FFIE
Intern rapport E–216
Referanse: Jobb 251/134
Dato: Juli 1973

FORELØPIG SYSTEM- OG PROGRAMBESKRIVELSE AV ET ILDLEDNINGS-SYSTEM FOR HURTIGE PATRULJEBÅTER

av

T Haugland

Godkjent
Kjeller 31 juli 1973

B Landmark
Superintendent

FORSVARETS FORSKNINGSINSTITUTT
Norwegian Defence Research Establishment
Postboks 25 – 2007 Kjeller
Norge

BEGRENSET
i h t Sikkerhetsinstruksen

FFIE
Intern rapport E–216
Referanse: Jobb 251/134
Dato: Juli 1973

FORELØPIG SYSTEM- OG PROGRAMBESKRIVELSE AV ET ILDLEDNINGS-SYSTEM FOR HURTIGE PATRULJEBÅTER

av

T Haugland

Godkjent
Kjeller 31 juli 1973

B Landmark
Superintendent

FORSVARETS FORSKNINGSINSTITUTT
Norwegian Defence Research Establishment
Postboks 25 – 2007 Kjeller
Norge

BEGRENSET
i h t Sikkerhetsinstruksen

INNHOLDSFORTEGNELSE

FORELØPIG SYSTEM- OG PROGRAMBESKRIVELSE AV ET ILDLEDNINGS-SYSTEM FOR HURTIGE PATRULJEBÅTER

SUMMARY

This report is a preliminary system and software description of a fire control system for fast patrol boats. The report also describes a software package which despite making very small demands on existing hardware should be a powerful tool in debugging system software and trying out the man-machine communication in a mock-up.

The work reported in this paper was carried out at the Norwegian Defence Research Establishment between March 1971 and December 1972.

(Preliminary system and software description of a fire control system for fast patrol boats)

1 INNLEDNING

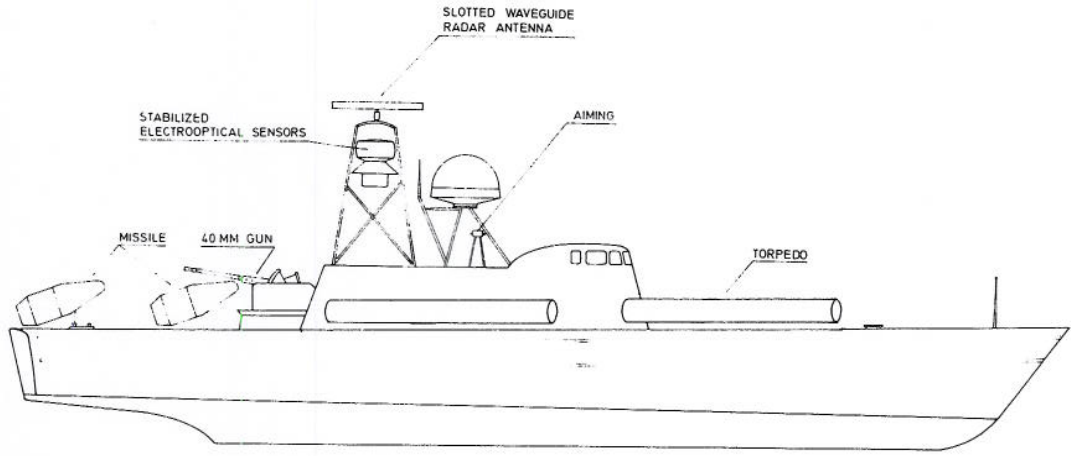
Ved Forsvarets forskningsinstitutt har det i perioden april 1971 til desember 1972 pågått arbeid med utvikling av software for et ildledningssystem beregnet for neste generasjons hurtige patruljebåter. Arbeidet har vært konsentrert om følgende oppgaver:

- Å utarbeide foreløpige spesifikasjoner for oppbygging av programsystemet. Til dette hører også kommunikasjon med perifert utstyr
- Å utarbeide et opplegg for uttesting av software og mann-maskin kommunikasjon
- Koding og debugging av programmer. Dette vil i praksis være programmer som er uavhengige av detaljerte systemspesifikasjoner.

Denne rapport vil i store trekk presentere resultatet fra arbeidet med spesifikasjon av programsystem og opplegg for uttesting. Det er først og fremst lagt vekt på hovedprinsipper ved oppbygning av programsystemet. Videre er programmene for innlesning av sensordata og for målfølging relativt grundig behandlet. Kontroll av våpen derimot er bare i liten grad beskrevet. Arbeidet kan ikke sees isolert fra en mer generell systembeskrivelse. Denne beskrivelsen følger i kapittel 2, 3 og 4.

2 FUNKSJONELL BESKRIVELSE

Ildledningssystemet er beregnet for neste generasjons hurtige patruljebåter (FPB). Et bilde av et fartøy slik som en kan tenke seg det med sensorer og våpen er vist på figur 2.1.



Figur 2.1 Fartøy

Systemet er et multisensor, multimål ildledningssystem som samtidig eller separat skal kunne utnytte passive og aktive sensorer. Systemet skal ved hjelp av sensordata kunne produsere måldata med tilstrekkelig nøyaktighet til å kunne fyre og om nødvendig styre valgte våpen mot målet.

2.1 Sensorer

De sensorer som skal kunne benyttes i systemet er:

- Primærradar
- Navigasjonsradar

  Ved hjelp av en radar video ekstraktor (ARVEX) vil det være mulig å følge opp til 16 mål samtidig
- TV
- Laser
- Billeddannende IR

  Disse sensorene (optisk pakke) vil være montert på en stabil plattform
- Optisk sikte (ustabilisert)
- Radar søkemottaker (for varsling)

2.2 Våpen

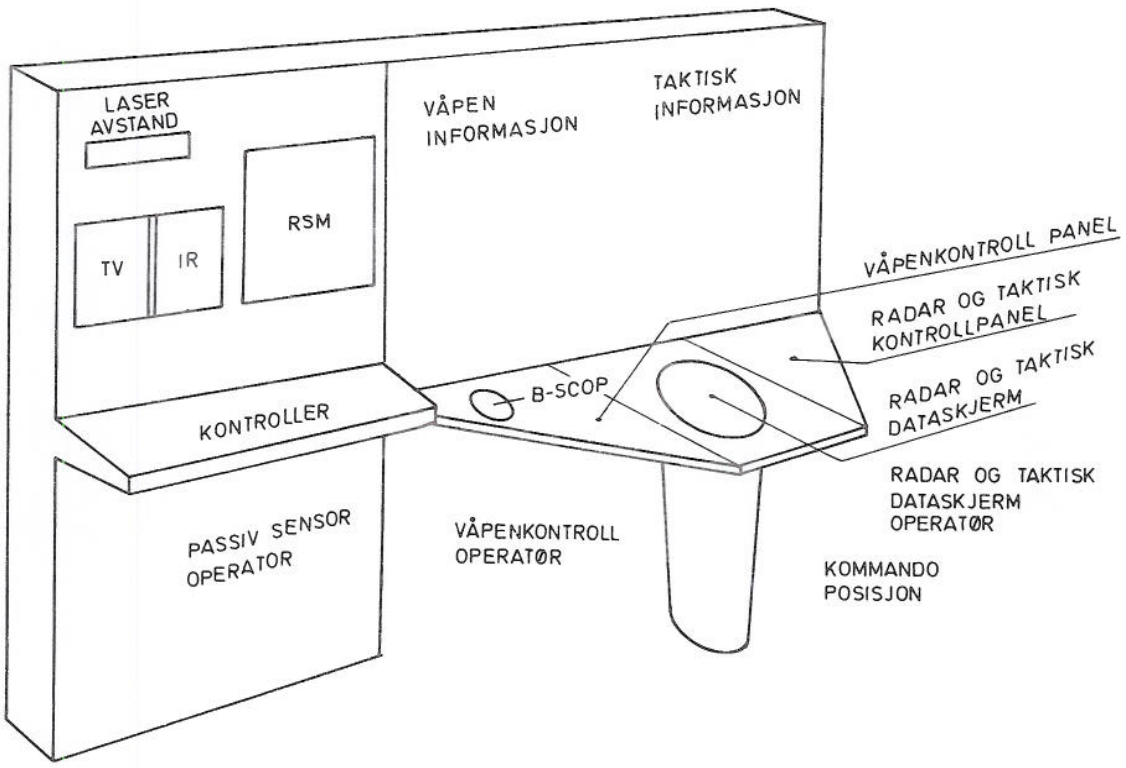
De våpen som en skal kunne benytte i systemet (se figur 2.2) er:

- Penguin raketter
- TI trådstyrte torpedoer
- 40 mm kanon

Figur 2.2 Våpen

2.3 Målfølging- og våpenkontroll

Figur 2.3 viser konsollet for målfølging og våpenkontroll. Figuren er tatt med for å illustrere hvordan systemet i grove trekke skal funksjonere, og representerer ikke noe konkret forslag til konsoll.



Figur 2.3 Konsoll for målfølging og våpenkontroll

Som figuren viser vil konsollet bestå av følgende hovedenheter:

– Passiv sensor panel (TV, laser, IR og radar søkemottaker)

– Horisontal radar- og taktisk dataskjerm

– Radar og taktisk kontrollpanel

– Våpenkontroll panel

Passiv sensor panel vil inneholde TV og IR dataskjerm, laser avstandsindikator og RSM dataskjerm. Det vil videre være indikatorer for stabil plattform inkludert tilt og peiling til optisk pakke. I tillegg vil det være knapper og vendere for kontroll og styring av de enkelte deler. Hovedoppgaven til operatøren ved dette panelet vil være styringen av den stabile plattform (det vil si å styre den optiske pakke mot ønskede mål), valg av sensorer, og å styre innlesingen av sensordata til regnemaskinen.

Radar og taktisk kontrollpanel vil bli operert av en *radar og taktisk operatør*. Hans oppgave er å velge radar og kontrollere og styre bruken av radar og taktisk dataskjerm samt å utføre manuelle plotteoppgaver.

Kommanderende offiser er plassert mellom radaroperatør og våpenoperatør.

Våpenkontroll panelet vil bli operert av en *våpenkontroll operatør*. Han vil være ansvarlig for styring av torpedoer og fyring av Penguin-raketter mot dedikerte mål. Han vil i tillegg ha en viss kontroll over styring av 40 mm kanon. Til hjelp under vanskelige forhold vil systemet være utstyrt med et B-skop.

3 BESKRIVELSE AV DATAMASKINKONFIGURASJON

På et tidlig tidspunkt ble det besluttet at en skulle vurdere mulighetene for å lage et typisk "bus"-orientert system rundt hovedregnemaskinen. Hensikten med dette var å tilstrebe en modularitet som ikke skulle låse systemet hverken med hensyn til regnekapasitet eller tilkobling av eksternt utstyr. Selv om en del av disse ideene ikke lenger er aktuelle er de i seg selv så interessante at en beskrivelse er tatt med i dette kapittel. En omtale vil så bli gitt av den datamaskinkonfigurasjon som i øyeblikket synes aktuell. Kapittelet vil også innehold en kort beskrivelse av den hovedregnemaskin som skal benyttes i systemet.

Den konfigurasjon som skal behandles er basert på at stabilplattform og radarekstraktor som utvikles ved FFI skal benyttes.

3.1 Regnemaskin

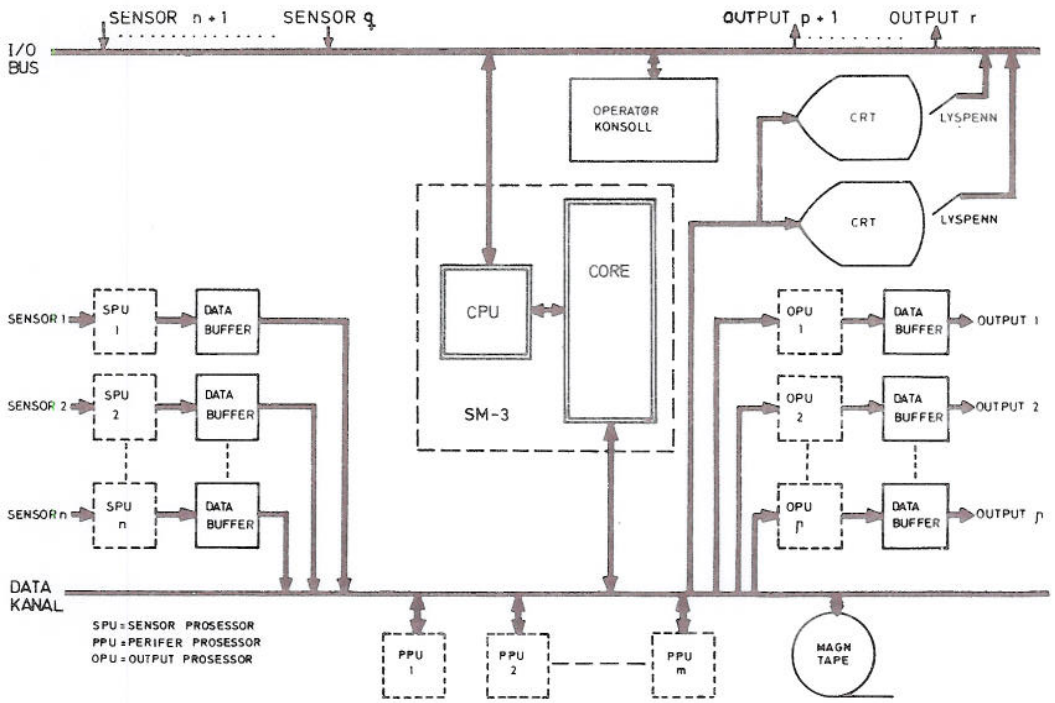
Ildledningssystemet vil være bygget opp omkring en programmerbar regnemaskin. Dette vil være en SM-3 maskin produsert av Kongsberg Våpenfabrikk. SM-3 er en 16 bits maskin med et relativt sterkt instruksjonssett og er spesielt anvendelig for prosess-styringsformål. En memory aksess vil ta ca 1,5 μs og det forutsettes at det er tilstrekkelig med ett nivås interrupt. Det kan benyttes 16, 24 eller 32 k ords hukommelse. Innlesing og utlesing fra regnemaskinen skjer via akkumulator. Et unntak i så måte er dataskjermprosessoren. Denne leser ut data fra SM-3's hukommelse via datakanal. Dette er beskrevet senere i dette kapittel.

3.2 Generell beskrivelse

Grove estimater viste at regnekapasiteten til SM-3 ville være utilstrekkelig. Dette skyldtes først og fremst to forhold:

- Bruk og valg av sensorer ville føre med seg relativt tidkrevende beregninger og samtidig ville belastningen på SM-3 på grunn av overhead bli stor da disse beregningene enten måtte skje ved interrupt eller ved at "sann-tid" klokkefrekvens ble øket fra 10 Hz til minimum 50 Hz.
- Ved målfølgingen ville det være aktuelt å benytte filteralgoritmer hvor en i det alt vesentlige måtte arbeide med flytende regning hvor SM-3 er svak.

Det ble derfor besluttet å gå for et modulært opplegg slik som skissert på figur 3.1, hvor den faste del er tegnet heltrukket, mens systemets utvidelsesmuligheter er tegnet stiplet. Den heltrukne delen består av en SM-3 hovedregneenhet. Videre finnes et operatør konsoll med forskjellig utstyr som knapper, vendere, tekst- og tallindikatorer o l. I dette systemet finnes også to grafiske dataskjermer med tilkobling av lyspenn og en båndstasjon for datainnsamling. Sensordata *inn* til systemet kommer enten via I/O-bus (innlesing av ett og ett ord til akkumulatoren) eller fra fordelte buffere via datakanal (data overføres i blokker direkte til core samtidig som CPU kan arbeide med andre oppgaver). Tilsvarende kan data skrives ut via I/O-bus eller datakanal.



Figur 3.1 Hardwarekonfigurasjon I

Den stiplede, modulære delen består av sensorprosessorer (SPU) hvor disse både kan samle og preprosessere sensordata og dermed frita hovedregneenheten for både regnearbeid og avbrudd. På utgangen kan en tilsvarende benytte output prosessorer (OPU), som viderebehandler data levert fra hovedregneenheten. I tillegg kan systemet ha et antall perifere prosessorer (PPU). Fra hovedregneenheten kalles en slik PPU opp via én instruksjon. Et datasett som hovedregneenheten på forhånd har gjort klart leses da fra core til et lager i den oppkalte PPU. Etter overføring kan PPU gjøre beregningene og

til slutt overføre data tilbake til core. Hovedregneenheten kan få beskjed om at resultatet er levert på to måter, enten ved interrupt eller ved at den tester på om data er levert.

Med de perifere prosessorene ville det være spesielt aktuelt å utføre matrisearitmetikk, idet det ville være mulig å bygge en enkel PPU slik at denne i seg selv er en multiprosessor. Alle prosessorene var forutsatte å skulle være av samme type mikroprogrammerte regneenhet. Denne er beskrevet i (4).

Det er i det foregående lagt vekt på å beskrive datastrømmen til og fra hovedregneenheten. Det er imidlertid ikke noe i veien for å overføre data direkte fra f eks en SPU til en OPU.

Med den type system som er vist på figur 3.1, vil en stå overfor et typisk ressursallokeringsproblem hvor det gjelder å finne en kombinasjon av prosessorer som gir best ytelse i forhold til kostnad. For å finne ut av dette i tillegg til at det var et generelt behov for en nøyere undersøkelse av eventuelle regnekapasitetsproblemer, ble det laget en tidsmodell av systemet skrevet i SIMULA 67. Denne er beskrevet i kapittel 7.1. Modellen har vist seg å være et nyttig hjelpemiddel ved valg av hardware og også software. Selv om det modulære konsept på figur 3.1 ikke blir realisert fullt ut, er det grunn til å summere opp de viktigste fordeler systemet har framfor å gå for et system med sentralisert regnekapasitet i form av én CPU:

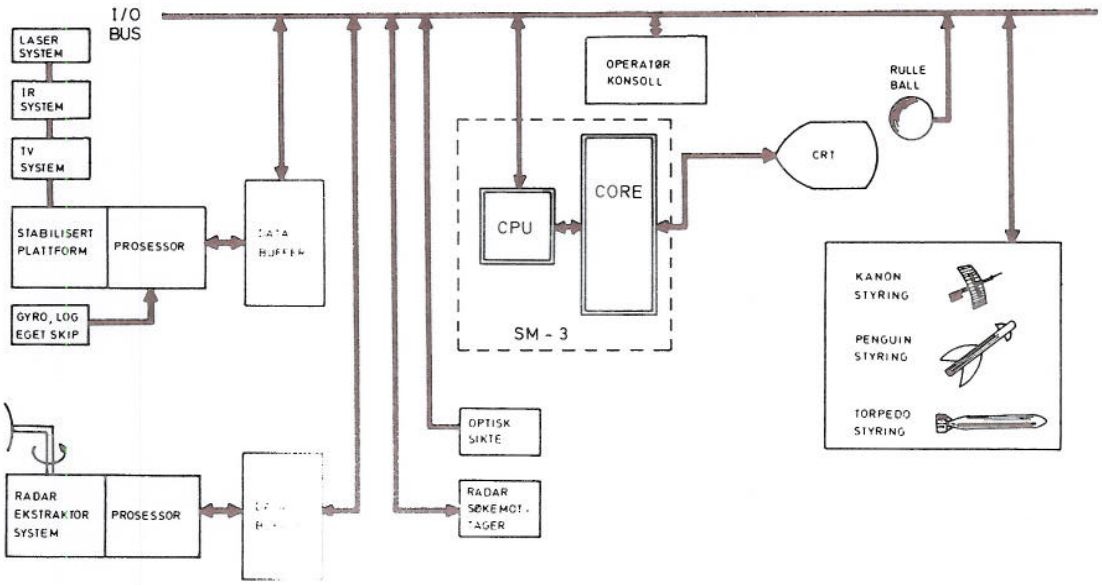
- Ved bruk av én regneenhet har en bundet systemets kapasitet til denne. Med et modulært system som på figur 3.1 har en bedre valgmuligheter og bedre utvidelsesmuligheter hvis dette senere skulle være aktuelt.
- De eksterne prosessorene er effektive "tallknusere" og er som sådanne prismessig fordelaktige i forhold til en større CPU.
- En får "desentralisert" software. Bl a kan sensordata leses inn til hovedregneenheten til tider og på en form som er velegnet.

Den største ulempen ved systemet er at hvis en skal benytte eksterne prosessorer, er datakanal en forutsetning, og dette er mer komplisert enn om data kan leses ved vanlig input/output.

3.3 Foreløpig systemkonfigurasjon, hardware

Et utkast til en aktuell systemkonfigurasjon er vist på figur 3.2. Dette skiller seg fra det som figur 3.1 viser på et par vesentlige punkter. For det første er datakanalen fjernet, slik at utveksling av data må skje ved ett og ett ord over vanlig input/output forbindelse. Med dette er muligheten for å bygge ut systemet med eksterne prosessorer blitt sterkt begrenset. En annen forandring er at lyspenn er skiftet ut med rulleball i tillegg til at systemet på figur 3.3 er forsynt med bare en dataskjerm som vil vise både radar video og syntetisk informasjon. Overgangen fra lyspenn til rulleball har indirekte stor betydning for hvordan software skal bygges opp, da dette vil innvirke på hvordan kommunikasjon mellom operatør og system skal være. Grunnen til at man valgte å gå for et noe svakere konsept enn det som er omtalt i forrige kapittel var endringer i valg og bruk av sensorer.

En usikkerhet ved systemet er hvorvidt Penguin-rakettene skal styres direkte fra SM-3 slik som figuren viser eller om de skal styres fra en egen datamaskin som før. (Måldata må i alle fall skaffes til veie av SM-3.) Dette spørsmål er imidlertid av underordnet betydning for denne rapporten som først og fremst vil beskrive sensorsiden og målfølging.



Figur 3.2 Hardwarekonfigurasjon II

Sensorsiden består av en stabil plattform. På denne er montert en optisk pakke som inneholder TV, billeddannende IR og laser avstandsmåler. Retningen på den optiske pakken (azimuth og tilt) styres av en operatør via SM-3. I forbindelse med plattformen finnes en prosessor som i tillegg til å være en integrert del av stabilisering- og styringssystemet for plattformen, også leverer filtrerte data om eget fartøys kurs og fart. Datautvekslingen mellom stabil plattform og resten av systemet vil skje via et buffer.

Radarsystemet, inklusive ekstraktor (ARVEX), vil på samme måte som stabil plattform kommunisere med omverdenen via et databuffer. Dataraten fra ARVEX antas å ville være maksimalt 50–60 ekko pr scan. Tid for hvert scan vil være 2–3 sek, og hver måling vil inneholde avstands- og peilingsinformasjon om ett ekko. Grunnen til at man kan forutsette så lav datarate er at *søkeområdet vil bli begrenset til ca 16 vinduer* hvor hvert vindu vil ha en side på 400–500 m. Midtpunktet for hvert søkeområde vil styres dynamisk fra SM-3. De 16 vinduene er valgt i samsvar med det antall mål som systemet skal kunne følge samtidig. Sensor innlesningen vil bli nærmere omtalt i neste kapittel.

Operatørkonsollet og hardware i forbindelse med dette vil være av samme type som det som ble benyttet i MSI-70U (5). Ved aktivering av knapper eller vendere vil et spesielt program i SM-3 bli aktivert, slik at all konsollstatus (lys i knapper, tallindikatorer etc) styres fra SM-3.

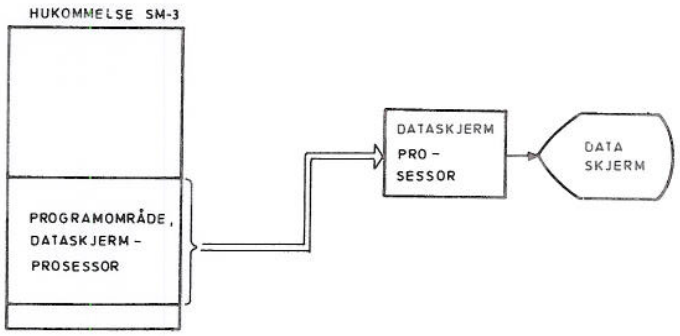
Dataskjermsystemet vil være identisk med det som ble benyttet i MSI-70U. Prinsippet er beskrevet i kapittel 3.5. Til hjelp for operatøren finnes det i forbindelse med dataskjermen én eller flere rulleballer. Et symbol på skjermen (styrt via et program) forflyttes i henhold til innholdet av to registre. Disse registre styres på sin side av rulleballens bevegelse i to retninger.

Det er gjort lite for å se på tilknytning mellom våpnene og resten av systemet. De vurderingene som er gjort tyder på at tilknytningen sett fra regnemaskinen neppe vil by på problemer. Data kan leses inn/ut via vanlig input/output samtidig som dataratene er lave. Med hensyn til hvordan Penguin-rakettene skal tilknyttes, er dette et problem som ikke vil bli behandlet i denne rapport.

3.4 Beskrivelse av dataskjermsystemet

Prinsippet er vist på figur 3.3. Katodestrålen styres ved hjelp av en *dataskjermprosessor* (DSP). Denne arbeider uavhengig av den sentrale regneenheten og har direkte aksess til dennes hukommelse ved hjelp av egen datakanal.

DSP har sitt eget programområde, og henter instruksjon for instruksjon og dekoder. Hver instruksjon kan være en beskjed om å tegne et punkt eller vektor etc. Programområdet for DSP vil genereres av hovedregneenheten, og vil fra denne være å betrakte som et dataområde.



Figur 3.3 Dataskjermprosessor, prinsippskisse

Prinsipielt utgjør CPU og DSP et to-prosessorsystem som arbeider på en felles hukommelse, men der DSP bare kan *lese* data mens CPU både kan lese og skrive.

4 DATA TIL OG FRA SENSORENE

Dette kapittel beskriver datautvekslingen mellom SM-3 og de viktigste sensorene. Beskrivelsen baserer seg på den hardwarekonfigurasjon som er gjennomgått i kapittel 3.3, figur 3.2. Hvis denne skulle bli vesentlig endret, vil dette kunne få stor innvirkning på dataflyten mellom SM-3 og det perifere utstyr.

4.1 Data til og fra stabil plattform

Dataoverføringen til og fra stabil plattform hvorpå den optiske pakke er plassert vil skje via et buffer som vist på figur 3.2.

Data *fra* stabil plattform vil være:

ψ_s Kurs, eget fartøy

θ_s Rull, eget fartøy

ϕ_s Stamp, eget fartøy

v_{SN}, v_{SE} Hastighet i Nord- og Øst-retning, eget fartøy (glattet)

a_{SN}, a_{SE} Akselerasjon i Nord- og Øst-retning, eget fartøy

l, e Posisjon (lengde og bredde) eget fartøy

ψ_{TV} Optisk pakkes retning

Data til dette buffer skrives ut *asynkront* med innlesingen til SM-3 eller annet eksternt utstyr som i prinsippet kan lese fra bufferet. (ARVEX vil f eks lese ψ_S, θ_S og ϕ_S direkte med en oppdaterings på ca 30 Hz.) Innskriving til og utlesing fra bufferet kan gjøres asynkron fordi oppdateringen av data til bufferet skjer så hurtig at forandringen fra den ene utskriving til den andre er uvesentlig for brukerne.

Data *til* den stabile plattform vil være:

ψ_{TVOR} Ønsket retning på den optiske pakke

$\dot{\psi}_{TVOR}$ Optisk pakke skal dreies med konstant dreierate $\dot{\psi}_{TVOR}$

$Tilt_N$, $Tilt_E$ Optisk pakke skal tiltes i henholdsvis Nord- og Øst-retning

Som det fremgår av ovenstående leverer plattformen all informasjon om eget fartøys bestikk og om retningen på den optiske pakke. Retning på pakken styres på sin side fra SM-3, enten ved at retningen settes direkte eller ved at svingeraten settes.

Data til og fra laser avstandsmåler er ennå ikke vurdert.

4.2 Data til og fra radar video ekstraktor (ARVEX)

På samme måte som for stabil plattform vil dataoverføring til og fra radarekstraktor skje via et buffer. Data fra et buffer vil prinsipielt inneholde denne informasjon:

$n + 1$ Antall datasett

ϕ_{REF} Referansevinkel

R_m, ϕ_m, S_m Sensordatasett, søkeområde nr m

R_s, ϕ_s, S_s Sensordatasett, søkeområde nr s

R_{m+n}, ϕ_{m+n}, S_{m+n} — Sensordatasett, søkeområde nr m+n

Data inn til radarekstraktor vil være:

$q + 1$ — Antall nye søkeområder

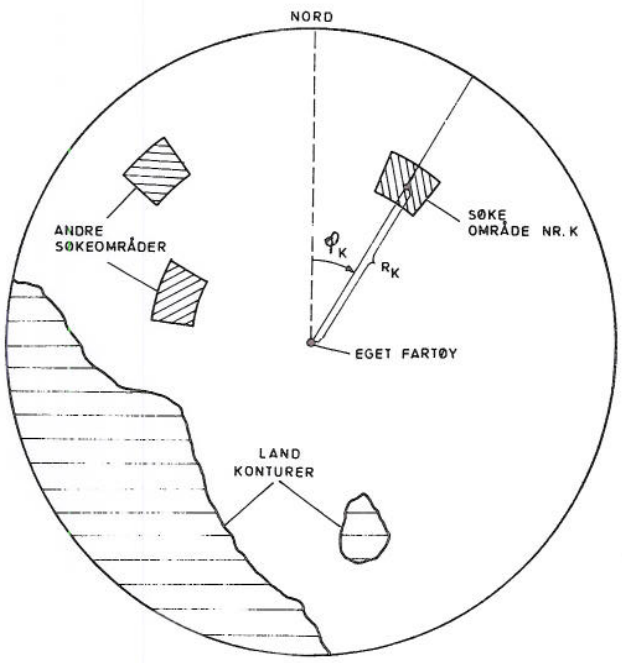
R_p, ϕ_p, S_p — Nytt søkeområde nr p

R_k, ϕ_k, S_k — Nytt søkeområde nr k

N_{p+q}, ϕ_{p+q}, S_{p+q} — Nytt søkeområde nr p+q

Data *fra* ARVEX vil det være naturlig å lese ut ved hvert klokkeinterrupt. Det vil være plass til 5 datasett i bufferet. Med det skulle en sikre at det ikke blir opphoping av data. Data til ARVEX vil det være naturlig å skrive ut hver gang et mål skal "startes opp" og siden en gang for hvert scan. Dette er for øvrig nærmere beskrevet i kapittel 6.2.

ARVEX søker etter mål i "vinduer" rundt hvert av punktene R_k, ϕ_k. Disse punktene skrives ut til bufferet (se figur 3.2) fra SM-3 og gir nye eller oppdaterer gamle søkeområder.



Figur 4.1 Søkeområder ved bruk av ARVEX

I datasettet som ARVEX genererer er:

n+1	Det totale antall mål R_s, ϕ_s som ekstraktoren har levert
ϕ_{REF}	Referansevinkel som viser hvor radarsweepet befant seg i det øyeblikk ARVEX leverte data fra et scan til bufferet. Grunnen til at denne vinkelen overføres er at man da kan vite når alle ekko innenfor ett vindu er innlest og prosessering kan begynne.
S_s	Forteller i hvilket område s målingen er funnet.

Det er foreløpig forutsatt at $N \leqslant 5$.

Data *til* radarekstraktor vil være av samme type.

4.3 Optisk sikte og radar søkemottaker

I tillegg til radar system og optisk pakke, vil sensorsiden bestå av et optisk sikte og radar søkemottaker (RSM). Det optiske sikte vil være fast montert til fartøyet og vil først og fremst være nyttig hvis den optiske pakke settes ut av drift. Bruken og tilkobling av disse sensorene er ennå ikke vurdert detaljert.

5 PROGRAMSYSTEMETS HOVEDSTRUKTUR

I dette kapittel vil programsystemets oppbygning bli gjennomgått, og det forutsettes at det som er skrevet i kapittel 1–4 er kjent. Rekkefølgen som stoffet er lagt fram på er i denne forbindelse vesentlig fordi erfaring fra dette og lignende systemer viser at kunnskaper om programsystemet bør basere seg på en forståelse av systemets funksjon og ikke minst hardwarekonfigurasjon.

5.1 Valg av programmeringsspråk

Alle operative programmer må skrives i symbolsk maskinkode (assembler). Dette er regningsvarende når en tar hensyn til at utstyret skal leveres i en større serie og ikke skal være forsynt med tromle eller platelager. Det har vært vurdert å kode hele eller deler av programsystemet i FORTRAN IV. Dette ville enten føre til høyere hardwareomkostninger i form av øket hukommelsesplass og større regnekapasitet eller til et system med dårligere ytelse. Noen forsøk viste at et program kodet i FORTRAN IV i beste fall ville kreve dobbelt så mye plass og dobbelt så mye CPU-tid som et program skrevet i assembly.

5.2 Tidsmonitor (jobbmonitor)

Programsystemet kan grovt sett deles i tre forskjellige kategorier:

– Tidsmonitor som styrer alle andre jobber (programmer)

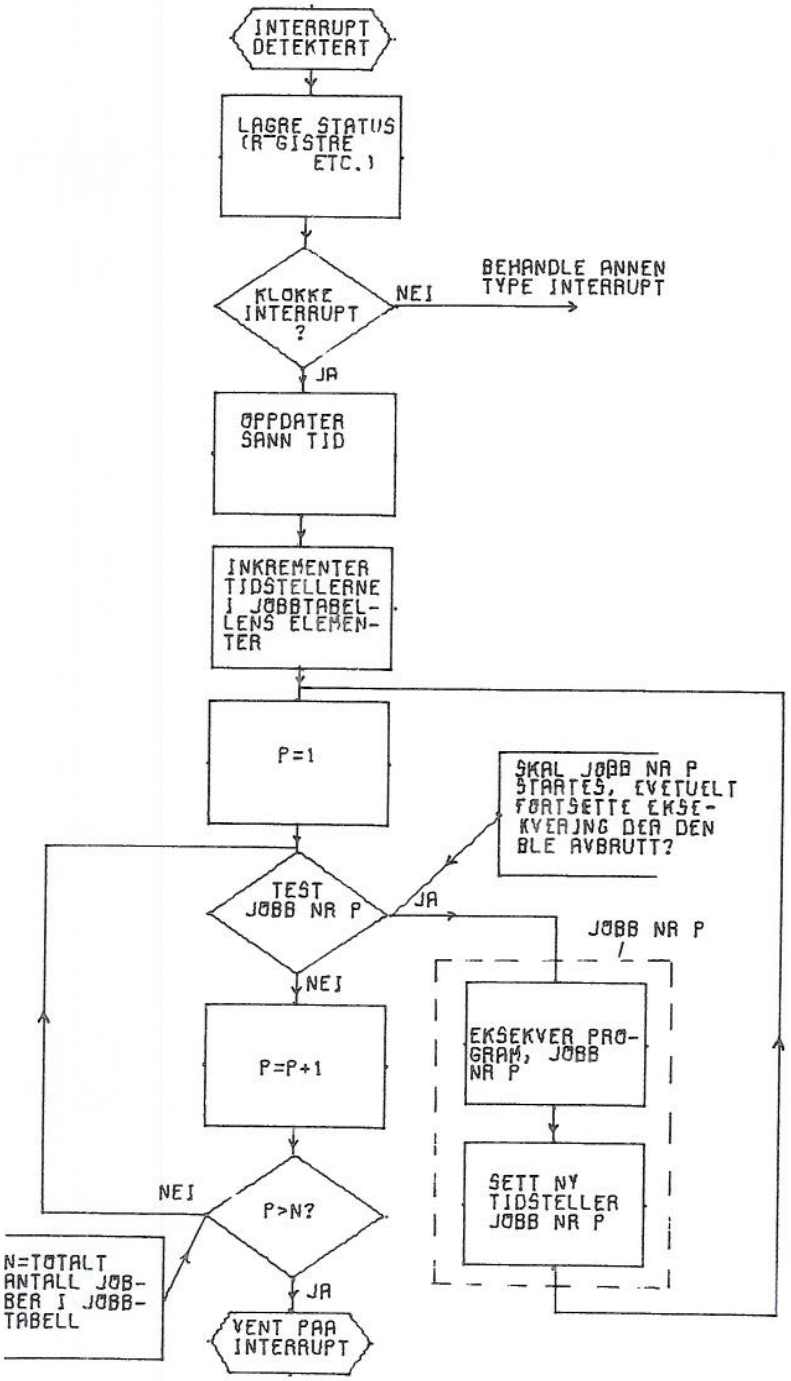
– Jobber som skal eksekveres umiddelbart p g a interrupt fra eksternt utstyr

– Jobber som tidsstyres fra monitoren

Den monitor som skal brukes er i prinsippet den samme som ble brukt i MSI-70U (5). Virkemåten til denne skal ganske kort beskrives.

Sentralt i beskrivelse av tidsmonitoren står den såkalte jobbtabell. Hver jobb i systemet står her oppført som ett *element*. Hvert element inneholder 3 ord. Det ene ordet er en *tidsteller* som forteller hvor lang tid det er til vedkommende jobb skal eksekveres. Det andre ordet er en adresse til vedkommende jobb (d v s hvilken coreadresse programmet for jobben starter i). Det tredje ordet er uten betydning i denne sammenheng. *Elementenes innbyrdes plassering i jobbtabellen angir jobbens prioritet.* D v s at dersom jobb nr n og jobb nr n+1 skal eksekveres samtidig, vil jobb nr n bli eksekvert først.

Tidsmonitorens virkemåte er forsøkt illustrert på figur 5.1. Hovedhensikten med flytdiagrammet er ikke å gjengi eksakt hvordan monitoren arbeider, men prøve å beskrive de viktigste prinsipper.



Figur 5.1 Flytdiagram, jobbmonitor

Fra toppen av er sekvensen som følger: Når interrupt er detektert lagres registre og annen status unna, og en kan teste hvilke eksternt utstyr som har gitt interrupt. Hvis dette er sanntidsklokken (som gir interrupt hver 0,1 sek), oppdateres "sann tid" i datamaskinen. Deretter inkrementeres tidstellerne for hver jobb. Jobbtabellen søkes nå igjennom fra element nr 1 til N hvor N er totalt antall elementer. Dersom en tidsteller er positiv, er den tilsvarende jobben klar for eksekvering og startes opp. *Som en del av en jobb inngår også en resetting av tidsteller.* Når jobben er ferdig, starter monitoren å søke fra toppen av jobbtabellen igjen. Når *alle* jobbene som skal eksekveres, er ferdige, står systemet og venter på nytt interrupt.

Noe som ofte vil inntreffe er at et interrupt kommer mens en jobb er under eksekvering. I så fall vil jobben som er under eksekvering bli avbrutt, og hvis det er et klokkeinterrupt, vil tidstelleren bli inkrementert og søkingen gjennom jobbtabellen blir gjennomført som forklart foran. En viktig forskjell er det imidlertid mellom tidligere avbrudte programmer og programmer som ikke er blitt avbrudt. Et program som ikke er avbrudt, vil bli startet fra vanlig startadresse, men et avbrudt program vil restartes fra det punkt hvor det ble avbrudt. Ved restart vil monitoren også sørge for at registre og annen status er som ved avbrudd.

Dersom en får et interrupt fra annet eksternt utstyr enn sanntidsklokken, vil en kunne innkalle jobber som *ikke* er oppført i jobbtabellen. Disse *jobbene* har *høyere* prioritet enn jobbene i jobbtabellen og bør kreve liten CPU-tid.

Den valget tidsmonitor er enkel og ved bruken av denne står en overfor en rekke begrensninger som en ikke kan komme inn på her. På plussiden kommer at den er enkel å forstå og bruke, den krever liten plass (ca 400 core lokasjoner) og overhead med denne monitoren er liten (2–4%), men dog avhengig av antall elementer i jobbtabellen.

5.3 Konsollmonitor

Foruten å kunne kommunisere med systemet ved hjelp av rulleball, vil operatøren(e) ha ett eller flere konsoller med forskjellige typer knapper og vendere. Disse er vist på figur 5.2.

Figur 5.2 Knapper og vendere

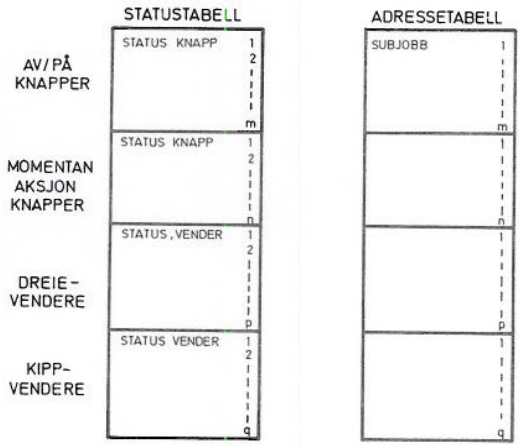
Type I er en vanlig knapp som kan ha status AV eller PÅ. *Statusstyringen* (indikert f eks ved lys) *skjer fra programsystemet.* Status kan dermed forandres i forbindelse med at knappen aktiveres, men statusforandring kan også skje helt uavhengig av dette.

Type II er en "momentan aksjon" knapp som bare vil forandre status i forbindelse med aktivering. Knappen vil vanligvis være i AV-status, og ved aktivering vil den komme i PÅ-status. Det vil nå typisk være en engangs jobb som skal utføres. Når jobben er utført vil knappen igjen settes i AV-status. All statusforandring skjer også her fra programsystemet.

Type III er en vanlig roterende vender hvor selve statusforandringen skjer via hardware, men hvor statusforandringen detekteres og lagres i programsystemet.

Type IV er en kippbryter som kan ha tre posisjoner. I midtposisjon er knappen AV og ingenting skjer. I hver av ytterstillingene vil ett register telles henholdsvis opp eller ned. Dette skjer fra programsystemet.

Den delen av programsystemet som skal lese av knapper (avføle aktivering), sette status og kalle opp programmer som skal eksekveres i forbindelse med statusavlesing er i denne rapporten omtalt som *konsollmonitor*, forkortet KM. Dette er på sin side et program som ligger med høyeste prioritet i tidsmonitorens jobbtabell.



Figur 5.3 Status- og adressetabell for konsollmonitor

Konsollmonitoren er tabelldrevet, og styres av to tabeller vist på figur 5.3. Den ene er en statustabell som inneholder ett ord for hver knapp/vender. For knappene vil statusord inneholde slike ting som informasjon om knappen er AV eller PÅ, om aktivering er tillatt etc. For dreievenderne er statusordet vesentlig enklere og angir bare hvilken stilling venderen står i. For kippvenderne vil statusordet være et register som telles opp eller ned når venderen blir satt i den ene eller andre ytterstilling.

Den andre tabellen er en adresse- eller subjobbtabell. For hver knapp/vender inneholder denne en adresse som er en startadresse for det *spesielle* programmet som skal eksekveres når knappen/venderen er aktivert.

Virkemåten til KM er vist på figur 5.4. Av dette skulle det fremgå at arbeidet i forbindelse med aktivering av en knapp/vender er delt i to:

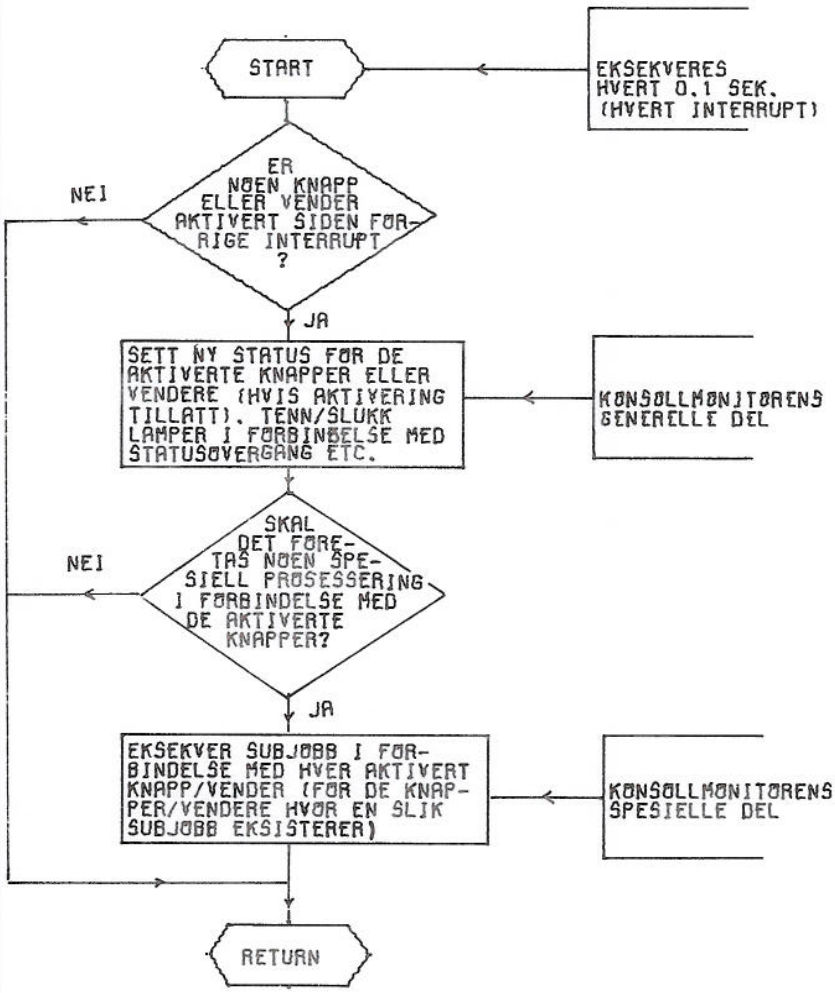
- For det første har vi den generelle delen som angår slike ting som setting av ny status, tenning/slukking av lampelys etc. For en knapp av en bestemt type er den programsekvensen felles.

- I tillegg har vi den spesielle delen av konsollmonitoren. Dette er i praksis et spesielt program for hver knapp, og startadressen til dette programmet er angitt i adressetabellen slik som vist på figur 5.3.

Den generelle delen av KM er implementert, og forsøk hittil har vist at denne oppbyggingen er oversiktlig og enkel, og det er fremfor alt enkelt å modifisere bruk av de forskjellige knapper og vendere.

Det må til slutt bemerkes at man ikke alltid kan utføre all prosessering i forbindelse med aktivering av en knapp/vender i forbindelse med eksekveringen av knappens/venderens subjobb (figur 5.3). KM har høy prioritet og kan således bare eksekvere viktige oppgaver. Når jobber med lav prioritet skal eksekveres som følge av aktivering av knapp/vender, kan disse aktiveres som vanlige jobber fra tidsmonitorens jobbtabell ved at det testes på knappens statusord etc. I bruken av dataskjermsystemet vil det vises et eksempel på hvordan jobben i forbindelse med aktiveringen av en knapp må

splittes opp i to. Her vil én del av jobben være en del av KM, mens en annen del vil være en vanlig jobb i tidsmonitorens jobbtabell.



Figur 5.4 Flytdiagram, konsollmonitor

5.4 Dataskjermsystemet

En forståelse av hvordan programsystemet er bygget opp kan først og fremst erverves gjennom en forståelse av sensordatainnlesning og behandling og dataskjermsystemet. I dette kapittel skal først en del av det som skal vises på dataskjermen beskrives, og dernest hvordan programsystemet omkring dette er bygget opp.

Den informasjon som skal vises i forbindelse med eget fartøys bestikk og målfølgingen er vist på figur 5.5. Det bør bemerkes at dette ikke er noe detaljert forslag til *hva* som skal vises og *hvordan*. Forandringer vil imidlertid ikke kunne få noen store konsekvenser for systemets virkemåte.

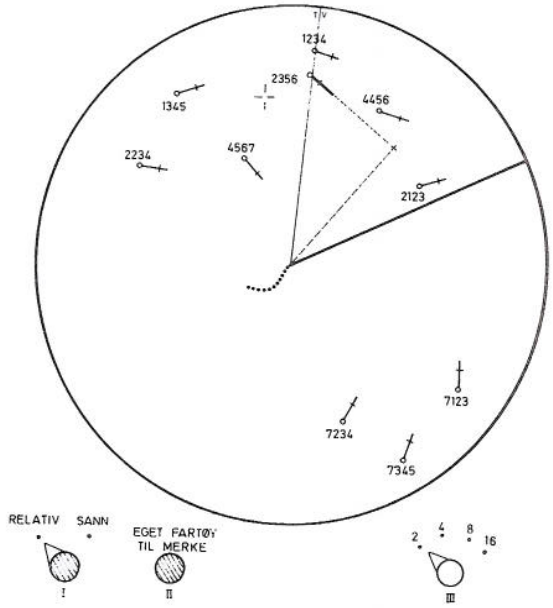
Bildet består av følgende deler:

– Eget fartøys posisjon og hastighet og eget fartøys historie. Hvis nåtiden er gitt som t, er historien gitt i form av punkter som angir posisjonene ved tidspunktene $t-\Delta T$, $t-2\Delta T$, $t-3\Delta T$, ... o s v. Fartøyets kurs er angitt som en vektor ut til kanten av skjermen og hastigheten ved et merke på kursvektoren, slik at hastigheten er proporsjonal med avstanden mellom fartøy og merke.

– Vektoren som er trukket med svak intensitet ut mot kanten av skjermen fra eget fartøy viser retning av den optiske pakke (som er montert på stabil plattform). Ved hjelp av den vektoren kan operatørene ved dataskjermen se hvor passiv sensor operatør konsentrerer oppmerksomheten.

– På skjermen finnes et antall målfartøyer. Disse er markert med posisjon og hastighet på samme måte som eget fartøy. Kursvektoren er imidlertid ikke trukket ut til kanten av skjermen. Hvert målfartøy er identifisert ved et fire siffers nummer.

– Korset til venstre for TV-vektoren er rulleballmerket, og posisjoneres ved hjelp av rulleballen.



Figur 5.5 Radar og taktisk dataskjerm

De stiplede linjene markerer møtet mellom eget fartøy og det ene målfartøyet hvis kurs og fart er som i øyeblikket.

På figur 5.5 er det også tegnet inn en knapp og to vendere som er av primær betydning for bildets utseende og programsystemets oppbygging. "SANN/REL" venderen angir om eget fartøy skal bevege seg "sant" på skjermen eller om bevegelsen skal være relativ, d v s eget fartøy ligger stille (bare med hensyn til posisjon) mens omgivelsene forflytter seg. "EGET FARTØY TIL MERKE" aktiveres når en ønsker at eget fartøy skal posisjoneres over rulleballmerket (omgivelsene vil selvfølgelig også flyttes) og til slutt finnes en vender (III) som angir enten dataskjermens radius skal tilsvare 2, 4, 8 eller 16 km.

For å forstå hvordan filene for dataskjermsystemet er bygget opp kreves det kjennskap til dataskjermprosessorens virkemåte. En beskrivelse av en aktuell prosessor er gitt i (1).

Dataskjermfilene (DSF) er bygget opp slik at det skal være så enkelt som mulig å arbeide med filene når en av knappene/venderne på figur 5.5 aktiveres. Likedan er det tatt hensyn til at DSF skal være enkle å oppdatere når eget fartøy forflytter seg.

På figur 5.6 er filstrukturen vist. Systemet består av en hovedfil hvor en setter koordinatene for de forskjellige delbildene som det totale bildet på dataskjermen kommer til å bestå av. For hver koordinatsetting vil det være et hopp til riktig delbilde (subbilde).

Det bildet som er vist på figur 5.5 kommer således til å bestå av følgende delbilder:

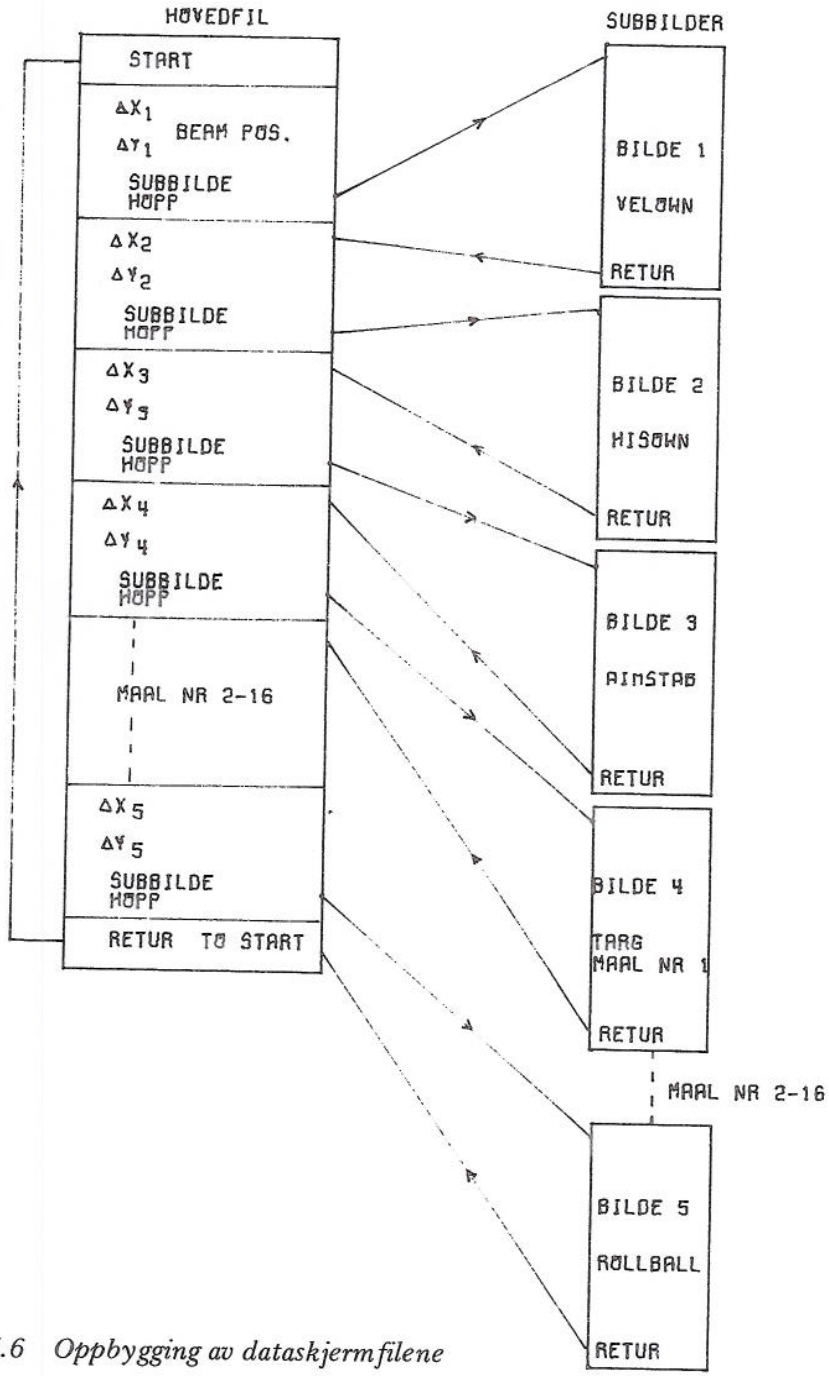
Bilde 1 Hastighetsvektor, eget fartøy (VELOWNP)

" 2 Posisjonshistorie, eget fartøy (HISOWNP)

" 3 Retningsvektor, stabil plattform (AIMSTABP)

" 4 Hastighetsvektor og identifikasjonsnummer for opp til 16 målfartøyer (et delbilde for hvert fartøy) (TARGP). For enkelhets skyld er dette angitt bare for et mål, men prinsipielt er dette uten betydning.

" 5 Rulleballmerket (ROLLBALLP)



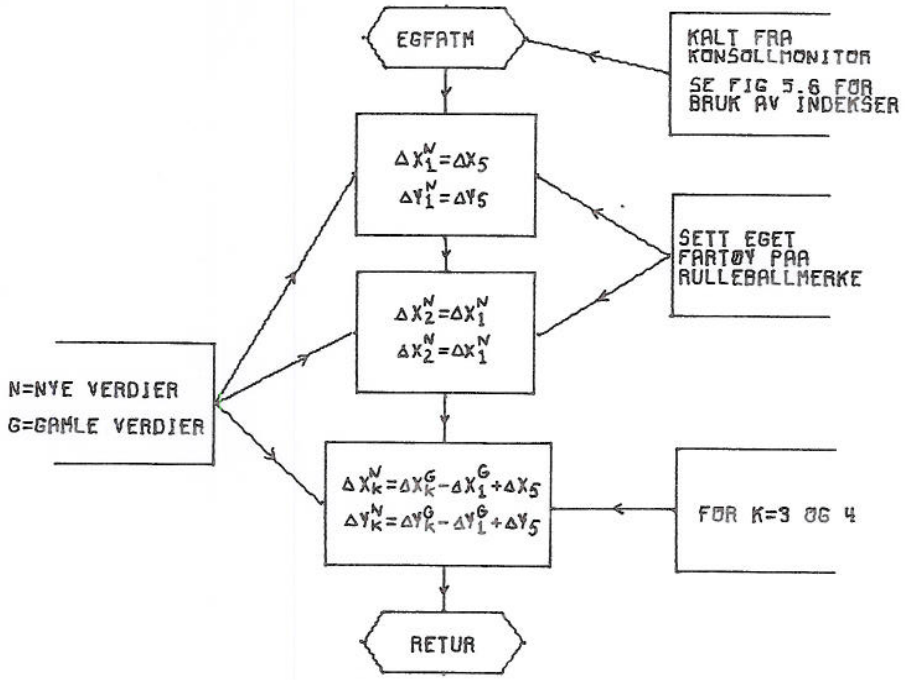
Figur 5.6 Oppbygging av dataskjermfilene

Tilsvarende vil det for hvert delbilde finnes et program, hvis eneste oppgave det er å generere delbildet ut i fra en gitt datastruktur. (Navnet på disse programmene er skrevet i parentes.) Det er på dette punkt viktig å peke på den prinsipielle forskjellen på datastruktur på den ene side og DSF på den andre. Mens datastrukturen til enhver tid inneholder systemets totalinformasjon, vil DSF kun være utledet av denne i den utstrekning det er ønskelig å vise denne informasjon på taktisk dataskjerm.

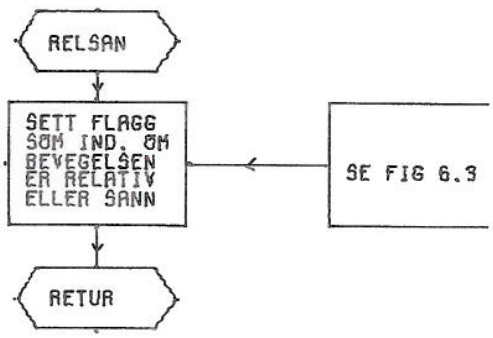
De "jobbene" som genererer DSF vil dels være jobber i KM, dels være jobber som kalles opp fra jobbmonitor. Fra KM kalles de tre programmene EGFATM, RELSAN og SKALA i forbindelsene med aktivering av knappene I, II og III på figur 5.5. Fra jobbmonitoren vil en kalle opp de forskjellige programmene VELOWNP, HISOWNP o s v i forbindelse med generering av de respektive delbildene VELOWN, HISOWN o s v som det totale bildet på dataskjermen består av.

I prinsippet kan de forskjellige programmene VELOWNP, HISOWNP o s v få hvert sitt element i jobbtabellen. I de tilfellene at jobbene kan ligge med samme prioritet bør imidlertid programmene kalles opp som subrutiner fra ett element da dette vil være tidsbesparende. For hvert program vil det være bekvemt å ha ett ord (flagg), slik at når dette ordet er 1, skal programmet eksekveres. Disse ordene (flaggene) i det følgende kalles VELOWNXX for program VELOWNP, HISOWNXX for program HISOWNP o s v.

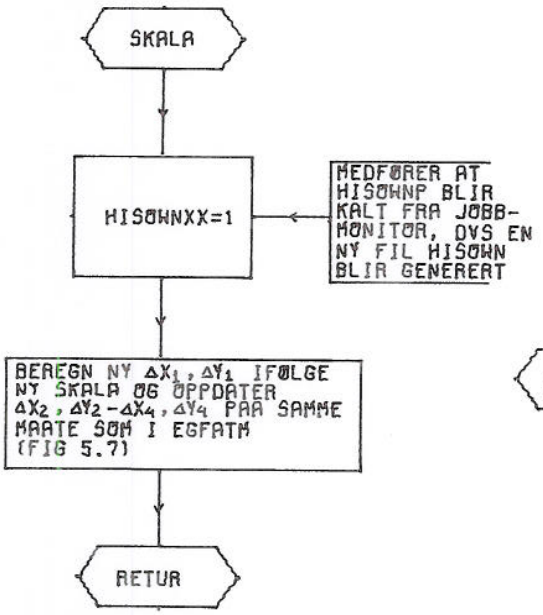
På figur 5.7 til 5.9 er det så vist flytdiagrammer for de enkelte programmer. Av de tingene som det er spesielt verdt å legge merke til er at de programmene som kalles opp fra KM, EFFATM, RELSAN og SKALA er svært lite tidkrevende. For det andre legger en merke til at det eneste program som skal kalles inn ved skalaforandring er HISOWNP. Det er her forutsatt at slike ting som fartsangivelser etc ikke skal forandres selv om skala forandres. Skal systemet vise annen informasjon som må skaleres med skalavelgeren er det intet i veien for det ved at flagget for de forskjellige DSF-genererende program settes slik at disse blir kalt inn fra jobbmonitor på samme måte som HISOWNP.



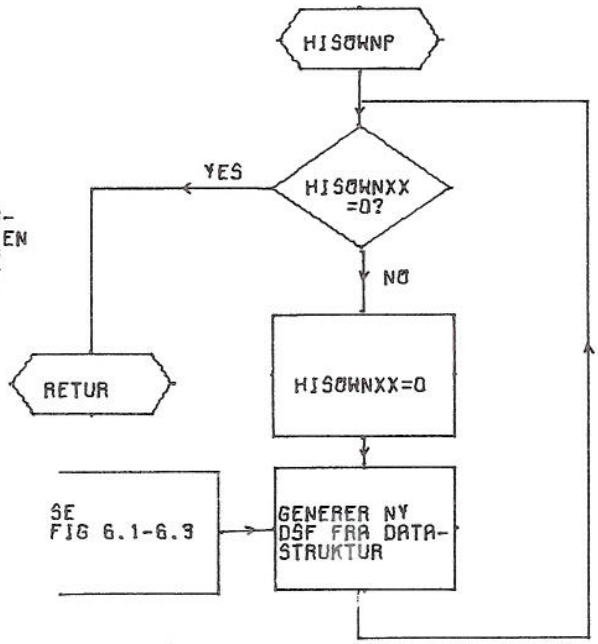
Figur 5.7 Flytdiagram for program når knapp "EGET FARTØY TIL MERKE" er aktivert



Figur 5.8 Flytdiagram for program når vender for rel/sann bevegelse er aktivert



Figur 5.9 Flytdiagram for program når vender for skalaendring er aktivert

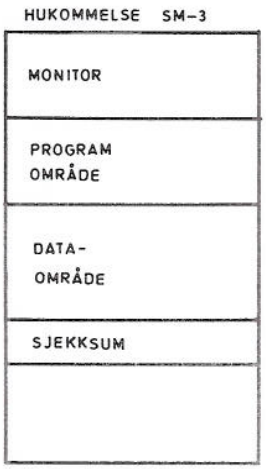


Figur 5.10 Flytdiagram for generering av dataskjermfil, eget fartøy

På figur 5.10 er flytdiagrammet for programmet som skal generere eget fartøys historie vist (HISOWNP). Det man spesielt bør legge merke til er bruken av flagget HISOWNXX. Dette blir nullsatt før generering av ny DSF. Etter at genereringen er ferdig blir flagget testet på nytt. Grunnen til dette er at mens genereringen skjer, kan programmet bli avbrutt av andre program med høyere prioritet, og HISOWNXX kan bli satt på nytt.

Med den fil og programstruktur som er foreslått for konsoll- og dataskjermsystem vil en oppnå å få en hensiktsmessig oppdeling av et sammensatt problemkompleks i mindre deler med en forholdsvis svak kobling mellom delene. Dette skulle sikre enkel implementering og fremfor alt enkel modifikasjon av programsystemet dersom dette skulle være nødvendig.

5.5 Selvsjekk av programsystemet



Figur 5.11 Plassering av data og program i hukommelsen

Erfaring viser at det er av stor betydning til ethvert tidspunkt å kunne verifisere at programsystemet i datamaskinen er intakt og i samme forfatning som ved første gangs innlesing. Dette kan gjøres enkelt og pålitelig ved at programsystemet deles fysisk i hukommelsen i en programdel og en datadel. Programdelen vil være tidsinvariant, mens datadelen vil forandres når programsystemet arbeider. Et bilde av hukommelsen er vist på figur 5.11.

Systemet sjekkes ved en såkalt sjekksum. Dette er summen (aritmetisk) av alle de lokasjonene som programdelen består av. Sjekksummen kan beregnes av et spesielt program etter at programsystemet er plassert i hukommelsen. Siden vil dette programmet ikke benyttes.

Som siste element i jobbtabellen kan man så ha et program som foretar en beregning av sjekksum og sammenligner med fasit som er utregnet av det nevnte program. En vil dermed ha en løpende beregning av sjekksum.

Dette systemet vil imidlertid ikke være brukbart dersom datadelen blir ødelagt. En kan allikevel spare innlesing av nyttprogram ved følgende prosedyre: Programsystemet utstyres med en *initialiseringsrutine*. Denne beregner først sjekksum av programdelen. Hvis denne er i orden *nullstilles* hele dataområde hvoretter dataområdet settes i startmodus v h a konstanter fra programdelen.

6 SENSORDATABEHANDLING OG MÅLFØLGING

I dette kapittel vil data- og programstrukturen for behandlingen av sensordata bli gjennomgått. Til dette kapittel er også lagt inn beskrivelse av hvordan data om eget fartøys kurs og posisjon avleses og behandles. Spesielt vil dataskjermfilen for eget fartøys historie beskrives.

6.1 Data og programstruktur, eget fartøys bestikk

Da stabil plattform slik som beskrevet i kapittel 4.1 vil levere filtrerte data om eget fartøys hastighet, rull og stamp, vil det programmet som skal oppdatere eget fartøys bestikk bli relativt enkelt. Hvis bestikket oppdateres med tiden Δt (som f eks kan være 0,5 sek), vil posisjonshistorien vist på dataskjermen ha et vesentlig lengre samplingsintervall ΔT (f eks 5 sek).

Flytdiagram og datastruktur er vist på figur 6.1, 6.2 og 6.3. På figur 6.1 er sammenhengen mellom posisjonshistorien slik den vises på skjermen og datastrukturen vist. Datastrukturen er inneholdt i filen OWNSYC, mens DSF er den før omtalte HISOWN. (Her er benyttet 5 punkter, men i virkeligheten vil dette være flere.) På figur 6.2 er omtrent det samme vist for dataskjermfilen. Det er her en prinsipielt viktig forskjell. Mens hvert element i datastrukturen gjenspeiler avstanden til forrige punkt, vil hvert element i DSF inneholde avstanden til det felles utgangspunkt.

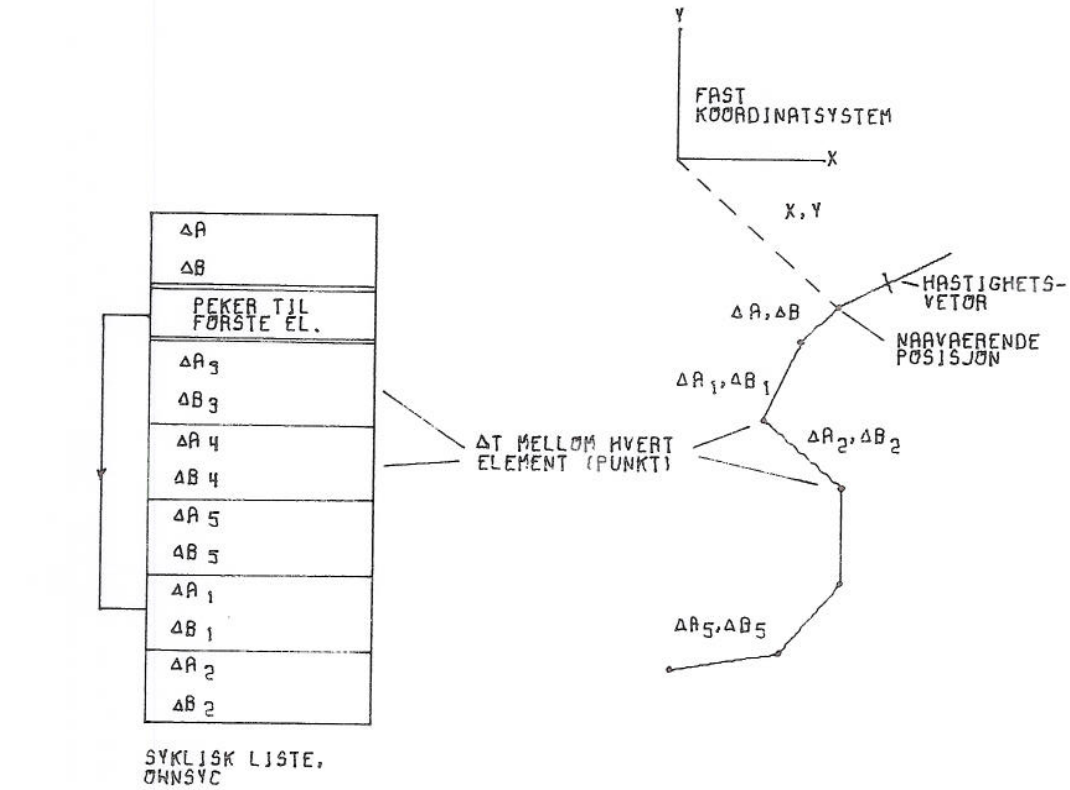
Grunnen til dette er at i OWNSYC vil oppløsningen være på 15 bits, mens oppløsningen i HISOWN vil være 10 bits. Man risikerer dermed at små inkrementer ved konvertering fra OWNSYC til HISOWN blir forkortet bort. Ved å benytte nevnte metode vil dette fortsatt kunne skje, men feilen vil ikke akkumuleres fra det ene punktet til det neste.

Det en taper på denne måten er at ved hvert tidsintervall ΔT, må HISOWN regenereres totalt. Hadde denne filen hatt samme oppbygging som kildefilen OWNSYC, ville man også her kunne nøye seg med å fjerne det eldste elementet og sette inn *ett* nytt.

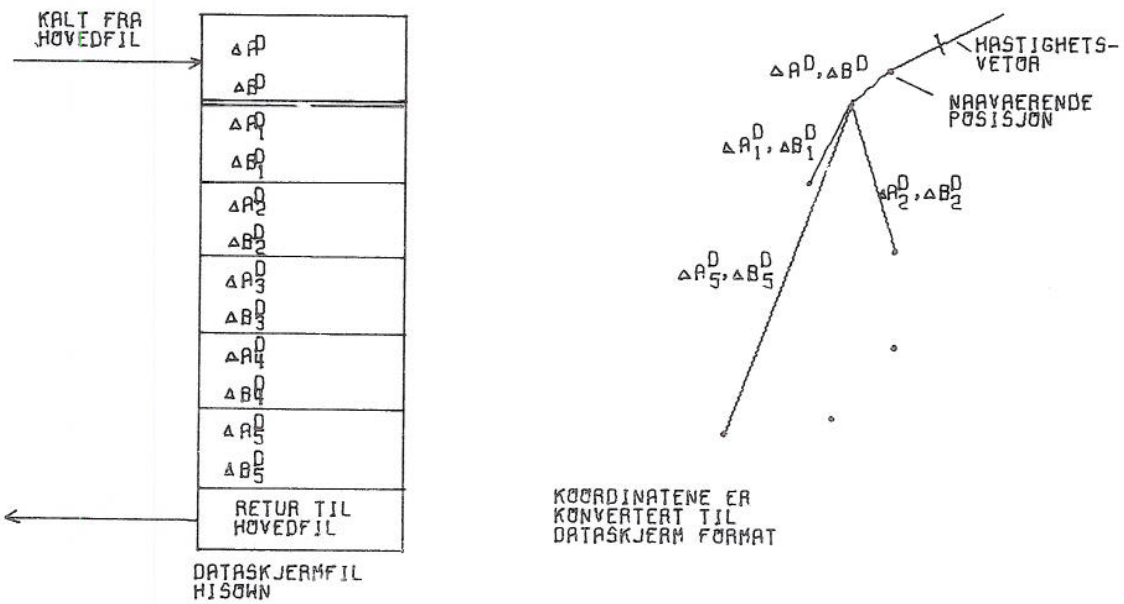
Flytdiagram for program som genererer datastruktur for eget fartøys kurs historie (DEADRECK) er vist på figur 6.3.

Det bør til slutt bemerkes at programmet HISOWNP er nøyaktig det samme som det omtalte i kapittel 4 i forbindelse med forandring av skala på skjermen, og dette programmet aner forsåvidt ikke hvilket program som har satt HISOWNXX.

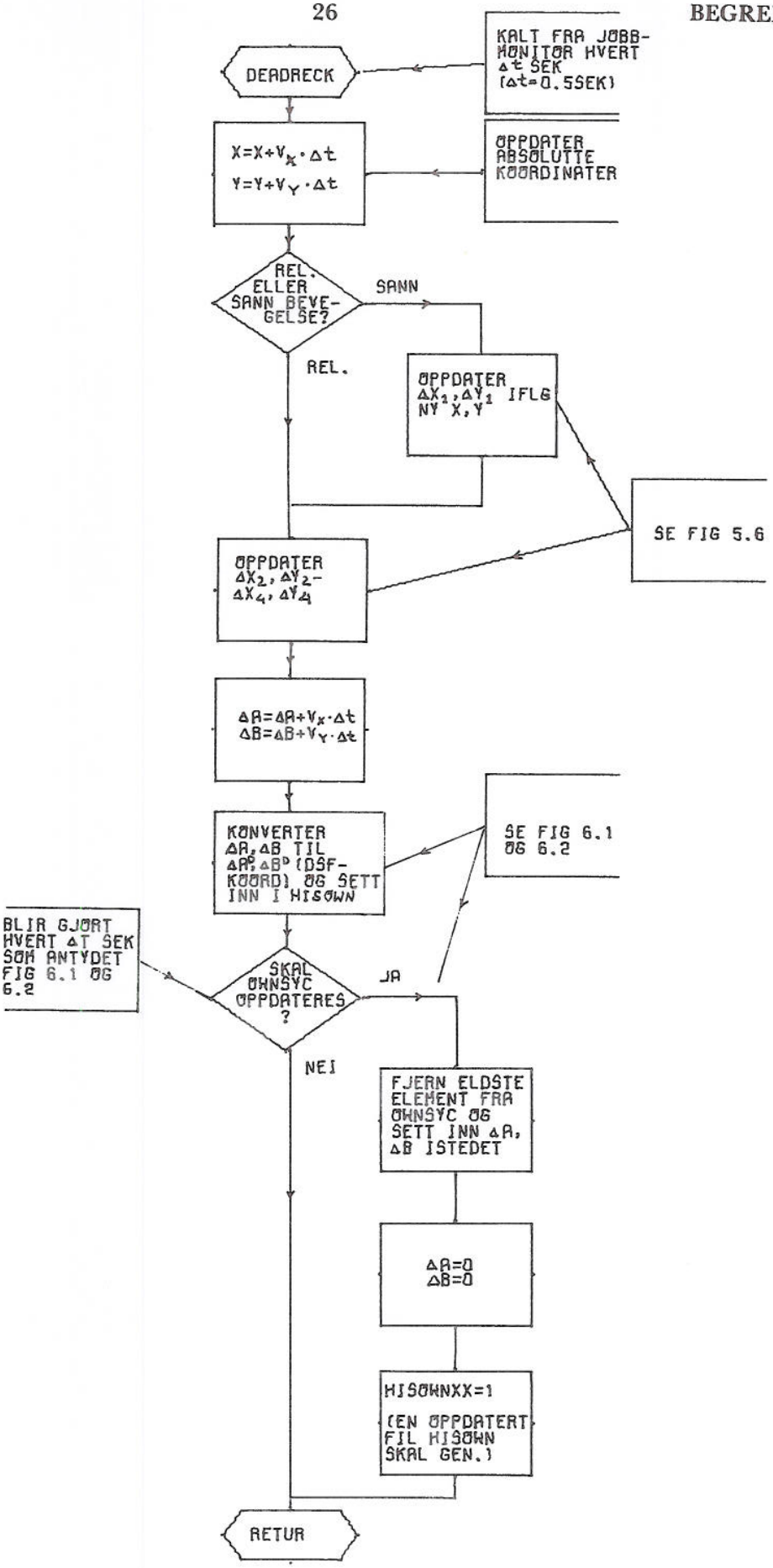
Med hensyn til programmet HISOWNP, vil dette være relativt enkelt, da det i prinsippet vil bestå i å hente ett og ett datapar fra OWNSYC (dette kan gjøres v h a spesielle subrutiner), addere til svaret fra forrige opphenting, konvertere til dataskjermformat og sette svaret inn i HISOWN.



Figur 6.1 Datastruktur, eget fartøys kurshistorie



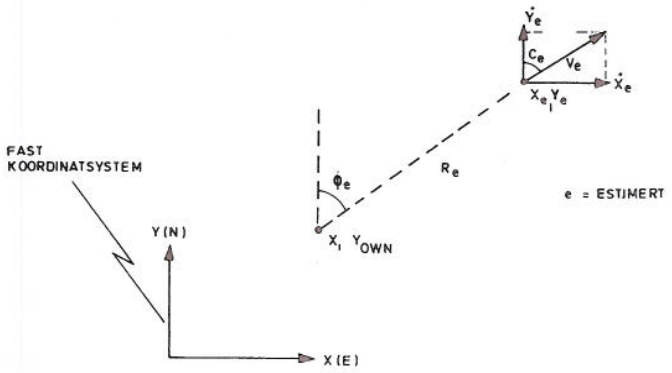
Figur 6.2 Dataskjermfil, eget fartøys kurshistorie



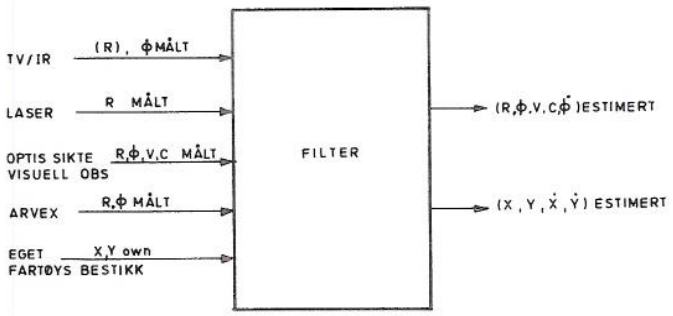
Figur 6.3 Flytdiagram for program som genererer datastruktur, eget fartøys kurshistoric

6.2 Generell beskrivelse av målfølgingen

Før totalsystemet skal beskrives, er det nødvendig med en prinsipiell beskrivelse av hvordan en finner de nødvendige parametre for et målfartøy. Dette er vist på figur 6.4 og 6.5.



Figur 6.4 Målfølgingsparametre



Figur 6.5 Målfølging, prinsippskisse

På grunnlag av målinger fra TV/IR, ARVEX, LASER og/eller det vi kan observere med optisk sikte eventuelt øyet og i tillegg eget fartøys posisjon, kan man ved hjelp av et filter finne et optimalt estimat X, Y og $\dot{X}$, $\dot{Y}$ som angir henholdsvis posisjon og hastighet i X- og Y-retning (som tilsvarer Nord- og Øst-retning). Filteret arbeider under forutsetning av at målfartøyet beveger seg med konstant hastighet. Dersom posisjonen i tidspunktet T er kjent, kan posisjonen ΔT senere lett finnes som

$$X_e(T+\Delta T) = X_e(T)+\dot{X}_e \cdot \Delta T$$

$$Y_e(T+\Delta T) = Y_e(T)+\dot{Y}_e \cdot \Delta T$$

Omregningen, dersom man ønsker å benytte polarkoordinater, er da triviell.

6.3 Data- og programstruktur, sensordatabehandling

Et forslag til oppbygging av data og programstruktur for behandling av sensordata er vist på figur 6.6. Datastrukturen er her vist i form av firkantede blokker mens programmene er inntegnet som sirkler. Alle programmene unntagen det som skal behandle data fra stabil plattform (STABEH) kalles opp fra jobbmonitoren. STABEH vil derimot sannsynligvis enklest kalles fra konsollmonitoren (KM) i det styring av inn- og utlesing her stort sett vil skje manuelt fra konsollet. Oppgaven til STABEH blir å skrive peiling og/eller peilerate ut til buffer for stabil plattform. Dernest skal STABEH lese peiling fra buffer dersom peiling skal leses inn til et valgt mål.

Peileraten ($\dot{\psi}$) kan da direkte være fremkommet på grunnlag av målestimatet for det målet man peker på, og dermed vil plattformen automatisk følge målet. $\dot{\psi}$ kan også være fremkommet ved bruk av f eks rulleball eller joystick. Dette blir nærmere omtalt i kapittel 7.5.

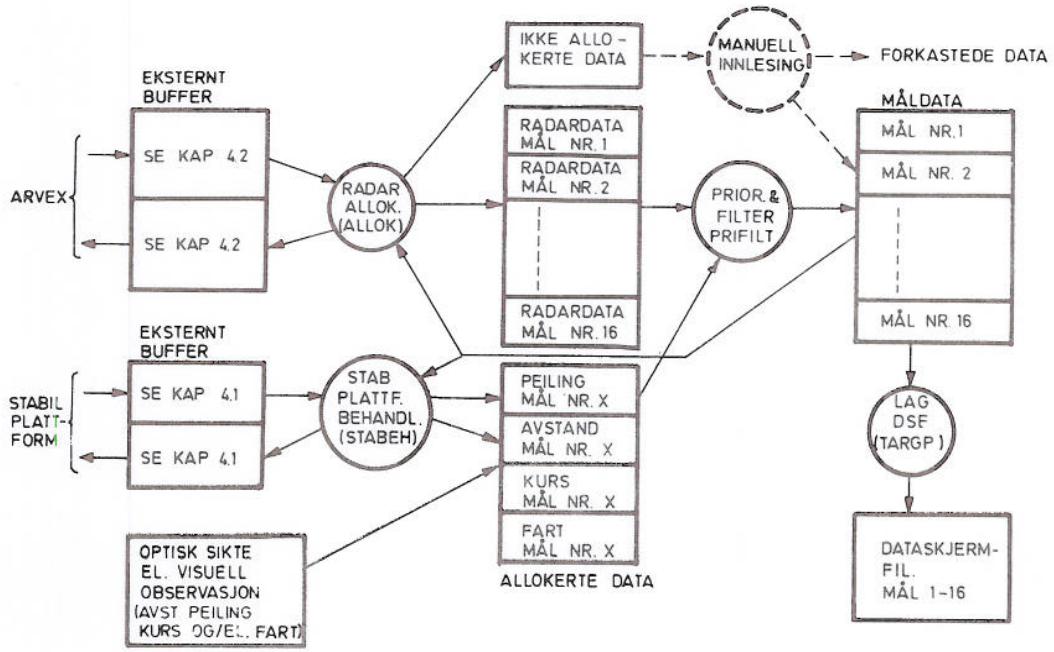
Figur 6.7 viser programmet som skal hente opp de allokerte sensordata og kalle opp filteralgoritmen. Dette programmet må *prioritere* målinger da det i filen for allokerte målinger vil kunne ligge mange målinger samtidig som skal filtreres. Utvelgelsen her vil være sterkt influert av systemspesifikasjonen.

Det må her tas hensyn til om det skal være spesielt prioriterte mål etc. En bør her spesielt være oppmerksom på at data kan tapes ved at nye data f eks fra radar kommer inn til filen for allokerte radardata før de gamle er blitt prosessert. I hvert fall for mål med lav prioritet vil det være naturlig syklisk å gjennomløpe filen for allokerte radardata. I forslaget figur 6.7 er data fra laser og plattform gitt høyest prioritet. Dette er gjort fordi dataraten fra disse sensorene er vesentlig lavere enn fra radar, samtidig som disse sensorenes nøyaktighet vil være større enn for radar.

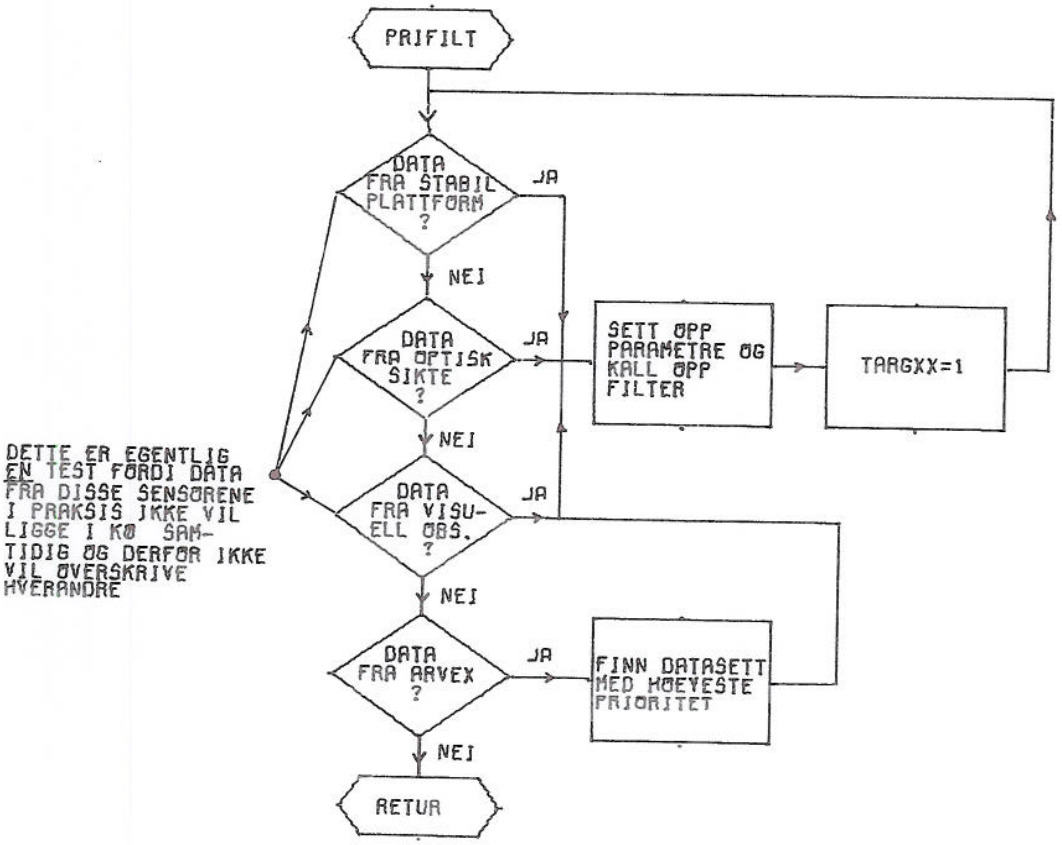
Det mest kompliserte av programmene vil være det som kommuniserer med radarekstraktor (ALLOK). Dette er vist på figur 6.8 og en hjelpefigur er tegnet på figur 6.9. Her opererer en med to "rammer', for hvert mål. Den største rammen (δR, $\delta\Phi$) benyttes av radarekstraktor, slik at alle ekko innenfor denne rammen vil bli lest inn til hovedregnemaskinen, mens den minste rammen (δr, $\delta\phi$) vil være den rammen som benyttes ved den endelige allokeringstest. Av figur 6.8 går det fram at ALLOK *både* leser data fra ARVEX og modifiserer senteret for rammene etter hvert som målene forflytter seg. Det er ikke tatt noen stilling til hva som skal skje dersom flere ekko kommer inn fra samme ramme, eventuelt at et ekko blir borte fra en ramme. Muligens vil det her være behov for manuelt å kunne gripe inn slik som antydet på figur 6.5.

Programmet TARGP som skal generere dataskjermfiler for fremvisning av målfartøyene er i prinsippet en ren konvertering. Dette programmet er det ikke tegnet flytdiagram for.

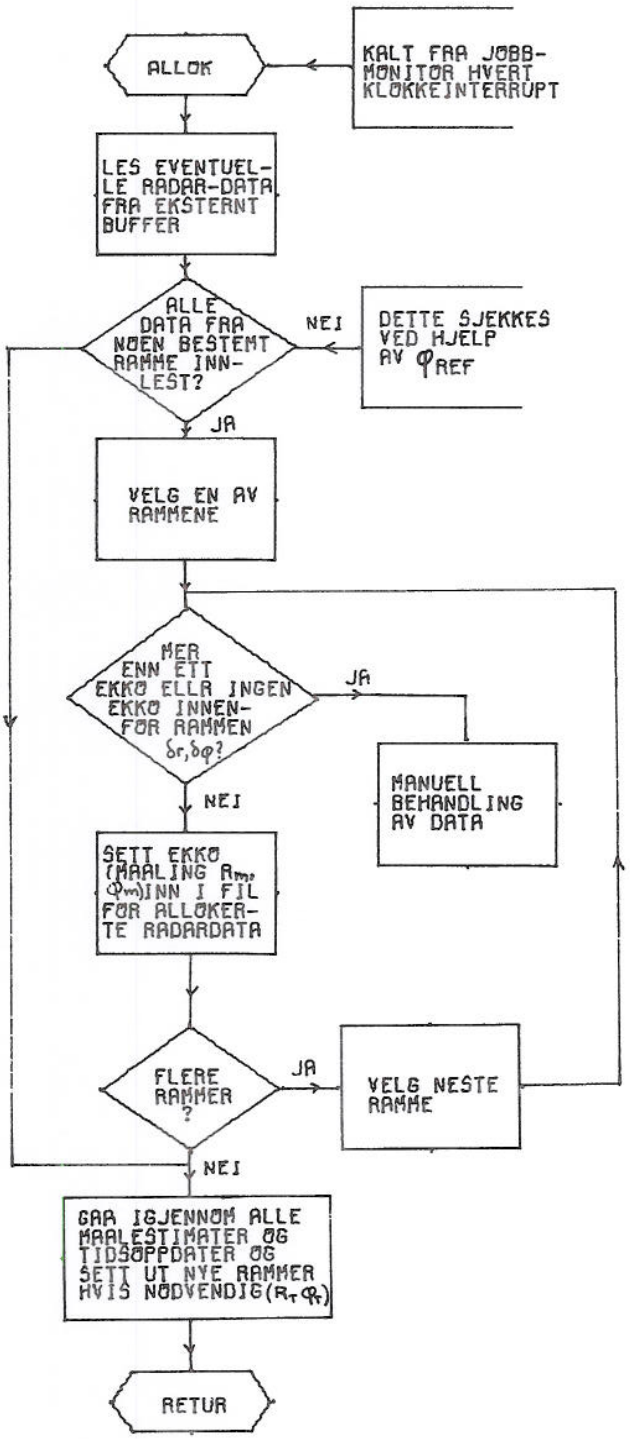
Jobbtabellens lay-out er vist på figur 6.10 og viser innbyrdes prioritet av de jobbene som her er omtalt. PRIFILT er her satt med laveste prioritet. Dette er gjort fordi dette er det enkeltprogrammet som vil kreve mesteparten av prosessortiden, og om dette programmet var plassert lenger foran ville jobbene bak kunne blokkeres lange tider av gangen. Hvis det skulle vise seg at kravet om rask prosessering av sensordata ikke kan forenes med den lave prioritet, kan dette problemet løses ved at jobben splittes i to. Jobben med høyest prioritet kan da behandle data for de prioriterte mål, mens de uprioriterte mål kan behandles av jobben med laveste prioritet. En ulempe med dette er at kravet til stackplass (midlertidig lagerplass) vil øke fordi filteralgoritmen vil kunne kalles fra begge jobbene.



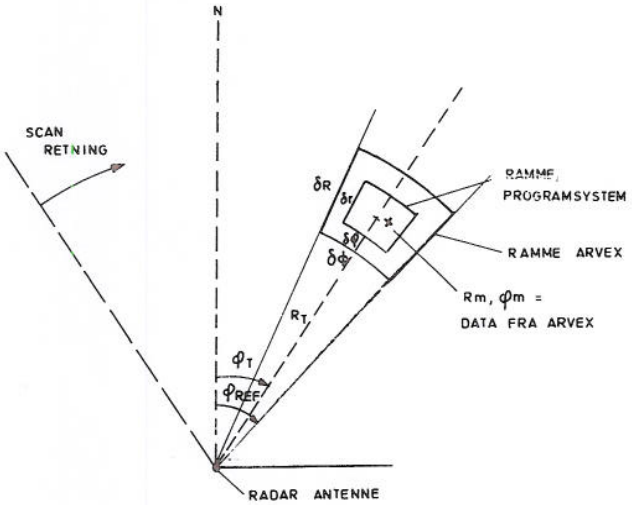
Figur 6.6 Oversikt over data og programstruktur for behandling av sensordata



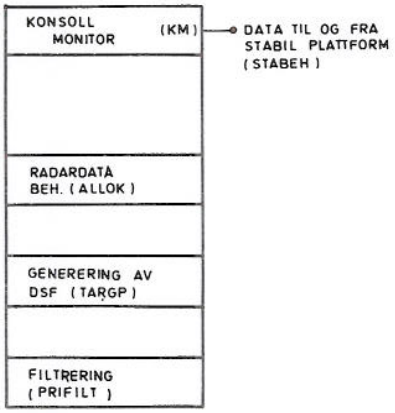
Figur 6.7 Flytdiagram for program som finner høyest prioriterte måling og kaller opp filteralgoritme



Figur 6.8 Flytdiagram for program som skal kommunisere med ARVEX



Figur 6.9 Bruk av rammer ved radartracking



Figur 6.10 Plassering av programmer i jobbtabell

7 UTPRØVING AV SOFTWARE OG MANN-MASKIN KOMMUNIKASJON

I det systemet som er skissert foran vil særlig to faktorer være kritiske:

- Systemet vil gi sterk CPU-belastning
- Det vil være vanskelig å simulere systemets sanne omgivelser og derigjennom få utprøvet mann-maskin kommunikasjonen og software i en mock-up

I det følgende vil det vises hvordan problemene kan minskes ved forskjellige metoder for simulering.

7.1 Simuleringsmodell for måling av CPU-belastning

For å få en oversikt over problemene forbundet med en høy belastning av regneressursene ble det besluttet å lage en tidsmodell av systemet, d v s en modell som mest mulig realistisk skulle fortelle hvordan systemet ble belastet ved varierende bruk av sensorer og våpen. Ikke minst ville denne modellen kunne si hvordan systemets yteevne ville forandres ved endringer i konstruksjon av både hardware og software.

For enklere å kunne vurdere og sammenligne forskjellige alternativer var det nødvendig med en definisjon av det vanligvis nokså diffuse begrep ytelse. I dette systemet ble (regnemessig) ytelse definert som

$$E = 1 - \sum_{i=1}^{n} K_i \frac{e_i}{m_i} \qquad \text{der } \Sigma K_i = 1$$

I dette uttrykket er

m_i – det totale antall målinger fra sensor nr *i* i et gitt tidsintervall

e_i – det totale antall tapte målinger i samme intervall. Tapte målinger er slike som blir overskrevet av nye eller blir foreldet.

K_i – vektfaktor som angir hvor stor relativ vekt vi legger på målinger fra sensor nr *i*.

Sagt med andre ord gir E et uttrykk for hvor stor del av den totale datamengde fra sensorene som maskinen kan behandle, hvis systemet skal arbeide i sann tid.

Ved å definere et slikt mål for ytelse var det lettere å vurdere ulike valg av konfigurasjon mot hverandre.

Omverdenens (sensorenes) dynamiske natur gjorde at det å finne E ikke var noe problem en kunne løse analytisk. En modell av systemet ble derfor beskrevet i språket SIMULA 67. Denne modellen skulle gi en realistisk beskrivelse av sekvensen i det hovedregneenheten foretok seg og gi en detaljert status for tilstanden i systemet, f eks antall målinger i kø i filene slik som vist på figur 6.6. Dessuten skulle modellen regne ut ytelsen eller effektiviteten E. Modellen innbefattet også en simulering av den jobbmonitor som skulle benyttes i systemet. Samtidig ble det nedlagt et relativt stort arbeid for å kunne estimere realistiske regnetider for de mest tidkrevende deler av systemet.

Erfaringene med denne modell har vært gode og en del av fordelene kan oppsummeres som følger:

a) Modellen gir en dynamisk korrekt gjengivelse av sekvensen i hovedregneenhet.

b) Modellen er enkel å beskrive p g a SIMULA-språkets innebygde egenskaper.

c) Modellen er fleksibel, d v s at kritiske deler av systemet kan beskrives detaljert, mens deler som er av mindre betydning kan forenkles eller sløyfes. Ny viten om systemet kan enkelt bygges inn etter hvert.

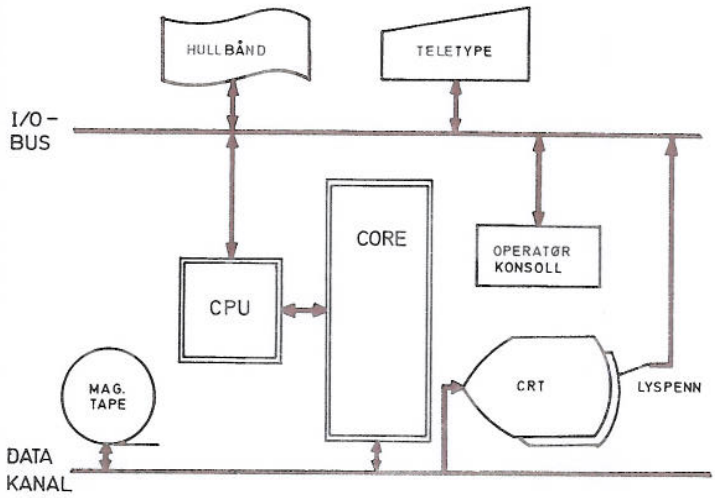
Ved FFI ga modellen i startfasen en god del nyttig informasjon, og selve arbeidet med modellen ga i seg selv ekstra utbytte ved at dette krevde en relativt detaljert formulering av programstrukturen til det endelige system. Modellen vil for øvrig stadig kunne være nyttig under den videre oppfølging av prosjektet.

7.2 Generell oversikt over testmetoder

Det er i de foregående kapitler vist hvordan det operative systemet er tenkt å virke, og det er vist hvordan en ved innlesning av sensordata er avhengig av et intimt samspill mellom hardware og software. Under uttesting av software og det man kan kalle mann-maskin kommunikasjon er dette samspillet ofte et stort problem. For det første vil tilstrekkelig hardware som regel komme på et sent tidspunkt og selv om hardware skulle finnes vil en få problemer med å gjenskape en taktisk situasjon i laboratoriet som vil gi en operatør en mulighet for å utprøve systemet.

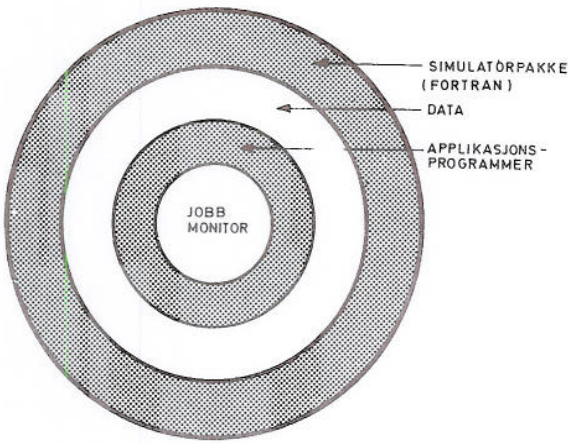
For å gjøre seg mest mulig uavhengig av eksisterende hardware eller gjenskapning av taktisk realistiske omgivelser vil det lønne seg å legge ned et betydelig arbeid for å bygge opp nødvendig hardware som er uavhengig av en endelig hardwarekonfigurasjon. I tillegg kan det konstrueres software som i en testperiode kan kompensere for manglende hardware og taktisk miljø.

På figur 7.1 er det vist hvilken hardware som er nødvendig for å kunne utprøve det systemet som er vist på figur 3.2. Dette er et system strippet for plattform, radar ekstraktor og våpen tilkobling, men hvor en i tillegg til det som er vist på figur 3.2 har en ekstra dataskjerm. Med den ene dataskjermen simuleres radar video, samtidig som den vanlige syntetiske informasjon slik som på figur 5.5 vises. På den andre dataskjermen simuleres TV-bildet (eller om en vil IR-bildet som prinsipielt ikke skiller



Figur 7.1 Testsystem konfigurasjon, hardware

seg fra TV-bildet). Konsollet kan enten være en mest mulig nøyaktig gjengivelse av det endelige konsollet, eller det kan være et generelt konsoll hvor knapper kan tillegges funksjon etter behov. Tilleggsutstyr er teletype, kortleser og hullbåndleser.



Figur 7.2 Oppbygging av programsystemet, prinsippskisse

Figur 7.2 viser software for dette systemet. Programsystemet er lagdelt, hvor de tre innerste ringene, jobbmonitor, applikasjonsprogrammer og dataområde utgjør den "operative" software, og hvor det ytterst er et simuleringsprogram skrevet hovedsaklig i FORTRAN. Dette er bl a i stand til å simulere opp til 16 mål og eget fartøys bestikk. I tillegg vil radar video for alle mål og TV-bilde av et mål simuleres.

Før systemet beskrives i sin helhet i mere detalj, vil neste kapittel beskrive simuleringspakken for 16 mål og eget fartøy.

7.3 Simuleringsprogram for eget fartøy og målfartøyer

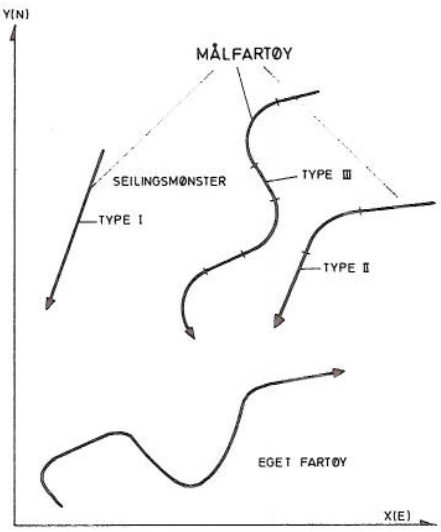
For å kunne simulere eget fartøy og målfartøy er det ved FFI utviklet et simuleringsprogram. Dette er skrevet i FORTRAN og er beskrevet i (2). Bruken av dette programmet er delt i tre faser:

a) I denne fase bestemmes hvilken taktisk situasjon som skal simuleres, d v s hvilke mønstre eget fartøy og opptil 16 målfartøyer skal seile etter. Et eksempel er vist på figur 7.3. Eget fartøy kan ha et relativt komplisert seilingsmønster. For hvert av de 16 mål kan man velge mellom tre seilingsmønstre. Det enkleste (I) på figur 7.3 er et mål som går med konstant kurs og fart (eventuelt med én akselerasjonsfase). (II) viser et mål som har en sving (etter en sirkelbue) og ellers går

rettløpet. (III) viser det mest kompliserte mønsteret. Her svinger målfartøyet vekselvis til venstre og høyre med et rett løp etter hver sving.

b) I denne fasen punches seilingsmønster for eget fartøy og målfartøyene inn på kort. En kan benytte flere kort for å beskrive seilingsmønsteret til eget fartøy, mens det trengs bare ett kort for hvert målfartøy. Kortene kan nå leses av simuleringsprogrammet. Når dette er gjort, er simuleringsprogrammet uavhengig av annet eksternt utstyr. De innleste parametre ligger nå lagret i simuleringsprogrammet.

c) Programmet er nå operativt og benyttes som en rekke subrutiner. Alle subrutiner kalles med "sann tid" som parameter, og man vil som resultat kunne få ut eget fartøys kurs og fart og målfartøyenes kurs, fart og peiling, alt referert til "sann tid". Ikke minst kan man få lagt på støy slik at de forskjellige sensorene kan simuleres.



Figur 7.3 Seilingsmønstre, eget fartøy og målfartøyer

7.4 Simulering av radar video og ARVEX

I forbindelse med den simuleringspakken som er beskrevet i 7.3 vil det være aktuelt å simulere radar video og data til og fra ARVEX. ARVEX vil sannsynligvis enklest simuleres ved at modellen beskrevet i 7.3 modifiseres slik at en ved subrutinekall kommuniserer med denne på samme måte som en skal kommunisere med ARVEX v h a I/O.

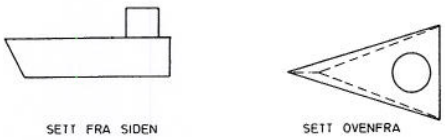
Radar video kan simuleres ved at det opprettes et eget entry i tidsmonitorens jobbtabell. Radarekko kan leses en gang pr scan fra simuleringsprogrammet (peiling referert til neste scan). Programmet kan da tilnærmet simulere radar ekko ved at det kalles inn hvert 0,1 sek og tegner ekko som er kommet inn i løpet av de siste 0,1 sek. Hvis en regner at et scan tar 3 sek, vil en ved hver opptegning få ekko innenfor en sektor på

$$\frac{360^\circ}{3} \cdot 0{,}1 \approx 12^\circ$$

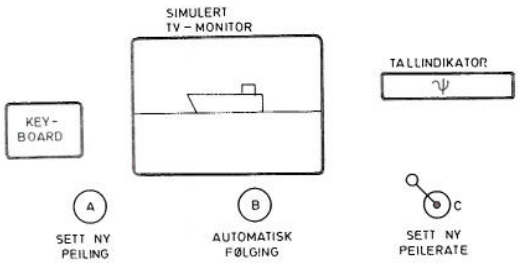
Med dette opplegget burde en få et noenlunde realistisk bilde av det som skjer i virkeligheten. Det ville sannsynligvis være en fordel om støy ble tegnet som punkter mens fartøyer kunne tegnes som vektorer.

7.5 Simulering av TV-bilde

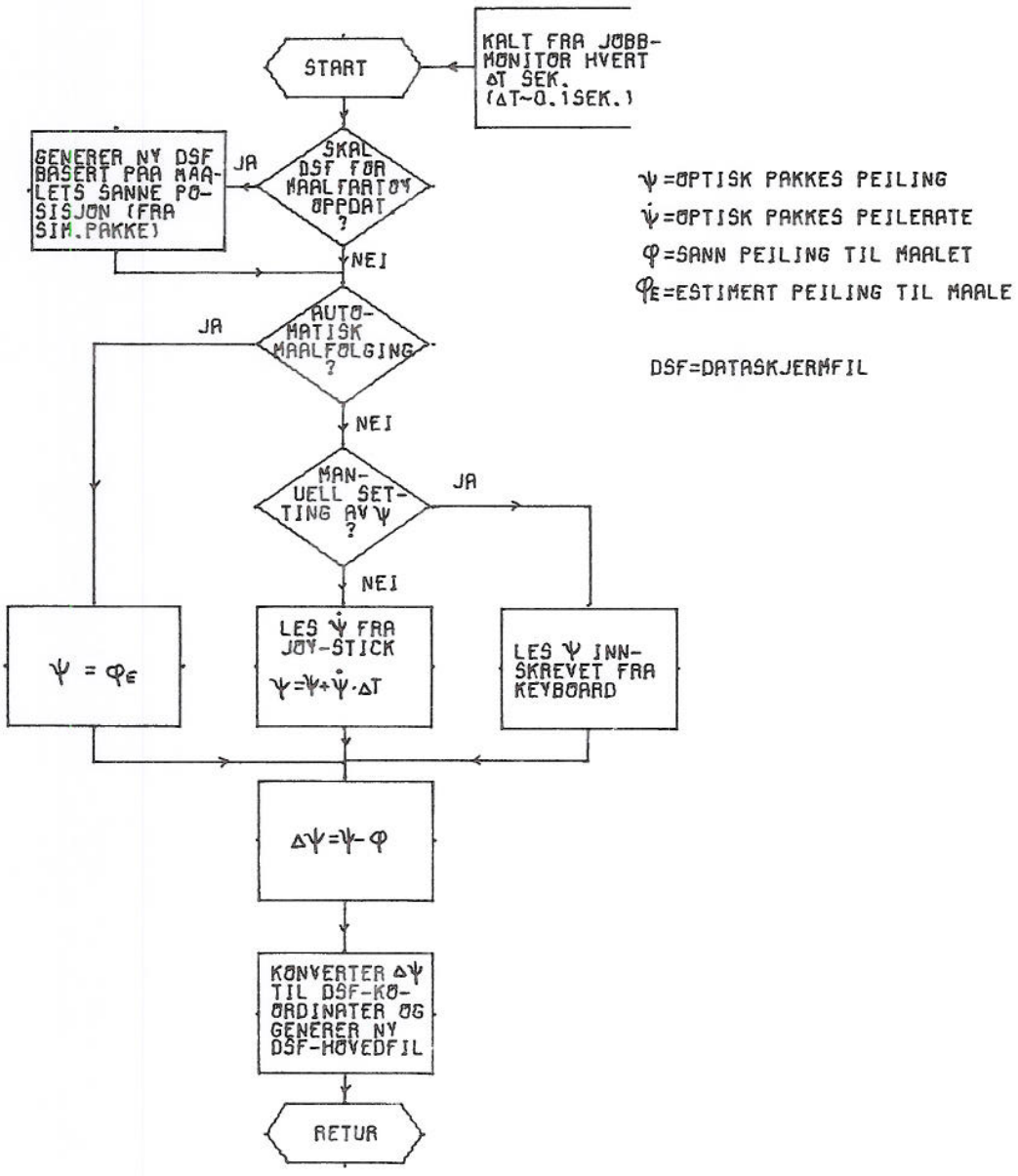
Ved bruk av systemet vil TV/IR inngå som en meget vesentlig faktor. Det vil derfor kunne være av stor verdi å kunne simulere denne sensoren. En metode å gjør dette på er v h a et syntetisk TV/IR-bilde som kan vises på en ekstra dataskjerm. En modell av en båt slik som figur 7.4 viser har vist seg å være forholdsvis enkel å tegne, samtidig som en ved at det benyttes skjulte linjer kan se om båter er på vei *mot* eget fartøy, *fra* eget fartøy etc.



Figur 7.4 Modell av båt for simulering av TV-bilde



Figur 7.5 Monitor og funksjoner



Figur 7.6 Flytdiagram for program for simulering av TV-bilde

Å kunne vise flere mål som samtidig befinner seg innenfor TV-monitorens synsfelt, vil belaste CPU så sterkt at operatøren vil miste følelsen av å arbeide i sann tid. Ved å simulere bare ett av målene (valgt av operatøren) vil derimot CPU-belastningen bli tolerabel, og operatøren vil arbeide i et tilnærmet virkelighetstro miljø.

Dataskjermfilene (DSF) for dette systemet kan bygges opp på samme måten som DSF for hovedsystemet (figur 5.6) ved hjelp av en hovedfile hvor plasseringen i x-retning på skjermen bestemmes. Denne må da oppdateres hvert klokkeinterrupt. Bildet av selve målfartøyet derimot vil forandre seg forholdsvis langsomt og en oppdatering hvert 3 sek burde være tilstrekkelig. Bildet av selve målfartøyet er generert i en egen DSF.

Hvis man antar at optisk pakke inneholder de betjeningsfunksjonene som vist på figur 7.5, vil simuleringsprogrammet ha et flytdiagram som vist på figur 7.6.

En ser at betjeningsfunksjonene består av en trykknapp for setting av ny peiling (ψ) og en joystick (eller tilsvarende hjelpemiddel) for setting av peilerate ($\dot{\psi}$).

I tillegg finnes en knapp for automatisk følging, slik at hvis denne er aktivert, vil optisk pakke følge målestimatet for et mål valgt av operatøren (ved at det settes ut ψ og/eller $\dot{\psi}$). En svakhet ved systemet er at bevegelsen av målfartøyet vil foregå i sprang med 10 Hz oppdateringsfrekvens.

8 KONKLUSJON

Denne rapport har beskrevet et forslag til programsystem ved utvikling av ildledningssystem for neste generasjons hurtige patruljebåter. Kontroll av våpen er ikke tatt med i denne beskrivelsen, men det må antas at denne delen av programsystemet vil være enkel å tilpasse innenfor den spesifiserte hovedramme. Idet jobbmonitoren fra MSI-70U er beholdt, vil også programsystemet naturlig nok ha mange likhetspunkter. Særlig når det gjelder filoppbygningen for dataskjermsystemet og den såkalte konsollmonitor burde det forslaget som er gitt representere en vesentlig forbedring fra MSI-70U.

Det foreslåtte system for utprøving av programsystem og mann-maskin kommunikasjon skulle være enkelt å implementere og gi brukbare resultater. Fremfor alt kreves det liten innsats for å konstruere ekstra hardware. Erfaringsmessig er dette tidkrevende og vil kreve bemanning som ellers kunne vært frigjort for utvikling av en eventuell prototyp.

Litteratur

(1) Bølstad, T — MAGDA display, Programmer's instruction manual, Teknisk notat E–484, Forsvarets forskningsinstitutt (1972)

(2) Løken, O — Simuleringssystem for måldatainnlesning og eget fartøys bestikk, Teknisk notat E–509, Forsvarets forskningsinstitutt (1973)

(3) Haugland, T — Problemer ved valg og implementering av typiske sanntidssystemer for prosesskontroll, Konferansepubl Nord-DATA 1972 (1972)

(4) Solesvik, A — The peripheral processor system, PEP, Teknisk notat E–462, Forsvarets forskningsinstitutt (1972)

(5) Rannestad, A — Dokumentasjon MSI-70U – Generell oversikt, Intern rapport E–190, Forsvarets forskningsinstitutt (1971)

(6) Holberg, K
A Solesvik — Preliminary main specifications for LOKE fire control system, Teknisk notat E–393, Forsvarets forskningsinstitutt (1971)

(7) Sørensen, J — SIMULOK–Et SIMULA-program som simulerer LOKE-systemets forbruk av regnetid, Work-Report Loke No 206, Forsvarets forskningsinstitutt (1971)