NTNU
Det skapende universitet

Instrumentering av autonomt ubemannet fly: CyberSwan

Edgar Bjørntvedt

Master i teknisk kybernetikk
Oppgaven levert: Juni 2007
Hovedveileder: Amund Skavhaug, ITK

Norges teknisk-naturvitenskapelige universitet
Institutt for teknisk kybernetikk

Oppgavetekst

Oppgavens tekst
Institutt for Teknisk Kybernetikk ønsker en generell plattform for et autonomt ubemannet fly med elektrisk fremdrift. Den skal kunne brukes til for eksempel:
- utføre overvåkningsoppdrag med live video overføring
- relay stasjon for høyhastighets datalink i ulendt terreng
- demonstrasjoner og nyrekruttering for Institutt for Teknisk Kybernetikk

Når flyet er ferdigstilt, skal det inneholde nødvendig instrumentering for å kunne å styre og navigere på egenhånd. I tillegg skal flyet kommunisere trådløst med en basestasjon på bakken.

Denne oppgaven består i å lage et utgangspunkt for en egnet generell instrumenteringsplattform for flyet. Instrumenteringsplattformen skal bygges og implementeres. Diplomoppgaven består i følgende momenter:
- finne frem til nødvendig måleinstrumenter og maskinvare
- sette sammen et førsteutkast til plattformen
- vurdere egen løsning

I den grad tiden tillater det, skal systemet testes og eventuelle modifikasjoner utføres.

Dette innebærer design, komponentvalg og sammensetning av disse. Hvis tid skal instrumenteringssystemet testes og eventuelt modifiseres etter testing.

Oppgaven gitt: 19. januar 2007
Hovedveileder: Amund Skavhaug, ITK

Sammendrag

Målet med CyberSwan er å lage en generell plattform for et autonomt ubemannet fly. Gjennom arbeidet som er beskrevet i denne rapporten er det kommet frem til et egnet instrumenteringsplattform for CyberSwan. Med instrumenteringssystemet som er beskrevet, styres flyet i utgangspunktet manuelt, men med en bryter på en manuell modellflyradiosender settes flyet i autonom modus. CyberSwan styrer seg selv så lenge denne bryteren holdes inne.

Instrumenteringssystemet er bygget rundt en PC/104 plattform som består av en PC/104 CPU enhet med lagring på et 2.0 GB Compact Flash-kort. Det er utviklet et IO-kort til PC/104 enheten som gjør det mulig å koble til IMU, GPS, pitotrør, mottager for manuell styring og seks servoer til alle rorene og fartsregulator i CyberSwan. Strømforsyningen til instrumenteringssystemet er også lagt på dette IO-kortet.

Styringssystemet kjøres på PC/104 enheten under Linux. Det er lagt inn programvare som gjør det mulig å overføre filer og starte opp styressystemet over Ethernet.

Det er utviklet en plattform i Simulink hvor CyberSwan sitt styresystem kan legges til. Denne plattformen inneholder blokker som kommuniserer med alle IO-enhetene i CyberSwan. Denne plattformen er godt beskrevet sammen med utviklingsmiljøet som ble brukt på arbeidsstasjonen.

CyberSwan sin viktigste målenhet er IMU. IMUen gir tilstrekkelig informasjon til at CyberSwan sitt styresystem kan stabilisere flyet mot en gitt retning. Denne IMUen er testet med gode resultater sammen med utviklet pådragsorgan til servoene under autonom flyvning.

Det er funnet frem til en GPS-modul som egner seg i CyberSwan. Modulen er liten og enkel å bruke, og den er integrert i plattformen i Simulink. Det er også funnet frem til en trykksensor som egner seg å bruke i et pitotrør for måling av hastighet. Denne målingen er ikke fullstendig integrert i instrumenteringssystemet, men trykkmåleren er testet og det er laget tilkobling til trykksensoren på IO-kortet.

Det er laget en mottaksenhet for seks servosignaler som er integrert i plattformen i Simulink. Når en modellflymottager kobles til disse inngangene, får styresystemet tilgang til servopådragene under manuell styring. Styresystemet vil dermed til en hver tid kjenne servoenes posisjon selv under manuell styring.

En komponent på IO-kortet fungerer ikke som forventet slik at en ikke får utnyttet det fulle potensiale IO-kortet har. Komponenten gjør at det ikke er mulig å måle manuelle servosignaler fra mottager under autonom flyvning. Dette er ikke kritisk for å kunne fly autonomt, men gir begrenset loggemuligheter og kompliserer bruken av CyberSwan. Denne enheten bør derfor byttes ut med en annen enheten i et eventuelt oppfølgingsprosjekt.

Det gjenstår en del arbeid for å fullføre integreringen av alle målingene i CyberSwan sitt instrumenteringssystem, men grunnlaget er lagt og resultatene viser at de delene som er fullført, fungerer som forventet.

Begrepsliste

ADC	Analog to Digital Converter
CPU	Central Processing Unit (norsk dagligtale: Prosessor)
GCC	GNU Compiler Collection
GGA	Global Positioning System Fixed Data
GPS	Global Positioning System
I/O	Input/Output (tilkobling til maskinvare)
I^2C	Inter-Integrated Circuit bus (også kalt TWI)
IMU	Inertial Measurement Unit
IDE	Integrated/Intelligent Drive Electronics, buss mellom PC og harddisk
IR	Infrared
LiPo	Litium Polymer
LSB	Least Significant Bit
NMEA	National Marine Electronics Association
OEM	Original Equipment Manufacturer
OSI model	Open Systems Interconnection Reference Model
PAL	Programmable Array Logic
PCB	Printed circuit board
PC/104	PC/104 er en liten PC med formfaktor på 90 x 96mm (l x b)
PCM	Pulse Code Modulation
PWM	Pulse-width modulation (norsk: Pulsbreddemodulering)
RF module	Radio Frequency Module
RMC	Recommended Minimum Specific GNSS Data
RS-232	Seriell kommunikasjonsprotokoll/COM-port
TTL-Level	Transistor-Transistor Logic Level[1]
TWI	Two-Wire Interface (også kalt I^2C)
UART	Universal Asynchronous Receiver/Transmitter
UAV	Unmaned Aeral Vehicle
VHDL	Very High Speed Intergrated Circuit Description Language
XML	Extensible Markup Language

Innhold

Figurer

Tabeller

Kapittel 1

Innledning

To studenters interesse for modellfly dannet grunnlaget for å benytte prosjekt- og dimplomoppgaven ved Institutt for Teknisk Kybernetikk til å arbeide med å utvikle og bygge en autonom Unmaned Aeral Vehicle (UAV). Instituttet ønsket på sin side å støtte opp under prosjektet for å kunne bruke flyet til å fange nye studenters interesse for studieretningen.

Flyet og prosjektet ble døpt CyberSwan, i tråd med navnet på instituttets eksisterende sveveplatform CyberEagle. Det langsiktige målet med CyberSwan var at det skulle utvikles en generell plattform for en UAV som skal kunne lette og lande autonomt, fly stabilt og følge en spesifisert rute. Siden CyberSwan er et omfattende prosjekt, skulle arbeidet påbegynnes av to til tre studenter og danne et grunnlag for senere arbeid gjennom prosjekt- og diplomoppgaver. Prosjektet ble startet høsten 2006, gjennom to prosjektoppgaver ved instituttet. Her ble det konstruert prototype på et elektrisk fly, og instrumenteringssystemet til flyet ble godt påbegynt. Sistnevnte er beskrevet i *Instrumenteringsdesign for autonomt ubemannet fly* fra desember 2006, og danner grunnlag for denne oppgaven. Prosjektet bygger også på erfaringer fra CyberEagle [4]. Denne våren har det også vært gjennomført prosjekter med Linux på AVR32, og integrering av Simulink med AVR32 som senere kan være et alternativ å inkludere i CyberSwan [12].

Det videre arbeidet med CyberSwan ble splittet i tre deler. Fokus i denne delen av prosjektet har vært å kartlegge hvilke komponenter som er nødvendige i CyberSwan sitt instrumenteringssystem, og å integrere disse i et fungerende styresystem. Denne rapporten begrenser seg derfor til instrumenteringssystemet som sitter inne i flyet. Rapporten beskriver først hvordan måleinstrumenter og maskinvare er integrert i CyberSwan, og hvilke komponenter som er benyttet. Alle komponentene er beskrevet og komponentvalg er begrunnet. Det er beskrevet hvordan komponentene er satt sammen i system og hvordan de kommuniserer med hverandre slik at det skal kunne gjenskapes senere. I tillegg beskriver rapporten plattform som ble valgt for styresystemet, og utviklingsvertøyene som er brukt til å lage denne plattformen. Denne plattformen henter inn målinger og gir kommandoer til pådragsorganene i flyet.

De andre delene i CyberSwanprosjektet har vært arbeidet med CyberSwan sitt styresystem som er beskrevet i [6], mens arbeidet med CyberSwan sin bakkestasjon er diskutert i [5].

Autonom UAV er lite utbredt i Norge, men det er andre UAV-prosjekter rundt om i verden; se for eksempel Nasa sin *Aerosonde* [10] og Monash UAV Research Group sin *Aerobotics* [9]. *Aerosonde* er en UAV med forbrenningsmotor og en del større en CyberSwan med totalvekt på 15 kg. *Aerobotics* er elektrofly på lik linje med CyberSwan men bruker en

ferdig AutoPilot enhet fra MicroPilot for Autonom styring. CyberSwan skal være generell plattform for en elektrisk drevet autonom UAV, og er derfor bygget sammen av uavhengige fullt konfigurerbare enheter.

CyberSwan er et kompisert og stort prosjekt som det krever flere år å fullføre. Det er derfor viktig at det er tydelige og klart definerte grensesnitt mellom delene i prosjektet. Det er også en fordel å benytte ferdig standardiserte løsninger der det er mulig for å begrense omfanget av hver enkelt komponent.

En annen utfordring med CyberSwan ligger i å holde vekten på et minimum slik at flyet kan bli så lite som mulig men samtidig ha lang rekkevidde og kunne holde seg lenge flyvende. Vekt og størrelse er også viktig for at det skal kunne bli praktisk å ta med seg flyet og vise frem flyet i rekrutteringsøyemed. CyberSwan må derfor ha energieffektive maskinvarekomponenter som gir tilstrekkelig regnekraft ved lavt strømforbruk. Måleinstrumentene må være nøyaktige og samtidig overholde kravene om lav vekt og strømforbruk. Det er også ønskelig at flyet skal kunne ta med nyttelast som for eksempel et kamera slik at live video kan overføres trådløst til bakken. Dette er ikke minst vesentlig for PR-formål.

Kapittelstrukturen i rapporten er som følger:

Kapittel 2 begrunner hvorfor de ulike måleinstrumentene er tatt med og hva slags informasjon de tilfører systemet. Andre viktige komponenter beskrives også med begrunnelse for hvorfor de er tatt med.

Kapittel 3 og 4 tar for seg Linux som operativsystem og programvaren i CyberSwan og utviklingsmiljøet på arbeidsstasjonen.

Kapittel 6 beskriver de ulike meldinger som brukes i CyberSwan, både internt i flyet og mellom bakkestasjon og flyet.

Kapittel 7 gir en forklaring på hvordan signalbehandlingen for mottak og generering av servosignaler foregår på IO-kortet.

Kapittel 8 beskriver IO-kortet som er utviklet og komponentene på kortet.

Kapittel 10 og 11 beskriver de to flyvningene som ble gjennomført og hvilke tilpasninger i instrumenteringssystemet som ble gjort i forkant av disse flyvningene.

Kapittel 12 gir en oversikt over styrker og svakheter med de valgte løsningene.

Kapittel 13 og 14 gir en oppsummering og konklusjon av arbeidet som er gjort og forslag til videre arbeid med CyberSwan er presentert.

Kapittel 2

Måleinstrumenter og maskinvarekomponenter

Dette kapittelet tar først for seg måleinstrumentene i CyberSwan, deretter er de viktigste komponentene i instrumenteringssystemet beskrevet. Det er forklart hva de ulike enhetene gjør, og det er begrunnet hvorfor de er tatt med og hvilken funksjon de har i instrumenteringssystemet. Hvordan enhetene er koblet sammen er beskrevet utover i rapporten, men figur 2.1 er lagt med i dette kapittelet for å gi en oversikt over det totale systemet i CyberSwan-prosjektet.

2.1 Inertial Measurement Unit - IMU

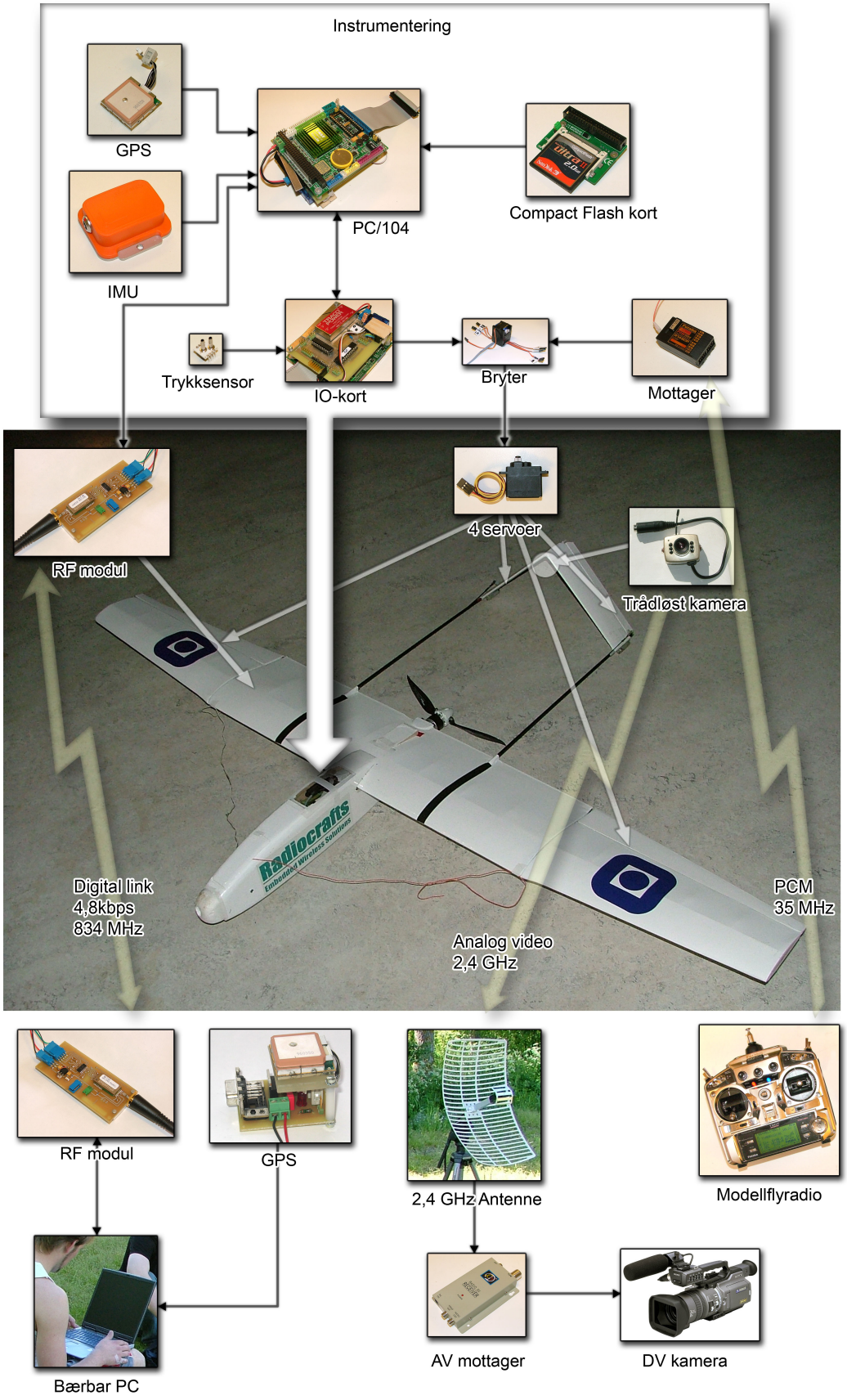
En IMU brukes til å måle flyets oppførsel og orientering i rommet. Den måler rotasjonshastighet rundt tre ortogonale akser med tre gyrometer, i tillegg til akselerasjon i tre ortogonale retninger med tre akselerometer. Pilene i figur 2.2 illustrerer dette. En IMU kan dermed gi ut informasjon om hvilke krefter som virker på flyet og hvordan flyet endrer orientering i rommet.

2.1.1 Magnetometer som referansemåling

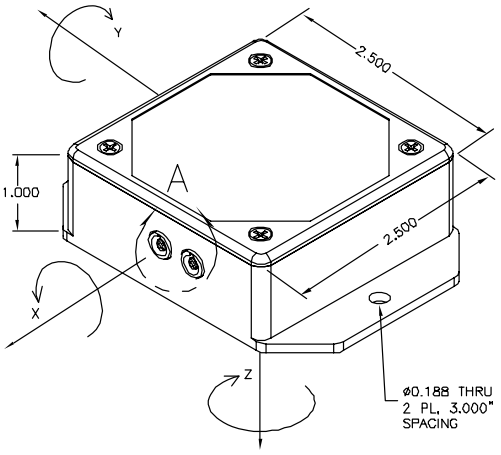
Hvis en IMU skal brukes til å hente ut orientering i rommet (vinkelposisjon), må IMUen kalibreres mot en referanse slik at den ikke drifter over tid. På bakken kan gravitasjonen brukes som referanse, men ved romlige bevegelser trengs det en ekstra måling for å få til en kalibrering som er stabil og nøyaktig over tid. Til dette kan man benytte et magnetometer, som er et instrument som måler jordens magnetiske felt. Et problem med magnetomere er naturlig nok andre magnetiske felt som virker i tillegg til jordmagnetismen. Siden motoren genererer magnetiske felter, må en sørge for at dette ikke forstyrrer målingen. Dette kan gjøres ved å plassere IMUen langt unna motoren eller skjerme støyen fra motoren.

2.1.2 Valg av IMU

IMUen er helt vesentlig for å kunne styre flyet autonomt. Dessverre koster en god IMU i overkant av 20.000 kroner. For å kunne forsvare en så stor utgift til et diplomprosjekt



Figur 2.1: Overordnet oversikt over elementer i prosjektet



Figur 2.2: Målinger fra en IMU

hos Institutt for Teknisk Kybernetikk, var det derfor viktig å finne en fleksibel enhet som kunne flyttes til andre prosjekter. Det ble lagt ned mye tid i å finne en god enhet som både skulle tilfredsstille kravene i dette prosjektet, og som senere vil være nyttig for instituttet i andre prosjekter.

Det finnes mange ulike typer IMUer i forskjellige prisklasser. Det er stor forskjell fra de enkleste til de mer avanserte IMUene. En god IMU må oppfylle følgende krav:

- Akselerometer, gyroskop og magnetometer
- Digitalt grensesnitt, RS-232 foretrekkes
- Enkle protokoller og utgangsverdier
- Selvstendig enhet som enkelt kan byttes ut
- Robust innkapsling

Det var bare to av de vurderte alternativene som var aktuelle. Det var *Xsens MTi* og *MicroStrain 3DM-GX1*. Disse hadde mange av de samme spesifikasjonene og begge IMUene oppfylte kravene over. Følgende egenskaper var felles:

- Akselerometer, gyroskop og magnetometer
- Tilkobling via RS-232/RS-485
- Innebygget DSP med kalmanfiltrering og temperaturkompensasjon
- Enkle utgangsverdier som blant annet Euler vinkler, quaternioner og orienteringsmatrise/rotasjonsmatrise
- Robust innkapsling og støtsikker opp til 500g (påslått)

Både *Xsens MTi* og *MicroStrain 3DM-GX1* har innebygget DSP[1] som filtrerer signalene. Enhetene har også temperaturkompensasjon og magnetometerkompensasjon som gir stabile målinger også over tid.

[1] DSP: Digital Signal Prosessor

2.1.3 xSens MTi eller MicroStrain 3DM-GX1

Figur 2.3: IMU - xSens MTi

xSens MTi har ifølge spesifikasjonen høyere nøyaktighet enn MicroStrain 3DM-GX1, men begge IMUene hadde nøyaktigheter som lå i størrelsesorden 0,1% av maksverdi. Siden dette trolig vil være tilstrekkelig på alle områder, var ikke nøyaktighet avgjørende for valg av enhet. Detaljerte spesifikasjoner og nøyaktigheter finnes forøvrig i datablad som medfølger på CD.

En svakhet som utmerket seg på xSens MTi er at ifølge spesifikasjonen er den ikke operativ med omgivelsestemperaturer under 0°C. Skal flyet brukes på vinterstid i Norge, må dette tas hensyn til.

Det meste av dokumentasjon på Microstrain 3DM-GX1 ligger ute på internett og protokoller er nøye beskrevet. Støtet denne enheten tåler er 500g påslått og 1000g avslått. Dette er ikke veldig mye, og en hard krasjlanding vil den har problemer med å takle. xSens MTi tåler 2000g ifølge dokumentasjonen fra xSens.

Ut ifra nettsiden og dokumentasjon som lå tilgjengelig på internett, virket xSens MTi som en mer kommersiell IMU enn MicroStrain sin variant. Et demokit kunne også bestilles til xSens MTi. Valget falt på xSens MTi siden den virket bedre egnet til prototyping enn MicroStrain 3DM-GX1. Den er også lettere og fysisk mindre en MicroStrain sin variant. En annen fordel er at den kan kobles til 5,1V strømforsyning (min 4,5V) som skal brukes på IO-kortet. Minstekravet til Microstrain 3DM-GX1 er 5,2V.

2.2 GPS

Mens IMUen sørger for et stabilt system og styring av selve flyet, er en GPS-modul nødvendig for at flyet skal kunne navigere etter en spesifisert rute. Navigasjonssystemet med målinger fra GPSen kan sette referanser til styresystemet slik at flyet kan følge en rute, men målingene fra GPSen er verken tilstrekkelige eller gode nok til å kunne stabilisere et fly.

Kvalitet og nøyaktighet i posisjoneringen fra GPS er blant annet avhengig av hvor mange satellitter som er innen rekkevidde. Spesifikasjon på GPS modulen har derfor ikke alt å si

på kvaliteten på posisjoneringen. Her er sentrale parametere til dette prosjektet listet opp for GlobalSat EM-411:

Horisontal presisjon:	<5 meter (50%),	<8 meter (90%)
Høydepresisjon:	<10 meter (50%),	<16 meter (90%)
Reposisjonering:	<2 sek. (90%)	

Oversikten her viser at for eksempel horisontal posisjonering vil med 50% sannsynlighet treffe innenfor 5 meter og med 90% sannsynlighet innenfor 8 meter.

GlobalSat EM-411 ble kjøpt inn til CyberSwan prosjektet høsten 2006, men det ble vurdert to andre alternativer også. Viktig for avgjørelsen var prisen, tilgjengeligheten og at tester av forgjengeren EM-406 viste at serien var enkel å bruke.

2.2.1 GlobalSat EM-411 GPS modul

GlobalSat EM-411 er en ny modul som kom på markedet første gang høsten 2006. Den er billig og har gode spesifikasjoner. GlobalSat EM-411 bygger på forgjengeren GlobalSat EM-406, og begge har SIRF III GPS-brikken med -159dB sensitivitet på GPS mottageren. GlobalSat EM-406 har tidligere vært brukt på instituttet med gode erfaringer, og forskjellen mellom de to er i hovedsak at EM-411 har støtte for flere GPS protokoller.

Fordelen med GlobalSat EM-411 er at den er meget enkel å bruke. Den har innebygget antenne, noe som er en stor fordel siden den er optimalisert for SIRF III-brikken, og den sitter godt montert med mindre muligheter ødeleggelser og feilkobling. Modulen er fra fabrikken satt opp til å sende data over UART (0V - 2.85V) med 4,8kbps. Når strømforsyning på 5V kobles på, begynner den å sende ut GPS meldinger definert i NMEA 0183-protokollen. Disse meldingene er godt beskrevet i dokumentasjonen og er samtidig ganske selvforklarende og greie.

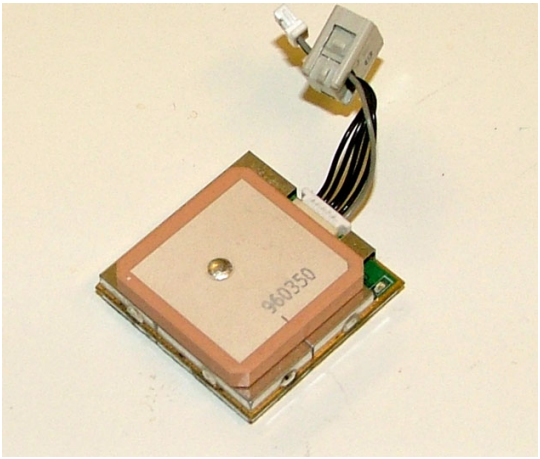
GlobalSat EM-411 gir ifølge den tekniske spesifikasjonen bare horisontal posisjonering (2D). Men tester viste imidlertid at den også gir ut høydeinformasjon. Nøyaktigheten på denne høydemålingen ble ikke testet grundig, men siden det i hovedsak ikke er i GPS modulen begrensninger i nøyaktigheten ligger, vil tilsvarende presisjonen som i tabellen over, også gjelde for GlobalSat EM-411.

Prisen på GlobalSat EM-411 var 450 kroner på ELFA.

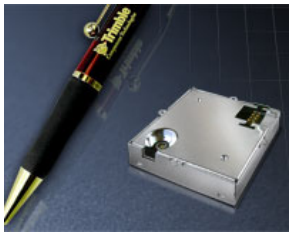
2.2.2 uBlox TIM-LF og Trimble Lassen iQ modul

Et annet alternativ som ble vurdert var uBlox TIM-LF som var tilgjengelig fra Instituttet ved prosjektstart. Siden disse modulene først kom ut i 2003 og hadde pinner som var loddet i stykker, var de lite aktuelle. Ifølge nettsiden til uBlox, skulle det komme ut en ny serie GPS moduler i starten av 2007. Dette kunne vært vurdert hvis GlobalSat EM-411 ikke oppfylte kravene.

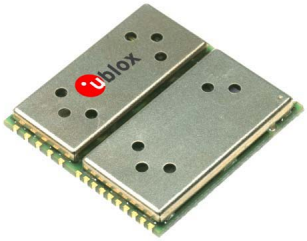
Et tredje alternativ var GPS fra Trimble som har vært brukt tidligere på instituttet. Trimble har også vært brukt tidligere i andre autonome UAV prosjekter som for eksempel NASAs *Aerosonde* [10] og Monash University, Australia [9]. Det ble derfor tidlig vurdert å bruke en GPS modul fra Trimble. Av sine egne GPS moduler anbefalte Trimble selv Trimble Lassen iQ modul. En ulempe var at den nåværende norske Trimbleleverandøren, EG Components, ikke hadde Trimbleprodukter som lagervare. Leveringstiden var derfor



(a) CyberSwan bruker GlobalSat EM-411



(b) Trimble Lassen iQ GPS



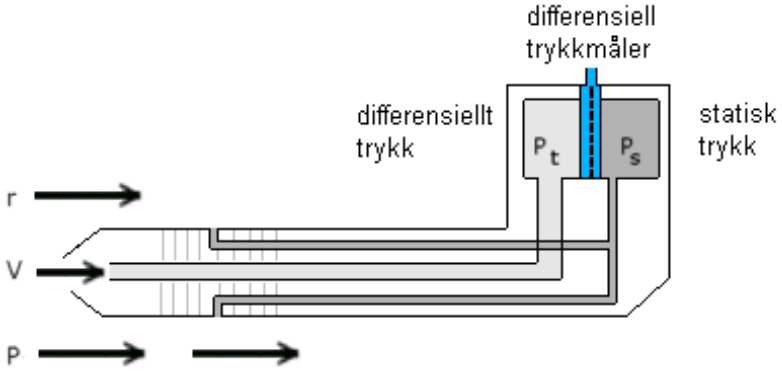
(c) uBlox TIM-LF

Figur 2.4: GPS moduler

3-4 uker. En annen ulempe med Lassen iQ modul var at den trengte ekstern antenne. På forespørsel kunne ikke leverandøren gi noen eksakt pris, men antydet at den ville ligge på godt over 1000 kroner + antenne.

2.3 Pitotrør for måling av hastighet

I de fleste vanlige fly brukes et pitotrør for å måle flyets relative hastighet i forhold til luften. Siden luftstrømmen genererer løftet i vingene er dette en nyttig måling.



Figur 2.5: Hastighetsmåler - Pitotrør (figuren er hentet fra [3])

Et pitotrør måler trykkforskjellen mellom dynamisk og statisk trykk. Det dynamiske trykket P_t [Pa] måles parallelt med fartsretningen, mens det statiske trykket P_s [Pa] måles ortogonalt på fartsretningen. Sammenhengen mellom trykkforskjellen $P_t - P_s$ og hastigheten V [m/s] er gitt av Bernouillis ligning (ligning 2.1):

$$P_s + \rho\frac{V^2}{2} = P_t \tag{2.1}$$

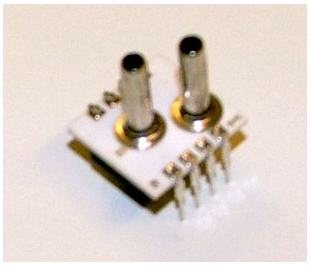
$$P_t - P_s = \rho\frac{V^2}{2} \tag{2.2}$$

Dette kan brukes til å dimensjonere trykksensoren som trengs i pitotrøret. CyberSwan er designet til å ha en marsjhastighet på 15-20 m/s [6], og det er antatt at flyet ikke vil komme opp i hastigheter høyere en 40m/s. Vi setter derfor 40 m/s som makshastighet. Ønsket måleområde for pitotrøret blir da [0, 40] m/s. Lufttettheten ρ er 1,225 kg/m^3, og vi får da:

$$(P_t - P_s)_{maks} = 1,225\frac{kg}{m^3} \cdot \frac{(40\frac{m}{s})^2}{2} \tag{2.3}$$

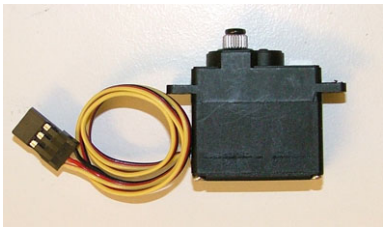
$$= 980Pa \tag{2.4}$$

Et tysk firma som heter AMSYS GmbH & Co. KG. hadde et godt utvalg av trykksensorer til svært gode priser sammenlignet med nordiske leverandører som Elfa og Farnell. De var meget behjelpelige med tips, og de anbefalte en serie som het SM5800 fra Silicon Microstructures. Som vi ser av svaret i 2.4 er det ønskelig med en sensor som kan måle fra 0-980 Pa. Det gir oss en ensidig differensiell trykkmåler av den mest følsomme typen i 5800-serien, som har maks utslag på 1,0 kPa. Denne vil kunne måle hastigheter i området [0, 40.4] m/s, og passer dermed svært bra til vårt bruk.



Figur 2.6: Ensidig differensiell trykkmåler fra Silicon Microstructures (SM5852-001-S-3-L)

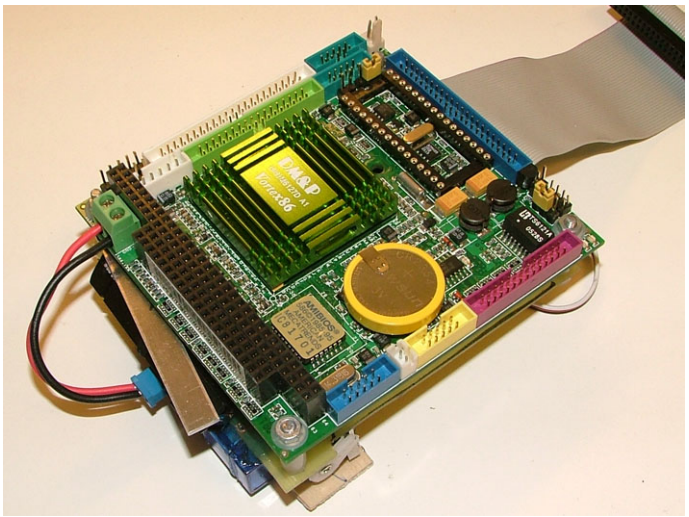
2.4 Servoer



Figur 2.7: Servo brukt til høyde/sideror og balanserorene

Det sitter fire servoer av typen HS-81 i flyet. Disse servoene styrer hvert sitt balanseror, og hvert sitt haleror. Det finnes mange forskjellige servoer som passer til ulike fly, og disse er vurdert med hensyn på moment, vekt, hurtighet og pris. De har også metallgir og er derfor noe mer holdbare enn andre, billigere servoer. Mer informasjon om disse servoene finnes i [6].

2.5 PC/104



Figur 2.8: ICOP-6070 PC/104 Embedded Vortex86 CPU Module

Flyet er designet for å ha plass til et PC/104 CPU kort med mulighet for et ekstra PC/104

kort tilkoblet dette kortet. Høyden på flykroppen er ca 12cm og åpningen i toppen er ca 12cm x 6cm. Dette setter begrensninger på størrelsen på CPU enheten.

Det eksisterte tre PC/104 CPU kort på instituttet ved prosjektstart, hvorav to av dem var aktuelle og ble testet grundig. Kortene hadde enten manglende dokumentasjon, manglende tilkoblinger, eller fungerte ikke som de skulle. Dette gjorde arbeidet med å få disse kortene til å fungere komplisert og tidkrevende. Resultatet var at det ble kjøpt inn et nytt kort til i underkant av 2000 kroner. Kortet som ble kjøpt inn var ICOP-6070 PC/104 Embedded Vortex86 CPU Module og er avbildet i figur 2.8. [2]

2.5.1 ICOP-6070 PC/104 Embedded Vortex86 CPU Module

ICOP-6070 har 2xUSB, 1xRS-232 port, 1xRS-232/RS-485 port, Ethernet, IDE (2 enheter maks), floppy disk, parallell port, LCD, VGA, tastatur, mus og sokkel for DiskOnChip.Den har i utgangspunktet for få serielle porter siden det trengs fire RS-232 tilkoblinger til CPU enheten, men med to USB-til-RS-232-adaptere koblet til de to USB portene kan en likevel få totalt fire serielle porter.

Det er skjerm, tastatur og nettverk på dette kortet, noe som gjør installasjon veldig enkel. Ethernet gjør også bruken av kortet betydelig enklere enn for eksempel Digital Logic i avsnitt 2.5.3.

Det kan kobles to IDE-enheter til kortet, slik at operativsystemet kan installeres på harddisk via CD-ROM slik som på en vanlig PC. Det er en 44-pins header på kortet, og når adapter og CD-ROM som begge har 40-pins IDE kontakt, skal kobles på, trengs det en egnet IDE-kabel. En slik kabel er vist i figur 3.2

De to RS-232 portene på kortet fungerte ikke da de ble bestet. Det ble derfor brukt to USB-til-RS-232 overganger. Dette gav begrenset funksjonalitet, men det gjorde det mulig å koble til både IMU og IO kortet og dermed fly autonomt.

2.5.2 Arcom Pegasus AMD Elan SC520 133MHz prosessor

Dette er et PC/104 CPU kort som har en 133MHz AMD Elan SC520-prosessor og 32MB RAM i tillegg til en 16MB Flash brikke. Kortet har fire serielle tilkoblinger - hvorav to av dem er RS-232, én er RS-422/485 og én er TTL. Den har også Ethernet og IDE med støtte for to IDE-enheter, parallell port, mus og tastatur, men den har ikke VGA- eller USB-tilkobling.

Kortet kan dessverre kun startes fra en spesiell CD som lastes ned fra ftp://download.arcom.com. Detaljert beskrivelse for hvordan Linux legges inn på den bootbare 16MB flash-brikken medfølger på denne CDen. En kan dermed ikke uten videre legge inn den distribusjonen en selv ønsker.

Etter biosoppgradering, installasjon av Linux på kortets interne Flashbrikke, nøye gjennomgang av dokumentasjonen til dette kortet og mange e-poster frem og tilbake med Arcoms support i England viste det seg at det var noe galt med kortet. De ba om å få sendt inn kortet slik at de kan få se nærmere på det. Kortet ble derfor gitt opp, men det bør likevel sendes inn til Arcom slik at det eventuelt kan brukes i senere prosjekter.

[2] størrelse, tilkoblinger, nødvendig krav, anskaffelse, problemer med ulike kort

2.5.3 Digital Logic MSM586SL med Elan520 133MHz prosessor

Dette kortet har en 133MHz prosessor, 64MB RAM, 4 serielle tilkoblinger, floppy, IDE og tastatur og mus. Siden kortet ikke har skjerm, ble det brukt et PC/104 skjermkort i tillegg.

Dokumentasjonen fra Digital logic var begrenset og vanskelig å finne på internett. Etter å ha kontaktet supportavdelingen sendte de tilbake en brukermanual på kortet som er vedlagt på CD (*MSM586SL_ CPU_ card_ Digital_ Logic.pdf*).

Det skulle vise seg at det ikke gikk å koble to IDE-enheter slik at Linux kunne installeres fra CD. Det ble derfor brukt en Bærbar PC til å installere Linux på som beskrevet i avsnitt 4.7.

DeLi Linux ble tidlig installert og fungerte godt på dette kortet. Men DeLi Linux er lite egnet som beskrevet i 3.1.2

Det ble også forsøkt å installere Fluxbuntu på MSM586SL via en bærbar PC som beskrevet i avsnitt 4.7. Fluxbuntu fungerte på den bærbare maskinen, men når Fluxbuntu ble startet på MSM586SL kom det først en *RSDP error* uten at systemet tok skade av det. Litt senere kom det opp en *Kernel Panic* som stoppet oppstarten av Fluxbuntu. Det ble forsøkt å reinstallere Fluxbuntu på et annet Compact Flash kort, men det samme skjedde igjen. Hva dette skyldes er uvisst, og søking på feilmeldingen på internett gav ikke resultater.

Tastaturkabelen ble gjort om og tilpasset Pegasus-kortet slik at det ikke ble testet andre distribusjoner enn DeLi Linux og Fluxbuntu på dette kortet.

2.5.4 Andre aktuelle CPU enheter

Et Mini-ITX hovedkort er større enn det er plass til i flyet, men Nano-ITX - som er et enda mindre hovedkort, vil kunne få plass. Fordelen med denne typen hovedkort er at de har kraftigere prosessorer og mye minne. De koster til gjengjeld betydelig mer enn en PC/104 CPU modul og bruker mer strøm. De har som regel heller ikke RS-232 tilkoblinger og krever strømforsyning med både 5V og 12V tilgjengelig. Dette gjør denne typen kort lite aktuelle.

Atmels AVR32 mikrokontroller er derimot et godt alternativ til PC/104. Det er imidlertid flere grunner til at det ikke ble brukt AVR32 i dette prosjektet. Det var ikke tilgjengelig verken AVR32-brikke eller utviklingskort på grunn av stor etterspørsel hos leverandøren. AVR32 har heller ikke tidligere vært brukt med Simulink og Real-Time Workshop. Det var derfor bedre å velge en sikker løsning som PC/104 i første omgang for så å vurdere andre løsninger senere.

2.6 2.0 GB Compact Flash kort som harddisk

Compact Flash kan brukes i stedet for en harddisk via en Compact Flash-til-IDE adapter. Til prosjektet ble det kjøpt inn to adaptere på ebay.com til ca 50 kroner stykket. Den ene er en adapter til 44 pins IDE mens den andre er en adapter til 40 pins IDE.[3] Sistnevnte,

[3] 44 pins IDE er vanlig tilkobling på 2,5"-harddisker i bærbare PCer, mens 40 pins IDE er vanlig tilkobling til 3,5"-harddisker i stasjonære PCer.

Figur 2.9: Compact Flash kort med adapter til 40 pins IDE

som vi ser i figur 2.9, må ha egen strømforsyningskabel siden det ikke er strømforsyning i en 40-pins IDE-kabel. Denne strømforsyningskontakten sitter på undersiden av kortet.

Under testingen av en rekke forskjellige Linux-distribusjoner (avsnitt 3.3) var det svært mange av distribusjonene som ikke klarte å fullføre installasjonen. Dette skjedde regelmessig etter en god stund, og det viste seg at adapteren med 44-pins IDE-tilkobling ikke var pålitelig. Adapteren og Compact Flash-kortet ble derfor byttet ut med en 2,5"-harddisk. Installasjonene fullførte uten problemer på første forsøk da adapteren ble byttet ut.

På 2,5"-disken var det installasjon av Xubuntu som fullførte uten problemer. Senere fullførte installasjonene av Xubuntu uten problemer også på Compact Flash-kortet, men da med en annen adapter enn den som tidligere var brukt. Dette var adapteren med 40-pins IDE-tilkobling (avbildet på figur 2.9). Dette tyder på at det kan være noe galt med 44-pins-adapteren. Mest sannsynlig er det dårlig kontakt med kabelen som er problemet med denne med adapteren siden den fungerer fint i en bærbar PC hvor adapteren ikke kobles via noen kabel.

2.7 Manuell flyradio og mottager

Flyet må kunne kontrolleres manuelt fra bakken. Til dette brukes radioutstyret som er avbildet på figur 2.10. Dette utstyret bruker 35MHz og støtter PCM[4], som har innebygget feilkorreksjon.

I et vanlig modellfly er alle servoene samt strømforsyning koblet direkte til mottageren i figur 2.10b, men i dette instrumenteringssystemet er det koblet inn en bryter mellom servoene og mottageren slik at det også skal være mulig å kjøre autonomt. Dette er illustrert i oversiktsbildet i figur 2.1. Denne bryteren i flyet styres med en egen knapp på flyradioen.

2.8 Digital kommunikasjon med bakken

I tillegg til radioforbindelse for manuell styring bør det være en toveis kommunikasjon mellom CyberSwan og en basestasjon på bakken. CyberSwan kan dermed sende informasjon om dens egen oppførsel via denne RF-linken til basestasjonen. Det kan også sendes

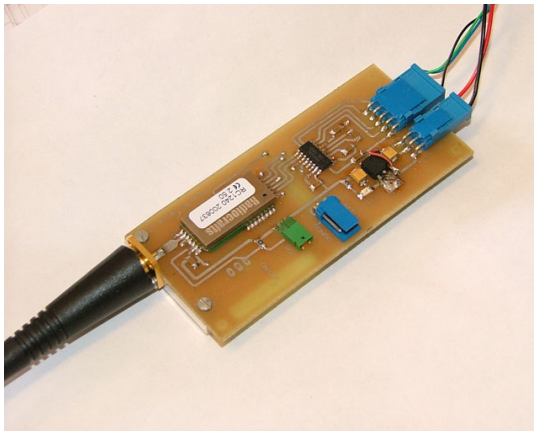
[4]PCM: Pulse Code Modulation

(a) PCM flyradiosender **(b)** PCM mottager i flyet

Figur 2.10: Radioutstyr for manuell styring av flyet

kontrollmeldinger til CyberSwan hvis det er ønskelig å endre parametere som for eksempel trimming av ror, ny kurs eller å hvis en ønsker å endre hva slags type informasjon CyberSwan skal sende til bakken.

Radiokommunikasjon mellom flyet og bakkestasjon er beskrevet i [5].



Figur 2.11: RF-modulen i flyet som er beskrevet i [5]

Kapittel 3

Operativsystem på PC/104

CyberSwan skal være en generell plattform, og instrumenteringssystemet som er beskrevet i denne rapporten danner et godt grunnlag som kan bygges ut med funksjonalitet og kompleksitet der det er ønskelig. Siden et operativsystem setter klare grenser mellom programvare og maskinvare, egner det seg godt i CyberSwan hvor hver enkelt byggesten bør være så selvstendig som mulig. Med et operativsystem kan CyberSwan sitt styresystem og annen programvare videreutvikles mer eller mindre uavhengig av maskinvare. Et operativsystem gjør det også enklere å koble til og kommunisere med eksterne enheter, og ekstern programvare er også enklere å benytte med et operativsystem i bunnen.

3.1 Linux som operativsystem

Linux ble valgt fremfor QNX, Windows og andre operativsystemer fordi utvikleren hadde noe kjennskap til Linux fra før av og ønsket å lære mer om Linux i et tilpasset datasystem. Andre operativsystemer kunne vært brukt, men det ble tidlig valgt å bruke Linux.

Linux er fri programvare som er distribuert under GPL[1]. I tillegg er det mange programmer og annen kildekode for Linux som også er distribuert under GPL og som kan lastes ned fritt fra nettet. Dette gir lave programvarekostnader og tilgang til mye kildekode som kan modifiseres og brukes etter eget behov, forutsatt at en ikke tjener penger uten å kompensere rettighetshaverne.

3.1.1 Sanntidsegenskaper

Linux er ikke et sanntidsoperativsystem og systemet er derfor designet slik at de tidskritiske delene er lagt på IO-kortet, hvor sanntidsegenskapene blir tatt bedre hånd om enn i Linux. Selve styresystemet opererer i utgangspunktet på 10 Hz, men dette kan økes hvis det skulle være behov for det.

Linux har en sanntidsutvidelse som heter RTLinux. RTLinux er ikke testet siden det ikke har vært behov for det, men en eventuell overgang til RTLinux vil i utgangspunktet medføre små eller ingen endringer i eksisterende plattform.

[1] GPL: GNU General Public License

3.1.2 Valg av distribusjon

Det er mulig å velge mellom en rekke Linux-distribusjoner. På Wikipedia finnes det en god oversikt over de mest kjente distribusjonene. Et alternativ kan også være å bygge opp en egen kjerne med kun de pakkene som trengs i systemet. En slik oppbygging av et eget tilpasset Linux system ble påbegynt, men det viste seg å være tidkrevende, og lite fordeler frem for å velge ferdige distribusjoner. Det største problemet var å få rett versjon av de ulike pakkene til å kompilere sammen uten at pakkene var gått ut på dato. Siden pakkene kunne bruke timer på å kompilere tok dette lang tid. Å ta utgangspunkt i en ferdig distribusjon er derfor en betydelig raskere løsning. Det er heller ikke noen dårlig løsning så lenge det er nok lagringskapasitet. Det ble derfor kjøpt inn et 2.0GB Compact Flash kort (avsnitt 2.6) som gir mer en nok plass til å legge inn en ferdig distribusjon.

Desktop Light Linux - eller DeLi Linux - er en distribusjon for eldre og svakere PCer. DeLi Linux sitt minimumskrav er 386-prosessor og 8 MB RAM. En full installasjon av DeLi Linux krever 350 MB med lagringsplass. DeLi Linux er i utgangspunktet beregnet for en arbeidsstasjon, men siden den har så lave systemkrav var det antatt at den passet godt til PC/104. Versjon 0.7.1 av DeLi Linux kom ut oktober 2006 og versjon 0.7.2 kom ut våren 2007. Det ble derfor antatt at DeLi Linux bestod av nye og oppdaterte pakker. Etter en del bruk og problemer ble det likevel oppdaget at DeLi Linux bygget på Linux 2.4-kjernen og ikke Linux 2.6 slik som arbeidsstasjonen (kapittel 4). Da dette ble oppdaget, valgte en å bytte distribusjon heller enn å kompilere programmer for Linux 2.4-kjerne.

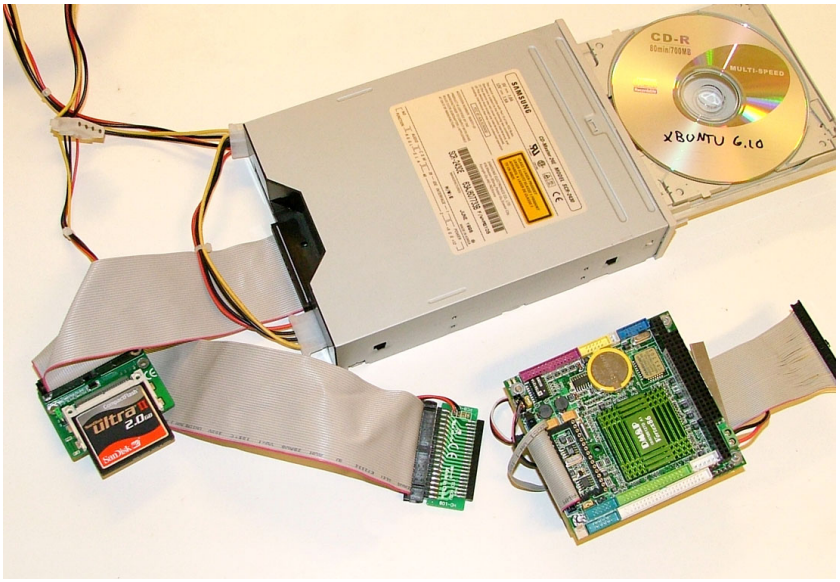
Etter å ha testet DeLi Linux kom det klart frem at det viktigste med valg av distribusjon er å ha et system som er så likt som mulig systemet på arbeidsstasjonen hvor programvaren utvikles. Programvaren kan dermed kompileres og testes på arbeidsstasjonen først. Det ble testet en del distribusjoner (avsnitt 3.3), men det beste resultatet fikk en ved å ta utgangspunkt i en server installasjon av Ubuntu-serien. En serverinstallasjon inneholder ikke et grafisk brukergrensesnitt, men siden det ikke behov for det i CyberSwan, sparer en systemresurser og lagringsplass på å ikke installere en full installasjon.

Ubuntuserien har utviklet seg raskt de siste årene parallelt med økt bruk og popularitet. Dette gjør at det er mye tilgjengelig programvare av god kvalitet gjennom Ubuntu sitt pakkebehandlingssystem. Dette var hovedårsaken til at det ble valgt å bruke Ubuntu-serien på arbeidsstasjonen fremfor RedHat eller andre Linuxversjoner. Dette valget påvirket også valg av Linux distribusjon på PC/104 som nevnt over.

Det er liten forskjell i basesystemet i Ubuntuserien siden det i utgangspunktet er det grafiske brukermiljøet som er forskjellig. Det spiller derfor liten rolle om en velger server installasjon av Ubuntu, Kubuntu eller Xubuntu. I CyberSwan er det installert Xubuntu 6.10-server.

3.2 Installasjon av Xubuntu

Xubuntu lastes ned fra www.xubuntu.org. Serverinstallasjonen finnes på den alternative CDen av Xubuntu. Etter at maskinen er startet opp fra Xubuntu-CDen velger en `Command-Line System`. Resten av installasjonen er ganske rett frem, men det er lagt ved en brukerveiledning på CD som er hentet fra www.osdcm.com: *Installing Xubuntu as a headless server.pdf*



Figur 3.1: Installasjon av Xubuntu på PC/104

3.2.1 openssh-server, minicom og screen

Det er noen nødvendige programmer som ikke følger med på Xubuntu-CDen, men som en laster ned via ubuntu sin pakkebehandler. Spesielt viktig er SSH serveren, som gjør det mulig å logge inn i en konsoll via Ethernet. MiniCom og Screen er også nødvendige.

Etter innlogging installeres `openssh-server`, `minicom` og `screen` via pakkebehandleren med følgende kommando:

```
sudo apt-get install openssh-server minicom screen
```

Innlogging til Xubuntu maskinen via SSH fra en annen Linux maskin gjøres med følgende kommando:

```
ssh <bruker>@<IP-adresse>
```

Minicom er et program for seriell kommunikasjon i konsollen. Serielle kommunikasjonsprogrammer og Minicom er beskrevet i avsnitt 4.8.

Screen gjør det mulig å starte et program vi SSH som ikke stopper i det en logger ut. Det er nødvendig for at styresystemet i CyberSwan skal kunne startes vi SSH før en flyvning. Bruken av screen er beskrevet i avsnitt 10.4.2.

3.3 Andre Linuxversjoner som ble testet

Det ble testet en rekke forskjellig Linuxdistribusjoner. Under er en liste over ulike Linuxdistribusjoner som er testet på ICOP-6070 CPU kortet (bortsett fra DeLi Linux som er testet på MSM586SL).

DeLi Linux var den første installasjonen som ble testet siden den egner seg for eldre og svake PCer med lite RAM og prosessorkraft. Pegasus kortet og MSM586SL er begge forholdsvis gamle og svake kort der DeLi Linux kunne egnet seg, men uten

grafisk brukergrensesnitt er besparelsen trolig minimal. DeLi Linux bygger på Linux 2.4 kjernen.

Fluxbuntu er en strippet ubuntu versjon beregnet for svakere desktop PCer enn det folk flest har. Det er fjernet mange pakker fra den opprinnelige ubuntu distribusjonen og lagt til Fluxbox Window Manager som er betydelig mindre og enklere enn GNOME. Erfaringen fra Fluxbuntu er at det fungerer greit, men at det er dårlig gjennomført og man sparer lite fremfor å bruke standard ubuntu. Uansett trengs ikke det grafiske brukergrensesnittet å legges inn. Det holder med serverinstallasjonen og da er forskjellen mindre.

Clark Connect er en kraftig, kommersiell og brukervennlig brannmur/ruter-distribusjon med utspring i Red Hat Enterprise Linux. På Clark Connect sine nettsider ligger det ute en gratisversjon av Clark Connect. CD-image av Clark Connect er på drøye 400 MB. Clark Connect bygger på Linux 2.6-kjernen. Den er enkel å installere, men den bruker veldig lang tid. Under installasjon av Clark Connect stoppet installasjon etter en stund. En mulig årsak er beskrevet i avsnitt 2.6.

Debian er beregnet for stabile servere. Debian har særdeles gode konfigureringsmuligheter og egner seg for maskiner med spesielle krav utenom det normale. Installasjonen er mer avansert en det som er nødvendig for operativsystemet i CyberSwan, og ble aldri fullført.

Coyote Linux er en liten brannmurdistribusjon der CD-imaget er på under 20 MB. Den bygger på Linux 2.6-kjernen og krever lite systemressurser. Coyote Linux er lett og rask å installere, men den er dårlig dokumentert. Under installasjonen ble det aldri spurt om å velge brukernavn og passord. Siden innloggingsinformasjonen ikke ble funnet på internett, var installasjonen ubrukelig.

Kapittel 4

Utviklingsmiljø på arbeidsstasjon

Dette kapittelet beskriver utviklingsmiljøet for styresystemet til CyberSwan under Linux. Å utvikle programvaren under Linux gjør at testing kan utføres på egen maskin før det testes på PC/104-enheten. Alternativet hadde vært å krysskompilere til Linux fra en Windowsmaskin og teste på PC/104-enheten. Siden en arbeidsstasjon med Kubuntu fungerer godt til daglig bruk, er det enklere å benytte Linux også under utvikling.

Kapittelet kan sees på som en brukerveiledning til verktøyene Matlab og Simulink, og til S-funksjoner i Simulink, mens selve programvaren i Simulink som danner grunnlaget for styresystemet til CyberSwan er beskrevet i kapittel 5.

I tillegg er lagt med følgende brukerveiledninger:

- Kildekoden som følger med xSens MTi
- Installasjonsveiledning av Linux på et PC/104-kort uten Ethernet eller USB forbindelse.
- Programmer for seriell kommunikasjon både på Linux og Windows.
- Konfiguering av Globalsat EM-411

4.0.1 Ubuntuserien - Kubuntu eller Ubuntu

Både Ubuntu og Kubuntu er enkle og bruke. I hovedsak er forskjellen mellom de to distribusjonene at Kubuntu bruker desktop manageren KDE, mens Ubuntu bruker GNOME. KDE er grafisk elegant, mens GNOME er mer sparsom på grafiske effekter som krever resurser. I GNOME står ytelse og enkelhet foran et flott utseende. Undertegnede opplevde KDE som mye mer behagelig og bruke og effektiviteten økte merkbart etter overgang fra Ubuntu til Kubuntu. Kubuntu anbefales derfor fremfor Ubuntu. Xubuntu kan også nevnes i denne sammenhengen. Xubuntu er en tilsvarende distribusjon, men med Xfce desktop manager. Dette er en enklere og mer effektiv desktop manager enn GNOME og KDE.

4.1 Matlab og Simulink med Real-Time Workshop

Simulink er et kraftig grafisk modelleringsverktøy som bygger på Matlab. I Simulink kan styresystemet til CyberSwan utvikles og simuleres før det testes i praksis. Real-Time Work-

shop gjør det mulig å generere kjørbare programmer av styresystemet som er utviklet i Simulink.

4.1.1 Installasjon

Installasjons-CD med Matlab og Simulink for Linux ble lånt fra instituttet. På denne CDen følger det med en god installasjonsveiledning. Instituttet har også lisens på Real-Time Workshop. Det er derfor viktig å bruke instituttet sin lisensfil og ikke lisens fra for eksempel orakel, da denne ikke innholder lisens på Real-Time Workshop.

4.1.2 Linux Soft Real-Time Target v2.2 (LNX2.2)

Linux Soft Real-Time Target (LNX) er en target definisjon til Real-Time Workshop som gjør at styresystemet som genereres kjører i soft realtime under Linux. I Simulink må det defineres en frekvens som styresystemet skal kjøre på. Denne frekvensen sammen med POSIX real-time clocks bruker Linux Soft Real-Time Target til å kjøre hvert tidsskritt i programmet med en fast periodetid [2]. Dette gjør at timingen som settes i Simulink ikke bare gjelder for simuleringer, men også i det ferdige programmet. Uten Linux Soft Real-Time Target ville kjøretiden på programmet vært bestemt av hvor mye regnekraft som var tilgjengelig, og ikke veggklokken på kjøkkenet. Uten LNX måtte systemet måtte da hatt en annen form for timing.

Linux Soft Real-Time Target lastes ned som en zip-fil fra *www.mathworks.com*. Som det står beskrevet i readme-fila som følger med, skal den pakkes ut i `[Matlabroot]/rtw/c/`. Denne stien spesifiseres i Matlab ved å velge *File -> Set Path*.

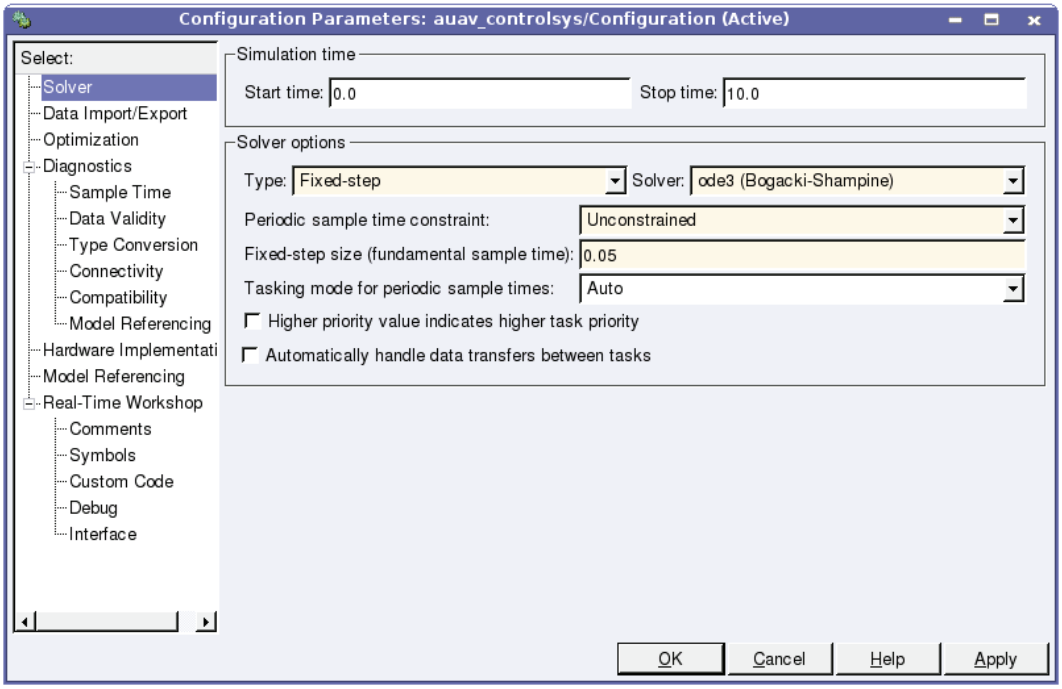
Når stien er satt i Matlab, blir det mulig å velge LNX som *System Target File* i Simulink. Dette gjøres under *Simulation -> Configuration Parameters*. Under *Real-Time Workshop* skal *System Target File* endres til `lnx.tlc` som beskrevet i 4.2.

4.2 Oppsett av Simulinkmodellen med Real-Time Workshop

Før en begynner å legge blokker på modellen i Simulink er det lurt å stille inn de parameterne som skal brukes. Velg *Simulation -> Configuration Parameters*, så kommer vinduet i figur 4.1 opp. Vi skal se nærmere på følgende faner:

Solver: *Simulation time* sier seg selv, og det er oppgitt i sekunder. *Solver options* må endres til *Fixed-step* i dette styresystemet, og *fixed-step size* settes til 0,1 som er det samme som 10Hz. 10Hz gir i utgangspunktet hakkete oppførsel på servoene, men det er implementert et glattefilter i driverkretsen på IO-kortet slik at det ikke er noe problem å kjøre på 10Hz. Uten dette glattefilteret må *fixed-step size* settes til 20Hz for å unngå hakking.

Real-Time Workshop: Her er det flere viktige parametere som skal settes. Vi begynner på toppen med *System Target file*. Hvis det er lagt inn støtte for Linux Soft Real-Time Target (se avsnitt 4.1.2), kan *System Target file* endres til `lnx.tlc`. *Language* må endres til C++ fordi kildekoden som skal integreres fra IMUen er skrevet i C++. S-funksjonen som kommuniserer med IMUen er derfor også skrevet i C++.



Figur 4.1: Konfigureringsvindu for modellen i Simulink

Custom Code: Her inkluderes eksterne kildefiler som for eksempel kildekoden for kommunikasjon med IMUen (`MTComm.cpp`). Linking til egne oppstarts- og termineringsfunksjoner kan også legges inn her. Koden som er lagt inn her, er som følger:

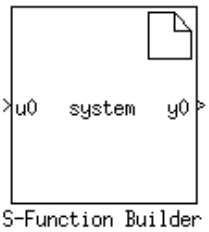
```
*Source file

*Header file
#include <sys/ioctl.h>
#define RF_STORE_LOCATION    "../RF_storage.txt"
#define RF_RECEIVE_LOCATION  "../read_RF.txt"
#define IMU_LOCATION         "/dev/ttyUSB1"
#define GPS_LOCATION         "../read_gps.txt"
#define RECEIVER_LOCATION    "../read_receiver.txt"
#define RECEIVER_SLEEP       sleep(1)
#define RECEIVER_LOCATION    "../read_receiver"
#define SERVO_LOCATION       "../Servo_storage.txt"
#define SERVO_FORMATTING     "%d\t%d\t%d\t%d\t%d\t%d\t%d %d\n"

extern void imu_start(void);
extern void imu_stop(void);
extern void gps_start(void);
extern void gps_stop(void);
extern void receiver_start(void);
extern void receiver_stop(void);
extern void RF_start(void);
extern void RF_stop(void);
```

```
*Initialize function
printf("************\n");
printf("***start****\n");
printf("************\n");
imu_start();
gps_start();
receiver_start();
RF_start();

*Terminate function
printf("************\n");
printf("***stop*****\n");
printf("************\n");
imu_stop();
gps_stop();
receiver_stop();
RF_stop();

*Include directories

*Source files
MTComm.cpp

*Libraries
```

Dette er nærmere beskrevet i kapittel 5.

Linux Target code generation options: *Linker options* settes til `-lm -lpthread -lrt -Wno-deprecated`. Fra før av vil det kun stå `-lm`, som legger til støtte for matematiske formler. `-lpthread` legger på sin side til støtte for POSIX-tråder som brukes i forbindelse med sampling av inndata fra IMUen. `-lrt` gjør at gcc linker med biblioteket `librt`. Dette biblioteket er Linux Real-Time utvidelsen av glibc. [11, Del II, Kap 6, Realtime Support in Linux]. Mindre viktig er `-Wno-deprecated`, som fjerner advarsler om utrangerte headerfiler det ikke er anbefalt å bruke.

4.3 Oppsett av S-funksjoner med S-Function Builder



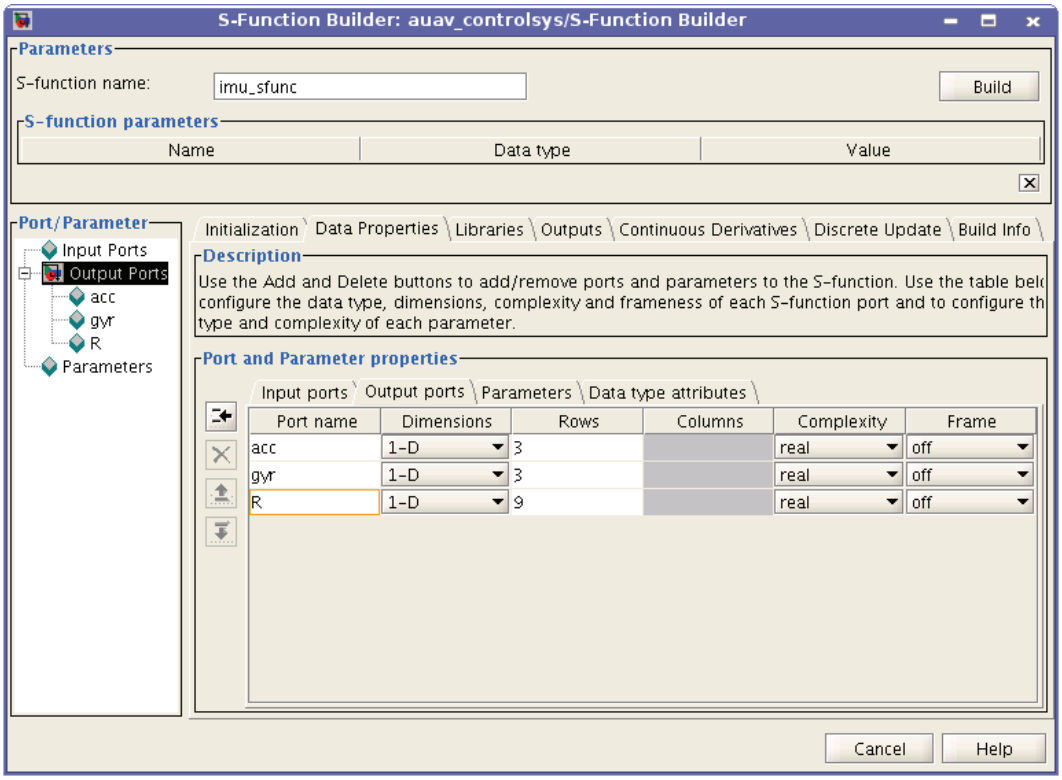
Figur 4.2: *S-Function Builder*

S-funksjoner gjør det mulig å integrere programmeringskode i Simulink. Dette gjør Simulink veldig fleksibelt siden en kan lage egne blokker hvis en mangler funksjonalitet i de ferdig

definerte blokkene i Simulink. S-funksjoner fungerer også meget godt sammen med Real-Time Workshop. En S-funksjon knyttes til kildekode som enten skrives i C/C++ eller som en M-fil.

Det er flere måter å lage et oppsett for S-funksjoner på, men beskrivelsen her tar utgangspunkt i Simulinkblokken *S-function Builder* som er vist i figur 4.2. Dette er en blokk som kan generere C/C++ filer som kan bruke i en S-funksjon senere. Før en bruker *S-Function Builder* må en ta stilling til om det skal lages S-funksjoner i C eller C++. Dette velges under *Simulation -> Configuration Parametrers* under oppsett for *Real-Time Workshop* som beskrevet i 4.2. I CyberSwan brukes C++ i alle S-funksjoner.

Blokken *S-Function Builder* legges inn i modellen. Ved å dobbeltklikke på blokken kommer det opp et vindu hvor S-funksjonen skal konfigureres som vist i figur 4.3. Her settes alle inn- og ut-parametere, filnavn og andre innstillinger. Det er mye som ikke trengs å endres, men vi skal gå gjennom de viktigste innstillingene.



Figur 4.3: Konfigureringsvindu til *S-function Builder*

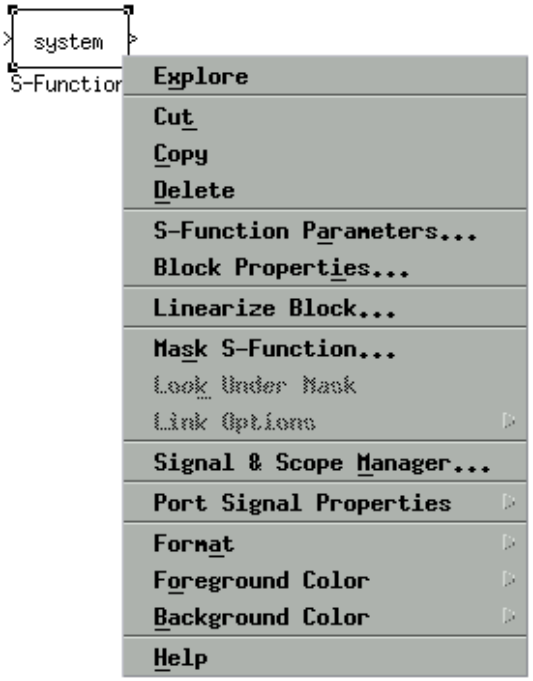
Hvis det er behov for initialisering av tilstander, gjøres dette under fanen *Initialization*. Det er viktig å passe på at *Sample Mode* står på *Inhereted* dersom en ikke har spesielle behov. *Inhereted* gjør at S-funksjonen arver innstillingene som er satt i Real-Time Workshop. Styresystemet genererte *Alarm clock*-feil etterfulgt av terminering av programmet da det var valgt *continuous* her. Det er riktignok ikke sikkert at det var dette som genererte *Alarm clock*-feilene siden det ble gjort mange endringer i neste forsøk. *Alarm clock* var lite dokument og det var få som hadde vært borte i problemet.

Under fanen *Data Properties* settes alle inn- og utganger til S-funksjonsblokken. *Parameters* brukes ikke her siden S-funksjonen ikke skal ta inn noen parametere. Hvis inn- og utgangene skal ha andre datatyper enn standardtypen *double*, kan dette endres under

Data type attributes.Funksjonaliteten og kodelinjene som kan legges inn i fanene: *Libraries*, *Outputs*, *Continuous Derivatives* og *Discrete Update* kan også legges direkte inn i C/C++ kildefilene senere. Siden S-function builder brukes her kun til å lage et oppsett av en s-funksjon er det bedre å legge inn disse kodelinjene senere.

Via knappen *Additional methods* under fanen *Build Info* huker en av for *Start* og *Terminate*. I kildekoden til S-funksjonen vil en da finne to metoder som heter `start()` og `terminate()`. Dette kan brukes hvis det er behov for å initialisere S-funksjonen før programmet starter eller hvis noe skal avsluttes samtidig som nedstengningen av programmet.

Til slutt skrives navnet på S-funksjonen inn i toppen av konfigureringsvinduet til *S-Function Builder* (fig 4.3) og trykker på *Build*. I eksempelet her brukes navnet `imu_sfunc` på S-funksjonen. Da lages det fire filer i prosjektkatalogen som heter `imu_sfunc.tlc`, `imu_sfunc_wrapper.cpp`, `imu_sfunc.cpp` og `imu_sfunc.mexglx`.



Figur 4.4: *S-function* blokk

Nå trengs ikke *S-Function Builder* lenger. En kan derfor slette blokken fra modellen og erstatte den med blokken *S-Function* som vist i figur 4.4. Det er denne blokken som blir S-funksjonsblokken med alle inn- og ut-parameterne en bestemmer seg for å ha i C++ koden.

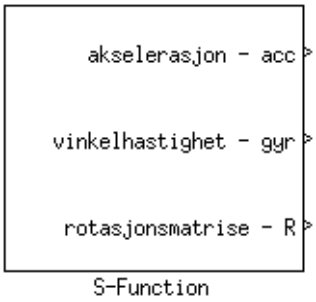
Høyreklikk på S-funksjonsblokken (figur 4.4) og velg *S-Function Parameters*. S-funksjonsblokken må kobles til filene som genereres med *S-Function Builder*. *S-function name* settes til `imu_sfunc` og *S-function modules* til `imu_sfunc_wrapper`. Feltet *S-function parameters* kan man la stå åpent.

Høyreklikk igjen på S-funksjonsblokken og velg *Mask S-Function*. Her settes inngang og utganger og navn som skal synes i blokken. I dette eksempelet brukes følgende kode:

```
port_label('output', 1, 'akselerasjon - acc')
port_label('output', 2, 'vinkelhastighet - gyr')
```

```
port_label('output', 3, 'rotasjonsmatrise - R')
```

Første parameter er enten `input` eller `output`. Andre parameter er rekkefølgen verdiene i *S-Function Builder* ble lagt inn. Hvis en bytter rekkefølgen på to utganger, vil de samme utgangene bytte plass i figur 4.5. Den siste parameteren er navnet på utgangen som skal synes i blokken som er vist i figur 4.5.



Figur 4.5: Eksempel på en S-funksjonsblokk med etiketter

Hvis alt er gjort rett, skal blokken endre seg slik at utgangene ser ut omtrent som i figur 4.5.

4.4 Eclipse med CDT

Eclipse er et gratis utviklingsmiljø som i utgangspunktet er beregnet for utvikling av Java-applikasjoner. Gjennom et prosjekt som heter CTD[1] er det lagt til støtte for C/C++ i Eclipse. Det finnes også en rekke andre utvidelser til Eclipse. Se www.eclipse.org.

Fordelen med Eclipse er at koden struktureres på en oversiktlig måte med blant annet en trestruktur på C++ klassene. Funksjoner, konstanter og globale variabler er enkelt å finne i denne trestrukturen. En kan også velge at Eclipse skal gi automatisk hint og valgfri autoutfylling underveis mens en skriver. Dette gjelder også på egenproduserte funksjoner. Oversiktlig og strukturert kode er viktig for å beholde oversikten i et større prosjekt. Eclipse gjør dette på en god måte.

4.4.1 Installasjon

Eclipse er tilgjengelig gjennom pakkeinstallasjonssystemet for Debian og Ubuntu. For å få Eclipse med støtte for CDT, må en installere to pakker; *eclipse* og *eclipse-cdt*. En kan bruke et grafisk pakkebehandlingsprogram som i Kubuntu heter *Adept Package Manager*, eller en kan bruke følgende kommando i konsollen:

```
sudo apt-get install eclipse eclipse-cdt
```

Etter at installeringen er ferdig, finnes Eclipse i programmenyen.

Da Eclipse skulle installeres, var ikke CDT-utvidelsen tilgjengelig gjennom pakkebehandlingssystemet. Pakkene måtte da lastes ned og kompileres fra kildekoden. Dette gikk uten

[1] Eclipse C/C++ Development Tooling - CDT

problemer med instruksjonene som fulgte med. Eclipse måtte deretter legges inn manuelt i programmenyen i Kubuntu.

Når Eclipse åpnes første gang, blir en bedt om å sette *workspace*. *Workspace* kan betraktes som *root*-katalogen i Eclipse. *Workspace* bør derfor settes slik at en slipper å endre denne senere. Under *workspace* kan opprettes nye prosjekter som lager hver sin katalog i *workspace*-katalogen.

Etter at *workspace* er satt, oppretter en et nytt prosjekt med *File -> New -> Project*. Under *C++* velges *Standard Make C++ Project*. I neste vindu taster en inn `auav_controlsys` som *Project name* og trykker *finish*. I workspacekatalogen er det nå opprettet en ny katalog som heter `auav_controlsys`. Alle kildefilene som skal med i prosjektet kopieres til katalogen `auav_controlsys/` og ved å trykke på F5 eller *refresh* i Eclipse sitt navigasjonsvindu, kommer alle de kopierte filene opp i Eclipse strukturert med en god trestruktur.

Vi valgte å ikke sette opp Eclipse til å styre byggingen av prosjektet siden det er bedre at Real-Time Workshop gjør dette.

4.5 Kommunikasjon med IMU

xSens MTi er godt dokumentert av xSens, og det ligger mye programvare og kildekode på CDen som følger med IMUen. Dette gjør integreringen av IMUen ganske grei selv om både kildekode og dokumentasjon er omfattende. I de følgende avsnittene er det derfor bare beskrevet de delene som er brukt i prosjektet og hvordan kildekoden er integrert med resten av systemet. Dokumentasjon ut over det som er relevant for dette prosjektet finnes på CDen merket 'MT Software Development Kit' som følger med xSens MTi. Denne CDen er det også lagt en kopi av på vedlagt CD i form av en ISO-fil.

Utgangspunktet for integreringen av IMUen er hentet fra kodeeksemplene som følger med xSens MTi. For å få tilgang til disse eksemplene, må programvaren som følger med legges inn på Windows. En vil altså ikke finne noe ved å utforske CDen fra xSens.

Det er særlig to filer som er sentrale og det er MTComm.cpp og MTComm.h som definerer C++ klassen MTComm. Ved å bruke de funksjonene som er definert i denne MTComm-klassen, får en full kontroll over det IMUen har å tilby av funksjonalitet. Denne klassen er omtalt som lavnivå kommunikasjon i xSens MTi sin dokumentasjon. Det følger også med høynivåkode gjennom en protokoll utviklet av xSens som heter MotionTracker Communication protocol. Siden sistnevnte er skrevet for Windows plattform og beregnet hovedsaklig på systemer med flere IMUer, er det kun MTComm klassen som er relevant for dette prosjektet.

4.5.1 Installasjon av xSens sin programvare (lisens, kodeeksempler)

Programvaren på CDen merket 'MT Software Development Kit' må legges inn før IMUen kan brukes, siden dette er eneste måte å få tilgang til dokumentasjon, testprogrammer og kodeeksempler fra xSens. Denne programvaren må installeres på Windows, men kildekode er utviklet både for Windows, Linux og andre operativsystemer. Etter installasjon kan enkelte ferdig kompilerte eksempler kopieres til en Linuxmaskin og kjøres der.

Det var problemer med lisens da programvaren skulle installeres. Enten manglet det lisenskode for IMUen som ble kjøpt, eller så er den forsvunnet. Siden instituttet tidligere hadde kjøpt tilsvarende IMU fra xSens kunne denne lisenskoden brukes. Det er nå Stefano Bertilli som har tilgang til lisenskoden.

4.5.2 Hvordan bruke ferdig kode fra xSens

C++ klassen MTComm, tar seg av lavnivå kommunikasjon med IMUen. For å lære seg hvordan denne klassen brukes, kan en ta utgangspunkt i et av de komplette eksemplene som følger med. Det er et ferdig kompilert Linuxeksempel som heter MTCommExample. Kildekoden til dette eksempelet er et greit utgangspunkt. Siden det er gode kommentarer i kildekoden, er det utelatt nærmere beskrivelse her. Hvordan denne koden er integrert i styresystemet er derimot beskrevet i avsnitt 5.2.

4.6 Filoverføring til en Linux installasjon på Compact Flash kort

Siden harddisken på Linuxinstallasjonen er et Compact Flash kort, kan kortet taes ut og settes inn i en minnekortleser. Hvis det ikke er minnekortleser på arbeidsstasjonen kan eksterne minnekortlesere kjøpes billig på Elfa. I skrivende stund koster en ekstern minnekortleser med USB-tilkobling et par hundre kroner.

Når Compact Flash kortet settes inn i minnekortleseren vil Kubuntu automatisk montere partisjonene som ligger på kortet. Denne automatiske monteringen kan fort slå feil siden Kubuntu for enkelte filsystem ikke klarer å lese riktig type. For eksempel vil Kubuntu montere ext3[2] feil, mens den fint klarer å montere FAT16[3].

Avmonter den automatisk monterte disken som i dette tilfelle heter `usbdisk`, og monter på nytt slik at filsystemets type kan spesifiseres. I dette tilfelle er partisjonen på Compact Flash kortet å finne på `\dev\sda1` med filsystem `ext3`.

```
knutedga@knutedga-kubuntu:~$ sudo umount /media/usbdisk
knutedga@knutedga-kubuntu:~$ sudo mount /dev/sda1 -t ext3 /media/usbdisk
```

Nå kan partisjonen både leses og skrives til. Overfør de aktuelle filene og avmontere partisjonen før kortet fjernes fra minnekortleseren.

```
knutedga@knutedga-kubuntu:~$ sudo umount /media/usbdisk
```

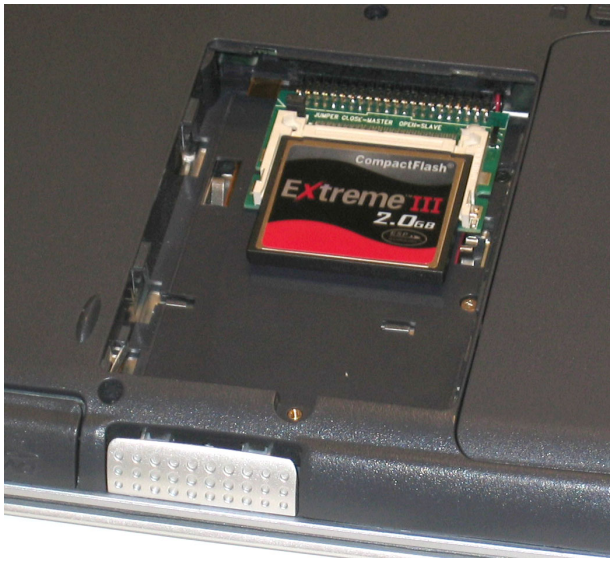
Det går også an å overføre filer fra Windows til minnekortet så sant Windows forstår filsystemet som ligger på kortet. I dette tilfelle hvor Compact Flash kortet har Extended 3 filsystem, må det legges inn drivere for extended 3 slik at en kan bruke utforskeren i Windows til å lese og skrive til Compact Flash kortet. En skal være litt forsiktig når en bruker slike drivere for selv om det kan se ut som det fungerer, kan filene skrives feil til disken og lage inkonsistens på minnekortet.

Et annet alternativ hvis filer skal overføres fra Windows, er å bruke egne programmer som forstår filsystemet. Et eksempel på det er PowerQuest Partition Magic som egentlig er et

[2] ext3: Extended File System version 3
[3] FAT16: File Allocation Table, 16 bit adressering

partisjoneringsverktøy, men som også kan brukes til å kopiere filer mellom ulike partisjoner.

4.7 Installasjon av Linux uten USB, Ethernet, eller to IDE enheter på PC/104



Figur 4.6: Compact Flash kort med adapter til 44 pins IDE i en bærbar PC

Hvis Linux skal installeres uten på en PC/104 uten USB, uten Ethernet, eller uten mulighet for å koble både CD-ROM og harddisk til PC-en samtidig, kan harddisken settes inn i en annen PC hvor installasjonen gjøres. Første restart etter installasjon bør gjøres på PC/104 enheten siden enkelte maskinvaredrivere fullføres under første restart.

4.8 Seriell kommunikasjon

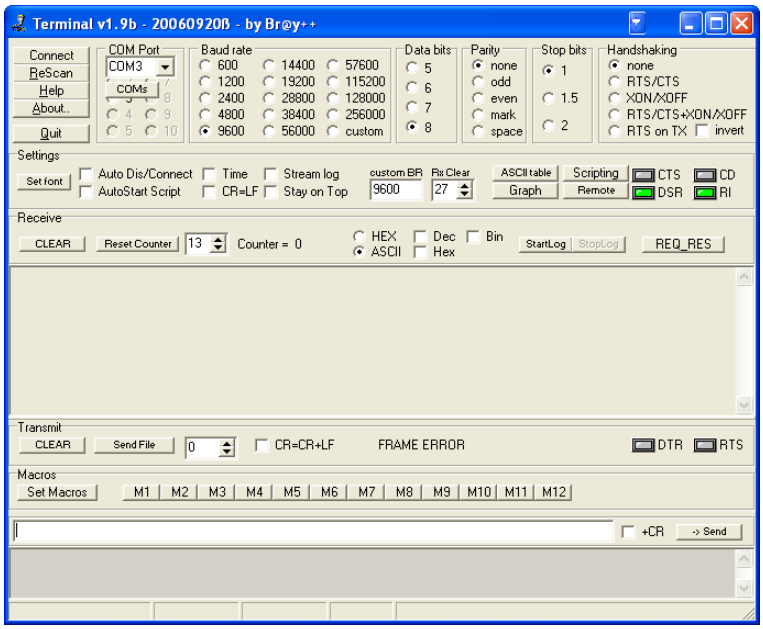
Kommunikasjonen med IO enhetene foregår med serielle linjer over RS-232. Under utvikling er det behov for å teste at kommunikasjonen fungerer. Etter litt prøving å feiling med ulike programmer, er det tre enkle programmer som har vist seg å fungere godt til sitt bruk.

4.8.1 Seriell kommunikasjon på Windows

Terminal er et enkelt og nyttig program som har det som trengs for å teste seriell kommunikasjon via RS-232 til en Windowsmaskin. Programmet kan brukes fritt og er vedlagt på CD. Skjermbilde av *Terminal* er vist i figur 4.7.

4.8.2 Seriell kommunikasjon på Linux

Under Linux er det to programmer som egner seg, *CuteCom* og *MiniCom*. Begge er gratis å bruke. Det enkleste av dem er *CuteCom* som er et grafisk program for KDE. Det ser veldig likt ut som *Terminal* fra forrige avsnitt.



Figur 4.7: Terminal v1.9b (vedlagt på CD)

CuteCom installeres på følgende vis:

```
sudo apt-get install debhelper
sudo apt-get install cmake
sudo apt-get install dpatch
sudo apt-get install libqt3-mt-dev
cd [installasjonsmappe]
wget http://ftp.debian.org/debian/pool/main/c/cutecom/ \\
        cutecom_0.14.1.orig.tar.gz
./configure
make
```

På Kubuntu 6.10 kompilerte *CuteCom* smertefritt etter installasjon av ovennevnte pakker. Figur 4.8a viser skjermbilde av CuteCom. Programmet kjøres med følgende kommando:

```
./cutecom
```

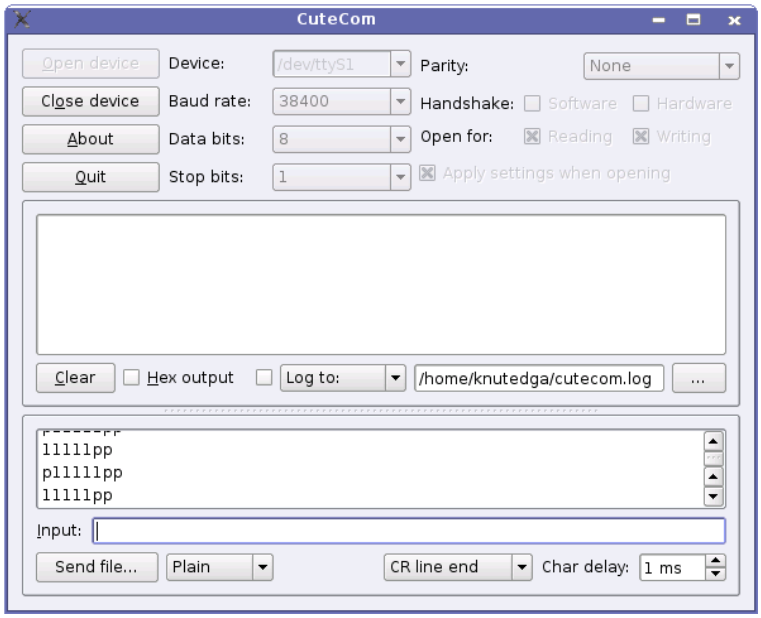
MiniCom er et konsollprogram i Linux og egner seg der det ikke er et grafisk miljø tilgjengelig. *MiniCom* er mer tungvindt og lære seg, men fungerer bra med litt trening og når programmet er satt opp riktig. Figur 4.8b viser Minicom i et konsollvindu.

Minicom installeres og kjøres slik:

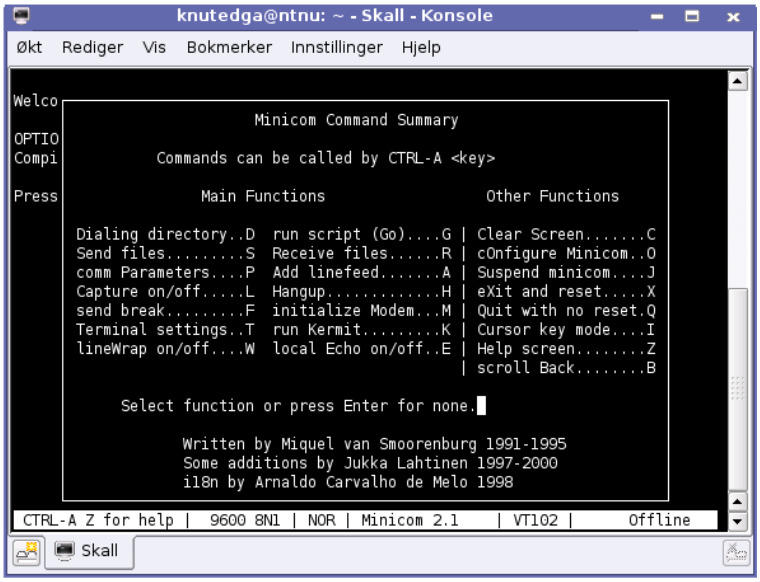
```
sudo apt-get install minicom
minicom
```

4.9 Konfigurering av Globalsat EM-411 GPS modul

Som beskrevet i avsnitt 5.3 skal styresystemet bruke GGA- og RMC-meldinger. GPS-modulen er konfigurert til å sende fire meldinger, deriblant GGA- og RMC- meldinger. Vi



(a) Skjermbilde av CuteCom på Kubuntu



(b) Skjermbilde av MiniCom i Kubuntus konsollvindu

Figur 4.8: Terminalprogram på Linux

ønsker å fjerne de unødvendige meldingene for å unngå unødvendig trafikk på serielinja. Vi ønsker også å øke hastigheten på den serielle linjen til 9600kbps siden det er standard hastighet på de serieporten i Linux. Det står dokumentert i brukerveiledningen hvordan GPS-modulen konfigureres, og meldingene som ble brukt er diskutert i de følgende avsnittene.

4.9.1 Velge mottak av NMEA-meldinger

Protokollen for å konfigurering av NMEA-meldingene som skal sendes, er som følger:

```
PSRF103,<msg>,<mode>,<rate>,<cksumEnable>*CKSUM<CR><LF>\\
```

Vi ønsker å sende følgende meldinger:

```
PSRF103, GGA, SetRate, 1sec, EnableChecksum *checksum
PSRF103, GGL, SetRate, 0sec, EnableChecksum *checksum
PSRF103, GSA, SetRate, 0sec, EnableChecksum *checksum
PSRF103, GSV, SetRate, 0sec, EnableChecksum *checksum
PSRF103, RMC, SetRate, 1sec, EnableChecksum *checksum
PSRF103, VTG, SetRate, 0sec, EnableChecksum *checksum
```

Meldingene ser da slik ut:

```
PSRF103,00,00,01,01*25<CR><LF>
PSRF103,01,00,00,01*25<CR><LF>
PSRF103,02,00,00,01*26<CR><LF>
PSRF103,03,00,00,01*27<CR><LF>
PSRF103,04,00,01,01*21<CR><LF>
PSRF103,05,00,00,01*21<CR><LF>
```

4.9.2 Konfigurere seriell kommunikasjon med GlobalSat EM-411

Protokollen for å konfigurere GPSens serielle overføring er som følger:

```
$PSRF100,<protocol>,<baud>,<DataBits>,<StopBits>,<Parity>*CKSUM
```

Vi ønsker å sende følgende melding:

```
$PSRF100, NMEA, 9600baud, 8DataBit, 1StopBit, NoneParity *CKSUM
```

Meldingen blir da seende slik ut:

```
$PSRF100,1,9600,8,1,0*0C<CR><LF>
```

Kapittel 5

Programvareoppsett for styresystemet

Styresystemet er utviklet i Simulink og eksportert med Real-Time Workshop til en kjørbar programfil. Dette kapittelet omhandler kun rammeverket for styresystemet. Selve styresystemet er beskrevet i detalj i [6].

I dette kapittelet er S-funksjonene[1] som kommuniserer med IO-enhetene i systemet beskrevet. Hensikten med dette kapittelet er å gi en oversikt over kildekoden til styresystemet. For mer spesifikke detaljer henvises det til vedlagte CD. Der er kildekoden detaljert kommentert linje for linje.

Under utvikling av programvaren ble [7] brukt som oppslagsverk sammen med nettsiden www.cplusplus.com [1].

5.1 Basismodell for CyberSwan i Simulink

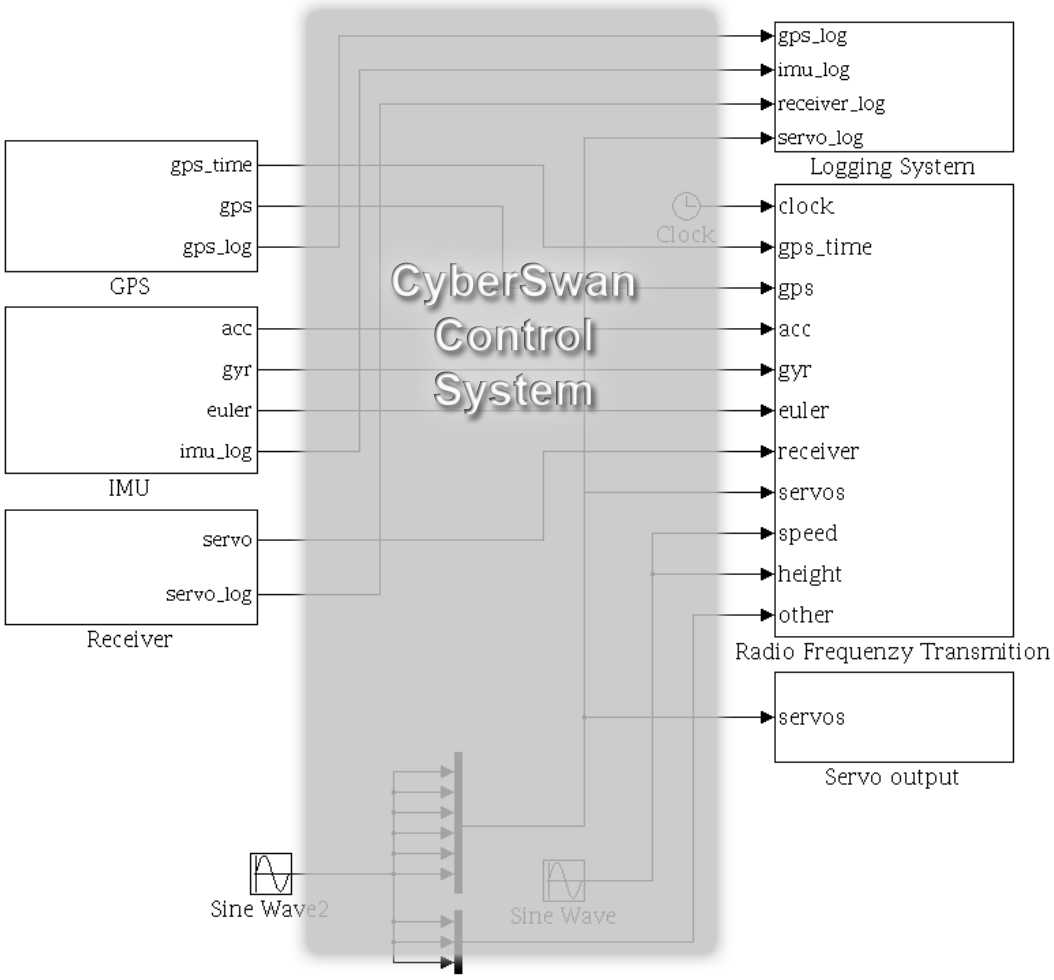
En S-funksjon gjør det mulig å integrere C/C++ kode på en enkel og strukturert måte. En S-funksjon er grei å sette opp og enkel å forstå (se kapittel 4.3). Kommunikasjonen med IO-enhetene er lagt i hver sin S-funksjonsblokk slik at det skal bli oversiktlig i Simulink. Programkoden til den enkelte IO-enheten blir også mer oversiktlig og enklere å vedlikeholde når hver IO-enhet representeres av en egen S-funksjon.

Figur 5.1 viser basismodellen i Simulink som ligger vedlagt på CD. Modellen logger inngangene til fil og sender ut et testsignal til servoene.

5.2 S-funksjon for IMU-målinger

- _kbit()
- do_settings()
- thread()

[1] S-Funksjoner er beskrevet i avsnitt 4.3



Figur 5.1: Rammeverk for CyberSwan sitt styresystem i Simulink (Simulinkmodellen er vedlagt på CD)

Figur 5.2: S-funksjon med målinger fra IMUen

- imu_start()
- imu_stop()
- imu_output_wrapper()

På CDen som følger med xSens MTi, ligger det et program som heter `MTCommExample`. Det er et konsollprogram som kommuniserer med IMUen og skriver IMU-data til skjermen. Kildekoden fra dette eksempelet danner basisen for kommunikasjonene med IMUen i denne S-funksjonen.

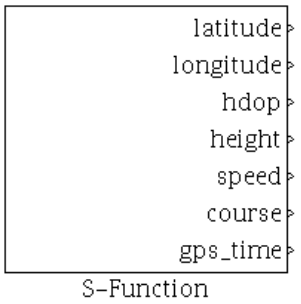
Det er en egen tråd som sørger for å lagre IMU-data. Denne opererer på 10Hz, men det er i utgangspunktet IMUen som styrer dette.

Kommunikasjonen med IMUen initialiseres i `imu_start()` hvor tråden `imu_thread()` startes og initialiseringsfunksjonene `_kbit()` og `do_settings()` fra MTCommExample kjøres. `imu_thread()` opprettholder kommunikasjonen med IMUen helt til `imu_stop()` terminerer tråden. `imu_stop()` er en funksjon som er definert i modellen i Simulink til å kjøre når programmet terminerer.

`imu_stop()` og `imu_start()` er lagt til som `Custom Code` som beskrevet i avsnitt 4.2.

`imu_output_wrapper()` er funksjonen som henter verdiene ut av S-funksjonen. Den kjøres med samme frekvens som definert i `Fixed-step size` i Simulink (avsnitt 4.2). For CyberSwan er dette 10Hz. `imu_output_wrapper()` gjør ikke annet enn å lese IMU-dataene fra et sett med variabler som tråden `imu_thread()` sørger for å holde oppdatert til enhver tid .

5.3 S-funksjon for GPS posisjonering



Figur 5.3: S-funksjon med GPS posisjonering

- gps_thread()
- gps_start()
- gps_stop()
- gps_output_wrapper()

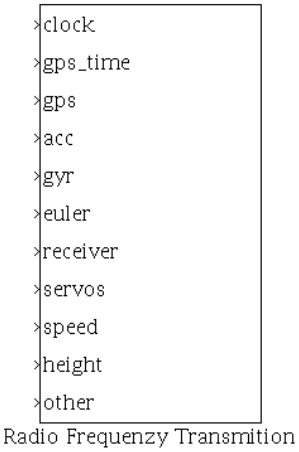
GPSens S-funksjon fungerer på tilsvarende måte som S-funksjon til IMUen. Tråden `gps_thread()` leser inn verdier som kommer fra GPSen helt til `gps_stop()` terminerer tråden når programmet avsluttes. `gps_stop()` og `gps_start()` er også lagt til som `Custom Code`, som beskrevet i avsnitt 4.2.

`gps_output_wrapper()` er funksjonen som henter verdiene ut av S-funksjonen, på tilsvarende måte som for IMUen.

Hovedforskjellen mellom S-funksjonene til IMUen og GPSen ligger i hva `imu_thread()` og `gps_thread()` gjør. I `gps_thread()` er det implementert innlesningen av to typer GPS meldinger; GGA og RMC. Begge disse meldingene er definert i NMEA 0183-protokollen. Denne typen meldinger består av en rekke verdier som ligger etter hverandre i en streng atskilt med komma. I `gps_thread()` leses disse verdiene ut og lagres som tallverdier. Disse tallverdiene hentes inn til styresystemet via funksjonen `gps_output_wrapper()`.

Det finnes ferdige biblioteker for NMEA 0183-protokollen som ligger fritt tilgjengelig på Internett, men det ble valgt å implementere en egen innlesing av GPS-meldingene fremfor å sette seg inn i bruken av slike biblioteker. Siden det er begrenset GPS-funksjonalitet som er nødvendig i CyberSwan, var det greit å implementere denne innlesingen.

5.4 S-funksjon for radiotransmisjon



Figur 5.4: S-funksjon med utganger til RF-linken

- send_imu()
- send_gps()
- send_receiver()
- send_servo()
- RF_thread()
- RF_start()
- RF_stop()
- RF_output_wrapper()

S-funksjonen som tar seg av om radiokommunikasjonen med bakken sender data fra styresystemet til basestasjonen på bakken via RF-linken. S-funksjonen får derfor inn verdier fra styresystemet, som vi ser av Simulinkblokken i figur 5.4. Disse verdiene sender den over RF-linken i henhold til protokollene som er definert i kapittel 6. `RF_output_wrapper()`

sender ikke verdier til styresystemet, men den får derimot inn verdier fra styresystemet som den sender videre. `RF_output_wrapper()` kjører funksjonene `send_XXXX()` som setter sammen de ulike meldingene som er definert, før de sendes til RF-modulen.

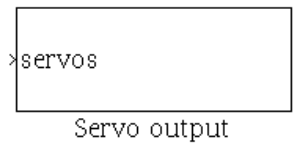
Funksjonaliteten i `RF_thread()` er ikke implementert i denne modellen siden det har vært litt uklart hva som faktisk skal sendes fra bakken til flyet. For implementasjon av denne funksjonen bør utgangspunktet være protokollene som er beskrevet i [5].

`RF_start()` starter i hovedsak `RF_thread()`, men her settes også perioden, starttid og stopptid til hver enkelt melding. Disse variablene kan dermed oppdateres med meldinger som mottas med `RF_thread()`, slik at en fra bakken kan bestemme hva slags meldinger og hvor ofte meldinger skal sendes fra CyberSwan.

```
void RF_start(void){
    ....
   // initialize the variables for testing
   next_imu_send       = 1;    // send first imu message after 1 second
   next_gps_send       = 2;    // send first gps message after 2 seconds
   next_receiver_send  = 3;    // send first rcv message after 3 seconds
   next_servo_send     = 4;    // send first srv message after 4 seconds
   next_other_send     = 5;    // not in use
   stop_imu_send       = 15;   // last imu msg will be sent on the 15th sec
   stop_gps_send       = 20;   // last gps msg will be sent on the 20th sec
   stop_receiver_send  = 10;   // last rcv msg will be sent on the 10th sec
   stop_servo_send     = 10;   // last srv msg will be sent on the 10th sec
   stop_other_send     = 15;   // not in use
   period_imu          = 0.2;  //   5Hz
   period_gps          = 2;    // 0,5Hz
   period_receiver     = 0.5;  //   2Hz
   period_servo        = 0.5;  //   2Hz
   period_other        = 0.5;  // not in use
}
```

I fila `RF_storage.txt` som ligger vedlagt på CD, er meldingene sendt til en fil i stedet for til serieporten under en testkjøring av modellen. Et utdrag av denne fila er vist i figur 5.8. Modellen kjørte i 30 sekunder, men som en ser av `stop_XXX_send`-variablene, skal det ikke sendes meldinger etter 20 sekunder. Det skal det heller ikke gjøres i `RF_storage.txt`. Fra utdraget i 5.8 ser en også at starttiden og perioden på de ulike meldingene stemmer overens med variablene som er definert i `RF_start()`.

5.5 S-funksjon for å sette ut servopådrag



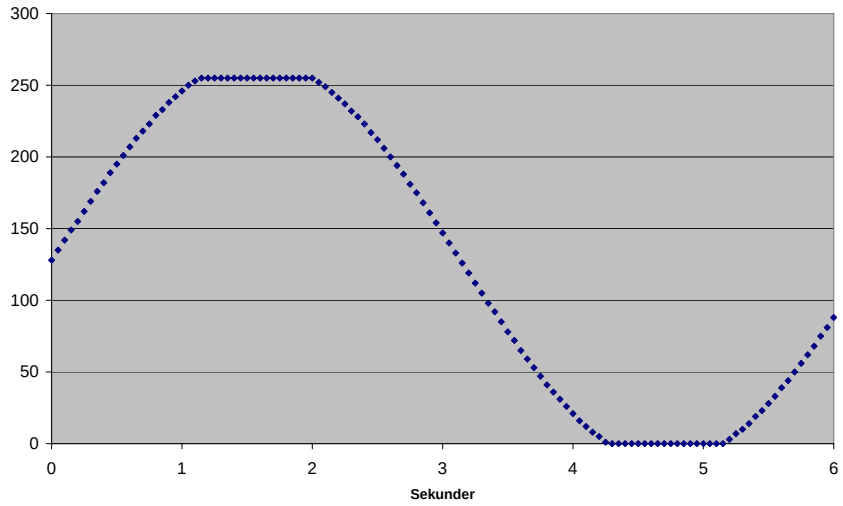
Figur 5.5: S-funksjon med pådrag til servoene

- send_servo_output()

- servo_output_wrapper()

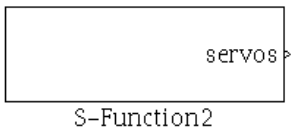
S-Funksjonen som sender meldinger til servoene er relativt enkel. Den får inn tallverdier på hver enkelt kanal fra styresystemet og sender disse videre til IO-kortet i henhold til protokollen som er definert i avsnitt 6.5. Det er `servo_output_wrapper()` som får inn tallverdiene. Denne funksjonen kjører `send_servo_output()` som så åpner serieporten og sender verdiene til serieporten før serieporten stenges igjen. Dette skjer med samme frekvens som er definert *Fixed-step size* i avsnitt 4.2, som i dette tilfelle er 0,1 sekunder eller 10Hz.

Det er lagt inn en begrensning som gjør at det ikke vil sendes et signal som er utenfor det definerte området til servoen. Hver kanal overføres som en byte til IO-kortet. S-funksjonen vil altså begrense signalet til å ligge i området [0, 255]. I figur 5.6 er det satt på en sinuskilde i Simulink som overskrider disse grensene. Serieporten er monitorert, og vi ser at S-funksjonen begrenser signalet.



Figur 5.6: Et servopådrag begrenses til definert område [0, 255] før det sendes til IO-kortet hvis kilden i Simulink overskriver maksgrensene.

5.6 S-funksjon for måling av mottagerkanaler



Figur 5.7: S-funksjon med pådrag fra mottageren

- receiver_thread()
- receiver_start()
- receiver_stop()
- receiver_output_wrapper()

S-funksjonen som kommuniserer med mottageren i flyet, får inn verdier fra modellflyradioen (figur 2.10a) nede på bakken. Styresystemet vet derfor til enhver tid utslagene på servoene selv om flyet styres manuelt fra bakken.

Siden det er IO-kortet som måler PWM-signalene fra mottageren (figur 2.10b) i flyet, får denne S-funksjonen meldinger fra IO-kortet. Disse meldingene er definert i avsnitt 6.4.

`receiver_start()` starter `receiver_thread()` som lytter kontinuerlig etter meldinger fra IO-kortet. Den lytter helt til `receiver_stop()` terminerer `receiver_thread()`. `receiver_thread()` oppdaterer globale variabler i S-funksjonen med samme frekvens som IO-kortet sender meldinger. Denne frekvensen er satt i IO-kortet til å være 10Hz 9.

`receiver_stop()` og `receiver_start()` er lagt til som `Custom Code` som beskrevet i avsnitt 4.2.

Styresystemet henter verdiene på kanalene fra mottageren med funksjonen `receiver_output_wrapper()`. Disse verdiene sørger `receiver_thread()` for er oppdatert til enhver tid. Dermed får styresystemet tilgang til utslagene på kanalene fra flyradioen nede på bakken.

5.7 Logging av data til mat-fil

Alle data logges til en mat-fil (`CyberSwan_log.mat`) i blokken *Logging System* i figur 5.1. For hvert tidsskritt logges alle data fra IMU, GPS og mottager samt alle pådragene som sendes til servoene. Etter flyvning kan mat-fila importeres og analyseres i Matlab. Siden hvert tidsskritt logges, blir både styresystemets og flyets oppførsel fullt dokumentert i mat-fila. I mat-fila har hver variabel sin egen rad slik at det legges til en ny kolonne for hvert tidsskritt programmet kjører.

Rekkefølgen på radene i mat-fila er som følger: time, acc_x, acc_y, acc_z, gyr_x, gyr_y, gyr_z, ϕ(roll), θ(pitch), ψ(yaw), GPStime, latitude, longitude, hdop, altitude, speed, course, rcv_{ch0}, rcv_{ch1}, rcv_{ch2}, rcv_{ch3}, rcv_{ch4}, rcv_{ch5}, srv_{ch0}, srv_{ch1}, srv_{ch2}, srv_{ch3}, srv_{ch4}, srv_{ch5}.

5.8 Tilkobling av IO-enheter som har forskjellig frekvens

IMUen, GPSen og IO-kortet sender meldinger til styresystemet med forskjellig frekvens. Siden alle trådene oppdaterer variablene i det tempoet den tilkoblede enheten ønsker, er frekvensen likegyldig. Styresystemet vil til en hver tid lese fullt oppdaterte verdier. For GPS vil dette si hvert sekund mens for IMU vil det si hvert tiendedels sekund og for manuell styring er et også hvert tiendedels sekund.

5.9 Testing av modellen

Det var ikke tilgjengelig nok RS-232 porter verken på arbeidsstasjonen eller PC/104 enheten (avsnitt 2.5.1) til å kunne teste modellen i sin helhet med alle IO-enhetene tilkoblet samtidig. Det ble derfor brukt filer under testingen av systemet for å simulere de IO-enhetene som ikke kunne kobles til. Meldinger fra en IO-enhet ble derfor lagt i en fil som styresystemet kunne åpne og lese på samme måte som fra en RS-232 port. Styresystemet kunne også skrive til fil på samme måte som skriving til en RS-232 port. En slik løsning er

mulig siden tilkobling til en RS-232 port i Linux foregår via en *file descriptor* på lik linje med aksessering av filer. Lik syntaks kunne derfor brukes på lesing og skriving til en fil som til en RS-232 port.

Under testingen var utgangene til systemet definert slik i modellen (avsnitt 4.2):

```
#define RF_STORE_LOCATION    "../RF_storage.txt"
#define SERVO_LOCATION       "../Servo_storage.txt"
```

Både `RF_storage.txt` og `Servo_storage.txt` er vedlagt på CD, men det er lagt med et utsnitt i figur 5.8. Siden servoprotokollen består av byte-verdier, er syntaksen endret noe for at den skal være leselig. Som det fremgår av utsnittet i fila (figur 5.8) og modellen i figur 5.1, er det et sinussignal som sendes på alle seks servokanalene. Disse kanalene er de seks første verdiene i servomeldingen som beskrevet avsnitt 6.5.

Under testingen var inngangene til systemet definert slik (avsnitt 4.2):

```
#define IMU_LOCATION         "/dev/ttyUSB1"
#define RF_RECEIVE_LOCATION  "../read_RF.txt" //not in use
#define GPS_LOCATION         "../read_gps.txt"
#define RECEIVER_LOCATION    "../read_receiver.txt"
```

IMUen var koblet til på `/dev/ttyUSB1`. `read_gps.txt` inneholdt to typer meldinger, en GGA- og en RMC-melding. Systemet vil alltid starte i starten av fila og lese ett og ett tegn fortløpende helt til den finner en ny linje. Når en ny linje er funnet vil den lukke *file descriptoren*. Den vil dermed alltid lese inn kun den første linja i fila. Hvis meldingen i den første linja endres, eller hvis rekkefølgen til de to meldingene endres under kjøring av systemet, vil en se at verdiene endrer seg i styresystemet i det samme fila lagres. Det samme gjelder for `read_receiver.txt`, hvor en melding i henhold til protokollen for mottageren ligger.

I tillegg til testing av systemet ble én og én IO-enhet testet opp imot styresystemet etter tur, alltid med IMUen tilkoblet.

IO-kortet ble testet med en et kamerastativ som ble styrt av to servoer. Dette kamerastativet kunne rotere og tilte kameraet med disse to servoene. IMUen ble brukt til å tilte kameraet opp og ned og til og panorere sideveis. Det ble lagt inn en integrator på rotasjonshastighetsmålingen i to retninger. En kunne dermed styre kameraet med IMUen. Dette fungerte overraskende bra og det var lite drift på kameraet. Etter mer enn 15 sekunder, eller mye variasjoner, kunne det merkes drift i signalet. Dette viste at både systemet fungerte som det skulle og at målingene fra IMUen er meget gode.

```
---Servo_storage.txt:-----------
141 141 141 141 141 141 6 13
128 128 128 128 128 128 6 13
128 128 128 128 128 128 6 13
129 129 129 129 129 129 6 13
129 129 129 129 129 129 6 13
...
--------------------------------

---RF_storage.txt:--------------
$CSIMU,1.000,-0.04,0.00,9.83,0.01,-0.01,-0.01,0.37,0.18,54.72*00
$CSIMU,1.200,-0.04,0.00,9.82,-0.00,0.01,0.02,0.38,0.16,54.77*00
$CSIMU,1.400,-0.05,0.00,9.83,0.01,0.01,0.01,0.40,0.18,54.80*00
$CSIMU,1.600,-0.04,-0.01,9.82,0.01,0.00,-0.00,0.39,0.19,54.79*00
$CSIMU,1.000,0.00,-0.01,9.79,0.02,0.00,0.01,0.36,0.22,54.24*00
$CSIMU,1.200,-0.04,-0.00,9.83,-0.00,-0.01,0.02,0.41,0.22,54.28*00
$CSIMU,1.400,-0.06,0.00,9.82,0.02,0.00,0.02,0.41,0.22,54.30*00
$CSIMU,1.600,-0.03,-0.01,9.80,0.01,-0.00,0.00,0.39,0.17,54.34*00
$CSIMU,1.800,-0.03,-0.01,9.81,-0.00,-0.00,0.01,0.36,0.13,54.31*00
$CSIMU,2.000,-0.04,-0.00,9.79,0.02,0.01,0.00,0.37,0.15,54.28*00
$CSGPS,2.000,,3723.2475,12158.3416,1.00,9.00,0.00,0.00*00
$CSIMU,1.000,-0.02,-0.02,9.79,-0.00,-0.01,0.02,0.29,0.23,54.26*00
$CSIMU,1.200,-0.02,-0.01,9.80,0.01,0.01,-0.00,0.30,0.24,54.24*00
$CSIMU,1.400,-0.04,-0.02,9.78,-0.01,0.01,0.02,0.33,0.26,54.21*00
$CSIMU,1.600,-0.03,-0.01,9.79,0.00,0.01,0.01,0.31,0.23,54.23*00
$CSIMU,1.800,-0.04,-0.01,9.84,0.00,-0.02,0.01,0.33,0.25,54.28*00
$CSIMU,2.000,-0.04,-0.01,9.84,0.01,0.01,-0.00,0.35,0.21,54.28*00
$CSGPS,2.000,,3723.2475,12158.3416,1.00,9.00,0.00,0.00*00
$CSIMU,2.200,-0.04,-0.01,9.78,0.01,0.01,0.01,0.36,0.22,54.27*00
$CSIMU,2.400,-0.02,-0.01,9.80,0.00,-0.02,-0.00,0.32,0.18,54.22*00
$CSIMU,2.600,-0.02,-0.02,9.79,-0.01,-0.01,0.02,0.31,0.14,54.21*00
$CSIMU,2.800,-0.02,-0.02,9.78,0.02,0.00,0.02,0.30,0.12,54.30*00
$CSRCV,3.000,121,128,32,123,0,0*00
$CSIMU,3.050,-0.03,-0.01,9.79,0.01,0.01,0.01,0.29,0.15,54.30*00
...
--------------------------------

---read_gps.txt:----------------
$GPGGA,161229.487,3723.2475,N,12158.3416,E,1,07,1.0,9.0,M,,,,0000*18
$GPRMC,161229.487,A,3723.2475,N,12158.3416,E,0.13,309.62,120598,,*10
--------------------------------

---read_receiver.txt:-----------
$MLSRV,121,128,32,123,0,0*00
$MLSRV,123,127,30,121,0,0*00
$MLSRV,120,126,31,124,0,0*00
--------------------------------
```

Figur 5.8: Filer som ble brukt i stedet for RS-232 porter under test av hele systemet

Kapittel 6

Protokoller for datatransmisjon

Meldingene som sendes mellom de ulike enhetene bør være på en bestemt kompakt form slik at mottager effektivt kan plukke ut og sortere innholdet. I dette kapittelet er protokollene som brukes for meldingsutveksling i CyberSwan beskrevet. Det er også skissert en alternativ type meldinger som kan brukes i CyberSwan. Styrker og svakheter med dette alternativet er belyst.

Det er flere former for meldingsutvikling i flyet. I dette kapittelet er det beskrevet protokoller for følgende meldinger:

- mellom CyberSwan og bakkestasjon
- mellom styresystem og IO-kort

Det er også andre typer meldinger i instrumenteringssystemet, men dette er ferdige protokoller som er dokumentert i brukermanualen for den enkelte enhet. Dette gjelder:

- meldinger fra GPSen
- meldinger til og fra IMUen

6.1 Meldinger fra CyberSwan til bakkestasjon

En melding fra CyberSwan til bakkestasjonen består av en tekststreng der alle elementene er lagt etter hverandre atskilt med komma som i figur 6.1. I starten av hver melding er det en tittel på meldingen, etterfulgt av en tidskode da meldingen ble generert. I slutten av hver melding er det en checksum som garanterer at dataen i meldingen er korrekt.

$CSIMU: CyberSwan IMU-meldingen inneholder 3D-akselerasjon, 3D-vinkelhastighet og 3D-eulervinkler som er direkte hentet fra IMUen. Benevningen på målingene finnes i dokumentasjon på IMUen.

$CSGPS: CyberSwan GPS-meldingen er satt sammen av informasjon fra GPS-meldingene GGA og RMC. Dette gjør at $CSGPS-meldingen består av absolutt 3D-posisjonering av CyberSwan i tillegg til hastighet og kursen CyberSwan beveger seg i. Det ligger også med unøyaktigheten i målingen i form av hdop i $CSGPS-meldingen. Benevningen på målingene finnes i dokumentasjon på GPSen.

$CSIMU,time,$\text{acc}_x$,$\text{acc}_y$,$\text{acc}_z$,$\text{gyr}_x$,$\text{gyr}_y$,$\text{gyr}_z$,$\phi$(roll),$\theta$(pitch),$\psi$(yaw)*CKSUM<CR><LF>

(a) IMU-melding

$CSGPS,time,latitude,longitude,hdop,altitude,speed,course*CKSUM<CR><LF>

(b) GPS-melding

$CSRCV,time,ch0,ch1,ch2,ch3,ch4,ch5*CKSUM<CR><LF>

(c) Mottakermelding

$CSSRV,time,ch0,ch1,ch2,ch3,ch4,ch5*CKSUM<CR><LF>

(d) Servomelding

$CSOTH,time,height,speed,CKSUM*<CR><LF>

(e) Diversemelding

Figur 6.1: Meldinger kodet i streng

$CSRCV: CyberSwan Receiver-meldingen inneholder seks utslag på seks kanaler fra mottageren i flyet. Mottageren får i sin tur informasjonen sin fra den manuelle radiosenderen på bakken. Verdiene ligger mellom [0, 255] i likhet med $CSSRV-meldingen.

$CSSRV: CyberSwan Servo-meldingen inneholder pådragene som styresystemet genererer til servoene. Disse pådragene ligger i området [0, 255] og tilsvarer [900, 2100] ms pulsbredde på servosignalet som i sin tur betyr [helt til venstre, helt til høyre] eller omvendt.

$CSOTH: CyberSwan Other-meldingen inneholder høydemåling fra en ultralydsensor som kan måle opp til 255 tommer (6,47 meter). Denne målingen er nødvendig under autonom landing. Det viktigste i denne meldingen derimot er hastighet på luftstrømmen målt av et pitotrør. [1]

6.2 Meldinger fra bakkestasjon til CyberSwan

Protokoll for meldinger som sendes fra bakkestasjonen til CyberSwan er beskrevet i [5]. Som nevnt i avsnitt 5.4 er denne funksjonaliteten ikke implementert, men det er tatt høyde for slike utvidelser i modellen. I [5] er det beskrevet protokoller som kan styre hvilke meldinger som skal sendes fra bakken og frekvensen på de ulike meldingene.

Det vil også være behov for å sende meldinger som påvirker oppførselen til CyberSwan. Dette kan være trimming av ror og andre parametere til flyets styre- og navigeringssystem. På den måten kan en endre CyberSwan sin oppførsel under en flyvning fortløpende uten å måtte lande flyet.

[1] Disse målingene er ikke implementert, og denne meldingen er ikke fullstendig integrert i eksempelet som ligger vedlagt på CD.

6.3 Alternativ meldingskoding fra CyberSwan til bakkestasjon

Et annet godt alternativ til å sende meldinger som tekststrenger, er å kode hvert element byte for byte i en binærprotokoll som i figur 6.2. Dette er en langt mer effektiv måte å kode meldinger på og brukes blant annet i xSens MTi. Den er derimot mindre leselig og mindre intuitiv enn en tekststreng.

'I'	time	acc_x	acc_y	acc_z	gyr_x	gyr_y	gyr_z	ϕ	θ	ψ	CKM	<CR><LF>

(a) IMU-melding (elementer på to byte)

'G'	time	lat_1	lat_2	lng_1	lng_2	hdp	alt	spd	course	CKM	<CR><LF>

(b) GPS-melding (elementer på to byte)

'S'	$time_1$	$time_2$	ch_0	ch_1	ch_2	ch_3	ch_4	ch_5	$cksm_1$	$cksm_2$	<CR>	<LF>

(c) Servomelding (elementer på en byte)

'O'	time	height	speed	CKSUM1	CKSUM1	<CR>	<LF>

(d) Diversemelding (elementer på en byte)

Figur 6.2: Alternative meldinger med binærprotokoll

De to meldingstypene har sine sterke og svake sider. Tabell 6.3 gir en oversikt over fordeler og ulemper med de to typene.

Type	*Fordeler*	*Ulemper*
Streng	Samme som GPS-meldingene (6.3.3) Lav kompleksitet (6.3.4) Intuitiv å forstå (6.3.4) Kan leses i terminal (6.3.4)	Sender mange ekstra tegn (6.3.1) Lite effektivt (6.3.1) Lav frekvens (maks 3Hz) (6.3.2)
Binær	God utnyttelse (6.3.1) Høy frekvens (maks 10Hz) (6.3.2)	Mer kompleks (6.3.4) Kan ikke leses i terminal (6.3.4) Høyere brukerterskel (6.3.4)

Tabell 6.1: Fordeler og ulemper med de to meldingstypene

6.3.1 Effektivitet

```
$CSIMU,12.80,-0.11,-0.09,9.80,0.02,0.00,-0.00,-0.45,0.74,-161.85*00
```

Figur 6.3: IMU-melding som er sendt fra CyberSwan

Observasjoner av målingene fra IMUen viser at hver enkelt måling varierer med minst ± 0.01. Det betyr at det er ikke nødvendig med bedre nøyaktighet enn to desimaler. Verken akselerasjon, vinkelhastighet eller eulervinkler vil komme utenfor ± 180. Det betyr at antall

forskjellige verdier på hver enkelt tallverdi er gitt ved:

$$\frac{definisjonsmengde}{opplsning} = \frac{2 \cdot 180}{0.01} = 36000 \tag{6.1}$$

To byte (16bit) kan representere 65536 forskjellig verdier (2^{16}). Siden $65536 > 36000$ vil to byte være nok til å representere hver enkelt måling fra IMUen i en binærprotokoll. Derfor er elementene i figur 6.2a på to byte. Til sammenligning bruker meldingen i figur 6.3 fra 4 til 7 byte per måling.

```
$CSIMU,9999.99,-50.00,-50.00,-50.00,-20.00,-20.00,-20.00,
-180.00,-90.00,-180.00*00<CR><LF>
```

Figur 6.4: IMU-melding med maksimal lengde

For å sammenligne de to formene for koding av meldinger er utgangspunktet *worst-case scenario*. IMU-meldingen i figur 6.4 består av 82 byte (*worst-case*). Ingen IMU-melding vil være lengre enn dette. IMU-meldingen i figur 6.2a har til sammenligning en fast lengde på 25 byte (*worst-case*). Lengden på meldingen kan dermed reduseres fra 82 til 25 byte uten å miste informasjon.

6.3.2 Frekvens på meldingene over RF-linken

Som vi så på i forrige avsnitt består IMU-meldingen av maksimalt 82 tegn eller 82 byte, mens binærprotokollen av IMU-meldingen har en fast lengde på 25 byte. Båndbredden som er tilgjengelig på RF-linken er på 4,8kbps eller 600 byte per sekund. Teoretisk er det derfor maksimalt mulig å overføre IMU-meldinger av strengtypen ved 7.3Hz, mens binærmeldingene kan sendes over RF-linken ved 24Hz.

Det er derimot ikke mulig å oppnå full kapasitet uten å bruke maksimal pakkestørrelse på RF-modulen. Dette, i tillegg til annet overhead i RF-modulen som er spesifisert i dokumentasjonen, gjør at antall meldinger per sekund reduseres kraftig. Det er sannsynlig å anslå at strengmeldingene vil kunne sendes ved 3Hz og binærmeldingene sendes ved 10Hz. RF-modulen er beskrevet i [5].

6.3.3 Standard protokoller

```
$GPGGA,124637.000,6325.0840,N,01024.1304,E,1,06,1.8,56.1,M,41.5,M,,0000*6B
$GPRMC,124637.000,A,6325.0840,N,01024.1304,E,0.16,46.98,150607,,*38
```

Figur 6.5: GPS-melding som er mottatt av GlobalSat EM-411

De to GPS-meldingene i figur 6.5 er mottatt med GPS-modulen. Både GGA og RMC er standard GPS-meldinger som er spesifisert av NMEA 0183-protokollen. Denne typen meldinger har en kjent form. Meldingene fra CyberSwan som vi finner i figur 6.1 bruker derfor samme type syntaks.

Dokumentasjonen på IMUen som ligger vedlagt på CD, beskriver en syntaks der det er satt av to eller fire byte til hver tallverdi ettersom det er heltall eller flyttallsverdier i meldingene. Syntaksen på disse meldingene er bygget opp på samme måte som meldingene i figur 6.2. Denne typer meldinger er vanlig i lavnivå kommunikasjon der hurtighet er viktig. Nå kan det nevnes at også NMEA 0183-protokollen har definert en binærprotokoll.

6.3.4 Lesbarhet og brukerterskel

Det er betydelig enklere å håndtere meldinger bestående av strenger. Slike meldingene kan mottas via et Terminalprogram for seriell kommunikasjon (se avsnitt 4.8 for anbefalte programmer). Dette er en stor fordel når systemet skal testes av en ny utvikler. Når det er maskinvare med mange ledninger og forskjellige koblinger som kan kobles feil, er det viktig å legge brukerterskelen på et så lavt nivå som mulig.

6.4 Protokoll for mottager-meldinger fra IO-kortet

```
\$MANUL,ch0,ch1,ch2,ch3,ch4,ch5*CKSUM<CR><LF>
```

Figur 6.6: Melding som sendes fra ATMega128 på IO-kortet med målinger av manuelle servopådrag

Meldignen i 6.6 sendes fra ATMega128 ved 10 Hz. Hver av kanalene på IO-kortet måles med avbrudd som beskrevet i 7.5. Disse meldingene var sentrale for logging av data under første flyvning som beskrevet i kapittel 10.

6.5 Protokoll for servostyring

ch0	ch1	ch2	ch3	ch4	ch5	x	x	x	x	x	x	x	x	x	x

Tabell 6.2: Protokoll for servostyring

Servomeldingen som sendes til IO-kortet er kodet binært som vi ser i figur 6.2. IO-kortet tar inn ett og ett tegn fortløpende. De seks første tegnene som IO-kortet mottar, vil definere utslaget på den enkelte kanalen som vist i figur 6.2. Dette forutsetter at en av X-ene er `<CR>`. `<CR>` genereres med [Enter] i terminalprogrammene beskrevet i 4.8. Hvis ingen av X-ene er en `<CR>` vil ingen kanaler endre verdi.

X-er kan ha forskjellige verdier:

- <CR> vil overføre ch0-ch6 til servoene og klargjøre for mottak av ny melding (ingen flere X-er godtas etter <CR>).
- 'y' vil skru på ACK. Når ACK er skrudd på, vil alle tegn som mottas, returneres.
- 'n' vil skru av ACK.
- 'p' vil returnere (*print*) alle kanalene med følgende syntaks:
  `pwm:128,128,128,128,128,128<CR><LF>`

6.5.1 Eksempel 1:

```
111111<CR>
zzzzzz<CR>
```

Når IO-kortet mottar siste tegn i første melding som er <CR>, vil alle kanalene settes til '1' som ifølge ASCII-tabellen er 49. En <CR> klargjør også for mottak av nye meldinger. Deretter vil alle kanalene settes til 'z' som har ASCII verdi 127. Hvis en eller flere servoer er koblet på, vil en se et kraftig utslag på servoene.

6.5.2 Eksempel 2:

```
111111AAAAA<CR>
zzzzzzAAAAAA<CR>
```

Disse meldingene vil gi nøyaktig samme resultat som over; siden ingen av x-ene er 'y', 'n', eller 'p', vil alle A-ene bli ignorert.

6.5.3 Eksempel 3:

```
111111p<CR>
zzzzzzp<CR>
zzzzzzp<CR>
```

Vil returnere:

```
pwm:[X],[X],[X],[X],[X],[X]<CR><LF>
pwm:49,49,49,49,49,49<CR><LF>
pwm:127,127,127,127,127,127<CR><LF>
```

Første melding vil sette alle kanalene til '1'=49, men dette skjer først når den mottar <CR>. Derfor vil den første p-en skrive ut de forrige kanalverdiene som vi ikke kjenner (illustrert med `[X]`). Neste melding vil skrive ut alle kanalene som nå er '1'=49 og deretter endre alle kanalene til 'z'=127. Siste melding vil ikke endre utslagene på servoene siden de er samme som forrige verdier, men den vil skrive ut alle kanalene som fortsatt er 'z'=127.

6.5.4 Eksempel 4:

```
<CR><CR><CR><CR><CR><CR><CR>
111111p<CR>
```

Vil returnere:

```
pwm:13,13,13,13,13,13<CR><LF>
```

Første melding vil sette alle kanaler til 13, som er ASCII-verdien for <CR>. Neste melding vil endre alle kanaler til '1'=49 og skrive ut forrige kanalverdier.

6.5.5 Eksempel 5:

```
111111<CR>p<CR>
111111<CR>
111111p<CR>
```

Vil returnere:

pwm:112,13,49,49,49,49<CR><LF>

Dette kan best beskrives som *feil syntaks* og er identisk med følgende meldinger:

111111<CR>
p<CR>1111<CR>
111111p<CR>

som vil returnere:

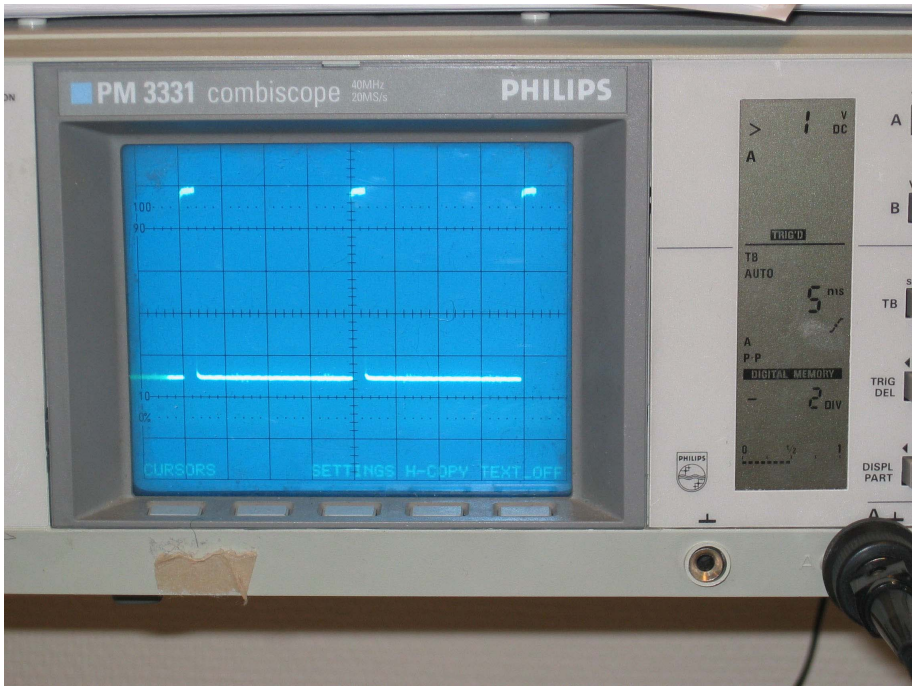
pwm:112,13,49,49,49,49<CR><LF>

Kapittel 7

PWM-signalbehandling på ATMega128

I dette instrumenteringssystemet blir PWM-signaler brukt til å definere utslaget på servoene. ATMega128 brukes til å ta imot og generere PWM-signaler på IO-kortet. Dette kapitlet tar for seg PWM-signaler og teorien rundt hvordan ATMega128 mottar og genererer PWM-signaler til servoene. I slutten av kapittelet er det beskrevet en test med mottak av PWM-signaler på IO-kortet som er beskrevet i kapittel 8.

7.1 Hva er et PWM-signal



Figur 7.1: Servostyringens PWM-signal generert av ATMega128

Et pulsbredde-modellert signal (PWM) gir ut en puls med en fast frekvens. Informasjonen i

signalet ligger i bredden på denne pulsen. Når dette brukes til servostyringen, angir bredden på pulsen utslaget på servoen. I servosignalet er periodetiden 20 ms og pulsbredden skal ligge mellom 0,9 ms og 2,1 ms. En pulsbredde på 0,9 ms setter servohodet i maksposisjon på den ene siden. En pulsbredde på 2,1 ms setter servohodet i på maksposisjon på den andre siden. En puls på 1,5 ms vil midtstille servoen.

7.2 Tellere på ATMega128

Det er tre interne tellere på ATMega128. Disse tellerne er direkte synkronisert med den klokkefrekvensen mikrokontrolleren opererer på. Dette gir disse tellerne bedre nøyaktighet enn hva som er mulig i programvare siden avbruddshåndering ikke vil ha påvirkning på tellernes oppførsel.

7.3 Oppløsning på servoene

Servoene i CyberSwan er av typen HS-81. Test av denne servoen viser at den har en nøyaktighet på 0,4° ±0,1°. Dette ble testet med en 30cm lang pinne festet på servohodet. Det minste utslaget som var mulig å få servoen til å produsere, var 2 mm på den 30 cm lange armen. ATMega128 ble brukt til å generere PWM-signalet i denne testen. Video av denne testen er vedlagt på CD.

PWM-signalet som skulle til for å endre servohodet med 0,4°, måtte ha 4 μs lengre pulsbredde enn det foregående. Pulsbredden på PWM-signalet til en servo, skal ligge mellom 900 μs og 2100 μs. Dette gir et aktivt område på 1200 μs. Siden servonøyaktigheten er på 4 μs, vil servoen ha ca 300 forskjellige utslag. Siden en byte er 8 bit og 8 bit gir 256 forskjellige verdier, vil det holde å bruke én byte i overføringen fra PC/104-enheten. Dette gir da litt færre enn de rundt 300 nivåene servoen takler, men det vil være mer enn nok. Å bruke 8 bit forenkler kompleksiteten og overføringsprotokollen.

Når verdien fra PC/104-enheten skal konverteres til en puls med en gitt bredde, gjøres dette med følgende formel:

$$pulsbredde\ [ms] = 911 + 4.6 \cdot PWMverdi \tag{7.1}$$

PWM-verdi (8 bit) i området [0, 255] gir en pulsbredde i området [911, 2084] ms mot maksimum [900, 2100] ms som servoer er definert for. Det er altså en liten margin i hver ende av signalet til servoen.

7.4 Generere PWM-signal

ATMega128 har en intern PWM-kontroller for generering av åtte PWM-kanaler hvorav seks av dem har 16 bits nøyaktighet og to har 8 bits nøyaktighet. Tabell 7.1 viser at en 8 bits timer ikke vil kunne levere flere enn 15 forskjellige vinkelutslag på servoen. Timer/Counter1 og Timer/Counter3 som begge er på 16 bit, kan generere tre PWM-signaler hver, altså seks PWM-signaler totalt.

Programvaren for ATMega128 er beskrevet i kapittel 9. Den er også vedlagt på CD.

Teller	*Bit*	*Antall nivåer på et servosignal*
Timer/Counter1	16	$\frac{2,1ms-0,9ms}{20ms} \cdot 2^{16} = 3932$
Timer/Counter2	8	$\frac{2,1ms-0,9ms}{20ms} \cdot 2^{8} = 15,4$
Timer/Counter3	16	$\frac{2,1ms-0,9ms}{20ms} \cdot 2^{16} = 3932$

Tabell 7.1: Antall mulige servoutslag som det er mulig å generere med de ulike tellerne på ATMega128

16 bit teller	*Servo (HS-81)*	*8 bit i overføring*
3932	300	256

Tabell 7.2: Konverteringsbegrensninger - antall mulige nivåer av servoutslag

7.5 Motta PWM-signal på ATMega128

ATMega128 har en *Input Capture Unit* som kan ta opp hendelser på en inngang og gi dem en tidskode. Det kan hende denne hadde vært fin å bruke til i ta opp PWM-signaler fra mottageren, men siden det var det var spesielle hensyn å ta i dette tilfelle, var det mer aktuelt å bruke avbrudd sammen med en 16 bits teller til å måle PWM-signalene fra mottageren.

Krav som måtte tilfredsstilles for mottak av PWM-signal:

- 16 bits teller for å få god nok nøyaktighet
- måling av seks kanaler fra mottageren
- bruke færrest mulig innganger på ATMega128
- må ikke komme i konflikt med PWM-utgangene

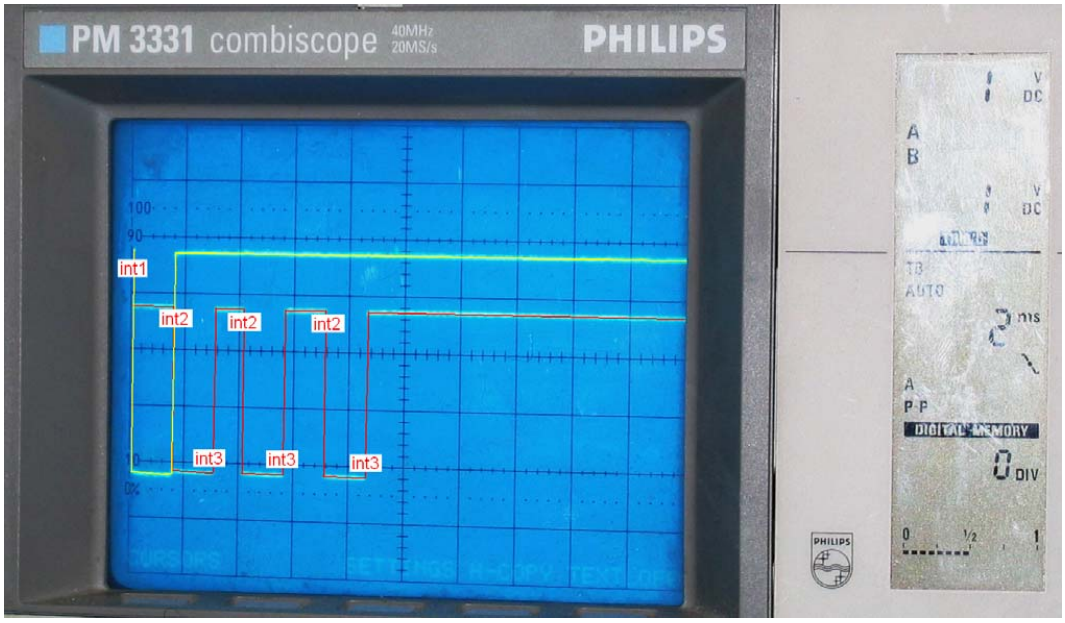
Spesielle forhold som kunne utnyttes:

- PWM-kanalene fra mottageren er faseforskøvet
- ledige utganger på PALen kan brukes til å multiplekse PWM-signalene før de sendes til ATMega128 siden de er faseforskøvet i forhold til hverandre
- PWM-inngangssignalene har samme periodetid (20 ms) som PWM-utgangssignalene
- PWM-inngangene kan bruke samme timer som PWM-utgangene uten å komme i konflikt med hverandre

Det å bruke færrest mulig innganger var meget aktuelt siden det ikke var flere enn fire ledige pinner til eksterne avbrudd på ATMega128. Mange innganger ville også gitt et større IO-kort siden seks parallelle baner ville tatt mye plass på kortet.

De seks PWM-kanalene fra mottageren i figur 7.3 er faseforskøvet, men ikke som først antatt. Det mest naturlige ville vært at de var faseforskøvet med 2,1 ms pluss en liten margin mellom hver kanal. De ligger imidlertid fortløpende etter hverandre, slik at den neste begynner idet den forrige slutter. Når kanal 0 går lav, går kanal 1 høy. Når kanal 1 går lav, går kanal 2 høy, osv. Kanalene ligger dermed helt inntil hverandre, og starten på en kanal er gitt ut i fra lengden på alle kanalene før.

Metoden for å måle pulsbredden på hver enkelt kanal er illustrert i figur 7.2. Starten på kanal 0 brukes til å starte en opptakssekvens for en periode. Fallende flanke på kanal 0



Figur 7.2: Avbrudd på multiplekset PWM-kanaler fra mottager

trigges med `INT1`[1] på ATMega128. Deretter trigges slutten på hver enkelt kanal fortløpende med `INT2` og `INT3` avhengig om det er stigende eller synkende flanke.

På denne måten er det nødvendig å føre to baner inn på tre innganger på ATMega128. `INT2` og `INT3` føres inn på hver sin inngang siden avbrudd enten kan trigge på positiv eller negativ flanke. `INT2` trigger på negativ flanke, mens `INT3` trigger på positiv flanke. En kunne sluppet unna med to innganger, men da måtte avbruddsrutinen selv endret avbruddet fra å trigge på positiv eller negativ flanke annenhver gang. Ved å bruke to avbrudd slipper en denne skiftingen og det er dermed mindre sjanse for feil. Hvis det hadde vært et ulikt antall innganger som skulle måles, for eksempel 5 eller 7, hadde det vært tilstrekkelig med en bane mindre og et avbrudd mindre, som det går an å se ut i fra figur 7.2.

Kritiske feilsituasjoner:

- **Alle seks kanaler må kobles til**[2]. Hvis for eksempel kanal seks ikke er koblet til, vil det ikke er være entydig om kanal seks ligger høy eller lav. Hvis den ligger høy hele tiden, vil den slette det ene signalet og `INT2` og `INT3` vil trigges.
- **Alle kanalene må kobles i rett rekkefølge.** Hvis dette gjøres feil, vil det skje verre ting enn at det måles feil kanal. Ved feil rekkefølge vil grafen på figur 7.2 se annerledes ut og avbruddene vil trigge på en annen måte enn forventet. Resultatet er at feilen vil forplante seg til alle resterende kanaler etter der feilen ligger. Hvis kanal 0 er rett, vil feilen nullstilles hver periode. Hvis for eksempel kanal 4 er byttet med kanal 3 vil kanal 3, 4 og 5 måles feil, mens kanal 0, 1 og 2 måles riktig.
- **Mottageren har en annen oppførsel en forventet**. Hvis ikke alle kanalene ligger fortløpende etter hverandre, vil ikke utgangssignalene se ut som i figur 7.2. Hvis dette blir et problem ved bytte av mottager vil det være behov for å modifisere PAL-kretsen,

[1] INT1: External Interrupt 1

[2] Dette gjelder strengt tatt bare kanal 1,2,4,6 siden det er disse kanalene som multiplekses av PALen, de andre kanalene er gitt implisitt.

PCB-printet og kanskje avbruddsrutinene på ATMega128.

7.6 Tidsforsinkelse for opptaksenheten av PWM-signal

PWM-signalet fra mottageren opererer på 50 Hz med en periodetid på 20 ms. Opptaksenheten på ATMega128 oppdaterer verdiene i interne variabler med samme frekvens. ATMega128 kan settes opp til å sende dataene ved valgfri frekvens, men dataene som mottas på PC/104-enheten vil aldri være eldre enn periodetiden på 20 ms i tillegg til forsinkelsen i serieforbindelsen. I tillegg vil det være en forsinklese før styresystemet mottar verdien som er mottatt på serieporten. CyberSwan sitt styresystem opererer på 10 Hz som gir en maksimal forsinkelse med periodetiden på 100 ms internt på PC/104 enheten. Dette er forutsatt at IO-kortet ikke sender meldinger med lavere frekvens enn 10 Hz. Den totale forsinkelsen er: forsinkelse fra bakke til mottager + periodetid på PWM-signalet + forsinkelse i serieforbindelse + periodetid til CyberSwan sitt styresystem.

Vi antar tallverdier på verdiene som ikke er kjent:

- forsinkelse fra bakke til mottager: (10 - 50) ms (worst case antagelse)
- periodetid på PWM-signalet: 20 ms (worst case antagelse)
- forsinkelse i serieforbindelse: (1-10) ms (worst case antagelse)
- periodetid til CyberSwan sitt styresystem: 100 ms (worst case)

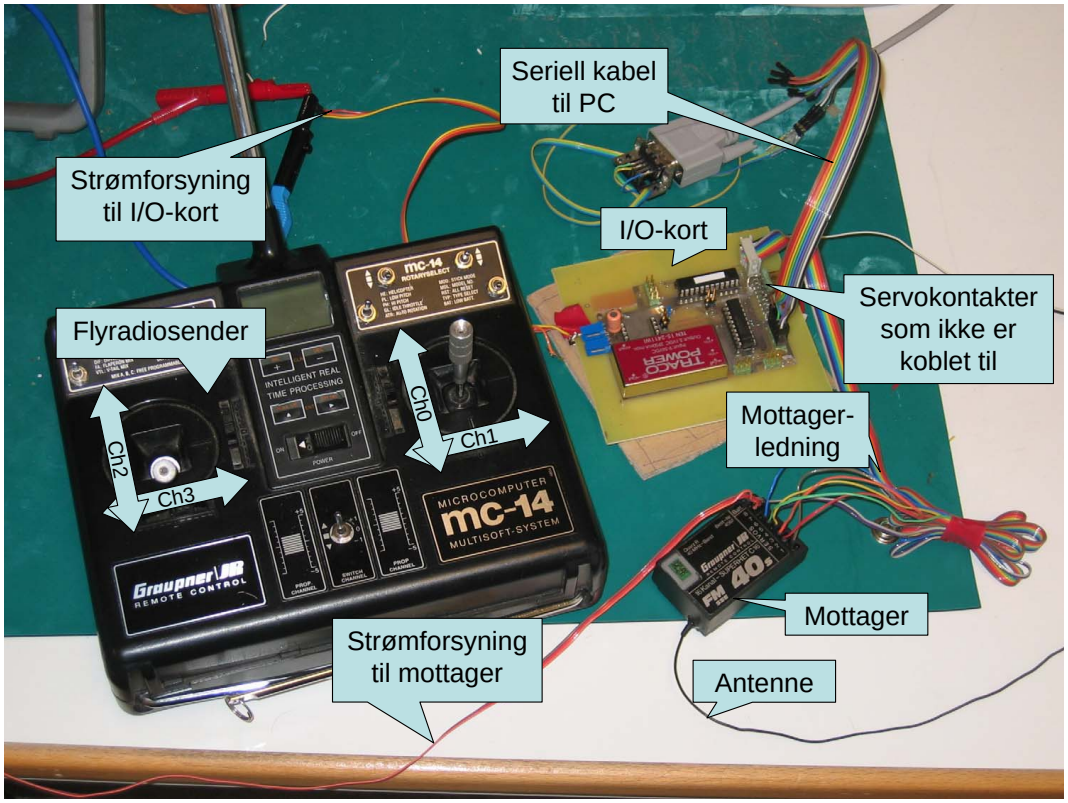
Total forsinkelse er da gitt ved: (10 - 50) ms + 20 ms + (1 - 10) ms + 100 ms = (131 - 180) ms (worst case)

I verste fall vil altså forsinkelsen ligge et sted mellom 131 ms og 180 ms fra et signal endres på modellflyradioen til det er registrert av CyberSwan sitt styresystem. En bør altså ta hensyn til at et mottagersignal kanskje vil ligge ett tidsteg bak de andre målingene, men denne tidsforsinklesen er ikke kritisk siden denne målingen ikke direkte har å gjøre med styringen av flyet på samme måte som for eksempel IMUen. Under flyturen hvor data ble logget (kapittel 10) var denne tidsforsinkelsen omtrent ikke merkbar.

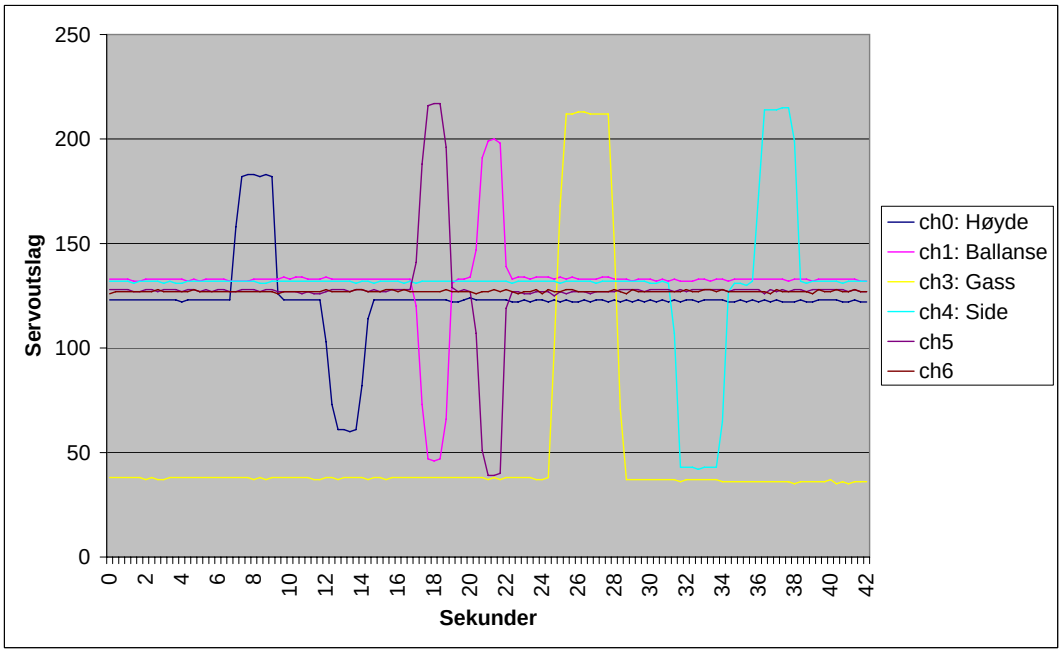
7.7 Testoppsett for mottaksenheten av PWM-signal på ATMega128

Mottageren ble koblet til inngangene til PALen via en egen kontakt på IO-kortet som er merket *Mottager*. RS-232-utgangen fra ATMega128 ble koblet til en serieport på PC-en som kjørte et terminalprogram som så leste dataene på serieporten. Det var ikke koblet til noen servoer i dette tilfelle siden det kun var målingen av PWM-signalene som skulle testes. ATMega128 sender data ved 3Hz i dette testoppsettet.

Figur 7.4 viser en sekvens der stikkene på flyradiosenderen beveges rolig til maks utslag til hver side etter tur. Alle kanalene fungerer som forventet. Gassen står gjerne på minimum, noe den også gjør her som vi ser på den gule grafen i figur 7.4, og som vi ser på bildet i figur 7.3. Kanal 5 er en invertert kopi av balanserorkanalen på kanal 1. Det er standardinnstillingene på radiosenderen som er satt opp på denne måten, slik at balanserorene kan kobles til hvert sitt signal. Det er ikke relevant her, men forklarer det som skjer på kanal 5. Dette



Figur 7.3: Testoppsett for logging av PWM-signal fra PPM-mottager



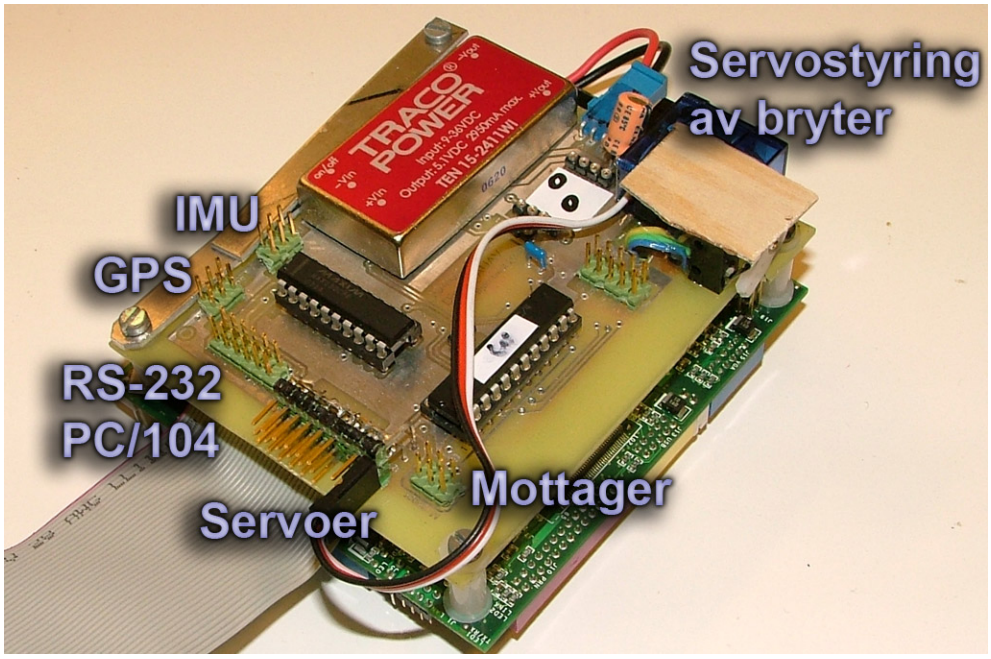
Figur 7.4: PWM-signal logget av ATMega128 ved 10 Hz

er et oppsett som gjelder for dette testutstyret (figur 7.3) og konfigureres på radiosenderen etter eget ønske alt ettersom hva slags utstyr som brukes. Dette utstyret er satt opp til ikke å bruke all tilgjengelig utslag også. Vi ser at maksutslag ikke går helt til bunn og topp. Det er også forskjellig maksutslag på hver kanal, og forskjellig midtposisjon på hver kanal. Dette skyldes at oppsettet på denne flyradioen er optimalisert for en bestemt flymodell, og har ingenting med unøyaktighet i målingen å gjøre.

Dataene som finnes på CDen viser at standardavviket i denne testen ligger på under 0,3%. Som vi ser fra grafen er dette neglisjerbart. Den fysiske slarken i servooverføringene vil for eksempel gi betydelig større usikkerhet enn det vi finner i denne målingen.

Kapittel 8

Utviklet maskinvare (Kretskort for komponenter og I/O)

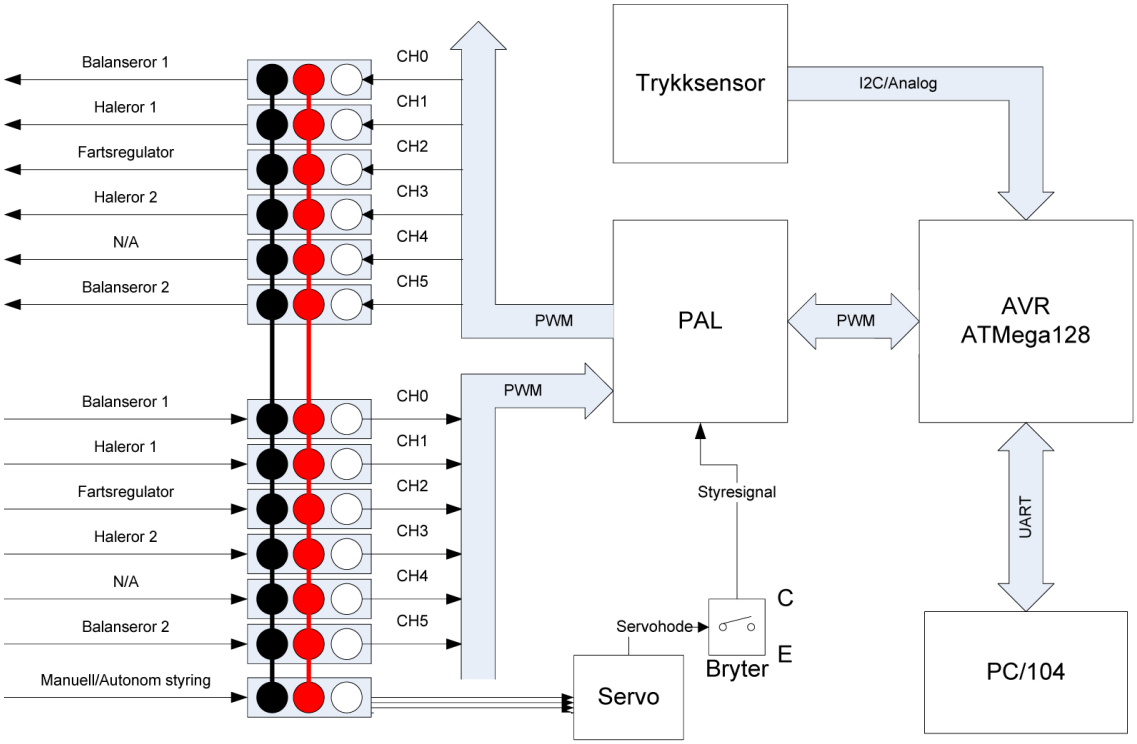


Figur 8.1: Dette er en oversikt over instrumenteringssystemet

Figur 8.1 viser et bilde av IO-kortet som er utviklet til CyberSwan. Funksjonen til dette IO-kortet er illustrert i figur 8.2. I tillegg til disse funksjonene er det lagt til tilkoblinger for GPS og IMU. Strømforsyningen til instrumenteringssystemet er også lagt på IO-kortet.

I dette kapitlet beskrives IO-kortet og alle dets komponenter. Det er også gjort en evaluering av IO-kortet i slutten av kapittelet. Programvaren til mikrokontrolleren på IO-kortet er beskrevet i kapittel 9 og signalbehandlingen i forbindelse med servoene er beskrevet i kapittel 7.

Kapitlet starter med beskrivelse av komponentene etterfulgt av hvilke tilkoblinger som finnes på IO-kortet og hva de brukes til. Strømforsyningen er også beskrevet etterfulgt av designverktøyet for IO-kortet samt en evaluering av IO-kortet helt til slutt.



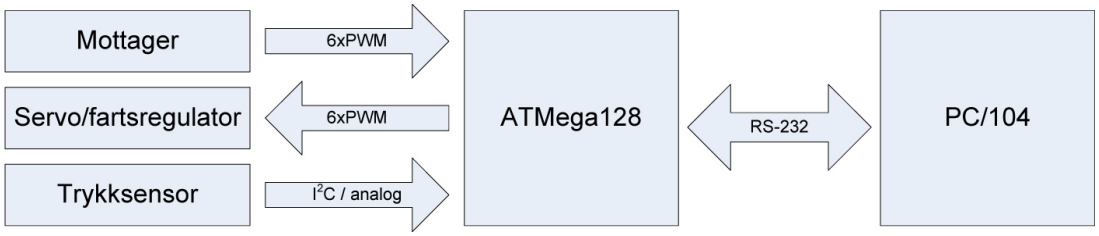
Figur 8.2: Funksjonsdiagram for IO-kort

8.1 ATMega128

I tillegg til å kommunisere med PC/104 enheten brukes ATMega128 i hovedsak til tre ting; generere PWM-signaler til servoene, motta PWM-signaler fra mottageren og til å ta imot sensordata via en analog inngang eller via I^2C på mikrokontrolleren. Det finnes riktignok egne IO-kort til PC/104 som kunne vært brukt til dette formålet, men dette ville både vært tungt, plasskrevende og upraktisk siden det ikke ville vært mulig å samle alle komponentene på ett kort hvis en skulle kjøpt en ferdig løsning.

Mikrokontrolleren ATMega128 støtter 8 PWM-utganger og den har innebygd ADC[1], I^2C-kontroller og UART-kontroller. Atmel var et naturlig valg siden Atmel sine mikrokontrollere er mye brukt på instituttet.

ATMega128 og dens oppgaver er nærmere beskrevet i kapittel 7 og kapittel 9



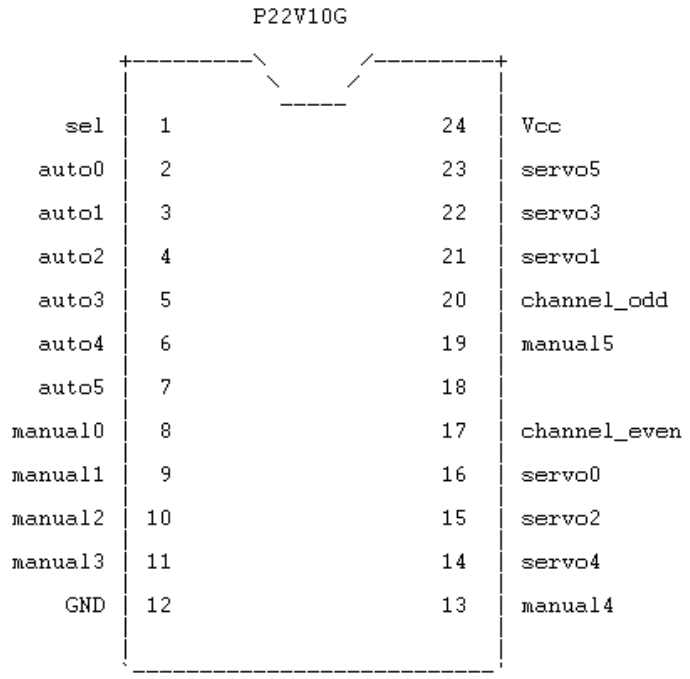
Figur 8.3: ATMega128s oppgaver

[1]ADC: Analog-Digital Converter

8.2 PAL-krets for å skifte mellom autonom/manuell styring

PAL-kretsen gjør det mulig å skifte mellom to forskjellige signalkilder til servoene. En slik skifting trengs for å kunne ha mulighet til både å styre manuelt og autonomt. PAL-kretsen har to innganger og én utgang, hver med seks signaler. Den har også styresignal og spenningsforsyning samt signal til ATMega128 for måling av kanalene fra mottageren.

8.3 Programvare på PAL-kretsen



Figur 8.4: Oversikt over pinnene på PALen (laget av VDHL-kompilatoren)

PALen må ha tolv innganger og seks utganger. Den vil ta kunne ta inn seks PWM-signaler fra både styringssystemet og mottageren for manuell styring og sende ut seks PWM-signaler til servoene. I tillegg må den ta inn et styresignal og strømforsyning samt to signaler til ATMega128 for måling av de manuelle kanalene. En PAL-krets opererer med nanosekunder fra inngang til utgang mens PWM-signalet som skal sendes igjennom, har timing på mikrosekunder. Det vil derfor ikke være problemer med forvrengning eller tidsforskyvning av PWM-signalet gjennom PALen.

For å programmere PALen ble det brukt en spesiell enhet fra Datamann som kan programmere logikkretser. Denne enheten ble lånt på Teknisk Kybernetikk sin sanntidssal. For å programmere slike kretser brukes VHDL[2]. VHDL-koden for PAL-kretsen er å finne i figur 8.5. Hvordan programmering av PAL-kretsen gjøres er beskrevet i øvingsopplegget til faget *TTK4155 - Industriell datasystemkonstruksjon*. Denne oppgaveteksten ligger vedlagt på CD.

Det ble testet om timingen av PWM-signalene i skiftingen kunne skape problemer for servoen. Som vi så på i avsnitt 7.1, skal PWM-signalet til servoene ha 20 ms periodetid.

[2]VHDL: Very High Speed Intergrated Circuit Description Language

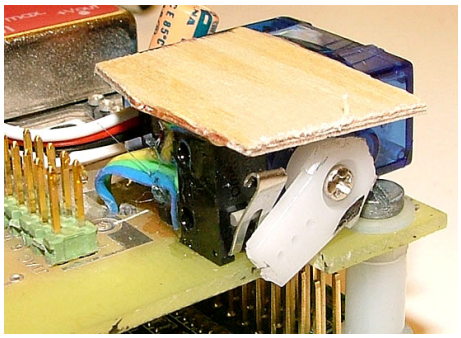
```
--------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity PAL is
PORT( sel : In  std_logic; --Definerer innport
auto0 : In  std_logic; --Definerer input fra auto0
auto1 : In  std_logic; --Definerer input fra auto1
auto2 : In  std_logic; --Definerer input fra auto2
auto3 : In  std_logic; --Definerer input fra auto3
auto4 : In  std_logic; --Definerer input fra auto4
auto5 : In  std_logic; --Definerer input fra auto4
manual0 : In  std_logic; --Definerer input til mottager0
manual1 : In  std_logic; --Definerer input til mottager1
manual2 : In  std_logic; --Definerer input til mottager2
manual3 : In  std_logic; --Definerer input til mottager3
manual4 : In  std_logic; --Definerer input til mottager4
manual5 : In  std_logic; --Definerer input til mottager4
channel_even : Out  std_logic; --Definerer utput fra alle mottagerkanalene
channel_odd  : Out  std_logic; --Definerer utput fra alle mottagerkanalene
servo0 : Out  std_logic; --Definerer utport til servo0
servo1 : Out  std_logic; --Definerer utport til servo1
servo2 : Out  std_logic; --Definerer utport til servo2
servo3 : Out  std_logic; --Definerer utport til servo3
servo4 : Out  std_logic; --Definerer utport til servo4
servo5 : Out  std_logic --Definerer utport til servo4
);
end PAL;

architecture behavioral of PAL is
begin
channel_even <= not (manual0);
channel_odd <= not (manual1 or manual3 or manual5);
servo0 <= ((not sel) and manual0) or (sel and auto0);
servo1 <= ((not sel) and manual1) or (sel and auto1);
servo2 <= ((not sel) and manual2) or (sel and auto2);
servo3 <= ((not sel) and manual3) or (sel and auto3);
servo4 <= ((not sel) and manual4) or (sel and auto4);
servo5 <= ((not sel) and manual5) or (sel and auto5);
end behavioral;
--------------------------------------
```

Figur 8.5: VHDL-kode for PAL-kretsen

En rask skifting mellom signalene vil kunne skape en periode som var langt kortere enn 20 ms. Under testen ble servoen HS-81 som brukes i flyet, brukt. ATMega128 ble brukt til å generere signalet fra 'styresystemet' mens en mottager ble brukt til å generere signalet fra 'manuell styring'. Altså et nøyaktig likt oppsett som i CyberSwan. En bryter som genererte et styresignal ble koblet til riktig inngang på PALen.

Denne testen viste ingen tegn til at servoen hadde problemer med timingen. Testen viste også at periodetiden ikke var kritisk viktig for utslaget. Om periodetiden var 19,5 ms eller 20,5 ms hadde ingen synlig konsekvens på servohodet. Video av denne testen er vedlagt på CD.

8.3.1 Servobryter



Figur 8.6: Bryter for å styre mellom autonom og manuell styring (Denne bryteren ble fjernet og erstattet på grunn av problemer med PALen)

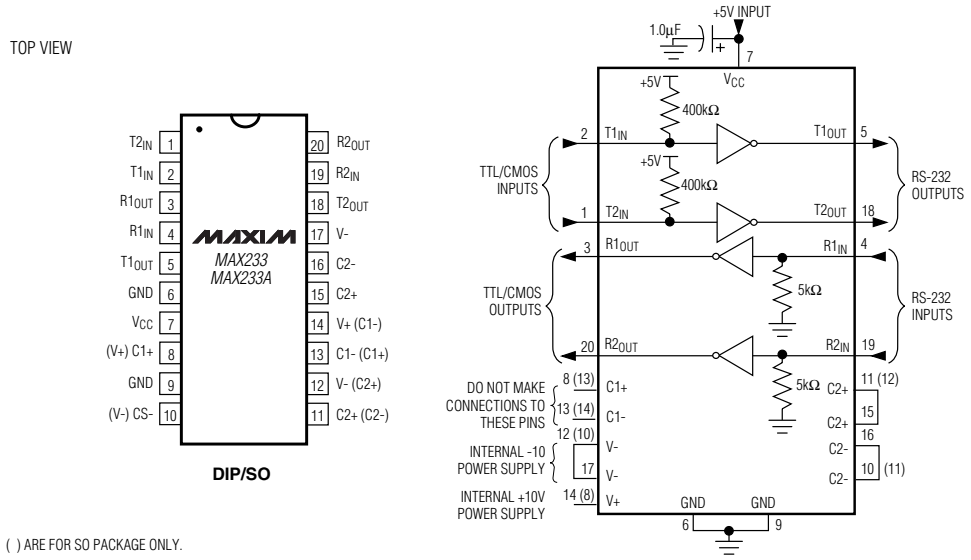
Det ble laget en bryterkrets som genererte styresignalet til PALen. Denne bryteren er avbildet i figur 8.6. Kretsen genererte et høyt eller lavt signal avhengig av om flyet skulle kjøres manuelt eller autonomt. En servo som kunne styres direkte fra radiosenderen på bakken kontrollerte denne bryteren. Operatøren kunne dermed skru av og på en bryter på radiosenderen nede på bakken og dermed velge om flyet skulle styres manuelt eller autonomt.

Denne bryteren fungerte fint, men siden det var en del problemer med PALen, ble PALen og denne bryteren skiftet ut med bryteren i figur 11.2.

8.4 MAX233

Max233 er en nivåomformer for RS-232 signaler. På CMOS-siden av MAX233 kan et signal på 0V-3,3V eller 0V-5V kobles til fra for eksempel ATMega128 sin UART-tilkobling. MAX233 omformer UART-signalet til nivåene i henhold til RS-232 standarden. RS-232-utgangen på MAX233 vil ifølge databladet til Maxim, ligge på ±8V under normal operasjon, men aldri lavere enn ±5V som er i henhold til spesifikasjonen EIA/TIA-232E og V.28. [8]

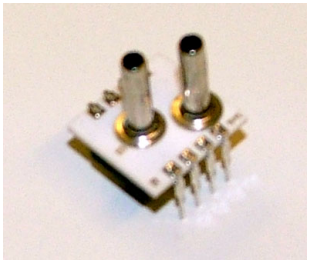
Det er flere andre MAX-kretser som kunne vært brukt, men MAX233 nøyer seg med 5V inngangsspenning og trenger ingen eksterne komponenter for å fungere. Den har to uavhengige kanaler. Disse kan derfor brukes til å lage RS-232 signaler av signalene fra



Figur 8.7: MAX233/MAX233A pinne oppsett og typisk funksjonskrets

GPSen og ATMega128. MAX233 var tilgjengelig på instituttets komponentlager da det var behov for den.

8.5 Trykksensor



Figur 8.8: Trykksensor beskrevet i avsnitt 2.3

Trykksensoren som vi ser i figur 8.8 har sin plass på IO-kortet, men den er ikke med på i figur 8.1 på grunn av at denne målingen ikke er fullstendig integrert i instrumenteringssystemet. Det er lagt opp til analog måling fra denne trykksensoren siden det ikke var tilgjengelig nok innganger på ATMega128 til å kunne bruke I^2C protokollen som denne trykksensoren støtter.

8.6 Kontakter for IMU og GPS

På bildet av IO-kortet i figur 8.1 ser vi at det er tilkobling til både IMU og GPS. Det er kun fire av de seks lederne som er i bruk på hver av kontaktene. Det er to signalledere til RS-232: RX og TX, samt 5V strømforsyning og jord. Det kan nevnes at GPSen bruker

0-3V nivå på RS-232 signalet mens IMUen bruker *true RS-232 levels*. Dette gjør at GPSen er koblet via MAX233 kretsen for å få *true RS-232 levels*.

Signalene fra IMU og GPS er å finne på kontakten som er merket *RS-232 PC/104* i figur 8.1.

8.7 Kontakt for 3 x RS-232 til PC/104

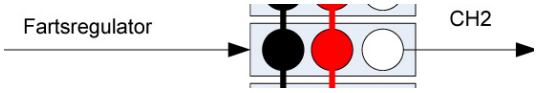
På bildet av IO-kortet i figur 8.1 er det en kontakt som er merket *RS-232 PC/104*. Denne består av 3 x RS-232. Dette er IMU, GPS og ATMega128. En skal være klar over at i forbindelse med kommunikasjon med PC/104 har begrepene 'ATMega128' og 'IO-kort' vært brukt om hverandre i denne rapporten. Men begge deler betyr det samme.

8.8 Kontakt for mottager

Mottageren som tar imot den manuelle styringen av flyet, kobles til kontakten som er merket Mottager på IO-kortet som vi ser i figur 8.1. Denne kontakten tar kun imot PWM-signalene fra mottageren, det må derfor kobles til en egen ledning med felles jord og strømforsyning til servoene på en av de to kontaktene på IO-kortet som sitter ved siden av tilkobling av servoene.

8.9 Tilkobling av seks servoer

- Servo til balanseror på venstre side
- Servo til balanseror på høyre side
- Servo til kombinert høyde/sideror (venstre side)
- Servo til kombinert høyde/sideror (høyre side)
- Fartsregulator til motor
- N/A



Figur 8.9: PWM-kontakt (Utsnitt av figur 8.2)

I figur 8.1 er det rekke med kontakter for tilkobling til servoer. Ledningene fra den enkelte servo og fartsregulator kobles til disse kontaktene.

8.9.1 Sort pinne (jord)

Den sorte pinnen i figur 8.9 er jord og er markert med en sort stripe på IO-kortet. Dette er felles jord med resten av IO-kortet. Alle de sorte pinnene på PWM-kontaktene er derfor

koblet sammen. Fartsregulator må ha felles jord med IO-kortet og mottager slik at PWM-signalene kan overføres med en felles referanse mot jord.

8.9.2 Rød pinne (strømforsyning)

Den røde pinnen i figur 8.9 er strømforsyningen til servoene. De røde pinnene er frakoblet fra resten av IO-kortet på grunn av at servoene trekker mye strøm. Fartsregulatorene skal derfor brukes som strømkilde til servoer og radiomottager slik som i et vanlig modellfly. Når fartsregulatorene er koblet til, vil derfor alle servoene samt mottageren få strøm fra fartsregulatoren og ikke IO-kortet. Stømforsyningen til IO-kortet blir derfor ikke belastet. Det er gunstig å gjøre det slik fordi strømforsyningen til instrumenteringssystemet da kan dimensjoneres uten å ta hensyn til at den skal levere strøm til servoer og mottager.

Motorens fartsregulator kan levere 2A til servoene. Dette er omtrent like mye som resten av instrumenteringssystemet kommer til å bruke. Det vil derfor være mye å spare på å la servoene trekke strøm fra fartsregulatoren. Se tabell 8.1.

Ulempen med å la servoene trekke strøm fra fartsregulatoren og dermed batteriene til fremdrift, er at flytiden reduseres. Servoene kan som nevnt trekke til sammen 2A fra fartsregulatoren, men dette vil ikke skje hele tiden. I hviletilstand trekker servoene lite eller omtrent ingen strøm. Trolig vil servoenes totale strømforbruk være minimal sett i forhold til motorens totale strømforbruk under flyvning. Det vil derfor være veldig liten reduksjon i flytid. Eneste konsekvens er:

- Kortere flytid

Alternativet med å la servoene trekke strøm fra instrumenteringssysteme ville introdusert følgende ulemper:

- Mindre predikterbar batteritid på instrumenteringssystemet på grunn av at strømtrekk hadde vært avhengig av hvor mye servoene beveger seg.
- Støy på IO-kortet.
- Overdimensjonering av spenningsregulatoren siden den måtte takle worst case scenario av strømtrekk fra servoene.

8.9.3 Hvit pinne (PWM-signal)

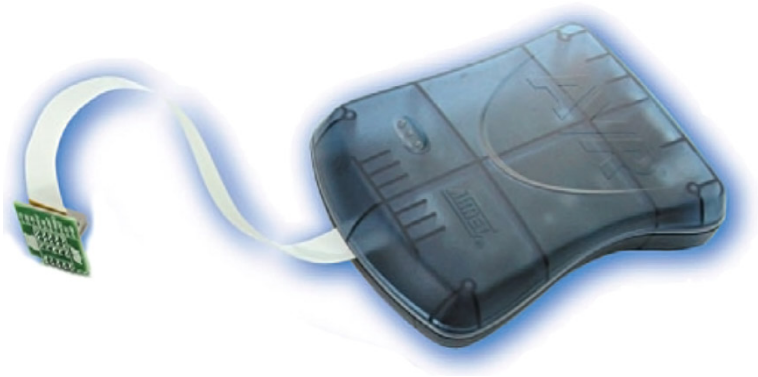
På de hvite pinnene ligger PWM-signalene til enhver tid, enten det går inn fra radiomottageren eller ut til servoene. På PWM-utgangene kommer signalet enten fra mottageren ved manuell styring eller fra ATmega128 ved autonom styring.

8.10 Analog inngang og inngang for eksternt avbrudd

Det er lagt til en analog inngang som er merket ADC1 på IO-kortet. Denne er koblet til en analog inngang på ATMega128 og kan brukes hvis det senere skal være behov for en analog måling.

Det er også lagt til mulighet for å legge inn et ekstra eksternt avbrudd på IO-kortet. Denne inngangen er merket INT7 og er koblet til inngangen som heter INT7 på ATmega128.

8.11 JTAG-kontakt for programmering av ATMega128



Figur 8.10: JTAG ICE mkII har USB tilkobling til PC

På IO-kortet er det en egen kontakt for tilkobling av JTAG. På IO-kortet brukes JTAG i bare til å programmere ATMega128, men med JTAG ICE mkII kan en også kjøre online debugging av programvaren på ATMega128.

8.12 Batteri og spenningsregulator

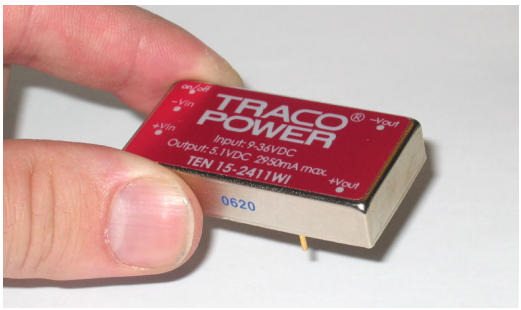
Enhet	*Produktnavn*	*Datablad*	*Maks forbruk*	*Spenning*
PC/104	ICOP-6070	0,68A(typ)/2A(maks)	2000mA (maks)	4,75V-5,25V
uC o.l. til IO	ATMega128	8MHz(active)- 19mW	50mA (maks)	2,7V - 5,5V
RF-modul	RC1240	26mA (maks)	30mA (maks)	2,8V - 5,5 V
GPS	EM-411	60mA (typ)	100mA (maks)	4,5V - 6,5V
IMU	Xsens MTi	350mW	200mA (maks)	4,5V - 15V
Trykksensor	SM5852	10 mA (maks)	20mA (maks)	4,75V-5,25V
Kamera		kobles utenom	0mA	(8V - 9V)
5-6 servoer		kobles utenom	0mA	(5V - 6V)
		Totalt	2400 mA (maks)	4,75V-5,25V

Tabell 8.1: Strømforbruk til alle komponentene i instrumenteringssystemet

Det er mange forskjellig spenningsregulatorer å velge mellom på markedet, på grunn av svært varierte behov i ulike applikasjoner. Behovene til instrumenteringen er ganske spesifikke i dette prosjektet:

- Kunne levere mer en 2400 mA (se tabell 8.1)
- Inngangspenning på 11,1 V (3 cellers LiPo-batteri) eller eventuelt 7,4V (2 cellers Lipo-batteri)
- Utgangspenning mellom 4,75V og 5,25V
- God effektivitet
- Lav vekt
- Ikke introdusere støy

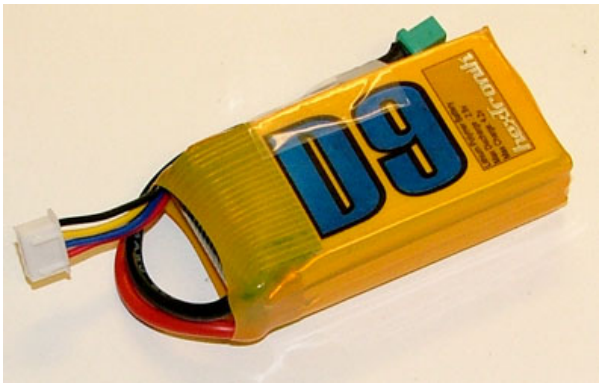
Siden effektivitet og fleksibilitet er viktig, ble det valgt en switcheregulatorer med mulighet for store variasjoner i inngangsspenningen. For å kunne dimensjonere spenningsregulatoren mest mulig presist, er det satt opp en oversikt over strømforbruket til de ulike enhetene i instrumenteringen i tabell 8.1.



Figur 8.11: Tracopower TEN-15WI serie

Tracopower har flere serier av gode switcheregulatorer som har intern støyfjerning og finnes i katalogen til Elfa. De fleste seriene har også metalldeksel som reduserer effekten av elektromagnetiske felter rundt regulatoren slik som vi ser av figur 8.11 at Tracopower TEN-15WI serien har. TEN-15WI er en serie med forskjellige utgang- og inngangsspenninger, men felles for alle i denne serien er at de kan levere 15 Watt og at de har samme fysisk form. Modellen TEN 15-2411 kan levere 2950mA @ 5,1V[3]. Denne kan da levere spenning til PC/104-enheten, ATMega128, RF-modulen, GPS, IMU og trykksensor. Servoene skal kobles utenom som beskrevet i avsnitt 8.9.

8.12.1 Batteri



Figur 8.12: Batteri til instrumenteringssystemet - HexTronik 1000mAh

Batteriene som brukes til motorfremdrift er 3-cellers LiPo-batterier[4] på 11,1V (3,7V per celle) med kapasitet på 1000mAh. Samme type batterier brukes også til styringsenhet slik at batteriene lettere kan brukes om hverandre. Spenningsregulatoren TEN 15-2411 godtar inngangsspenning mellom 9-36VDC.

[3] 2950mA @ 5,1V betyr 2950mA ved 5,1Volt
[4] Litium Polymer er spesielt lette batterier som er brukes i modellfly

Alle enhetene til sammen bruker maksimum 2400mA som vi ser av oversikten i tabell 8.1.

$$\frac{11,1V}{5,1V} \cdot \frac{1000mAh}{2400mA\ (maks)} \cdot 82\%\ effektivitet = 45minutter \tag{8.1}$$

Vi ser av ligning 8.1 at i verste fall vil kapasiteten være oppbrukt etter 45 minutter. Typisk forbruk på alle enhetene vil gi et mer realistisk anslag av det totale forbruket. Typisk forbruk er på ca 900mA totalt og da vil det være strøm nok til nesten to timers drift.

8.13 Design av IO-kortet med Eagle

Instituttet har etseutstyr som gjør det mulig å lage kretskort av PCB-tegninger. Bruker-veiledning for hvordan en etser et kretskort, ligger på instituttet sitt etserom.

Eagle er et program for tegning av PCB[5] print. Det er tilgjengelig gratis på Internett på www.cadsoft.de. Gratisversjonen har riktignok begrensning på størrelsen på kortet som kan lages (10x10cm), men dette var ikke noe problem under utvikling av dette IO-kortet. Eagle er i hovedsak delt inn i to forskjellige tegnebrett; *Schematic* og *Board*.

8.13.1 Schematic

Skjermbildet av IO-kortet i *Schematic* er vist i figur 8.13a. I *Schematic* utvikles alt det funksjonelle kortet består av uten tanke på fysisk plassering eller ruting av baner på kortet. Alle komponentene legges inn og baner trekkes mellom de ulike komponentene i *Schematic*. De fleste komponentene er som regel å finne i Eagle sitt interne bibliotek, som består av svært mange komponenter. Hvis enkelte komponenter ikke finnes, er det godt dokumentert i Eagle sitt hjelpesystem hvordan en legger til nye komponenter i biblioteket. De fysiske målene på en gitt komponent pleier å være beskrevet i databladet til komponenten. Det er derimot litt forskjellig fra datablad til datablad om tegningen er sett fra over- eller undersiden av komponenten.

8.13.2 Board

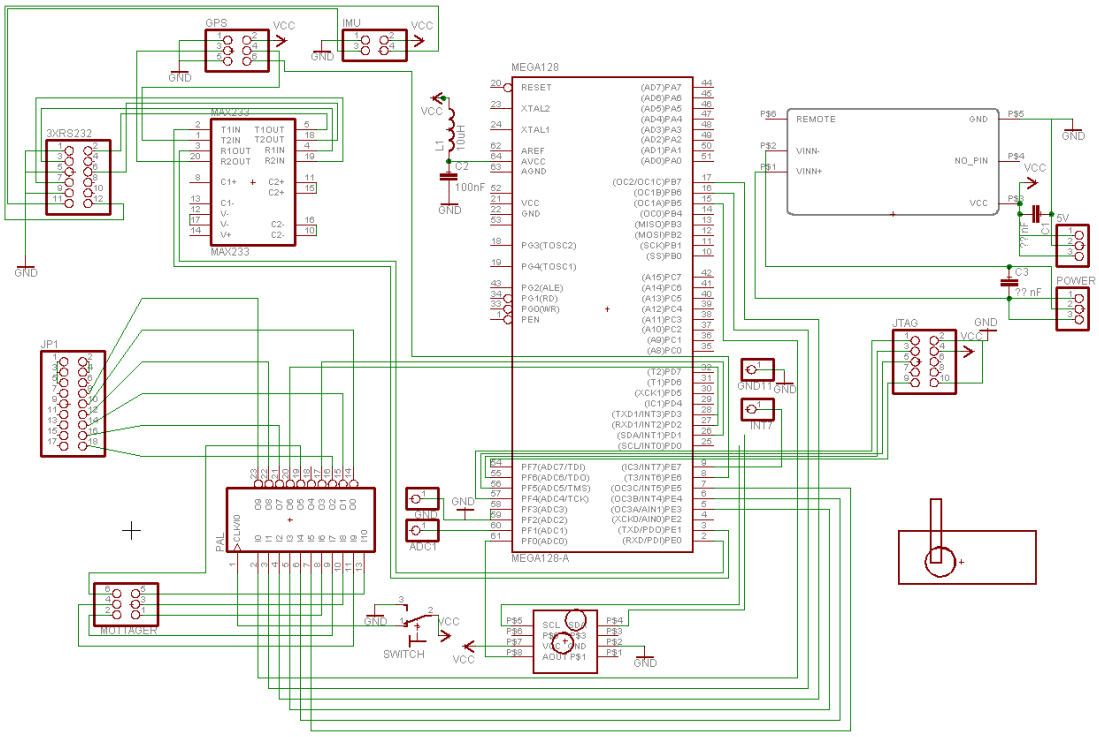
Skjermbildet av IO-kortet i *Board* er vist i figur 8.13b. I *Board* tegner en hvordan kortet fysisk skal se ut. Komponentene må plasseres og ruteres fornuftig slik at det blir færrest mulig via[6] å sette inn på kortet. Kontaktene må plasseres slik at det er hensiktsmessig å koble til eksterne enheter. Ruting av et tolagskort tar mye tid, men det lønner seg å gjøre det grundig slik at en slipper å gjøre feil og måtte etse et nytt kort eller gjøre hurtigloddinger for å rette opp i feil.

8.13.3 Tips

Det er viktig å huske på at det skal plasseres kontakter på headere på kortet. Hvis det skal brukes en latch på en flatkabel, bruker den litt ekstra plass på hver side av headeren, noe for eksempel ikke et berghus gjør. Berghus er forøvrig en del høyere enn en latch.

[5] PCB: Printed Circuit Board

[6] Via er små metallhylser som stikkes gjennom et tolagskort for å skape kontakt mellom de to sidene på kortet



(a) *Schematic*

(b) *Board* - Blå er underside, rød er overside

Figur 8.13: Skjermbilde av IO-kortet i Eagle

Det kan være lurt å tenke gjennom hvor tykke baner som trengs på kortet. Så lenge en bruker godt etseutstyr kan en bruke tynnere baner enn det som er brukt på dette kortet (se figur 8.13b og Eagletegningene på vedlagt CD). Tynne baner gjør det enklere og rute kretskortet, og en kan klare seg med færre via.

Det kan bli unødvendig tungvindt å plassere via hvis de er for små. På dette kortet er det brukt et 0,9mm bor til hullene og via med ytre diameter på 0,8mm. Det er greit å håndtere.

Det lønner seg å skrive ut kretskortet og fysisk kontrollere at komponentene passer. En får da sjekket om størrelsen på kortet er riktig og at komponentene passer før en begynner å etse. Spesielt viktig er det å gjøre dette med egne komponenter som er har lagt til i biblioteket.

8.14 Resultat og drøfting av IO-kortet

Resultatet av IO-kortet ble meget bra. Kortet gjorde det mulig å logge CyberSwan sin oppførsel samt å kunne fly autonomt med CyberSwan. Det ble derimot funnet noen svakheter med komponentene på kortet. Dette, sammen med mangler ved PC/104-kortet ICOP-6070, gjorde at funksjonaliteten i kortet ikke kunne utnyttes til det fulle.

Kortet ble revidert én gang for å legge opp baner for måling av mottageren på AMTega128. PALen ble da programmert om slik at dette skulle bli mulig. Spenningsregulatoren ble også redesignet i Eagle, siden den ble feildesignet slik at den havnet på undersiden av kortet i første revisjon.

ATMega128 fungerte meget godt, og JTAG for programmering av ATMega128 fungerte også veldig bra. En ulempe med IO-kortet var at det ikke var noen reset-knapp. Det hendte at det var behov for å resette IO-kortet etter oppstart hvis serieporten ikke ble initialisert riktig på PC/104 kortet. En nullstilling av IO-kortet løste dette problemet, men uten reset-knapp måtte dette gjøres med JTAG. Det fungerte såpass greit at det ikke ble laget en revisjon tre.

PAL-kretsen fungerte meget godt med PPM-mottageren i testoppsettet i avsnitt 7.7. Både måling av alle kanalene fra mottageren på ATMega128, bryterkretsen for å skifte signal til servoene og generering av PWM-signaler til servoene fra ATMega128 fungerte som forventet. Mottak og generering av PWM-signaler på ATMega128 er beskrevet i kapittel 7.

Som vi ser i avsnitt 7.5, viste det seg å bli kritisk å bytte mottager. PCM-mottageren som ble brukt i CyberSwan, var ikke tilgjengelig for testing da det var behov for den. Programvaren ble derfor utviklet for en annen mottager (avbildet i figur 7.3), med antagelser om at det kom til å være små endringer i programvaren ved bytte av mottager. Problemene som oppstod med PCM-mottageren i CyberSwan er beskrevet i 10.1 og førte til at PALen og bryterkretsen måtte byttes ut med bryteren i figur 11.2 for å kunne fly autonomt.

GPSen ble først koblet til den ledige UART-inngangen på ATMega128,men denne ble senere flyttet til inngangen på MAX233 på grunn av at det ville bli komplisert og uoversiktlig å både sende mottagermålingene og GPS meldinger på samme serielinje til PC/104.

MAX233-kretsen fungerte godt hele tiden, og den var grei å koble til.

Tilkoblingen til IMUen via kontakten 3 x RS-232 fungerte bra, men siden USB-omformeren fra xSens ble brukt i stedet for RS-232, var det enklere å ikke koble IMUen via IO-kortet.

Den kunne hente strømforsyning fra USB-porten i stedet. Dette gav mindre kabling i CyberSwan. Ved fortsettelse av prosjektet vil det være en bedre løsning å koble til IMUen via IO-kortet, men det forutsetter at serieportene fungerer som de skal på PC/104-kortet.

Spenningsforsyningen egnet seg godt til å forsyne IO-kortet, PC/104-kortet og resten av instrumenteringssystemet, men det skulle vise seg at batteriene var dårligere enn det som stod i spesifikasjonen. De hadde ikke mer enn 800mAh kapasitet og spenningen sank til 9 V før all kapasitet var brukt opp. Etter 1 time og 15 minutter på et slikt batteri stoppet instrumenteringssystemet på grunn av at spenningen på batteriet hadde sunket under 9 V. Den havnet altså under det som er minimumskravet til spenningsregulatoren. Oppetiden til systemet ble dermed dårligere enn antatt, men det var mer enn nok til å kunne teste CyberSwan. Ved å velge bedre batterier eller flere batterier kan kjøretiden på systemet økes.

Trykksensoren til pitotrøret ble aldri testet på IO-kortet, men det ble lagt opp analoge baner til ATMega128. Hvis det skal benyttes digital overføring via I^2C fra trykksensoren, må det enten settes på en ny mikrokontroller eller velges færre PWM-signaler, siden det ikke vil være nok innganger på ATMega128.

Kapittel 9

Programvareoversikt på ATMega128

ATMega128 gjør i hovedsak fire ting, så derfor består kildekoden av fire filer. Koden for å generere PWM-signalene til servoene ligger i `pwm.c`. Mottak av PWM-signaler gjøres ved at det trigges eksterne avbrudd, og kildekoden for dette ligger i `ext_int.c`. Kildekode for mottak og kommunisering over UART ligger i `uart.c`. Her er også servoprotokollen som definerer hvordan PC/104 enheten sender kommandoer fra styresystemet implementert. Denne servoprotokollen er beskrevet i avsnitt 6.5.

IO-kortet sender også meldinger tilbake til styresystemet ved 10 HZ. Dette er implementert i `main.c`.

Hensikten med dette kapittelet er å gi en oversikt over kildekoden på ATMega128. For mer spesifikke detaljer henvises det til vedlagte CD. Der er kildekoden detaljert kommentert linje for linje.

9.1 main.c

- void main(void)

main() kjører initialiseringsfunksjonene til UART, PWM og EXT_INT før den går inn i en løkke der den sender $MANUL-meldinger (avsnitt 6.4) til styresystemet med 10 Hz. main()-funksjonen henter verdienemottager fra globale variable som er definert i `ext_int.c` Disse vil ha verdien 0 helt til mottageren kobles til.

Løkken i main() kjører helt til mikrokontrolleren skrues av eller startes på nytt. main()-funksjonen har ikke noe med generering av servosignaler og gjøre. Servogenerering skjer kun ved hjelp av avbrudd. Meldingen som sendes til styresystemet er omtalt i avsnitt 6.4 men den ser slik ut: `$MANUL,ch0,ch1,ch2,ch3,ch4,ch5*CKSUM<CR><LF>`.

9.2 uart.c

- void USART_Init(void)
- int USART_write0(unsigned char c)
- unsigned char USART_read0(void)

- void reset_buffer(void)
- void add_data_to_buffer(char data)
- void print_buffer(void)
- void copy_buffer_to_pwm_channels(void)
- SIGNAL(SIG_USART0_RECV)
- SIGNAL(SIG_USART1_RECV)

I initialiseringsfunksjonen settes UARTen til å operere på 38400 bps med 8 data bit, 2 stop bit og ingen flytkontroll. Transmisjonen settes også til å være asynkron på enkel hastighet (dobbel hastighet er deaktivert). Funksjonen `printf()` kobles til `USART_write0()` slik at C-funksjonen `printf()` vil skrive data til serieporten. En kan dermed bruke `printf()` som en er vant til når en programmerer C, men i stedet for å skrive til konsollvinduet vil ATMega128 skrive ut data på serieporten.

9.3 pwm.c

- void pwm_init(void)
- void set_width_on_pwm_pulse(char channel, int value)
- SIGNAL(SIG_OVERFLOW1)
- SIGNAL(SIG_OVERFLOW3)

Initialiseringsfunksjonen setter i hovedsak opp Timer/counter1 og Timer/counter3 med Fast PWM mode.

Når funksjonen `set_width_on_pwm_pulse(char channel, int value)` kjøres, settes kun midlertidig variabler som er definert for hver kanal. Først når PWM-signalet går høyt, kopieres verdien som ligger i de midlertidig variablene inn i PWM-kontrolleren på ATMega128. Dette gjøres fordi da har ATMega128 minimum 900 μs før signalet går lavt igjen. Ved å gjøre det på denne måten, vil det helt sikkert ikke bli overlapp i skrivingen til den 16 bits verdien som definerer når hver kanal skal gå lav.

Det er også implementert et glattefilter som sørger for en mindre hakkete oppførsel for servoene. Denne er tilpasset styresystemet når styresystemet går på 10 Hz. Hvis denne frekvensen endres, bør også denne glattefunksjonen endres. `set_width_on_pwm_pulse(char channel, int value)` vil kjøres hver gang IO-kortet mottar en melding. Dette vil være ved samme frekvens som styresystemet, altså 10 Hz. Siden servoene sender PWM-pulser med 50 Hz, vil det sendes omtrent fem nye PWM-pulser før det kommer en ny melding fra styresystemet. I stedet for å endre pulsbredden momentant ved 10 Hz, bruker den disse 5 PWM-pulsene til gradvis å endre pulsbredden til den nye referansen. Dette gjøres vet at `set_width_on_pwm_pulse(char channel, int value)` regner ut et stigningstall den må endre PWM-verdien med. Så endrer den PWM-verdien med dette stigningstallet maks fem ganger. Etter fem ganger vil den få en ny verdi fra styresystemet. Det ble merkbart bedre oppførsel på servoen etter denne glattingen.

9.4 ext_int.c

- void interrupt_init(void)
- unsigned char calculate_channel_value(
  unsigned int last_channel_value. unsigned int channel_value)
- void start_channels(void)
- void store_channel(void)
- unsigned int TIM16_ReadTCNT1(void)
- SIGNAL(INT0_vect)
- SIGNAL(INT1_vect)
- SIGNAL(INT2_vect)
- SIGNAL(INT3_vect)

Når ATMega128 mottar PWM-signaler, bruker den de samme tellerne som er satt opp i `pwm.c`. Når et avbrudd inntreffer, lagres tiden når avbruddet inntraff ved hjelp av den ene telleren som brukes til PWM-generering.

Initialiseringsfunksjonen setter opp eksternt avbrudd 1 (`INT1`) og eksternt avbrudd 2 (`INT2`) til å trigge på fallende flanke og eksternt avbrudd 3 (`INT3`) til å trigge på reisende flanke, slik som beskrevet i 7.5.

`calculate_channel_value()` beregner hvor stort utslag en servo har ut i fra to tidskoder som den får inn. Forskjellen i disse tidskodene er gitt av servosignalspesifikasjonen og vil ligge mellom 900 ms og 2100 ms. Returverdien er på en byte har dermed verdier mellom 0 og 255.

`start_channels()` kjøres av eksternt avbrudd 1 som trigges 50 ganger per sekund av mottageren, og funksjonen initialiserer opptakssekvensen av seks kanaler.

`store_channel()` kjøres av eksternt avbrudd 1 og eksternt avbrudd 2 som trigges til sammen 50x6 ganger per sekund. Funksjonen lagrer tidskoder for seks kanaler fortløpende. Når den har mottatt siste kanal, beregner den utsalget til alle kanalene med funksjonen `calculate_channel_value()`.

`TIM16_ReadTCNT1( void )` er en funksjon med atomisk avlesning av en integer på to byte. Den er hentet fra databladet til ATMega128. Hemmeligheten med denne funksjonen er at den kobler ut alle avbrudd for å unngå avbrudd mellom første og siste byte.

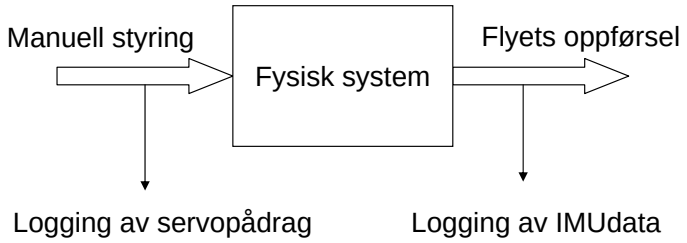
`SIGNAL()`-funksjonene er avbruddsrutinene til hvert enkelt avbrudd. Det er tre avbruddsrutiner knyttet til opptak av PWM-kanalene fra mottakeren. Disse tre er eksterne avbrudd som er koblet til hvert sitt pinne på ATMega128 og kjører `start_channels()` og `store_channels()` som definert i 7.5. Se figur 7.2 for oversikt over når disse trigges.

`SIGNAL(INT7)` er avbruddsrutinen til den eksterne avbruddspinnen på IO-kortet som er merket `INT7` og beskrevet i avsnitt 8.10. Det er ikke laget noen avbruddshåndtering for avbrudd på denne pinnen. Dette kan derfor legges til i etterkant hvis det skal være behov for det.

Kapittel 10

Flyvning 1 - logging av data

Målet med første flyvning var å logge flyets oppførsel. Når man vet hva som går inn i et system og måler hva som kommer ut av systemet, kan man lage en modell av systemet. Med en modell av flyet, kan styresystemet simuleres og en kan da verifisere at regulatorene fungerer som forventet. Under første flyvning ble det derfor loggført hvilke utslag som ble gitt til servoene sammen med målingene fra IMUen. Disse dataene ble brukt til å lage en modell av flyet og i neste rekke et styresystem for flyet. Dette er beskrevet i [6]. Dette kapittelet tar for seg de tilpasningene som ble gjort i instrumenteringssystemet for å få logget de nødvendige dataene. Det var spesielt nødvendig med en del tilpasninger på IO-kortet på grunn av forhold ved PALen og PCM-mottageren, men styresystemet ble også tilpasset for å få mest mulig optimal og sikker logging av dataene.

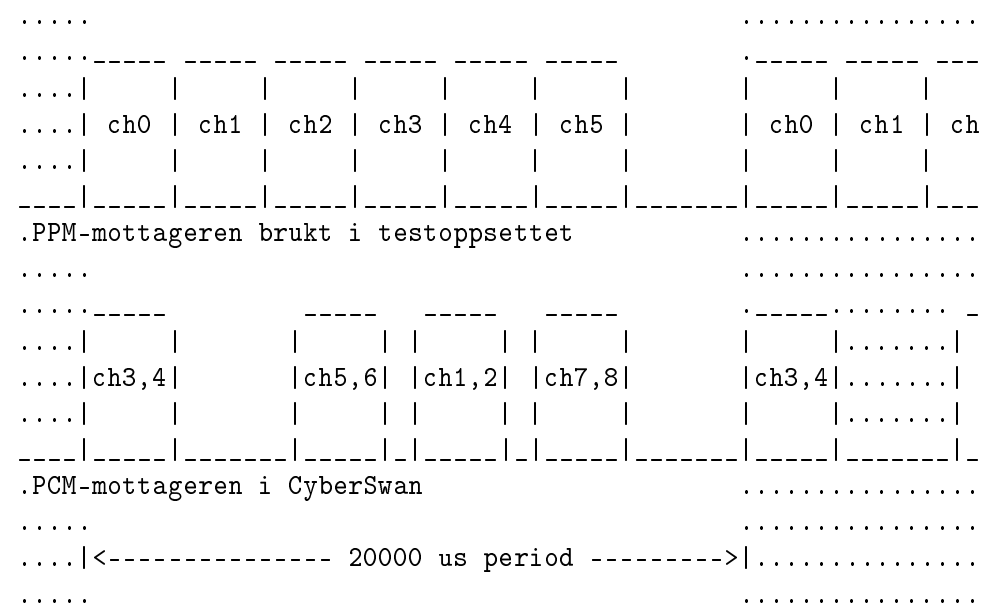


Figur 10.1: Logging av inngang og utgang

Kapitlet beskriver først tilpasninger som ble gjort til PCM-mottageren, etterfulgt av en beskrivelse av loggesystemet som ble brukt. Oppstarten og bruken av programmet Screen over SSH er også beskrevet. Til slutt er det presentert resultatene av flyvningen.

10.1 Målinger av servopådrag fra mottager

IO-kortet var tilpasset en annen mottager enn den som ble brukt under første flyvning. Forskjellen på PCM-mottageren som ble brukt i CyberSwan og PPM-mottageren som ble brukt i testoppsettet i avsnitt 7.5, var at kanalene ikke var faseforsøvet likt i de to mottagerne. I figur 10.2 er denne faseforskyvningen illustrert. Som vi ser av figur 10.2 er to og to kanaler faseforsøvet likt på PCM-mottageren som ble brukt i CyberSwan. Dette gjør at det trengs en mer avansert form for måling med to baner tilgjengelig til ATMega128



Figur 10.2: Forskjellen i faseforskyvning på kanalene på PCM- og PPM-mottageren

enn med PPM-mottageren. Med tre baner kunne det vært gjort på tilsvarende måte som beskrevet i 7.5. For å gjøre utviklingen enklest mulig, ble det derfor målt bare fem kanaler i stedet for seks. Men det var ikke behov for måling av flere enn fem kanaler så dette var ikke noe problem. De fem kanalene ser vi i figur 10.3, hvor de to banene er målt hver for seg med et combiscop.

Figur 10.3 viser signalet på de to banene som føres inn til ATMega128, over en periode på 20 ms. På den ene banen ligger bare kanal 3, mens kanal 4, 6, 2 og 7 ligger på den andre banen. For å få til dette måtte PALen programmeres om slik at riktig kanaler ble ført inn på riktig bane til ATMega128. Følgende linjer ble brukt:

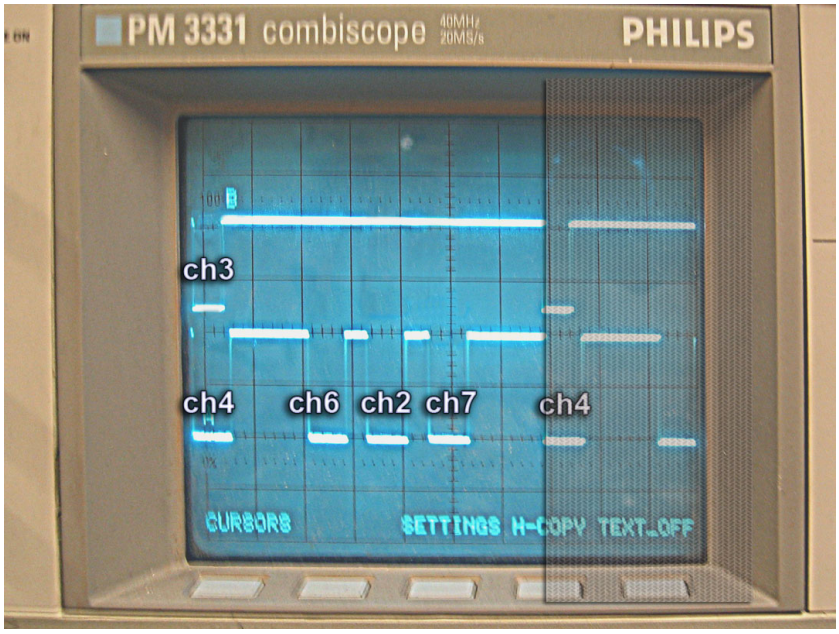
```
channel_even  <=  not (ch3);
channel_odd   <=  not (ch4 or ch6 or ch2 or ch7);
```

Vi ser at det ble valgt ut fem kanaler som ble målt slik som vist i figur 10.3. Da kunne den ene banen nullstille perioden på negativ flanke og starte en ny opptakssekvens. Så ble det laget avbruddsrutiner slik at fallende flanke startet en ny kanal, og stigende flanke lagret gjeldene kanal. Dette ble betydelig enklere og mer feilsikkert enn å legge flere kanaler også på den øverste banen.

Programfil (.hex) og kildekoden som ble brukt på IO-kortet under første flyvning ligger vedlagt på CD.

10.2 PALen lagde støy på servosignalet

Faseforskyvningen på PCM-mottageren skapte flere problemer enn måling av kanalene. PAL-kretsen introduserte støy på servoene da den nye PCM-mottageren ble koblet til. Det var lenge et mysterium hvor denne støyen kom fra. Det skulle vise seg at isolasjonen mellom kanalene i PALen ikke var god nok. Dette var ikke noe problem med PPM-mottageren i testoppsettet ettersom det da ikke var kanaler som lå høye samtidig.



Figur 10.3: Avbrudd på multiplekset PWM-kanaler fra PCM-mottageren i CyberSwan

Som vi ser av combiscopet i figur 10.3, starter kanal 3 og kanal 4 samtidig. Det gjør også kanal 1 og 2 (illustrasjon i figur 10.2). Kanal 1 ut fra PALen er vist i figur 10.4a hvor det også er markert kanal 2 på inngangen av PALen. I figur 10.4b er det vist hva slags signal servoen tror den får fra PALen. Servoen tror altså ikke at den får et signal med støy i dette tilfelle. I figur 10.5 er det vist et annet tilfelle. Her tror servoen at den får et en ekstra puls. Den får da et signal som har to pulser i hver periode og servohodet beveger seg helt ukontrollert.

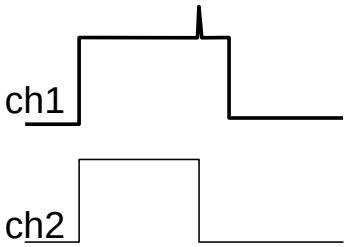
Fallende flanke på kanal 2 lager altså en støypuls på kanal 1 på grunn av at det var for dårlig isolering mellom de ulike kanalene i PALen. Dette gikk riktignok fint så lenge kanal 1 hadde lengre pulsbredde enn kanal 2. Når kanal 2 har lengre pulsbredde enn kanal 1 ser servoen en ekstra puls og servoen er ikke mulig å kontrollere.

10.2.1 Løsning på støyen i PALen

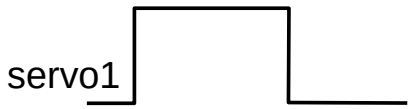
Støypulsen mellom kanalene i PALen kan mest sannsynlig fjernes med en kondensator på hver kanal. Det er ikke gjort beregninger på hvor stor kondensatoren må være, men den må være en del mindre enn 100nF. En test med en 100nF avkoblingskondensator viste at PWM-pulsen blir betydelig avrundet. Et så avrundet PWM-signal gir for dårlig signal til servoen slik at servoene blir stående å vibrere siden flanken ikke blir klart nok definert. En kondensator på mindre enn 1 pF kunne kanskje fungert forutsatt at PALen godtar kondensatorer på utgangene.

Siden vi ikke skulle fly autonomt denne første flyturen, hadde støyen på utgangen lite å si, så disse kondensatorene bli ikke satt inn på kortet.

Det var to PCM-mottagere tilgjengelig. Det ble derfor satt inn en egen mottager som instrumenteringssystemet kunne måle signalene fra som var helt dekoblet fra servoene. CyberSwan ble altså fløyet med et vanlig modellflyoppsett bestående av en mottager som

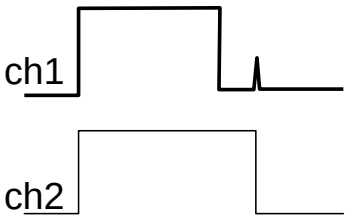


(a) Signal fra utgang 1 på PALen, med plot av hvordan kanal 2 innvirker på signalet

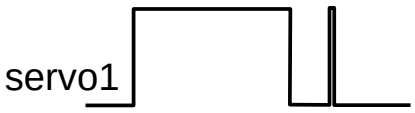


(b) Signal som servo 1 mottar er støyfritt

Figur 10.4: Når kanal 1 har mindre utslag enn kanal 2 fungerer servoen fint



(a) Signal fra utgang 1 på PALen, med plot av hvordan kanal 2 innvirker på signalet



(b) Servo 1 oppfatter støyen som en ekstra puls. Resultatet er ekstreme utslag på servoen

Figur 10.5: Når kanal 1 har større utslag enn kanal 2, fungerer servoen ikke siden PWM-signalet forvrenges av kanal 2

sendte signaler til servoer og fartsregulator. I tillegg var det en ekstra PCM-mottager som tok imot de samme signalene som servoene fikk, denne PCM-mottageren var koblet til IO-kortet.

10.3 Loggesystem på PC/104 under Xubuntu

Det var problemer med å få Simulinkmodellen som er beskrevet i kapittel 5 til fungere stabilt før første flyvning. Hovedproblemet lå i at det ble generert `Alarm Clock` etter variabel tid etter at systemet var startet. Når en `Alarm Clock` inntraff, terminerte programmet. Det var mangelfull dokumentasjon på `Alarm Clock`, og det var liten tid til å feilsøke på dette problemet. Det var derfor en enklere og sikrere løsning å plukke ut de nødvendige kodelinjene fra S-funksjonene til IMU og mottager og lage selvstendige programmer av dem. Dette var ikke noen dårligere løsning bortsett fra at det måtte startes to programmer og ikke ett.

Et program med utgangspunkt i S-funksjonen fra IMUen ble derfor satt sammen til et eget lite, kjørbart program. Dette programmet skrev IMU-meldinger til `auav_logg_first_run.txt`. Et annet program med utgangspunkt i S-funksjonen for mottageren ble også laget. Dette programmet lyttet på meldinger fra IO-kortet og skrev meldinger som ble mottatt til samme tekstfil; `auav_logg_first_run.txt`.

De to programmene som heter `receiver_logging` og `imu_logging`, ble kjørt samtidig i hver sin terminal som beskrevet i avsnitt 10.4.2.

I etterkant ble alle dataene i `auav_logg_first_run.txt` importert og analysert i Matlab. På CDen er det vedlagt en fil som heter *Første flyvning - logging.xls* hvor alle dataene er strukturert slik at det var mulig å analysere flyets oppførsel.

10.4 Oppstart av CyberSwan

Når batteriet kobles i, starter IO-kortet og Xubuntu på PC/104 samtidig. ATMega128 starter umiddelbart og begynner å sende data på serieporten. Xubuntu bruker et par minutter på oppstarten, men etter fullført oppstart, kan en logge seg inn på Xubuntu med en bærbar PC via SSH og starte styresystemet.

Grunnen til at det ble brukt en bærbar PC til å starte systemet, er at en på den måten kan verifisere at systemet starter som det skal. Programmene kunne vært lagt inn sammen med oppstarten av Xubuntu, men da hadde en hadde hatt mindre kontroll på at systemene startet som forventet.

10.4.1 Fast IP-adresse på CyberSwan og bærbar PC

Xubuntu setter IP-adressen sin til 192.168.0.10 i oppstarten. Dette gjør den siden følgende linje er lagt til i filen `/etc/rc.local`:

```
ifconfig eth0 192.168.0.10 netmask 255.255.255.0
```

Alle kommandoer i `/etc/rc.local` kjøres som root i slutten av oppstarten av Xubuntu. Ved å kommentere ut linja over vil Xubuntu motta sin IP-adresse via DHCP. Det kan

nevnes at en vil trolig kunne få en raskere oppstart ved å deaktivere konfigureringen av nettverket i oppstarten siden Linux vil få en *timeout* på DHCP forespørsel før den går videre.

På den bærbare PCen som skal kobles til, må det også defineres en tilsvarende IP-adresse for å kunne koble seg til CyberSwan. Følgende kommando ble brukt til dette:

```
sudo ifconfig eth0 192.168.0.11 netmask 255.255.255.0
```

Med en krysset nettverkskabel kan en da koble seg til CyberSwan via SSH.

10.4.2 Virtuelle terminaler med programmet `Screen`

Screen er et program som setter opp virtuelle terminaler i Linux. Ved å bruke screen kan oppstarten av CyberSwan gjøres via SSH. På den måten trengs ikke skjerm, og tastatur til oppstarten, men kun en bærbar PC med nettverk og SSH-klient. Et annet alternativ hadde vært å kjøre oppstarten over serieporten, men siden det var problemer med serieportene på PC/104 kortet var Screen over SSH en enklere å bedre løsning.

Etter innlogging opprettes alle virtuelle terminalene som trengs med screen. Hvert sitt program kan da startes opp og kjøres i sin egen terminal. En kan så koble seg fra disse terminalene mens de fortsetter og kjøre. Før nettverkskabelen dras ut, bør SSH kobles ifra, selv om det strengt tatt ikke nødvendig så lenge alle terminalene er *detached*.

Innlogging via SSH:

```
ssh knutedga@192.168.0.10
password:
```

Før flyvning startes en virtuell terminal på PCen i flyet via SSH med kommandoen:

```
screen
```

En liste over virtuelle terminaler som er startet, vises men kommandoen:

```
screen -ls
```

Etter landing kan en eksisterende terminal med PID=`2329.0-xubuntu` hentes tilbake igjen men kommandoen:

```
screen -d -r 2329.0-xubuntu
```

En vilkårlig terminal kan hentes tilbake igjen med kommandoen:

```
screen -D -RR
```

En virtuell terminal avsluttes med kommandoen:

```
exit
```

`Screen` installeres med følgende kommando:

```
sudo apt-get install screen
```

`Screen` er godt dokumentert, og hurtigtastene for å kontrollere terminalene som er startet er logisk. Det er vedlagt en brukerveiledning til `Screen` på CD.

10.5 Nedstengning av systemet

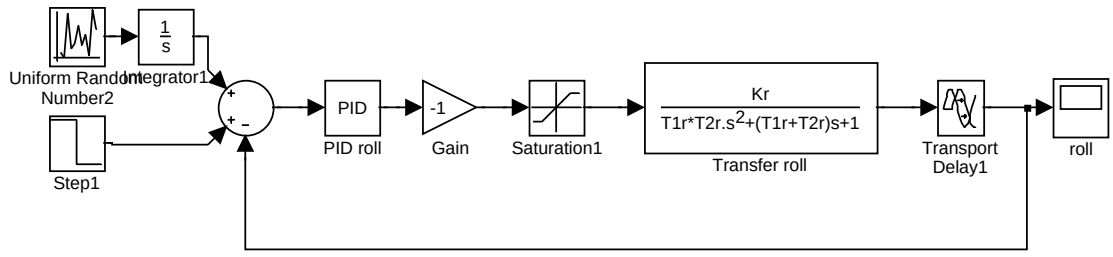
Nedstegning av systemet kan enten gjøres brutalt ved å koble fra batteriet, eller bedre ved å logge seg inn i via SSH, avslutte de to programmene, avslutte de virtuelle terminalene, kopiere over loggfila og kjøre kommandoen `sudo shutdown -h now`. SSH-serveren er noe av det første som avsluttes, derfor mister en kommunikasjonene med CyberSwan ganske raskt. Men siden Xubuntu bruker ti til tjue sekunder på å avslutte helt, bør en derfor vente en liten stund før batteriet kobles fra.

10.6 Resultater av første flyvning

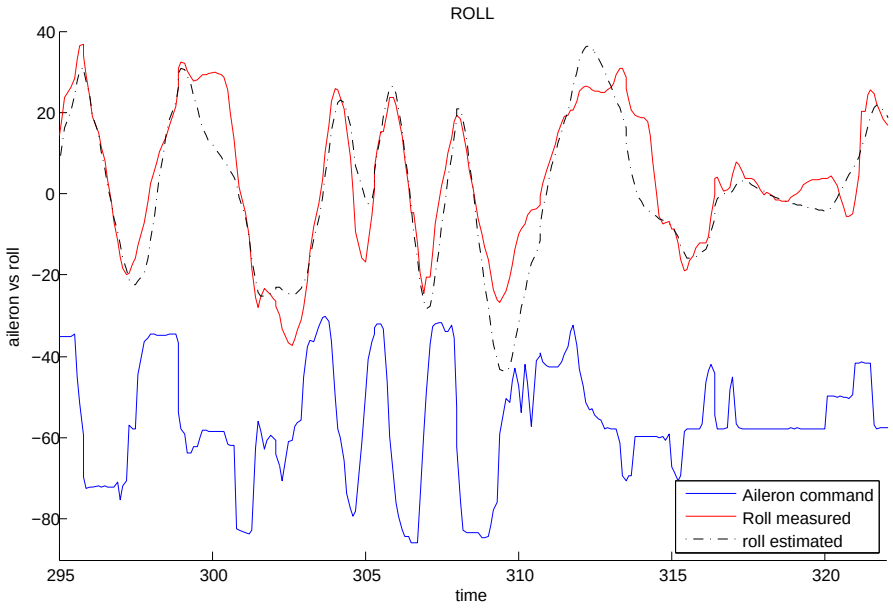
Loggfila fra første flytur ligger vedlagt på CD. Ut ifra loggfila kan det se ut som det er noe støy på målingene fra mottageren. Og enkelte kanaler har mer støy enn andre. Det er litt usikkert hva støyen kommer av, men mest sannsynlig har det å gjøre med at det i enkelte tilfeller genereres ekstra avbrudd i signalene som sendes til ATMega128 (signalet er vist i figur 10.3). Det var likevel ikke mer støy enn at målingene kunne brukes til å lage en god modell av flyet. Modellen kan estimere roll ganske bra som vi ser i figur 10.7.

Analyseringen av dataene fra flyturen er ikke en del av denne oppgaven, men det er beskrevet i [6]. Figurene 10.6 og 10.7 er lagt med for å illustrere hva dataene ble brukt til. Ved å implementere PID-regulatoren i figur 10.6 kunne CyberSwan regulere *roll* under den autonome flyvingen. Tilsvarende regulatorer ble utviklet for *pitch* og *yaw* også.

Figur 10.7 viser at målingene gjorde det mulig å lage et godt estimat av flyets oppførsel, noe som er viktig for å kunne lage et godt styresystem.



Figur 10.6: Reguleringssløyfe for roll med en andre ordens modell av CyberSwan i Simulink (figuren er hentet fra [6])



Figur 10.7: Her er roll estimert ut i fra en andre ordens modell av CyberSwan. Rollen er plottet sammen med målingene av roll og pådrag til balanserorene under første flyvning (figuren er hentet fra [6])

Kapittel 11

Flyvning 2 - Autonom styring

Figur 11.1: Klargjøring for autonom flyvning av CyberSwan

Med bakgrunn i dataene som ble samlet under første flyvning ble det laget reulatorer til CyberSwan. Disse regulatorene ble satt sammen til et styresystem som ble testet under autonom flyvning. Det var laget regulatorer for høyde- og sideror samt for balanserorene, men ikke for hastighet. Den autonome flyvningen var altså ikke fullstendig autonom siden hastigheten ble justert manuelt.

Dette kapitlet beskriver hvilke endringer som måtte gjøres med eksisterende plattform før det var mulig å fly autonomt, samt erfaringer og resultater av flyvningen.

kapitlet tar først kort for seg styresystemet som ble brukt, etterfulgt av en beskrivelse av de tilpasninger som ble utført i instrumenteringssystemet før flyvningen. Til slutt er logging

av data samt oppsummering og resultatene fra flyvningen beskrevet.

11.1 CyberSwan sitt styresystem under Xubuntu på PC/104

Loggesystemet som er omtalt i avsnitt 10.3 fungerte godt under første flyvning. Systemet ble derfor brukt videre som utgangspunkt til CyberSwan sitt styresystem sammen med kildekode fra S-funksjonen som setter ut pådrag til servoene (avsnitt 5.5).

Styresystemet som ble laget for autonom styring av CyberSwan ligger vedlagt på CD sammen med kildekode for programmet. Programmet heter `controlsys_imu-in_servo-out`. Som filnavnet tilsier, ble CyberSwan styrt med målingene fra IMUen. Når dette styresystemet får inn målinger fra IMUen, genereres pådragene av reguleringsalgoritmen før pådragene sendes til IO-kortet som lager PWM-signaler til servoene. Dette skjer med en frekvens på 10Hz og styres i hovedsak av IMUen. Reguleringsalgoritmene som ble benyttet er beskrevet i detalj i [6].

11.2 PAL koblet ut

Som nevnt i avsnitt 10.2, lagde PALen støy på servosignalet når PCM-mottageren i CyberSwan ble brukt. Under den autonome styringen var det ikke like kritisk å måle signalet fra mottageren, derfor ble PALen fjernet helt og byttet ut med bryteren i figur 11.2.

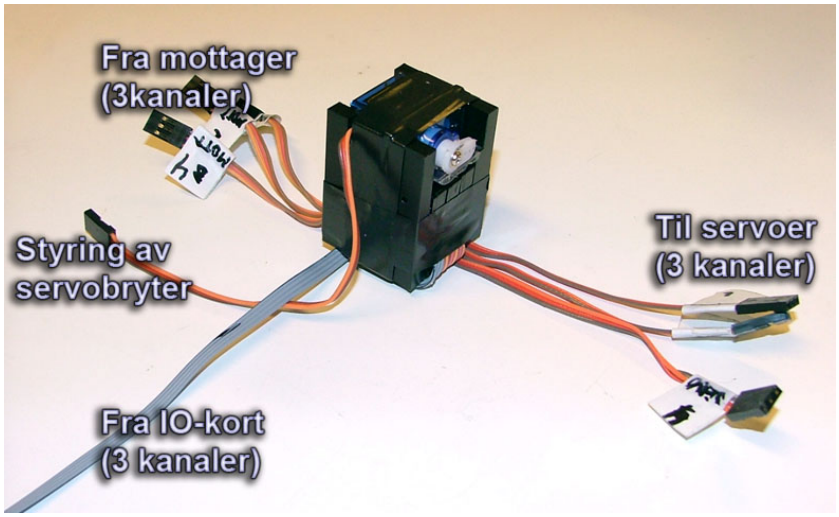
Utskifting av PALen gav noen uheldige begrensninger. Uten PALen var det ikke mulig å måle servosignalene fra mottageren under manuell styring. Dette skyldtes at PALen multiplekset PWM-signalene til ATMega128. Uten PALen hadde det bare vært mulig å måle to PWM-signaler siden det bare var to baner til ATMega128. Men måling av bare to PWM-signaler hadde gitt lite informasjon om den manuelle styringen, og dette ble derfor ikke gjort.

Selv om manuell styring ikke var kritisk for selve styringen, kunne en med slike målinger likevel ha endret referansene på regulatorene underveis. For eksempel kunne en kanal ha vært brukt til å trimme høyderorreferansen til flyet. Denne muligheten mistet man ved å fjerne PALen.

11.3 Bryter lagt til

PALen sin oppgave var å styre hvor servoene fikk sin PWM-kilde fra. For å få samme funksjonalitet uten PALen, ble det laget en bryter bestående av tre brytere som hadde to innganger og en utgang hver. Denne bryteren kunne dermed ta inn tre PWM-signaler fra ATMega128 og tre PWM-signaler fra mottageren. Fire servoer var koblet til utgangene på denne bryteren. De tre PWM-signalene var fordelt på servoene som følger: De to servoene på haleroret hadde hvert sitt PWM-signal, mens balanserorene fikk samme PWM-signal. Dette var mulig siden balanserorene i CyberSwan går hver sin vei.

Det var en servo som kontrollerte om de tre bryterne skulle være i autonom eller manuell modus. Denne servoen ble styrt med en bryter på radiosenderen på bakken. Servoen fun-



Figur 11.2: Denne bryteren erstattet PALen og bryterkretsen på IO-kortet

gerer altså på samme måte som servoen i bryterkretsen som stod på IO-kortet (avbildet i figur 8.6).

I sokkelen hvor PALen hadde stått, ble de stukket ledninger som koblet rett inngang med rett utgang.

11.4 Logging av måling og pådrag

Alle data som ble mottatt fra IMUen er logget med 10 Hz. Alle pådragene som ble sendt til servoen er også logget. Det er dermed mulig å sammenligne disse dataene med dataene som ble loggført under første flyvning. Men siden det ikke kommer frem nøyaktig når det er fløyet autonomt kan det være vanskelig å skille når flyet er fløyet autonomt og når det er fløyet manuelt. For å finne ut når flyet er fløyet autonomt kan man enten generere en modell av systemet med *Identification Toolbox* som beskrevet i [6] og se hvor modellen stemmer med dataene, eller bruke videoen aktivt til å analysere særegenheter. Den beste løsningen er trolig en kominasjon. Loggfila ligger vedlagt på CD.

11.5 Oppsummering og resultater av autonom flyving

CyberSwan ble startet på samme måte som beskrevet i kapittel 10, ved at en bærbar PC ble benyttet til å starte styresystemet med Screen via SSH.

Det ble på forhånd kompilert opp en rekke programmer med ulike reguleringsparametere. Alle disse er vedlagt på CD. En liten ulempe var at det ikke var mulig å kompilere programmer mens vi var ute og fløy, ettersom det ikke var lagt inn g++ kompilator på den bærbare PCen.

CyberSwan fløy autonomt i 10 til 15 sekunder av gangen med instrumenteringssystemet som er beskrevet. Hovedårsaken til at flyet ikke kunne fly autonomt over lengre tid var at det var for liten tid til å lage og teste et reguleringssystem som kunne svinge og følge en rute over tid. Flytiden på 10 til 15 sekunder var gitt ved at dette var tiden det tok

CyberSwan å fly fra den ene til den siden av tilgjengelig flyområde. Resultatet viser likevel at instrumenteringsplattformen fungerer og at den danner et godt utgangspunkt for videre arbeid med CyberSwan, og at det skal være grunnlag for å kunne fly autonomt over lengre tid.

Flyvingen ble filmet med kamera på bakken og med et trådløst kamera oppe i flyet. Denne videoen hvor det også er lagt med forklaringer underveis er vedlagt på CD.

Kapittel 12

Diskusjon

Gjennom arbeidet med CyberSwan sitt instrumenteringssystem er det gjort mange erfaringer som er viktig å ta med seg videre ved en eventuell fortsettelse av prosjektet. Dette kapittelet gir en oversikt over styrkene og svakhetene ved de mest sentrale valgene som er gjort i designet av systemet. Forslag til forbedringer er også presentert og begrunnet i dette kapittelet, men nærmere konkretisert i kapittel 14.

12.1 PC/104 med Linux på Compact Flash

CyberSwan skal være en generell plattform. Det ble derfor valgt en instrumenteringsplattform basert på PC/104 for at det skulle være enkelt å koble til eksterne enheter, og for at det skulle kunne legges inn et operativsystem og annen eksisterende programvare. Sammenlignet med alternativet AVR32 gjorde PC/104 det enklere å begrense omfanget på denne delen av systemet. Integreringen av AVR32 med Linux er heller ikke så god som på en PC/104. På lang sikt kan imidlertid AVR32 være en god løsning siden den har betydelig lavere strømforbruk og tar en del mindre plass. I [12] er det beskrevet integrering av Real-Time Workshop med AVR32.

Det ble valgt å bruke et Compact Flash-kort som lagringsenhet til PC/104-enheten. Det var noen problemer med Compact Flash-adapteren, men når denne ble byttet ut, var dette en god løsning med lav vekt og 2.0 GB lagringskapasitet.

I et prosjekt som CyberSwan der prosjektet skal gå i arv og fullføres av andre er det viktig at hver byggesten er så selvstendig så mulig slik at en ikke bør bruke tid på deler som allerede fungerer. Den PC/104-løsningen som er skaffet til dette prosjektet fungerer godt og bør brukes videre i et oppfølgingsprosjekt. Etter installasjon av Linux foregikk all testing og bruk av PC/104 via SSH over Ethernet. Kommunikasjon, filoverføring, installasjon av programmer på PC/104, testing av styresystem og endring av Linuxoppsett ble gjort over SSH fra arbeidsstasjonen.

12.2 Linux på arbeidsstasjon og i CyberSwan

Linux ble brukt både til utvikling av programvare på arbeidsstasjonen og i CyberSwan på PC/104. *Kubuntu* ble brukt på arbeidsstasjonen og *Xubuntu server* på PC/104. Begge

disse distribusjonene bygger på Ubuntu 6 serien som har en Linux 2.6 kjerne. Det lønner seg å ha så like systemer som mulig slik at programvaren som kjører på arbeidsstasjonen også kjører på PC/104. Dette sparer mye tid under utviklingen.

Ubuntuserien er i løpet av de siste årene blitt meget populær. Erfaring viser at andel programmer som er tilgjengelig gjennom pakkesystemet for Debian og Ubuntu er meget stor og raskt økende. Nytten av å ha nettverkskort for å kunne legge inn nødvendige programvare med pakkebehandleren bør forøvrig ikke undervurderes. Dette gjelder først og fremst på arbeidsstasjonen, men i noen grad også på PC/104.

DeLi Linux 0.7.2 ble testet og fungerte tilsynelatende greit, men DeLi Linux bygger på Linux 2.4-kjernen. Dette introduserer unødvendig problemer ettersom samtlige programmer må kunne kjøre under Linux 2.4. Siden Linux 2.6 kjernen kom første gang i des 2003 er mye programvare utviklet for Linux 2.6. Argumentene for å bruke DeLi Linux var at DeLi Linux har lave krav til maskinvareresurser, men når styresystemet kjørte med under Xubuntu 6.10 server, var den totale CPU bruken på maksimalt 5-10

RedHat Linux er et annet alternativ som fortsatt kanskje har best støtte i 'embedded-verden', men PC/104-løsningen i CyberSwan som har 2.0GB lagringskapasitet ligner mer på en standard Linux server enn en embedded Linux. Fordelene med en embedded Linux vil trolig først gjøre seg gjeldene hvis en velger AVR32 i stedet for PC/104.

Linux som generelt plattformvalg var uproblematisk. Matlab og Simulink kjørte problemfritt på Linux på arbeidsstasjonen, og Simulinkdiagrammer laget under Windows fungerte fint også under Linux. SSH over Ethernet gir også mange muligheter under Linux. SSH gjorde filoverføring meget enkelt, og er helt essensielt sammen med *Screen* under oppstart av CyberSwan før flyvning. Når CyberSwan sitt styresystem er grundig testet og helt sikkert kjører stabilt uten behov for overvåkning, kan styresystemet legges inn under oppstarten av Linux.

12.3 Matlab og Simulink med Real-Time Workshop

Matlab og Simulink ble valgt til å utvikle styresystemet for CyberSwan, og erfaringene med dette var meget gode. Med en slik plattform legges det ingen begrensninger for kompleksiteten i styresystemet. Så lenge en utvikler kjenner til Matlab og Simulink, trengs det lite kunnskaper om instrumenteringssystemet i CyberSwan. Styresystemet i CyberSwan bør kunne utvikles i Matlab og Simulink hvis prosjektet skal drives videre av nye personer.

Erfaringen med kodegenerering av Simulinkdiagrammer gjennom Real-Time Workshop er også gode. Grunnlaget for integreringen av instrumenteringen med Matlab og Simulink er lagt gjennom modellen *auav_ controlsys.mdl* som ligger vedlagt på CD. S-funksjoner som kommuniserer med alle IO-enhetene er laget og fungerer.

Styresystemet som ble eksportert med Real-Time Workshop genererte *Alarm Clock* fra tid til annen. Verken dokumentasjonen på Real-Time Workshop eller Linux Soft Real-Time Target (LNX) omtalte *Alarm Clock*. Det ble derfor søkt etter *Alarm Clock* i forbindelse med Real-Time Workshop på Internett uten at det ga resultater. Siden LNX bruker POSIX-timere til å lage Soft Real-Time, er det trolig lurere å lete etter *Alarm Clock* i dokumentasjon til POSIX eller Linux. Et annet alternativ er å bruke en annen timing enn LNX.

På grunn av problemet med *Alarm Clock* ble det valgt å ikke bruke modellen som er laget i Simulink under flyvningen. Kildekoden i de aktuelle S-funksjonene ble satt sammen til et loggesystem som ble brukt under første flyvning og et styresystem som ble brukt under andre flyvning. Dette ble utviklet i C++ uten Simulink og Real-Time Workshop. Siden systemene hadde begrenset kompleksitet, ville det ta mindre tid og klippe ut nødvendige kodelinjer enn å feilsøke i ukjente problemer. Ettersom det meste av koden var laget og testet, gikk det greit å lage både loggesystem og styresystem som var stabile. Det vil imidlertid ikke være gunstig å fortsette og utvide styresystemer i C++ i senere prosjekter siden et komplekst styresystem i C++ vil være vanskelig å vedlikeholde.

12.4 IO-kort

IO-kortet som er utviklet, fungerer bra, men det er behov for modifikasjoner for å få full utnyttelse av mulighetene kortet har. Detaljert evaluering av IO-kortet er lagt til slutten av kapittelet som omhandler IO-kortet (kapittel 8), men hovedpunkter er trukket frem i dette avsnittet.

Problemet med IO-kortet ligger i hovedsak at PALen ikke fungerer som planlagt med PCM-mottageren som ble brukt. På grunn av at isolasjonen mellom utgangene på PALen ikke er god nok, støyer de ulike kanalene til servoene for hverandre. Enten kan PCM-mottageren byttes ut, eller så kan PALen byttes ut i nye revisjoner av IO-kortet. Alternativt kan det hende det er tilstrekkelig å bytte denne til en nyere PAL med bedre egenskaper. Trolig var PALen som ble benyttet eldre enn fem år.

På grunn av disse problemene med PALen kan ikke IO-kortet ta inn PWM-signaler fra PCM-mottageren (manuell styring) og sende PWM-signaler ut til servoene samtidig. Det ble derfor gjort to flyvninger. Under første flyvningen logget systemet pådragene til servoene under manuell flyvning sammen med flyets oppførsel (IMU data). Under den andre flyvningen var PAL-kretsen byttet ut med en bryter som styrte hvilke signal servoene skulle få, enten fra PCM-mottageren (manuell styring) eller fra IO-kortet (autonom styring). Siden bryteren kunne styres med modellflyradioen på bakken, kunne en velge under flyvning om flyet skulle kjøres manuelt eller autonomt.

Strømforsyningen på IO-kortet fungerte meget godt og det samme gjorde ATMega128 og MAX233. Dette er derfor komponenter som bør brukes videre.

Kildekoden som er laget til ATMega128, fungerer meget godt med servoer og PPM-mottageren i testoppsettet, men altså ikke PCM-mottageren i CyberSwan. Hvis det skal brukes en annen løsning enn multipleksing av PWM-signaler via PAL, må avbruddsrutinene gjøres om, men teorien beskrevet i kapittel 7 egner seg godt til å bruke videre. Kildekoden på ATMega128 danner også et godt grunnlag for både generering og mottak av PWM-signaler.

12.5 Måleinstrumenter

IMUen (xSens MTi) var stabil og leverte gode målinger av flyets oppførsel. Målingene fra både første og andre flyvning ligger vedlagt på CD. Det var ventet at motoren kom til å lage varierende magnetfelt og dermed støye en del for magnetometeret i IMUen. IMUen

ble derfor plassert frem i flyet så den kom langt unna motoren. Målingene viser ingen tegn til støy fra motoren.

IMUen var brukervennlig, og den var godt dokumentert. Det fulgte med gode eksempler på kildekode og programvare for å teste og konfigurere IMUen. Disse verktøyene er nyttig når det er flere som skal dele på IMUen da hvert enkelt prosjekt ikke nødvendigvis bruker samme konfigurasjon. Rekonfigurasjon er enkelt å gjøre med medfølgende programvare.

GPSen GlobalSat EM-411, fungerte fra første stund. Den kunne virke lite dokumentert, men den var såpass enkel å bruke at den korte dokumentasjonen var tilstrekkelig. EM-411 støtter en rekke GPS-meldinger som er definert i NMEA 0183-protokollen. EM-411 kan konfigureres slik at den sender bare de meldingene det er behov for med en valgfri periodetid som kan settes i hele sekunder. Det finnes riktignok en binærprotokoll i NMEA 0183, men meldingene som brukes i CyberSwan består av tekststrenger som er lett leselig i et terminalprogram.

Trykksensoren er testet, men ikke implementert. Det er beregnet at den vil fungere i et pitotrør, men ikke i en høydemåler. Det er både analog og digital tilkoblingsmulighet til trykksensoren som er kjøpt inn til prosjektet. På grunn av mangel på innganger på ATMega128 er det kun lagt opp baner for analog overføring fra trykksensoren til ATMega128. Det som gjenstår for å få et fungerende pitotrør med denne trykksensoren, er implementering av analog måling på ATMega128 samt konstruere et pitotrør og trekke slanger mellom pitotrøret og trykksensoren.

Kapittel 13

Konklusjon

I denne rapporten er det beskrevet hvilke komponenter det er behov for i CyberSwan for å kunne styre og navigere CyberSwan autonomt. Det er testet at alle komponenter fungerer, og de viktigste komponentene er integrert i CyberSwan sitt styresystem. Det gjenstår en del arbeid for full integrering av alle komponenter, men erfaringene som er dokumentert i denne rapporten og valgene som er gjort, gir et godt grunnlag å bygge på i et eventuelt videre arbeid med CyberSwan.

CyberSwan bruker målinger fra en IMU til å styre flyet mot en gitt retning. Denne IMUen gir gode og stabile målinger som gir tilstrekkelig informasjon til å kunne stabilisere flyet. Bortsett fra manuell hastighetsregulering, ble CyberSwan fløyet autonomt basert på 3D euler-vinkler og 3D rotasjonshastighet fra IMUen.

CyberSwan sitt styresystem kjører under Linux på PC/104. Det er gjort gode erfaringer både med Linux og PC/104 i CyberSwan. Valgte Linuxdistribusjon kjører stabilt på PC/104-kortet med strømforsyning fra IO-kortet og lagring på Compact Flash. Det er laget et rammeverk og lagt inn programvare i CyberSwan som gjør at oppstart, filoverføring, monitorering og testing av CyberSwan er stabil og godt fungerende.

Det er utviklet et IO-kort til CyberSwan som genererer pådragene til servoene under autonom flyvning. Dette IO-kortet ble også brukt til å måle de manuelle pådragene fra radiostyringen på bakken. På grunn av problemer med en PAL på IO-kortet, kan imidlertid ikke kortet generere både servopådrag og måle manuelle pådrag samtidig. Det er ikke kritisk for å fly autonomt, men CyberSwan vil få en bedre oppførsel hvis systemet kjenner servoenes posisjon før den skal ta over styringen selv. Loggingen av data vil også bli mer fullstendig siden en kan se nøyaktig når flyet er fløyet autonomt og når det er fløyet manuelt. For å kunne måle de manuelle pådragene og generere servopådrag fra IO-kortet samtidig, må PALen byttes ut med en enhet som isolerer kanalene bedre, eller det må settes inn en mikrokontroller som tar seg av dette.

IO-kortet leverer også strøm til servoene og har kontakter for tilkobling av både IMU og GPS. Det er også en trykksensor på IO-kortet. Denne trykksensoren er testet og egner seg til å brukes i et pitotrør, men det er ikke implementert programvare som tar inn målinger fra denne sensoren.

Målinger fra IMU, GPS og mottager samt pådragsorgan til servoene er implementert i et styresystem sammen med radiooverføring til bakken. Dette styresystemet er utviklet i Simulink og tester av dette systemet er utført på arbeidsstasjon med godt resultat. På

grunn av problemer med *Alarm Clock* som ikke ble løst, ble det brukt en forenklet utgave av dette styresystemet som utgangspunkt under de to flyvningene som ble gjennomført med CyberSwan. Instrumenteringssytemet fungerte som forventet under disse flyvningene tatt i betraktning de begrensningene som er beskrevet i forbindelse med PALen.

IO-kortet som er laget, viser at de fleste enhetene fungerer godt og egner seg til å bruke videre. Mikrokontrolleren ATMega128 ble brukt til å motta og generere PWM-signaler til servoene, Denne fungerte som forventet og egner seg godt til formålet. Erfaringer viser at PWM-signalene bør ikke gå via en PAL slik som det er gjort på dette IO-kortet. Alle PWM-signalene fra mottageren bør kobles inn på hver sin eksterne avbruddspinne på ATMega128. Det vil da være helt uavhengig hva slags mottager som brukes i CyberSwan.

Det kan være en god løsning at ATMega128 genererer nye PWM-signaler ut i fra PWM-signaler som måles fra mottageren, selv under manuell styring. Med en slik løsning vil en kritisk feil på ATMega128 sende flyet i bakken. Systemet bør i et slik tilfelle derfor være godt beskyttet mot feil som kan oppstå. Det kan derfor være tryggere og bruke en lignende fysisk bryterkrets som ble brukt under autonom flyvning og som er beskrevet i denne rapporten. Denne bryterkretsen må bestå av like mange brytere som PWM-signaler som trengs i CyberSwan, og for unngå for mye kabling i CyberSwan, bør bryterkretsen integreres på IO-kortet.

Kapittel 14

Videre arbeid

Instrumenteringssystemet i CyberSwan er komplisert og består av mange elementer og komponenter. Grunnlaget er lagt for integreringen av målinger fra IMU, GPS, modellflymottager og pitotrør, samt integrering av pådragsorganer i et modellfly, men det gjenstår å fullføre integreringen av enkelte komponenter.

PC/104-løsningen med lagring på Compact Flash egner seg godt til å bruke videre, men det vil bli behov for å koble til flere enheter enn IMU og IO-kort. Til dette er det to ledige RS-232-porter, selv om disse ikke fungerte da de ble testet. Det gjenstår å se om RS-232-portene er ødelagt og om en eventuelt bør lodde på nye driverkretser.

På sikt kan AVR32 byttes ut med PC/104 for å spare plass og strømforbruk, men integreringen av AVR32 er komplisert og det kan hende det trengs å gjøre mange tilpasninger før AVR32 egner seg i CyberSwan. Det kan derfor være lurt å bruke PC/104 frem til CyberSwan sin instrumenteringsplattform er fullført, slik at behovene til CPU-enheten er klart definert. Da vil det være et bedre grunnlag for å ta stilling til om AVR32 vil tilfredsstille CyberSwan sine behov.

Det bør utvikles et nytt IO-kort med enten to mikrokontrollere eller en annen mikrokontroller med flere tilkoblingspinner. Dette skyldes at det ikke vil være nok tilkoblinger på ATMega128 hvis IO-kortet skal motta seks PWM-signaler, generere seks PWM-signaler, og kommunisere via UART og I^2C. Tilkoblingen til GPS og IMU bør imidlertid fortsatt ligge på IO-kortet sammen med strømforsyning og trykksensor. Dermed unngår man for mye kabling i CyberSwan, og forholdene legges til rette for et stabilt instrumenteringssystem. Nytten av å samle flest mulig komponenter på IO-kortet viste seg tydelig når alt skulle plasseres nede i flyet.

Det bør også bygges et nytt fly hvor det er bedre plass til instrumenteringssystemet slik at alle enheter i instrumenteringssystemet kan kobles sammen før det settes ned i flyet.

Pitotrøret ble nedprioritert siden hastighetsmåling ikke var en kritisk viktig måling for å få flyet til å fungere. Det er laget sokkel for trykksensoren på IO-kortet, og det er lagt opp en bane på IO-kortet for analog avlesning på en av ATMega128 sine analoge innganger. Det som gjenstår for å integrere hastighetsmåling i CyberSwan er å fysisk lage pitotrøret, trekke trykkslanger mellom pitotrøret og IO-kortet, samt utvikle programvare for avlesning og overføring av informasjon til styresystemet. Det kan også legges opp til digital avlesning av målingen. Dokumentasjon av kommunikasjon med trykksensoren over I^2C er vedlagt på CD.

Erfaringene er gode med Matlab og Simulink med Real-Time Workshop. Når det skal integreres mange IO-enheter vil det bli betydelig mer oversiktlig å bruke en slik plattform. Et godt grunnlag er lagt for integrering av IO-enhetene i Simulink, men det er behov for å få en mer stabil kjøring av programvaren ved å finne kilden til *Alarm Clock*. Det trengs også simulatorer for IO-enhetene slik at det enklere kan kjøres gode simuleringer under utvikling av CyberSwan sitt styresystem. Disse simulatorene bør kunne erstatte hver enkelt av S-funksjonene i plattformen for styresystemet i Simulink.

Bibliografi

[1] The C++ Resources Network. web, www.cplusplus.com, apr 2007.

[2] Mathworks.com. web, www.mathworks.com, mars 2007.

[3] Wikipedia. web, www.wikipedia.org, apr 2007.

[4] Ole-Johan Ellingsen. The CyberEagle Project, Automatic control of a model helicopter. Diplomoppgave, Teknisk Kybernetikk, NTNU, jun 2002.

[5] Mikael K. Eriksen. Ground Station and Hardware Peripherals for Fixed-wing UAV: CyberSwan. Diplomoppgave, Teknisk Kybernetikk, NTNU, jun 2007.

[6] Jon Bernhard Høstmark. AUAV Fixed Wing Prototyp. Diplomoppgave, Teknisk Kybernetikk, NTNU, jun 2007.

[7] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall PTR, 2 edition, 1988.

[8] Maxim Integrated Products. *+5V-Powered, Multichannel RS-232 Drivers/Receivers*. www.maxim-ic.com, 2006. Datasheet.

[9] Australia Monash University. Monash uav research group (aerobotics). web, www.ctie.monash.edu.au/AEROBOTICS, 2006.

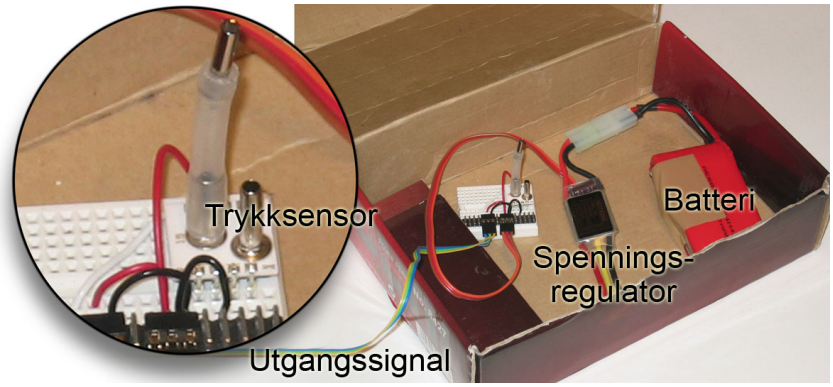
[10] NASA. Aerosonde - autonom UAV. web, www.aerosonde.com.au, 2006.

[11] Ismael Ripoll, Pavel Pisa, Luca Abeni, Paolo Gai, Agnes Lanusse, and Sergio Saez. *RTOS State of the Art Analysis*. http://mnis.fr/ocera_support/rtos/book1.html, 2002.

[12] Øyvind Netland. Developing Embedded Control System Platform using Atmel AVR32 Processor. Diplomoppgave, Teknisk Kybernetikk, NTNU, jun 2007.

Tillegg A

Test av trykksensor



Figur A.1: Test av Silicon Microstructures SM5852 differensielle trykksensor

A.1 Utstyr

- Trykksensor Silicon Microstructures SM5852-001-D-3-L
- Batteri
- Spenningsregulator
- Comboskop
- to kodensatorer, ledninger og tilkoblingsplate

A.2 Hensikt

Hensikten med denne testen er å se om trykksensoren som er kjøpt inn, vil fungere til høydemålemåling. Nøyaktighet og sensorens totale måleområde skal testes.

A.3 Beskrivelse

Testen gikk ut på å flytte trykksensoren vertikalt og se om det gav et forventet utslag på det analoge utgangssignalet.

Trykksensor og strømforsyning ble lagt i en boks som enkelt kunne flyttes opp og ned. Denne boksen var viktig for å redusere støy på grunn av temperaturforskjell og vind. Batteri og spenningsregulator ble tatt fra et modellfly og koblet til trykksensoren oppe i boksen siden det var en enkel og god løsning. Signalkabelen ble strekt ut av boksen og koblet til et comboskop som ble stillt inn slik at signalet var lesbart. To kondensatorer ble koblet i parallell mellom jord og signalet. Dette mer en halverte rippelen i signalet.

Comboskopet ble stilt inn på en oppløsning på 100 mV/div. Boksen ble løftet raskt opp og ned samtidig som signalet som konstant var under drift (ca 1-5 mV per sekund) ble lest av.

A.4 Feilkilder

Det var vanskelig å få klargjort alle elementene slik at målingene kunne leses av skikkelig. Dette av flere grunner:

- Det var begrensede muligheter til å justere offset på comboscopet siden man måtte ha så høy oppløsningen for å få lest av nøyaktig nok.
- Referansetrykket måtte justeres slik at det kom inn i et lestbart område på oscilloskopet
- Målingen driftet ved berøring, trolig på grunn av økt temperatur i referansetrykket.
- Analog måling introduserer støy
- Sensoren var som forventet ekstremt følsom, og rippelen på signalet var på ca 20-60 mV selv med to kondensatorer på 10 μF i parallell ved probemålingen.
- Drift (trolig temperatur) gjorde at man raskt måtte se utslag på målingen og dermed gjøre raske bevegelser på trykksensoren. Det introduserte noe vindstøy, men å legge måleinstrumentet oppe i en boks fjernet noe drift og vindstøy.
- Høyde over havet der testen ble utført var ca 60 meter

A.5 Resultater

$$P = P_0 e^{-\frac{Mgz}{RT}} \tag{A.1}$$

hvor

P	er trykk,
P_0	er trykk ved havoverflaten (101,325 kPa),
M	er masse av 1 mol luft (M = 0,029 kg mol^{-1}),
g	er gravitasjon (g $\approx$ 9,8 m/s^2),
z	er høyde over havet,
R	er gasskonstanten (R = 8,314 J K^{-1} mol^{-1}),
T	er temperaturen (20°C=293°K)

Det er differensielt trykk mellom P_1 ved høyde z_1 og P_2 ved høyde z_2 som er interresant

$$P_2 - P_1 = P_0 e^{-\frac{Mg}{RT}}(e^{z_2} - e^{z_1}) \tag{A.2}$$

Vi kan antar en startposisjon ved z_1=200 m. Vi antar at flyet ikke vil stige mer en 100 m så vi setter z_2=300 m. Vi setter dette inn i ligning A.2

$$P_2 - P_1 = -1148Pa \tag{A.3}$$

Trykket vil altså synke med 1.148kPa på disse 100 meterne. Den mest nærliggende sensoren i SM5800-serien har maks utslag på 1.0kPa. Da blir $P_2 - P_1$=1.0 kPa og det gir med ligning A.2 en makshøyde på 87 meter.

Trykksensoren ble løftet ca 1 meter vertikalt opp og ned, og da ble det observert et sted mellom 20 mV og 60 mV endring i signalet.

Måleområdet på sensoren ble målt til $[0, 5]$ V. Vi regner ut hvor mange høydemeter sensoren maks kan måle:

$$\frac{\Delta h}{\Delta x} = \frac{h_{maks}}{x_{maks}} \tag{A.4}$$

$$\frac{\Delta h}{\Delta x} \cdot x_{maks} = h_{maks} \tag{A.5}$$

hvor Δh er høydeforskjellen i testen, Δx målt utslag på comboskopet, x_{maks} er maks måleområde og h_{maks} er maks høyde trykkmåleren kan måle. Vi setter inn måleverdier i ligning A.5:

$$\frac{1m}{(40\ pm20)mV} \cdot (5V - 0V) = \begin{cases} 83m & \text{hvis} \quad \Delta x = 20mV \\ 125m & \text{hvis} \quad \Delta x = 40mV \\ 250m & \text{hvis} \quad \Delta x = 60mV \end{cases} \tag{A.6}$$

Her ser vi at sensoren har et måleområde h_{maks} som strekker seg over 125 meter, men på grunn av unøyaktige målinger kan det i minste fall være 83 meter og i største fall 250 meter.

SM5852-001-D-3-L har et måleområde på [0, 1.0] kPa. Vi regnet ut at dette skulle gi en høyde på 87 meter (eller ca 85 meter hvis man skal ta hensyn til at testen ble utført 60 meter over havet, og ikke 200 meter).

Målefeil/avlesningsfeil gitt i prosent:

$$\frac{(\pm 20)mV \cdot 100\%}{(5V - 0V)} = (\pm 0, 4)\% \tag{A.7}$$

Det viste seg at trykksensoren var meget følsom på temperaturforandringer. Få sekunders berøring gav et mer enn, og til dels mye mer enn 10mV endring på oscilloskopet. Hvor nært koblet til temperatur denne trykkmålingen er, kan vi regne ut med utgangspunkt i generell gasslov. Vi kan da finne hvor stor temperaturforandring som skal til for å øke trykket i referansekammeret like mye som måleområdet til trykksensoren som er på 1.0 kPa.

$$\frac{P_1 V_1}{T_1} = \frac{P_2 V_2}{T_2} \tag{A.8}$$

Vi antar at volumet er konstant ($V_1 = V_2$) og får:

$$T_1 = T_2 \frac{P_1}{P_2} \tag{A.9}$$

Vi setter P_1 til atmosfæretrykket (0 meter over havet) og $P_2 = P_1 + 1,0kPa$ som er trykket som må til for å maksimere trykkmåleren. Vi setter $T_2 = 20°C = 293°K$

$$T_1 = 293°K \frac{101,325kPa}{102,325kPa} = 290,1°K = 17,1°C \tag{A.10}$$

Av dette ser vi at en temperaturforskjell på 2,9° C i referansekammeret vil alene være nok til å maksimere trykkmåleren.

A.6 Konklusjon

Vanskelige avlesningsforhold og høy rippel gav unøyaktig måling. Alt i alt var det vanskelig å få noe konkret ut av selve testen. Det som testen viser er at høydemåling med den nøyaktigheten som det her er snakk om, kan bli vanskelig i praksis.

Testen viser riktignok at det teoretiske måleområdet ligger i nærheten og det som ble målt.

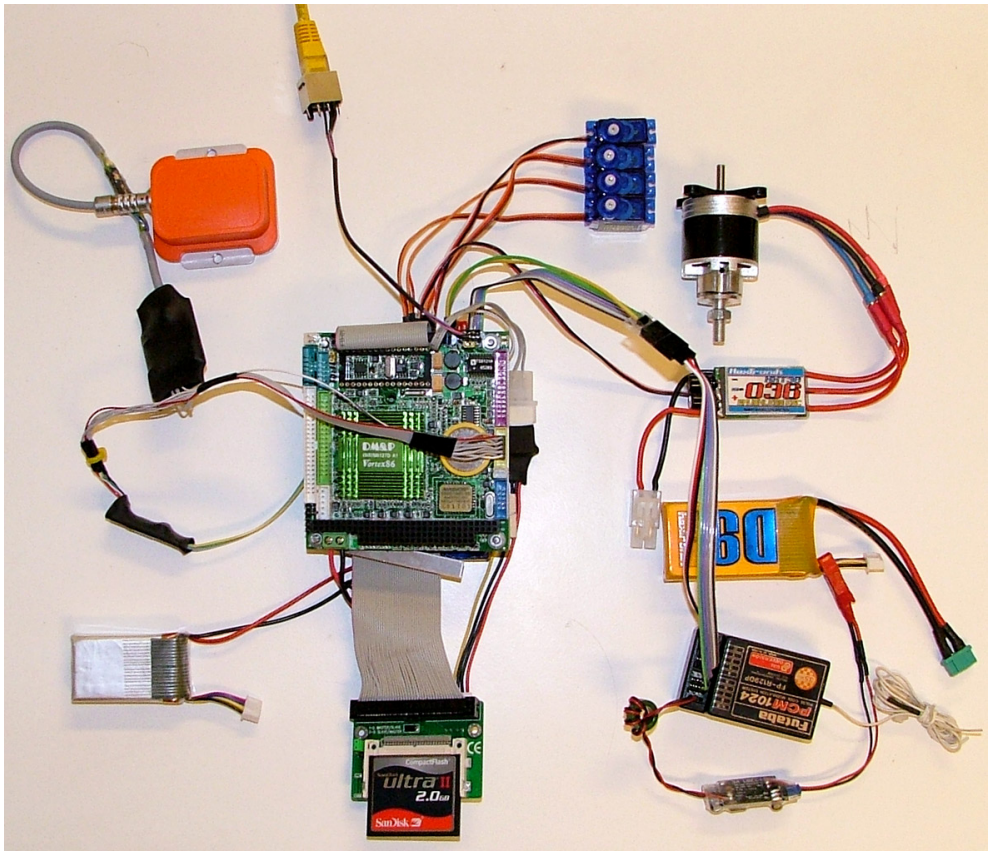
Testen gir ingen konkret oppløsning på signalet, men den kan sammenlignes med målefeilen som var på ±0,4% av maksutslag. Ved samme målefeil vil opløsningen bli større en 0,8 % av maksutslag eller ca 1 meter. Målingen ble derimot gjort analogt. Ved å bruke den digitale utgangen på sensoren, vil signalmålefeilen fjernes og oppløsningen vil trolig bli betydelig bedre.

Det største problemet denne testen avdekker, er at trykksensoren er meget følsom for temperatursvigninger. 2,9°C temperaturendringer vil alene maksimere trykkmåleren uansett hva målingen viste før endringen.

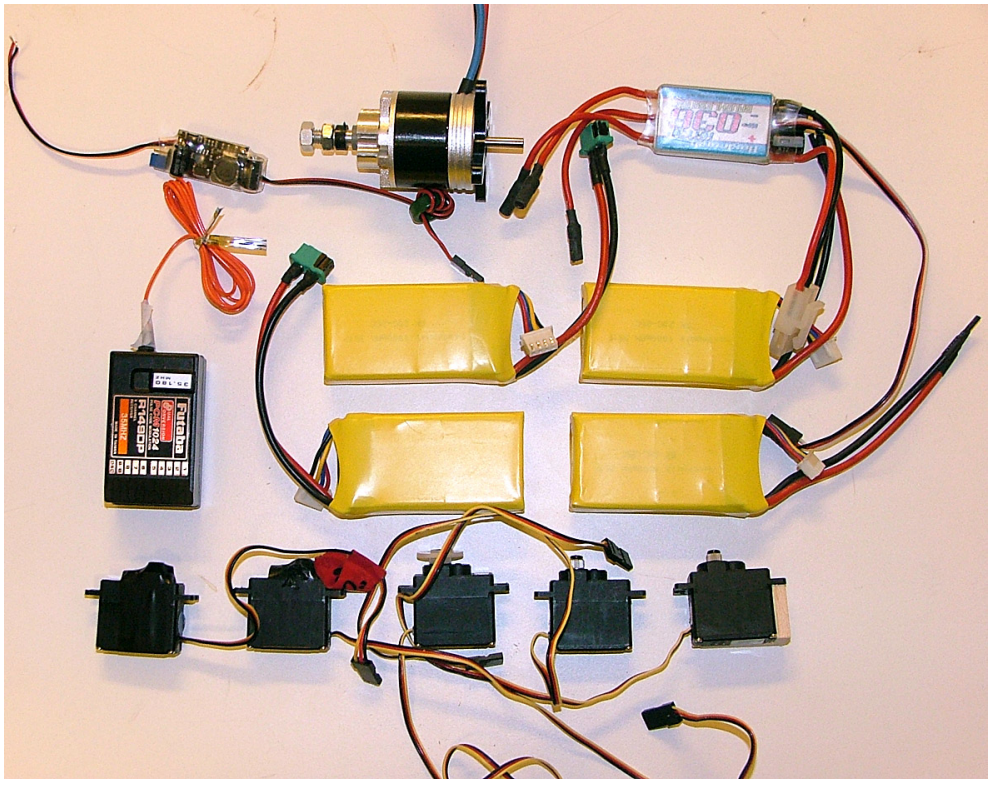
Vi kan anta at det vil være temperaturforandringer oppe i flyet, og det er ikke sikkert endringene vil være mindre enn 2,9°C under en flyvning. Dette gjør en slik trykkmåler dårlig egnet som høydemåler.

Tillegg B

Bilder



Figur B.1: Instrumenteringssystem, med pådragsorganer



Figur B.2: RC-deler ved manuell flyvning uten instrumenteringssystem

Tillegg C

Katalogoversikt på CD

```
./CyberSwan Plattform - oppsett i Simulink:
./Datablad:
./Datablad/gps:
./Datablad/ICOP-6070 - Vortex86:
./Datablad/imu:
./Datablad/imu/xSens Software:
./Datablad/msm586sl:
./Datablad/Nano ITX hovedkort:
./Datablad/pegasus:
./Datablad/pegasus/howto:
./Datablad/PM-GX_datasheets:
./Flyvning 1 - Logging:
./Flyvning 1 - Logging/kildefiler:
./Flyvning 2 - Autonom Styring:
./HowTos:
./IO-kort:
./IO-kort/ATMega128 kildekode under autonom flyvning:
./IO-kort/ATMega128 kildekode under logge flyvning:
./IO-kort/ATMega128 kildekode under testoppsett:
./IO-kort/IO-kort (Eagle):
./IO-kort/pal:
./Latex kildekode:
./Programmer:
./Programmer/CuteCom:
./Programmer/CuteCom/packages needed:
./Programmer/Terminal v1.9b:
./Video:

Instrumentering av autonomt ubemannet fly - CyberSwan.pdf
```