Unmanned Aerial Vehicle

Hovedoppgave
våren 2010

Geir Bø Lunde

Institutt for data- og elektroteknikk,
Det teknisk- naturvitenskapelige fakultet,
Universitetet i Stavanger

Sammendrag

Det er i denne oppgaven sett på metoder for å stabilisere et helikopter ved hjelp av elektronikk. Hovedvekten er da lagt på å finne orienteringen til helikopteret ved bruk av softsensor.

Det er bygget et fire-motors helikopter. Dette styres ved å endre pådraget på de forskjellige motorene. Det er også blitt utviklet egen elektronikk, som kan styre motorene og samle inn data fra sensorene. Denne sender i tillegg informasjonen tilbake til datamaskinen. På datamaskinen er det utviklet et eget program, som presenterer denne informasjonen for brukeren.

Det er ingen lett oppgave å måle orienteringen til helikopteret. Det er blitt vist at de billige sensorene ikke gir bra nok målinger for dette. Det er derfor utviklet en softsensor, som kombinerer flere målinger. For å gjøre målingene er det brukt tre gyroer. Disse lider av bias-drift og blir derfor feil over tid. For å kompensere for bias-driften er det brukt et tre-aksers akselerometer. Dette måler tyngde-akselerasjonen og bruker denne til å beregne orienteringen. Alle kreftene som virker på helikoptret vil også gi utslag på akselerometeret. Denne målingen kan derfor heller ikke stoles på. For å kombinere disse målingene er det blitt sett på flere forskjellige softsensorer. Det er vist at en oppnår best resultat ved å bruke et andre ordens komplementær-filter. For å vise dette er det blitt gjort simuleringer av de forskjellige metodene i Matlab. Disse simuleringene er da basert på virkelig data hentet fra sensorene på helikopteret. Det er i tillegg vist at en ikke oppnår bedre resultater ved å bruke støyfjernings-filter før softsensoren.

For å gjøre selve reguleringen er det brukt PID-regulatorer, som er implementert på en mikrokontroller fra Atmel. Denne henter også inn all informasjon fra sensorene og kjører softsensor-algoritmen. Det er også blitt designet et eget kretskort hvor alle sensorene er montert. Brukeren kan da styre helikopteret ved hjelp av en joystick koblet til gjennom USB-porten på datamaskinen.

Systemet er utviklet med mulighet for modulbasering. De forskjellige modulene kan da kobles sammen ved hjelp av I2C. Dette gir mulighet for videre arbeid uten for mye kjennskap til hovedsystemet. Det er også blitt utviklet en kommunikasjons-modul, som lar helikopteret kommuniserer med datamaskinen ved hjelp av ZigBee. Dette er et eget kretskort, som er koblet til hovedsystemet ved hjelp av en flatkabel.

Innhold

1 Viktige begreper **1**

2 Introduksjon **2**

2.1 Rapportens innhold og struktur 5

3 Konstruksjon **6**

3.1 Motorer 6

3.2 Hastighets-kontroller 7

3.2.1 PPM-signal 7

3.2.2 I2C-styring 8

3.3 Batteri 8

3.4 Propeller 8

3.5 Helikopterkroppen 9

3.6 Koblingsskjema 11

3.7 Pris 11

4 Modulbasering **12**

4.1 Modul-forslag 13

4.1.1 Orienterings-estimator 13

4.1.2 GPS/INU 13

5 Valg av hjerne **14**

5.1 Vanlig mikrokontroller 14

5.2 FPGA 15

5.3 Sanntids linux 16
5.4 Valg 16

6 Måleinstrumenter **17**

6.1 Differensiell GPS 17
6.2 Termiske sensorer (IR) 17
6.3 Gyroskop 18
6.3.1 Mekanisk 18
6.3.2 Optisk gyro 19
6.3.3 MEMS-gyro 19
6.4 Akselerometer 20
6.5 Magnetometer 20
6.6 Valg 21

7 Implementering av sensorer **22**

7.1 Gryo 22
7.2 Akselerometer 25

8 Kretskort **27**

9 Målestøy **31**

9.1 Logisk filter 32
9.2 Gjennomsitts-filter 33
9.3 Avansert filtrering 34
9.3.1 Digitalt filter 36

9.3.2 Analogt filter 41

10 Representasjon **43**

10.1 Eulers-vinkler (enkel metode) 43

10.2 Avansert ulineær metode 45

10.2.1 Rotasjonsmatriser 46

10.2.2 DCM Utledning 47

10.2.3 Normalisering 50

11 Beregning av orientering med akselerometer **51**

11.1 Separate vinkler 51

11.2 Metode 2 53

12 Softsensor **54**

12.1 Kalmanfilter 54

12.1.1 Kalman-implementasjon 55

12.2 Kalmanfilter på ulineært system 57

12.3 Komplementært-filter 58

12.3.1 1. orden 58

12.3.2 2.orden 59

13 Resultat **61**

13.1 Representasjon 62

13.1.1 Lineær 62

13.1.2 Ulineær 63

13.1.3 Sammenligning 64
13.2 Softsensor 65
13.2.1 Lineær-representasjon 66
13.2.2 Ulineær 67
13.2.3 1.orden og 2.orden 69
13.3 Oppsummering 73

14 Matematisk-modell **74**
14.1 Parameter identifikasjon 77

15 Regulatordesign **79**
15.1 Parameter-tuning 80

16 Firmware **81**
16.1 Kalibrering av hastighets-kontrollere 83

17 Hovedprogrammet **84**

18 INS **86**
18.1 GPS 86
18.2 Akselerometer 86
18.3 Kamera 87
18.4 Softsensor 87
18.5 Beregning av høyde 88
18.5.1 Sonar 88
18.5.2 Barometer 88

19 Kommunikasjons-modul **89**

19.1 Analog-mottaker 89

19.2 Zigbee 89

20 Videre arbeid **92**

21 Diskusjon og konklusjon **93**

21.1 Diskusjon 93

21.2 Konklusjon 93

Referanser **95**

Vedlegg **97**

A integrer.m **97**

B mitt_comp_filter.m **97**

C euler.m **98**

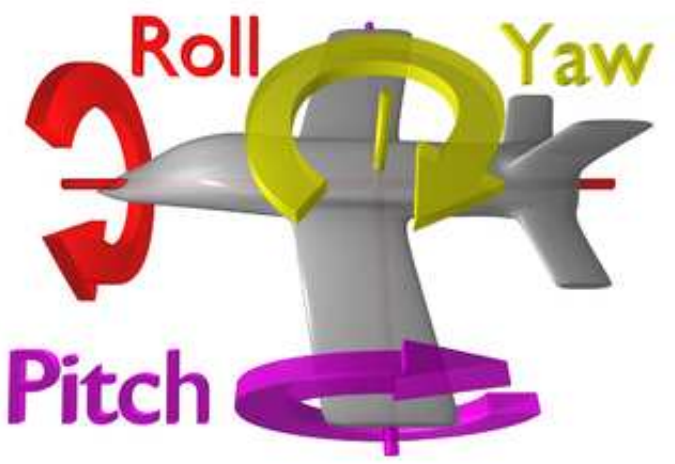
D DCM.m **99**

E Kretsskjema **101**

F Kommunikasjons-modul **102**

1 Viktige begreper

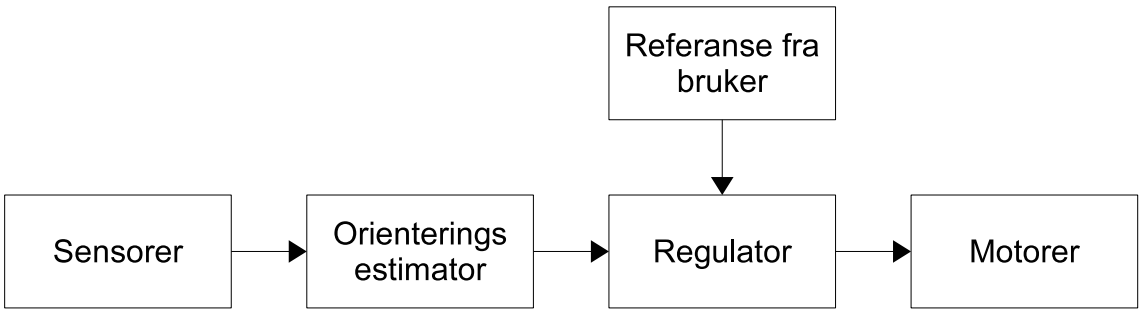
Helikopter og fly har seks frihetsgrader. Dette gjør at en trenger egne navn for å beskrive disse. For de som beskriver rotasjonen til objektet rundt massesentret brukes *yaw*, *roll* og *pitch*. I masteroppgaven vil det først og fremst være disse som skal reguleres. Å ha egne navn for rotasjonen gjør det svært mye enklere å forklare bevegelsen til helikopteret. Figur 1.1 viser hvilken vinkel hver av disse beskriver. I oppgaven er det valgt å kalle rotasjonen til helikopteret, oreintering.
De tre siste frihetsgradene beskriver bevegelsen i X, Y og Z retningen. Disse kalles ofte surge, sway og heave. Disse blir i oppgaven kalt posisjon.
Det er blitt valgt å bruke de engelske ordene. Siden disse brukes som norske innen dette fagfeltet. Et annet begrep som kommer til nytte er *hover*, som er en fornorsking av det engelske ordet. Dette brukes når roll og pitch er null og helikopteret altså svever.



Figur 1.1: Figuren er hentet fra http://en.wikipedia.org/wiki/Flight_dynamics

2 Introduksjon

Det er i den senere tid blitt mer og mer oppmerksomhet rundt UAV-teknologi. Det finnes mange bruksområder for en UAV, innen overvåkning, rekognosering, redningsarbeid og så videre. Den kan brukes i alle situasjoner hvor en ikke ønsker å sende en pilot. Eller hvor det er såpass dårlig plass at en ikke kan sende et fartøy med plass til pilot. Skal en foreksempel inn i en krigssone vil det alltid være fare for menneskeliv. Dette slipper en å tenke på når en bruker en UAV. Forskningen rundt dette jobber med enorme budsjetter. Her er blant annet NASA og det amerikanske forsvaret store aktører. Det er vanskelig å konkurrere med disse, men utvikling på et lavt budsjett er svært forskjellig fra et høyt budsjett. Mesteparten av forskningen som blir gjort er heller ikke gjort tilgjengelig.
Innen UAV-enme finnes det svært mange interessante problemstillinger. Disse passer perfekt til kybernetikk-fagfeltet. Dette gjelder blant annet bruk av softsensor, regulering, signal-forbedring og kommunikasjon. For å løse utfordringene må en nesten innom alle fagene på masterstudiet. Det ble derfor valgt å utvikle et slikt system.
Det første valget som måtte tas, sto mellom helikopter eller fly. Det finnes svært mange fordeler med begge, men det er noe enklere å gjøre helikopter-tester innendørs. Det finnes også mer ferdig dokumentasjon rundt utvikling av fly-systemer.
Det ble derfor valgt å utvikle et drone-helikopter. Disse er ofte svært dyre, men fremskritt innen MEMS(Micro Electrical Mechanical System) teknologi har gjort det mulig å utvikle en billig versjon. Budsjettet går ut over sensor-kvaliteten og en stor del av oppgaven blir derfor å estimere tilstanden til helikopteret. Som en ser finnes det svært mange problemer, som må løses innen dette emne. For å begrense oppgaven noe er det satt et hovedmål. Dette vil være å gjøre helikopteret stabilt i hover-tilstand. Altså å utvikle en stabil luftplattform, som kan stabilisere seg selv uten påvirkning av brukeren. Brukeren har kun mulighet for å sette referansene til regulatorene. Systemet skal deretter plassere helikopteret i den ønskede orienteringen. En slik luftplattform kan blant annet brukes for å filme eller fotografere områder.
For å begrense oppgaven enda mer er det lagt mest vekt på å finne orienteringen til helikopteret. Dette siden det er umulig å skapet et stabilt system uten å kjenne denne. Denne delen har også flest interessante problemer. En kan nå sette opp en oversikt over hvilke systemer, som må implementeres i oppgaven.



Figur 2.1: Figuren viser en oversikt over systemene, som skal implementeres.

Valg av helikoptertype ble gjort allerede i forprosjektet,[36]. Det ble her valgt å bygge et fire-motors helikopter. Siden dette har et enkelt mekanisk design og kan bygges relativt billig. Designet er også svært stabilt. Her bruker en elektronikk for å balansere ved hjelp av å regulere hastigheten på de fire propellene. For å forklare dette deler en inn i to motor-par. En ser da at ved å endre hastigheten på den ene motoren og øke hastigheten på den andre. Vil dette gi en endring i roll eller pitch. Et motor-par styrer altså pitch, mens det andre styrer roll. Yaw kontrolleres ved å la de to motor-parene rotere hver sin vei. Hvis et par går raskere enn det andre vil en få en endring i yaw.

Figur 2.2: Figuren viser helikopteret, som skal byggges i oppgaven

Et fire-rotors helikopter er ikke nytt. Det finnes allerede prosjekter, som jobber med å utvikle slike til hobbybruk. Disse prosjektene er ofte svært dårlig dokumentert. Og en fellesnevner er at de bygger på en prøv og feil metode. Et eksempel på dette er at en tar ferdig regulatorkode fra et annet prosjekt og implementerer dette direkte. Deretter prøver en nye regulator-parametere til helikopteret blir nokså stabilt. Dette går igjen for alt som brukes på disse systemene. Denne metoden gir helikoptre som er mulig å fly, men krever svært mye av "piloten". Det er altså ikke et autonomt-system.
Denne oppgaven vil derfor basere seg på å gjøre et analytisk studie. Dette gjøres ved å hentet data direkte fra sensorene og behandle disse i et simulerings-system på datamaskinen. På denne måten vil en kunne se eventuelle problemer i plott i stede for å gjette seg frem på parametere. En får da se problemene de forskjellige sensorene og metodene har direkte. Og kan finne løsninger på disse uten å gjøre tester med helikopteret i luften. Det er ikke funnet noe tidligere studie, som går gjennom dette på denne måten.

Et prosjekt som er blitt brukt som referanse i dette prosjektet er AEROQAUD, [2]. Her jobbes det med å utvikle et helikopter, som baserer seg på Arduino, [3]. Dette er et utviklings system som i det siste er blitt svært populært blant hobbyentusiaster. Firmwaren skrives da i et eget rammeverk, hvor de fleste metodene er ferdig skrevet i c. Dette gjør at brukeren trenger svært lite forståelse av hva som skjer i bakgrunnen, siden alt her er ferdig skrevet av andre. Det er mulig å fly dette helikopteret, men det krever veldig mye av piloten. Det er altså ikke særlig stabilt. Informasjonen som er hentet herfra er hovedsakelig mekanisk informasjon, som motor, bygging av selve helikopteret og så videre.

Informasjonen som finnes om dette emne er både vanskelig å finne og mye er direkte feil. Dette gjør litteraturstudie til en omfattende oppgave. Og det er brukt mye tid på litteratur fra forskjellige kilder, som viser seg å være feil. Mye av litteraturen er basert på bra oppgaver og virker derfor troverdig. Mye informasjon er også kommet frem etter diskusjon på forskjellige forum, da spessilet www.rcgroups.com.
Mange av elementene i oppgaven vil være felles for både fly og helikopter. Dette gjelder blant annet orienterings-estimeringen. Det er derfor ønskelig å utvikle systemet så generelt som mulig. Slik at de med et firmware bytte kan brukes på det andre systemet. På denne måten slipper en å utvikle ny hardware for forskjellige systemer.

Det vil bli endel begrensninger på hva som er mulig å implementere på helikopteret. Endel av disse begrensningene oppstår på grunn av løftekraften og størrelsen til helikopteret. I motsetning til større helikopter må systemet implementeres på en liten mikrokontroller. Dette gir begrensninger både innen plass og prosessorkraft. Et viktig mål i prosjektet er at det skal kunne bygges så billig som mulig. Det skal også være mulig for en lekmann å bygge sin egen versjon av helikopteret. Materialene som brukes må derfor være lett tilgjengelige og billige. Dette setter ytterlige begrensninger på hva som er mulig å oppnå.
Det finnes svært lite informasjon om dynamikken til systemet. Det er derfor vanskelig å vite hvor gode målingene må være og hvor rask sampletid en trenger. Det er derfor i stede valgt å prøve å gjøre disse best mulig, med de begrensningene systemet setter. En kan se at et så lite system vil få svært rask dynamikk. Det vil derfor bli forsøkt å presse sampletiden lavest mulig. Og på denne måten få med mest mulig informasjon, som senere kan brukes til å lage modell.
For å oppsummere, vil problemstillingen bli å designe en stabil luftplattform. Hvor systemene testes i et simuleringsverktøy i stede for å implementeres direkte. Systemet skal også bygges på et lavt budsjett slik at hvem som helst kan bygge sitt eget. En må da først utvikle elektronikk, som kan hente inn den nødvendige informasjonen. Videre må selve helikopteret bygges. Deretter må en utvikle programvare, som kan bruke sensor-data til å beregne orienteringen til helikopteret. Og gi pådrag til motorene basert på denne informasjonen.

2.1 Rapportens innhold og struktur

Kapittel 1 går gjennom viktige begreper brukt i oppgaven.

Kapittel 2 er en kort innledning som tar for seg bakgrunn og problemstilling for oppgaven.

Kapittel 3 gir en beskrivelse av hvordan selve helikopteret er konstruert.

Kapittel 4 tar for seg muligheten for modulbasering.

Kapittel 5 går gjennom valg av hjerne ("mikrokontroller") til helikopteret.

Kapittel 6 beskriver de forskjellige måleinstrumentene, som er blitt vurdert til oppgaven.

Kapittel 7 viser hvordan de forskjellige måleinstrumentene er blitt implementert.

Kapittel 8 er en kort beskrivelse av kretskort designet

Kapittel 9 viser metoder for å fjerne målestøy. Ved hjelp av støyfjernings-filter.

Kapittel 10 beskriver metoder for å representere orienteringen til helikopteret.

Kapittel 11 viser metoder for å beregne orientering ved hjelp av akselerometer

Kapittel 12 går gjennom forskjellige typer softsensorer, som kan brukes for å estimere orienteringen

Kapittel 13 viser resultatet av å bruke metodene i de tidligere kapitlene. Valg av beste metode blir også tatt her.

Kapittel 14 går gjennom metoder for å finne den matematiske-modellen.

Kapittel 15 er en beskrivelse av regulator-designet.

Kapittel 16 er en kort beskrivelse av firmwaren til helikopteret.

Kapittel 17 er en kort beskrivelse av dataprogrammet, utviklet i oppgaven

Kapittel 18 beskriver metoder for å finne posisjonen til helikopteret

Kapittel 19 beskriver kommunikasjons-modulen utviklet i oppgaven

Kapittel 20 går gjennom muligheter for videre utvikling av oppgaven

Kapittel 21 konklusjon for oppgaven

3 Konstruksjon

Det første som måtte gjøres i oppgaven var å bygge det fysiske helikoptret. Det er ikke lagt mye vekt på konstruksjon i denne oppgaven. Og dette blir derfor gjennomgodt nokså raskt. Et av målene var her at det skulle kunne bygges billig hjemme, uten for mye avansert utstyr. Slik at hvem som helst skulle kunne bygge sin egen versjon. Elektronikken utviklet i resten av oppgaven skal deretter kunne brukes for å kontrollere helikopteret. Dette gjør at brukeren ikke trenger en inngående forståelse av systemet. Designet blir derfor gjort på enklest mulig måte. Mange av ideene ble hentet fra [2]. Blant annet motorer og motorstyring.

3.1 Motorer

Noe av det første som måtte velges i oppgaven var hvilke motorer som skulle brukes. Her finnes det et hav av muligheter. Dette er derfor et vanskelig valg, som må tas basert på erfaring. Valget ble tatt ut fra anbefalinger på [2]. Her ble det anbefalt å bruke børsteløse motorer av typen, TowerPro Brushless Outrunner 13A. Disse er svært billige og løftekraften skal være bra. De var relativ lang bestillingstid, disse ble derfor bestill uten videre overveiing. Motorene ble kjøpt fra, [13]

Figur 3.1: Bilde er hentet fra, [13]

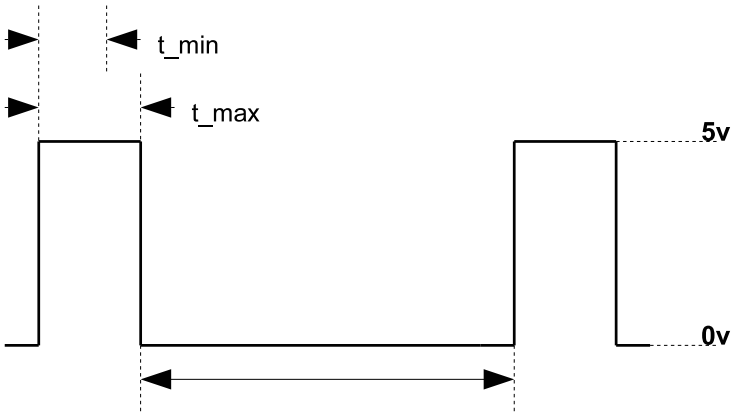
3.2 Hastighets-kontroller

Som en ser har de børsteløse motorene tre ledninger inn. En trenger derfor spesielle hastighets-kontrollere, ESC (Electric Speed Controller) for å kontrollere hastigheten på motorene. Disse har egne mikrokontrollere, som styrer sekvensen som sendes inn gjennom de tre ledningene. Valget ble også her tatt basert på pris og anbefalinger fra, [2] Her ble det valgt å bruke TURNIGY Plush 18amp. Disse kontrolleres ved hjelp av PPM-signal. Dette blir gjennomgodt under. Disse ble også kjøpt fra [13]

Figur 3.2: Bilde er hentet fra, [13]

3.2.1 PPM-signal

Den valgte hastighets-kontrolleren styres som sagt ved hjelp av et PPM(Pulse Position Modulation)-signal. Dette er samme signal som brukes for å styre vanlige servoer, [38]. Signalet kan ses i figur 3.3.



Figur 3.3: Figuren viser et PPM-signal

Her sendes det en puls som ligger mellom 1 til 2 ms, deretter kommer det en pause på 20ms. For så å repetere hele prosessen. Pulsen styrer da altså pådraget til motorene. En ser at det da tar fra 20 fil 22 ms for å oppdatere pådraget til motorene. Dette vil gi en oppdateringsfrekvens på (1/22ms) = 45 Hz. Dette bør være bra nok til vårt formål. AeroQuad prosjektet kjører de samme kontrollerne ved hjelp av et vanlig PWM-signal. Hvor pulsen altså er like lang som pausen. Dette ble derfor også testet i dette prosjektet, med svært gode resultater. Dette gjør at de fire pådragene nå enkelt kan gis fra fire PWM kanaler.

3.2.2 I2C-styring

En annen måte å kontrollere hastighets-kontrollerne på, er å bruke I2C. Denne metoden passer svært bra til dette prosjektet siden en da kunne koblet seg rett inn på I2C-nettverket. En hadde på denne måte ikke brukt noen av timerene på mikrokontrolleren. Denne typen kontroller kan kjøpes ferdig, men det vanligste er å bygge om PPM-kontrollere. Siden kontrollerne inneholder en mikrokontroller er det mulig å bytte programvaren. En kan da selv velge hvordan en vil kommunisere med kontrolleren. Dette er endel arbeid og ble derfor ikke påbegynt i denne oppgaven. Denne oppdateringen kunne heller vært en ide til en senere bacheloroppgave.

3.3 Batteri

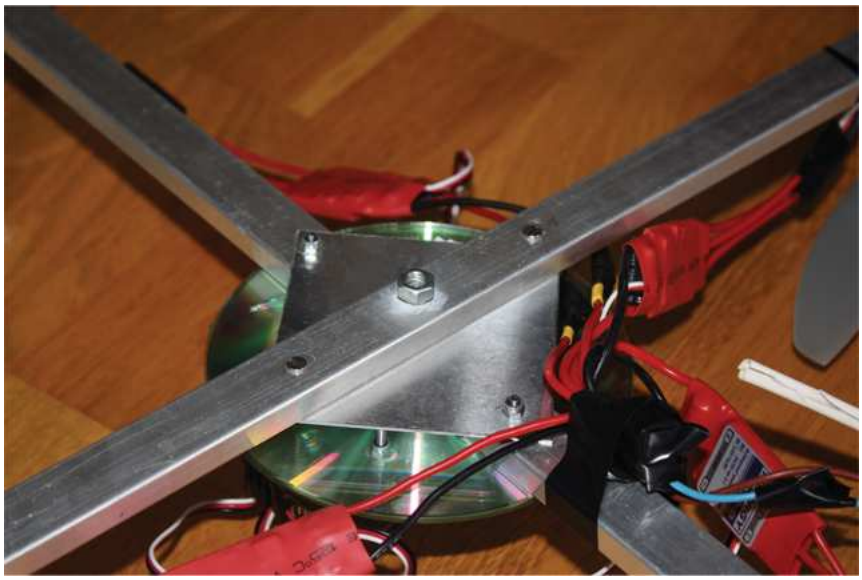
For å kunne fly helikopteret var det tungvindt å være avhengig av strøm fra bakken. Det var derfor nødvendig med et batteri. Igjen ble valget tatt basert på erfaringer fra AeroQuad prosjektet. Valget falt på ZIPPY Flightmax 4000mAh 3S1P 20C. Dette skal gi en flytid på ca 15min og passer derfor fint til dette formålet. Også dette ble kjøpt fra, [13].

3.4 Propeller

For at helikopteret skulle kunne fly, var en avhengig av å montere propeller på motorene. Siden de to motor-parene har propeller som går hver sin vei, trenger en to typer propeller. Grunnen til dette er at hvis en snur retningen på en vanlig propell vil den presse luften feil vei. Valget ble igjen tatt basert på tips fra AeroQuad. Her ble APC 10x4.7 Slow Flyer og APC 10x4.7 Slow Flyer Pusher valgt. Disse ble kjøpt fra carancho.com

3.5 Helikopterkroppen

Det siste som må bygges er selve kroppen til helikopteret. Her er det svært viktig med sterke, stive og ikke minst lette materialer. Det ble prøvd flere forskjellige materialer bandt annet PVC. Valget falt til slutt på bruk av Aluminium siden dette er lett, sterkt og relativt billig. Trox Auranor, kunne skaffe to 1m lange aluminiums bjelker. Disse var formet som rektangler med mål 1cm x 2cm. Denne typen er akkurat bred nok til å kunne feste motorene direkte i bjelken. Og ble derfor valgt til oppgaven.



Figur 3.4: Figuren viser hvordan helikopterkroppen ble satt sammen

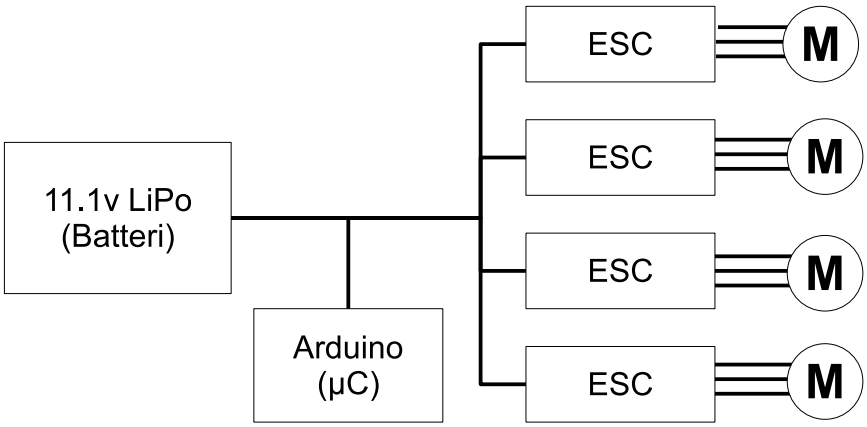
I et optimalt helikopter vil alle motorene stå i samme høyde. For å gjøre designe så enkelt som mulig ble det i stede valgt å kappe bjelken i to like deler. For så å legge disse delene over hverandre for å danne et kryss. For å stabilisere dette ble det skåret til en aluminiumsplate. Denne ble lagt mellom bjelkene. Deretter ble det brukt en håndholdt drill til å lage huller, til skruer. For å feste elektronikken ble det brukt to vanlige cd-plater, som ble limt sammen for å gjøre dem stivere. Disse ble brukt for at en skal slippe å ha elektronikken i kontakt med aluminium. Tre huller ble drillet helt ytterst på hver av bjelkene slik at en kunne montere motorene her. Motorene kom med alt festemateriell i pakken. Dette kan en se på bilde under. Nå gjenstod det bare å sette i skruer og systemet var ferdig.

Figur 3.5: Bilde viser hvordan motoren ble montert.

Som en kan se er dette et relativt raskt bygg, men hoved-fokuset i oppgaven er ikke konstruksjon. Kravet til denne delen var derfor bare å lage noe som fungerte. Resultatet kan ses i figur 2.2.

3.6 Koblingsskjema

En har nå alle delene som trengs for den mekaniske delen av oppgaven. Neste steg var å koble alt sammen. Hvordan dette ble gjort vises i det forenklede koblingskjemaet under.



Figur 3.6: Koblingskjema

3.7 Pris

Det her her tatt med en rask oversikt over prisen på de forskjellige delene til helikopteret. Som en ser kan dette bygges svært billig. Aluminiums delene var gratis og det er derfor ikke funnet pris på disse.

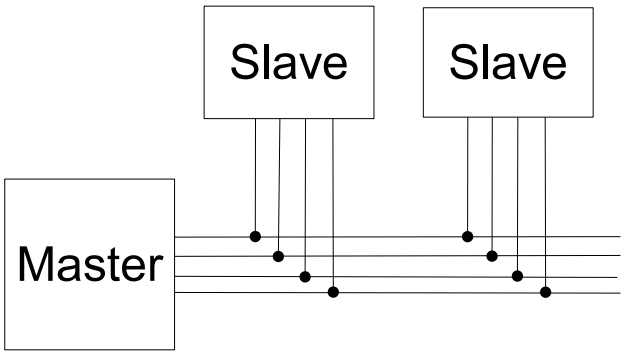
Del	Navn	Pris (USD)	Antall	Sum (USD)
Propell	GWS HD8040 3 Blade Prop 2PK	2.69	1	2.69
Propell	GWS HD8040 3 Blade Prop CR 2PK	2.69	1	2.69
Motor	TowerPro Brushless Outrunner 13A	6.39	4	25.56
ESC	TURNIGY Plush 18amp Speed Controll	11.31	4	45.24
Batteri	ZIPPY Flightmax 4000mAh 3S1P 20C	19.99	1	19.99

Som en ser kan hele systemet bygges for 96.17 USD eller 624 NOK. Dette blir da uten elektronikken, men fortsatt svært billig.

4 Modulbasering

For å gjøre systemet så allsidig som mulig. Ble det valgt å gi systemet mulighet for modulbasering. Disse modulene skal kunne operere uavhengig av hverandre. Slik at de enkelt kan byttes ut, uten å påvirke resten av systemet. Det blir altså en hoved-enhet/kort og flere slave-enheter. En kan foreksempel ha to kommunikasjons-kort, ett som kommuniserer via ZigBee. Og ett som styres ved hjelp av vanlig kontroller. Eller skulle en ønske å styre systemet ved hjelp av Wifi er også dette mulig. Disse kortene kan da enkelt byttes ut etter hvilke spesifikasjoner en ønsker på helikopteret. En av det store fordelene med modulbasering er at en kan spre arbeidsmengden over flere mikrokontrollere. Hvor hver slave gjør en spesifikk oppgave. Deretter kan masteren hente inn informasjon og fordele den dataen som trenges. Et eksempel kan være en modul som gjør orienterings-estimering. Masteren kan da hente inn denne informasjonen herfra for å sende den videre til en annen modul som tar seg av reguleringen.
Modulbaseringen gjør også at systemet blir enklere å videre utvikle. Skal det legges til funksjoner på helikopteret trenger en ikke nødvendigvis forstå hele systemet. En kan lage en egen modul som gjør den nye oppgaven. En trenger da bare en enkel oversikt over hva de forskjellige modulene forventer inn og ut.

Får å få systemet modulbasert må kortene kunne kommunisere. En trenger da en kommunikasjonsprotokoll. For å gjøre dette enkelt ble det valgt å bruke I2C. De fleste nyere mikrokontrollere har dette innebygget. Det blir derfor en enkel sak å gjøre dette i praksis. En annen fordel er at I2C er et bus-system. En trenger altså ikke en kabel fra hver modul. Det kan i stede brukes en flatkabel som går innom hver modul. Denne må ha minimum tre ledere til jord, data og klokke. For å slippe å ha egen spennings-regulator på hvert kort ble det også lagt til 5v og 3.3v i denne.



Figur 4.1: Figuren viser bus-systemet

Hvert modul bygges da med en egen mikrokontroller. Denne settes opp som I2C-slave. Videre står en fritt til å bygge hva en ønsker. For å spare plass bygges systemet i etasjer, modulene plasseres altså over hverandre.

4.1 Modul-forslag

Her er det tatt med noe ideer til moduler som kan utvikles. Det vil senere i oppgaven bli vist en implementering av en egen kommunikasjons-modul. Her vil rammeverket bli satt for hvordan moduler kan utvikles.

4.1.1 Orienterings-estimator

Det ble vurdert å gjøre orienterings-estimeringen i en egen modul. Denne kunne hatt egen mikrokontroller, som hentet inn data fra sensorene og kjørte disse gjennom filter algoritmen. Deretter kunne hovedsystemet hentet all data om orientering helt uten å måtte bruke tid på dette. Siden dette ville kunne skape mye problemer ble det valgt å la alt dette være på hovedkortet. Dette vil heller kunne gjøres i en senere implementering.

4.1.2 GPS/INU

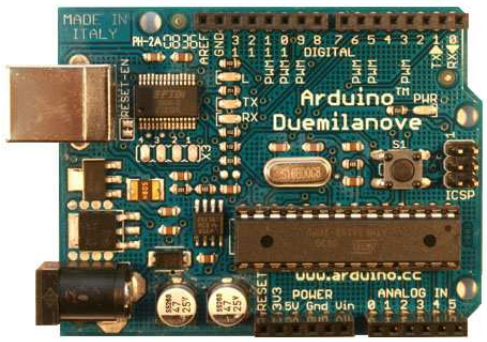
En annen interessant ide er å bygge en INS(Inertial Navigation System)-modul. Hvordan en slik kunne fungert vil bli gjennomgått i et eget kapittel, men aldri implementert. Implementerer en denne som en modul kan den beregne posisjonen til helikopteret. Modulen kan videre bruke en regulator, som forteller master-modulen hvor mye orienteringen til helikopteret må endres for å komme seg til den ønskede posisjonen. Denne passer også godt for en senere implementering.

5 Valg av hjerne

Valget av "hjerne"(mikrokontroller) til helikopteret var et sentralt tema i oppgaven, siden det er denne som skal gjøre alt tungarbeidet. Et viktig element for å gjøre helikopteret stabilt er at sampleraten blir høy nok. Den må oppfatte endringer i orienteringen på et tidlig tidspunkt for å kunne oppdatere motor-pådraget raskt. Siden prosjektet bygges på budsjett må den også være billig. Mikrokontrolleren må kunne hente inn ny data fra sensorene, behandle denne, for deretter å kjøre softsensor-algoritmen. Den må videre bruke denne informasjonen til å regulere orienteringen til helikopteret. På samme tid skal data kunne hentes ut fra systemet. For å kunne bruke denne informasjonen til simulering og videre utvikling på datamaskinen. Må den hentes ut i hvert tidssteg. Dette blir relativt mye data og det stiller derfor strengere krav til mikrokontrolleren.

5.1 Vanlig mikrokontroller

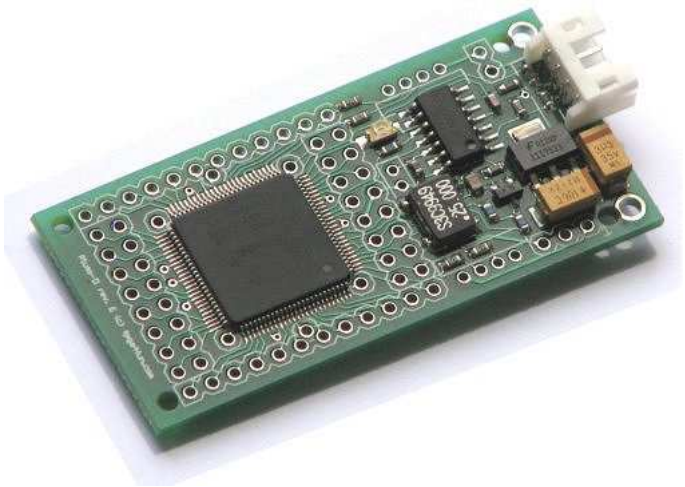
Det mest nærliggende valget var å bruke en vanlig mikrokontroller. Disse er laget for å gjøre en oppgave(kjøre ett program). Men det er mulig å få et tilnærmet "multitasking" system ved å bruke kreative avbrudd. Og passe på at systemet ikke blir hengende med en oppgaven. Dette er også en svært billig løsning og nyere mikrokontrollere kan være svært raske. Det ble valgt å bruke et utviklingskort for å slippe problemene som egne kort ofte fører med seg. Siden referanseprosjektet, [2] bruker Arduino, [3]. Hadde det vært interessant å bruke denne også i denne oppgaven. Arduino er egentlig laget for hobbybruk og har som nevnt tidligere et eget utviklings rammeverk. Hvor all lavnivå-koden er skrevet av andre. Dette gjør det lett å utvikle en rask prototype, men ikke fleksibelt nok til å desine programmet helt etter kravene. Men ved å bruke en egen flash programmerer, er det mulig å laste sin egen kode direkte inn. Denne koden kan da skrives dirkete i c. Mikrokontrolleren som står på kortet er Atmega328,[4]. Denne er satt opp til å kjøre på 16mhz, som er relativt bra etter dagens standard. Den har også 32 kb flash-minne, altså god plass til programmet. Arduino er laget for å feste såkalte "shields" på toppen. Dette er egne kretskort. En kan da utvide utviklingskorte med egne kretser.



Figur 5.1: Bilde er hentet fra, [3]

5.2 FPGA

Et alternativ til å bruke mikrokontroller, var å bruke en FPGA (Field Programmable Gate Array). Her kan hele systemet implementere som logiske blokker. For å gjøre implementasjonen enklest mulig kan dette implementeres i HDL (Hardware Description Languages). De vanligste er Verilog og VHDL. Her designes systemet i et språk som ikke er for ulikt c. Fordelen med å gjøre denne implementeringen er at systemet blir utrolig raskt. En kan også kjøre flere oppgaver i parallell. En kan foreksempel kjøre tre regulatorer på samme tid. Et eksempel på en slik implementasjon kan ses i [22]. Et slik system vil bli noe dyrere enn å bruke mikrokontroller, men det finnes også her blillige utviklingskort. Et eksempel kan være et av pluto kortene fra [16]. Disse koster ikke mer enn 39.95 usd for den billigste typen. Det finnes også gratis versjoner av utviklings verktøyet som brukes for denne FPGA-en.

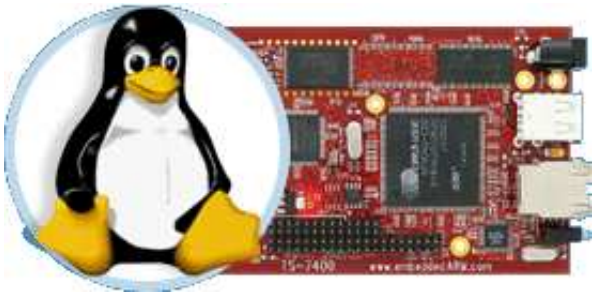


Figur 5.2: Bilde er hentet fra, [16]

Problemet med denne typen design er at det er svært komplisert. Mens prosjektet er i prototype-fasen gjøres det svært mange radikale endringer. Og ting skal testes opp mot hverandre. Hele oppsette med derfor bygges om flere ganger. Dette kan derfor i stede være en god oppgave for videre utvikling. Et alternativ til å implementere hele systemet. Kan være å utvikle dette som en modul. Denne kan da foreksempel ta seg av regulering og motor styring (ESC). En kunne da gitt måle og referansedata direkte til denne modulen ved hjelp av I2C. Problemet med trege hastighets-kontrollere blir på denne måten også løst.

5.3 Sanntids linux

Her brukes en kraftig mikrokontroller, med mulighet for å kjøre et sanntids-operativsystem. Dette kan foreksempel være en spesiell versjon av linux. Dette gjør at den opererer som en vanlig datamaskin og kan kjøre flere oppgaver på samme tid. Mange av muligheten i linux kan da også brukes, som wlan-kontroll og video-prosessering. Det vil altså bli som å plassere en datamaskin på helikopteret.
Et eksempel på mikrokontroller kan være ARM9, denne kan kjøre en sanntid linux-kjerne. Og kan ha en klokkefrekvens på 250 Mhz. Som en ser er dette en helt annen verden enn vanlige mikrokontrollere.



Figur 5.3: Bilde er hentet fra, [28]

Bakdelen med denne er at det er en dyr løsning, TS-4500 koster 92 USD fra [28]. Altså like mye som hele helikopteret. Hvis systemet senere skal utvides med flere krevende funksjoner, foreksempel video-prosessering. Vil dette være en mer gunstig løsning.

5.4 Valg

Etter å ha gått gjennom mulighetene over, ser en at det mest logiske valget er å gå for en vanlig mikrokontroller. Denne har alle mulighetene som trengs i prosjektet og er i tillegg den billigste løsningen. Det ble derfor valgt å kjøpe et Arduino-kort.

6 Måleinstrumenter

For å kunne stabilisere helikopteret, trenger en informasjon om orienteringen. Det trengs altså en IMU (Inertial Measurement Unit), denne består ofte av flere sensorer. Den brukes for å gi en nøyaktig måling av orienteringen til helikopteret. I store prosjekter kan dette gjøres ved hjelp av svært dyre og tunge måleinstrumenter. Utfordringen vil her være å finne orienteringen ved hjelp av billige og lette sensorer. Disse er kjent for å være sterkt påvirket av bias-drift og støy. Siden helikopteret er så lite, blir dynamikken svært rask og sensorene må være raske nok til å oppfatte dette. Løftekraften er også begrenset, så lav vekt er et viktig krav. Til slutt må også prisen være lav. Det er ønskelig å måle både vinkel og vinkelhastighet siden vinkelhastigheten kan brukes av regulatoren. På denne måten trengs ingen derivasjon. Det vil under bli gjennomgodt forskjellige metoder for å beregne orienteringen ved hjelp av forskjellige måleinstrumenter.

6.1 Differensiell GPS

Denne metoden går ut på å ha flere GPS-mottakere monter forskjellige steder på helikopteret. En får da informasjon om høydeforskjell og posisjons forskjell mellom de forskjellige GPS-mottakerne. Dette kan videre brukes til å beregne pitch, roll og yaw. Dette er den eneste metoden hvor alle frihetsgradene kan beregnes. Denne metoden brukes blant annet på helikopter som reparerer høyspentlinjer i USA. Problemet er her at helikopteret er så lite, en kan ikke regne med å få bedre nøyaktighet enn en meter på GPS-mottakeren. Når helikopter kroppen ikke er større enn 0.5 x 0.5 m gir dette alt for dårlig presisjon. Og er derfor utelukket for vårt formål. Metoden kom fram etter samtaler med Hans Gunnar Richardsen.

6.2 Termiske sensorer (IR)

Denne metoden går ut på å ha sensor-par som måler temperaturen. Ved å la disse peke i motsatt retning kan en utnytte temperatur-differansen mellom himmelen og jorden. Himmelen vil gi kald stråling mens jorden stråler en noe høyere temperatur. Ved å måle denne differansen kan en beregne både roll og pitch med relativt god nøyaktighet. Denne metoden har sine svakheter i at den vil kunne gi feil resultater når terrenget endrer seg. Denne vil også være umulig å bruke innendørs og alle steder som ikke har fri sikt til himmelen. Et eksempel på mulige sensorer er MLX90247ESF-DSA-ND fra digikey.com 17,12 usd per stk. Metoden er hentet fra [21]

6.3 Gyroskop

Et gyroskop brukes for å bestemme vinkel eller vinkelhastigheten. Det er nettopp dette som skal bestemmes i denne oppgaven. Her finnes det svært mange typer å velge mellom. De vanligste vil bli gjennomgodt. Et problem som er felles for alle typene er at de lider av bias-drift. At det er bias-drift betyr at det er en sakte varierende bias, som legger seg til målingene. Denne kommer av støy og temperatur endringer. Det er vanskelig å forutsi hvordan den vil utvikle seg. Dette vil si at vinkelen vil begynne å drifte vekk fra null selv om helikopteret står helt stille. Målingen vil altså bli upålitelige over tid. En kan best se dette ut fra målelikningen, se ligning 6.1. Denne delen av oppgaven bygger på data fra [41]

$$\omega_m = \omega + b + \mu \tag{6.1}$$

ω er sann vinkelhastighet. μ er gausisk målestøy. b er en sakte varierende bias (ikke-stokastisk). Og ω_m er målt vinkelhastighet. Biasen kommer av temperatur effekter og vibrasjon.

6.3.1 Mekanisk

Den eldste mekaniske typen er den mest kjente. Denne har et roterende hjul som er hengt opp slik at det kan bevege seg fritt. Her brukes prinsippet om bevaring av bevegelsesenergi. Og dette hjulet motvirker derfor bevegelse og ønsker å holde samme stilling som det har. Rammen som hjulet henger i vil derimot bevege seg med fly-kroppen. Hvis en da måle forskjellen mellom rammen og hjulet får en vinkel. Denne typen gyro måler altså vinkler i motsetning til mer moderne typer som måler vinkelhastighet. Denne typen ble blant annet brukt i v2 raketten under 2.verdenskrig. Her var også problemet bias-drift. Raketten hadde derfor en unøyaktighet på størrelse med en by. Som var akkurat det som trengtes for å treffe London. Dette var på den tiden svært bra, men ikke bra nok for denne oppgaven. Denne typen gyro inneholder mange mekaniske deler. Og friksjon gjør at den er sterkt utsatt for bias-drift. Den er også stor og tung og derfor helt utelukket til våre formål.

6.3.2 Optisk gyro

Her finnes det to hovedtyper, fiberoptisk gyro (FOG) og Ring laser gyro (RLG). Begge opererer på samme prinsippet. En lysstråle splittes, deretter sendes de to strålene i motsatt retning rundt en ring. Hvis systemet roterer vil den lysstrålen som går samme vei som rotasjonen oppleve en lengre vei enn den andre. Dette kalles Sagnac effekten. Når dette måles er det mulig og regne ut vinkelhastigheten. Denne typen gyro kalles også et Sagnac interferometer, [26]. Denne typen har ingen bevegelige deler og er svært nøyaktige. Bias-drift er ned mot 0.0035 $^\circ/h$ Problemet med denne er at slike gyroer er ekstremt dyre. Denne typen brukes blant annet i F16 og er ikke funnet til salg for privat personer. GG1308 skal være av de billigste. Vekten er også relativt høy 454g for GG1308. Selv om denne ville løst svært mange problemer i oppgaven er den dessverre utelukket på grunn av tilgjengelighet og pris.

6.3.3 MEMS-gyro

Den desidert billigste løsningen er å bruke MEMS-gyroer (Micro Electrical Mechanical System). Dette er gyroskoper som er implementert på en IC. Disse bruker Coriolis-effekten til å beregne vinkelhastighet. Inne i IC-en vibrerer et objekt i en bestemt retning. Når gyroen roteres, vil det oppstå vibrasjon som står vinkelrett på den første vibrasjonen på grunn av Coriolis-effekten. Ved å måle denne vibrasjonen kan vinkelhastigheten beregnes. MEMS-sensorer er ikke i nærheten av presisjonen til optiske, men det er antatt at den kan være det i fremtiden. Disse er svært billige og vekten er svært lav. Denne har større problemer med bias-drift enn den optiske-gyroen. Den er vanligvis opp mot $70^\circ/h$.

6.4 Akselerometer

Akselerometer brukes som navnet tilsier til å måle akselerasjon. Denne typen sensor er i det siste blitt svært små og billige. Ved hjelp av et tre-aksers akselerometer hvor aksene står ortogonalt på hverandre får en all akselerasjons-data om helikopteret. I denne oppgaven er det hovedsakelig orientering en er ute etter. For å finne denne bruker en at tyngdeakselerasjonen er en tilnærmet konstant loddrett kraft på 9.81 m/s^2. Hvis en da lar disse tre målingene danne et vektorpunkt i rommet kan en bruke vektorteori til å finne vinkelen. Er foreksempel X og Y målingen lik null og Z lik 9.81 vil denne vektoren peke rett ned. En vet da at roll og pitch er lik null. Måler en derimot Z og X lik null mens Y blir 9.81 vil det bety at roll er 90 grader og pitch er null.
Et av det store problemene med bruk av akselerometer til orienterings-bestemmelse. Er at det finnes mange forstyrrende-elementer, som vil gi utslag på akselerometeret. Dette er elementer som vibrasjon fra propellene og sentripetalkraft. For å bruke gravitasjons-vektoren er det ønskelig å skille mellom systemets akselerasjon og denne. Dette kan en se i måleligningen, ligning 6.2

$$a_m = g + a + v \tag{6.2}$$

Her er g gravitasjons-vektoren, a er sann akselerasjon for systemet og v er gaussisk målestøy. a_m er målt akselerasjon.
For dette systemet finnes det ingen modell som er god nok til å skille mellom akselerasjonen til systemet og tyngdeakselerasjonen. Men siden systemet har som hovedoppgave å hovre er det en god tilnærming og si at akselerasjonen i gjennomsnitt vil være lik null. En kan altså si at ved lave-frekvenser er den målte akselerasjonen tilnærmet lik tyngdeakselerasjonen. Dette siden helikopteret hovedsakelig skal brukes som en luft-plattform. Akselerometer gir oss altså informasjon nok til å beregne roll og pitch ved lave frekvenser. Det vil også gi akselerasjons-informasjons som senere kan brukes til navigering.

6.5 Magnetometer

Magnetometer brukes for å måle retningen på et magnetfelt. Jorden omgis som kjent av magnetfelt. Ved å måle retningen på dette i forhold til helikopteret, kan en beregne hva som er magnetisk nord. Det kan altså brukes som en kompass-modul. Honeywell har designet en ferdig modul som bruker et to aksers magnetometer til å bestemme magnetisk nord HMC6352, [ref]. Denne bruker I2C for å kommunisere med mikrokontrolleren å kan dermed kobles direkte til I2C bussen. Dette vil da gi en metode for å finne retningen helikopteret flyr. En har fra før denne informasjonen fra gyroen, men som nevnt tidligere vil denne drifte over tid. For å kunne holde retningen nøyaktig trenger en denne informasjonen. På grunn av de fire motorene som omgir helikopteret vil det kunne oppstå problemer ved bruk av magnetometer. Disse inneholder store magneter som vil kunne gi feil resultater på kompasset. Det er i denne oppgaven valgt å ikke implementere magnetometer, siden det ikke er nødvendig for å holde helikopteret stabilt i luften. Om retningen drifter litt har ikke mye å si, siden hovedmålet er at helikopteret skal kunne hovre. For å gjøre dette trenger en bare roll og pitch.

6.6 Valg

Med bakgrunn i informasjonen over ble det valgt å bruke MEMS-gyro på grunn av pris, tilgjengelighet, lavt strømtrekk og vekt. Disse leverer vinkelhastighet som trengtes for regulatorene. Siden disse lider av bias-drift ble det også valgt å bruke akselerometer siden disse ikke lider av drift. Dette gir oss to målinger hvor den ene oppveier for problemene til den andre. Det vil senere i kapittelet bli gjennomgodt hvordan disse målingene kan kombineres.

7 Implementering av sensorer

Det vil i denne delen bli gjennomgodt hvilke fysiske sensorer som er blitt valgt og implementeringen av disse. Valgene er også her hovedsakelig tatt etter pris og vekt.

7.1 Gryo

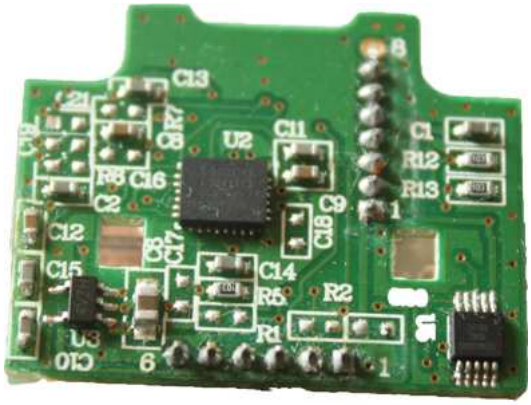
MEMS-gyroene leveres i svært små pakninger og lodding er derfor svært vanskelig. Det var derfor ønskelig med et utviklingskort. For å kunne detektere alle de tre aksene trengtes tre gyroer. Etter tips på et forum viste deg seg at Wii MotionPuls fra Nintendo inneholdt tre gyroer. Dette er et tillegg som gjør Nintendos Wii kontroller bedre. Dette tillegget har mulighet for å måle vinkelhastighet på alle tre aksene. Det er akkurat hva som trengs i denne oppgaven. Den har også innebygd 14-bit A/D omformer, sik at data kan hentes ut via I2C. Dette blir ikke en løsning som passer for masseproduksjon, men en perfekt løsning for et gjør det selv prosjekt. Siden denne er billig og enkel å få tak i for privat personer.
Problemet med denne er at det ikke finnes datablad. Den er ikke laget for denne bruken og Nintendo har derfor ikke gitt ut noen form for spesifikasjoner. Det er derfor vanskelig å si noe om presisjonen. Men siden den er så billig blir den likevel kjøpt. Skulle den vise seg å være for dårlig for oppgaven kan den byttes ut. Enheten kan ses i figuren under.

Figur 7.1: Wii MotionPlus

For å oppnå det samme kunne en foreksempel bruk disse produktene fra Sparkfun.
$http://www.sparkfun.com/commerce/product_info.php?products_id = 9410(39.95usd)$
$http://www.sparkfun.com/commerce/product_info.php?products_id = 9165(19.95usd)$

Dette gir oss en samlet pris på 59.9 USD før frakt og toll. En billig kopi av Wii MotionPuls ble funnet hos Dealextreme.com. Denne koster ikke mer enn 15.03 USD med frakt. Det var altså en betydelig prisforskjell.

WM+ ble demontert og inneholdt et svært lite kretskort. På dette fantes det to gyro IC-er. Den ene var IDG600, som måler både roll og pitch. Mens den andre var X3500W som brukes for å måle yaw. Det er ikke funnet datablad på noen av disse. De er nokså sikkert spesial designet for Nintendo. I stede er databladet for IDG650 blitt brukt, [14]. Litt enkelt lodding ga tilgang til I2C klokke og data. Kortet opererer på 3.3v.



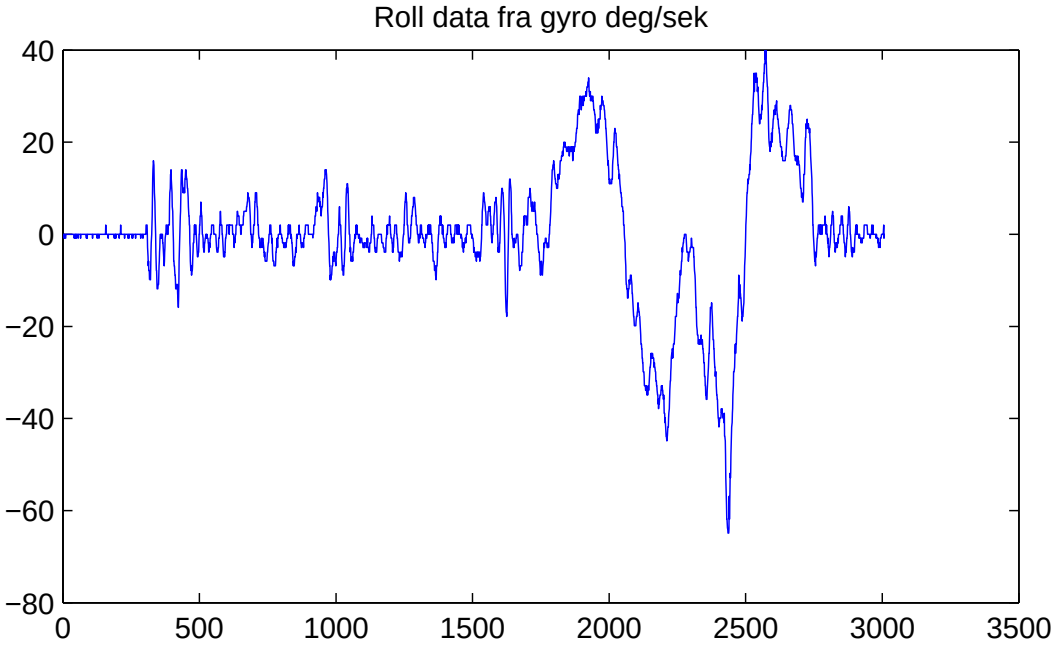
Figur 7.2: Kretskortet funnet i MotionPlus

For å teste kretsen ble det designet et lite grensesnittkort med en spenningsnivå omformer. For å gjøre omformingen ble TXS0104EDR,[29] brukt. Dette er en IC, som konverterte fra 3.3v logikk til 5v logikk. Og WM+ kunne dermed enkelt kobles til mikrokontrolleren for testing. Denne omformingskretsen ble senere implementert på samme måte på kortet. I2C var allerede implementert for å kunne kommunisere mellom modulene og kortet ble derfor koblet direkte inn på denne bussen.
Neste steg var å dekode protokollen. WM+ er ikke beregnet for denne typen bruk og protokollen er derfor ikke gjort tilgjengelig av Nintendo. Denne informasjonen måtte derfor finnes ved hjelp av en logisk-analyse. Dette var allerede gjort av andre og all informasjon om WM+ og andre Wii kontrollere ble funnet på http://wiibrew.org/.
Kommunikasjonen skjer som nevnt over ved hjelp av I2C. For å aktivere WM+ sendes først 0xFE04 til adresse 0x53. WM+ vil da skifte adresse til 0x52. For å lese av gyroene sendes nå 0x00 til adresse 0x52 og 6 byte kan deretter leses av. Disse 6 bytene inneholder informasjon om de tre vinkelhastigheten. De inneholder også informasjon om skaleringen på målingen. Gyroene har nemlig to skaleringer, en for å måle raske bevegelser(2000deg/sek) og en for å måle sakte(500 deg/sek) mer nøyaktig. Det er viktig at programmet også henter ut denne for å bruke rett skalering. Det trengs mer enn 1 byte for å representere vinkelhastigheten. Dette er gjort med 14 bit, det må en dekoding til for å få informasjonen ut fra de 6 bitene.

Tabell under viser en oversikt over hvordan informasjonen hentes ut. Tabellen er hentet fra wiibrew.org. Nede til høyre i tabellen kan en se f/s. Dette bitet forteller hvilken skalering målingen er gjort med (rask eller sakte).

	bit							
byte	7	6	5	4	3	2	1	0
0	yaw<7:0>							
1	roll<7:0>							
2	pitch<7:0>							
3	yaw<13:8>						yaw f/s	pitch f/s
4	roll<13:8>						roll f/s	ext.
5	pitch<13:8>						1	0

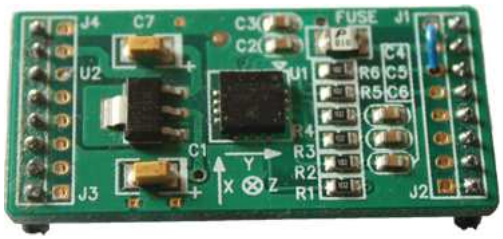
Dette ble implementer på mikrokontrolleren å data ble hentet ut ved hjelp av USB. Disse dataene ble så hentet inn til Matlab for videre behandling. Figure 7.3 viser et plott av roll-data sentrert om null. Helikopteret har her blitt holdt i handen, mens det roteres 90 grader i alle retninger. Støyen på signalet kommer av at det er vanskelig å holde helikopteret helt stille. Det kan tydelig ses fra plottet når roll gjøres. X-aksen er antall sampler.



Figur 7.3: Plott av roll data

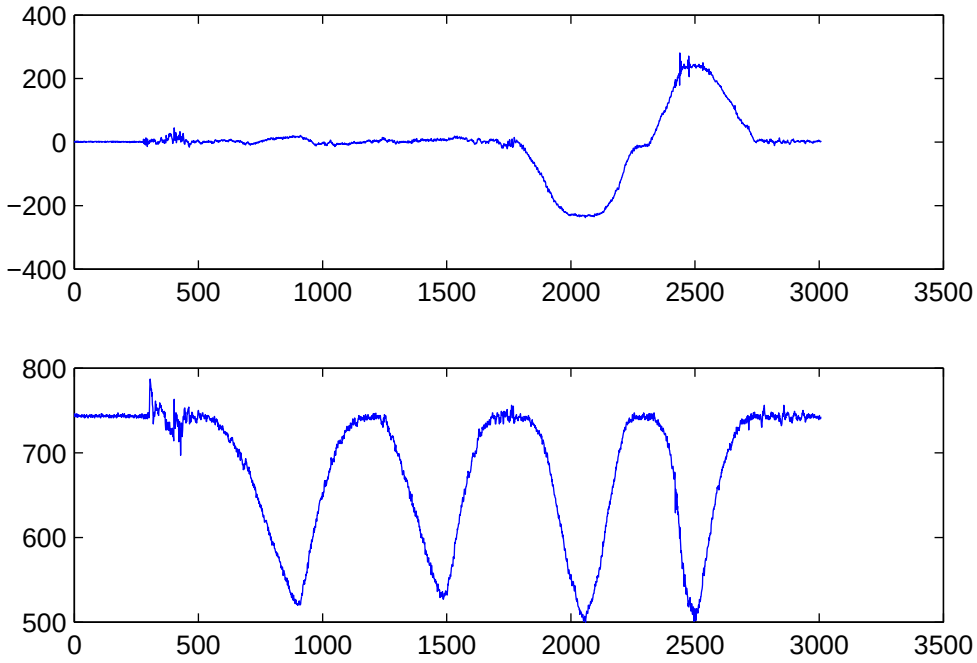
7.2 Akselerometer

Det var allerede før master-oppgaven ble påbegynt kjøpt et akselerometer for eksperimentering. Dette ble kjøpt fra Sure-Electronics,[27] på ebay. Akselerometeret var allerede montert på et utviklingskort og og var dermed enkelt å implementere. Akselerometeret var MMA7260, [19]. Dette har høy sensitivitet 800mV/g som gjør at det ikke har problemer med å måle tyngdeakselerasjon. Det har også innebygd lavpass-filter og temperatur-kompensering. Det bør derfor passe godt til oppgaven.
Akselerometeret gir ut akselerasjonen som tre analoge spenninger. Det ble derfor koblet til de analoge inngangene på mikrokontrolleren. Det finnes bare en A/D omformer på mikrokontrolleren, den leser derfor av aksene en etter en. Dette gir noe unøyaktighet med tanke på at helikopteret kan ha flyttet seg på den tiden dette tar. Disse verdiene leses heller ikke av på samme tid som gyroen. Det er valgt å se bort fra dette i denne oppgaven siden dette skjer så raskt at det bør være neglisjerbart. I følge databladet til mikrokontrolleren, [4] tar en avlesning fra 13 til 260μS. En løsning kunne eventuelt vært å bruke en ekstern A/D som kommuniserte med mikrokontrolleren over I2C. Dette ville også gjort det mulig å lese av gyro og akselerometer data på samme tid. Et forslag til dette kan være å bruke MAX1238 som er en 12 kanalers A/D omformer.



Figur 7.4: Akselerometer-kort fra Sure-Electronics

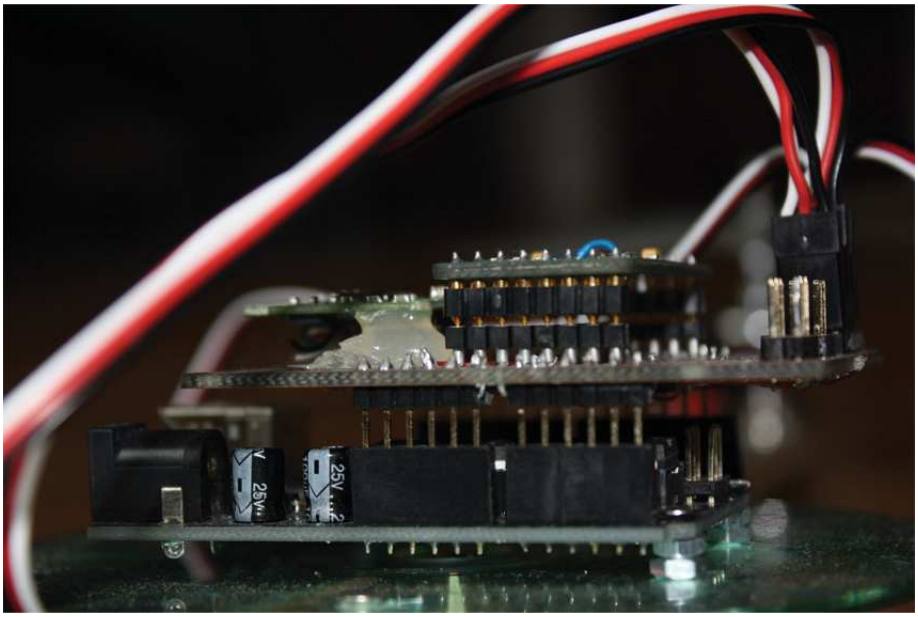
Dette ble også implementert på mikrokontrolleren. Data ble her hentet ut fra A/D kanalene. Figur 7.5 viser data fra x og z aksen på akselerometeret. Også her blir helikopter hold i handen, mens det roteres 90 grader i alle retninger. Det kan også her tydelig ses når vridningen av helikopteret skjer. Y aksen er her spenningen fra akselerometeret, representert med 10bit (800mV/g). Dette er altså akselerasjonen til systemet. X aksen er antall sampler.



Figur 7.5: Data hentet fra akselerometeret

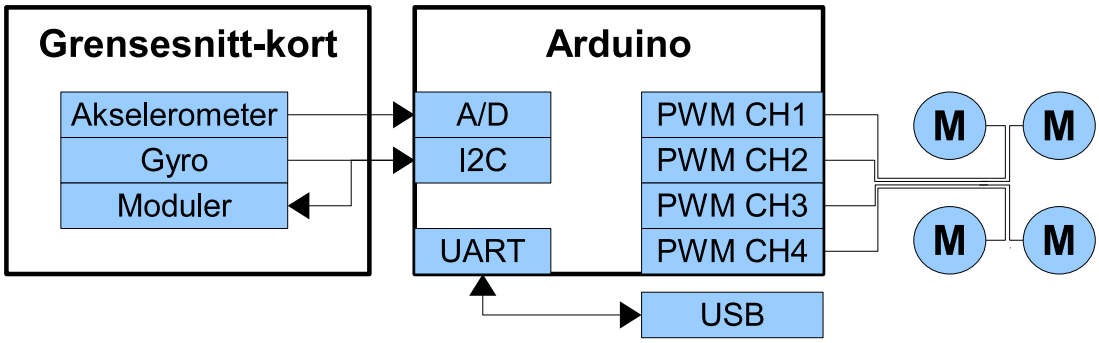
8 Kretskort

For at alle sensorene skal kunne kommunisere med mikrokontrolleren. Var det nødvendig å utvikle et grensesnitt-kort. Som kjent er det valgt å bruke en Arduino,[3] som utviklingskort. Dette gjør at kretskortet må bygges som et såkalt skjold (shield). Dette betyr at grensesnittkortet har pinner som er slik at det kan settes direkte ned i Arduinokortet, Se figure 8.1. Selve skjoldet var det allerede laget en ferdig mal på denne ble funnet hos, [17]. Dette var bare plasseringen av pinnene, resten av kortet ble designet i oppgaven.



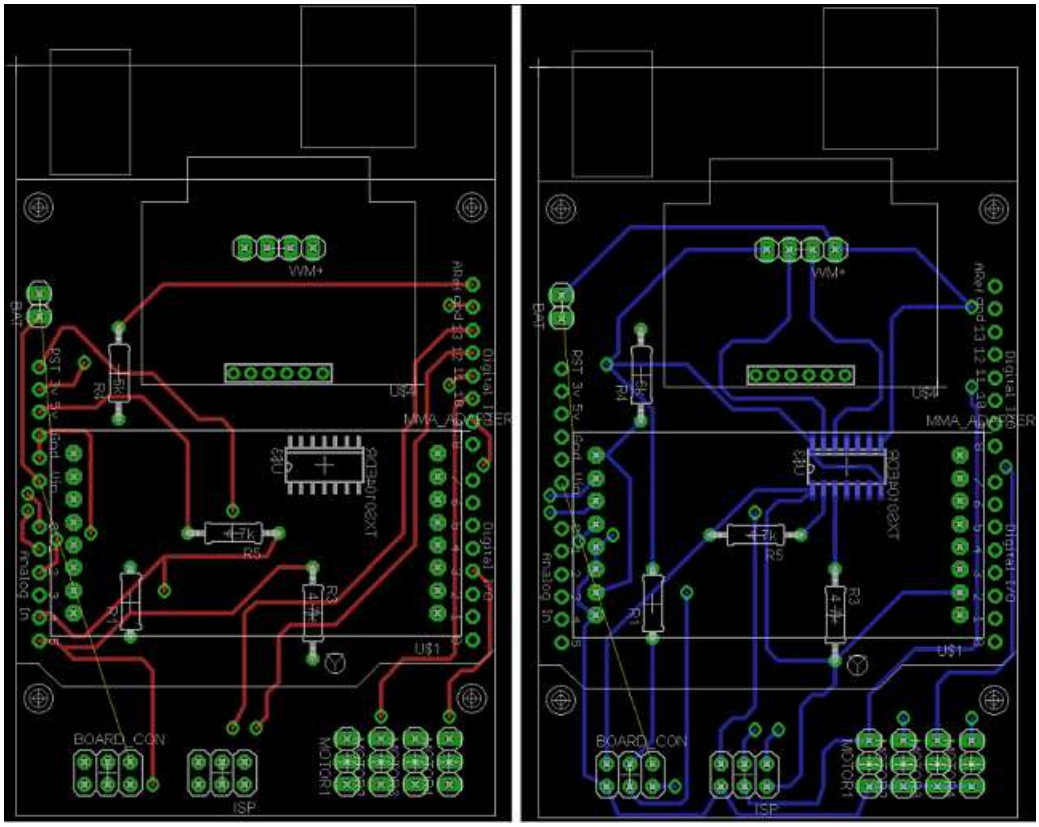
Figur 8.1: Figuren viser kortene over hverandre

Kretsen må da basere seg på skjemategningene for Arduinokortet. Disse kan ses på [3]. Hvordan alle sensorene og motorene skal kobles til, er blitt beskrevet tidligere. I tillegg er det også blitt laget en egen tilkobling for en flatkabel. Slik at de andre modulene kan kobles til her. Under er det satt opp en oversikt som viser hele systemet i ett, se figur 8.2.

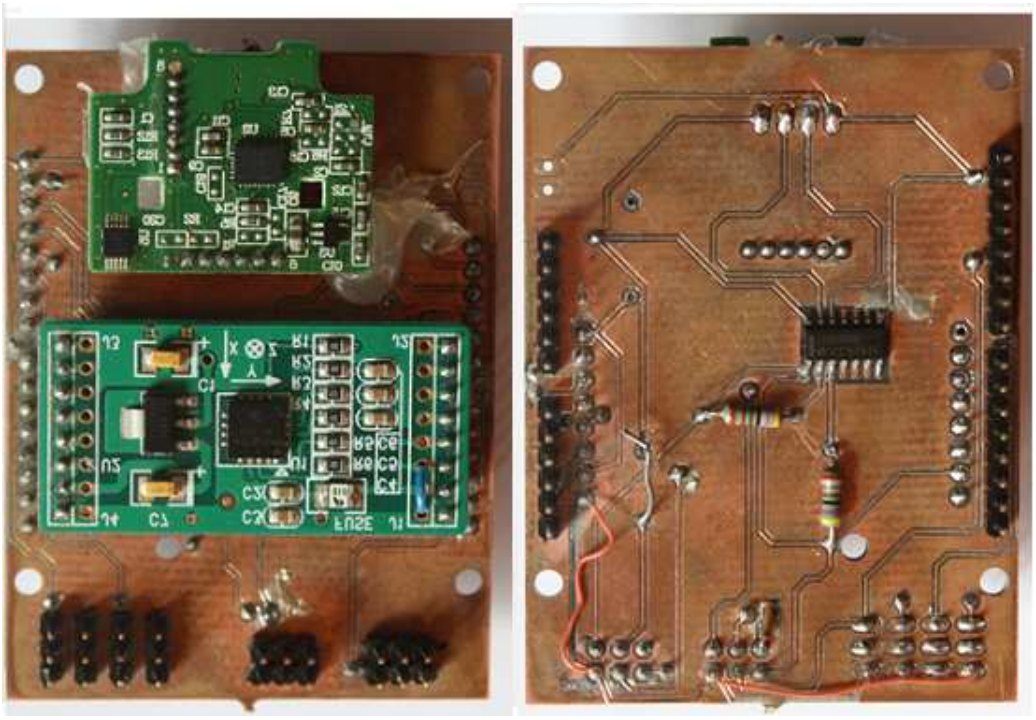


Figur 8.2: Figuren viser en oversikt over koblingene, som skal gjøres på kretskortet.

Arduinokortet har alt som trengs for å kjøre mikrokontrolleren. En kan også hente 5v og 3.3v herfra. Dette gjør at en slipper egne spenningsregulatorer på grensesnitt-kortet. Som en husker opererer gyroen på 3.3v logikk, mens Arduino kortet kjører på 5v. Dette er blitt løst med en enkel IC som oversetter fra 3.3v logikk til 5v og omvendt, TXS0104EDR,[29]. Neste steg i prosessen er nå å utvikle selve kortet. Dette ble designet ved hjelp av programmet EagleCad, [12]. Dette programmet kan brukes gratis til ikke kommersielle design. Resultatet ble som vist i figurene under.



Figur 8.3: Figuren viser kretskort-utlegget, topp og bunn

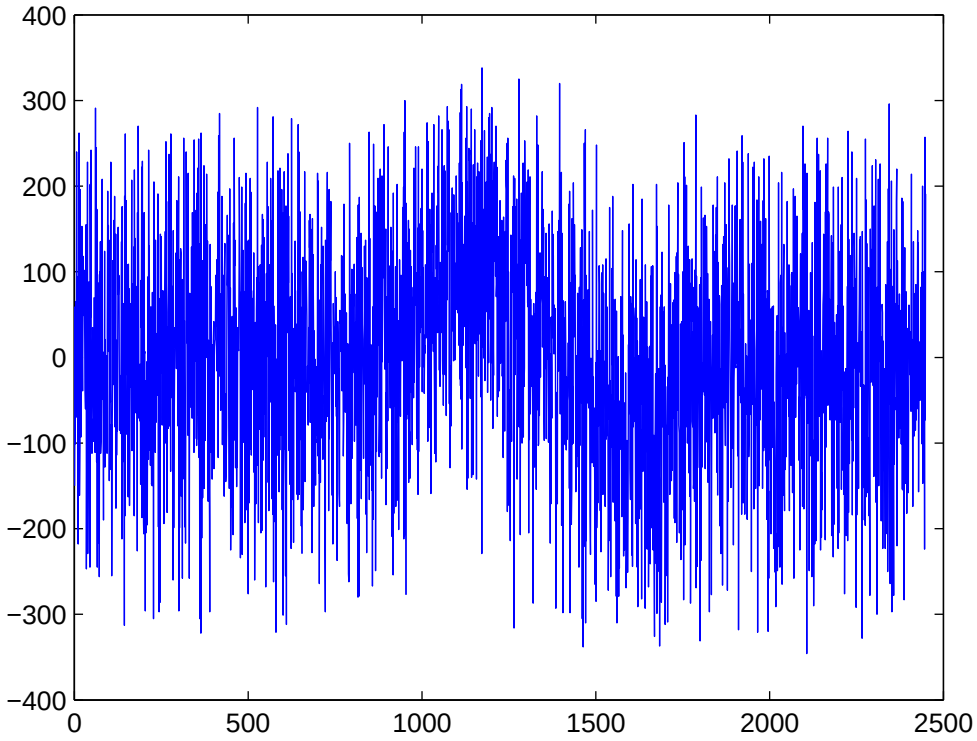


Figur 8.4: Figuren viser det ferdige kretskortet, topp og bunn

Kretsskjema kan ses i vedlegg, E. Det er blitt brukt svært mye tid på kretskortet, siden det har oppstått små kortslutninger. Disse har vist seg å være svært vanskelige å finne. Etter endel arbeid fungerer kretskortet nå svært bra.

9 Målestøy

Det var medregnet at noe målestøy ville oppstå på grunn av propellene. Det var også antatt at dette ville dempes sterkt av lavpass filteret, som var på utviklingskortet til akselerometeret. Alle testene som ble gjort før systemet var ferdig bygget viste også at dette burde gå bra. Da systemet var ferdig implementert viste det seg allikevel at støy ble et stort problem. Når alle de fire motorene går på et relativt høyt turtall ble måledata uleselige på grunn av vibrasjonen som oppstår. Dette vises tydelig i figur, 9.1.



Figur 9.1: Data hentet fra akselerometer, uten filtrering

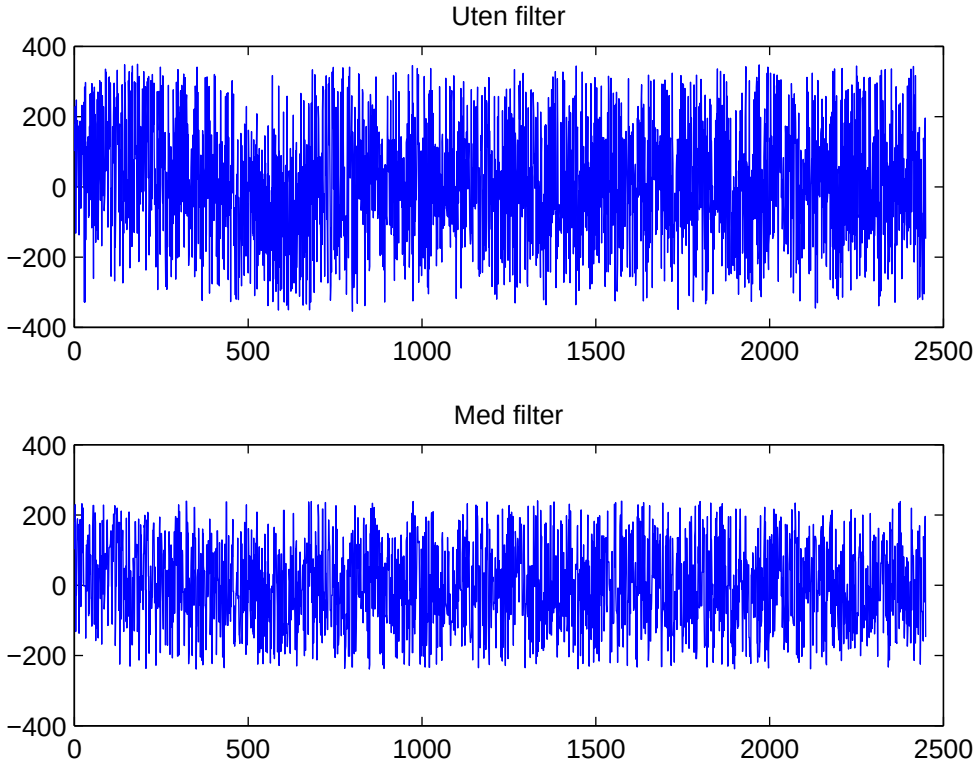
Det var spesielt akselerometeret dette ga utslag på, siden det måler vibrasjon. Flere tiltak ble gjort for å løse dette. Det ble blant annet gjort tester med bruk av skumgummiplater for demping mellom kretskortet og selve helikopterkroppen. Dette gjorde dessverre bare problemet enda større og ble derfor fjernet. Det var altså vanskelig å gjøre noe med støyen rent mekanisk. Lenge så det ut som akselerometer data skulle være helt ubrukelige. Det var også for sent å gjøre noe med elektronikken. Et digitalt filter måtte derfor implementeres. Videre vil det bli gjennomgått de forskjellige filtrene som ble testet.

9.1 Logisk filter

En svært enkel måte å fjerne noe av feilbidraget på var å lage et logisk filter. Dette er et enkelt filter, som fjerne ulogiske verdier. Det vil si at hvis akselerometeret måler verdier som er større en 9.81. Kan en trykt anta at dette ikke er tyngdeakselerasjon. Dette kan gjøres som i ligningene under.

$$-9.81 > a_m(k) > 9.81 \qquad a_m(k) = a_m(k-1)$$
$$-9.81 < a_m(k) < 9.81 \qquad a_m(k) = a_m(k)$$

Dette ble implementert ved hjelp av if setninger i Matlab og ga resultater som i figur, 9.2. Y aksen er her spenningen fra akselerometeret representert med 10bit (800mV/g). Dette er altså akselerasjonen til systemet. X aksen er antall sampler. En g er som kjent 9.81 m/s^2.

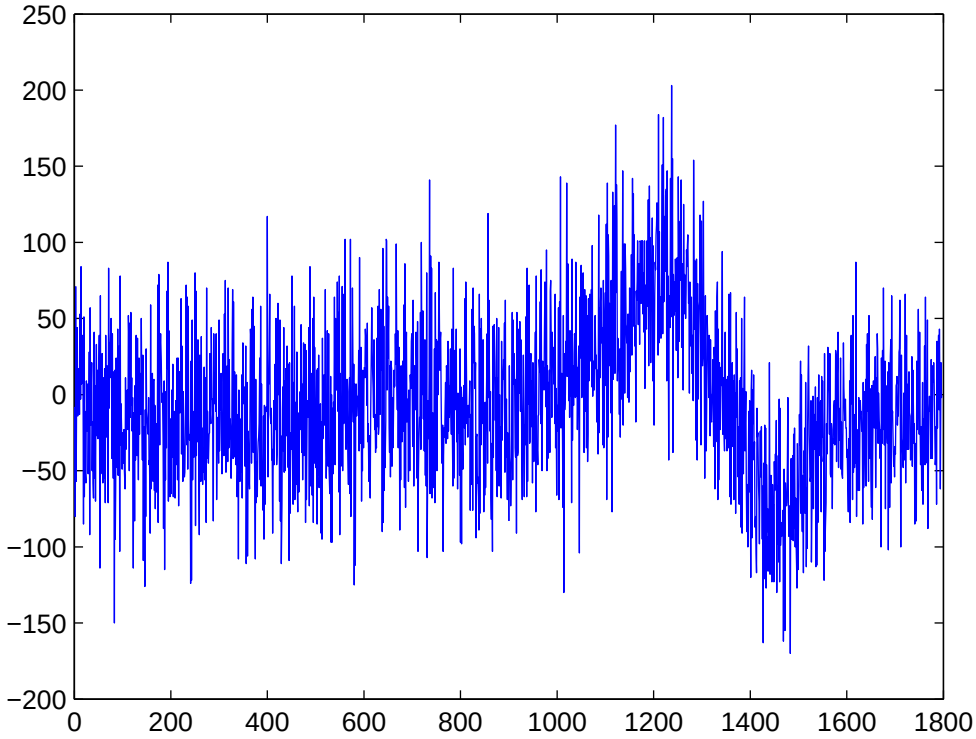


Figur 9.2: Sammenligning med og uten logisk filter

Som en kan se reduserer dette støyen noe, men resultatet blir ikke perfekt. Styrken er at det fjerner ekstremalverdier og er svært lite ressurskrevende. Dette er derfor blitt implementert på det ferdige systemet.

9.2 Gjennomsitts-filter

Dette er en annen enkel måte å få filtrert dataene på. Her kjøres målingen flere ganger, for deretter å regne et gjennomsnitt av disse. Som en husker tar en måling $13 - 260\mu S$. Det blir altså ikke store forsinkelsen. Dette bør fjerne mye av den høyfrekvente dataen. Denne metoden ble også implementert på mikrokontrolleren, målingene kjøres her åtte ganger. Dette ga resultater som i figur, 9.3.



Figur 9.3: Plot av data filtrert med gjennomsnitts-filteret.

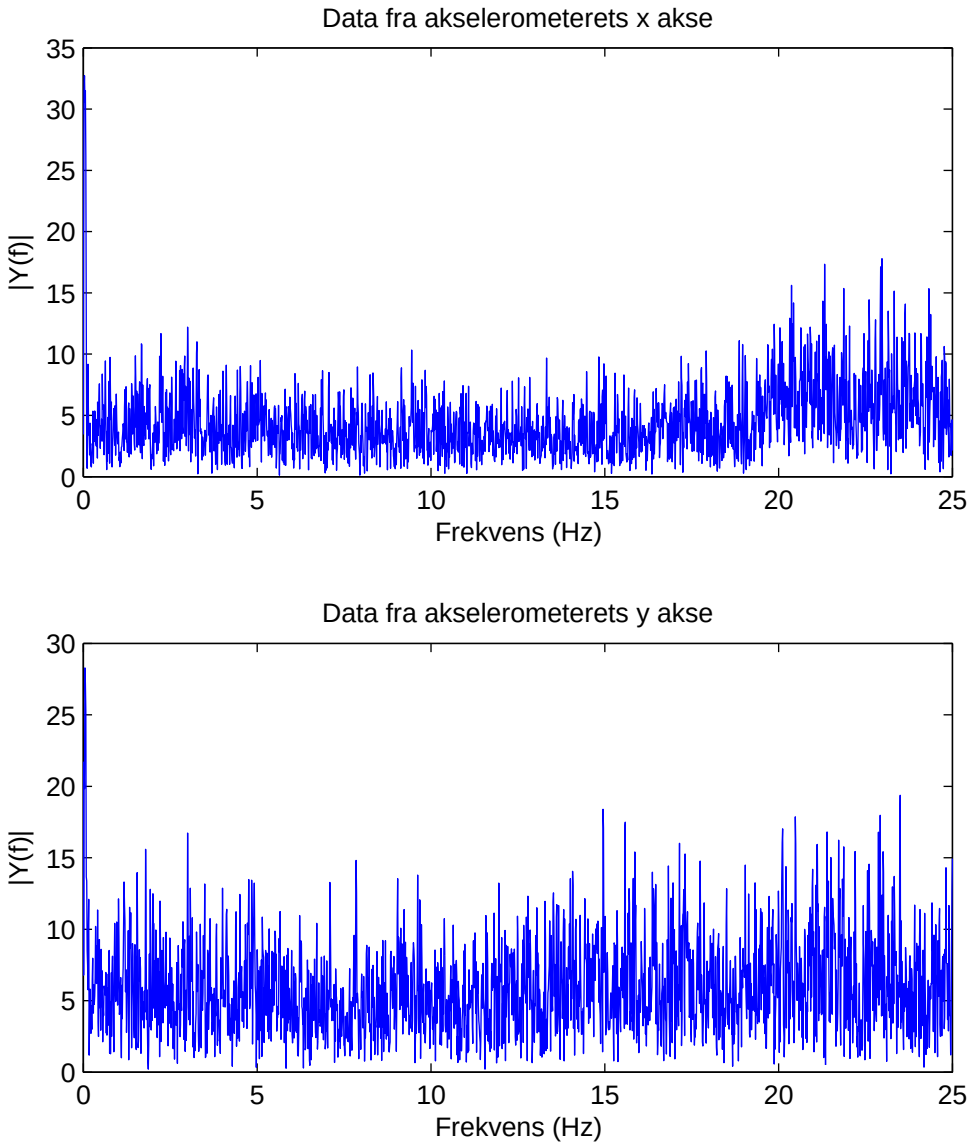
Hvis en sammenligner med figur 9.1 er det en klar forbedring, men langt fra perfekt resultat. Det er derimot en enkel metode som ikke krever svært mye ressurser.

9.3 Avansert filtrering

Siden de forrige eksperimentene ikke ga tilfredsstillende resultater, vil det i denne delen bli forsøkt en mer avansert fremgangsmåte. I stede for å implementere test-filtrene direkte på mikrokontrolleren, vil data hentes fra helikopteret og bli forsøkt filtrert i Matlab. Matlab har svært mange kraftige filtrerings-verktøy tilgjengelig. En vanlig måte å starte filterdesign på er å finne frekvensinnholdet i signalet. Dette kan en gjøre ved hjelp av FFT (Fast Fourier Transform). Det finnes en kommando i Matlab, som gjør dette. Denne er dessverre ikke helt rett frem å bruke, det måtte derfor skrives en liten kode. Denne vil nå gi ut et plot av frekvensinnholdet i signalet.

```
function plot_fft(inn, Fs) % Signal,samplerate

L = length(inn);
NFFT = 2^nextpow2(L);
Y = fft(inn,NFFT)/L;
f = Fs/2*linspace(0,1,NFFT/2+1);

%% Plot
figure
plot(f,2*abs(Y(1:NFFT/2+1)))
xlabel('Frekvens (Hz)')
ylabel('|Y(f)|')
```

Det var et håp at frekvensen på støyen skulle være direkte avhenging av frekvensen til propellene. En kunne da brukt et båndstopp-filter til å fjerne dette. Filteret kunne flyttet stopp-båndet etter frekvensen på propellene. Når FFT ble kjørt viste dette seg å være en umulighet. Som en ser fra figur, 9.4 ligger støyen sprett over hele frekvensspekteret. Det bør allikevel være mulig å fjerne mye av dette ved hjelp av et godt lavpass-filter. Neste steg blir derfor å utvikle et lavpass-filter.



Figur 9.4: FFT på signalet fra akselerometeret

Etter mye forskning ble det valgt å bruke et Butterworth-filter. Dette er et IIR-filter som gir så flat frekvensrespons som mulig i passbåndet. Dette filteret bør passe svært bra til vårt formål. Det hadde vært en fordel å implementert filteret i hardware. Dette for å slippe de ekstra beregningene, som fører til lavere samplerate. Problemet var at hardware allerede var konstruert og det var derfor for sent å endre dette. Det ble derfor valgt å utvikle et digitalt-filter. Denne delen bygger på [8], [11] og [9].

9.3.1 Digitalt filter

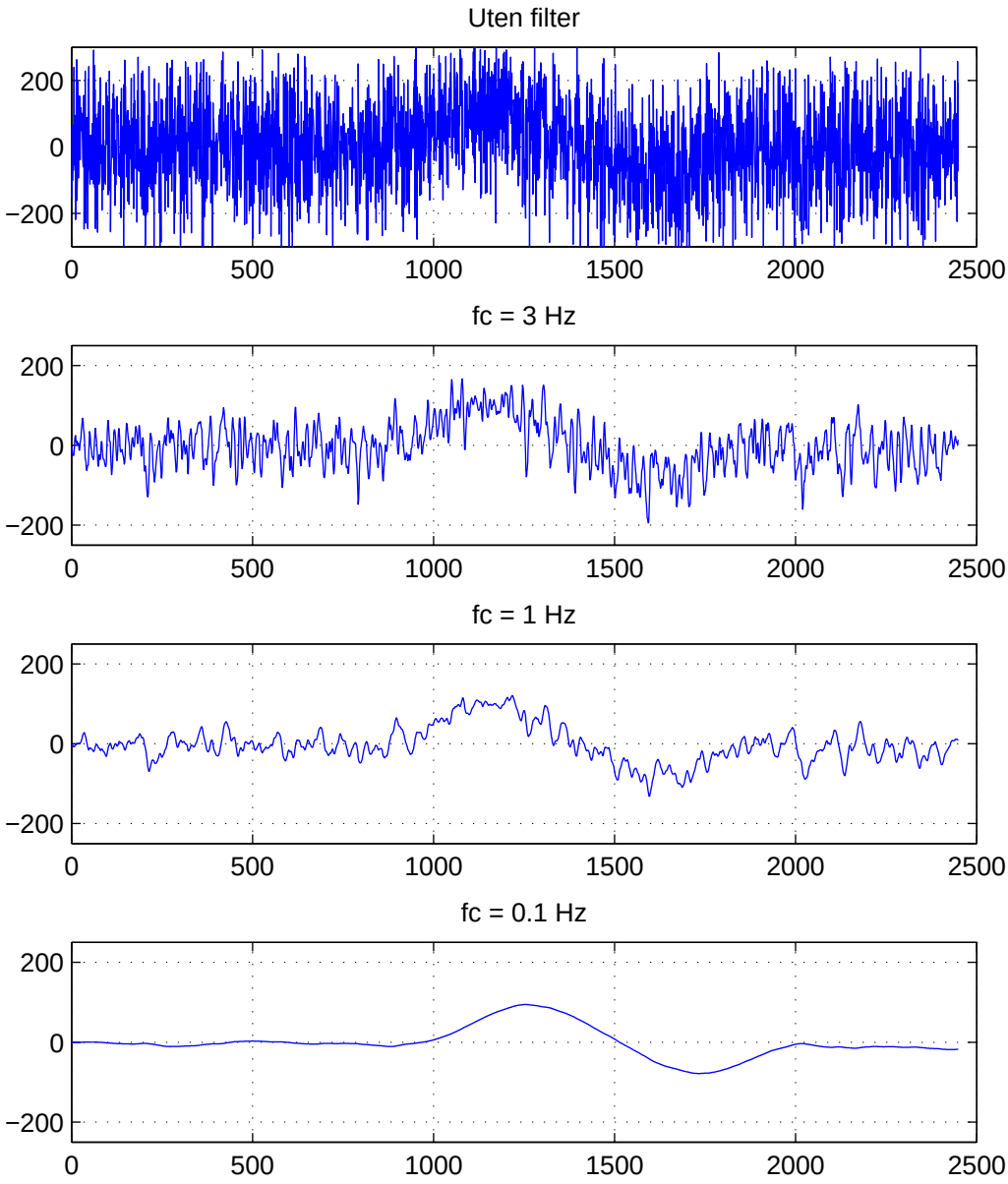
Det vil her bli designet et digitalt Butterworth-filter, som kan implementeres på mikrokontrolleren. For å designe et digitalt IIR-filter må en først finne filter-koeffisienten. Dette kan enkelt gjøres i Matlab ved hjelp av kommandoen [B,A] = butter(N,Wn). Denne gir oss koeffisientene til et n.ordens-Butterworth-lavpass-filter, med Wn som knekkfrekvens. Wn må være et tall mellom 1 og 0, hvor 1 gir en knekkfrekvens på halvparten av samplefrekvensen. Wn kan derfor regnes ut ved hjelp av ligningen under. Her er f_c knekkfrekvens(Hz) og f_s samplefrekvens(Hz).

$$W_n = \frac{2f_c}{f_s} \tag{9.1}$$

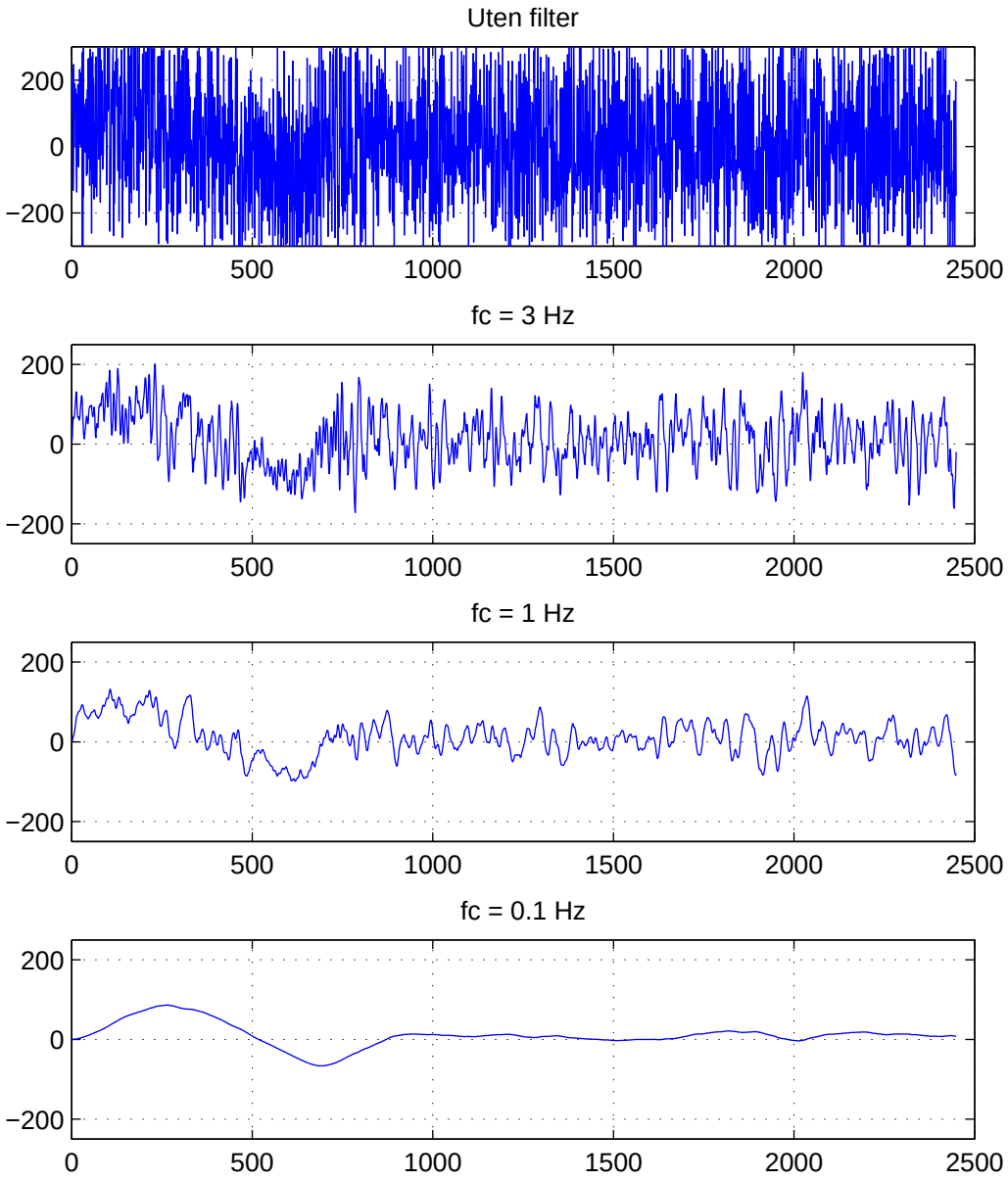
For å finne utgangen fra filteret kan en bruke den generelle differensiallinkningen for et LTI(lineært tids-invariant) filter, se ligning 9.2. En ser her at en regner ut neste sample ved å bruke tidligere sampler. Siden dette er et IIR-filter vil M være ulik null. Dette gjør at en også må ta hensyn til tidligere utgangs-sampler. Alle disse samplene må derfor lagres i minnet. Her vil den største av M og N være ordenen til filteret. Det er derfor en fordel å ha så lav orden som mulig. Dette for å gjøre filteret minst mulig ressurskrevende.

$$y[k] = \sum_{j=0}^{N} b_j \cdot x[k-j] + \sum_{i=1}^{M} (-a_i) \cdot y[k-i] \tag{9.2}$$

For å teste filteret i Malab kan en enkelt regne ut hva utgangen blir ved hjelp av kommandoen "y = filter(b,a,signal)". En kan da teste forskjellige knekkfrekvenser og orden. Filteret ble testet med forskjellige parametere. For å få en effektiv implementasjon ble 2. orden brukt. Dette gjør også at den analoge-implementeringen i neste delkapittel blir mye enklere. Resultatet vises i figurene under. Her er data hentet fra akselerometeret med motorene på et relativt høyt turtall.

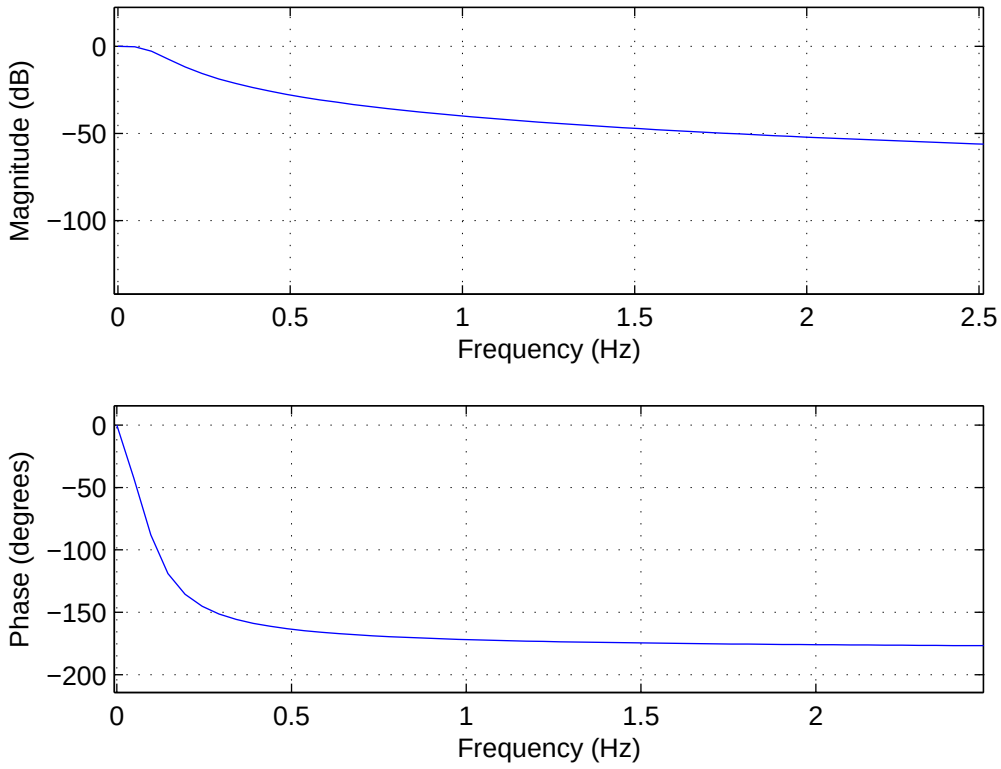


Figur 9.5: Filtrert data fra x-aksen til akselerometeret

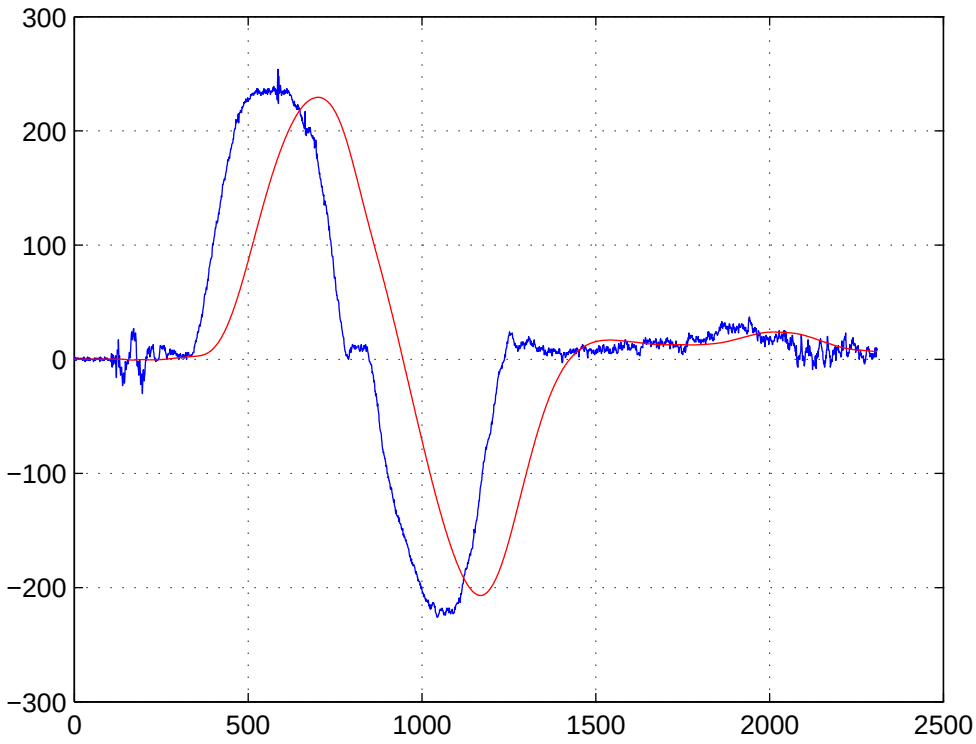


Figur 9.6: Filtrert data fra y-aksen til akselerometeret

Som en ser ut fra plottet kan en fjerne svært mye støy ved hjelp av denne typen filter. Når knekkfrekvensen kommer ned i 0.1 Hz er signalet tilnærmet perfekt. Problemet med alle typer filter er at signalet i passbåndet kan bli dempet og faseforskjøvet. Det er derfor laget et bode-plott av filteret, se figur 9.7. Her ser en at alt utenom svært lave frekvenser blir dempet og fase forskjøvet. En ønsker selvsagt at frekvensene i stopp-båndet skal dempes. Men problemer oppstår når frekvenser i passbåndet også endres. For å teste dette ble det laget et sammenlignings plot, se figur 9.8. Her er et støyfritt signal kjørt gjennom filteret. Begge signalene er så plottet i samme figur. En ser her at det er noe dempning og ganske betydelig forskyvning. Såpass lav knekkfrekvens kan derfor gi problemer. Dette gjør at når knekkfrekvensen er lav nok til å fjerne all støy blir signalet forskjøvet. En må ta hensyn til dette når en implementerer, slik at verken støy eller forskyvning skaper problemer. Atmel har et omfattende implementerings-eksempel for nettopp dette filteret, se [11]. Her følger det med kode skrevet i ASM. Denne koden kan da enkelt implementeres, og koeffisientene endres til de funnet med Matlab.



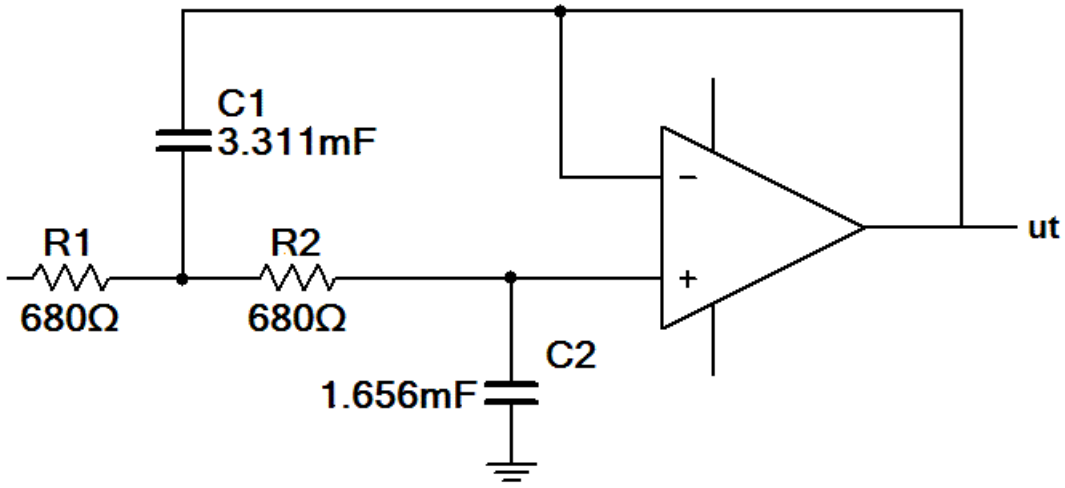
Figur 9.7: Bodeplott av filteret med knekkfrekvens 0.1 Hz



Figur 9.8: Sammeligning med og uten filter

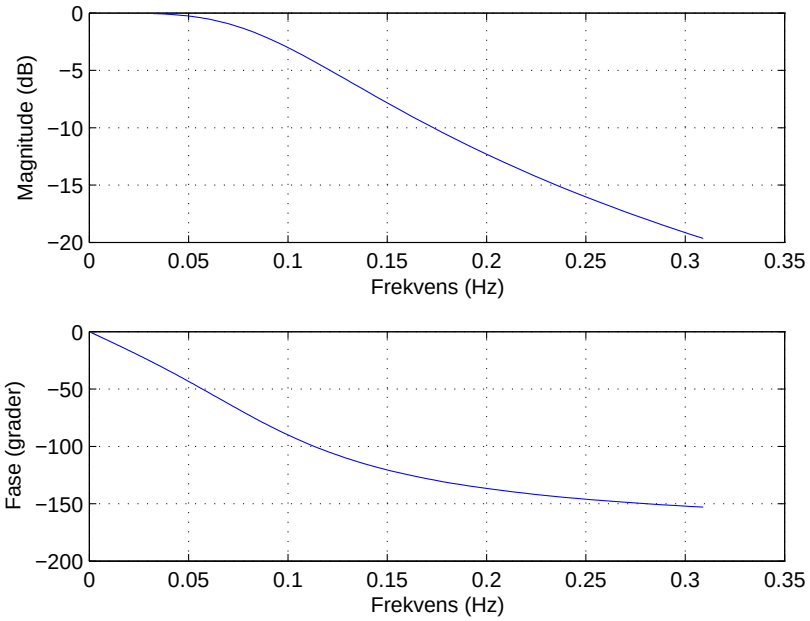
9.3.2 Analogt filter

En ser over at et digital Butterworth-filter kan gi sterke signal forbedringer. Det sammen kan en også oppnå ved hjelp av et analogt-filter. På denne måten slipper en de ekstra beregningene, som fører til lavere samplerate. Denne delen vil ikke bli implementert siden hardware allerede var ferdig da dette ble påbegynt. Det er i stede tenkt til senere implementering. Det finnes flere måter å implementere et analogt Butterworth-filter. Før var det vanlig å bare bruke passive komponenter. Dette gir et svært enkelt filter, men bruk av induktanser gir ofte lavere nøyaktighet. Siden det er vanskelig å produsere disse like. Det ble derfor valgt å gjøre en Sallen-Key implementering. Her bruker en i stedet en operasjons-forsterker. Kretsen ble utviklet og simulert i programmet Multisim, [20]. Kretsen vises i figur, 9.9. Fra simuleringen ble bode-plottet hentet ut, dette ble plottet ved hjelp av Matlab se figur 9.10.



Figur 9.9: Analogt Butterworth-filter

Verdiene på komponentene i kretsen ble beregnet ved å bruke kalkulatoren på, [9]. Her kan en legge inn filter-spesifikasjonene og enkelt designe om filteret etter behov. Filter-spesifikasjonene er de samme som i forrige delkapittel.



Figur 9.10: Bodeplot for analogt Butterworth-filter, med knekkfrekvens 0.1 Hz

Som en ser fra bode-plottet bør det være mulig å oppnå samme resultat som ved det digitale filteret. Dette filteret kan enkel implementeres mellom akselerometeret og de analoge-inngangene på mikrokontrolleren. Men som nevnt over er dette noe som må gjøres i neste versjon av kretskortet.

10 Representasjon

Har nå vinkelhastigheten for alle tre aksene. Har også nok akselerasjons-data til å beregne roll og pitch. Neste steg er å beregne orienteringen og utvikle en metode for å representere denne. Det er mest nærliggende å bruke gyro-data til å beregne orientering, for deretter å bruke akselerasjons-data som sann måling for å rette gyro-målingen. For at regulatorene skal kunne bruke gyro-dataene må det utvikles en likning, som gjør vinkelhastighet om til orientering. Det trengtes altså en ligning eller lignings-sett som tar gyro-data inn og gir orientering ut. Dette er et av temaene som har tatt opp mye tid i denne master-oppgaven. Det vil i denne delen bli gjennomgått to metoder for å representere orienteringen til helikopteret. En enkel lineær og en avansert ulineær.

10.1 Eulers-vinkler (enkel metode)

Den enkleste og mest intuitive metoden er en beskrivelse ved hjelp av Eulers-vinkler roll, pitch og yaw. Her behandles hver av aksene hver for seg gjennom hele prosessen. Problemstillingen er å lage tre likninger som tar inn vinkelhastighet fra gyroen og gir ut orienteringen til helikopteret.
Som kjent er den deriverte av vinkelen med hensyn på tid, vinkelhastighet, se ligningene under. For å finne vinkel fra vinkelhastighet kan en da integrerer utgangen fra gyroen. Dette gir tre enkle og lineære ligninger som beskriver orienteringen til helikopteret. Denne metoden gjør at de tre aksene behandles hver for seg gjennom hele prosessen.

$$\dot{\theta}(t) = \frac{d\theta(t)}{dt} = \omega = \text{vinkelhastighet} \tag{10.1}$$

$$\theta(t) = \int_0^t \omega(t)\, dt \tag{10.2}$$

I mikrokontrolleren kan det ikke brukes kontinuerlige ligninger. Må derfor finne en diskret integrator. For å gjøre dette så enkelt som mulig er det i denne oppgaven valgt å bruke Eulers forover, ligning 10.3. Dette vil ikke gi et nøyaktig svar, men vil spare mye prosessering i forhold til mer avanserte integrasjons metoder. Et alternativ kunne vært å bruke en Runge-Kutta integrator. Denne er noe mer krevende, men gir en bedre tilnærming [25].

$$\dot{\omega}(k) \approx \frac{1}{T}\big(\omega(k+1) - \omega(k)\big) \tag{10.3}$$

$$\omega(k+1) = \dot{\omega}(k)T + \omega(k) \tag{10.4}$$

Ligning 10.4 kan nå enkelt implementeres i c eller matlab som:

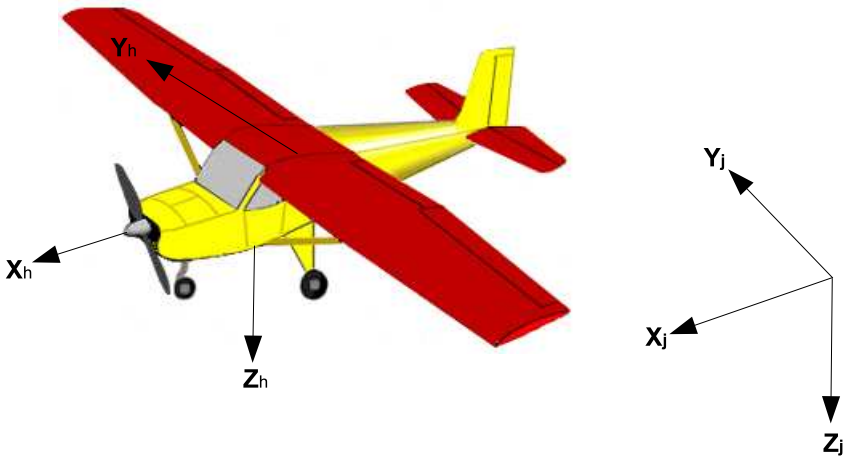
```
roll = roll + inn*T;
```

Hvor inn er vinkelhastigheten og T er sampletiden.

Det er nå utviklet en enkel lineær metode som gir oss vinkel fra vinkelhastighet. Problemet med denne metode er at det er en lineær tilnærming til et ulineært problem. Det er heller ikke tilstrekkelig å beskrive orienteringen til helikopteret med Eulers-vinkler. Rotasjon kan ikke gjøres i tilfeldig rekkefølge. Dette kan best forklares gjennom et eksempel. La oss si at du roterer i pitch retningen 90 grader. Deretter gjør du en roll på 90 grader. Dette gjør at nesen/x-aksen på helikopteret peker rett opp. Gjøres dette derimot i omvendt rekkefølge vil retningen bli helt forskjellig. X-aksen vil nå peke mot siden.
Siden helikopteret hovedsakelig skal operere rundt hover-tilstand, vil de tre vinklene ofte bli null. Når alle vinklene er null må helikopteret stå som i utgangspunktet. Dette blir altså, som å starte på ny. Alle feil som er akkumulert fjernes altså hver gang helikopteret hovrer. Siden vinkelutslagene rundt hover skal være små. Blir dette også som å definere et arbeidspunkt i et ulineært system. Denne metoden kan derfor være god nok rundt arbeidspunktet.

10.2 Avansert ulineær metode

Problemstillingen er her å lage en likning som kan gi orienteringen til helikopteret over hele spekteret, altså ikke bare rundt hover-tilstand. Isteden for å beskrive Eulers-vinklene direkte, bruker en to koordinatsystemer og beskriver sammenhengen mellom disse. Dette er en metode som først ble brukt i satellitt-teori. Et koordinatsystem er koblet til jorden og ett til helikopteret. Det første er statisk og vil ikke endres, mens koordinatsystemet til helikopteret endres med helikopterkroppen. For å beskrive sammenhengen mellom disse kan en bruke quaternations eller rotasjonsmatriser. I studiene er det rotasjonsmatriser som er blitt brukt. Det er derfor mest nærliggende å bruke disse. Denne delen bygger på [41], [10] og [31].



Figur 10.1: Kordinatsystemene bruk i denne delen av oppgaven. Bilde er hentet fra www.rc-airplane-world.com

10.2.1 Rotasjonsmatriser

Rotasjonsmatriser brukes for å konvertere vektorer og koordinater fra ett koordinatsystem til et annet. Det vil altså kunne inneholde all informasjon om oreinteringen til helikopteret. En annen fordel med denne metoden er at matrisen også kan inneholde informasjon om posisjonen til helikopteret gjennom translasjon.
For en fullstendig beskrivelse av orienteringen trengs det tre rotasjons-matriser, en for hver akse. Ved å multiplisere disse får en en 3x3 matrise, som gir en fullstendig beskrivelse. I ligningene nedenfor er ϕ = roll, θ = pitch, ψ = yaw. For en detaljert beskrivelse se [31].

$$R_{3D} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & sin\phi \\ 0 & -sin\phi & cos\phi \end{bmatrix} \begin{bmatrix} cos\theta & 0 & -sin\theta \\ 0 & 1 & 0 \\ sin\theta & 0 & cos\theta \end{bmatrix} \begin{bmatrix} cos\psi & sin\psi & 0 \\ -sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{10.5}$$

$$= \begin{bmatrix} cos\theta cos\psi & sin\phi sin\theta cos\psi - cos\phi sin\psi & cos\phi sin\theta cos\psi + sin\phi sin\psi \\ cos\theta sin\psi & sin\phi sin\theta sin\psi + cos\phi cos\psi & cos\phi sin\theta sin\psi - sin\phi cos\psi \\ -sin\theta & sin\phi cos\theta & cos\phi cos\theta \end{bmatrix} \tag{10.6}$$

Hvis en nå multiplisere denne med en vektor fra det ene koordinatsystem, får en den samme vektoren i det andre koordinatsystem. For å gå tilbake, bruker en den inverse av matrisen multiplisert med vektoren. En spesiell egenskap for rotasjonsmatrisen er at den inverse av rotasjonsmatrisen er det samme som den transponerte til rotasjonsmatrisen. Den transponerte er som kjent å bytte rader og kolonner og altså svært mye enklere enn å finne den inverse. Kolonnene og radene i matrisa kan ses på som enhetsvektorer. Ut fra dette kan en si at kolonnen i matrise er aksene i det ene systemet mens radene er aksene i det andre. En kan altså la kolonnene beskrive de tre aksene på helikopteret (subskript h). Da vil radene beskrive aksene på kordinatsystemet på bakken (subskript j). Dette kan ses i figuren under. Har nå en fullverdig metode for å beskrive orienteringen. Videre må det utvikles en metode for å oppdatere rotasjons-matriseren ved hjelp av vinkelhastighetene fra gyroen. Denne metoden er ikke ny, men det er ikke funnet simulerings-resultater rundt den.

$$\begin{matrix} & X_h & Y_h & Z_h & \\ \begin{bmatrix} \\ \\ \\ \end{bmatrix} & \begin{matrix} r_{xx} \\ r_{yx} \\ r_{zx} \end{matrix} & \begin{matrix} r_{xy} \\ r_{yy} \\ r_{zy} \end{matrix} & \begin{matrix} r_{xz} \\ r_{yz} \\ r_{zz} \end{matrix} & \begin{matrix} X_j \\ Y_j \\ Z_j \end{matrix} \end{matrix}$$

10.2.2 DCM Utledning

Det skal her utvikles en metode for å finne rotasjonsmatrisen i neste tidssteg. Hvis en da tenker som i den enkle metoden, delkapittel 10.1. Hvor en bruker en enkel integrator og summerer opp rotasjonen. Kan en ta rotasjonsmatrisen fra forrige tidssteg og legge til rotasjonen som er skjedd til neste tidssteg. Dette blir altså en Eulers forover-integrasjon. Som kjent vil en multiplikasjon mellom to rotasjonsmatriser gi en rotasjonsmatrise med begge rotasjonene. Det vil si hvis R_1 har en rotasjon på 30 grader om X aksen og R_2 har en rotasjon på 10 grader. Vil en multiplikasjon mellom disse resultere i en matrise som gir en rotasjon på 40 grader om X aksen.
For å summere opp rotasjonen kan en derfor ta rotasjonsmatrisen fra forrige tidssteg og multiplisert denne med en oppdaterings-matrise(Ω), som har rotasjonen mellom tidsstegene. Dette vil gi summen av rotasjon så langt. Altså en rotasjonsmatrise R som gir fullstendig informasjon om orienteringen til helikopteret.

$$R(t + dt) = R(t) \cdot \Omega \tag{10.7}$$

Videre må Ω finnes. Dette skal være en rotasjonsmatrise med rotasjonen fra forrige tidspunkt til nå. Tar en da utgangspunkt i en vanlig rotasjonsmatrise, ligning 10.6. Og antar at tidsintervallet (dt) er lav, vil vinkelendringen(dVinkel) mellom hvert tidssteg bli svært liten. Matrisa kan da forenkles ved å gjøre tilnærmingene under. Dette gir oss en svært enkel matrise som i ligning 10.11.

$$\lim_{dt \to 0} cos(\theta) \approx 1 \tag{10.8}$$

$$\lim_{dt \to 0} sin(\theta) \approx \theta \tag{10.9}$$

$$\lim_{dt \to 0} sin(\theta) \cdot sin(\psi) \approx 0 \tag{10.10}$$

$$\Omega = \lim_{dt \to 0} R = \begin{bmatrix} 1 & -d\psi & d\theta \\ d\psi & 1 & -d\phi \\ -d\theta & d\phi & 1 \end{bmatrix} \tag{10.11}$$

Neste tidssteg kan nå finnes ved å bruke vinkelendringene. Det er ønskelig å finne denne ved å bruke vinkelhastigheten. Dette kan enkelt gjøres ved å ta utgangspunkt i at den deriverte av vinkel er vinkelhastighet, se ligning 10.12. Skrives denne om får en ligning,10.13. Denne viser at den deriverte multiplisert med tidsintervallet er endringen av vinkel. Dette kan nå settes direkte inn i oppdateringsmatrisen Ω og en får ligning, 10.14.

$$\frac{d\theta}{dt} = \dot{\theta} = \omega \tag{10.12}$$

$$d\theta = \omega \cdot dt \tag{10.13}$$

$$\Omega = \begin{bmatrix} 1 & -\omega_z dt & \omega_x dt \\ \omega_z dt & 1 & -\omega_y dt \\ -\omega_x dt & \omega_y dt & 1 \end{bmatrix} \tag{10.14}$$

Hvis en nå endrer t til k og dt til T(sampletid) får en, en diskret ligning som gir oss rotasjonsmatrisen i neste tidssteg. Dette vil gi en fullstendig ulineær beskrivelse av orienteringen. En setter ganske enkelt vinkelhastighetene inn i oppdateringsmatrisen og multipliserer med R(t) fra forrige tidssteg se ligning 10.16.

$$\Omega = \begin{bmatrix} 1 & -\omega_z T & \omega_x T \\ \omega_z T & 1 & -\omega_y T \\ -\omega_x T & \omega_y T & 1 \end{bmatrix} \tag{10.15}$$

$$R(k+1) = R(k)\Omega \tag{10.16}$$

Denne metoden er bare en grov tilnærming, som gir oss det samme som Eulers forover integrasjon. Ønsker en å bruke en annen type integrator kan en jobbe videre med ligning 10.7. Settes denne inn i definisjonen for den grensederiverte, ligning 10.17 kan en få den kontinuerlige differensial likningen. Denne kan deretter diskretiseres ved hjelp av en annen type integrator for eksempel Runge-Kutta.

$$\dot{R}(t) = \lim_{dt \to 0} \frac{R(t+dt) - R(t)}{dt} \tag{10.17}$$

Samme ligning kan også utledes ved å bruke kinematikk. En beskriver da hver av enhetsvektorene i rotasjonsmatrisen ved hjelp av rotasjonsvektorer, se ligning 10.18. Denne kan da diskretiseres før den settes inn i matrisa. Bruker en Eulers forover vil dette gi nøyaktig samme svar som over.

$$\frac{r(t)}{dt} = \omega(t) \otimes r(t) \tag{10.18}$$

For å oppsummere er det nå utvikler en metode for å finne rotasjonsmatrisen i neste tidssteg ved hjelp av vinkelhastighetene. Ligningen for dette kan ses under.

$$R(k+1) = R(k) \begin{bmatrix} 1 & -\omega_z T & \omega_x T \\ \omega_z T & 1 & -\omega_y T \\ -\omega_x T & \omega_y T & 1 \end{bmatrix} \tag{10.19}$$

Som en ser er dette en relativt stor og komplisert algoritme. Den vil kreve svært mye av mikrokontrolleren. Deler av denne metoden ble implementert i c for simulering, hvor den viste seg å være svært prosessorkrevende. Bare multiplikasjon av rotasjonsmatrisen med oppdaterings-matrisen brukte 4ms ved bruk av float-point. Hvis det bare var denne som skulle gjøres er en allerede nede i en sampletid på (1/0.004s) = 250 hz. Med alle beregningene som skal gjøres vil det være vanskelig å holde denne metoden over 50 Hz med den nåværende prosessoren. Dette er på alle måter en metode som er mer rettet mot systemer i fly hvor orienteringen kan være forskjellig fra null over lengre tid. For å gjøre denne algoritmen mer prosessorvennlig kunne en foreksempel brukt quaternions. Dette ville gitt en fullstendig representasjon ved hjelp av bare tre variabler. En vil da unngå de store matrise multiplikasjonene. For en fullstendig besrkivelse av quaternions se [23].

10.2.3 Normalisering

Tilnærmingene som er gjort for å gå fra vinkelhastighet til rotasjons-matrise. Vil over tid gi et feilbidrag. Får dette skje over lengre tid. Vil feilbidraget kunne gjøre at rotasjonsmatrisen ikke lengre oppfyller kravene for å være en rotasjonsmatrise. Dette vil gi store problemer siden algoritmene utnytter mange av egenskapene til rotasjonsmatrisen. For å løse dette kan en utnytte at alle aksene skal står ortogonalt på hverandre. Det må derfor finnes en metode for å sørge for at aksene rettes opp hvis de ikke er ortogonale. Det ble foreslått en metode for å løse dette i [10] og denne er blitt implementert i Matlab for simulering. I denne metoden finner en først hvor mye aksene har driftet. Dette kan en gjøre ved å ta prikk produktet mellom to av dem. Står det 90 grader på hverandre skal dette gi null, se ligning 10.20.

$$A \odot B = |A||B| \cdot cos(\theta_{AB}) = \text{feilbidrag} \tag{10.20}$$

Videre deler en dette feilbidraget mellom aksene og roterer begge tilbake. Dette gir ikke en nøyaktig løsning, men når dette gjøres i hvert tidssteg vil det bli en god tilnærming. For å finne den siste aksen kan en ta kryss produktet mellom de to første. Dette vil gi en akse, som står normalt på begge. Dette er altså den siste aksen. Deretter skaleres hver av aksene slik at magnituden er en. Aksene kan nå settes direkte inn i rotasjonsmatrisen. Denne metoden ble implementert i Matlab og fungerte svært bra. Implementasjonen kan ses nedenfor.

```
function [DCM] = normal(DCM)
X= DCM(1,:)';
Y= DCM(2,:)';
Z= DCM(3,:)';

    error = dot(X,Y);
    Xo = X - (error/2) * Y;
    Yo = Y - (error/2) * X;

    Zo = cross(Xo,Yo);

    Xn = 0.5*(3-dot(Xo,Xo))*Xo;
    Yn = 0.5*(3-dot(Yo,Yo))*Yo;
    Zn = 0.5*(3-dot(Zo,Zo))*Zo;

DCM(1,:) = Xn';
DCM(2,:) = Yn';
DCM(3,:) = Zn';
```

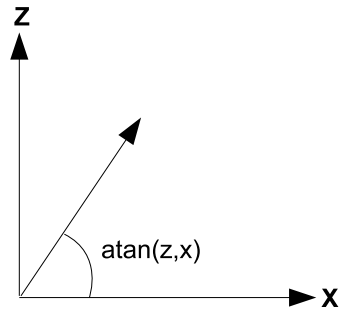
11 Beregning av orientering med akselerometer

Det er nå utviklet en metode for å gjøre gyro-data til orientering. Det samme må nå gjøres for akselerometer-data. Det må utvikles en metode til bruk sammen med den enkle representasjonen hvor hver vinkel regnes ut hver for seg. Og en som kan brukes sammen med den avanserte. De tre målingene vil i hvert av tilfellene bli brukt som en vektor i planet. Når denne ikke påvirkes av andre krefter, vil den alltid peke i samme retning som Z-aksen på helikopteret. Den vil altså representere sann Z_h. Derfor vil vinkelen mellom vektoren og tyngdeakselerasjon være orienteringen.

$$a = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{11.1}$$

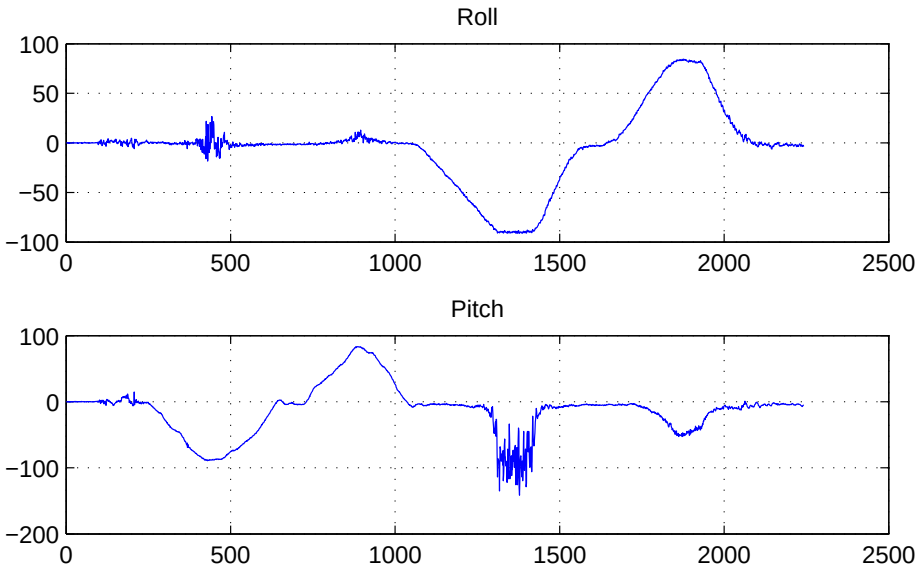
11.1 Separate vinkler

For å finne de to vinklene hver for seg brukes Z-målingen sammen med en av de andre aksene til å danne et punkt i 2D planet. Vinkelen til dette punktet fra den positive x-aksen kan enkelt beregnes ved hjelp av atan2(z,x) funksjonen. Se figur 11.1. Dette gir en enkel metode for å beregne roll og pitch hver for seg.

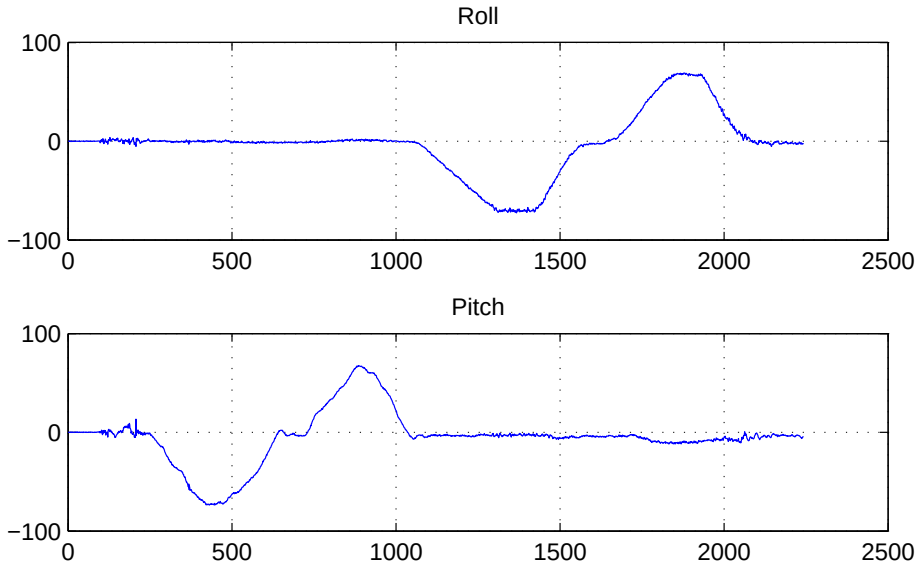


Figur 11.1: Bruk av atan2 funksjonen

Problemer oppstår når Z-målingen sammen med en av de andre målingene blir svært lav. Dette skjer når foreksempel pitch er null og roll er stor. Tyngdeakselerasjonen vil da måles av y som blir stor mens x og z blir små. Hvis x da går under null vil dette gi en feil på 180 grader, som er veldig kritisk. Siden formelen er lineær kan dette løses ved å legge til litt på z slik at den aldri blir så lav som null. Deretter kan resultatet multipliseres slik at det stemmer igjen. Figur 11.2 viser et eksempel på dette. Her kan en se at feilen er over 100 grader på det verste.



Figur 11.2: Orientering beregnet ved hjelp av atan2



Figur 11.3: Orientering beregnet ved hjelp av atan2 med høyere z

11.2 Metode 2

Denne metoden skal brukes sammen med rotasjonsmatrise representasjonen. Her brukes ikke Eulers vinkler og metoden over vil derfor ikke fungere. En ønsker å sammenligne akselerasjons-vektoren med gravitasjons-vektoren. For å gjøre dette lages det en gravitasjons-vektor som alltid peker rett ned, ligning 11.2. Denne vil være i jordens koordinatsystem. Hvis en nå bruker rotasjonsmatrisen til å flytte denne fra jordens koordinatsystem til helikopteret. Vil en få en vektor som peker i samme retning som akselerasjons-vektoren forutsatt at rotasjonsmatrisen ikke har driftet, se ligning 11.3.

$$g = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{11.2}$$

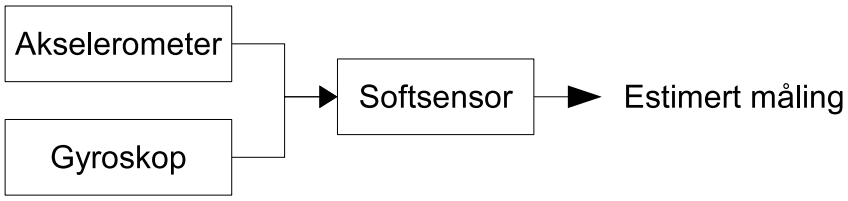
$$\begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{bmatrix}^T \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} r_7 \\ r_8 \\ r_9 \end{bmatrix} \tag{11.3}$$

En ser fra ligning 11.3 at når gravitasjons-vektoren flyttes vil den bli nøyaktig lik nederste rad i rotasjonsmatrisen også kalt Z_j. Isteden for å bruke tid på å flytte vektoren i hver iterasjon. Kan en derfor bruke Z_j direkte, som gravitasjons-vektor i helikopter-planet. En har nå to vektorer. En som er estimert gravitasjon, hentet fra rotasjonsmatrisen og en som er sann gravitasjon hentet fra akselerometeret. Fra dette kan en beregne hvor mye rotasjonsmatrisen har driftet. For å gjøre dette kan en ta kryss produktet mellom de to vektorene. Dette gir som kjent en vektor som står normalt på begge, se ligning 11.4. Magnituden til denne vektoren vil henge direkte sammen med hvor mye matrisen har driftet. Denne kan derfor brukes direkte i softsensoren. Denne metoden ble foreslått i [10], men aldri forklart eller simulert. Metoden ble derfor implementer i Matlab. Simuleringene vil bli vist under resultater.

$$|Z_e \otimes Z_{acc}| = |Z_e||Z_{acc}| \cdot sin\theta \tag{11.4}$$

12 Softsensor

Det finnes altså ingen billige sensorer som kan måle vinkel perfekt. Gyroene har problemer med bias og akselerometeret måler bare vinkel ved lave frekvenser. En ser at ved å bruke begge vil disse sensorene utfylle hverandre. Løsningen blir da å kombinere disse. For å kunne bruke det beste fra begge verdener. Til dette kan en softsensor brukes. En softsensor kan implementeres i software og er en algoritme for estimering av en prosessvariabel. En vil i dette kapittelet se på løsninger for å gjøre nettopp dette. Softsensoren bruker informasjon fra gyro og akselerometer til å estimere vinkelen. Som i resten av oppgaven er et viktig krav at softsensoren ikke er for prosessorkrevende.
Softsensoren bygger ofte på en matematiskmodell av det dynamiske systemet. På denne måten kunne en hatt enda et estimat av vinkelen. Dette estimatet kunne bygget på motorpådrag og hvordan orienteringen skulle vært ut fra den matematiskemodellen. Det er i denne oppgaven valgt å ikke basere softsensoren på dette. Dette ville gjort systemet mer prosessorkrevende og det ville også gjort det spesifikt for dette helikopteret. Det er ønskelig med et mer generelt filter. Et annet problem er at en trenger filteret for å få gode nok data til å utvikle modellen. Felles for filtrene i denne oppgaven er at de bruker akselerasjons-data som virkelig verdi. Det finnes svært mange metoder for å implementere en softsensor. Noen av disse vil bli forklart i dette kapittelet. Denne delen er basert på [34], [39] og [10].



Figur 12.1: Oversikt over softsensor implementasjonen

12.1 Kalmanfilter

Det mest nærliggende er å bruke et kalmanfilter. Et kalmanfilter kan brukes for å estimere en tilstand, som ikke kan måles. Det er nettopp dette softsensoren skal gjøre. Filteret vil gi et optimalt tilstandsestimat i den forstand at aposteriori-estimeringsavvikets varians minimeres. En del forutsetninger må da oppfylles for å kunne si at filteret er optimalt. Prosessen må være påvirket av tilfeldig (stokastisk) prosess-støy og det må være stokastisk målestøy. Kalmanfilteret er et verktøy som bare kan brukes på lineære systemer. Som kjent har en i denne oppgaven en lineær og en ulineær representasjon, som skal estimeres. Det vil altså bare fungere for den ene represetasjonen. Det finnes andre typer kalmanfilter som kan kompensere for dette. Disse vil bli beskrevet senere.

12.1.1 Kalman-implementasjon

Kalmanfilter baseres vanligvis på en ferdig matematisk-modell av dynamikken til systemet. Som kjent var dette ikke ønskelig i denne oppgaven. Kalmanfilteret må derfor implementeres på en noe spesiell måte. En kan da ta utgangspunkt i den lineære ligningen som blir brukt for å gå fra vinkelhastighet til vinkel, se kapittel 10.1. Hvis en her legger til en variable for bias vil denne kunne brukes direkte i filteret, se ligning 12.1. Denne delen baserer seg på [39] og [18]

$$\theta(k+1) = (\omega(k) - \beta)\,T + \theta(k) \tag{12.1}$$

Her er θ vinkel, ω vinkelhastighet, β bias og T sampletid. For å bruke kalmanfilter-algoritmen er det ønskelig å ha denne på lineær tilstandsromform. Den generelle lineære tilstandsrom-modellen er vist under. Tabellen er hentet fra, [39].

$$x(k+1) = \Phi(k)x(k) + \Gamma(k)u(k) + \Omega(k)v(k) \tag{12.2}$$
$$y(k) = D(k)x(k) + w(k) \tag{12.3}$$

Navn	Symbol	Dimensjon
Transisjonsmatrise	Φ	n $\times$ n
Pådragsmatrise	Γ	n $\times$ s
Forstyrrelsematrise	Ω	n $\times$ n
Målematrise	D	l $\times$ n
Direktekoblingsmatrise	E	l $\times$ s

Hvis en nå lar vinkel og bias være tilstander og vinkelhastighet være pådrag. Kan en sette ligning 12.1, på tilstandsrom-form og får ligningen under. Det er da vanlig å ikke ta med støyleddene.

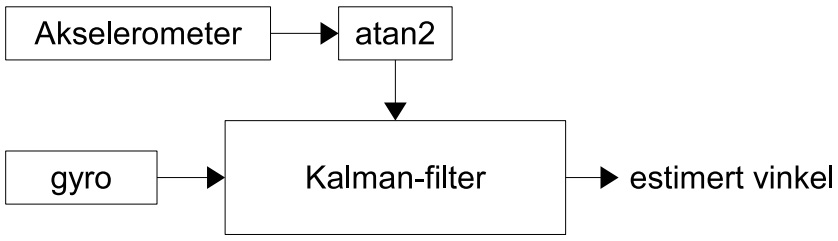
$$\begin{bmatrix} \theta \\ \beta \end{bmatrix}_{(k+1)} = \begin{bmatrix} 1 & -T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \theta \\ \beta \end{bmatrix} + \begin{bmatrix} T \\ 0 \end{bmatrix} \omega \tag{12.4}$$

Har nå en ferdig lineær tilstandsrom-modell, som kan brukes direkte i kalman-algoritmen. Videre lar en $y(k)$ eller måleinngangen til filteret være vinkel beregnet fra akselerometer-data. Kalmanfilteret vil da gi et optimalt vinkel estimat for hver iterasjon.

Under vises selve kalman-algoritmen. Matrisene fra tilstandsrom-modellen kan nå settes direkte inn i denne. Filter-parameteren Q og R settes vanligvis ved hjelp av eksperimentering. Disse ligningene må gjøres i hvert tidssteg. Ligningene er hentet fra [39]

$$\overline{x}(k) = \Phi\hat{x}(k-1) + \Gamma u(k-1)$$
$$\overline{P}(k) = \Phi\hat{P}(k-1)\phi^T + Q$$
$$K(k) = \overline{P}(k)D^T(D\overline{P}(k)D^T + R)^{-1}$$
$$\hat{x} = \overline{x}(k) + K(k)[y(k) - D\overline{x}(k)]$$
$$\hat{P}(k) = (I - K(k)D)\overline{P}(k)$$

Kalmanfilter-algoritmen vil være nokså rett frem å implementere i c, men algoritmen har svært mange matrise-operasjoner. Dette gjør at den blir relativt prosessorkrevende. Den vil derfor presse sampletiden sterkt ned. Dette er også en lineær metode, som bare vil fungere for den enkle representasjonen. Det krever også en del erfaring å finne filter parameterne Q og R. Dette kan bli et problem for andre som ønsker å implementere systemet siden disse må passe til deres system.



Figur 12.2: Oversikt over kalman-implementasjonen

12.2 Kalmanfilter på ulineært system

Den vanligste måten gjøre tilstandsestimering på i et ulineært system er å linearisere systemet omkring et fast arbeidspunkt. Problemet er at en får et filter som bare er gjeldene rundt dette arbeidspunktet.
Dette kan løses ved å bruke et utvidet kalmanfilter(EKF), som gir en tilnærming til det optimale estimatet. Det utvidede kalmanfilteret linearisere rundt et flytende arbeidspunkt, en lar arbeidspunktet ligge rundt det siste tilstandsestimatet. Det må da beregnes nye filterparametere for hvert tidssteg og dette er svært krevende for prosessoren. Ligningene under viser den generelle algoritmen for et EKF. Ligningene er hentet fra, [39].

$$\overline{x}(k) = f\left[\hat{x}(k-1), u(k-1), ...\right] \tag{12.5}$$

$$\Phi(k) = \frac{\partial f(\cdot)}{\partial x}|_{\hat{x}(k-1),u(k-1)} \tag{12.6}$$

$$\overline{P}(k) = \Phi(k)\hat{P}(k-1)\Phi^T(k) + Q \tag{12.7}$$

$$D(k) = \frac{\partial g(\cdot)}{\partial x}|_{\overline{x}(k)} \tag{12.8}$$

$$K(k) = \overline{P}(k)D^T(k)\left(D(k)\overline{P}(k)D^T(k) + R\right)^{-1} \tag{12.9}$$

$$\overline{y}(k) = g\left[\overline{x}(k), ...\right] \tag{12.10}$$

$$\hat{x}(k) = \overline{x}(k) + K(k)\left[y(k) - \overline{y}(k)\right] \tag{12.11}$$

$$\hat{P}(k) = \left(I - K(k)D(k)\right)\overline{P}(k) \tag{12.12}$$

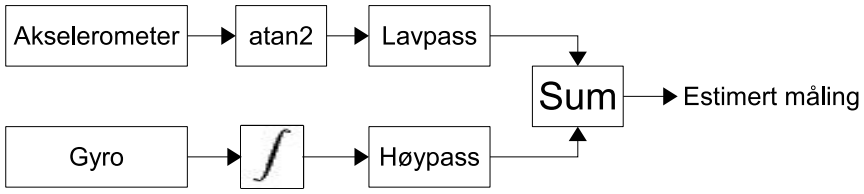
Filteret kan implementeres ved å bruke representasjonen fra kapittel 10.2, som modell. De 9 elementene i rotasjonsmatrisen vil da bli tilstander. Som over blir vinkelhastighetene pådrag og akselerometer-data blir måling. Dette vil bli en svært stor algoritme og derfor ikke implementerbart på en vanlig mikrokontroller.

12.3 Komplementært-filter

Komplementær-filter brukes for å fusjonere målinger. I motsetning til kalmanfilteret, bygger ikke Komplementær-filter på en matematisk-modell. Det lager et tilstandsestimat, som bare baserer seg på de forskjellige målingene. Det er nettopp dette som er målet i denne oppgaven. Det finnes to aktuelle typer typer, 1.orden og 2.orden. Disse er satt opp på nokså forskjellige måter, men har samme funksjon. Begge gir en enkel og oversiktlig tilstandsestimering. Dette er ikke et veldig utbredt filter, det er derfor relativt vanskelig å finne god informasjon om det. Denne oppgaven bygger derfor på [10] og [32]

12.3.1 1. orden

For å forklare dette filteret. Tas det utgangspunkt i at akselerometer-data er rett hvis den måles over lengre tid, altså ved lave frekvenser. For å fjerne feil kan dette derfor lavpass-filtreres. En har da en metode for å få nøyaktige data. Denne filtreringen vil dessverre gjøre at hurtige orienterings-endringer forsvinner. For å løse dette problemet brukes gyro-data, som har det omvendte problemet. De har en sakte bias-drift, de blir altså feil over tid. Det er da logisk å la denne gå gjennom et høypass-filter. En har nå høypass og lavpass filtrert data. Settes disse sammen vil det gi en estimert-måling som er rett både for raske endringer og over lengre tid.
Et 1. ordens Komplementært-filter er altså et høypass og et lavpass filter i kombinasjon. Se figur 12.3. Dette gir en svært enkel ligning, 12.13. Denne er diskretisert ved hjelp av Eulers forover integrator.



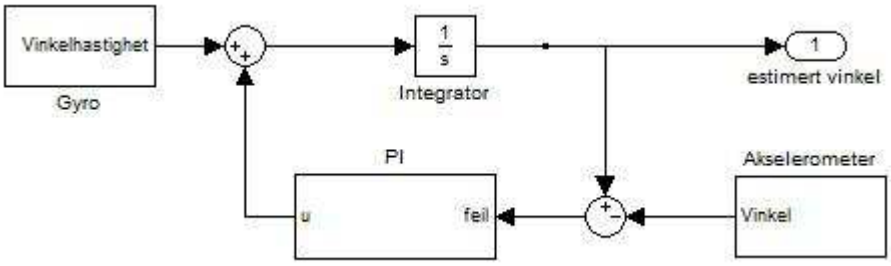
Figur 12.3: 1. ordens Komplementært-filter

$$\theta(k+1) = k(\theta(k) + \omega(k)T) + (1-k)\theta_a \tag{12.13}$$

Her er θ vinkel, ω vinkelhastighet, θ_a er akselerometer-vinkelen og T sampletid. K er filter konstanten, denne settes mellom 1 og 0. Er denne høy vil hovedvekten legges på gyro-måling. Denne settes derfor etter hvor gode akselerometer-dataene er. Er det mye støy på disse målingene kan en sette k høy og dataene vil da akumuleres sakte opp. Dette er altså et filter som er svært enkelt å implementere i c. Det krever heller ikke mye av prosessoren og det er enkelt for en bruker å forså hvordan filter parameteren skal finnes.

12.3.2 2.orden

Et 2.ordens Komplementært-filter er i realiteten en PI-regulator. En regulator tar inn avviket og gir ut hvor mye du må legge til for å komme opp til ønsket verdi. Hvis en da lar ønsket verdi være vinkel beregnet fra akselerometer-data. Og trekker denne vinkelen fra estimert-vinkel,vil dette gi avviket. Regulatoren kan da regne ut et pådrag som må legges til gyroen for å komme opp til akselerometer-data. En kan på denne måten designe et filter som vil være mer eller mindre intuitivt for alle med en regulerings-bakgrunn. Et blokkskjema av filteret kan ses i figur 12.4. Her vises filteret for den enkle representasjonen, samme type filter vil også bli brukt på den avanserte.



Figur 12.4: Blokkskjema

For å utvikle et 2.ordens komplementært-filter tas det utgangspunkt i formelen for en vanlig PI-regulator. Se ligning, 12.14.

$$u(t) = K_p e(t) + \frac{K_p}{T_i} \int_0^t e(t)dt \tag{12.14}$$

Her er $u(t)$-verdien som skal legges til vinkelhastigheten. e(t)-feil, K_p og T_i blir filter-parametre. For å bruke denne i mikrokontrolleren må den være på diskretform. Tilnærmer derfor integralet som i ligning 12.15. Dette er en grov tilnærming, men det gjør algoritmen enklere og mindre prosessorkrevende. Har nå ligningen for en PI-regulator på diskretform, 12.16.

$$\int_0^t e(t)dt = \sum_{n=0}^{k} e(n)T \tag{12.15}$$

$$u(k) = K_p e(k) + \frac{K_p T}{T_i} \sum_{n=0}^{k} e(n) \tag{12.16}$$

Denne kan nå forenkles videre slik at mikrokontrolleren skal måtte gjøre færrest mulig operasjoner. Kan da bruke ligning,12.17 til å erstatte deler av ligning,12.16. Dette gir den ferdig ligningen 12.18. Denne kan nå brukes direkte som et filter i mikrokontrolleren.

$$K_i = \frac{K_p T}{T_i} \tag{12.17}$$

$$u(k) = K_p e(k) + K_i \sum_{n=0}^{k} e(n) \tag{12.18}$$

Det er tydelig at denne implementasjonen er mer krevende for prosessoren enn 1.ordens filteret. Allikevel vil den kreve svært mye mindre enn foreksempel kalmanfilteret. Den har også flere tuning parametere enn 1.ordens filteret, dette bør gi bedre fleksibilitet. Dette gjør den til en sterk kandidat for bruk i det ferdige systemet. Under vises et eksempel på hvor enkelt denne kan implementeres i Matlab.

```
function [vinkel] = mitt_comp_filter(acc_v, gyro,T,k,ki)
%% mitt filter 2.ordens komplimenter filter

int_reg(1) = 0; % integralet som bygger seg opp i regulatoren
int_y(1) = 0;   % integral av vinkelhastigheten

for i=1:size(gyro)-1,
    avvik = acc_v(i) - int_y(i);
    g = avvik * k;
    int_reg(i+1) = g*ki + int_reg(i);
    int_y(i+1) = (g + int_reg(i+1) + gyro(i)) * T + int_y(i);
end

vinkel = int_y;
```

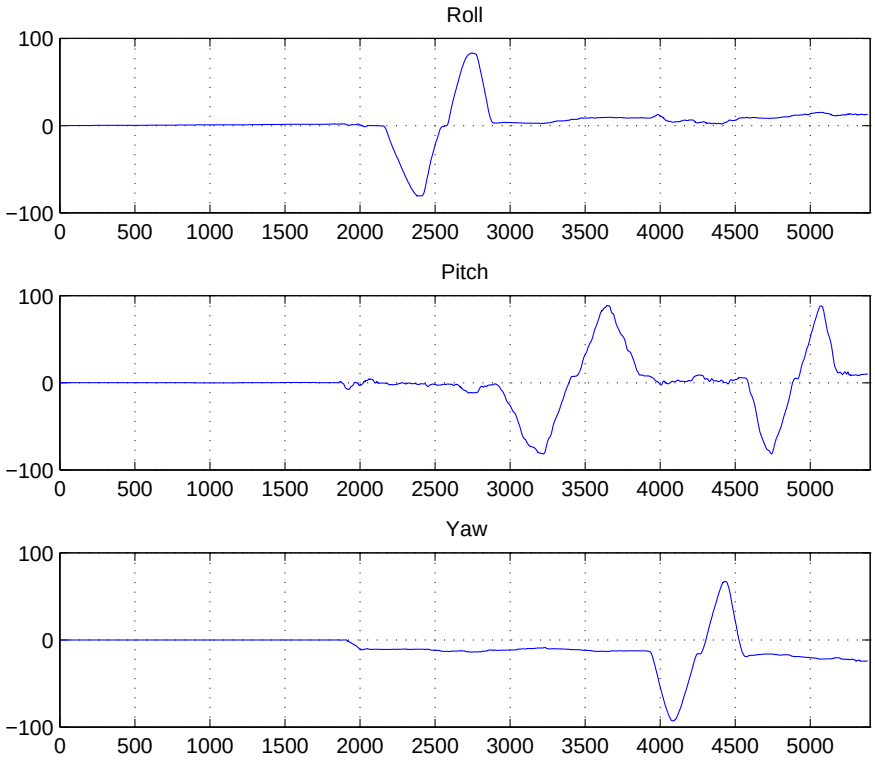
13 Resultat

Det er nå utviklet flere forskjellige metoder for å finne orienteringen til helikopteret. Det er også utviklet en rekke metoder for å forbedre resultatene. Videre skal det testes hva som er den beste kombinasjonen. Her er det svært mange krav og ta hensyn til. Men spesielt vil det være hvor mye ressurser algoritmen krever, mot hvor bra resultatene blir. For at helikopteret skal kunne stabilisere seg selv, må resultatene være tilnærmet perfekt. En liten "vipp" vil sette helikopterkroppen i bevegelse. For å oppsummere hvilke metoder som nå skal testes. Finnes det nå to metoder for å beregne representasjon fra vinkelhastighet. Videre har en tre typer softsensor, Kalmanfilter, 1.ordens og 2.ordens komplementær-filter. Har også en metode for å fjerne målestøy ved hjelp av Butterworth-filter.
Noen av metodene kan fjernes uten videre testing. Dette gjelder spesielt Kalman-algoritmen. Hadde det vært tilgang til ubegrenset regnekraft, ville en implementasjon av utvidet-kalmanfilter sammen med DCM-metoden vært å foretrekke. Dette ville som nevnt over gitt en tilnærmet optimal tilstandsestimering av den ulineære representasjonen. Dette er dessverre ikke en mulighet, siden rask sampletid er et viktig krav i dette prosjektet. Kalmanfilter-algoritmen trenger svært mange regne-operasjoner i hver iterasjon. Det er derfor valgt å bruke 1. og 2. ordens komplementær-filter til videre testing i denne oppgaven. For å teste hva som er den beste kombinasjonen ble de forskjellige metodene implementert i Matlab som m-filer. Dette gir mulighet for en rask implementering, som ikke er svært forskjellig fra implementasjonen i c. På denne måten får en samme resultat, som en implementering på mikrokontrolleren vil gi. I Matlab er det også enkelt å få oversiktlige resultater ved hjelp av plott. Test-data er da hentet fra helikopteret og importert til Matlab. For å gjøre koden oversiktlig er de fleste algoritmer implementert som funksjoner.

13.1 Representasjon

13.1.1 Lineær

Starter her med å vise resultatene av den lineære representasjons-metoden. Dette blir gjort med funksjonen roll_int = integrer(roll,'sum',T). Hvor roll er vinkelhastighet hentet fra mikrokontrolleren, sum er integrasjons-metode og T er sampletid. Hvordan funksjonen er implementert kan ses i vedlegg A. Helikopteret blir her holdt i hånden og snudd 90 grader i alle retninger uten propellene i gang.

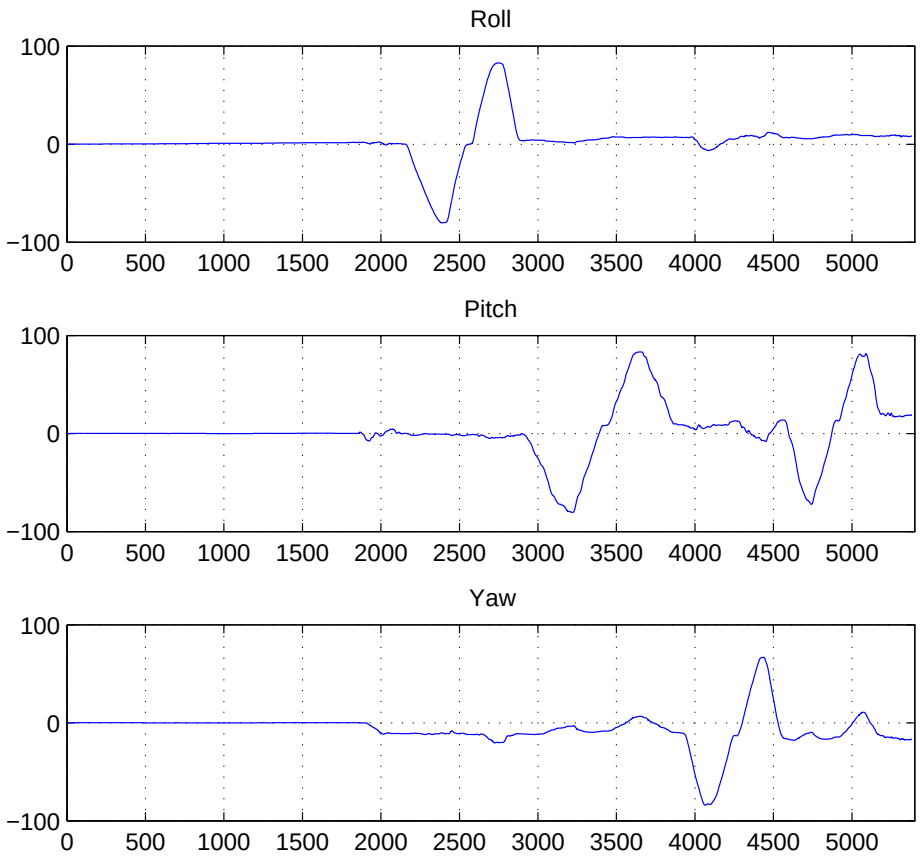


Figur 13.1: Representasjon beregnet ved hjelp av den enkle metoden. Her er y aksen grader, mens x aksen er sampler

En ser fra plottet at denne metoden gir svært gode resultater, men det er viktig å huske på at dette er en lineær tilnærming til et ulineært problem. En kan også se at selv på så korte tester er det en viss bias-drift. Små ripplene kommer av at helikopteret holdes i hånden når det roteres.

13.1.2 Ulineær

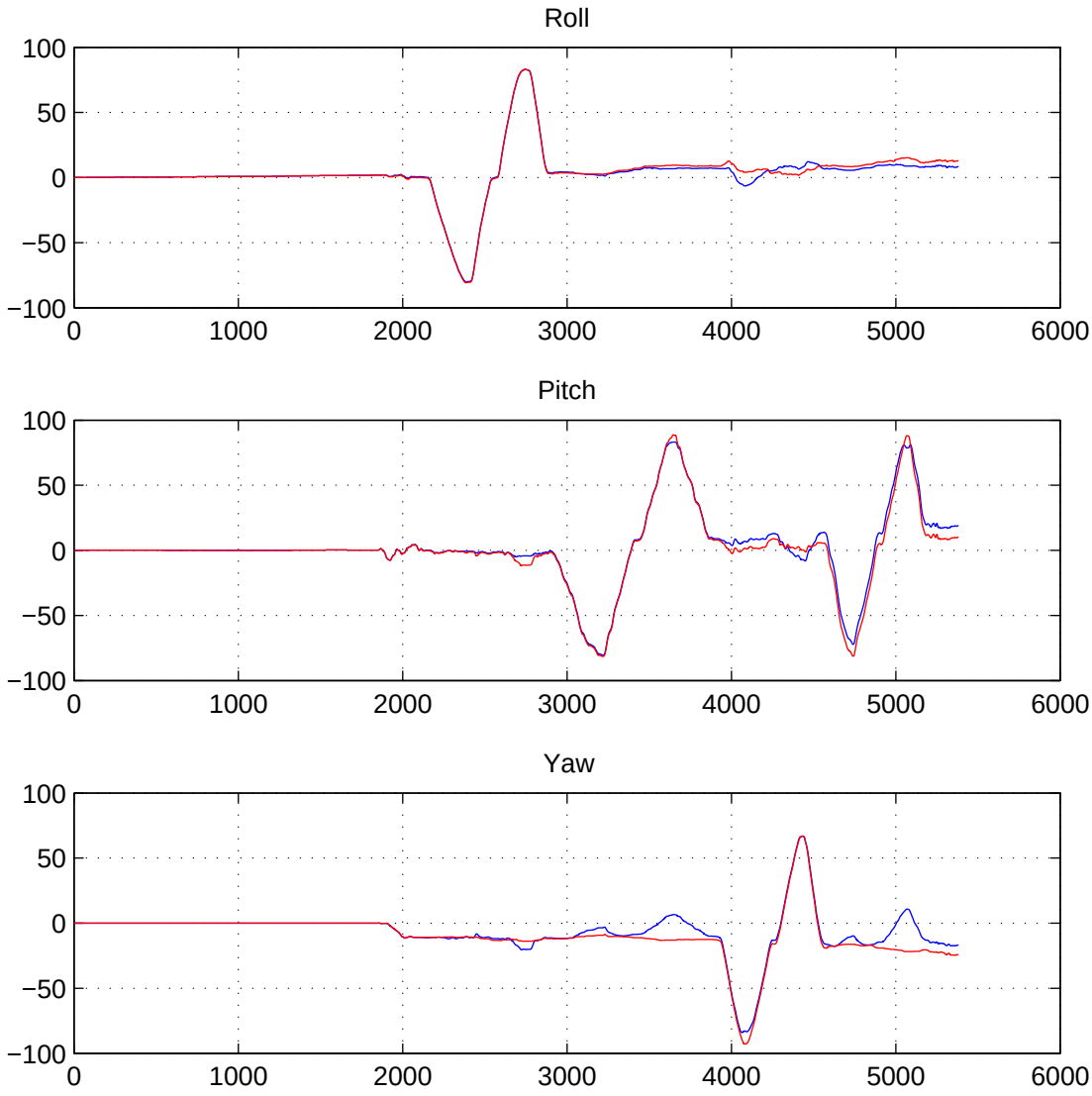
Denne metoden er noe mer omfattende, men implementert på samme måte som den enkle. Metoden er så omfattende at selv Matlab har problemer med å kjøre den. Dette kommer av alle matrise-multiplikasjonene. Den må derfor gi betydelig mye bedre resultater for å bli valgt til implementering på mikrokontrolleren. Plottet er laget med metoden [pitch_u,roll_u,yaw_u,ut] = DCM(roll,yaw,pitch,acc_x,acc_y,acc_z,T,0). Inngangene til funksjonen er vinkelhastighetene fra gyroen og data fra akselerometeret. Den siste parameteren brukes for å skrue av og på filteret. Utgangen er Eulers vinkler, hentet ut fra rotasjonsmatrisen. Dette er ikke rett fram og derfor gjort med en egen funksjon eulers(DCM). Vinklene blir derfor bare en tilnærming, siden disse ikke gir en like bra representasjon som selve matrisen. Tilnærmingen gjelder spesiell yaw, siden denne må hentes ut på en spesiell måte. En kan se denne metoden i vedlegg C. Dette vil ikke være et problem i en regulerings situasjon, siden en der bruker rotasjonsmatrisen direkte. Implementasjonen av filteret kan sees i vedlegg D.



Figur 13.2: Representasjon beregnet ved hjelp av den ulineære metoden.

13.1.3 Sammenligning

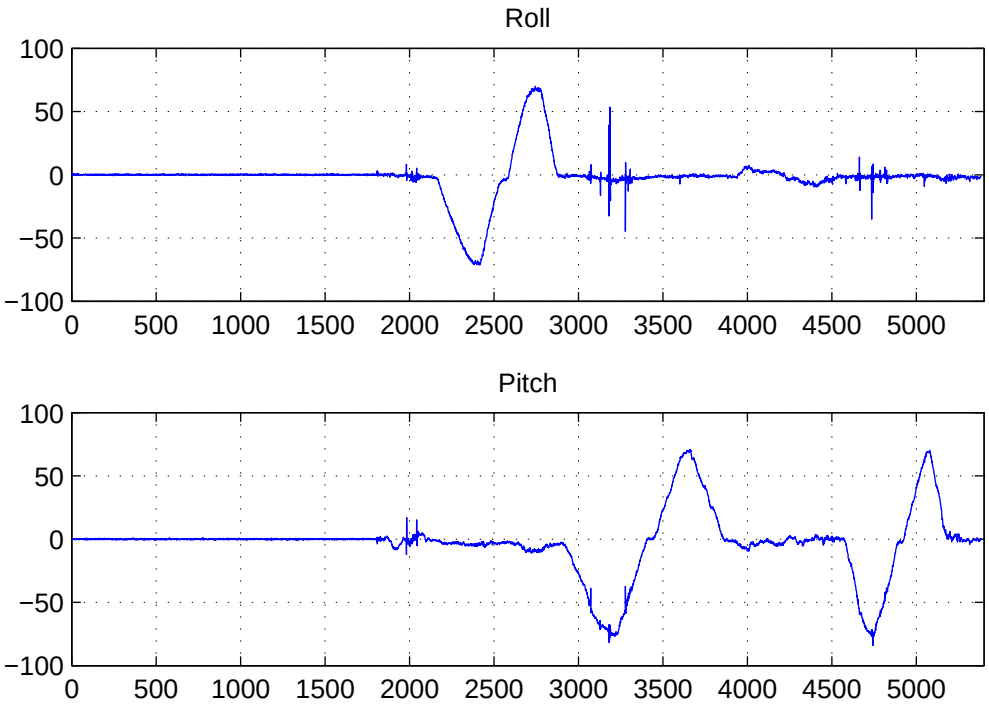
En ser at begge metodene gir et nokså likt resultat. Det er derfor her gjort et sammenlignings plot som viser begge metoden plottet sammen.



Figur 13.3: Sammenligning av representasjonene. Rød lineær metode, blå ulineær metode

13.2 Softsensor

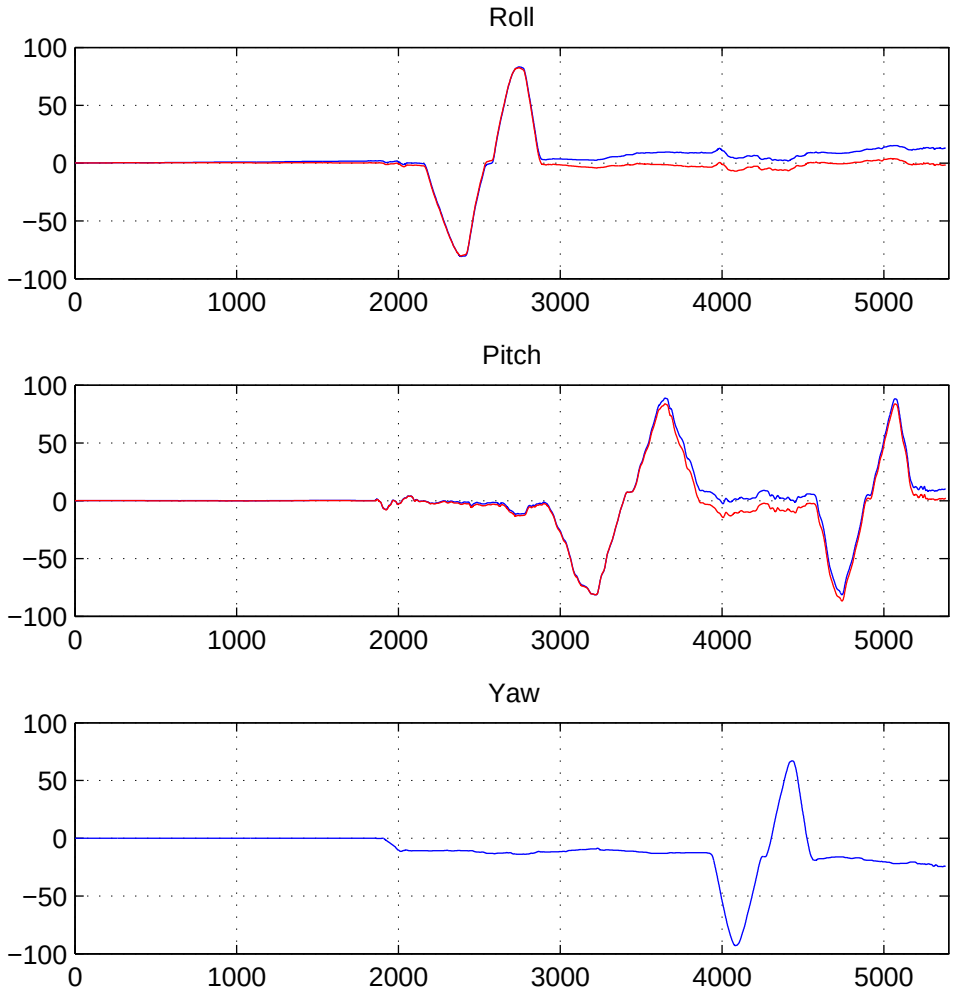
Siden begge metodene over gir tilnærmet likt resultat. Blir det gjort videre tester med softsensor implementert. For å se om en av metodene blir bedre enn den andre. Begge blir da implementert med et 2.ordens komplementær-filter. Siden dette kan brukes på begge representasjons-metodene. En bruker da akselerometer-data, som sann måling. Disse dataene er plottet ved hjelp av atan2 metoden og kan ses i figur 13.4. En kan se at det i dette plottet ikke er problemer med bias-drift, men det er noe støy spesielt på roll aksen. Denne støyen kommer av at helikopteret holdes i hånden. Det vil senere bli arbeidet mer med filteret til den metoden som gir best resultater. Filter-parameterne er derfor ikke perfekt tunet.



Figur 13.4: Data hentet fra akselerometeret

13.2.1 Lineær-representasjon

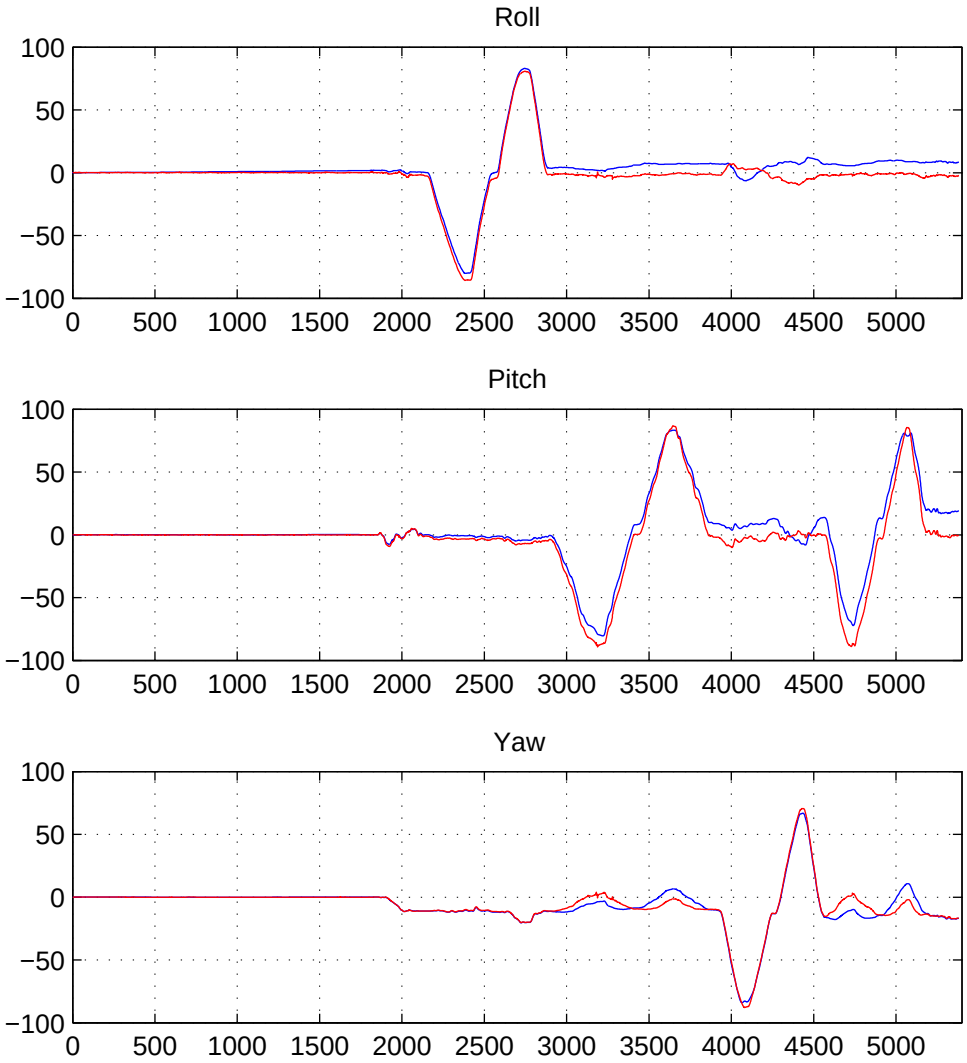
Filteret som blir brukt her er det samme som blir beskrevet i delen om 2.ordens komplementær-filter. Også dette er implementert som en funksjon i Matlab. En bruker filteret ved hjelp av funksjonen mitt_comp_filter(aks, vinkelhastighet,T,K_P,K_ti), se vedlegg B. Dette må da gjøres for både roll og pitch. Det finnes ikke korrigeringsdata for yaw, det gjøres derfor ingenting med denne.



Figur 13.5: Filtrert data fra gyroene. Det blå plottet er orginaldata, mens det røde plottet er filtrert

13.2.2 Ulineær

For å aktivere filteret i den ulineær metoden, setter en ganske enkelt den siste parameteren i DCM() funksjonen til 1. Et 2.ordens komplementær-filter vil da bli brukt til å kompensere for bias-driften. En bruker da samme metode som vist i akselerometer-delen (11.2) til å finne estimerings-avviket. Dette er som kjent en vektor, som deretter kan kjøres gjennom en PI-regulator. En får da ut en vektor, som kan legges direkte til vinkelhastigheten fra gyroene.

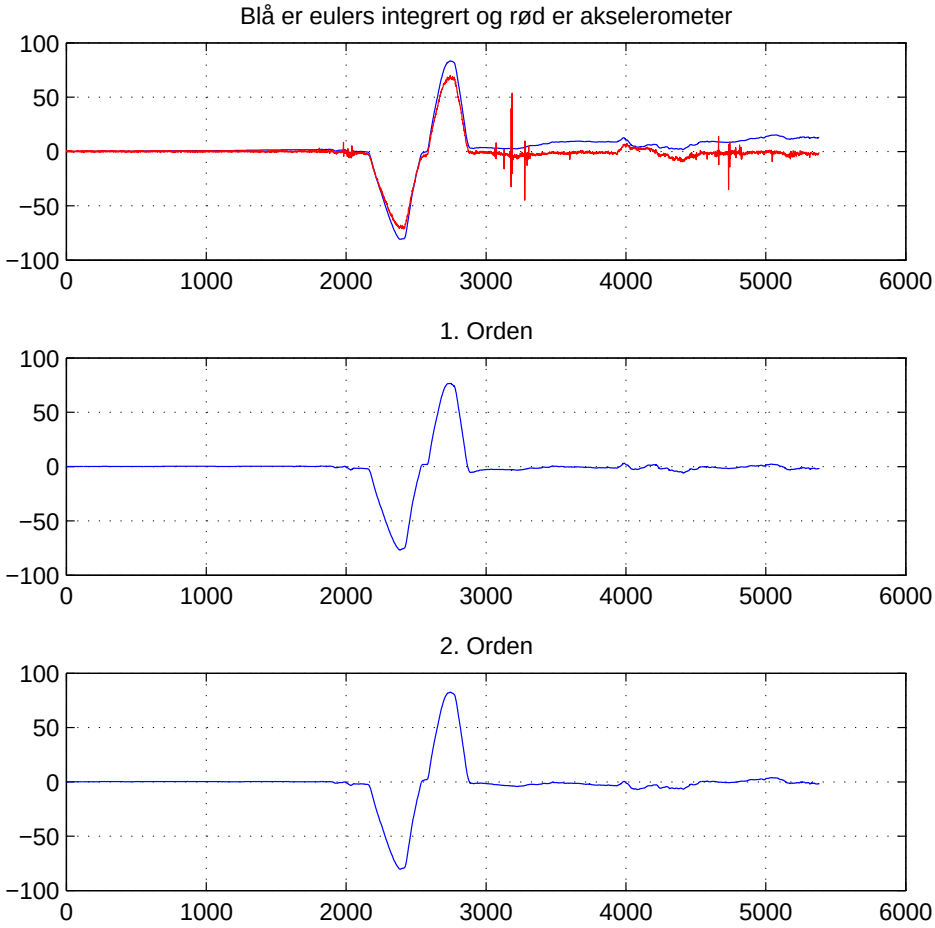


Figur 13.6: Filtrert data fra gyroene. Det blå plottet er orginaldata, mens det røde plottet er filtrert

Også her gir metodene nokså like resultater. Det er selvsagt vanskelig å si om metoden er god nok bare ut fra et plott. Siden en ikke kan se 3D-representasjonen. Problemene som kommer på grunn av overgangen mellom rotasjonsmatrise og Eulers-vinkler, kan også gi et noe feil bilde. Men siden den ulineær representasjonen krever svært mye mer enn den lineære. Kan en allikevel si at bakdelene ved å ofre sampletiden er større enn en tilnærming rundt et arbeidspunkt. Det er derfor valgt å anta at den lineære representasjonen med en tilnærmingen rundt et arbeidspunkt er god nok. Spesielt siden helikopteret hovedsakelig skal operere rundt hover-tilstand. Hadde en jobbet med et fly ville en måttet brukt den ulineære metoden, siden en der vanskelig kan definere et arbeidspunkt. Det er derfor valgt å bruke den lineære representasjonen videre i oppgaven. Selv om denne er en tilnærming bør den gi gode resultater rundet hover tilstand.

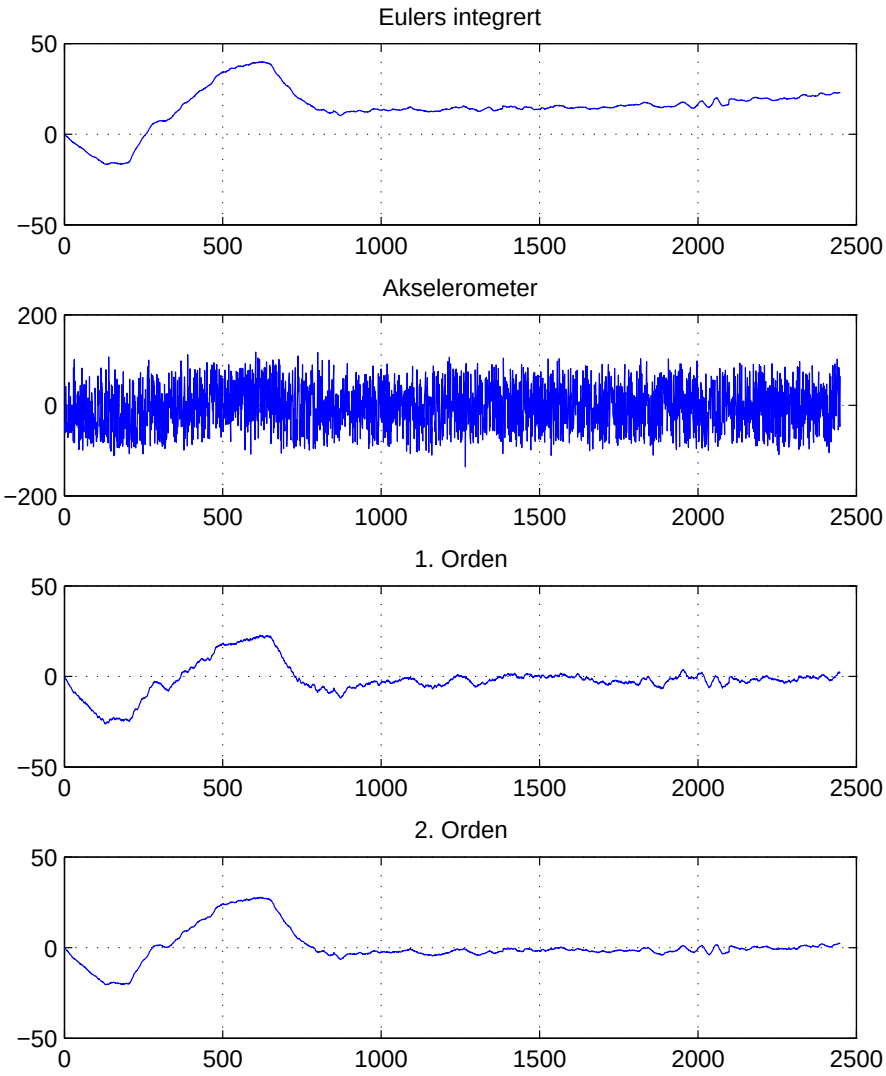
13.2.3 1.orden og 2.orden

Det er nå blitt valgt å bruke den enkle representasjonsmetoden. På denne kan en bruke både 1. og 2. ordens komplementær-filter. Det vil derfor her bli gjort en sammenligning av disse. En vet at 1.ordens filteret trenger mindre regnekraft, det må derfor være gode forbedringer for å bruke 2.ordens filteret.



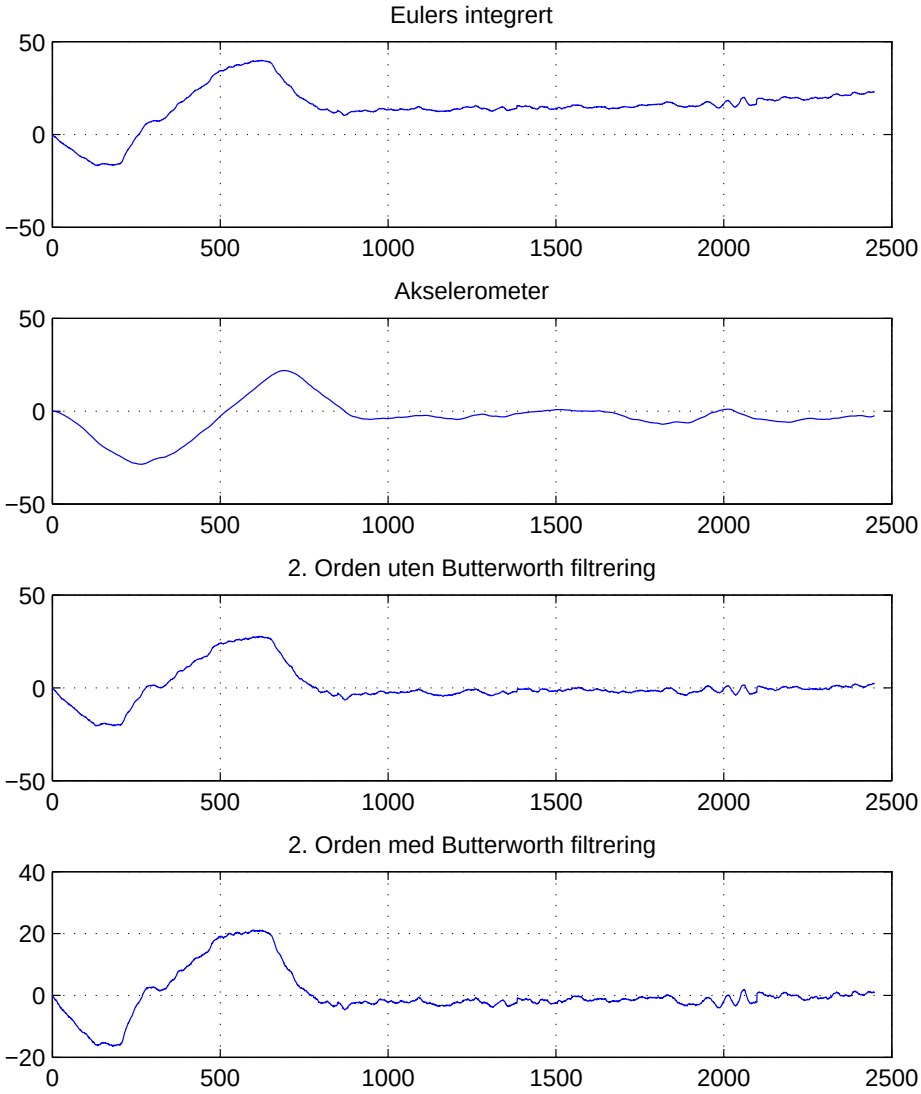
Figur 13.7: Sammenligning av 1. og 2. ordens filter

Som en ser fra testen gir begge filtrene svært gode resultater og dette gjør at en sammenligning blir vanskelig. Det blir derfor gjort videre tester med propellene på et relativt høyt turtall. Turtallet blir satt til rett rundt hvor helikopteret kan holde seg selv oppe. Akselerometer-data blir da tilsynelatende helt uleselig. En legger også merke til at det er bias-drift på Eulers integralet. Den har driftet med ca 22 grader iløpet av svært kort tid, dette vil altså være katastrofalt. Her brukes altså de to øverste plottene for å lage de to nederste.



Figur 13.8: Sammenligning av 1. og 2. ordens filter med støy

Ser nå at 2.ordens filteret gir et noe bedre resultat. Filtrene er såpass enkle at begge kan implementeres i c. Og la brukeren velge mellom dem før kompilering.
Turtallet som er brukt på motorene under testen er akkurat i grenseland for hva som skal til for å holde helikopteret i hover-tilstand. Signalet forverres derfor mer og mer jo høyere turtallet blir. Selv om komplementær-filteret gjør en god jobb kan data hvor turtallet er høyt fortsatt ikke brukes. En må altså ha betydelig forbedring for å kunne bruke akselerometer-data. Dårlig akselerometer-data gjør mer skade enn forbedring.
Neste steg blir derfor et forsøk på å forbedre signalet ved hjelp av Butterworth-filteret utviklet over. I figur 13.9 vises akselerometer-data butterworth-filtrert. De to siste plottene viser komplementær-filteret. Hvor det ene bruker filtrert akselerasjons-data, mens det andre bruker ufiltrert.



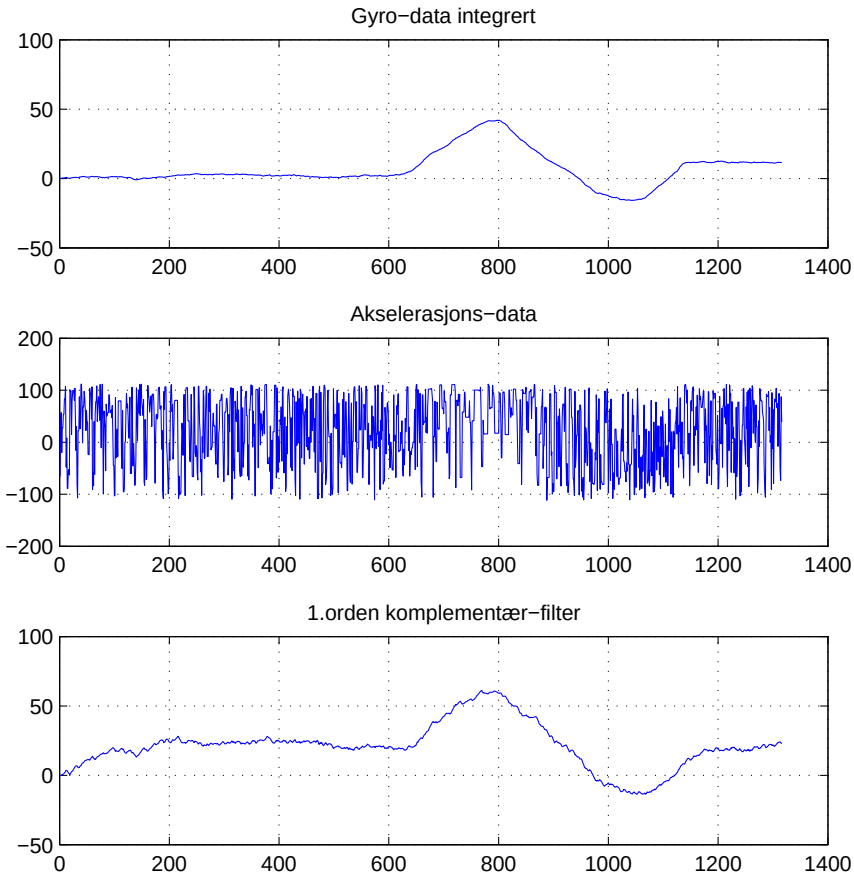
Figur 13.9: Sammenligning med og uten Butterworth-filter

En kan her se at Butterworth-filteret gir en klar forbedring av rådata. Det gir derimot ikke noe betydelig utslag på orienteringen. Derimot er data blitt forverret etter bruk av filteret. Dette filteret vil altså bare skape ekstra regne operasjoner og ble derfor droppet. Softsensoren gjør en bra nok jobb alene.

13.3 Oppsummering

Testene over viser altså at det er mest hensiktsmessig å bruke den enkle lineære representasjons-metoden. Sammen med et 2.ordens komplementær-filter for å kompensere for bias-driften til gyroen. Testene har vist at denne metoden bør være tilstrekkelig for å oppnå stabil hover. Hvis en ønsker bedre metoder, må en ha en kraftigere prosessor.

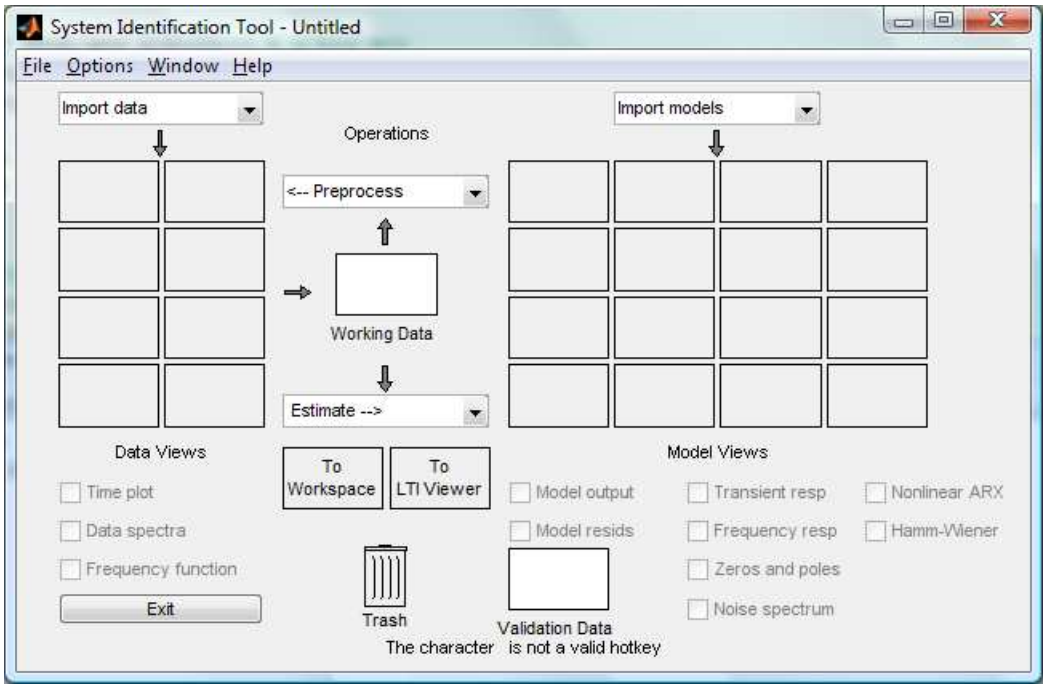
Når systemet var ferdig implementert viste deg seg likevel at akselerasjons-data ble ubrukelig. Pådraget på motorene måtte være svært høyt for at helikopteret skulle kunne fly. Dette skapte mye vibrasjons-støy, som det ikke var mulig å fjerne. Dette har gjort det umulig å beregne orienteringen til helikoptret, mens det flyr. Dette kan en se i figuren under. Selv om gyro-data har driftet en del, er den mye bedre enn estimatet som kommer fra softsensoren. Estimatet er helt ubrukelig for videre regulering. Eneste løsning på dette er å bytte akselerometer.



Figur 13.10: Figuren viser problemene som oppstår på høyt turtall

14 Matematisk-modell

For å få inngående kjennskap til systemet kan en utvikle en matematisk-modell. Dette gir muligheter til blant annet å simulere systemet på datamaskinen. For å se hvordan det vil reagere på forskjellige pådrag. Det er også en stor fordel å ha modell når en skal gjøre regulatordesign. Siden de fleste metodene for å finne regulatorparametere ofte baserer seg på en ferdig modell. Dette kapittelet bygger på [33], [37] og [40]. Det var derfor ønskelig å finne den matematiske-modellen til systemet. Her finnes det flere fremgangsmåter. En kan starte med kraftbalansen og sette opp modellen basert på fysiske lover. Eller en kan bruke svartboks metode, hvor en tenker på systemet som en svartboks. Deretter måler en hva de forskjellige inngangs-signal gjør med utgangene. I dette tilfellet vil en ha pådraget til motorene som innganger og orienteringen til helikopteret som utgang. Når en har denne dataen finnes det flere verktøy som kan brukes for å identifisere systemet. Blant annet System Identification Toolbox i Matlab, som kan startes ved å skrive ident i kommandovinduet, se figur 14.1. Det finnes også mer avanserte metoder hvor en bruker nevrale-nett til å identifisere systemet. Dette er å foretrekke for kompliserte ulineær modeller.



Figur 14.1: Figuren viser System Identification Toolbox i Matlab

En annen metode er å finne sprang-responsen og fra dette sette opp transfer-funksjonene til systemet. For å kunne bruke disse metodene på helikopteret er en dessverre avhengig å ha et ferdig system. Helikopteret må kunne holde seg i luften lenge nok til å gjøre alle målingene. Orienterings-data er foreløpig ikke bra nok til dette, på grunn av problemene med akselerometeret. Dette må derfor bli en mulig oppgave til videreutvikling. Det ble forsøkt å bruke ident verktøyet i Matlab med helikopteret i hånden. Men resultatet ble svært dårlig å ingen modell kunne identifiseres.

En står da igjen med å måtte bruke fysiske lover til å finne modellen. Det er da vanlig å starte med å sette opp kraftbalansen. Når en jobber med å beskrive bevegelsen av et stive legeme. I vårt tilfelle helikopterkroppen. Er det vanlig å ta utgangspunkt i de ferdige fysiske formlene for dette, (rigid body dynamics), [24]. Dette gjelder også modellering av fly. Fra dette kan det settes opp en generell beskrivelse av bevegelsen til et objekt. Her finnes det formler for å beskriver vinkelhastighet, hastighet og posisjon. Fra dette kan en sette opp en modell som passer til helikopteret. For å sette opp denne modellen må en vite hvilke krefter som påvirker systemet. Helikopteret har som kjent seks frihetsgrader, som styres av fire pådrag. Bare ut fra dette kan en si at det blir en komplisert modell. Dynamikken til systemet er heller ikke dekoblet. Senker en foreksempel hastigheten til den venstre motoren, vil dette resultere i en rotasjon om X-asken(roll). Dette vil igjen føre til at helikopteret begynner å bevege seg mot venstre. Når en motor går senere vil dette også gjøre at yaw begynner å endre seg. Endring av ett pådrag resulterer altså i minst tre endringer i orientering og posisjon. Dette viser at mulighetene for å beskrive systemet ved hjelp av lineære ligninger er små.
Bevegelse skapes av de fire motorene, som driver propellene. Disse skaper fire krefter som går i samme retning som Z aksen på helikopteret. Kraften propellene skaper motvirkes av tre krefter, gravitasjon, treghet og luftmotstand. I tillegg til disse tre vil også motoren skape en gyro effekt. Siden de er masse som roterer rundt et massesenter, vil dette motvirke bevegelse i både roll og pitch retningen. Også dette viser at systemet ikke er dekoblet. Skal en finne den matematiske-modellen må en finne fysiske formler for alle disse kreftene. Dette er altså en svært krevende oppgave. Dette vil trolig være nok arbeid for en hel master-oppgave.

Dette arbeidet er heldigvis allerede gjort av andre. Beskrivelsen av hvordan modellen er funnet er dessverre noe kort. Og det hadde derfor vært en fordel å laget sin egen for en fullstendig forståelse, men dette er det som sagt ikke tid til i denne oppgaven. Den ferdige matematiske-modellen ble funnet i [40]. Og gir en fullstendig beskrivelse av både vinkelhastighet, posisjon, hastighet og orientering. Siden det i vår oppgave bare er tatt hensyn til orientering er kun dette tatt med i formlene under.

$$\begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} = \begin{bmatrix} \left(\frac{J_{yy}-J_{zz}}{J_{xx}}\right)\omega_z\omega_y + \left(\frac{d\cdot C_f}{J_{xx}}\right)(\Omega_2^2-\Omega_4^2) - \left(\frac{J_r}{J_{xx}}\right)\omega_y(\Omega_2+\Omega_4-\Omega_1-\Omega_3) \\ \left(\frac{J_{zz}-J_{xx}}{J_{yy}}\right)\omega_z\omega_x + \left(\frac{d\cdot C_f}{J_{yy}}\right)(\Omega_3^2-\Omega_1^2) - \left(\frac{J_r}{J_{yy}}\right)\omega_x(\Omega_2+\Omega_4-\Omega_1-\Omega_3) \\ \left(\frac{J_{xx}-J_{yy}}{J_{zz}}\right)\omega_x\omega_y + \left(\frac{C_r}{J_{zz}}\right)(\Omega_1^2+\Omega_3^2-\Omega_2^2-\Omega_4^2) \end{bmatrix} \tag{14.1}$$

$$\begin{bmatrix} \dot{\varphi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} \omega_x + \omega_y sin(\varphi)tan(\theta) + \omega_z cos(\varphi)tan(\theta) \\ \omega_y cos(\varphi) - \omega_z sin(\varphi) \\ \omega_y sin(\varphi)sec(\theta) + \omega_z cos(\varphi)sec(\theta) \end{bmatrix} \tag{14.2}$$

Her er J - treghetsvektor, ω - vinkelhastighet. Videre er d,C_f,C_r,J_{re} konstanter. Pådraget blir da $\Omega_{1,2,3,4}$ som er vinkelhastigheten til rotorene.

Dette gir en generell matematisk modell for et fire motors helikopter. For at modellen skal kunne beskrive helikopteret i denne oppgaven, må parametrene identifiseres. Dette blir sett på under.

14.1 Parameter identifikasjon

For å kunne bruke modellen til noe fornuftig må parameterne identifiseres. Det finnes flere alternative metoder for å gjøre dette. En metode å gjøre dette på er å måle alle variablene direkte. En må da bygge spesielle rigger, som kan brukes til målingen. Skal en foreksempel måle kraften motorene gir, basert på spenningen inn. Kan en bygge en rigg som måle løftekraften motorene gir for forskjellige spenninger. Og på denne måten finne parameterne. Dette er svært tungvindt. En må da bygge rigger for å måle alle parametrene i modellen. Skal en da bytte noe må alle målingene gjøres på ny. En annen måte blir foreslått i [40]. Her brukes det avanserte CAD-verktøy, til å beregne blant annet tregheten til systemet. Også ved denne metoden må dette gjøres på ny hver gang noe byttes på helikopteret. Metodene over vil også bare gi en tilnærming til virkelig verdi, slike metoder vil derfor alltid gi en viss usikkerhet. Metodene vil også gjøre det svært vanskelig for en lekmann, som ønsker å bygge sin egen versjon av helikopteret.
Det finnes en annen metode, som vil være enkel for hvem som helst å bruke. Denne metoden vil også gi modell-parametere som passer til helikopteret det ble laget med. Metoden går ut på at en lar helikopteret fly en runde. Mens helikopteret er i luften måler en inngangene og utgangene til systemet. En gir altså pådrag til motoren og ser deretter hva som skjer med orienteringen. Dette kan så overføres via radiolinken til datamaskinen. På datamaskinen kan det nå brukes en brute-force metode. Hvor en tester alle mulige parametere i modellen. For å teste de forskjellige parameterne sette en pådragene hentet fra den virkelige testen inn på modellen. En kan da sammenligne utgangen av modellen med virkelig verdi hentet fra helikopter testen. Når en har funnet de parametrene som passer best med virkelig verdier. Har en altså funnet modell parametrene. Det kan skrives egen programvare på datamaskinene som gjør nettopp dette. Det vil gjøre det enkelt for hvem som helst å identifisere parameterne for sitt helikopter. Alt en trenger å gjøre er en rask flytur for å samle data. Deretter er det bare å vente på at datamaskinen skal bli ferdig å identifisere parametere.
Problemet med metoden er at det er en svært prosessor krevende arbeid. Det kan derfor ta svært lang tid. Men siden personlige datamaskiner nå begynner å bli svært kraftige. Kan slike operasjoner kanskje allikevel gjøres av en vanlig datamaskin. Skulle oppgaven allikevel vise seg å bli for stor kan en bruke programvare, som lar flere datamaskiner samarbeide. En kan på denne måte få hjelp av venner og bekjente. Hadde en foreksempel latt alle datamaskinene på Universitetet samarbeide, hadde en fått en enorm regnekraft.

Det finnes gratis programvare som brukes til nettopp slike formål. Programmet kalles BOINC(Berkeley Open Infrastructure for Network Computing),[7] se figur 14.2.Vanlige brukere utnytter langt fra det fulle potensialet til datamaskinen. Dette programmet kan da brukes til å utnytte den resterende prosessorkraften til noe nyttig. Programmet brukes nå til alt fra å lete etter liv i rommet til kreft forskning. For å bruke BOINC settes det opp en server som deler ut oppgaver til alle klientene. Klient programmet installeres enkelt på de datamaskinene, som ønsker å donere prosessortid. Hver maskin får da en lite del av oppgaven å jobbe med. Siden akselerometeret ikke er bra nok for oppgaven, vil heller ikke orienteringsdata bli god nok til å gjøre nettopp dette. Det kan derfor gjøres i en senere master-oppgave. Dette ville blitt en svært spennende oppgave. Det finnes også mye hjelp til hvordan dette kan settes opp på, [7].

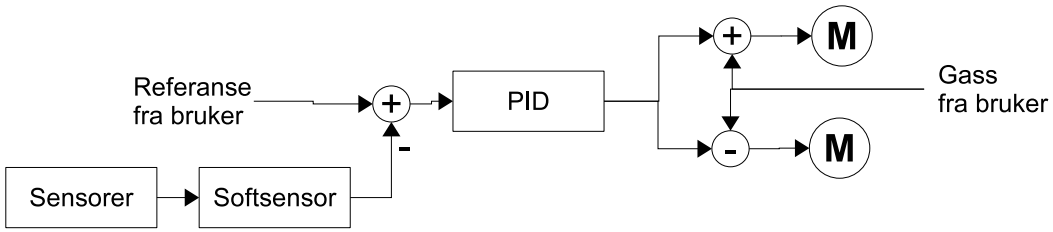
Figur 14.2: Figuren viser BOINC-klienten, som jobber med kreftforskning

15 Regulatordesign

For at helikopteret skal kunne fly må det nå utvikles et regulator-system. Dette er et stort og tidkrevende emne. Men siden regulator-design ikke er et hovedmål i dette prosjektet er det valgt å bruke PID-regulatorer. Det er denne som er blitt brukt i studiet og den er også svært vanlig i industrien. Andre alternativer kunne vært MPC, SDRE,[35] og så videre. Her finnes det et hav og velge mellom.
PID-regulatorer er som kjent beregnet for lineære system. Som en har sett i modell kapittelet, er dette systemet langt fra lineært. Men siden en allerede har begrenset systemet ved å bruke en orienterings-representasjon som kun er gyldig i et arbeidspunkt vil ikke dette være et problem. Helikopteret skal som kjent hovedsakelig jobbe rundt hover-tilstand. Regulatoren blir derfor basert rundt et arbeidspunkt. En kunne eventuelt brukt gain-scheduling, hvor en setter nye regulator parametere etter som en flytter seg over i nye arbeidspunkt. Det finnes også regulatorer som er designet for ulineær system(SDRE), men dette blir ikke nødvendig her.
I modell kapittelet kan en også se at systemet er langt fra dekoblet. En har altså et multivariabelt reguleringssystem, hvor en inngang påvirker flere utganger. For å løse dette må en bruke dekobling. Det finnes flere måter å gjøre dette på, men felles for alle er at en trenger en ferdig modell for å designe dekobleren. Dette blir derfor ikke mulig i denne oppgaven og reguleringen vil derfor ikke bli perfekt.
En har nå tre målinger og fire pådrag. Det trengs derfor en kreativ løsning for å kunne styre helikoptret. For at brukeren skal kunne kontrollere alle seks frihetsgradene må han kunne styre roll, yaw, pitch og i tillegg ha et gass-pådrag. Gass-pådraget brukes for å styre hvor hurtig helikopteret går opp eller ned. Det skal ikke være en egen regulator for dette, men brukeren skal kunne styre det manuelt. For orienteringen er det ønskelig å ha egne regulatorer. En trenger altså et minimum av tre PID-regulatorer, disse skal styre fire pådrag. Hvis en da tenker tilbake til hvordan helikoptret styres. Ser en at et motor par styrer roll og det andre styrer pitch. En setter altså ned hastigheten på den ene motoren og opp hastigheten på den andre. Hvis en da lar regulatoren gi ut en balanse verdi, som legges til det ene pådrag og trekkes fra det andre. Kan en styre roll og pitch ved hjelp av to regulatorer. Yaw kontrolleres ved å la de to motorparene rotere hver sin vei. Hvis et par går raskere enn det andre vil en få en endring i yaw. En kan derfor la en siste regulator gi ut en balanse verdi som trekkes fra et motor par og legges til det andre. Gass-pådrage setter hastigheten til rotorene og balanseverdiene trekkes fra eller legges til denne. En kan da regne ut det fire motorpådragene ved hjelp av ligningene under. Oppsettet for å kontrollere roll eller pitch kan også ses i figur. De to siste regulatorene er ikke tatt med for å gjøre figuren oversiktlig.

$$\text{pådrag } 1 = \text{gass} - b_r + b_y \tag{15.1}$$
$$\text{pådrag } 2 = \text{gass} + b_r + b_y \tag{15.2}$$
$$\text{pådrag } 3 = \text{gass} + b_p - b_y \tag{15.3}$$
$$\text{pådrag } 4 = \text{gass} - b_p - b_y \tag{15.4}$$

Her er b_r - roll balansen, b_y - yaw balansen og b_p - pitch balansen



Figur 15.1: Figuren viser hvordan en kan kontrollere roll eller pitch.

Atmel har et eksempel på hvordan en diskret PID-regulator kan implementeres i c. Dette kan ses i [5] Dette er blitt bygget videre på og det er blitt implementert tre diskrete regulatorer. Disse jobber helt uten flyttall. Hver regulator bruker da 877 CPU-sykluser eller 54.81 ţS på en gjennomkjøring. Som en ser gir dette et utrolig raskt system.

15.1 Parameter-tuning

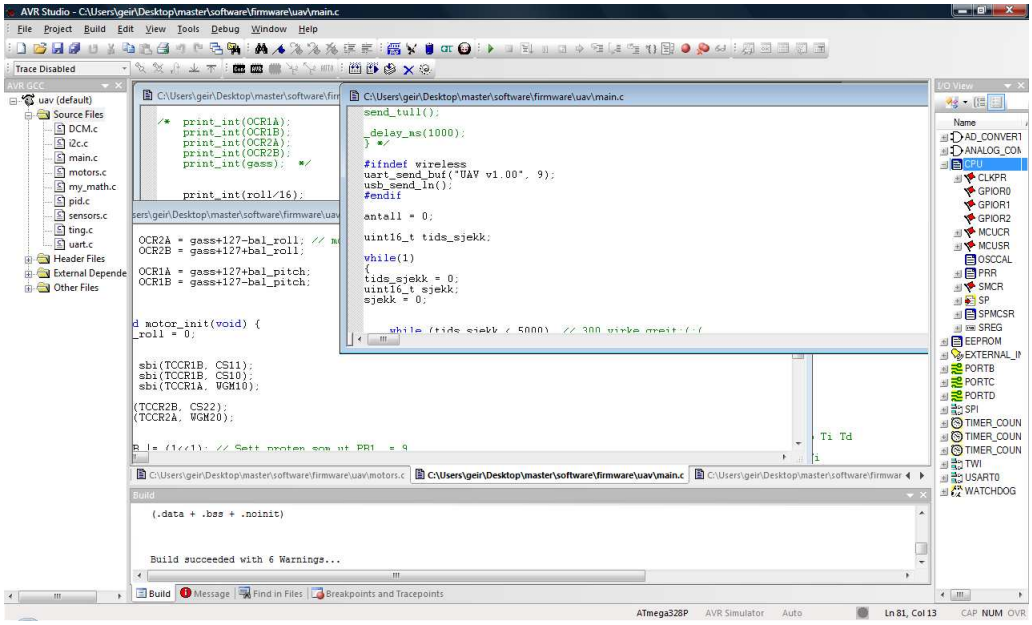
For å oppnå en stabil hover, må en tune regulatorene. De fleste metoder for å gjøre dette bygger på en ferdig modell. Også dette vil derfor ikke være mulig i denne oppgaven. Det er dessverre ikke mulig å fly helikopteret uten regulatorene innkoblet derfor er alle metoder som baserer seg på åpensløyfe uaktuelle. Det finnes også en hel del metoder som kan brukes med lukket sløyfe. En av de vanligste er Ziegler-Nichols metode. Denne metoden går ut på at en fjerner T_i og T_d leddene. Deretter øker en forsterkningen til systemet blir ustabilt. En kan nå finne kritisk forsterkning(K_c) og perioden (P_c). Deretter kan ligningene under brukes til å regne ut K, T_i og T_d direkte. Dette er hentet fra [5].

$$P = 0.6K_c \tag{15.5}$$
$$T_i = 0.5P_c \tag{15.6}$$
$$T_d = 0.12P_c \tag{15.7}$$

Denne metoden ble forsøkt på helikopteret, men siden det er problemer med akselerometeret kunne K_c og P_c dessverre ikke identifiseres. Det samme gjelder alle andre åpensløyfe metoder, som er blitt funnet. Parametrene ble derfor funnet ved hjelp av en prøv og feil metode. Dette sammen med at akselerometer problemene gjorde at systemet aldri ble helt stabilt.

16 Firmware

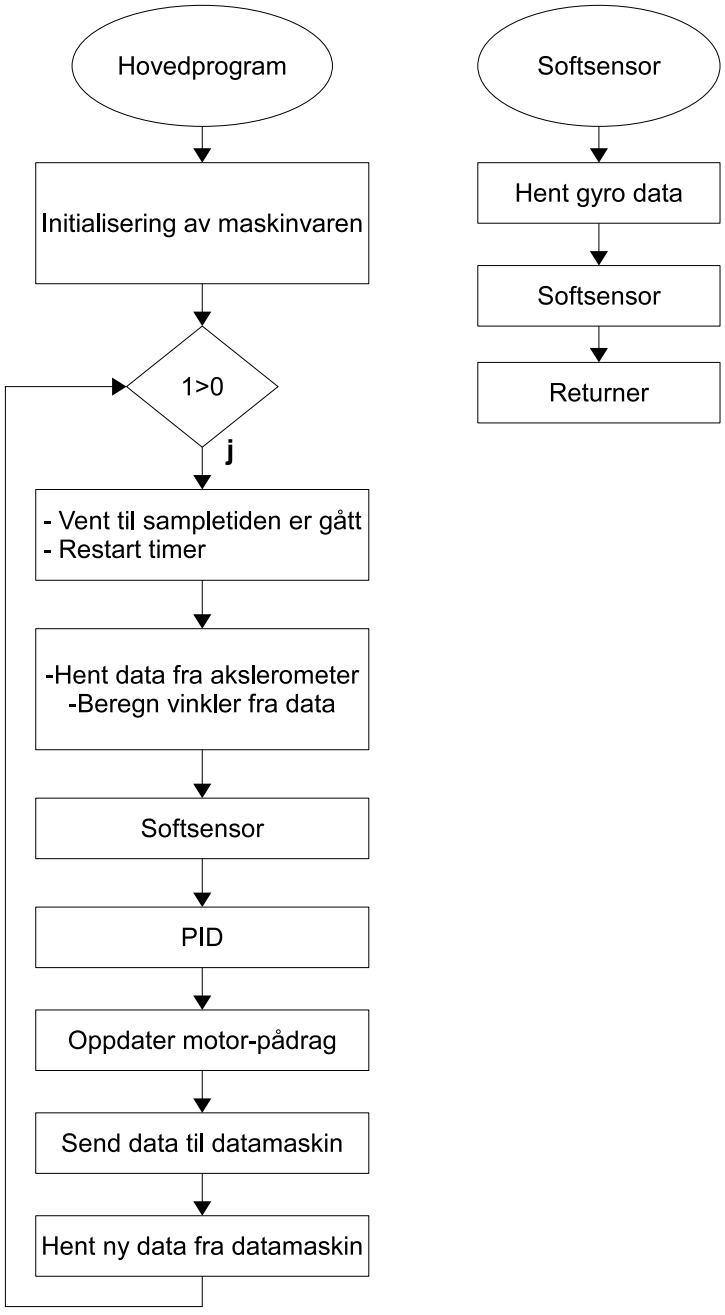
Alle valgene rundt hvordan systemet skal implementeres er nå tatt. Neste steg blir å skrive programvaren til helikoptret. Alle funksjonene som skal implementeres er allerede gjennomgått tidligere i oppgaven. Implementeringen på mikrokontrolleren er såpass lik implementeringen i Matlab, at dette var en enkel sak.
Programvaren ble skrevet i AVR studio4 [6], som er et gratis utviklingsverktøy fra Atmel. Det var ønskelig å skrive programmet i c, siden utvikling i ASM tar unødvendig lang tid. Derfor trengtes også en kompilator, WinAVR [30]. Dette er en gratis kompilator med åpen kildekode. Den gir AVR studio støtte for c/c++. Utviklingsprogrammet kan ses i figur 16.1.



Figur 16.1: Figuren viser utviklingsprogrammet brukt i oppgaven.

En viktig faktor i program utviklingen var å gjøre programmet så raskt som mulig. Mye av tiden har derfor gått med til å unngå flyttall. Det er i programmet gitt mulighet for å velge mellom USB eller den trådløse-modulen. I testene er det hovedsakelig USB, som er blitt brukt. Siden denne har vist seg å være mest pålitelig. Det er også mulig å gjøre filter valg. Alt dette gjøres ved hjelp av definisjons setninger (#define). Dette gjør at kompilatoren fjerner de delene som ikke skal brukes. På denne måten bruker ikke de delene som er utelatt plass på mikrokontrolleren.

Programmet jobber i en hovedløkke, ved hver gjennomkjøring er det en vente-rutine som venter til den satte sampletiden er gått. Sampletiden måles ved hjelp av timer0. Dette gjør at hver gjennomkjøring tar like lang tid. Etter dette kjøres alle metodene en etter en. Dette kan en best se i det forenklede blokk-diagrammet under. Programmet er testet og fungerer svært bra.



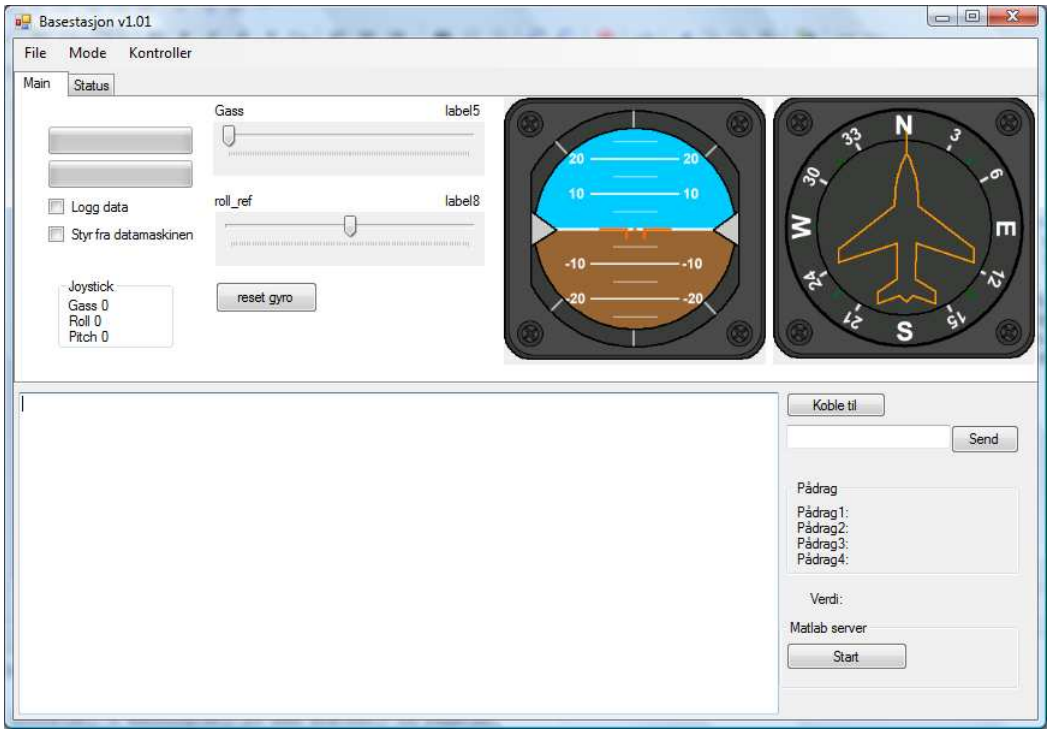
Figur 16.2: Figuren viser blokk-diagram av programmet som kjøres på mikrokontrolleren

16.1 Kalibrering av hastighets-kontrollere

Da testingen av motorene begynte var det mye problemer med at de startet på helt forskjellige pådrag. De gikk heller ikke likt. Det viste seg at hastighets-kontrollerne måtte kalibreres. Det ble derfor skrevet en egen kalibrerings-sekvens inn i programvaren. Denne brukes for at alle fire motor-kontrollerne skal kalibreres likt. For å bruke sekvensen kobles helikopteret til gjennom USB. På datamaskinen bruker en et terminalprogram, for å kommunisere med helikopteret. Deretter kommer det opp melding om at systemet er klart på datamaskinen. Batteriet kobles så til helikopteret, motordriverne gir en lyd som indikerer at de er klare. Deretter trykker brukeren en tast på tastaturet for å kalibrere. Kalibreringen skjer da automatisk. Denne algoritmen ligger i firmwaren, men er kommentert bort. For å bruke algoritmen fjernes ganske enkelt kommentar tegnene.

17 Hovedprogrammet

Basestasjons programvaren var allerede startet på i forprosjektet, [36]. Den er iløpet av masteren blitt satt opp med noen nye funksjoner. Programmet hadde allerede mulighet for kommunikasjon med basestasjonen. Dette ble også endre litt på slik at programmet også kunne kommunisere dirkete med helikopteret gjennom USB. Som nevnt tidligere er USB raskere og informasjonen må ikke gå gjennom like mange ledd. USB blir derfor mer pålitelig. Neste steg var å gjøre den mottatte informasjonen enkelt tilgjengelig for brukeren. Dette ble gjort ved å sette instrumenter på hovedvinduet, se figur 17.1. Disse viser orienteringen til helikopteret og oppdateres hver gang det kommer ny data fra helikopteret. Det var også behov for å kunne lagre den informasjonen som kom fra helikopteret, slik at den skal kunne brukes i Matlab. Dette ble løst ved at programmet skriver alt det mottar til en egen fil kalt logg.txt. Denne funksjonen kan skrues av og på ved hjelp av avkryssingsboksen i hovedvinduet. Resten av funksjonene kan ses i, [36].



Figur 17.1: Figuren viser hovedprogrammet

Neste utfordring som måtte løses, var å gi brukeren en enkel måte å sende data til helikoptret. Slik at han rask skal runne endre referansene på regulatorene Det ble allerede i forprosjektet utviklet en metode for å styre helikopteret ved hjelp av kontroller. Denne metoden gikk ut på å koble kontrolleren direkte til basestasjonen. Denne dekoder PPM signalet fra kontrolleren og sender verdiene videre til helikopteret. Denne metoden fungerer fint, men det var ikke mulig å sende data via USB. Basestasjons programvaren ble derfor utvidet med mulighet for å koble til kontroller (joystick), se figur 17.2.

Figur 17.2: Figuren viser kontrolleren som brukes for å styre helikopteret

For å bruke kontrolleren var det nødvendig med et grensesnitt mellom programvaren og kontrolleren. Et slikt grensesnitt eksisterte allerede, [15]. Men det var dessverre svært dårlig skrevet og passet dårlig til vårt formål. Et nytt grensesnitt som bygger på dette ble derfor utviklet. Grensesnittet ble skrevet som et eget objekt, slik at det enkelt kan gjenbrukes av andre. Det ble også skrevet en kalibrerings-algoritme. Denne brukes som navnet tilsier til å kalibrere kontrolleren, slik at pådragene går fra null til ønsket verdi. Kalibreringen gjør en da enkelt ved å velge Kalibrering under Kontroller fanen. Deretter beveger en alle pådragene fra null til fullt. Programmet lagrer deretter kalibrerings-verdiene slik at dette bare må gjøres en gang.
Det er nå altså mulig å koble en joystick til datamaskinen ved hjelp av USB-porten. Brukeren kan altså selv velge den joysticken han føler seg komfortabel med.

18 INS

INS er ikke implementert i denne oppgaven. Men det er det neste naturlige steget for en slik oppgave. Og blir derfor dekket her. INS står for Inertial navigation system og brukes for å finne posisjonen til helikopteret. Det finnes flere metoder for å gjøre dette noen er nevnt under.
Å kjenne posisjonen nøyaktig vil være avgjørende for å kunne navigere og holde posisjon. Dette vil blant annet være et godt hjelpemiddel i vind. Selve possisjons-bestemmelsen kan gjøres i en egen modul, som gir beskjed til hovedsystemet hvor mye orienteringen må endres for å komme til en bestemte posisjon. Det er i denne delen av oppgaven at rotasjonsmatrisen virkelig hadde vært nødvendig. Denne metoden beskriver som nevnt orientering, men kan også beskrive posisjon ved hjelp av translasjon.

18.1 GPS

Den mest nærliggende metoden er å bruke GPS-mottaker. Denne vil være svært enkelt å implementere og gir posisjon og høyde direkte. Problemet med denne metoden er at den ikke gir en nøyaktig posisjon. En GPS-mottaker som er nøyaktig ned til en meter regnes som svært bra. Mange er dessverre ofte opp imot 10m. Et annet problem er at sampletiden vil bli svært lav. Fra en god GPS kan data hentes med en frekvens på 10Hz, med andre ord ikke særlig ofte. Med begge disse svakhetene vil det være vanskelig å bestemme hvor helikopteret egentlig står. Dette kan likevel brukes på steder hvor feilmarginen kan være stor. Hvor det altså ikke er fare for å komme borti noe. Men til navigasjon i trangere områder vil dessverre dette systemet være ubrukelig.

18.2 Akselerometer

En annen mulighet er å bruke akselerometer til å bestemme posisjon. Det finnes allerede akselerometer som måler alle tre aksene på helikopteret. Dette kan altså gjøres uten å tilføre flere sensorer. Som kjent er dobbelt integralet av akselerasjon avstanden en har beveget seg. For å finne posisjonen må en altså bare vite retningen en har beveget seg i. Dette kjenner en også gjennom orienteringen. En trenger altså ingen endring i hardware for å bruke denne metoden. Metoden brukes blant annet i den sivile fly-trafikken. Boing 737 bruker denne metoden for å navigere, [Kjell Sverre Voll]. Problemet med metoden er at dobbelt integral gjør at selv små feil raskt vil akkumuleres til enorme feil. En kan derfor ikke stole blindt på denne målingen alene.

18.3 Kamera

En svært interessant måte å navigere på er ved hjelp av kamera. En bruker ett eller to kamera, som peker i Z-retningen. Kameraet ser altså bakken. En kan da bruke en algoritme som velger ut punkter , som er lette å kjenne igjen (vei punkter). Dette gjør det mulig å bruke korrelasjons algoritmer til å kjenne igjen punktene. Ut fra bevegelsen til punktene kan en da beregne posisjonen til helikopteret. Bruker en to kamera kan en også beregne høyden. Systemet kan også huske punkter å bruke denne informasjonen til å kjenne seg igjen på steder den har vert før. Det finnes mye forskning rundt slike metoder da spesielt for kjøretøy. En felles betegnelse er SLAM (Simultaneous Localization and Mapping). Denne metoden er beskrevet i [38].
En annen fordel med denne metoden er at en også kan hente svært mye annen informasjon gjennom videoen som hentes inn. Den kan blant annet sendes tilbake til brukeren på bakken, slik han han får en sanntids video-feed.
Problemet med denne metoden er at det er svært store datamengder som skal behandles. Det vil altså være svært krevende for prosessoren og vil kreve mye mer enn en enkel mikrokontroller. Dette vil heller være et alternativ hvis en velger å bruke en sanntids linux prosessor, se kapittel 5.

18.4 Softsensor

Også i dette tilfelle kan en implementere systemet ved hjelp av en softsensor. En ser at GPS gir et relativ nøyaktig resultat, men har lav samplerate. Akselerasjons-metoden kan gjøres ved høy samplerate, men gir feil over tid. En har altså samme problem som en hadde i kapittel 12. Problemet kan også her løses ved å bruke en softsensor for å estimere posisjonen. En kan også kombinere dette med kamera data for å få et veldig nøyaktig resultat.

18.5 Beregning av høyde

Å kunne måle høyden til helikopteret vil være svært anvendelig for å gjøre systemet autonomt. Når en skal endre orienteringen vil det ofte være et problem at også høyden endres. Har en da en høydemåling kan en bruke en egen regulator for å holde en bestemt høyde. Som nevnt over kan høyden finnes både ved hjelp av GPS og kamera. Men det finnes også flere metoder for å gjøre dette.

18.5.1 Sonar

En sonar-modul kan monteres under helikopteret slik at den peker i Z retningen, altså mot bakken. Denne sender en ultralyds puls mot bakken. Pulsen reflekteres og sensoren måler hvor lang tid det går før pulsen kommer tilbake. En kan da enkelt bruke denne tiden til å beregne avstanden til bakken. Denne metoden ble brukt i bacheloren, [38] og blir nøye beskrevet der.
Problemet med denne metoden er at den er svært avhengig av underlaget. Siden en skal kunne bruke disse dataene i en regulator er dette et stort problem. Skal en foreksempel fly over gress eller en dal vil helikopteret tro at høyden er stor å gå ned i dalen. Vanlige sonar-moduler fungerer heller ikke på lengre avstander enn 6m.

18.5.2 Barometer

Som kjent vil trykket endre seg ettersom helikopteret endrer høyde. En kan da bruke et barometer for å måle trykke. Når en kjenner trykk-differansen mellom bakken og helikopteret kan en enkelt regne ut høyden til helikopteret. Dette vil gi et godt mål på høyden, som ikke varierer etter underlaget. Innendørs vil derimot trykke kunne variere veldig, dette er derfor en sensor som kun kan brukes utendørs. Et barometer er lite og lett og kan derfor enkelt monteres på et kretskort.

19 Kommunikasjons-modul

For å kunne styre helikopteret fra bakken, måtte det implementeres et kommunikasjons-system. Siden systemet er designet slik at det skal være mulig å legge til moduler. Ble det valgt å designe en egne modul som kan kommunisere trådløst med datamaskinen. Det er flere gode alternativer når det gjelder kommunikasjons-system og denne metoden vil derfor gjøre at disse enkelt kan byttes på. Nye kommunikasjons-systemer kan også utvikles uten kjennskap til hovedsystemet. Det vil under bli gjennomgodt et par mulige metoder for å kommunisere med helikopteret.

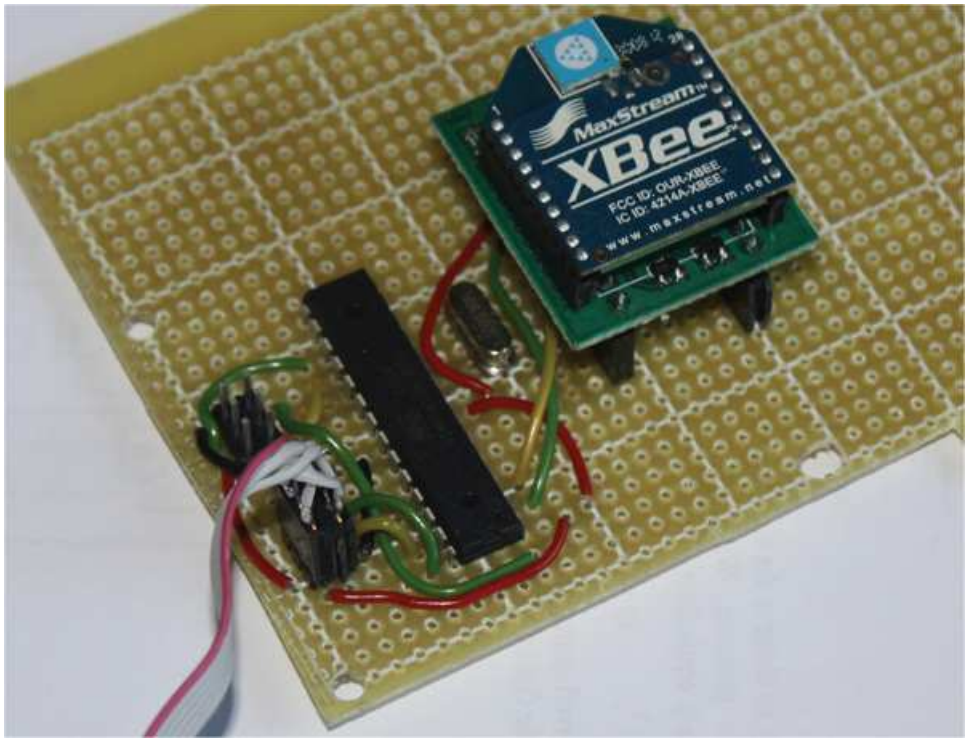
19.1 Analog-mottaker

Her brukes en vanlig analog-mottaker av samme type som vanligvis brukes på radiostyrte helikopter. Denne kan motta data fra kontrolleren i figur, 17.2 Denne har fra fire til seks kanaler ut fra disse kommer et PPM-signal. Dette signalet kan enkelt leses av en mikrokontroller. Bruker en denne modulen kan altså helikopteret styres, som et vanlig radiostyrt helikopter. Ved bruk av denne metoden er det dessverre ikke mulig å sende data tilbake. For å løse dette kan en koble en minnekort av samme type som brukes i digitale kamera til gjennom SPI-porten på mikrokontrolleren. All data kan da lagres på dette for å senere bli analysert og behandlet. Det ble kjøpt en analog-mottaker fra dealextreme.com. Denne ble dessverre levert så sent at det ikke ble tid til å implementere dette systemet. Den kan derfor være en ide til en senere oppgave.

19.2 Zigbee

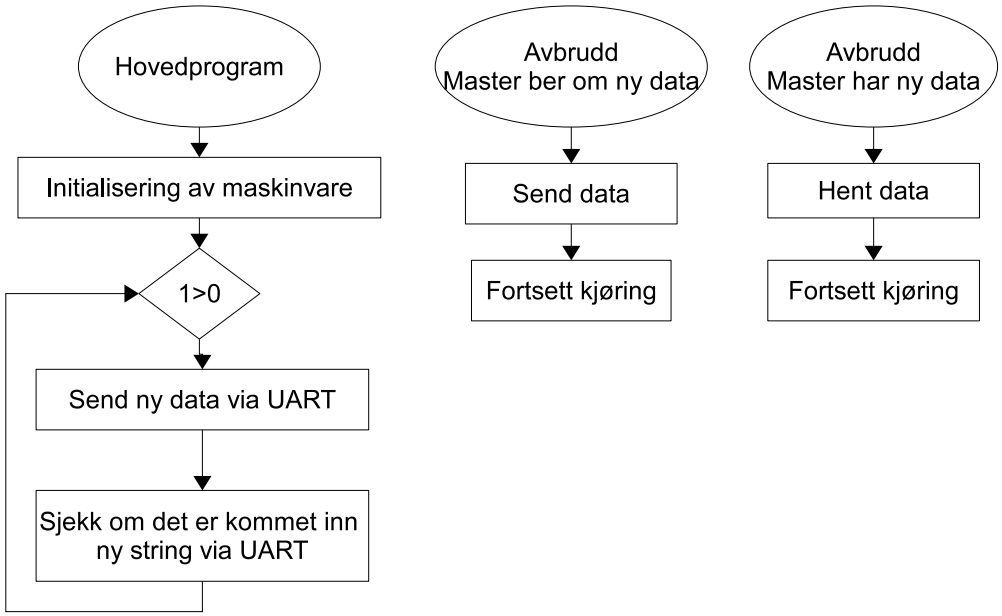
I forprosjektet ble det utviklet en basestasjon som har mulighet for å kommunisere med et eksternt system gjennom Zigbee-protokollen, [36]. Hele systemet på bakken var altså ferdig utviklet. Radiomodemet som skulle brukes var også allerede valgt i forprosjektet. Det gjensto da bare å utvikle en modul, som kunne kommunisere gjennom dette. Siden modulen skulle kobles inn på I2C-nettverket måtte den ha egen mikrokontroller. Alt en trenger for å bygge modulen er altså et radiomodem og mikrokontroller. Radiomodemet kobles til mikrokontrollene via UART og mikrokontrolleren kobles til hovedsystemet via I2C. Mikrokontrollerens oppgave blir da bare og formidle informasjon mellom hovedsystemet og radiomodemet. Fordelene med å ha et slikt system er at sending og mottak av data er en relativt prosessor og minne krevende oppgave. Når systemet implementeres slik fjernes denne oppgaven fra hovedsystemet.

Det ble valgt å bruke ATMEGA168,[4] fra Atmel. Dette er forgjengeren til den som brukes i hovedsystemet. Den har noe mindre lagringsplass, men er også billiger. Ellers er den lik på alle måter.
Modulen kobles som nevnt tidligere til hovedsystemet ved hjelp av en flatkabel. Denne gir tilgang til 5v, 3.3v, jord, data og klokke linje. En trenger altså ikke egen spenningsregulator på modulen. Kretsskjema ble tegnet og kan ses i vedlegg F. Kretsen ble implementert på et eksperimentkort, resultatet kan ses i figur 19.1.



Figur 19.1: Figuren viser kommunikasjons-modulen

Det måtte også utvikles programvare til modulen. Dette ble gjort i c, på samme måte som programvaren til hovedsystemet. Programvaren ble satt opp for å være avbruddsstyrt. Et avbrudd oppstår når masteren ber om eller er klar til å sende ny data. Data blir da sendt direkte. Et avbrudd skjer også hver gang det kommer ny data til UART. Denne blir så lagt i et buffer, hvor den behandles fortløpende av hovedløkka. Data som skal sendes via UART legges først i et buffer. Deretter sendes ett og ett bit ved hjelp av et avbrudd, som kalles hver gang UART-modulen er klar for nytt tegn. For å vise dette er det satt opp et forenklet blokkdiagram som viser funksjonen til programmet.



Figur 19.2: Forenklet blokkdiagram

Kommunikasjonsmodulen er ikke blitt mye brukt i eksperimenteringen. USB er både raskere og informasjonen må ikke gå gjennom like mange ledd. Dette gjør feil søking mye enklere. Programvaren ble derfor bare satt opp som et ekperiment, for å se at det virket. Den sender derfor ikke all data kun roll og pitch vinklene. I tillegg sendes gass-pådraget.

20 Videre arbeid

Som en har sett gjennom oppgaven, er mulighetene for videre arbeid så og si uendelige. Det vil derfor her bli satt opp noen forslag til videre arbeid med oppgaven. Det finnes oppgaver som passer bra både for master og bachelor nivå.

En veldig spennende oppgave er blitt foreslått i kapittel, 14. Denne går ut på å utvikle en metode for å finne parameterne i den matematiske-modellen. Her er det blitt foreslått å bruke et program som lar flere datamaskiner samarbeide om oppgaven.

Når modellen er funnet er det mulig å gjøre et bedre regulator-design. Siden systemet ikke er dekoblet, kan en med fordel implementere et multivariabelt regulerings-system. Det kan også brukes andre mer avanserte regulatorer. Eller ganske enkelt bedre metoder for å finne regulator parameterne.

En av de mest nærliggende tingene på et slik helikopter er å montere kamera. Her kan en se på trådløst overføring av video tilbake til brukeren på bakken. Dette finnes det ferdige systemer for. Oppgaven passer derfor kanskje best som en bachelor-oppgave. Velger en å utvikle sin egen overførings-enhet kan det allikevel bli en svært avansert oppgave.

En kan også se på bruk av andre typer "hjerne". Det er i denne oppgaven blitt foreslått FPGA eller sanntids-linux systemer. Implementeres dette kan en også bruke mer avanserte algoritmer for å estimere orientering. Tenker da spesielt på utvidet-kalmanfilter.

Utvikling av en INU-enhet er også et naturlig videre steg. Denne kan utvikles som en egen modul med softsensor, som kombinerer forskjellige typer posisjons-målinger.

En siste ide er også mer bachelor rettet. Dette vil være å bygge sin egen hastighets-kontroller for motorene. Dette kan være en enhet, med mulighet for å styre alle fire motoren. Denne kan også implementeres som en egen modul, som kommunisere ved hjelp av I2C.

21 Diskusjon og konklusjon

21.1 Diskusjon

Hovedmålet i oppgaven var å beregne orienteringen til helikopteret. Dette er oppnådd, men gir feil resultat når turtallet på motorene er høyt. Dette kommer av at akselerometeret, som skal brukes for å måle tyngdeakselerasjonen også måler vibrasjonen skapt av propellene. Dette har gitt store problemer gjennom hele oppgaven. Det er foreslått flere metoder for å fjerne vibrasjons-støyen, men på høye turtall har dette vist seg å være umulig. For å løse disse problemene må det implementeres en annen type akselerometer. Når problemene ble oppdaget var det dessverre for sent å bestille nytt akselerometer. Det vil derfor her bli valgt et alternativt akselerometer for en senere oppgave.

Det er dessverre vanskelig å velge et nytt akselerometer bare ut fra data i databladet. Hvordan det reagerer på vibrasjon, er ikke godt nok dokumentert her. Men det er likevel funnet et, som kan være et godt alternativ ADXL335, [1]. Dette skal ifølge [17] være det nyeste og beste fra Analog Devices. Her er det også mulighet for å sette båndbredden akselerometeret måler, helt ned til 0.5 Hz. Går en tilbake til kapittel 9.3, ser en at denne båndbredden bør gi et tilnærmet perfekt resultat. Akselerometeret koster 20 USD fra, [17] og passer derfor også godt til bygging på budsjett.

21.2 Konklusjon

Et fire-motors helikopter er bygget og fungerer fint. Elektronikk som kan hente sensor-informasjon og styre motorene er også utviklet. Systemet er blitt bygget slik at det er mulighet for modulbasering. Videre er det vist to forslag til orienterings-representering og tre forslag til softsensor for å kombinere sensor-data. De forskjellige metodene er blitt testet mot hverandre i Matlab.

Hovedmålet i oppgaven var å beregne orienteringen til helikopteret. Det er blitt vist at denne kan representeres ved hjelp av Eulers-vinkler. Det er videre vist at det er mulig å kompensere for bias-drift ved hjelp av softsensor. Matlab testene har vist at dette best kan gjøres ved hjelp av et 2.ordens komplementær-fileter. Dette gir et svært godt estimat av orienteringen. Hovedmålet er altså nådd. Men på grunn av dårlig akselerometer-sensor. Fungerer bare metoden frem til motorene når et visst turtall. Dette turtallet er dessverre akkurat der hvor helikopteret flyr av seg selv. Dette er blitt forsøkt løst ved hjelp av støy-filtrering. Og det er blitt vist at dette gir sterke forbedringer av rådata. Men at det derimot ikke gir noen forbedring på orienterings-estimatet etter softsensor. En helt stabil hover, er derfor ikke oppnådd i denne oppgaven.

Akselerometer-problemene har også gjort det umulig å finne fullstendig modell for systemet. Men det er blitt vist en generell versjon uten parametere. Det er også blitt vist hvordan disse parameterne kan identifiseres, når en har bedre akselerometer-data. I dette kapitlet er det også vist at systemet er ulineært og at prosessen ikke er dekoblet.

At det ikke finnes modell har også gjort regulator-designet vanskelig, siden de fleste metoder bygger på en ferdig modell. Men det er implementert PID-regulatorer i c, som gir en tilfredsstillende regulering av systemet.

Modulbasering har vist seg å være en god ide, spesielt med tanke på videre utvikling. Det er blitt vist at nye moduler kan bygges uten for mye kjennskap til hovedsystemet. Det er også bygget en kommunikasjons-modul. Denne kan kommunisere trådløst med datamaskinen.

Videre har hele systemet blitt implementert på en mikrokontroller fra Atmel. Denne har vist seg å ha nok ressurser for oppgaven. Skal systemet utvikles videre er det også vist alternativer til å bruke mikrokontroller.

Referanser

[1] Adxl335. Webside. http://www.analog.com/static/imported-files/data_sheets/ADXL335.pdf.

[2] Aeroquad. Webside. http://www.aeroquad.info.

[3] arduino. Webside. http://www.arduino.cc/.

[4] Atmega328. Webside. http://www.atmel.com/dyn/products/product_card.asp?PN=ATmega328P.

[5] Atmel pid. Webside. http://www.atmel.com/dyn/products/app_notes.asp?family_id=607.

[6] Avrstudio. Webside. http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725.

[7] Boinc. Webside. http://boinc.berkeley.edu.

[8] Butterworth filter. Webside. http://en.wikipedia.org/wiki/Butterworth_filter.

[9] Butterworth kalkulator. Webside. http://www.daycounter.com/Filters/Sallen-Key-LP-Calculator.phtml.

[10] Dcm-draft. Webside. http://gentlenav.googlecode.com/files/DCMDraft2.pdf.

[11] Digital filters with avr. Webside. http://www.atmel.com/dyn/resources/prod_documents/doc2527.pdf.

[12] Eaglecad. Webside. http://www.cadsoft.de/.

[13] Hobbycity. Webside. http://www.hobbycity.com/.

[14] Idg650. Webside. http://invensense.com/mems/gyro/idg650.html.

[15] Joystick interface. Webside. http://www.codeproject.com/KB/directx/joystick.aspx.

[16] Knjn. Webside. http://www.knjn.com.

[17] Ladyada. Webside. http://www.ladyada.net.

[18] Mav-blog. Webside. http://tom.pycke.be/mav/71/kalman-filtering-of-imu-data.

[19] Mma7260qt. Webside. http://www.freescale.com/files/sensors/doc/data_sheet/MMA7260QT.pdf.

[20] Multisim. Webside. http://www.ni.com/multisim/.

[21] Paparazzi. Webside. http://paparazzi.enac.fr/wiki/Infrared_Sensors.

[22] Pid controller in fpga. Webside. http://www.atlantixeng.com/pdf-files/PID_Controller_in_FPGA.pdf.

[23] Quaternion. Webside. http://en.wikipedia.org/wiki/Quaternion.

[24] Rigid body dynamics. Webside. http://en.wikipedia.org/wiki/Rigid_body.

[25] Runge-kutta. Webside. http://www.ee.nthu.edu.tw/bschen/files/c16-1.pdf.

[26] Sagnac effect. Webside. http://en.wikipedia.org/wiki/Sagnac_effect.

[27] Sure electronics. Webside. http://stores.ebay.com/Sure-Electronics.

[28] Ts-4500. Webside. http://www.embeddedarm.com.

[29] Txs0104e. Webside. http://focus.ti.com/lit/ds/symlink/txs0104e.pdf.

[30] Winavr. Webside. http://winavr.sourceforge.net/.

[31] Ivar Austvoll. Lecture notes for mik170 image processing.

[32] Shane Colton. The balance filter. http://web.mit.edu/scolton/www/filter.pdf.

[33] S.Bouabdallah R.Siegwart D. Schafroth, C.Bermes. Modeling,system identification and robust control of a coaxial microhelicopter.

[34] F. Haugen. *Regulering av dynamiske system*. Tapir, 3 edition, 2003.

[35] Tayfun Çimen. State-dependent riccati equation (sdre) control.

[36] Geir Lunde. Forprosjekt kybernetikk.

[37] P. McKerrow. Modelling the draganflyer four-rotor helicopter.

[38] Geir Lunde og Benny Michal Andtbacka. Sonar-robot.

[39] Karl Skretting. Forelesningsnotater for mik130 systemidentifikasjon.

[40] Antal Turóczi. Flight control system of an experimental unmanned quad-rotor helicopter. Webside. http://old.bmf.hu/conferences/cinti2009/74_cinti2009_submission.pdf.

[41] Oliver J. Woodman. An introduction to inertial navigation. http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-696.pdf.

A integrer.m

```
function [ret] = integrer(inn, type, T)
%% Eulers integrator
if (strcmp(type,'sum'))
    s=0;
    for i=1:size(inn),
        s = s + inn(i)*T;
        ret(i) = s;
    end
    return;
end;
```

B mitt_comp_filter.m

```
function [vinkel] = mitt_comp_filter(acc_v, gyro,T,k,ki)
%% mitt filter 2.ordens komplimenter filter

int_reg(1) = 0; % integralet som bygger seg opp i regulatoren
int_y(1) = 0;   % integral av vinkelhastigheten

for i=1:size(gyro)-1,
   avvik = acc_v(i) - int_y(i);
   g = avvik * k;
   int_reg(i+1) = g*ki + int_reg(i);
   int_y(i+1) = (g + int_reg(i+1) + gyro(i)) * T + int_y(i);
end

vinkel = int_y;
```

C euler.m

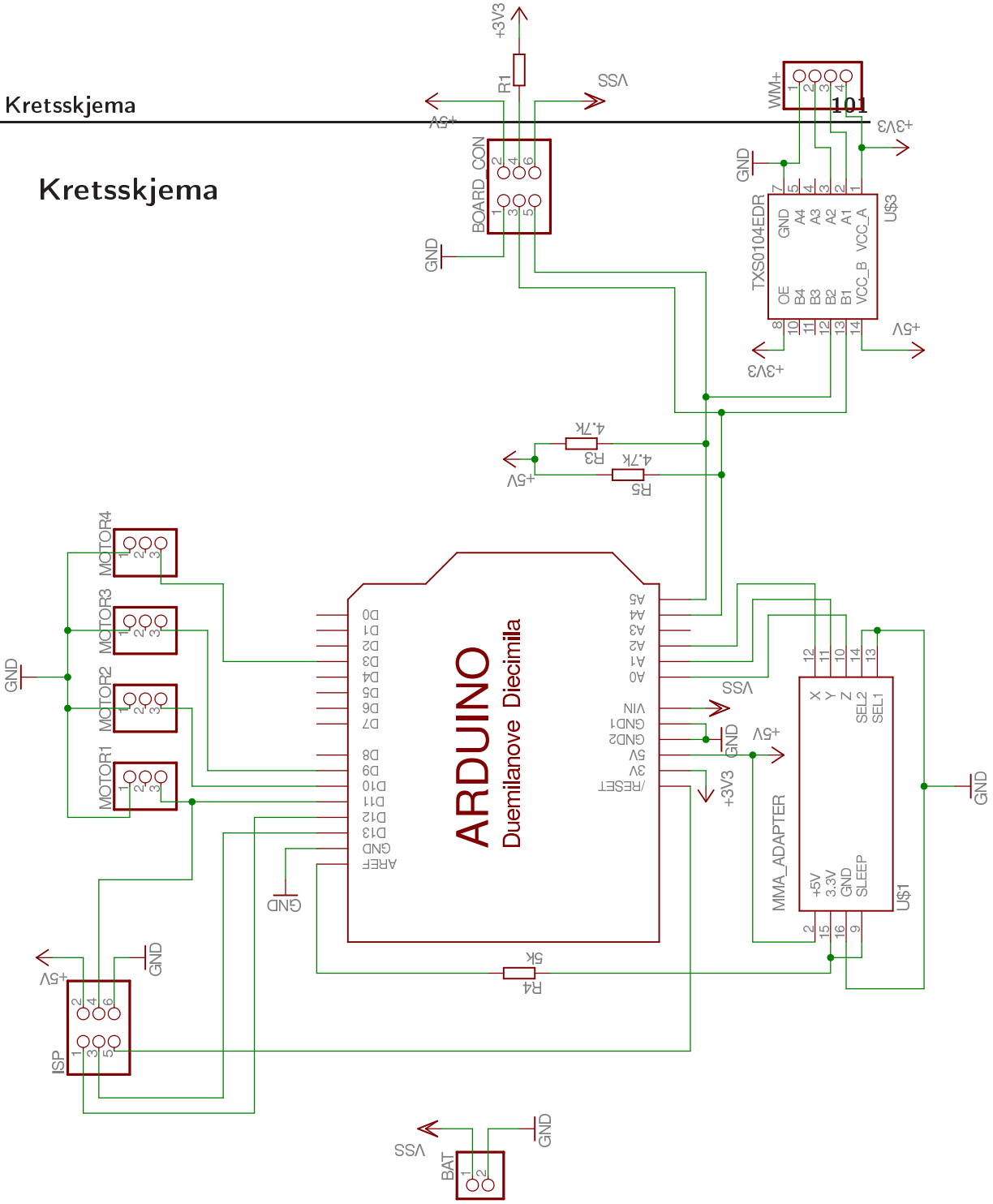
```
function [pitch,roll,yaw] = euler(DCM)
  %% Hent ut vinkler

  % pitch kan hentes ut direkte fra matrisa
    pitch = -asin_n(DCM(3,1));

  % Henter ut roll
    roll = asin_n(DCM(3,2));

    % Det finnes ingen direkte metode for å hente ut yaw. Det kan gjøres ved hjelp
    % av atan2, men dette gir dårlige resultater.
    % Denne metoden er den som har gitt best resultat til nå
    % **** Prøv å fiks yaw ***************
    % Normaliser vektorane
    vekt = DCM(:,1);
    vekt2 = [0 -1 0]'; % Ønsket retning

    magn = sqrt(vekt(1)*vekt(1)+vekt(2)*vekt(2)+vekt(3)*vekt(3));
    magn2 = sqrt(vekt2(1)*vekt2(1)+vekt2(2)*vekt2(2)+vekt2(3)*vekt2(3));
    vekt = vekt/magn;
    vekt2 = vekt2/magn2;
    yaw = acos(dot(vekt,vekt2))-deg2rad(90);

    %til grader
    pitch = rad2deg(pitch);
    roll = rad2deg(roll);
    yaw = rad2deg(yaw);
```

D DCM.m

```
function [pitch,roll,yaw,ut] = DCM(roll,yaw, pitch,acc_x,acc_y,acc_z,T, filter) % filter

g = mean(acc_z(1:20)); % g til acc
Omega_I=[0 0 0]';
Omega_P=[0 0 0]';

 %% Init data
DCM = [1 0 0
       0 1 0
       0 0 1];
% Gjør om til radianer
 roll = roll*(pi * 2) / 360;
 pitch = pitch*(pi * 2) / 360;
 yaw = yaw*(pi * 2) / 360;

 %% selve filteret
 for i=1:size(roll),
  Gryo_vektor = [roll(i)
                 pitch(i)
                 yaw(i)]; % Lag gyro vektor

if (filter == 1)
    omega = Gryo_vektor-Omega_I;
    Omega_Vektor =omega-Omega_P;
else
    Omega_Vektor = Gryo_vektor;
end;

  Omega_Vektor = Omega_Vektor*T;
  Omega = [1 -Omega_Vektor(3) Omega_Vektor(2)
           Omega_Vektor(3) 1 -Omega_Vektor(1)
           -Omega_Vektor(2) Omega_Vektor(1) 1];

  DCM = DCM*Omega; % Regn ut neste tidssteg

    % Normaliser
  DCM = normal(DCM);

  [pitch(i),roll(i),yaw(i)] = euler(DCM);

 % finn korreksjons greine
Accel_Vektor = [acc_x(i)
                acc_y(i)
```

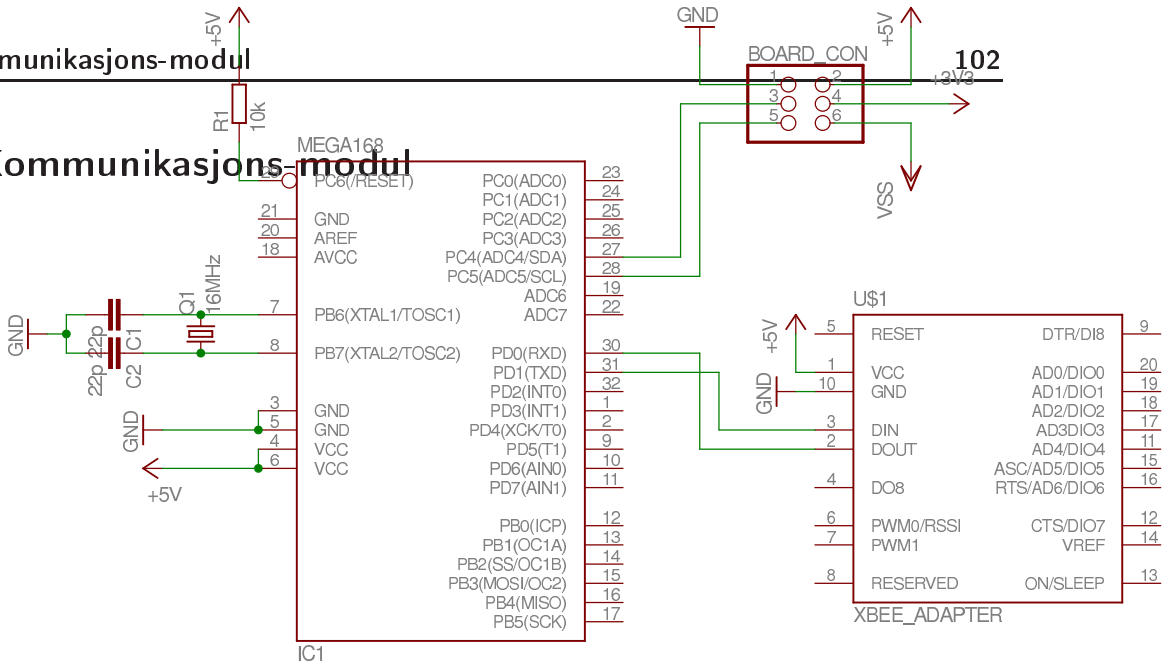
```
                acc_z(i)];

acc_error = cross(DCM(3,:)',Accel_Vektor);

Kp_ROLLPITCH = 0.014;
Ki_ROLLPITCH = 0.000011;

Omega_P = acc_error * Kp_ROLLPITCH;
temp = acc_error * Ki_ROLLPITCH;
Omega_I = Omega_I + temp;

%ut = 0;
ut(i,:) = [DCM(1,:) DCM(2,:) DCM(3,:)];
 end
```

E Kretsskjema



Figur E.1: Kretsskjema laget i eagle

F Kommunikasjons-modul



Figur F.1: Kretsskjema laget i eagle