NTNU
Det skapende universitet

Modellering og regulering av sirkulært opphengt invertert multipendel

Øyvind Bjørnson-Langen

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

Norges teknisk-naturvitenskapelige universitet
Institutt for teknisk kybernetikk

Oppgavetekst

Institutt for teknisk kybernetikk ønsker å benytte en invertert multipendel som "imponator". Arbeidet ble startet av Amund Skavhaug i 1989, og det er siden gjort arbeid både på instrumenteringen og matematikken til systemet. Oppgavens hensikt er å ferdigstille totalsystemet slik at flere pendler kan svinges opp.

Oppgaven består av:

Sette seg inn i tidligere arbeid samt nødvendig teori.

Utføre nødvendige forbedringer på datainnsamling og finne parametre til pådragsenhet.

Videreutvikle reguleringsstrategi og implementere programvare for realisering av totalsystem.

Vurdere godhet av løsning.

Oppgaven gitt: 09. juli 2007
Hovedveileder: Amund Skavhaug, ITK

Forord

Denne rapporten tar for seg arbeidet utført med diplomoppgaven på institutt for teknisk kybernetikk. Diplomoppgaven går ut på å knyte sammen arbeider gjort i tidligere diplomoppgaver og prosjektoppgaver som har omhandlet modellering, regulering og instrumentering av et trippelinvertert pendelsystem. Målet har vært å få til oppsving og balansering av både en og to inverterte pendler. Når det gjelder denne rapporten er det forutsatt at leser har kunnskaper innenfor kybernetikk fra før. Forklaringer er korte der det lar seg gjøre og ofte med henvisninger til andre kilder om man ønsker mer informasjon. Det er absolutt å anbefale å lese gjennom [14] for innføring i hvordan datainnsamlingen er bygd opp og [13] om man ønsker mer informasjon om hvordan modeller og regulatorer er utledet.

Under følger en del tanker rundt fordeler og utfordringer i forbindelse med arbeidet som er utført. Jeg håper at noe av informasjonen her kan hjelpe andre studenter i fremtiden med å gjøre riktige valg. Mitt arbeid med pendelen startet som prosjektoppgave høsten 2006. Grunnet studieopphold i utlandet var det nødvendig å bruke våren 2007 for å ta ekstra fag før arbeidet med pendelen ble fortsatt høsten 2007. Det å ha hatt et halvt år på å tenke og reflektere samt ta relevante fag har vist seg fordelaktig i det videre arbeidet med pendelen.

Men selv med et halvt år mellom prosjekt og diplom har det vært vanskelig med veivalg i starten. I Etterpåklokskapens lys har man ofte lurt på hva det var man egentlig tenkte da man først prøvde å løse et problem. Noen problemer kan man til en viss grad forutse og ved hjelp av erfaring løse enkelt, mens andre feil vil under hele prosessen fra identifisering til løsnig utarte seg som meget kryptiske. Disse siste feilen er ofte relaterte til hardware, og for å illustrere dette er det tatt med et eksempel i vedlegg som forklarer en uforståelig interrupt-feil på en mikrokontroller.

Arbeid med hardware innebærer at man må ha pågangsmot hele veien og ikke la seg stoppe av merkelige feil. Det kan være greit å jobbe med flere ting parallelt eller veksle mellom områder av systemet man jobber med. Dette for å kunne reflektere og få nødvendig distanse fra vanskelige problemer som kan oppstå underveis. Feil som oppdages må og takles på en strukturert måte. For å finne feil må man verifisere og kontrollere alle elementer som kan produsere feilen, altså dele opp systemet i mindre deler helt til man finner den delen hvor feilen ligger. For at dette skal være enkelt er det nødvendig at man under systemdesign fortløpende verifiserer arbeid for å minimerer risikoen for feil. Det verste som kan skje er om en observert feil er et resultat av en kombinasjon av feil i forskjellige deler av systemet. Dette er nesten umulig å unngå når et system blir stort og komplekst, om

man ikke sørger for modulbasert oppbygning. Arbeidet utført i diplomoppgaven er derfor forsøkt delt opp i moduler som hver for seg verifiseres og sjekkes for feil.
I utviklingen av hardware er dette meget viktig. Selv om det ikke er nevnt i rapporten så er det for eksempel under utvikling av hardwareestimering koblet opp testsystem med bruk av utviklingsbrettet stk600 og breadboard. På denne måten har det vært mulig å utvikle og teste kode samt analysere problemområdene før kretsdesign er gjort. Dette faktum viser og at det tar tid å utvikle hardware, og at man helst ikke skal stresse arbeidet. Om man stresser risikerer man siden å lage kretskort som enten må modifiseres eller er helt ubrukelige på grunn av uoppdagede feil.
I diplomoppgaven har det og vært nødvendig å redesigne og bygge nytt fysisk system. Dette er en prosess som kan ta tid, spesielt fordi verkstedet på instituttet har alt for mye å gjøre. For nye studenter som trenger å få produsert deler anbefaler jeg å lære seg bruk av et CAD-program. Da sikrer man at produksjon går fort og uten feil.
Denne oppgaven inneholder konfidensielt materiale som ikke må gjøres tilgjegelig for allmenheten før etter februar 2007. Grunnen til dette er bruk av mikrokontrolleren XMEGA før den er lansert på markedet. Ønskes det mer informasjon angående XMEGA kan man kontakte Kristian Saether[1] hos Atmel.
Det er selvfølgelig en del personer som fortjener en takk for hjelpen i forbindelse med min utførsel av diplomoppgave

- Amund Skavhaug for veiledning
- Atmel ved Kristian Sæther for at de turde å starte samarbeid med meg helt på slutten av diplomperioden
- Verkstedet ved Terje Haugen og Hans Jørgen Berntsen.
- Komponentverkstedet for at de er der og da spesielt John Olav Horrigmo samt Stefano Bertelli og Jan Leistad, samt Per-Inge Løvold
- Elkraft for utlån av transformatorer og hallsensorer.

[1] Kristian.Saether@atmel.com

Sammendrag

Denne rapporten tar for seg arbeidet med å gå fra simulering av inverterte pendelsystemer, til fungerende oppsving og balansering på et fysisk system. Oppgaven omhandler mye av problematikken som man støter på i den forbindelse, blant annet forbedring av mekanikk, forbedring av sensor og målesystem, implementering av estimatorer, parameterestimering, tilpasning av regulatorer, implementering av sikkerhetsmekanismer og generell tuning.
Ukjente parametre for motor og motorregulator er funnet ved parameterestimering, disse er så blitt kontrollert ved bruk av ulineært kalmanfilter på det fysiske systemet. Resultatene fra disse forsøk viser at filteret følger tilstandene bra selv når tilbakekoblingen fra det fysiske sytemet er lav.
Siden sensorene på systemet kun måler vinkler må vinkelhastighet estimeres. Dette er gjort blandt annet ved ulineært kalmanfilter i programmvare og ved å måle tiden mellom to vinkelposisjoner ved bruk av mikrokontroller. Denne siste løsningen er implementert på mikrokontrolleren XMEGA. Disse estimatene er følsomme for vibrasjoner i systemet og gir derfor store oscillasjoner i pådraget under balansering. Det ulineære kalmanfilteret er det som fungerer best, men man har alikevel problemer med noe avvik i vinkelhastighetsestimatene.
Det er blitt utført oppsving og balansering av enkelpendel, enkelpendel hver side og dobbelpendel på det fysiske systemet. Dette verifiserer langt på vei at arbeide som tidligere har vært utført i simulering fungerer i praksis. For å øke robustheten ved oppsving og balansering av multipendelsystem er det nødvendig med redesign av regulatorer for å minske nødvendig tilbakekobling fra vinkelhastighetene eller utvikle bedre hastighetsestimering.

Innhold

Figurer

Kapittel 1

Innledning

1.1 Bakgrunn

Et fungerende pendelsystem er ønsket brukt som imponator for å lokke nye studenter og være motivasjon for allerede eksisterende studenter samt kunne brukes som utstilling for å vise frem noe av arbeidet som utføres ved institutt for teknisk kybernetikk.
Her er en kort sammenfatning av tidligere arbeid på Institutt for teknisk kybernetikk som har omhandlet dette inverterte pendelsystemer. Ved instituttet ble arbeidet på systemet denne rapporten omhandler startet opp av Amund Skavhaug i 1989, og siden har det fra tid til annen vært gjort studentprosjekter og diplomoppgaver. Modelleringen og utvikling av regulatorer er i den senere tid blitt utført i diplomoppgaver av Ekrem Misimi, [6] og Vegard Åstebøl Larssen, [13]. Misimi har tatt for seg noe av modelleringen og gjort noe arbeid på utledning av regulatorer. Men det er først med arbeidet til Larsen at både modelleringen og regulatorutledningen er gjort stegvis og oversiktlig. Han har utført modelleringen av systemet med Lagrange dynamikk. Oppsvingsregulatorene tar i bruk lineariserende tilbakekobling hvor pådraget er gitt med hensyn på å pumpe energi inn i systemet. Mens all stabilisering av pendlene i vertikal posisjon er gjort ved linearisering om arbeidspunktet og bruk av LQR algoritme for å regne ut optimalt pådrag. Dette arbeidet er så simulert og verifisert i `Matlab`, hvor det ble oppdaget singulariteter i modellen for trippelinvertert pendel.
Instrumenteringen av systemet er utført i Diplomoppgavene til Ekrem Misimi[6] og Hans Jørgen Svendsen[10], samt prosjektoppgaven til Øyvind Bjørnson-Langen[14]. Det virker som Misimi har kommet med en løsning samt kretsskjema for hvordan kunne løse instrumenteringen. Svendsen har implementert en løsning basert på enkodere for måling av vinkler. Dataene er så blitt behandlet av mikrokontrollere som sender disse videre via diodeoverføring og overføring ved RS232. En mikrodatamaskin som kjørte `VxWorks` var benyttet for å ta imot dataene.
Det har vært vanskelig å finne dokumentasjon på om noen av disse arbeidene har resultert i oppsving og stabilisering av det fysiske pendelsystemet. I [6] er det lagt ved plot som viser at enkelpendel har vært svingt opp og balansert uten noe mer dokumentasjon. Ingen av arbeidene før [14] har gjort noe med kommunikasjon mot regulatorkortet til `Baldor`,

men Svendsen har vist at datainnsamlingen fungerte. I arbeidet til Bjørnson er instrumenteringen gjort samt koblingen mot `Simulink` og `RTW` og det er vist ved forsøk at oppsving og balansering kan styres fra en styringsdatamaskin.
Arbeidet i denne diplomen omhandler å rette opp feilene og problemene som er kommentert i [14] og [13], og få til oppsving og balansering av så mange pendler som mulig.

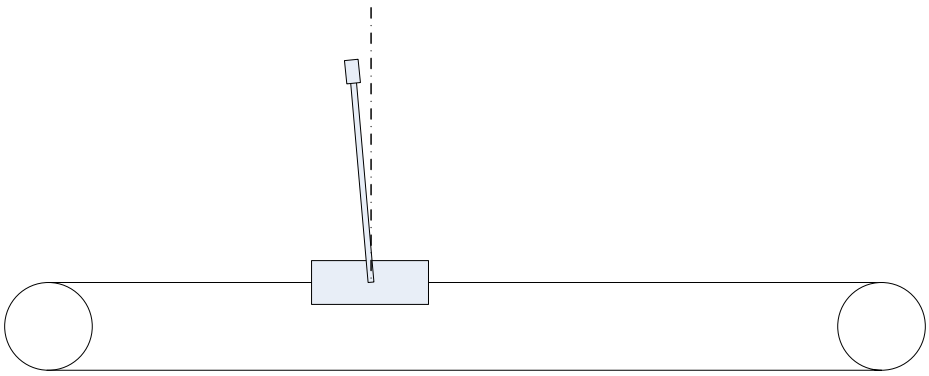
1.2 Rapportdisposisjon

- Kapittel 2 - Her er det en kort innføring i inverterte pendelsytemer generelt.
- Kapittel 3 - Viser systemet ved oppstart og hva som fungerer og ikke fungerer.
- Kapittel 4 - Dette kapittelet tar for seg nødvendig teori for de resterende kapittlene i oppgaven.
- Kapittel 5 - Her blir den eksisterende simulatoren kommentert samt arbeidet som er gjort for å tilpasse koden for å kunne brukes i sammenheng med `RTW` blir gjennomgått.
- Kapittel 6 - Kapittelet tar for seg forandringer som har vært nødvendig å gjøre både i hardware og software for å få til vellykket parameterestimering.
- Kapittel 7 - Dette kapittelet går gjennom design av nye kretskort og design av nytt mekanisk system.
- Kapittel 8 - Kapittelet viser oppnådde resultater og det blir diskutert hva disse betyr.
- Kapittel 9 - Konklusjon
- Kapittel 10 - Videre arbeid
- Kapittel 11 - Referanser
- De resterende kapittlene er vedlegg

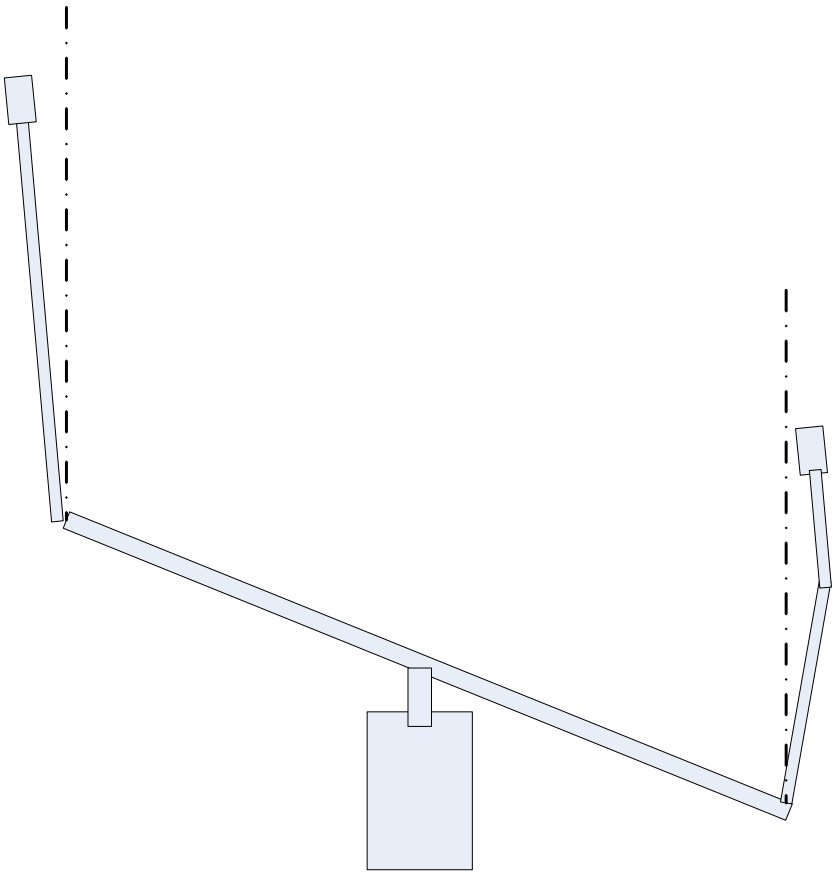
Kapittel 2

Inverterte pendelsystemer

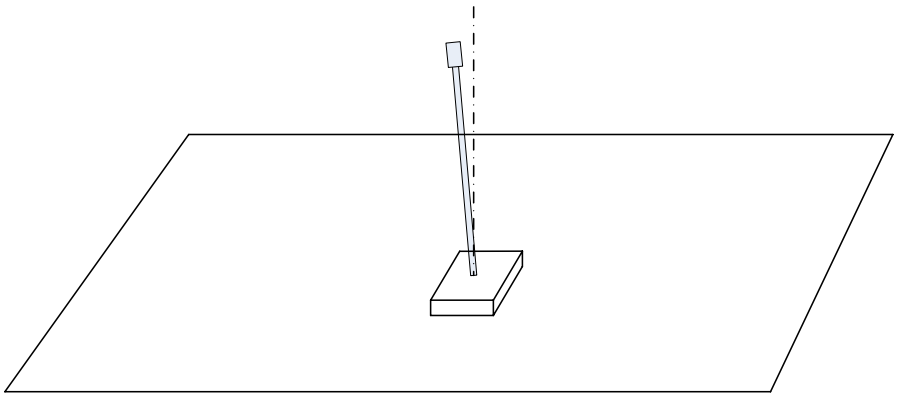
Det finnes flere hovedtyper av pendelsystemer, blant annet pendel montert på vogn som beveger seg i horisontalplanet, se figur 2.1, kule som ruller på en arm, furuta pendel, se figur 2.2 på neste side som er en sirkulært opphengt pendel og pendel montert på vogn som beveger seg på en flate, se figur 2.3 på neste side. Av disse systemene så er de to første forholdsvis enkle å modellere og regulere, mens de to siste øker betraktelig i kompleksitet. Pendelsystemet som er omhandlet i denne rapporten er av type furuta pendel med flere armer.



Figur 2.1: Plot hentet fra [14] viser pendel montert på vogn



Figur 2.2: Plot hentet fra [14] viser sirkulæropphengt pendel

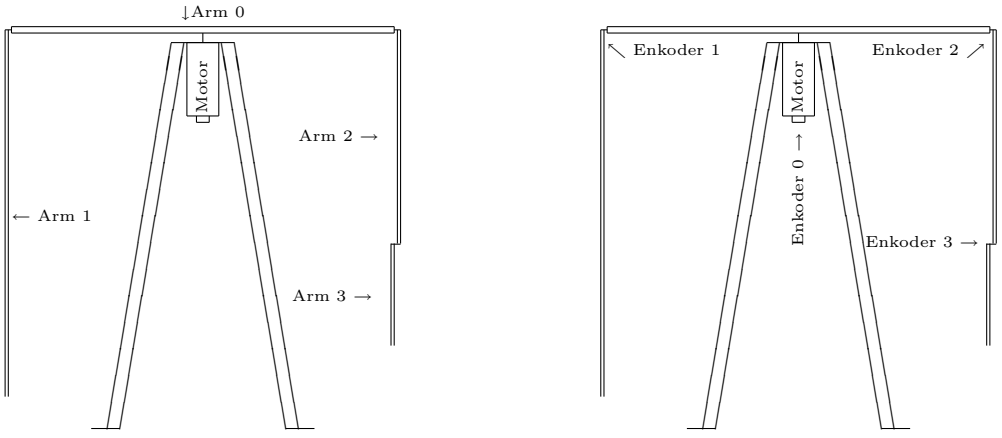


Figur 2.3: Plot hentet fra [14] viser todimensjonal pendel

Kapittel 3

Systemet ved oppstart

Systemet slik det var ved oppstart og slik det fortsatt er etter dette arbeidet, er satt sammen av tre hovedkomponenter. Den fysiske pendelen, regulatorkabinett og styringspc. Den fysiske pendelen består av bein, motor, armer samt elektronikk for datainnsamling. Regulatorkabinettet består av styringselektronikk for motoren samt et kort `node_reg` som fungerer som grensesnitt mellom styringspc og pendelsystem. Mens styringspc, heretter kalt `target_pc`, tar seg av koblingen mot systemet ved bruk av programvare laget i [14], samt programmer for regulering av systemet generert ved hjelp av `RTW`.
Det er nødvendig å definerer de forskjellige armene og sensorene på systemet og disse defineres på samme måte som i [14]. I figur 3.1(a) ser man den generelle definisjonen som brukes i denne rapporten, men desverre krasjer dette i forhold til hva som er gjort i [13]. Defineringen der er ikke konsistent mellom de forskjellige oppsett av pendler. For å lette arbeidet med implementasjon av estimatorer og regulatorer er defineringen fra [13] brukt på kode som er skrevet mot `Matlab`, noe som desverre kan skape forvirring.
Elektronikken som er brukt for datainnsamling er delt opp i tre kretskort. `node_reg` som



(a) Definering av armene

(b) Definering av sensorer

Figur 3.1: Definering av armer og enkodere. Figurene er hentet fra [14]

er nevnt over tar seg av siste grensesnitt mot `target_pc` samt måler vinkelen til `arm_0`. `node_0` sitter på `arm_0` og måler vinkelene til `arm_1` og `arm_2`, sender data til `node_reg` og tar imot data fra `node_2` som sitter på `arm_2`. Dette siste kortet måler vinkelen til `arm_3`. Styringselektronikken til motor er satt sammen av 1-fase transformator, likeretterkort og regulatorkort hvor de to siste er fra produsenten `Baldor`.
Kjøring av systemet etter prosjektoppgaven fra [14] innebar justering av fire potmetre på rotasjonsregulatoren, som er en integrert del av regulatorkortet, samt tuning av filtre og parametre i `Simulink`. Det at det i det hele tatt fungerte(rigtignok kun i fåtall av forsøkene) å svinge opp og stabilisere en pendel(mens de to andre også var montert og virket inn på systemet) kan nesten betegnes som flaks(og iherdig tuning).

3.1 Maskinvare

Kretskortene som ble produsert under prosjektperioden fungerer slik som ønsket. Det eneste problemet som er registrert med disse kortene er bruken av radiooverføring for sending av data fra `node_2` til `node_0`. Denne overføringen lider av støyproblemer, noe som fører til at det blir avvik over tid i vinkelverdien for `arm_3`.
Regulatorkortet fra `Baldor` som er benyttet til motorstyring har kun tilgjengelig turtallsstyring for bruk til regulering, og for å kunne bruke dette for momentstyring er det derfor nødvendig å vite nøyaktig verdien til parametrene som er brukt av denne regulatoren. Dette vanskeliggjøres av at flere av parametrene kan justeres med potmetre, og at regulator og motor i visse tilfeller er blitt observert å ha en ulineær oppførsel. Løsningen som har vært benyttet i prosjektperioden for å oppnå strømregulering har vært å filtrere strømmålingen som regulatorkortet gir ut før det benyttes som tilbakekobling. Differansen mellom ønsket pådrag og denne filtrerte strømmålingen blir så sendt gjennom en PID-regulator. Med denne løsningen har det da vært nødvendig å gjøre en avveiing mellom forsinkelse og støy, noe som ikke har fungert tilfredsstillende.

3.2 Programvare

Datainnsamlingen på systemet fungerte bra ved oppstart. Dataene som sendes med RS-232 til `target_pc` blir gjort tilgjengelig via ett hierarki av drivere. Å benytte et hierarki av drivere har gjort det nødvendig med bufring av data på hvert nivå, noe som kan gi forsinkelse og bruker unødvendig datakraft. Det mangler og en watchdog for å overvåke at systemet oppfører seg innenfor ett forhåndsdefinert sett med grenser. Dette er en nødvendighet for å ungå skade på system og omgivelser.
Det ble bevist at oppsving og stabilisering av enkelpendel fungerte ved avsluttet prosjektperiode, men disse forsøkene led noe av manglende robusthet og til tider farlige pådrag. Regulatorene laget fra [13] fungerer bra i simulering, men har altså ikke blitt utprøvd og justert for bruk på det fysiske systemet. Siden disse regulatorene er skrevet for bruk i simulering vil de derfor måtte skrives om noe for å takle problemene som kan oppstå når

man skal regulere et fysisk system.

3.3 Den fysiske delen av systemet

Tilstanden til det fysiske systemet ved oppstart av diplomoppgaven var ikke tilfredsstillende. Det stammer tilbake fra oppstarten i 1989 da det ble laget. Det eneste som har vært oppdatert siden, er beina og all elektronikken som er blitt laget. Mekanisk lider systemet av slitasje og feilbehandling, som i kombinasjon med noen uheldige mekaniske løsninger har ført til slark i alle ledd samt tretthetsbrudd i området hvor `arm_0` er festet til motoraksling. Det har allikevel vist seg mulig å svinge opp og balansere enkelpendel på dette systemet, men disse problemene gjør det både ustabilt og også farlig om ingen tiltak blir gjort for å øke robustheten til mekanikken.

Kapittel 4

Teori

Dette kapitlet omhandler nødvendig teori for arbeid gjort i de etterfølgende kapitlene. Målet med de neste underkapitlene er å presentere metoder for tilstandsestimering, parameterestimering og estimering av vinkelhastighet ved bruk av mikrokontroller som heretter vil bli kalt hardwareestimering. Tilstandsestimeringen er nødvendig for å finne vinkelhastighetene til systemet. Vinkelhastigheter og vinkelmålinger brukes av regulatorene som er benyttet for oppsving og balansering. Flere av parametrene til systemet er vanskelige å finne og derfor er det nødvendig med parameterestimering slik at man kan finne nøyaktige verdier som så kan brukes i regulatorene, estimatorene og i modeller brukt under simulering av systemet. Hardwareestimering er en av metodene som benyttes for å finne vinkelhastigheter.

4.1 Modellbasert tilstandsestimator

Modellbasert tilstandsestimator er som navnet tilsier avhengig av en modell av systemet hvor man ønsker å estimere en eller flere av tilstandene. Grunnen til at man ønske å bruke en modell av systemet er at denne metoden gir bedre estimater enn bruk av for eksempel derivering eller filtrering. Hvis man setter opp systemet på tilstandsromformen

$$\dot{x} = Ax + Bu + Ev \tag{4.1}$$
$$y = Cx + Du + w \tag{4.2}$$

hvor v og w er henholdsvis prosesstøy og målestøy, vil en generell tilstandsestimator være gitt av

$$\dot{\hat{x}} = A\hat{x} + Bu + K(y - \hat{y}) \tag{4.3}$$
$$\hat{y} = C\hat{x} + Du \tag{4.4}$$

hvor u og y er pådrag og målinger fra det reelle systemet. Matrisen K styrer innsvingningen av estimatoren i forhold til målingene fra det reelle systemet. For lineære tilfeller er det enkelt å bevise stabilitet for estimatoren. Dette kan gjøres ved å sette opp error-systemet

$$\dot{x}_e = \dot{x} - \dot{\hat{x}} = Ax + Bu - A\hat{x} - Bu - K(y - \hat{y}) \tag{4.5}$$
$$= A(x - \hat{x}) - KC(x - \hat{x}) = (A - KC)x_e \tag{4.6}$$

hvor man så må bevise at alle polene til $(A-KC)$ er negative. Man har allikevel problemet med parameterusikkerheten når man skal bruke dette på et fysisk system, men dette kan motvirkes ved å gi polene høye negative verdier. Det er anbefalt i [1] at verdiene bør velges 4-10 ganger mer negative enn polene til systemet.
Siden pendelsystemet er ulineært må det enten benyttes en lineær tilstandsestimator på en linearisert modell av systemet som kun kan benyttes på ett begrenset område, eller man kan benytte en tilstandsestimator beregnet for ulineære systemer. Om man har systemet

$$\dot{x} = f(x,u) + Ev \tag{4.7}$$
$$y = h(x,u) + w \tag{4.8}$$

Hvor v og w er henholdsvis prosesstøy og målestøy, vil da en generell tilstandsestimator for systemet være

$$\dot{\hat{x}} = f(\hat{x},u) + K(y-\hat{y}) \tag{4.9}$$
$$\hat{y} = h(\hat{x},u) \tag{4.10}$$

Hvor K er forsterkningen som sørger for at feilen mellom tilstandsestimatoren og den reelle systemet går mot null. For lineære systemer kan dette vises ved å finne egenverdiene til error-systemet. For ulineære systemer derimot blir dette mye vanskeligere og veldig avhengig av hva slags system man gjør det for, se [5].
Generelt for tilstandsestimatorer kan forsterkningen, K, enten regnes ut på forhånd eller dynamisk under kjøring. Dette er hovedforskjellen mellom forskjellige typer modellbaserte tilstandsestimatorer, både for lineære og ulineære versjoner. For eksempel representerer det lineære kalmanfilteret begge typer. For tidsinvariante system bruker man gjerne en statisk forsterkning, mens for tidsvarierende systemer bruker man gjerne dynamisk forsterkning. For ulineære systemer derimot er det ikke lenger mulig å gjøre noe enkelt skille, men det finnes mange forskjellige klasser av tilstandsestimatorer som benytter seg av begge typer. Ulineær high-gain observer benytter ofte statisk forsterkning selv om det og finnes utvidede versjoner med dynamisk forsterkning. Det ulineære kalmanfilteret bruker dynamisk forsterkning.
Det er valgt å bruke ulineært kalmanfilter ved bruk av modellbasert estimering av tilstandene til pendelsystemet. Det har vært gjort forsøk på bruk av både ulineær high-gain observer og kombinasjoner av lineære kalmanfiltre brukt på lineariserte modeller, men ingen av disse har kommet i nærheten av stabiliteten til det ulineære kalmanfilteret. Informasjon om disse estimatorene finnes i [5] og [3] og vil ikke bli gjennomgått her.

4.1.1 Ulineært kalmanfilter

Ulineært kalmanfilter er brukt til to ting i denne oppgaven, tilstandsestimator og parameterestimering. I oppgaven er det kun brukt diskrete versjoner av kalmanfilteret selv om `Matlab` støtter implementasjon av kontinuerlige systemer i sine s-funksjoner. Om man hadde benyttet seg av kontinuerlig implementasjon ville det skjedd en diskretisering under kjøring av simuleringen eller sanntidsfilene som kan lages med `RTW`. Type diskretisering

velger man da ved å velge `solver` i `Simulink`. Selv om det er mulig å gjøre det på denne måten er det valgt å benytte en diskret implementasjon hvor systemet diskretiseres på enkleste og minst resurskrevende måte. Det er benyttet euler-diskretisering som er en første ordens approksimasjon. Ulempen med en første ordens approksimasjon er at det er nødvendig med små tidsskritt for å sørge for stabilitet, og til forskjell fra de innebygde diskretiseringsalgoritmene i `Matlab` tar den ikke høyde for jitter[1] i periodetiden under kjøring på `target_pc`. Under implementasjonen er denne antatt å være minimal.
Ligningene for kalmanfilteret er delt inn i to deler, filterligninger og prediksjonsligninger. Prediksjonsligningen estimerer ett tidskritt videre i tid ut fra forrige estimerte tilstand og pådrag, samt regner ut kovariansmatrisen eller riktigheten for denne prediksjonen. Filterligningen regner først ut filterforsterkningen, K, som er gitt ut fra riktigheten til prediksjonen, før det så gjøres en filtrering mellom prediksjonen og målte verdier. Dette blir så satt sammen til et tilstandsestimat som kalmanfilteret gir ut. Under følger ligningene for filteret. Her er det brukt en egen notasjon for de forskjellige variablene. Denne notasjonen følger ikke hva som er vanlig i lærebøkene, men bør være lett å sette seg inn i når man har kjennskap til ligningene for kontinuerlig tilstandsrommodeller.

$$\hat{x}_k = \bar{x}_k + K_k(y_k - g(\bar{x}_k)) \tag{4.11}$$
$$\hat{X}_k = [I - K_k C_k]\bar{X}_k \tag{4.12}$$

der

$$K_k = \bar{X}_k (C_k)^T [C_k \bar{X}_k (C_k)^T + W_k]^{-1} \qquad C_k = \frac{\partial g}{\partial (x_k)^T}(\bar{x}_k) \tag{4.13}$$

Prediksjonligningene er

$$\bar{x}_{k+1} = f(\hat{x}_k, u_k) \tag{4.14}$$
$$\bar{X}_{k+1} = A_k \hat{X}_k (A_k)^T + E_k Z_k E_k^T \tag{4.15}$$

Hvor A_k finnes ved å partiellderivere f med hensyn på x_k, og sette inn $\hat{x}_k$

$$A_k = \frac{\partial f}{\partial (x_k)^T}(\hat{x}_k, u_k) \qquad Z_k = E[v_k v_k^T] \tag{4.16}$$

Det fins flere ulemper med det ulineære kalmanfilteret. Som med alle tilstandsestimatorer som bruker systemmodeller, er det en absolutt nødvendighet å ha så gode parametre for modellen som mulig. Dette gjør seg spesielt gjeldende med ulineære systemer hvor det ofte er vanskelig å garantere for stabiliteten til estimatoren fordi man kan risikere at de estimerte tilstandene kommer utenfor det stabile området til systemmodellen. Problemet her øker ettersom skrittlengden for estimatoren blir større, eller dynamikken øker i systemet man ønsker å estimere tilstanden til. Et annet problem kan være om systemmodellen har singulariteter. Dette vil i så fall føre til totalkrasj for estimatoren.

[1] I dette tilfellet: forskjellige periodetider

4.2 Parameterestimering

Som med tilstandsestimatorer finnes det flere forskjellige metoder for parameterestimering. Måle vekter og lengder og ut fra dette beregne parametre er en meget sikker måte å få gode parametre på, og vil derfor benyttes for de fleste parametre. For de parametrene som man ikke på noen enkel måte kan måle med linjal eller vekt, må det benyttes andre mer matematiske metoder. Det finnes i prinsippet to forskjellige måter å gjøre parameterestimering på, ved bruk av kjent systemmodell og uten bruk av systemmodell. Den første kan brukes on-line[2] eller off-line[3], mens den siste gjerne brukes off-line. Det er blitt gjort undersøkelse både på ulineært kalmanfilter for parameterestimering og subspace identification. Subspace identification er en metode for off-line-estimering uten bruk av systemmodell, mens ulineært kalmanfilter for parameterestimering tar i bruk systemmodell og kan brukes både on-line og off-line. Teorien bak subspace identification vil ikke bli gjennomgått her, men det henvises til [4] for en innføring. Fordelen med denne metoden er at den gir ut alle systemmatrisene samt kovariansmatrisen for støyen som er observert. Ulempen derimot er at den ikke sier noe om at parametrene man ønsker å estimere er fra et ulineært system, annet enn at kovariansmatrisen vil få høye verdier. Med ulineært kalmanfilter for parameterestimering er det derimot mulig å undersøke parametrene og de estimerte tilstandene opp mot de reelle under kjøring, se kapittel 8.1.2 på side 62. Hvis modellen man har valgt ikke passer, vil det dette fort observeres ved parametre som beveger seg uten mål og mening og estimerte tilstander som ikke ligner på de reelle. Ulempen med ulineært kalmanfilter for parameterestimering er derimot at man må kjenne systemmodellen. Det er i denne oppgaven gjort mest arbeid med parameterestimering ved hjelp av ulineært kalmanfilter fordi det i starten var ulineariteter i dataene som ble brukt og dette skapte en del forvirring rundt subspace identification.

4.2.1 Ulineært kalmanfilter for parameterestimering

Ulineært kalmanfilter kan brukes for å estimere parametre både on-line og off-line. For å kunne benytte seg av ulineært kalmanfilter må man kjenne modellen til systemet man ønsker å estimere parametre for. Parametrene som ønskes estimert, implementeres som tilstander i kalmanfilteret og tilnærmes ved hjelp av random walk, $p_{k+1} = p_k + \xi_k$ hvor ξ_k er hvit støy. Det at parametrene forandrer seg over tid og multipliseres med systemets tilstander er bakgrunnen for at det er nødvendig med et ulineært kalmanfilter selv i tilfeller hvor systemmodellen er lineær. En fordel med at det må benyttes et ulineært kalmanfilter er at man også kan finne parametre til ulineære systemmodeller.
Det er meget viktig ved bruk av ulineært kalmanfilter at samplingsfrekvensen er i forhold til ulinearitetene. Grunnen til dette er at hvis systemets tilstander forandrer seg for mye, kan man risikere at estimatoren får stort avvik fordi den trekkes mot feil likevektspunkt til systemmodellen. Dette gjør det vanskelig å bruke ulineært kalmanfilter og parameterestimering on-line på utstyr med lav regnekraft og rask dynamikk. I tilfellet i denne rapporten hvor man ønsker å finne parametrene til motor, regulator og `arm_0` er det ønskelig å sample

[2] estimering av parametre i sanntid
[3] parametrene estimeres ut fra et sett pådrag og et tilhørende sett målinger

systemet med så liten periode som mulig. Dette har gjort det nødvendig å kjøre parameterestimeringen off-line for ikke å bruke datakraft til dette mens systemet kjører.
Ligningene utledes her i det diskret tilfellet siden det er denne versjonen som er implementert. Det kan nevnes at det ikke er noe problem å finne parametre til et kontinuerlig system selv om de blir funnet ved hjelp av en diskret implementasjon av filteret. Det er nettopp denne løsningen som er valgt under implementasjonen gjort i vedlegg G på side 105.
Det er flere betingelser for at det skal være mulig å estimere parametre i et system. Pådraget som benyttes må være persistent eksiterende, som er en betegnelse på hvor rikt signalet er. For en mer presis definering se s.179 i [3]. For å kunne identifisere flere parametre er det viktig at transferfunksjonen har en spesiell struktur. Hvis man har systemet

$$H(s) = K\frac{\beta_0 + \beta_1 s + \cdots + \beta_m s^m)}{\alpha_0 + \alpha_1 s + \cdots + \alpha_n s^n} \tag{4.17}$$

vil det være mulig å estimere $n + m + 1$ parametre, hvor det siste ett-tallet kommer på grunn av at transferfunksjonen inneholder den konstante forsterkningen K. Pådraget som man da setter på systemet bør inneholde frekvenser som er i nærheten av knekkfrekvensene som polene og nullpunktene representerer. Pådrag som inneholder flere frekvenser som hvit støy og firkantpulser er de beste signalene å bruke til parameterestimering.
I tilfellet senere i rapporten hvor man ønsker å finne parametre for motor, regulator og `arm_0`, er det nødvendig å påtrykke signaler som er raske nok til å identifisere strømsløyfen og signaler som er trege nok til å identifisere rotasjonssløyfen. Hvilke frekvenser som er nødvendige å bruke vil da avhenge av frekvensresponsen til de forskjellige sløyfene, noe som i praksis betyr at man må prøve og feile for å finne gode pådrag om man ikke allerede har en idé om hvor frekvensresponsen til systemet ligger.
Ligningene som følger for parameterestimering ved hjelp av ulineært kalmanfilter tar basis i et lineært system, fordi motor og regulator har lineær dynamikk. Det er ikke noe i veien for å skrive om disse ligningene for parameterestimering på ulineære systemer, selv om dette da gir større krav til samplingsfrekvenser på implementasjonen. Implementasjonene som derimot er laget, se vedlegg G på side 105, støtter parameterestimering både på lineære og ulineære systemer. Hvis man har det lineære systemet under, hvor A, B og C kan ha parametre som skal estimeres

$$x_{k+1} = A(p_k)x_k + B(p_k)u_k + Ev_k \tag{4.18}$$
$$y_k = C(p_k)x_k + w_k \tag{4.19}$$

og hvor

$$p_{k+1} = p_k + \xi_k \tag{4.20}$$

er parametrene man ønsker å estimere. Disse implementeres da som ekstra tilstander $x_k^a = [x_k\ p_k]^T$. Den nye differensialligningen for x_k^a som man så får kan deles opp i to uttrykk,

ett for systemet og ett for støyen.

$$x_{k+1}^a = f^a(x_k^a, u_k) + E_k^a \zeta_k \tag{4.21}$$

$$f^a(x_k^a, u_k) = \begin{bmatrix} A(p_k)x_k + B(p_k)u_k \\ p_k \end{bmatrix} \tag{4.22}$$

$$\zeta_k = \begin{bmatrix} v_k \\ \xi_k \end{bmatrix} \tag{4.23}$$

Med ligningen for y_k

$$y_k = g^a(x_k^a) + w_k \tag{4.24}$$

Filtreringsligningene for det ulineære kalmanfilteret blir da

$$\hat{x}_k^a = \bar{x}_k^a + K_k(y_k - g(\bar{x}_k^a)) \tag{4.25}$$

$$\hat{X}_k^a = [I - K_k C_k^a]\bar{X}_k^a \tag{4.26}$$

der

$$C_k^a = \frac{\partial g^a}{\partial (x_k^a)^T}(\bar{x}_k^a) = \left[C(p_k) \quad \frac{\partial}{\partial p_k^T}[C(p_k)x_k] \right]_{x_k=\bar{x}_k,\ p_k=\bar{p}_k} \tag{4.27}$$

$$K_k = \bar{X}_k^a (C_k^a)^T [C_k^a \bar{X}_k^a (C_k^a)^T + W_k]^{-1} \tag{4.28}$$

Prediksjonligningene blir så

$$\bar{x}_{k+1}^a = f^a(\hat{x}_k^a, u_k) \tag{4.29}$$

$$\bar{X}_{k+1}^a = A_k^a \hat{X}_k^a (A_k^a)^T + E_k^a Z_k (E_k^a)^T \tag{4.30}$$

Hvor A_k^a finnes ved å partiellderivere f med hensyn på x_k^a, og sette inn $\hat{x}_k^a$

$$A_k^a = \frac{\partial f^a}{\partial (x_k^a)^T}(\hat{x}_k^a, u_k) = \begin{bmatrix} A(p_k) & \frac{\partial}{\partial p_k^T}[A(p_k)x_k + B(p_k)u_k] \\ 0 & I_{n_p} \end{bmatrix}_{x_k=\hat{x}_k,\ p_k=\hat{p}_k} \tag{4.31}$$

$$Z_k = E[\zeta\zeta^T] = E\left[\begin{bmatrix} v_k \\ \xi_k \end{bmatrix} [v_k^T \ \xi_k^T] \right] = \begin{bmatrix} E[v_k v_k^T] & E[v_k \xi_k^T] \\ E[\xi_k v_k^T] & E[\xi_k \xi_k^T] \end{bmatrix} \tag{4.32}$$

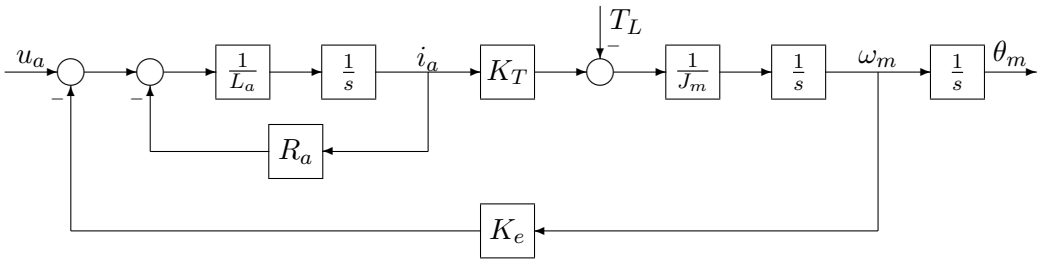
4.3 Motormodell

For å kunne benytte seg av parameterestimering ved hjelp av et ulineært kalmanfilter er det som nevnt tidligere nødvendig å utlede modeller for systemet man ønsker å finne parametrene til. Motoren som brukes for å styre pendelen er av type likestrømsmotor med permanentmagneter. Den fysiske utledningen ut fra strøm og krefter er gjort i [7], mens resultatene fra dette er gjengitt i ligningene under

$$\dot{\theta}_m = \omega_m \tag{4.33}$$

$$\dot{\omega}_m = \frac{K_T}{J_m} i_a - \frac{1}{J_m} T_L \qquad \text{Rotasjonssløyfe} \tag{4.34}$$

$$\dot{i}_a = -\frac{R_a}{L_a} i_a - \frac{K_E}{L_a} \omega_m + \frac{1}{L_a} u_a \qquad \text{Strømsløyfe} \tag{4.35}$$



Figur 4.1: Motormodellen

Denne modellen er gjengitt som blokkdiagram i figur 4.1. En generell karakteristikk for elektromotorer er at strømsløyfen er veldig rask i forhold til dynamikken til den mekaniske delen av motoren. Dette vil brukes i flere forenklinger som er gjort både på motormodellen og regulatorstrukturen. Det er nødvendig å skrive om strømsløyfen som vist under

$$
\begin{aligned}
i_a s &= -\frac{R_a}{L_a} i_a - \frac{K_E}{L_a}\omega_m + \frac{1}{L_a}u_a && (4.36)\\
\left(s + \frac{R_a}{L_a}\right) i_a &= -\frac{K_E}{L_a}\omega_m + \frac{1}{L_a}u_a && (4.37)\\
i_a &= -\frac{\frac{K_E}{L_a}}{\left(s + \frac{R_a}{L_a}\right)}\omega_m + \frac{\frac{1}{L_a}}{\left(s + \frac{R_a}{L_a}\right)}u_a && (4.38)
\end{aligned}
$$

før denne kan settes inn i en noe omskrevet rotasjonssløyfe

$$
\begin{aligned}
\omega_m s &= \frac{K_T}{J_m} i_a - \frac{1}{J_m} T_L\\
\omega_m s &= \frac{K_T}{J_m}\left(-\frac{\frac{K_E}{L_a}}{(s + \frac{R_a}{L_a})}\omega_m + \frac{\frac{1}{L_a}}{(s + \frac{R_a}{L_a})}u_a\right) - \frac{1}{J_m}T_L\\
\left(s^2 + \frac{R_a}{L_a}s\right)\omega_m &= -\frac{K_T K_E}{J_m L_a}\omega_m + \frac{K_T}{J_m L_a}u_a - \frac{s + \frac{R_a}{L_a}}{J_m}T_L\\
\left(s^2 + \frac{R_a}{L_a}s + \frac{K_T K_E}{J_m L_a}\right)\omega_m &= \frac{K_T}{J_m L_a}u_a - \frac{s + \frac{R_a}{L_a}}{J_m}T_L\\
\left(\frac{J_m L_a}{K_T K_E}s^2 + \frac{R_a J_m}{K_T K_E}s + 1\right)\omega_m &= \frac{1}{K_E}u_a - \frac{R_a}{K_T K_E}\left(\frac{L_a}{R_a}s + 1\right)T_L\\
\omega_m &= \frac{\frac{1}{K_E}u_a - \frac{R_a}{K_T K_E}\left(\frac{L_a}{R_a}s + 1\right)T_L}{\frac{J_m L_a}{K_T K_E}s^2 + \frac{R_a J_m}{K_T K_E}s + 1}\\
\omega_m &= \frac{\frac{1}{K_E}u_a - \frac{R_a}{K_T K_E}(1 + T_a s)T_L}{T_a T_m s^2 + T_m s + 1}\\
\omega_m &\approx \frac{\frac{1}{K_E}u_a - \frac{R_a}{K_T K_E}(1 + T_a s)T_L}{(T_a s + 1)(T_m s + 1)}
\end{aligned}
$$

Hvor $T_a = \frac{L_a}{R_a}$ er den elektriske tidskonstanten til motoren og $T_m = \frac{J_m R_a}{K_E K_T}$ er den mekaniske tidskonstanten til motoren. Siste omregning benytter at den elektriske tidskonstanten er mye mindre enn den mekaniske. Ut fra dette får man to transferfunksjoner som representerer motorens respons på strømpådrag og ytre pådrag.

$$\frac{\omega_m}{u_a} = \frac{\frac{1}{K_E}}{T_m s + 1} \tag{4.39}$$

$$\frac{\omega_m}{T_L} = -\frac{\frac{R_a}{K_T K_E}}{T_m s + 1} \tag{4.40}$$

Motormodellen viser at en likestrømsmotor er passiv sett at lasten også er passiv, se [7]. Det er bevist i [13] at alle pendelmodellene er passive, slik at totalsystemet med motor og pendel altså er passivt.

4.4 Regulatormodeller for motorstyring

Regulatorstrukturen for likestrømsmotorer er bygd opp i trinn. I første trinn sørger man for å styre strømmen til ønsket verdi. I andre trinn styrer man omdreiningsturtallet og i tredje trinn styrer man posisjonen. På `Baldor`-regulatoren blir det både i strømregulatoren og hastighetregulatoren benyttet PI-regulering noe som er det mest vanlige for servokontrollere.

4.4.1 Strømregulering

For å finne en strømregulator må man først finne transferfunksjonen fra i_a til u_a. fra (4.36) og (4.39) får man

$$\begin{aligned}
L_a i_a s &= -R_a i_a - \frac{K_E K_T}{J_m s} i_a - u_a && (4.41)\\
\left(L_a s^2 + R_a s + \frac{K_E K_T}{J_m}\right) i_a &= u_a s && (4.42)\\
\frac{i_a}{u_a} &= \frac{s}{L_a s^2 + R_a s + \frac{K_E K_T}{J_m}} && (4.43)\\
&= \frac{\frac{J_m}{K_E K_T} s}{\frac{L_a J_m}{K_E K_T} s^2 + \frac{R_a J_m}{K_E K_T} s + 1} && (4.44)\\
&= \frac{\frac{J_m}{K_E K_T} s}{T_a T_m s^2 + T_m s + 1} && (4.45)\\
&= H_a(s) && (4.46)
\end{aligned}$$

Proposjonalregulering

Her kan man så velge å bruke en proposjonalregulator $u_a = K_{ip}(i_d - i_a)$ som da gir

$$\begin{aligned} i_a &= H_a(s)u_a && (4.47)\\ &= H_a(s)K_{ip}i_d - H_a(s)K_{ip}i_a && (4.48)\\ (1 + H_aK_{ip})i_a &= H_a(s)K_{ip}i_d && (4.49)\\ \frac{i_a}{i_d} &= \frac{H_a(s)K_{ip}}{1 + H_a(s)K_{ip}} && (4.50) \end{aligned}$$

og her ser vi at når $K_{ip} \to \text{inf}$ vil $i_a \approx i_d$.

PI-regulering

Hvis man velger en PI-regulator, som er brukt på `Baldor`-regulatoren, setter man denne på formen under

$$u_a = \left(K_{ip} + \frac{1}{s}K_{ii}\right)(i_d - i_a) = \frac{K_{ii}}{s}\left(1 + \frac{K_{ip}}{K_{ii}}s\right)(i_d - i_a) \quad (4.51)$$

Og sørger for å velge $\frac{K_{ip}}{K_{ii}} = T_m$, samtidig som man forutsetter at $T_a << T_m$ som var antatt under utledningen av motormodellen. Så får man ut fra likning (4.45).

$$\begin{aligned} i_a &= \frac{\frac{J_m}{K_EK_T}s}{(1 + T_as)(1 + T_ms)}u_a && (4.52)\\ &= \frac{\frac{J_m}{K_EK_T}s}{(1 + T_as)(1 + T_ms)}\frac{K_{ii}}{s}(1 + T_ms)(i_d - i_a) && (4.53)\\ &= \frac{\frac{J_m}{K_EK_T}}{(1 + T_as)}K_{ii}(i_d - i_a) && (4.54) \end{aligned}$$

Her kan man velge K_{ii} stor slik at $i_a \approx i_d$.
Uansett om man velger P eller PI regulatorer ender man opp med modellen for rotasjonssløyfen under

$$\omega_m = \frac{1}{J_ms}(K_Ti_d - T_L) \quad (4.55)$$

Dette betyr at om man har implementert en strømregulator ut fra kravene over, kan man anse den som så rask at man kan se bort fra dynamikken i strømsløyfen under regulering av rotasjonssløyfen. Resultatet er dermed at strømpådraget man gir kan brukes direkte som pådrag til rotasjonssløyfen.

4.4.2 Hastighetsregulering

For utledningen av hastighetsregulator bruke man resultatet over, altså (4.55) som gir transferfunksjonen

$$\frac{\omega_m}{i_d} = \frac{K_T}{J_ms} \quad (4.56)$$

Proposjonal

Ved proposjonalregulering har man $i_d = K_{\omega p}(\omega_d - \omega_m)$

$$\omega_m = \frac{K_T}{J_m s} K_{\omega p}(\omega_d - \omega_m) \tag{4.57}$$

$$\omega_m = \frac{\frac{K_T K_{\omega p}}{J_m s}}{s + \frac{K_T K_{\omega p}}{J_m s}} \omega_d \tag{4.58}$$

$$\omega_m = \frac{1}{1 + \frac{J_m}{K_T K_{\omega p}} s} \omega_d \tag{4.59}$$

PI-regulert

Ved PI-regulering har man $i_d = (K_{\omega p} + \frac{1}{s} K_{\omega i})(\omega_d - \omega_m)$ som gir

$$\omega_m = \frac{K_T}{J_m s} \left(K_{\omega p} + \frac{1}{s} K_{\omega i} \right) (\omega_d - \omega_m) \tag{4.60}$$

$$\omega_m = \frac{K_T}{J_m s} \frac{K_{\omega p} s + K_{\omega i}}{s} (\omega_d - \omega_m) \tag{4.61}$$

$$J_m s^2 \omega_m + K_T K_{\omega p} s \omega_m + K_T K_{\omega i} \omega_m = K_T (K_{\omega p} s + K_{\omega i}) \omega_d \tag{4.62}$$

$$\omega_m = \frac{K_T (K_{\omega p} s + K_{\omega i})}{J_m s^2 + K_T K_{\omega p} s + K_T K_{\omega i}} \omega_d \tag{4.63}$$

$$\omega_m = \frac{1 + \frac{K_{\omega p}}{K_{\omega i}} s}{1 + \frac{K_{\omega p}}{K_{\omega i}} s + \frac{J_m}{K_T K_{\omega i}} s^2} \omega_d \tag{4.64}$$

velger man $K_{\omega i}$ stor får man

$$\omega_m = \frac{1 + \frac{K_{\omega p}}{K_{\omega i}} s}{\left(1 + \frac{K_{\omega p}}{K_{\omega i}} s\right) \left(1 + \frac{J_m}{K_T K_{\omega p}} s\right)} \omega_d \tag{4.65}$$

$$\omega_m = \frac{1}{1 + \frac{J_m}{K_T K_{\omega p}} s} \omega_d \tag{4.66}$$

Som gir tilnærmet samme transferfunksjon med P-regulator som ved PI-regulator.

4.5 Furuta pendel

Dette kapittelet vil kort gjennomgå hvordan modellen for Furuta-pendelen er funnet. Det henvises til [13] og [7] for en grundigere gjennomgang av både teori og utregninger. Den matematiske bevegelsesmodellen som er blitt laget for pendelsystemet bygger på Lagrange ligningen. For å kunne benytte seg av denne må man finne den potensielle og kinetiske energien til systemet. Disse energiene må så relateres til et inertielt koordinatsystem. Den

kinetiske energien til enkelpendelsystemet er sammensatt av den kinetiske energien til `arm_0` samt den kinetiske energien til `arm_1`.

$$K_{\text{total}} = K_{arm_0} + K_{arm_1} = \frac{1}{2}J_0\dot{\theta}_0^2 + \frac{1}{2}m_1v^2 + \frac{1}{2}J_1\dot{\theta}_1^2 \tag{4.67}$$

disse energiene må så representeres fra det inertiale koordinatsystem. Siden `arm_0` allerede er representert i det inertiale koordinatsystemet er det kun ligningen for `arm_1` som må forandres på.

$$K_{\text{total}} = K_{arm_0} + K_{arm_1} = \frac{1}{2}J_0\dot{\theta}_0^2 + \frac{1}{2}m_1(v_{01}^0)^2 + \frac{1}{2}J_1\dot{\theta}_1^2 \tag{4.68}$$

Man gjør det på samme måte for den potensielle energien. Her er det kun `arm_1` som har noe bidrag

$$P_{\text{total}} = P_{arm_1} = m_1gL_1cos(\theta_1) \tag{4.69}$$

For å finne bevegelsesligningene setter man så opp Lagrange-ligningen

$$L = K_{\text{total}} + P_{\text{total}} \tag{4.70}$$

løser man ut får man disse to ligningene

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{dt}}\left(\frac{\partial L}{\partial \dot{\theta}_0}\right) - \frac{\partial L}{\partial \theta_0} =& m_1(2\dot{\theta}_0\cos(\theta_1)\dot{\theta}_1 l_1^2\sin(\theta_1) - \sin(\theta_1)\dot{\theta}_1^2 l_1 L_0 \\
&+ \cos(\theta_1)\ddot{\theta}_1 l_1 L_0 + \ddot{\theta}_0 l_1^2 - \ddot{\theta}_0 l_1^2\sin^2(\theta_1) + \ddot{\theta}_0 L_1^2) + I_0\ddot{\theta}_0 \\
\frac{\mathrm{d}}{\mathrm{dt}}\left(\frac{\partial L}{\partial \dot{\theta}_1}\right) - \frac{\partial L}{\partial \theta_1} =& - m_1 l_1(-\ddot{\theta}_1 l_1 - \cos(\theta_1)\ddot{\theta}_0 L_0 \\
&+ \sin(\theta_1)\dot{\theta}_0^2 l_1\cos(\theta_1) + \sin(\theta_1)g)
\end{aligned} \tag{4.71}$$

Samme fremgangsmåte kan brukes for alle pendelkombinasjoner.

4.6 Friksjon

I [13] var det brukt en friksjonsmodell i simulatoren som kun tar for seg den viskøse friksjonen, altså

$$\tau_{\text{friksjon}} = \alpha\dot{\theta} \qquad \alpha = \text{konstant} \tag{4.72}$$

uten at det er nærmere presisert hvor parameteren α er funnet fra. Undersøkelser av systemet viser at en mer presis friksjonsmodell vil være en som tar med stiksjon, coloumb-friksjon, viskøs-friksjon og luftmotstand.

$$\tau_{\text{friksjon}} = \text{stiksjon} + \text{sign}(\dot{\theta})\beta_1 + \beta_2\dot{\theta} + \beta_3\text{abs}(\dot{\theta})\dot{\theta} \tag{4.73}$$

Siden systemet kjører med "lave" rotasjonshastigheter vil ikke luftmotstanden utgjøre en stor faktor for den horisontale armen. For pendlene som roterer kan luftmotstanden spille inn, spesielt siden armene som benyttes er laget av lette materialer. Stiksjonen gjør

seg spesielt gjeldende på ledd som benytter sleperinger, dette gjelder da `arm_0` og `arm_2`. For `arm_1` og `arm_3` er ikke stiksjonen målbar. Den viskøse friksjonen gir lite virkning på systemet. Den er meget liten og vanskelig å detektere ved de rotasjonshastighetene som benyttes. Hvis den skulle hatt innvirkning måtte systemet operert på et mye større hastighetsspekter. Ved høye rotasjonshastigheter ville antageligvis luftmotstanden være enerådende. Friksjonsmodellen som er benyttet videre for systemet er kun coloumb-friksjon, dette fordi stiksjon og viskøs-friksjon er liten i forhold til coloumb-friksjon mens det er antatt at luftmotstanden ikke spiller inn i nevneverdig grad. Videre i oppgaven brukes derfor modellen

$$\tau_{\text{friksjon}} = \text{sign}(\dot{\theta})\beta_1 \tag{4.74}$$

hvor β_1 kan måles med kraftmåler på systemet. Siden bruk av signumfunksjonen kan skape oscillasjoner i systemet er det valgt å bruke tilnærmingen gitt i ligningen under

$$\tau_{\text{friksjon}} = \tanh(k\dot{\theta})\beta_1 \tag{4.75}$$

Hvor k-verdien avgjør hvor god tilnærmingen skal være. Høy k gir bedre tilnærming, mens lav k gir glattere overgang fra -1 til 1.

4.7 Modeller av motor, motorregulator og arm_0 for parameterestimering

For å kunne gjøre parameterestimering er det nødvendig å sette sammen modellene av motor, motorregulator og last, som i dette tilfellet er `arm_0`, samt sette det på en form som kan benyttes av s-funksjonen som er skrevet, se vedlegg G på side 105.

4.7.1 Enkel modell for rotasjonssløyfen

Gitt likningen (4.39) hvor T_m byttes ut med $T_{\text{tot}} = \frac{(J_m+J_{\text{arm\_0}})R_a}{K_E K_T}$ for å ta med inertia til `arm_0`. Dette gir likningene

$$\omega_m = \frac{\frac{1}{K_E}}{T_{\text{tot}}s+1}u_a \tag{4.76}$$

$$T_{\text{tot}}s\omega_m + \omega_m = \frac{1}{K_E}u_a \tag{4.77}$$

$$\omega_m s = -\frac{1}{T_{\text{tot}}}\omega_m + \frac{1}{T_{\text{tot}}K_E}u_a \tag{4.78}$$

Hvor $u_a = \omega_d$. Det kan nevnes at (4.76) i [2] kalles første ordens Nomoto-modell. Systemmodellen blir da

$$\dot{\theta}_m = \omega_m \tag{4.79}$$

$$\dot{\omega}_m = -\frac{1}{T_{\text{tot}}}\omega_m + \frac{1}{T_{\text{tot}}K_E}\omega_d \tag{4.80}$$

For å kunne bruke dette i det ulineære kalmanfilteret for parameterestimering må man sette det på formen

$$\dot{\theta}_m = \omega_m \tag{4.81}$$
$$\dot{\omega}_m = p_1\omega_m + p_2\omega_d \tag{4.82}$$

Med $x^a = [\theta_m\ \omega_m\ p_1\ p_2]$ hvor man får estimert $p_1 = -\frac{1}{T_{\text{tot}}}$ og $p_2 = \frac{1}{T_{\text{tot}}K_E}$.

4.7.2 Modeller for strømsløyfe og rotasjonssløyfe

For å kunne estimere parametre i strømsløyfa er det nødvendig å sette sammen motorens strømsløyfe med de forskjellige regulatorene. Først settes P-regulatoren $u_a = K_{ip}(i_d - i_a)$ inn i strømligningen (4.36)

$$si_a = -\frac{R_a}{L_a}i_a - \frac{K_E}{L_a}\omega_m + \frac{1}{L_a}K_{ip}(i_d - i_a) \tag{4.83}$$
$$= -\frac{1}{L_a}(R_a + K_{ip})i_a - \frac{K_E}{L_a}\omega_m + \frac{K_{ip}}{L_a}i_d \tag{4.84}$$

For PI-regulatoren $u_a = \left(K_{ip} + \frac{1}{s}K_{ii}\right)(i_d - i_a)$ innsatt i (4.36) får man

$$si_a = -\frac{R_a}{L_a}i_a - \frac{K_E}{L_a}\omega_m + \frac{1}{L_a}\left(K_{ip} + \frac{1}{s}K_{ii}\right)(i_d - i_a) \tag{4.85}$$
$$= -\frac{1}{L_a}(R_a + K_{ip})i_a - \frac{K_{ii}}{L_a}\frac{1}{s}i_a - \frac{K_E}{L_a}\omega_m + \frac{1}{L_a}\left(K_{ip} + \frac{1}{s}K_{ii}\right)i_d \tag{4.86}$$
$$= -\frac{1}{L_a}(R_a + K_{ip})i_a - \frac{1}{L_a}\left(K_E + \frac{K_{ii}J_m}{K_T}\right)\omega_m + \frac{1}{L_a}\left(K_{ip} + \frac{1}{s}K_{ii}\right)i_d \tag{4.87}$$

hvor det er brukt at

$$\frac{1}{s}i_a = \frac{J_m}{K_T}\omega_m \tag{4.88}$$

som man finner fra (4.39).

I disse to uttrykkene, (4.84) for P-regulert og (4.87) for PI-regulert kan man så sette inn for både P og PI-regulert rotasjonshastighet og da oppnå til sammen fire systemmodellikninger.
For P-regulert strøm og P-regulatoren $i_d = K_{\omega p}(\omega_d + \omega_m)$ for hastighet får man

$$si_a = -\frac{1}{L_a}(R_a + K_{ip})i_a - \frac{K_E}{L_a}\omega_m + \frac{K_{ip}}{L_a}K_{\omega p}(\omega_d + \omega_m) \tag{4.89}$$
$$= -\frac{1}{L_a}(R_a + K_{ip})i_a - \frac{1}{L_a}(K_E + K_{ip}K_{\omega p})\omega_m + \frac{K_{ip}K_{\omega p}}{L_a}\omega_d \tag{4.90}$$

For å benytte seg av parameterestimering med ulineært kalmanfilter er det nødvendig å sette systemet opp på formen

$$\dot{\theta}_m = \omega_m \tag{4.91}$$
$$\dot{\omega}_m = p_1 i_a \tag{4.92}$$
$$\dot{i}_a = p_2 i_a + p_3\omega_m + p_4 u \tag{4.93}$$

med $x^a = [\theta_m\ \omega_m\ i_a\ p_1\ p_2\ p_3\ p_4]$ hvor

$$p_1 = \frac{K_T}{J_m + J_{\text{arm\_0}}} \tag{4.94}$$

$$p_2 = -\frac{1}{L_a}(R_a + K_{ip}) \tag{4.95}$$

$$p_3 = -\frac{1}{L_a}(K_E + K_{ip}K_{\omega p}) \tag{4.96}$$

$$p_4 = \frac{K_{ip}K_{\omega p}}{L_a} \tag{4.97}$$

De resterende 3 modellene er satt i vedlegg F på side 103

4.8 Furuta pendel, motor, motorregulator og friksjonsmodell

Man kan tenke seg at det fysiske systemet er delt i to. Strømsløyfen, inkludert strømregulator og strøm gjennom motor, og fysiske system representert av pendel og mekaniske respons til motor. Det er gjort på denne måten fordi det ikke er ønskelig eller nødvendig å ta med strømsløyfen i tilstandsestimeringen under kjøring på pendelsystemet. Under simulering vil man derimot gjerne ha med strømsløyfen i modellen av systemet. Dette var enkleste måten å gjøre det på siden det var ønskelig å gjenbruke kode fra [13].
For å modifisere pendelmodellen er det da nødvendig å legge til friksjonsmodell og inertia for `arm_0`. Man setter da $\tau = K_T i_a - J_{\text{arm\_0}}\dot{\omega}_m - \tau_{\text{friksjon}}$.
Ligningen for strømsløyfen under er implementert som en transferfunksjon før modellen av pendel med i_d som inngang og i_a som utgang.

$$\dot{i}_a = p_2 i_a + p_3\omega_m + p_4\theta + p_5 u + p_6\eta_1 \tag{4.98}$$

4.9 Estimering av hastighet ved hjelp av hardware (hardwareestimering)

Hele dette kapittelet forutsetter bruk av optisk enkoder for måling av vinkel, og kvadraturteller for kommunikasjon mot enkoderen. En grunnleggende innføring i hvordan dette fungerer finner man for eksempel i [14] eller man kan lese databladene [11] og [12].
Estimering av hastighet kan gjøres med likningen

$$\omega = \frac{\Delta\theta}{\Delta t} \tag{4.99}$$

Det finnes to prinsipielle måter å benytte seg av denne ligningen. Enten så holder man Δt på en fast verdi og måler forskjellen i vinkel mellom to tidspunkt, heretter kalt frekvensestimering. Eller man setter $\Delta\theta$ til ett forhåndsbestemt intervall og måler tiden i mellom, heretter kalt periodeestimering. Begge disse metodene setter meget høye krav til reaksjonen til hardware. På pendelsystemet kan man med andre ord ikke tillate seg å gjøre

målingene på `target_pc` fordi man da allerede er blitt utsatt for forsinkelse i dataoverføring fra nodene. Denne type estimering må gjøres så nærme enkoderen som mulig, og gjerne i samarbeid med kvadraturtelleren. Verdien Δt er meget relevant i reguleringshensyn fordi den sier noe om hvor ofte man får inn en måling på hastigheten. Denne verdien vil heretter kalles samplingsintervall.

4.9.1 Frekvensestimering

Siden frekvensestimering går ut på at man måler en diskret forandring i vinkel mellom to tidskritt, er det viktig at enten tidsskrittene er store nok, eller rotasjonshastigheten høy nok til at det skjer nok forandring i løpet av et tidsintervall. Feilen i estimatet er invers proporsjonal med antall forandringer som skjer mellom to tidsskritt. Hvis man tenker seg at tiden for et estimat går ut rett før den 101. forandringen i vinkel. Mens ved neste estimat får man med seg denne 101. forandringen i vinkel vil dette tilsvare en forskjell på 1%. Dette kan brukes for på forhånd kunne definere den maksimale feilen man kan godta i forhold til andre valgbare parametre som oppløsning til enkoder og samplingsintervall. Dette eksempelet viser også at desto høyere rotasjonshastigheten er, desto sikrere blir estimatet. Ligningen under viser estimert hastighet fra frekvensestimering

$$\omega \approx \frac{\#\text{transisjoner}\frac{2\pi}{\text{oppløsning fra kvadraturteller}}}{\Delta t} \tag{4.100}$$

Hvor *#transisjoner* er antallet vinkelforandringer. Om man ønsker en forhåndsdefinert maksimal feil på f.eks 1%, kan man skrive om likningen over til

$$\omega \geq \frac{\frac{100}{\text{feilprosent}}\frac{2\pi}{\text{oppløsning fra kvadraturteller}}}{\Delta t} \tag{4.101}$$

Likningen over gir et uttrykk for minimumsrotasjonshastighet som man må holde seg over for ikke å risikere større feil en den ønskede feilprosent.

4.9.2 Periodeestimering

Rotasjonshastighet ved periodemåling går ut på å bruke en høyoppløselig teller til å måle tiden mellom byttet fra en vinkelposisjon til en annen. Med denne løsningen er det viktig at enten rotasjonshastigheten er lav nok eller telleren er rask nok, slik at telleren rekker å telle mange nok ganger mellom transisjonene til å kunne gi et godt hastighetsestimat. Feilen er da invers proporsjonal med tellerverdi mellom to transisjoner. Om man tenker seg at transisjonen skjer akkurat før telleren har telt til hundre på første estimat, mens telleren teller til 101 på neste, vil man da få en feil på 1%. Dette kan, som for frekvensestimering, brukes til å definere en forhåndsbestemt maksimal feil i forhold til telleroppløsning og oppløsning til enkoder. Eksempelet viser og at nøyaktigheten øker desto lengre telleren rekker å telle, som da skjer ved lave rotasjonshastigheter. Likningen under viser estimering av rotasjonshastighet ved bruk av periodetid

$$\omega \approx \frac{\frac{2\pi}{\text{oppløsning fra kvadraturteller}}}{\Delta t_{\text{målt}}} \tag{4.102}$$

Ved å skrive om ligningen kan man så gi inn ønsket feilprosent

$$\omega \leq \frac{\frac{2\pi}{\text{oppløsning fra kvadraturteller}}}{\frac{100}{\text{feilprosent}}\frac{1}{\text{tellerfrekvens}}} \tag{4.103}$$

Noe som setter en øvre grense for rotasjonshastigheten. Denne øvre grensen kan økes om man istedenfor å måle tiden det tar for en transisjon, måler tiden det tar for to eller flere transisjoner. Dette kan brukes om man ønsker å bruke periodeestimat ved høye hastigheter. En fordel til med å måle mellom flere transisjoner er om det er mye varians på enkoderen.

4.9.3 Valg av verdier

For å velge verdier for enkoderoppløsning, tellerfrekvens, samplingsintervall og ønsket feilprosent er det viktig å vite hva slags rotasjonshastigheter systemet skal operere på og om det er mye dynamikk i systemet hvor man ønsker å estimere. Er det mye dynamikk i systemet vil nødvendigvis samplingsfrekvensen måtte være høy. Ulempen da er at høy samplingsfrekvens kan gi problemer om man ønsker å detektere lave rotasjonshastigheter. Ved lave rotasjonshastigheter kan det gå lengre tid enn den ønskede samplingstiden før det skjer en tilstandstransisjon som kan brukes til å regne ut vinkelhastighet. Likningen under viser dette

$$\omega = \frac{\frac{2\pi}{\text{oppløsning fra kvadraturteller}}}{\Delta t} \tag{4.104}$$

Likningen gir minimums rotasjonshastighet som er observerbar innenfor ønsket samplingtid. Blir rotasjonshastigheten lavere enn dette vil man få forsinkelse i estimatet. Det finnes to muligheter for å løse dette problemet. Enten senke samplingsfrekvensen eller øke oppløsningen til enkoderen. I tillegg finnes det en programvareløsning som kan benyttes ved avtagende hastigheter. Hvis man undersøker med jevne mellomrom, gjerne tilsvarende samplingsintervallet, om tellerverdien er høyere enn forrige tellerverdi kan man begynne å estimere vinkelhastighet fortløpende med den løpende tellerverdien helt til ny vinkelposisjon mottas og tellerverdien blir låst. På denne måten får man et noe glattere hastighetsestimat ved avtagende hastighet.
Tabell 4.9.3 på neste side viser laveste rotasjonshastighet systemet kan ha for forskjellige enkoderoppløsninger om systemet skal overholde samplingsintervallet på 1ms. For pendelsystemet som har samplingsintervall på 1ms og oppløsning på 2000 ser man at minimums rotasjonshastighet blir $3.14 rad/s$. Dette er en forholdsvis høy verdi, og som man kan se senere skaper dette problemer. Nå kan man gå videre og finne nødvendig oppløsning til telleren og ved hvilke rotasjonshastighet man må bytte mellom frekvensestimat og periodeestimat for å overholde ønsket feilprosent. Nødvendig tellerfrekvens kan finnes ved å sette sammen likningene (4.101) og (4.103). Da får man uttrykket

$$\text{Frekvens}_{\text{teller}} \geq \frac{100^2}{\text{feilprosent}^2}\frac{1}{\Delta t} \tag{4.105}$$

Om tellerfrekvensen skulle bli for høy i forhold til hva som er mulig, er det nødvendig å gjøre modifisering av periodeestimeringen. Som nevnt over kan man da måle tiden mellom

Oppløsning fra kvadraturteller	Minimums fart ved 1ms
800	7.85
2000	3.14
4000	1.53
16000	0.31

Tabell 4.1: Tabellen viser minimums rotasjonshastighet som det skjer forandring i innenfor samplingsperiode på 1ms

Minimum tellerfrekvens	#transisjoner
10Mhz	1
5Mhz	2
3.33Mhz	3
2.5Mhz	4

Tabell 4.2: Tabellen viser minimums klokkefrekvens ved forskjellig antall transisjoner

to eller flere transisjoner og da sørge for å bytte mellom estimeringsligningene ut fra rotasjonshastighet. Tellerfrekvensen som trengs for estimering som måler tid mellom flere transisjoner vil da kunne regnes ut fra likningen

$$\text{Frekvens}_{\text{teller}} \geq \frac{100^2}{\text{feilprosent}^2} \frac{1}{\Delta t} \frac{1}{\#\text{transisjoner}} \tag{4.106}$$

For så å finne rotasjonshastigheten hvor det er nødvendig å veksle mellom metodene setter man inn verdier i (4.101), hvor man bytter ut $\leq$ med $=$.
Tabell 4.2 viser tellerfrekvens for forskjellige antall transisjoner regnet ut fra likning (4.106). Det er forutsatt at samplingsintervall er valgt til 1ms. Siden denne minimumsfrekvensen er nødvendig i rotasjonshastighetsområdet hvor det skal foregå bytte til frekvensestimering kan man slakke litt av på kravene om man vet at systemet ikke kommer opp i disse hastigheten. For pendelsystemet hvor man har enkoderoppløsning på 2000 er hastigheten da gitt til å være $314 rad/s$. Pendelsystemet kommer aldri opp i hastigheter over $50 rad/s$ så tellerfrekvensen kan være en sjettedel av hva som er oppgitt i tabellen. Dette faktum tilsier og at det ikke er nødvendig å implementere frekvensestimering av vinkelhastighet.

4.9.4 Implementasjonshensyn

Likningen gitt over forutsetter at det ikke er noen forsinkelse i implementasjonen, noe som ikke er sannsynlig. Derfor gjelder det å gjøre det som er mulig for å få hardware man bruker til å reagere så fort som mulig på hendelser i forbindelse med teller og enkoder. De fleste mikrokontrollere har interrupt på teller slik at frekvensestimeringen vil kunne implementeres uten problem. For å få til gode periodeestimat kan det derimot være vanskelig å bruke en ekstern kvadraturteller. Man ville da være nødt til å polle på denne så ofte som mulig for å registrere en transisjon og lese ut verdi av teller. En bedre løsning vil da være å

bruke enten en FPGA som gjort i [8], eller en mikrokontroller med innebygd kvadraturteller som gir ut interrupt ved transisjon. Det er først de siste årene at slike mikrokontrollere har kommet på markedet, først fra Microchip og nå også fra Atmel på deres nye serie XMEGA. I kombinasjon med å benytte interrupt er det ønskelig å velge så høye klokkefrekvenser som mulig for å minimere reaksjonstidene til mikrokontrolleren og tiden det tar for å utføre annen kode.
Det bør nevnes at periodeestimering er følsom for støy på A og B signalene som enkoder gir ut. Falske transisjoner på signallinjene vil altså trigge hastighetsestimat. Man kan da risikere at disse estimatene har ekstremt høye verdier noe som kan være katastrofalt i reguleringssammenheng. Det kan derfor være fordel å bruke skjermet kabel eller benytte seg av differensial signalering fra enkoder hvis dette er mulig.

4.9.5 Feilkilder enkoder

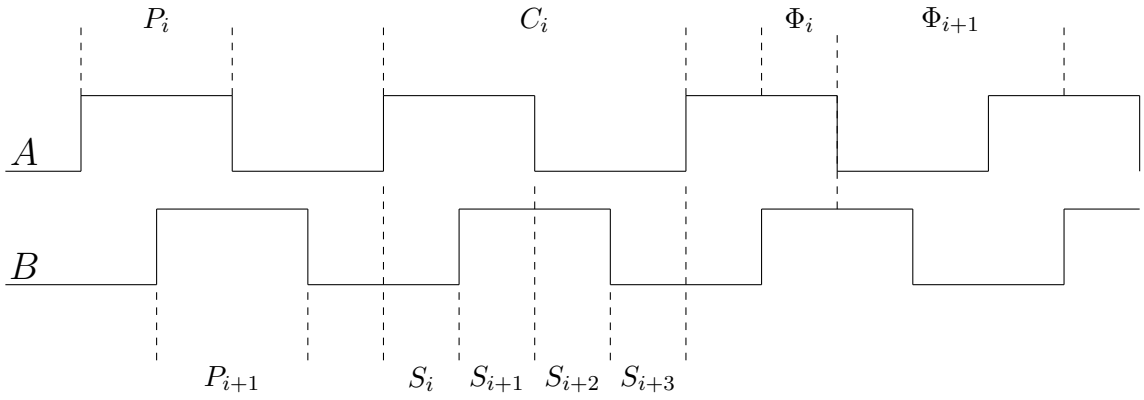
En stor feilkilde for hastighetsestimering ved periodemåling er den fysiske unøyaktigheten til enkoderen. For at hastighetsestimatet skal være godt er det meget viktig at avstanden til stolpe og vindus-parene er uniforme. A og B signalene fra enkoderen gis ut fra lys som stoppes av stolpene eller går igjennom vinduene på enkoderhjulet. Hvis en stolpe eller et vindu har feil form vil dette gi signaler som kommer for tidlig eller for sent. Det er mange mulige feilkilder for enkoderhjulet og definisjonen under er hentet fra databladet [12].

- **Antall(N)** - Antallet stolpe og vindus-par som er på enkoderhjulet
- **En sykel(C)** - 360 elektriske grader($e\,\text{deg}$) eller et stolpe og vindus-par. Hvis man tenker seg at A og B signalet er 90 grader faseforskjøvet, så vil da 360 elektriske grader tilsvare at A signal går høy så går B signal høy før A går lav etterfulgt av at B går lav.
- **Sykel-feil(ΔC)** - Er et uttrykk for hvor uniform syklene er.
- **En aksel omdreining** - tilsvarer 360 mekaniske grader
- **Posisjons Feil($\Delta\Theta$)** - Normalisert vinkeldifferanse mellom akslingens faktiske posisjon og posisjonen som blir oppgitt fra kvadraturteller. Feilkilden her ligger i monteringen. Hvis for eksempel hullet i enkoderskiven er litt for stort vil skiven bli montert med en liten radiell forskyvning. Dette vil da forplante seg som en vinkelfeil.
- **Pulsbredde(P)** - Antallet elektriske grader en puls er høy. Bør være 180 elektriske grader
- **Pulsbreddefeil(ΔP)** - Avviket på pulsbredden i elektriske grader
- **Tilstandsbredde(S)** - Antall elektriske grader mellom et tilstandbytte i den ene kanalen til et tilstansbytte i den andre kanalen. Er 4 slike for hver sykel som hver skal være på 90 elektriske grader.
- **Tilstandsbreddefeil(ΔS)** - Avviket i tilstandsbredde fra normalen på 90 elektriske grader

Parameter	Typisk	Maksimal	Enhet
ΔP	7	45	elektriske grader
ΔS	5	45	elektriske grader
$\Delta \Phi$	2	20	elektriske grader
$\Delta \Theta$	10	40	min. of arc
ΔC	3	5.5	elektriske grader

Tabell 4.3: Tabellen viser typisk og maksimal feil på enkoderparametre

- **Fase(Φ)** - Antall elektriske grader mellom midtpunktet på et høyt utgangssignal fra den ene kanalen til et høyt utgangssignal på den andre kanalen. Skal være 90 eller 270 elektriske grader
- **Fasefeil($\Delta\Phi$)** - Avvik i elektriske grader på fasen.



Figur 4.2: Figuren viser A og B signal fra enkoder og defineringen av parametre

Fysisk unøyaktighet er uten tvil det som utgjør den største feilkilden ved periodeestimering. Og til forskjell fra andre feilkilder er det lite man kan gjøre med dem uten å påvirke samplingsfrekvensen. ΔS er for `HEDS-5500` oppgitt til å være typisk 5 elektriske grader. Men i verste fall kan den være helt opp til 45 elektriske grader, se tabell 4.3. Tabellen viser og verdier for de andre feilen som kan være på enkoderen. Disse verdiene gjelder for enkoderhjul når de kommer direkte fra produksjon. Om man i ettertid har skadet enkoderhjulet kan det være mye større feil enn dette. Og skader kan lett skje ved montering og demontering om man er uforsiktig.

4.9.6 Andre feilkilder

For hastighetsestimering ved kombinering av frekvensestimering og periodeestimering har man tre problemområder. Ved veldig lave rotasjonshastigheter vil ikke samplingsfrekvensen bli overholdt, og om man har en teller med lav oppløsning kan man risikere å få overflow

og dermed en begrensning for hvor lav hastighet hardware klarer å gi ut. Ved området hvor det byttes fra periodeestimering til frekvensestimering vil det være et toppunkt på feilen. Til slutt ved meget høye rotasjonshastigheter blir det problem for hardware. Man kan risikere overflow av kvadraturteller, problemer med å registrere vinkelforandringer for fotodioder i enkoderen og problemer med å registrere vinkelforandring fordi kvadraturteller vil at A og B signal skal holdes ett minimum antall klokkesykler før det registrers som en transisjon. Alle disse problemene må man ta i betrakning under valg av oppløsning til enkodere, ønsket samplingsfrekvens, krystallfrekvens og godtagbare feilprosenter.

Kapittel 5

Simulatoren

Oppbygningen av simulatoren fra [13] er noe mangelfull fordi den ikke er gjort med hensyn på å brukes i `RTW`. Pendelmodellene er implementert som `level-2-c-file-s-function` noe som sikrer raske simuleringer. Regulatorer og observator som bytter mellom de forskjellige regulatorene er implementert som `embedded-m-function`, og i tillegg spredt utover i mange blokker. Siden flere av regulatorene og observatoren bruker de samme ligningene er dette en unødvendig resurskrevende implementasjon. Oppstartsparametre for både pendelmodell, regulator og observator initialiseres ved hjelp av en init-fil som må kjøres før simulering.

5.1 Init-fil

I init-fila fra [13] velger man ønskede parametre for pendelsystemet som masse og lengde til de forskjellige armene. Init-fila sørger så for å kjøre en ny fil `LinearizeAboutEquilibrium` som regner ut LQR-forsterkningene brukt i balanseringsregulatorene. Denne init-fila er blitt modifisert til å implementere resultater fra parameterestimering og nødvendige parametre for ulineært kalmanfilter samt nødvendige parametre for den nye regulatorstrukturen som er implementert med hensyn på å kunne kjøre på det fysiske systemet. En grundigere gjennomgang av init-fil samt kodeeksempel finnes i vedlegg K på side 115

5.2 Implementering av pendelmodeller

5.2.1 Kommentar til implementasjonen i [13]

I [13] er det ikke forklart hvordan modellene er implementert i simulatoren. Det er derfor nødvendig med en forklaring på hvordan dette er gjort fordi denne implementasjonen fungerer som grunnlag både for de modifiserte pendelmodellene som er laget og for implementasjonen av ulineært kalmanfilter. Man tar systemet fra likning (4.71) og setter det på formen

$$M(\Theta)\ddot{\Theta} + C(\Theta, \dot{\Theta})\dot{\Theta} + D(\Theta, \dot{\Theta}) + G(\Theta) = \tau \tag{5.1}$$

Hvor $\Theta = [\theta_0\ \theta_1]^T$. Ved å gange uttrykket over med $M(\Theta)^{-1}$ får man systemet på standard form for ulineære systemer som gir to ligninger, en for $\dot{\omega}_0 = \ddot{\theta}_0$ og en for $\dot{\omega}_1 = \ddot{\theta}_1$. Ved å augmentere systemet med tilstandene $\dot{\theta}_0 = \omega_0$ og $\dot{\theta}_1 = \omega_1$, får man en total modell for enkelpendelsystemet. Det er disse likningene som er implementert i en s-funksjon for å representere dynamikken til pendelsystemet.

5.2.2 Modifisert implementasjon

Pendelmodellen fra [13] var implementert som `level 2 c-file s-function`. Det er kun blitt lagt til en ekstra transferfunksjon til denne som etterligner strømsløyfen. For å få implementert parameterene for rotasjonssløyfen som er funnet fra parameterestimeringen, og legge til ny friksjonsmodell er systemet blitt manipulert på samme vis som gjort i 4.8 på side 22.
Enkelt forklart betyr dette at total inertia for motor og arm som er oppgitt i initfil er forandret fra

$$I_0 = \frac{m_0(2L_0)^2}{12} + I_m \tag{5.2}$$

Hvor I_m er inertia som er oppgitt på motor og brøken er en utregning av inertia til `arm_0` ut fra vekt, lengde og type rør. Utregningen av inertia til `arm_0` er en mangelfull metode fordi den forutsetter at vekten er jevnt fordelt på armen noe som ikke er tilfellet. Ved å forandre dette til

$$I_0 = \frac{K_T}{p_1} \tag{5.3}$$

Hvor parametrene er hentet fra parameterestimering vil man få verdier som representerer det fysiske systemet.
Forsterkningen som er på pådraget K_T og parametre for friksjonsmodellen blir og gitt via init-fil, men her har det vært nødvendig å modifisere i s-funksjonen slik at $\tau = K_T i_a - \tau_{\text{friksjon arm\_0}}$.

5.3 Implementering av regulator

5.3.1 Kommentar til Regulator og regulatorstruktur i [13]

Regulatorene som er brukt er hentet fra [13]. Reguleringen av systemet bygger på bytte mellom flere regulatorer ut fra tilstanden til systemet. Oppsvingsregulering er oppnådd ved lineariserende tilbakekobling hvor selve pådraget er konstruert for å pumpe energi inn i systemet. Balanseringsregulatorene er laget ved hjelp av linearisering om arbeidspunktene for så å finne optimalt pådrag ut fra `lqr`-funksjonen i `Matlab`. Observatoren som veksler mellom regulatorer ut fra tilstanden til systemet samt regulatorene er implementert som `embedded m-function`. Denne løsningen innebærer at samme parametre regnes ut flere ganger i hvert tidsskritt fordi flere av regulatorene samt observatoren bruker de samme ligningene. Denne løsningen må forbedres, spesielt med tanke på at regulatorene skal brukes på sanntidssystemet som skal styre det fysiske systemet. En bedre løsning er derfor å implementere alle regulatorene samt observatoren i en enkelt `level 2 c-file s-function`. Dette vil gi en bedre utnyttelse av datakraft. Men det er forbundet både fordeler

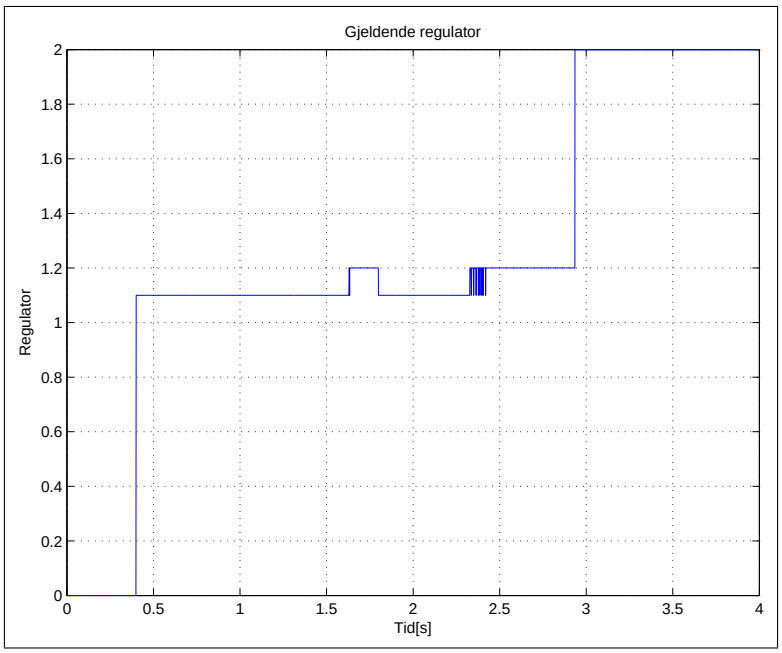
og ulemper ut fra om man vil simulere eller kjøre RTW-kode. Om man ønsker å forandre på koden er det nødvendig med en `mex`-rekompilering om man vil simulere systemet, mens med `embedded m-function` er det mulig å kjøre vanlig simulering uten noen form for kompilering. Om man bruker `embedded m-function` ved bruk av RTW og ønsker å gjøre forandringer i koden, må man gjøre en total rekompilering som beskrevet i vedlegg B.2 på side 95. Mens ved bruk av `level 2 c-file s-function` trenger man kun å gjøre en rekompilering på `target_pc`.

5.3.2 Modifisert implementasjon

Siden regulatoren som brukes i simulering også skal kunne benyttes på det fysiske systemet, er det viktig at implementasjonen gjøres ut fra fysiske hensyn. For det fysiske systemet er det viktig at det i byttet mellom regulatorer er noe overlapping. Det vil si at hvis det byttes fra en regulator til en annen ut fra at en tilstand har krysset en forhåndsbestemt grense, så vil den fortsette å benytte denne regulatoren selv om tilstanden går utenfor denne grensen igjen. Byttet kan først skje om tilstanden beveger seg videre og går forbi en annen forhåndsbestemt grense. Det er altså nødvendig med to sett av parametre for å bytte mellom to regulatorer. Bruk av en slik observator som bytter mellom regulatorer er nødvendig for å ungå oscillasjoner relatert til raske bytter frem og tilbake mellom regulatorer. Figur 5.1 på neste side viser bytte av regulator som skjer i oppsving og balansering av enkelpendel på fysisk system. Her er det ikke brukt overlapp på regulatorene 1.1 som sørger for å pumpe inn energi under oppsving og 1.2 som tar ut energi ved å ikke gi pådrag. Resultatet er at det oscillerer mellom regulatorene. For hvert bytte inn og ut av en regulatortype har det vist seg nødvendig med opptil 4 parametre inn og 4 parametre ut. Dette gjør det nødvendig å tune mange parametre, noe som er tidkrevende. Man kan heller ikke regne med at parametre som fungere i simulering nødvendigvis vil fungere bra på det fysiske systemet.
Selve c-koden brukt for å implementere regulator samt kommentarer og forklaringer til denne er satt i vedlegg L på side 119.

5.4 Implementering av rotasjonshastighetsestimator

Pendelmodellen fra [13] er implementer på to måter. En ideell versjon og en versjon som skal etterligne virkeligheten. Den siste er implementert med en kvantisering på 1024 og lavpassfiltrering på pådraget til pendelen. På målingene er det satt en kvantisering på 1024 både på vinkel og vinkelhastighet. Denne løsningen er noe mangelfull fordi det da er forutsatt at man har perfekte vinkelhastighetsmålinger, noe som ikke er realiteten. Siden regulatorene som er utviklet i [13] baserer seg på full tilstandstilbakekobling, og at det ikke finnes noen målinger på vinkelhastighetene, må dette derfor løses med en tilstandsestimator.



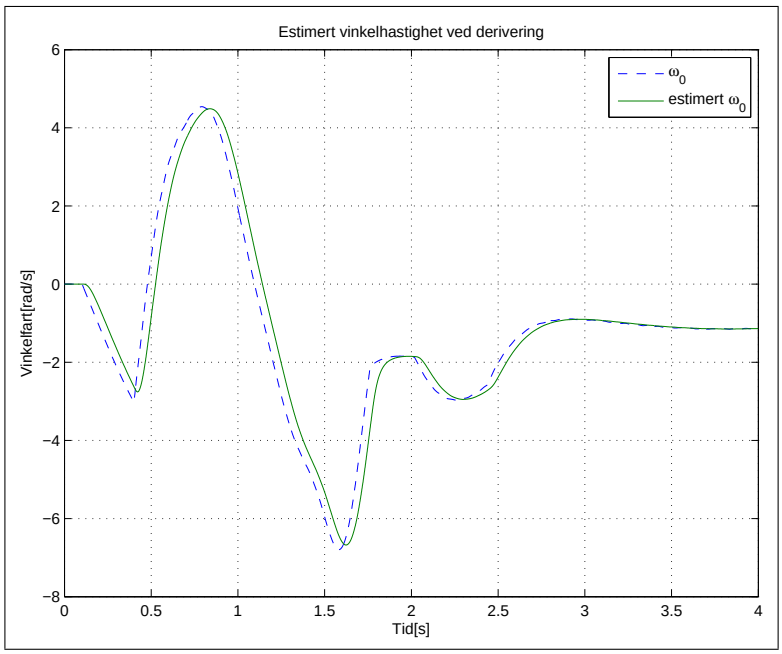
Figur 5.1: Plot viser bytte mellom regulatorer. Her ser man oscillering mellom regulator 1.1 og 1.2 som er et resultat av at det ikke er laget overlapp

5.4.1 Estimering av rotasjonshastigheter ved derivering

Vinkelhastighetene er forsøkt estimert ved hjelp av derivering av vinkelposisjonen. Dette fungerer meget dårlig fordi vinkelposisjonene er kvantiserte. Det er derfor nødvendig å glatte vinkelposisjonen med for eksempel filtrering eller moving average. Problemet med en slik løsning er at det gir forsinkelse på den estimerte rotasjonshastigheten i forhold til den reelle, se figur 5.2 på neste side som er tatt fra simulering. Her ser man den reelle vinkelhastigheten fra pendelmodellen sammen med vinkelhastigheten estimert fra deriveringen. Det er en tydelig forsinkelse på områdene hvor det er dynamikk i systemet samt meget støyfullt på de områdene hvor vinkelen veksler mellom to kvanteposisjoner. Problemet med denne løsningen er at man må gjøre en avveiing mellom forsinkelse og støy.

5.4.2 Estimering av rotasjonshastigheter ved bruk av ulineært kalman-filter

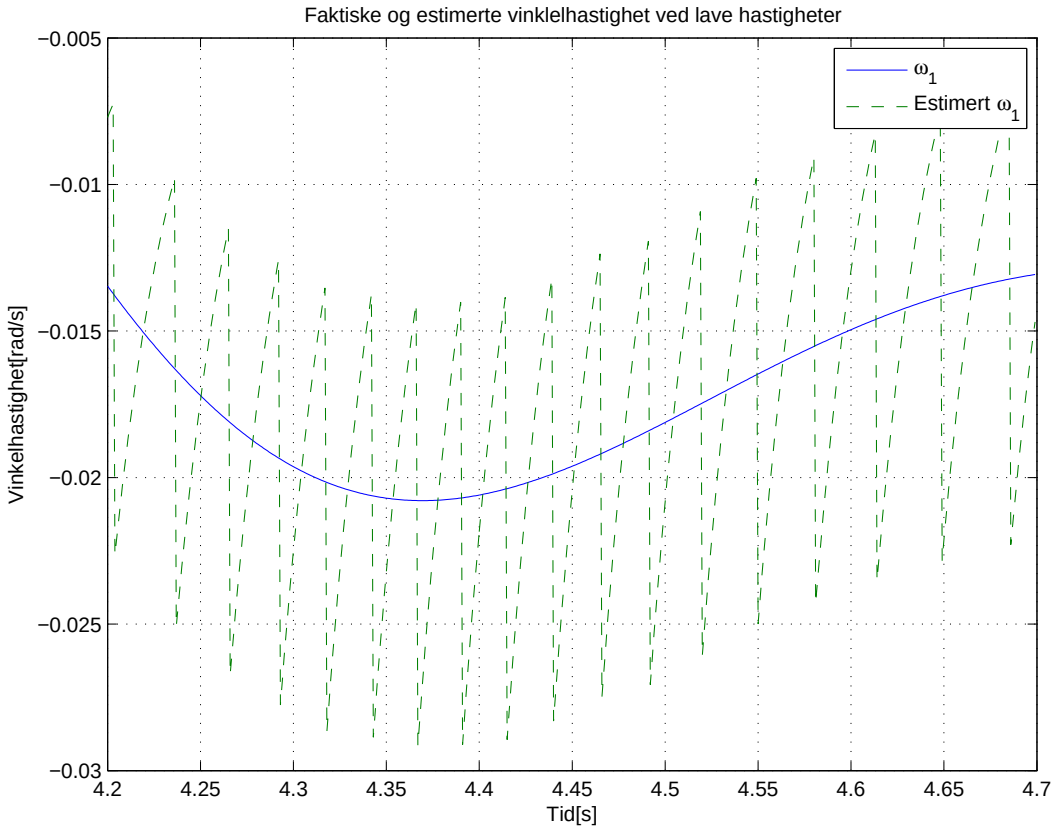
Implementasjonen er også her gjort i level 2 c-file s-funksjon. Implementasjonen er og optimalisert ved at alle matrisemultiplikasjoner, summasjoner og inverteringer er utført på forhånd. Det vil si at hver enkelt likning for hvert enkelt element i alle matrisene til kovarians prediksjon, kovarians filter osv er skrevet inn direkte. Dette ble gjort mulig ved å bruke `Maple` til å foreta den algebraiske utregningen på forhånd. Deriveringen av $f(x, u)$ måtte derimot gjøres for hånd.
Mer om hvordan implementasjonen er utført og kommentarer finnes i vedlegg H på side 109.



Figur 5.2: Plot viser forsinkelsen mellom faktiske vinkelhastighet og vinkelhastighet estimert ved filtrering og derivering

5.4.3 Generelt problem med tilstandsestimering fra kvantisert signal

Problemet med alle løsningene for estimering av vinkelhastighet er det kvantiserte signalet. Estimatorene prøver å svinge seg inn til dette, og selv på estimatorer som tar i bruk pendelmodell fører dette til nærmest hopp i estimert hastighet, se figur 5.3 på neste side. Ved balansering på det reelle systemet skaper dette problemer, og i kombinasjon med slark i det fysiske systemet og forsinkelser kan dette føre til oscillasjon.



Figur 5.3: Plot viser reell vinkelhastighet og estimert vinkelhastighet for **arm_1** ved bruk av ulineært kalmanfilter

Kapittel 6

Forandringer i programvare og maskinvare som er et resultat av forsøk på parameterestimering

De påfølgende kapitlene er kronologisk ut fra arbeidets gang med å finne parametre til systemet. Underveis i arbeidet er det oppdaget problemer som ikke var ventet og som da har gjort det nødvendig å endre retning på det videre arbeidet. Det kunne blitt rotete og vanskelig å forstå hvordan problemer er oppdaget og løst om arbeidet med parameterestimeringen hadde vært presentert på en annen måte. Presentasjonen her viser utviklingen og tankegangen underveis. Dette kapittelet vil derfor inneholde mye arbeid rettet mot både programvare og maskinvare som ikke er direkte relatert til parameterestimering. Dette er fordi problemer underveis har resultert i at det har vært nødvendig å gjøre mye forandringer for å oppnå et fungerende fysisk system hvor det er mulig å utføre parameterestimering. For eksempel ble omgjøring av driver og implementasjon av watchdog utført i en periode da parameterestimeringen ikke fungerte og feilen ikke ble funnet. Da implementasjonen var ferdig fanget den nye programvaren opp feilen på en slik måte at det var enklere å spesifisere hvor i systemet den lå. På samme måte er de resten av underkapitlene relevante på veien mot et system som egner seg for parameterestimering.

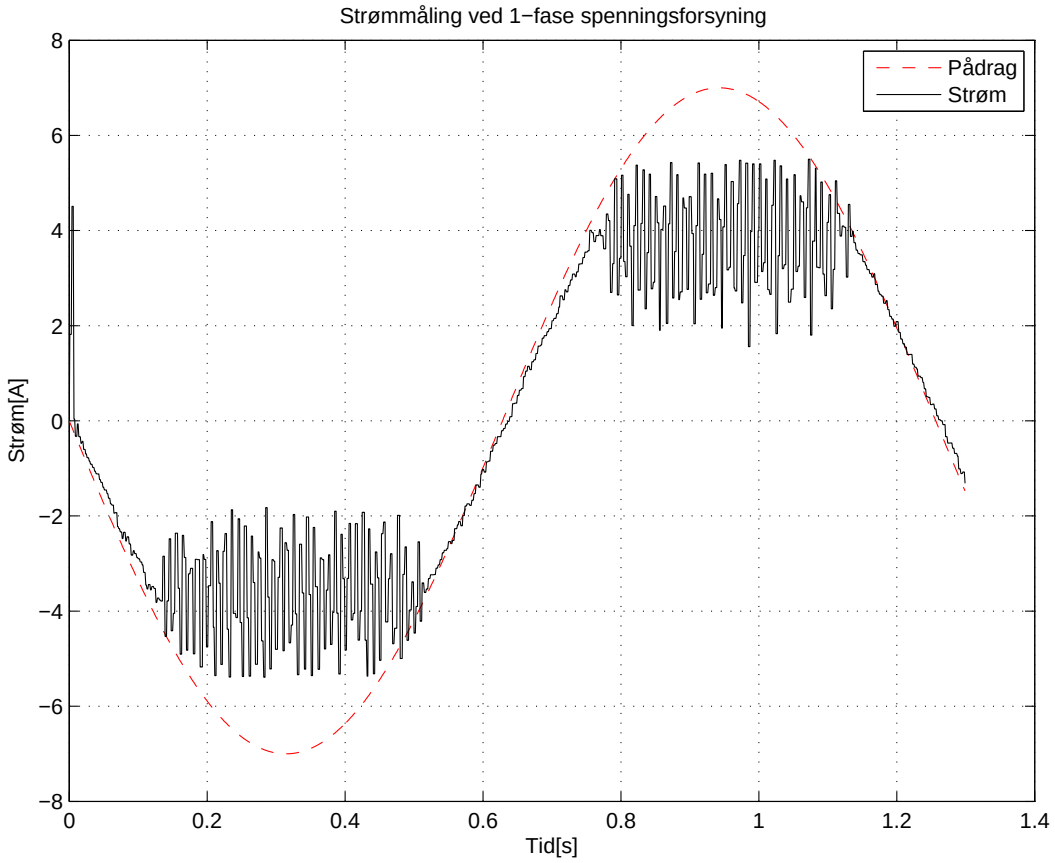
6.1 Feilaktig strømmåling

Det første og viktigste som måtte undersøkes da arbeidet startet på det fysiske systemet var om det var mulig å få ut parametrene fra regulator og motor. Det var i starten antatt at målingen fra regulatorkortet som angir strøm ga ut toppverdier som tilsvarer 12A siden dette var oppgitt som maksimal kontinuerlig strøm. Her har det nok en gang vært noe forvirring i forbindelse med databladet fra `Baldor`. Det viste seg at strømmålingen gir ut toppverdier tilsvarende 30A som tilsvarer maksimal peak-strøm. Dette ble oppdaget helt i slutten av diplomoppgaven. Det skal nevnes at det i rapporten til [6] er funnet en kopi av karakteristikken til regulatorkortet hvor det står at peak-strøm tilsvarer 32A. Hva som er riktig er ikke funnet ut. Dette forklarer unormalt lave strømmålinger som vises i

påfølgende plot. Første del av dette kapittelet vil derfor inneholde plot med utslag som tilsvarer en tredjedel av riktige verdier, dette for å vise noe av motivasjonen for bytte av utstyr som resulterer i mulighet til å trekke større strømmer. De riktige strømverdiene vil allikevel oppgis i parentes. Starten av parameterestimeringen er altså utført i den tro at det gikk mye mindre strøm gjennom transformator, likeretter, regulator og motor enn hva som faktisk var tilfellet.
Det er blitt gjort forsøk på parameterestimering av motor og regulator ut fra modellene fra kapittel 4.7 på side 20. Måten dette er blitt utført på, er ved å bruke ett sett med pådrag på systemet og så måle tilstandene. Disse dataene er så brukt for å beregne parametre til modellene. Det er brukt en samling av sinus-signaler med forskjellig frekvens som pådrag for å få ett så rikt signal som mulig. Det er og prøvd med firkant signaler. I alle forsøk ble det registrert store ulinearitеter som gjorde at parameterestimeringen aldri ble vellykket. Denne ulineariteten kan ses i figur 6.1 på neste side, og her må altså strømtrekket ganges med 3 for å få riktig verdi. Det må nevnes at figurene i dette og påfølgende kapitler er laget etter at det ble funnet en metode for å bruke regulatorkortet fra `Baldor` som kun strømregulator. Det var altså ikke mulig å få ut noe plot som på enkel måte viste problemet da regulatorkortet hadde hastighetsregulering. Dette er bakgrunnen for at det har gått med mye tid for å analysere problemet og finne en løsning.
Problemet kommer av at likeretteren ikke klarer å levere nok strøm i forhold til hva hastighetsregulatoren gir som pådrag til strømregulatoren. Siden det er den interne hastighetsregulatoren til `Baldor` som bestemmer strømpådraget til motor, er det ikke mye annet å gjøre enn å sørge for at det blir nok strøm tilgjengelig.

6.2 Bytte av transformator

Transformatoren som var montert var av 1-fase type noe som det ikke ble lagt merke til i løpet av prosjektperioden. Likeretterkortet er beregnet å ta inn 3-fase 150V. Dette har ført til at maks strøm levert til motor var ut fra tidlige målinger antatt å være 3.5A(10.5A). Ved å bytte til tre-fase spenningsforsyning fra en `variac` transformator økte den fra 3.5A(10.5A) til 5A(15A), som vist i figur 6.2 på side 38.
Siden `variac`-transformatoren kun var på utlån fra elkraft ble det forsøkt å finne en permanent løsning. Det er utprøvd en gammel tre-fase-transformator som er beregnet for å øke spenning fra 220V til 440V. Ved å bytte mellom primær og sekundærspoler fungerer denne nå fra 220V til 110V som likerettet blir 165V. Dette er helt på grensen for spesifikasjonene fra `Baldor` som er 150V $\pm$10%. Problemet med denne transformatoren var først at den ikke klarte å levere mer enn 2A(6A). Grunnen til dette var at det var montert variable høyeffektsmotstander på hver av de tre transformatorene. Da disse ble fjernet økte konstant strømtrekk til 4A(12A).
Siden det fortsatt ble antatt at strømmen var for lav i forhold til hva som burde kunne oppnås er det derfor funnet et stort kondensatorbatteri som gjør det mulig å opprettholde et strømtrekk på 7A(21A) i korte øyeblikk, se figur 6.3 på side 39. Siden strømtrekket altså har vist seg å være 3 ganger høyere enn først antatt og kondensatorbatteriet klarer å levere store strømmer i korte øyeblikk i forbindelse med regulering av systemet, er det

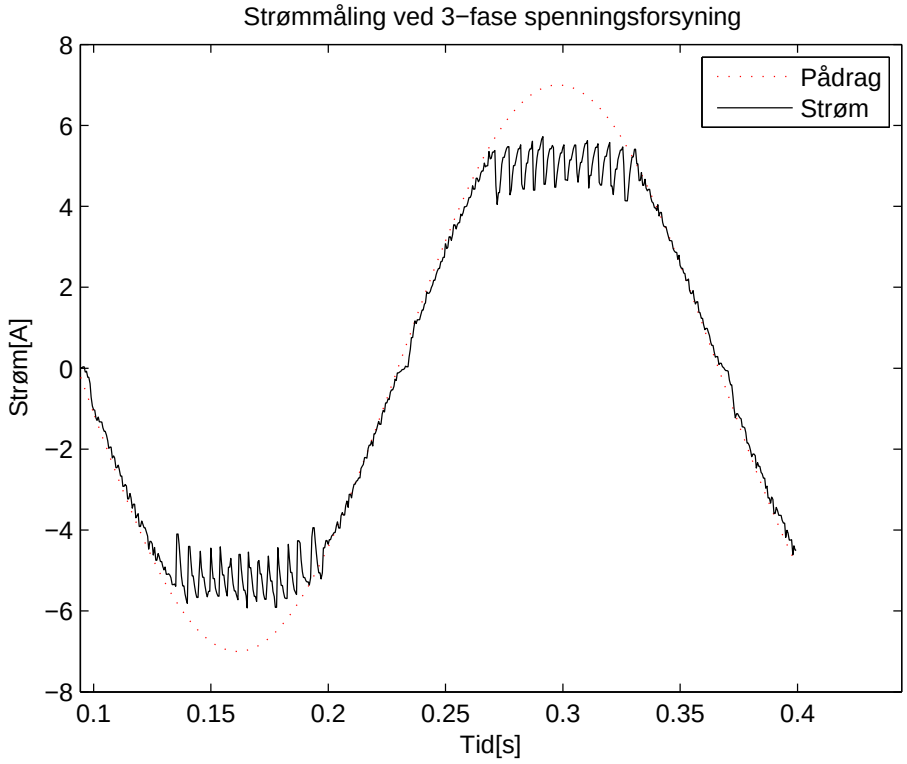


Figur 6.1: Plot viser at likeretteren ikke klarer å levere nok strøm når den har 1-fase spenningsforsyning. Maks strømtrekk ligger på ca 10.5A(3.5A)

gått tilbake til 1-fase transformatoren som er innebygd i regulatorkabinettet. Som nevnt i neste kapittel er det satt på hurtigkontakter for om ønskelig kunne bytte transformator på en enkel måte.

6.3 Installering av vifte og generell opprydding i regulatorkabinett

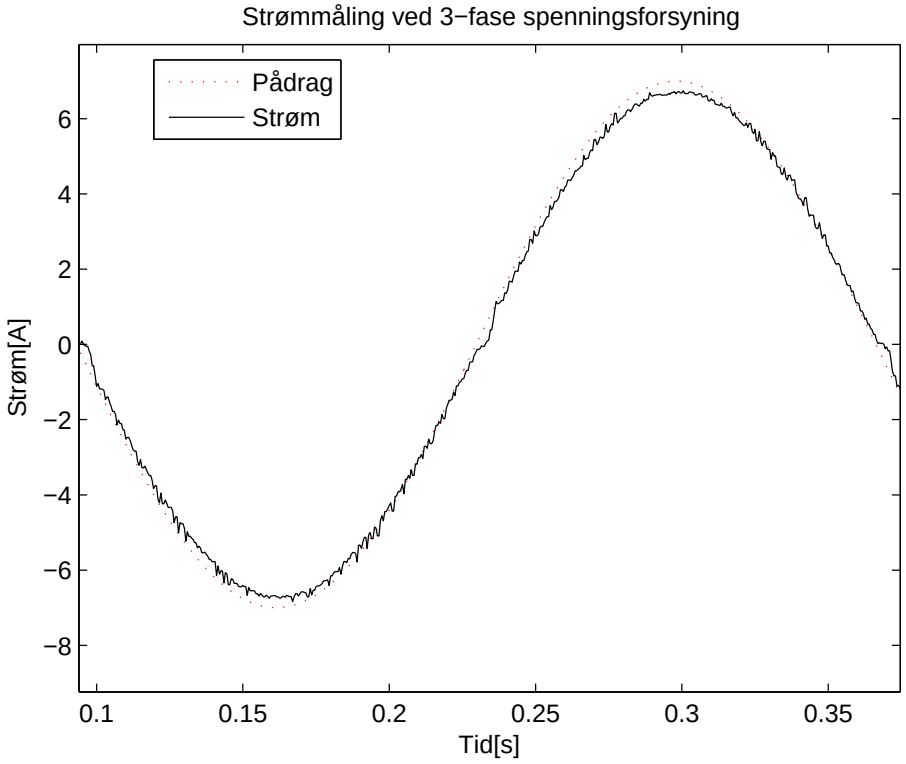
Med de større strømmer som blir trukket gjennom systemet under parameterestimering blir både motor, regulator og likeretter varme. Det er derfor montert vifte i kabinettet for å kjøle regulator og likeretter. Motoren er det vanskelig å montere kjøling på så den må man kontrollere for hånd om man trekker mye strøm over lengre tid, noe som ikke skjer under vanlig bruk. Bilde 6.4 på side 40 viser regulatorkabinett etter installering av vifte, kondensatorbatteri og bytte til mindre PSU for forsyning av nodene. Det er og montert banankontakter på forsyningsledningene til likeretter slik at det enkelt kan veksles mellom intern 1-fase transformator og ekstern 3-fase transformator.



Figur 6.2: Plot viser at likeretteren heller ikke klarer å levere nok strøm når den har 3-fase fra en variac spenningsforskyvning. Maks strømtrekk ligger på ca 5A(15A)

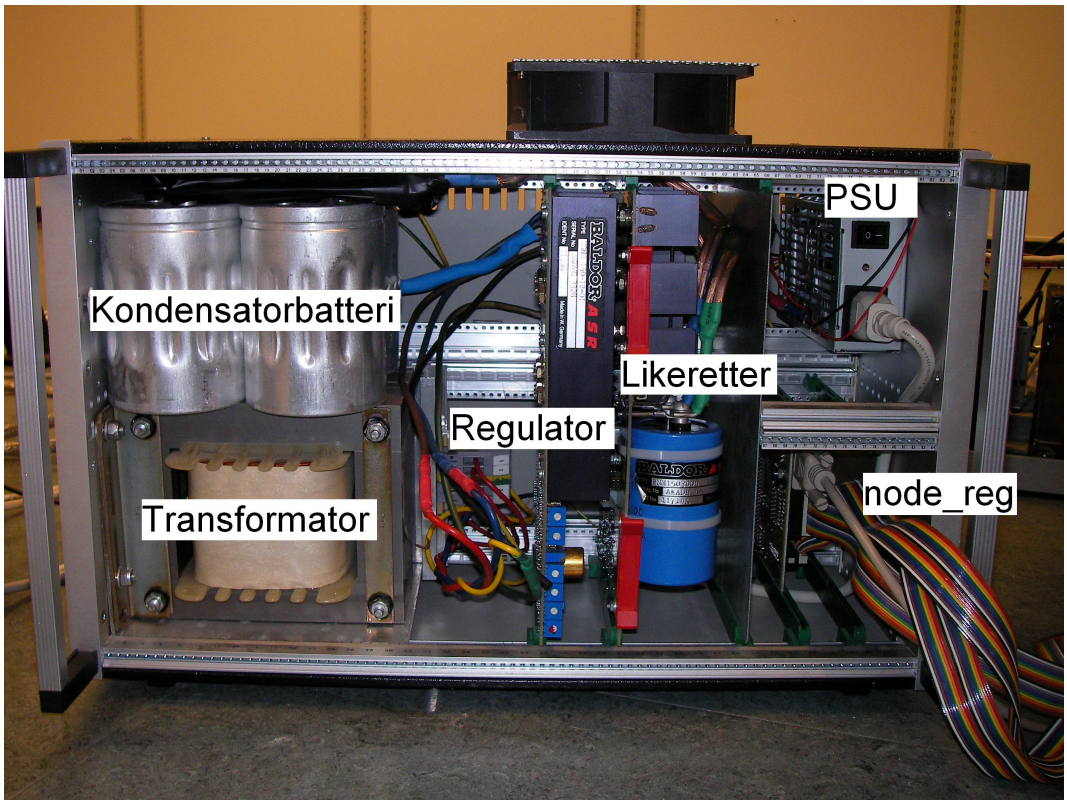
6.4 Metode for å omgå rotasjonsregulatoren

Da noe av ulineariteten på strømmålingen var løst ble det gjort videre forsøk på parameterestimering. Disse forsøkene ble gjort uten belastning på motoren, noe som ikke fungerte i det hele tatt. Ved å montere på masse med lavere inertia enn `arm_0` var det mulig å få parametre som viste tendenser til å stabilisere seg. Ved å øke inertia enda mer ved å montere `arm_0` ble det registrert mye rare lyder som tydet på at regulatorene eller transformatorene gikk i metning. Dette ble prøvd justert på de tre forskjellige potmetre som styrer forsterkning til tachometer-spenning, forsterkning til pådragsspenning og forsterkningen i rotasjonsregulatoren. Dette viste seg å være vanskelig, spesielt om man monterte `arm_0` på motorakslingen. Det endte da opp med to motstridigheter. Man ønsker mye masse på akslingen for å få en presis parameterestimering, mens rotasjonsregulatoren ikke likte at det ble montert ekstra masse. Det ble derfor antatt at rotasjonsregulatoren spilte såpass negativt inn på systemet at det ble vanskelig å estimere parametre. I ettertid har det derimot vist seg at det kan ha vært flere andre grunner til problemene og disse blir gjennomgått i senere kapitler.
Det at motordynamikken ble for ulineær skapte til slutt så mye frustrasjoner at det ble lagt



Figur 6.3: Plot viser forsøk med å gi 7A(21A) pådrag. Dette kan anses som grensen for hva 3-fase transformator og kondensatorbatteri klarer å levere

inn betraktelig arbeid på å finne en løsning for å bruke eksisterende `Baldor`-regulatorkort til kun strømregulering. Igjen må det nevnes at det har vært vanskelig å orientere seg i bruksanvisningen til `Baldor`. Bilder og informasjon som er relatert til hverandre er plassert spredt utover i hele dokumentet, uten at det noen sted finnes informasjon om at det er mulig å bruke kortet som kun strømregulator.
Kortet inneholder diverse testpunkter, som kunne vært brukt for å gi inn signal til strømregulatoren. En slik løsning ville kreve at man fysisk kutte baner og slik omgår rotasjonsregulator. Dette kunne vært en mulighet siden det så ut som strømregulatoren og hastighetsregulatoren er plassert på to deler av kretskortet. Men andre mindre drastiske muligheter ble først undersøkt. En av dem kunne vært å benytte seg av tilgjengelige pinner som er beregnet for å kunne koble til eksternt potmeter for begrensning av strømpådraget til motor. I kombinasjon med å gi makspådrag til turtallsregulatoren kan man justere strømpådraget ved å bruke et digitalt potmeter for å gi strømbegrensning. Heldigvis ble det ved nærmere analyse og testing funnet at det derimot ved hjelp av disse inngangene er mulig å gi signal direkte inn til strømregulatoren. Det gis altså et spenningssignal inn på pinnen `armature current command signal`. Dette gjør det nå mulig å omgå rotasjonsregulatoren på en enkel måte. Dette gjør arbeidet med parameterestimering av motor og



Figur 6.4: Bildet viser regulatorkabinett etter modifisering

regulator samt implementering av momentregulering betraktelig enklere. Dette gjør og at man kan benytte modellen fra kap 4.7 på side 20 som kun inneholder strømregulatoren som modell for parameterestimering.
Det at rotasjonsregulatoren er omgått har gjort det mulig å omgå opptil 6 forskjellige potmetre for justering av forsterkninger og begrensninger. Det er kun et potmeter igjen som man kan benytte for å justere forsterkningen i strømregulatoren. Denne er nå justert optimalt for motoren ut fra hva som er oppgitt i databladet. Dette kan man se på innsvingningen som er gitt i figur 6.8 på side 45 som er tatt fra oscilloskopmåling. Som oppgitt i databladet har strømmen her to oversving.
Noe som ble oppdaget da direkte strømpådrag til regulator ble oppnådd var at I/O-kortet ikke gir ut mer enn ±5V og det var derfor ikke mulig å gi høyere pådrag enn 5A(15A). Bruksanvisningen som finnes for I/O-kortet, og sier at det er mulig å velge mellom 5V og 10V som kilde for DA, er desverre for en nyere versjon av kortet (PCL812PG) og det gjelder altså ikke for PCL812. For å løse dette er det blitt byttet om to motstander på `node_reg` for å doble forsterkningen i differensialforsterkeren, se [14].
Det ble og funnet at potmeter VR1 på `PCL812` gjelder for å sette maksutslag på D/A 1, mens potmeter VR2 setter maksutslag på D/A 2(med klokka for mindre utslag), noe som heller ikke er oppgitt i databladet

6.5 Omgjøring av drivere og annen programvare

Nye forsøk er igjen blitt utført for å finne parametrene, men nå uten rotasjonsregulator. Denne gang ble det påtrykt et firkantsignal direkte til strømregulatoren med høy frekvens(10Hz). Ved å lytte til systemet ble det nå registrert at frekvensen ikke ble opprettholdt, men med jevne mellomrom fikk en liten tidsforskyvning, noe som kan ha bidratt til de observerte ulinearitetene fra tidligere kapittel. Dette problemet ble det brukt lang tid på å finne en løsning på og det ble derfor i mellomtiden implementert nytt system i driverhierarkiet.
Systemet med drivere og kommunikasjon mellom driver er omgjort i forhold til hva som var gjort i [14]. Figur 6.5 på neste side viser hvordan løsningen var gjort. `Device driver Trippelinvertert Pendel` var implementert som en device-manager hvor man på en enkel måte kunne få ut vinkelverdier og strømmåling samt gi pådrag til systemet. Denne løsningen har derimot ulempen med at det skjer en bufring av alle vinkelverdier fordi de må dynamisk leses ut fra serielldriveren `devc-ser8250`. Og siden blir også vinkelverdiene bufret med jevne mellomrom i s-funksjonen. Her er det altså et unødvendig lag som riktig nok gir en enkel kobling mot pendelsystemet, men resulterer i forsinkelser på målinger og unødvendig bruk av datakraft. Omgjøringen som er gjort er at s-funksjonen er satt til å lese av verdiene direkte fra både serielldriveren og PCL812-driveren og lagrer dem i global buffer slik at de blir tilgjengelig for RTW. Figur 6.6 på side 43 viser dette nye oppsettet.

6.5.1 Watchdog

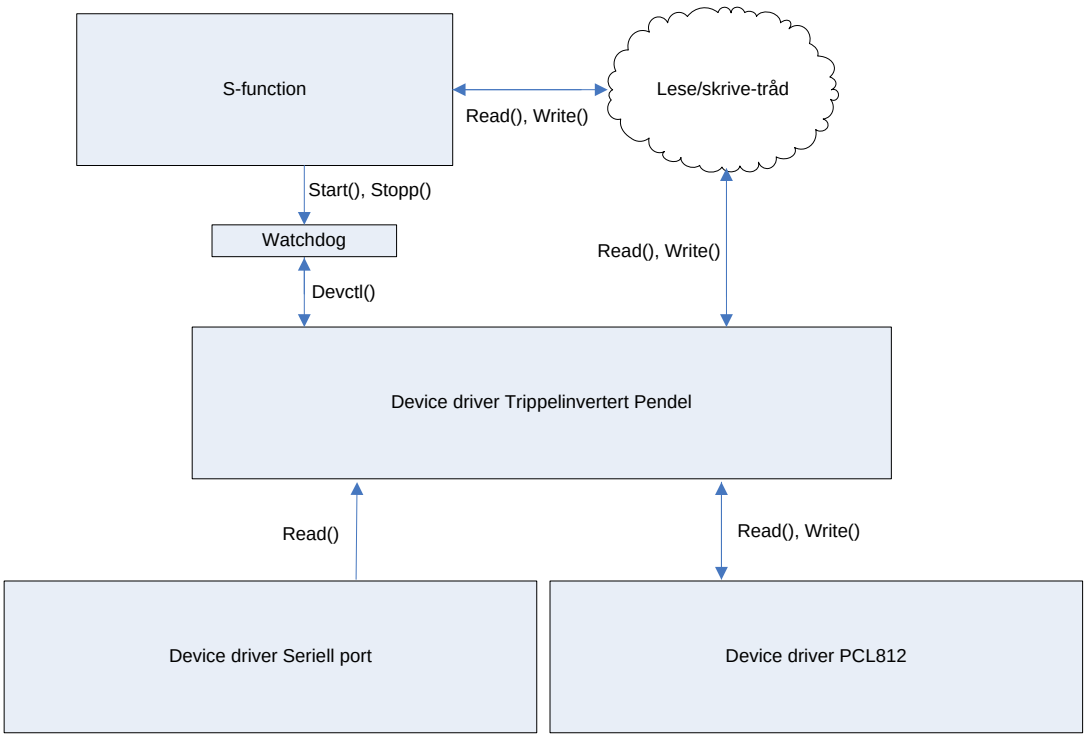
Grunnen til at det er installert en wachdog i systemet er ønsket om sikkerhet. Det er derfor watchdog er satt til å styre et signal mot `Baldor`-regulatorkortet som bestemmer om det skal kunne gis pådrag til motor. Dette signalet heter `total disable` og er grundigere kommentert i [14]. Watchdog stiller med 3 forskjellige `devctl`-kall som en prosess kan benytte seg av etter at den har opprettet kontakt med driveren ved hjelp av standard unix-kallet `open()`. Kallene som er tilgjengelig er

- devctl(fd_watchdog, WATCHDOG_DEVCTL_START, &samplingsfrekvens, **sizeof**(samplingsfrekvens), NULL) - Dette kallet sørger for å si fra at watchdog skal starte opp og sørge for at det er mulig å gi pådrag til motor. Samplingsfrekvensen til RTW blir og gitt med slik at watchdog kan avgjøre om denne blir overholdt av RTW.

- devctl(fd_watchdog, WATCHDOG_DEVCTL_UPDATE, &vinkel_0, **sizeof**(vinkel_0), NULL) - Dette kallet sørger for å gi tilbakemelding til watchdog hver samplingsperiode for å si fra om at det fortsatt er liv i RTW. Det blir og gitt med `vinkel_0` slik at watchdog kan overvåke om rotasjonshastigheten er for høy.

- devctl(fd_watchdog, WATCHDOG_DEVCTL_STOP, NULL, NULL, NULL) - Dette kallet sier fra til watchdog at RTW holder på å avslutte på normal måte.

Alle disse kallene blir sendt via s-funksjon-wrapperen `s_func_trippelinvertert_pendel_wrapper` og sier i fra om at reguleringen av pendelsystemet er oppe og kjører. Hvis ikke WATCHDOG_DEVCTL_U



Figur 6.5: Kommunikasjon mellom device managere og S-function slik det var implementert fra prosjektperioden. Watchdog var vel og merke ikke implementert.

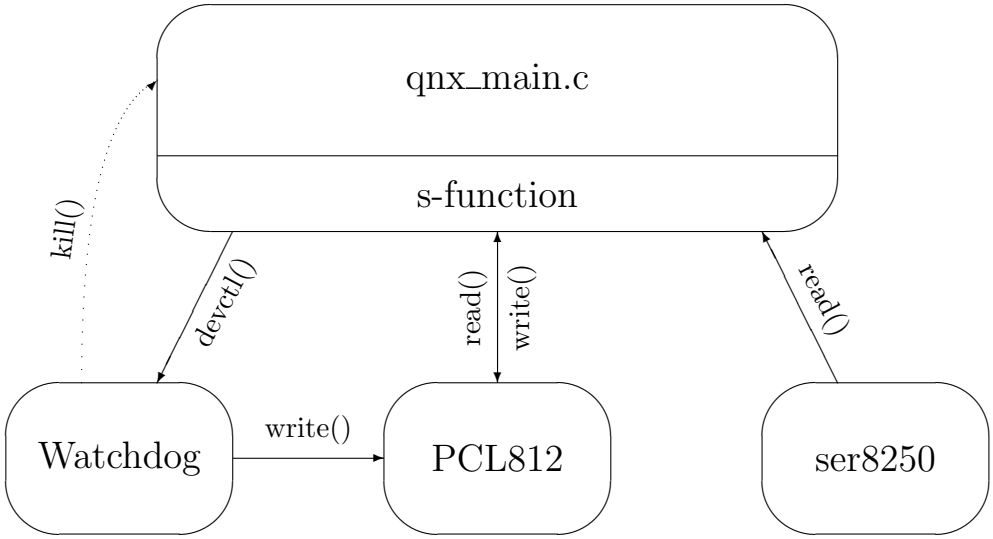
blir sendt med riktig periode vil watchdog sørge for å skru av pådrag til motor og drepe RTW-prosessen. Det samme skjer om `vinkel_0` har en slik verdi at rotasjonshastigheten er for høy. Implementasjonen av watchdog og spesifikke kommentarer til hvordan deler av koden fungerer er lagt i vedlegg M på side 123.

6.5.2 s_func_trippelinvertert_pendel

Denne s-funksjonen er satt sammen av to filer, **s_func_trippelinvertert_pendel** hvor rammeverket er generert ved hjelp av `s-function-builder`-blokken i `Simulink` og tilhørende wrapperfil `s_func_trippelinvertert_pendel_wrapper` som er filen hvor man setter inn ønsket brukerkode. Her er altså all nødvendig kode for kommunikasjon mot det fysiske pendelsystemet og watchdog implementert. Hvordan hoveddelen av denne implementeringen er gjort og kommentarer til dette er satt i vedlegget J på side 113.

6.5.3 Prioriteter og qnx_main.c

Da watchdoggen ble implementert og systemet igjen ble testet ble det oppdaget flere svakheter. Watchdoggen er satt til å stoppe systemet hvis RTW overstiger sitt periodekrav på $2 * (\text{periodetid})$. Dette periodekravet ble oversteget hver eneste gang. Det ble så implemen-



Figur 6.6: Figuren viser koblingen mellom RTW og drivere

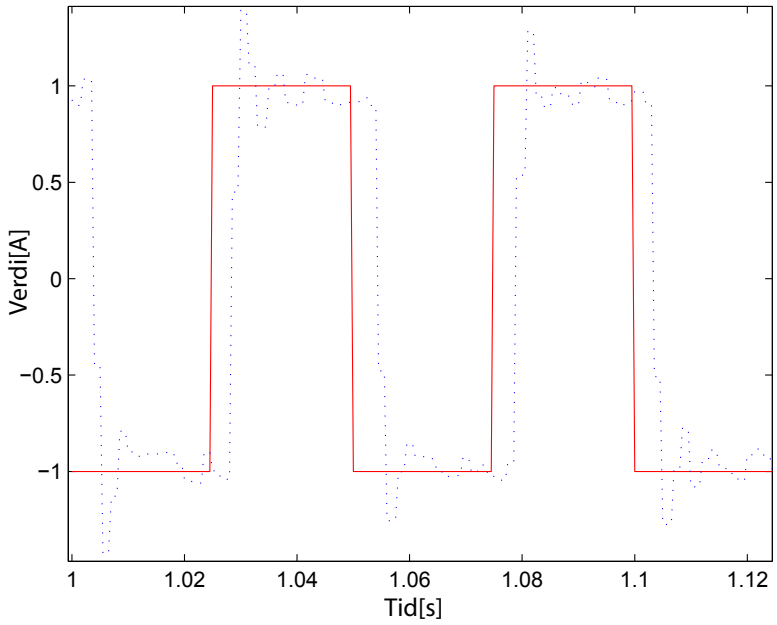
tert kode for å kunne analysere periodetid og utførelsestid på `Rt_OneStep()`. Dette er en tråd som er en del av `qnx_main.c`. Denne kjører hvert tidsskritt og tar seg av utregningen av innholdet i simulinkdiagrammet. Det er kun worst-case periodetid og utførelsestid som blir lagret, men disse forteller jo også alt. Man kan under ingen omstendigheter godta at disse øker urimelig mye. Det er viktig å merke seg at forandringer som gjøres i **qnx_main.c** må gjøres i fila som er lagret på `target_pc`.
Worst-case tiden for perioden ble observert til å være over 10ms selv om den var satt til 1ms. Det viste seg at det i `qnx_main.c` ikke var satt noe økt prioritet på denne prosessen i forhold til en hvilken som helst annen bruker prosess i QNX. Prioriteten var derfor satt til standard verdien 10 på en prioritetsskala som går fra 1-256 hvor 256 er høyeste prioritet. For et sanntidssystem er dette alt for lavt, spesielt med tanke på at dette er standardverdien som alle prosesser får om man ikke aktivt velger en annen prioritet. Ved å øke denne prioriteten opp til over 100 kjører nå RTW uten å bli stanset av watchdog.

6.6 Forsinkelse i strømmåling

Nå som strømpådrag var mulig og prosessene overholdt sine perioder ble det igjen påtrykt et firkantsignal for å undersøke responsen og om mulig gjøre parameterestimering. Det ble da oppdaget en forsinkelse mellom pådrag og strømmålingen da disse ble kontrollert opp mot hverandre. I figur 6.7 på neste side ser man denne forsinkelsen som er på $5-6$ms. For å undersøke nærmere hva som forårsaker denne forsinkelsen er det blitt koblet inn oscilloskop på inngangen til regulatorkortet og på utgangen(strømmålingen). I figur 6.8 på neste

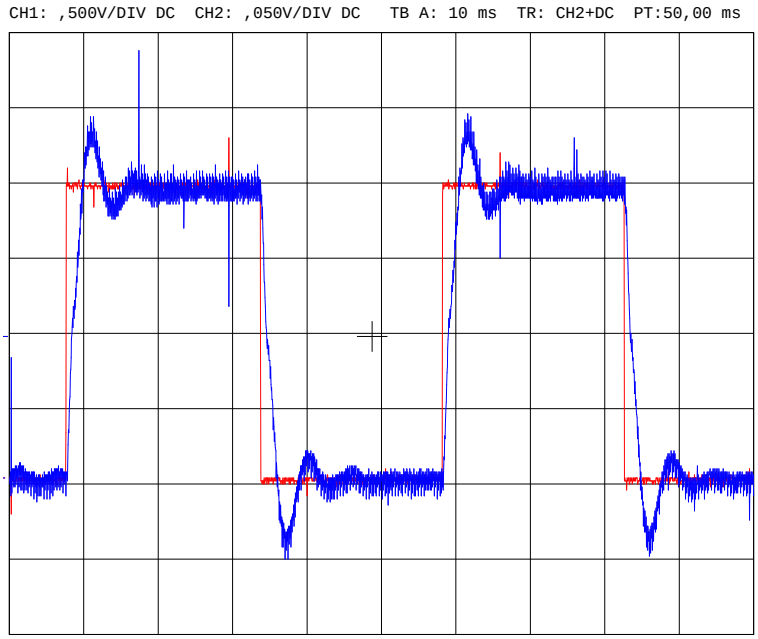
side ser man pådragssignalet og det målte signalet. Figuren viser responsen til strømregulatoren. Her er det ikke noe synlig forsinkelse. Forsinkelsen ligger da enten mellom RTW og pådragsinngangen, eller mellom strømmålingsutgangen og RTW. Tester og undersøkelser viste utrolig nok at feilen var et resultat av en funksjon sleep(1), som normalt skulle gi forsinkelse på 1ms. Årsaken til at denne har gitt forsinkelse på 5ms er ikke funnet. Siden det ikke skapte noen problemer å fjerne funksjonen ble dette gjort som løsning på problemet. Figur 6.9 på neste side viser strømmålingen etter at problemet er rettet. Her ser man at forsinkelsen er nede i 1-2ms.
Figuren som er fra oscilloskopet som viser hvor raskt strømregulatoren klarer å svinge inn strømmen i forhold til pådraget, verifiserer at det i de modellbaserte tilstandsestimatorene kan ses bort fra strømsløyfen og anta at $i_a \approx i_d$.

Disse siste justeringene gjør det nå mulig å få til vellykket parameterestimering på sys-

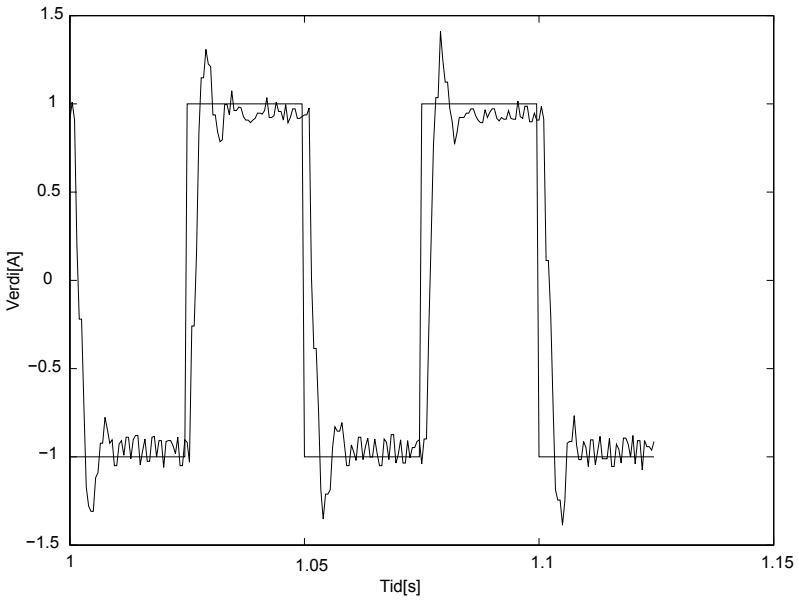


Figur 6.7: Plot viser forsinkelsen mellom pådrag og måling av strøm i **Simulink**. Forsinkelsen er 5-6ms

temet. Resultatene gjennomgås i kapittel 8.1 på side 61.



Figur 6.8: Plot viser reaksjonen på regulator mellom pådrag og strømmåling



Figur 6.9: Plot viser forsinkelsen mellom pådrag og måling av strøm i simulink etter at problemet er fikset. Forsinkelsen er nå 1-2ms

Kapittel 7

Fornyelse av utstyr og elektronikk

7.1 Datalink mellom node_2 og node_0

Fra prosjektperioden ble det registrert at radiolinken på `node_2` som sender vinkelen for `arm_3` til `node_0` ikke fungerte tilfredsstillende. Det er blitt gjort flere forsøk på å finne og om mulig rette problemet. I databladet [9] er det spesifisert antenne laget av å spinne 0.5mm enamelert kobbertråd til en spole. Fra prosjektet var det benyttet en mye tynnere enamelert aluminiumstråd. Det er forsøkt å bytte til spesifisert tråd uten hell. Også forsøk på å trekke ut lengden på antennespolen samt justere retningene til antennene mot hverandre, slik som foreslått i databladet resulterte og uten forbedringer. En grundigere undersøkelse hos produsenten viser at de selger spesialiserte radiosendere for UART/RS232 med innebygd koding og feilsjekk. Problemet med de valgte radiobrikkene er at de ikke inneholder noen form for koding av signalet og er derfor følsomme for frekvensstøy, faseskift og jitter. Man kunne ha løst noe av problemet ved å implementere for eksempel manchesterkoding men dette ville føre til en halvering av den tilgjengelige overføringshastigheten på 10000bit/s noe som er uønsket.
Det er og undersøkt andre muligheter i programvare for å gjøre overføringen robust nok. Overføringen fungere ved at tellerverdien til kvadraturtelleren overføres til `target_pc` og der omgjøres til en vinkelposisjon. Det registreres kun om tellerverdien har inkrementert eller dekrementert noe som fører til at en buffer som representerer vinkelen teller opp eller ned. Denne løsningen hindrer problemer med overflow. For å øke robustheten kunne man helt ute på `node_2` beregnet vinkelposisjonen ut fra tellerverdien til kvadraturtelleren, slik at `target_pc` får den faktiske vinkelposisjonen og ikke trenger å regne ut denne selv fra datapakker som kan inneholde feil. Denne løsningen derimot ville kreve at man øker meldingslengden fra 2 byte til 3 byte for ikke å risikere overflow. Det er ikke sikkert at denne løsningen ville gitt en overføring som er stabil nok til å kunne bli brukt i regulering på systemet, på grunn av lavere samplingsintervall og fordi vinkelposisjonen som gis til `Simulink` allikevel ikke kan garanteres å være riktig.

7.1.1 Design av IRDA-overføring

På grunn av problemene nevnt over med radiolinken er det derfor valgt å konstruere en ny løsning for overføring av tellerverdien fra `node_2`. Løsningen som er valgt er `IRDA`, noe som og er anbefalt som alternativ løsning i [14]. Fordelen med `IRDA` er at man kan oppnå hastigheter helt opp til 4Mbs. Selv om det ikke er nødvendig med så høye hastigheter på dette systemet er det uansett bra å ha mulighet til å øke hastigheten opp mot 100kbs. Valg av komponenter er gjort ut fra hva som er lett å implementere på det eksisterende systemet. IRDA må altså kunne styres direkte fra UART slik at det ikke er nødvendig med større modifiseringer av nodene. Kortene er i tillegg laget så små som mulig og trenger altså kun strøm og UART-signal for å fungere. Kortene er satt sammen av en IRDA-kontroller og en IRDA-tranceiver samt nødvendig støtteelektronikk.
IRDA-kontrolleren som er valgt er `mcp2120` fordi det er enkelt å styre denne om man kun har tilgjengelig UART-signaler. Det fantes flere andre kretser å velge mellom, men hvor de enten var mer kompliserte å bruke og dermed trengte mer støtteelektronikk eller man måtte gi inn ett klokkesignal. På `mcp2120` kan man bruke vanlig krystall som klokkekilde, og baudraten er mulig å sette enten i hardware eller i software. På denne måten slipper man kontrollsignaler ved at man kan sette baudraten med jumpere. Det finnes mange tranceivere på markedet og valget falt på `TFDU6102`. Denne har støtte for høye hastigheter har gode overføringsegenskaper og er forholdsvis enkel å koble opp. Ulempen er at den trenger en del støtteelektronikk, blandt annet tantalum kondensatorer.
Alle komponenter som er valgt for kretsdesignet er av type SMD[1] for å gjøre kortene så små som mulig. Figur 7.1 på neste side viser utlegget for kretsen og bildet 7.2 på side 50 viser den ferdig kretsen. Legg merke til loddejumperene som er blitt laget for minimering av plassbehov.

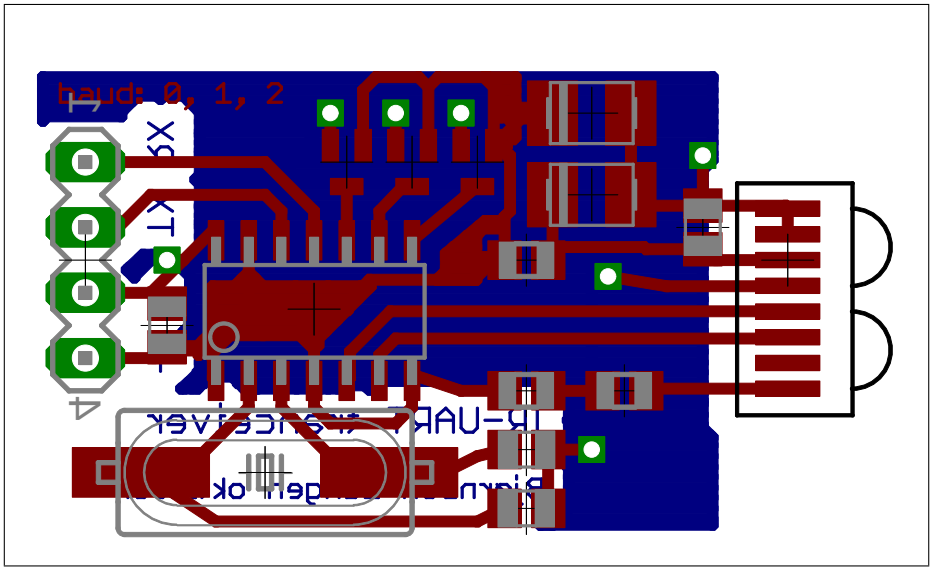
7.2 Implementering av vinkelhastighetsestimering på hardware

problemet med singulariteter i det ulineære kalmanfilteret for trippelinvertert pendel samt at det uansett allerede er ustabilitet i modellbaserte estimatorer allerede ved forsøk på dobbelpendel gjør det nødvendig å estimere vinkelhastighet på annet vis. En mulig løsning er å gjøre dette på hardware ut fra metodene beskrevet i kapittel 4.9 på side 22.

7.2.1 Valg av mikrokontroller og støtteelektronikk

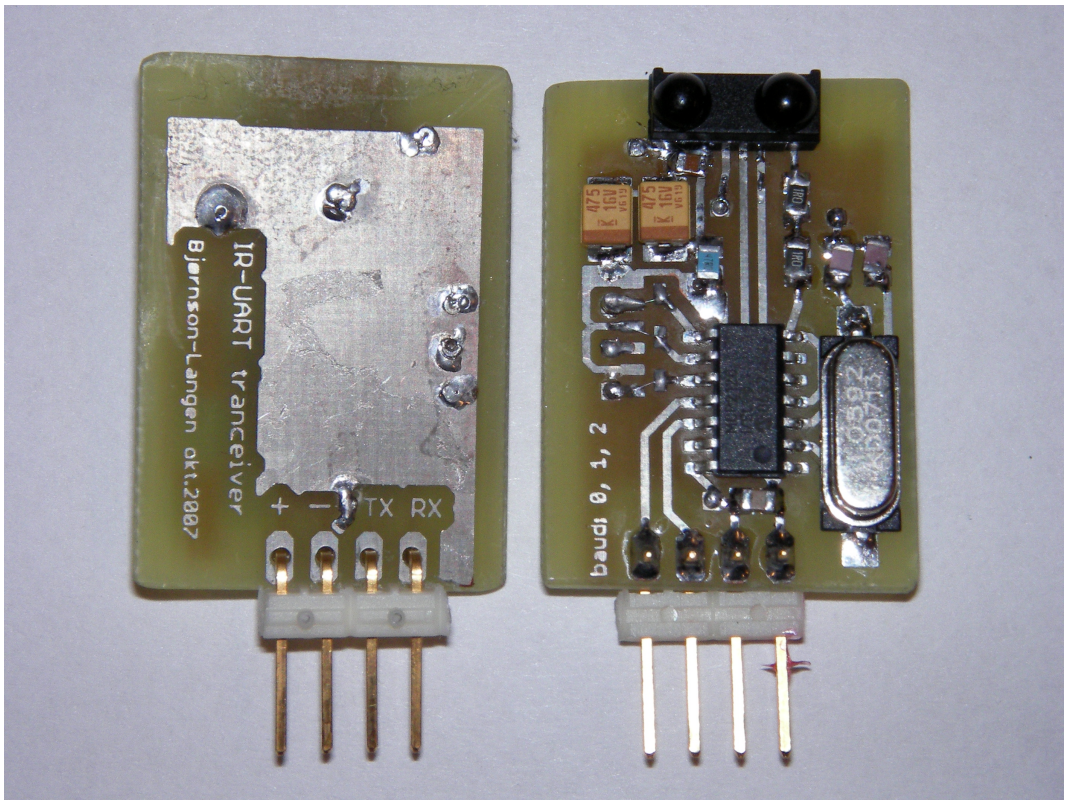
Valg av mikrokontroller falt på XMEGA grunnet støtte for kvadraturteller og mulighet til bruk av eventsystemet for å gjøre det mulig å estimere vinkelhastighet på en enkel måte, se vedlegg C på side 97. Arbeidet med hardwareestimering ble satt igang en måned før diplominnlevering grunnet problemene som er oppdaget med andre estimeringsmetoder. I den sammenheng ble det startet opp et samarbeid med `Atmel` med test av deres nye `XMEGA`-mikrokontroller. Arbeidet med implementasjon av hardwareestimering og design av

[1] SMD(surface mounted devices) - overflatemonterte komponenter



Figur 7.1: Figuren viser IRDA-kretsen

nye kretskort ble startet opp i slutten av oktober 2007 fire måneder før brikken skulle lanseres på markedet. Jobbing med så tidlige revisjoner av mikrokontrolleren har selvfølgelig resultert i flere mystiske feil, men disse har heldigvis blitt løst fort i samarbeid med `Atmel`. Valget av `XMEGA` er ikke noen selvfølge, man kunne liksågodt ha valgt en FPGA som gjort i [8]. Men med en slik løsning måtte man på egenhånd ha implementert både hardware-estimeringen og kvadraturteller. Selv om det har vært noe merarbeidet med en så ny mikrokontroller er det ansett at det har vært en raskere utviklingsprosess enn hva det ville vært på en FPGA.
Mikrokontrolleren er beregnet å kjører på 3.3V. Enkoderene er beregnet kjørt på 5V, men dette er kun en anbefaling i databladet. Det ble derfor testet å kjøre enkoderene på 3.3V. Dette fungerte bra om man ikke har lav rotasjonshastighet. Ved lav rotasjonshastighet ble det observert støy ved tilstandstransisjoner på A og B-signalet. Støy her fører til at tiden som beregnes mellom to transisjoner bli liten, noe som tilsvarer meget høye hastigheter. Ved lav rotasjonshastighet risikerer man da at den feilaktig estimerte høye hastigheten blir gjeldene til det kommer ett nytt estimat som dermed sletter det gamle. Det ble forsøkt med filtrering for å fjerne støy, men da var det nødvendig å gjøre en overveielse mellom om man ønsket å registrere lave eller høye hastigheter. Ved å benytte seg av 5V forsvinner denne problematikken helt. Det er derfor montert to spenningsregulatorer, en for 3.3V og en for 5V på kortene som er laget. For å senke signalspenningen på 5V for A og B signalet ned til 3.3V er det benyttet ett motstandsnettverk.
For overføring er det valgt kretsene `MAX3221` for `RS-232` og `MCP2515` sammen med `SN65HVD233` for CAN. Muligheten for to protokoller er ut fra diskusjonen gjort i [14]. Det er der ansett som en fornuftig måte å muliggjøre bytte til kraftigere `target_pc` hvor I/O-kortet `PCL812` som trenger ISA-port ikke lenger er støttet. Oppsett av CAN-bus og design av ny `node_reg` er da en løsning for å få tilgang til I/O.



Figur 7.2: Bildet viser kretskortene som tar seg av IRDA-linken mellom **arm_2** og **arm_0**

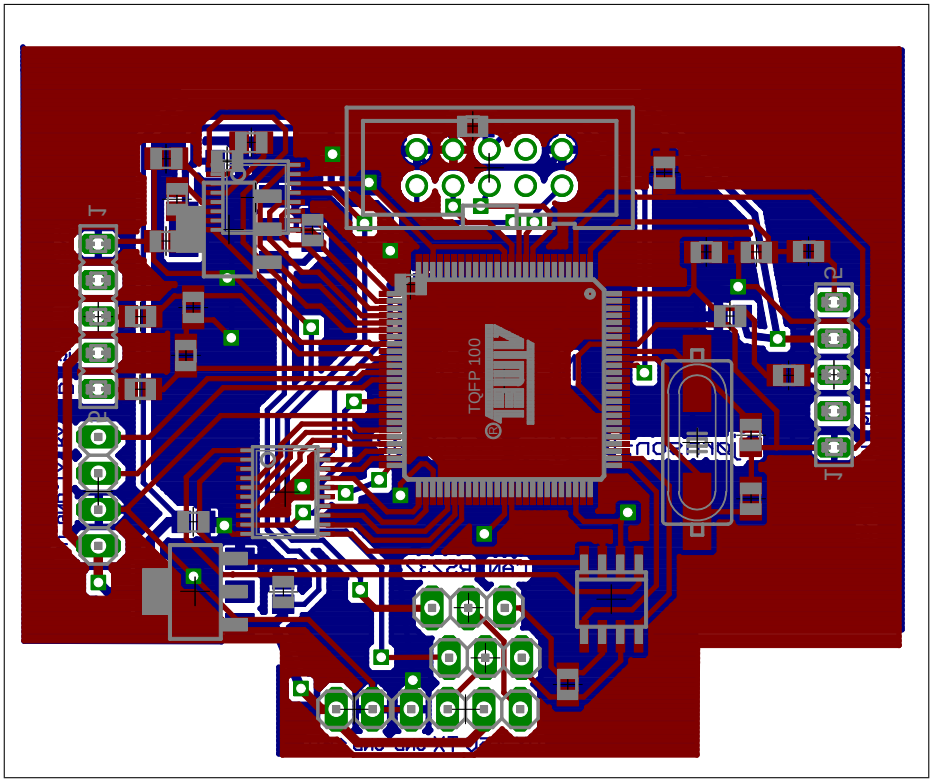
Komponenter som er valgt er generelt av type SMD, dette for å kunne lage kretskortene små.

7.2.2 Node_0

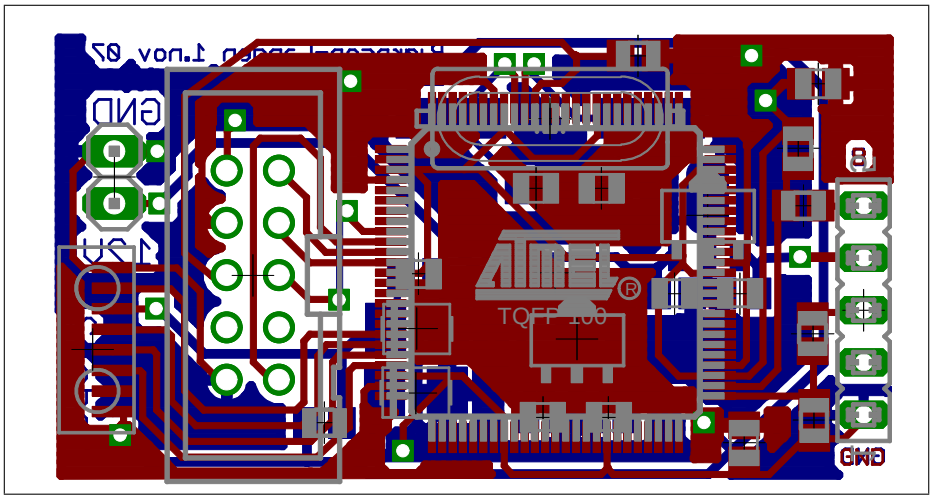
Oppkobling er gjort ut fra databladene, men grunnet dårlig tid er det gjort noen feil. Disse er blitt rettet på kretskortene ved modding. Figur 7.3 på neste side er andre revisjon av utlegget for `node_0`. Her er alle feilene som er oppdaget i hardware rettet. Bilde 7.7 på side 57 viser det ferdige kretskortet montert på pendelsystemet.

7.2.3 Node_2

Dette kretskortet er koblet opp ut fra databladene. I databladet er det oppgitt at `XMEGA` skal ha innebygd IRDA-kontroller. En misforståelse gjorde at det ikke ble registrert at denne kontrolleren ikke var implementert på revisjonen av mikrokontrollerene som ble mottatt fra `Atmel`. Figur 7.4 på neste side viser kretsen som om IRDA-kontrolleren skulle vært implementert, kortet på bilde 7.10 på side 60 viser kretsen som er laget. Denne er blitt moddet i ettertid for bruk av det eksterne IRDA-kontrollerkort som er laget i 7.1.1 på side 48.



Figur 7.3: Figuren viser node_0



Figur 7.4: Figuren viser node_2

7.2.4 Programvare

Programvaren som kjører på mikrokontrollerene er delt opp i funksjoner med spesifikke oppgaver. Siden mye av funksjonaliteten er gitt ut fra oppsett gjort på event-systemet kjører mye av programmet som init-funksjoner. Under følger en forklaring på disse funksjonene.

- `init_klokke()`: Denne funksjonen tar seg av å starte opp ekstern krystall og vente til denne er klar. Så starte opp PLL med krystall som klokkekilde og vente til denne er klar, Før PLL velges som systemklokke. På denne måten er det oppnådd en klokkefrekvens på 32MHz fra en krystall på 16MHz.
- `init_enkoder1()`: Denne funksjonen setter opp nødvendig registre for eventsystem og teller. To etterfølgende pinner på en port velges som inngang til eventsystemet som gir ut event for disse ut fra at det er A og B-signaler fra en enkoder. Det er og valgt et digitalt filter på 8 klokketikk, som altså krever at A og B-signaler holdes i minimum 8 klokketikk før det kan gis ut event.
- `init_enkoder2()`: Samme som over men for enkoder til arm 2.
- `init_hastighet1()`; Denne funksjonen slår sammen to tellerregistre for å få en 32bit teller. Det blir så valgt at det skal telles ut fra systemklokken og at event fra enkoder 1 skal trigge compare, slik at tellerverdien lagres i buffer.
- `init_hastighet2()`: Samme som over men for enkoder til arm 2.
- `init_usartD0()`: Setter opp nødvendig signaler mot `MAX3221` samt initierer registre for bruk av `RS-232` på hastighet 115200Hz.
- `init_CAN_controller()`: Her blir oppsettet for `MCP2515` gjort. Denne er holdt i reset. Om man ønsker å benytte seg av CAN siden er det nødvendig med justeringer her
- `init_CAN_transceiver()`: Her blir oppsette for `SN65HVD233` gjort. Denne er satt i standby mode.

I tillegg til initfunksjonene er det nødvendig med et sett av interruptrutiner. En forklaring på disse er gitt her.

- `ISR(TCD0_CCA_vect)`: Denne interrupt-funksjonen behandler interrupt trigget på telleren for beregning av hastighet. Her sørges det for å lagre tellerverdiene i buffere før telleren resettes.
- `ISR(TCE0_CCA_vect)`: Samme som over men for vinkelhastighet til arm 2
- `ISR(TCD1_OVF_vect)`: Denne interrupt-funksjonen behandler overflow i telleren for hastighetsestimering. Ved overflow settes tellerbufferen til å være maksimalverdi.
- `ISR(TCE1_OVF_vect)`: Som over men for vinkelhastighet til arm 2.

- `ISR(USARTD0_TXC_vect)`: Denne metoden sørger for å sette et flagg slik at main-metoden får lov til å sende ny byte over `USART/RS-232`.
- `ISR(USARTD0_RXC_vect)`: Denne metoden sørger for å sette et flagg slik at main-metoden får lov til å ta imot ny byte som kommer via IRDA.

Main-metoden sørger for å kjøre init-funksjonene før de nødvendige interruptnivåene skrus på og global interrupt skrus på. I while-løkken i main-metoden er det sørget for å sende data over RS-232.
While-løkka i main-metoden vist i kodebit 7.1 sørger for å sende vinkelposisjon og vinkelhastighet til alle armene i en bestemt rekkefølge. Det er først når interruptrutinen `ISR(USARTD0_TXC_vect)` setter et flagg at data flyttes over i sendebufferet. Først sendes vinkelverdi så hastighet. Dette gjøres for alle tre armene før syklusen gjentas. For å sende vinkelposisjonen er det bare å lese av telleren som er avsatt til dette. Siden denne er 16bit må disse leses av i en atomisk operasjon hvor man så først sender de 8 MSB og så de 8 LSB fra denne leseoperasjonen. Grunne til at man ungår datakorrupsjon når man bruker 16bit med en 8bit mikrokontroller er at det er støtte for "atomisk" lesing av 16bit tellere på spesifikke minneområder på XMEGA.
For å sende tiden mellom to vinkelposisjoner som benyttes for hastighetsestimering kreves det noe mer kode siden det er brukt to tellere på 16 bit for å beregne tiden mellom to vinkelposisjoner. Disse 32 bitene kan kun leses av i to atomiske operasjoner. I verste fall risikerer man å få en interrupt midt mellom lesingen av de to tellerene som oppdaterer verdiene. Dette vil føre til inkonsistens i verdien man leser ut. Den minst resurskrevende måten å ungå dette problemet på er å skru av global interrupt under lesing av disse 32 bitene og så skru på interrupt igjen når lesingen er ferdig. Hvis det skulle skje at det kommer en interrupt fra telleren under lesing vil den da behandles umiddelbart når global interrupt skrus på igjen.
For å få til et glattere hastighetsestimat ved avtagende hastighet benyttes både den faktiske tellerverdien og verdien gitt fra siste capture-interrupt. De 16 MSB fra disse samenlignes. Hvis tellerverdien er høyere enn siste capture-verdi vil tellerverdien sendes. Hvis ikke vil capture-verdien sendes. Siden det bare er de 16MSB som sammenlignes må det ha gått maksimum en tid tilsvarende $1/(32000000/2^{16}) \approx 0.002048$ etter at tellerverdien er blitt høyere enn capture-verdien før tellerverdien sendes.
Et problem med den siste løsningen er at man egentlig skulle ha stoppet telleren når man leser av verdien for å ungå datakorrupsjon. Dette var ønskelig å gjøre ved å trigge en programvaregenerert capture fra eventsystemet. Men det er ikke funnet en måte for å få til å trigge forskjellige capture fra forskjellige kilder.

Listing 7.1: While-løkke på mikrokontroller

```
1 for(;;) {
2   if(flag == 1){
3     if(i==0){//sender vinkel_0 msb
4       buffer_vinkel_1= HEADER_0<<8 | (0x3fff & TCC0.CNT);
5       USARTD0.DATA = buffer_vinkel_1>>8;
6     }
7     else if(i==1){//sender vinkel_0 lsb
```

```
8         USARTD0.DATA = buffer_vinkel_1;
9       }
10      else if(i==2){//sende tid_0 msb
11        cli();//skrur av global interrupt for å lese av teller
12        buffer_CNT_1_D = TCD1.CNT;
13        buffer_CNT_0_D = TCD0.CNT;
14        sei();//skrur på igjen global interrupt
15         //siste mulighet for oppdatering
16        cli();//skrur av global interrupt for å lese av capture
17        buffer_CCA_1_D = CCA_1_D;
18        buffer_CCA_0_D = CCA_0_D;
19        sei();//skrur på igjen global interrupt
20        if(buffer_CNT_1_D > buffer_CCA_1_D){//sjekker om teller er større enn capture
21          buffer_tid_fart_1[1] = buffer_CNT_0_D;
22          buffer_tid_fart_1[0] = buffer_CNT_1_D;
23        }
24        else{
25          buffer_tid_fart_1[1] = buffer_CCA_0_D;
26          buffer_tid_fart_1[0] = buffer_CCA_1_D;
27        }
28        USARTD0.DATA = buffer_tid_fart_1[0]>>8;
29      }
30      ........................................
31      Sender 24 gjenstående bit av tidsestimat
32      ........................................
33      Samme oppsett for vinkel og tid for arm_2
34      ........................................
35      Videresender data mottatt fra node_2
36      ........................................
37      i++;
38      if(i>17){
39        i=0;
40      }
41      flag = 0;
42    }
43  }
```

Resultatet av implementasjonen som er gjort er at det tilslutt er brukt 6 av de 8 tilgjengelige tellerene. Da er det fortsatt 2 tellere igjen hvor man trenger en av dem for å gi klokkesignal til CAN-controlleren om man ønsker å benytte denne.
For sending av kun vinkelposisjon, det vil si 6 byte, to for hver arm. Blir samplingsintervallet som vist i tabellen 7.1 på neste side. For sending av vinkelposisjon og estimert vinkelhastighet, det vil si 18 byte, to for vinkel og fire for vinkelhastighet for hver arm, er samplingsintervallet som oppgitt i tabellen 7.2 på neste side. Hastigheten 115200bit/s er den maksimalt lovlige hastigheten for `target_pc`. Selv med denne hastigheten er samplingsintervallet lavere enn samplingsintervallet til regulatoren, som er 1ms, om man velger å bruke hardwareestimering av vinkelhastighetene. Dette kan løses ved å gå over til CAN-protokollen.

Baudrate[bit/s]	Sendetid[s]
57600	0.0011
115200	0.000573

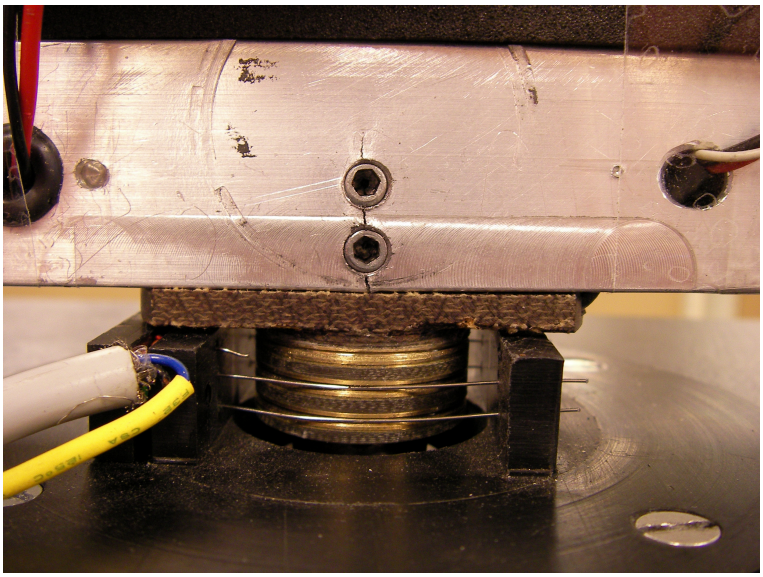
Tabell 7.1: Tabellen viser samplingshastighet for sending av posisjon ved forskjellige baudrater

Baudrate[bit/s]	Sendetid[s]
57600	0.0034
115200	0.0017

Tabell 7.2: Tabellen viser samplingshastighet for sending av posisjon og hastighet ved forskjellige baudrater

7.3 Pendelsystem

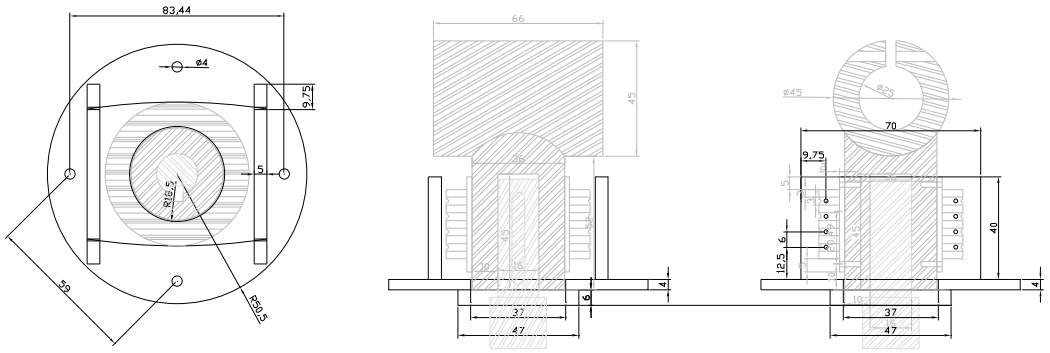
Det eksisterende pendelsystemet ble bygget ved prosjektets oppstart. Det er noe utslitt grunnet både uheldig behandlig og lagring samt deler av konstruksjonen er underdimensjonert. Festeanordningen som er laget for motorakslingen er ikke tilfredsstillende. Den griper kun om halvparten av motorakslingens lengde for å kunne gi rom for sleperingene. Belastningen som denne konstruksjonen har ført til på aluminiumet har resultert i metalltretthet rundt festeskruene noe som kan ses som sprekkdannelse på bilde 7.5. Det har derfor vært nødvendig å designe nytt feste.
Feste for pendelarmene er som nevnt i [14] slarkete. Her er det en kobinasjon av feil bruk av kulelager samt underdimensjonert konstruksjon som fører til slark. Det har derfor også her vært nødvendig å designe nytt feste.



Figur 7.5: Bildet viser resultatet av metalltrettheten rundt festeskruene på gamle **arm_0**

7.3.1 Hub

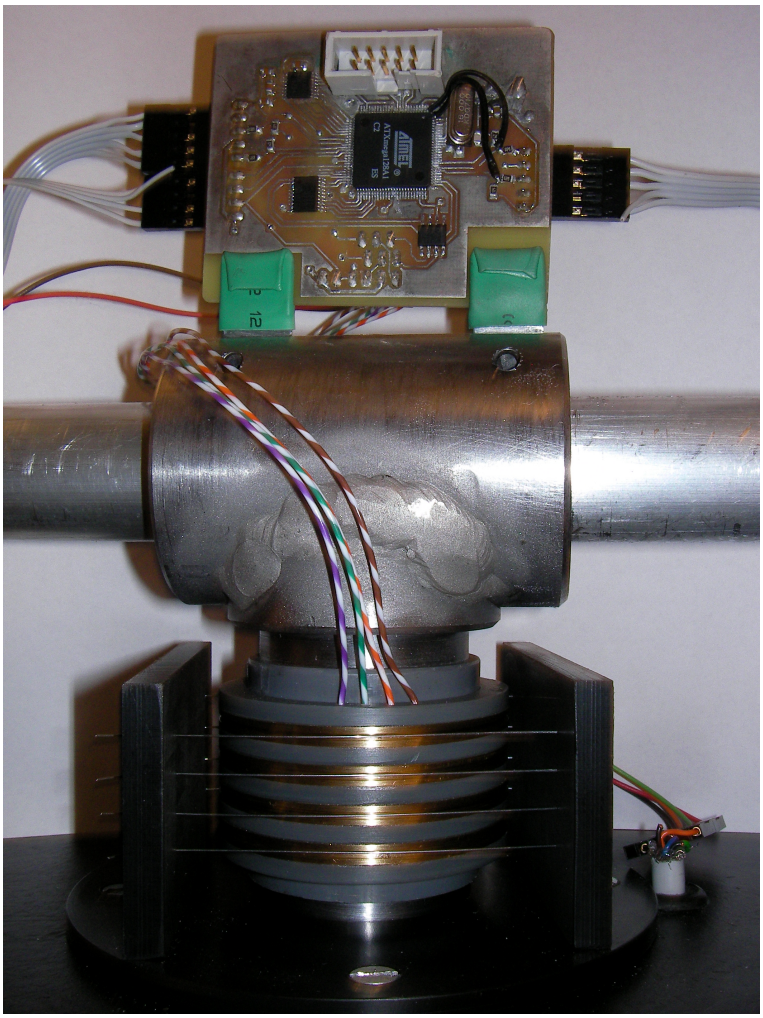
Det nye festet mellom motor og den horisontale armen er laget som en T. Hele motorakslingens lengde blir nå brukt, og innfestingen er byttet fra to settskruer til feste med kile som er den beregnede måten å gjøre det på ut fra motorkonstruksjonen. Figur 7.6 viser CAD-tegningene av den nye konstruksjonen, mens Bilde 7.7 på neste side viser de ferdigproduserte delene.



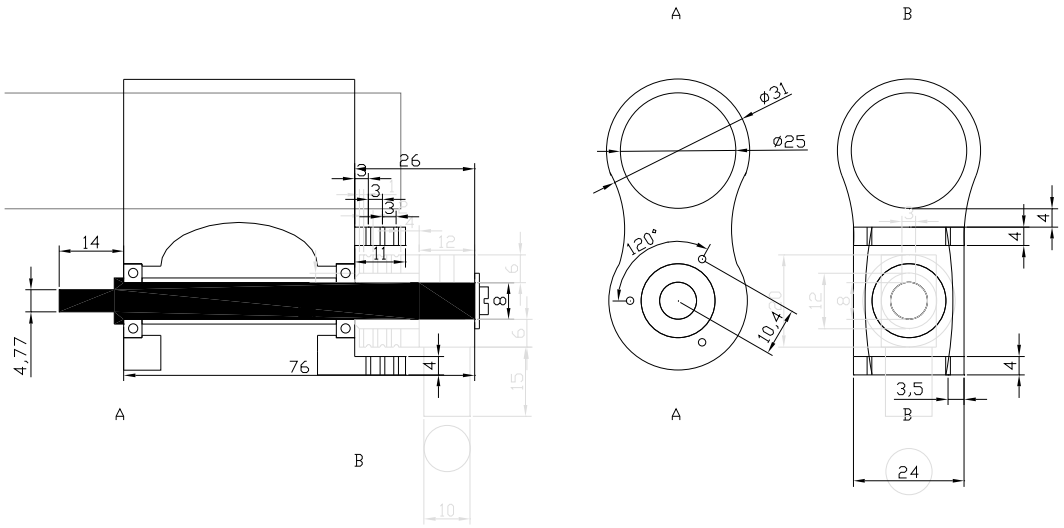
Figur 7.6: Figuren viser CAD-tegningen av pendelhub, sleperinger og slepetrådholder

7.3.2 Pendeloppheng

Pendelarmenes oppheng er som nevnt over slarkete. Dette er fordi kulelagrene som er brukt er for tett plassert mot hverandre og noe små, samt at de er feil montert. Akslingen som går gjennom lagrene flyter fritt. Det vil si at det ikke er noe som hindrer at den kan bevege seg frem og tilbake i lagrene bortsett fra friksjon. Dette er uheldig med tanke på at enkoderskiven sitter fast i akslingen og dermed beveger seg sammen med denne inne i enkoderhuset. I verste fall kan dette føre til skade på enkoderskivene. Siden disse løsningene gir mye slark kan det og være noe av opphavet til ulinearitеter i systemet. Disse løsningene på opphenget av pendelsystemet har gjort det nødvendig å konstruere og bygge opp ett helt nytt system. Det er derfor blitt laget nye festeanordninger for `arm_1` og `arm_2`. Fokuset under design av nytt system har vært på minimering av slark og å lage det så robust at det tåler de ekstreme påkjenningene som kan komme om systemet kommer ut av kontroll. Derfor er dimensjonen på både lagrene og akslingene blitt økt. På figur 7.8 på side 58 ser man `CAD`-tegningen av det nye systemet. På den nye konstruksjonen ser man at kulelagrene er montert med en spennhylse mellom seg som kun har kontakt med innerringen på lagrene.



Figur 7.7: Bildet viser hub og slepering. Legg merke til de grønne festet som er laget for kretskort. Bildet viser og det nye kretskortet for **node_0**

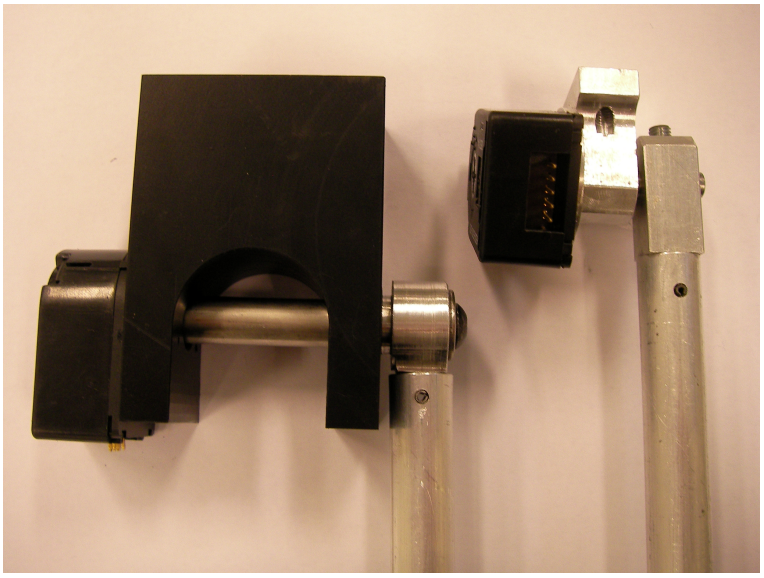


Figur 7.8: Figuren viser CAD-tegningen av pendeloppheng samt sleperingkonstruksjonen

Sammen med strammeskruen som er brukt for å skru fast pendelarmen oppnår man nå at alle elementene i konstruksjonen blir presset mot hverandre uten at det hindrer for fri rotasjon. Denne konstruksjonen er viktig både for å ungå slark i systemet samt fjerne mulighet for skade på enkoderene ved at akslingen ikke kan bevege seg frem og tilbake i lagrene. Bilde 7.9 på neste side viser det nye systemet sammen med det gamle. Man ser her tydelig at dimensjonene er økt.

7.4 Bytte av targetpc

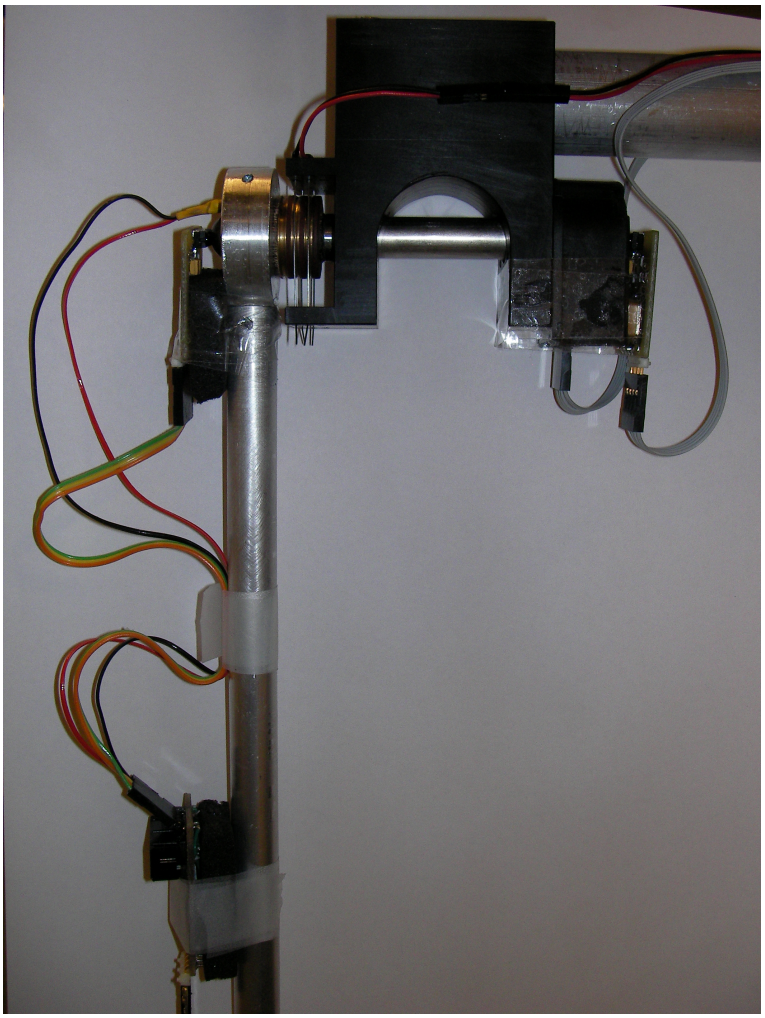
Grunnet `NTNU` sin stadige fornyelse av dataparken er det blitt tatt en leterunde for å finne ny og kraftigere `target_pc` blant gamle pensjonerte maskiner. Kravene for `target_pc` er at den har ISA-buss og ellers har så kraftig prosessor som mulig. Denne leterunden resulterte i at det ble funnet nye pc med prosessor på 800MHz istedenfor 266MHz som var før. Tabellen 7.3 på neste side viser prosessorbruk på gamle og nye pc samt kjøretider på gamle og nye pc. Det går klart frem her at man nå har tilgjengelig mye mer resurser for regulering av pendelsystemet.



Figur 7.9: Bildet viser pendelarmoppheng

prosess/stråd	gammel pc	ny pc
`devc-ser8250`	16%	4.6%
`dev_PCL812`	15%	1.7%
`watchdog`	3%	0.5%
`sinus`(RTW-prosess)	32%	3.7%
totalt for pendelprosesser	66%	10.5%
maks periodetid `Rt_OneStep()`	1.9ms	1.3ms
min periodetid `Rt_OneStep()`	0.3ms	0.7ms
maks utførelsestid `Rt_OneStep()`	0.3ms	0.1ms
min utførelsestid `Rt_OneStep()`	0.1ms	0.1ms

Tabell 7.3: Tabellen viser sammenligning av prosessorbruk og kjøretid på gammel og ny **target_pc**. Periodetiden skal her være 1ms.



Figur 7.10: Bildet viser pendelarmoppheng med slepering samt oppkobling med elektronikk

Kapittel 8

Resultater og diskusjon

8.1 Parameterestimering

Dette kapittelet tar for seg oppsett for parameterestimering på det fysiske sytemet og resultatene som er blitt oppnådd. Alle verdier som er funnet er gjengitt i tabellform i kapittel E på side 101.

8.1.1 Parametre funnet med vekt, kraftmåler og linjal

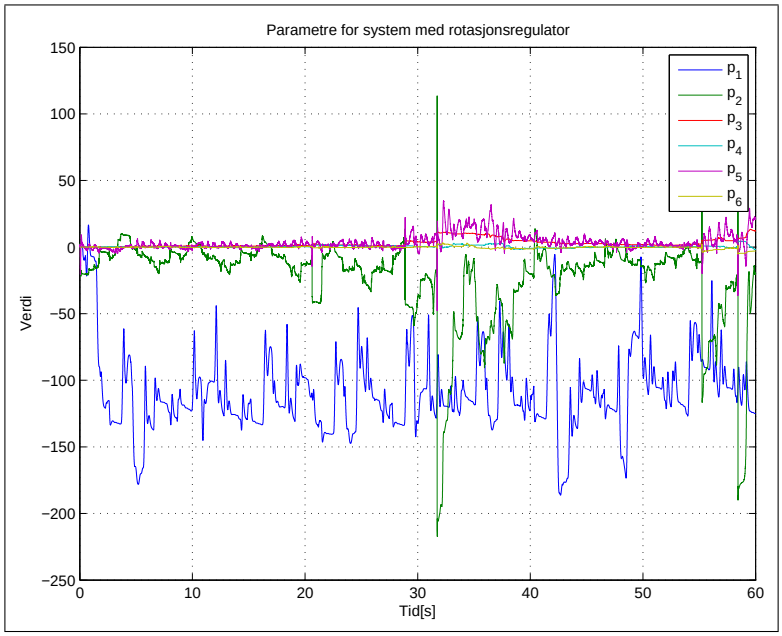
Oppsettet for å måle parametre til systemet avhenger av hvilke parametre man ønsker å finne. For å finnet parametrene til pendelarmer er disse demontert fra systemet før vekt og lengde er målt. Den horisontale armen, `arm_0` er det kun målt lengde på. Vekten på denne er ikke relevant alene men et uttrykk for inertia blir isteden funnet ved parameterestimering.
Friksjonen på systemet er forsøkt målt med kraftmålere. For pendelarmene `arm_1`, `arm_2` uten sleperinger og `arm_3` er friksjonen så liten at den har vært vanskelig å måle. For `arm_0` og `arm_2` med sleperinger var det mulig å måle verdier. Grunnen til at det er gjort måling på `arm_2` både med og uten sleperinger er at det kun er ved dobbelpendel og trippelpendelsystemet at sleperingene blir brukt. Ved bruk av enkelpendel hver side er det benyttet en pendel som ikke tar i bruk eller bli påvirket av sleperinganordningen. Friksjon for `arm_0` er målt til 20g eller $\tau_{\text{statisk friksjon arm\_0}} \approx 0.02[kg] * 9.81[m/s^2] * 0.6[m] \approx 0.12[Nm]$. For `arm_2` er den målt til $\tau_{\text{statisk friksjon arm\_2}} \approx 0.01[kg] * 9.81[m/s^2] * 0.6[m] \approx 0.06[Nm]$
Det er og gjort måling av kreftene som motor gir i forhold til pådraget gitt i `Simulink`. Disse målingene er gjort uten sleperinger for å unngå påvirkning fra disse. Kreftene er målt med en kraftmåler på enden av `arm_0` og er gjengitt i tabell 8.1 på neste side. Disse målingen gir en lineær sammenheng mellom pådrag og krefter. Fra tabellen får man $K_T \approx 0.13 \cdot 9.81 \cdot 0.6 \approx 0.77$, noe som er litt lavere en verdien 0.84 som er oppgitt på motoren. Dette kommer av at pådrag gis fra `Simulink` og går igjennom diverse programvare og maskinvare som kan gjøre feil med skaleringen av signalet.

Pådrag	vekt[g]
1.5	190
3	390
4.5	580
6	770
7.5	970

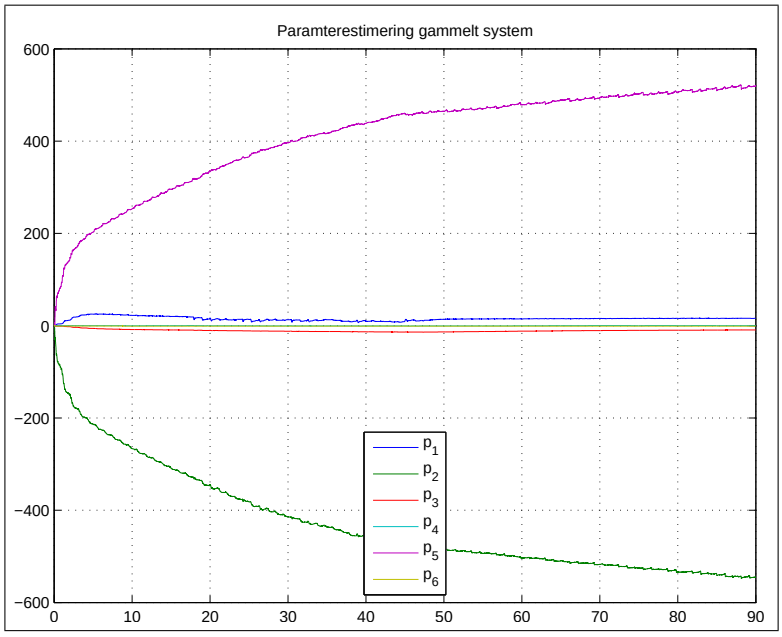
Tabell 8.1: Tabellen viser pådragsverdien gitt i **Simulink** og målt vekt det utgjør på enden av en 0.6m arm

8.1.2 Parametre funnet med ulineært kalmanfilter

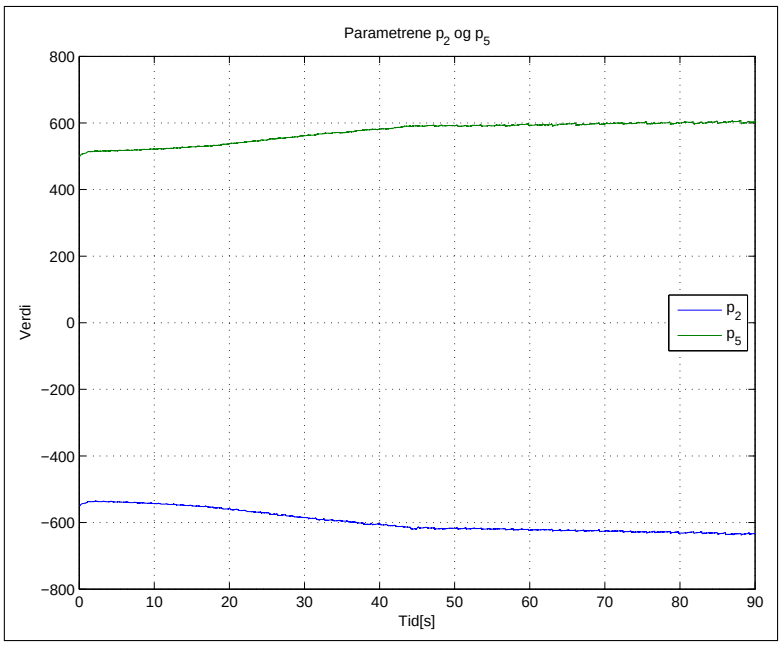
Parameterestimering med ulineært kalmanfilter er gjort på pendelsystemet med kun `arm_0` montert. Sleperingene er ikke montert for å ungå påvirkning fra disse. Innsamling av data er så gjort ved å påtrykke en samling med sinussignaler og utføre måling av responsen til systemet. Samplingsperioden er satt til 0.5ms som er den laveste verdien som er mulig uten at watchdog stopper system. Målingene som tas inn er fra strømmålingen fra `Baldor`-regulatorkortet og vinkelen til `arm_0`. Alle data fra pådrag og måling er lagret for siden å kunne brukes til parameterestimering.
Selve parameterestimeringen kjører offline ved å sende de lagrede pådrag og målinger gjennom det ulineære kalmanfilteret for parameterestimering, se vedlegg G på side 105. Det har vært forsøkt å finne parametre til alle modeller som er kommentert både i kap 4.7 på side 20 og vedlegg F på side 103.
I figur 8.1 på neste side ser man parametrene for testdata hentet inn før rotasjonsregulatoren ble omgått. Av verdien til parametrene ser man tendenser men ikke stabile nok estimat til å kunne konkludere med verdiene til parametrene. Her ser man bakgrunnen for at forsøk i starten ved bruk av subspace identification var mislykkede. Dette er og noe av bakgrunnen for utviklingen som er gjort i kapittel 6 på side 35. Etter at rotasjonssløyfen ble omgått er det kun strømsløyfelikning (4.87) som er benyttet i modellen (F.1). Parameterestimeringen er gjort både på det gamle og det nye fysiske systemet for å kunne sammenligne om inertia til det nye systemet er høyere eller lavere enn på det gamle. Figur 8.2 på neste side viser utviklingen for alle de seks parametrene under simulering av data samlet inn i løpet av 90 sekunder på det gamle systemet. Parametrene p_2 og p_5 ser man her svinge flott inn mot sine tilhørende verdier. På figur 8.3 på side 64 ser man parametrene p_2 og p_5 sin utvikling etter ny simulering med de samme 90 sekunder med testdata. Her er derimot startverdier på simuleringen satt til å være sluttverdiene fra forrige simulering. Flere tilsvarende forsøk hvor sluttverdiene er brukt som startverdier resulterer i $p_2 \approx -600$ og $p_5 \approx 600$. Figur 8.4 på side 65 viser parameteren p_1 sin utvikling etter at sluttverdi er brukt som startverdi. Her derimot ser man et tydelig avvik som skjer i det 17. sekundet. Dette avviket kommer av at armen fysisk er blitt forsøkt holdt innenfor et spesifikt vinkelområde i et forsøk på å ungå ulinearitetet. Ut fra hva man ser i figuren så fungerte ikke dette forsøket, men det verifiserer derimot at strømsløyfen er delvis dekoblet fra rotasjonssløyfen siden dette avviket ikke er synlig på parametrene p_2 og p_5 som gjelder for strømsløyfen. Hvis man så ser på figur 8.5 på side 65 ser man



Figur 8.1: Parametre for gammelt system da rotasjonsregulator fortsatt var i bruk



Figur 8.2: Figuren viser innsvingningen for alle parametrene til det gamle systemet



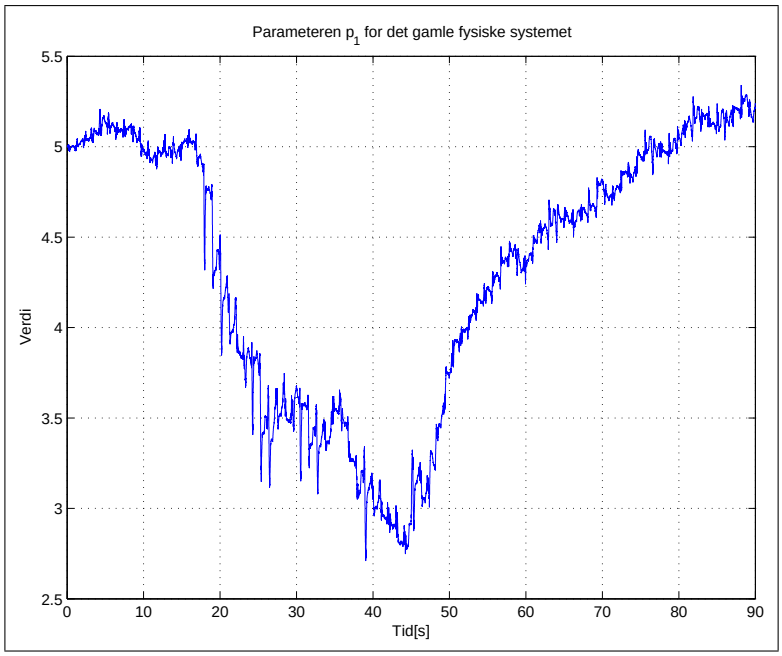
Figur 8.3: Figuren viser simulering nummer 2 for å finne parametre for p_2 og p_5

her også at parametrene p_3 og p_4 som er koblingen fra rotasjonssløyfen mot strømsløyfen prøver å kompensere for disse avvikene. Under parameterestimering med forskjellige data sett er det kun parametrene p_1, p_2 og p_5 som finner tilbake til tilærmet samme verdier hver gang. De resterende parametrene kan man ikke anta har funnet riktige verdier fordi de kompenserer for ulinearitеter i systemet. Disse ulinearitetene ser man tydelig på figur 8.8 på side 68 hvor man kan se at vinkelen til `arm_0` driver betraktelig.

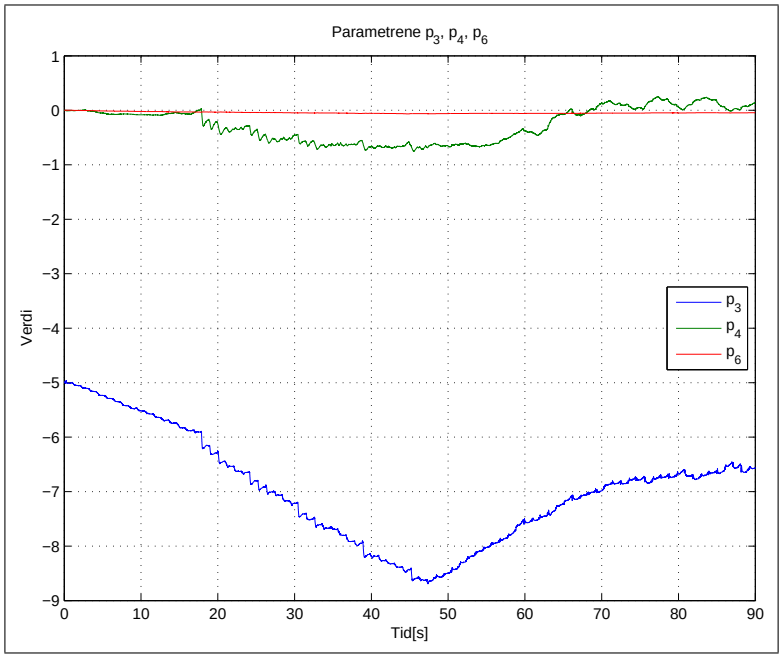
I figur 8.6 på side 66 ser man resultatet av parameterestimering på data hentet fra det nye fysiske systemet. Her legger man merke til at parametrene p_2 og p_5 finner tilbake til tilnærmet samme verdier som ble oppnådd fra parameterestimering på gamle fysiske system, noe som verifiserer at strømsløyfen ikke blir nevneverdi påvirket av forskjellig masse som monteres på motorakslingen. Parameteren p_1 har derimot fått en annen verdi. Som likningene tilsier så blir denne parameteren påvirket av vekt på motoraksling. Det at verdien har økt fra 5.2 til 7 er bra siden dette innebærer at inertia for `arm_0` er lavere for det nye fysiske sytemet enn for det gamle. Dette ser man av likningen

$$J_{tot} = K_T/p_1 \tag{8.1}$$

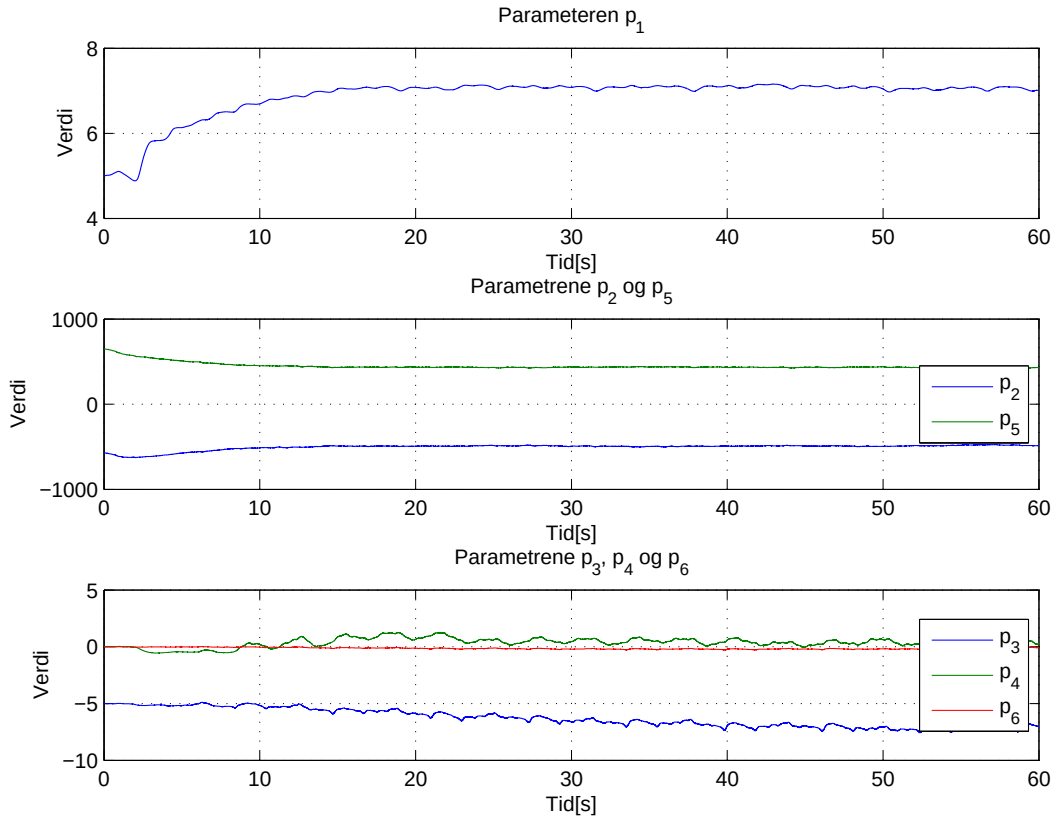
Parametrene p_3 og p_4 ser man også her at de er veldig lave og kun fungerer for å fjerne ulinearitеter i måledata. Figur 8.7 på side 67 viser støммåling fra måledata og strømestimat fra start og slutt av estimeringsprosessen. Her ser man hvordan estimatet svinger inn ettersom parametrene finner riktig verdi. Figur 8.8 på side 68 viser vinkelmåling og estimat fra samme forsøk. Hadde det fysiske systemet vært lineært slik som det er modellert ville ikke vinkelen fått drift slik som plotene viser. Selv om det er problemer med drift ser man tydelig at den estimerte vinkelen svinger seg inn og dermed at parametrene har funnet



Figur 8.4: Figuren viser innsving for parameteren p_1 på det gamle systemet

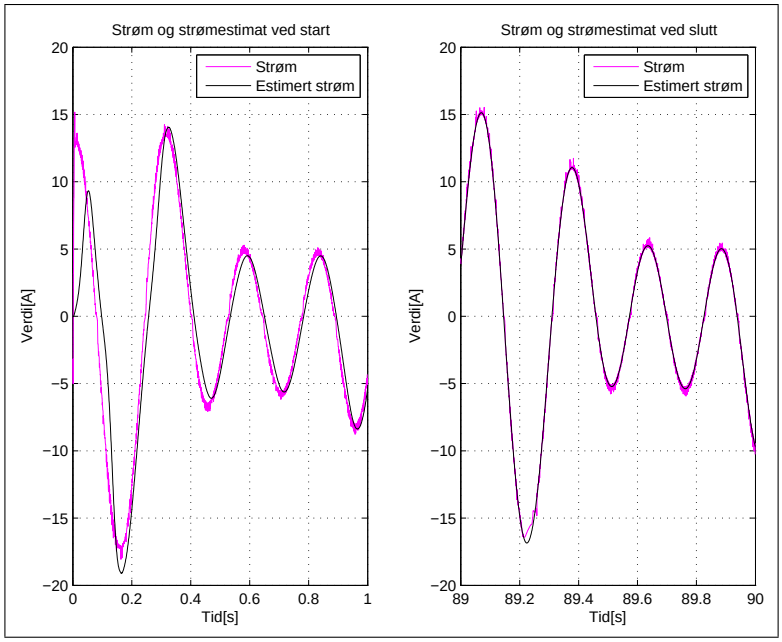


Figur 8.5: Figuren viser innsving for parameteren p_3, p_4 og p_6 på det gamle systemet



Figur 8.6: Parametre for det nye systemet

tilnærmet riktig verdi. Figur 8.9 på neste side viser den estimerte vinkelhastigheten for



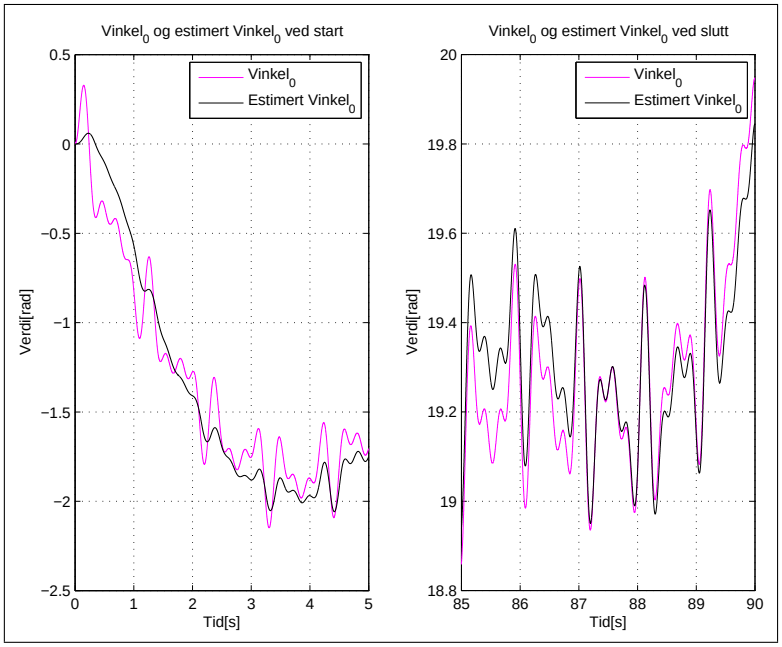
Figur 8.7: Strømestimat under parameterestimering

testdataene som er brukt.

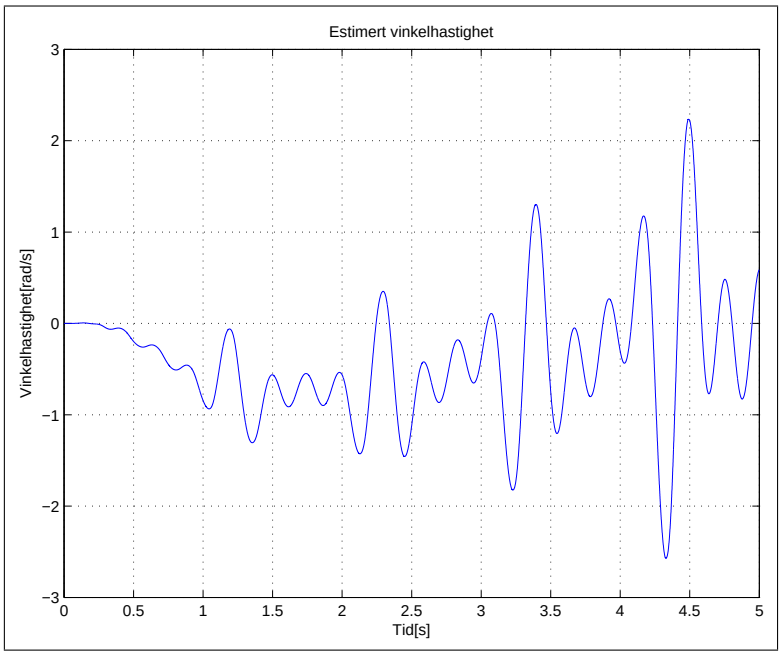
Det skal nevnes at man ikke kan stole helt på parametrene for strømsløyfen. Hvis man så ser på figur 8.10 på side 69 ser man stepresponsen lest direkte fra måling og stepresponsen ut fra et system hvor parametrene fra parameterestimeringen er brukt. Selv om de svinger inn omtrent like fort så ser man at deler av dynamikken ikke er med i det simulerte systemet. Forskjellen i dynamikk er såpass liten at det i praksis ikke har noe å si, verken i simulering eller under kjøring på det fysiske systemet.

8.2 Overføring av data med IRDA fra arm_2 til arm_0

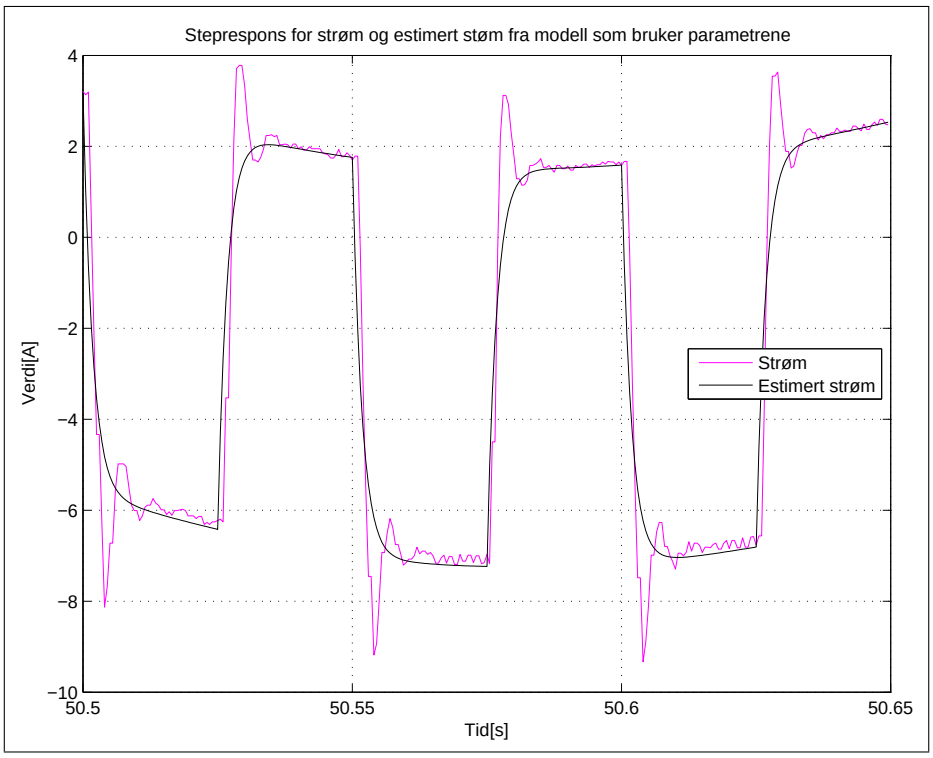
Løsningen med overføring av data med IRDA er vellykket. Til forskjell fra radiooverføringen som var benyttet før er dette en meget stabil og støysikker overføring. IRDA-kortene som er laget har vært prøvd ut ved å stille dem rygg mot rygg slik at senderen sender i stikk motsatt retning av hvor mottaker er. For å få til en overføring holder det å holde en hånd foran sender slik at noe av lyset blir reflektert. Selv om IRDA-standarden spesifiserer at sender og mottaker skal være innenfor visse vinkler og avstander klarer altså transceiverene som er valgt mye bedre enn dette. I figur 8.11 på side 70 ser man plot av vinkel og vinkelhastighet til `arm_2`. Forsøket som er gjort er å rotere armen 8-10 runder for så å rotere tilbake igjen og se om vinkelen ender opp i 0 igjen. Ved radiooverføring fikk man en markant feil på denne testen, mens med IRDA går det mye bedre. Man legger alikevel merke til at det er noe feil, men dette kommer av friksjon i leddene mellom



Figur 8.8: Vinkelestimat under parameterestimering

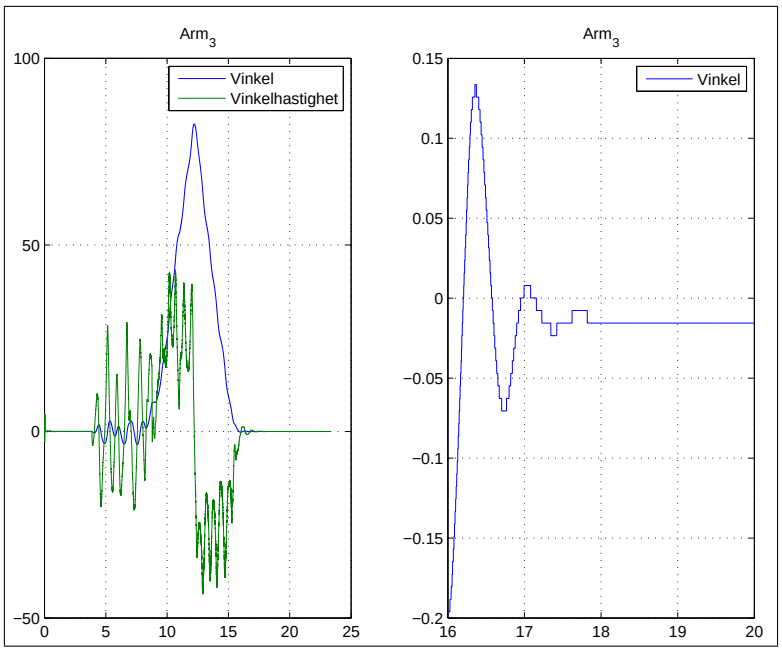


Figur 8.9: Estimert vinkelhastighet på testdata brukt på parameterestimering



Figur 8.10: Figuren viser strømmåling fra systemet sammen med estimert strøm ut fra parametrene som er funnet

armene som gjør at pendelarmene ikke går perfekt tilbake til utgangsposisjonene sine. Resultatene i kapittel 8.5.3 på side 78 viser vellykkede forsøk med oppsving og balansering av dobbelpendel. Dette verifiserer at IRDA-linken er stabil nok også når den benyttes i reguleringssammenheng.



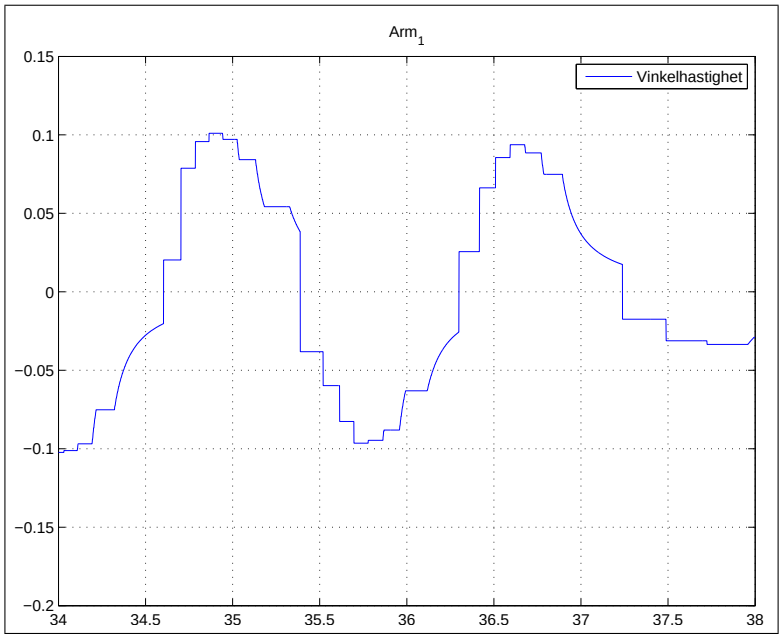
Figur 8.11: Figuren viser vinkel og vinkelhastighet for **arm_3**. Offset kommer av friksjonen i oppheng til **arm_2** og **arm_3**

8.3 Estimering av hastighet i programvare

Estimeringen av hastigheten er forsøkt gjort på flere måter. I software er den estimert ved hjelp av filtrering og derivering av posisjonsdataene, noe som etter noe tuning fungerer tilfredsstillende ved oppsving og balansering av enkelpendel. Estimeringen er og gjort ved kombinasjon av derivering og kalmanfilter om linearisert område hvor pendelen balanserer på høykant. Denne løsningen ga bedre resultater. Men estimeringen ved hjelp av ulineært kalmanfilter har gitt best resultater. Det er først i forsøk ved oppsving og balansering av enkelpendel på hver side at det har blitt problemer. Med den kraftige tilbakekoblingen fra vinkelhastigheten som er nødvendig for å kunne balansere to pendler er det nødvendig med veldig presise tilstandsestimeringer. Siden det ulineære kalmanfilteret lider av noe drift fungerer ikke oppsving og balansering konsekvent. Forsøk og resultater kan ses i de senere kapitlene som omhandler oppsving og balansering.

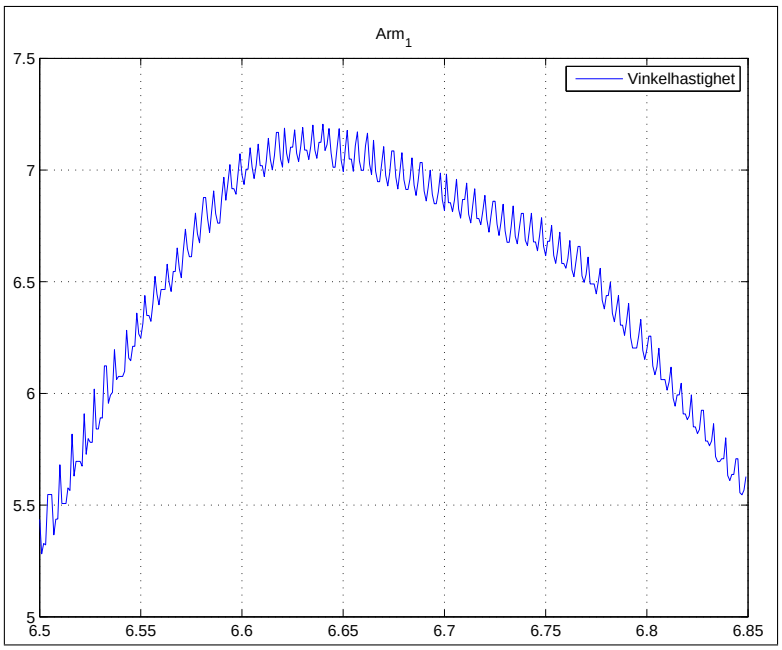
8.4 Estimering av hastighet med hardware

Dette er en måte som i teorien skulle gitt gode estimater, men som i realiteten er avhengig av at hardware og mekanisk system er av god kvalitet. Det er antatt at problemene mye gjelder for periodeestimering, grunnet måten estimatet utføres. Siden frekvensestimering har krevd et høyere hastighetsområde er det ikke denne metoden blitt implementert. Enkle tester som gjøres ved å la pendel fritt dingle frem og tilbake avslører støy som antageligvis er relatert til produksjonsunøyaktigheter på enkoderhjulet, se figur 8.13 på neste side. Støyen som man ser her tilsvarer ca 3% av utslaget noe som stemmer godt overens med hva som er oppgitt i databladet [12] som er gjengitt i tabell 4.3 på side 27.
Figur 8.12 viser estimat av lave rotasjonshastigheter på pendel. Her ser man tydelig at det ved avtagende hastighet er en glatt innsvingning mot null.



Figur 8.12: Figuren viser innsving ved lav vinkelhastighet for **arm_1**

Problemet oppstår derimot når man setter på regulatorsløyfen. Periodeestimering er naturlig følsom for vibrasjoner, det vil si at vibrasjoner i en pendel vil forplante seg som bevegelser på enkoderen og trigge målinger. Kombinasjonen av forsinkelse og frekvensresponsen til regulatoren er slik at vibrasjoner bli forsterket, noe som både ser og høres ille ut. Hadde det derimot vært mulig å bruke frekvensestimering ville man hatt en estimeringsmetode som naturlig demper vibrasjoner. Men denne metoden kan ikke brukes sett at man ikke øker oppløsningen til enkoderene betraktelig og/eller senker samplingstiden.



Figur 8.13: Figuren viser støy ved høy vinkelhastighet for **arm_1**. Årsaken er antageligvis unøyaktigheter på enkoderhjul

8.5 Test av regulering på reelle system

Testforsøkene som er gjort har fulgt en stegvis utvikling. Først har det vært utprøvd å regulere enkelpendel så enkelpendel hver side og til slutt dobbelpendel. Underveis har det ofte kommet ideer om hvordan problemer kan løses. Ideene har da først blitt prøvd ut i simulering, før det har blitt tatt gradvis ut på det fysiske systemet sett at simuleringene har vært vellykket. Dette kapittelet kunne inneholdt veldig mange plot som viser testresultat med forskjellig oppsett. Det er derfor gjort et forsøk på å samle så mye som mulig informasjon i de plottene som er tatt med.
På medfølgende cd er det flere videoer som viser oppsving og balansering av de forskjellige systemene. Det er og tatt med videor som viser problem med vibrasjoner og noen tilfeller hvor oppsving ikke har vært vellykket.

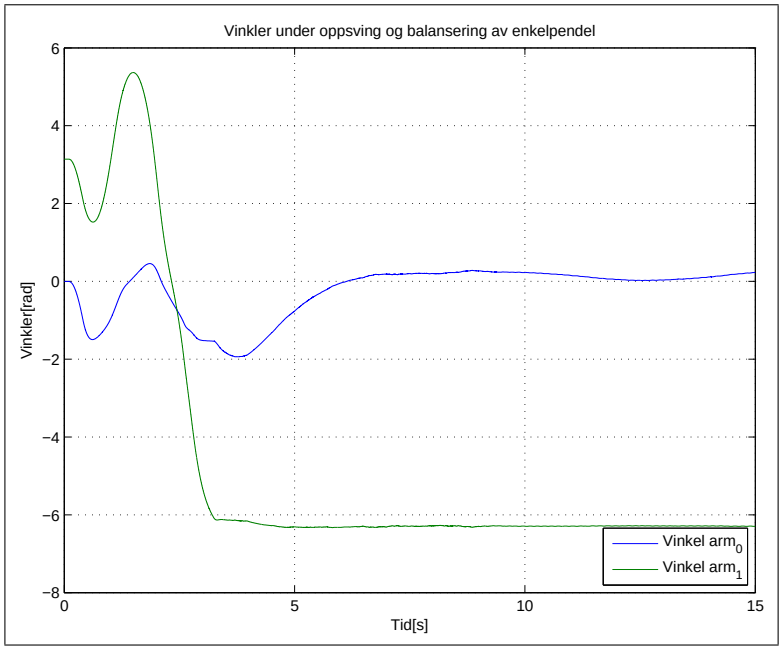
8.5.1 Oppsving og balansering enkel pendel

Oppsving og balansering av enkelpendel har fungert ved bruk av alle de forskjellige vinkelhastighetsestimeringsmetodene. Men noen av metodene har utmerket seg i positiv og negativ retning

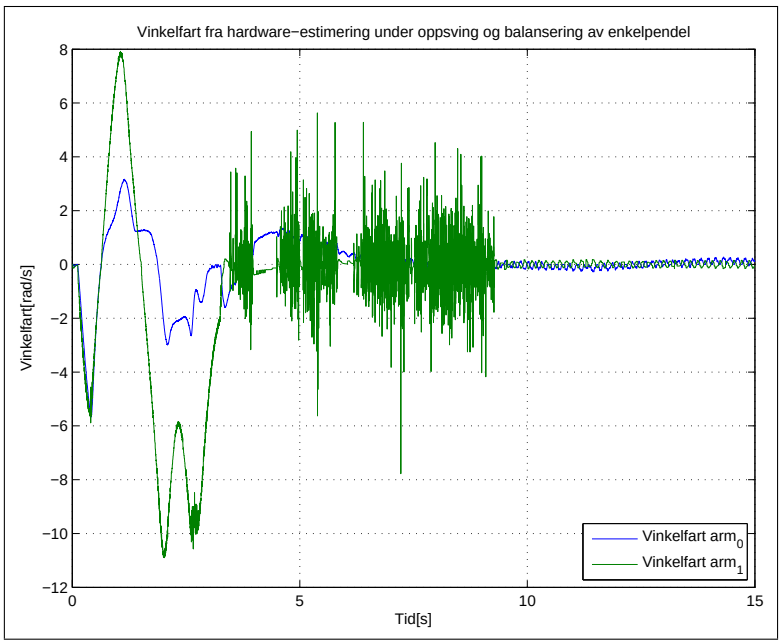
Ved bruk av hardwareestimering

Bruk av hardwareestimering for å estimere vinkelhastighetene har vist seg vanskelig. Pendelarmen som normalt brukes i enkelpendelsytemet er over en meter lang og noe fleksibel.

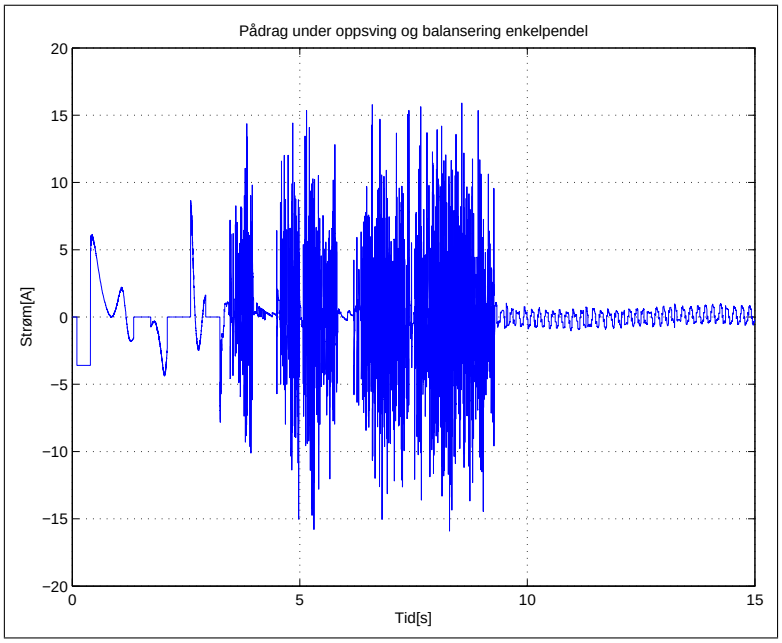
Dette i kombinasjon med hardwareestimering har ført til ukontrollerte vibrasjoner under balansering. Oppsving har til gjengjeld fungert tilfredsstillende. Det er blitt forsøkt tre forskjellige løsninger for å hindre vibrasjoner. Det er forsøkt å sette ett første ordens filter både på pådrag og vinkelhastighetsestimat, uten hell. Det er forsøk med flere forskjellige filterverdier men resultatet har alltid vært at det bringer med seg forsinkelse. Problemet ligger uansett i hvordan periodeestimeringen fungerer ved å måle tid mellom to posisjoner. Man må altså kreve bruk av stivt legeme for å ungå vibrasjoner. Dette har vært prøvd oppnådd på to måter. Fylle pendelrøret med bygningsskum i håp om at dette skulle dempe vibrasjoner og bruk av kortere pendel med høyere resonansfrekvens. Bruk av skum var ikke vellykket, men bruk av kortere pendel ga vellykkede resultater i de fleste tilfeller.
I figur 8.14 ser man vinkelverdiene under oppsving og balansering av kort enkelpendel ved bruk av hardwareestimering. Alt ser bra ut her, men om man derimot ser på vinkelhastighetene i figur 8.15 på neste side ser man at sytemet har funnet resonansfrekvensen noen ganger. Systemet har derimot ikke klart å opprettholde denne høye resonansfrekvensen over lengre tid. Figuren viser altså både vellykket oppsving og balansering samt problemet med hardwareestimering. Figur 8.16 på neste side viser pådrag under samme forsøk. Her ser man at amplituden under vibrasjonene er kraftigere en pådraget som er nødvendig for oppsving. Det puttes altså mye energi inn i systemet.



Figur 8.14: Figuren viser vinkler for **arm_0** og **arm_1**. I dette forsøket er det brukt en kort pendel



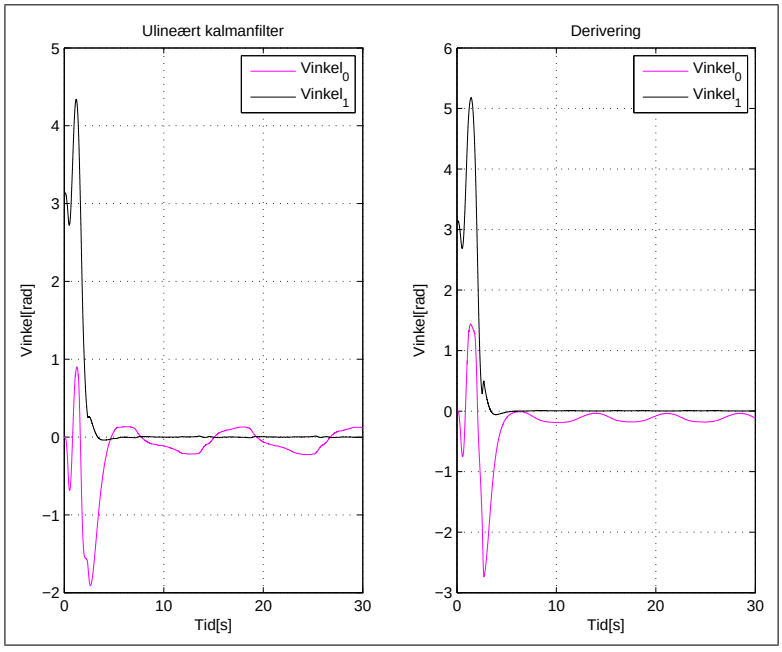
Figur 8.15: Figuren viser vinkelhastighetene for **arm_0** og **arm_1**. I dette forsøket er det brukt en kort pendel. Legg merke til områdene hvor systemet finner resonansfrekvensen til metallet i pendelen



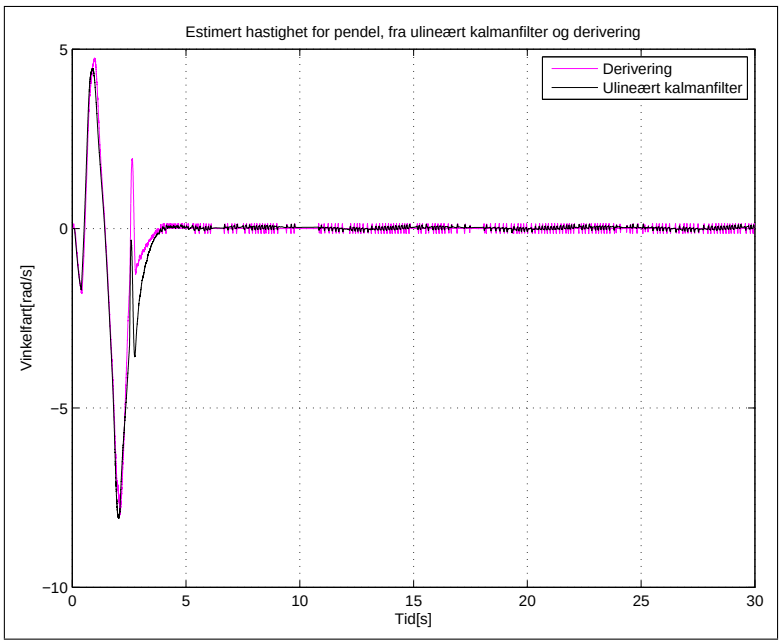
Figur 8.16: Figuren viser strømpådrag under oppsving og balansering. Legg merke til de enorme pådragene som gis på grunn av at vibrasjoner kommer med i tilbakekoblingen

Ved bruk av softwareestimering

Bruk av estimering ved filtrering og derivering av posisjonsdata led under tidlige forsøk av samme problem som ved hardwareestimering. Men problemet ble fjernet ved å justere filterparametre, slik at det nå og kan benyttes lang pendel. På figur 8.17 ser man vinklene for oppsving og balansering. Her legger man merke til at vinkelen for `arm_0` svinger inn ganske bra mot null og det er kun en minimal oscillasjon rundt dette området som er et resultat av at systemet må motvirke friksjonen i sleperingene.
Bruk av ulineært kalmanfilter er metoden som har gitt glatteste pådrag fordi estimatene som tilbakekobles er glatte. I figur 8.17 ser man resultat av oppsving og balansering. Til forskjell fra forsøk hvor filtrering og derivering er brukt, ser man her at vinkel til `arm_0` oscillerer med større amplitude rundt nullverdi. Det er kun når `arm_1` kommer mye ut av likevektspunktet sitt at pådraget blir stort nok til å motvirke friksjon. Først da svinger `arm_0` på seg noe som fører til større oscillering. Dette kunne vært motvirket ved å bruke pådrag som motvirker friksjonen i sleperingene. Grunnen til at dette ikke skjer ved bruk av filtrering og derivering er at det naturlig blir sendt gjennom støy fra estimeringen ut til pådraget som gjør at pådraget hele tiden er nærmere grenseverdien for å motvirke friksjonen i sleperingene. Ved bruk av ulineært kalmanfilter i tilbakekoblingen kan man derimot tillate kraftigere tilbakekbling, noe som ville gitt samme resultat. Hastighetsestimat for begge tilfeller ser man i figur 8.18 på neste side.



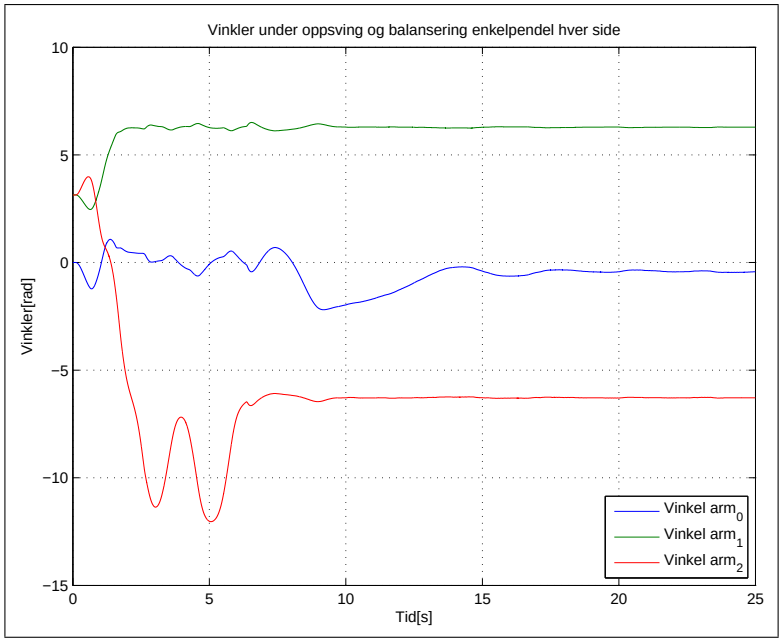
Figur 8.17: Figuren viser vinkler under oppsving og balansering av enkelpendel. Resultater vises både ved bruk av ulineært kalmanfilter og ved bruk av filtrering og derivering



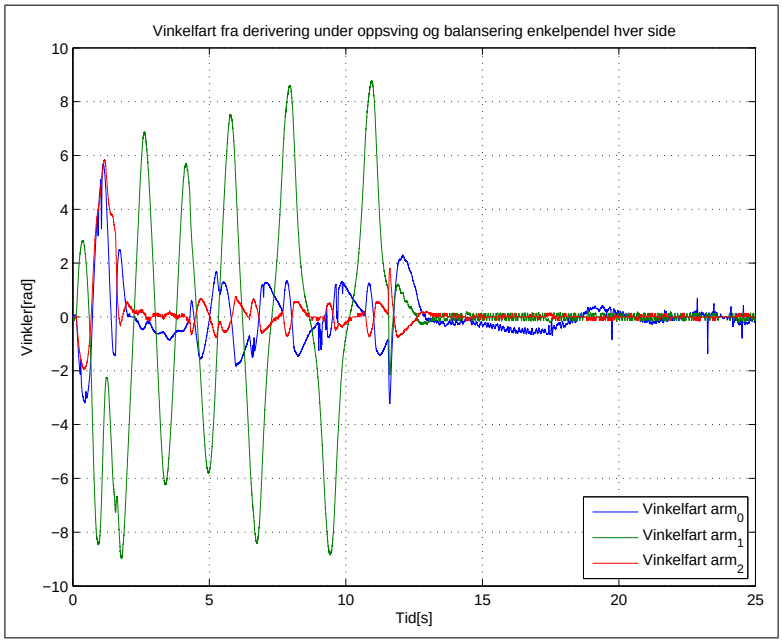
Figur 8.18: Figuren viser vinkelhastighet under oppsving og balansering av enkelpendel. Vinkelhastighetene estimeriert ved bruk av ulineært kalmanfilter og ved bruk av filtrering og derivering. Estimatene er tatt fra samme forsøk

8.5.2 Oppsving og balansering enkel pendel hver side

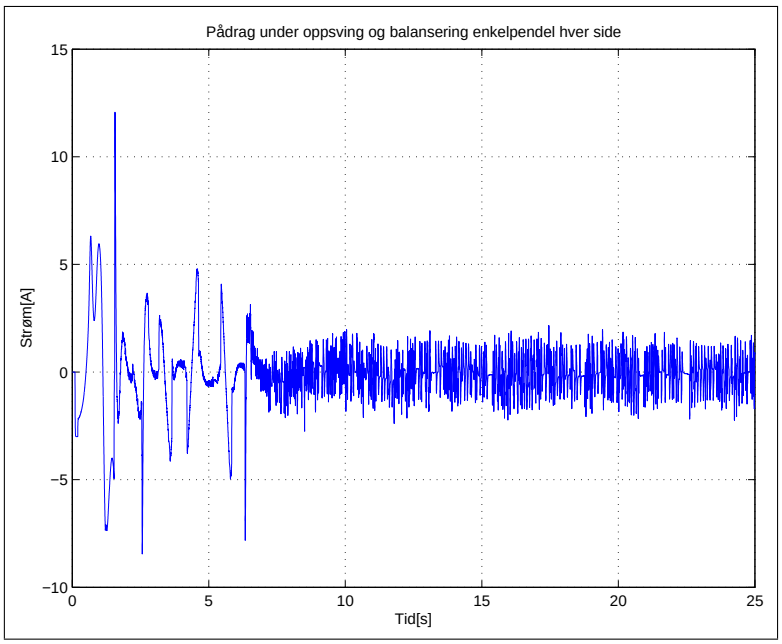
Her er det kun bruk av ulineært kalmanfilter for estimering av vinkelhastighet som har gitt mulighet til oppsving og balansering. Det har vært prøvd å bruke filtrering og derivering av vinkelposisjon, men selv om oppsving har fungert har det ikke vært noen tilfeller med vellykket balansering. Når balanseringsregulatoren slår inn er tilbakekoblingen fra vinkelhastighetene så høye at systemet starter å oscillere. Grunnen til dette er tidsforsinkelsen som filtrering og derivering skaper på hatighetsestimatet.
Det har og blitt forsøkt å bruke hardwareestimering og da med korte pendler. På samme vis har det fungert med oppsving, men i det øyeblikket det byttes til balanseringsregulator blir tilbakekoblingen så stor at systemet får ekstreme vibrasjoner.
Figur 8.5.2 på neste side viser vinkelverdier under vellykket oppsving og balansering ved bruk av ulineært kalmanfilter for estimering av vinkelhastighet. Oppsving og balansering fungerer her i de fleste tilfeller, men av og til ender filteret opp med en liten drift på hastighetsestimatene. I noen tilfeller så klarer ikke systemet da å balansere pendlene, mens i andre tilfeller resulterer det i konstantavvik på vinkelverdien til `arm_0` noe som er synlig i figuren. Figur 8.5.2 på neste side viser vinkelhastighetene til systemet. Her ser man små vibrasjoner på lave hastigheter som er et resultat av at kalmanfilteret justerer inn vinkelposisjonsestimatet til kvantifisert måling som nevnt i kapittel 5.4.3 på side 33. Forplantningen av disse vibrasjonene ser man videre i figur 8.5.2 på side 78 hvor man ser pådraget.



Figur 8.19: Figuren viser vinklene under oppsving og balansering av enkelpendel hver side



Figur 8.20: Figuren viser de estimerte vinkelhastigheten under oppsving og balansering av enkelpendel hver side



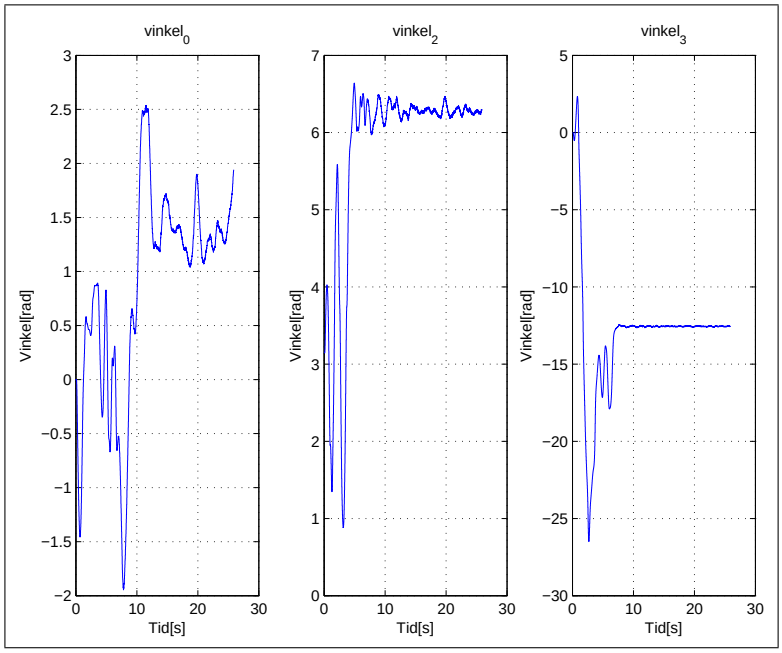
Figur 8.21: Figuren viser strømpådraget under oppsving og balansering av enkelpendel hver side

8.5.3 Oppsving og balansering dobbel pendel

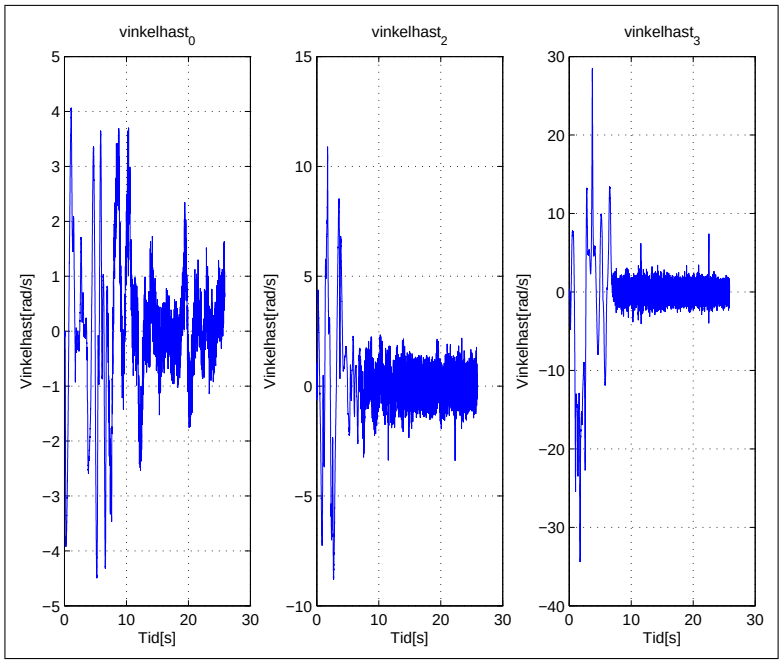
Dette fungerer bra i simulering, men har vært vanskeligere å få til på pendelsystem. Grunnen til dette har vært valg av lengde og vekt på pendlene. Pendelen `arm_2` blir naturlig nok tyngre enn ønskelig når man monterer kretskortet `node_2` og enkoderen. Det er da meget viktig å velge fordelaktige lengder på pendelarmene. Lengdene var valgt en eller annen gang før [14] til å benytte lengre arm for `arm_2` enn for `arm_3`. Lengedene er nå valgt så like hverandre som mulig, og i så lette materialer som mulig.
Det at det er mulig å utføre oppsving og balansering av dobbelpendel viser at overføringen med IRDA fungerer i praksis.

hardwareestimering

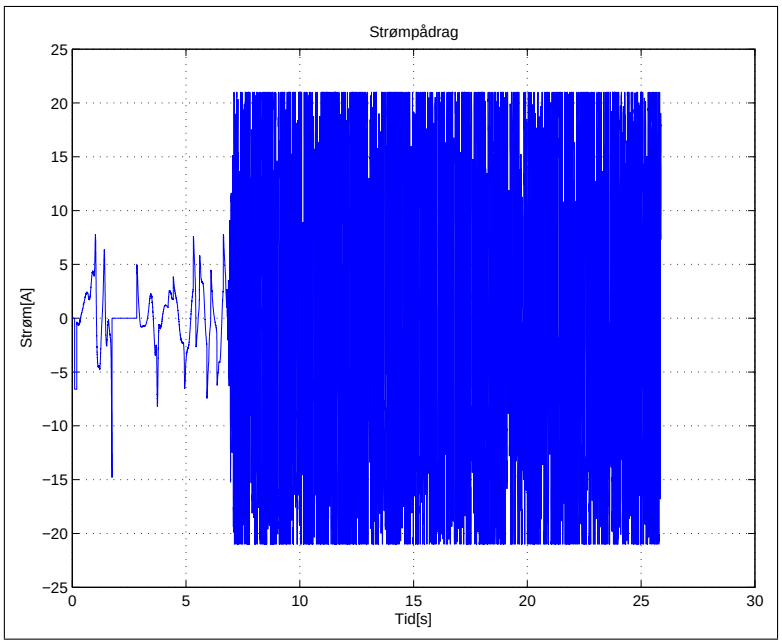
Forsøk på oppsving og balansering ved bruk av hardwareestimering fungerer. I figur 8.5.3 på neste side ser man vinklene til systemet. Selv om pendlene balanserer ser man av vinkelverdien at de er ganske støyfulle. Av figur 8.5.3 på neste side ser man estimerte vinkelhastigheter som er meget støyfulle og som når de tilbakekobles gjennom regulatoren gir pådraget i figur 8.5.3 på side 80. Strømpådraget ser man er normalt sammenlignet med de andre pendelsystemene helt til stabiliseringsregulatoren slår inn. Strømpådraget oscillerer da mellom maksimum og minimum pådrag, noe som fører til enorme vibrasjoner.



Figur 8.22: Figuren viser vinkel for armene til dobbelpendelsystemet ved bruk av hardwareestimering



Figur 8.23: Figuren viser vinkelfart fra hardwareestimering for armene til dobbelpendelsystemet



Figur 8.24: Figuren viser pådrag for dobbelpendelsystemet ved bruk av hardwareestimering

Derivering

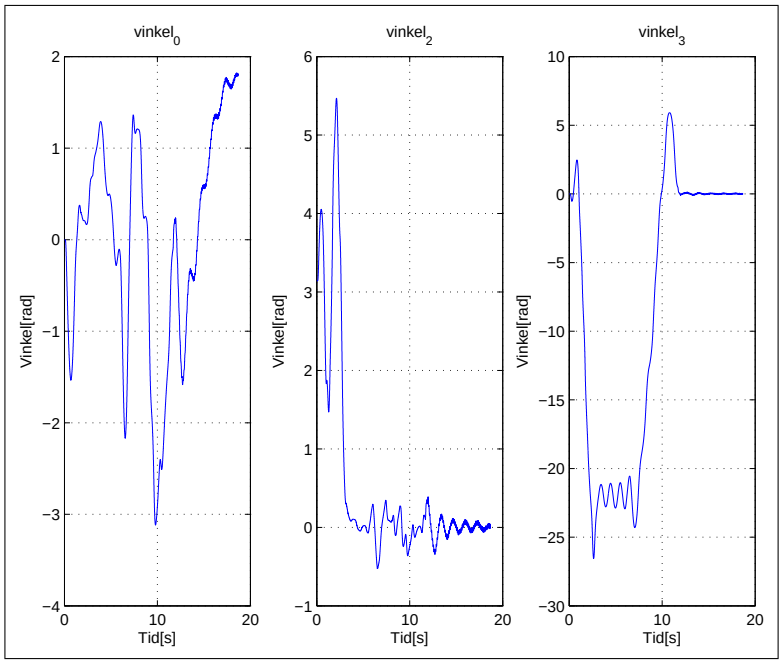
Resultatene fra forsøk gjort med vinkelhastighet estimert fra derivering kan ses i figur 8.5.3 på neste side og 8.5.3 på neste side. Resultatene er tilsvarende de oppnådd fra forsøk gjort med hardwareestimering. Pådraget oscillere mellom metning i begge endepunkter når systemet går over til regulator for å balansere begge pendlene på høykant.

Ulineært kalmanfilter

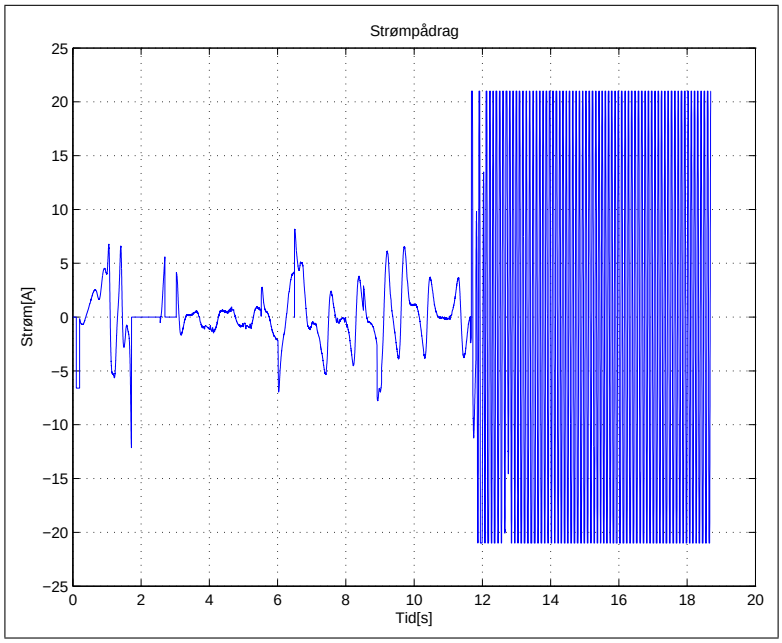
Oppsving går meget fint med vinkelfart estimert fra ulineært kalmanfilter, men i det øyeblikket systemet bytter til stabiliseringsregulator for å holde begge pendlene på høykant gjør en liten drift i vinkelhastighetsestimatet at systemet blir ustabilt. Det er ikke funnet noen løsning på dette problemet i form av justering av parametre for kalmanfilteret.

8.5.4 Trippelpendel

Det er forsøkt implementert ulineært kalmanfilter for trippelpendel systemet. Det er nok datakraft til å kjøre systemet om man øker samplingsperioden fra 1ms til 2ms. Men problemet med implementasjonen er at modellen har singulariteter noe som fører til at estimatene går mot uendelig etter kort tid.



Figur 8.25: Figuren viser vinkel for armene til dobbelpendelsystemet ved bruk av filtrering og derivering for estimat av vinkelhastighetene



Figur 8.26: Figuren viser vinkelfart estimert ved derivering for armene til dobbelpendelsystemet

Kapittel 9

Konklusjon

Regulatorkortet som benyttes til å gi pådrag til motor inneholder både strømregulator og rotasjonsregulator. Selv om det ikke er oppgitt i databladet er det funnet en måte for å omgå rotasjonsregulator, og dermed oppnå strømregulering/momentregulering av motor.
For parametre som ikke kan måles med linjal, vekt eller kraftmåler, er det nå mulig å finne parametrene til systemet ved bruk av ulineært kalmanfilter for parameterestimering. Ved å benytte disse parametrene i en ulineær tilstandsestimator og bruke lave verdier på tilbakekoblingen mellom det fysiske systemet og estimatoren er det så mulig å kontrollere riktigheten til parametrene. Forsøk viser at tilstandene til estimatoren følger de målte tilstandene bra helt til ulinearitetene skaper drift. Denne driften fjernes ved å øke tilbakekoblingen mellom system og estimator og man har dermed en tilstandsestimator som fungerer bra. Dette resultatet verifiserer og både at parametrene som er funnet fra målinger og parameterestimering er tilstrekkelig korrekte og at modellene som er utviklet i [13] stemmer overens med dynamikken til det fysiske systemet.
Regulatorene for oppsving og balansering krever full tilstandstilbakekobling av vinkler og vinkelhastigheter, mens det fysiske systemet kun har måling av vinkler. Det er derfor forsøkt implementert flere forskjellige metoder for tilstandsestimering for å finne vinkelhastigheter. I programvare er estimering i hovedsak forsøkt med filtrering og derivering av vinkelverdi og ved bruk av ulineært kalmanfilter. Ved bruk av mikrokontroller er estimering gjort ved å benytte en høyoppløselig teller for å måle tiden mellom to vinkelposisjoner, kalt hardwareestimering. Denne siste løsningen er implementert på mikrokontrolleren `XMEGA` fra `Atmel` som ved innleverig for denne oppgaven har tre måneder igjen av utvikling før den skal ut på markedet. Selv om det har gitt ekstra utfordringer å jobbe med et produkt som ikke er ute på markede, har det ved bruk av nye funksjonaliteter på mikrokontrolleren vært mulig å implementere vinkelhastighetsestimering på en forholdsvis enkel måte. Dette arbeidet har og resultert i at det er laget nye kretskort for `node_0` og `node_2`.
Det er blitt laget nye kretskort for å ta seg av overføringen av data fra `node_2` til `node_0`. Overføringen baserer seg nå på bruk av IRDA istedenfor bruk av radiolink. Løsningen er robust og fungerer uten problem under regulering av det fysiske systemet
Regulator for oppsving og balansering er skrevet om for å kunne brukes på fysisk system. Det er derfor laget overlapp i byttet mellom de forskjellige regulatorene som benyttes. På

denne måten ungår man nå oscillering i strømpådraget under bytte fra en regulator til en annen.
Både regulator og ulineært kalmanfilter er implementert i `level 2 c-file s-function`. Dette har sikret raske kjøretider selv ved tilstandsestimering av 6 tilstander i et ulineært system. Det er dermed oppnådd en periodetiden ved kjøring på det fysiske systemet på 1ms.
For å sikre at systemet ikke skal kunne komme ut av kontroll er det impementert en overvåkningsmekanisme i form av watchdog. Denne er implementert med høyeste prioritet i forhold til resten av prosessene som er nødvendig for å regulere det fysiske systemet. Denne watchdoggen overvåker at koden generert fra `RTW`, som er generert fra simulinkdiagrammene som representerer de forskjellige systemoppsettene, overholder sin periodetid på 1ms og at vinkelverdien for `arm_0` ikke forandrer seg for mye innenfor et kort tidsintervall. Etter at denne løsningen ble implementert har ikke systemet lenger kommet ut av kontroll under forsøk som er gjort.
Det er oppnådd oppsving og balansering av enkelpendel, enkelpendle hver side og dobbelpendel på det fysiske systemet. Ved oppsving og balansering av enkelpendel fungerer alle metoder for estimering av vinkelhastighet. Men om man benytter hardwareestimering må man benytte en kortere pendel i forhold til de andre metodene. Dette er fordi denne estimeringsformen har en innebygd følsomhet for vibrasjoner. Kombinasjonen av forsinkelse i estimatet og at det allerede er noe forsinkelse i dataoverføring og i regulator gir en totalforsinkelse som er stor nok til å forsterke og øke vibrasjoner i metallet i pendelen. Ved å benytte kortere pendel får man en høyere resonansfrekvens på pendelen noe som gjør at systemet ikke klarer å forsterke vibrasjoner.
Oppsving og balansering av enkelpendel hver side er kun mulig ved bruk av ulineært kalmanfilter for tilstandsestimering. Det er noe problemer med drift i estimatene som gjør at balansering ikke alltid fungerer fordi høye verdiene i tilbakekoblingen for balanseringsregulatoren forutsetter at tilstandsestimatene er tilnærmet perfekte. Det har vært forsøk å benytte hardwareestimering sammen med kortere pendler, men vibrasjonene har selv da blitt for store. Ved bruk av filtrering og derivering for å estimere vinkelhastighetene har man samme problem som ved bruk av hardwareestimering.
Oppsving og balansering av dobbelpendel fungerer om vinkelhastighetene estimeres ved bruk av hardwareestimering eller ved bruk av filtrering og derivering. Om man benytter ulineært kalmanfilter får man for stor drift i estimatet som da fører til at balanseringen ikke blir vellykket. De to andre estimeringsmetodene fungere ikke bra nok selv om det fungere å svinge opp og balansere. Under balansering oscillere pådragsverdien mellom maksimum og minimum noe som skaper store vibrasjoner i systemet, selv om det altså ikke blir ustabilt.
Med disse problemene nevnt over både for enkelpendel hver side og for dobbelpendel har det derfor ikke vært forsøkt med oppsving og balansering av trippelpendelsystemet. Det skal sies at regulatorene for dette systemet heller ikke er utviklet ferdig og at systemmodellen som er laget inneholder singulariteter. Man må altså først få til robust oppsving og balansering av enkelpendel hver side og dobbelpendel samt få til oppsving og balansering av trippelpendelsystemet i simulering før det er mulig å få dette til å fungere på det fysiske systemet.
Arbeidet som er gjort har godt på vei verifisert arbeidet fra [13] ved fysiske forsøk. Alle

elementer fra "videre arbeid" i [14] og [13] er og blitt utført i diplomperioden. Og alt dette har ført til at man nå kan utføre oppsving og balansering av enkelpendel, enkelpendel hver side og dobbelpendel på det fysiske systemet. Det gjenstår derimot en god del arbeid før det er mulig å få til oppsving og balansering av trippelpendelsystemet.
Resultatene fra denne diplomoppgaven viser at det kan være nødvendig å gjøre en revurdering av valg av regulatorer. Bruk av balanseringsregulatorer som benytter en linearisert modell rundt toppunktene for pendelarmene, og hvor regulatorparametrene er regnet med `lqr`-funksjonen i `Matlab` krever kraftige tilbakekoblinger fra vinkel og vinkelhastighet. Det kan hende at en løsning ved bruk av andre mer avanserte regulatorstrukturer som H_∞, backstepping eller MPC kan være fordelaktig.
Arbeidet gjort med implementasjonen av ulineært kalmanfilter for parameterestimering er gjort på en slik måte at det er lett å tilpasse til nye systemmodeller. Det er derfor mulig å bruke dette på andre fremtidige prosjektoppgaver og diplomoppgaver. Dette gjelder og arbeidet som er gjort i forbindelse med hardwareestimering av vinkelhastighet.

Kapittel 10

Videre arbeid

Det er fortsatt mye arbeid som kan gjøres på pendelsystemet. Videre arbeid krever nesten at man velger en retning for hvordan man vil gå videre. Man kan enten velge å bruke regulatorstrukturen fra [13] vider eller utvikle nye. Det er også uttrykt et ønske fra `Atmel` om å gjøre forsøk på å svinge opp og balansere enkelpendel, i første omgang, ved at regulatoralgoritmen kjører på deres mikrokontrollere. Under følger en liste for videre arbeid som kan gjøres.

- Redesign av regulator
  - Forsøke bruk av ulineær balanseringsregulator. Teorien viser for eksempel at bruk av `sliding mode`-regulator fungerer godt selv om man har umodellerte tilstander i systemet. Dette kan være en fordel med tanke på at det fysiske pendelsystemet ikke er et stivt legeme.
  - Det kan og forsøkes å benytte mer avanserte multivariable regulatorer som ikke gjør tilbakekobling fra vinkelhastighetene. H_∞ regulatoren fungere godt selv om systemet har umodellerte tilstander slik som med `sliding mode`-regulatoren.
  - Bruk av MPC regulator kunne være en interessant ting å prøve. Men med denne trenger man mye datakraft for å kunne kjøre med høy samplingsfrekvens. Et alternativ kunne være å gjøre en form for gain-scheduling hvor verdiene regnes ut på forhånd for alle mulige kominasjoner av vinkler og vinkelhastigheter. Regulatoren kunne da gjøre tabelloppslag istede.
- Forbedring av estimeringsmetoder for å kunne benytte tilgjengelige regulatorer
  - Det har vært forsøkt å tilbakekoble vinkelhastighetsestimat fra hardwareestimering inn i det ulineære kalmanfilteret. Dette har ikke vært vellykket grunnet vibrasjoner, men man kan forsøke å augmentere med en modell for vibrasjonene i pendelene. Det finnes og nyere teori for ulineært kalmanfilter, det såkalte `Unscented Kalman Filter for Nonlinear Estimation`, som vistnok skal fungere bedre enn versjonen brukt i denne oppgaven.

  - Når det gjelder estimeringsmetoden som benytter filtrering og derivering kunne det tenkes at om man implementerer et mer avansert filter med høyere grad og brattere knekkfrekvens kunne man oppnå bedre resultater.
  - Et annet alternativ kan være å bruke en mindre resurskrevende tilstandsestimator, som `high-gain-observer`, og gjøre en full tilstandstilbakekobling fra vinkelmålinger og vinkelhastighet fra hardwareestimering.

- Mulige forbedringer i hardware
  - En økning av oppløsningen til enkoderene fra 500 til 4000 kan føre til at man får gode nok estimat ved bruk av for eksempel frekvensestimering. Men dette krever at man går over til enkodere med annen fysisk størrelse som igjen krever redesign av mekanikken. Om man ønsker å utforske muligheten bør man titte på enkoderene fra `Avago`.
  - Bytte overføringsprotokoll til CAN kan være fordelaktig for å få mulighet til å øke samplingshastigheten for vinkelverdiene. Dette gjør det og mulig å bytte ut `target_pc` med en nyere modell som da ikke har ISA-buss. Da må man altså benytte CAN og mikrokontroller for å utføre handlingene I/O-kortet `pcl812` gjør i dag.

- Mekanisk
  - Bytte av materiale i armen fra aluminium til for eksempel karbonfiber kan løse problemene med vibrasjoner i systemet. Aluminium har noe mer fleksibilitet enn hva som er ønskelig.

- Oppgaver som kan gjøres i samarbeid med `Atmel`
  - Utvikle nødvendig kode for å gi mulighet for hurtigprogrammvareutvikling med `RTW` for å kunne kjøre systemer designet i `Simulink` på mikrokontrollerene fra `Atmel`
  - Finne regulatorer for oppsving og balansering som er raske nok til å kunne kjøres på mikrokontrollere fra `Atmega`

Kapittel 11

11.1 Referanser

[1] Chi-Tsong Chen. *Linear System Theory and Design.* Oxford, 1999.

[2] Thor I. Fossen. *Marine Control Systems.* Marine Cybernetics, 2002.

[3] Rolf Henriksen. *Stokastiske systemer.* Institutt for teknisk kybernetikk, August 1998.

[4] Rolf Henriksen. *Subspace Identification of Linear Systems.* Institutt for teknisk kybernetikk, February 2001.

[5] Hassan K. Khalil. *Nonlinear Systems.* Prentice Hall, 2002.

[6] Ekrem Misimi. Control of a triple inverted pendulum, July 2001.

[7] Jan Tommy Gravdahl Olav Egeland. *Modeling and Sinulation for Automatic Control.* Marine Cybernetics, June 2003.

[8] Blake Hannaford Pamela Bhatti. Single chip velocity measurement system for incremental optical encoders, June 1997.

[9] R.F. Solutions. *FM Transmitter and Receiver Modules: FM-TX2-XXX and FM-RX2-XXX.* Vedlagt på CD.

[10] Hans Jørgen Svendsen. Instumentering av sirkulært opphengt invertert pendel, Mai 2002.

[11] Agilent Technologies. *Quadrature Decoder/Counter Interface ICs: HCTL-2000, HCTL-2016, HCTL-2020.* Vedlagt på CD.

[12] Agilent Technologies. *Quick assembly two and three channel optical encoders: HEDM-550x/560x, HEDS-550x/554x/560x/564x.* Vedlagt på CD.

[13] Vegard Åstebøl Larssen. Modeling and control of inverted pendulum systems, August 2006.

[14] Øyvind Bjørnson-Langen. Instrumentering og programvaresystem for trippelinvertert sirkulæropphengt pendel, Desember 2006.

11.2 Bibliografi

[15] Advantech. *User's Manual for PCL-812, PCL-812PG*, 2000. Vedlagt på CD.

[16] BALDOR ASR. *Operating and service manual*, 1989. Vedlagt på CD.

[17] Atmel. *ATMEGA128*. Vedlagt på CD.

[18] Atmel. *ATMEGA168*. Vedlagt på CD.

[19] Atmel. *ATxmega128A1*. Vedlagt på CD.

[20] Atmel. *ATxmega128A1 Manual*. Vedlagt på CD.

[21] Rolf Henriksen. Subspace identification of deterministic linear systems using a single short experiment, Februar 2002.

[22] Texas Instruments. *SN65HVD233, 3.3V CAN Transceivers*. Vedlagt på CD.

[23] Texas Instruments. *uA78L00 Series Positive-voltage Regulators*. Vedlagt på CD.

[24] Maxim. *MAX3221, RS-232 Transceiver*. Vedlagt på CD.

[25] Michochip. *MCP2515, Stand-Alone CAN Controller with SPI Interface*. Vedlagt på CD.

[26] Microchip. *MCP2120, Infrared Encoder,Decoder*. Vedlagt på CD.

[27] Microchip. *MCP2551, High Speed CAN Transceiver*. Vedlagt på CD.

[28] National Semiconductor. *LM1117, 800mA Low-Dropout Linear Regulator*. Vedlagt på CD.

[29] Vishay. *TFDU6102, Fast Infrared Transceiver Module*. Vedlagt på CD.

[30] www.qnx.com. *Writing a Resource Manager*.

[31] www.qnx.com. *Documentations*, 2006. http://www.qnx.com/.

Tillegg A

Installasjon og oppsett helt frem til kjøring av system

Dette kapittelet omhandler alt som er nødvendig å gjøre fra man har datamaskiner uten noe installert samt pendelsystemet, til alt er satt opp og klart til kjøring. Dette vil lette arbeidet for personer som ønsker å bruke systemet i fremtiden. Det kan være en fordel å ta en titt i tilsvarende kapittel i [14].

A.1 Installasjon av QNX-kjerne og utviklingsmiljø

På `target-pc` installerer man QNX direkte fra installasjons-cd, og svarer på spørsmålene under installasjonen på en fornuftig måte.
Installasjon av `QNX Momentics` har derimot vist seg noe problematisk på vertsmaskiner som kjører `Windows`. Systemet har fungert frem til man oppdaterer til `SP3`. Etter dette har det skjedd at kompilering slutter å fungere og justeringer man gjør på oppsettet ikke blir lagret. Eneste tilfellet denne installasjonen har fungert er når man gjør alt fortløpende uten å bruke utviklingsmiljøet før man installerer `SP3`.

A.2 Oppsett av filer

Som nevnt i [14] er det nødvendig med ett sett av filer både på `target_pc` og vertsmaskin som gjør det mulig å bygge, kompilere og kjøre `RTW`-kode i `QNX`. På `target_pc` trenger man filene

- `MATLAB_ROOT/rtw/c/qnx`
- `MATLAB_ROOT/rtw/c/tools/qnxtools.mk`
- `MATLAB_ROOT/rtw/c/libsrc`
- `MATLAB_ROOT/rtw/c/src`
- `MATLAB_ROOT/rtw/c/tools`

- MATLAB_ROOT/simulink/include
- MATLAB_ROOT/extern/include

Disse filene må lagres ut fra denne mappestrukturen, som er valgt i qnx_unix.tmf-fila. Som standardverdi vil dette bety at MATLAB_ROOT er satt til å være

- /matlab

På vertsmaskinen er det og nødvendig med noen filer. Disse er

- qnx.tlc
- qnx_main.c
- qnx_unix.tmf

Som må lagres under

- $matlabroot/rtw/c/qnx

som også må inkluderes i Matlab ved å tykke File->Set Path og legge til mappen.
Ved oppstart av diplom var det nødvendig å reinstallere Windows grunnet virus. Det var da ikke lenger mulig å få tak i en Matlab versjon som var gammel nok til at generering av kode med RTW fungerte uten problemer. Problemet som var med de nye versjonene av Matlab var at man ikke fikk mulighet til å velge tcpip i external mode. Får å få frem dette alternativet må man modifisere fila <matlab_root>/toolbox/simulink/simulink/extmode_transports.m hvor man setter inn en tilsvarende elseif for qnx.tlc som for grt.tlc.

A.3 Oppsett av system og hardware

Velg simulinkmodell ut fra hvilke pendelsystem som skal kjøres. Her er det viktig å velge simulinkdiagram som er laget for RTW-kompilering og ikke de som er laget for simulering. Navnsettingen på mappestrukturen skal være selvforklarende her. Det viktig å velge hvilket fysisk system som skal benyttes, hvilken hardware som skal benyttes og om hardwareestimering av vinkelhastighet skal brukes. Om det er ønskelig å ha så høy samplingsrate for vinklene som mulig, se 7.1 på side 55 og 7.2 på side 55, kan man reprogrammere node_0 for kun å sende vinkelverdier. I såfall **må** man bruke softwareestimering av vinkelhastighetene. Kode for begge deler er vedlagt på cd. Det neste som må gjøres er å montere opp systemet med riktige armer, dette gjøres enkelt ved å skru av festeskruer og bytte ut armer. Det er viktig å huske å skru av strømmen når man modifiserer systemet. Både for å umuliggjøre utilsiktet oppstart, og fordi om man ikke gjør det går det strøm ut til kretskortene. Det er da fare for kortsluttninger, spesielt når man tar av eller setter på armer med sleperinger.

A.4 Bygging av kode og Kompilering

Når system og hardware er valgt og man har funnet tilhørende simulinkdiagram kan denne bygges med `Ctrl+B`. Man må huske å velge riktig IP-adresse for `target_pc`. Etter bygging kan den genererte c-koden kompileres på `target_pc`. Enkleste måten å få til dette er ved bruk av nfs-server for å mappe opp filer fra verts-pc som startes opp på `target_pc` med kommandoen

```
fs-nfs2 -t 129.241.154.103:/c/mappe1 /mappe2 &
```

hvor mappe1 er mappen på verts-pc hvor c-filene fra byggingen er, mens mappe2 er mappen på `target_pc` hvor filene skal gjøres tilgjengelig. På `target_pc` går man så til mappen hvor c-filene er mappet opp og kompilering gjøres med

```
make -f filnavn.mk
```

A.5 Kjøring av kode

Om man prøver å kjøre den kompilerte c-koden vil man få feilmelding om at drivere ikke er starte. Start først opp driveren `DEVICEMANAGER_PCL812` og så `WATCHDOG`. Dette kan enten gjøres fra `QNX Momentics` eller direkte på `target_pc`. Driverene kan startes på `target_pc` med

```
./devicemanager_pcl812
```

Mens RTW-koden må ha input argumenter

```
./filnavn -tf inf -w
```

Hvor -tf sier fra at programmet skal kjøre i inf sekunder. -w sier fra at programmet ikke skal starte opp før `Simulink` har koblet til fra host-pc. Det vil si at hvis man vil kjøre pendel uten bruk av host-pc må man starte driverene direkte på `target_pc` og starte opp RTW-koden uten å ta med argumentet -w, og da gjerne med en fastsatt tid. Programmet kan alltid stoppes ved å bruke `Ctrl+C`.

A.5.1 Styre hele sytemet fra verts-pc

Det er mulig å styre hele systemet fra verts-pc om man ønsker det. Eneste forutsetning er at man installerer ssh-klient på `target_pc`, og starter opp nødvendige nettverksprosesser som qconn, nfs og ssh før man kobler til eksternt. ssh installeres ved å bruke pakkehåndtererprogrammet til `QNX` og installere openssh. For å kunne starte ssh-daemon første gang er det nødvendig å kjøre `/etc/openssh/genhostkeys.sh`. Når dette er gjort kan man starte opp `sshd` som trengs for å logge på fra verts-pc. Før man kan gjøre en remote-login må man opprette en ekstra bruker i qnx. Dette gjør man ved å bruke funksjonen

`passwd <nytt brukernavn>` i ett terminalvindu.
Selve tilkoblingen kan man gjøre fra verts-pc ved hjelp av et ssh-program som for eksempel `f-secure`.

Tillegg B

Kjente problemer

B.1 Error in 'modell/VR Sink'

Hele feilkoden er: **Error in 'modell/VR Sink' while executing C MEX S-function 'vrsfunc' (mdlStart), at time 0. MATLAB error message:.**. Dette er en feil man får om man gjør forandringer i init-fila etter å ha åpnet simulink-modellen, og kjører denne før man igjen prøver å kjøre simulering. For å få lov til å kjøre igjen må man enten lukke simulinkdiagrammet før man reåpne og simulerer på nytt. Eller man kan reåpne VR-modellen, siden den er blitt lukket automatisk da man kjørte init-fila, og starte simuleringen derfra.

B.2 CRC checsum mismatch

Om man forandrer noe i simulink-diagrammet etter først å ha kompilert og kjørt det på `target_pc`, får man feilmeldingen `crc checksum-mismatch`. Får å løse dette problemet må man kjøre `make clean -f filnavn.mk`, lukke matlab og slette RTW-filene og mappene som er blitt generert ved forrige build-kall. Så kan man starte `Matlab` på nytt, bygge RTW-kode og kjøre `make`. Dette problemet skjer kun om man forandrer noe i simulink-diagrammer og ikke om man forandrer på *.c-filene som RTW bruker(s-func, qnx_main.c osv). Da trenger man kun å gjøre en rekompilering på `target_pc`.

B.3 Kun siste del av data fra sanntidskjøring blir lagret

Default-bufferen for data som lagres i RTW når man kjører den kompilerte koden er satt til 1000 verdier. Dette vil si at man kun får lagret det siste sekundet med verdier etter kjøring av RTW-koden om samplingsraten er satt til 1ms. For å øke dette bufferet må man justere i `Tools/External Mode Control Panel`, trykke på `Signal and Triggering` og forandre på parameteren `Duration`.

B.4 Data blir ikke sendt riktig, pendlene dingler frem og tilbake

Dette er et meget merkelig problem som ble registrert på det gamle `node_0`-kortet etter at det ble modifisert for bruk av IRDA istedenfor radiolink. Problemet kan observeres under kjøring ved at pendlene dingler frem og tilbake om man bruker modellbasert tilstandsestimering, eller at systemet står stille om man bruker annen form for tilstandsestimering. Under en periode var det mulig å gjenskape feilen og da ble det funnet at problemet kom av at mikrokontrolleren ikke ville sende data over USART0 om den ikke mottok data-/interrupt fra USART1. Mikrokontrolleren ble satt i debug-mode for å kontrollere om interruptflagget for sendefuffer til USART0 ble satt, og det ble det. Det ble og satt break i USART1 sin interrupt-rutinen, og det var først da programmet var innom denne breaken at det begynte å sende over USART0. Interrupt rutinen for USART0 vil alikevel ikke kjøre selv om den har høyere prioritet enn interrupt-rutinene for USART1. Det er forsøkt å slå av hele USART1 men det gir ingen forskjell.
En mulig grunn til at dette ikke har vært problem før kan være enten noen ukjente feil med ny GCC-kompilator som er installert i forbindelse med XMEGA, eller at da radiolinken var brukt så sendte mottakeren hele tiden ut data selv om det ikke ble sendt noe til den. Dette kan ha sørget for å trigge interrupt på USART1. Etter at IRDA ble implementert var den eneste måten å få `node_0` til å sende på, å sørge for trigging av interrupt i USART1 ved å sende noe over IRDA eller så kunne man stryke litt på banen til USART1 på kretskortet med fingrene. Dette siste fører til at det mottas interrupt på USART1, og da begynner USART0 å sende data. En senere total omprogrammering gjorde at problemet forsvant for en periode. Men det er siden registrert noen tilfeller hvor problemet har gjentatt seg. Skulle det skje igjen så er det bare å stryke fingrenen over baksiden av kortet.

B.5 Avvik på strømmåling

Strømmåling får en sakteforandrende konstantavvik i forhold til pådraget i noen tilfeller. Dette er noe man må være oppmerksom på ved parameterestimering. Feilen har vært undersøkt ved å koble inn oscilloskop. Disse målingene viser korrekte resultater fra regulatorkortet, noe som betyr at feilen ligger ett eller annet sted mellom regulatorkortet og `Simulink`. En mulig årsak kan være at kabelen som går mellom `target_pc` og `node_reg` er uskjermet, men dette problemet er ikke utforsket videre. Under normal bruk er ikke strømmålingen relvant, men når man samler inn data for parameterestimering må man forsikre seg om at målingene er korrekte.

Tillegg C

XMEGA

XMEGA er i skrivende stund Atmel sin nyeste mikrokontroller med revolusjonerende forandringer samelignet med ATMEGA mikrokontrolerene. XMEGA har mye flere konfigurasjonsmuligheter og periferienheter enn tidligere kontrollere. Systemklokken kan velges i programvare og kan være opp til 32MHz. Det er støtte for DMA og kryptering. Det er flere tellere, USART, SPI og TWI enn før. De påfølgende kapittlene tar kun med elementer som er viktig for denne diplomen.

C.1 Klokke

Systemklokken kan velges i programvare mellom flere forskjellige interne og eksterne enheter. Det er og mulig å skalere både opp og ned et allerede tilgjengelig klokkesignal. Bytte av systemklokke kan derimot kun skje om kilden er stabil. Dette blir angitt av mikrokontrolleren slik at man selv kan sjekke og bytte når det passer seg.

C.2 Event systemet

Eventsystemet er en revolusjonerende tankegang for behandling av styresignaler mot periferienheter uten bruk av prosessortid. Systemet bygger på 8 multiplexere som man kan sette opp til å sende forhåndsbestemte signaler fra ønsket pereferienhet til en eller flere andre. For eksempel kan man få overflow i en teller til å sende en event til en ny teller som så inkrementerer. På denne måten kan man slå sammen flere tellere for å øke oppløsningen. Det finnes to typer event. Ett signal-event og ett data-event. Disse brukes på forskjellig måte avhengig av mottaker. Det er heller ikke alle mottakere som har mulighet til å benytte seg av data-event. I tilfellet over hvor en annen teller skal inkrementere ved overflow benyttes kun signal-event.
Event-systemet har mulighet for å benytte ett digitalt filter. Dette fungerer på en slik måte at man velger en verdi mellom 1 og 8 som pereferienheten må trigge eventet, før eventet sendes videre til mottakere.
Event-systemet har og støtte for kvadraturteller. Når man benytter seg av denne sier man fra til eventsystemet hvilke to etterfølende pinner A og B-signalet fra enkoderen kommer

inn. Disse signalene kan man så velge å kjøre gjennom det digitale filteret. Kombinasjonen med Schmitt-trigger på port-pinnene og det digitale filteret gir en tisvarende kvadraturteller i XMEGA som kvadraturtellerbrikker `HCTL2000` som er brukt på de gamle kretskortene. Tabellen under viser hvordan en teller oppsatt for kvadraturtelling, reagerer på de forskjellige eventene.

Signal	Data	Data-event bruker	Signal-event bruker
0	0	Ingen event	Ingen event
0	1	Index/Reset	Ingen event
1	0	Tell opp	Signal-event
1	1	Tell ned	Signal-event

C.3 Tellere

Det er tilgjengelig 8 stykker 16-bits tellere i XMEGA. Disse kan styres enten ved direkte skriving til sine tilhørende registre eller ved hjelp av eventsystemet. Tellerene har og mulighet til å gi event eller interrupt som man så kan bruke som man vil.
I kombinasjonen sammen med eventsystemet hvor man ønsker å ha kvadraturteller blir både signal-event og data-event benyttet for å inkrementere, dekrementere eller eventuelt resette telleren. Det er og mulig å sette opp telleren til kun å bruke signal-event for å registrere når det skjer en tilstandstransisjon på en kvadraturteller. Dette kan benyttes for å finne tiden det tar mellom to vinkelposisjoner, ved å benytte en teller som inkrementerer med systemklokka sin frekvens til å trigge en capture ved signal-event. Det som skjer da er at tellerverdien blir lagret i ett register. Når dette skjer kan man sette opp systemet til å trigge en interrupt slik at man kan behandle dataene videre.

C.4 Interrupt systemet

Interrupt systemet er blitt bygd om til å ha støtte for tre forskjellige prioriteter. Det er allerede en innebygd prioritet ut fra hvilken verdi interrupt-vektoren har. Men nå kan man også fordele interruptene i tre prioritetsnivåer. Inbyrdes i hvert nivå følges prioriteten til interrupt-vektoren, bortsett fra på laveste nivå hvor man har mulighet til å velge round-robin scheduling for å unngå starvation.

Tillegg D

Kretskortproduksjon

Dette er et supplement til teksten om kretskortproduksjon fra [14] fordi det viste seg å dukke opp nye problemer som ikke var forventet. Den fysiske størrelsen til XMEGA er av type TQPF100. Den har altså samme fysiske størrelse som en ATMEGA128, men istedenfor 64 bein har den 100. Dette stiller meget strenge krav til alle ledd i produksjonen. Det som viste seg å bli et problem var kvaliteten på utskriften av overheaden. Verken laserskrivere eller blekkskrivere som ble prøvd hadde god nok oppløsning til å lage print av høy nok kvalitet. Resultatet av dette er at man må skrape bort metall mellom "pads"-ene for XMEGA. Grunnet allerede liten fysisk størrelse er dette problematisk og man risikerer å ødelegge kretskortet. Det anbefales derfor at man prøver å finne skrivere med høyere oppløsning enn 1200X1200 pixler, som er det som har vært brukt.

Tillegg E

Parametertabeller

Dette kapittelet inneholder tabeller med parameterverdier for systemet.

Enkodere på gamle system	HEDS-5500 (E05)
Oppløsning	200cpr
Enkodere på nye system	HEDS-5500 (A05 og ombygd E05)
Oppløsning	500cpr

Tabell E.1: Tabellen viser hvilke enkodere som er brukt på gamle og nye sys

Arm	Vekt[kg]	Lengde(L_i)[m]	lengde(l_i)[m]
Arm_0	0.645	1	0.5
Arm_1	0.195	1.18	0.65

Tabell E.2: Tabellen viser målte parametre for armer brukt på enkelpendelsystem

Arm	Vekt[kg]	Lengde(L_i)[m]	lengde(l_i)[m]
Arm_0	0.645	1	0.5
Arm_1	0.195	1.18	0.65
Arm_2	0.1	0.63	0.25

Tabell E.3: Tabellen viser målte parametre for armer brukt på systemet enkelpendel hver side

Arm	Vekt[kg]	Lengde(L_i)[m]	lengde(l_i)[m]
Arm_0	0.645[1]	1	0.5
Arm_2	0.15	0.5	0.265
Arm_3	0.055	0.438	0.175

Tabell E.4: Tabellen viser målte parametre for armer brukt på dobbelpendelsystem

Strømligning	$G(s) = \frac{p_5 s^2 + p_6 s}{s^3 - p_2 s^2 - p_1 p_3 s - p_1 p_4}$
p_1	5.2
p_2	-600
p_3	≈ 0
p_4	≈ 0
p_5	550
p_6	≈ 0

Tabell E.5: Tabellen viser parametre funnet fra parameterestimering på det gamle systemet

Strømligning	$G(s) = \frac{p_5 s^2 + p_6 s}{s^3 - p_2 s^2 - p_1 p_3 s - p_1 p_4}$
p_1	7
p_2	-600
p_3	≈ 0
p_4	≈ 0
p_5	550
p_6	≈ 0

Tabell E.6: Tabellen viser parametre funnet fra parameterestimering på det nye systemet

Motor	Baldor 22-15 A1
K_T	0.84
Stall torque	3Nm
Max armature pulse amps	18A
Max voltage	180V

Tabell E.7: Tabellen viser parametre for motor

Regulator	TSNM 150-12-01
Continuous current	12A
Max peak current	30A(32A)
Voltage($\pm$10%)	150V

Tabell E.8: Tabellen viser parametre for regulatorkortet

Tillegg F

Motormodeller

Med PI-regulert strøm og P-regulert hastighet får man

$$\begin{aligned} si_a &= -\frac{1}{L_a}(R_a + K_{ip})i_a - \frac{1}{L_a}\left(K_E + \frac{K_{ii}J_m}{K_T}\right)\omega_m + \frac{1}{L_a}\left(K_{ip} + \frac{1}{s}K_{ii}\right)i_d \\ &= -\frac{1}{L_a}(R_a + K_{ip})i_a - \frac{1}{L_a}\left(K_E + \frac{K_{ii}J_m}{K_T}\right)\omega_m + \frac{1}{L_a}\left(K_{ip} + \frac{1}{s}K_{ii}\right)K_{\omega p}(\omega_d + \omega_m) \\ &= -\frac{1}{L_a}(R_a + K_{ip})i_a - \frac{1}{L_a}\left(K_E + \frac{K_{ii}J_m}{K_T} + K_{ip}K_{\omega p}\right)\omega_m \\ &\quad + \frac{K_{ii}K_{\omega p}}{L_a}\theta_m + \frac{K_{\omega p}}{L_a}\left(K_{ip} + \frac{1}{s}K_{ii}\right)\omega_d \end{aligned}$$

her ser man at det er nødvendig å augmentere modellen med en ekstra tilstand for å få med den integrerte av pådraget. For P-regulert strøm og PI-regulert hastighet $i_d = (K_{\omega p} + \frac{1}{s}K_{\omega i})(\omega_d - \omega_m)$ får man

$$\begin{aligned} si_a &= -\frac{1}{L_a}(R_a + K_{ip})i_a - \frac{K_E}{L_a}\omega_m + \frac{K_{ip}}{L_a}(K_{\omega p} + \frac{1}{s}K_{\omega i})(\omega_d - \omega_m) \\ &= -\frac{1}{L_a}(R_a + K_{ip})i_a - \frac{1}{L_a}(K_E + K_{ip}K_{\omega p})\omega_m - \frac{K_{ip}K_{\omega i}}{L_a}\theta_m + \frac{K_{ip}}{L_a}(K_{\omega p} + \frac{1}{s}K_{\omega i})\omega_d \end{aligned}$$

hvor man og trenger å augmentere med en tilstand for å få med den integrerte av pådraget. Disse to siste modellene kan man estimere parametrene i ved å sette opp på formen

$$\dot{\eta}_1 = u \tag{F.1}$$
$$\dot{\theta}_m = \omega_m \tag{F.2}$$
$$\dot{\omega}_m = p_1 i_a \tag{F.3}$$
$$\dot{i}_a = p_2 i_a + p_3\omega_m + p_4\theta + p_5 u + p_6\eta_1 \tag{F.4}$$

med $x = [\eta_1\ \theta_m\ \omega_m\ i_a\ p_1\ p_2\ p_3\ p_4\ p_5\ p_6]$.

Til slutt med PI-regulert strøm og PI-regulert hastighet får man

$$
\begin{aligned}
si_a &= -\frac{1}{L_a}(R_a + K_{ip})i_a - \frac{1}{L_a}\left(K_E + \frac{K_{ii}J_m}{K_T}\right)\omega_m + \frac{1}{L_a}\left(K_{ip} + \frac{1}{s}K_{ii}\right)i_d \\
&= -\frac{1}{L_a}(R_a + K_{ip})i_a - \frac{1}{L_a}\left(K_E + \frac{K_{ii}J_m}{K_T}\right)\omega_m + \frac{1}{L_a}\left(K_{ip} + \frac{1}{s}K_{ii}\right)(K_{\omega p} + \frac{1}{s}K_{\omega i})(\omega_d - \omega_m) \\
&= -\frac{1}{L_a}(R_a + K_{ip})i_a - \frac{1}{L_a}\left(K_E + \frac{K_{ii}J_m}{K_T}\right)\omega_m \\
&\quad + \frac{1}{L_a}\left(K_{ip}K_{\omega p} + (K_{ip}K_{\omega i} + K_{ii}K_{\omega p})\frac{1}{s} + K_{ii}K_{\omega i}\frac{1}{s^2}\right)(\omega_d - \omega_m) \\
&= -\frac{1}{L_a}(R_a + K_{ip})i_a - \frac{1}{L_a}\left(K_E + \frac{K_{ii}J_m}{K_T} + K_{ip}K_{\omega p}\right)\omega_m - \frac{K_{ip}K_{\omega i} + K_{ii}K_{\omega p}}{L_a}\theta_m - \frac{K_{ii}K_{\omega i}}{L_a}\frac{1}{s}\theta_m \\
&\quad + \frac{1}{L_a}\left(K_{ip}K_{\omega p} + (K_{ip}K_{\omega i} + K_{ii}K_{\omega p})\frac{1}{s} + K_{ii}K_{\omega i}\frac{1}{s^2}\right)\omega_d
\end{aligned}
$$

hvor man må augmentere med 3 ekstra tilstander og sette på formen

$$
\begin{aligned}
\dot{\eta}_1 &= \theta \\
\dot{\eta}_2 &= u \\
\dot{\eta}_3 &= \eta_2 \\
\dot{\theta}_m &= \omega_m \\
\dot{\omega}_m &= p_1 i_a \\
\dot{i}_a &= p_2 i_a + p_3\omega_m + p_4\theta + p_5\eta_1 + p_6 u + p_7\eta_2 + p_8\eta_3
\end{aligned}
$$

med $x^a = [\eta_1\ \eta_2\ \eta_3\ \theta_m\ \omega_m\ i_a\ p_1\ p_2\ p_3\ p_4\ p_5\ p_6\ p_7\ p_8]$. Denne siste er modellen som representerer `Baldor`-regulatoren og motoren. Denne modellen gir derimot ikke ut noe mer nyttig informasjon enn hva modellene over gjør.

Tillegg G

Implementering av ulineært kalmanfilter for parameterestimering i level 2 m-file s-function

Hele bakgrunnen for at implementasjonen er mulig er funksjonen `inline`. Denne funksjonen gjør at man kan gi inn ulineære systemer til kalmanfilteret. Under følger et eksempel på en ulineær funksjon og hvordan denne skrives inn i en inline-funksjon

$$F(x,u) = \begin{cases} x_1^2 + u \\ x_1 + \sin x_2 \end{cases} \tag{G.1}$$

$$\mathrm{F} = \mathrm{inline}\,(\text{'x(1)\^{}2+u(1);x(1)+sin(x(2))'},\text{'x'},\text{'u'}) \tag{G.2}$$

Ved å skrive F(x,u) i matlab kan man så gi inn vilkårlig x og u. Inline-funksjonen er derfor benyttet i s-funksjonen for kalmanfilteret der hvor det har vært nødvendig å representere ulineariteter. I figur G.1 på side 107 ser man brukergrensesnittet til kalmanfilteret. Man kan velge mellom forskjellige filtertyper i nedtrekksmenyen noe som vil gjøre at resten av brukergrensesnittet forandrer seg i forhold til hva man velger. Det er implementert filter for tidsinvariante lineære og ulineære systemer, samt filter for parameterestimering. Ut fra hva man velger vil det komme opp et lite eksempel for hvordan man skal skrive inn de forskjellige verdiene. Man kan og trykke på hjelp og få opp en mer utførlig bruksanvisning for hvordan man skal bruke filteret. Fordelen med å bruke inline-implementasjonen er at man ved ulineære systemer kan skrive inn ligningene direkte. Det er viktig å merke seg at ligningene man skriver inn skal være for kontinuerlige systemer. Diskretiseringen skjer i filteret ved hjelp av eulerdiskretisering. Avkryssningsboksen `Debugging` sørger for at innovasjonsprosessen og kalman-gain også blir gitt som utganger i `Simulink` slik at man kan analysere hvor bra filterparametrene fungerer. Som et lite eksempel følger her ligningene for F(x,u) og DF(x,u) for systemet (F.1)$\cdots$(F.4) slik de må skrives i `Matalab`.

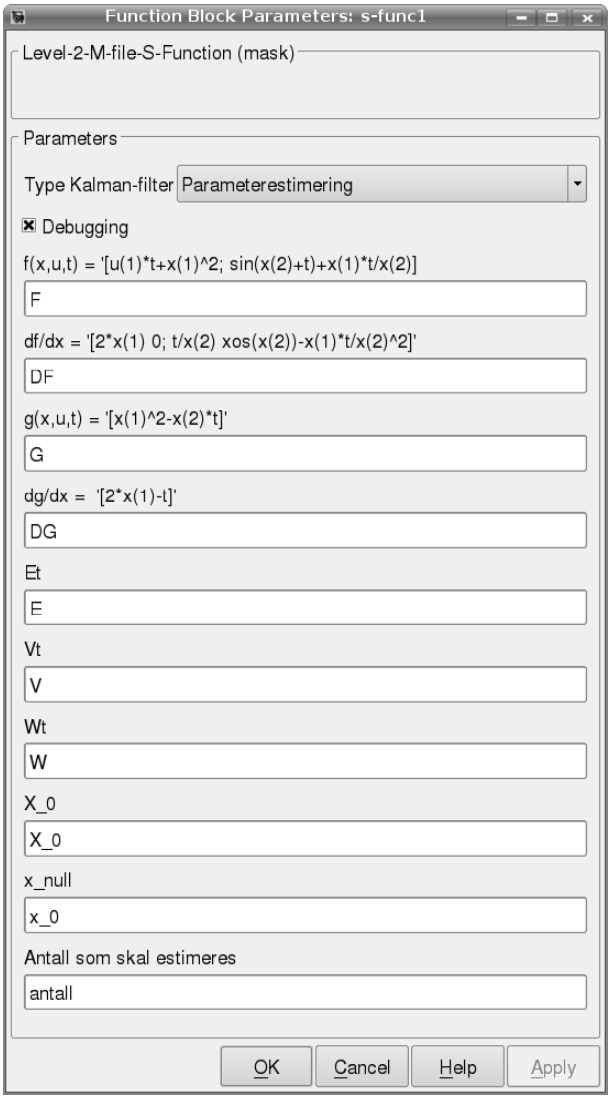
```
F = '[u(1);  ...
     x(3);  ...
     x(5)*x(4);  ...
     x(6)*x(4)+x(7)*x(3)+x(8)*x(2)+x(9)*u(1)+x(10)*x(1);  ...
     0;0;0;0;0;0]';

DF = '[0  0  0  0  0  0  0  0  0  0;  ...
       0  0  1  0  0  0  0  0  0  0;  ...
       0  0  0  x(5)  x(4)  0  0  0  0  0;  ...
       x(10)  x(8)  x(7)  x(6)  0  x(4)  x(3)  x(2)  u(1)  x(1);  ...
       0  0  0  0  0  0  0  0  0  0;  ...
       0  0  0  0  0  0  0  0  0  0;  ...
       0  0  0  0  0  0  0  0  0  0;  ...
       0  0  0  0  0  0  0  0  0  0;  ...
       0  0  0  0  0  0  0  0  0  0;  ...
       0  0  0  0  0  0  0  0  0  0]';
```

Her legger man merke til at alle de site leddene er 0 istedenfor enere diagonalt nedover. Dette har samenheng med hvordan diskretiseringen er implementert.

Vedlagt på cd er det en mappe som inneholder eksempelsystemer som det kan gjøres parameterestimering på i simulering. Det er både lagt med et lineært og et ulineært system hvor det er klargjort kode for estimering av flere forskjellige kombinasjoner av parametre. For å bruke filene åpner man ønsket m-fil og i denne finner man ut hvilken modell som simuleres, dette står i sim('system.mdl'). Ved å åpne denne tilhørende mdl-filen og så åpne oscilloskopet som viser parametrene kan man under kjøring se hvordan de svinger inn. For å kjøre det hele er det bare å starte m-fila med for eksempel `F5`.

G.1 Valg av startparametre for parameterestimering

Valg av startparametre må gjøres med omhu for å sikre rask og presis innsvingning av parametrene man ønske å estimere. Det er mange parametre som man må velge verdi på hvor flere av dem er meget relevante for hvordan estimeringen vil utløpe seg. Kovariansmatrisen for systemstøy V, og kovariansmatrisen for målestøy W setter premissene for hvor raskt systemet svinger inn. En god start er å velge V og W som diagonalmatriser, og så må elementene i V som sier hvor mye man kan stole på systemmodell og parametrene velges med omhu. Systemmodellen vil man gjerne stole mye på så her velger man verdier rundt $0.001 - 0.00001$. For parametrene bør verdier velges ut fra hvilken verdi parametrene har, eller kommer til å få. Parametre som har en høy verdi må ha høy verdi i V matrisen, mens parametre som har lav verdi må ha lav verdi. Velg for eksempel verdien 1 for den høyeste parameteren og gjør en skalering for de resterende parametrene i forhold til den med lavest verdi. Dette er for å sikre at filteret prøver å svinge inn parametre med høy verdi like mye som det prøver å svinge inn parametre med lav verdi. Verdiene $\hat{x}_0$ og $\hat{X}_0$ må og velges med omhu, og da særlig startverdiene til systemet. I tillegg til at man velger starttilstander så velger man og startverdier for parametre som skal estimeres. Feil valg



Figur G.1: Figuren viser brukergrensesnittet til kalmanfilteret brukt for parameterestimering

her kan gjøre at filteret ikke svinger inn, eller svinger inn til feil verdier. Det er absolutt fordel å ha en idé om hva slags verdi parametrene skal ha.

Tillegg H

Implementasjon av ulineært kalmanfilter for tilstandsestimering i level 2 c-file s-function

Implementasjon av ulineært kalmanfilter uten bruk av matrisebibliotek er en ganske kompleks operasjon. Grunnen til at det ikke har vært brukt matrisebiblioteket til matlab i implementasjonene i s-funksjon er fordi man har mer kontroll på hvilke operasjoner som faktisk utføres. Siden både A og C matrisen av det lineariserte systemet er sparse gjør dette at matrisemultiplikasjon, summeringer, subtraheringer og ikke minst inverteringer ikke er nødvendig å utføre på alle elementene i matrisene siden disse er 0. Det har derfor blitt benyttet en løsning med å regne ut algebraisk alle matriselikninger på forhånd for så å implementere ligningene for hvert enkelt matriseelement hver for seg i s-funksjonen. Det sier seg selv at det blir mye data å holde styr på om man skulle gjort dette for hånd. Heldigvis går det ann å bruke programmet `Maple` for å regne ut algebraisk på forhånd. Selv med bruk av `Maple` har ikke dette vært problemfritt. Når man regner ut for dobbel og trippelpendel bli resultatene meget store, spesielt i ligningen hvor det skjer invertering. Når man så prøver å kopiere resultatet for å implementere dette i s-funksjonen ender man opp med at `Maple` bruker opp all tilgjengelig ram, som på dataen som har vært brukt tilsvarer 2Gig, og krasjer. Eller så hender det og at programmet bruker opp all ram uten at noe blir kopiert. For å overkomme dette problemet har det vært nødvendig å splitte opp flere av ligningene i mindre elementer.

En ting man desverre ikke kommer unna å gjøre for hånd er lineariseringen av systemet som må gjøres ved å derivere $f(x, u)$ for å finne A-matrisen. Løsningen for å få til dette har vært å ta alle elementer i M, C og G matrisen og derivere disse med hensyn på alle tilstander. Det sier seg selv at sannsynligheten for å gjøre feil blir ganske stor på grunn av alle ligningene man da ender opp med. Det henvises til `regulator.c` for å se på

oppdelingen og metodene som er brukt for å få til dette på en oversiktlig måte. Noe som er verdt å merke seg her er at allerede fra starten av er det tatt hensyn til at man skal kunne bytte friksjonsmodell på en enkel måte. Om man bytter modell er det kun nødvendig å skrive inn den nye, samt regne ut den deriverte.

Selve plassering av kalmanligningene i s-funksjonen er gjort ut ifra hvilke deler av s-funksjonen som kjøre på hvilke tidspunkt. MdlOutput kjører først, så MdlUpdate. Kalmanfilterets oppbygning tilsier da at man plasserer filterligningene i MdlOutput og prediksjonsligningene i MdlUpdate slik som koden under viser.

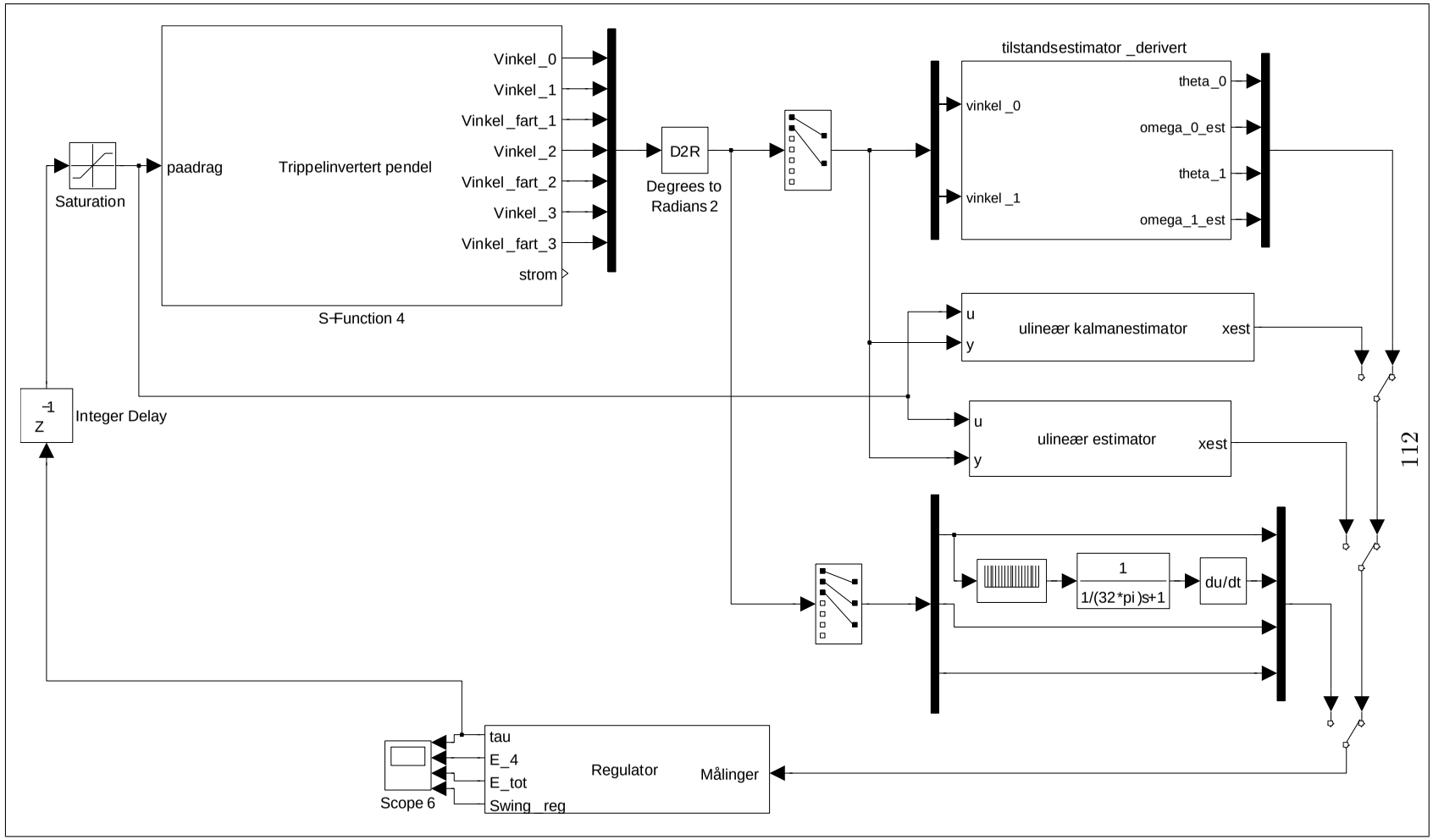
```
static void mdlOutputs(SimStruct *S, int_T tid){
  if(Tid == 0){
    //lagre initial Xhat
    //lagre initial xhat
    //gi ut initial xhat
  }
  else{
    //regne ut K
    //regne ut og lagre Xhat
    //regne ut og lagre xhat
    //gi ut xhat
  }
}
static void mdlUpdate(SimStruct *S, int_T tid){
  //regne ut M, C og G matrisene
  //derivere M, C og G matrisene
  //sette opp den diskrete A matrisen
  //regne ut Xbar og lagre denne
  //regne ut xbar og lagre denne
```

I mdlUpdate er det viktig å huske å utføre euler-diskretiseringen av pendelmodellen.

Tillegg I

Simulinkdiagram for RTW-modell

Figur I.1 på neste side viser simulinkmodellen for enkelpendel. Figuren viser det typiske utlegget i `Simulink` som er gjort for alle pendelmodeller både for simulering og for `RTW`. De simulinkmodellene som faktisk er brukt inneholder derimot mange flere oscilloskop slik at det skal være mulig å sammenligne og analysere de forskjellige signaler.
Oppe til venstre ser man simulinkblokken som representerer pendelsystemet. I simulinkdiagrammene som brukes for simulering er denne blokken en s-funksjon som inneholder de matematiske modellene hvor det er en inngang og like mange utganger som det er tilstander i systemet. For RTW-diagrammene er denne blokken en s-funksjon som er kommentert i J på side 113.
Hele høyre del av simulinkmodellen er blokker for forskjellige tilstandsestimatorer. I figur I.1 på neste side er det fra øverst til nederst, tilstandsestimering ved derivering, ulineært kalmanfilter, high-gain observer og hardwareestimering. På denne siste er problemet med at det ikke er noe hardwareestimering på vinkelhastigheten til `arm_0` løst ved å estimere ved filtrering og derivering.
Nederst i simulinkdiagrammet ser man regulatoren. Denne tar inn en vektor med tilstander og regner så ut pådrag. Blokken gir og ut energiene til hver enkelt arm i systemet samt totalenergien. Den nederste utgangen angir hvilken regulator som til enhver tid kjører.



Figur I.1: Figuren viser et eksempelblokkdiagram for hvordan systemet er satt opp i Simulink

Tillegg J

s-funksjonen for interface mellom RTW og pendelsystem

For å forstå koden fult ut er det anbefalt å se på fila `sfunc_trippelinvertert_pendel_wrapper` hvor utsnittet er hentet fra. denne fila består av over 400 linjer, det er derfor det bare er tatt med et utsnitt under som inneholder de forskjellige funksjonene. Det finnes to versjoner av denne fila, avhengig av om det blir sendt kun vinkelverdier fra `node_0`, eller om det og blir send verdier fra hardwareestimeringen.

```
1  void sfunc_trippelinvertert_pendel_Outputs_wrapper(const real_T *paadrag,
                            real_T *vinkel_0,
3                           real_T *vinkel_1,
                            real_T *vinkel_fart_1,
5                           real_T *vinkel_2,
                            real_T *vinkel_fart_2,
7                           real_T *vinkel_3,
                            real_T *vinkel_fart_3,
9                           real_T *strom)
   {
11   //sørge for lagring av verdier som skal frem om tilbake mellom RTW og trådene under
   }
13
   void *dev_pendel(void *c){
15   //lese vinkel_0
     //lese strom
17   //skrive paadrag
   }
19
   void les_quadratur(void){
21   //leser av kvadraturtelleren HCTL2000 som er koblet til enkoder_0
     //gjøres via driveren for PCL812
23 }

25 void skriv_paadrag(float verdi){
     //gir pådrag til motor
27   //gjøres via driveren for PCL812
   }
29
   void les_strom(void){
31   //leser av strømmen på motor
     //gjøres via driveren for PCL812
33 }

35 #define BAUDRATE 115200
   void *les_rs232_vinkel(void *c){
37   //leser av vinkel_1, vinkel_2 og vinkel_3
     //leser av vinkel_fart_1, vinkel_fart_2 og vinkel_fart_3
39   //gjøres via serielldriveren devc-ser8250
   }
41
   void pendel_start(void){
43   //initialiserer variabler
     //oppretter tråder
45   //setter prioriteter på tråder
```

```
    //åpner forskjellige drivere
47  //starter tråder
    //sier fra til watchdog at systemet starter opp
49 }

51 void pendel_stopp(void){
    //sier fra til watchdog at systemet stopper
53  //stopper tråder
    //lukker derivere
55 }
```

Tillegg K

Eksempel på init-fil

Under følger init-fila som er brukt for enkelpendel hver side. Oppsette på denne og de andre initfilene er identiske bortsett fra forskjellig antall parametre og forskjellige verdier. Første del av koden er for oppsett av modellparametre og initiering av starttilstander. Andre del er oppsett av variabler for regulator og dertil nødvendige parametre samt oppsett av parametre for det ulineære kalmanfilteret som er brukt for tilstandsestimering.

```
%length and masses from measurements:
clear all;
clc

K_T = 2.2;%fra kraftmålinger
p_1 = 23.5;%fra parameterestimering
p_2 = -615;
p_3 = 0;
p_4 = 0;
p_5 = 600;
p_6 = 0;

NUM = [p_5 p_6 0];
DEN = [1 -p_2 -p_1*p_3 -p_1*p_4];

%rod
m_1=0.852;   L_1=1/2;   l_1=0;

I_1=K_T/p_1; %kgm^2 ... bruker for arm og motor fra parameterestimering

%short single pendulum
m_2=0.236; m_2=0.0637;  L_2=0.65;    l_2=L_2/2;   I_2= (m_2*L_2^2)/12;

%long single pendulum
m_3 =0.183;  L_3 = 1.16;  l_3 = 0.55; I_3 = (m_3*(L_3)^2)/12;

g=9.81;

theta_1    = 0;
thetaDot_1 = 0;
friction_1 = 0.11772;%fra kraftmåling

%korte pendel
theta_2    = pi;
thetaDot_2 = 0;
friction_2 = 0.001;%noe usikker

%lange pendel
theta_3    = pi;
thetaDot_3 = 0;
friction_3 = 0.001;%noe usikker

constants2Sfun=[m_1 L_1 l_1 I_1 m_2 L_2 l_2 I_2 m_3 L_3 l_3 I_3 g K_T];
constants=constants2Sfun;
initCond=[theta_1 thetaDot_1 theta_2 thetaDot_2 theta_3 thetaDot_3];
friction=[friction_1 friction_2 friction_3];

%regner ut gain for balanseringsregulatorer
linearizedAboutEquilibriums
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%definering av indexer i g-(gain)-vektoren i output-funksjonen
%setter parametre for skifte mellom regulatorer
sul_inn_E1  = -0.3; %swing_up_long_E1. får svinge opp lang pendel om E_1 er under denna
sul_ut_E1   = 0.3;  %swing_up_long_E1. får svinge opp lang pendel om E_1 er under denna
UD_inn_t1   = 0.5;  %abs(theta_3) er under denna OG
UD_inn_td1  = 3;    %abs(thetaDot_3) er under denna
UD_ut_t1    = 0.65; %abs(theta_3) er over denne ELLER
UD_ut_td1   = 3.5;  %abs(thetaDot_3) er over denne
sus_inn_E1  = 0.1;  %får starte oppsving av korte pendel om abs(E_4) er under denne OG
sus_inn_t0  = 0.5;  %abs(theta_1) er under denne OG
sus_inn_td0 = 0.5;  %abs(thetaDot_1) er under denne
sus_inn_E2  = 0;    %får starte å pumpe inn energi i kort pendel om E_2 er under denne
sus_ut_E2   = 0.08; %får ikke fortsette å pumpe inn energi om E_2 er over denne
sus_ut_E1   = 0.2;  %går tilbake til balansering lang pendel om abs(E_4) er over denne ELLER
sus_ut_t1   = 0.5;  %abs(theta_3) er over denne ELLER
sus_ut_td1  = 3;    %abs(thetaDot_3) er over denne
UU_inn_t1   = 0.2;  %får starte balansering om abs(theta_3) er under denna OG
UU_inn_td1  = 1.5;  %abs(thetaDot_3) er under denna OG
UU_inn_t2   = 0.2;  %abs(theta_2) er under denna OG
UU_inn_td2  = 1.5;  %abs(thetaDot_2) er under denna
UU_ut_t1    = 0.5;  %går tilbake til oppsving kort pendel om abs(theta_3) er over denne ELLER
UU_ut_td1   = 3.5;  %abs(thetaDot_3) er over denne ELLER
UU_ut_t2    = 0.5;  %abs(theta_2) er over denne ELLER
UU_ut_td2   = 4.5;  %abs(thetaDot_2) er over denne

%diverse forsterkninger
start_strom = -1;   %for å starte opp det hele
kl_1        = 4;    %forsterkning oppsving lang: holde theta_0 = 0
kl_2        = 4;    %forsterkning oppsving lang: holde thetaDot_0 = 0
kl_3        = 16;   %forsterkning oppsving lang: oppsvinget
%K_UD fås fra linearizedAboutEquilibriums
kk_1        = -4;  %forsterkning oppsving kort: holde theta_0 = 0
kk_2        = -4;  %forsterkning oppsving kort: holde thetaDot_0 = 0
kk_3        = 14;  %forsterkning oppsving kort: oppsvinget
%K_UU fås fra linearizedAboutEquilibriums

% kk_1        = 0;  %forsterkning oppsving kort: holde theta_0 = 0
% kk_2        = 0;  %forsterkning oppsving kort: holde thetaDot_0 = 0
% kk_3        = 13; %forsterkning oppsving kort: oppsvinget

gain = [sul_inn_E1 sul_ut_E1 ...%for oppsving lang
        UD_inn_t1 UD_inn_td1 UD_ut_t1 UD_ut_td1 ...%for stab lang
        sus_inn_E1 sus_inn_t0 sus_inn_td0 sus_inn_E2 sus_ut_E2 sus_ut_E1 sus_ut_t1 sus_ut_td1 ...%for oppsving kort
        UU_inn_t1 UU_inn_td1 UU_inn_t2 UU_inn_td2 UU_ut_t1 UU_ut_td1 UU_ut_t2 UU_ut_td2 ...%for stab UU
        start_strom kl_1 kl_2 kl_3 K_DU kk_1 kk_2 kk_3 K_UU] %diverse forsterkninger


% definering av diskrete tilstander(og parametre som kommer inn via KALMANFILTERPARAMETRE)
xhat1  = 0;
xhat2  = 0;
xhat3  = pi;%husk at denne er = pi
xhat4  = 0;
xhat5  = pi;%husk at denne er = pi
xhat6  = 0;
Xhat11 = 1;%ganske sikker på at satrtvinkel_0=0
Xhat12 = 0;
Xhat13 = 0;
Xhat14 = 0;
Xhat15 = 0;
Xhat16 = 0;
Xhat21 = 0;
Xhat22 = 1;%ganske sikker på at startfart_0=0
Xhat23 = 0;
Xhat24 = 0;
Xhat25 = 0;
Xhat26 = 0;
Xhat31 = 0;
Xhat32 = 0;
Xhat33 = 1;%ganske sikker på at startvinkel_1=pi
Xhat34 = 0;
Xhat35 = 0;
Xhat36 = 0;
Xhat41 = 0;
Xhat42 = 0;
Xhat43 = 0;
Xhat44 = 1;%ganske sikker på at startfart_1=0
Xhat45 = 0;
Xhat46 = 0;
Xhat51 = 0;
Xhat52 = 0;
Xhat53 = 0;
```

```
134 Xhat54 = 0;
135 Xhat55 = 1;%ganske sikker på at startvinkel_2=0
136 Xhat56 = 0;
137 Xhat61 = 0;
138 Xhat62 = 0;
139 Xhat63 = 0;
140 Xhat64 = 0;
141 Xhat65 = 0;
142 Xhat66 = 1;%ganske sikker på at startfart_2=0
143 W_1 = 0.1;%gode målinger?
144 W_3 = 0.1;%gode målinger?
145 W_5 = 0.1;%gode målinger?
146 V_11 = 1;
147 V_22 = 100;
148 V_33 = 1;
149 V_44 = 100;
150 V_55 = 1;
151 V_66 = 100;
152
153 kalmparam = [xhat1 xhat2 xhat3 xhat4 xhat5 xhat6...
154               Xhat11 Xhat12 Xhat13 Xhat14 Xhat15 Xhat16...
155               Xhat21 Xhat22 Xhat23 Xhat24 Xhat25 Xhat26...
156               Xhat31 Xhat32 Xhat33 Xhat34 Xhat35 Xhat36...
157               Xhat41 Xhat42 Xhat43 Xhat44 Xhat45 Xhat46...
158               Xhat51 Xhat52 Xhat53 Xhat54 Xhat55 Xhat56...
159               Xhat61 Xhat62 Xhat63 Xhat64 Xhat65 Xhat66...
160               W_1 W_3 W_5 V_11 V_22 V_33 V_44 V_55 V_66]
161
162 clear m_1 L_1 l_1 I_1 m_2 L_2 l_2 I_2 m_3 L_3 l_3 I_3 g
163 disp('initialisation complete');
```

Tillegg L

Implementasjon av regulator i level 2 c-file s-function

Her følger observatorstrukturen for regulatoren for enkelpendel hver side og under er en forklaring på hva de forskjellige delene gjør. Dette er bare en liten del av koden som utgjør s-funksjonen til regulatoren, blant annet mangler alle matematiske utregninger og oppsett i forbindelse med s-funksjonen.

```
1  if(tiden < 0.2){//reg 0: oppstart
     regulator = 1;//må settes lik 1 for å kunne gå videre
3    under_reg = 1;//må settes lik 1 for å kunne gå videre
     if(tiden>0.1){
5      tau[0] = g[start_strom];
     }
7  }
   //for å hindre skade på systemet ved for høy hastighet
9  else if(thetaDot_1>maks_rot_hast || thetaDot_1<-maks_rot_hast){
     if(thetaDot_1>0){
11     tau[0]=-2;
     }
13   else{
       tau[0]=2;
15   }
17 }else{
     if(regulator==1){//reg 1: oppsving
19     //sjekk om det skal byttes til balanseringsregulator lang
       if(theta_3>-g[UD_inn_t1] && theta_3<g[UD_inn_t1] && thetaDot_3>-g[UD_inn_td1] && thetaDot_3<g[UD_inn_td1]){
21     regulator=2;
       }
23     else{
       if(under_reg==1){
25       if(E_3b > g[sul_ut_E1]){
           under_reg=2;
27       }
         else{
29         u_ = -g[kl_1]*theta_1 -g[kl_2]*thetaDot_1 + g[kl_3]*cos(theta_3)*atan(-thetaDot_3);
           tau[0] = M_1*u_ + C_1*thetaDot_1 + C_2_1*thetaDot_2 + C_2_2*thetaDot_3;
31         reg[0]=1.1;
         }
33     }
       if(under_reg==2){
35       if(E_3b < g[sul_inn_E1]){
           under_reg=1;
37       }
         else{//droppe pådrag
39         tau[0]=0;
           reg[0]=1.2;
41       }
       }
43     }
     }
45   if(regulator==2){//reg 2: balansering lang
       //sjekke om det skal byttes tilbake til oppsvingsregulator lang
47     if(theta_3<-g[UD_ut_t1] || theta_3>g[UD_ut_t1] || thetaDot_3<-g[UD_ut_td1] || thetaDot_3>g[UD_ut_td1]){
```

```
    regulator=1;
49  tau[0]=0;
    reg[0]=regulator;
51  under_reg=1;
    }
53  //sjekke om det skal byttes til balansering lang og oppsvingsregulator kort
    else if(E_3b>-g[sus_inn_E1] && E_3b<g[sus_inn_E1] && theta_1>-g[sus_inn_t0] && theta_1<g[sus_inn_t0] && thetaDot_1>-g[
55  regulator=3;
    under_reg=1;
57  }
    else{//balansering
59  tau[0] = -g[K_UD_1]*theta_1 -g[K_UD_2]*thetaDot_1 -g[K_UD_3]*theta_2 -g[K_UD_4]*thetaDot_2 -g[K_UD_5]*theta_3 -g[K_UD_
    reg[0]=regulator;
61  }
  }
63 if(regulator==3){//reg 3: oppsving kort
    //sjekke om det skal byttes tilbake til stabiliseringsregulator lang
65  if(E_3b>g[sus_ut_E1] || E_3b<-g[sus_ut_E1] || theta_3<-g[sus_ut_t1] || theta_3>g[sus_ut_t1] || thetaDot_3<-g[sus_ut_td
    regulator=2;
67  tau[0]=0;
    reg[0]=regulator;
69  }
    //sjekke om det skal byttes til balansering lang og balansering kort
71  else if(theta_3>-g[UU_inn_t1] && theta_3<g[UU_inn_t1] && thetaDot_3>-g[UU_inn_td1] && thetaDot_3<g[UU_inn_td1] &&
    theta_2>-g[UU_inn_t2] && theta_2<g[UU_inn_t2] && thetaDot_2>-g[UU_inn_td2] && thetaDot_2<g[UU_inn_td2]){
73  regulator=4;
    }
75  else{
    if(under_reg==1){
77    if(E_2b>g[sus_ut_E2]){
        under_reg=2;
79    }
      else{
81      u2_=-g[kk_3]*cos(theta_2)*atan(thetaDot_2);
        u_=-g[kk_1]*theta_1 -g[kk_2]*thetaDot_1 + u2_;
83      tau[0]= M_1*u_ + C_1*thetaDot_1 + C_2_1*thetaDot_2 + C_2_2*thetaDot_3 -g[K_UD_1]*theta_1 -g[K_UD_2]*thetaDot_1 -g[
        //tau[0]=u2_ -g[K_UD_1]*theta_1 -g[K_UD_2]*thetaDot_1 -g[K_UD_3]*theta_2 -g[K_UD_4]*thetaDot_2 -g[K_UD_5]*theta_3
85      reg[0]=3.1;
      }
87  }
    if(under_reg==2){
89    if(E_2b<g[sus_inn_E2]){
        under_reg=1;
91    }
      else{//droppe pådrag
93      tau[0]= -g[K_UD_1]*theta_1 -g[K_UD_2]*thetaDot_1 -g[K_UD_3]*theta_2 -g[K_UD_4]*thetaDot_2 -g[K_UD_5]*theta_3 -g[K_
        reg[0]=3.2;
95    }
    }
97  }
  }
99 if(regulator==4){//reg 4: balansering UU
    //sjekke om det skal byttes tilbake til oppsving kort
101 if(theta_3<-g[UU_ut_t1] || theta_3>g[UU_ut_t1] || thetaDot_3<-g[UU_ut_td1] || thetaDot_3>g[UU_ut_td1] || theta_2<-g[UU
    regulator=3;
103 tau[0]=0;
    reg[0]=regulator;
105 }
    else{//balansering UU
107 tau[0] = -g[K_UU_1]*theta_1 -g[K_UU_2]*thetaDot_1 -g[K_UU_3]*theta_2 -g[K_UU_4]*thetaDot_2 -g[K_UU_5]*theta_3 -g[K_UU_
    reg[0]=regulator;
109 }
  }
111 }
```

Det er viktig å merke seg at `tau[0]` er pådraget mens `g[*]` er parametre som er gitt inn til regulatoren via init-fila. Alle parametre som begynner med `E` er energier hvor det etterfølgende tallet forteller hvilken arm den gjelder for. For parametrene som hentes ut med `g[*]` finnes det to alternativer. De som begynner med `K` er forsterkninger, mens de resterende er betingelser for bytte mellom regulatorer.

- `if(tiden < 0.2){//reg 0: oppstart` - Denne if sørger for oppstart av systemet. Her blir det gitt et pådrag som man velger i init-fila, som holdes i ett kort tidsintervall. Dette er for å sørge for at systemet får en identisk oppstart hver gang. Her blir også parametrene som velger regulator initiert.

- `else if(thetaDot_1>maks_rot_hast || thetaDot_1<-maks_rot_hast){` - Denne else if er meget viktig. Dette er en av to sikkerhetsfunksjonene som hindrer ukontrollerte høye rotasjonshastigheter. Det er implementert en sjekk i watchdogen som er skrevet, se kap 6.5.1 på side 41, som ikke tar høyde for ekstreme akselerasjoner av systemet. Dette ble oppdaget under forsøk da systemet rakk å spinne opp i meget høy hastighet i løpet av et halvt sekund. Denne else if stopper dette effektivt ved å bremse ned systemet hvis rotasjonshastigheten overgår en forhåndsbestemt verdi.

- `else{` - Her begynner selve regulatorstrukturen, og her er det og veldig mange parametre som kan justeres fra init-fil. under følger forklaring for hver.

    1. `if(regulator==1){//reg 1: oppsving` - Dette er oppsvingsregulatoren for lang pendel. Denne vil alltid kjøre om begge pendlene er utenfor sine stabiliseringsområder. Her er det tre ting som kan skje. Det blir først sjekket mot ett sett med parametre som setter kravene for å bytte til balanseringsregulator for lange pendel. Om ikke kravene for å gjøre bytte er oppfylt er det så to muligheter. Fortsette å pumpe energi inn i systemet ved å gi pådrag eller sørge for å ta energi ut av systemet ved å ikke gi pådrag. Bytte mellom disse to alternativene skjer også med overlapping
    2. `if(regulator==2){//reg 2: balansering lang` - Dette er balanseringsregulatoren for den lange pendelen. Her er det tre mulige ting som kan skje. Det sjekkes for om pendelen er utenfor området som kreves for å balansere, hvis dette er tilfellet byttes det tilbake til oppsvingsregulator. Hvis den lange er balansert og rotasjonshastighet og vinkel til `arm_0` er under en forhåndsbestemt verdi byttes det til "balansering lang og oppsving av kort"-regulator. Tredje mulighet er at det gis pådrag for å balansere den lange pendelen.
    3. `if(regulator==3){//reg 3: oppsving kort` - Denne regulatoren sørger for å svinge opp korte pendel etter at den lange er svingt opp og balansert. Her er det fire ting som kan skje. Det kan byttes tilbake til balansering-lang-regulator eller videre til balansering lang og kort. Kravene for disse byttene settes i initfila. Om det skal gis pådrag for å pumpe energi inn i den korte pendel, eller om det kun skal fokuseres på å fortsette å balansere den lange pendelen for å trekke energi ut av den korte pendelen avhenger av tilstanden til systemet. Byttet mellom disse to regulatorene er overlappende.
    4. `if(regulator==4){//reg 4: balansering UU` - Om begge pendlene er på høy kant og tilfredsstiller krav for rotasjonshastighet og posisjon vil denne regulatoren sørge for å gi pådrag som sørger for å holde dem balansert. Hvis ikke dette er mulig vil det byttes tilbake til forrige regulator.

Tillegg M

Watchdog

Under følger de viktigste elementene i koden for watchdog. Watchdog-en er implementert som device manager og det er derfor en god del støttestruktur som ikke er tatt med i kodeeksempelet under.

```
1  //definering av funksjoner
   int devctrl_PENDEL(resmgr_context_t *ctp, io_devctl_t *msg, RESMGR_OCB_T *ocb);
3  void *timer_traad(void *c);

5  //definering av melding
   typedef union {
7          struct _pulse   pulse;
           /* your other message structures would go
9             here too */
   } my_message_t;
11
   #define TIMER_FYRA     _PULSE_CODE_MINAVAIL  //definering av pulse for timeren
13 #define SJEKKE_VINKEL _PULSE_CODE_MINAVAIL+1  //definering av pulse for vinkel
   struct itimerspec        itime, vinkel_itime;          //tidsstrukt
15 timer_t                   timer_id, vinkel_timer_id;       //timerstrukt
   struct sigevent          event, vinkel_event;          //eventstrukt
17 int                      chid;         //int for kanalnr
   my_message_t             msg;          //opprettelse av melding
19 pid_t            pid;         //for prosessid til kallende prosess
   float            *data, gammel_vinkel; //global for samplingsfrekvens/vinkel
21
   int main(int argc, char *argv[]){
23   //Åner nødvendige drivere for PCL812
     //initialiserer tråder
25   //generelt oppsett av driveren
   }
27
   //devctl() for å muligjøre start/stopp/resetting av pendelsystem
29 int devctrl_PENDEL(resmgr_context_t *ctp, io_devctl_t *msg, RESMGR_OCB_T *ocb){
     switch (devctrl melding) {
31     case WATCHDOG_DEVCTL_START :
         //oppretter timer som tråd
33       //setter timerintervall ut fra data sendt fra RTW-koden
         //starter timer
35       //skrur på muligheten for å gi pådrag til motor

37     case WATCHDOG_DEVCTL_STOP:
         //skrur av mulighet for å gi pådrag til motor
39       //stopper og sletter timertråd

41     case WATCHDOG_DEVCTL_UPDATE:
         //resetter watchdog-timer
43   }
   }
45
   //tråd som kjører og sender kill signal til RTW hvis den ikke gir lyd fra seg
47 void *timer_traad(void *c){
     int rcvid;
49   //printf("prioritet = %i\n", getprio(0));
     for(;;){
51       rcvid = MsgReceivePulse(chid, &msg, sizeof(msg), NULL);
         if (rcvid == 0) { /* we got a pulse */ //hvis rcvid>0 så er det melding
53           if (msg.pulse.code == TIMER_FYRA) {
```

```
            printf("RTW krasjet. Stopper programmet\n");
            //skrur av pådrag
            set_bit(devctl_maske, TOTAL_DISABLE);
            set_bit(devctl_maske, RESET);
            clear_bit(devctl_verdi, TOTAL_DISABLE);
            clear_bit(devctl_verdi, RESET);
            devctl_buffer[0] = devctl_maske;
            devctl_buffer[1] = devctl_verdi;
            pthread_mutex_lock(&mutex);
            write(fd_dig_out, &devctl_buffer, sizeof(devctl_buffer));
            pthread_mutex_unlock(&mutex);
            //stopper RTW
            kill(pid,SIGINT);
            //rydder opp
            timer_delete(timer_id);
            timer_delete(vinkel_timer_id);
            ChannelDestroy(chid);
            pthread_exit(NULL);
              } /* else other pulses ... */
              if(msg.pulse.code == SJEKKE_VINKEL){
                if(abs(gammel_vinkel - *data) > 180){
                  //printf("Rotasjonshastighet for stor");
                  printf("Rotasjonshastighet for stor = %f\n", gammel_vinkel - *data);
                  //stoppe RTW
              set_bit(devctl_maske, TOTAL_DISABLE);
              set_bit(devctl_maske, RESET);
              clear_bit(devctl_verdi, TOTAL_DISABLE);
              clear_bit(devctl_verdi, RESET);
              devctl_buffer[0] = devctl_maske;
              devctl_buffer[1] = devctl_verdi;
              pthread_mutex_lock(&mutex);
              write(fd_dig_out, &devctl_buffer, sizeof(devctl_buffer));
              pthread_mutex_unlock(&mutex);
                  //stopper RTW
                  kill(pid,SIGINT);
                  //rydder opp
                  timer_delete(timer_id);
                  timer_delete(vinkel_timer_id);
                  ChannelDestroy(chid);
                  pthread_exit(NULL);
                }
                gammel_vinkel = *data;
              }
        } /* else other messages ... */
    }
}
```

- main() - starter opp driveren osv....
- `devctrl_PENDEL` - denne funksjonen kjører når man kaller `devctl()` fra s-funksjonen.
  - `WATCHDOG_DEVCTL_START` - Denne koden sørger for å sette opp timere og initialisere disse med riktige verdier og starte opp tråden `timer_traad()`. Skrur og på mulighet for å gi pådrag
  - `WATCHDOG_DEVCTL_STOP` - stopper tråder og timere samt skrur av mulighet for å gi pådrag.
  - `WATCHDOG_DEVCTL_UPDATE` - restter timer
- `timer_traad()` - er en tråd som venter på mottak av signal fra timer som sjekker om det er liv i RTW og en timer som teller en forhåndsbestemt tid(0.5s) hvor den da sjekker `vinkel_0` mot hva den var forrige gang. Denne tråden har mulighet til å drepe RTW

Tillegg N

Innhold på cd

Vedlagt følger en cd med alle nødvendige filer for systemet, samt videoer av oppsving og balansering. Under følger en punktliste med mappene og deres innhold.

- Hardware - inneholder filer som har med kretskortdesign å gjøre.
- Hjelpefiler - inneholder datablader og annen nyttig tekst.
- Matlab - inneholder all kode som er nødvendig for `Matlab`, inkludert eksempler for parameterestimering.
- Mekanikk - inneholder CAD-tegninger for det fysiske systemet.
- QNX - inneholder programvare utviklet for QNX.
- Rapport - inneholder denne rapporten.
- Software - inneholder programvare for mikrokontrollerene.
- Video - inneholder video av systemet.