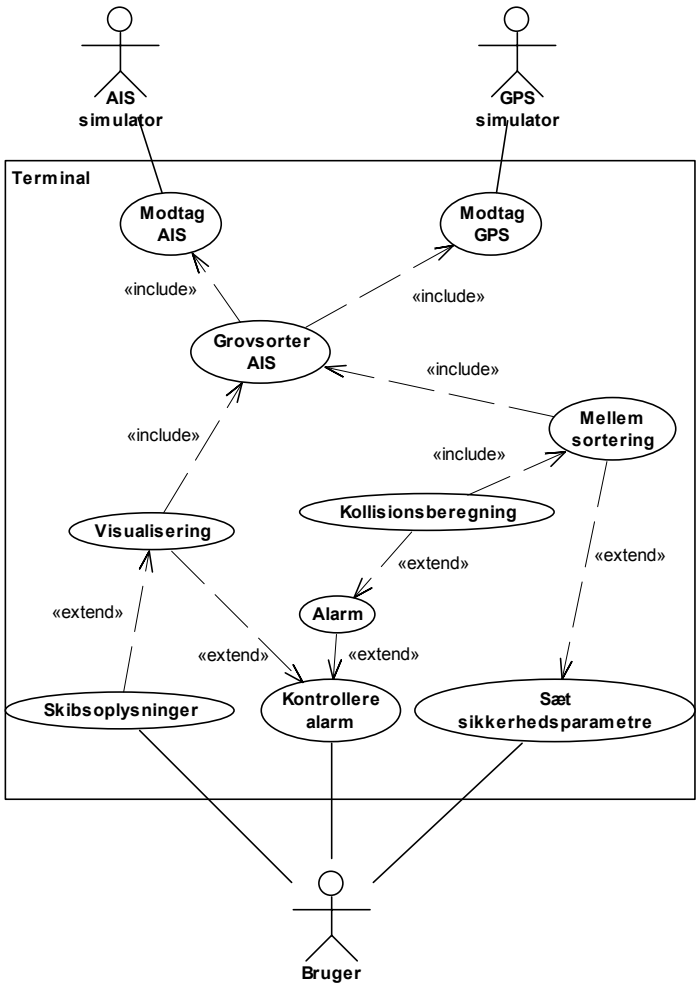


Figur 8.5: Oversigt af klasser i AIS-simulatoren

Af figur 8.5 fremgår, at det er valgt, at inddele AIS-simulatoren i seks klasser.
Klassen TSerielport gør det muligt for klassen TAISAdm at videresende simulerede AIS-informationer til terminalen.
Klassen TAISAdm administrerer timingen og styringen af AIS-simulatoren ved at hente AIS-informationer fra databasen vha. klassen TAISDataAccess. Herefter videresendes disse informationer til klassen TAISSim, der koder og pakker AIS-informationer i henhold til den udvidede NMEA 0183 standard.
Endvidere benyttes klassen TTimer af klassen TAISAdm som timer.
Endelig benyttes klassen TAISForm til at styre simuleringerne af de enkelte scenarier. Superbrugeren har således mulighed for bl.a. at starte, stoppe og pause simuleringerne, samt vælge scenarier via klassen TAISForm.

8.3 Terminal

I designet af softwaren til terminalen tages der udgangspunkt i det opstillede use case diagram i figur 8.6:



Figur 8.6: Use case diagram over terminalen

Terminalen skal opfylde de opstillede krav fra kravspecifikationen som er opsummeret nedenfor:

- Skal kunne modtage AIS-informationer
- Skal kunne modtage GPS-informationer
- Skal kunne modtage og behandle AIS-informationer svarende til 450 skibe
- Skal give en grafisk visualisering, herunder en zoomfunktion af den omkringliggende skibstrafik
- Skal give en visuel og auditiv alarm ved kollisionsrisiko
- Auditiv alarm skal kunne deaktiveres af brugeren
- Skal kunne give en række AIS-informationer vedrørende de omkringliggende skibe
- Skal give brugeren mulighed for at indstille sikkerhedsafstand og reaktionstid

8.3.1 Grænseflader

Simulatorens eksterne grænseflader er fastlagt til henholdsvis Bruger, AIS-simulator og GPS-simulator:

Bruger

Brugeren har flere forskellige muligheder for at opnå indflydelse i forbindelse med terminalens funktioner. For det første har brugeren mulighed for, igennem terminalen, at indhente skibsoplysninger vedrørende alle skibe der er repræsenteret i skærmbilledet. Brugeren har endvidere

muligheden for at slukke for alarmen hvis denne lyder, eller helt deaktivere alarmen. Brugeren kan ligeledes vælge at indstille værdier for henholdsvis sikkerhedsafstand og reaktionstid, vha. terminalen. Endeligt kan brugeren vælge hvilken visualisering han ønsker, en radius mellem 1 og 25 sømil.

AIS-simulator
AIS-simulatoren skal afsende AIS-signaler til terminalen i henhold til den udvidede NMEA 0183.

GPS-simulator
GPS-simulatoren skal afsende GPS-signaler til terminalen i henhold til NMEA 0183.

8.3.2 Beskrivelse af use cases

De til terminalen hørende use cases, som er illustreret i figur 8.6, vil efterfølgende blive beskrevet.

Modtag AIS
Denne use case skal modtage AIS-informationer fra AIS-simulatoren og herefter sørge for, at disse informationer bliver dekodet fra NMEA 0183 til AIS-data og lagret i de tilhørende variable.

Modtag GPS
Denne use case skal modtage GPS-informationer fra GPS-simulatoren, behandle disse data og opdatere eget skibs position ud fra dataene.

Grovsorter AIS
De første beregninger foregår i denne use case. Det er her dataene fra GPS'en omregnes til x,y-koordinater, og eget skib placeres i origo. Herefter afgøres hvilke skibe der er kategori 1 og hvilke der er kategori 2.

Mellemsortering
I denne use case afgøres det hvorvidt de resterende skibe er kategori 2, 3 eller 4.

Kollisionsberegning
I use casen "kollisionsberegning" foregår den beregning på kategori 4 skibe der er beskrevet i appendiks C.4.

Alarm
Dette er use casen der skal slå alarmen til i det tilfælde, at et skib er på kollisionskurs.

Visualisering
Benyttes til at indhente og plotte skibsdata, samt ændre visualiseringen af skærmbilledet mellem 1 og 25 sømil.

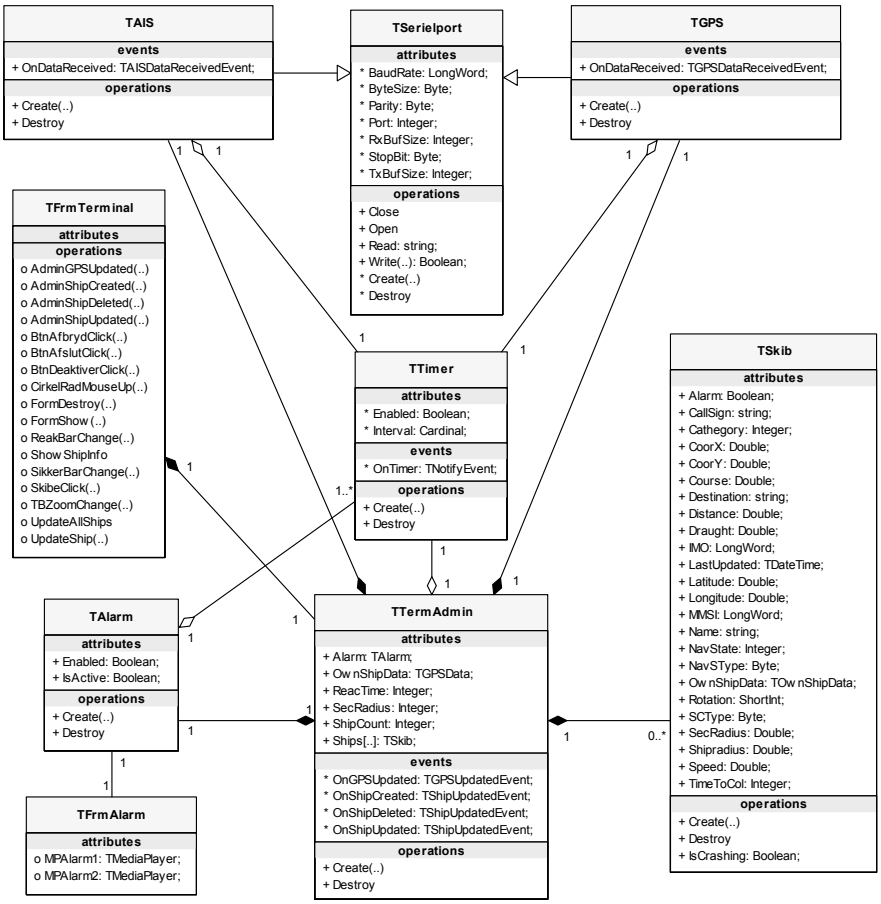
Skibsoplysninger
Denne use case giver brugeren mulighed for at vælge et skib indenfor skærmbilledet og få alle relevante data vist på skærmen.

Kontrollere alarm
Denne use case giver brugeren muligheden for at slå alarmen fra i det tilfælde at denne lyder. Brugeren kan også vælge helt at deaktivere alarmen eller aktivere den igen.

Sæt sikkerhedsparametre
Her kan brugeren indtaste værdier for henholdsvis sikkerhedsafstand og ønsket reaktionstid, vha. terminalen.

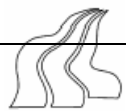
8.3.3 Klasser

Terminalen er inddelt i ni klasser, og det er valgt, pga. af klassernes størrelse, kun at vise de metoder og property's der er public, hvilket er illustreret i figur 8.7



Figur 8.7: Oversigt over klasser til terminalen

Klassen TSerielport har to subklasser TGPS og TAIS, der hver især dekoder henholdsvis GPS- og AIS-informationer, modtaget fra GPS- og AIS-simulatoren.
TGPS dekoder GPS-informationer hvorefter disse data sendes videre til klassen TTermAdmin via et event. Dette kontrolleres af et objekt af klassen TTimer, der sætter dekodningen i gang.
Klassen TAIS dekoder modtaget data hver gang TTimer instruerer den til dette, og sender data videre til TTermAdmin.
TSkib foretager beregninger på baggrund af modtaget data fra TTermAdmin og sender resultatet tilbage til TTermAdmin. Her videresendes dette til TFrmTerminal. Klassen TFrmTerminal benytter disse resultater til bl.a. at plotte omkringliggende skibe på brugerfladen samt indhente skibsdata for disse skibe.
Endelig aktiverer og deaktiverer klassen TAlarm alarmen på foranledning af klassen TTermAdmin.



9 Proces-/moduldesign

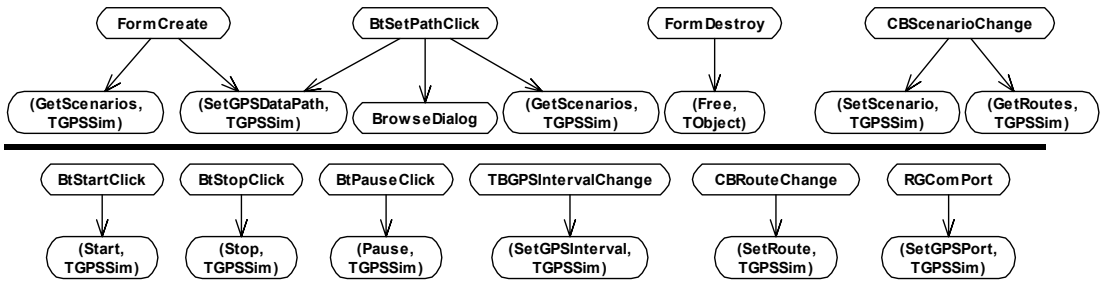
Efter det overordnede programdesign er klarlagt, og de enkelte klasser og metoder dermed er bestemt, beskrives programmet fra et lavere abstraktionsniveau. Dette afsnit vil derfor behandle funktionerne af de enkelte metoder der er nævnt i programdesignet.
Beskrivelsen af de enkelte metoder er delt ind i følgende fire hovedafsnit: GPS-simulator, AIS-simulator, Terminal og fælles klasser.
Disse hovedafsnit vil blive behandlet én klasse ad gangen. I forbindelse med hver klasse beskrives sammenhængen mellem klassens egne metoder og andre klassers metoder.
Det er således valgt at kombinere proces- og moduldesign i dokumentationen af disse.

9.1 GPS-simulator

GPS-simulatoren består, som nævnt i afsnit 8.1.3, af klasserne TGPSForm, TGPSSim og TGPSDataAccess samt fællesklasserne TserielPort og TTimer. Samtlige metoder i klasserne TGPSForm og TGPSSim, samt de relevante metoder i klassen TGPSDataAccess vil blive gennemgået i det følgende. Metoderne i klassen TSerielPort beskrives i afsnit 9.4.

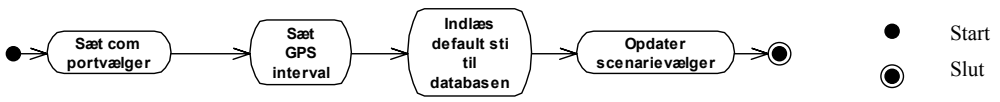
9.1.1 TGPSForm

Klassen TGPSForm gør det muligt for superbrugeren at kontrollere GPS-simulationen. Dette gøres vha. af fire knapper start, stop, pause og sæt samt en serielport-, rute og scenarievælger. TGPSForm udgør således GPS-simulatorens Graphic User Interface (GUI).
Inden beskrivelse af de enkelte metoder i klassen TGPSForm er det valgt at illustrere sammenhængen mellem de enkelte metoder, hvilket ses i figur 9.1.



Figur 9.1: Metodesammenhæng for klassen TGPSForm

FormCreate

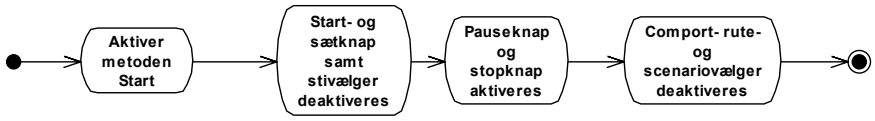


Figur 9.2: Aktivitetsdiagram over metoden FormCreate

FormCreate (se figur 9.2) køres når GPS-simulatoren startes op, og opretter et objekt af klassen TGPSSim. Dernæst sætter metoden serielporten til default porten 1.
Intervallet hvormed superbrugeren ønsker, at GPS-informationer skal afsendes, sættes ved opstart til én gang i sekundet.
I det tilfælde, at GPS-simulationen har været kørt tidligere, vil stien til databasen være gemt i en ini fil. Denne fil tilgås, og default stien til databasen sættes til stien angivet i filen. Dette sker ved at skrive til property'en GPSDataPath, der henviser til SetGPSDataPath, i objektet GPS af klassen

TGPSSim. Indeholder filen en gyldig sti til databasen, vil FormCreate afslutningsvis opdatere scenarievælgeren GetScenarios, i klassen TGPSSim.

FormDestroy
Benyttes til at nedlægge Objektet GPS af klassen TGPSSim vha. metoden Free, i klassen TObject. Denne klasse er en indbygget delphi klasse.

BtStartClick


Figur 9.3: Aktivitetsdiagram over metoden BtStartClick

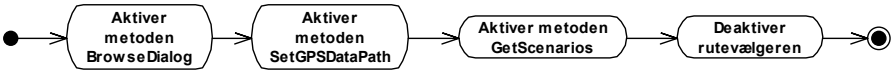
Metoden BtStartClick (se figur 9.3) har til formål at starte et GPS-scenarie, der simulerer GPS-informationer for eget skib, hvilket gøres ved at trykke på knappen start.
Efter tryk på startknappen skal metoden Start, i klassen TGPSSim, aktiveres, hvorved GPS-simulationen påbegyndes. Efterfølgende skal startknappen deaktiveres da det ikke skal være muligt at opstarte et nyt scenarie, når et sådant allerede er sat i gang. Ligeledes skal det ikke længere være muligt at vælge databasestien vha. sætknappen. Derpå aktiveres pause- og stopknapperne, således scenariet kan pauses eller stoppes. Endelig deaktiveres serielport-, rute- og scenarievælgerne da disse ikke skal kunne benyttes efter opstart af et scenarie.
Formålet med serielportvælgeren er at gøre det muligt for superbrugeren at bestemme, hvilken serielport der skal benyttes til afsendelse af GPS-informationer til terminalen, der også skal modtage AIS-informationer. Scenarievælgeren gør det muligt for superbrugeren at vælge mellem scenarier i databasen, medens rutevælgeren gør det muligt at bestemme rute indenfor det valgte scenarie.

BtStopClick


Figur 9.4: Aktivitetsdiagram over metoden BtStopClick

Metoden BtStopClick (se figur 9.4) har til formål at stoppe en allerede aktiveret GPS-simulation, hvilket gøres ved at trykke på knappen stop.
Dette gøres ved at aktivere metoden Stop i klassen TGPSSim.
Efter tryk på stopknappen skal denne og pauseknappen deaktiveres, da disse ikke skal være tilgængelige når en GPS-simulation ikke er aktiv.
Serielport-, rute- og scenarievælgerne skal aktiveres, idet det efter tryk på stopknappen skal være muligt, at ændre i parametrene for GPS-simulationen. Ligeledes skal start- og sætknappen aktiveres, da det naturligvis skal være muligt at starte GPS-simulationen ud fra det samme eller et andet scenarie.

BtPauseClick
Metoden BtPauseClick har til formål at pause en allerede aktiv GPS-simulation, således denne kan genopstartes fra samme tidspunkt den blev pauset.
Dette opnås ved at aktivere metoden Pause, i klassen TGPSSim.

BtSetPathClick


Figur 9.5: Aktivitetsdiagram over metoden BtSetPathClick

BtSetPathClick (se figur 9.5) benyttes til at vælge stien til databasen med GPS-scenarier.
Først aktiveres metoden BrowseDialog. Denne metode giver superbrugeren mulighed for at specificere stien til den aktuelle database. Herefter initialisereres databasen med den nye sti. Dette sker ved at skrive til property'en GPSDataPath, der henviser til SetGPSDataPath, i objektet GPS af klassen TGPSSim.
Endvidere indlæses nye scenarietitler i scenarievælgeren, på baggrund af den nyvalgte database. Dette sker ved at læse property'en Scenarios, der henviser til GetScenarios, i objektet GPS af klassen TGPSSim.
Endelig deaktiveres rutevælgeren, idet superbrugeren endnu ikke har valgt et scenarie.

CBScenarioChange

Figur 9.6: Aktivitetsdiagram over metoden CBScenarioChange

Metoden CBScenarioChange (se figur 9.6) aktiveres af scenarievælgeren, og benyttes til at vælge hvilket scenarie, det ønskes at simulere GPS-informationer for.
Indledningsvis aktiveres det af superbrugeren valgte scenarie. Dette sker ved at skrive til property'en Scenario, der henviser til SetScenario, i objektet GPS af klassen TGPSSim.
Herpå aktiveres rutevælgeren, da det nu skal være muligt at vælge en rute for det valgte scenarie.
Afslutningsvis hentes de mulige ruter, der findes i det valgte scenarie, ved at læse property'en Routes, der henviser til GetRoutes, i objektet GPS af klassen TGPSSim.

CBRouteChange

Denne metode aktiveres af rutevælgeren og informerer databasen om hvilken rute der er valgt, ved at skrive til property'en Route, der henviser til SetRoute, i objektet GPS af klassen TGPSSim.

RGCOMClick

Denne metode aktiveres af serielportvælgeren og informerer klassen TserielPort om hvilken serielport der skal benyttes. Dette sker ved at skrive til property'en GPSPort, der henviser til SetGPSPort, i objektet GPS af klassen TGPSSim.

TBGPSIntervalChange

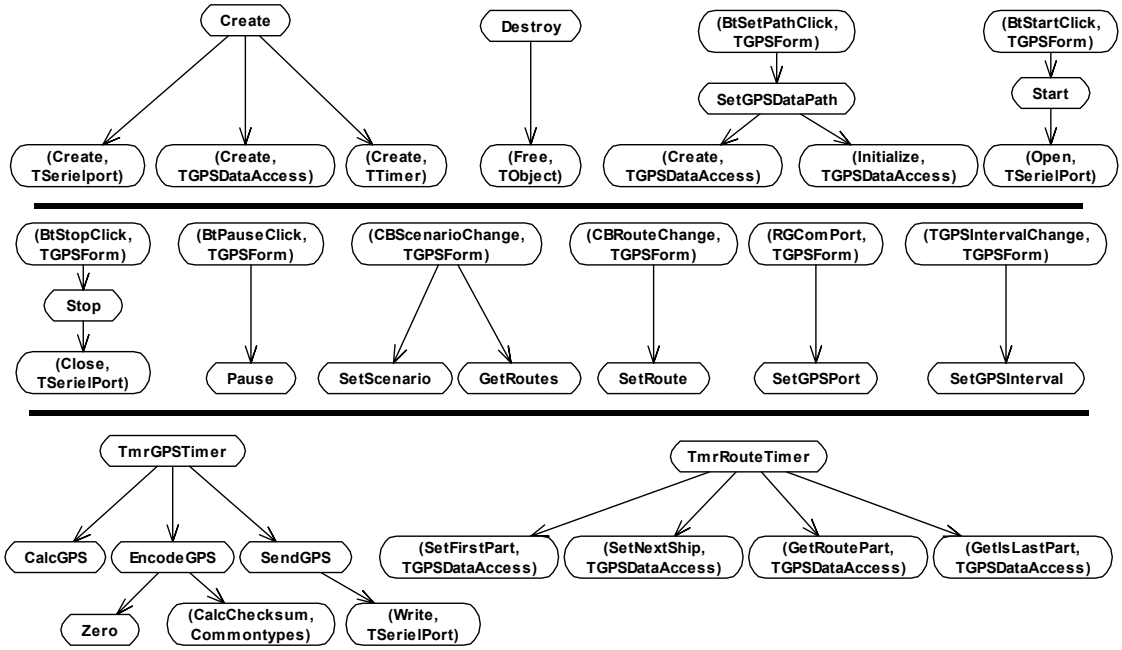
Metoden TGPSIntervalChange aktiveres af intervalvælgeren. Metoden sætter det valgte GPS-interval. Dette udføres ved at skrive til property'en GPSInterval. Denne property henviser til SetGPSInterval, i objektet GPS af klassen TGPSSim.

FormClose

Metoden FormClose aktiveres når GPS-simulatoren afsluttes, og har som eneste formål, at oprette et objekt hvori den valgte databasesti lægges i en ini fil. Herefter nedlægges objektet vha. metoden Free, i klassen TObjekt.

9.1.2 TGPSSim

Klassen TGPSSIM indeholder de metoder, i GPS-simulatoren, der omhandler kodning, pakning og administrering af GPS-informationerne.
Sammenhængen mellem de enkelte metoder, er illustreret i figur 9.7.



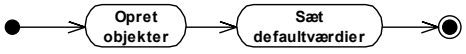
Figur 9.7: Metodesammenhæng for klassen TGPSSim

Til kommunikation klasserne imellem, er properties'ene i tabel 9.1 indeholdt i TGPSSim:

Navn	Datatype	Læser fra	Skriver til
GPSPort	Integer	FGPSPort	SetGPSPort
GPSInterval	Integer	FGPSInterval	SetGPSInterval
GPSDataPath	String	FGPSDataPath	SetGPSDataPath
Scenarios	TStringlist	GetScenarios	-
Scenario	String	-	SetScenario
Routes	TStringlist	GetRoutes	-
Route	Integer	-	SetRoute
IsRunning	Boolean	FIsRunning	-

Tabel 9.1: Properties indeholdt i klassen TGPSSim

Create



Figur 9.8: Aktivitetsdiagram over metoden Create

Metoden Create (se figur 9.8) aktiveres af metoden CreateForm i klassen TGPSForm og har til formål at oprette objekter af klasserne TSerielport, TTimer og TGPSDataAccess.
Der oprettes således ét objekt til kontrol af serielporten, to timerobjekter, der styrer timingen i rutevælgeren og intervallet hvormed GPS-informationer skal afsendes og ét objekt der giver adgang til databasen.
Endvidere sættes defaultværdierne for disse objekter.

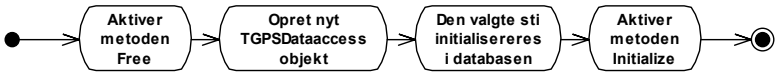
Destroy

Benyttes til at nedlægge de objekter der blev oprettet i metoden Create, i klassen TGPSSim. Dette gøres vha. metoden Free, i klassen TObject.

GetScenarios

Metoden GetScenarios aktiveres af metoden FormCreate eller BtSetPathClick, beliggende i klassen TGPSForm, igennem property'en Scenarios.
Den har til formål at indlæse scenarietitler i scenarievælgeren. Dette sker ved at skabe en forbindelse mellem Scenarievælgeren og objektet GPSDatabase.

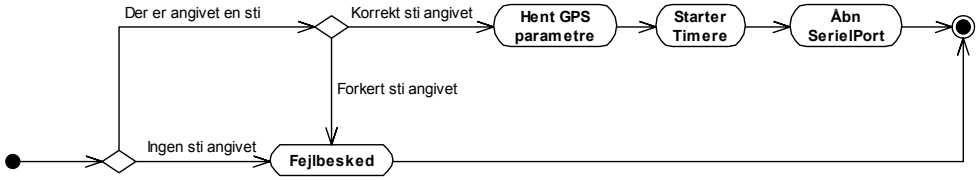
SetGPSDataPath



Figur 9.9: Aktivitetsdiagram over metoden SetGPSDataPath

SetGPSDataPath (se figur 9.9) kaldes af metoderne FormCreate og BtSetPathClick, i klassen TGPSForm, igennem property'en GPSDataPath.
Først aktiveres metoden Free, beliggende i klassen TObject. Denne metode er en indbygget Delphi metode, der er i stand til at nedlægge objekter. Metoden benyttes her til at nedlægge database objektet DBAIS beliggende i TGPSDataAccess klassen. Dernæst oprettes dette objekt, og dermed databasen igen. Dette gøres via metoden Create, beliggende i klassen TGPSDataAccess.
Den valgte sti associeres nu, i klassen TGPSDataAccess, med objektet GPSDatabase, således der nu er adgang til den valgte database.
Endeligt aktiveres metoden Initialize, igen beliggende i klassen TGPSDataAccess, der initialiserer objektet GPSDatabase.

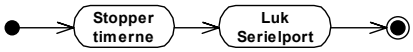
Start



Figur 9.10: Aktivitetsdiagram over metoden Start

Metoden Start (se figur 9.10) kaldes af metoden BtStartClick i klassen TGPSForm, og har til opgave at hente GPS-parametre fra den valgte database og starte GPS-simulationen.
Først kontrolleres hvorvidt en sti til databasen er angivet eller ej. Er dette ikke tilfældet meldes en fejlbesked. Er der derimod angivet en sti kontrolleres det hvorvidt denne sti eksisterer eller ej. Igen meldes en fejlbesked hvis dette ikke er tilfældet. Eksisterer stien hentes GPS-parametre omhandlende position, fart og rotation. Herpå startes timerne for henholdsvis rutevælgeren og intervallet hvormed GPS-informationer skal afsendes. Endelig åbnes serielporten vha. metoden Open i klassen TSerielPort.

Stop



Figur 9.11: Aktivitetsdiagram over metoden Stop

Metoden Stop (se figur 9.11) aktiveres af metoden BtStopClick i, klassen TGPSForm og har til opgave at stoppe GPS-simulationen.
Dette gøres ved at stoppe de to førnævnte timere, hvorefter serielporten lukkes vha. metoden Close, i klassen TSerielPort.

Pause
Metoden Pause aktiveres af metoden BtPauseClick, beliggende i klassen TGPSForm.
Kører GPS-simulationen som normalt skal timerne stoppes, hvorimod de skal startes i det tilfælde at GPS-simulationen allerede *er* pauset.

SetScenario
SetScenario kaldes af metoden CBScenarioChange, i klassen TGPSForm, igennem property'en Scenario.
Formålet med denne metode er at fortælle databasen hvilket scenarie superbrugeren har valgt.

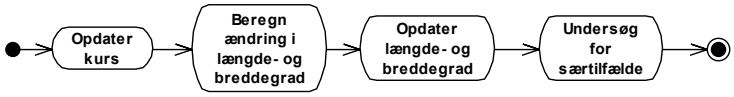
GetRoutes
Denne metode kaldes af metoden CBScenarioChange, i klassen TGPSForm, igennem property'en Routes.
Hensigten med denne metode er at hente de ruter, der er at vælge i mellem, for det nu valgte scenarie.

SetRoute
SetRoute kaldes af metoden CBRouteChange, i klassen TGPSForm, igennem property'en Route.
Formålet med denne metode er at fortælle databasen hvilken rute superbrugeren har valgt.

SetGPSPort
Metoden kaldes af RGComPort, placeret i klassen TGPSForm, igennem property'en GPSPort.
Denne metode angiver, hvilken serielport superbrugeren ønsker at benytte, til afsendelse af GPS-informationer. Dette gøres ved at lægge valget over i en variabel, i klassen TSerielPort.

SetGPSInterval
Metoden kaldes af metoden TGPSIntervalChange, i klassen TGPSForm, igennem property'en GPSInterval.
Hensigten med denne metode er, at angive med hvilket interval GPS-informationerne ønskes afsendt. Dette gøres ved at lægge det valgte interval over i en variabel, i klassen TTimer.

TmrGPSTimer
Metoden TmrGPSTimer aktiveres af den timer (TmrGPS) der bestemmer, hvor ofte GPS-informationer skal afsendes. Når dette sker skal metoderne CalcGPS, EncodeGPS og SendGPS aktiveres.

CalcGPS


Figur 9.12: Aktivitetsdiagram over metoden CalcGPS

Metoden CalcGPS (Se figur 9.12) aktiveres af metoden TmrGPSTimer, i klassen TGPSSim.
Formålet med CalcGPS er at beregne eget skibs position, ud fra et sæt startbetingelser, og timeren TmrGPS´s timinginterval.
Først beregner metoden en ny kurs ud fra ligning 9.1

$$\text{Kurs} = \text{Kurs}_{\text{opr}} + \left(\frac{\text{Rotation} \cdot \text{Timinginterval}}{60000} \right) \qquad (9.1)$$

Den opdaterede kurs findes ved at addere den oprindelige kurs, der er angivet i databasen, med et led givet ved rotationen og den tid der er gået siden sidste opdatering. Indholdet i parentesen findes

ved at gange rotationen sammen med timingintervallet og dividere dette med 60000. Der divideres med 60000 da rotationen er givet i grader pr. minut og timingintervallet er givet i millisekunder. Efterfølgende tages der højde for det tilfælde, at kursen ikke ligger i intervallet 0-360 grader. Ændringen i længde per timinginterval findes ud fra ligning 9.2.

$$\Delta\text{Longitude} = \frac{\text{Fart} \cdot \text{Timinginterval} \cdot \cos(\text{kurs}) \cdot \cos\left(\frac{\text{Latitude}}{60}\right)}{3600000} \qquad (9.2)$$

I ligning 9.2 tages der hensyn til hvilken bredde eget skib befinder sig på, samt at den startposition er givet i minutter. Endvidere tages der hensyn til at farten er givet i knob, hvorfor der divideres med 3600000 for at få farten i sømil per millisekund. Dette gøres da timingintervallet er givet i millisekunder. Ændringen i længde per timinginterval er således givet i minutter per timinginterval. Ændringen i bredden per timinginterval findes ud fra ligning 9.3

$$\Delta\text{Latitude} = \frac{\text{Fart} \cdot \text{Timinginterval} \cdot \sin(\text{kurs})}{3600000} \qquad (9.3)$$

Efterfølgende lægges resultaterne af ligning 9.2 og 9.3 til positionen inden den aktuelle opdatering. Endelig undersøges hvorvidt eget skib har krydset datolinien. Dette udføres ved at undersøge, om længden er numerisk større end 180º, og korrigere for dette.

EncodeGPS

EncodeGPS kaldes af metoden TmrGPSTimer i klassen TGPSSim.
Formålet med denne metode er at pakke GPS-informationerne i henhold til NMEA 0183 standarden. I forbindelse med denne metode kaldes metoden Zero og metoden CalcChecksum, beliggende i unit'en Commontypes.

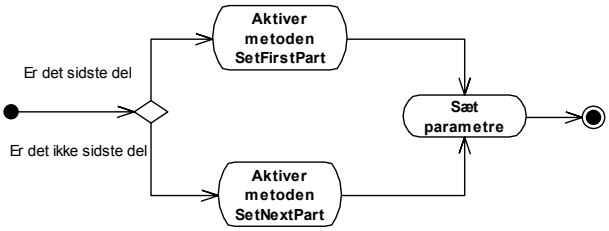
Zero

Metoden Zero aktiveres af metoden EncodeGPS, i samme klasse.
Denne metode har til formål at kunne formatere et tal til at optage et specifikt antal pladser jf. NMEA 0183.

SendGPS

SendGPS kaldes af metoden TmrGPSTimer i klassen TGPSSim.
Formålet med denne metode er at afsende pakken af GPS-informationer som metoden EncodeGPS har pakket. Dette gøres vha. metoden Write i klassen TSerielport.

TmrRouteTimer



Figur 9.13: Aktivitetsdiagram over metoden TmrRouteTimer

Da den, i databasen, valgte rute kan indeholde en kombination af ruter, i form af forskellige parametre, er det nødvendigt at bestemme hvor ofte disse parametre skal ændres. Dette gøres vha. timeren TmrRoute, der aktiverer metoden TmrRouteTimer (se figur 9.13).
Er den aktuelle rutedel, af en valgt rute, den sidste del i denne rute, skal metoden SetFirstPart, i klassen TGPSDataAccess aktiveres. Ellers skal metoden SetNextPart, ligeledes i klassen TGPSDataAccess, aktiveres. Disse to metoder vælger den første eller næste record i databasen over rutedele.
Afslutningsvis sættes parametrene for den nye rutedel.

9.1.3 TGPSDataAccess

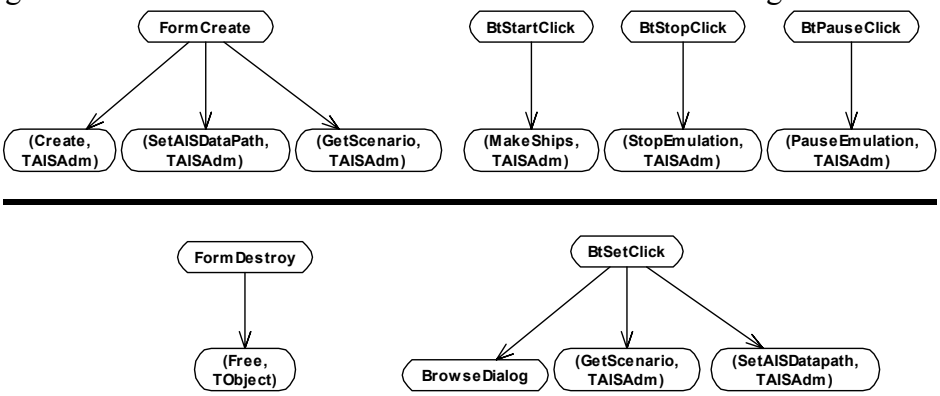
Denne klasse giver adgang til databasen. Den er beskrevet nærmere under afsnit 10.2.3.

9.2 AIS-simulator

AIS-simulatoren indeholder som førnævnt klasserne TAISSim, TAISAdm, TAISForm, TAISDataAccess samt fællesklasserne TSerielPort og TTimer.
Samtlige metoder i de tre førstnævnte klasser, samt de relevante metoder i klassen TAISDataAccess vil blive gennemgået i det følgende. Metoderne i klassen TSerielPort beskrives, som førnævnt, i afsnit 9.4.

9.2.1 TAISForm

Klassen TAISForm gør det muligt for superbrugeren at kontrollere AIS-simulationen. Dette gøres, i lighed med TGPSForm vha. af fire knapper start, stop, pause og sæt samt en serielport- og scenarievælger.
Sammenhængen mellem de enkelte metoder i klassen TAISForm ses i figur 9.14



Figur 9.14: Metodesammenhæng for klassen TAISForm

FormCreate

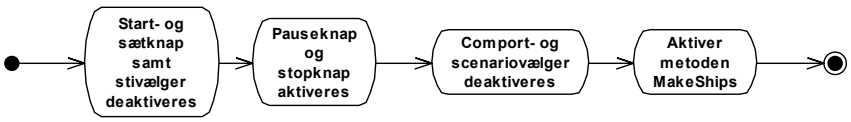
Figur 9.15: Aktivitetsdiagram over metoden FormCreate

FormCreate (se figur 9.15) køres når AIS-simulatoren startes op, og opretter et objekt af klassen TAISAdm. Dernæst sætter metoden serielporten til default porten 1.
I det tilfælde, at AIS-simulationen har været kørt tidligere, vil stien til databasen være gemt i en ini fil. Denne fil tilgås, og default stien til databasen sættes til stien angivet i filen. Dette sker ved at skrive til property'en AISDataPath, der henviser til SetAISDataPath, i objektet AIS af klassen TAISAdm.

Indeholder filen en gyldig sti til databasen, vil FormCreate afslutningsvis opdatere scenarievælgeren via metoden GetScenario, i klassen TAISAdm.

FormDestroy
Benyttes til at nedlægge objektet af klassen TGPSAdm oprettet i FormCreate, i klassen TAISForm, vha. metoden Free, i klassen TObject.

BtStartClick



Figur 9.16: Aktivitetsdiagram over metoden BtStartClick

Metoden BtStartClick (se figur 9.16) har til formål at starte et scenarie af skibe der skal simuleres, hvilket gøres ved at trykke på knappen start.
Efter tryk på startknappen skal denne deaktiveres da det ikke skal være muligt at opstarte et nyt scenarie når et sådant allerede er sat i gang. Ligeledes skal det ikke længere være muligt at vælge database stien vha. sætknappen. Efterfølgende aktiveres pause- og stopknapperne, således scenariet kan pauses eller stoppes. Serielport- og scenarievælgerne deaktiveres da disse ikke skal kunne benyttes efter opstart af et scenarie.
Formålet med serielportvælgeren er at gøre det muligt for superbrugeren at bestemme, hvilken serielport der skal benyttes til afsendelse af AIS-informationer til terminalen, der også skal modtage GPS-informationer. Scenarievælgeren gør det muligt for superbrugeren at vælge mellem scenarier i databasen.
Endeligt skal proceduren MakeShips aktiveres, hvilken beskrives under klassen TAISAdm afsnit 9.2.2.

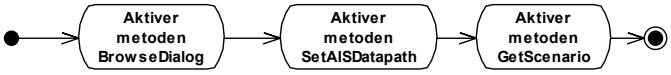
BtStopClick

Figur 9.17: Aktivitetsdiagram over metoden BtStopClick

Metoden BtStopClick (se figur 9.17) har til formål at stoppe en allerede aktiveret AIS-simulation, hvilket gøres ved at trykke på knappen stop.
Efter tryk på stopknappen skal denne og pauseknappen deaktiveres, da disse ikke skal være tilgængelige når en AIS-simulation ikke er aktiv.
Serielport- og scenarievælgerne skal aktiveres, idet det efter tryk på stopknappen skal være muligt, at ændre i parametrene for AIS-simulationen. Ligeledes skal start- og sætknappen aktiveres, da det skal være muligt at starte AIS-simulationen ud fra det samme eller et andet scenarie. Endeligt skal metoden StopEmulation, i klassen TAISAdm, aktiveres.

BtPauseClick
Metoden BtPauseClick har til formål at pause en allerede aktiv AIS-simulation, således denne kan genopstartes fra samme punkt den blev pauset.
Dette opnås ved at aktivere metoden PauseEmulation, i klassen TAISAdm.

BtSetClick



Figur 9.18: Aktivitetsdiagram over metoden BtSetClick

BtSetClick (se figur 9.18) benyttes til at vælge stien til databasen med AIS-scenarier.
Først aktiveres metoden BrowseDialog. Denne metode giver superbrugeren mulighed for at specificere stien til den aktuelle database. Herefter initialiseres den nye database med sti, ved at skrive til property'en AISDataPath, der henviser til SetAISDataPath, i objektet AIS af klassen TAISAdm.
Endeligt aktiveres metoden GetScenario, beliggende i TAISAdm. Denne indlæser nye scenarietitler i scenarievælgeren, på baggrund af den nyvalgte database.

RGCOMClick

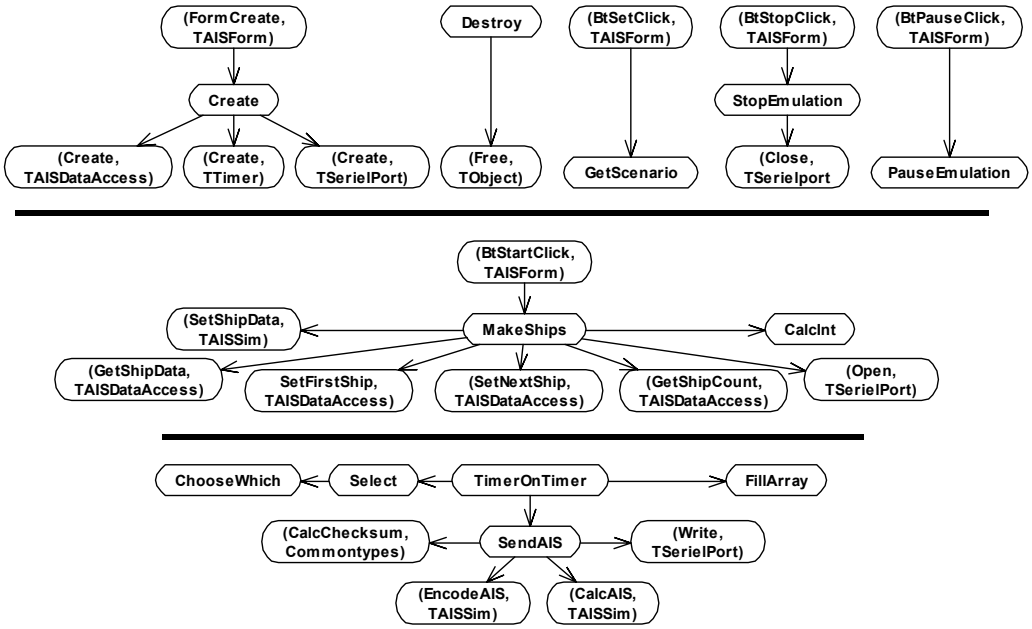
Denne metode aktiveres af serielportvælgeren og informerer under metoden MakeShips, i klassen TAISAdm, klassen TSerielPort om hvilken serielport der skal benyttes.

FormClose

Metoden FormClose aktiveres når AIS-simulatoren afsluttes, og har som eneste formål, at oprette et objekt hvori den valgte databasesti lægges i en ini fil. Herefter nedlægges objektet vha. metoden Free, i klassen TObject.

9.2.2 TAISAdm

Klassen TAISAdm indeholder de metoder, i AIS-simulatoren, der administrerer timingen og styringen af AIS-simulationen. Sammenhængen mellem disse metoder ses i figur 9.19



Figur 9.19: Metodesammenhæng for klassen TAISAdm

Til kommunikation klasserne imellem, er properties'ene i tabel 9.2 indeholdt i TGPSDataAccess:

Navn	Datatype	Læser fra	Skriver til
AISPort	Byte	-	FAISPort
AISDataPath	String	-	SetAISDataPath

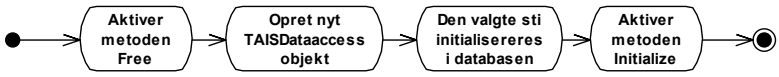
Tabel 9.2: Properties indeholdt i klassen TAISAdm

Create

Metoden Create opretter objekter af klasserne TAISDataAccess, TTimer og TSerielport. Dernæst sætter metoden parametrene for objekterne i de to sidstnævnte klasser.

Destroy
Benyttes til at nedlægge de objekter der blev oprettet i metoden Create, i klassen TAISAdm. Dette gøres vha. metoden Free, i klassen TObject.

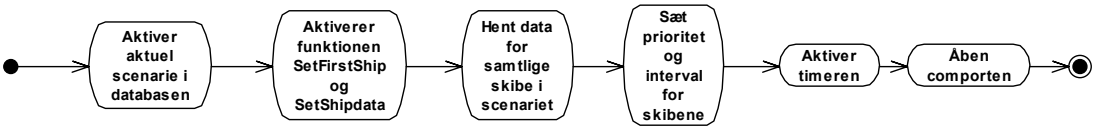
SetAISDatapath



Figur 9.20: Aktivitetsdiagram over metoden SetAISDatapath

SetAISDatapath (se figur 9.20) aktiveres af metoden BtSetClick, i klassen TAISForm, igennem property'en AISDataPath. Først aktiveres metoden Free, beliggende i klassen TObject. Metoden benyttes til at nedlægge database objektet DBAIS af klassen TAISDataAccess. Dernæst oprettes dette objekt, og dermed databasen igen. Dette gøres via metoden Create, beliggende i klassen TAISDataAccess.
Den valgte sti associeres nu, i klassen TAISDataAccess, med objektet DBAIS, således der nu er adgang til den valgte database.
Endeligt aktiveres metoden Initialize, igen beliggende i klassen TAISDataAccess, der initialiserer objektet DBAIS.

MakeShips



Figur 9.21: Aktivitetsdiagram over metoden Makeships

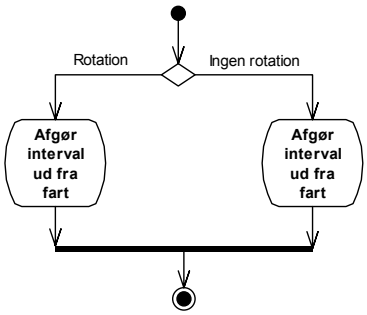
Metoden Makeships (se figur 9.21) aktiveres når superbrugeren, vha. startknappen, har valgt at opstarte en AIS-simulation.
Makeship starter med at aktivere, det i scenarievælgeren, valgte scenarie. Dette gøres ved at associere navnet på det valgte scenarie, med det tilsvarende scenarie i databasen. Dernæst gøres data for det første skib klar ved at aktivere metoden SetFirstShip, i klassen TAISDataAccess. Der oprettes nu et objekt, Ship[0], tilknyttet det første skib og data lægges i dette objekt. Efterfølgende aktiveres gemmes informationer for det enkelte skib, indhentet fra databasen. Dette sker ved at skrive til property'en ShipData, der henviser til SetShipData, i det aktuelle objekt, Ship[0]. Dette er nødvendigt, idet disse informationer kun indhentes én gang fra en record.
Efter data for første skib er overført, aktiveres metoden SetNextShip, også i klassen TAISDataAccess, der klargør data for næste skib i scenariet. Disse data lægges tilsvarende i et nyt objekt, Ship[1]. Dette gentages indtil alle skibe i scenariet har fået oprettet sit eget objekt, Ship[x], med tilhørende data. Alle objekter er placeret i arrayet Skib, der således er et array af objekter.
Sideløbende med dette lægges i samtlige objekter en værdi, beregnet ud fra det enkelte skibs fart og ændring i kurs, der indikerer med hvilket interval det enkelte skib skal udsende pakke 1, 2 eller 3 AIS-informationer. Dette interval bestemmes vha. funktionen CalcInt placeret i TAISAdm.
Intervallet for udsendelse af pakke 5 AIS-informationer er fast på 360s [AIS s. 3].
Jævnfør standarden[AIS] må det interval, indenfor hvilket en bestemt pakke skal være afsendt, ikke variere med mere end 10 procent af intervallet. For at opfylde dette krav, er det fundet nødvendigt at udstyre det enkelte skib med en varierende prioritet, således det er muligt at bestemme hvilket skib der skal afsende en pakke. Til dette oprettes en række arrays:

- *Countpak1 og Countpak5:* Disse to arrays indeholder et tal for hvert skib, der svarer til hvor lang tid siden det enkelte skib sidst afsendte en pakke 1,2,3 eller pakke 5. Dette tal stiger med værdien 1 for hvert timerinterval, og nulstilles når pakken afsendes.

- *Priopak1 og Priopak5:* Disse arrays indeholder prioriteten for det enkelte skib med hensyn til den enkelte pakketype. Prioriteten starter på det interval, der er mellem to ens pakketyper der afsendes, plus 10 procent. Prioriteten falder med værdien 1 for hvert timerinterval, og resettes når pakken afsendes. En pakke har højest prioritet når denne er faldet ned til værdien 1.
- *Intpak1:* Dette array indeholder det enkelte skibs interval for en pakke 1,2 eller 3 afsendelse, målt i antal timerintervaller. Pakke 5 vil altid have samme interval.

Når disse arrays, og objekter er sat op, startes timeren Timer i klassen TTimer, og den valgte serielport aktiveres via metoden Open, beliggende i klassen TSerielport.

CalcInt



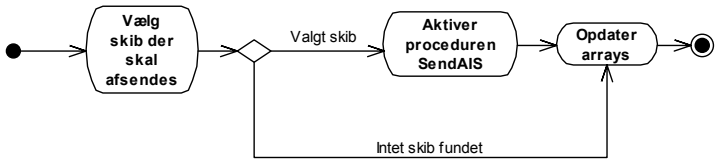
Figur 9.22: Aktivitetsdiagram over metoden CalcInt

Metoden CalcInt (se figur 9.22) benyttes af metoden Makeships til at beregne antallet af timerintervaller, der skal være mellem afsendelse af en pakke 1,2 eller 3.
Det afgøres først hvorvidt det aktuelle skib er ved at foretage en kursændring eller ej. Herefter findes intervallet ud fra skibets fart. Sammenhængen mellem interval, kursændring og fart, for en pakke 1,2 eller 3, kan ses i tabel 9.3.

Situation	Interval
Ligger for anker	3 min.
Sejler 0-14 knob	12 sek.
Sejler 0-14 knob og ændrer kurs	4 sek.
Sejler 14-23 knob	6 sek.
Sejler 14-23 knob og ændrer kurs	2 sek.
Sejler over 23 knob	3 sek.
Sejler over 23 knob og ændrer kurs	2 sek.

Tabel 9.3: Interval for afsendelse af pakke 1,2 og 3[AIS s. 3]

TimerOnTimer



Figur 9.23: Aktivitetsdiagram over metoden TimerOnTimer

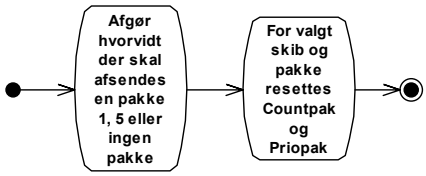
Metoden TimerOnTimer (se figur 9.23) aktiveres efter hvert timerinterval.
Det skal først afgøres hvilket skib skal sende hvilken pakke, hvis en pakke overhovedet skal afsendes efter det aktuelle timerinterval. Denne udvælgelse foregår ved at aktivere metoden Select, også placeret i TAISAdm.

Returnerer Select en værdi der indikerer at ingen pakke skal afsendes, i dette timerinterval, skal værdierne i arrayene Countpak1 og 5 hæves med 1, medens værdierne i arrayene Priopak1 og 5 skal sænkes med 1. Denne opdatering af arrayene sker ved at aktivere metoden FillArray, ligeledes placeret i TAISAdm.
Returnerer Select derimod en værdi, der indikerer en specifik pakke der skal afsendes, aktiveres metoden SendAIS.
Herefter opdateres arrayene med metoden FillArray.

FillArray

Metoden FillArray aktiveres af TimerOnTimer, og har til formål at opdatere arrayene Countpak1 og 5 samt Pripak1 og 5. Dette gøres ved at hæve værdierne i arrayene Countpak1 og 5 med værdien 1, medens værdierne i arrayene Priopak1 og 5 sænkes med 1.

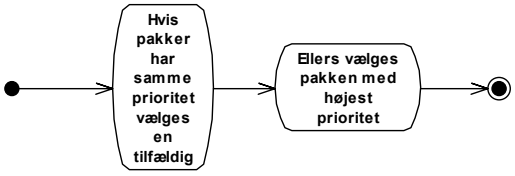
Select



Figur 9.24: Aktivitetsdiagram over metoden Select

Denne metode aktiveres af metoden TimerOnTimer, og har til formål at vælge hvilket skib og hvilken pakke der skal afsendes, i det pågældende timerinterval, hvis en pakke overhovedet skal afsendes (se figur 9.24).
Først undersøges hvorvidt et skib skal afsende en pakke 1,2 eller 3, hvilket gøres ved at sammenligne værdierne af arrayene Countpak1 og Intpak1, for de enkelte skibe. Er værdien af Countpak1 større end værdien af Intpak1, skal pakke 1,2 eller 3 fra dette skib afsendes. Findes mere end ét skib, der skal afsende en pakke 1, sammenlignes prioriteterne for disse pakker. Dette gøres vha. metoden ChooseWhich, fra klassen TAISAdm, der vælger den pakke med højest prioritet.
Dette gøres tilsvarende for alle skibe med hensyn til pakke 5. Da metoden ChooseWhich er i stand til at sammenligne prioriteterne på både pakke 1,2,3 og 5, vil pakken med højest prioritet nu være valgt, uanset pakketype.
For valgt skib og pakke resettes efterfølgende Countpak og priopak.

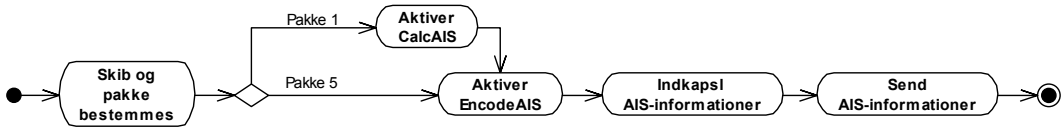
ChooseWhich



Figur 9.25: Aktivitetsdiagram over metoden ChooseWhich

Metoden ChooseWhich (se figur 9.25) aktiveres af metoden Select, og har til formål at vælge hvilken af to allerede specificerede pakker, der har højest prioritet. Metoden er i stand til at prioritere pakker fra samme eller forskellige skibe, og pakker af forskellige typer.
Først undersøges det hvorvidt de specificerede pakker har samme prioritet, i hvilket tilfælde metoden vælger en tilfældig pakke. Har pakkerne forskellig prioritet vælges den pakke med højest prioritet.

SendAIS



Figur 9.26: Aktivitetsdiagram over metoden SendAIS

SendAIS (se figur 9.26) aktiveres af metoden TimerOnTimer, og har til formål at afsende AIS-informationer til terminalen gennem den valgte serielport.
Metoden Select har returneret en integerværdi der fortæller SendAIS, hvilket skib og hvilken pakke, der skal afsendes.
I det tilfælde at den specificerede pakke er af typen 1,2 eller 3 skal metoden CalcAIS, i klassen AISSim, aktiveres. Denne metode bestemmer skibets position.
Det er ikke nødvendigt at benytte metoden CalcAIS såfremt en pakke 5 skal afsendes, idet pakke 5 udelukkende består af statisk data.
Herefter aktiveres metoden EncodeAIS, i klassen AISSim, der pakker AIS-informationerne i datadele af maksimalt 62 bit, i henhold til NMEA 0183[AIS 2001 s. 100]. Efterfølgende indkapsles de pakkede AIS-informationer, med start- og slutsekvens igen i henhold til NMEA 0183. I slutsekvensen indgår bl.a. en checksum, der beregnes ved hjælp af metoden CalcChecksum, beliggende i unit'en Commontypes.
Afslutningsvis aktiveres metoden Write, i klassen TSerielPort, der bevirker, at AIS-informationerne sendes til terminalen via den valgte serielport.

GetScenario

Metoden GetScenario aktiveres af metoden BtSetClick og FormCreate, beliggende i klassen TAISForm. Den har til formål at indlæse scenarie titler i scenarievælgeren. Dette sker ved at skabe en forbindelse mellem Scenarievælgeren og objektet DBAIS af klassen TAISDataAccess.

StopEmulation

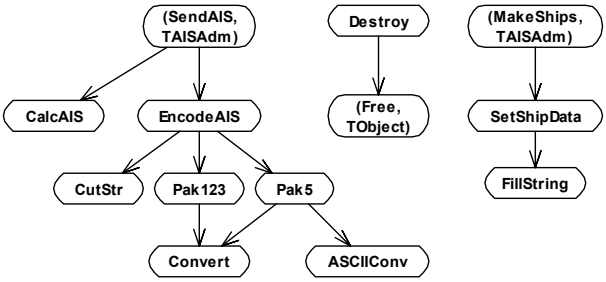
Metoden StopEmulation aktiveres af metoden BtStopClick, beliggende i klassen TAISForm. Metoden har to formål, for det første at deaktivere timeren, og for det næste at lukke serielporten vha. klassen TSerielport.

PauseEmulation

Metoden PauseEmulation aktiveres af metoden BtPauseClick, beliggende i klassen TAISForm. Kører AIS-simulationen som normalt skal timeren stoppes, hvorimod den skal startes i det tilfælde at AIS-simulationen allerede *er* pauset.

9.2.3 TAISSim

Klassen TAISSIM indeholder de metoder, i AIS-simulatoren, der omhandler kodning og pakning af AIS-informationerne. Figur 9.27 illustrerer sammenhængen mellem disse metoder.



Figur 9.27: Metodesammenhæng for klassen TAISSim

Til kommunikation klasserne imellem, er properties'ene i tabel 9.4 indeholdt i TAISSim:

Navn	Datatype	Læser fra	Skriver til
Interval	Integer	FAISInterval	FAISInterval
ShipData	TAISShipData	-	SetShipData
Speed	Integer	FSpeed	-
Rotation	Shortint	FRotation	-

Tabel 9.4: Properties indeholdt i klassen TAISSim

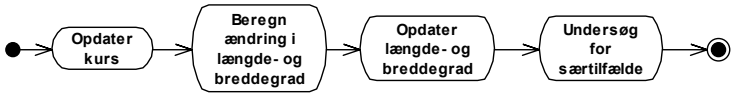
SetShipData

Metoden SetShipdata aktiveres af metoden MakeShips, i klassen TAISAdm, igennem property'en ShipData. Metoden har til formål at indlæse AIS-informationer fra databasen, pakket i en record, ind i variable i det korrekte skibsobjekt. Endvidere sørger SetShipData for at omdanne evt. lowercase bogstaver til uppercase.

FillString

Metoden FillString kaldes af metoden SetShipData, i det tilfælde, at antallet af bogstaver ikke udfylder den plads der er til rådighed. FillString sørger så for at fylde den overskydende plads med tomme pladser.

CalcAIS



Figur 9.28: Aktivitetsdiagram over metoden CalcAIS

Metoden CalcAIS (Se figur 9.28) aktiveres af metoden SendAIS, i klassen TAISAdm. Formålet med CalcAIS er at beregne et skibs position, ud fra et sæt startbetingelser og timeren Timer´s (fra TAISAdm) timingsinterval.
Først beregner metoden CalcAIS en ny kurs ud fra ligning 9.4

$$\text{Kurs} = \text{Kurs}_{\text{opr}} + \left(\frac{\text{Rotation} \cdot \text{Timinginterval}}{6000} \right) \qquad (9.4)$$

Den opdaterede kurs findes ved at addere den oprindelige kurs, der er angivet i databasen, med et led givet ved rotationen og den tid der er gået siden sidste opdatering. Dette led findes ved at gange rotationen sammen med timingintervallet og dividere dette med 6000. Der divideres med 6000 da rotationen er givet i tiendedele grader pr. minut og timingintervallet er givet i millisekunder. Efterfølgende tages der højde for det tilfælde, at kursen ikke ligger i intervallet 0-360 grader. Ændringen i længden per timinginterval findes ud fra ligning 9.5

$$\Delta\text{Longitude} = \frac{\text{Fart} \cdot \text{Timinginterval} \cdot \cos\left(\frac{\text{Latitude}}{600000} \right) \cdot \cos(\text{kurs})}{3600} \qquad (9.5)$$

I ligning 9.5 tages der hensyn til hvilken bredde eget skib befinder sig på, samt at startpositionen er givet i titusindedele minutter. Endvidere tages der hensyn til at farten er givet i tiendedele knob, og positionen skal angives i titusindedele minutter per timinginterval, hvorfor der divideres med 3600. Ændringen i bredden per timinginterval findes ud fra ligning 9.6:

$$\Delta\text{Latitude} = \frac{\text{Fart} \cdot \text{Timinginterval} \cdot \sin(\text{kurs})}{3600} \qquad (9.6)$$

Efterfølgende lægges resultaterne af ligning 9.5 og 9.6 til positionen inden den aktuelle opdatering. Endelig undersøges hvorvidt eget skib har krydset datolinien. Dette udføres ved at undersøge, om længden er numerisk større end 180º, og korrigere for dette.

EncodeAIS

Metoden EncodeAIS aktiveres af metoden SendAIS, fra klassen TAISAdm, og har til formål at pakke AIS-informationerne i datadele af maksimalt 62 tegn, i henhold til NMEA 0183[AIS 2001 s. 100]. Metoden modtager den pakketype, der skal sendes. Afhængigt af dette kaldes enten Pak123 eller Pak5. Herefter kaldes CutStr med resultatet af dette som parameter.

CutStr

CutStr aktiveres af metoden EncodeAIS, og klipper strengen af AIS-informationer i x antal stykker af 62 tegn, alt efter størrelsen af den pakke der skal sendes.

Pak123

Metoden Pak123 kaldes af EncodeAIS og pakker AIS-informationer for en pakke 1,2 eller 3 såfremt en sådan skal afsendes.
Dette gøres ved at tage AIS-informationerne og pakke disse i 6 bit stykker, i arrayet p123. AIS-informationerne pakkes som illustreret i tabel 9.5.

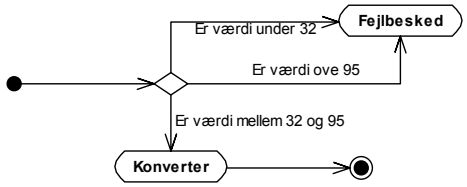
	Bit
1: Message ID,	
2: Data terminal equipment, Data indicator, MMSI	
3:	
4:	
5:	
6:	
7: Navigational status, Rate of turn	
8:	
9: Speed over ground	
10: Position accuracy, Longitude	
11:	
12:	
13:	
14:	
15: Latitude	
16:	
17:	
18:	
19:	
20: Course over ground	
21:	
22: Heading	
23: Time stamp	
24: Spare	
25:	
26: Communication state	
27:	
28:	

Tabel 9.5: Tabeloversigt over hvorledes bittene pakkes

Formålet med at pakke AIS-informationerne på denne måde er, at disse skal overføres til terminalen vha. den tegnbaseret kommunikationsprotokol RS-232.

Grunden til at AIS-informationerne ikke umiddelbart pakkes i 8 bit stykker repræsenteret ved et 8 bit ASCII-tegn er, at dette vil kunne medføre fejl når AIS-informationerne dekodes i terminalen. Dette vil eksempelvis være tilfældet, hvis AIS-informationer repræsenteres ved et specialtegn, i intervallet 0-31.
Efter at AIS-informationerne er pakket i 6 bit stykker, repræsenteres disse ved ét af 64 8 bit ASCII tegn, i intervallet 32-95. Denne representation foretages via metoden Convert.
Metoden Pak5 anvendes til en pakke 5 på samme måde som Pak123 anvendes til pakke 1,2 og 3.
I Pak5 benyttes endvidere metoden ASCIIConv til at konvertere AIS-informationer givet ved store bogstaver i 8 bit ASCII-værdier til 6 bit ASCII, inden disse pakkes i 6 bit stykker.

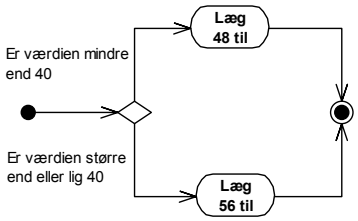
ASCIIConv



Figur 9.29: Aktivitetsdiagram over metoden ASCIIConv

Metoden ASCIIConv (se figur 9.29) kaldes af Pak5 og benyttes til at konvertere bogstaver givet ved 8 bit ASCII-værdier til 6 bit ASCII-værdier.
Det bestemmes hvorvidt 8 bit ASCII-værdien for det enkelte bogstav ligger i intervallet 32-95 eller ej. Er dette ikke tilfældet meldes en fejlbesked. Ligger værdien derimod i det tilladte interval konverteres bogstavet til 6 bit ASCII.
Begrundelsen for at værdien skal ligge i intervallet 32-95 er, at de tegn der skal kunne benyttes alle ligger i dette interval.

Convert



Figur 9.30:Aktivitetsdiagram over metoden Convert

Convert (se figur 9.30) kaldes af metoderne Pak123 eller Pak5 alt efter hvilken pakke der skal afsendes.
Eneste formål med dennne metode er at konvertere AIS-informationerne, der nu er pakket i 6 bit stykker til tilsvarende 8 bit ASCII-værdier, i henhold til NMEA standarden [AIS 2001 s. 111].

9.2.4 TAISDataAccess

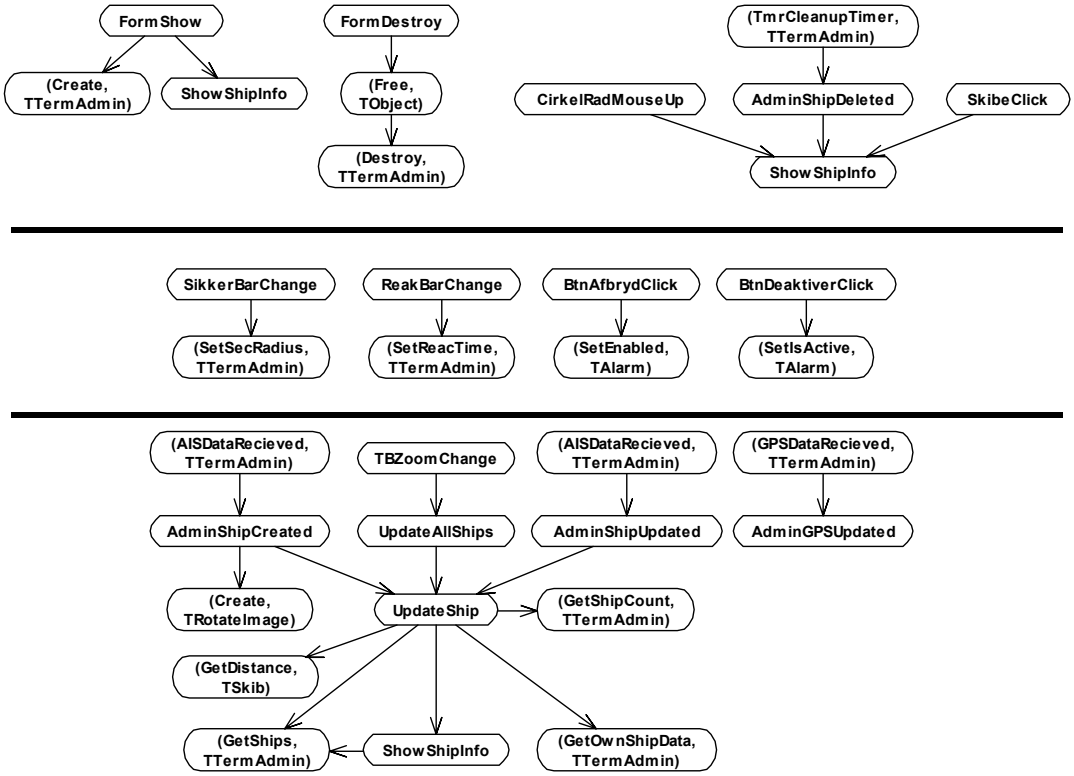
Denne klasse giver adgang til databasen. Den er beskrevet nærmere under afsnit 10.2.2.

9.3 Terminal

Terminalen består, som nævnt i afsnit 8.3, af klasserne TFrmTerminal, TTermAdmin, TGPS, TAIS, TSkib, TAlarm, samt fællesklasserne TserielPort og TTimer. Samtlige metoder i de førstnævnte klasser, vil blive gennemgået i det følgende. Metoderne i klassen TSerielPort beskrives i afsnit 9.4.

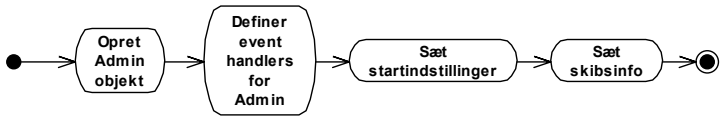
9.3.1 TFrmTerminal

Klassen TFrmTerminal gør det muligt for brugeren at benytte kollisionsdetektoren, med dennes funktionaliter. TFrmTerminal udgør således kollisionsdetektorens GUI, hvilket er uddybet i afsnit 11.
Inden beskrivelse af de enkelte metoder i klassen TFrmTerminal er det valgt at illustrere sammenhængen mellem de enkelte metoder, hvilket ses i figur 9.31.



Figur 9.31: Metodesammenhæng for klassen TFrmTerminal

FormShow

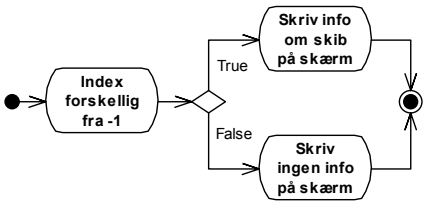


Figur 9.32: Aktivitetsdiagram over metoden FormShow

Denne metode (illustreret i figur 9.32) kaldes ved opstart af programmet, efter alle forme er oprettet. Det første der sker er, at objektet Admin af klassen TTermAdmin oprettes. Der defineres endvidere eventhandlers for events fra Admin. Disse events er OnGPSUpdated, OnShipCreated, OnShipUpdated og OnShipDeleted, og de håndteres af metoderne AdminGPSUpdated, AdminShipCreated, AdminShipUpdated og AdminShipDeleted, beliggende i klassen TFrmTerminal.
Startindstillingerne sættes herefter. Sikkerhedsafstanden sættes til 50 m, reaktionstiden sættes til 5 minutter (300 sekunder) og zoom sættes til 25 sømil.
Endeligt sættes indholdet af boksen ”Skibsinformationer” via metoden ShowShipInfo.

FormDestroy
Når TFrmTerminal lukkes, vil denne metoden FormDestroy fjerne objektet Admin, således allokeret hukommelse frigøres. Dettes sker ved at kalde at kalde metoden Free, beliggende i klassen TObject, der igen kalder metoden Destroy, beliggende i klassen TTermAdmin.

ShowShipInfo



Figur 9.33: Aktivitetsdiagram for metoden ShowShipInfo

Metoden ShowShipInfo (se figur 9.33) kan aktiveres af følgende fire metoder: CirkelRadMouseUp, AdminShipDeleted, SkibeClick og UpdateShip, alle beliggende i klassen TfrmTerminal. ShowShipInfo undersøger, hvorvidt der er valgt et skib til visning, i boksen ”Skibsinformationer”, ved at kontrollere indekset benævnt ShipInfoIndex. I det tilfælde, at indekset er forskelligt fra -1, vil et skib være valgt. Information for skibets indeks vil herefter blive vist i boksen ”Skibsinformationer”. Dette sker ved, at ShowShipInfo kalder metoden GetShips, beliggende i klassen TTermAdmin, igennem property’en Ships, ligeledes beliggende i klassen TTermAdmin. GetShips returnerer et det valgte objekt af klassen TSkib, hvori ShowShipInfo kan læse data og formidle disse videre til boksen ”Skibsinformationer”.
Er indekset lig -1, vil evt. skibsinformationer i boksen blive slettet.

CirkelRadMouseUp
Denne metode kaldes når brugeren klikker på den figur, der hedder CirkelRad, også kaldet skærmcirklen. Når dette sker, skal der, i boksen ”Skibsinformationer”, ikke vises information for noget skib. Dette opnås ved at sætte ShipInfoIndex til -1 og kalde metoden ShowShipInfo.

AdminShipDeleted
Metoden AdminShipDeleted aktiveres af metoden TmrCleanupTimer, beliggende i klassen TTermAdmin, igennem property’en OnShipDeleted. I dette tilfælde skal AdminShipDeleted slette det tilsvarende skib på skærmen.
Hvis skibets information bliver vist på skærmen, vil dette blive slettet ved at sætte ShipInfoIndex til -1 og kalde ShowShipInfo.

SkibeClick
Når der klikkes på et af de skibe, der vises på skærmen vil denne metode blive kaldt. Metoden starter med at undersøge hvilket skib, der er klikket på. Dette gøres ved at sammenligne det objekt, der modtages, som parameter til metoden, med alle de objekter, der findes på skærmen. Herefter sættes ShipInfoIndex lig indeks for det fundne skib, og metoden ShowShipInfo kaldes.

SikkerBarChange
Denne metode kaldes, når trackbaren SikkerBar, ændres af brugeren, der således ønsker at ændre sin sikkerhedsafstand. Metoden sætter den valgte sikkerhedsafstand i objektet Admin, via metoden SetSecRadius, igennem property’en SecRadius, hvorefter SikkerBarChange skriver indstillingen ud på skærmen.

ReakBarChange
Denne metode kaldes, når ReakBar, ændres af brugeren, der således ønsker at ændre reaktionstiden. Metoden sætter den valgte reaktionstid i objektet Admin, via metoden SetReacTime, igennem

property'en ReacTime, hvorefter ReakBarChange regner indstillingen om til minutter og sekunder og skriver den ud på skærmen.

BtnAfbrydClick
Ved klik på knappen BtnAfbryd, vil denne metode blive kaldt, hvilket vil forårsage, at alarmen afbrydes, i tilfælde af at denne er aktiv. Dette sker ved at ativere metoden SetEnabled, beliggende i klassen TAlarm, igennem property'en Enabled, med parameteren false.

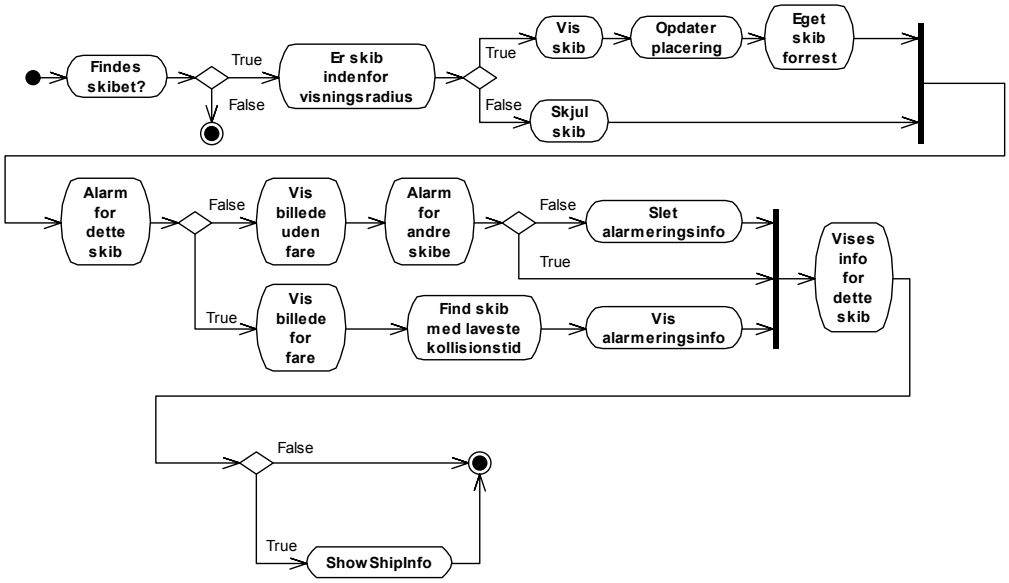
BtnDeaktiverClick
Ved klik på knappen BtnDeaktiver, vil denne metode blive kaldt, hvilket vil forårsage, at alarmen deaktiveres, i tilfælde af at denne er aktiveret, og aktiveres i tilfælde af at denne allerede er deaktiveret. Dette sker ved at kalde metoden SetIsActive, beliggende i klassen TAlarm, igennem property'en IsActive. Angives parameteren false deaktiveres alarmen, og angives parameteren true genaktiveres alarmen.

BtnAfslutClick
Ved klik på knappen BtnAfslut vil denne metode lukke programmet.

TBZoomChange
Denne metode kaldes når TBZoom, ændres af brugeren, der således ønsker at ændre zoom indstillingen. Metoden skriver den nye radius ud på skærmen og gemmer den i den tilhørende variabel. Herefter kaldes metoden UpdateAllShips.

UpdateAllShips
Denne metode kalder metoden UpdateShip for alle skibe.

UpdateShip



Figur 9.34: Aktivitetsdiagram for metoden UpdateShip

Denne metode (se figur 9.34) opdaterer visningen af det skib, der modtages som parameter. Først undersøges hvorvidt skibet er oprettet. Er dette ikke tilfældet, afsluttes. Er skibet oprettet, undersøges det hvorvidt skibet er indenfor visningsradius, via metoden GetDistance, beliggende i klassen TSkib, igennem property'en Distance. Er dette tilfældet vises skibet, og placeringen opdateres, ellers skjules skibet. Til dette benyttes metoden GetShips, beliggende i klassen TermAdmin, igennem property'en Ships.

Hvis alarmen er aktiveret for skibet, vil skibets billede blive sat til billedet for fare. Skibet med laveste kollisionstid findes, og alarmeringsinfo for dette skib vises på skærmen.
Hvis alarmen ikke er aktiveret for skibet, vil skibets billede blive sat til billedet uden fare. Hvis der heller ikke er andre skibe, der har aktiveret alarmen, vil alarmeringsinfo på skærmen blive fjernet.
Hvis der vises informationer for dette skib på skærmen, vil ShowShipInfo blive kaldt for at opdatere disse informationer.

AdminShipCreated

Metoden AdminShipCreated aktiveres af metoden AISDataRecieved, beliggende i klassen TTermAdmin, igennem property'en OnShipCreated. AdminShipCreated opretter et nyt objekt af klassen TRotateImage med samme index, som angivet i metoden AISDataRecieved. Der sættes default parametre for det nye objekt, og eventhandler'en OnClick associeres med metoden SkibeClick.
Endeligt aktiveres metoden UpdateShip, således skibets dynamiske parametre opdateres.

AdminShipUpdated

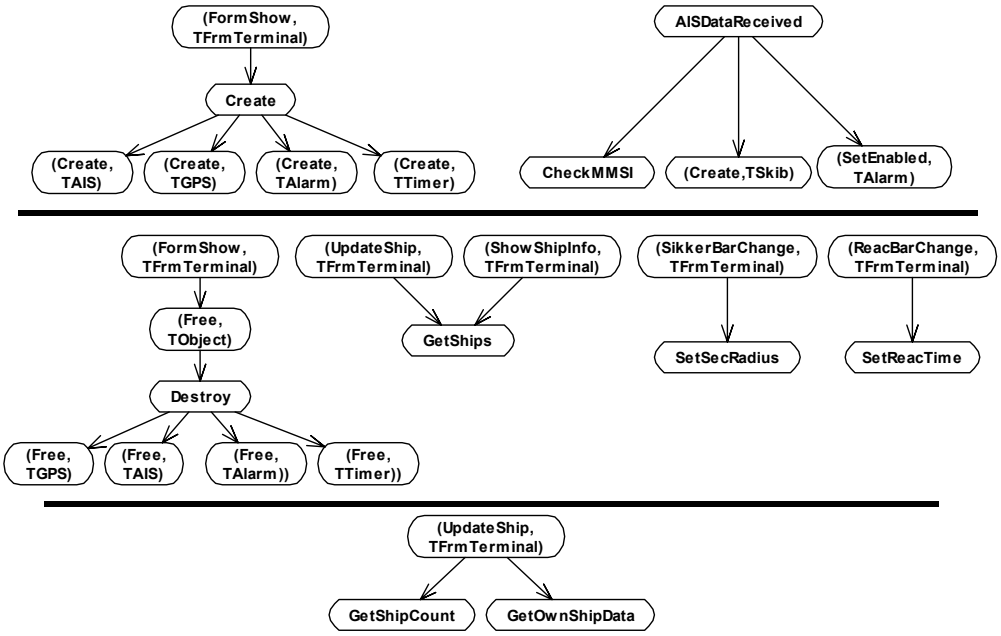
Metoden AdminShipUpdated aktiveres af metoden AISDataRecieved, beliggende i klassen TTermAdmin, igennem property'en OnShipUpdated. I dette tilfælde vil AdminShipUpdated kalde metoden UpdateShip, således informationerne for det aktuelle skib opdateres.

AdminGPSUpdated

Metoden AdminGPSUpdated aktiveres af metoden GPSDataRecieved, beliggende i klassen TTermAdmin, igennem property'en OnGPSUpdated
GPS data bliver skrevet op på skærmen, og placeringen af kompasnålen på skærmen opdateres.

9.3.2 TTermAdmin

Klassen TTermAdmin indeholder de metoder, i terminalen, der administrerer kommunikationen mellem de respektive klasser i Terminalen. Sammenhængen mellem disse metoder ses i figur 9.35



Figur 9.35: Metodesammenhæng for klassen TTermAdmin

Til kommunikation klasserne imellem, er properties'ene i tabel 9.6 indeholdt i TSkib:

Navn	Datatype	Læser fra	Skriver til
SecRadius	Integer	FSecRadius	SetSecRadius
ReacTime	Integer	FReacTime	SetReacTime
Ships[No: integer]	TSkib	GetShips	-
ShipCount	Integer	GetShipCount	-
Alarm	TAlarm	FAlarm	FAlarm
OwnShipData	TGPSData	SetOwnShipData	-
OnShipUpDated	TShipUpdatedEvent	FOnShipUpDated	FOnShipUpDated
OnShipCreated	TShipUpdatedEvent	FOnShipCreated	FOnShipCreated
OnShipDeleted	TShipUpdatedEvent	FOnShipDeleted	FOnShipDeleted
OnGPSUpdated	TGPSUpdatedEvent	FOnGPSUpdated	FOnGPSUpdated

Tabel 9.6: Properties indeholdt i klassen TTermAdmin

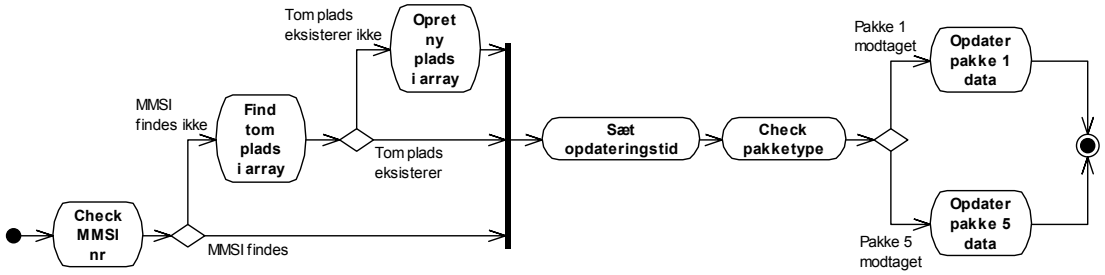
Create

Denne metode starter med at køre metoden Create for superklassen TComponent. Herefter oprettes objekter af klasserne TAIS og TGPS, der modtager data fra henholdsvis AIS og GPS modtager. OnDataReceived events for de 2 objekter defineres til henholdsvis AISDataReceived og GPSDataReceived. Objektet FAlarm af klassen TAlarm oprettes.
FMMSITable oprettes som en TStringList. Dette objekt indeholder en tabel over MMSI numre på de aktive skibe.
Timeren TmrCleanup oprettes og initialiseres. Metoden, der køres ved timer event, defineres til TmrCleanupTimer.
Til slut initialiseres anvendte variabler i klassen.

Destroy

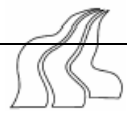
Objekterne AIS, GPS, FAlarm og TmrCleanup slettes, ved at kalde metoden Free for de respektive klasser. Herefter gennemgås array'et med skibe, hvor alle eksisterende skibe slettes. Endeligt slettes FMMSITable og destructoren for superklassen TComponent kaldes.

AISDataReceived



Figur 9.36: Aktivitetsdiagram over metoden AISDataReceived

Metoden AISDataRecieved (se figur 9.36) aktiveres af event'et OnDataRecieved genereret af objektet AIS af klassen TAIS. AISDataRecieved starter med at undersøge, hvorvidt det skib, der er modtaget som parameter, er oprettet som objekt i det globale array over skibe, benævnt FShips og værende af klassen TSkib. Dette sker ved at kalde metoden CheckMMSI. Er skibet ikke oprettet, undersøges hvorvidt der findes en tom plads i arrayet. Herefter indsættes skibet i arrayet på den tomme plads, eller hvis en sådan ikke eksisterer, vil skibet blive indsat sidst i arrayet. Skibet oprettes som et objekt af klassen TSkib, via metoden Create, beliggende i klassen TSkib. Findes skibet allerede i arrayet, opdateres de variable, der ifølge parameteren PakType er opdateret, samt



de variable, der definerer data for eget skib. Herefter aktiveres enten event'et OnShipUpdated, eller event'et OnShipCreated, alt afhængig af, om skibet er oprettet, eller opdateret.
Udgør skibet en kollisionsrisiko, vil funktionen IsCrashing, i objektet af klassen TSkib, returnere booleanen true. Dette vil medføre, at metoden SetEnabled, i klassen TAlarm, igennem property'en Enabled, aktiveres.
Afslutningsvis fjernes det skib, der er modtaget som parameter.

GPSDataReceived

Metoden GPSDataRecieved aktiveres af event'et OnDataRecieved genereret af objektet GPS af klassen TGPS. Metoden gemmer de data, der modtages som parameter, af typen TGPSData, i en lokal record, hvorefter event'et OnGPSUpdated aktiveres.

GetShips

Metoden GetShips aktiveres af metoderne UpdateShip eller ShowShipInfo, begge beliggende i klassen TFrmTerminal, igennem property'en Ships. Metoden modtager en parameter indeholdende et indeks, hvorefter den returnerer det skib i arrayet FShips, der har det tilsvarende indeks.

CheckMMSI

Metoden CheckMMSI aktiveres af metoden AISDataRecieved. Denne metode modtager et MMSI nr. som parameter, hvorefter den returnerer indeks for det tilhørende skib i arrayet. Findes skibet ikke returneres værdien -1.

SetSecRadius

SetSecRadius kaldes af metoden SikkerBarChange, beliggende i klassen TFrmTerminal, igennem property'en SecRadius. Metoden modtager en parameter indeholdende en sikkerhedsafstand i meter. Denne sikkerhedsafstand gemmes i den tilhørende globale variabel.

SetReacTime

SetReacTime kaldes af metoden ReacBarChange, beliggende i klassen TFrmTerminal, igennem property'en ReacTime. Metoden modtager reaktionstiden som parameter, og har til formål at gemme reaktionstiden i de tilhørende variable.

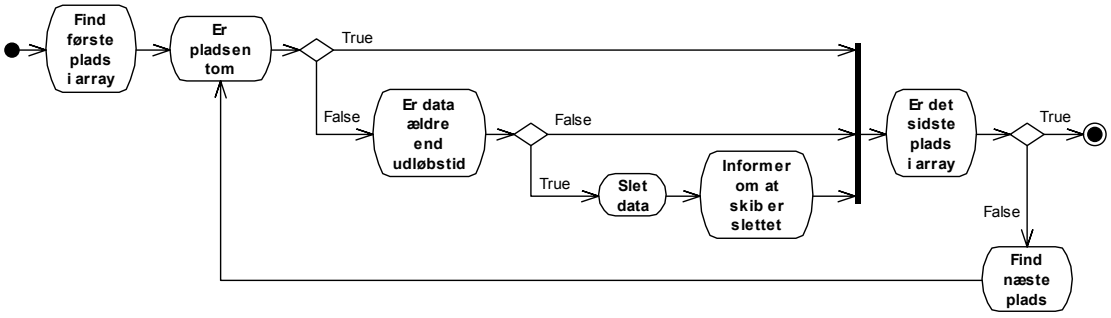
GetShipCount

GetShipCount aktiveres af metoden UpdateShip, beliggende i klassen TFrmTerminal, igennem property'en ShipCount. Metoden returnerer længden af det globale array FShips, der indeholder skibe.

GetOwnShipData

GetOwnShipData aktiveres af metoden UpdateShip, beliggende i klassen TFrmTerminal, igennem property'en OwnShipData. Denne metode returnerer data for eget skib, i form af en record. Et af felterne i denne record burde specificere UTC-tiden, her vil dog blot ligge computerens egen tid.

TmrCleanupTimer

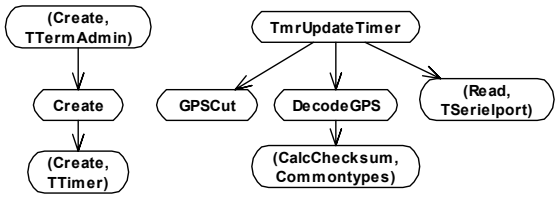


Figur 9.37: Aktivitetsdiagram over metoden TmrCleanupTimer

Denne metode kontrollerer, med faste intervaller, alle skibes sidste opdateringstidspunkt. Dette gøres for at kunne slette de skibe, der ikke har været opdateret indenfor en nærmere fastsat tid. Dette er nødvendigt, for at undgå at arrayet FShips fyldes med unødvendige data.
Metoden starter med at finde første plads i array'et, hvorefter det undersøges hvorvidt data forekommer data på denne plads. Er dette tilfældet, vil opdateringstidspunktet for skibet blive undersøgt, og er opdateringstidspunktet ældre end udløbstiden, vil disse data blive slettet. Ved sletning af data sendes event'et FOnShipDeleted med specifikation af indeks for slettet skib.
Er den aktuelle plads i array'et ikke er den sidste, undersøges næste plads. Dette gentages indtil samtlige pladser er blevet undersøgt.

9.3.3 TGPS

Klassen TGPS indeholder de metoder, der står for dekodning og implementering af de GPS-informationer terminalen modtager. Sammenhængen for disse ses på figur 9.38.



Figur 9.38: Metodesammenhæng for klassen TGPS

Til kommunikation klasserne imellem, er properties'ene i tabel 9.7 indeholdt i TGPS:

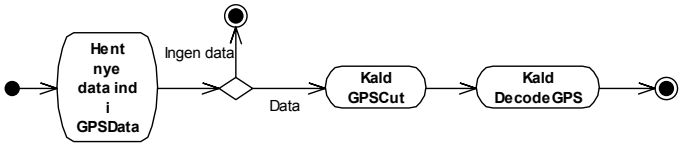
Navn	Datatype	Læser til	Skriver til
OnDataReceived	TGPSDataReceivedEvent	FOnDataReceived	FOnDataReceived

Tabel 9.7: Properties indeholdt i klassen TGPS

Create

Metoden Create aktiveres af metoden Create, beliggende i klassen TTermAdmin. Metoden starter med at køre metoden Create for superklassen TSerielport. Dernæst skal den oprette et objekt TmrUpdate af klassen TTimer. TmrUpdates indstillinger sættes til at opdatere hvert sekund, og ved timerevent skal metoden TmrUpdateTimer kaldes. Endeligt sættes baudrate til 4800 baud og serielporten sættes til COM 1, hvorefter serielporten åbnes via metoden Open, beliggende i klassen TSerielport.

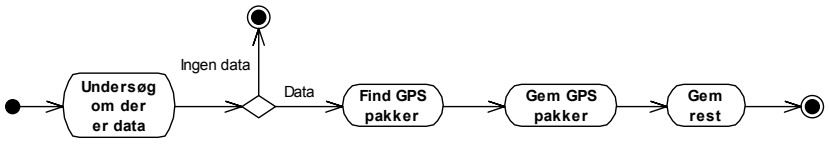
TmrUpdateTimer



Figur 9.39: Aktivitetsdiagram over metoden TmrUpdateTimer

Metoden TmrUpdateTimer (se figur 9.39) henter GPS-informationer via metoden Read, beliggende i klassen TSerielport. GPSCut aktiveres i det tilfælde, der er data tilstede. Efterfølgende kaldes metoden DecodeGPS, med parameteren sat til resultatet af metoden GPSCut.

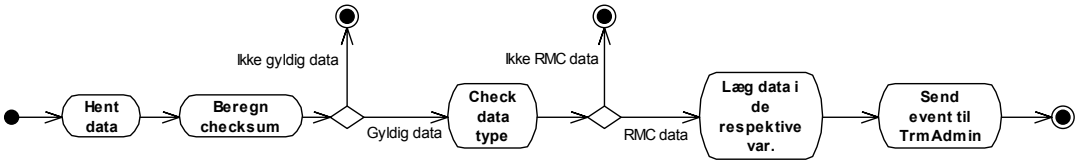
GPSCut



Figur 9.40: Aktivitetsdiagram over metoden GPSCut

GPSCut (se figur 9.40) aktiveres af metoden TmrUpdateTimer, og formålet med metoden GPSCut er at opdele data fra bufferen, i GPS pakker. Dette er nødvendigt, idet indholdet fra bufferen ikke nødvendigvis består af fuldstændige GPS pakker. Først kontrolleres det, hvorvidt der er data fra bufferen. Er dette tilfældet undersøger GPSCut hvorvidt dataene indeholder en startsekvens. Findes en startsekvens, indlæses dataene i en midlertidig string indtil slutsekvensen er fundet. Den midlertidige string gemmes i en stringlist. En evt. rest ved læsning af bufferen gemmes, til de nye data kommer ind.

DecodeGPS

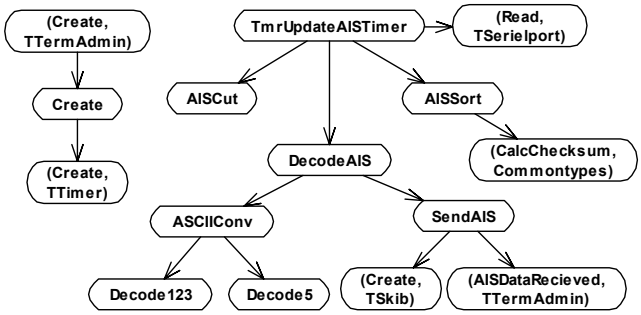


Figur 9.41: Aktivitetsdiagram over metoden DecodeGPS

Metoden DecodeGPS (se figur 9.41) aktiveres af metoden TmrUpdateTimer, og starter med at hente data fra den stringlist, den modtager som parameter. Herefter aktiveres metoden CalcChecksum, beliggende i unit'en Commontypes, for at kontrollere, at dataene er korrekte. Dernæst kontrolleres, hvorvidt det er den rigtige NMEA sætning, hvilket gøres ved at undersøge den første del af dataene, hvori det fremgår, hvilken sætning type dataene består af. Er det er den rigtige sætning, lægges dataene i de respektive variabler. Endvidere undersøges hvorvidt GPS'en er i fejltilstand. Er dette tilfældet skrives en fejlmeddelelse ud på den grafiske brugerflade. Når alle data er gemt aktiveres event'et, OnDataRecieved, der fortæller, at der foreligger nye data.

9.3.4 TAIS

Klassen TAIS indeholder de metoder, der står for dekodning og implementering af de AIS-informationer terminalen modtager. Sammenhængen ses i figur 9.42.



Figur 9.42: Metodesammenhæng for klassen TAIS

Til kommunikation klasserne imellem, er properties'ene i tabel 9.8 indeholdt i TAIS:

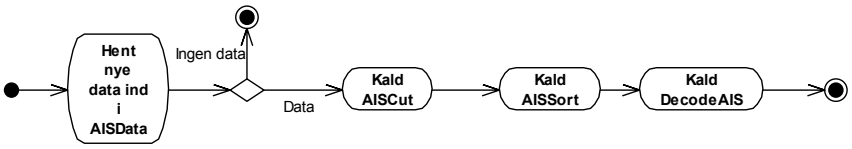
Navn	Datatype	Læser til	Skriver til
OnDataReceived	TAISDataReceivedEvent	FOnDataReceived	FOnDataReceived

Tabel 9.8: Properties indeholdt i klassen TAIS

Create

Metoden Create aktiveres af metoden Create, beliggende i klassen TTermAdmin, og starter med at køre metoden Create for superklassen TSerielport. Dernæst oprettes objektet TmrUpdateAIS af klassen TTimer. TmrUpdateAIS's indstillinger sættes til at opdatere hvert 1/10 sekund, og ved timerevent skal metoden TmrUpdateAISTimer kaldes. Endvidere sættes baudrate til 38.400 baud og serielporten sættes til COM 2. Herefter aktiveres metoden Open, beliggende i klassen TSerielport. Endeligt oprettes en stringlist, der benyttes til at gemme AIS data i, og eventuelt indhold bliver slettet.

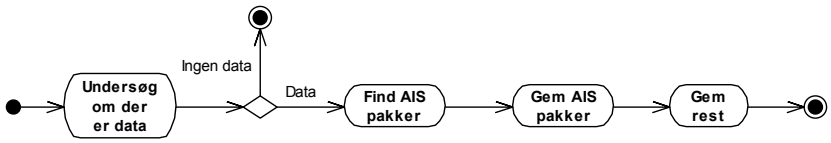
TmrUpdateAISTimer



Figur 9.43: Aktivitetsdiagram over metoden TmrUpdateAISTimer

Metoden TmrUpdateAISTimer (se figur 9.43) henter AIS-informationer vha. metoden Read, beliggende i klassen TSerielport. AISCut aktiveres, i det tilfælde, at der er data tilstede. Efterfølgende kaldes metoden AISSort, med parameteren sat til resultatet af metoden AISCut. Endeligt kaldes metoden DecodeAIS, med parameteren sat til resultatet af metoden AISSort.

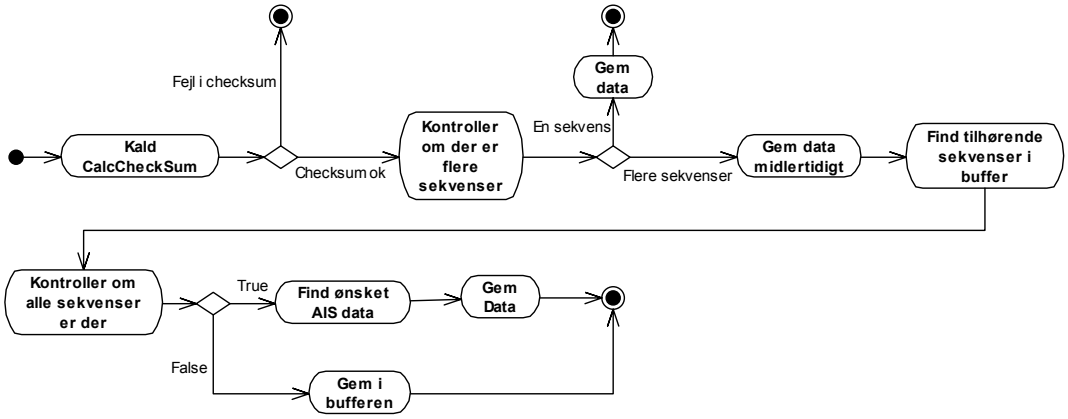
AISCut



Figur 9.44: Aktivitetsdiagram over metoden AISCut

Metoden AISCut (se figur 9.44) aktiveres af metoden TmrUpdateAISTimer. Formålet med denne metode er at opdele data fra bufferen i AIS pakker. Dette er nødvendigt, idet indholdet fra bufferen ikke nødvendigvis består af fuldstændige AIS pakker. Indledningsvis kontrolleres det, hvorvidt der er data fra bufferen. Er dette tilfældet undersøger AISCut hvorvidt dataene indeholder en startsekvens. Findes en startsekvens, indlæses dataene i en midlertidig string indtil slutsekvensen er fundet. Den midlertidige string gemmes i en stringlist. En evt. rest, ved læsning af bufferen, gemmes til nye data kommer ind.

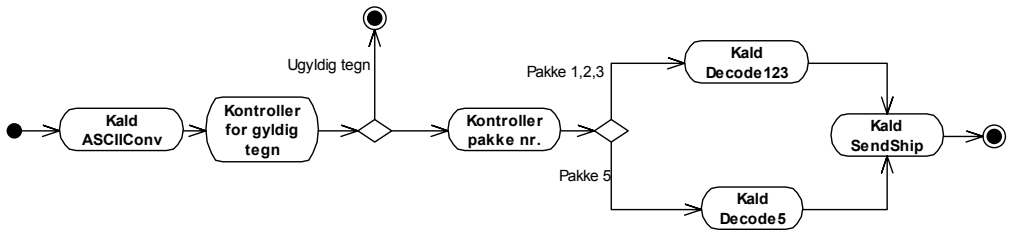
AISSort



Figur 9.45: Aktivitetsdiagram over metoden AISSort

Metoden AISSort (se figur 9.45) aktiveres af metoden TmrUpdateAISTimer, og har til formål at sortere de AIS sekvenser, metoden AISCut har frembragt.
AISSort starter med at aktivere metoden CalcChecksum, beliggende i unit'en Commontypes, hvorefter det kontrolleres hvor mange sekvenser den aktuelle sætning består af. Antallet af sætninger kan, i NMEA sætningen, læses af x-pladsen, i det nedenstående eksempel på en NMEA sætning jf. afsnit 2.3.
!AIVDM,x,y,z,a,s—s,f*hh<CR><LF>
Y-pladsen beskriver hvilket ID nummer den aktuelle sekvens har, og z er sætningens id nummer. I det tilfælde, at x og dermed ligeledes y er 1, skal sekvensen gemmes og metoden afsluttes. Er dette ikke tilfældet, skal sekvensen gemmes midlertidigt og id nummeret (z) sammenlignes med samtlige sekvenser beliggende i bufferen. Findes flere sekvenser af samme sætning, lægges disse til den aktuelle sekvens. Derefter undersøges, hvorvidt alle sekvenser er i sætningen. Er dette ikke tilfældet skal sætningen gemmes i bufferen, indtil alle sekvenser er til stede, hvorefter metoden starter igen og undersøger en ny sekvens.
Findes hele sætningen skal alle sekvenserne klippes sammen, hvilket sker ved at sekvenserne indlæses i en stringlist. Til sidst fjernes data fra buffer.

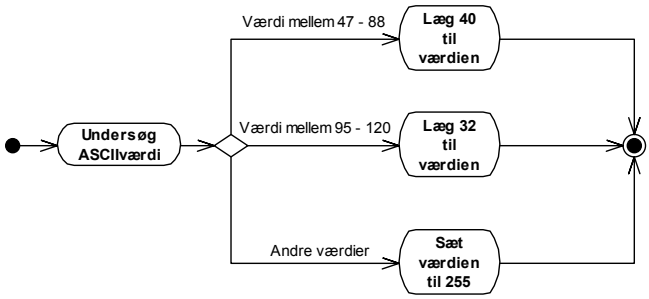
DecodeAIS



Figur 9.46: Aktivitetsdiagram over metoden DecodeAIS

Metoden DecodeAIS (se figur 9.46) aktiveres af metoden TmrUpdateAISTimer. Metoden kalder som det første metoden ASCIIConv, der kontrollerer hvorvidt dataene indeholder ugyldige værdier, hvorefter ASCII karakterne omskrives til binære data. Dernæst bestemmes hvilken type AISpakke de aktuelle data tilhører. Efterfølgende aktiveres en af metoderne Decode123 eller Decode5, der dekoder den aktuelle pakke. Afslutningsvis kalder DecodeAIS metoden SendShip med en tilhørende parameter, alt efter hvilken pakke der er dekodet.

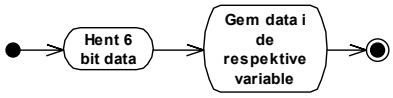
ASCIIConv



Figur 9.47: Aktivitetsdiagram over metoden ASCIIConv

Metoden ASCIIConv (se figur 9.47) aktiveres af metoden DecodeAIS, og har til formål at konvertere 8 bit ASCIIværdier til 6 bit værdier. Indledningsvis undersøges samtlige ASCIIværdier, og det kontrolleres hvorvidt disse ligger inden for de specificerede værdier. Er dette ikke tilfældet returneres værdien 255, idet denne værdi er valgt til fejlværdi.
Konverteringen foregår ved at hente 8 bit ASCIIværdien ind og bestemme hvilken decimalværdi karakteren repræsenterer. Er værdien mellem 47 og 88 lægges værdien 40 til, og er værdien mellem 95 og 120 lægges værdien 32 til. Dette gøres for at undgå, at karakterer, der i 6 bit koden ikke kan bruges, optager plads. Endeligt and'es den binære værdi med 111111 (63) for at opnå en 6 bit værdi.

Decode123



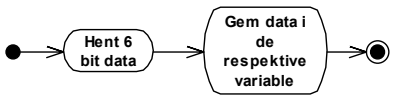
Figur 9.48: Aktivitetsdiagram over metoden Decode123

Metoden Decode123 (se figur 9.48) aktiveres af metoden DecodeAIS, og har til formål at udlede de relevante data fra de modtagne data, og skrive disse data i de dertil oprettede variable. De aktuelle variable er vist i tabel 9.9.

FMMSI
FNavState
FRotation
FSpeed
FLongitude
FLatitude
FCourse

Tabel 9.9: Variable benyttet af Decode123

Decode5



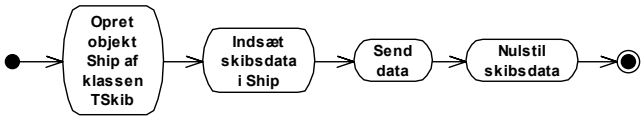
Figur 9.49: Aktivitetsdiagram over metoden Decode5

Metoden Decode5 (se figur 9.49) aktiveres af metoden DecodeAIS, og har, som Decode123, til formål at udlede de relevante data fra de modtagne data, og skrive disse data i de dertil oprettede variable. De aktuelle variable er vist i tabel 9.10.

FMMSI
FIMO
FCallSign
FName
FSCType
FShipRadius
FDrought
FDestination

Tabel 9.10: Variable benyttet af Decode5

SendShip

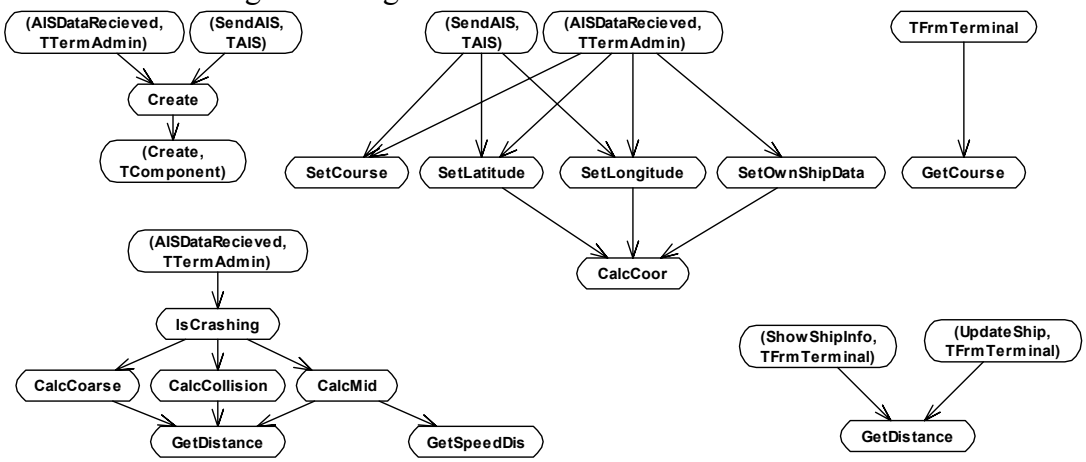


Figur 9.50: Aktivitetsdiagram over metoden SendShip

Metoden SendShip (se figur 9.50) aktiveres af metoden DecodeAIS. Denne metode benyttes til at informere terminalen om, at der er modtaget AIS-informationer.
Indledningsvis oprettes et objekt Ship af klassen TSkib. I dette objekt lægges de data, der i metoderne Decode123 og Decode5, er blevet klargjort. Objektet får endvidere vedhæftet en parameter, der oplyser hvilken datapakke Ship indeholder.
Endeligt aktiveres event'et TAISDataRecievedEvent, igennem property'en OnDataRecieved. Dette event aktiverer, i klassen TTermAdmin, metoden og eventhandler'en AISDataRecieved.
Afslutningsvis nulstilles variablerne i tabel 9.9 og 9.10.

9.3.5 TSkib

Klassen TSkib indeholder de metoder, der står for kollisionsberegninger og datalagring for det enkelte skib. Sammenhængen ses i figur 9.51.



Figur 9.51: Metodesammenhæng for klassen TSkib

Til kommunikation klasserne imellem, er properties'ene i tabel 9.11 indeholdt i TSkib:

Navn	Datatype	Læser fra	Skriver til
SpeedDis	Double	GetSpeedDis	-
Cathegory	Integer	FCathegory	FCathegory
CoorX	Double	FCoorX	-
CoorY	Double	FCoorY	-
Speed	Double	FSpeed	FSpeed
Course	Double	GetCourse	SetCourse
Latitude	Double	FLatitude	SetLatitude
Longitude	Double	FLongitude	SetLongitude
IMO	Longword	FIMO	FIMO
SCType	Byte	FSCType	FSCType
NavSType	Byte	FNavSType	FNavSType
Draught	Double	FDraught	FDraught
Destination	String	FDestination	FDestination
Rotation	Shortint	FRotation	FRotation
MMSI	Longword	FMMSI	FMMSI
Name	String	FName	FName
NavState	Integer	FNavState	FNavState
Shipradius	Double	FShipradius	FShipradius
SecRadius	Double	FSecRadius	FSecRadius
LastUpdated	TDateTime	FLastUpdated	FLastUpdated
CallSign	String	FCallSign	FCallSign
Distance	Double	GetDistance	-
OwnShipData	TOwnShipData	-	SetOwnShipData
Alarm	Boolean	FAlarm	FAlarm
TimeToCol	Integer	FTimeToCol	-

Tabel 9.11: Properties indeholdt i klassen TSkib

Create
Metoden Create aktiveres af enten metoden AISDataRecieved, beliggende i klassen TTermAdmin, eller metoden SendAIS, beliggende i klassen TAIS.
Metoden kalder constructoren for superklassen TComponent, hvorefter alle variable initialiseres.

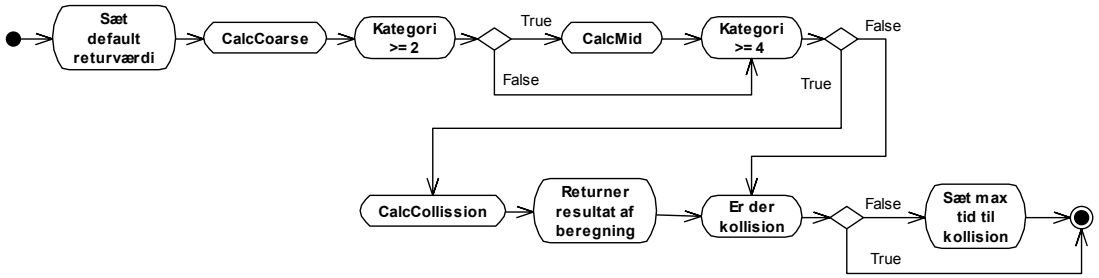
Destroy
Klassen TSkib opretter ingen objekter, der skal fjernes, så denne destructor kalder kun destructoren for superklassen TComponent.

SetLatitude
Metoden SetLatitude aktiveres af enten metoden AISDataRecieved, beliggende i klassen TTermAdmin, eller metoden SendAIS, beliggende i klassen TAIS, igennem property'en Latitude. Denne metode gemmer den bredde, der modtages som parameter, i den tilhørende globale variabel, hvorefter CalcCoor køres for at opdatere koordinater.

SetLongititude
Metoden SetLongitude aktiveres af enten metoden AISDataRecieved, beliggende i klassen TTermAdmin, eller metoden SendAIS, beliggende i klassen TAIS, igennem property'en Longitude. Denne metode gemmer den længde, der modtages som parameter, i den tilhørende globale variabel, hvorefter CalcCoor køres for at opdatere koordinater.

SetOwnShipData
Metoden SetLatitude aktiveres af metoden AISDataRecieved, beliggende i klassen TTermAdmin, igennem property'en OwnShipData.
Metoden gemmer data, modtaget som parameter, i de respektive variable. Et af felterne i record'en Value er kursen, og her modtages kursen i grader med geografisk reference. Denne kurs omregnes til matematiske grader, og omsættes til radianer.
Endeligt aktiveres metoden CalcCoor for at opdatere koordinater.

IsCrashing



Figur 9.52: Aktivitetsdiagram over metoden IsCrashing

Metoden IsCrashing (se figur 9.52) aktiveres af metoden AISDataRecieved, beliggende i klassen TTermAdmin.
Indledningsvis sætter metoden returværdien til false, således denne er defineret fra starten, hvorefter metoden CalcCoarse aktiveres. Er den resulterende kategori større end eller lig med 2, aktiveres metoden CalcMid.
Er kategorien efterfølgende større end eller lig med 4, aktiveres metoden CalcCollision, og returværdien for IsCrashing sættes lig returværdien for CalcCollision. Er kollision ikke en mulighed vil TimeToCol property'ens værdi blive sat til 1 større end reaktionstiden, således skibet ikke kan forvekles med skibe, der indebærer en kollisionsrisiko.

CalcCoarse
Metoden CalcCoarse aktiveres af metoden IsCrashing. Denne metode undersøger hvorvidt afstanden til skibet er under 25 sømil, via metoden GetDistance, igennem property'en Distance. Hvis dette er tilfældet sættes kategorien til 2, ellers 1.

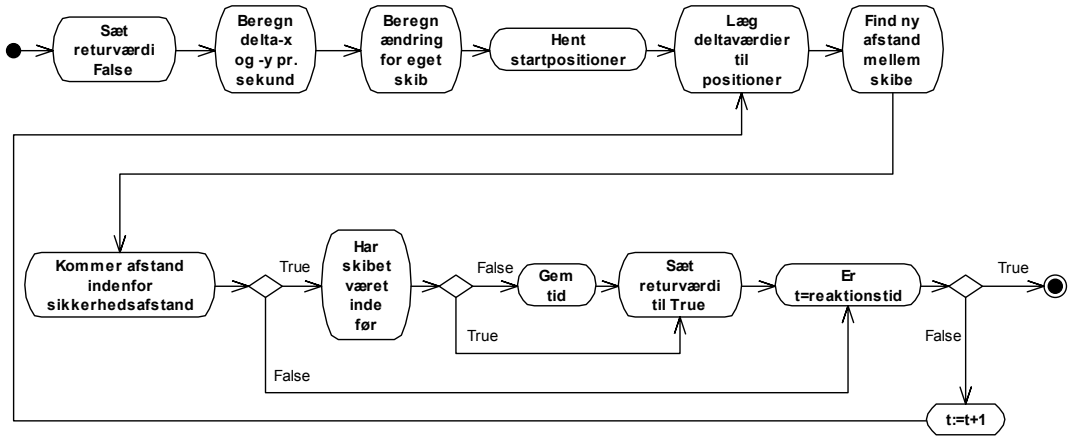
GetDistance
Metoden GetDistance kan aktiveres af en af følgende metoder: ShowShipInfo eller UpdateShip i klassen TFrmTerminal, eller CalcCourse, CalcMid eller CalcCollision.
Metoden GetDistance returnerer afstanden mellem eget skib og det aktuelle skib, udregnet efter Pythagoras sætning.

CalcMid
Metoden CalcCoarse aktiveres af metoden IsCrashing. Metoden starter med at undersøge, hvorvidt afstanden til det aktuelle skib kan tilbagelægges indenfor den fastsatte reaktionstid. Dette foretages vha. metoden GetDistance, og ved en anslået max hastighed, jf. afsnit 5.2.
Kan afstanden tilbagelægges indenfor reaktionstiden sættes kategorien til 3, ellers sættes kategorien til 2.
I det tilfælde, at kategorien sættes til 3, undersøges det endnu engang hvorvidt afstanden kan tilbagelægges indenfor reaktionstiden. Denne gang foretages udregningen med de aktuelle hastigheder, vha. metoden GetSpeedDis, igennem property'en SpeedDis. Kan afstanden tilbagelægges indenfor reaktionstiden sættes kategorien til 4, ellers sættes kategorien til 3.

GetSpeedDis

Metoden GetSpeedDis aktiveres af metoden CalcMid.
Metoden udregner den afstand de to skibe sammenlagt kan tilbagelægge indenfor reaktionstiden. Dertil lægges sikkerhedsafstand og skibsradius, og dette resultat returneres.

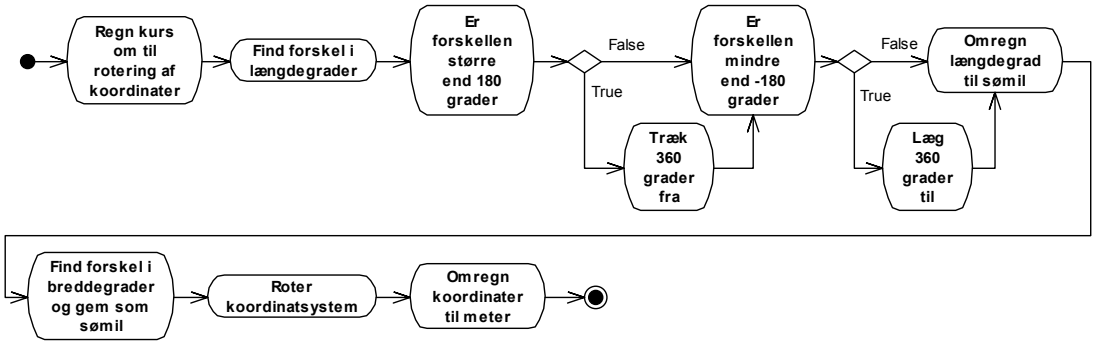
CalcCollision



Figur 9.53: Aktivitetsdiagram over metoden CalcCollision

Metoden CalcCollision (se figur 9.53) aktiveres af metoden IsCrashing. Denne metode sætter indledningsvis returværdien til false, således denne er defineret fra starten. Herefter beregnes ændringen i koordinaterne pr. sekund, deltaværdierne, for det aktuelle skib og for eget skib, hvorefter startpositionerne for de to skibe hentes.
I en løkke lægges deltaværdierne til positionerne for skibene, og afstanden mellem skibene beregnes. Er denne afstand indenfor sikkerhedsafstanden, vil returværdien blive sat til True. Har det aktuelle skib ikke været indenfor sikkerhedsafstanden tidligere, vil tiden blive gemt. Når løkken har kørt alle tider op til reaktionstiden, afsluttes.

CalcCoor



Figur 9.54: Aktivitetsdiagram over metoden CalcCoor

Metoden CalcCoor (se figur 9.54) aktiveres af metoderne SetLatitude, SetLongitude og SetOwnShipData.
Metoden omregner geografiske koordinater til et koordinatsystem med enheden meter, hvorefter koordinatsystemet roteres, på en sådan måde, at eget skib har kurs 0. Udregningen af rotationen af koordinaterne ses i ligning 9.7.

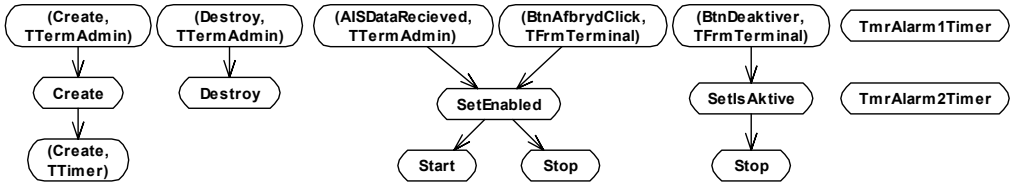
$$x1 = x \cdot \cos(v) + y \cdot \sin(v)$$
$$y1 = -x \cdot \sin(v) + y \cdot \cos(v) \qquad (9.7)$$

Først findes den vinkel, v, som koordinaterne skal roteres efter.
Dernæst findes forskellen i længderne, og i det tilfælde, at denne forskel er numerisk større end 180 grader, vil dette indikere, at skibet enten ligger på den anden side af datolinjen, eller sender fra den anden side af jorden, hvilket dog er meget usandsynligt. Er forskellen numerisk større end 180 grader, vil 360 grader enten blive lagt til eller trukket fra, således skibet vil få sammenlignelige koordinater. Disse koordinater vil være ugyldige, da længden numerisk vil komme over 180 grader, men matematisk har dette ingen betydning. Dette forbehold taget i betragtning, omregnes længden til sømil, idet der tages hensyn til den aktuelle bredde, hvorefter længde gemmes som x-koordinat. Efterfølgende findes forskellen i bredderne. Denne forskel vil umiddelbart svare til y-koordinaten målt i sømil.
Efter omregning til koordinater roteres koordinatsystemet, således eget skib placeres i kurs 0. Endeligt omregnes koordinaterne fra sømil til meter.

GetCourse
Metoden GetCourse aktiveres af en af metoderne ShowShipInfo eller UpdateShip, begge beliggende i klassen TFrmTerminal.
Metoden returnerer skibets kurs, efter at have omregnet denne fra matematiske radianer, til geografiske grader.

SetCourse
Metoden SetCourse aktiveres af enten metoden AISDataRecieved, beliggende i klassen TTermAdmin, eller metoden SendAIS, beliggende i klassen TAIS, igennem property'en Course.
Metoden sætter skibets kurs, efter at have omregnet denne fra geografiske grader, til matematiske radianer.

9.3.6 TAlarm
Klassen TAlarm indeholder de metoder, der står for alarmens funktionalitet. Sammenhængen mellem disse ses i figur 9.55.



Figur 9.55: Metodesammenhæng for klassen TAlarm

Til kommunikation klasserne imellem, er properties'ene i tabel 9.12 indeholdt i TSkib.

Navn	Datatype	Læser fra	Skriver til
IsActive	Boolean	FIsActive	SetIsActive
Enabled	Boolean	FEnabled	SetEnabled

Tabel 9.12: Properties indeholdt i klassen TAlarm

Create
Metoden Create aktiveres af metoden Create, beliggende i klassen TTermAdminCreate, og modtager en parameter, AOwner, der definerer hvilket objekt, der ejer det objekt, der oprettes her. Metoden starter med at kalde Create for superklassen TComponent. Herefter initialiseres 2 mediaplayer objekter, MPAlarm1 og MPAlarm2, der er placeret på formen FrmAlarm. I disse to objekter indlæses de lydklip, der kombineres til den alarmlyd, der afspilles ved alarm. Endvidere oprettes og initialiseres 2 timere, TmrAlarm1 og TmrAlarm2, der kontrollerer afspilningen af lyden. Endeligt initialiseres globale variable.

Destroy
Metoden Destroy aktiveres af metoden Destroy, beliggende i klassen TTermAdmin.
Metoden fjerner de 2 timere TmrAlarm1 og TmrAlarm2, hvorefter destructoren for superklassen TComponent kaldes.

TmrAlarm1Timer
MPAlarm1 skal spole tilbage, hvorefter afspilningen skal starte forfra.

TmrAlarm2Timer
MPAlarm2 skal spole tilbage, hvorefter afspilningen skal starte forfra.

SetEnabled
Metoden SetEnabled aktiveres af metoden AISDataRecieve, beliggende i klassen TTermAdmin, eller metoden BtnAfbrydClick, beliggende i klassen TFrmTerminal.
Metoden SetEnabled kører metoden Start i det tilfælde, at metodens inputparameter og property'en IsActive er sande. Ellers aktiveres metoden Stop.

SetIsActive
Metoden SetIsActive aktiveres af metoden BtnDeaktiverClick, beliggende i klassen TFrmTerminal. Metoden sætter variablen FIsActive, ud fra den parameter den er kaldt med. Skal alarmen deaktiveres køres Stop, og skal alarmen aktiveres gøres intet.

Start
Metoden Start aktiveres af metoden SetEnabled. Metoden sætter indledningsvis variablen FEnabled til True, hvilket indikerer, at alarmen er startet. Efterfølgende aktiveres TmrAlarm1 og TmrAlarm2. Endeligt startes afspilningen i objekterne MPAlarm1 og MPAlarm2.

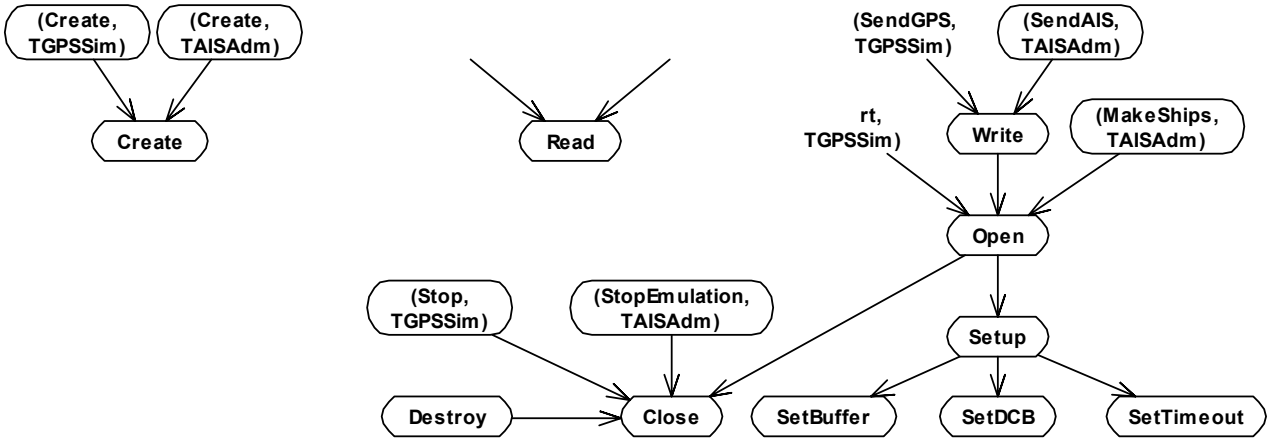
Stop
Metoden Stop aktiveres af metoden SetEnabled eller metoden SetIsActive. Metoden sætter indledningsvis variablen FEnabled til False, hvilket indikerer, at alarmen er stoppet. Efterfølgende deaktiveres TmrAlarm1 og TmrAlarm2. Endeligt stoppes afspilningen i objekterne MPAlarm1 og MPAlarm2.

9.4 Fælles klasser og units

De fælles klasser består af klasserne TSerielport og TDataAccess, endvidere benytter kollisionsdetektoren en række fælles records, der er placeret i unit'en Commontypes.
TSerielport og Commontypes beskrives efterfølgende, hvorimod klassen TDataAccess beskrives i forbindelse med databasen i afsnit 10.

9.4.1 TSerielport

Klassen TSerielport indeholder de metoder, der står for interfacet mellem programmellet og den aktuelle PC's serielport. Sammenhængen mellem disse ses i figur 9.56.



Figur 9.56: Metodesammenhæng for klassen TSerielport

Til kommunikation klasserne imellem, er properties'ene i tabel 9.13 indeholdt i klassen TSerielport

Navn	Datatype	Læser fra	Skriver til
BaudRate	LongWord	FBaudRate	FBaudRate
ByteSize	Byte	FByteSize	FByteSize
Parity	Byte	FParity	FParity
Port	Integer	FPort	FPort
StopBit	Byte	FStopBit	FStopBit
RxBufSize	Integer	FRxBufSize	FRxBufSize
TxBufSize	Integer	FTxBufSize	FTxBufSize

Tabel 9.13: Properties indeholdt i klassen TSerielport

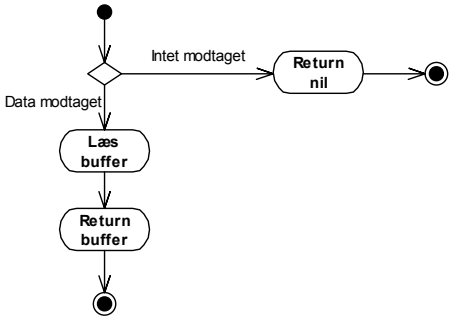
Create

Metoden Create aktiveres af metoden create, beliggende i klassen TGPSSim, eller metoden Create beliggende i klassen TAISAdm.
Denne metode starter med at kalde metoden Create for superklassen TComponent. Dernæst sættes default værdier for parametrene i metoden.

Destroy

Denne destructor aktiverer metoden Close, således serielporten lukkes ned. Endvidere kaldes destructoren for superklassen TComponent.

Read

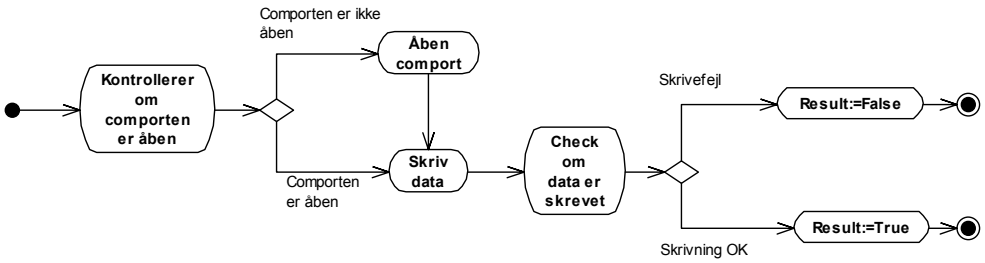


Figur 9.57: Aktivitetsdiagram over metoden Read

Metoden Read (Se figur 9.57) aktiveres af metoden TmrUpdateTimer, beliggende i klassen TGPS, eller metoden TmrUpdateAISTimer, beligggende i klassen TAIS.

Metoden kontrollerer, hvorvidt der er data på serielporten. Er dette tilfældet læses data og disse data gemmes i inputbufferen.

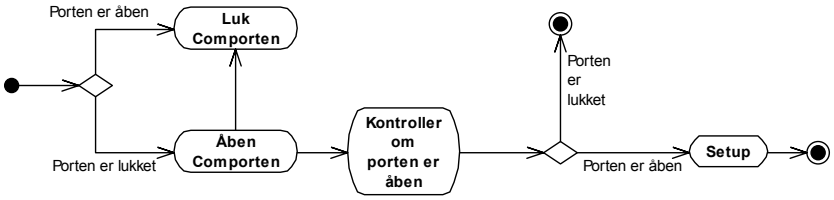
Write



Figur 9.58: Aktivitetsdiagram over metoden Write

Metoden Write (Se figur 9.58) aktiveres af metoden SendGPS, beliggende i klassen TGPSSim, eller metoden SendAIS, beligggende i klassen TAISAdm.
Metoden kontrollerer, hvorvidt der skal skrives til serielporten. Er dette tilfældet, undersøges hvorvidt serielporten er åben. Er serielporten ikke åben aktiveres metoden Open, hvorefter data afsendes. Metoden afsluttes med at kontrollere hvorvidt data er afsendt eller ej.

Open



Figur 9.59: Aktivitetsdiagram over metoden Open

Metoden Open (Se figur 9.59) aktiveres af en af de følgende metoder: Write, Start, beliggende i klassen TGPSSim eller MakeShips, beliggende i klassen TAISAdm.
Metoden kontrollerer, hvorvidt serielporten er åben eller ej. Er dette tilfældet aktiveres metoden Close. Er serielporten lukket, åbnes denne, hvilket returnerer et handle til serielporten. Det undersøges igen, hvorvidt serielporten er åben eller ej, vha. handle't til serielporten. Er serielporten stadig lukket sendes en fejlmeddelelse, hvorimod metoden Setup aktiveres, i det tilfælde, at serielporten er åben.

Close

Metoden Close aktiveres af en af de følgende metoder: Destroy, Open, Stop, beliggende i klassen TGPSSim eller StopEmulation, beliggende i klassen TAISAdm.
Close kontroller hvorvidt serielporten er åben, og er dette tilfældet lukkes handle't til serielporten.

Setup

Figur 9.60: Aktivitetsdiagram over metoden Setup

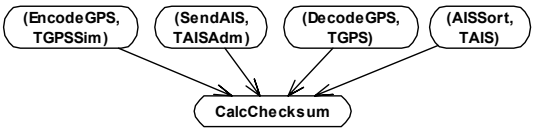
Metoden Setup (Se figur 9.60) aktiveres af metoden Open. Metoden aktiverer metoderne SetBuffer, SetDCB og SetTimeout.

SetBuffer
Metoden SetBuffer aktiveres af metoden Setup. Denne metode aktiverer Windows funktionen SetupComm, der initialiserer kommunikationsparametrene. Parametrene der bliver sat i SetupComm er et handle til serielporten, samt størrelsen på input- og outputbuffer.

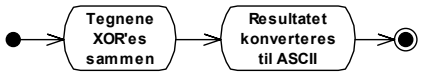
SetDCB
Metoden SetDCB aktiveres af metoden Setup. Denne metode kontrollerer hvorvidt serielporten er åben, og er dette tilfældet hentes DCB-indstillingerne ind. Endeligt sættes de nødvendige indstillinger for serielporten, hvorefter disse gemmes.

SetTimeout
Metoden SetTimeout aktiveres af metoden Setup. Denne metode sætter timeout parametrene for alle læse- og skriveoperationer for serielporten. Eksempelvis den højeste tid mellem modtagelse af to karakterer.

9.4.2 Commontypes
Unit'en Commontypes indeholder en metode og en række records, der benyttes til kommunikation og dataoverførsel klasserne imellem. Metoden CalcChecksum vil blive beskrevet efterfølgende, og en oversigt over de benyttede records følger af nedenstående tabeller.

CalcChecksum


Figur 9.61: Metodesammenhæng for metoden CalcChecksum



Figur 9.62: Aktivitetsdiagram over metoden CalcChecksum

Metoden CalcChecksum (se figur 9.61 og 9.62) aktiveres af en af følgende metoder: EncodeGPS, beliggende i klassen TGPSSim, SendAIS, beliggende i klassen TAISAdm, DecodeGPS, beliggende i klassen TGPS eller AISSort, beliggende i klassen TAIS.
Metoden har til formål at beregne checksummen til sikring af data ved transmission.
De aktuelle tegn XOR'es sammen fortløbende, hvorefter resultatet konverteres til hexadecimal.

TAISShipData
Record'en TAISShipData's opbygning fremgår af tabel 9.14.

Navn	Datatype
MMSI	Longword
NavState	Byte
IMO	Longword
CallSign	String
Name	String
SCType	Byte
GNSSPosA	Word
GNSSPosB	Word
GNSSPosC	Byte
GNSSPosD	Byte
NavSType	Byte
Draught	Double
Destination	String
Rotation	Shortinteger
Speed	Double
Latitude	Double
Longitude	Double
Course	Double

Tabel 9.14: Record'en TAISShipData

TGPSRoute

Record'en TGPSRoute's opbygning fremgår af tabel 9.15.

Navn	Datatype
Longitude	Double
Latitude	Double
Course	Double

Tabel 9.15: Record'en TGPSRoute

TGPSPart

Record'en TGPSPart's opbygning fremgår af tabel 9.16.

Navn	Datatype
Speed	Double
Rotation	Shortinteger
ChangeTime	Integer

Tabel 9.16: Record'en TGPSPart

TGPSData

Record'en TGPSData's opbygning fremgår af tabel 9.17.

Navn	Datatype
Speed	Double
Course	Double
Latitude	Double
Longitude	Double
DateTime	TDateTime

Tabel 9.17: Record'en TGPSData

TOwnShipData

Record'en TOwnShipData's opbygning fremgår af tabel 9.18.

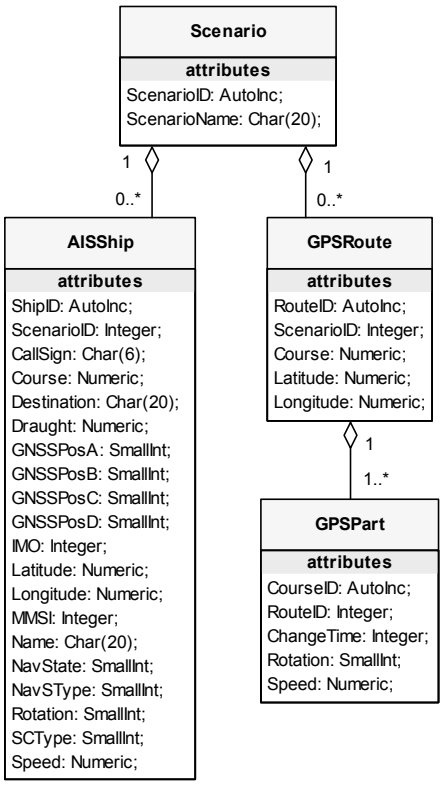
Navn	Datatype
Speed	Double
Course	Double
Latitude	Double
Longitude	Double
ReacTime	Integer

Tabel 9.18: Record'en TOwnShipData

10 Database

10.1 Database

For at kunne reproducere simulerede scenarier, gemmes disse i en database. For hvert scenarie skal der defineres en række skibe til en AIS-simulator, samt en eller flere ruter i nærheden af disse skibe. En rute deles op i flere rutedele, så det er muligt at lave kursændringer og hastighedsændringer midt i en simulation.
For at kunne opnå dette, opbygges databasen med en hovedtabel (master) til scenarier og to undertabeller (detail) til data for henholdsvis AIS-simulering og GPS-simulering. GPS tabellen har endvidere en undertabel til delruter. Både AIS- og GPS-tabellerne er defineret i samme database. Dette gøres for nemmere at kunne holde overblikket over de forskellige scenarier.
Sammenhængen mellem tabellerne er illustreret i klassediagrammet på figur 10.1.



Figur 10.1: Klassediagram over databasen

I klasserne er medtaget primærnøgler og fremmednøgler for tabellerne. En primærnøgle identificerer entydigt en række (record) i den aktuelle tabel. En fremmednøgle identificerer en record i en detailtabel, der tilhører den record i mastertabellen med den tilsvarende primærnøgle.
De datatyper, der er anvendt i databasen er som angivet i tabel 10.1.

Integer:	32 bit heltal med fortegn.
AutoInc:	Integer, som ved oprettelse af en record automatisk får tildelt en værdi, der er 1 højere end den sidst oprettede record. Første værdi er 1.
Smallint:	16 bit heltal med fortegn.
Numeric:	64 bit kommatal med fortegn
Char():	Tekst. Maksimal længde er defineret i parentes

Tabel 10.1: Tabel over brugte datatyper i databasen

I denne database er det valgt at definere alle primærnøgler med datatypen AutoInc. Dette sikrer, at primærnøglen for den enkelte record altid er unik.
Koblingen mellem master- og detailtabel udføres i praksis ved i detailtabellen at definere et filter. Dette filter viser kun de records med en fremmednøgle, der er identisk med primærnøglen for den aktive record i mastertabellen.

10.1.1 Scenario

Navn	Datatype
ScenarioID	Autoincrement
ScenarioName	Char (længde 20)

Tabel 10.2: Attributter til tabellen Scenario

Tabellen Scenario indeholder attributterne i tabel 10.2, og fungerer som mastertabel for hele databasen. Ud over primærnøglen, ScenarioID, indeholder denne tabel kun en attribut, ScenarioName af typen Char(20), der gør det muligt at navngive scenarierne. Dette navn er det eneste, der er fælles for de undertabeller, der bruges til henholdsvis AIS- og GPS-simulatoren.

10.1.2 AISShip

Navn	Datatype
ShipID	Autoincrement
ScenarioID	Long integer
CallSign	Char (længde 6)
Course	Number
Destination	Char (længde 20)
Draught	Short integer
GNSSPosA	Short integer
GNSSPosB	Short integer
GNSSPosC	Short integer
GNSSPosD	Short integer
IMO	Long integer
Latitude	Number
Longitude	Number
MMSI	Long integer
Name	Char (længde 20)
NavState	Short integer
NavSType	Short Integer
Rotation	Short integer
SCType	Short integer
Speed	Number

Tabel 10.3: Attributter til tabellen AISShip

Tabellen AISShip indeholder attributterne i tabel 10.3, og indeholder oplysninger om de skibe, der skal simuleres i AIS simulatoren. Den har primærnøglen ShipID, samt fremmednøglen ScenarioID, hvilket henviser til, at tabellen er en detailtabel til Scenario. Hver record i Scenario kan være associeret med nul, en eller flere records i AISShip.
Tabellen indeholder både de statiske oplysninger, der sendes i en AIS pakke 5, samt startværdierne for de dynamiske oplysninger, der sendes i en AIS pakke 1, 2 og 3. Dataformatet for attributterne er identisk med dataformatet for de tilsvarende felter i AIS pakkerne med den undtagelse, at nogle af felterne i AIS pakkerne sendes som i brøkdele, hvorimod værdierne gemmes med den oprindelige enhed i databasen. For eksempel sendes længden og bredden i 1/10000 dele af minutter, men i databasen gemmes disse tal i minutter.

10.1.3 GPSRoute

Navn	Datatype
RouteID	Autoincrement
ScenarioID	Long integer
Course	Number
Latitude	Number
Longitude	Number

Tabel 10.4: Attributter til tabellen GPSRoute

Tabellen GPSRoute indeholder attributterne i tabel 10.4, og indeholder oplysninger om de ruter, der simuleres i GPS simulatoren. Den har primærnøglen RouteID samt fremmednøglen ScenarioID, hvilket henviser til, at tabellen er en detailtabel til Scenario. Hver record i Scenario kan være associeret med nul, en eller flere records i GPSRoute.
Tabellen indeholder startposition og startkurs for hver GPS rute. Positionen er angivet i minutter, medens kursen er angivet i grader.

10.1.4 GPSPart

Navn	Datatype
CourseID	Autoincrement
RouteID	Long integer
ChangeTime	Long integer
Rotation	Number
Speed	Number

Tabel 10.5: Attributter til tabellen GPSPart

Tabellen GPSPart indeholder attributterne i tabel 10.5, og indeholder oplysninger om dele af de ruter, der simuleres i GPS simulatoren. Den har primærnøglen CourseID samt fremmednøglen RouteID, hvilket henviser til, at tabellen er en detailtabel til GPSRoute.
Tabellen indeholder hastighed og drejningsrate for hver del af en GPS rute, samt den tid denne delrute skal simuleres. Hastigheden gemmes i knob, drejningsraten gemmes i grader/min, og tiden gemmes i millisekunder.

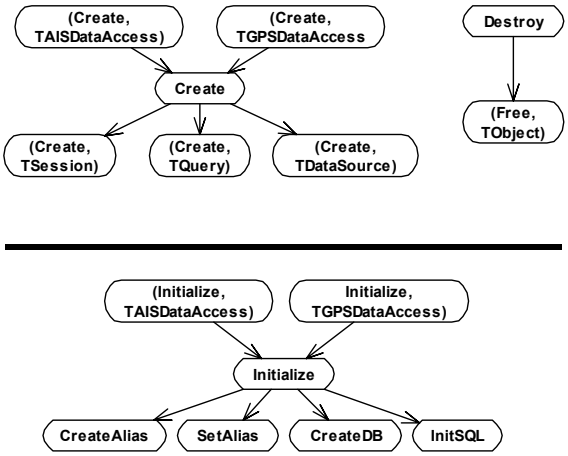
10.2 Tilgang til database

For at kunne tilgå og manipulere data og database, benyttes objekter af klasserne TSession, TQuery og TDataSource.
TSession objektet holder den overordnede kontrol med databasen, herunder det alias, der bruges internt i programmet til at referere til databasen.

TQuery objekter fungerer som interface til den enkelte tabel i databasen, og alle læsninger og skrivninger til tabellen går gennem dette objekt. Det er også her tabeller kan oprettes, modificeres og slettes. Kontrol med TQuery objektet foregår ved hjælp af sproget SQL, der er et sprog for tilgang til relationelle databaser. En relationel database er en database, der kun består af tabeller, og ikke indeks eller lignende til at håndtere opslag. SQL består af to dele, en datadefinitionsdel, DDL, og en datamanipulationsdel, DML. DDL omfatter de dele af sproget, der manipulerer selve tabellerne, dvs. oprettelse og sletning af tabeller, definition af attributter og datatyper osv. DML omfatter de dele af sproget, der manipulerer med data i tabellerne.
TDataSource objekter fungerer som interface mellem visuelle database komponenter på en form, og et TQuery objekt. Det er ligeledes objekter af klassen TDataSource, der fungerer som kobling fra en mastertabel til detailtabeller.

10.2.1 TDataAccess

Denne klasse indeholder de funktioner, der er fælles for adgangen til både AIS databasen og GPS databasen. Dette inkluderer definition af databaseplacering samt alle funktioner og procedurer, der relaterer til Scenario tabellen. Sammenhængen for metoderne er illustreret i figur 10.2.



Figur 10.2: Metodesammenhæng for klassen TDataAccess

Create

Metoden Create aktiveres af constructorerne for subklasserne TAISDataAccess og TGPSDataAccess. Den starter med at kalde constructoren for superklassen TComponent. Herefter sætter den en default databasesti, der defineres som biblioteket Database under programbiblioteket. Hvis denne sti ikke eksisterer, oprettes den, og efterfølgende oprettes objektet SData af klassen TSession.
Herefter oprettes objekterne QryScenario og DsScenario, henholdsvis af klasserne TQuery og TDataSource.
Endeligt defineres navnet på det alias, der anvendes som reference til databasen.

Destroy

Denne metode kaldes af destructorerne for subklasserne TAISDataAccess og TGPSDataAccess. Den starter med at deaktivere SData, hvorefter dette objekt fjernes. Herefter lukkes QryScenario's adgang til databasen, hvorefter dette objekt også fjernes. Afslutningsvis fjernes DsScenario, og destructoren for superklassen TComponent kaldes.

Initialize
Metoden Initialize sørger for at initialisere databasen, så den er klar til brug. Dette foregår ved at kalde metoderne CreateAlias, SetAlias, CreateDB og InitSQL, hvis den definerede sti til databasen eksisterer.

CreateAlias
Denne metode opretter et alias til databaseplaceringen. Hvis der allerede eksisterer et alias til databasen, modificeres dette, ellers oprettes et nyt.

SetAlias
Denne metode sætter det nyligt oprettede alias for QryScenario, således det defineres hvilken database, der skal tilgås. Desuden defineres, QryScenario således databasen tilgås med skriveadgang.

CreateDB
Metoden CreateDB undersøger, om tabellen Scenario er oprettet, og hvis dette ikke er tilfældet, oprettes denne tabel efter definitionen i afsnit 10.1.1. Dette foregår ved hjælp sætninger fra DDL-delen af SQL sproget. Disse SQL kommandoer eksekveres ved hjælp af QryScenario.

InitSQL
I metoden InitSQL defineres hvilken tabel, og hvilke attributter fra denne, der skal tilgås fra QryScenario. Det vælges at tilgå alle attributter i tabellen Scenario. Dette foregår ved hjælp af sætninger fra DML delen af SQL sproget. Disse sætninger lægges i QryScenario, og tabellen åbnes. Herefter vil alle operationer med QryScenario operere i forhold til disse sætninger.

GetScenarios
Denne metode, der tilgås af property'en Scenarios vil oprette en TStringList med navnene på alle de scenarier, der findes i Scenario. Dette gøres ved at læse attributten ScenarioName for alle records i tabellen, hvorefter disse tilføjes til listen. Herefter returneres denne liste.

SetFirstScenario
Denne metode vil forsøge at vælge den første record i tabellen Scenario, hvorefter den vil returnere en boolean, der fortæller om dette er lykkedes.

GetScenarioCount
Metoden SetScenariocount returnerer antallet af records, der matcher de SQL sætninger, der er defineret i QryScenario.

GetScenarioName
Hvis der eksisterer records, der matcher SQL sætningerne i QryScenario, vil denne metode returnere indholdet af attributten ScenarioName for den aktive record.

SetNextScenario
Denne metode vil forsøge at vælge den næste record i tabellen Scenario, hvorefter den vil returnere en boolean, der fortæller om dette er lykkedes.

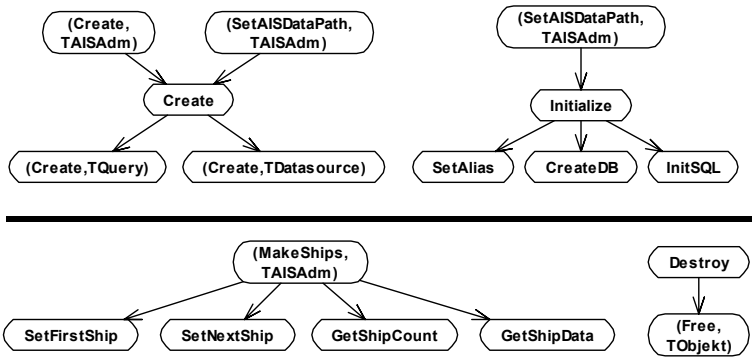
SetScenario
Denne metode vil vælge den record i Scenario, hvor indholdet af attributten ScenarioName er lig den parameter, der modtages. Hvis der ikke findes en sådan record, vil der komme en fejlbesked.

SetScenarioName
Metoden SetScenarioName ændrer indholdet af attributten ScenarioName for tabellen Scenario. Medens indholdet bliver ændret, deaktiveres eventhandleren for OnDataChange i objektet DsScenario. Dette sker for at undgå, at skrivningen til tabellen bliver afbrudt.

NewScenario
Metoden NewScenario vil føje en ny record til tabellen Scenario. Indholdet af denne record er specificeret af den parameter, der modtages til metoden.

DelScenario
Denne metode sletter den aktive record i tabellen Scenario, hvorefter den efterfølgende record bliver aktiv. Er den slettede record er den sidste i tabellen, aktiveres den forrige record aktiv.

10.2.2 TAISDataAccess
Klassen TAISDataAccess er, en subklasse til klassen TDataAccess. Formålet med denne klasse er at skabe en forbindelse mellem AIS-simulatoren og databasen, således AIS-informationer kan indhentes.
Metodesammenhængen for metoderne i klassen TAISDataAccess ses i figur 10.3.



Figur 10.3: Metodesammenhæng for klassen TAISDataAccess

Til kommunikation klasserne imellem, er properties'ene i tabel 10.6 indeholdt i TAISDataAccess:

Navn	Datatype	Læser fra	Skriver til
ShipData	TAISShipData	GetShipData	SetShipData
ShipCount	Integer	GetShipCount	-
Ships	Tstringlist	GetShips	-

Tabel 10.6: Properties indeholdt i klassen TAISDataAccess

Create
Metoden Create kaldes af metoderne Create eller SetAISDataPath, i klassen TAISAdm.
Metoden starter med at kalde constructoren for superklassen TDataAccess.
Endvidere oprettes objekterne QryAISShip og DsAISShip af klasserne TQuery og TDataSource.

Destroy
Fjerner de objekter, der blev oprettet under Create. Dette gøres vha. metoden Free, i klassen TObjekt. TQuery objekter skal dog først lukkes for at lukke adgangen til databasen. Til sidst kaldes destructoren for superklassen TDataAccess.

Initialize
Metoden Initialize kaldes fra metoden SetAISDataPath, beliggende i klassen TAISAdm. Denne metode har til formål at initialisere databasen.
Metoden starter med at kalde Initialize for superklassen TDataAccess, hvorefter metoderne SetAlias, CreateDB og InitSQL kaldes.

SetAlias
Her sættes aliaset fra TDataAccess for QryAISShip. Endvidere sættes også properties for QryAISShip, der definerer, at databasen tilgås med skriveadgang, og at QryAISShip bruger DsScenario fra superklassen TDataAccess som link til en masterklasse.

CreateDB
Metoden undersøger, hvorvidt tabellen AISShip er oprettet, og hvis dette ikke er tilfældet, oprettes denne tabel efter definitionen i afsnit 10.1.2. Dette foregår ved hjælp sætninger fra DDL-delen af SQL sproget. Disse SQL kommandoer eksekveres ved hjælp af QryAISShip.

InitSQL
I denne metode defineres hvilken tabel, og hvilke attributter fra denne, der skal tilgås fra QryAISShip. Det vælges at tilgå alle attributter i tabellen AISShip og alle records, hvor fremmednøglen ScenarioID matcher primærnøglen fra Scenario. Dette foregår ved hjælp af sætninger fra DML delen af SQL sproget. Disse sætninger lægges i QryAISShip, og tabellen åbnes. Herefter vil alle operationer med QryAISShip operere i forhold til disse sætninger.

SetFirstShip
Metoden SetFirstShip aktiveres af metoden MakeShips, i klassen TAISAdm. Denne metode har til formål at vælge den første record i tabellen AISShip, således denne record kan læses gennem property'en ShipData.

SetNextShip
Metoden SetNextShip aktiveres ligeledes af metoden MakeShips, i klassen TAISAdm. Denne metode har til formål at vælge den næste record i tabellen AISShip, således denne record kan læses gennem property'en ShipData.

GetShipCount
Metoden GetShipCount kaldes tilsvarende af metoden MakeShips, i klassen TAISAdm, igennem property'en ShipCount. Denne metode returnerer antallet af skibe i tabellen AISShip, der tilhører det aktive scenarie.

GetShipData
Metoden GetShipData kaldes også af metoden MakeShips, i klassen TAISAdm, igennem property'en ShipData. Denne metode henter de AIS-informationer, der er i den aktive record i tabellen AISShip, og returnerer disse i en record af typen TAISShipData.

SetShipData
Denne metode kaldes gennem property'en ShipData. Den skriver den modtagne record af typen TAISShipData til den aktive record i tabellen AISShip.

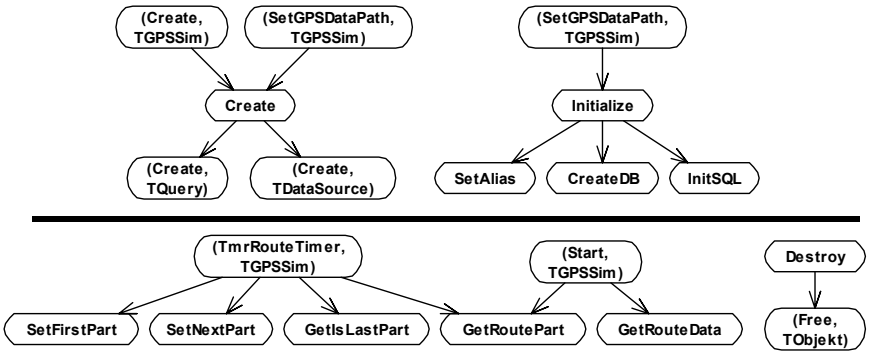
NewShip
Denne metode føjer en ny record til tabellen AISShip. Data til denne record modtages som en parameter af typen TAISShipData.

DelShip
Metoden DelShip sletter den aktive record i tabellen AISShip.

DelShips
Denne metode kører metoden DelShip lige så mange gange, der er records i AISShip. Dette vil slette alle de skibe, der hører under det aktive scenarie.

CheckMMSI
Metoden CheckMMSI vil søge i tabellen AISShip efter det MMSI nummer, der modtages som parameter. Hvis der findes en record med det angivne MMSI nummer, vil metoden returnere true, ellers false.

10.2.3 TGPSDataAccess
Klassen TGPSDataAccess er en subklasse til klassen TDataAccess.
Formålet med denne klasse er at skabe en forbindelse mellem GPS-simulatoren og databasen, således GPS-informationer kan indhentes.
Metodesammenhængen for metoderne i klassen TGPSDataAccess ses i figur 10.4.



Figur 10.4: Metodesammenhæng for klassen TGPSDataAccess

Til kommunikation klasserne imellem, er properties'ene i tabel 10.7 indeholdt i TGPSDataAccess:

Navn	Datatype	Læser fra	Skriver til
RouteData	TGPSRoute	GetRouteData	SetRouteData
RoutePart	TGPSPart	GetRoutePart	SetRoutePart
RouteCount	Integer	GetRouteCount	-
PartCount	Integer	GetPartCount	-
RouteID	Integer	GetRouteID	-
Routes	TStringlist	GetRoutes	-
Route	Integer	-	SetRoute
IsLastPart	Boolean	GetIsLastPart	-

Tabel 10.7: Properties indeholdt i klassen TGPSDataAccess

Create
Metoden Create kaldes af metoderne Create eller SetGPSDataPath, i klassen TGPSSim.
Først kaldes constructoren for superklassen TDataAccess. Dernæst oprettes objekterne QryGPSRoute og QryGPSPart af klassen TQuery og objekterne DsGPSRoute og DsGPSPart af klassen TDataSource.

Destroy
Denne metode benyttes til at nedlægge de objekter, der blev oprettet i metoden Create, i klassen TGPSDataAccess. Dette gøres vha. metoden Free, i klassen TObjekt. TQuery objekter skal dog lukkes før de kan fjernes.
Til sidst køres destructoren for superklassen TDataAccess.

Initialize
Metoden Initialize kaldes fra metoden SetGPSDataPath, beliggende i klassen TGPSSim. Denne metode har til formål at initialisere databasen.

Metoden starter med at kalde initialize for superklassen TDataAccess, hvorefter metoderne SetAlias, CreateDB og InitSQL kaldes.

SetAlias
Denne metode sætter aliaset fra TDataAccess for QryGPSRoute og QryGPSPart. Endvidere sættes også properties for disse objekter, der definerer, at databasen tilgås med skriveadgang. QryGPSRoute og QryGPSPart bruger henholdsvis DsScenario fra superklassen TDataAccess og DsGPSRoute som link til en masterklasse.

CreateDB
Metoden undersøger, hvorvidt tabellerne GPSRoute og GPSPart er oprettet. Er dette ikke tilfældet, oprettes den/de der mangler efter definitionen i henholdsvis afsnit 10.1.3 og 10.1.4. Dette foregår ved hjælp sætninger fra DDL-delen af SQL sproget. Disse SQL kommandoer eksekveres ved hjælp af objektet QryGPSRoute. Grunden til at GPSPart ikke bruges til dette er, at dette objekt ikke kan aktiveres før QryGPSRoute er færdigdefineret. Dette skyldes det link fra QryGPSPart til DsGPSRoute, der er defineret.

InitSQL
I metoden defineres hvilke tabeller, og hvilke attributter fra disse, der skal tilgås fra henholdssvis QryGPSRoute og QryGPSPart. QryGPSRoute tilgår alle attributter i tabellen GPSRoute og alle records, hvor fremmednøglen ScenarioID matcher primærnøglen fra Scenario. QryGPSPart tilgår alle attributter i tabellen GPSPart og alle records, hvor fremmednøglen RouteID matcher primærnøglen fra GPSRoute. Dette foregår ved hjælp af sætninger fra DML delen af SQL sproget. Disse sætninger lægges i de to TQuery objekter, og tabellerne åbnes. Herefter vil alle operationer med QryGPSRoute og QryGPSPart operere i forhold til disse sætninger.

SetFirstPart
Metoden kaldes af metoden TmrRouteTimer, i klassen TGPSSim.
Denne metode har til formål at klargøre rute-informationer, for den første del af den valgte rute, i databasen.
Metoden forsøger at vælge den første record under QryGPSPart, hvorefter den returnerer en boolean, der fortæller hvorvidt dette er lykkedes.

SetNextPart
Metoden kaldes af metoden TmrRouteTimer, i klassen TGPSSim.
Denne metode har til formål at klargøre rute-informationer, for den næste del af den valgte rute, i databasen.
Metoden forsøger at vælge den næste record under QryGPSPart, hvorefter den returnerer en boolean, der fortæller hvorvidt dette er lykkedes.

GetIsLastPart
GetIsLastPart kaldes af metoden TmrRouteTimer, i klassen TGPSSim, igennem property'en IsLastPart.
Formålet med denne metode er at undersøge, hvorvidt den aktuelle rute del er den sidste, i den valgte rute. Dette returnerer en boolean.

GetRoutePart
Metoden kaldes af metoden TmrRouteTimer, i klassen TGPSSim, igennem property'en RoutePart. GetRoutepart returnerer den aktive record fra QryGPSPart i en record, som klassen TGPSSim har adgang til. Dette gøres efter, at en af metoderne SetFirstPart eller SetNextPart har valgt denne.

GetRouteID
Denne metode returnerer ID for den aktive rute. Dette sker ved at læse primærnøglen, RouteID, fra QryGPSRoute.

GetRouteCount
Denne metode returnerer antallet af ruter i det aktive scenarie, det vil sige de ruter, der er i QryGPSRoute.

GetRoutes
Denne metode returnerer en TStringlist indeholdende RouteID fra alle de ruter, der er i QryGPSRoute.

SetFirstRoute
Metoden vælger den første rute i scenariet, dvs. i QryGPSRoute. Der returneres true eller false afhængigt af, hvorvidt dette er lykkedes.

SetNextRoute
Metoden vælger den næste rute i scenariet, dvs. i QryGPSRoute. Der returneres true eller false afhængigt af, om dette er lykkedes.

GetPartCount
Denne metode returnerer antallet af delruter i det aktive scenarie, det vil sige de delruter, der er i QryGPSPart.

GetRouteData
Denne metode returnerer indholdet af den aktive record i tabellen over ruter, det vil sige fra QryGPSRoute.

SetRouteData
Metoden skriver modtagne data i den aktive record i tabellen over ruter, hvilket betyder skrivning til QryGPSRoute.

SetRoutePart
Metoden skriver modtagne data i den aktive record i tabellen over delruter, hvilket betyder skrivning til QryGPSPart.

SetRoute
Denne metode forsøger at vælge den rute i QryGPSRoute, hvor attributten RouteID er lig det ID, der er modtaget som parameter. Hvis dette ikke lykkes, vises en fejlbesked.

NewRoute
Denne metode opretter en ny rute indeholdende de data, der modtages som parameter til metoden. Disse data tilføjes tabellen i QryGPSRoute.

NewPart
Denne metode opretter en ny delrute indeholdende de data, der modtages som parameter til metoden. Disse data tilføjes tabellen i QryGPSPart.

DelRoute
Denne metode sletter den aktive rute i QryGPSRoute. Først køres metoden DelParts for at slette data i undertabellen, hvorefter den aktive rute slettes.

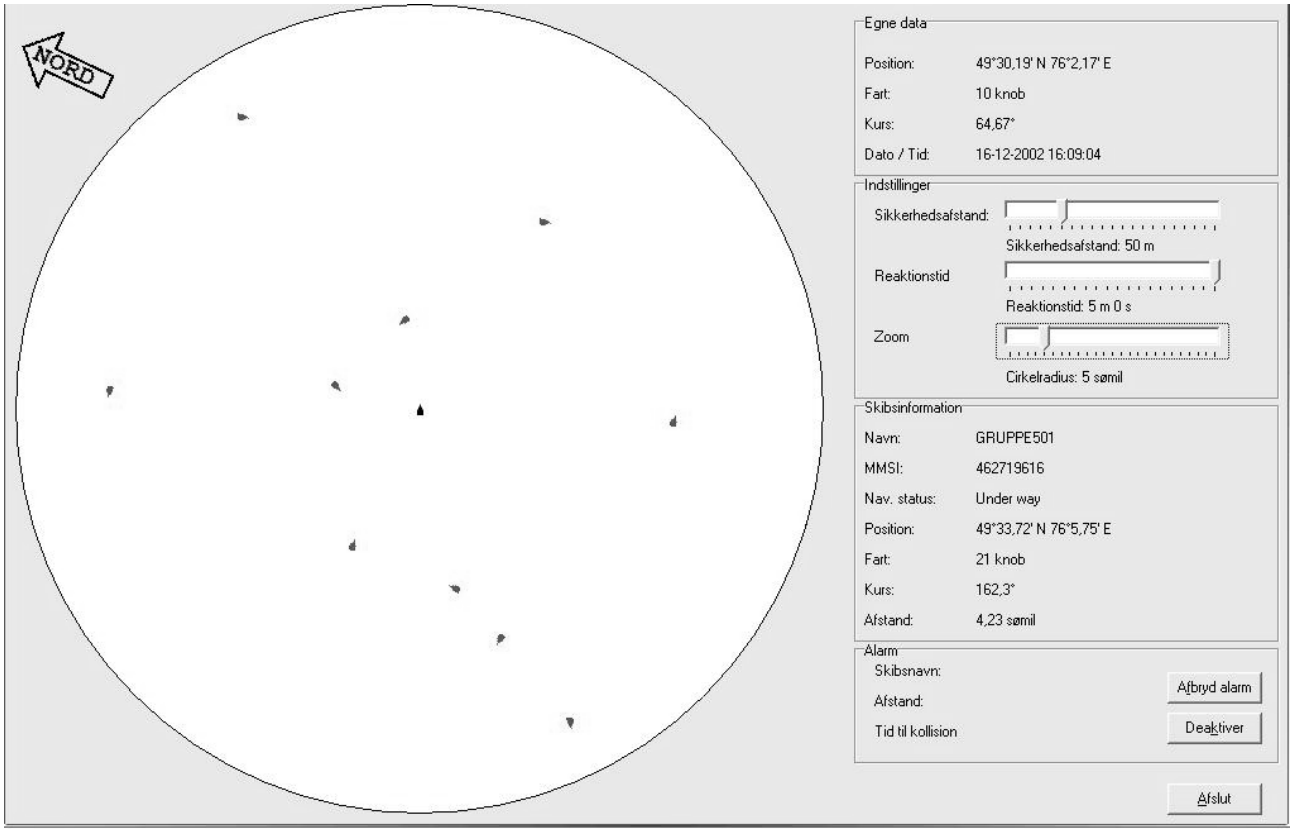
DelRoutes
Denne metode sletter alle ruter, der hører til et bestemt scenarie. Dette gøres ved at køre DelRoute lige så mange gange som der er records i QryGPSRoute.

DelPart
Metoden sletter den aktive delrute i QryGPSPart.

DelParts
Denne metode sletter alle delruter, der hører til en bestemt rute. Dette gøres ved at køre DelPart lige så mange gange som der er records i QryGPSPart.

11 GUI

I det følgende afsnit vil kollisionsdetektorens Graphic User Interface (GUI), blive beskrevet. Et snapshot af kollisionsdetektorens brugerflade, under drift, ses i figur 11.1.



Figur 11.1: Kollisionsdetektorens brugerflade under drift.

Med baggrund i figur 11.1 vil brugerfladens funktionalitet samt specifikke egenskaber blive beskrevet.

I kravspecifikationen er opstillet en række krav til funktionen af kollisionsdetektoren. En række af disse krav står i direkte forbindelse til brugerfladen, hvorfor beskrivelsen af GUI vil tage udgangspunkt i disse krav.
Et af kravene til kollisionsdetekteren lyder:

- Skal kunne give en grafisk visualisering af omkringliggende AIS kompatible skibe, vha. en PC.

Dette krav opfyldes, idet brugerfladen er udstyret med en skærmcirkel, indenfor hvilken alle skibe, der er AIS kompatible vil blive plottet. Eget skib er placeret i centrum af denne skærmcirkel, og omkringliggende skibe plottes i forhold til eget skib. Skærmcirklens radius angiver indenfor hvilken afstand skibe vil blive plottet. Denne radius kan sættes og varieres af brugeren.

Et andet krav, til kollisionsdetektoren, lyder:

- Skal kunne afgøre, hvorvidt der er risiko for kollision, og i tilfælde af dette advare brugeren visuelt og auditivt.

Dette krav opfyldes, idet skibe der udgør en kollisionsrisiko, indenfor skærmcirklen, vil fremtræde med en rød farve og således advare brugeren. Endvidere vil en kollisionsrisiko aktivere en auditiv alarm.

De to førnævnte krav vedrørte brugerfladens visuelle og auditive udformning, hvorimod de følgende krav omhandler brugerens muligheder for aktivt at benytte brugerfladen:

- Skal kunne give brugeren mulighed for at indhente informationer omkring AIS-kompatible skibe.
- Skal kunne give brugeren mulighed for at indstille sikkerhedsafstanden og reaktionstiden til omkringliggende skibe.
- Skal kunne give brugeren mulighed for at afbryde audioalarmen.

Disse krav opfyldes af brugerfladens specifikke egenskaber.

11.1 Specifikke egenskaber

Brugerfladens specifikke egenskaber er implementeret i form af en række knapper og trackbars, placeret på brugerfladens højre halvdel. Disse egenskaber beskrives efterfølgende.

Egne data

Boksen ”Egne data” giver løbende oplysninger om eget skib. Der gives således oplysninger omhandlende position, fart, kurs samt dato og tid. Disse oplysninger erhverves via GPS-simulatoren, og formidles til brugeren i form af fire labels. De ønskede oplysninger associeres med disse labels via metoden AdminGPSUpdated, beliggende i klassen TFrmTerminal jf. afsnit 9.3.1.

Indstillinger

Boksen ”Indstillinger” giver brugeren mulighed for, vha. tre trackbars, at ændre parametrene for henholdsvis sikkerhedsafstand, reaktionstid og zoom.
Sikkerhedsafstanden kan indstilles til minimum 10m og maksimum 200m, i intervaller af 10m. Dette sker via metoden SikkerBarChange, beliggende i klassen TFrmTerminal.
Reaktionstiden kan indstilles til minimum 15s og maksimum 5 minutter, i intervaller af 15s. Dette sker via metoden ReakBarChange, beliggende i klassen TFrmTerminal.
Zoom kan indstille radius af skærmcirklen til et minimum af 1 sømil og et maksimum af 25 sømil, i intervaller af 1 sømil. Dette sker via metoden TBZoomChange, beliggende i klassen TFrmTerminal.

Skibsinformation

Boksen ”Skibsinformation” giver brugeren relevante oplysninger omhandlende et omkringliggende skib. I det tilfælde, at brugeren ønsker disse oplysninger omkring et skib, skal brugeren blot klikke på det ønskede skib.
De oplysninger, der fremkommer i boksen ved klik på et skib, vil bestå af følgende informationer: Navn, MMSI nummer, navigations status, position, fart, kurs og afstand til eget skib.
Hvis brugeren klikker indenfor skærmcirklen, men dog ikke på et skib, vil skibsinformationerne slettes.
Denne funktionalitet tilgås via metoden SkibeClick, beliggende i klassen TFrmTerminal.

Alarm

I det tilfælde, at et skib udløser en alarm, vil det, af tre labels i boksen ”Alarm”, fremgå hvilket skib, der har udløst alarmen, den nuværende afstand til skibet og tiden indtil sikkerhedsafstanden brydes. Dette sker via metoden UpdateShip, beliggende i klassen TFrmTerminal.
Endvidere vil brugeren have mulighed for at afbryde alarmen, eller deaktivere denne.
Er alarmen udløst, har brugeren mulighed for at afbryde denne, vha. knappen ”Afbryd alarm”. Afbrydes alarmen vil denne starte igen, næste gang kollisionsdetektoren detekterer en

kollisionsrisiko, fra samme eller et andet skib. Vælger brugeren at deaktivere alarmen, vil denne forblive deaktiveret, indtil det tidspunkt at brugeren aktiverer alarmen igen.
Knappen ”Afbryd alarm” styres af metoden BtnAfbrydClick, beliggende i klassen TFrmTerminal, og knappen ”Deaktiver alarm” styres af metoden BtnDeaktiverClick, ligeledes beliggende i klassen TFrmTerminal.

Diverse

Udover de førnævnte funktionaliteter, indeholder brugerterminalen en kompasnål i øverste venstre hjørne. Denne kompasnål indikerer i hvilken retning eget skib sejler. Dette er nødvendigt, idet eget skib altid vil have retning lodret opad på skærmcirklen. Kompasnålen styres af metoden AdminGPSUpdate, beliggende i klassen TFrmTerminal.
Som tidligere nævnt vil et skib, der har udløst en alarm, fremtræde med en rød farve. Endvidere vil andre skibe fremtræde med en grøn farve, og eget skib med en sort farve.

12 Modul-/modulintegrationstest

Formålet med modultesten er at teste hver enkelt metode, for således at sikre, at metoden fungerer som beskrevet i metodedesignet. Modultesten skal endvidere teste den enkelte metodes grænseflade. Det er valgt at kombinere modultesten med modulintegrationstesten, idet mange af metoderne er afhængige af hinanden.

Da en metodetest af samtlige metoder vil være meget tidskrævende, er det valgt at teste udvalgte metoder.
Dette er gjort ved at teste metoder, benyttet i henholdsvis GPS-simulatoren, AIS-simulatoren og Brugerterminalen.
Metoderne er, i lighed med metodedesignet, beskrevet under den klasse de tilhører.
For hver klasse er der lavet et testprogram til test af metoderne i den aktuelle klasse. Disse testprogrammer er placeret på den medfølgende CD i mappen "Test".

12.1 GPS-simulator

12.1.1 Modultest af udvalgte metoder af TGPSSim

Navnet på filen til testprogrammet til denne klasse er GPSSIMTEST.exe.

Zero

Formålet med metoden Zero er, at formatere tal til det format, der skal anvendes til afsendelse fra GPS-simulatoren.
Eksempel:
I GPS-indkapslingen, skal tallet 3,6430952325 laves om til 03.64309. Metoden kaldes med parametrene: Cifre før komma, Decimaler og selve tallet, der skal formateres.
Ved test er det konstateret, at denne metode fungerer korrekt.

CalcGPS

CalcGPS metodens formål er at beregne en ny position ud fra den foregående, fart, kurs, rotation og sendeinterval. CalcGPS-metoden er testet ved parametrene i tabel 12.1

parametre	Værdi
Længde	3252,533 minutter = 54° 12' 533
Bredde	1353,5333 minutter = 22° 33' 533
Kurs	0,4 π =72°
Fart	12 knob
Rotation	13 grader per minut
Interval	2 sek

Tabel 12.1: GPS-parametre.

Testen fandt følgende differencer per sendeinterval:

Længde: 0,00586946851499063
Bredde: 0,00201210197172903

Disse resultater kontrolleres i ligning 12.1 og 12.2:

$$\Delta\text{Længde} = \frac{12\text{knob} \cdot 2000\text{ms} \cdot \cos\cdot(2,5\pi - 0,4\pi) \cdot \cos\left(\frac{\left(\frac{1353,5333\text{minutter}}{60}\right)\cdot\pi}{180}\right)}{3600000} \qquad (12.1)$$

$$\Downarrow$$

$$\Delta\text{Længde} = 0,0058552474\,\text{minutter}/\text{interval}$$

$$\Delta\text{Bredde} = \frac{12\text{knob} \cdot 2000\text{ms} \cdot \sin(2,5\pi - 0,4\pi)}{3600000} = 0,0020601133\,\text{minutter}/\text{interval} \qquad (12.2)$$

Forskellene mellem de testede og de beregnede resultater kan tilskrives at sinus og cosinus er implementeret forskelligt i delphi og i den, til beregningerne, anvendte lommeregner. Denne påstand er blevet testet og bekræftet.
På baggrund af dette kan det konkluderes at CalcGPS-proceduren virker efter hensigten.

EncodeGPS

Formålet med denne metode er at pakke GPS-informationerne i henhold til NMEA 0183 standarden.
På baggrund af de beregnede GPS-informationer (Tabel 12.1) genererer metoden Encode GPS følgende sætning:

$GPRMC,163602.00,A,2233.53531,N,05412.53883,E,012.00,072.43,131202,0.0,E,A*31

Forskellene mellem de oprindeligt indtastede værdier (se tabel 12.1), og de udsendte, skyldes at dataene er blevet beregnet og opdateret af CalcGPS på det tidspunkt EncodeGPS laver GPS-strengen. EncodeGPS pakker GPS-strengen ud fra variable, der er blevet opdateret af CalcGPS. Disse variable er blevet kontrolleret, og fundet korrekte, i forhold til ovenstående streng.
Det kan konkluderes, at EncodeGPS-metoden fungerer korrekt.

12.2 AIS-simulator

12.2.1 Modultest af udvalgte metoder af TAISAdm

Navnet på filen til testprogrammet til denne klasse er AISTEST.exe.

CalcInt

Metoden CalcInt benyttes til at beregne hvor ofte pakke 1,2 eller 3 skal afsendes.
Dette gøres ved, i testprogrammet, at indtaste variablene fart og rotation. Sættes disse variable, inden for de angivne intervaller, i tabel 2.4 i afsnit 2.2, kan resultatet aflæses ved at køre testprogrammet.
Da resultaterne af testprogrammet er identisk med det forventede i tabel 12.2 må det konkluderes at metoden fungerer korrekt.

Situation	Interval
Ligger for anker	3 min.
Sejler 0-14 knob	12 sek.
Sejler 0-14 knob og ændrer kurs	4 sek.
Sejler 14-23 knob	6 sek.
Sejler 14-23 knob og ændrer kurs	2 sek.
Sejler over 23 knob	3 sek.
Sejler over 23 knob og ændrer kurs	2 sek.

Tabel 12.2: Interval for afsendelse af pakke 1[AIS s. 3]

CalcChecksum

Denne metode har til formål at beregne checksummen af dele af de AIS-informationer, der skal afsendes som NMEA.
Ved test hentes testsætninger fra NMEA 0183 standarden [NMEA s.8]:

GPGLL,5057.970,N,00146.110,E,142451,A og
GPVTG,089.0,T,,,15.2,N,,

Checksummene for disse sætninger er henholdvis 27 og 7F, hvilket er identisk med resultatet af testen. Det kan derfor konkluderes at metoden virker.

Choosewhich

Metoden ChooseWhich har til formål at vælge hvilken af to allerede specificerede pakker, der har højest prioritet. Metoden er i stand til at prioritere pakker fra samme, eller forskellige skibe, og pakker af forskellige typer.
Har den ene pakke højere prioritet, end den anden, er det den pakke med højest prioritet, der skal vælges. Har de to pakker lige høj prioritet, skal valget foretages, med en tilfældighedsfunktion.
Funktionen skal kunne sammenligne de 2 pakkertype, 1-3 og 5, ligegyldigt hvilke pladser parametrene har i metoden. Endvidere kan prioritetsparametrene, have samme prioritet, eller forskellige prioriteter. Dette giver 8 kombinationer, hvilket er illustreret i tabel 12.3:

Testnummer	Pakke nr.	Prioritet	Skib
1	1 og 1	Identisk	Tilfælde
		Forskellig	Højest prioritet
2	1 og 5	Identisk	Tilfælde
		Forskellig	Højest prioritet
3	5 og 1	Identisk	Tilfælde
		Forskellig	Højest prioritet
4	5 og 5	Identisk	Tilfælde
		Forskellig	Højest prioritet

Tabel 12.3: Prioriteringstabel

Test viser, at resultaterne er i overensstemmelse med indholdet i tabel 12.3, hvorfor det må konkluderes at metoden virker efter hensigten.

Select

Denne metode har til formål at vælge hvilket skib og hvilken pakke, der skal afsendes i det pågældende timerinterval, i det tilfælde, at en pakke skal afsendes.
Select udvælger det skib, der skal afsendes, og returnerer skibets nummer.
Er der flere skibe, hvis tællere indikerer, at de skal afsendes, anvendes metoden ChooseWhich, til at bestemme hvilket af disse skibe, der skal afsendes.

I testen sættes startparametrene til 2 skibe, der begge har både pakke 1 og 5, til afsendelse, alle med lige høje prioriteter. Select skal således vælge et af disse skibe tilfældigt.
Under test af modulet blev det konkluderet, at denne metode virker efter hensigten.

SendAIS

Formålet med metoden SendAIS, er at afsende den pakke, der i metoden Select er valgt til afsendelse. I denne metode, kaldes metoderne CalcAIS og EncodeAIS, der ligger i klassen TAISSim. Disse metoder testes i afsnit 12.2.2 og udkommenteres således i testkoden. Endvidere erstattes EncodeAIS, med en Stringlist, som EncodeAIS kunne have returneret i det integrerede program. Formålet er herefter at indkapsle denne streng korrekt, alt efter om det er en pakke 1-3, eller en pakke 5, der er klippet op i flere stykker. Er den returnerede streng af den sidste pakketype, skal EncodeAIS, kunne indkapsle dataene med, antal sekvenser, sekvenssnummer og ID nummer, som det er defineret i AIS-standarden.
Er den returnerede Stringlist, på kun én sekvens, skal den afsendes på samme måde, som en sætning med flere sekvenser, men uden ID nummret.
I SendAIS, skrives disse strenge ud på serielporten. Klassen TSerielport testes selvstændigt, og af denne grund skrives resultatet ud som tekst på skærmen.
Det skal kontrolleres hvorvidt proceduren indkapsler dataene korrekt, jævnfør AIS-standarden.

I tilfælde af, at det er en pakke 1-3, der skal afsendes, er det valgt at følgende data-streng skal indkapsles:

1;o;U`Wv@>>KdOPhD6S83wp80000

En pakke 5 er sat til følgende datastrenge:

5<ene<C;McC7?18003BW=80000000000000001E?A2<W2whuPb8McTP0000000 00000000

Disse data er klippet i to dele på max 62 tegn, da indkapslingens længde, ikke må overstige 82 tegn, jf. NMEA-standarden. Dataene, der benyttes i testen, vil fra starten være klippet op i det korrekte antal strenge, idet den udkommenterede metode EncodeAIS skulle have sørget for dette.

For pakke1 er den returnerede streng:

!AIVDM,1,1,,A,1;o;U`Wv@>>KdOPhD6S83wp80000,0*61

For pakke5 er de returnerede strenge:

!AIVDM,2,1,1,A,5<ene<C;McC7?18003BW=80000000000000001E?A2<W2whuP b8McTP0000000,0*53

og

!AIVDM,2,2,1,A,00000000,0*17

Indkapslingen sker korrekt, antal sekvenser, sekvenssnummer samt ID nummer sættes korrekt af proceduren.

12.2.2 Modultest af udvalgte metoder af TAISSim

Navnet på filen til testprogrammet til denne klasse er AISTEST.exe.

CalcAIS

CalcAIS metodens formål er at beregne en ny position ud fra den foregående, fart, kurs, rotation og sendeinterval. Med disse Parametre er der lavet 4 forskellige scenarier:

1: *Datolinien*
Det skal kontrolleres, hvorvidt beregningerne tager hensyn til muligheden for at krydse datolinien.
2: *Rotation*
Det skal kontrolleres, hvorvidt beregningerne drejer skibet korrekt i forhold til rotationen.
3: *Positiv*
Det skal kontrolleres hvorvidt positionen er korrekt angivet i positiv længde og bredde.
4: *Negativ*
Det skal kontrolleres hvorvidt positionen er korrekt angivet i negativ længde/bredde.

Der sættes en timer, der kalder denne procedure med et hvis interval, og resultaterne skrives ud på skærmen. De vigtige resultater er: Længden, Bredden og Kursen.

I scenarie 1, sættes skibet til at sejle i en cirkel over datolinien, og det kan på længde-koordinaten ses, at når datolinien overskrides, skifter metoden koordinaten over til den anden side af datolinien.

I Scenarie 2, skal rotationen kontrolleres. Rotationen er sat til 127 grader/min., og intervallet er sat til 2 sek. Dermed kan grader/interval beregnes: 127/30 = 4,2333 grader/interval.
Det ses i testen, at der lægges 4,2 grader til kursen for hvert interval. Forskellen mellem den beregnede og den for koden aktuelle kurs, skyldes at koden begrænser kursen til 1/10 grader, og derfor ikke får mere end 1 decimal med.
Af dette kan konkluderes, at CalcAIS beregner kursændringen korrekt.

I Scenarie 3 kontrolleres det, hvorvidt beregningerne er korrekte i forhold til positive koordinater. Samtidig kontrolleres det, hvorvidt positionsændringen per interval er korrekt.
Følgende parametre er sat for scenarie 3:
Skibet sejler 102,4 knob ved en kurs på 45°.
Ændringen i længden per timinginterval beregnes i ligning 12.3:

$$\Delta Længde = \frac{1024 \cdot 2000 \cdot \cos\left(\frac{\left(\frac{239009775}{600000}\right) \cdot \pi}{180}\right) \cdot \cos\left(\frac{(450-45)\cdot\pi}{180}\right)}{3600} = 315,47 \, {}^{1}\!/_{\text{10000minuter}}\!/_{\text{interval}} \qquad (12.3)$$

I testen er forskellen 316. Forskellen kan igen tilskrives den egenskab, at implementationen af cosinus og sinus i Delphi, er forskellig fra implementationen på den benyttede lommeregner.
Ændringen i bredden per timinginterval beregnes i ligning 12.4

$$\Delta Bredde = \frac{1024 \cdot 2000 \cdot \sin\left(\frac{(450-45)\cdot\pi}{180}\right)}{3600} = 402,27 \, {}^{1}\!/_{\text{10000minuter}}\!/_{\text{interval}} \qquad (12.4)$$

I testen er forskellen 402.

Det er kontrolleret, at koordinaterne flytter den rigtige vej i forhold til kursen, og af dette kan det således konkluderes, at der tælles den rigtige vej.

I Scenarie 4 er det blevet kontrolleret, hvorvidt flytning af skibet fungerer på negative koordinater, hvilket er tilfældet.

Convert

Metoden Convert skal kunne konvertere et array af værdier mellem 0 og 63 (6 bit) til en streng af korrekte 8 bit ASCII-tegn.
Til testen er der benyttet de, i tabel 12.4, illustrerede arrays.

Test nr.	Tegn1	Tegn2	Tegn3	Tegn4	Tegn5	Tegn6	Tegn7	Tegn8	Tilsvarende streng
Test 1	28	43	17	8	32	63	58	20	LcA8PwrD
Test 2	63	24	34	12	34	56	45	23	wHR<RpeG

Tabel 12.4: Testarrays for konvertering til 8 bit ASCII-tegn

Ved manuel gennemgang af værdierne og den tilsvarende streng, sammenlignet med en ASCII-tabel, er det konkluderet at metoden Convert fungerer korrekt.

ASCIIConv

Denne metode konverterer navne/strenge af store bogstaver til 6 bit værdier. Store bogstavers ASCII-værdier ligger i området 65 for A til 90 for Z. Det er testet, hvorvidt metoden ASCIIConv kan konvertere disse tegn til 6 bit ASCII-værdier. Dette er gjort ved at angive strenge og den ønskede aflæste position i strengen. Det kan konkluderes, at metoden er i stand til dette.

Pak123

Metoden Pak123, kaldes af metoden EncodeAIS i det tilfælde, at det er en pakke 1-3 der skal afsendes. Formålet med metoden er at pakke variable i arrays af 6 bit værdier.
Følgende variable er sat:

MMSInr: 573503023	binær 30 bit: 1000***100010***11101111011000101111
Rotation: -127	binær 8 bit: 10000001
Fart: 83,7	binær 10 bit: ***110100***0101
Længde: -3256,8765	binær 28 bit: 11100000111100001***001000***011
Bredde: -2390,0975	binær 27 bit: 11010010011010011001101000 1
Kurs: 40,6	binær 12 bit: 001110***010110***

Følgende positioner i arrayet er kontrolleret: 3, 9, 14 og 21. De på positionen forventede bitsekvenser/tal er uddraget af tabel 9.5.
Resultaterne for de forventede og fundne bitsekvenser/tal er illustreret i tabel 12.5

Position i array	Forventet:	Fundet
Pos 3:	100010 = 34	34
Pos 9:	110100 = 52	52
Pos 14:	001000 = 8	8
Pos 21:	010110 = 22	22

Tabel 12.5: Tabeloversigt over de forventede og fundne bitsekvenser/tal.

På baggrund af testen kan det konkluderes at metoden fungerer korrekt.

Pak5

Metoden Pak5, kaldes af metoden EncodeAIS i det tilfælde, at det er en pakke 5 der skal afsendes.

Princippet i pakke5 er det samme som pak123. Resultatet af testen er, at metoden Pak5 fungerer efter hensigten.

CutStr

Metoden CutStr's input er en streng. Output skal være den samme streng, klippet op i stykker af maksimalt 62 tegn. Retur-parameteren er en StringList med de forskellige dele af den oprindelige streng.
Efter indsættelse af en streng i metoden er resultatet kontrolleret for antal tegn, og på baggrund af dette kan konkluderes, at metoden fungerer korrekt.

12.3 Brugerterminal

12.3.1 Modultest af udvalgte metoder af TSkib

Navnet på filen til testprogrammet til denne klasse er TSSkibtest.exe.

GetCourse

Metodens formål er at returnere skibets kurs, efter at have omregnet denne fra matematiske radianer til geografiske grader.
Ved test indsættes to vinkler i radianer og programmet returnerer en grad. Ved indsættelse af vinklerne 0,21 og 1,2 radianer for henholdsvis andet skib (FCourse) og eget skib (FGPSCourse), skal resultatet blive som vist i ligning 12.5.

$$450 - \frac{360}{2\pi}\left(\text{FCourse} - \text{FGPSCourse} - ½\pi\right) = 459,213 \qquad (12.5)$$

Da dette stemmer overens med programmet virker GetCourse korrekt.

CalcCourse

Metoden undersøger, hvorvidt afstanden til et andet skib er under 25 sømil. Hvis dette er tilfældet sættes kategorien til 2, ellers til 1.
For at teste CalcCourse sættes radius for grænsen mellem kategori 1 og 2 skibe til 25 sømil.
Konklusionen på testen er, at metoden fungerer efter hensigten.

CalcCoor

Denne metode omregner geografiske koordinater til et koordinatsystem med enheden meter, hvorefter koordinatsystemet roteres, så eget skib har kurs 0.
Til denne test indtastes længden og bredden for henholdsvis eget skib (FGPSLongitude og FGPSLattitude) og andet skib (FLongitude og FLattitude). Disse værdier indsættes i minutter.
Til eksemplet indsættes værdierne som vist i tabel 12.6

FLongitude	3021,33 E
FLattitude	1235,25 N
FGPSLongitude	3020,33 E
FGPSLattitude	1210,17 N

Tabel 12.6: Position for andet og eget skib

Kursen for eget skib sættes til 0°.
Med udgangspunkt i ovenstående, beregnes koordinaterne for det andet skib, efter dette er lagt ind i forhold til eget skib. Egen kurs er desuden korrigeret således, at kursen regnes med matematisk reference.
De forventede x- og y-koordinater for det andet skib beregnes i ligning C.30 og C.31 i afsnit C kolisionsberegning til henholdsvis 0,937 og 25,08 sømil inden kursen korrigeres.

Herefter bestemmes den korrigerede kurs for eget skib i ligning 12.6:

$$v = 0° - \frac{\pi}{2} = -\frac{\pi}{2} \qquad (12.6)$$

På baggrund af dette bestemmes x- og y-koordinaterne for det andet skib i ligning 12.7 og 12.8

$$x = \left(0,937\text{sm} \cdot \cos\left(-\frac{\pi}{2} \right) + 25,08\text{sm} \cdot \sin\left(-\frac{\pi}{2} \right) \right) \cdot 1852 \,\text{meter}/\text{sm} = -46448,15\text{meter} \qquad (12.7)$$

$$y = \left(-0,937\text{sm} \cdot \sin\left(-\frac{\pi}{2} \right) + 25,08\text{sm} \cdot \cos\left(-\frac{\pi}{2} \right) \right) \cdot 1852 \,\text{meter}/\text{sm} = 1736,08\text{meter} \qquad (12.8)$$

Da disse koordinater er identiske med resultatet af testen, konkluderes det, at metoden CalcCoor fungerer efter hensigten.

CalcMid

Denne metode returnerer den længde de to skibe sammenlagt kan tilbagelægge indenfor reaktionstiden plus sikkerhedsafstand og skibsradius.
Grænsen mellem et kategori 2 og 3 skib er tidligere fastsat til 4,92 sømil =9155,7 meter.
Kan de to skibe tilbagelægge denne afstand indenfor reaktionstiden, er skibet således et kategori 3 skib.
I testen tages udgangspunkt i parametrene i tabel12.7:

Parametre	Værdier
Fart(eget skib)	10knob
Reaktionstid	300sek.
Sikkerhedsafstand:	40meter
Skibsradius:	10meter
Fart(andet skib):	22knob

Tabel 12.7: Testparametre for metoden Calcmid

Disse parametre indsættes i ligning 12.9.

$$d = \left(\left(\left(v_1 + v_2 \right) \cdot \frac{1852}{60 \cdot 60} \right) t_r + \text{skibsradius+sikkerhedsafstand} \right)$$

$$\Downarrow \qquad (12.9)$$

$$d = \left(\left(\left(10\text{knob} + 22\text{knob} \right) \cdot \frac{1852}{60 \cdot 60} \right) 300\text{sek.} + 10 \text{ meter} + 40\text{meter} \right) = 4988,7\text{meter}$$

Følgende tabel, for hvilken kategori det aktuelle skib befinder sig i, kan således opstilles:

Kategori	Distance
2	x≥9155,7 meter
3	9155,7>x≥4988,7 meter
4	x<4988,7 meter

Tabel 12.8: kategoriinddelling (x= reel afstand)

Ved udførelse af testen blev de rigtige kategorier vist, hvorfor det må konstateres at metoden virker korrekt.

CalcCollision

Denne metode beregner, hvorvidt der er risiko for kollision, med et andet skib, inden for reaktionstiden.
For at kunne teste metoden opsættes tre scenarier, der undersøger, hvorvidt der er fare for kollision. Fælles for disse scenarier er at reaktionstiden er 1 minut og 40 sekunder, skibsradius er 10 meter sikkerhedsafstand er 40 meter og egen kurs er 0°.
De tre scenarier er vist i tabel 12.9:

Scenarie	FCoorX	FCoorY	FCourse	FSpeed	FGPSSpeed	Alarm
1	555,6 meter	740,8 meter	270 grader	9,8 knob	13,4 knob	Nej
2	555,6 meter	740,8 meter	270 grader	10 knob	13,6 knob	Nej
3	555,6 meter	740,8 meter	270 grader	10,8 knob	14,4 knob	Ja

Tabel 12.9: Testscenarier

Det forventede resultat for scenarie 1 er beregnet i det følgende:

Den tilbagelagte afstand for eget skib efter 100 sek. inklusiv skibsradius og sikkerhedsafstand er beregnet i ligning 12.10.

$$d = \left(\left(13,4\text{knob} \cdot \frac{1852}{3600} \right) \cdot 100\text{sek} \right) + \left(10 + 40\right)\text{meter} = 748\text{meter} \qquad (12.10)$$

Den tilbagelagte afstand for andet skib efter 100 sek. er beregnet i ligning 12.11.

$$d = \left(9,8\text{knob} \cdot \frac{1852}{3600} \right) \cdot 100\text{sek} = 504,16\text{meter} \qquad (12.11)$$

På baggrund af ligning 12.10 kan det konkluderes, at der er tale om en kollision i y-retningen. Til gengæld viser ligning12.11, at der ikke er kollision i x-retningen, hvorfor der ikke skal alarmeres.
Beregningerne for de øvrige scenarier er foretaget på tilsvarende måde.
Da det forventede resultat er identisk med resultatet af testen, må det konkluderes, at metoden virker korrekt.
Vælges det i testen, selv at indtaste kurs for andet skib, foretages beregningerne på baggrund af følgende kurs: ((450-indtastet kurs)*2π / 360), da der benyttes matematiske radian tal.

12.3.2 Modultest af udvalgte metoder af TGPS og TAIS

Navnet på filen til testprogrammet til denne klasse er TAISTGPStest.exe.

GPSCut

Formålet med metoden GPSCut er at opdele data fra bufferen i GPS pakker, da indholdet fra bufferen ikke nødvendigvis er fuldstændige GPS pakker.
Metoden skal dele en GPS-streng op i flere strenge. Disse strenge skal starte med $ og slutte med hex-værdien #$0A (LF). Ved test af metoden indsættes følgende data fra bufferen:

'streng1' + #$0D + #$0A + '$,streng2' + #$0D + #$0A + '$,streng3' + #$0D + #$0A + '$,streng4' + #$0D.

Af dette ses, at resultatet skulle blive to strenge med indholdet:
$,streng2 <CR> og $,streng3 <CR>.
Ved test af programmet blev resultatet som det forventede.

DecodeGPS

DecodeGPS har til formål at hente oplysninger ud af variablen GPSData, og lægge dem i de korrekte variable. For at kontrollere, hvorvidt resultatet er korrekt indsættes følgende GPS streng:

$GPRMC,162256.00,A,3335.10132,N,05010.20149,E,001.00,034.00,121202,0.0,E,A*38

Resultatet af dekodningen skal således være identisk med indholdet af tabel 12.10:

Parametre	Værdi
UTC	162256.00
Status	A
Bredde	33 35.10132
N/S	N
Længde	050 10.20149
Ø/V	E
Fart	1
Kurs	34
Dato	121202

Tabel 12.10: Testparametre for metoden DecodeGPS

Af testprogrammet fremgår det, at strengen er dekodet rigtigt, idet alle data ligger i deres respektive variable. Af dette kan konkluderes, at metoden fungerer korrekt.

AISCut

Formålet med metoden AISCut er, at opdele data fra bufferen i AIS-pakker, idet indholdet fra bufferen ikke nødvendigvis er fuldstændige AIS-pakker.
Metoden skal dele en AIS-streng op i flere strenge, og disse strenge skal starte med ! og slutte med hex-værdien #$0A. For at gøre testen mere overskuelig laves testen på en sådan måde, at resultatet skal blive det samme som i GPSCut, med undtagelse af starttegnet. Ved test af metoden indsættes følgende data fra bufferen:

'streng1' + #$0D + #$0A + '!,streng2' + #$0D + #$0A + '!,streng3' + #$0D + #$0A + '!,streng4' + #$0D.

Af dette ses, at resultatet skulle blive to strenge med indholdet:
!,streng2 <CR> og !,streng3 <CR>
Ved test af programmet blev resultatet som det forventede.

AISSort

Formålet med metoden AISSort er, at sortere de sekvenser der kommer fra AISCut. AISSort starter med at beregne checksummen, hvorefter den kontrollerer hvor mange sekvenser, der er i den aktuelle sætning.
Metoden sorterer AIS-strengene og sætter sekvenser sammen, hvis alle sekvenser der har samme sekventielle ID er til stede. I testen tages udgangspunkt i sekvenserne i tabel 12.11. Disse sekvenser er lavet ud fra følgende sætnig:

!AIVDM,x,y,z,a,s—s,f*hh<CR><LF>

Ved første test indsættes 5 strenge med samme ID nr., for at se om metoden indsætter data i rigtig rækkefølge, alt efter hvilken sekvens data ligger i.

Antal Sekvenser	Sekvens nr.	ID	Streng
5	2	7	!AIVDM,5,2,7,a,2:,0*24
5	5	7	!AIVDM,5,5,7,a,5,0*24
5	1	7	!AIVDM,5,1,7,a,1:,0*24
5	4	7	!AIVDM,5,4,7,a,4:,0*24
5	3	7	!AIVDM,5,3,7,a,3:,0*24

Tabel 12.11: AIS-information indeholdende fem sekvenser

Resultatet, ved indsættelse af disse strenge, forventes at give følgende rækkefølge af datadelen: 1:2:3:4:5.
Det forventede resultat er identisk med resultatet af testen.

Næste test har til hensigt at vise, hvorvidt metoden undlader sætninger der ikke er komplette. Dette gøres ved at tage udgangspunkt i strengene i tabel 12.12. Tre af disse strenge danner en sætning, i modsætning til de øvrige, der skal gemmes i bufferen.

Antal Sekvenser	Sekvens nr.	ID	Streng
3	2	6	!AIVDM,3,2,6,a,6:2;,0*24
4	1	5	!AIVDM,4,1,5,a,5:1;,0*24
3	3	6	!AIVDM,3,3,6,a,6:3;,0*24
2	2	2	!AIVDM,2,2,2,a,2:2;,0*24
3	1	6	!AIVDM,3,1,6,a,6:1;,0*24
4	4	5	!AIVDM,4,4,5,a,5:4;,0*24

Tabel 12.12: AIS-informationer med forskelligt ID

Da samtlige sekvenser af sætningen med ID nummer 6 er til stede, er det forventede resultat, at denne sætning skrives ud i boksen ”Hele sætninger”.
Resten af sekvenserne skal gemmes i bufferen, og i testen skrives ud i boksen ”Buffer”.
Testen viser, at det forventede resultat er identisk med resultatet af testen.

Den sidste del af testen skal vise, at metoden AISSort kan tage sekvenser beliggende i bufferen og sætte disse sammen med de nye sekvenser.
Til at teste dette tages udgangspunkt i strengene i tabel 12.12. Testen udføres i to etaper. I den første del udføres metoden AISSort med de to første sekvenser. Da de to første strenge ikke udgør en komplet sætning, skal disse strenge lægges i bufferen, og skrives ud i boksen ”Buffer”.
Derefter køres programmet med de resterende sekvenser, og resultatet skrives ud som i test 2.
Efter denne test kan det konkluderes, at metoden AISSort fungerer efter hensigten.

ASCIIConv

Formålet med metoden ASCIIConv er at konvertere 8 bit ASCIIværdier til 6 bit værdier.
Til at teste, hvorvidt dette gøres korrekt bruges de samme strenge, der blev genereret i metoderne Pak123 og Pak5 i klassen TAISSim. Strengen genereret af metoden Pak123 er:

18Rsn;j1lGhN58NTlkA6Fwr`0000

Resultatet af konverteringen bliver derpå:

1;8;34;59;54;11;50;1;52;23;48;30;5;8;30;36;53;51;17;6;22;63;58;40;0;0;0;0;

Strengen genereret af metoden Pak5 er:
58Rsn;jj0:066:=FBFn7:Frn7:VFOG:jVnGCBGtwwwww2whu2`st9Kb8IqK`st9Kb8IqKP

Resultatet af denne konvertering bliver:
5;8;34;59;54;11;50;50;0;10;0;6;6;10;13;22;18;22;54;7;10;22;58;54;7;10;38;22;31;23;10;50;38;54;
23;19;18;23;60;63;63;63;63;63;2;63;48;61;2;40;59;60;9;27;42;8;52;57;27;40;59;60;9;27;42;8;52;57;27;32;

I TAISSim blev der taget følgende stikprøver:
Pak123: [3]=34, [9]=52, [14]=8 og [21]=22. Da dette er i overensstemmelse med det forventede, må det konstateres, at metoden virker efter hensigten.

Test af Pak5 er foretaget på tilsvarende måde med tilsvarende forventet resultat.

Decode123

Metoden Decode123 har til formål at hente, de data metoden ASCIIConv har konverteret.
For at undersøge om Decode123 dekoder AIS-strengen korrekt, undersøges, som i metoden ASCIIConv, den kendte streng fra metoden Pak123, i klassen TAISSim.
AIS-informationerne der blev pakket af Pak123 er vist i tabel 12.13:

Parametre	Værdier
MMSI	573503023
NavState	0
Rotation	-127
Speed	83,7 (837/10)
Course	40,6 (406/10)
Longitude	-3256,8765 (-325687650/10000)
Latitude	-2389,6879 (-238968790/10000)

Tabel 12.13: Testparametre for metoden Decode123

Efter gennemførelse af testen kan det konstateres at metoden virker korrekt. Det er således de ønskede AIS-informationer der bliver dekodet.

Decode5

Metoden Decode5 har, i lighed med metoden Decode123, til formål at hente, de data metoden ASCIIConv har konverteret.
For at undersøge, hvorvidt Decode5 dekoder AIS-strengen korrekt, undersøges, som i metoden ASCIIConv, den kendte streng fra metoden Pak5, i klassen TAISSim.
AIS-informationerne, der blev pakket af Pak5 er vist i tabel 12.14:

Parametre	Værdier
MMSI	573503026
IMO	746596353
GNSSA	511
GNSSB	511
GNSSC	63
GNSSD	63
Draught	10
CallSign	ABC5DE
Name	MARENMARIEGURLIMETTE
Destination	COPENHAGENCOPENHAGEN

Tabel 12.14: Testparametre for metoden Decode5

Efter gennemførelse af testen kan det konstateres at metoden virker korrekt. Det er således de ønskede AIS-informationer der bliver dekodet

DecodeAIS

DecodeAIS skal bestemme hvilken pakketype der skal dekodes, og på baggrund af dette kalde metoden Decode123 eller decode5. I testen skrives den metode ud der bliver kaldt for følgende fire sætninger:

1<ene<Gt?R>KG?`hBJg3SwwH0000,
2>SpEhGuAf>PQi@hCEw6OwtP0000,
36;LjH7t0v>R1A@hElncEwwH0000 og
5=L80pGw@r>Q1t0hEpe2vwtP0000

Det forventede resultat er, at metoden Encode123 bliver kaldt i de tre første sætninger, medens Decode5 skal i den sidste sætning. Da dette også er resultatet af testen, må det konkluderes at metoden virker korrekt.

13 Procesintegrationstest

Procesintegrationstesten udføres efter modulintegrationstesten er afviklet, idet denne test tester funktionalitet af de sammenkoblede processor.
Dette er gjort ved at teste udvalgte klasser benyttet i henholdsvis GPS-simulatoren, AIS-simulatoren, Brugerterminalen og Databasen..
For hver klasse er der lavet et testprogram til test af den aktuelle klasse. Disse testprogrammer er placeret på den medfølgende CD i mappen "Test".

13.1 GPS-simulator

13.1.1 Klassen TGPSSim

Navnet på filen til testprogrammet af denne klasse er GPSSimulatorTest.exe

Klassen TGPSSim har til formål at beregne og afsende GPS-signaler.
For at teste klassen TGPSSim, skal denne inkluderes i en anden klasse, der kan tilgå kommandoerne i TGPSSim og anvende disse. Den kode, der er skrevet til den færdige GPS-simulator, udfører allerede denne opgave, hvorfor det er valgt at benytte den allerede eksisterende kode i testen
Til testen er der valgt følgende scenarier:

- Test af datolinien og rotation.
- Test af koordinatændring i forhold til. kurs og fart.

Ved test af datolinien og rotation sættes et scenarie op på en sådan måde, at skibet har kurs mod datolinien og startpositionen er tæt på datolinien. I dette scenarie sættes skibet samtidig til at rotere. Dette er valgt af to grunde, for det første for at kontrollere hvorvidt rotationen af skibet udføres korrekt af simulatoren. For det andet vil rotationen af skibet medføre, at det vil sejle i en cirkel, der krydser datolinien to steder. Det kan således påvises, hvorvidt korrigeringen, i forhold til datolinien, udføres korrekt af simulatoren i begge retninger.

Til test af datolinien og rotation blev testparametrene i tabel 13.1 benyttet:

Parametre	Værdier
Længde	10798,45 min.
Bredde	1203,65 min.
Kurs	90°
Rotation	23 grader/min.
Fart	30 knob.

Tabel 13.1: : Testparametre for klassen TGPSSim

GPS-Simulatoren sættes i dette scenarie til at sende data med intervallet 1 sek.
Til kontrol af resultatet sættes en PC, med hyperterminal, til at "optage" aktiviteten på serieludgangen, fra GPS-Simulatoren. Derved fremkommer følgende data:

$GPRMC,092139.00,A,2003.64689,N,17959.97803,E,030.00,093.**83**,161202,0.0,E*33
$GPRMC,092140.00,A,2003.64632,N,17959.98627,E,030.00,094.**22**,161202,0.0,E*36
$GPRMC,092141.00,A,2003.64563,N,**17959.99359,E**,030.00,094.**60**,161202,0.0,E*3B

$GPRMC,092142.00,A,2003.64494,N,**17959.99817,W**,030.00,094.**98**,161202,0.0,E*25
$GPRMC,092143.00,A,2003.64414,N,17959.99084,W,030.00,095.**37**,161202,0.0,E*2A
$GPRMC,092144.00,A,2003.64334,N,17959.98260,W,030.00,095.**75**,161202,0.0,E*27
…..
…..
$GPRMC,092903.00,A,2001.16604,N,17959.97894,W,030.00,263.**65**,161202,0.0,E*29
$GPRMC,092904.00,A,2001.16524,N,17959.98627,W,030.00,264.**03**,161202,0.0,E*21
$GPRMC,092905.00,A,2001.16444,N,**17959.99451,W**,030.00,264.**42**,161202,0.0,E*20
$GPRMC,092906.00,A,2001.16364,N,**17959.99817,E**,030.00,264.**80**,161202,0.0,E*34
$GPRMC,092907.00,A,2001.16295,N,17959.98993,E,030.00,265.**18**,161202,0.0,E*36
$GPRMC,092908.00,A,2001.16226,N,17959.98260,E,030.00,265.**57**,161202,0.0,E*3D

Af strengene ses det, at GPS-simulatoren tager hensyn til datolinien begge veje.
Det kan endvidere kontrolleres hvorvidt rotationen er korrekt.
Den forventede rotation beregnes i ligning 13.1:

$$\frac{23 grader / \min}{60 sek. / \min} = 0,38333 \, grader / sekund \qquad (13.1)$$

I GPS-strengene kan det konstateres, at differencen i kursen pr interval, varierer mellem 0,38 og 0,39° per sek.
Det kan således konkluderes, at GPS-simulatoren tager hensyn til datolinien, og beregner kursændring korrekt.

Test af koordinatændring i forhold til kurs og fart.

Formålet med denne test er, at fastslå hvorvidt beregningen af positionsændring pr. tidsinterval fungerer korrekt.
Til dette tages udgangspunkt i testparametrene i tabel 13.2 for eget skib:

Parametre	Værdier
Længde	3798,45 min. = 63° 18′45Ø
Bredde	1203,65 min. = 20° 03′65N
Kurs	60°
Rotation	0 grader/min.
Fart	30 knob.
Tidsinterval	2 sek.

Tabel 13.2: Testparametre for klassen TGPSSim

Den beregnede ændring i længde per tidsinterval beregnes i ligning 13.2.

$$\Delta\text{længde} = \frac{30\text{knob} \cdot 2000\text{ms} \cdot \cos\left(2,5\pi - 1/3\,\pi\right) \cdot \cos\left(\frac{\left(\frac{1203,65\,\text{min.}}{60}\right) \cdot \pi}{180}\right)}{3600000} \qquad (13.2)$$

$$\Downarrow$$

$$\Delta\text{længde} = 0,0135537199 \, \text{min.} / \text{interval}$$

På tilsvarende måde beregnes ændring i bredden per tidsinterval i ligning13.3.

$$\Delta\text{Bredde} = \frac{30\text{knob} \cdot 2000\text{ms} \cdot \sin\left(2,5\pi - \frac{1}{3}\pi\right)}{3600000} = 0,00833333\,\text{min}/\text{interval} \qquad (13.3))$$

Følgende streng er optaget med Windows hyperterminal:

$GPRMC,095904.00,A,**2003.65833**,N,**06318.46367**,E,030.00,060.00,161202,0.0,E*31

Forskellen i bredde og længde kan derefter beregnes:

Forskellen i bredde: 2003,65833-2003,65 = 0,00833 min., hvor det forventede var 0,00833.
Forskellen i længde: 6318,46367-6318,45 = 0,01367 min., hvor det forventede var 0,01355.

I beregningerne af længden kan det konstateres, at der er afvigelser mellem det forventede og resultatet af testen.
Denne fejl tilskrives forskellig implementation af sinus og cosinus i henholdsvis Delphi og den benyttede lommeregner. Da resultatet af ligning 13.2 er resultatet af produktet af to cosinus-faktorer sættes fejlen i anden.

På baggrund af de to tests konkluderes det, at klassen TGPSSim og GPS-simulatoren fungerer korrekt.

13.2 AIS-simulator

13.2.1 Klassen TAISSim

Navnet på filen til testprogrammet af denne klasse er TAISKlassetest.exe

Klassen TAISSim har til formål at beregne og pakke AIS-informationer for et enkelt skib, og derpå returnere dette resultat til klassen, der har kaldt på data fra TAISSim.
TAISSim klassen skal ikke afsende dataene, kun pakke dem. Disse data indkapsles i klassen TAISAdm, hvorfra de efterfølgende afsendes.
I denne test inkluderes TAISSim-klassen i en ny GUI. Denne GUI går således ind og overtager AISAdm-klassens placering. I stedet for at sende data ud på serielporten, skrives det, i testen, ud på skærmen.
I testen får brugeren af testprogrammet mulighed for manuelt, at indtaste data, til klassen TAISSim. Desuden er der mulighed for at indlæse de forudbestemte testparametre i tabel 13.3:

Parametre	Værdier
Callsign	Bubber
Kurs	92,3°
Destination	badekarsfabrikken
Draught	21meter
GNSS	32
IMO	674908263
Latitude	3917,2509 min.
Longitude	1244,5232 min.
MMSI	987145236
Name	Bubbers badebaad
Navstate	1
NavStype	1
Rotation	13° per min.
Sctype	12
Speed	23,4knob
Opdaterings Interval	2000ms

Tabel 13.3: Testparametre for klassen TAISSim

Ovenstående værdier anvendes til denne test:
Der er lavet 2 timere, der hver opdaterer pakke 1 og pakke 5. Når timererne kalder metoden EncodeAIS i TAISSim-klassen, returnerer denne metode den data der skal returneres. Denne data skrives ud på skærmen, hvorved følgende pakker er fremkommet for henholdsvis en pakke 1 og 5:

1>eJT54=>b5sk<jEKVG>0wuP0000

5>eJT52PrB6N;F::G:;F::G;<2:6BF:66@0000<40PPP2whul``I9JpLdq`Hdb
JrqKP000

Den første pakke kontrolleres på samme måde som i modultesten af metoden Pak123 (se afsnit 12.2.2).

MMSInr:	987145236	Binært:	1110***101101***011010***100100***00010100
Fart:	234	Binært:	***001110***1010

De markerede binære værdier svarer til positionerne: 3, 5, 9.
I pakke 1 strengen svarer disse positioner til bogstaverne: e, T, >.

Det testes ikke for længden, bredden og kursen i pakken, da disse er opdateret til nye værdier på det tidspunkt, TAISSim returnerer strengen. Det vil derfor ikke være muligt at føre kontrol af disse værdier, i forhold til det oprindeligt indtastede.

Konverteres de valgte tre bogstaver til 6-bit-værdier, fås følgende værdier:

ASCII-værdi af e: 101

101+40 er større end 128, derfor adderes 32, jævnfør [AIS2001 s.112]. e's 6-bit værdi: 6 LSB af 173 (101+40+32) = 10***101101***, og dermed er dette resultat korrekt.

ASCII-værdi af T: 84

84+40 er mindre end 128, derfor adderes 40, jævnfør [AIS2001 s.112]
T's 6-bit værdi: 6 LSB af 164 (84+40+40) = 10***100100***, og dermed er dette resultat korrekt.

ASCII-værdi af >: 62

62+40 er mindre end 128, derfor adderes 40, jævnfør [AIS2001 s.112]
T's 6-bit værdi: 6 LSB af 142 (62+40+40) = 10***001110***, og dermed er dette resultat korrekt.

En tilsvarende test udføres for pakke 5. Det er valgt at teste position 8, 20 og 62.

Position 8 svarer til bit 3-8 i IMO nummeret:
IMO: 674908263 I binært: 10***100000***111010010010000011001111

Position 8 i pakke 5 strengen er P.

ASCII-værdi af P: 80

80+40 er mindre end 128, derfor adderes 40, jævnfør [AIS2001 s.112]
T's 6-bit værdi: 6 LSB af 160 (80+40+40) = 10***100000***, og dermed er dette resultat korrekt.

Position 20 svarer til 4 LSB af tegn 2, samt 2 MSB af tegn 3 i Skibsnavnet.
Tegn 2 og 3 i skibsnavnet er U og B (store bogstaver, idet navnet konverteres til store bogstaver inden navnet pakkes jf. [AIS2001]).
Da området begrænses til 6 bit anvendes ASCII-værdierne mellem 32 og 95 da dette giver 64 forskellige tegn. I det nævnte område er der kun store bogstaver, derfor begrænses navnene til at benytte store bogstaver.
Da værdierne i dette område er lineære, kan konverteringen foregå ved at trække 32 fra ASCII-værdien, som foreskrevet i AIS-standarden.

U's 6-bit værdi:
ASCII-værdi af U: 85

85-32 = 53 = 11***0101***

B's 6-bit værdi:
ASCII-værdi af B: 66

66-32 = 34 = ***10***0010

Dermed skal værdien af tegn nummer 20 i pakke 5 stregen være 010110 = 22.

Position 20 i Pakke 5 strengen er tegnet F. Da hele ASCII-området i af/indkodningen anvendes konverteres tegnet jf. [AIS2001 s.112].

F's 6-bit værdi:
ASCII-værdi af F: 70

70+40 er mindre end 128, derfor adderes 40,
T's 6-bit værdi: 6 LSB af 150 (70+40+40) = 10***010110***.

Det kan hermed konkluderes at dette tegn er afsendt korrekt.

Position 62 i pakke 5 består af 2 LSB af tegn 12 og 4 MSB af tegn 13 i destinationen. Tegn 12 og 13 er henholdsvis R (ASCII: 82) og I (ASCII: 73).

R: 82-32 = 50 = 1100***10***

I: 73-32 = 41 = ***1010***01

Den konverterede værdi af tegn 62 i pakke 5 skal dermed være **101010** = 42. Tegn 62 i pakke 5 er b.

ASCII-værdi af b: 98

98+40 er større end 128, hvorfor der adderes med 32.
6 LSB af 170 (98+40+32) = 10***101010***

Det kan hermed konkluderes at dette tegn er afsendt korrekt.

Det ovenstående beviser dog ikke, hvorvidt alle de pakkede værdier i pakke 1 og 5 er korrekte. Af denne grund vil de ovenstående strenge blive anvendt i klassetesten af klassen TAIS (se afsnit 13.3.4), der modtager disse data og afkoder dem.

13.3 Brugerterminal

13.3.1 Klassen TSkib

Navnet på filen til testprogrammet af denne klasse er TSkibklassetest.exe

Klassen TSkibs formål er at beregne kollisionsfare for et enkelt skib i forhold til eget. Dette gøres ved at indsætte data for eget og andet skib.
Til hele testen tages udgangspunkt i parametrene i tabel 13.4 for eget skib:

Parametre	Værdier
Reaktionstid	100sek.
Kurs	0°
Længde	3020,33 min.
Bredde	1210,17 min.

Tabel 13.4: Testparamatre for klassen TSkib eget skib

Farten for eget skib varierer fra test til test.

Fælles for alle andre skibe er testparametrene i tabel 13.5:

Paramtre	Værdier
SCType	1
NavSType	1
NavState	1
Shipradius	10
SecRadius	40
MMSI	123456789
IMO	987654321
Destination	'HAVNEBY'
Rotation	12
Name	'ISABELLA'
CallSign	'A1B2C3'
Draught	8

Tabel 13.5: Testparamatre for klassen TSkib andre skibe

Test1

Test1 tager udgangspunkt i scenarie 1 i tabel 12.9. Formålet med testen er at vise, hvorvidt kollisionsberegningerne er foretaget korrekt.
I scenarie 1 i tabel 12.9 er koordinaterne for andet skib i forhold til eget skib givet således: (555,6;740,8).
Disse koordinater regnes først tilbage, til den reelle position, for det andet skib. Herefter benyttes dette resultat til at kontrollere, hvorvidt kollisionsberegningerne er foretaget som forventet.

Først beregnes y-koordinaten for afstanden mellem eget og andet skib i ligning 13.4

$$y = 740,8\,\text{meter} = \frac{740,8\text{meter}}{1852} = 0,4\,\text{sm} \qquad (13.4)$$

Herefter omregnes denne afstand til et koordinat, der angiver det andet skibs placering i forhold til eget skibs placering, i y-retningen. Dette er beregnet i ligning 13.5. Dette gøres for kunne følge de beregninger, der er gennemgået i appendiks C, omhandlende kollisionsberegninger.

$$\begin{gathered} 0,4\text{ sm} = y - 1210,17\text{minutter} \\ \Downarrow \\ y = 1210,57\text{minutter} = 20,1762° \end{gathered} \qquad (13.5)$$

Herefter beregnes x-koordinaten for det andet skib i forhold til eget skib på tilsvarende måde, hvilket er gjort i ligning 13.6 og 13.7.

$$x = 555,6\,\text{meter} = \frac{555,6\text{meter}}{1852} = 0,3\,\text{sm} \qquad (13.6)$$

$$0{,}3\text{sm}=\text{Cos}\left(\frac{20{,}1695°+20{,}1762°}{2}\right)\cdot\left(\text{x}-3020{,}33\text{minutter}\right)$$
$$\Downarrow \qquad (13.7)$$
$$\text{x}=3020{,}6496\text{minutter}$$

Data for andet skib er angivet i tabel 13.6:

Parametre	Værdier
Bredde	1210,57 min.
Længde	3020,65 min.
Fart	9,8knob
Kurs	270°

Tabel 13.6: Testparametre for klassen TSkib

I testen tages udgangspunkt i disse data for andet skib. Resultatet bliver således, at alarmen ikke skal aktiveres. Desuden kan det konstateres, at koordinaterne for det andet skib er som forventet jf. scenarie 1 i tabel 12.9.
Resultatet er testen er således i overensstemmelse med det forventede.

Test2

Denne test undersøger det samme som test1, blot med den undtagelse at der nu benyttes værdierne for scenarie 3 i tabel 12.9. Resultatet af testen skal derfor være identisk med resultatet af test 1, blot med den undtagelse, at alarmen skal aktiveres, hvilket testen efterviser.

Test3

Denne test skal undersøge, hvorvidt et kategori 3 skib bliver behandlet korrekt.
Med udgangspunkt i tabel 12.8 og metoden CalcMid vælges afstanden til andet skib til 6000 meter. X-koordinaten sættes til 1700 meter, hvorfor y-koordinaten kan beregnes ud fra ligning 13.8:

$$\text{y}=\sqrt{6000^2\text{meter}-1700^2\,\text{meter}}=5756{,}13\text{meter}=3{,}1081\text{sm} \qquad (13.8)$$

Herefter omregnes dette koordinat til et koordinat, der angiver det andet skibs placering i forhold til eget skibs placering, i y-retningen. Dette er beregnet i ligning 13.9

$$3{,}1081\text{sm}=\text{y}-1210{,}17\text{minutter}$$
$$\Downarrow \qquad (13.9)$$
$$\text{y}=1213{,}2781\text{minutter}=20{,}2213°$$

Herefter beregnes x-koordinaten for det andet skib, i forhold til eget skib,på tilsvarende måde, hvilket er gjort i ligning 13.10 og 13.11.

$$\text{x}=1700\,\text{meter}=\frac{1700\text{meter}}{1852}=0{,}917934\,\text{sm} \qquad (13.10)$$

$$0{,}91793=\text{Cos}\left(\frac{20{,}1695°+20{,}2213°}{2}\right)\cdot\left(x-3020{,}33\text{minutter}\right)$$
$$\Downarrow \qquad (13.11)$$
$$x=3021{,}3081\text{minutter}$$

Med udgangspunkt i de beregnede værdier, kan det konkluderes, at testprogrammet kategoriserer det andet skib korrekt. Endvidere kan det konstateres, at koordinaterne for det andet skib er som forventet.

Test4

Denne test skal undersøge, hvorvidt et kategori 2 skib bliver behandlet korrekt.
Med udgangspunkt i tabel 12.8, og metoden CalcMid, vælges afstanden til andet skib til 15000 meter. X-koordinaten sættes igen til 1700 meter, hvorfor y-koordinaten kan beregnes i ligning 13.12:

$$y=\sqrt{15000^2\,\text{meter}-1700^2\,\text{meter}}=14903{,}36\text{meter}=8{,}05\text{sm} \qquad (13.12)$$

Beregningerne foretages efterfølgende i lighed med de øvrige tests . Resultatet af beregningerne giver således følgende reelle koordinatsæt for bredde og længde: (1218,2172 , 3021,3083).
Med udgangspunkt i de beregnede værdier, kan det konkluderes at testprogrammet kategoriserer det andet skib korrekt. Endvidere kan det konstateres, at koordinaterne for det andet skib er som forventet.

Test 5

Den sidste test, test5, skal undersøge hvorvidt klassen behandler et kategori 1 skib korrekt.
Grænsen for afstanden til et kategori 1 skib er 25 sømil svarende til 46300 meter. Afstanden til et andet skib vælges af denne grund til 50000 meter.
X-koordinaten sættes igen til 1700 meter, hvorfor y-koordinaten kan beregnes i ligning 13.13:

$$y=\sqrt{50000^2\,\text{meter}-1700^2\,\text{meter}}=49971{,}09\text{meter}=26{,}98\text{sm} \qquad (13.13)$$

Beregningerne foretages efterfølgende i lighed med de øvrige tests . Resultatet af beregningerne giver således følgende reelle koordinatsæt for bredde og længde: (1237,1522 , 3021,3093).
Med udgangspunkt i de beregnede værdier, kan det konkluderes at testprogrammet kategoriserer det andet skib korrekt. Endvidere kan det konstateres, at koordinaterne for det andet skib er som forventet.
Det kan således konstateres at der i testen af klassen TSkib ikke kan konstateres nogen fejl.

13.3.2 Klassen TAlarm

Navnet på filen til testprogrammet af denne klasse er TAlarmklassetest.exe

Klassen TAlarm har til formål at administrere alarmen, på baggrund af properties'ene IsActive og Enabled. IsActive sættes af brugeren vha. brugerfladen på knappen "deaktiver", medens Enabled sættes når metoden IsCrashing, i klassen TSkib returnerer værdien true.
For at den auditive alarm skal starte, skal begge properties'ene være true.
Har brugeren sat IsActive false, eller returnerer metoden IsCrashing værdien false, skal den auditive alarm ikke starte.
Dette er kontrolleret i testprogrammet og resultatet er i overensstemmelse med forventede.

13.3.3 Klassen TSerielPort

Navnet på filen til testprogrammet af denne klasse er TSerielklassetest.exe

Klassen TSerielPort er opbygget, på en sådan måde, at der med simple læse og skrive kommandoer kan læses og skrives til serielporten på en PC.
Testen af klassen er foregået på følgende måde:
På én computer åbnes Windows hyperterminal med parametrene: 4800 bps, 8 databit, ingen paritet, 1 stopbit og ingen flowstyring.

Klassen TSeriel indstilles til de samme parametre.
Læse og skrive kommandoerne for klassen anvendes nu til at sende og modtage beskeder fra hyperterminalen.

Denne test har vist, at Serielport-klassen fungerer korrekt, da det med ovenstående fremgangsmåde er lykkedes at kommunikere begge veje med serielport-klassen.

13.3.4 Klasserne TGPS og TAIS

Navnet på filen til testprogrammet af denne klasse er TAISTGPSKlassetest..exe

Klasserne TGPS og TAIS testes, ved at sende kendte GPS- og AIS-informationer, fra de to simulatorer, gennem serielportene, til brugerterminalen. Disse informationer dekodes i klasserne TGPS og TAIS i brugerterminalen, hvorpå resultatet af dette sammenlignes med de afsendte informationer.
Da de afsendte informationer, er identiske med de modtagne dekodede informationer, kan det konstateres at klasserne TGPS og TAIS virker efter hensigten.
På baggrund af dette, må det tillige konkluderes, at klasserne TAISSim og TGPSSim fungerer korrekt.

13.4 Databasen

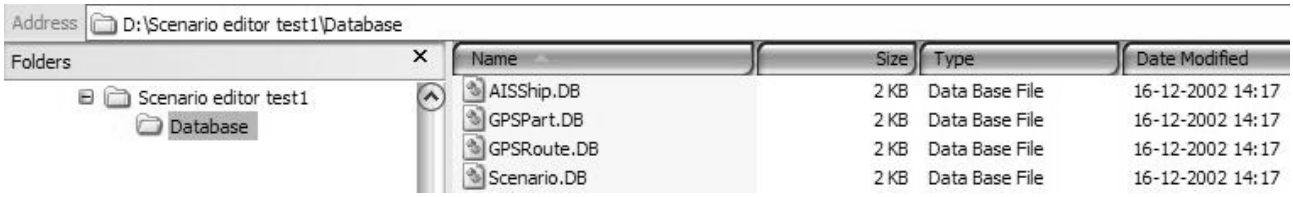
13.4.1 Klasserne TDataAccess, TAISDataAccess og TGPSDataAccess

Navnet på filen til testprogrammet af disse klasser er ScenarioEditProj..exe

Klasserne har til formål at fungere som interface til en database, hvor simulationsparametre til AIS- og GPS-simulatorerne lagres.
Da disse klasser er meget tæt forbundne, er de testet i sammenhæng. Til test af klasserne, er der lavet et program, en Scenario Editor med filnavnet ScenarioEdiProj.exe, der ved hjælp af disse klasser kan oprette og modificere scenarier. Testene er udført ved at definere scenarier i Scenario Editoren, hvorefter resultatet kontrolleres, ved at åbne tabellerne i MS Access.

Test 1: Oprettelse af database

Testen udføres ved at starte Scenario Editoren i et tomt bibliotek. Ved opstart af programmet oprettes objekter af klasserne TAISDataAccess og TGPSDataAccess. Disse skal, hvis alt fungerer som det skal, oprette et underbibliotek med de fire tabeller, der er specificeret i afsnit 10.1. Tabellernes filer kan ses i figur 13.1.



Figur 13.1: Filer til tabellerne i databasen

Ved at åbne tabellerne i Access ses, at tabellernes specifikationer passer til det tidligere beskrevne. Tabellerne ses i figurerne 13.2, 13.3, 13.4 og 13.5.

Scenario : Table

Field Name	Data Type
ScenarioID	AutoNumber
ScenarioName	Text

Figur 13.2: Aflæsning af attributterne fra Scenario

AISShip : Table

Field Name	Data Type
ShipID	AutoNumber
ScenarioID	Number
MMSI	Number
NavState	Number
IMO	Number
CallSign	Text
Name	Text
SCType	Number
GNSSPosA	Number
GNSSPosB	Number
GNSSPosC	Number
GNSSPosD	Number
NavSType	Number
Draught	Number
Destination	Text
Rotation	Number
Speed	Number
Latitude	Number
Longitude	Number
Course	Number

Figur 13.3: Aflæsning af attributterne fra AISShip

GPSRoute : Table

Field Name	Data Type
RouteID	AutoNumber
ScenarioID	Number
Longitude	Number
Latitude	Number
Course	Number

Figur 13.4: Aflæsning af attributterne fra GPSRoute

GPSPart : Table

Field Name	Data Type
CourseID	AutoNumber
RouteID	Number
Rotation	Number
Speed	Number
ChangeTime	Number

Figur 13.5: Aflæsning af attributterne fra GPSPart

Test 2: Oprettelse og modificering af scenarie

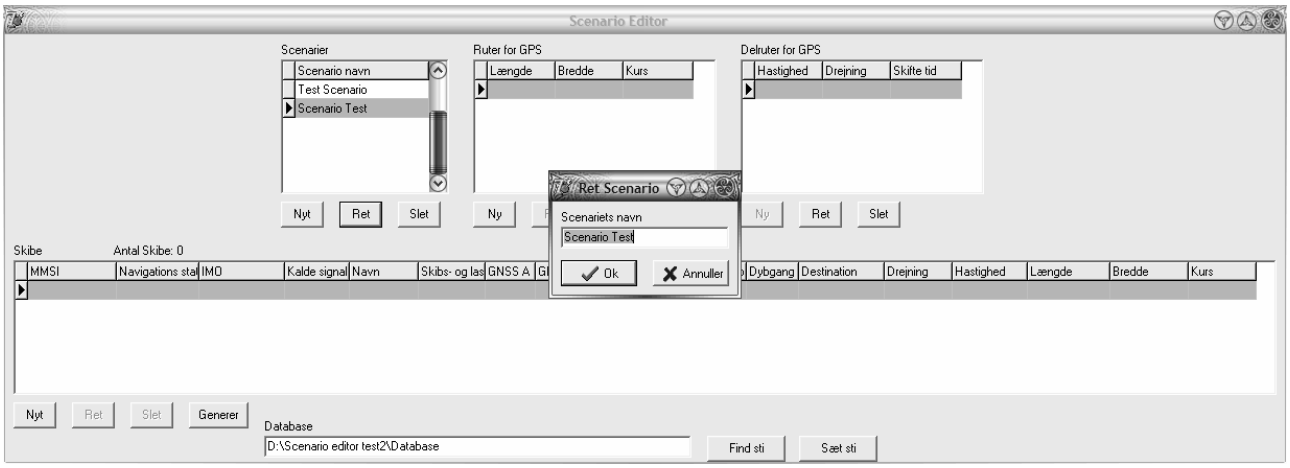
Testen udføres ved at klikke på knappen ”Nyt” under listen med scenarier. Et vindue herved komme frem, og i dette indtastes scenariets navn, Test Scenario. Det samme gøres med navnet Scenario Test. Programmet lukkes ned, og Scenario tabellen åbnes i Access. Her verificeres det, at disse navne står i tabellen, og at rækkefølgen er identisk med indtastningsrækkefølgen. Dette kan ses i figur 13.6.

Scenario : Table

ScenarioID	ScenarioName
1	Test Scenario
2	Scenario Test

Figur 13.6: Data i Scenario tabellen

Scenario Editoren åbnes igen, og Scenario Test vælges. Der trykkes ”Ret”, og et vindue kommer frem. Det verificeres, at der står Scenario Test i dette, hvilket viser, at der kan læses fra databasen. Dette ses i figur 13.7.



Figur 13.7: Det nuværende navn indlæses i boksen

Teksten i boksen rettes til Anden Scenario Test. Dette verificeres i Access, hvilket ses i figur 13.8.

Scenario : Table

ScenarioID	ScenarioName
1	Test Scenario
2	Anden Scenario Test

Figur 13.8: Navnet er ændret

Til sidst slettes Anden Scenario Test, og verificeres i Access.

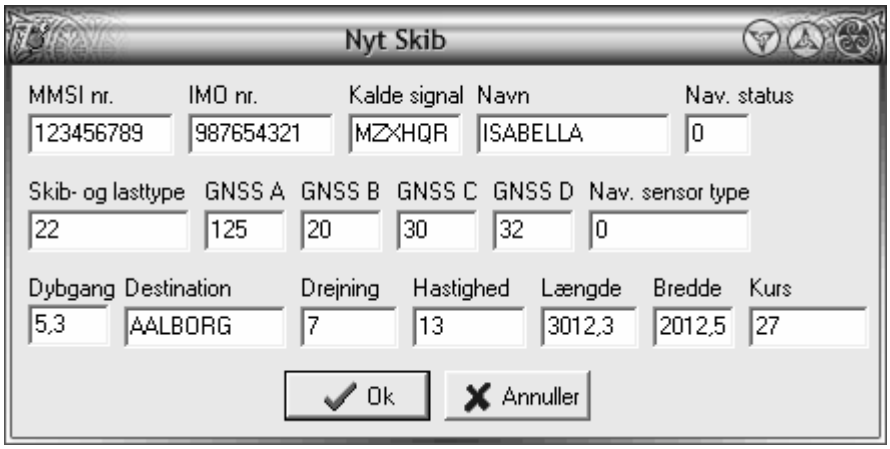
Scenario : Table

	ScenarioID	ScenarioName
▶	1	Test Scenario

Figur 13.9: Nederste scenarie er slettet

Test 3: Oprettelse og modificering af skibe

Der startes med databasen fra Test 2.
Testen udføres ved at klikke på knappen ”Nyt” under listen med skibe. Herved kommer et vindue frem, og skibets informationer indtastes. Disse informationer ses i figur 13.10.



Figur 13.10: Dataene som de tastes ind

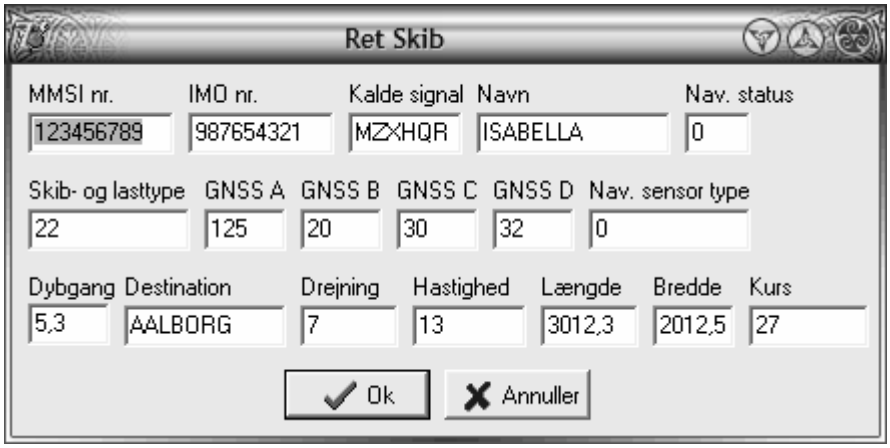
Programmet lukkes ned, og AISShip tabellen åbnes i Access. Her verificeres det, at disse informationer står i tabellen. Dette kan ses i figur 13.11.

AISShip : Table

	ShipID	ScenarioID	MMSI	NavStat	IMO	CallSign	Name	SCType	GNSSPosA	GNSSPosB	GNSSPosC	GNSSPosD	NavSTyp	Draught	Destination	Rotation	Speed	Latitude	Longitu	Course
▶	1	1	123456789	0	987654321	MZXHQR	ISABELLA	22	125	20	30	32	0	5,3	AALBORG	7	13	2012,5	3012,3	27

Figur 13.11: Data læst i Access

Til sidst åbnes programmet igen, og det verificeres, at informationerne læses, når der trykkes ”Ret”. Dette ses i figur 13.12.



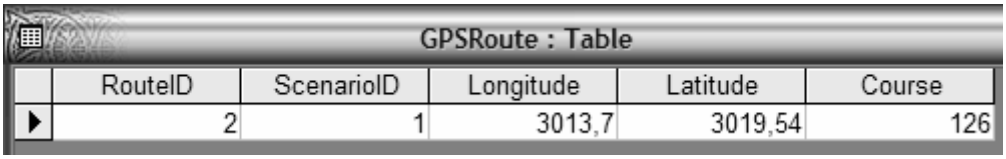
Figur 13.12: Data hentet fra databasen

Test4: Oprettelse og modificering af ruter

Der startes med databasen fra test 2.
Testen udføres ved at klikke på knappen ”Ny” under listen med ruter. Et vindue kommer frem, og rutens informationer indtastes. Disse informationer ses i figur 13.13.

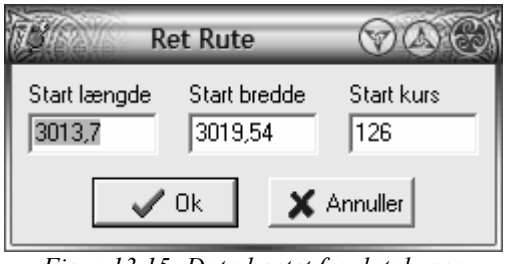
Figur 13.13: Data’ene, som de tastes ind

Programmet lukkes ned, og GPSRoute tabellen åbnes i Access. Her verificeres det, at disse informationer står i tabellen. Dette kan ses i figur 13.14. RouteID er 2 fordi der har været tastet en rute ind, der er blevet slettet igen.



Figur 13.14: Data læst i Access

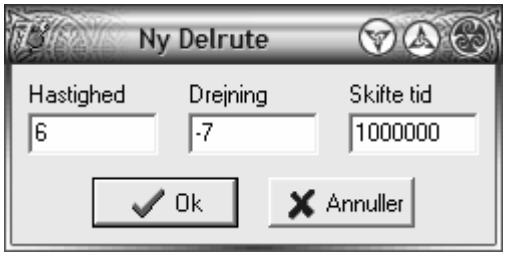
Til sidst åbnes programmet igen, og det verificeres, at informationerne læses, når der trykkes ”Ret”. Dette ses i figur 13.15.



Figur 13.15: Data hentet fra databasen

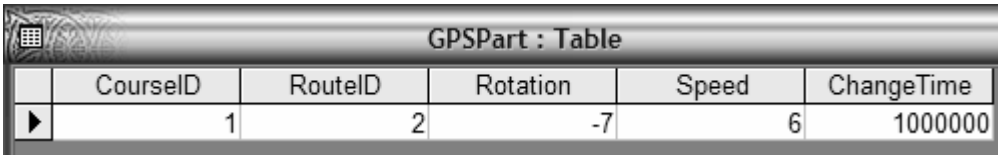
Test5: Oprettelse og modificering af rutedele.

Der startes med databasen fra test 4.
Testen udføres ved at klikke på knappen ”Ny” under listen med rutedele. Herved kommer et vindue frem, og informationer for delruten indtastes. Disse informationer ses i figur 13.16.



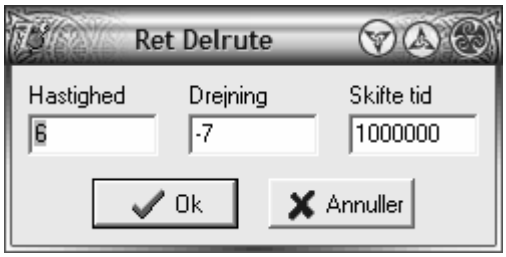
Figur 13.16: Data’ene som de tastes ind

Programmet lukkes ned, og GPSPart tabellen åbnes i Access. Her verificeres det, at disse informationer står i tabellen. Dette kan ses i figur 13.17.



Figur 13.17: Data læst i Access

Til sidst åbnes programmet igen, og det verificeres, at informationerne læses, når der trykkes ”Ret”. Dette ses i figur 13.18.



Figur 13.18: Data hentet fra databasen

Det kan konkluderes, at data placeres de rigtige steder, i de forskellige tabeller. Endvidere ses, at data kan læses fra tabellerne.
Det kan ligeledes konkluderes, at fremmednøglerne for tabellerne bliver lig primærnøglerne for de respektive mastertabeller (jf. afsnit 10.1).

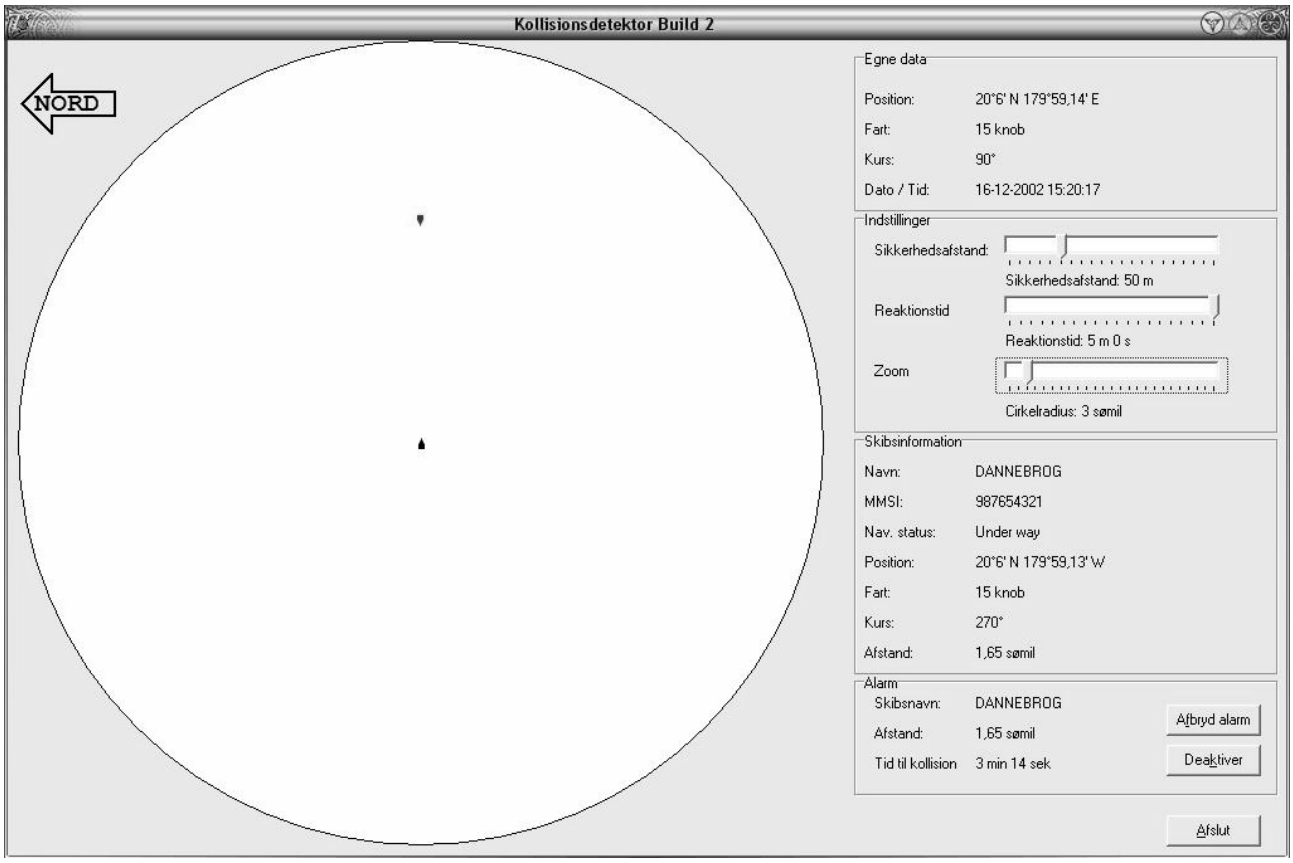
14 Accepttest

Accepttesten har til formål at verificere, at samtlige specifikke krav opstillet i kravspecifikationen er opfyldt. Desuden undersøges hvorvidt programmellet opfylder den ønskede prioritering af kvalitetsfaktorerne, opstillet i kravspecifikationen. Sideløbende med dette testes systemet ved diverse spidsbelastninger og ekstreme situationer, således eventuelle fejl eller begrænsninger findes.

I accepttesten tages der udgangspunkt i, at brugerterminalen er placeret på en PC med en AMD XP 1600+ processor, medens de to simulatorer er placeret på to PC'er begge med Intel P4 1800MHz processorer. De valgte scenarier, der benyttes i testen, er indtastet i Scenario Editoren.

14.1 Stress-test

Kollisionsdetektoren skal testes for det tilfælde, at et skib krydser datolinien.
Testen foretages således, at eget og andet skib placeres lige langt fra hinanden, på hver sin side af datolinien.
Herefter sættes begge skibe til at sejle den korteste vej mod datolinien med samme fart, hvorfor skibene burde støde sammen på datolinien.
Resultatet af testen er, at skibene støder sammen på datolinien og alarmen sættes som forventet.
Testen er illustreret i figur 14.1



Figur 14.1: Stress-test.

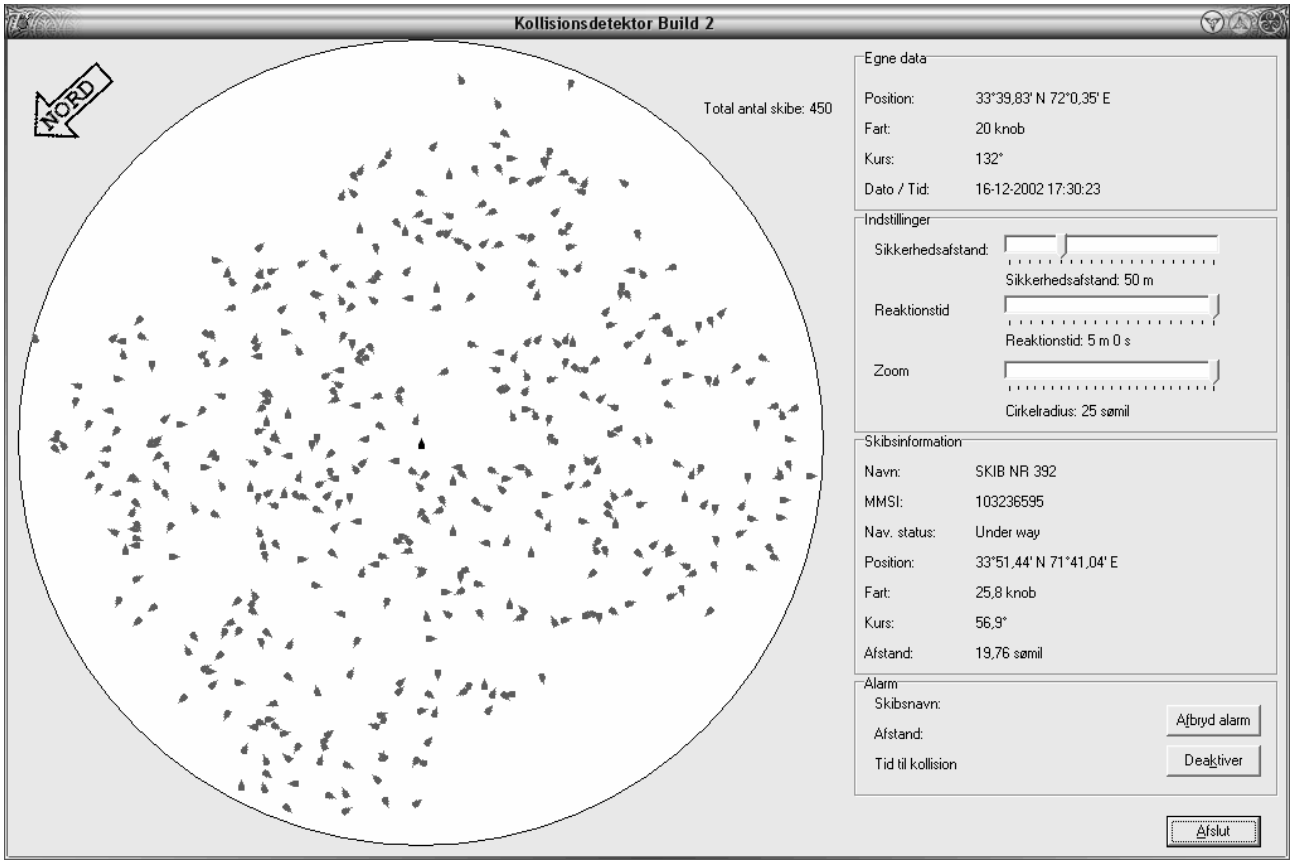
14.2 Konfigurationstest

Denne test skal undersøge, hvorvidt kollisionsdetektoren fungerer korrekt ved tilslutning til en GPS-modtager.
Til testen benyttes en GPS-modtager af typen: Shipmate RS5820 No. II med serienummeret 3924153201.
Konklusionen på denne test er, at det er muligt for kollisionsdetektoren at modtage GPS-informationer fra en GPS-modtager. Det kan således konstateres, at GPS-simulatoren simulerer GPS-informationer i det rette format.

14.3 Test af ydeevne

I denne test undersøges kollisionsdetektorens evne til at opdatere AIS-informationer, i henhold til AIS standarden [AIS s.3]. Endvidere undersøges det, hvorvidt kollisionsdetektoren kan behandle og plotte AIS-informationer fra 450 skibe.
Testen er foretaget ved, at AIS-simulatoren tilsluttes en hyperterminal på en anden PC, vha. serielporten. Efterfølgende afsendes AIS-informationer for et skib der sejler 15 knob og roterer, hvorfor afsendingsintervallet skal være 2 sek.
Ved gennemførelse af testen, kan det konkluderes, at der er overensstemmelse mellem resultatet af testen og det forventede resultat.
Efterfølgende foretages den samme test for ti tilfældige skibe samtidig.
Resultatet er ligeledes i denne test i overensstemmelse med det forventede.

Endelig kan det konkluderes, at kollisionsdetektoren kan behandle og plotte 450 skibe (se figur 14.2). Dette kan konkluderes, idet udnyttelsen af processorkraften ikke overstiger 15%, ved plotningen af 450 skibe. I testen er det valgt, at implementere en skibstæller til kontrol af, at AIS-informationer for samtlige skibe er modtaget.



Figur 14.2: Test af ydeevne.

14.4 Pålidelighedstest

Kollisionsdetektoren skal testes ved kontinuert normaldrift i 48 timer.
Denne test blev udført i perioden 14.-16. december 2002. Resultatet af denne test forløb uden form for fejlmeddelelser, og programmellet for både terminalen og simulatorerne fungerede korrekt ved testens afslutning.

14.5 Fejlbehandlingstest

Kollisionsdetektoren skal testes for det tilfælde, at der ikke længere modtages AIS-informationer fra et bestemt skib. I dette tilfælde skal informationerne for dette skib slettes fra brugerterminalens hukommelse vha. en time-out funktion. Formålet med denne funktion, skal være at undgå, at skibe der bevæger sig uden for sendeafstand stadig optager hukommelse. Endvidere er formålet, at undgå muligheden for spøgelsesskibe. Med dette menes AIS-informationer for skibe, der ikke længere befinder sig på den sidst registrerede position, men har bevæget sig uden for sendeafstand.
Testen er foretaget ved, at stoppe AIS-simulatoren i forbindelse med eksekvering af de endelige programmer. Herved modtager brugerterminalen således ikke længere AIS-informationer.
Det forventede resultat er, at hukommelsen slettes efter maksimalt otte minutter. Begrundelsen for dette er, at der hvert minut kontrolleres hvor længe der er gået siden der sidst er modtaget AIS-informationer fra de enkelte skibe. Er der gået mere end syv minutter for et skib, skal informationer for dette skib slettes.

14.6 Konklusion af accepttesten

Accepttestens formål har været at verificere, at samtlige specifikke krav opstillet i kravspecifikationen er opfyldt. Disse krav er opsummeret i det følgende for de enkelte hoveddele af kollisionsdetektoren:

14.6.1 AIS simulator

- Skal kunne simulere op til 450 skibe, af forskellig fart og kurs inden for et afgrænset område på op til 25 sømil i radius. Dette gøres da en realistisk situation ikke vil kunne overstige dette scenarie.
- Skal simulere AIS-informationer i en uafhængig PC og videresende disse via NMEA 0183 protokollen
- Skal udstyres med en pausefunktion
- Skal afsende AIS-informationer med det i AIS standarden angivne tidsinterval
- Skal kunne hente AIS scenarier fra en bestemt database

14.6.2 GPS simulator

- Skal hente GPS scenarier fra en bestemt database
- Skal simulere GPS-informationer, for eget skib, i en uafhængig PC og videresende disse via NMEA 0183 protokollen
- Skal udstyres med en pausefunktion

14.6.3 Brugerterminalen

- Skal kunne modtage AIS-informationer
- Skal kunne modtage GPS-informationer
- Skal kunne modtage og behandle AIS-informationer svarende til 450 skibe.
- Skal give en grafisk visualisering, herunder en zoomfunktion af den omkringliggende skibstrafik
- Skal give en visuel og auditiv alarm ved kollisionsrisiko
- Auditiv alarm skal kunne deaktiveres af brugeren
- Skal kunne give en række AIS-informationer vedrørende de omkringliggende skibe
- Skal give brugeren mulighed for at indstille sikkerhedsafstand og reaktionstid

På baggrund af resultatet af de enkelte tests i accepttesten og beskrivelsen af brugerfladen i afsnit 11, kan det konkluderes, at samtlige specifikke krav i kravspecifikationen er opfyldt.
Endvidere konkluderes det, at kollisionsdetektorens programdesign og funktionalitet, er i overensstemmelse med kvalitetsfaktorerne, opstillet i kravspecifikationen.

15 Konklusion

Formålet med dette projekt har været at designe og konstruere en kollisionsdetektor, der advarer lystbåde om risikoen for kollision.
Dette er gjort ved at dele designet af kollisionsdetektoren ind i følgende fire hoveddele:

- AIS-simulator
- GPS-simulator
- Brugerterminal
- Database

Samtlige hoveddele har været softwarebaseret.
Designet af de enkelte hoveddele er foretaget i overensstemmelse med AIS- og NMEA-standarderne, og har taget udgangspunkt i "Unified modeling language".
Efter færdiggørelse af designet, er de enkelte hoveddele blevet testet enkeltvis og samlet. Testen har således været inddelt og foretaget, i følgende tre dele, med forskelligt abstraktionsniveau:

- Modultest
- Integrationstest
- Accepttest

Konklusionen af disse test har været, at samtlige test verificerer en overensstemmelse mellem de forventede resultater og testresultaterne.

På baggrund af de foretagne test, kan det derfor konkluderes, at det er lykkedes at konstruere en kollisiondetektor, der lever op til samtlige krav stillet i kravspecifikationen.
Endvidere konkluderes det, at projektets udviklingsforløb har levet op til E-studienævnets projektenhedsbeskrivelse.

16 Litteraturliste

Reference	**UML**	**SPU**
Titel	UML Toolkit	Struktureret Program Udvikling
ISBN	0-471-19161-2	87-571-1046
Forlag	John Wiley & Sons, Inc.	Teknisk Forlag A/S
Udgave	1.	1.
Årstal/Dato	1998	1996
Forfatter	Hans-Erik Erikson & Magnus Penker	S.Biering-Sørensen m.fl.
Link		
Reference	**AIS**	**GPS**
Titel	ITU-R M.1371	The global positioning system
ISBN		1-57504-017-4
Forlag		Ann Arbor press, Inc.
Udgave		1.
Årstal/Dato	1998	1996
Forfatter	International Telecommunication Union	Michael Kennedy
Link	CD\Standarder\AIS	
Reference	**NMEA**	**AIS 2001**
Titel	Standard For Interfacing Marine Electronic Devices	IEC 61993-2
ISBN		
Forlag		
Udgave	2.30	1.
Årstal/Dato	1998	2001
Forfatter	National Marine Electronics Associaion	International Electrotechnical Commission
Link	CD\ Standarder \NMEA	CD\ Standarder \AIS_2001
Reference	**TRI**	**OMK**
Titel		
ISBN		
Forlag		
Udgave		
Årstal/Dato	26/11 2002	16/12 2002
Forfatter		
Link	www.ictp.trieste.it/~radionet/ghanal 1998/GPS/PRC.HTM	http://130.227.242.101/hjemmesider/spektrum/kap01-763.pdf
Reference	**NAV**	
Titel	Navigation	
ISBN	87-7790-014-6	
Forlag	Iver C. Weilbach & Co. A/S	
Udgave	1.	
Årstal/Dato	1993	
Forfatter	Jens Aage Jensen m.fl.	
Link	www.ictp.trieste.it/~radionet/ghanal 1998/GPS/PRC.HTM	

Appendiks

A Søvejsregler

De fleste søvejsregler er baseret på sund fornuft, "godt sømandskab" og individuel bedømmelse. Da dette er grundprincipper for søvejsreglerne er disse regler meget generelle. Det er således den sunde fornuft der råder, hvis der opstår et problem.
Et udpluk af disse regler følger:

Udkig
Ethvert skib er forpligtet til at holde udkig efter andre skibe. Dette kan ske vha. visuelt udsyn, radar eller en anden form for teknologiske hjælpemidler.

Sikker fart
Reglen om "sikker fart" hentyder til, at der altid skal sejles efter forholdene. En skibsfører skal således altid tage hensyn til faktorer såsom tåge, høj sø og tæt trafik. Kort sagt skal en skibsfører altid bruge sin sunde fornuft.

Forholdsregler for at undgå sammenstød
Ved enhver form for trafik vil der altid være en risiko for kollision. Da det er i alles interesse at undgå dette, har enhver skibsfører pligt til at tage alle de forholdsregler der er ham tilgængelig for at undgå dette.
Er to skibe på kollisionskurs, skal begge skibsførere foretage en kursændring mod styrbord.

B OSI-modellen

OSI modellen er opbygget af 7 lag, der beskriver de forskellige abstraktionsniveauer der anvendes til beskrivelse af kommunikationen mellem 2 systemer. OSI modellens 7 lag er illustreret i tabel B.7

7	Applikation
6	Præsentation
5	Session
4	Transport
3	Netværk
2	Data link
1	Fysisk lag

Tabel B.7: OSI modellen

Herunder følger en kort beskrivelse af arbejdsopgaverne for de enkelte lag i OSI modellen.

Lag 1: Fysiske lag:
- Transmission af rå bit over et fysisk medie.
- Fastlæggelse af bit i forbindelse med spændinger, tid, frekvens eller fase.
- Duplex, halv duplex eller simplex.
- Mekaniske og elektriske interfaces.

Lag 2: Data link:
- Transmission af data rammer over et fysisk medie.
- Fejldetektion og genoprettelse.
- Flow kontrol.

Lag 3: Netværk:
- Transport af pakker fra kilde til destination gennem forskellige undernetværk.
- Statisk eller dynamisk transport: Rutetabeller eller ruteprotokoller.

Lag 4: Transport:
- Modtager data fra sesssion laget, hvorefter dette deles op i mindre dele og sendes videre til netværks laget.
- Sikrer fejlfri transport til det andet system.
- Flow kontrol
- Multipleksing af flere forbindelser over netværket.

Lag 5: Session:
- Sessions laget giver brugerere på forskellige maskiner mulighed for at etablere sessioner mellem maskinerne.
- Dialog kontrol (holder styr på hvis tur det er til at sende).
- Token management (sørger for at to maskiner ikke benytter den samme funktion på samme tid)
- Synkronisering (sætter checkpoint i lange transmissioner, således at maskinerne kan opstarte korrekt i tilfælde af, at disse evt. er gået ned).

Lag 6: Præsentation:

- Præsentation af data.
- Oversættelse mellem forskellige dataformater (ASCII vs. EBCDIC eller CR+LF vs. CR).

Lag 7: Applikation:

- Indeholder en række applikationprotokoller (eksempelvis HTTP).

C Kollisionsberegninger

Til beskrivelse af beregningerne for skibe i kategori 1-4, tages udgangspunkt i 4 eksempler, og beregningerne for skibe i kategori 4 udledes.

C.1 Kategori 1 skibe

I de følgende eksempler tages der udgangspunkt i følgende relevante data for eget skib:

Position:
20° 10′17N:

$$y = 20° \cdot 60\,\text{min} + 10{,}17\,\text{min} = 1210{,}17\,\text{min} \approx 20{,}17° \qquad (C.14)$$

50° 20′33Ø:

$$x = 50° \cdot 60\,\text{min} + 20{,}33\,\text{min} = 3020{,}33\,\text{min} \qquad (C.15)$$

Det er valgt at omregne koordinaterne således disse angives i bueminutter.

Sikkerhedsafstand: = 50 meter
Reaktionstid: = 5 minutter = 300 sekunder
Fart:

$$5\,\text{knob} = 5\,{}^{\text{sm}}\!/_{\text{time}} \cdot \frac{1{,}852\,{}^{\text{km}}\!/_{\text{sm}}}{3{,}6\,\frac{{}^{\text{km}}\!/_{\text{time}}}{{}^{\text{m}}\!/_{\text{s}}}} = 5\,{}^{\text{sm}}\!/_{\text{time}} \cdot 0{,}514\,\frac{\text{time} \cdot \text{m}}{\text{sm} \cdot \text{s}} = 2{,}57\,\text{m/s} \qquad (C.16)$$

Kurs: = 0°
Relevant data for andet skib i kategori 1 er givet ved:

Position:
20° 35′25 N:

$$y = 20° \cdot 60\,\text{min} + 35{,}25\,\text{min} = 1235{,}25\,\text{min} = 20{,}58° \qquad (C.17)$$

50° 21′33 Ø:

$$x = 50° \cdot 60\,\text{min} + 21{,}33\,\text{min} = 3021{,}33\,\text{min} \qquad (C.18)$$

Da eget skibs koordinater lægges ind i origo bestemmes andet skibs koordinater i forhold til dette:
y koordinat:

$$y = (1235{,}25\,\text{min} - 1210{,}17\,\text{min}) \cdot 1\,{}^{\text{sm}}\!/_{\text{min}} = 25{,}08\,\text{sm} \qquad (C.30)$$

x koordinat:

$$x = \cos\left(\frac{20{,}58° + 20{,}17°}{2}\right) \cdot (3021{,}33\,\text{min} - 3020{,}33\,\text{min}) \cdot 1\,{}^{\text{sm}}\!/_{\text{min}} = 0{,}9374\,\text{sm} \qquad (C.31)$$

Til bestemmelse af afstanden mellem eget og andet skib benyttes ligning C.21.

$$d = \sqrt{(25,08\text{sm})^2 + (0,9374\text{sm})^2} = 25,10\,\text{sm} \qquad (C.21)$$

Da denne afstand er større end skærmbillederadiusen, på 25 sømil, er dette skib et kategori 1 skib.

C.2 Kategori 2 skibe

Relevant data for andet skib i kategori 2 er givet ved:

Position:
20° 02′00N:

$$y = 20° \cdot 60\,\text{min} + 2\,\text{min} = 1202\,\text{min} = 20,03° \qquad (C.22)$$

50° 21′33Ø:

$$x = 50° \cdot 60\,\text{min} + 21,33\,\text{min} = 3021,33\,\text{min} \qquad (C.23)$$

Kurs:181°

Da eget skibs koordinater igen lægges ind i origo bestemmes andet skibs koordinater i forhold til dette:
y koordinat:

$$y = (1202\,\text{min} - 1210,17\,\text{min}) \cdot 1\,{}^{\text{sm}}\!/_{\text{min}} = -8,17\text{sm} \qquad (C.24)$$

x koordinat:

$$x = \cos\left(\frac{20,03° + 20,17°}{2}\right) \cdot (3021,33\,\text{min} - 3020,33\,\text{min}) \cdot 1\,{}^{\text{sm}}\!/_{\text{min}} = 0,9391\text{sm} \qquad (C.25)$$

Til bestemmelse af skibets kategori benyttes igen afstandsformlen:

$$d = \sqrt{(-8,17\text{sm})^2 + (0,9391\text{sm})^2} = 8,224\,\text{sm} \qquad (C.26)$$

Da denne afstand er mindre end skærmbillederadiusen, på 25 sømil, og større end radius på beregningscirklen, på 4,92sm, er dette skib et kategori 2 skib.

C.3 Kategori 3 skibe

Relevant data for andet skib i kategori 3 er givet ved:

Position:
20° 08′17N:

$$y = 20° \cdot 60\,\text{min} + 8,17\,\text{min} = 1208,17\,\text{min} = 20,13° \qquad (C.27)$$

50° 21′33 Ø:

$$x = 50° \cdot 60\,\text{min} + 21,33\,\text{min} = 3021,33\,\text{min} \qquad (C.28)$$

Skibsradius: = 10 meter
Fart: = 8 knob
Kurs: = 181°

Da eget skibs koordinater igen lægges ind i origo bestemmes andet skibs koordinater i forhold til dette:
y koordinat:

$$y=(1208{,}17\,\text{min}-1210{,}17\,\text{min})\cdot 1\,{}^{\text{sm}}\!/_{\text{min}}=-2{,}0\,\text{sm} \qquad (C.29)$$

x koordinat:

$$x=\cos\left(\frac{20{,}13°+20{,}17°}{2}\right)\cdot(3021{,}33\,\text{min}-3020{,}33\,\text{min})\cdot 1\,{}^{\text{sm}}\!/_{\text{min}}=0{,}9388\,\text{sm} \qquad (C.30)$$

Til bestemmelse af skibets kategori benyttes igen afstandsformlen:

$$d=\sqrt{(-2\text{sm})^2+(0{,}9388\text{sm})^2}=2{,}209\text{sm} \qquad (C.31)$$

Da denne afstand er mindre end radius på beregningscirklen, på 4,92sm, er dette skib enten et kategori 3 eller 4 skib.
I ligning C.32 bestemmes hvor meget de to skibe har nærmet sig hinanden, indenfor reaktionstiden, i det tilfælde, at skibene har kurs direkte mod hinanden. Hertil lægges skibsradius og sikkerhedsafstand.

$$d=\left((v_1+v_2)\cdot t_r+\text{skibsradius+sikkerhedsafstand}\right)$$
$$\Downarrow \qquad (C.32)$$
$$d=(5+8)\,\text{knob}\cdot 0{,}514\,\frac{\text{time}\cdot\text{m}}{\text{sm}\cdot\text{s}}\cdot 300\,\text{s}+50\ \text{m}+10\,\text{m}=2060{,}7\,\text{m}=1{,}11\,\text{sm}$$

Da resultatet i ligning C.32 er mindre end ligning C.31 er der tale om et kategori 3 skib, og ikke et kategori 4 skib.

C.4 Kategori 4 skibe

Relevant data for andet skib i kategori 4 er givet ved:

Position:
20° 11′00 N:

$$y=20°\cdot 60\,\text{min}+11\,\text{min}=1211\,\text{min}=20{,}18° \qquad (C.33)$$

50° 20′35 Ø:

$$x=50°\cdot 60\,\text{min}+20\,\text{min}+\frac{21\cdot 100}{60\cdot 100}\,\text{min}=3020{,}35\,\text{min} \qquad (C.34)$$

Skibsradius: = 10 meter
Kurs: = 181°
Fart: = 8 knob
Da eget skibs koordinater igen lægges ind i origo bestemmes andet skibs koordinater i forhold til dette:
y koordinat:

$$y=1211-1210{,}17=0{,}83\,\text{sm} \qquad (C.35)$$

x koordinat:

$$x = \cos\left(\frac{20{,}18° + 20{,}17°}{2}\right) \cdot (3020{,}35\,\text{min} - 3020{,}33\,\text{min}) \cdot 1\,{}^{\text{sm}}\!/_{\text{min}} = 0{,}0188\,\text{sm} \qquad (C.36)$$

Til bestemmelse af skibets kategori benyttes igen afstandsformlen:

$$d = \sqrt{(0{,}83\text{sm})^2 + (0{,}0188\text{sm})^2} = 0{,}8302\text{sm} \qquad (C.37)$$

Da denne afstand er mindre end radius på beregningscirklen, på 4,92sm, er dette skib enten et kategori 3 eller 4 skib.
I ligning C.38 bestemmes hvor meget de to skibe har nærmet sig hinanden, indenfor reaktionstiden, i det tilfælde, at skibene har kurs direkte mod hinanden. Hertil lægges skibsradius og sikkerhedsafstand.

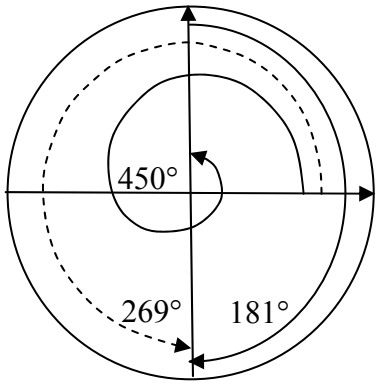
$$d = \left((v_1 + v_2) \cdot t_r + \text{skibsradius+sikkerhedsafstand}\right)$$
$$\Downarrow$$
$$d = (5+8)\,\text{knob} \cdot 0{,}514\,\frac{\text{time} \cdot \text{m}}{\text{sm} \cdot \text{s}} \cdot 300\,\text{s} + 50\,\text{m} + 10\,\text{m} = 2064{,}6\,\text{m} = 1{,}11\,\text{sm} \qquad (C.38)$$

Da resultatet i ligning C.38 er større end i ligning C.37 er der tale om et kategori 4 skib

Det er herefter nødvendigt at beregne positionerne for henholdsvis eget og andet skib som Funktion af tiden t.
Nyt x-koordinat for andet skib:

$$x_{\text{dem}} = v_x \cdot t + x_{gammel}$$
$$\Downarrow$$
$$x_{\text{dem}} = \cos(450° - 181°) \cdot 8\text{knob} \cdot 0{,}514\,\frac{\text{time} \cdot \text{m}}{\text{sm} \cdot \text{s}} \cdot t + 0{,}0188\text{sm} \cdot 1852\,{}^{\text{m}}\!/_{\text{sm}} \qquad (C.39)$$
$$\Downarrow$$
$$x_{\text{dem}} = -0{,}0717\,{}^{\text{m}}\!/_{\text{s}}\, t + 34{,}82\text{m}$$

Begrundelsen for at trække kursen fra 450° er, at det er 90 graders forskydning mellem kursen 0° og 0° på enhedscirklen. Endvidere gør det at kursen får en negativ omløbsretning på enhedscirklen. Virkningen af dette, for det aktuelle kategori 4 skib, ses i figur C.3



Figur C.3: Sammenhæng mellem kompaskurs og grader på enhedscirklen.

Nyt y-koordinat for andet skib:

$$\begin{gathered} y_{dem} = v_y \cdot t + y_{gammel} \\ \Downarrow \\ y_{dem} = \sin(450° - 181°) \cdot 8\text{knob} \cdot 0{,}514 \frac{\text{time} \cdot \text{m}}{\text{sm} \cdot \text{s}} \cdot t + 0{,}83\text{sm} \cdot 1852 \,\text{m}/\text{sm} \\ \Downarrow \\ y_{dem} = -4{,}111 \,\text{m}/\text{s} \cdot t + 1537\text{m} \end{gathered} \qquad (C.40)$$

X-koordinat for eget skib er altid 0, hvorfor det kun er nødvendigt at beregne nyt y-koordinat:

$$\begin{gathered} y_{os} = v \cdot t \\ \Downarrow \\ y_{os} = 5 \cdot 0{,}514 \frac{\text{time} \cdot \text{m}}{\text{sm} \cdot \text{s}} t = 2{,}57 \,\text{m}/\text{s}\, t \end{gathered} \qquad (C.41)$$

Herefter beregnes om summen af sikkerhedsafstanden og skibsradius brydes indenfor reaktionstiden. Hertil benyttes ligning C.42:

$$\begin{gathered} 0 > \sqrt{(y_{dem} - y_{os})^2 + {x_{dem}}^2} - (\text{Sikkerhedsafstand+skibsradius}) \\ \Downarrow \\ 0 > \sqrt{\left((v_y \cdot t + y_{gammel}) - v \cdot t\right)^2 + (v_x \cdot t + x_{gammel})^2} - (\text{Sikkerhedsafstand+skibsradius}) \\ \Downarrow \\ 0 > \sqrt{\left(\left(-4{,}111 \,\text{m}/\text{s}\, t + 1537\text{m}\right) - 2{,}57 \,\text{m}/\text{s}\, t\right)^2 + \left(-0{,}0717 \,\text{m}/\text{s}\, t + 34{,}82\text{m}\right)^2} - 60\text{m} \\ \Downarrow \\ 221{,}5s < t < 238{,}6s \end{gathered} \qquad (C.42)$$

Summen af sikkerhedsafstanden og skibsradius brydes altså efter 221,49s hvorfor alarmen skal have lydt. Efter 238,61s skal alarmen igen slå fra.