NTNU
Det skapende universitet

Modellering og Regulering av Trippelinvertert Sirkulær Pendel

Sindre Hansen

Master i kybernetikk og robotikk
Innlevert: juni 2015
Hovedveileder: Amund Skavhaug, ITK

Norges teknisk-naturvitenskapelige universitet
Institutt for teknisk kybernetikk

Modellering og Regulering av Trippelinvertert Sirkulær Pendel

Sindre Hansen

15. juni 2015

Oppgavetekst

Institutt for teknisk kybernetikk har ønske om å legge inn styring til et trippelinvertert sirkulært pendelsystem. Arbeid på systemet ble startet opp av Amund Skavhaug i 1989. Siden den gang har flere prosjekt blitt utført og systemet har opplevd gradvis oppgradering, men det har hittil ikke blitt funnet noen endelig løsning for å balansere alle tre pendler. Oppgavens hensikt er å finne en god modell for systemet slik at det kan utvikles modellbasert regulator til balansering av pendler.

Oppgaven består av:

Sette seg inn i nødvendig teori og tidligere arbeid

Utlede en god matematisk modell

Finne systemparametrene

Utvikle regulator til balansering av pendel

Forord

Denne rapporten omhandler arbeid utført med diplomoppgave ved institutt for teknisk kybernetikk. Oppgaven går ut på å balansere pendler i et trippelinvertert pendelsystem. Systemet ble opprettet i 1989 av Amund Skavhaug. Det er siden den gang blitt utført en rekke prosjekter på systemet, men hittil har det ikke blitt funnet noen endelig løsning for å balansere alle tre pendlene. I 2006 ble det utført et prosjekt [10] som fokuserte på å utlede bevegelsesligningene til pendelsystemet. Ettersom alle prosjekt siden den gang har basert seg på disse ligningene var det ønskelig å utlede bevegelsesligningene på nytt ved å benytte en annen metode. I [13] ble det utført arbeid med å regulere det virkelige systemet, men på grunn av slingring, vibrasjon og lav enkoderoppløsning ble reguleringen vanskelig. På systemet er det nå mindre slingring og det er montert enkodere med høyere oppløsning.
Under prosjektet var det i utgangspunktet ønskelig å forsøke balansering av pendler på det virkelig systemet, men på grunn av en hardware feil var det kun mulig å måle vinkel på en av pendlene om gangen. Å skulle reparere feilen lå utenfor oppgavens omfang. Siden det fortsatt var mulig å hente inn nødvendig data for å modellere systemet ble det besluttet å fokusere på utvikling av en nøyaktig modell hvor regulatoren kunne implementeres. På denne måten var det fortsatt mulig å forholde seg til målene i oppgaven.
For å estimere modellparametrene ble det i [13] benyttet et ulineært kalmanfilterble. Denne metoden var begrenset i at den krevde derivasjon av bevegelsesligningene til motoren. Motoren er derimot påvirket av ulineær coulmbfriksjon som ikke kan deriveres. Metoden ble dermed noe unøyaktig. I dette prosjektet er modellparametrene i stedet estimert ved hjelp av en genetisk algoritme som baserer seg på biologisk evolusjon. Denne algoritmen ble til slutt også benyttet for å finne regulatorparametre.
I forbindelse med oppgaven vil jeg takke

- Amund Skavhaug for veiledning
- Verkstedet ved Terje Haugen
- Stefano Bertelli for bistand til feilsøking av elektronikken

Sammendrag

Denne rapporten tar for seg arbeid utført i forbindelse med diplomoppgave ved NTNU institutt for teknisk kybernetikk. Oppgaven omhandler modellering og stabilisering av et tippelinvertert pendelsystem. Under prosjektet ble bevegelsesligningene for systemet utledet. For å finne parametrene i den ulineære servomodellen ble det benyttet en metode som kombinerer genetisk evolusjons-algoritme med Nelder-Mead simplexmetoden. Resultatene viser at metoden fungerer ganske bra og demonstrerer at metoden kan benyttes på ulineære systemer der andre metoder feiler. det ble implementert LQ-regulator til balansering av enkeltpendel. Som alternativ til LQR ble den genetiske algoritmen også benyttet til å finne regulatorparametre. Metoden ga like gode resultat som med LQR, men uten behov for linearisering av bevegelsesligningene.

Abstract

This report documents the work that has been perform in regards to a Masterthesis at the Norwegian University of Science and Technology, department of cybernetics. The thesis is about modelling and control of a triple inverted pundulum system. The equations of motion was derived using Newton-Euler's method. The nonlinear servomotor parameters were estimated using a method that combines a genetic evolutionary algorithm with the Nelder-Mead algorithm. The results were rather promising and serves to demonstrate that the estimation scheme can be used for estimating parameters in systems where other methods might fail. A Linear Quadratic Regulator was implemented for stabilization of the single pendulum system. The genetic algorithm was also used for finding control parameters as an alternative to the LQR. The controller that was found with the genetic algorithm yielded results that were as good as with the LQR, but the method did not require derivation of the linearized system equations.

Innhold

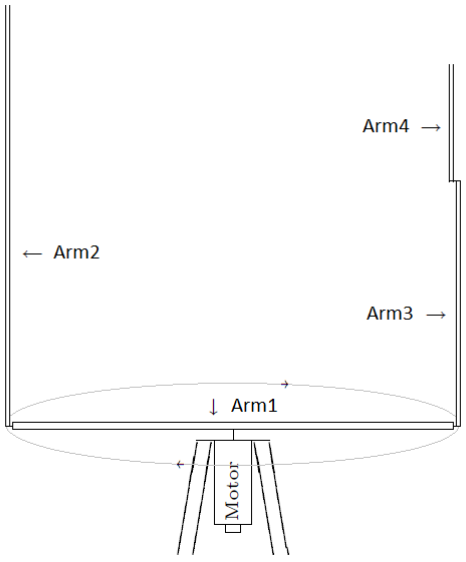
Figurer

Tabeller

Kapittel 1

Introduksjon

Ved NTNU's institutt for teknisk kybernetikk har det blitt utført en rekke prosjekter med fokus på utvikling av et fungerende trippelinvertert pendelsystem. Systemet ønskes å brukes som et «showcase»-objekt for å lokke nye studenter til linjen, samt motivere eksisterende studenter. Et invertert pendelsystem er et reguleringsproblem der en pendel forsøkes å balanseres opp ned. Inverterte pendler har en komplisert, ulineær og ustabil dynamikk. Allikevel er det mulig å finne en ganske nøyaktig matematisk beskrivelse av dynamikken. Av den grunn egner systemene seg ganske bra til testing og utvikling av reguleringsstrategier. Det finnes mange variasjoner av inverterte pendelsystemer, men felles for alle systemtypene er at de er underaktuerte. Dette vil si at de har flere frihetsgrader enn de har aktuatorer. Av den grunn er inverterte pendelsystemer generelt sett utfordrende å kontrollere. Oppgaven i prosjektet omhandler et sirkulært pendelsystem der pendlene er festet til hver ende av en horisontal arm. Den horisontale armen roteres av en motor der målet er å rotere armen slik at pendlene svinger opp til invertert posisjon hvor de så må balanseres.



Figur 1.1: Trippelinvertert sirkulært pendelsystem

I tidligere utførte prosjekter har det blitt implementert regulator til balansering av enkeltpendel, men med mangel på repeterbare resultater for balansering av flere pendler samtidig. Mange av de tidligere utfordringene skyldes teknologiske begrensinger i blant annet instrumentering, kommunikasjon og datakraft. Nå ønskes det å benytte den moderne teknologien til å finne en mer nøyaktig systemmodell og for å oppnå bedre regulering slik at det på sikt kan være mulig å stabilisere alle tre pendlene samtidig.

1.1 Tidligere arbeid

Arbeid på pendelsystemet ble først startet opp i 1989 av Amund Skavhaug. Systemet har gjennomgått en del forandring siden den gang, men det finnes fortsatt uforandrede komponenter, slik som servomotoren.

I [10] ble bevegelsesligningene for pendlene utledet ved å bruke Lagrange metode som tar utgangspunkt i energibevaring. Selv om ligningene gir et godt bilde av dynamikken, har det blitt gjort noen forenklinger under utledningen som muligens kan gjør modellen noe unøyaktig. Simulatoren som ble utviklet i [10] har også andre unøyaktigheter. Her ble ikke motormodell beskrevet. pådraget fra regulatoren sendes i stedet rett inn på pendelligningene som påtrykt moment. Simulatoren blir dermed en unøyaktig representasjon. I [13] ble motoren modellert og simulatoren fra [10] ble forbedret.

I [13] ble det implementert en regulator som kunne balansere enkeltpendler. Det ble også utført balansering av flere pendler ved hjelp av LQR, men det rapporteres at balanseringen av flere pendler bare fungerte av og til. Dårlige tilstandsestimat, slingring og vibrasjoner i systemet klandres for problemene med å balansere flere pendler. Siden den gang har slingringen blitt redusert. Det forklares i [13] at dårlige tilstandsestimat skyldes for lav oppløsning i enkoderne som måler pendelvinklene. Enkoderoppløsningen var den gangen en teknologisk begrensning som ikke kunne forbedres siden det ikke fantes passende enkodere med høyere oppløsning enn de som allerede var montert. Siden den gang har det blitt anskaffet enkodere med høyere oppløsning. Enkoderoppløsningen er nå dobbelt så stor som i [13].

I [8] ble det gjort forsøk på å implementere **M**odel **P**redictive **C**ontrol til å stabilisere pendlene som alternativ til LQR. Det ble konkludert med at MPC er utilstrekkelig som reguleringstrategi til stabilisering av pendlene. Det nevnes at en ulineær MPC kanskje vil kunne fungere.

I [5] ble det skrevet nytt programvare for datainnsamling og serieportkommunikasjon mot PC. For å styre systemet fra datamaskin ble det tidligere benyttet en ekstra maskin med sanntidsoperativsystemet QNX. I [4] ble Simulink-modulen, **R**eal-**T**ime **W**indows **T**arget benyttet til å styre systemet i sanntid direkte fra Windows. Det ble satt opp en modell i Simulink-RTWT som behandler og overfører data til og fra systemet med den nye kommunikasjonsprotokollen som ble implementert i [5]

1.2 Mål

Det overordnede målet i prosjektet er å implementere regulator til balansering av invertert pendel. For å oppnå dette ønskes det en god systemmodell. Til dette formålet, og for eget læringsutbytte, er det aktuelt å regne ut bevegelsesligningene på nytt ved å benytte en annen metode enn den som har blitt brukt i tidligere utledning.

Siden systemet har flere ulineære komponenter med ukjente parametre er det nødvendig med en god parameterestimeringsstrategi dersom det skal finnes en god systemmodell.

Det ønskes å gjenskape tidligere resultat med å benytte LQR til balansering av pendel. Det ønskes også å undersøke andre strategier for regulering som alternativ til LQR.

Dersom tiden tillater ønskes det å implementere balanseringsregulatoren fra mikrokontrolleren

1.3 Omjustering av oppgaven

Et par måneder ut i prosjektet ble det oppdaget en feil som hadde oppstått i datainnsamlingselektronikken slik at vinkelmålingen fra den ene pendelen ble ubrukelig. Det ble forsøkt å feilsøke elektronikken, men på grunn av manglende dokumentasjon ble denne oppgaven tidskrevende. I [5] ble det utført arbeid med datainnsamling som omhandlet programmering av mikrokontroller og retting av loddefeil, men det er uvisst hvem som faktisk har laget kretskortet som det er feil på. Arbeidet i [5] ble utført slik at det i dette prosjektet skulle kunne fokuseres på modellering og regulering av systemet. Å skulle lage ny elektronikk lå utenfor oppgavens omfang.

Ved å bytte om på inngangene til innsamlingelektronikken var det fortsatt mulig å lese vinkelen til alle pendlene, men kun for en pendel av gangen. Av den grunn var det fortsatt mulig å hente inn nødvendig data til å modellere systemet. For å holde prosjektet mest mulig til målene ble det besluttet å fokusere på utvikling av en treffsikker simulator hvor regulator kan implementeres.

1.4 Rapportdisposisjon

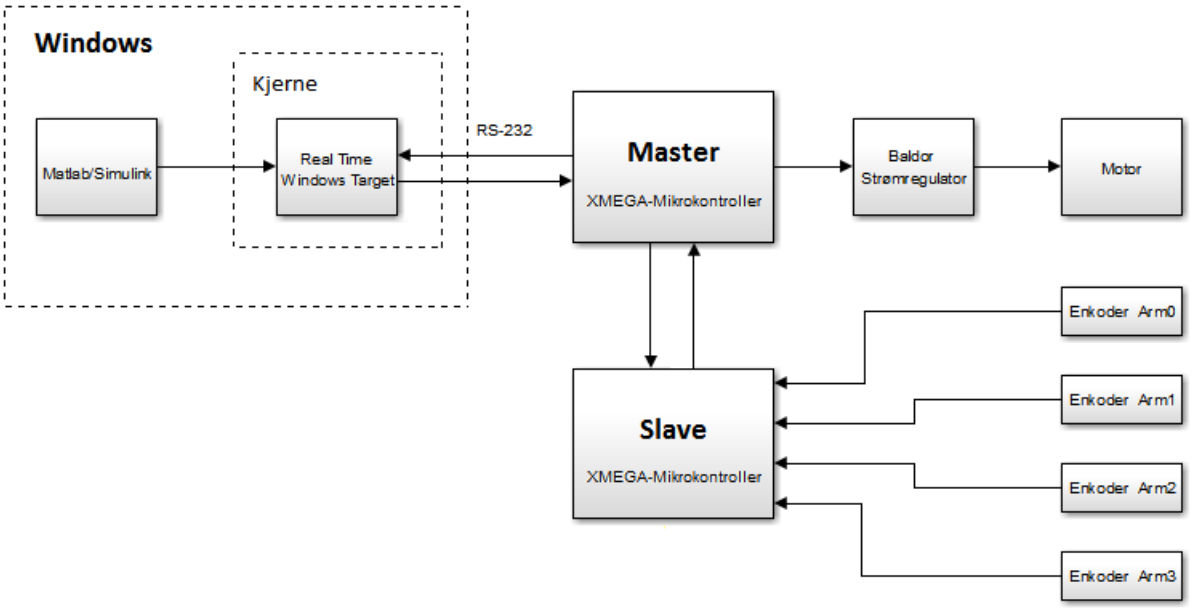
- Kapittel 2 - Beskrivelse av system, hardware og software
- Kapittel 3 - Presentasjon av nødvendig teori for arbeidet som er utført
- Kapittel 4 - Systemmodellering. Utleding av bevegelsesligningene, modellering av friksjon, ulineær motordynamikk og servo
- Kapittel 5 - Feilsøking av målefeil i pendelvinkel
- Kapittel 6 - Identifikasjon av pendelfriksjon og implementering av parameterestimering med genetisk algoritme

- Kapittel 7 - Implementer av regulator
- Kapittel 8 - Resultater og diskusjon
- Kapittel 9 - Konklusjon
- Kapittel 10 - Videre arbeid
- Vedlegg...

Kapittel 2

Systembeskrivelse

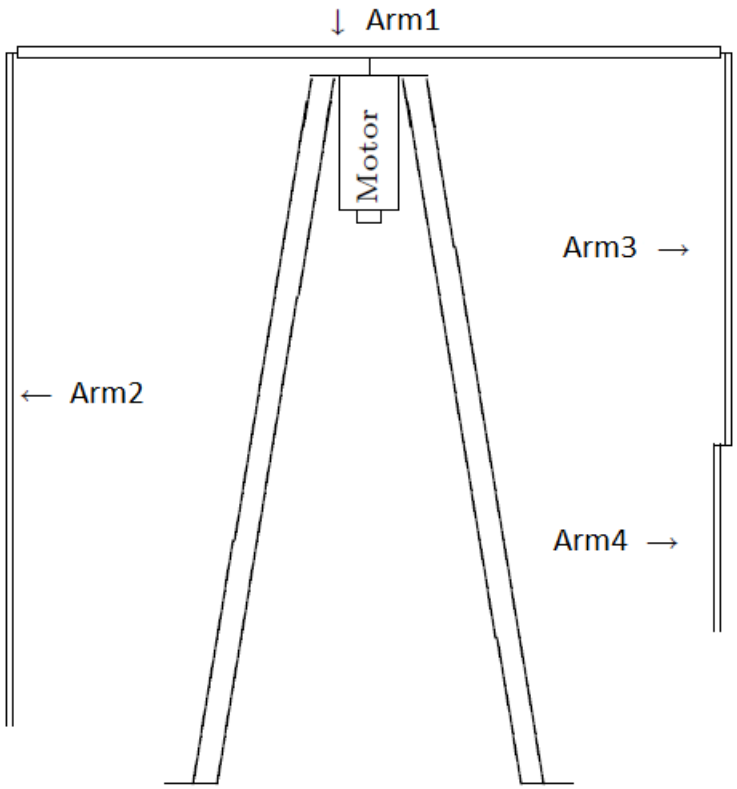
Pendelsystemet består av en horisontal arm som på midten er koblet til en likestrømsmotor. På hver ende av den horisontale armen er det festet pendler som kan rotere fritt. Pendlenes vinkel blir målt av optiske enkodere. Enkodersignalene mottas av elektronikk for datainnsamling som overfører dataen videre til hovedmikrokontrolleren. Denne mikrokontrolleren sender pådragsignal til servosystemet som regulerer strømmen i motoren slik at systemet kan styres. En datamaskin kan benyttes til å kommunisere med hovedmikrokontrolleren over serieport. Fra datamaskinen kjøres Simulink-modulen, Real-Time Windows Target, som benytter Windows-kjernen til å kjøre Simulink-modeller i sanntid. Regulator for å svinge opp og balansere pendlene kan da implementeres fra Simulink eller direkte fra mikrokontrolleren.



Figur 2.1: Kommunikasjonsflyt for pendelsystem

2.1 Pendlene

Den horisontale staven, Arm1, er festet til en motor slik at den vil rotere når motoren roterer. Armen er festet direkte til motor uten noe gir imellom slik at vinkelhastigheten til Arm1 vil være den samme som vinkelhastigheten til motoren. På hver ende av Arm1 er det festet en pendel som kan roterer fritt. Pendlene er dimensjonert til forskjellige lengder slik at systemet skal være kontrollerbart. På enden av den korte pendelen, Arm3, er det festet enda en pendel, Arm4. Lengden til Arm4 er dimensjonert slik at den totale lengden til Arm3 og Arm4 ikke blir den samme som lengden til pendelen, Arm2, som er festet på den andre enden av Arm1. Ved oppstart av prosjektet mangler det festeoppheng til Arm4.

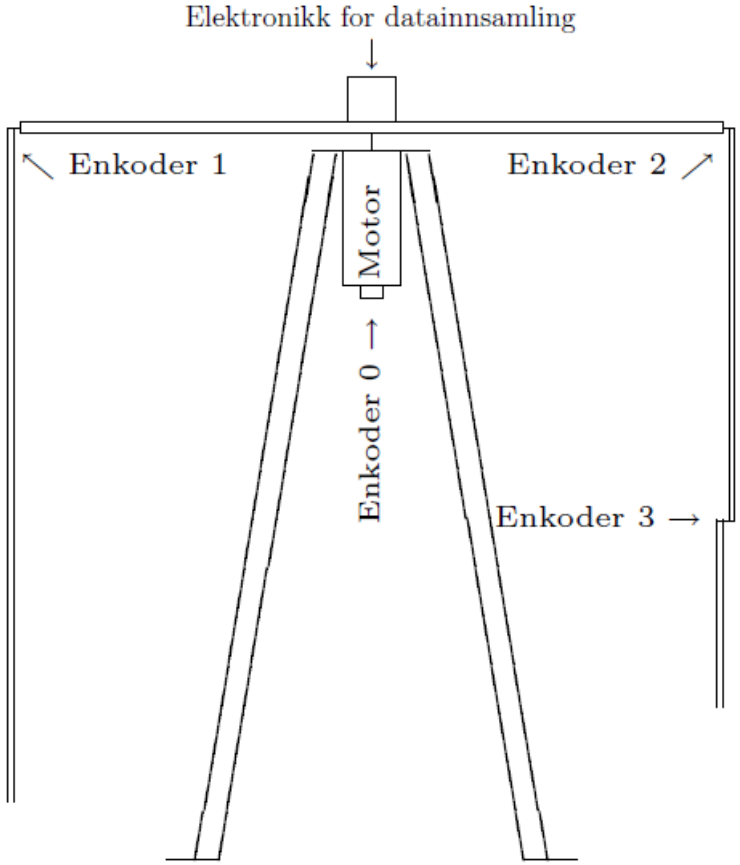


Figur 2.2: Pendelsystem

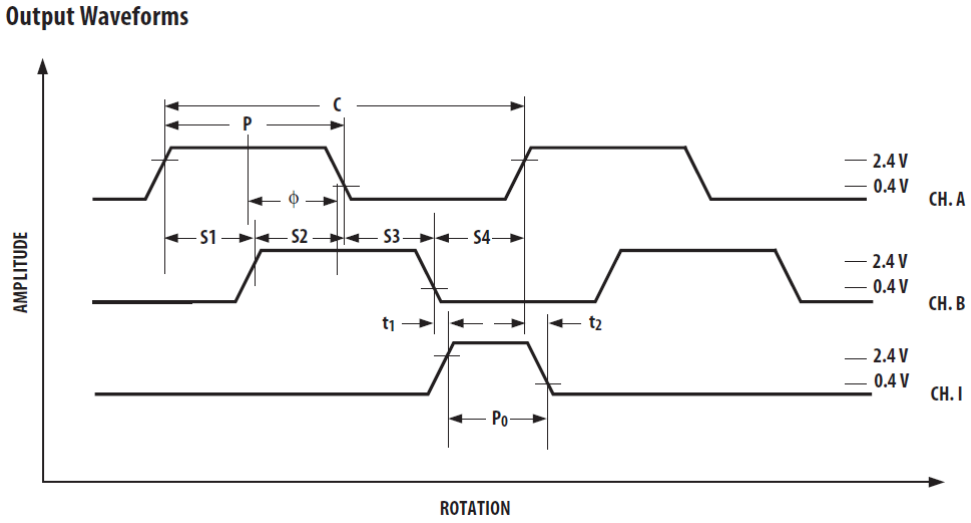
2.2 Instrumentering og datainnsamling

Ved hver arm er det montert optiske inkrementell-enkodere av typen Avago HEDM 5540 som har en oppløsning på 1024 punkt per omdreining. Når pendlene roterer sender enkoderne ut pulstog over to forskjellige kanaler. De to pulstogene står 90° faseforskjøvet i forhold til hverandre slik at retningen på rotasjonen kan bestemmes. I tillegg vil kombinasjonen av høy og lav på de to kanalene kunne benyttes til å øke den effektive oppløsningen med 4 ganger oppgitt oppløsning. Figur 2.4 illustrerer dette konseptet hvor områdene S1, S2, S3 og S4 kan brukes til å registrere 4 punkt for hver puls. I figuren gir kanal I ut en

puls for hver omdreining, men denne kanalen benyttes ikke i pendelsystemet.



Figur 2.3: Enkoderplassering



Figur 2.4: Enkoderkanalene

Signalene fra enkoderne mottas av elektronikk for datainnsamling som er festet på toppen av Arm1. På dette kretskortet sitter en AVR XMEGA mikrokontroller som prosesserer enkoder-dataen og estimerer vinkelhastighetene. mikrokontrolleren fungerer som en slave og overfører dataen videre til master-mikrokontrolleren som sitter i elektronikk-kabinettet som står plassert under systemet.

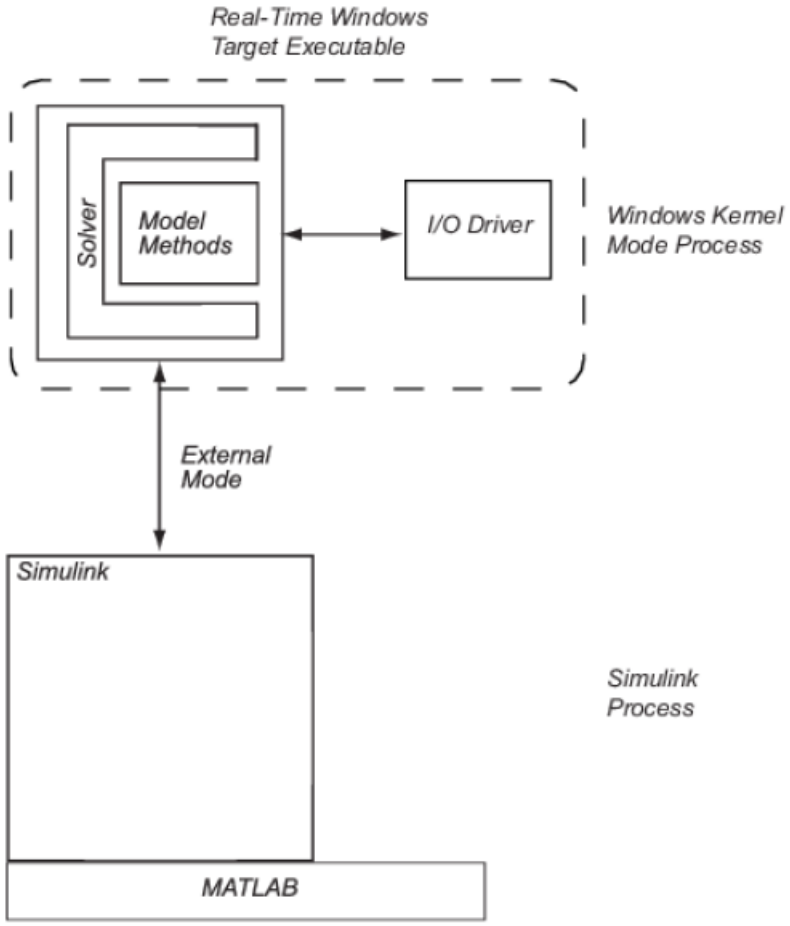
2.3 Servomotor

DCmotoren styres fra et regulatorkort[1]. Regulatorkortet ble utgitt i 1989 og kan stilles inn ved hjelp av en rekke potmeterer. Kortet mottar en referanseverdi fra master-mikrokontrolleren og regulerer vinkelhastigheten til motoren. Utgangen fra PI hastighets-regulatoren brukes som referanse til en PI-strømregulator. Utgangen fra denne strømregulatoren påtrykkes motoren via en likeretterbro. I tidligere prosjekt på systemet ble hastighetsregulatoren koblet ut fordi den ikke fungerte tilfredsstillende. Dermed benyttes det kun strømregulering hvor inngangen til kortet brukes som strømreferanse.

2.4 Real-Time Windows Target

Regulator som utvikles i Matlab/Simulink kan implementeres i sanntid på systemet ved å benytte modulen **R**eal-**T**ime **W**indows **T**arget i kombinasjon med Simulink-coder for kodegenerering. RTWT genererer kode for å kjøre modellalgoritmen som en ekstern prosess i Windows kjernen. Modellen blir da gitt en høy prioritet slik at den rekker å bli ferdig med modelloppdateringer innen tidsfristene. Data fra den eksterne prosessen sendes så tilbake til Matlab.

[1] Baldor TSNM



Figur 2.5: Illustrasjon av Real Time Windows Target

Kapittel 3

Teori

I denne delen presenteres nødvendig teori til arbeidet som har blitt utført. Det blir presentert teori som er brukt under modellering og regulering. Parameterestimeringsmetoden blir forklart. Metoden benytter seg av sorteringsalgoritmen mergesort. Denne Sorteringsalgoritmen blir derfor også forklart.

3.1 Modelleringsteori

3.1.1 Rotasjonsmatriser

Rotasjonsmatriser brukes til å utføre rotasjon matematisk. Rotasjonsmatriser karakteriseres som ortogonale. Dette vil si at for rotasjonsmatrisen R så er $R^T = R^{-1}$. I tillegg til dette har alle rotasjonsmatriser determinant lik 1. Det finnes i hovedsak to forskjellige konvensjoner. Rotasjon av objekt og rotasjon av koordinatsystem.

Rotasjon av objekt utføres i forhold til et fast koordinatsystem. objektet kan beskrives som et punkt eller eventuelt en vektor. Figur 3.1 illustrerer rotasjon av vektor.

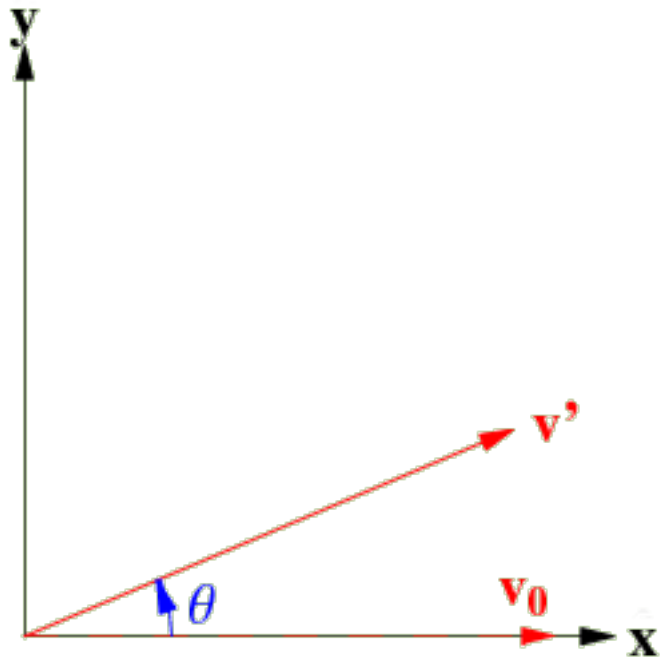
v_0 beskriver vektoren ved 0 grader.

$$v' = Rv_0 \tag{3.1}$$

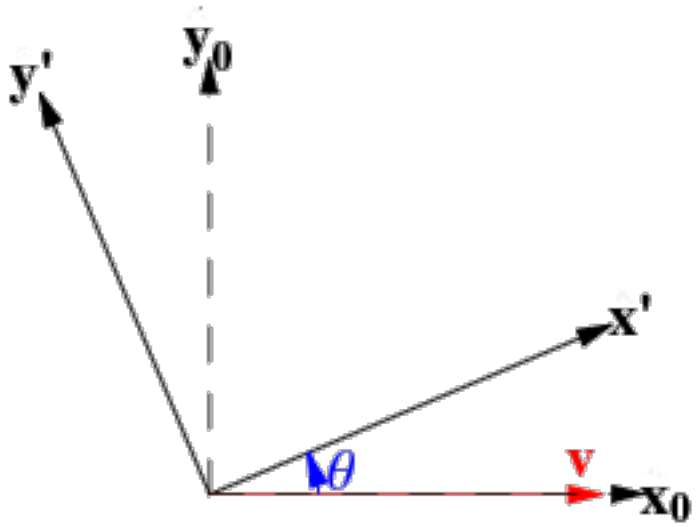
der

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{3.2}$$

Denne rotasjonsmatrisen kan altså brukes til å rotere vektor eller punkt der positiv vinkel går mot klokka. En annen konvensjon er å rotere selve koordinatsystemet slik at det oppnås et relativt koordinatsystem rotert mot klokka i forhold til grunnkoordinatsystemet. Hvis det så ønskes å kartlegge en vektor til det relative koordinatsystemet må det benyttes en transponert versjon av rotasjonmatrisen over. Fra det relative koordinatsystemet vil dette være ekvivalent til å rotere vektoren med klokka.



Figur 3.1: Rotasjon av vektor



Figur 3.2: Rotasjon av koordinataksene

Rotasjonen kan her beskrives ved

$$v' = Rv \tag{3.3}$$

der

$$R = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \tag{3.4}$$

Her brukes altså rotasjonsmatrisen til å kartlegge vektoren til det roterte koordinatsystemet.

Sett fra 3 dimensjoner foregår denne rotasjonen rundt z-aksen. Matrisen kan da utvides slik at

$$R_z = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.5a}$$

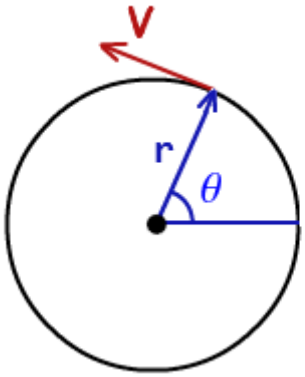
På likt vis kan rotasjon om de to andre aksene beskrives med følgende matriser

$$R_y = \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{3.5b}$$

$$R_z = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) \end{bmatrix} \tag{3.5c}$$

3.1.2 Sentripetal og tangentiell akselerasjon

Når en gjenstand roterer med variabel vinkelhastighet kan akselerasjonen deles inn i to komponenter. Den ene er sentripetalakselerasjonen som virker inn mot sentrum av rotasjonen. Den andre er en tangentiell akselerasjon som forårsaker forandring i banehastigheten.



Figur 3.3: Hastighet under rotasjon

Den lineære hastigheten kan uttrykkes som

$$\boldsymbol{v} = \frac{\mathrm{d}\boldsymbol{r}}{\mathrm{d}t} = \boldsymbol{\omega} \times \boldsymbol{r} \tag{3.6}$$

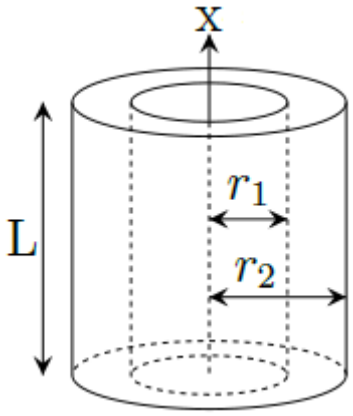
der $\boldsymbol{\omega}$ er vinkelhastigheten. akselerasjonen kan da utrykkes som

$$\boldsymbol{a} = \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} = \frac{\mathrm{d}}{\mathrm{d}t}(\boldsymbol{\omega} \times \boldsymbol{r}) = \frac{\mathrm{d}\boldsymbol{\omega}}{\mathrm{d}t} \times \boldsymbol{r} + \boldsymbol{\omega} \times \frac{\mathrm{d}\boldsymbol{r}}{\mathrm{d}t} = \dot{\boldsymbol{\omega}} \times \boldsymbol{r} + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \boldsymbol{r}) \tag{3.7}$$

Det første leddet representerer sentripetalakselerasjonen. Det andre leddet representerer den tangentielle akselerasjonen.

3.1.3 Treghetsmoment

Treghetsmoment for hul stav med uniformt fordelt masse



Figur 3.4: Hul sylinder

For en hul stav med uniformt fordelt masse vil massetettheten være

$$\rho = \frac{m}{V} = \frac{m}{\pi(r_2^2 - r_1^2)L} \tag{3.8}$$

Der m er massen. Matrisen for treghetsmomentet kan da finnes ved integrasjon

$$I = \int_V \rho(x,y,z) \begin{bmatrix} y^2+z^2 & -xy & -xz \\ -xy & x^2+z^2 & -yz \\ -xz & -yz & x^2+y^2 \end{bmatrix} dzdydx \tag{3.9}$$

$$= \begin{bmatrix} \frac{1}{2}m(r_1^2+r_2^2) & 0 & 0 \\ 0 & \frac{1}{12}mL^2 + \frac{1}{4}m(r_1^2+r_2^2) & 0 \\ 0 & 0 & \frac{1}{12}mL^2 + \frac{1}{4}m(r_1^2+r_2^2) \end{bmatrix} \tag{3.10}$$

Dersom tykkelsen og radiusen til staven anses som ubetydelig ($r_1 \to r_2$, $r_2 \to 0$) kan matrisen for treghetsmoment forenkles til

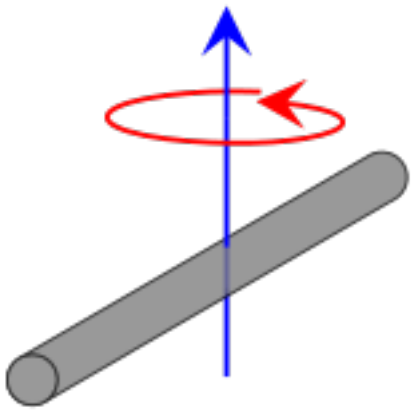
$$I = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \frac{mL^2}{12} & 0 \\ 0 & 0 & \frac{mL^2}{12} \end{bmatrix} \tag{3.11}$$

Massetettheten kan uttrykkes ved

$$\rho = \frac{m}{AL} \tag{3.12}$$

Dersom staven roteres rundt midten kan treghetsmomentet uttrykkes ved

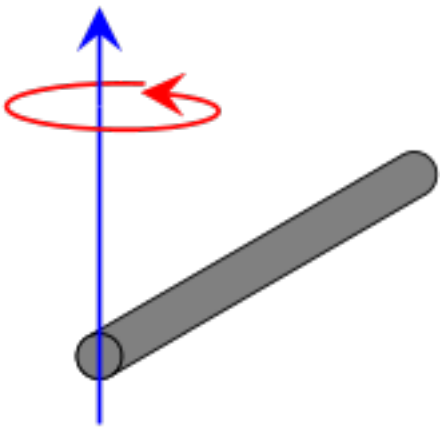
$$I = A\rho \int_{-L/2}^{L/2} x^2 dx = \frac{1}{12}mL^2 \tag{3.13}$$



Figur 3.5: Stang som roteres rundt midten

Dersom staven roteres rundt den ene enden kan treghetsmomentet uttrykkes ved

$$I = A\rho \int_0^L x^2 dx = \frac{1}{3} mL^2 \tag{3.14}$$



Figur 3.6: Stang som roteres rundt den ene enden

Treghetsmoment for pendel med ikke-uniform massefordeling

Dersom en pendel har ikke-uniform og ukjent massefordeling kan treghetsmomentet finnes ved å benytte den naturlige frekvensen til pendelens svingning. Bevegelsen til en pendel kan uttrykkes ved Differensiallikningen

$$J\ddot{\theta} = -mgl \sin\theta \tag{3.15}$$

der J er treghetsmomentet, m er massen og l er lengden til pendelen. $\sin\theta$ kan uttrykkes som en maclaurinrekke slik at

$$\sin\theta = \theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \frac{\theta^7}{7!}... \tag{3.16}$$

Fra denne rekken kan man tydelig se at $\sin\theta \approx \theta$ ved små vinkler ettersom at de andre leddene i rekken blir ubetydelig små. f.eks. dersom man har en vinkel $\theta = 0.1rad$ så vil $\sin\theta = 0.0998$. Ved små vinkler kan differensiallikning altså approksimeres slik at

$$\ddot{\theta} \approx -\frac{mgl}{J}\theta \tag{3.17}$$

Løsningen for denne type differensiallikning tar formen

$$\theta = c_1 \sin(\sqrt{\frac{mgl}{J}}t) + c_2 \cos(\sqrt{\frac{mgl}{J}}t) \tag{3.18}$$

Fra denne løsningen kan man se at frekvensen er gitt ved

$$\omega_N = \sqrt{\frac{mgl}{J}} \tag{3.19}$$

Den naturlige frekvensen, ω_N, defineres her som frekvensen til pendelens svingninger ved små vinkler. Ved større vinkler vil approksimasjonen bli unøyaktig og uttrykket for ω_N vil være ugyldig. Dersom man uttrykker frekvensen ved hjelp av periodetid og snur på likningen får man

$$J = \frac{mglT^2}{4\pi^2} \tag{3.20}$$

Den naturlige periodetiden kan enkelt finnes ved å måle gjennomsnittlig periodetid på svingningene til pendelen ved små vinkler.

3.2 Parameterestimering med genetisk algoritme og Nelder-Mead

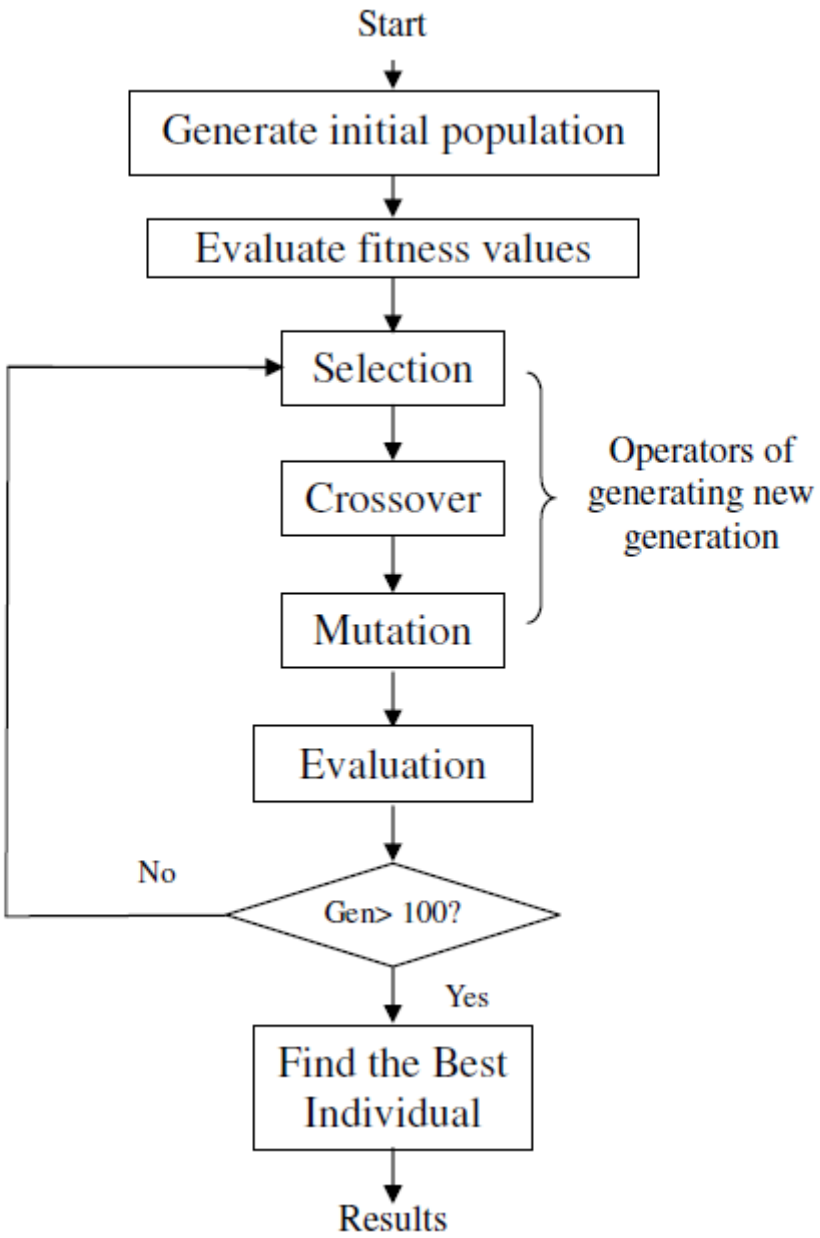
En genetisk algoritme er en form for stokastisk søkealgoritme som baserer seg på evolusjonsteori fra biologien. Ideen om å benytte en datamaskin til å etterligne evolusjon ble for første gang foreslått av Alan Turing i [11]. Siden den gang har det blitt utviklet flere typer algoritmer som baserer seg på evolusjon. Genetiske algoritmer ble først foreslått av Holland [6] og skiller seg fra andre biologiske algoritmer ved at den etterligner genspleising. Algoritmen har flere anvendelsesområder og gjør seg spesielt nyttig under optimaliseringsproblemer hvor det finnes flere lokale optimale løsninger. Dette fordi algoritmen har en utmerket evne til å finne området rundt den globale optimale løsningen uten å sette seg fast i en relativt dårlig lokal løsning. I [12] beskrives det hvordan algoritmen kan brukes til å estimere parameter i ulineære systemer. Resultatene som legges frem i artikkelen viser at metoden fungerer veldig bra til parameterestimering under situasjoner der andre metoder feiler på grunn av ulinearitetet i modellen. I masteroppgaven fra 2007 som omhandler pendelsystemet [13] var det nettopp ulineæritetene i motormodellen som gjorde parameterestimeringen av DC-motoren mislykket. Av den grunn ble det gjort forenklinger i motormodellen slik at hele rotasjonssløyfen av motordynamikken ble fjernet fra modellen. Til tross for at genetiske algoritmer er veldig gode på å finne området rundt den globale optimale løsningen, er de elendige på å faktisk konvergere til det optimalt punktet. Som en løsning på dette problemet blir det i [3] benyttet en kombinasjon av genetisk algoritme og Nelder-Mead simplex-algoritmen til identifikasjon av parametere for en ulineær DC-motormodell. I motsetning til genetiske algoritmer har simplex-metoden en veldig god evne til å raskt konvergere til den nærmeste lokale løsningen. Dette er også metoden sin svakhet da den har vanskelighet for å finne den globale løsningen når det finnes flere lokale løsninger. Den genetiske algoritmen blir derfor først brukt til å finne området i nærheten av den globale løsningen. Resultatet fra den genetiske algoritmen blir deretter brukt som initialverdi for simplex-algoritmen som så konvergerer til den globale optimale løsningen. Resultatene som legges frem i artikkelen ser meget lovende ut og det konkluderes med at metoden både er nøyaktig og effektiv. En av de største ulempene ved bruk av genetiske algoritmer er at det teoretisk sett ikke kan garanteres på noen som helst måte at metoden faktisk kommer til å finne nærområdet til den globale løsning, men i praksis ser metoden ut til å fungere meget bra. En annen ulempe er at det må utføres ganske mange simuleringer med forskjellige parametersett. På grunn av dette egner algoritmen seg kun til offline-estimering. Det finnes heller ingen god beskrivelse eller metode for hvordan algoritmen sine parametere skal velges. Disse må velges ut ifra generelle anbefalinger.

3.2.1 Beskrivelse av den genetiske algoritmen

En genetisk algoritme består i hovedsak av 4 oppgaver som gjentas inntil algoritmen avsluttes. Først blir befolkningen av kromosomer[1] evaluert og rangert etter hvor bra de passer inn. Deretter blir et visst antall av befolkningen valgt ut, med preferanse på de best rangerte, til å produsere neste generasjon. Par blir satt sammen til å danne variasjon

[1] kodesekvens som utgjør et individ

og allikevel opprettholde gener[2] gjennom en krysningsprosess. Deretter utføres tilfeldig mutasjon for å muliggjøre opprettelsen av ny informasjon. Prosessen gjentas så med den nye generasjonen som blir produsert.



Figur 3.7: Flytskjema over en genetisk algoritme

Koding For at algoritmen og de forskjellige mekanismene skal ha noen som helst betydning må det defineres en form for kode som algoritmen har i oppgave å prosessere. Et biologisk DNA beskrives med 4 forskjellige typ baser. I et menneskeskapt datasystem er

[2]stykkevis informasjon

det mer fornuftig og anvende binærtall for å kode informasjonen. Under parameterestimering kan man se på hvert parametersett som et kromosom og hver av parametrene i settet som et gen. Antall bit som kreves for å beskrive hver parameter kan finnes ved

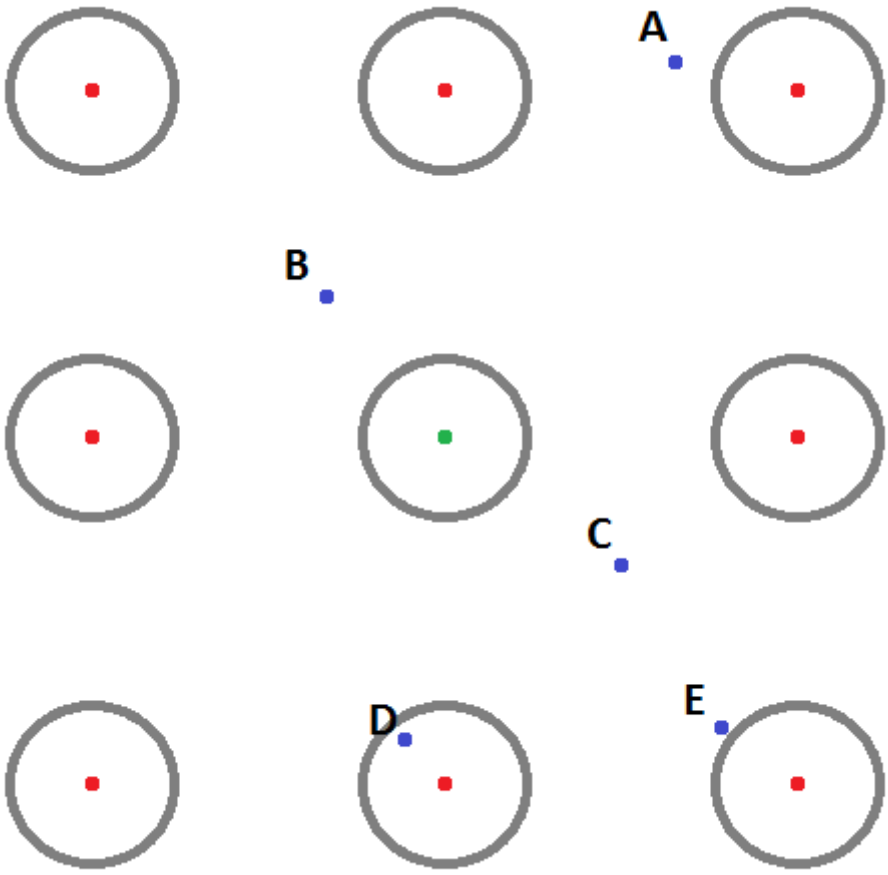
$$\text{Lengde}(K) = \text{ceil}(\log_2(K_{max} \cdot 10^{\delta})) \tag{3.21}$$

hvor δ er presisjon i antall desimaler og K_{max} er den maksimale verdien som det tillates at parameteren kan ta.

Initialisering Før evolusjonsprosessen kan iverksettes må det skapes en initialbefolkning. Ved parameterestimering blir det opprettet M-antall sett med parametere som genereres stokastisk innenfor optimaliseringsområdet.

Evaluering Under parameterestimering er evalueringen av kromosomer den mest krevende delen av algoritmen. I denne delen må hvert sett med parametere simuleres og resultatene av hver simulering evalueres etter hvor bra parametrene passer inn. Det må da defineres en såkalt fitness-funksjon som tildeler hver kromosom et tall. Ved parameterestimering er denne funksjonen gjerne avhengig av det akkumulerte avviket mellom responsen til simuleringen og responsen til systemet som skal identifiseres. Til utvelgelsesoppgaven kan det her også være nyttig å rangere alle kromosomene relativt til hverandre fra best til dårligst. Til denne rangeringen benyttes en sorteringsalgoritme.

Utvalg Når Kromosomene skal velges ut til reproduksjon kan det kanskje være naturlig å tenke at man velger ut de D-antall beste kromosomene og forkaster resten, men dersom man gjør dette kan man faktisk ende opp med å ødelegge effektiviteten og nøyaktigheten til algoritmen. Figur 3.8 illustrerer et hypotetisk optimaliseringsområde der den grønne prikken i midten er den globalt optimale løsningen. De røde prikkene rundt er lokale optimale løsninger. De blå prikkene er tilfeldig genererte parametersett klare for utvalg til neste generasjons parametersett. Dersom man bare velger de beste verdiene til reproduksjon vil algoritmen velge D og E mens B og C vil bli forkastet selv om disse i realiteten er nærmere den globalt optimale løsningen. Den neste generasjonen vil dermed inneholde parametere nærmere de lokale løsningene og lenger unna den globale løsningen. Dette vil da kunne føre til at algoritmen låser seg inn på en lokal løsning og hele poenget med å bruke en genetisk algoritme er borte. I stedet for å garantere at den beste løsningene blir valgt og at de dårligste blir forkastet, ønsker man heller at det er en viss mulighet for at et hvert parametersett kan bli valgt til reproduksjon. For at algoritmen skal kunne ha noen progresjon må det fortsatt være større sannsynlighet for å velge ut de beste løsningene enn de dårlige. For å velge ut kromosomer benyttes en funksjon som tildeler en sannsynlighet P_i til hver kromosom. en sekvens med M antall tilfeldige tall $U_i(0,1)$ blir så generert og sammenlignet med den kumulative sannsynligheten $C_j = \sum_{i=1}^{j} p_i$. kromosom i blir valgt dersom $C_{j-1} < U_i(0,1) \leq C_i$. Algoritmen kan implementeres til å skape en variabel befolkning, men kan også implementeres til å holde befolkningen konstant. Når



Figur 3.8: Illustrasjon av hypotetisk optimaliseringsområde. Grønn prikk er global optimum, Røde prikker er lokale optimum, Grå sirkler er indikerer nærområdene til optimumene. A,B,C,D og E illustrerer forskjellige kromosomer som er tilfeldig generert innenfor området

det ønskes en konstant befolkning er det vanlig at algoritmen programmeres slik at hvert valgte kromosom produserer ett nytt kromosom.

Krysning Krysning skaper nye kombinasjoner av gener og akselererer evolusjonen betraktelig. De valgte kromosomene har en sannsynlighet P_c for å gjennomgå krysning. De resterende kromosomene gjennomgår kun mutasjon. Blant kromosomene som skal krysses plukkes det ut tilfeldige par. Det finnes flere måter krysningen kan implementeres. Krysningen kan f.eks. foregå på kromosomnivå. I parameterestimeringssammenheng vil dette si å bytte om parametere mellom to parametersett for å skape to nye parametersett. I [12] forklares det at krysning vil være mer effektivt om det foregår binært innenfor hver enkelt parameter. En av de enkleste måtene å implementere krysningen på er ved såkalt 1.-punktskrysning. Ved denne metoden velges det ut en tilfeldig posisjon der informasjonen etter denne posisjonen krysses. Dersom parameter K1 består av 12 bit og krysningspunktet tilfeldig velges til 4

Kromosom1.K1 xxxx|xxxxxxxx
Kromosom2.K1 yyyy|yyyyyyyy

blir parametrene i de nye kromosomene

KromosomA.K1 xxxx|yyyyyyyy
KromosomB.K1 yyyy|xxxxxxxx

Mutasjon Uten mutasjon vil krysning over tid føre til at et fåtall gener begynner å dominere og algoritmen vil konvergere til den beste kombinasjonen av eksisterende gen. Det vil da ikke være mulighet for at ny informasjon kan oppstå og med mindre man er særdeles heldig vil løsningen verken konvergere til det globalt optimale nærområdet eller en lokal optimal løsning. For å kunne nå ut til hele optimaliseringsområdet kreves det mangfold. Dette oppnås med mutasjon. Mutasjon er rett og slett tilfeldig forandring av informasjonen. En enkel måte å implementere mutasjon på er å tilegne hver eneste bit en sannsynlighet P_m for å bytte verdi.

Valg av algoritmeparametere Som nevnt tidligere finnes det ingen godt beskrevet metode for å velge parametere. Generelt sett burde sannsynligheten for mutasjon P_m settes rimelig lav slik at variasjonen ikke blir alt for stor. Da vil det bli vanskelig for algoritmen å sikte seg inn på et optimalt område, men dersom det er for lite mutasjon vil det kunne ta fryktelig lang tid å utforske optimaliseringsområdet for å finne det globalt optimale området. Krysning gjør det mulig å oppdage heldige kombinasjoner av eksisterende informasjon i løpet av få generasjoner og vil dermed kunne resultere i kombinasjoner som ville tatt veldig lang tid å oppdage med mutasjon alene. Av Den grunn burde sannsynligheten for krysning P_c settes rimelig høy, men ikke så høy at genene i befolkningen blir ensformig. I [3] hevdes det at vanlige verdier for disse parametrene ligger mellom 0.001 - 0.2 for P_m og 0.25 - 0.75 for P_c.

Avslutning av algoritmen Den genetiske algoritmen kan avsluttes etter forskjellige kriterier. Den kan f.eks. avsluttes etter å ha kjørt en bestemt tid. En av de vanligste kriteriene er å sette en maksimal generasjon slik at algoritmen avsluttes når denne generasjonen er nådd. Å bruke konvergens som et avslutningskrav er derimot en dårlig ide fordi algoritmen i stor grad baserer seg på å opprettholde et visst mangfold blant individene. I likhet med biologien har ikke evolusjon noe endelig mål, men fortsetter kontinuerlig med å lete etter bedre løsninger.

3.2.2 Beskrivelse av Nelder-Mead algoritmen

Nelder-Mead simplex-algoritmen, først foreslått i [9], er en heuristisk søkealgoritme som kan brukes på ulineære problemer innenfor fler-dimensjonale optimaliseringsområder. Metoden benytter seg av en simplex bestående av n+1 forskjellige punkter innenfor n di-

mensjoner og kan brukes til å finne en approksimasjon av det nærmeste lokale optimumet.

Det lokale optimumet approksimeres ved å bevege og manipulere størrelsen til simplexen ved hjelp av 4 forskjellige operasjoner. Refleksjon, ekspansjon, sammentrekning, og reduksjon.

Refleksjon I utgangspunktet beveges sentrum av simplexen med en viss steglengde i retning av lavere objektiv verdi. Dette gjøres ved å bevege det dårligste punktet i simplexen gjennom sentrum til motsatt side. På den måten bevares størrelsen til simplexen mens den beveges rundt.

Ekspansjon Dersom det reflekterte punktet er objektivt sett bedre enn alle andre punkt i simplexen, flyttes punktet ekstra langt. På den måten ekspanderes simplexen slik at den kan flyttes med større steglengder.

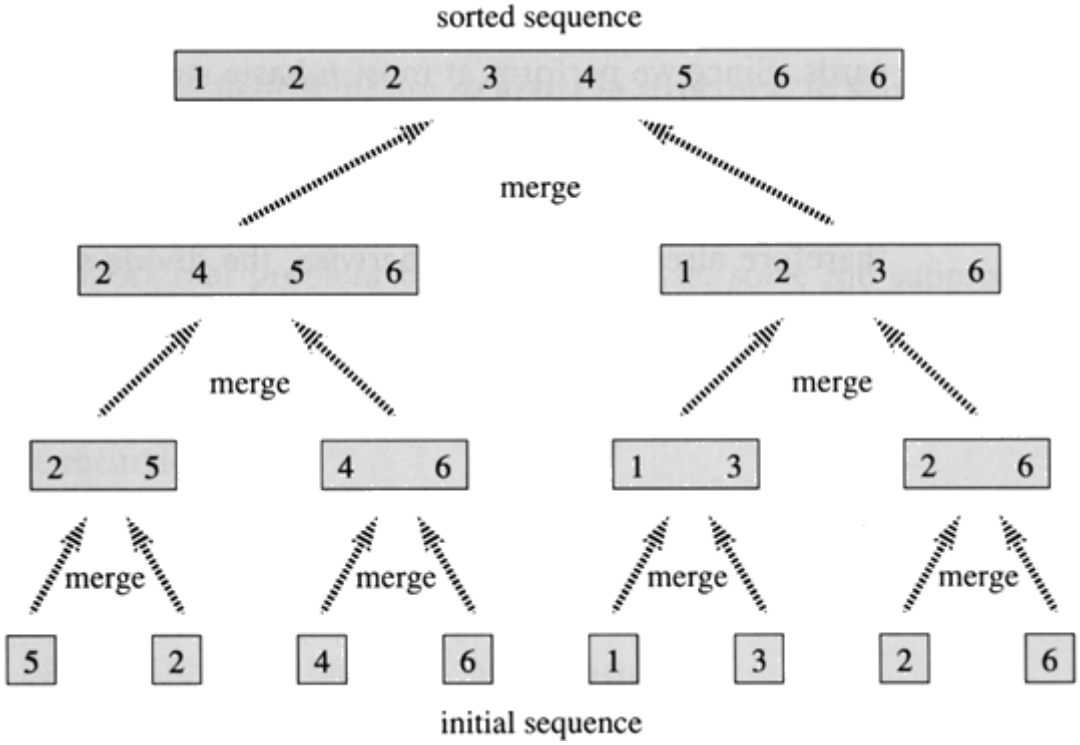
Sammentrekning Dersom det reflekterte punktet er objektivt sett dårligere enn alle de andre punktene, flyttes det reflekterte punktet et lite steg tilbake i retningen den kom fra. På den måten krymper simplexen slik at den kan flyttes rundt mer forsiktig.

Reduksjon Dersom det reflekterte punktet er minst like dårlig som det opprinnelige punktet, krympes hele simplexen sammen i alle retninger for å sikte seg inn mot det beste punktet.

Ved å benytte seg av disse operasjonene kan algoritmen konvergere mot en løsning, men løsningen vil ikke være ekte optimal.

3.3 Merge sort

Merge sort er en rekursiv og stabil sorteringsalgoritme med konsekvent skaleringsytelse på $O(n \log n)$. Algoritmen begynner med å dele opp alle elementene i listen til enkeltelementer. Deretter settes elementene sammen til sorterte par. hvert par settes så sammen to og to i sortert rekkefølge også videre til hele listen er sortert.



Figur 3.9: Merge sort algoritmen

3.4 Linear-quadratic regulator

LQ-regulatoren er en modellbasert regulator som regnes ut ved å minimere en kostfunksjon. Med systemet på tilstand-rom-form

$$\dot{x} = Ax + Bu \tag{3.22}$$

implementeres regulatoren ved

$$u = -Kx \tag{3.23}$$

Metoden kan brukes på ulineære systemer ved å linearisere systemet rundt arbeidspunktet. K kan finnes ved å minimere kostnadsfunksjonen

$$J = \int x^T Q x + u^T R u dt \tag{3.24}$$

der Q og R er diagonale matriser hvor hvert element i matrisen representerer kostnaden for å ha en tilstandverdi eller et pådrag. Dersom det er ønskelig at en tilstand har mindre

utslag enn de andre tilstandene økes det tilsvarende elementet i Q. Dersom det er ønskelig å bruke minst mulig av et visst pådrag økes tilsvarende element i R. Programvare slik som Matlab har innebygde funksjoner for å regne ut K som minimerer kostnadsfunksjonen.

Kapittel 4

Modellering

Denne delen av rapporten dokumenterer den matematiske modellen og fremgangsmåten for hvordan den er funnet. I [10] ble bevegelsesligningene for pendelsystemet utledet med Lagrange metode som tar utgangspunkt i energibalansen. Til eget læringsutbytte, og for å eventuelt verifisere at den tidligere utledningen er korrekt, ble ligningene i dette prosjektet utledet på nytt ved å benytte Newton-Eulers metode. I stedet for å ta utgangspunkt i energibalansen tar Newton-Eulers metode utgangspunkt i momentbalansen. I [10] forklares det at Newton-Eulers-metode vil være en veldig komplisert måte å utlede ligningene. Man vil da ende opp med latterlig store utrykk i utregningen av kryssproduktene. Heldigvis lever vi i en verden med kraftige datamaskiner som lett kan utføre den type oppgave. For å regne ut bevegelsesligningene benyttes derfor symbolsk regning i Matlab. Matlab klarer fint å regne ut ligningene, men mangler et overordnet system for å sortere dem. Resultatet blir dermed en lang kaotisk remse med utrykk der vilkårlige parametere er faktorisert ut til å danne parenteser. Mange av leddene i ligningen kan samles og forenkles med blant annet trigonometriske identiteter. Mange av leddene vil også kansellere hverandre, men på grunn av Matlab sin kaotiske sortering vil disse leddene være låst inn i forskjellige parenteser. For å forenkle ligningene må altså alle parentesene multipliseres ut. Alle leddene må så sorteres etter variablene som inngår. Problemet er at mange av bevegelsesligningene er forferdelig lange når de skrives ut i Matlab. For å unngå tidskrevende slavearbeid ble det skrevet et program i Python som har i oppgave å sortere ligningene som printes i Matlab. Programmet multipliserer ut ligningene, sorter dem til mer fornuftige parenteser og sorterer parametrene i hvert ledd. Når ligningene er sortert gjenstår kun arbeidet med å forenkle ligningen. Python og Matlab-koden som brukes til å regne ut bevegelsesligningene er oppgitt i vedlegg A

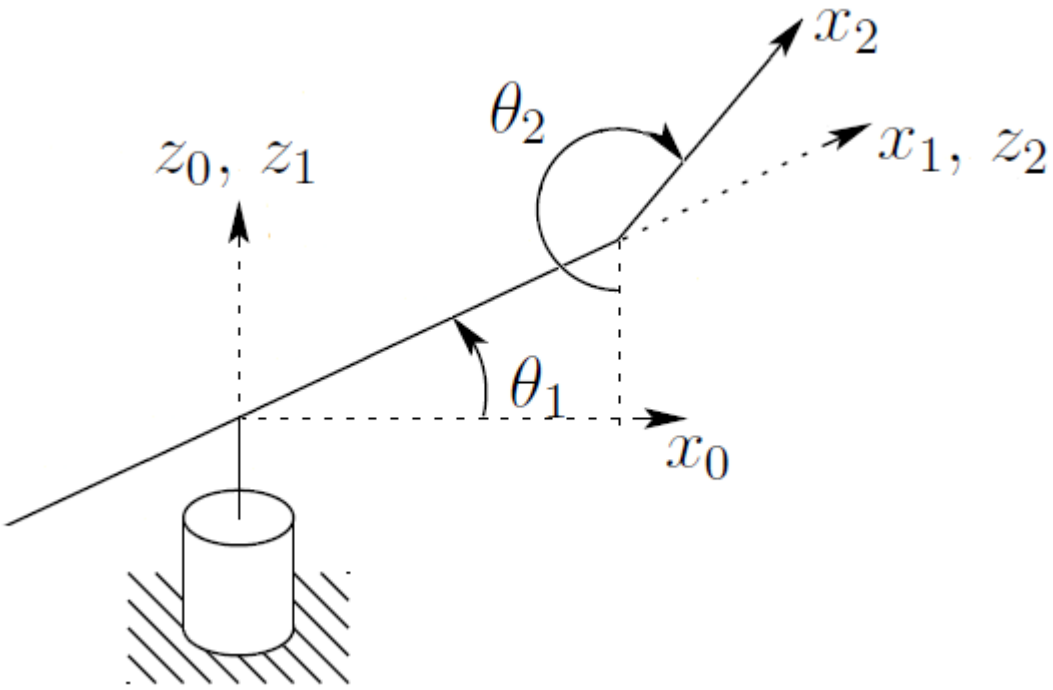
Videre i dette kapittelet forklares det hvordan pendelligningene er utledet. Systemet blir modellert i fire omganger med økende kompleksitet.

- Modell med en enkeltpendel
- Enkeltpendel på hver side av horisontal arm
- Dobbelpendel uten pendel på andre side av horisontal arm
- Modell med alle tre pendler

Friksjonsmodellen blir her ekskludert under utledningen siden denne kan legges til i etterkant.

4.1 System med enkeltpendel

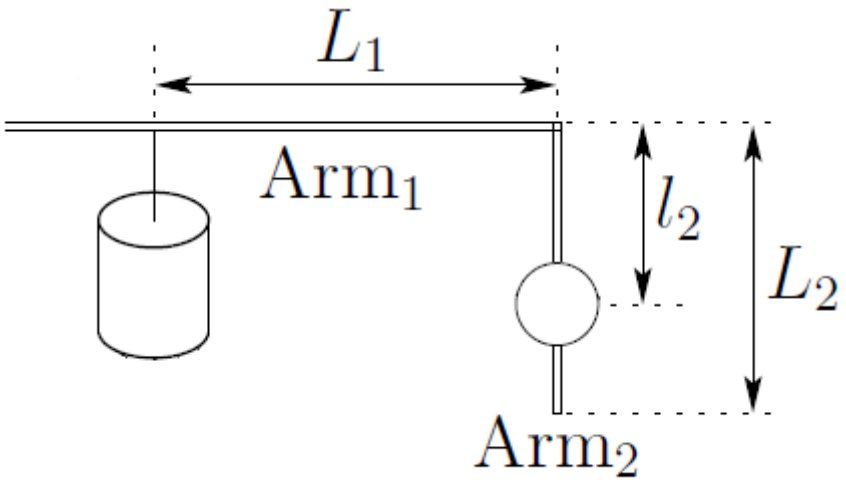
For å modellere systemet benyttes det flere relative koordinatsystemer og kinematikk for å overføre vektorer mellom de forskjellige koordinatsystemene. Først blir koordinatsystemene definert. Deretter settes rotasjonsmatrisene opp. Akselerasjonene ved massesenteret på pendelen blir uttrykt som en vektor slik at denne kan settes inn i momentbalansen.



Figur 4.1: System med enkel pendel

	Variabel	Enhet
Vinkel, Arm1	θ_1	[rad]
Vinkel, Arm2	θ_2	[rad]
	Paramenter	
Masse, Arm1	m_1	[Kg]
Treghetsmoment, Arm1	I_1	[Kg·m^2]
Avstand fra sentrum til enden av Arm1	L_1	[m]
Masse, Arm2	m_2	[Kg]
Treghetsmoment, Arm2	I_2	[Kg·m^2]
Lengde, Arm2	L_2	[m]
Avstand til massesenter på Arm2	l_2	[m]

Tabell 4.1: Variabler og parametere for system med enkeltpendel



Figur 4.2: Lengder til system med enkel pendel

4.1.1 Kinematikk

I tidligere matematiske utledninger gjort I [10] har det blitt benyttet kinematikk slik at alle vektorer kan beskrives i forhold til den faste grunnrammen[1]. Dette ville vært praktisk dersom det var interessant å f.eks. beskrive den absolutte posisjonen til pendlene. Slik er ikke tilfellet ved dette systemet. målet er å beskrive bevegelsen ved hjelp av vinkler, vinkelhastigheter og vinkelakselerasjoner. Det er derfor mye lettere å forholde seg til de relative koordinatsystemene. Under modelleringen i dette prosjektet brukes det derfor enkle rotasjonsmatriser for å kartlegge vektorer mellom rammene i stedet for store rotasjonsmatriser for å overføre samtlige vektorer til grunnrammen.

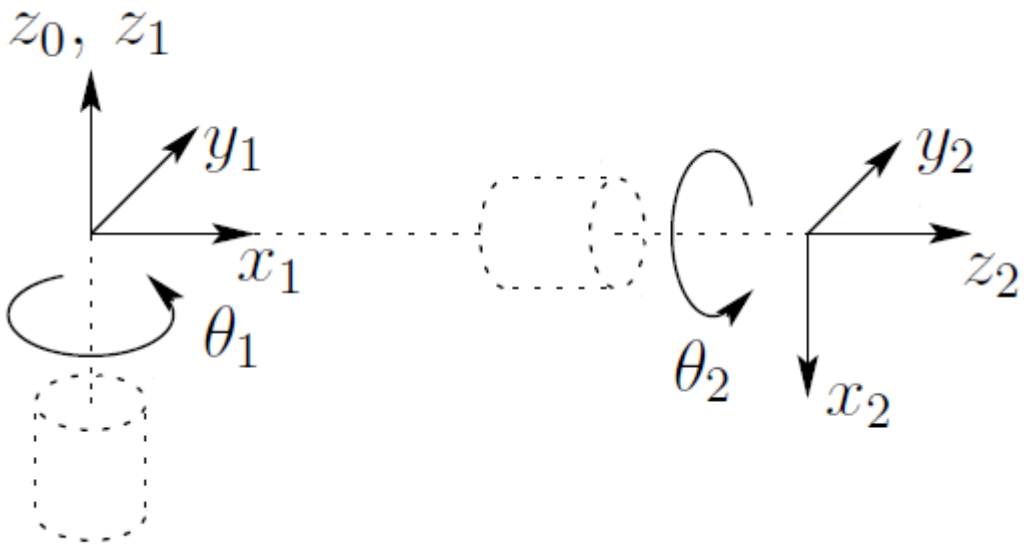
Det blir definert 3 koordinatrammer. Koordinatsystemene defineres etter høyrehåndsregelen der z-aksen peker i leddets retningen. x-aksene defineres til å peke i armenes retning. θ_1 er vinkelen mellom grunnrammen og arm 1. θ_2 er pendelens vinkel. Fra grunnrammen til ramme 1 roteres det θ_1 rundt z-aksen.

$$R_1 = \begin{bmatrix} \cos(\theta_1) & \sin(\theta_1) & 0 \\ -\sin(\theta_1) & \cos(\theta_1) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

Fra ramme 1 til ramme 2 roteres først y-aksen med 90° etterfulgt av θ_2 rundt z-aksen

$$\begin{aligned} R_2 = R_{z,\theta_2} R_{y,90^\circ} &= \begin{bmatrix} \cos(\theta_2) & \sin(\theta_2) & 0 \\ -\sin(\theta_2) & \cos(\theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(90^\circ) & 0 & -\sin(90^\circ) \\ 0 & 1 & 0 \\ \sin(90^\circ) & 0 & \cos(90^\circ) \end{bmatrix} \\ &= \begin{bmatrix} 0 & \sin(\theta_2) & -\cos(\theta_2) \\ 0 & \cos(\theta_2) & \sin(\theta_2) \\ 1 & 0 & 0 \end{bmatrix} \end{aligned} \tag{4.2}$$

[1] (x_0, y_0, z_0)



Figur 4.3: Koordinatrammer festet til enkeltpendelsystem

4.1.2 Treghetsmoment

Treghetsmomentet langs aksene på armene kan beskrives med matrisen

$$\boldsymbol{I} = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix} \tag{4.3}$$

For Arm1 vil I_x og I_y være ubetydelig siden denne armen kun roterer rundt z-aksen. Pendlene roterer rundt z-aksen på de relative koordinatsystemene, men siden disse koordinatsystemene roterer når Arm1 roterer vil det virke inn moment på pendlene langs flere akser. I [10] ble treghetsmomentet for pendlene kun beskrevet for z-aksen. Treghetsmomentet langs de andre aksene ble i effekt satt til 0. Under modellering av furutapendel i [2] ble treghetsmomentet langs x-aksen tilnærmet 0 ettersom at pendelen er ganske tynn. Videre ble det antatt at pendelen har rotasjonssymmetri slik at treghetsmomentet langs de to andre aksene er like. Den samme forenklingen benyttes også i dette prosjektet

$$\boldsymbol{I}_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & I_2 & 0 \\ 0 & 0 & I_2 \end{bmatrix} \tag{4.4}$$

4.1.3 Hastighet og akselerasjon

for å representere effekten av gravitasjon på pendelen, settes Arm1 sin akselerasjon til g i positiv z-retning.

$$\boldsymbol{a}_1 = \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \tag{4.5}$$

For pendelen vil dette være ekvivalent til å akselerere pendelen med g i negativ retning. Dynamikken til Arm1 vil derimot ikke påvirkes av gravitasjon uansett retning siden denne armen står i vater til en hver tid. Vinkelhastigheten til Arm1 er

$$\boldsymbol{\omega}_1 = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 \end{bmatrix} \tag{4.6}$$

Vinkelakselerasjonen til Arm1 blir da

$$\boldsymbol{\alpha}_1 = \dot{\boldsymbol{\omega}}_{\mathbf{1}} = \begin{bmatrix} 0 \\ 0 \\ \ddot{\theta}_1 \end{bmatrix} \tag{4.7}$$

For Arm2 er vinkelhastigheten

$$\boldsymbol{\omega}_2 = R_2\boldsymbol{\omega}_1 + \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} -\dot{\theta}_1 \cos(\theta_2) \\ \dot{\theta}_1 \sin(\theta_2) \\ \dot{\theta}_2 \end{bmatrix} \tag{4.8}$$

Vinkelakselerasjonen kan finnes ved derivering

$$\boldsymbol{\alpha}_2 = \dot{\boldsymbol{\omega}}_{\mathbf{2}} = \begin{bmatrix} -\ddot{\theta}_1 \cos(\theta_2) + \dot{\theta}_1\dot{\theta}_2 \sin(\theta_2) \\ \ddot{\theta}_1 \sin(\theta_2) + \dot{\theta}_1\dot{\theta}_2 \cos(\theta_2) \\ \ddot{\theta}_2 \end{bmatrix} \tag{4.9}$$

Den lineære hastigheten til enden av Arm1 kan finnes ved

$$\boldsymbol{v}_{1e} = \frac{\mathrm{d}\boldsymbol{L}_1}{\mathrm{d}t} = \boldsymbol{\omega}_1 \times \boldsymbol{L}_1 \tag{4.10}$$

der $\boldsymbol{L}_1$ er lengdevektoren til Arm1.

$$\boldsymbol{L}_1 = \begin{bmatrix} L_1 \\ 0 \\ 0 \end{bmatrix} \tag{4.11}$$

I referanseramme 2 blir den lineære hastigheten

$$\boldsymbol{v}_2 = R_2 \left(\boldsymbol{\omega}_1 \times \boldsymbol{L}_1 \right) \tag{4.12}$$

Dette er altså hastigheten til Arm2 sitt svingningspunkt. Den lineære akselerasjonen til svingningspunktet kan finnes ved

$$\begin{aligned}\boldsymbol{a}_2 &= R_2\left(\frac{\mathrm{d}\boldsymbol{v}_{1e}}{\mathrm{d}t} + \boldsymbol{a}_1\right) \\ &= R_2\left(\frac{\mathrm{d}}{\mathrm{d}t}\left(\boldsymbol{\omega}_1 \times \boldsymbol{L}_1\right) + \boldsymbol{a}_1\right) \\ &= R_2\big(\dot{\boldsymbol{\omega}}_{\mathbf{1}} \times \boldsymbol{L}_1 + \boldsymbol{\omega}_1 \times (\boldsymbol{\omega}_1 \times \boldsymbol{L}_1) + \boldsymbol{a}_1\big)\end{aligned} \tag{4.13}$$

Den lineære akselerasjonen ved pendelens massesentrum er

$$\begin{aligned}\boldsymbol{a}_{2c} &= \frac{\mathrm{d}}{\mathrm{d}t}\left(\boldsymbol{\omega}_2 \times \boldsymbol{l}_2\right) + \boldsymbol{a}_2 \\ &= \dot{\boldsymbol{\omega}}_{\mathbf{2}} \times \boldsymbol{l}_2 + \boldsymbol{\omega}_2 \times (\boldsymbol{\omega}_2 \times \boldsymbol{l}_2) + \boldsymbol{a}_2\end{aligned} \tag{4.14}$$

Der $\boldsymbol{l}_2$ er lengdevektoren til massesenteret på Arm2.

$$\boldsymbol{l}_2 = \begin{bmatrix} l_2 \\ 0 \\ 0 \end{bmatrix} \tag{4.15}$$

Nå som akselerasjonene er beskrevet kan de brukes til å finne krefter og moment i momentbalansen.

4.1.4 Momentbalanse

Kraften som virker inn på pendelen og Arm1 er

$$\boldsymbol{F}_1 = m_1\boldsymbol{a}_1 \tag{4.16}$$

$$\boldsymbol{F}_2 = m_2\boldsymbol{a}_{2c} \tag{4.17}$$

Momentvektorene kan finnes ved

$$\boldsymbol{N}_1 = \boldsymbol{I}_1\boldsymbol{\alpha}_1 + \boldsymbol{\omega}_1 \times \boldsymbol{I}_1\boldsymbol{\omega}_1 \tag{4.18}$$

$$\boldsymbol{N}_2 = \boldsymbol{I}_2\boldsymbol{\alpha}_2 + \boldsymbol{\omega}_2 \times \boldsymbol{I}_2\boldsymbol{\omega}_2 \tag{4.19}$$

der treghetsmomentet for Arm1 er tilnærmet til

$$\boldsymbol{I}_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & I_1 \end{bmatrix} \tag{4.20}$$

siden Arm1 kun roterer rundt z-aksen. Momentvektoren for Arm1 blir da

$$\boldsymbol{N}_1 = I_1\boldsymbol{\alpha}_1 \tag{4.21}$$

Momentet som Arm2 påtrykker Arm1 kan finnes ved

$$\boldsymbol{n}_2 = \boldsymbol{N}_2 + \boldsymbol{l}_2 \times \boldsymbol{F}_2 \tag{4.22}$$

Kraften $\boldsymbol{F}_2$ fra Arm 2 virker tilbake på Arm 1 slik at det resulterende momentet blir

$$\boldsymbol{l}_1 \times R_2^T \boldsymbol{F}_2 \tag{4.23}$$

Momentet som Arm1 påtrykker basen blir

$$\boldsymbol{n}_1 = \boldsymbol{N}_1 + \boldsymbol{R}_2^T \boldsymbol{n}_2 + \boldsymbol{l}_1 \times R_2^T \boldsymbol{F}_2 \tag{4.24}$$

Momentbalansen kan da settes opp slik

$$\begin{bmatrix} \boldsymbol{n}_1^T \\ \boldsymbol{n}_2^T \end{bmatrix} \hat{\boldsymbol{z}} + \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} = \begin{bmatrix} \tau_m \\ 0 \end{bmatrix} \tag{4.25}$$

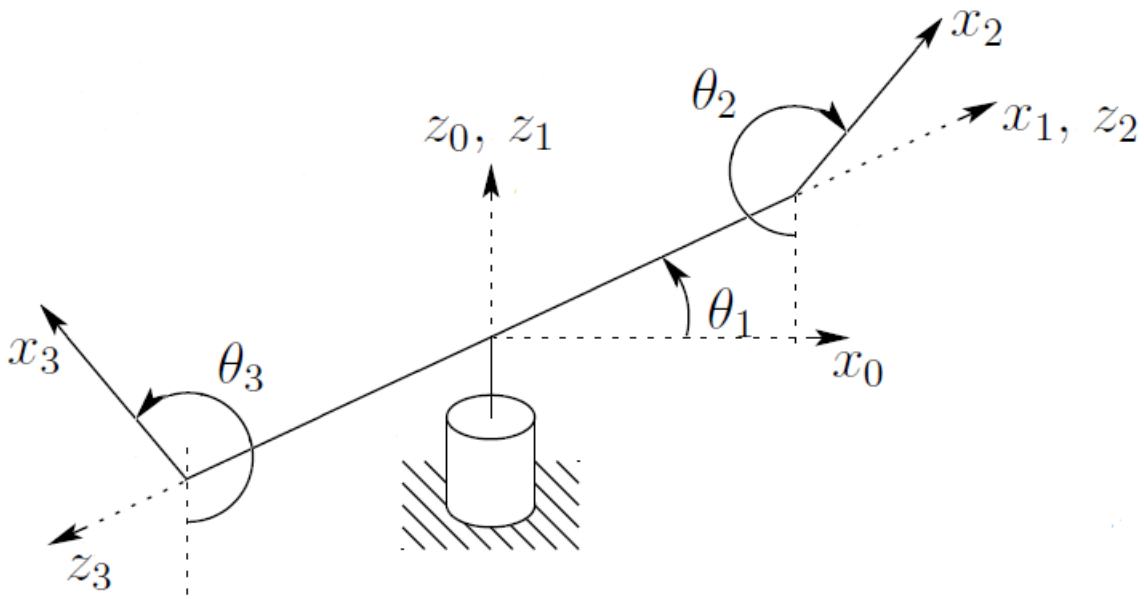
der

$$\hat{\boldsymbol{z}} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{4.26}$$

er enhetsvektoren en z-retning. τ_1 og τ_2 er friksjonsmomentet til de to armene. τ_m momentet som produseres av motoren. De utledede ligningene er oppgitt i Vedlegg B

4.2 System med pendel på hver side

Modelleringen av system med pendel på hver side foregår ganske likt som for enkeltpendel. De to pendlene vil ikke påvirke hverandre direkte, men vil i stedet være knyttet sammen gjennom dynamikken til Arm1.



Figur 4.4: System med pendel i begge ender

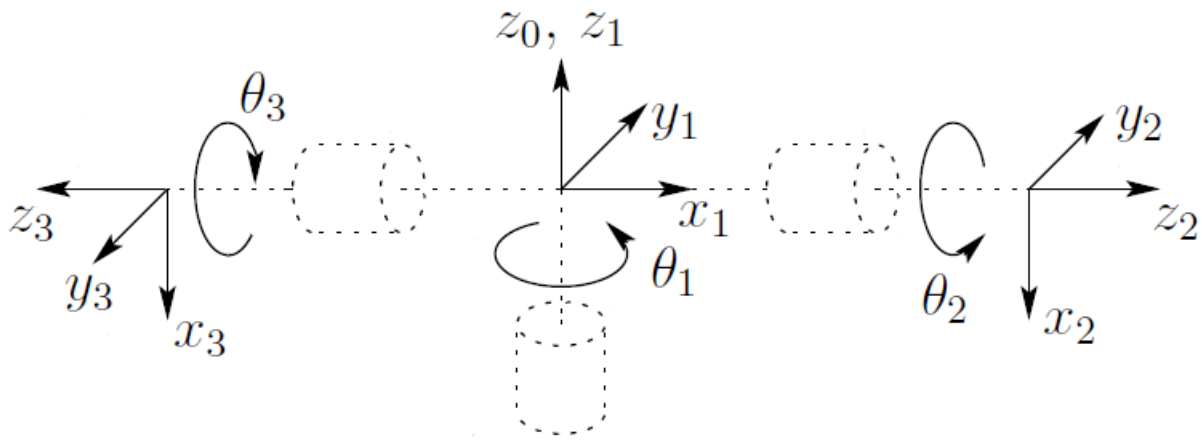
	Paramenter	Enhet
Masse	m_3	[Kg]
Treghetsmoment	I_3	[Kg·m^2]
Lengde	L_3	[m]
Avstand til massesenter på Arm2	l_2	[m]

Tabell 4.2: Variabler og parametere Arm3

4.2.1 Kinematikk

Rotasjonsmatrisene for koordinatrammene til Arm1 og Arm2 som ble utledet i 4.1.1 kan fortsatt benyttes for system med pendel på hver side. Koordinatsystemet til Arm3 settes opp etter samme retningslinjer. Fra ramme 1 til ramme 3 roteres y-aksen med $-90°$, deretter roteres z-aksen med $180°$ slik at x-aksen peker rett ned når $\theta_3 = 0$. z-aksen roteres så med θ_3.

$$
\begin{aligned}
R_3 &= R_{z,\theta_3} R_{z,180°} R_{y,-90°} \\
&= \begin{bmatrix} \cos(\theta_3) & \sin(\theta_3) & 0 \\ -\sin(\theta_3) & \cos(\theta_3) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(180) & \sin(180) & 0 \\ -\sin(180) & \cos(180) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(-90°) & 0 & -\sin(-90°) \\ 0 & 1 & 0 \\ \sin(-90°) & 0 & \cos(-90°) \end{bmatrix} \\
&= \begin{bmatrix} 0 & -\sin(\theta_2) & -\cos(\theta_2) \\ 0 & -\cos(\theta_2) & \sin(\theta_2) \\ -1 & 0 & 0 \end{bmatrix}
\end{aligned}
\tag{4.27}
$$



Figur 4.5: Koordinatrammer festet til system med pendel på hver side

4.2.2 Hastighet og akselerasjon

For Arm3 blir vinkelhastigheten

$$\boldsymbol{\omega}_3 = R_3\boldsymbol{\omega}_1 + \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_3 \end{bmatrix} = \begin{bmatrix} -\dot{\theta}_1\cos(\theta_3) \\ \dot{\theta}_1\sin(\theta_3) \\ \dot{\theta}_2 \end{bmatrix} \tag{4.28}$$

Vinkelakselerasjonen blir

$$\boldsymbol{\alpha}_3 = \dot{\boldsymbol{\omega}}_{\boldsymbol{3}} = \begin{bmatrix} -\ddot{\theta}_1\cos(\theta_3) + \dot{\theta}_1\dot{\theta}_3\sin(\theta_3) \\ \ddot{\theta}_1\sin(\theta_3) + \dot{\theta}_1\dot{\theta}_3\cos(\theta_3) \\ \ddot{\theta}_3 \end{bmatrix} \tag{4.29}$$

Den lineære akselerasjonen til svingningspunktet på Arm3 kan finnes på likt vis som for Arm2

$$\boldsymbol{a}_3 = R_3\big(\dot{\boldsymbol{\omega}}_{\boldsymbol{1}} \times \boldsymbol{L}_1 + \boldsymbol{\omega}_1 \times (\boldsymbol{\omega}_1 \times \boldsymbol{L}_1) + \boldsymbol{a}_1\big) \tag{4.30}$$

Den lineære akselerasjonen ved pendelens massesentrum er

$$\boldsymbol{a}_{3c} = \dot{\boldsymbol{\omega}}_{\boldsymbol{3}} \times \boldsymbol{l}_3 + \boldsymbol{\omega}_3 \times (\boldsymbol{\omega}_3 \times \boldsymbol{l}_3) + \boldsymbol{a}_3 \tag{4.31}$$

Der $\boldsymbol{l}_3$ er lengdevektoren til massesenteret på Arm3.

$$\boldsymbol{l}_3 = \begin{bmatrix} l_3 \\ 0 \\ 0 \end{bmatrix} \tag{4.32}$$

4.2.3 Momentbalanse

Kraften og momentvektoren til Arm3 er

$$\boldsymbol{F}_3 = m_3\boldsymbol{a}_{3c} \tag{4.33}$$

$$\boldsymbol{N}_3 = \boldsymbol{I}_3\boldsymbol{\alpha}_3 + \boldsymbol{\omega}_3 \times \boldsymbol{I}_3\boldsymbol{\omega}_3 \tag{4.34}$$

hvor

$$\boldsymbol{I}_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & I_3 & 0 \\ 0 & 0 & I_3 \end{bmatrix} \tag{4.35}$$

Momentet som Arm3 påtrykker Arm1 kan finnes ved

$$\boldsymbol{n}_3 = \boldsymbol{N}_3 + \boldsymbol{l}_3 \times \boldsymbol{F}_3 \tag{4.36}$$

Kraften $\boldsymbol{F}_3$ fra Arm 3 virker tilbake på Arm 1 slik at det resulterende momentet blir

$$\boldsymbol{l}_1 \times R_3^T\boldsymbol{F}_3 \tag{4.37}$$

Momentet som Arm1 påtrykker basen blir da

$$\boldsymbol{n}_1 = \boldsymbol{N}_1 + \boldsymbol{R}_2^T\boldsymbol{n}_2 + R_3^T\boldsymbol{n}_3 + \boldsymbol{l}_1 \times R_2^T\boldsymbol{F}_2 + \boldsymbol{l}_1 \times R_3^T\boldsymbol{F}_3 \tag{4.38}$$

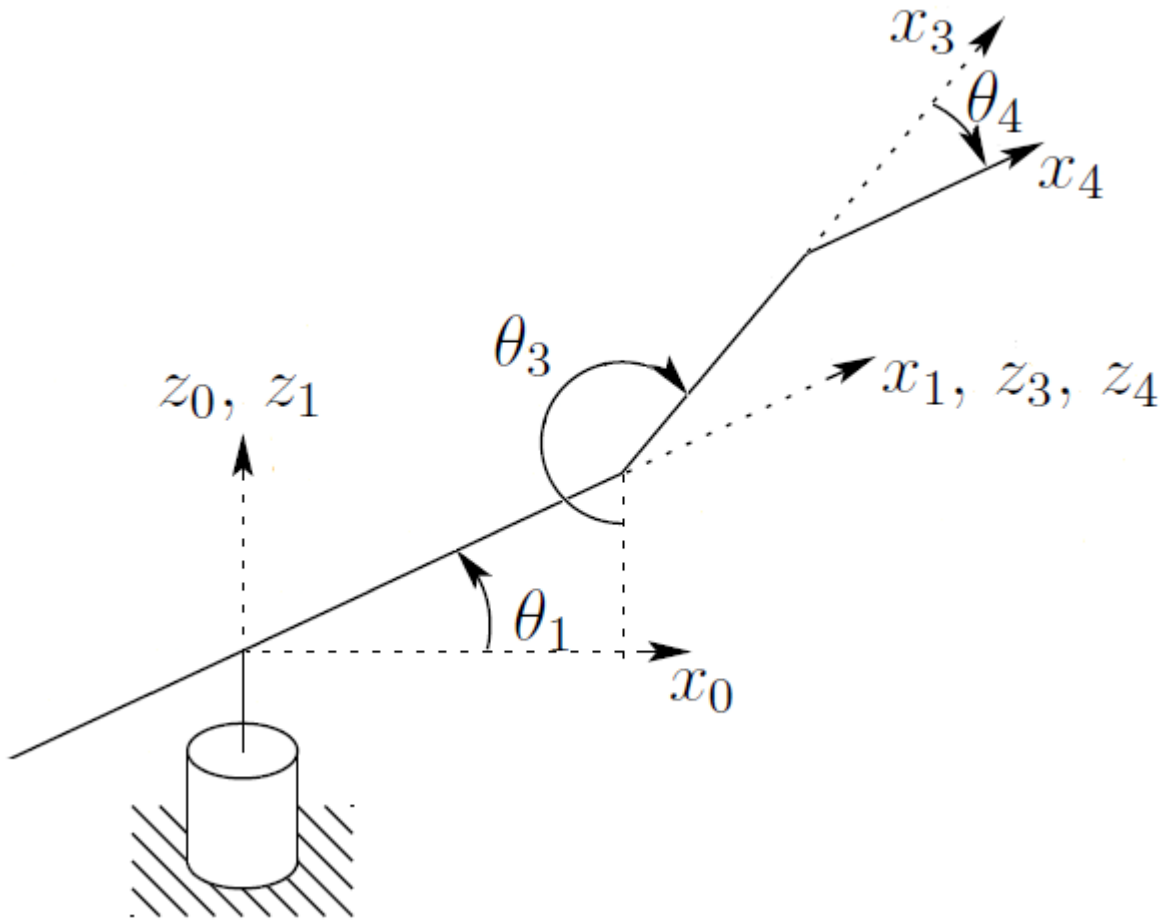
Momentbalansen blir

$$\begin{bmatrix} \boldsymbol{n}_1^T \\ \boldsymbol{n}_2^T \\ \boldsymbol{n}_3^T \end{bmatrix} \hat{\boldsymbol{z}} + \begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} = \begin{bmatrix} \tau_m \\ 0 \\ 0 \end{bmatrix} \tag{4.39}$$

Fra momentbalansen for Arm1 kan man se at utrykkene for moment og kraft som virker på Arm1 fra pendlene inngår i uavhengige ledd. En mulig strategi for å modellere systemet kan være å modellere hvert enkeltpendelsystem for så å kombinere utrykkene i bevegelsesligningen til Arm1.

4.3 System med dobbelpendel

Med dobbelpendelsystemet blir bevegelsesligningene en del større. I dette systemet er ikke de to pendlene isolert fra hverandre slik som i system med pendel på hver side. Kraft og moment fra Arm4 vil virke inn på kraften og momentet som Arm3 påtrykker Arm1.



Figur 4.6: System med Dobbelpendel

4.3.1 Kinematikk

Kartlegging fra ramme 3 til ramme 4 er ganske grei. Den består av en enkel rotasjon om z-aksen og kan dermed beskrives med en enkel rotasjonsmatrise

$$R_4 = \begin{bmatrix} \cos(\theta_4) & \sin(\theta_4) & 0 \\ -\sin(\theta_4) & \cos(\theta_4) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.40}$$

4.3.2 Akselerasjon

For Arm4 er vinkelhastigheten

$$\boldsymbol{\omega}_4 = R_4\boldsymbol{\omega}_3 + \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_4 \end{bmatrix} = \begin{bmatrix} -\dot{\theta}_1\cos(\theta_3+\theta_4) \\ \dot{\theta}_1\sin(\theta_3+\theta_4) \\ \dot{\theta}_3+\dot{\theta}_4 \end{bmatrix} \tag{4.41}$$

Vinkelakselerasjonen kan finnes ved

$$\begin{aligned} \dot{\boldsymbol{\omega}}_{\boldsymbol{4}} = \boldsymbol{\alpha}_4 &= R_4\dot{\boldsymbol{\omega}}_{\boldsymbol{3}} + R_4\boldsymbol{\omega}_3 \times \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \ddot{\theta}_4 \end{bmatrix} \\ &= \begin{bmatrix} -\ddot{\theta}_1\cos(\theta_3+\theta_4) + \dot{\theta}_1\dot{\theta}_3\sin(\theta_3+\theta_4) + \dot{\theta}_1\dot{\theta}_4\sin(\theta_3+\theta_4) \\ \ddot{\theta}_1\sin(\theta_3+\theta_4) + \dot{\theta}_1\dot{\theta}_3\cos(\theta_3+\theta_4) + \dot{\theta}_1\dot{\theta}_4\cos(\theta_3+\theta_4) \\ \ddot{\theta}_3+\ddot{\theta}_4 \end{bmatrix} \end{aligned} \tag{4.42}$$

Akselerasjonen til svingningspunktet på Arm4 kan finnes ved

$$\boldsymbol{a}_4 = R_4\left(\dot{\boldsymbol{\omega}}_{\boldsymbol{3}} \times \boldsymbol{L}_3 + \boldsymbol{\omega}_3 \times (\boldsymbol{\omega}_3 \times \boldsymbol{L}_3) + \boldsymbol{a}_3\right) \tag{4.43}$$

der

$$\boldsymbol{L}_3 = \begin{bmatrix} L_3 \\ 0 \\ 0 \end{bmatrix} \tag{4.44}$$

Akselerasjonen på massesenteret til Arm4 blir

$$\boldsymbol{a}_{4c} = \dot{\boldsymbol{\omega}}_{\boldsymbol{4}} \times \boldsymbol{l}_4 + \boldsymbol{\omega}_4 \times (\boldsymbol{\omega}_4 \times \boldsymbol{l}_4) + \boldsymbol{a}_4 \tag{4.45}$$

4.3.3 Momentbalanse

Kraften som virker på massesenteret til Arm4 og momentvektoren uttrykkes ved

$$\boldsymbol{F}_4 = m_4\boldsymbol{a}_{4c} \tag{4.46}$$

$$\boldsymbol{N}_4 = \boldsymbol{I}_4\boldsymbol{\alpha}_4 + \boldsymbol{\omega}_4 \times \boldsymbol{I}_4\boldsymbol{\omega}_4 \tag{4.47}$$

Momentet som Arm4 påtrykker Arm3 er

$$\boldsymbol{n}_4 = \boldsymbol{N}_4 + \boldsymbol{l}_4 \times \boldsymbol{F}_4 \tag{4.48}$$

Momentet som Arm3 påtrykker Arm1 blir da

$$\boldsymbol{n}_3 = \boldsymbol{N}_3 + \boldsymbol{l}_3 \times \boldsymbol{F}_3 + \boldsymbol{R}_4^T \boldsymbol{n}_4 + \boldsymbol{L}_3 \times R_4^T \boldsymbol{F}_4 \tag{4.49}$$

Kraften som virker inn på Arm1 fra Arm2 kan beskrives ved

$$\boldsymbol{F}_3 + R_4^T \boldsymbol{F}_4 \tag{4.50}$$

Momentet som Arm1 påtrykker basen blir da

$$\boldsymbol{n}_1 = \boldsymbol{N}_1 + \boldsymbol{R}_3^T \boldsymbol{n}_3 + \boldsymbol{l}_1 \times \boldsymbol{R}_3^T (\boldsymbol{F}_3 + \boldsymbol{R}_4^T \boldsymbol{F}_4) \tag{4.51}$$

Momentbalansen blir

$$\begin{bmatrix} \boldsymbol{n}_1^T \\ \boldsymbol{n}_3^T \\ \boldsymbol{n}_4^T \end{bmatrix} \hat{\boldsymbol{z}} + \begin{bmatrix} \tau_1 \\ \tau_3 \\ \tau_4 \end{bmatrix} = \begin{bmatrix} \tau_m \\ 0 \\ 0 \end{bmatrix} \tag{4.52}$$

4.4 Trippelpendelsystem

For å finne momentbalansen til systemet med alle tre pendler trenger man bare å sette sammen utrykkene for Arm2 og dobbelpendelsystemet. Momentet som Arm1 påtrykker basen blir dermed

$$\boldsymbol{n}_1 = \boldsymbol{N}_1 + \boldsymbol{R}_2^T \boldsymbol{n}_2 + \boldsymbol{R}_3^T \boldsymbol{n}_3 + \boldsymbol{l}_1 \times \boldsymbol{R}_2^T \boldsymbol{F}_2 + \boldsymbol{l}_1 \times \boldsymbol{R}_3^T (\boldsymbol{F}_3 + \boldsymbol{R}_4^T \boldsymbol{F}_4) \tag{4.53}$$

Momentbalansen blir

$$\begin{bmatrix} \boldsymbol{n}_1^T \\ \boldsymbol{n}_2^T \\ \boldsymbol{n}_3^T \\ \boldsymbol{n}_4^T \end{bmatrix} \hat{\boldsymbol{z}} + \begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \\ \tau_4 \end{bmatrix} = \begin{bmatrix} \tau_m \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.54}$$

4.5 Pendelfriksjon

Pendlene påvirkes av forskjellige friksjonskomponenter. I [10] blir det benyttet en viskøs friksjonsmodell.

$$\tau_{\text{friksjon}} = b\dot{\theta} \tag{4.55}$$

I [13] blir det brukt en mer korrekt ulineær coulombfriksjonsmodell

$$\tau_{\text{friksjon}} = b\text{sign}(\dot{\theta}) \tag{4.56}$$

Det blir i [13] også foreslått en komplett friksjonemodell som har med luftmotstand i tillegg til coulombfriksjon, men siden pendlene skal balanseres rundt små vinkelhastigheter ble luftmotstanden ansett som ubetydelig. Luftmotstanden vil derimot ha en ganske stor påvirkning under oppsving av pendlene. I dette prosjektet var det dermed ønskelig å inkludere påvirkningen av luftmotstand siden det skulle utvikles en mer nøyaktig simulator. friksjonsmodellen for de isolerte pendlene blir da

$$\tau_{\text{friksjon}} = b_l \dot{\theta}^2 + b_c \text{sign}(\dot{\theta}) \tag{4.57}$$

Luftmotstanden som påvirker pendlene vil også være avhengig av hastigheten som motoren roterer med. Denne delen av luftmotstanden vil også være avhengig av vinkelen til pendelen. Dersom pendelen henger rett ned vil luftmotstanden være mye større enn om pendelen står 90°. Følgende friksjonsmodell foreslås for å ta hensyn til luftmotstanden som oppstår når motoren roterer.

$$\tau_{\text{friksjon}} = b_l\dot{\theta}_2^2 + b_{lm}\dot{\theta}_1^2\cos(\theta_2) + b_c\text{sign}(\dot{\theta}_2) \tag{4.58}$$

der θ_1 er vinkelen til horisontal arm og θ_2 er vinkelen til pendelen. Med denne friksjonsmodellen vil pendelen få en stasjonær vinkel dersom motoren roterer med konstant hastighet.

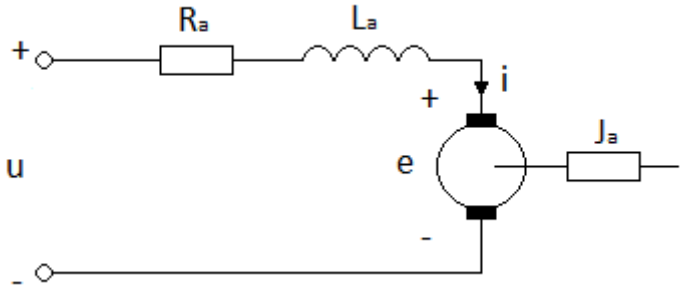
4.6 DC-motormodell

I [7] forklares det hvordan den lineære DC-motormodellen blir unøyaktig dersom motoren ofte skifter retning. Ifølge [7] vil likestrømsmotorer i stor grad være påvirket av coulombfriksjon, dødsoneulineæritet og hysterese. I [13] benyttes det en motormodell med coulombfriksjon, men denne modellen klarer ikke å representere stribeckeffekten under lave hastigheter og blir dermed unøyaktig under små vinkelhastigheter. Siden motoren er nødt til å generere små vinkelhastigheter når pendlene skal balanseres. Det foreslås derfor å bruke en modifisert coulombfriksjonsmodell

$$\tau_{\text{friksjon}} = (b_{c1} + b_{c2}e^{-\alpha|\dot{\theta}|})\text{sign}(\dot{\theta}) + b_v\dot{\theta} \tag{4.59}$$

der b_{c1} er tradisjonell coulombfriksjon. b_{c2} utgjør stribeck-effekten og b_v er viskøs friksjon.

I pendelsystemet benyttes en likestrømsmotor slik som vist i Figur 4.7.



Figur 4.7: Ekvivalentskjema for likestrømsmotor

	Variabel	Enhet
Klemmespenning	u	[V]
Strøm	i_a	[A]
Motindusert spenning	e	[V]
Vinkelhastighet	$\dot{\theta}$	[rad/s]
Utviklet motormoment	τ_m	[NM]
Ytre belastningsmoment	T_L	[NM]
	Parameter	Enhet
Indre motstand	R_a	[Ω]
Induktans	L_a	[H]
Treghetsmoment	J_a	[kgm^2]
Spenningskonstant	K_e	[V/(rad/s)]
Momentkonstant	K_T	[Nm/A]

Tabell 4.3: variabler og parametere i motormodellen

Motindusert spenning er gitt ved

$$e = K_e\dot{\theta} \tag{4.60}$$

Utviklet motormoment er gitt ved

$$\tau_m = K_T i_a \tag{4.61}$$

For likestrømsmotorer er det vanligvis lite jerntap slik at K_e og K_T vil få tilnærmet samme verdi. Dette er også tilfellet ved motoren i pendelsystemet hvor det er oppgitt samme verdi for disse konstantene. Spenningsbalansen kan uttrykkes ved

$$\begin{aligned} 0 &= u - R_a i_a - L_a\dot{i}_a - K_e\dot{\theta} \\ \dot{i}_a &= \frac{1}{L_a}(u - R_a i_a - K_e\dot{\theta}) \end{aligned} \tag{4.62}$$

momentbalansen kan uttrykkes

$$\begin{aligned} J_a\ddot{\theta} &= K_T i_a - T_L - \tau_{\text{friksjon}} \\ \ddot{\theta} &= \frac{1}{J_a}(K_T i_a - T_L - \tau_{\text{friksjon}}) \end{aligned} \tag{4.63}$$

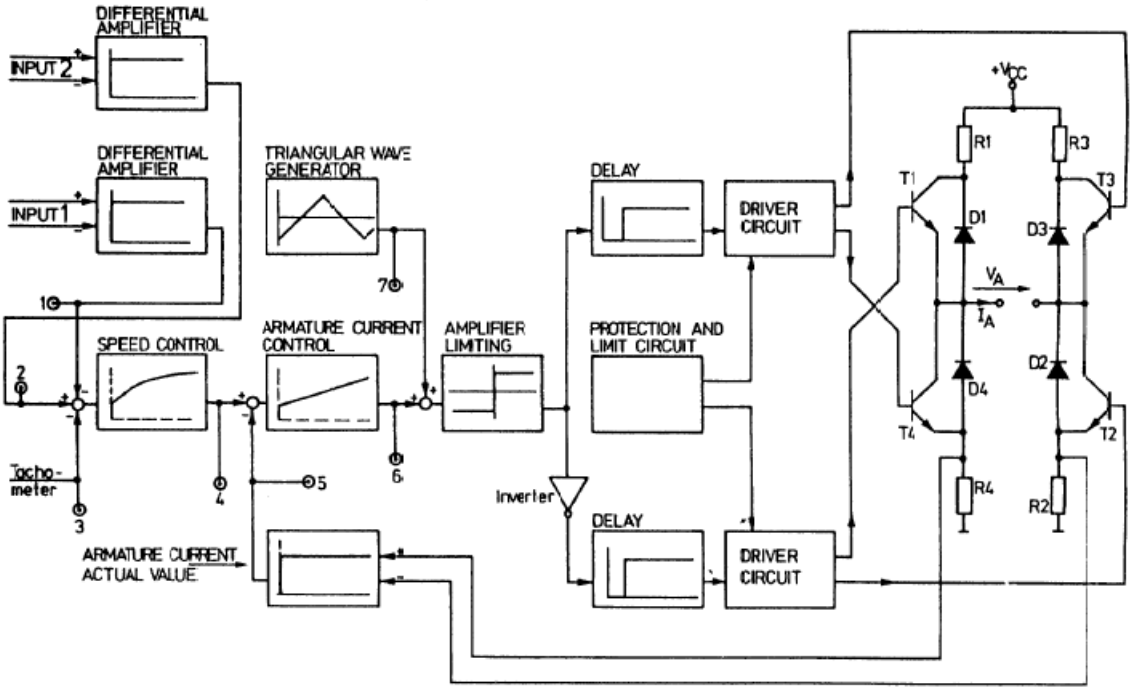
Der T_L er ytre belastningsmoment. I pendelsystemet vil T_L være påtrykt moment til pendelligningene. Med last vil treghetsmomentet bli $J = J_a + J_{\text{last}}$.

4.7 Servo

Inngangspenningen til motoren reguleres fra baldor TSNM. Et skjema av dette regulatorkortet er vist i Figur 4.8. I [13] ble hastighetsregulatoren koplet ut slik at regulatorkortet nå kun benytter strømregulering. Modellen for strømregulatoren ble i [13] satt opp slik

$$u = (K_{pi} + \frac{1}{s}Kii)(i_d - i_a) \tag{4.64}$$

Der i_d er referansesignalet fra mikrokontrolleren og i_a er strømmen i motoren slik som motorligningene beskriver. Regulatormodellen fra [13] regulerer altså strømmen i motoren slik at effekten fra motindusert spenning blir kansellert. Denne modellen blir derimot unøyaktig i forhold til hvordan regulatoren faktisk fungerer. Fra Figur 4.8 kan man se at strømmen måles bak likeretterbroa. Når motoren trekker energi vil denne målte strømmen være den samme som strømmen i motorkretsen(i_a). når motoren ikke lenger trekker energi, men opererer i generatormodus, vil likeretterbroen og transistorene sperre slik at det ikke måles noen strøm. I dette tilfellet vil ikke regulatoren kunne motvirke dempingen fra motindusert spenning. I Simulink kan i_a lett trekkes tilbake til regulatoren, men det er vanskelig å kun måle strømmen som motoren trekker. En løsning kan være å regne ut avvik mellom absoluttverdi av påtrykt spenning og motindusert spenning $avvik = |u| - |e|$ for så å benytte en bryter til å koble ut regulatoren dersom dette $avvik < 0$. I så fall må også i_a kobles ut på inngangen til regulatoren slik at pådraget på utgangen ikke ligger i metning neste gang regulatoren kobles inn. Denne metoden ble forsøkt implementert, men fungerte dårlig da Simulink ga ut feilmelding om algebraisk loop som følge av at u kobles ut med sin egen verdi som kriterie.



Figur 4.8: Baldor TSNM regulatorkort

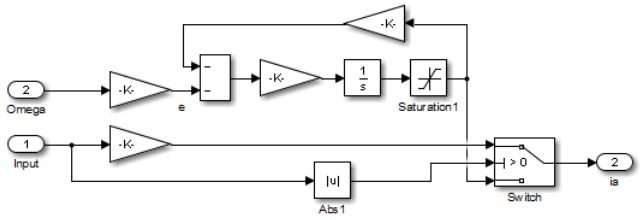
Resultatene i [13] viser at strømregulatoren er såpass rask at det er mulig å approksimere modellen ved å sende referanseverdien rett inn i rotasjonssløyfen på motormodellen[2]. Om modellen ikke inneholder regulator, vil det heller ikke være noen problemer med å få trukket riktig verdi tilbake til regulatoren. Den approksimerte modellen forsøkes implementert slik som vist i Figur 4.9. Når modellen mottar inngangsverdi brukes det at $i_a = i_d$ for å etterligne en ideell regulator. $i_d = 0$ representerer situasjonen hvor motoren

[2] $i_a \approx i_d$

ikke trekker strøm. Da vil ikke regulatoren kunne gi noe bidrag til å motvirke motindusert spenning. I modellen kobles derfor strømsløyfen inn når $i_d = 0$.

$$i_a = \begin{cases} -\frac{1}{L_a s}(K_e\dot{\theta}_1 + R_a i_a) & \text{Hvis } i_d = 0 \\ i_d & \text{ellers} \end{cases} \tag{4.65}$$

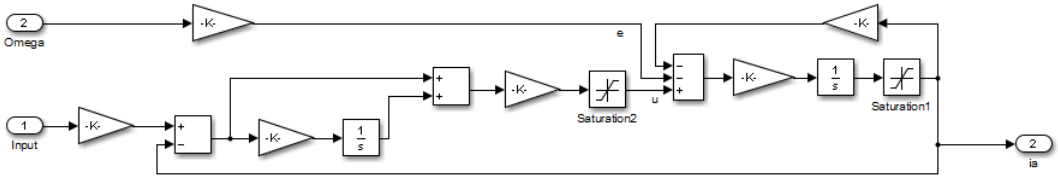
$i_d = K_k u_{inn}$ er konvertering av inngangsverdien som sendes fra mikrokontrolleren.



Figur 4.9: Motormodell uten regulator og med innkobling av motindusert spenning

Modellen har altså ideell regulator med innkobling av motindusert spenning. Om denne modellen skulle bli unøyaktig ønskes det fortsatt å undersøke om regulatormodellen som ble brukt i [13] kan gi bedre resultat. I denne modellen approksimeres ikke regulatoren til å være ideell, men regulatoren vil til en hver tid motvirke strømsløyfen.

$$\dot{i}_a = \frac{1}{L_a}(-K_e\dot{\theta}_1 - R_a i_a + (K_{pi} + \frac{1}{s}Kii)(i_d - i_a)) \tag{4.66}$$



Figur 4.10: Motormodell med regulering av i_a

Kapittel 5

Observerte målefeil

Det ble oppdaget en rekke måleavvik på systemet. På den horisontale armen ble det konsekvent målt en vinkelhastighet på $0.0016 rad/s$ når systemet står i ro. Det ble også observert at vinkelen til denne armen gikk fra $0 - 364.3°$.

Det ble oppdaget at systemet ikke stod helt i vater slik at det ble observert variabelt nullpunkt[1] på pendlene avhengig av posisjonen til den horisontale armen. På det meste kunne det måles opp til en grad avvik.

5.1 Feil i elektronikk for datainnsamling

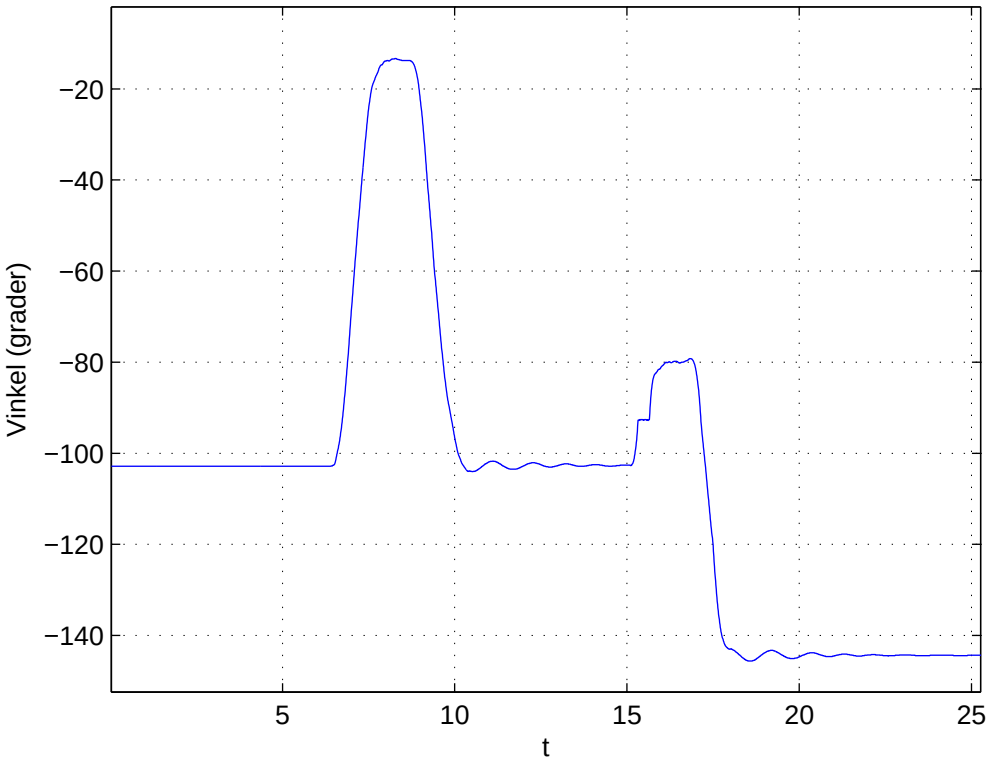
Et par måneder inn i prosjektet ble det oppdaget at demonstrasjonsregulatoren fra [5] ikke lenger fungerte slik den skulle. Systemet bestod her kun av to pendler[2] siden den siste pendelen fortsatt manglet festeoppheng. Demonstrasjonsprogrammet fra [5] startes automatisk når det blir satt strøm på systemet. Først skal den korte pendelen svinges opp og balanseres. Den korte pendelen slippes så ned igjen og deretter skal den lange pendelen svinges opp og balanseres. Programmet benyttet finjusterte PID-regulatorer som tidligere fungerte utmerket til balansering av enkeltpendler. problemet var at balanseringen av den korte pendelen ikke lenger fungerte. Pendelen ble svingt opp, men falt bare ned igjen. Ved kjøring av Simulinkmodellen fra [4] ble det observert at vinkelen til den korte pendelen ikke lenger ble målt til 0° når pendelen hang rett ned. Det var på dette tidspunkt ikke utført noen forandringer i mikrokontroller fra [5].

Det ble undersøkt hvorvidt feil nullpunkt kunne skyldes at det fortsatt lå strøm på mikrokontrolleren selv etter systemet var slått av slik at tilstander i systemet ble liggende til neste oppstart. Andre årsaker kunne blant annet skyldes feil i signal fra enkoderen eller feiltolking av signalet.

Det ble oppdaget at målefeilen oppstod etter systemet ble startet og ikke i initieringen. Ved å rotere armen manuelt ble det også oppdaget at feilen har en tilknytning til hastigheten som pendelen roterer med.

[1] Verdien som måles når pendelen henger rett ned

[2] System med pendel på hver side



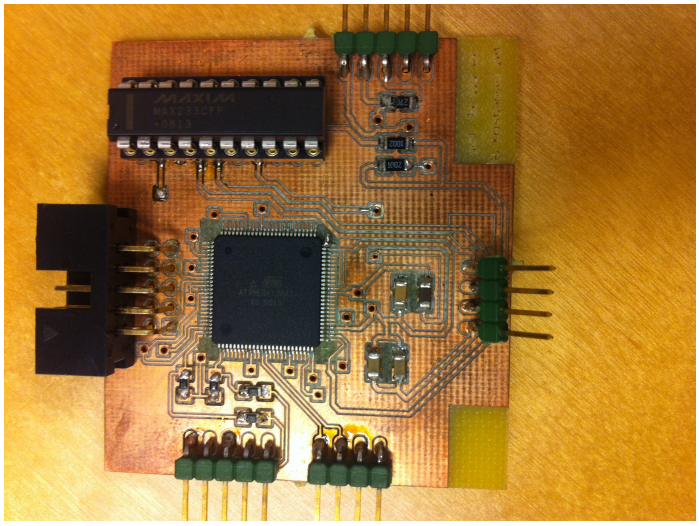
Figur 5.1: Forskyvning av nullpunkt

Plott i Figur 5.1 demonstrerer feilsymptomene. I starten av plottet måles det ca. $-100°$ når armen henger rett ned. Mellom $t = 5$ og $t = 10$ blir armen manuelt rotert opp til $90°$ og ned igjen. På dette punktet måles det fortsatt $-100°$ når armen henger rett ned. Mellom $t = 15$ og $t = 20$ roteres armen på nytt opp til $90°$, men med litt høyere hastighet. Man kan da se et trappetrinn i måling når hastigheten er på sitt høyeste. Etter at armen slippes ned igjen måles det ca. $-140°$ når armen henger rett ned. Det kan virke som at enkoderen hopper over punkt når hastigheten blir for høy. Ved nærmere fysisk undersøkelse ble det oppdaget slingring i feste til den pendelen som hadde målefeil. Et Kulelager satt løst og beveget seg litt når man beveget pendelen. Det ble satt på pakning i kulelageret og slingringen forsvant, men dette rettet ikke målefeilen.

Feilen opptrådte kun på den korte pendelen og ikke på den lange. For å isolere feilen ble ledningene fra de to enkoderene byttet om ved inngangen til kortet som mottar enkoderinformasjonen. Det ble nå observert at målefeilen opptrådte på den lange pendelen og ikke på den korte. Ettersom at det ikke var gjort noen forandringer i mikrokontrollerprogrammet ble det antatt at feilen lå i slavekortet[3]. Det skulle vise seg utfordrende å feilsøke slave-kortet ettersom at det ikke eksisterer noen dokumentasjon eller tegninger på det. I [5] ble det rettet noen loddefeil på kortet, men hvem som faktisk har laget det

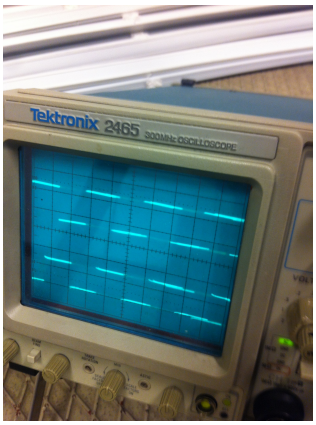
[3]Elektronikken som mottar enkoderdataen

er uvisst. I [8] ble det designet et nytt slave-kort, men Eagle-tegningene og avbildingen av kortet i rapporten samsvarer ikke med kortet som står montert på riggen.



Figur 5.2: Slave-kort

Det ble undersøkt om det kunne være noen åpenbare feil som brudd eller kortslutning. Loddingene ble gjennomgått og motstanden mellom kanalene på begge enkoderinngangene ble målt for å se om det var noen forskjell. Da det ikke ble funnet noen åpenbare feil ble kortet undersøkt med Oscilloskop. Skopet ble koblet til de to enkoderkanalene på den siden der feilen fremtrer for å se om det kunne oppdages noen uforventede spenningsforandringer.



Figur 5.3: Enkoderkanalene målt på oscilloskop

På skopet ser alt ut til å stemme og når målt vinkel igjen undersøkes fra PC ser feilen ut til å ha forsvunnet. Demonstrasjonsregulatoren ble testet og fungerte nå slik som den skulle uten at det ble funnet noen forklaring. Over de neste ukene dukket feilen opp og forsvant igjen ved jevne mellomrom, men til slutt ble målefeilen permanent.

Det ble bestemt at det ikke skulle brukes mer tid på å feilsøke denne elektronikken. Det var fortsatt mulig å måle begge pendelvinklene hver for seg ved å bytte om inngangene på kortet. Det ble dermed tatt en avgjørelse om å fokusere på å få laget en god simulator til å utvikle regulator i.

Kapittel 6

Parameterestimering og identifikasjon

Når det skal utvikles modellbasert regulator kreves det estimat av de ukjente parameterne i bevegelsesligningene. Noen av parameterne kan finnes med vekt og målebånd mens andre parametere ikke kan finnes like direkte. I [13] ble mange av parameterne estimert ved hjelp av ulineært kalmanfilter. Kalmanfilteret ga noe brukbare resultat slik at det var mulig å bruke den ganske robuste LQ-reguleringsmetoden til å balansere enkeltpendler. Det nevnes allikevel i [13] at resultatene er en del unøyaktig, spesielt når det gjelder parametrene for motoren. Under estimeringen ble det benyttet en forenklet lineær DC-motormodell som ikke tok hensyn til den ulineære friksjonen. I dette prosjektet ønskes det en mer nøyaktig modell slik at det kan settes opp en mer nøyaktig simulator. Motormodellen inkluderer derfor coulombfriksjon, stribeck-effekten og hysterese. Med denne modellen er det ganske mange ulineære parametere som må estimeres og det ulineære kalmanfilteret er rett og slett ikke en god nok metode. For å estimere motorparametere benyttes en metode for bruk av genetisk algoritme som har hatt suksess til estimering av ulineære motorparametere i [3].

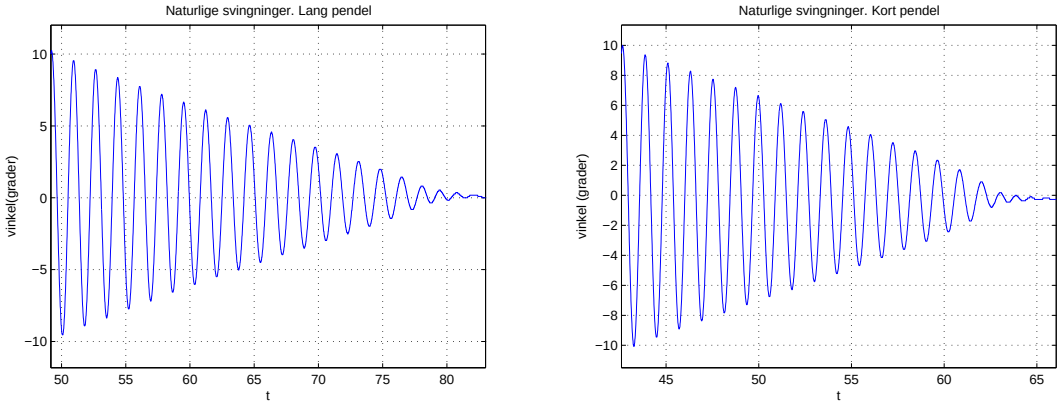
I motsetning til motoren kan mange av pendelparametrene lett finnes ved måling og utregning. I [10] ble det antatt at pendlene hadde uniform massefordeling slik at treghetsmomentet kunne finnes matematisk ved bruk av målt lengde og masse. I dette prosjektet ønskes det å unngå denne antagelsen slik at det kan finnes mer nøyaktige verdier for treghetsmoment og avstand til massesenter på pendlene. For å finne avstand til massesenteret ble pendlene balansert på en kant. balanseringspunktet ble markert og lengden ble målt. For pendlene benyttes en friksjonsmodell som tar hensyn til både luftmotstand og coulombfriksjon. Disse friksjonsparametrene er ganske enkle å finne manuelt ettersom at de forskjellige friksjonskomponentene dominerer ganske betydelig under forskjellige vinkelhastigheter. Luftmotstanden kan finnes ved å studere dynamikken under høy vinkelhastighet og coulombfriksjonen kan finnes ved å studere dynamikken under mindre vinkelhastigheter. På grunn av målefeilen som står beskrevet i Kapittel 5 var det ikke mulig å studere dynamikk til begge pendlene samtidig. Ved å bytte om enkoderinngangene på slavekortet var det derimot mulig å studere en pendel av gangen. De forskjellige isolerte delene av systemet identifiseres hver for seg. Motoren estimeres ved å demontere pendlene. Pendlene identifiseres ved å låse den horisontale armen.

6.1 Identifikasjon av kort pendel

Under Identifikasjon av pendlene benyttes modellen som ble utledet i kapittel 4.1. I modellen låses Arm1 ved å settes $\ddot{\theta}_1 = 0$ og $\dot{\theta}_1 = 0$.

6.2 Treghetsmoment

Formelen som ble utledet i teoridel 3.1.3 ble brukt til å finne treghetsmomentet til pendlene. Den naturlige periodetiden ble funnet ved å ta et snitt over flere svingninger med amplitude under 10°. Periodetidene som ble funnet for de to pendlene er



Figur 6.1: Små svingninger til utregning av treghetsmoment

$$T_{lang} = 1.714 \tag{6.1}$$
$$T_{kort} = 1.220 \tag{6.2}$$

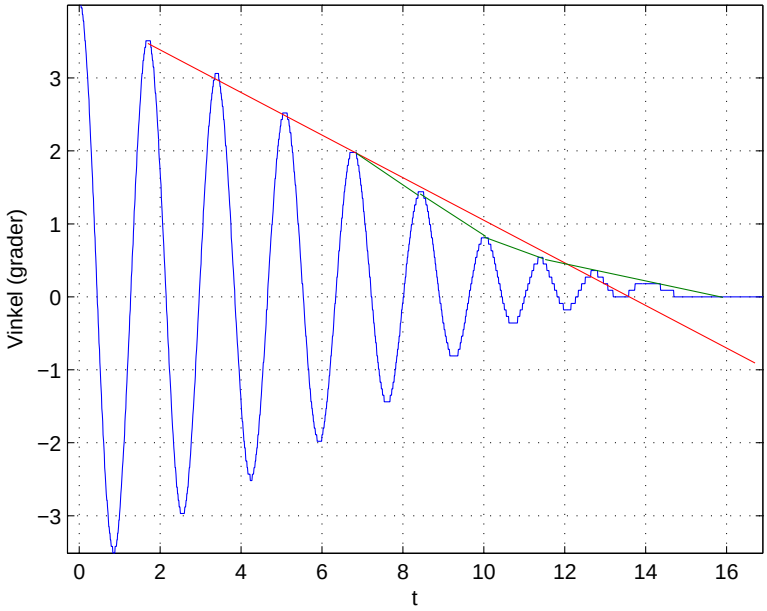
Treghetsmomentene blir

$$J2 = \frac{m_2 g l_2 T_{lang}^2}{4\pi^2} = 0.0735 \tag{6.3}$$
$$J3 = \frac{m_3 g l_3 T_{kort}^2}{4\pi^2} = 0.0057 \tag{6.4}$$

6.3 Identifikasjon av isolerte enkeltpendler

For å finne friksjonsparameterne som påvirker enkeltpendlene ble den horisontale armen låst slik at pendlene isoleres fra det fullstendige systemet. Pendlene ble så sluppet slik at de svinger fritt. Svingningsforløpene ble registrert ved hjelp av simulinkmodellen fra [4]. Den matematiske modellen for pendlene ble implementert i simulink og resultatet fra modellen sammenlignes med responsen fra det virkelige systemet. I denne delen av rapporten blir forklart hvordan den lange pendelen, Arm2, blir identifisert. Identifisering av kort pendel foregår på samme måte.

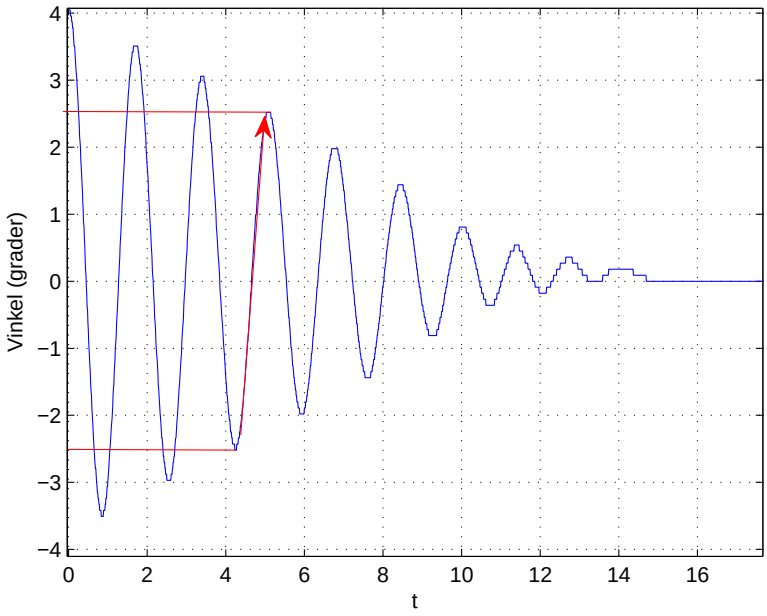
I første omgang blir den lange pendelen analysert med initialt utgangspunkt på 4.05°. Under dette svingningsforløpet vil ikke vinkelhastigheten være stor nok til at luftmotstanden dominerer. Coulomb-friksjonen vil i stedet dominere. Figur 6.2 viser hvordan amplituden



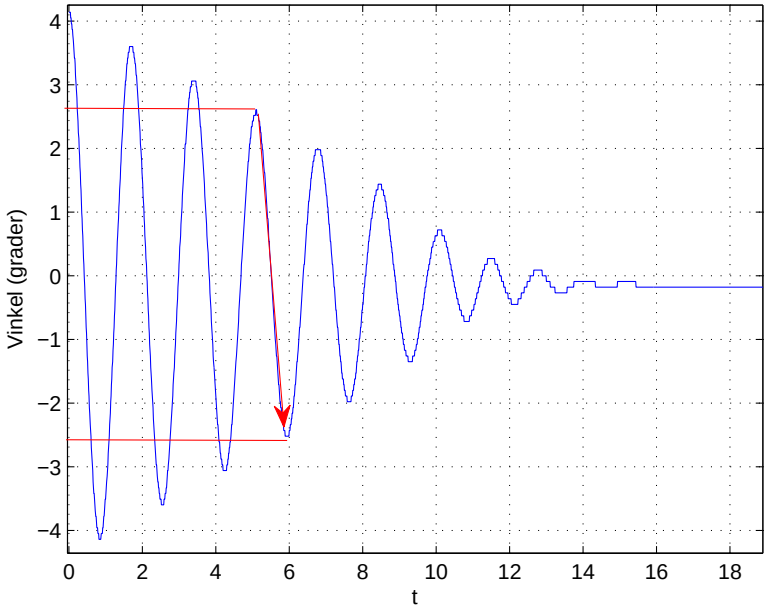
Figur 6.2: Svingningsrespons fra 4.05°

på svingningene avtar. I den første delen av svingningsforløpet observeres den lineære avtagelsen slik den tradisjonelle coulombfriksjonsmodellen beskriver. Når amplituden går under 2° mot slutten av forløpet observeres en invers eksponentiell avtagelse som minner mer om viskøs friksjon. På forløpet oppdages også et uforventet fenomen der dempingen under positiv flanke ser ut til å være veldig mye mindre enn dempingen under negativ flanke slik som vist Figur 6.3. Dette fenomenet kan være et tegn på hysterese i friksjonen. I Kapittel 5 blir det forklart at systemet ikke står helt i vater. Dette kan også være en forklaring på den usymetriske dempingen. For å undersøke dette nærmere utføres samme respons med 180° forskyvning på den horisontale armen. Dersom effekten skyldes gravitasjon vil det forventes at den usymmetrisk dempingen får omvendt effekt og blir mindre i negativ flanke enn i positiv. Dersom effekten skyldes hysterese vil det forventes samme resultat uavhengig av posisjonen til den horisontale armen.

Responsen som gjengis i Figur 6.4 viser at den usymmetriske dempingen antageligvis skylles påvirkning av gravitasjon som følge av at systemet ikke står i vater. Det oppdages at effekten kun har en liten påvirkning når amplituden på svingningene er ganske små, men at frekvensen på svingningene ikke ser ut til å bli påvirket i noen stor grad. Modellavviket ignoreres og parameterestimeringen fortsetter.



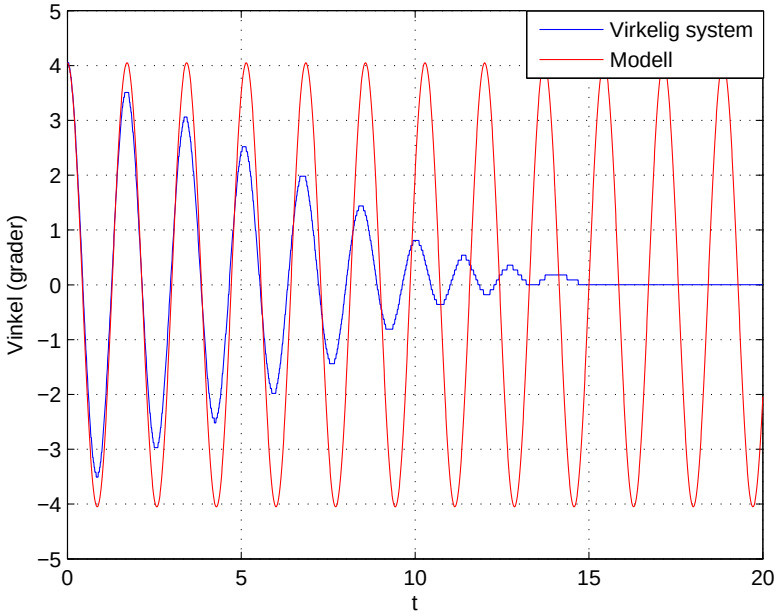
Figur 6.3: Manglende demping i positiv flanke. Figuren viser at amplituden har relativt samme verdi når pendelen svinger fra negativ vinkel til positiv



Figur 6.4: Manglende demping i positiv flanke. Figuren viser at amplituden har relativt samme verdi når pendelen svinger fra negativ vinkel til positiv

6.3.1 Identifikasjon av lang pendel under liten amplitude

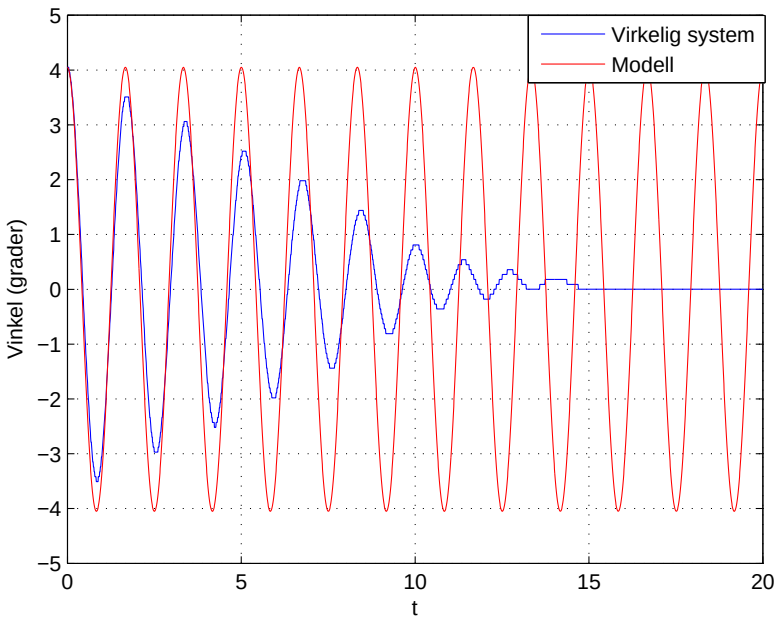
Coulombfriksjonen vil ikke påvirke svingningsfrekvensen slik den står beskrevet i Kapittel 4.5. For å undersøke hvorvidt dette stemmer i praksis blir modellen først simulert uten noen friksjonsmodell.



Figur 6.5: Svingningsrespons fra 4.05° uten friksjon

Fra Figur 6.5 kan man se at modellen har en tregere svingningsfrekvens enn det virkelige systemet. Til analyseringsformål ble parameteren for avstand til massesenter midlertidig justert opp for å øke frekvensen til svingningene i modellen. Resultatet er gjengitt i Figur 6.6.

Fra responsen kan det observeres at frekvensen til svingningene i modellen er noe raskere i starten av forløpet. Ved t=10 er svingningene synkronisert og det er tydelig at frekvensen i modellen er betydelig tregere mot slutten av forløpet når amplituden i det virkelige systemet går under 2°. Økning av frekvensen i det virkelige systemet ser ut til å ha en sammenheng med hva som ble observert tidligere når coulomb-friksjonsmodellen blir unøyaktig i slutten av svingningsforløpet. Friksjonsmodellen ser derimot ut til å kunne være en god approksimasjon når amplituden er større enn 2°.

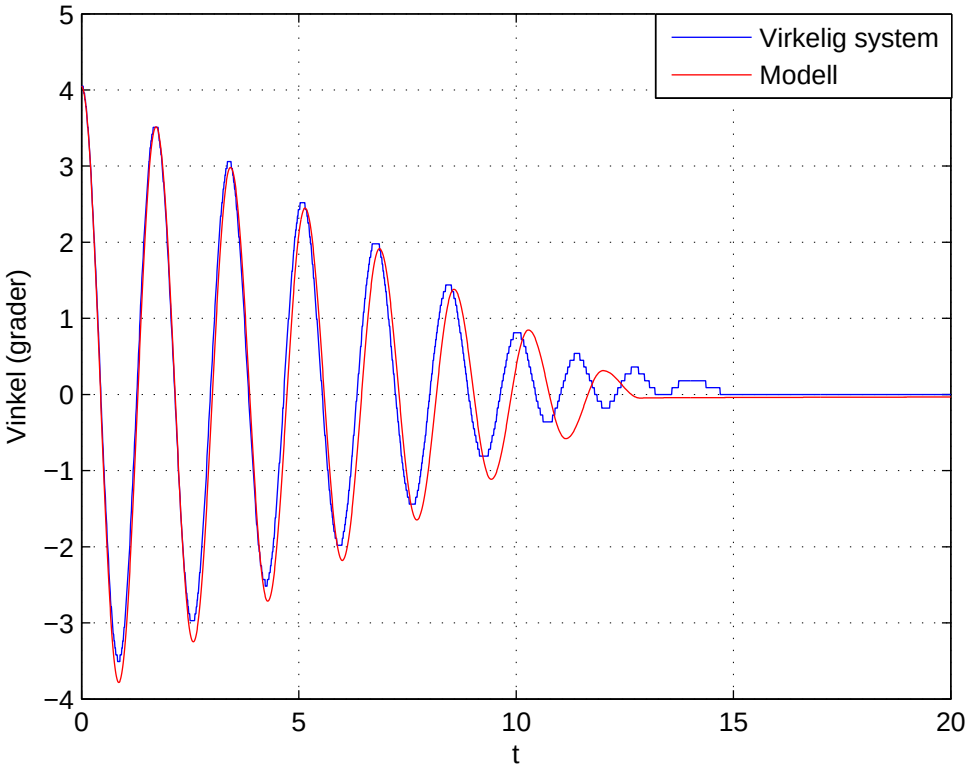


Figur 6.6: Svingningsrespons fra 4.05° uten friksjon og $l_2 + 0.03m$

Coulomb-friksjonen implementeres ved

$$\tau_f = b_c \text{sign}(\dot{\theta}) \tag{6.5}$$

Avstanden til massesenteret justeres tilbake slik at frekvensen blir riktig under første del av forløpet. Friksjonsparameteren finjusteres manuelt til $b_c = 0.0023$. figur 6.3.1 viser resultatet. Som forventet fungerer friksjonsmodellen fint under de første delene av forløpet men blir litt unøyaktig med hensyn til både amplitude og frekvens mot slutten av forløpet.



Figur 6.7: Svingningsrespons fra 4.05° med coulomb-friksjon

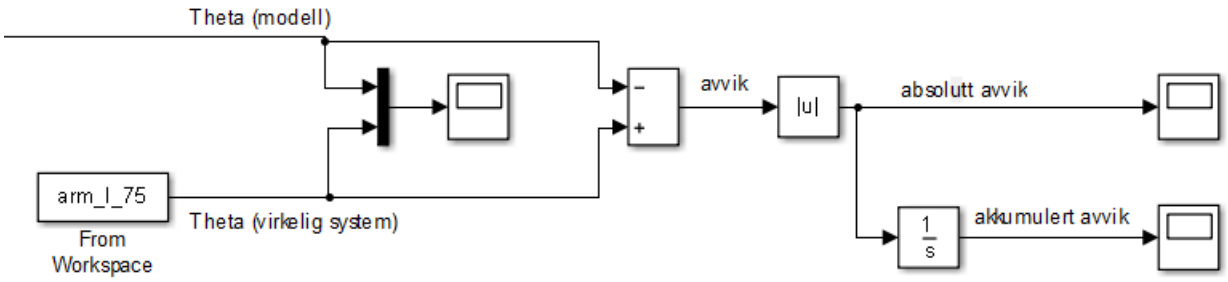
6.3.2 Identifikasjon av lang pendel under stor initialvinkel

Når vinkelhastigheten til pendelen er stor vil luftmotstanden dominere blant dempekomponentene. Luftmotstanden vil da skape både eksponentiell demping i amplitude og økning i svingningsfrekvens. Selv om luftmotstanden og coulomb-friksjonen vil dominere i hvert sitt område av svingningsforløpet, vil det fortsatt være noe overlapping. Små marginer kan da gjøre store utslag. For å finjustere friksjonparametrene benyttes det, i tillegg til visuell analyse, en analysering av det akkumulerte avviket mellom simulering og virkelig system. Denne analyseringsmodellen er implementert slik som vist i figur 6.3.2. ved å studere avviket kan simuleringesresponsen analyseres mer objektivt og det kan også gjøre det lettere å finne ut hvilke områder hvor modellen er mest unøyaktig.

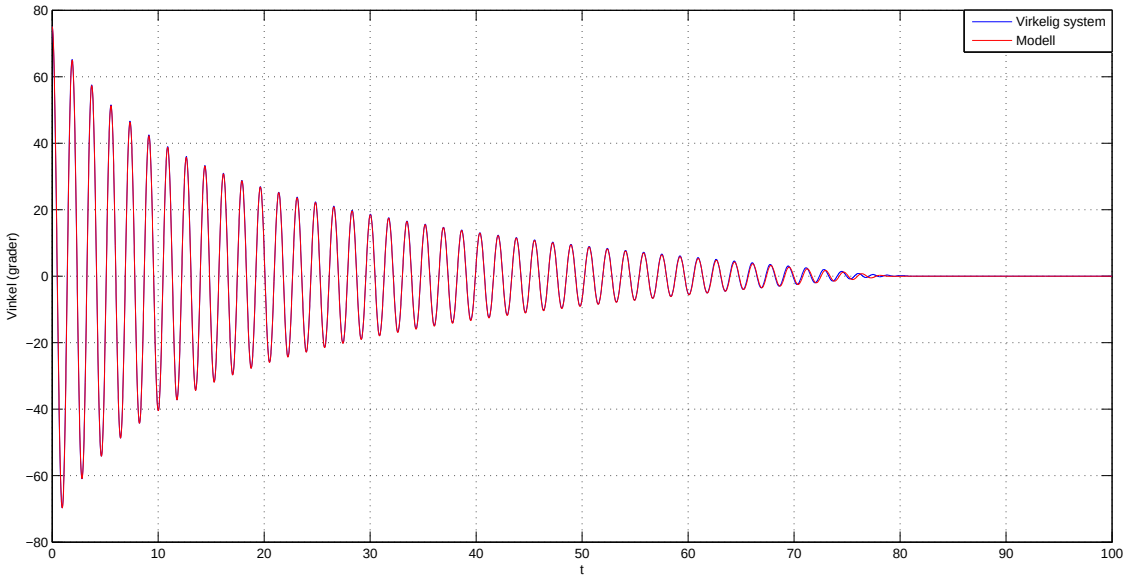
luftmotstanden implementeres ved

$$\tau_l = b_l \dot{\theta}|\dot{\theta}| \tag{6.6}$$

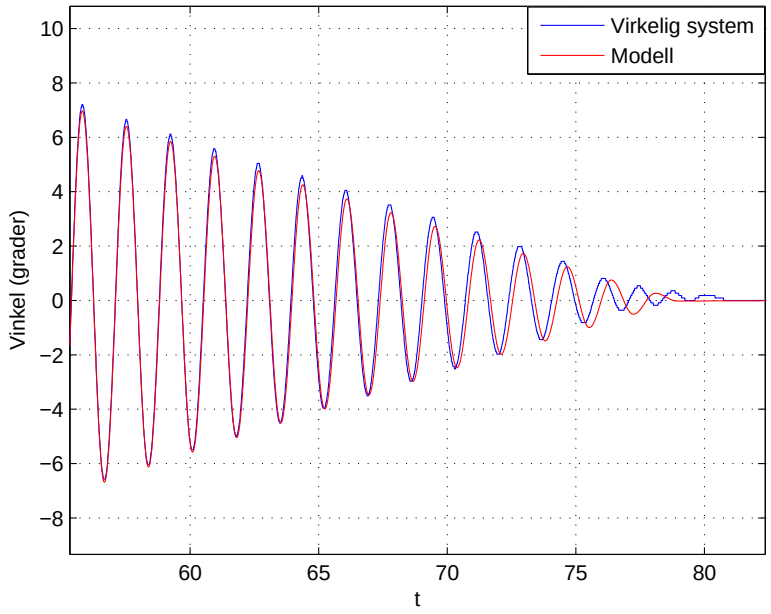
b_l justeres for å gi lavest mulig akkumulert avvik. Figur 6.9 viser den resulterende responsen. Responsen er for det meste ganske nøyaktig, men på slutten observeres den samme unøyaktigheten som tidligere.



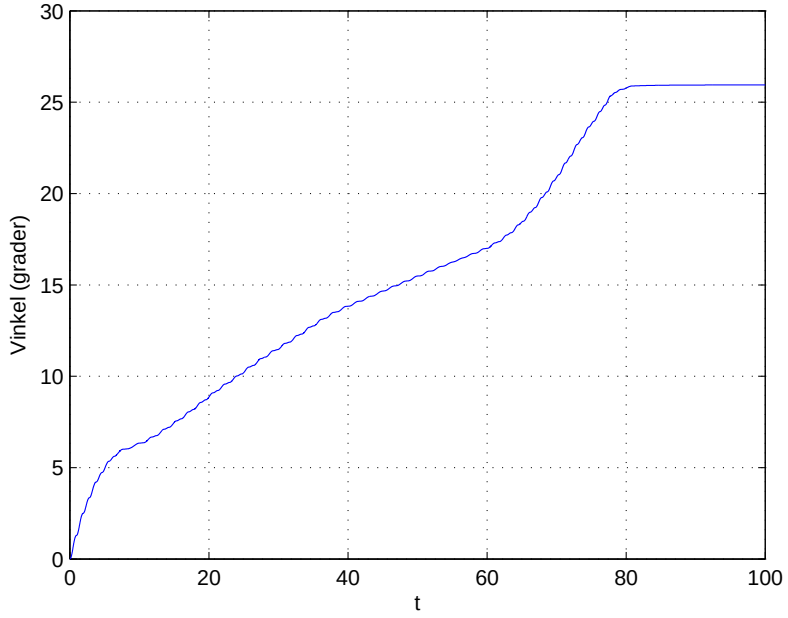
Figur 6.8: Modell for analysering av avvik



Figur 6.9: Svingningsrespons fra 75° med finjustert luftmotstand



Figur 6.10: Avslutningen på svingningsrespons fra 75°



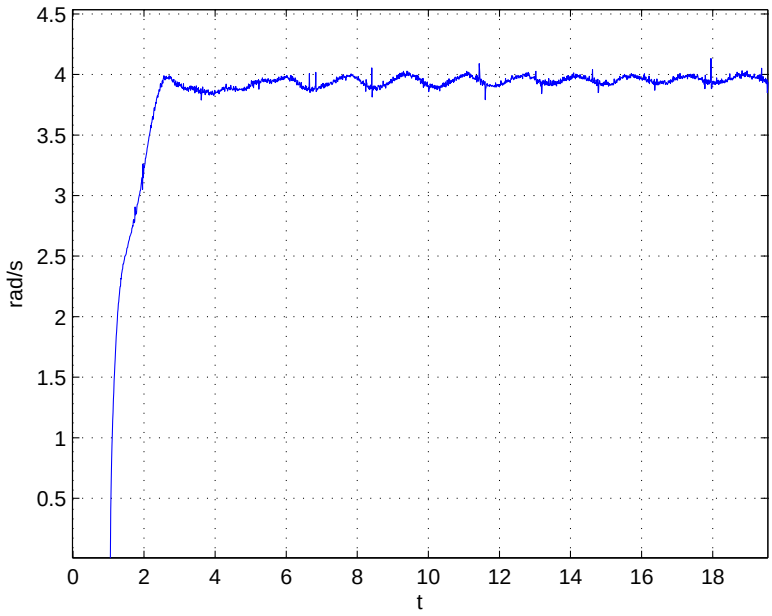
Figur 6.11: Akkumulert avvik i svingningsrespons fra 75°

6.3.3 Identifikasjon av modifisert luftmotstand

Pendlene vil påvirkes av luftmotstandskomponenter som er avhengig av vinkelhastigheten til motoren. Når motoren roterer med konstant hastighet vil ikke pendlene henge rett ned, men de vil få en stasjonær vinkel. Denne luftmotstanden er også avhengig av vinkelen på pendelen. Dersom pendelen står 90° vil denne luftmotstandskomponenten være på sitt laveste. Dersom pendelen står i invertert posisjon eller henger rett ned vil luftmotstandskomponenten være på sitt høyeste. I Kapittel 4.5 ble denne luftmotstanden beskrevet slik

$$b_{lm}\dot{\theta}_1^2 \cos(\theta_2) \tag{6.7}$$

For å finne en verdi for parameteren b_{lm} studeres systemet ved å benytte hastighetsregulatoren fra [4] som regulerer vinkelhastigheten til motoren. Motoren reguleres med $4rad/s$ referanse slik som vist i Figur 6.12. Fra figuren kan man se at hastigheten har stående svingninger rundt $3.9rad/s$ Fra Figur 6.13 kan man se at pendelen svinger rundt 33°



Figur 6.12: Vinkelhastigheten, $\dot{\theta}1$, til motoren under regulering av hastighet

når motoren svinger rundt $3.9rad/s$. Det antas at pendelen ville lagt seg på 33° dersom motoren hadde oppnådd en stasjonær vinkelhastighet på $3.9rad/s$. Dette kan brukes til å finne luftmotstandskoeffisienten. Med stasjonær vinkelhastighet på Arm1 og stasjonær vinkel på Arm2 er

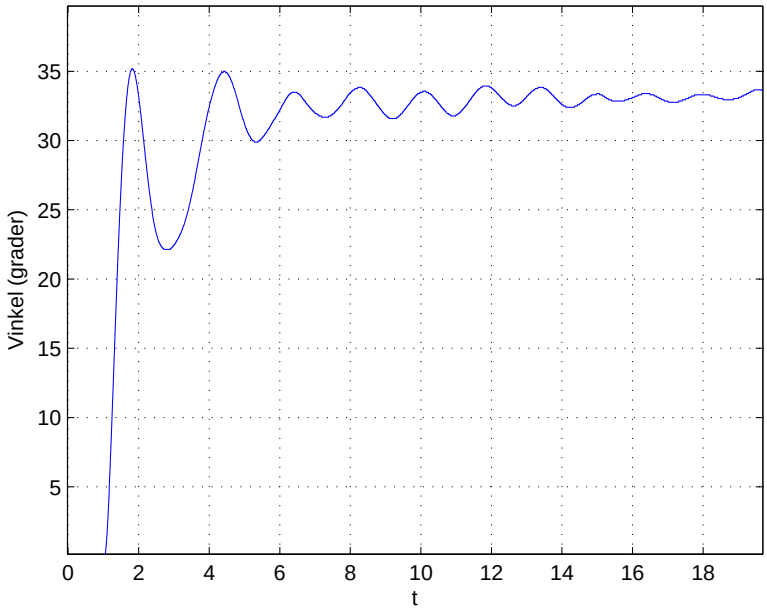
$$\ddot{\theta}_1 = \ddot{\theta}_2 = \dot{\theta}_2 = 0 \tag{6.8}$$

bevegelsesligningen for Arm2 blir da

$$-\dot{\theta}_1^2 \frac{1}{2} J_2 \sin(2\theta_2) + l_2 m_2 g \sin(\theta_2) + b_{lm}\dot{\theta}_1^2 \cos(\theta_2) = 0 \tag{6.9}$$

$$b_{lm} = J_2 \frac{\sin(2\theta_2)}{2\cos(\theta_2)} - \frac{l_2 m_2 g \sin(\theta_2)}{\dot{\theta}_1^2 \cos(\theta_2)} \tag{6.10}$$

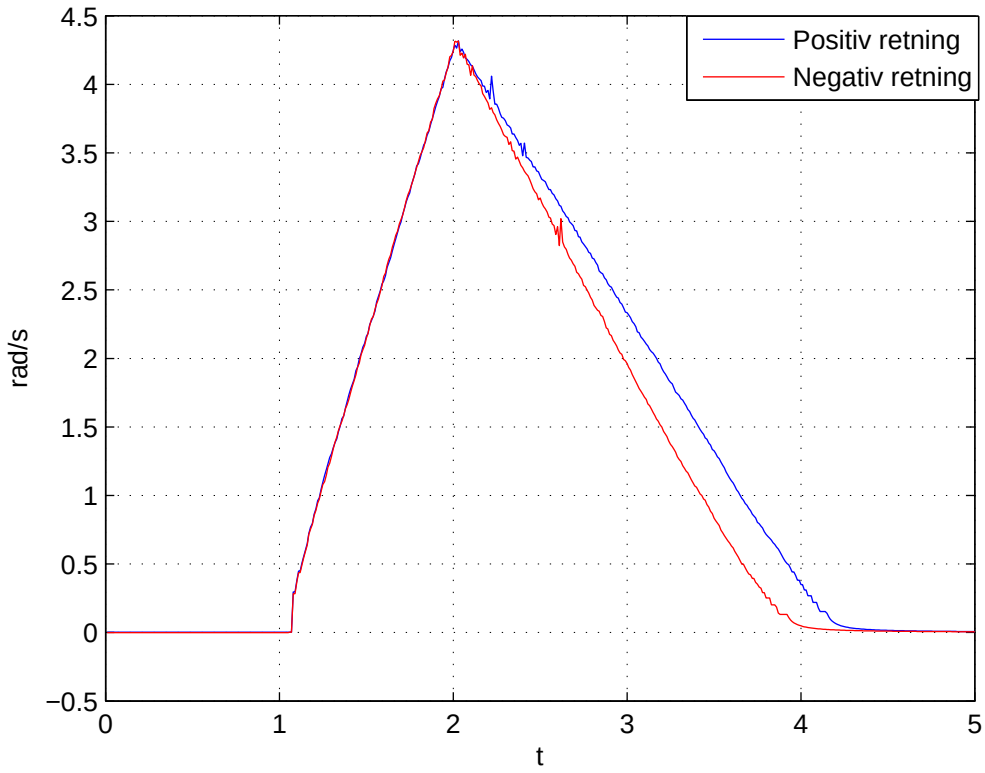
Ved å sette inn $\dot{\theta}_1 = 3.9$, $\theta_2 = 33 * \pi/180$ og verdi for modellparametrene kan det finnes at $b_{lm} = -2.14 \cdot 10^-3$



Figur 6.13: Vinkelen, $\theta 2$, for den lange pendelen under regulering av hastighet

6.4 Implementasjon av estimeringsalgoritme til estimering av motorparametre

Slik som beskrevet i Kapittel 4.6 og 4.7 har servomotoren en del ukjente parametre. Noen av parametrene, slik som spenningskonstanten og momentkonstanten, er oppgitt på merkeskiltet til motoren. I [13] ble det derimot funnet at disse konstantene i virkeligheten ikke stemte med hva som var oppgitt. Motoren er jo faktisk 25 år gammel og det kan jo godt hende at enkelte parametre forandrer seg noe over tid. Med hele servosystemet er det totalt 11 ukjente parametre som må identifiseres. I [1], [3] og [7] blir det forklart at likestrømsmotorer ofte opplever hysterese i friksjonen. For å undersøke om dette også er en betydelig realitet i baldormotoren utføres to like sprangresponser med forskjellig retning. For at det skal bli lettere å sammenligne disse responsene har fortegnet blitt snudd i plottet av den negative responsen.
Fra responsen i Figur 6.14 kan det observeres at motoren reagerer tilnærmet identisk på input i begge retninger. Når systemet ikke lenger mottar input ved t=2 kan det observeres at hastigheten dør ut med forskjellig demping avhengig av retning. Med hysterese i friksjonskomponentene blir det totalt 14 ukjente parametre som må estimeres. Med såpass mange ukjente parametere kan man se nødvendigheten med en estimeringsalgoritme. På grunn av ulineæritetene i modellen blir det vanskelig å ta i bruk estimeringsmetoder som benytter seg av gradient og derivasjon. Et alternativ er å benytte en søkealgoritme. Problemet med søkealgoritmer slik som Nelder-Mead er at de har en tendens til å konvergere



Figur 6.14: Sprangrespons av motoren i positiv og negativ retning. Sprang fra t=1 til 2. Plott viser tilstedeværelse av hysterese i friksjonen

mot nærmest lokale løsning. De krever derfor et godt startpunkt å jobbe fra. Med 14 ukjente parametere er det heller ingen lett oppgave å skulle foreslå et bra utgangspunkt uten videre. Som en mulig løsning ble det implementert en genetisk algoritme. se Kapittel 3.2. Denne algoritmen er dårlig på å konvergere, men er derimot god på å søke igjennom hele optimaliseringsområdet for mulige forbedringer. Resultatet kan så brukes som utgangspunkt for Nelder-Mead algoritmen. Dette er en treg estimeringsmetode som kun egner seg til offline-estimering, men kan brukes til å finne ganske gode estimat i et hvilket som helst system.

6.4.1 Definering av estimeringproblem

Estimeringsproblemet er av typen SISO med u som inngang og y som utgang. Problemet kan beskrives slik

$$\begin{aligned} \dot{\mathbf{x}} &= f(\mathbf{x}, \mathbf{z}, u) \\ y &= C\mathbf{x} + h \end{aligned} \tag{6.11}$$

der f er en ulineær funksjon, $\mathbf{x}$ er tilstandsvektoren til systemet, C er koeffisientmatrisen og h er målestøy. Oppgaven er å identifisere parametrene

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} \tag{6.12}$$

ved å benytte inngangen u og utgangen y til det virkelige systemet

6.4.2 Implementasjon av genetisk algoritme

Den genetiske algoritmen er implementert i Matlab og kjører simuleringene i Simulink. koden er oppgitt i Vedlegg E. Ved initiering av algoritmen opprettes det en n*m-matrise, K_g, der n er antall parametre som skal estimeres og m er antallet parametersett i befolkningen. Parametersettene i K_g genereres stokastisk innenfor optimaliseringsområdet.

Rangering De forskjellige parametersettene i K_g simuleres i simulinkmodellen og resultatene vurderes etter funksjonen

$$F_g(i) = \sum_{j=1}^{N} (\hat{Y}(i) - Y(i)) \tag{6.13}$$

for parametersett nr. i hvor N er antall datapunkt i simuleringen og $\hat{Y} - Y$ er avviket mellom simulering og virkelig system. Parametersettene sorteres så fra best til dårligst ved hjelp av Mergesort-algoritmen.

Utvalg For å velge ut parametersett til å skape neste generasjon benyttes den normaliserte geometriske funksjonen

$$P_i = q'(1-q)^{r-1} \tag{6.14}$$

som tildeler en utvelgelsessannsynlighet til hvert parametersett avhengig av rangering r der $r = 1$ er den beste rangeringen. I funksjonen er q den ønskede sannsynligheten for å velge beste parametersett. For at summen av alle utvelgelsessannsynlighetene skal bli 1 settes

$$q' = \frac{q}{1-(1-q)^m} \tag{6.15}$$

der m er er antall parametersett. Det genereres så m-antall tilfeldige tall, $U(0,1)$, som en etter en sammenlignes med de kumulative sannsynlighetene

$$C_i = \sum_{j=1}^{i} P_j \tag{6.16}$$

Dersom $C_{i-1} < U < C_i$ velges parametersett nr. i. På den måten er det også mulig at de samme parametersettene bli valgt flere ganger inntil det har blitt valgt et sett for hver $U(0,1)$.

Mutasjon De nye parametersettene gjennomgår binærmutasjon slik at hvert eneste bit i hver eneste parameter har en sannsynlighet, p_m, for å bli tooglet. For å utføre denne mutasjonen opprettes det ganske simpelt en ny sekvens bit med lengde(a) for hver parameter, a, der hvert bit har en p_m-sannsynlighet for å bli en 1-er og blir ellers valgt til 0. Det utføres deretter bitvis-eksklusiv-eller-operasjon mellom denne bit-sekvensen og parameteren som skal muteres.

Krysning Hvert utvalgte parametersett har en sannsynlighet, p_c, for å bli valgt ut til krysning. De som ikke blir valgt ut gjennomgår kun mutasjon. De utvalgte parametersettene blir satt sammen i par og informasjonen i hver parameter blir krysset på binært nivå. Under krysningen velges det ut en tilfeldig plassering hvor informasjonen mellom de to parametersettene byttes om.

Terminering av algoritme Når mutasjon og krysning er utført starter loopen på nytt med neste generasjon. Etter k generasjoner avsluttes algoritmen og den nyeste generasjonen rangeres.

Valg av estimeringsparametre Som forklart i Kapittel 3.2 finnes det ingen god metode for å velge estimeringsparamtrene. Disse har blitt valgt etter generelle anbefalinger.

	Parameter	Verdi
Befolkning	m	80
Antall generasjoner	k	100
Mutasjonssannsynlighet	p_m	0.1
Krysningssannsynlighet	p_c	0.6
Utvelgelsessannsynlighet	q	0.08

Tabell 6.1: Estimeringsparametre for genetisk algoritme

6.4.3 Implementasjon av Nelder-Mead

Matlab-koden for Nelder-Mead-algoritmen er oppgitt i Vedlegg F. Fra den siste generasjonen av den genetiske algoritmen, velges de n+1 beste parametersettene til å danne simplexen som skal brukes i Nelder-Mead algoritmen. Algoritmen benytter eksakt skalering av refleksjonspunktet slik at refleksjonsoperasjonen kun flytter simplexen uten å forandre størrelsen. Ekspansjons og krympekoeffisientene velges slik at simplexen ekspanderer raskt når den finner et bra parametersett og krymper sakte når den skal konvergere. Som følge av dette blir algoritmen veldig treg, men finner til gjengjeld gode estimat. Som termineringskriterie benyttes

$$\sqrt{\frac{1}{n+1}\sum_{i=1}^{n+1}(||P_i - P_1||)^2} \leq 10^{-\epsilon} \tag{6.17}$$

hvor n er antall parametre som skal estimeres, P er simplexen, P_1 er det beste punktet i simplexen og ϵ er presisjon i antall desimaler. Dette stoppkriteriet sjekker konvergensen til algoritmen. Dersom alle punktene i simplexen har samme verdi med opp til ϵ-desimaler nøyaktighet blir algoritmen avsluttet.

	Parameter	Verdi
Refleksjonskoeffisient	ρ	1
Ekspansjonskoeffisient	μ	2
Sammentrekningskoeffisient	λ	0.75
Krympekoeffisient	σ	0.99

Tabell 6.2: Estimeringsparametre for genetisk algoritme

Kapittel 7

Regulator

I prosjektet ble det utviklet regulator til oppsving og balansering av enkeltpendelsystemet med kort pendel. Denne delen av rapporten forklarer implementeringen av regulatorene. For å svinge opp pendelen benyttes energiregulering slik at den potensielle energien til pendelen i invertert tilstand kan brukes som referanse. Dette er en metode som har fungerte bra til oppsving under tidligere prosjekt på systemet.

I [10] og [13] benyttet LQ-regulator til å balansere pendlene. I [10] styres systemet kun i simulator. Her blir resultatene litt urealistisk siden utgangen på regulatoren benyttes direkte som påtrykt moment i pendelligningene uten noen motormodell i mellom. Modellen for det fullstendige systemet med motor og pendel ble beskrevet i [13]. Har ble regulatoren implementert på det virkelige systemet med noe dårlig resultat under balansering av system med flere pendler. Det rapporteres at balanseringsproblemene skyldes dårlige tilstandsestimat og ikke reguleringsmetoden som er brukt. Det konkluderes likevel med at andre reguleringstrategier, slik som MPC[1], kan være fordelaktig siden LQR krever kraftige tilbakekoblinger fra tilstandene. Det forklares ikke hvorfor det skal være problematisk med kraftig forsterkning i tilbakekoblingene. I [8] ble det forsøkt implementert en MPC til balansering av pendel i simulator. Det konkluderes med at denne metoden er utilstrekkelig, men at en ulineær MPC[2], kanskje kan fungere. I følge [8] vil nMPC også benytte kraftige forsterkninger i tilbakekoblingene.

På grunn av den tidligere suksessen med bruk av LQR blir denne reguleringsstrategien også implementert til balansering av enkeltpendel i dette prosjektet. På grunn av tidsbegrensninger ble det ikke utviklet regulator til balansering av flere pendler. For å benytte LQR er det nødvendig å linearisere bevegelsesligningene. For enkeltpendelsystemet kan dette fint gjøres for hånd, men som man kan se fra Vedlegg B blir ligningene ganske store under dobbel- og trippel-pendelsystemet. Her er det antageligvis best å utnytte datakraft for å linearisere ligningene.

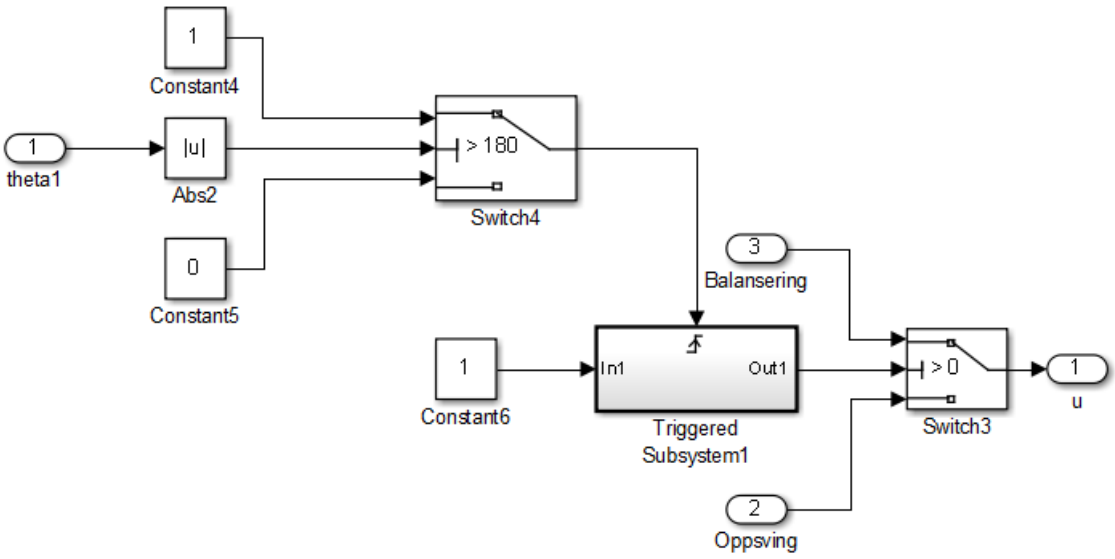
For å demonstrere bruksområdet til den genetiske algoritmen som er benyttet til parameterestimering, ble denne algoritmen også brukt til å finne regulatorparametre i simulator som alternativ til LQR. Denne metoden er fryktelig lett å implementere og krever ingen

[1] Model predictive control
[2] nMPC

modifikasjon av algoritmen slik som den er skrevet. Fordelen med metoden er at man ikke trenger bruke tid på å linearisere ligningene. Ulempen er at man nødvendigvis må bruke metoden i en simulator. Det ville ellers tatt alt for lang tid å finne regulatorparametrene i sanntid på et virkelig system. Parametrene som blir funnet kan selvfølgelig benyttes til å balansere pendlene i det virkelige systemet gitt at simulatoren er en god nok analog.

7.1 Oppsvingsregulator

Dersom pendelen svinges opp i kun ett trinn, vil energien måtte nå referansen før pendelen oppnår 90°. Dette vil kreve en høy vinkelakselerasjon for motoren og vinkelhastigheten kommer til å være ganske høy etter at pendelen har svingt opp. En bedre strategi vil være å utføre oppsving i to trinn slik at pendelen først svinges opp til 90°, deretter snur motoren retning slik at pendelen svinges opp til 180°. På den måten vil motoren ha en relativt liten hastighet når pendelen står i invertert tilstand. Dette vil gjøre jobben lettere for balanseringsregulatoren. Etter at pendelen har svingt opp må oppsvingsregulatoren kobles ut og balanseringsregulatoren kobles inn. For å utføre dette benyttes en switch hvor utgangen går fra 0 til 1 dersom vinkelen er 180°. På utgangen til switch-en benyttes det flanketrigging for å holde en konstant høy verdi. Denne verdien brukes deretter som kriterie for bytte mellom regulator. På denne måten vil ikke oppsvingsregulatoren kobles inn igjen dersom balanseringsregulatoren feiler.



Figur 7.1: Bytte mellom oppsvingsregulator og balanseringsregulator

7.2 LQR

Først ble ligningene linearisert rundt arbeidspunktet

$$a = \begin{bmatrix} \theta_{1e} \\ \theta_{3e} \\ \dot{\theta}_{1e} \\ \dot{\theta}_{3e} \end{bmatrix} = \begin{bmatrix} 0 \\ \pi \\ 0 \\ 0 \end{bmatrix} \tag{7.1}$$

slik at den lineariserte modellen blir

$$\dot{x} = Ax + Bu \tag{7.2}$$

$$A = \frac{1}{J_1 J_3 - m_3^2 l_1^2 l_3^2} \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & g m_3^2 l_3^2 l_1 & -b_1 J_3 & 0 \\ 0 & g m_3 l_3 J_1 & b_1 m_3 l_1 l_3 & 0 \end{bmatrix} \tag{7.3}$$

$$B = \frac{1}{J_1 J_3 - m_3^2 l_1^2 l_3^2} \begin{bmatrix} 0 \\ 0 \\ J_3 \\ -m_3 l_1 l_3 \end{bmatrix} \tag{7.4}$$

$$x = \begin{bmatrix} \theta_1 \\ \theta_3 - pi \\ \dot{\theta}_1 \\ \dot{\theta}_3 \end{bmatrix} \tag{7.5}$$

$$u = i_d \tag{7.6}$$

Regulatoren settes opp

$$u = -Kx \tag{7.7}$$

der

$$K = \begin{bmatrix} K_1 & K_2 & K_3 & K_4 \end{bmatrix} \tag{7.8}$$

K ble så funnet i Matlab ved å minimere kostfunksjonen

$$\int x^T Q x + R u^2 \mathrm{d}t \tag{7.9}$$

hvor Q og R velges

$$Q = \begin{bmatrix} 0.1 & 0 & 0 & 0 \\ 0 & 1000 & 0 & 0 \\ 0 & 0 & 1000 & \\ 0 & 0 & 0 & 100 \end{bmatrix} \tag{7.10}$$

$$R = 0.1 \tag{7.11}$$

7.3 Utvikling av regulator ved hjelp av genetisk algoritme

Når den genetiske algoritmen brukes til å finne regulatorparametrene kan man se på det som at programmet "lærer" seg å balansere pendlene. Metoden ble brukt på simulatoren for enkeltpendelsystemet med kort pendel. Det ble benyttet samme regulatorstruktur som med LQR-metoden

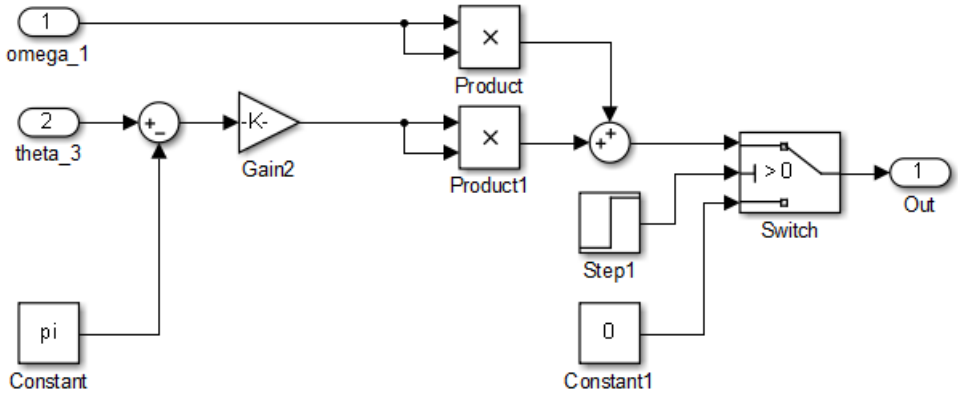
$$u = -Kx \tag{7.12}$$

Posisjonen til den horisontale armen er urelevant til balansering av pendel. Siden det ikke er ønskelig å regulere denne tilstanden settes $K_1 = 0$. Dermed er det bare 3 parametere som algoritmen må finne.

Man vet i utgangspunktet ikke så mye om hvordan regulatorparametrene burde være. Siden det er ønskelig å legge størst vekt på at $\theta_3 \to \pi$ burde $|K_2|$ antageligvis være høyere enn de andre parametrene. Optimaliseringsområdet settes til

$$K_2 \in [-1000, 1000] \tag{7.13}$$
$$K_3 \in [-200, 200] \tag{7.14}$$
$$K_4 \in [-200, 200] \tag{7.15}$$

Det siste som gjenstår da er å ordne utgangen i simuleringsmodellen. Det er ønskelig at regulatoren minimerer $|\theta_3 - \pi|$ og $\dot{\theta}_1$. Utgangen på simulatoren ble derfor satt opp slik som vist i Figur 7.2. For å unngå store utgangsverdier under oppsvingsdelen ble det brukt en switch som setter verdien 0 på utgangen under oppsvingningen. Dermed blir objektiv funksjonsverdi veldig liten under de stabile løsningene i forhold til de ustabile. For å prioritere minimering av $\theta_3 - \pi$ benyttes det en forsterkning.



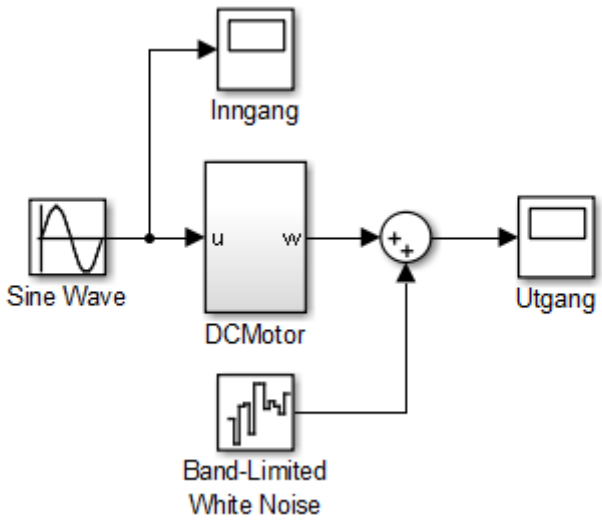
Figur 7.2: Utgang på simuleringsmodell som benyttes til den genetiske algoritmen sin objektive funksjon

Kapittel 8

Resultater

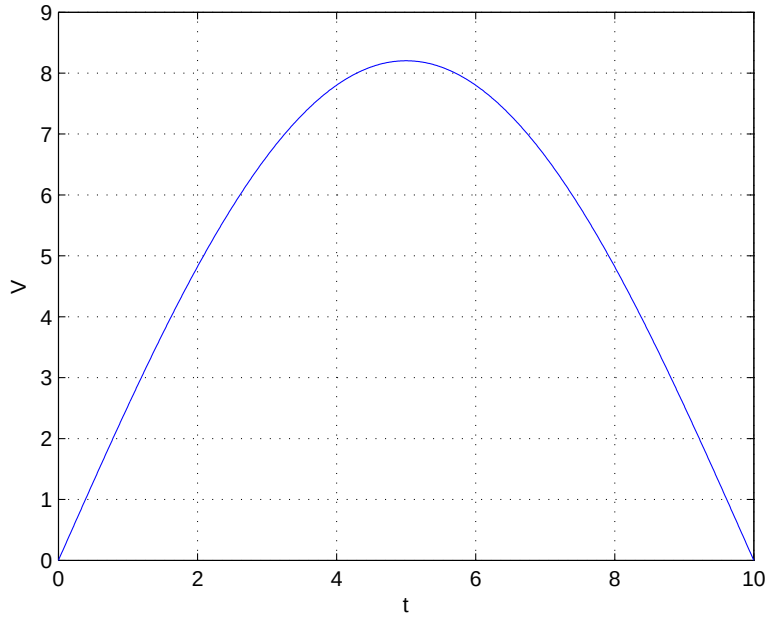
8.1 Verifisering av estimeringsalgoritmen

For å verifisere om den implementere algoritmen fungerer like bra som i [3] blir det satt opp en modell med de samme parametrene som ble funnet i [3]. Det blir lagt til hvit støy på utgangen for å gjøre forsøket mer realistisk. Det ønskes ikke bare å undersøke hvor bra algoritmen konvergerer, men også hvor lang tid det tar og hvor mange iterasjoner som utføres.

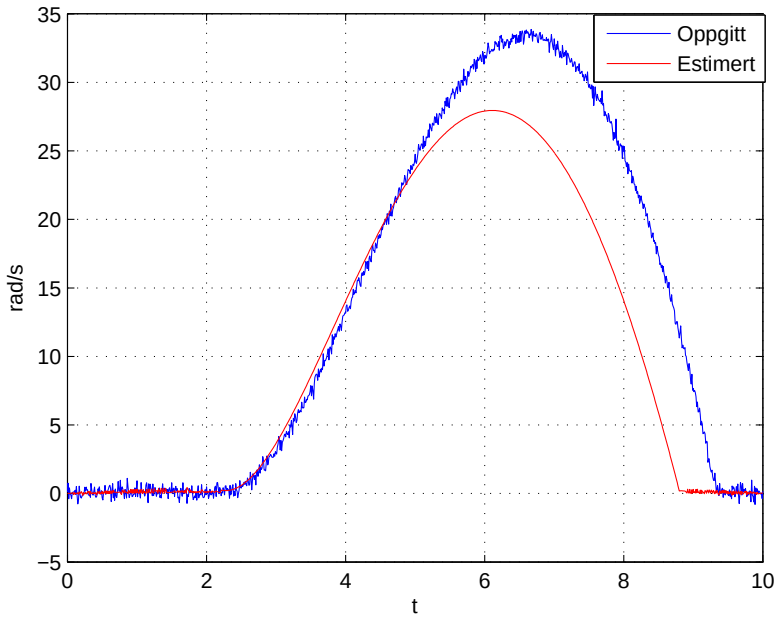


Figur 8.1: Likestrømsmotormodell for oppretting av testdata under verifisering av algoritmen

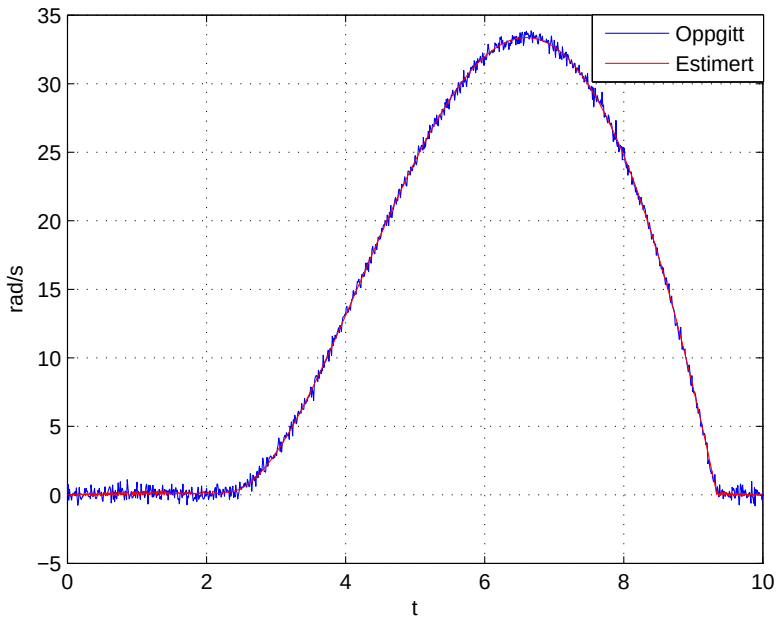
På inngangen brukes samme signal som ble brukt under estimering i [3].



Figur 8.2: Inngangsignal



Figur 8.3: Resultat av etter genetisk algoritme



Figur 8.4: Resultat av etter Nelder-Mead

Resultatene viser at estimeringsalgoritmen finner like bra estimat som i [3] og med færre iterasjoner. Nelder-Mead algoritmen konvergerer etter rundt 1200 iterasjoner i motsetning til de 1500 som utføres [3]. Algoritmen utfører tilsammen rundt 14 000 simuleringer og bruker en total tid på ca. 2400s. Dette er mer enn dobbelt så mye tid som programmet bruker i [3]. Algoritmen vil trolig bruke mindre tid dersom den kjøres fra en raskere datamaskin enn den som er brukt i prosjektet.

Algoritmen finner parametre innenfor optimaliseringsområdet som ble definert i [3], men verdien av de estimerte parametrene er noe anderledes. Dette kan tyde på at inngangsignalet ikke er rikt nok slik at det finnes flere gode løsninger. Ved å sammenligne plot i Figur 8.4 og fra [3] er det umulig å si hvilke estimerte parametersett som er best. I begge tilfeller har algoritmen utført oppgaven sin slik den skal.

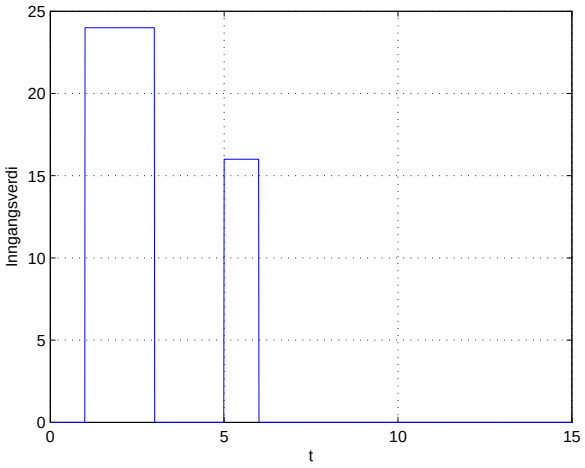
Selv om algoritmen bruker en del tid viser resultatene at den kan benyttes til å estimere parametre i ulineære systemer hvor det ikke er mulig å regne ut gradient av bevegelsesligningene slik som kreves av de fleste optimale estimatorer. Det eneste som kreves er en nøyaktig modell til å utføre simuleringene i. Begrensningene for metoden ligger i tiden algoritmen bruker. Denne tiden er avhengig av antall simuleringer som utføres og tiden simuleringene bruker. Den genetiske algoritmen utfører like mange simuleringer uavhengig av antall parametre som skal estimeres og er dermed kun begrenset av simuleringstiden. Nelder-Mead vil derimot utføre et antall simuleringer som er veldig avhengig av antallet parametre som skal estimeres. På grunn av disse tidsbegrensningene vil det være upraktisk å bruke metoden på trege systemer med mange datapunkter. Motordynamikken i dette prosjektet er derimot ganske raskt slik at det ikke kreves mer enn 10-15s med 10ms samplingstid for å fange en god respons til inngangen.

8.2 Estimering av motorparametre

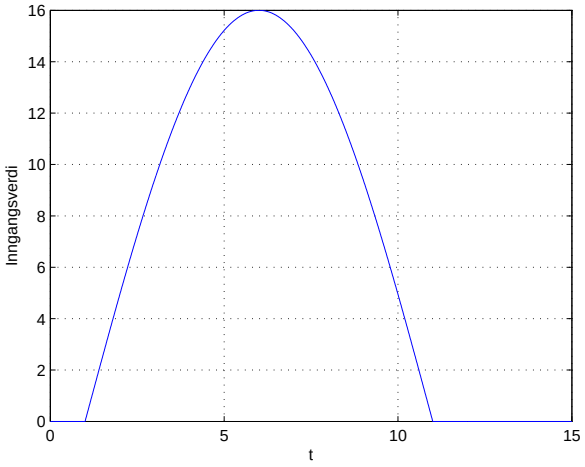
Motorparametrene ble estimert med en firkantpuls på inngangen.

$$i_d = 24u(t-1) - 24u(t-3) + 16u(t-5) - 16u(t-6) \tag{8.1}$$

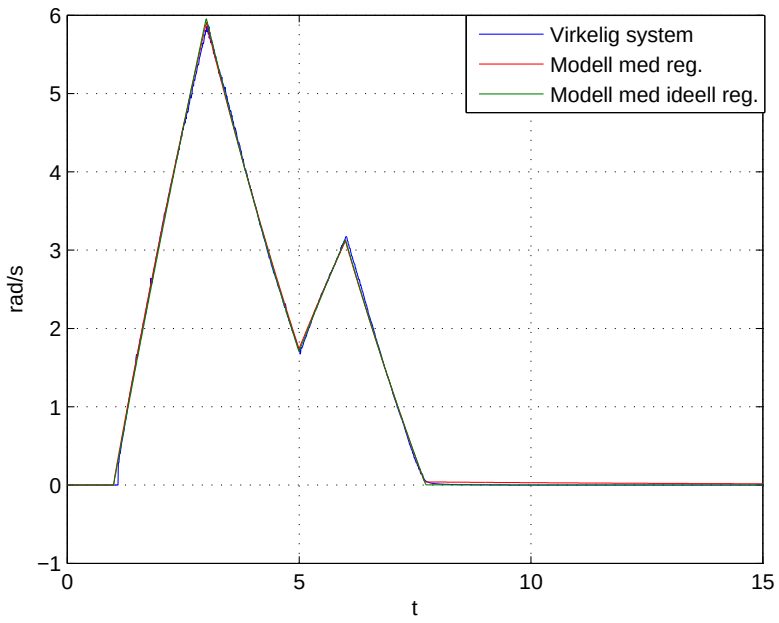
Det ble deretter brukt et sinussignal for å teste de estimerte modellene med en annen inngang.



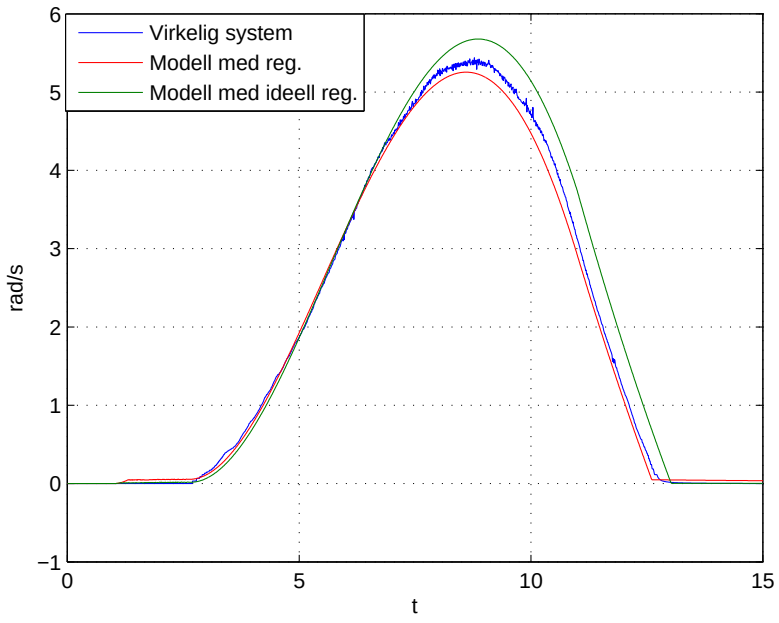
Figur 8.5: Firkantpulstog under estimering av motorparametre



Figur 8.6: Sinussignal til test av de estimerte modellene



Figur 8.7: Respons under estimering av to to motormodellene

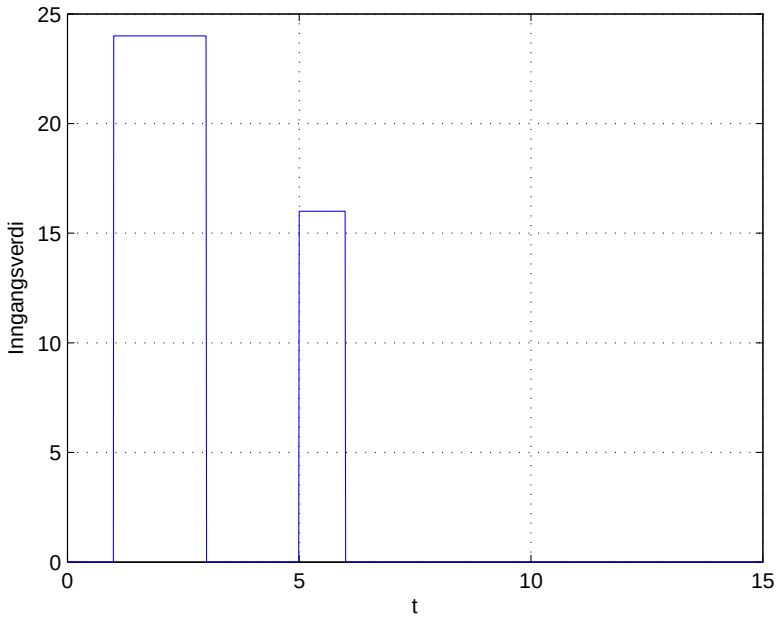


Figur 8.8: Respons av estimerte motormodeller med sinus på inngangen

Resultatene viser at begge motormodellene fungerer som approksimasjoner. Modellen med ideell regulator og som kobler inn og ut strømsløyfen gir noe dårligere resultat.

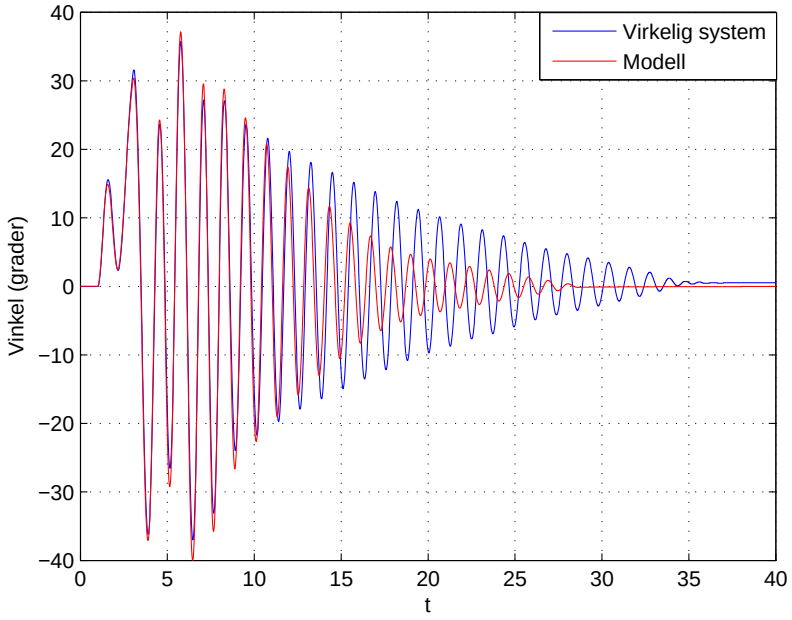
8.3 Enkeltpendelmodell

Motormodellene ble satt sammen med modellen for kort pendel som ble identifisert ved hjelp av å låse den horisontale armen. Den totale dynamikken ble så sammenlignet med det virkelige enkeltpendelsystemet ved å påtrykke en firkantpuls.

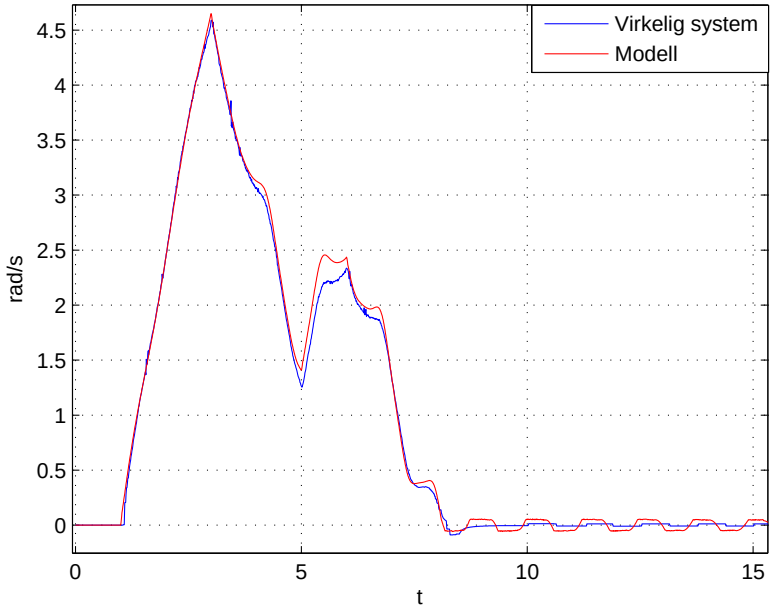


Figur 8.9: Inngangsignal

8.3.1 Motormodell med strømregulator

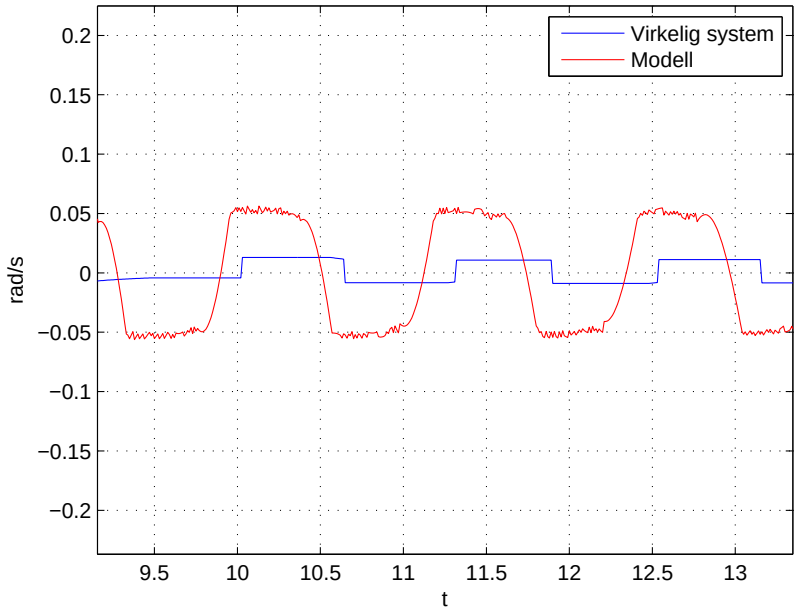


Figur 8.10: Vinkel, Kort pendel. Respons for enkeltpendelsystem med strømregulator



Figur 8.11: Vinkelhastighet, Arm1. Respons for enkeltpendelsystem med strømregulator

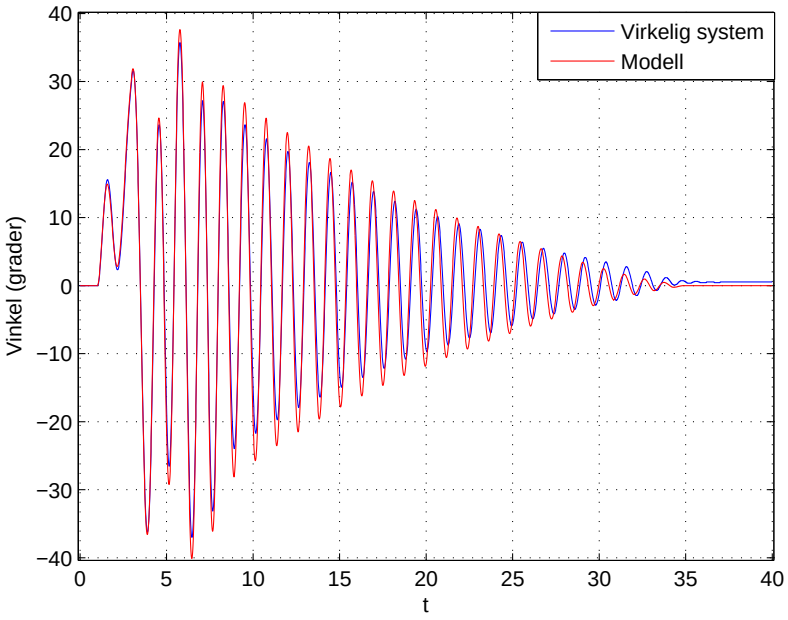
Fra Figur 8.10 kan det observeres at responsen for vinkelen til den korte pendelen begynner ganske lovende. Når vinkelhastigheten til Arm1 er på sitt høyeste ved $t = 3$ kan man se at pendelen svinger opp til ca. 30° slik som i det virkelige systemet. Dette tyder på at den utregnede luftmotstandkomponenten stemmer. Ved $t = 10$ opplever pendelen alt for stor demping i modellen og faller dermed til ro mye raskere enn i det virkelige systemet. I Figur 8.10 kan det observeres ved $t = 10$ at dødsonen til motormodellen er mye større enn i det virkelige systemet. Under estimeringen av motorparametre hadde ikke systemet



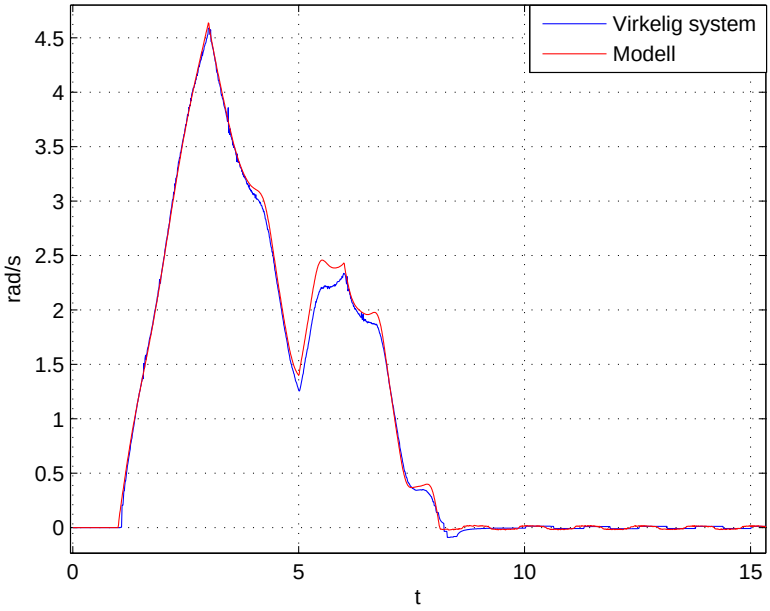
Figur 8.12: Dødsone for motoren

noen pendel som dinglet frem og tilbake etter at motoren falt til ro. Av den grunn var heller ikke dødsonen like synlig. For å rette opp i feilen forandres dødsonen fra 0.05 til 0.0129 slik at den er lik som i det virkelige systemet. Modellen simuleres på nytt med den nye dødsonen. Resultatene vises i Figur 8.13 og Figur 8.14

Med den nye dødsone kan man se at modellen stemmer veldig bra. Den korte pendelen svinger litt for mye, men de dynamiske komponentene ser veldig riktig ut. Vinkelhastigheten til motoren øker litt for mye etter at den andre firkantpulsen påtrykkes systemet. Utenom dette tyder resultatene på at identifikasjon av både pendel og moter er ganske bra.

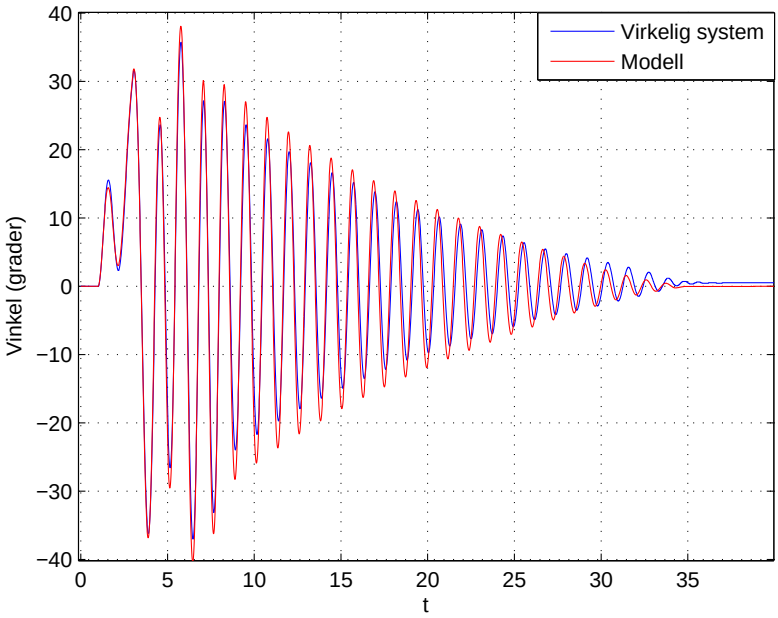


Figur 8.13: Vinkel, Kort pendel. Respons for enkeltpendelsystem med strømregulator og riktig dødsone på motoren

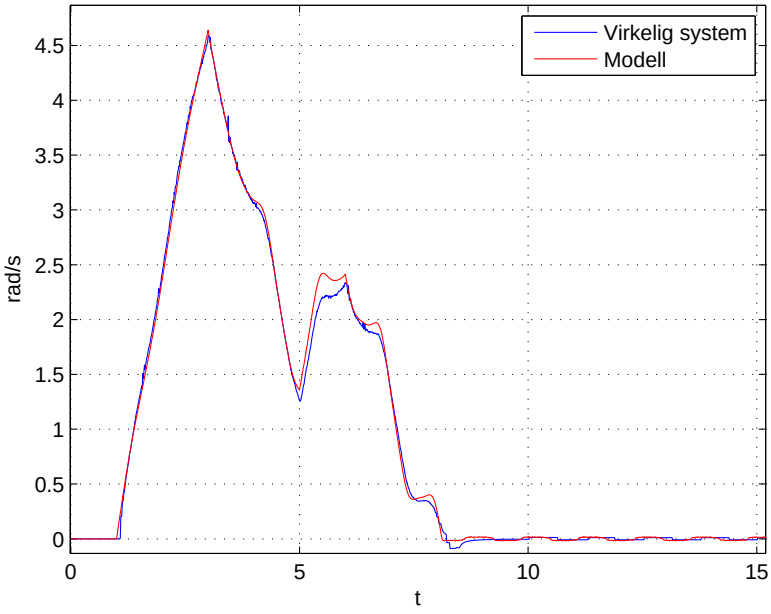


Figur 8.14: Vinkelhastighet, Arm1. Respons for enkeltpendelsystem med strømregulator og riktig dødsone på motoren

8.3.2 Motormodell med innkobling og frakobling av strømsløyfe



Figur 8.15: Vinkel, Kort pendel. Respons for enkeltpendelsystem med ideell strømregulering



Figur 8.16: Vinkelhastighet, Arm1. Respons for enkeltpendelsystem med ideell strømregulering

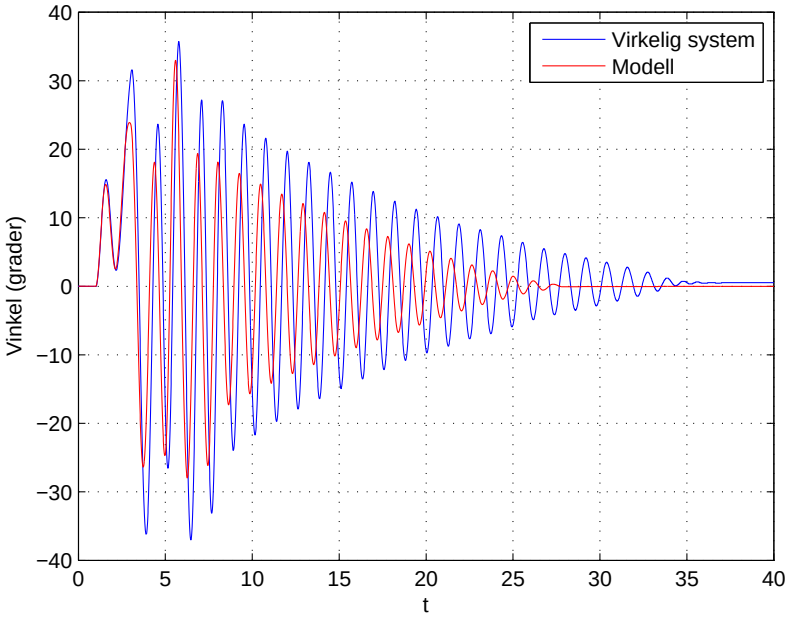
De to motormodellene gir nesten helt identisk respons. Dette tyder på at strømreguleringen er veldig rask i det virkelige systemet og at den motindusert spenning er ganske ubetydelig under dempingen i forhold til friksjonen. Hvilken modell som benyttes har altså ikke så mye å si, men det er muligens bedre å benytte modellen med strømregulator for å unngå ulineæriteten med å koble strømsløyfen inn og ut. Et siste alternativ vil være å droppe både strømsløyfe og regulator slik at man ender opp med en modell som ligner på den som ble funnet i [13]. Den ulineære friksjonsmodellen viser seg derimot viktig for å få riktig motordynamikk, men fortsatt finnes det ukjente komponenter i motordynamikken som ikke har blitt modellert.

8.3.3 Alternative pendelligninger

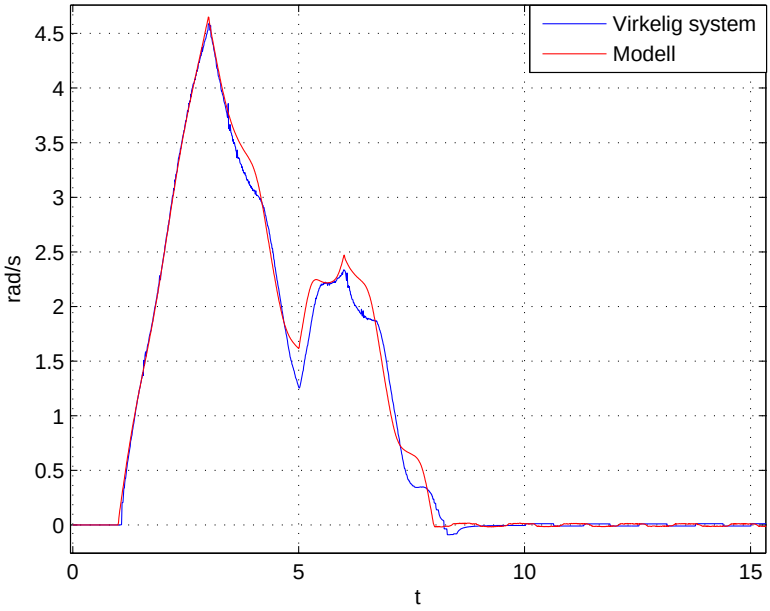
I [10] ble det funnet bevegelsesligninger som var noe anderledes enn de som ble funnet i 4.1. Dette skyldes at det kun ble brukt treghetsmoment rundt z-aksen under utledningen i [10] mens treghetsmomentet ble beskrevet langs flere akser i 4.1. I [13] konkluderes det allikevel med at pendelligningene fra [10] må være riktig fordi regulatoren som ble implementert fungerer tilfredsstillende til balansering av enkeltpendel. Av den grunn var det ønskelig å sammenligne pendelmodellen fra både 4.1 og [10] med det virkelige systemet for å se hvilken som faktisk er mer nøyaktig.

Pendelmodellen i [10] ser identisk ut for isolert pendel der horisontal arm er låst. Av den grunn blir de identifiserte friksjonsparametrene fortsatt gyldig med unntak av den utregnede luftmotstandenskoeffisienten som beskriver luftmotstanden på pendelen når motoren roterer. Denne ble derfor regnet ut på nytt til ligningene fra [10]. Det benyttes samme inngangsignal

Resultatene i Figur 8.17 og Figur 8.18 viser at ligningene gir en litt dårligere respons. pendelen gir for lite utslag etter bevegelse i Arm1. Som følge av dette blir Arm1 påvirket litt anderledes fra pendelen. Dette tyder på at det blir mer riktig å benytte treghetsmoment langs flere akser når ligningene utledes.



Figur 8.17: Vinkel, Kort pendel. Respons for enkeltpendelsystem med Larssen sine ligninger

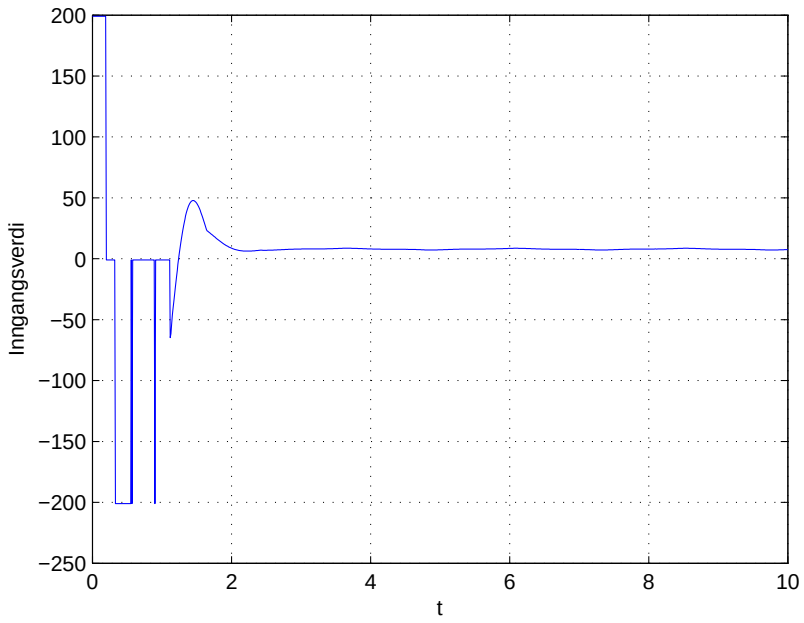


Figur 8.18: Vinkelhastighet, Arm1. Respons for enkeltpendelsystem med Larssen sine ligninger

8.4 Balansering av pendel

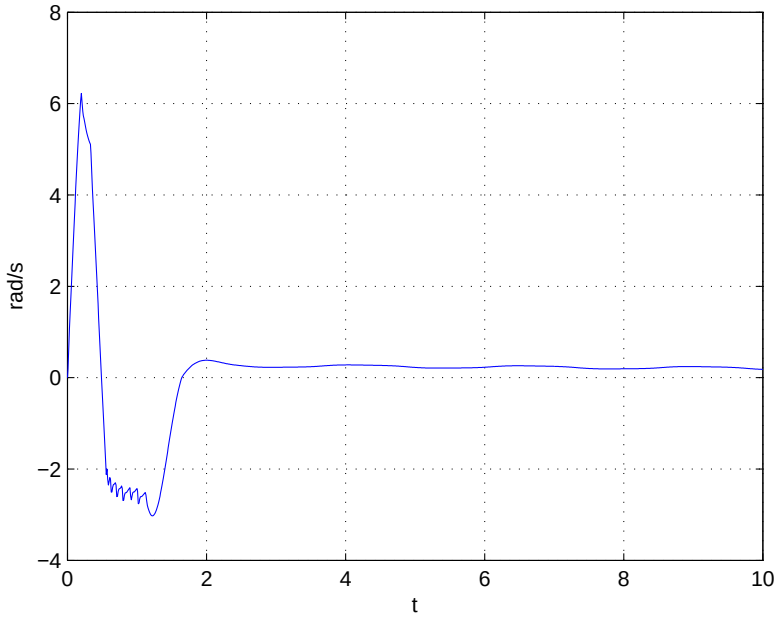
Regulatorene for balansering av kort pendel ble implementert i simulatoren. Her benyttes oppsvingsregulatoren til å svinge opp pendelen.

8.4.1 LQR

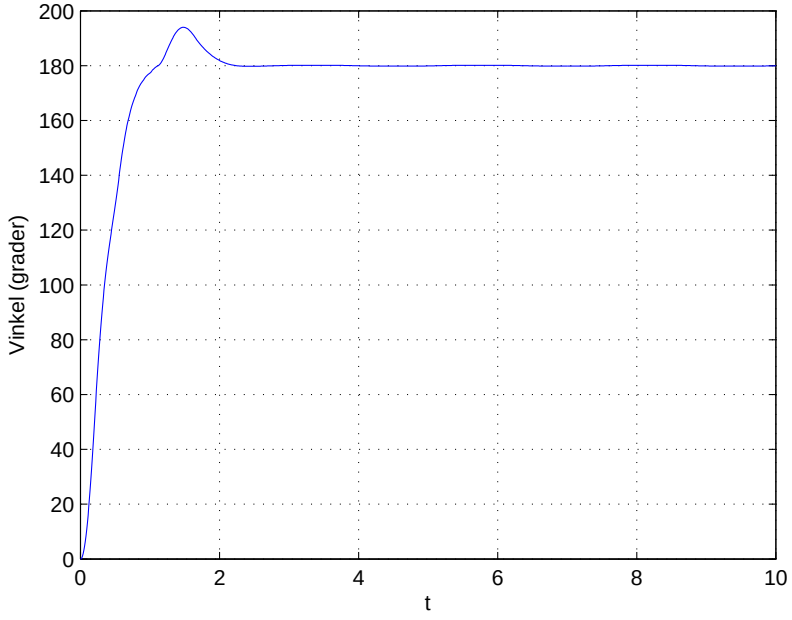


Figur 8.19: Paadrag til motor under balansering med LQR

Som man kan se i Figur 8.21 fungerer regulatoren ganske bra. Det er noe oversving, men vinkelen konvergerer ganske raskt til 180°. En mulighet for å redusere oversving kunne vært å øke kostnaden for θ_3 i Q. Dette ville gitt en større forsterkning i tilbakekoblingen til θ_3, men som man kan se fra Figur 8.19 så holder utgangen av regulatoren seg veldig rolig rundt relativt små verdier. Det ville gått helt fint med større forsterkning av tilstandene. I Figur 8.19 kan man se hvordan motoren bremser og snur retning under oppsvingningen etter at pendelen oppnår 90°. Under balanseringen legger vinkelhastigheten seg på en ganske lav stasjonær verdi.



Figur 8.20: Vinkelhastighet, Arm1. LQR



Figur 8.21: Vinkel, Kort pendel. LQR

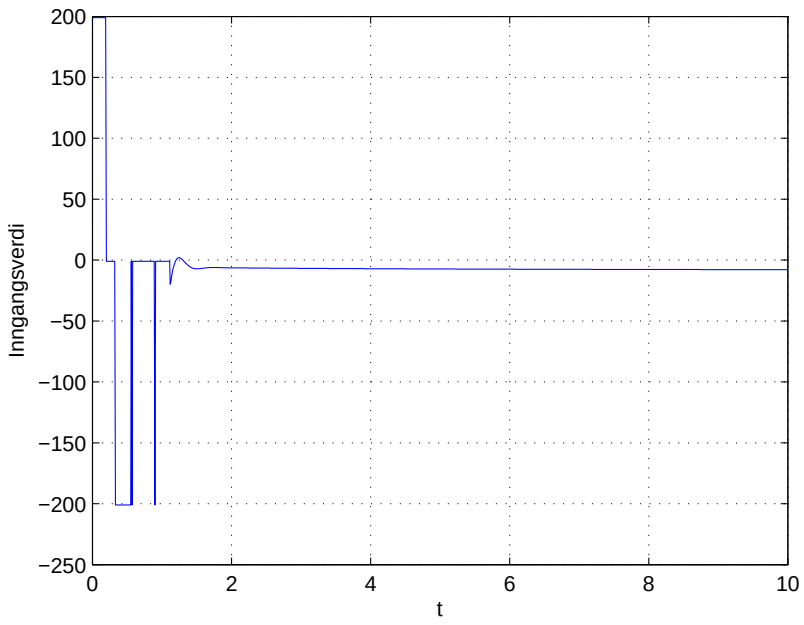
8.4.2 Regulator funnet med genetisk algoritme

Algoritmen brukte ganske kort tid på å finne en stabil løsning. når den stabile løsningen først var funnet, ble den spred gjennom befolkningen på få generasjoner. Det var derfor ikke nødvendig å kjøre algoritmen i mer en 20 generasjoner, noe som tok i underkant av 10-minutter.

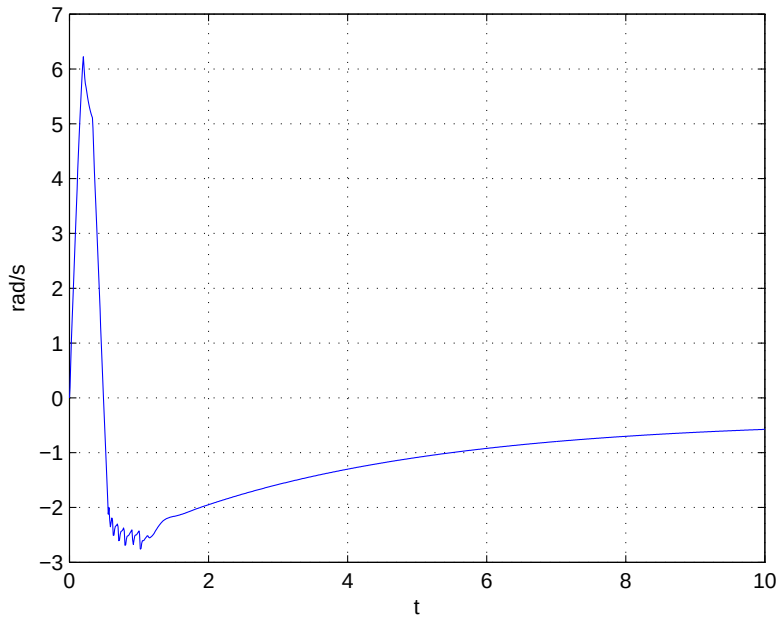
Resultatene viser at regulatoren lett kan finnes ved å benytte den genetiske algoritmen. I motsetning til resultatene for LQ-regulatoren, er det ikke noe oversving i pendelens vinkel, men som man kan se fra Figur 8.23 så bruker motoren lenger tid på å nå en stasjonær verdi. Dette skyldes at forsterkningen for avviket $\theta_3 - pi$ på utgangen av simulatoren var satt ganske høy da algoritmen ble kjørt. Av den grunn ble det lagt stor vekt på å minimere $\theta_3 - pi$, noe som man kan se i Figur 8.24 hvor oppnår stasjonær verdi ganske raskt. Med denne metoden kan også regulatormodellen ganske lett utvides. F.eks. kunne man lagt til integrasjon av avviket i regulatormodellen slik at

$$u = -k_1 x_1 - K_2 \frac{1}{s}(x - pi) - K_3 x_3 - K_4 x_4 \tag{8.2}$$

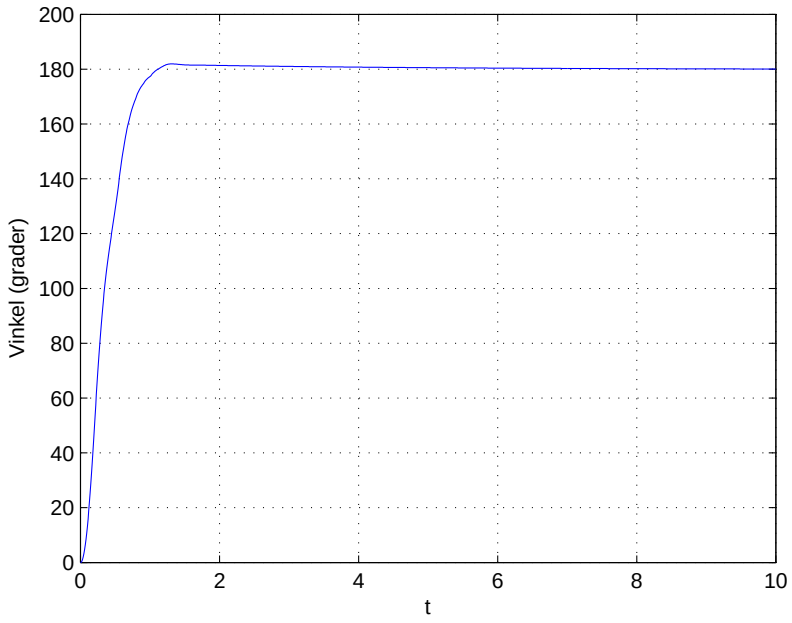
Om man skulle implementert dette med LQR måtte man satt opp en ny linearisert modell med en ekstra tilstand. Om man bruker genetisk algoritme kan man derimot lett implementere denne effektene uten mye ekstra arbeid.



Figur 8.22: Paadrag. Regulator funnet med genetisk algoritme



Figur 8.23: Vinkelhastighet, Arm1. Regulator funnet med genetisk algoritme



Figur 8.24: Vinkel, Kort pendel. Regulator funnet med genetisk algoritme

Kapittel 9

Konklusjon

I prosjektet ble systemet modellert på nytt med færre forenklinger og approksimasjoner enn hva som ble benyttet under modellering i tidligere prosjekt på systemet. Bevegelsesligningene for pendeldynamikken ble utledet ved å benytte Newton-Eulers metode. De resulterende ligningene og ligningene utledet i tidligere prosjekt ble satt opp i simulator. Responsen fra simulatoren ble så sammenlignet med responsen til det virkelige systemet. Resultatene viser at bevegelsesligningene som ble utledet i dette prosjektet gir en mer nøyaktig respons enn de tidligere ligningene.
Det ble satt opp en ulineær DC-motormodell som inkludere coulombfriksjon med stribeck-effekt, dødsone og hysterese i friksjonen. For å estimere parametrene til denne motormodellen ble det implementert en genetisk algoritme der resultatet fra denne algoritmen benyttes som initiering av en Nelder-Mead algoritme. Estimeringsmetoden utførte en god del simuleringer og brukte dermed relativt lang tid. Resultatet ble derimot veldig bra og ga en motormodell som reflekterte det virkelige systemet mye bedre enn motormodellene fra tidligere prosjekt. Resultatene demonstrerer også at metoden kan benyttes til å estimere parametre i ulineære systemer der gradienten til modelligningene ikke kan regnes ut.
Det ble implementert to forskjellige servostrukturer til motoren. I den ene servomodellen benyttes PI-regulering av strømmen i motoren mens den andre modellen benytter ideell strømregulering. Resultatene viser at begge modellene gir ganske lik respons, men at modellen med PI-regulator er noe mer nøyaktig i forhold til det virkelige systemet. Resultatene tyder på at strømreguleringen er ganske rask slik at systemet med ideell regulator fungerer bra som approksimasjon.
I simulatoren for enkeltpendelsystemet ble det implementer regulator for oppsving og balansering av pendel. Det ble implementert to forskjellige strategier for å finne regulatorparametrene. Først og fremt ble LQR-metoden brukt. Med LQR ble det oppnådd tilfredsstillende resultat, men metoden krever linearisering av bevegelsesligningene. For enkeltpendelsystemet går det fint an å linearisere ligningene med penn og papir. For dobbel og trippelpendelsystemet vil ligningene være en god del større og lineariseringen vil da begynne å bli problematisk. Man burde da regne ut de lineariserte ligningene fra en datamaskin. Som et alternativ til LQR forsøkes det å benytte den genetiske algoritmen til å finne regulatorparametre. Man kan se på det som at den genetiske algoritmen "lærer" seg hvordan systemet kan best reguleres. Det ble da brukt samme reguleringslov som med LQR. Denne metoden viste seg å være veldig lettvint å benytte siden algoritmen allerede

var implementert under estimering av modellparametre. I motsetning til LQR som finner en optimal løsning til den lineære approksimasjonen, så finner den genetiske algoritmen en løsning til den ulinære modellen. denne metoden antas derfor å fungere bedre enn LQR dersom lineariseringen av systemet skulle gi en dårlig approksimert modell. Dersom den lineariserte modellen rundt arbeidspunktet skulle være en veldig god approksimasjon antas det at LQR vil kunne gi bedre resultat ettersom at metoden finner optimal løsning. Begge regulatorene ga gode resultat til balansering av pendel. Dette tyder på at metoden med genetisk algoritme er et godt alternativ til LQR slik at metoden kan forsøkes til balansering av flere pendler i fremtidige prosjekt dersom LQR skulle feile. Metoden har fortsatt noen svakheter. For det første må regulatorparametrene finnes i en simulator ettersom at det ville tatt for lang tid å benytte metoden i sanntid på det virkelige systemet. For at regulatorparametrene skal fungere på det virkelige systemet kreves det at simulatoren er en god analog. Det kan også hende at systemet med dobbel og trippelpendel vil ha et ganske smalt område av regulatorparametre som gir et stabilt system. Da kan det hende at det blir for vanskelig for den genetiske algoritmen å finne en stabil løsning. I dette tilfellet kan det forsøkes å justere algoritmeparametrene eller redusere optimaliseringsområdet.
På grunn av feil i hardware ble ikke regulatorene implementert på det virkelige systemet. Det ble heller ikke tid til utvikling av regulator for balansering av flere pendler. Fortsatt har det blitt gjort mye nyttig arbeid som vil kunne være til hjelp i fremtidige prosjekt. Det håpes at de nye metodene som er introdusert i prosjektet kan hjelpe til i fremtiden med å finne en endelig løsning på problemet med å balansere alle tre pendler.

Kapittel 10

Videre arbeid

Det er fortsatt ting som må gjøres for at trippelpendelsystemet skal kunne balanseres.

- Datainnsamling
  - Elektronikken for datainnsamling må repareres slik at vinkel for alle pendlene kan måles samtidig.
  - Det må lages et system for overføring av enkoderdata fra Arm4. f.eks. ved bruk av slepering, radiosignal eller infrarødt
- Mekanisk
  - Det må lages et festeoppheng til Arm4 slik at denne pendelen kan benyttes
- Mikrokontroller
  - Mikrokontrollerenprogrammet som ble laget i [5] er veldig kaotisk og krever en opprydning dersom det skal kunne implementeres regulator direkte fra mikrokontrolleren
  - Dataen for systemtilstandene er ikke helt riktig. Det ble oppdaget vinkelområder fra $0 - 364.3^\circ$ og stasjonære vinkelhastigheter når systemet står i ro. Dette burde fikses
- Regulator
  - Det må finnes en regulator som kan repeterbart balansere flere pendler
  - Alternative reguleringsmetoder kan undersøkes. i [8] frempekes ulineær MPC som en mulighet

Tillegg A

Kildekode for matematisk modellering

I prosjektet er Matlab benyttet for symbolsk utregning av bevegelsesligningene til pendlene. Matlab sorterer disse ligningene på kaotisk vis under mellomregningene. For å kunne forenkle bevegelsesligningene må de ganges ut og sorteres på mer systematisk vis (sorteres etter variabler og ikke etter vilkårlige parametre). Ettersom at disse ligningene er ganske lange har det blitt skrevet et program i Python som utfører oppgaven med å sortere ligningene.

	Matematisk symbol	Programsymbol
Vinkel	θ	`th`
Vinkelhastighet	$\dot{\theta}$	`d_th`
Vinkelakselerasjon	$\ddot{\theta}$	`dd_th`

Tabell A.1: Modellvariabler

A.1 Matlabscript for symbolsk utregning av bevegelsesligninger

```
%Utregning av furutapendel bevegelsesligninger

%Definering av symbolske uttrykk
th1=sym('th1');
d_th1=sym('d_th1');
dd_th1=sym('dd_th1');
th2=sym('th2');
d_th2=sym('d_th2');
dd_th2=sym('dd_th2');
th3=sym('th3');
d_th3=sym('d_th3');
dd_th3=sym('dd_th3');
th4=sym('th4');
```

```
d_th4=sym('d_th4');
dd_th4=sym('dd_th4');
g=sym('g');
l1=sym('l1');
l2=sym('l2');
l3=sym('l3');
l4=sym('l4');
L3=sym('L3');
tau=sym('tau');
tau_f=sym('tau_f');
m1=sym('m1');
m2=sym('m2');
m3=sym('m3');
m4=sym('m4');
I1=sym('I1');
I2=sym('I2');
I3=sym('I3');
I4=sym('I4');
J1=sym('J1');
J2=sym('J2');
J3=sym('J3');
J4=sym('J4');

%Rotasjonsmatriser
R1=[cos(th1) sin(th1) 0;-sin(th1) cos(th1) 0; 0 0 1];
R2=[0 sin(th2) -cos(th2);0 cos(th2) sin(th2); 1 0 0];
R3=[0 -sin(th3) -cos(th3); 0 -cos(th3) sin(th3); -1 0 0];
R4=[cos(th4) sin(th4) 0;-sin(th4) cos(th4) 0; 0 0 1];

%arm1
%Inertiamatrix
I_1=[0 0 0;0 I1 0;0 0 I1];
%Lengdevektor
l_1=[l1;0;0];
%Lineaer akselerasjon
a1=[0; 0; g];
%Vinkelhastighet
omega1=[0; 0; d_th1];
%Vinkelakselerasjon
d_omega1=[0; 0; dd_th1];
%Lineaer hastighet i enden av arm1
v1_e=cross(omega1,l_1);

%arm2
%inertiamatrix
I_2=[0 0 0;0 I2 0;0 0 I2];
%lengdevektor
l_2=[l2; 0; 0];
%Vinkelhastighet
omega2=R2*omega1+[0; 0; d_th2];
%Vinkelakselerasjon
d_omega2=[-dd_th1*cos(th2)+d_th1*d_th2*sin(th2);dd_th1*sin(th2)+d_th1*d_th2*cos(th2);dd_
%Lineaer hastighet
v2=R2*v1_e;
%Lineaer akselerasjon
```

```
69 a2=R2*(cross(d_omega1,l_1)+cross(omega1,v1_e)+a1);
70 %Lineaer akselerasjon ved massesenter
71 a2_c=cross(d_omega2,l_2)+cross(omega2,cross(omega2,l_2))+a2;
72
73 %arm3
74 %inertiamatrix
75 I_3=[0 0 0;0 I3 0;0 0 I3];
76 %lengdevektor
77 l_3=[l3; 0; 0];
78 L_3=[L3; 0; 0];
79 %Vinkelhastighet
80 omega3=R3*omega1+[0; 0; d_th3];
81 %Vinkelakselerasjon
82 d_omega3=[-dd_th1*cos(th3)+d_th1*d_th3*sin(th3);dd_th1*sin(th3)+d_th1*d_th3*cos(th3);dd_
83 %Lineaer hastighet
84 v3=R3*v1_e;
85 %Lineaer akselerasjon
86 a3=R3*(cross(d_omega1,l_1)+cross(omega1,v1_e)+a1);
87 %Lineaer akselerasjon ved massesenter
88 a3_c=cross(d_omega3,l_3)+cross(omega3,cross(omega3,l_3))+a3;
89
90 %arm4
91 %inertiamatrix
92 I_4=[0 0 0;0 I4 0;0 0 I4];
93 %lengdevektor
94 l_4=[l4; 0; 0];
95 %Vinkelhastighet
96 omega4=R4*omega3+[0; 0; d_th4];
97 %Vinkelakselerasjon
98 d_omega4=[-dd_th1*cos(th3+th4)+d_th1*d_th3*sin(th3+th4)+d_th1*d_th4*sin(th3+th4);dd_th1*
99 %Lineaer akselerasjon
100 a4=R4*(cross(d_omega3,L_3)+cross(omega3,cross(omega3,L_3))+a3);
101 %Lineaer akselerasjon ved massesenter
102 a4_c=cross(d_omega4,l_4)+cross(omega4,cross(omega4,l_4))+a4;
103
104 %Newton-euler
105
106 %Krefter
107 F1=m1*a1;
108 F2=m2*a2_c;
109 F3=m3*a3_c;
110 F4=m4*a4_c;
111
112 %Momentvektor
113 N1=I_1*d_omega1;
114 N2=I_2*d_omega2+cross(omega2,I_2*omega2);
115 N3=I_3*d_omega3+cross(omega3,I_3*omega3);
116 N4=I_4*d_omega4+cross(omega4,I_4*omega4);
117
118 %Utregning av enkeltpendel
119 %n1=N1+R2.'*n2+cross(l_1,R2.'*F2)
120 %n2=N2+cross(l_2,F2)
121
122 %Utregning av dobbelpendel
123 f3=F3+R4.'*F4;
```

```
124 n4=N4+cross(l_4,F4);
125 n3=N3+cross(l_3,F3)+R4.'*n4+cross(L_3,R4.'*F4);
126 n1=N1+R3.'*n3+cross(l_1,R3.'*f3);
```

Koden regner ut $\boldsymbol{n}$-vektorene. z-komponenten av disse kan hentes ut ved kommandoene n1(3), n2(3), n3(3) og n4(3)

A.2 Python-kode for sortering av bevegelsesligning

Programmet tar .txt som input. Ligningen som printes i Matlab kan da kopieres sammenhengende over til en tekst-fil. tekst-filen benyttes som stdin til pythonprogrammet.

Eksempel på inputfil Input.txt

```
I2*dd_th2 + l2*m2*(- l2*cos(th2)*sin(th2)*d_th1^2 + dd_th2*l2 +
g*sin(th2) + dd_th1*l1*cos(th2)) - I2*d_th1^2*cos(th2)*sin(th2)
```

Programmet ganger ut alle parentesene i ligningen, sorterer leddene til nye parenteser og sorterer til slutt alle parametrene i hvert ledd. Resultatene blir skrevet til Output.txt hvor

Eksempel på outputfil Output.txt

```
dd_th1
(
    +m2*l1*l2*cos(th2)
)
+
dd_th2
(
    +I2
    +m2*l2^2
)
+
d_th1^2
(
    -m2*l2^2*sin(th2)*cos(th2)
    -I2*sin(th2)*cos(th2)
)
+
+m2*l2*g*sin(th2)
```

Etter at ligningen er sortert kan den lettere forenkles. Ligningen i eksempelet kan ikke forenkles så veldig mye, men for de større pendelligningene kan størrelsen på ligningene reduseres til rundt en ti-del og mindre av opprinnelig størrelse.

```
#LigningSortering.py

#Utforer sortering av bevelgelseslikninger som har blitt utledet symbolsk i matlab.
#Programmet leser inn likning som streng, ganger ut parenteser og sorterer ledd.
#Resultatet skrives ut til 'Output.txt'

from sys import stdin

def SamleUtrykk(Likning,Utrykk):
  #Finner alle substreng av Utrykk i Likning
  #og samler disse i begynnelsen av likningen
  nLikning=Likning      #Initierer modifiseringsligning
  n=0             #Antallet gjentakelser av Utrykk i Likning
  pos=Likning.find(Utrykk)  #Finner den forste forekomst av Utrykk
  size=len(Utrykk)      #Lengden av Utrykk
  if len(nLikning)>(size+1):

    while (pos!=-1): #Saa lenge det gjenstaar forekomst av Utrykk i nLikning

      if (len(nLikning)>pos+size+1):
        if(nLikning[(pos+size)]=='^'):    #Dersom Utrykket er opphoyet
          n+=int(nLikning[(pos+size+1)])    #okes antallet oppdagede utrykk
          nLikning=nLikning[:pos+size]+nLikning[(pos+size+2):] #opphoyning slettes
        else:                       #Ellers
          n+=1                        #Inkrementeres n
      else:
        n+=1

      #Det oppdagede utrykket slettes fra likningen sammen med
      #multiplikasjonstegnet i forkant eller etterkant
      if pos == 1:      #Dersom Utrykk forekommer i starten av likning
        nLikning=nLikning[0]+nLikning[(pos+size+1):] #Slettes tegn i etterkant
      else:                               #Ellers
        nLikning=nLikning[:(pos-1)]+nLikning[(pos+size):] #Slettes tegn i forkant

      pos=nLikning.find(Utrykk) #Finner neste forekomst

  #Oppdaget antall Utrykk settes tilbake i begynnelsen av ligningen
  if n == 1:
    nLikning = nLikning[0] + Utrykk + '*' + nLikning[1:]
  if n > 1:
    nLikning = nLikning[0] + Utrykk + '^' + str(n) + '*' + nLikning[1:]

  return nLikning

def SorterLedd(Likning):

  #Sorterer utrykkene i Likningen
  Likning=SamleUtrykk(Likning,'cos(th3+th4)')
  Likning=SamleUtrykk(Likning,'sin(th3+th4)')
  Likning=SamleUtrykk(Likning,'cos(th4)')
  Likning=SamleUtrykk(Likning,'cos(th3)')
  Likning=SamleUtrykk(Likning,'cos(th2)')
  Likning=SamleUtrykk(Likning,'sin(th4)')
  Likning=SamleUtrykk(Likning,'sin(th3)')
```

```
56    Likning=SamleUtrykk(Likning,'sin(th2)')
57    Likning=SamleUtrykk(Likning,'g')
58    Likning=SamleUtrykk(Likning,'L4')
59    Likning=SamleUtrykk(Likning,'L3')
60    Likning=SamleUtrykk(Likning,'L2')
61    Likning=SamleUtrykk(Likning,'l4')
62    Likning=SamleUtrykk(Likning,'l3')
63    Likning=SamleUtrykk(Likning,'l2')
64    Likning=SamleUtrykk(Likning,'l1')
65    Likning=SamleUtrykk(Likning,'m4')
66    Likning=SamleUtrykk(Likning,'m3')
67    Likning=SamleUtrykk(Likning,'m2')
68    Likning=SamleUtrykk(Likning,'m1')
69    Likning=SamleUtrykk(Likning,'I4')
70    Likning=SamleUtrykk(Likning,'I3')
71    Likning=SamleUtrykk(Likning,'I2')
72    Likning=SamleUtrykk(Likning,'I1')
73
74    return Likning
75
76  #Main
77  ligning=stdin.readline().split()
78  #Likningen leses inn og deles opp ved hvert mellomrom.
79  #Dette betyr at man vil faa et enkelt fortegn ved annenhvert element
80  #siden Matlab skriver ut ligningen med mellomrom mellom hvert fortegn
81
82  a=""            #Streng til lagring av modifiserte likningsledd
83  b=[""]            #Liste til lagring av uttrykk under utganging av parenteser
84  oppdelt_ligning=[]     #Utganget og oppdelt modifisert likning
85  fortegn="+"         #Fortegn til ledd i likning
86  op=""           #Operasjonsidentifisering
87
88  #Lister med ledd for Nye Inndelingsparenteser
89  dd_th1=[]         #Dobbelderivert av theta1
90  dd_th2=[]
91  dd_th3=[]
92  dd_th4=[]
93  d_th1_2=[]          #Kvadrat av theta1 derivert
94  d_th2_2=[]
95  d_th3_2=[]
96  d_th4_2=[]
97  d_th1_d_th2=[]        #Produkt av theta1 derivert og theta2 derivert
98  d_th1_d_th3=[]
99  d_th1=[]          #Derivert av theta1
100 d_th2=[]
101 d_th3=[]
102 d_th4=[]
103 rest=[]           #Restrende uttrykk i likningen
104
105
106 for ledd in ligning:
107
108   if len(ledd)>1:   #Dersom elementet inneholder et uttrykk og ikke bare et fortegn
109     a+=b[-1]        #Initieres leddet ved aa fylle inn et eventuelt utgangingsuttrykk
110
```

```
111     #Dersom et ledd inneholder en trigonometrisk funksjon med
112     #en sum. f.eks. sin(theta1 + theta2). Vil dette leddet splittes
113     #naar det leses inn. ['sin(theta1', '+', 'theta2)'].
114     #I dette tilfellet er det onskelig at elementene samles til sin(theta1+theta2)
115     if (op == 'si') or (op == 'co'): #Hvis trigonometrisk funksjon ble oppdegaet
116       a+=fortegn      #Skrives fortegn inn paa slutten av a
117       fortegn = a[0]    #Fortegn i begynnelsen av a lagres
118       a=a[1:]       #Fortegn i begynnelsen av a slettes
119       b.pop()       #Utrykket fra forrige iterasjon slettes
120 
121   #Dersom et ledd er paa insiden av en parentes som skal ganges ut
122   #maa fortegnet ordnes dersom utgangingsutrykket er negativt
123   if (len(a)>0) and (op != 'si') and (op != 'co'):
124     if (a[0]=='-') and (fortegn =='+'):
125       fortegn = '-'
126     elif (a[0]=='-') and (fortegn =='-'):
127       fortegn = '+'
128     a=a[1:]   #Fortegn fjernes fra utrykket siden det skal legges til mot slutten
129 
130   for c in ledd:  #Undersoker hvert tegn i strengen som utrykker ledddet
131 
132     #Den neste delen har som maal aa bestemme hvorvidt det kommer en parentes
133     #som maa ganges ut. Starten paa en parentes kan oppdages ved
134     #tegnkombinasjonen '*('. Slutten paa parentesen kan kun oppdages ved tegnet ')'.
135     #Siden sinus og cosinus ogsaa besrives med parenteser er det ogsaa nodvendig
136     #aa oppdage disse funksjonene. Ellers vil man risikere aa tolke slutten paa
137     #en trigonometrisk funksjon som enden paa en parentes som skal ganges ut
138 
139     if ((c=='*') or (c=='s') or (c=='c')) and (op != 'si') and (op != 'co'):
140       op=c
141     elif len(op)==1:
142       op+=c
143     if (len(op)==2) and (op!='*(') and (op!='si') and (op!='co'):
144       op=""
145 
146     if (c==')') and (op != 'si') and (op != 'co'):
147       b.pop()
148     if (c==')') and ((op == 'si') or (op == 'co')):
149       op=""
150       a+=c
151 
152     if op=='*(':      #Dersom begynnelsen paa en parentes er oppdaget
153       b.append(fortegn+a)   #Lagres utgangingselementet i enden av b
154       op=""             #og operasjonsidentiteten slettes
155     else:
156       if (len(a) > 0) and (c=='-'):
157         a=c
158       else:
159         if c != ')':
160           a+=c    #tegn skrives til a
161 
162   if (a=='+') or (a=='-'):  #Dersom a kun inneholder et enkelt fortegn
163     fortegn=a           #lagres fortegnet
164   else:
165     if (op != 'si') and (op != 'co'): #Hvis ikke trigonometrisk funksjon
```

```
166         oppdelt_ligning.append(fortegn+a) #lagres a til liste med modifiserte ledd
167       else:                           #Ellers
168         b.append(fortegn+a)      #overfores a til neste iterasjon
169     a=""  #a resettes for neste iterasjon
170
171 #I neste del sorteres hvert ledd i modifisert likning til de nye inndelingene
172 for ledd in oppdelt_ligning:
173
174     pos = ledd.find('dd_th1')
175     if pos!=-1:
176       dd_th1.append(SorterLedd(ledd[:(pos-1)]+ledd[(pos+6):]))
177     if pos==-1:
178       pos = ledd.find('dd_th2')
179       if pos!=-1:
180         dd_th2.append(SorterLedd(ledd[:(pos-1)]+ledd[(pos+6):]))
181     if pos==-1:
182       pos = ledd.find('dd_th3')
183       if pos!=-1:
184         dd_th3.append(SorterLedd(ledd[:(pos-1)]+ledd[(pos+6):]))
185     if pos==-1:
186       pos = ledd.find('dd_th4')
187       if pos!=-1:
188         dd_th4.append(SorterLedd(ledd[:(pos-1)]+ledd[(pos+6):]))
189
190     if pos==-1:
191       pos = ledd.find('d_th1^2')
192       if pos!=-1:
193         d_th1_2.append(SorterLedd(ledd[:(pos-1)]+ledd[(pos+7):]))
194     if pos==-1:
195       pos = ledd.find('d_th2^2')
196       if pos!=-1:
197         d_th2_2.append(SorterLedd(ledd[:(pos-1)]+ledd[(pos+7):]))
198     if pos==-1:
199       pos = ledd.find('d_th3^2')
200       if pos!=-1:
201         d_th3_2.append(SorterLedd(ledd[:(pos-1)]+ledd[(pos+7):]))
202     if pos==-1:
203       pos = ledd.find('d_th4^2')
204       if pos!=-1:
205         d_th4_2.append(SorterLedd(ledd[:(pos-1)]+ledd[(pos+7):]))
206
207     if pos==-1:
208       pos = ledd.find('d_th1*d_th2')
209       if pos!=-1:
210         d_th1_d_th2.append(SorterLedd(ledd[:(pos-1)]+ledd[(pos+11):]))
211     if pos==-1:
212       pos = ledd.find('d_th1*d_th3')
213       if pos!=-1:
214         d_th1_d_th3.append(SorterLedd(ledd[:(pos-1)]+ledd[(pos+11):]))
215
216     if pos==-1:
217       pos = ledd.find('d_th1')
218       if pos!=-1:
219         d_th1.append(SorterLedd(ledd[:(pos-1)]+ledd[(pos+5):]))
220     if pos==-1:
```

```
221       pos = ledd.find('d_th2')
222       if pos!=-1:
223         d_th2.append(SorterLedd(ledd[:(pos-1)]+ledd[(pos+5):]))
224     if pos==-1:
225       pos = ledd.find('d_th3')
226       if pos!=-1:
227         d_th3.append(SorterLedd(ledd[:(pos-1)]+ledd[(pos+5):]))
228     if pos==-1:
229       pos = ledd.find('d_th4')
230       if pos!=-1:
231         d_th4.append(SorterLedd(ledd[:(pos-1)]+ledd[(pos+5):]))
232
233     if pos==-1:
234       rest.append(SorterLedd(ledd))
235
236 #Likning skrives til Output.txt
237 text_file = open("Output.txt", "w")
238 if dd_th1:
239     text_file.write('dd_th1\n(')
240     for i in dd_th1:
241       text_file.write('\n\t' + i)
242     text_file.write('\n)\n+\n')
243
244 if dd_th2:
245     text_file.write('dd_th2\n(')
246     for i in dd_th2:
247       text_file.write('\n\t' + i)
248     text_file.write('\n)\n+\n')
249
250 if dd_th3:
251     text_file.write('dd_th3\n(')
252     for i in dd_th3:
253       text_file.write('\n\t' + i)
254     text_file.write('\n)\n+\n')
255
256 if dd_th4:
257     text_file.write('dd_th4\n(')
258     for i in dd_th4:
259       text_file.write('\n\t' + i)
260     text_file.write('\n)\n+\n')
261
262 if d_th1_2:
263     text_file.write('d_th1^2\n(')
264     for i in d_th1_2:
265       text_file.write('\n\t' + i)
266     text_file.write('\n)\n+\n')
267
268 if d_th2_2:
269     text_file.write('d_th2^2\n(')
270     for i in d_th2_2:
271       text_file.write('\n\t' + i)
272     text_file.write('\n)\n+\n')
273
274 if d_th3_2:
275     text_file.write('d_th3^2\n(')
```

```
276     for i in d_th3_2:
277       text_file.write('\n\t' + i)
278     text_file.write('\n)\n+\n')
279 
280 if d_th4_2:
281     text_file.write('d_th4^2\n(')
282     for i in d_th4_2:
283       text_file.write('\n\t' + i)
284     text_file.write('\n)\n+\n')
285 
286 if d_th1_d_th2:
287     text_file.write('d_th1*d_th2\n(')
288     for i in d_th1_d_th2:
289       text_file.write('\n\t' + i)
290     text_file.write('\n)\n+\n')
291 
292 if d_th1_d_th3:
293     text_file.write('d_th1*d_th3\n(')
294     for i in d_th1_d_th3:
295       text_file.write('\n\t' + i)
296     text_file.write('\n)\n+\n')
297 
298 if d_th1:
299     text_file.write('d_th1\n(')
300     for i in d_th1:
301       text_file.write('\n\t' + i)
302     text_file.write('\n)\n+\n')
303 
304 if d_th2:
305     text_file.write('d_th2\n(')
306     for i in d_th2:
307       text_file.write('\n\t' + i)
308     text_file.write('\n)\n+\n')
309 
310 if d_th3:
311     text_file.write('d_th3\n(')
312     for i in d_th3:
313       text_file.write('\n\t' + i)
314     text_file.write('\n)\n+\n')
315 
316 if d_th4:
317     text_file.write('d_th4\n(')
318     for i in d_th4:
319       text_file.write('\n\t' + i)
320     text_file.write('\n)\n+\n')
321 
322 for i in rest:
323     text_file.write(i + '\n')
324 text_file.close()
325 
326 print 'Ferdig'
```

Tillegg B

Pendelligninger

Treghetsmomentet rundt svingningspunktet til pendlene er

$$J_1 = I_1 \tag{B.1}$$
$$J_2 = I_2 + m_2 l_2^2 \tag{B.2}$$
$$J_3 = I_3 + m_3 l_3^2 \tag{B.3}$$
$$J_4 = I_4 + m_4 l_4^2 \tag{B.4}$$

B.1 Enkeltpendel

Arm1

$$\begin{aligned} &\ddot{\theta}_1(J_1 + m_2 l_1^2 + J_2 \sin^2(\theta_2)) + \ddot{\theta}_2 m_2 l_1 l_2 \cos(\theta_2) \\ &- \dot{\theta}_2^2 m_2 l_1 l_2 \sin(\theta_2) + \dot{\theta}_1 \dot{\theta}_2 J_2 \sin(2\theta_2) + \tau_1 = \tau_m \end{aligned} \tag{B.5}$$

Arm2

$$\ddot{\theta}_1 m_2 l_1 l_2 \cos(\theta_2) + \ddot{\theta}_2 J_2 - \dot{\theta}_1^2 \frac{1}{2} J_2 \sin(2\theta_2) + l_2 m_2 g \sin(\theta_2) + \tau_2 = 0 \tag{B.6}$$

B.2 Pendel hver side

Arm1

$$\begin{aligned} &\ddot{\theta}_1(J_1 + m_2 l_1^2 + m_3 l_1^2 + J_2 \sin^2(\theta_2) + J_3 \sin^2(\theta_3)) + \ddot{\theta}_2 m_2 l_1 l_2 \cos(\theta_2) \\ &-\ddot{\theta}_3 m_3 l_1 l_3 \cos(\theta_3) + \dot{\theta}_3^2 m_3 l_1 l_3 \sin(\theta_3) - \dot{\theta}_2^2 m_2 l_1 l_2 \sin(\theta_2) \\ &+\dot{\theta}_1 \dot{\theta}_2 J_2 \sin(2\theta_2) + \dot{\theta}_1 \dot{\theta}_3 J_3 \sin(2\theta_3) + \tau_1 = \tau_m \end{aligned} \tag{B.7}$$

Arm2

$$\ddot{\theta}_1 m_2 l_1 l_2 \cos(\theta_2) + \ddot{\theta}_2 J_2 - \dot{\theta}_1^2 \frac{1}{2} J_2 \sin(2\theta_2) + l_2 m_2 g \sin(\theta_2) + \tau_2 = 0 \tag{B.8}$$

Arm3

$$-\ddot{\theta}_1 m_3 l_1 l_3 \cos(\theta_3) + \ddot{\theta}_3 J_3 - \dot{\theta}_1^2 \frac{1}{2} J_3 \sin(2\theta_3) + l_3 m_3 g \sin(\theta_3) + \tau_3 = 0 \tag{B.9}$$

B.3 Dobbelpendel

Arm1

$$
\begin{aligned}
&\ddot{\theta}_1(I_1 + m_3 l_1^2 + m_4 l_1^2 + (J_3 + m_4 L_3^2)\sin^2(\theta_3) + J_4 \sin^2(\theta_3 + \theta_4)) \\
&+\ddot{\theta}_1 2 m_4 l_4 L_3 \sin(\theta_3)\sin(\theta_3 + \theta_4) - \ddot{\theta}_3((m_3 l_1 l_3 + m_4 l_1 L_3)\cos(\theta_3) + m_4 l_1 l_4 \cos(\theta_3 + \theta_4)) \\
&-\ddot{\theta}_4 m_4 l_1 l_4 \cos(\theta_3 + \theta_4) - \dot{\theta}_1^2 m_4 l_1 l_4(\sin(\theta_4)\cos(\theta_3) + \sin(\theta_3)\cos(\theta_4)\sin^2(\theta_3 + \theta_4)) \\
&+\dot{\theta}_3^2(m_3 l_1 l_3 + m_4 l_1 L_3)\sin(\theta_3) + \dot{\theta}_4^2 m_4 l_1 l_4 \sin(\theta_3 + \theta_4)) \\
&+\dot{\theta}_1\dot{\theta}_3((J3 + m_4 L_3^2)\sin(2\theta_3) + \frac{1}{2} J_4 \sin(2(\theta_3 + \theta_4)) + \frac{1}{2} m_4 l_4 L_3(\sin(\theta_4) + 3\sin(2\theta_3 + \theta_4))) \\
&+\dot{\theta}_1\dot{\theta}_4(J_4 \sin(2(\theta_3 + \theta_4)) + 2 m_4 l_4 L_3 \sin(\theta_3)\cos(\theta_3 + \theta_4)) \\
&+\dot{\theta}_1 m_4 l_1 l_4 \sin(\theta_4)\cos(\theta_3)(\cos(\theta_3 + \theta_4) + \sin(\theta_3 + \theta_4)) + \dot{\theta}_1 m_4 l_1 l_4 \sin(\theta_3)^2 \cos(\theta_4)\sin(\theta_3 + \theta_4) \\
&+\dot{\theta}_3((J_4 + m_4 l_1 l_4)\sin(\theta_3 + \theta_4) + m_4 l_4 L_3 \sin(\theta_3)) + \tau_1 = \tau_m
\end{aligned}
\tag{B.10}
$$

Arm3

$$
\begin{aligned}
&-\ddot{\theta}_1((m_3 l_3 l_1 + m_4 l_1 L_3)\cos(\theta_3) + m_4 l_4 l_1 \cos(\theta_3 + \theta_4)) + \ddot{\theta}_3(J_3 + J_4 + m_4 L_3^2 + 2 m_4 l_4 L_3 \cos(\theta_4)) \\
&+\ddot{\theta}_4(J_4 + m_4 l_4 L_3 \cos(\theta_4)) - \dot{\theta}_1^2(\frac{1}{2}(J_3 + m_4 L_3^2)\sin(2\theta_3) + J_4 \sin(\theta_3)\cos(\theta_4)\cos(\theta_3 + \theta_4)) \\
&-\dot{\theta}_1^2 \frac{1}{2} m_4 l_4 L_3 \sin(\theta_3)\cos(\theta_3 + \theta_4)(\cos(2\theta_4) + 3) + \dot{\theta}_3^2 m_4 l_4 L_3 \sin(\theta_4) - \dot{\theta}_4^2 m_4 l_4 L_3 \sin(\theta_4) \\
&-\dot{\theta}_1(J_4 \sin(\theta_4)\cos(\theta_3) + m_4 l_4 L_3 \sin(\theta_4)(\cos(\theta_3)\cos(\theta_4) + \cos(\theta_3)\sin(\theta_4) + \sin(\theta_3)^2 \cos(\theta_4))) \\
&-\dot{\theta}_3 m_4 l_4 L_3 \sin(\theta_4) + (m_3 + m_4) l_3 g \sin(\theta_3) + m_4 l_4 g \sin(\theta_3 + \theta_4) + \tau_3 = 0
\end{aligned}
\tag{B.11}
$$

Arm4

$$
\begin{aligned}
&-\ddot{\theta}_1 m_4 l_1 l_4 \cos(\theta_3 + \theta_4) + \ddot{\theta}_3(J_4 + m_4 l_4 L_3 \cos(\theta_4)) + \ddot{\theta}_4 J_4 \\
&-\dot{\theta}_1^2(J_4 \sin(\theta_3)\cos(\theta_4)\cos(\theta_3 + \theta_4) + m_4 l_4 L_3 \sin(\theta_3)\cos(\theta_3 + \theta_4)) \\
&+\dot{\theta}_3^2 m_4 l_4 L_3 \sin(\theta_4) - \dot{\theta}_1 J_4 \sin(\theta_4)\cos(\theta_3) + m_4 l_4 g \sin(\theta_3 + \theta_4) + \tau_4 = 0
\end{aligned}
\tag{B.12}
$$

B.4 Trippelpendel

Arm1

$$\begin{aligned}
&\ddot{\theta}_1(I_1 + m_2 l_1^2 + m_3 l_1^2 + m_4 l_1^2 + J_2 \sin(\theta_2)^2 + (J_3 + m_4 L_3^2)\sin^2(\theta_3) + J_4 \sin^2(\theta_3 + \theta_4)) \\
&+\ddot{\theta}_1 2 m_4 l_4 L_3 \sin(\theta_3)\sin(\theta_3 + \theta_4) + \ddot{\theta}_2 m_2 l_1 l_2 \cos(\theta_2) - \ddot{\theta}_3(m_3 l_1 l_3 + m_4 l_1 L_3)\cos(\theta_3) \\
&-\ddot{\theta}_3 m_4 l_1 l_4 \cos(\theta_3 + \theta_4) - \ddot{\theta}_4 m_4 l_1 l_4 \cos(\theta_3 + \theta_4) \\
&-\dot{\theta}_1^2 m_4 l_1 l_4(\sin(\theta_4)\cos(\theta_3) + \sin(\theta_3)\cos(\theta_4)\sin^2(\theta_3 + \theta_4)) - \dot{\theta}_2^2 m_2 l_1 l_2 \sin(\theta_2) \\
&+\dot{\theta}_3^2(m_3 l_1 l_3 + m_4 l_1 L_3)\sin(\theta_3) + \dot{\theta}_4^2 m_4 l_1 l_4 \sin(\theta_3 + \theta_4)) + \dot{\theta}_1 \dot{\theta}_2 J_2 \sin(2\theta_2) \\
&+\dot{\theta}_1 \dot{\theta}_3((J3 + m_4 L_3^2)\sin(2\theta_3) + \frac{1}{2} J_4 \sin(2(\theta_3 + \theta_4)) + \frac{1}{2} m_4 l_4 L_3(\sin(\theta_4) + 3\sin(2\theta_3 + \theta_4))) \\
&+\dot{\theta}_1 \dot{\theta}_4(J_4 \sin(2(\theta_3 + \theta_4)) + 2 m_4 l_4 L_3 \sin(\theta_3)\cos(\theta_3 + \theta_4)) \\
&+\dot{\theta}_1 m_4 l_1 l_4 \sin(\theta_4)\cos(\theta_3)(\cos(\theta_3 + \theta_4) + \sin(\theta_3 + \theta_4)) + \dot{\theta}_1 m_4 l_1 l_4 \sin(\theta_3)^2 \cos(\theta_4)\sin(\theta_3 + \theta_4) \\
&+\dot{\theta}_3((J_4 + m_4 l_1 l_4)\sin(\theta_3 + \theta_4) + m_4 l_4 L_3 \sin(\theta_3)) + \tau_1 = \tau_m
\end{aligned} \tag{B.13}$$

Arm2

$$\ddot{\theta}_1 m_2 l_1 l_2 \cos(\theta_2) + \ddot{\theta}_2 J_2 - \dot{\theta}_1^2 \frac{1}{2} J_2 \sin(2\theta_2) + l_2 m_2 g \sin(\theta_2) + \tau_2 = 0 \tag{B.14}$$

Arm3

$$\begin{aligned}
&-\ddot{\theta}_1((m_3 l_3 l_1 + m_4 l_1 L_3)\cos(\theta_3) + m_4 l_4 l_1 \cos(\theta_3 + \theta_4)) + \ddot{\theta}_3(J_3 + J_4 + m_4 L_3^2 + 2 m_4 l_4 L_3 \cos(\theta_4)) \\
&+\ddot{\theta}_4(J_4 + m_4 l_4 L_3 \cos(\theta_4)) - \dot{\theta}_1^2(\frac{1}{2}(J_3 + m_4 L_3^2)\sin(2\theta_3) + J_4 \sin(\theta_3)\cos(\theta_4)\cos(\theta_3 + \theta_4)) \\
&-\dot{\theta}_1^2 \frac{1}{2} m_4 l_4 L_3 \sin(\theta_3)\cos(\theta_3 + \theta_4)(\cos(2\theta_4) + 3) + \dot{\theta}_3^2 m_4 l_4 L_3 \sin(\theta_4) - \dot{\theta}_4^2 m_4 l_4 L_3 \sin(\theta_4) \\
&-\dot{\theta}_1(J_4 \sin(\theta_4)\cos(\theta_3) + m_4 l_4 L_3 \sin(\theta_4)(\cos(\theta_3)\cos(\theta_4) + \cos(\theta_3)\sin(\theta_4) + \sin(\theta_3)^2 \cos(\theta_4))) \\
&-\dot{\theta}_3 m_4 l_4 L_3 \sin(\theta_4) + (m_3 + m_4) l_3 g \sin(\theta_3) + m_4 l_4 g \sin(\theta_3 + \theta_4) + \tau_3 = 0
\end{aligned} \tag{B.15}$$

Arm4

$$\begin{aligned}
&-\ddot{\theta}_1 m_4 l_1 l_4 \cos(\theta_3 + \theta_4) + \ddot{\theta}_3(J_4 + m_4 l_4 L_3 \cos(\theta_4)) + \ddot{\theta}_4 J_4 \\
&-\dot{\theta}_1^2(J_4 \sin(\theta_3)\cos(\theta_4)\cos(\theta_3 + \theta_4) + m_4 l_4 L_3 \sin(\theta_3)\cos(\theta_3 + \theta_4)) \\
&+\dot{\theta}_3^2 m_4 l_4 L_3 \sin(\theta_4) - \dot{\theta}_1 J_4 \sin(\theta_4)\cos(\theta_3) + m_4 l_4 g \sin(\theta_3 + \theta_4) + \tau_4 = 0
\end{aligned} \tag{B.16}$$

Tillegg C

Modellparametre

C.1 Motor og Arm1

Parameter	Verdi
K_e	0.84 [Vs/rad]
K_t	0.84 [Nm/rad]
J_a	0.0016 [Kgm^2]
Max I	30 [A]

Tabell C.1: Oppgitte Motorparametre

Parameter	Verdi
m_1	0.645 [Kg]
l_1	0.5 [m]
I_1	0.0016 [Kgm^2]

Tabell C.2: Målte parametre for Arm1

Under estimering er motorligningene satt opp slik

$$\ddot{\theta}_1 = -K_{m1}\dot{\theta}_1 + K_{m2}i_a - K_{m3}\text{sign}(\dot{\theta}_1) - K_{m4}e^{-K_{m5}|\dot{\theta}_1|}\text{sign}(\dot{\theta}_1) \tag{C.1}$$

$$\dot{i}_a = K_{m8}(-K_{m6}\dot{\theta}_1 - K_{m7}i_a + u) \tag{C.2}$$

$$u = K_{m11}(1 + K_{m10}\frac{1}{s}(K_{m9}i_d - i_a)) \tag{C.3}$$

Parameter	Positiv retning	Negativ retning
K_{m1}	0.1249	0
K_{m2}	20.0636	-
K_{m3}	0.8265	0.4653
K_{m4}	0.7843	1.4039
K_{m5}	0.0025	-
K_{m6}	1.1343	-
K_{m7}	15	-
K_{m8}	0.9708	-
K_{m9}	0.0103	-
K_{m10}	289.8932	-
K_{m11}	65.4959	-

Tabell C.3: Estimerte parametre for motor. parametre som er like i begge retninger markeres med '-'

C.2 Arm2, lang pendel

Målte Parameter	Verdi	Identifiserte parametre	Verdi
m_2	0.19 [Kg]	b_{c2}	0.0021
L_2	1.105 [m]	b_{l2}	0.00276
l_2	0.53 [m]	b_{lm2}	$-2.14 \cdot 10^{-}3$
J_2	0.0735 [Kgm^2]		

Tabell C.4: Målte parametre for Arm2

C.3 Arm3, kort pendel

Målte parameter	Verdi	Identifiserte parametre	Verdi
m_3	0.064 [Kg]	b_{c3}	0.00032
L_3	0.565 [m]	b_{l3}	0.00016
l_3	0.24 [m]	b_{lm3}	$-0.3 \cdot 10^{-}3$
J_3	0.00569 [Kgm^2]		

Tabell C.5: Målte parametre for Arm3

Tillegg D

Regulatorparametre

Parameter	Verdi
K_1	-1
K_2	-355.7785
K_3	-32.1367
K_3	-63.3726

Tabell D.1: LQR-parametre

Parameter	Verdi
K_1	0
K_2	-1000
K_3	-15.1879
K_3	-71.4523

Tabell D.2: Regulatorparametre funnet med genetisk algoritme

Tillegg E

Genetisk algoritme

Den genetiske algoritmen kan lett brukes ved å sette antall parametre, `n`, som skal estimeres og definere optimaliseringsområdets øvre grense i `K_max`. Nedre grense i algoritmen er alltid 0. Dersom det ønskes en annen nedre grense kan dette implementeres i modellen ved å benytte parametre på formen `K_min+Pr` hvor `Pr` er parametren som skal estimeres. Navnet på simulinkmodellen fylles inn i algoritmen. Utgangen i modellen bestemmer objektiv funksjon. Algoritmen akkumulerer utgangen fra modellen for å regne verdi av objektiv funksjon.

```
function [ F ] = Simulation( K,modell,N )
%SIMULATION simulerer modell med parametersett K og returnerer resultat
%
%K       - Parametersett
%modell - Streng med navn paa simuleringsmodell
%N       - Antall datapunkt i simuleringen
%
%F       - Resultat


global Pr sim_n
F=0;

%Simuler modell
Pr=K;
sim(modell)
sim_n=sim_n+1;

%regner ut objektiv funksjon
for j = 1:N
    F=F+yout(j);
end

end
```

```
function [ Ks, Fs ] = GenAlg_Rangering( K,modell,N )
%GENALG_RANGERING Rangerer parametersett i K fra best til daarligst
%
```

```
4  %Funksjonen returnerer resultatene Fs i sortert rekkefolge fra best til
5  %daarligst med tillhorende parametersett Ks.
6  %
7  %K       - Parametersett
8  %modell - Streng med navn paa simuleringsmodell
9  %N       - Antall datapunkt i simuleringen
10 %
11 %Ks      - Parametersett i sortert rekkefolge fra best til daarligst
12 %Fs      - Resultater av parametersett
13
14 n = size(K,2);
15 F=zeros(n,1);
16
17
18 %Simulerer parametersett i K og legger resultatene i F
19 for i=1:n
20     F(i)=Simulation(K(:,i),modell,N);
21 end
22
23 %Setter opp indeksmatrise slik at I = [1  2  3  4...]^T
24 I=zeros(n,1);
25 for i = 1:(n)
26     I(i)=i;
27 end
28
29 FI=[F I];                %Kombinerer F og I
30 FI_s=MergeSort(FI,n);    %Sorteringsalgoritme. Sorterer med hensyn paa F
31 Fs=FI_s(:,1);            %Resultater i sortert rekkefolge
32 r=FI_s(:,2);            %Rekkefolge paa indekser etter sortering
33
34 %Setter parametersett i rangert rekkefolge
35 for i = 1:n
36     Ks(:,i)=K(:,r(i));
37 end
38
39 end
```

```
1  function [ rv,rc ] = GenAlg_Selection( C,p_c )
2  %GENALG_SELECTION Utvalg av parametersett til neste generasjon
3  %
4  %Funksjonen finner stokastisk utvalg av parametersett til neste generasjon
5  %paa bakgrunn av de kumulative sannsynelighetene og returnerer indeks paa
6  %utvalgte parametere hvor det 1 er den beste. Av de utvalgte parameterene
7  %plukkes det stokastisk ut par til krysning paa bakgrunn av
8  %Krysningskoeffisienten.
9  %
10 %C   - Kumulative utvelgelsessannsynligheter
11 %p_c - Krysningskoeffisient
12 %
13 %rv  - Indeks paa utvalgte parametersett
14 %rc  - Indeks paa parametersett i rv som er utvalgt til krysning
15
16 n=size(C,1);          %Finner antall parametersett
17 rv=zeros(n,1);
```

```
18 rc=zeros(2,n/2);
19
20 U=rand(n,1);          %Genererer stokastiske verdier til sammenligning
21 Uc=rand(n,1);
22
23
24 %For hvert element i U velges tilhorlig indeks etter  det geometriske
25 %normaliseringsprinsippet.
26 for i=1:n
27     if U(i) <= C(1)
28          rv(i)=1;
29          continue
30     end
31     for j=2:n
32          if (U(i) <= C(j))&&(U(i) > C(j-1))
33              rv(i)=j;
34              continue
35         end
36     end
37 end
38
39 %For hvert element i Uc plukkes tilhorende indeks til krysning med
40 %en sannsynlighet lik p_c
41 j=1;
42 l=1;
43 for i=1:n
44     if Uc(i) <= p_c
45          rc(j,l)=i;
46          j=j+1;
47          if j==3
48               j=1;
49               l=l+1;
50          end
51     end
52 end
53
54 end
```

```
1 function [ K_m ] = GenAlg_Mutation( K,K_max,p_m,pr )
2 %MUTATION. Binaer mutasjon av K
3 %
4 %Funksjonen bytter om hvert bit i K med en sannsynlighet lik p_m.
5 %
6 %K_max - maksimal verdi som tillates for K
7 %p_m   - mutasjonssannsynlighet
8 %pr    - presisjon (antall desimaler)
9 %
10 %K_m   - Mutert versjon av K
11
12 K_m=0;
13 a='';
14 n=ceil(log2(K_max*(10^pr))); %regner ut ordlengde
15
16
```

```
%Plaserer bit for bit i a med p_m sannsynelighet for aa plasere en 1-er
for i = 1:n
    if rand < p_m
        a=strcat(a,'1');
    else
        a=strcat(a,'0');
    end
end

b=floor(K*10^pr);
%For hver 1-er i a blir tilsvarende bit togglet i K ved hjelp av ekslusiv
%eller
K_m=bitxor(bin2dec(a),b)/(10^pr);

%Begrenser K_m til ovre tillatte grense
if K_m > K_max
    K_m=K_max;
end

end
```

```
function [ K_ab,K_ba ] = GenAlg_Crossover( K_a,K_b,K_max,pr )
%CROSSOVER. Enkel binaer 1.-punkts krysning mellom K_a og K_b
%
%Funksjonen plukker ut et punkt i den binaere koden for K_a og K_b.
%Informasjonen etter dette punktet blir saa byttet om.
%
%K_a   - Parametersett a
%K_b   - Parametersett b
%K_max - Hoyeste verdi som tillates
%pr    - Presisjon
%
%K_ab  - Informasjon fra K_a for og K_b etter krysningspunkt
%K_ba  - Informasjon fra K_b for og K_a etter krysningspunkt

%Binaer enkoding med ordlengde n
n=ceil(log2(K_max*(10^pr)));
a=dec2bin(K_a*10^pr,n);
b=dec2bin(K_b*10^pr,n);

%Stokastisk generering av krysningspunkt
s=randi(n-1);

%Sammensetning
ab=strcat(a(1:s),b(s+1:n));
ba=strcat(b(1:s),a(s+1:n));

%Begrensning av totalverdi
K_ab=bin2dec(ab)/(10^pr);
if K_ab > K_max
    K_ab=K_max;
end
K_ba=bin2dec(ba)/(10^pr);
if K_ba > K_max
```

```
34      K_ba=K_max;
35  end
36
37  end
```

```
1  %Genetisk algoritme
2
3  %Initiering
4  n=11;                    %Antall parametere som skal estimeres
5  m=80;                    %Antall parametersett i befolkning
6  gen=100;                 %Antall generasjoner
7  p_c=0.6;                 %Krysningskoeffisient
8  p_m=0.1;                 %Mutasjonskoeffisient
9  q=0.08;                  %Sannsynlighet for utvelgelse av beste parametersett
10 pr=4;                    %Presisjon
11
12 global Pr sim_n
13 Pr=zeros(n,1);           %Simuleringsmodellparametere
14 sim_n=0;                 %Til opptelling av antall simuleringer
15 N=1000;                  %Antall datapunkt i simuleringsrespons
16 modell='Reg';%Simuleringsmodell
17
18 Kg=zeros(n,m);           %Befolkningsmatrise
19 Kng=zeros(n,m);          %Neste generasjons befolkningsmatrise
20 new_gen=zeros(n,1);      %Utvalgte parametersett
21 Kmax=zeros(n,1);         %ovre grense av optimaliseringsomraadet
22 Fg=zeros(m,1);           %Simuleringsresultatmatrise
23
24 Kmax(1)=1;
25 Kmax(2)=20;
26 Kmax(3)=150;
27 Kmax(4)=50;
28 Kmax(5)=10;
29 Kmax(6)=30;
30 Kmax(7)=20;
31 Kmax(8)=1;
32 Kmax(9)=1;
33 Kmax(10)=1;
34 Kmax(11)=1;
35 %Generering av initialbefolkning
36 for i=1:n
37 Kg(i,:)=rand(1,m)*Kmax(i);
38 end
39
40
41 %Utregning av individuell og kumulativ uvelgelsessannsynlighet
42 P=zeros(m,1);          %Individuelle uvelgelsessannsynligheter
43 C=zeros(m,1);          %Kumulative utvelgelsessannsyneligheter
44 Q=q/(1-(1-q)^m);
45 for i=1:m
46      P(i)=Q*((1-q)^(i-1));    %Utregning av P
47 end
48
49 for i=1:m
```

```
50      for j=1:i
51          C(i)=C(i)+P(j);      %Utregning av C
52      end
53  end
54
55
56  %Algoritmestruktur
57  for i=1:gen
58
59      %Rangering og sortering av parametersett
60      [Kg,Fg] = GenAlg_Rangering( Kg,modell,N );
61
62      %Utvelgelse
63      %new_gen inneholder indekser paa utvalgte parametersett
64      %rc inneholder indekser paa par til krysning
65      [new_gen,rc] = GenAlg_Selection( C,p_c );
66
67      %Mutasjon
68      for j=1:m
69
70          %De utvalgte parametersettene plasseres i Kng
71          Kng(:,j)=Kg(:,new_gen(j));
72
73          %Hver parameter muteres med funksjonen GenAlg_Mutation
74          for k=1:n
75              [ Kng(k,j) ] = GenAlg_Mutation( Kng(k,j),Kmax(k),p_m,pr );
76          end
77
78      end
79
80      %Krysning
81      for j=1:m
82
83          %Underoker at det forsatt er flere par som skal krysses.
84          %Dersom det ikke er flere par igjen blir for-sloyfen brutt
85          if (rc(1,j)*rc(2,j))>0
86              %Parameterparene krysses med funksjonen GenAlg_Crossover
87              for k=1:n
88                [ Kng(k,rc(1,j)),Kng(k,rc(2,j)) ] = ...
89                GenAlg_Crossover( Kng(k,rc(1,j)),Kng(k,rc(2,j)),Kmax(k),pr );
90              end
91          else
92              break
93          end
94      end
95
96      %Den nye generasjonen Kng plasseres i befolkningen Kg
97      Kg=Kng;
98
99      %Skriver ut iterasjonsnummeret
100     i
101
102 end
103
```

```
104 %Naar algoritmen er over sorteres den siste generasjonen av praktiske
        hensyn
105 [Kg,Fg] = GenAlg_Rangering( Kg,modell,N );
```

Tillegg F

Nelder-Mead Simplex

Nelder-Mead algoritmen benytter `Kg` fra den genetiske algoritmen som inngang.

```
function [ F ] = NelderMead_Rangering( K,N,modell )
%NELDERMEAD_RANGERING Rangering av simplexelementer
%
%K       - Simplex
%N       - Antall datapunkt i simplexen
%modell - Simuleringsmodell
%
%F       - Simuleringsresultat

n=size(K,1);
F=zeros(n+1,1);

%Simulerer hvert punkt i simplexen
    for i=1:(n+1)
        F(i)=Simulation(K(:,i),modell,N);
    end

end
```

```
function [ Ks, Fs ] = NelderMead_Sort( K,F )
%NELDERMEAD_SORT Sorterer simplexelementene i sortert rekkefolge
%
%K  - Simplex
%F  - Simuleringsresultater
%
%Ks - Sortert simplex
%Fs - Sorterte simuleringsresultater

n=size(K,1);
Ks=zeros(n,n+1);

%Oppretter indekseringsnummerering
I=zeros(n+1,1);
for i = 1:(n+1)
     I(i)=i;
end
```

```
18
19 FI=[F I];
20 FI_s=MergeSort(FI,n+1); %Sorteringsalgoritme
21 Fs=FI_s(:,1);           %Sorterte simuleringsresultater
22 r=FI_s(:,2);            %indekseringsrekkefolge
23
24 %Setter parametersett i rangert rekkefolge
25 for i = 1:(n+1)
26      Ks(:,i)=K(:,r(i));
27 end
28
29 end
```

```
1  function [ Kce ] = NelderMead_Centroid( K )
2  %NELDERMEAD_CENTROID Regner ut tyngdepunkt av simplex K
3  %
4  %K   - Simplex
5  %Kce - Tyngdepunkt
6
7  n=size(K,1);
8  Kce=zeros(n,1);
9
10 %Utregning av tyngdepunkt
11 for i=1:n
12     Kce=Kce+K(:,i);
13 end
14 Kce=Kce/n;
15
16 %Begrenser Kce slik at den ikke blir negativ
17 for i=1:n
18     if Kce(i) < 0
19         Kce(i)=0;
20     end
21 end
22
23 end
```

```
1  function [ Kr, Fr ] = NelderMead_Reflection( Kce, Kn1, ro, N, modell )
2  %NELDERMEAD_REFLECTION Utregning av refleksjonspunkt
3  %
4  %Kce    - Tyngdepunkt
5  %Kn1    - Daarligste punkt(n+1) i simplexen
6  %ro     - Refleksjonskoeffisient
7  %N      - Antall datapunkt i simuleringsrespons
8  %modell - Simuleringsmodell
9  %
10 %Kr     - Ekspansjonspunkt
11 %Fr     - Simuleringsresultat av Kr
12
13 n=size(Kce,1);
14
15 %Utregning og begrensning
16 Kr = Kce+ro*(Kce-Kn1);
```

```
for i=1:n
    if Kr(i) < 0
        Kr(i)=0;
    end
end

%Simulering
Fr=Simulation(Kr,modell,N);

end
```

```
function [ Ke, Fe ] = NelderMead_Expand( Kce, Kr, mu, N, modell )
%NELDERMEAD_EXPAND Utregning av ekspansjonspunkt
%
%Kce     - Tyngdepunkt
%Kr      - Refleksjonspunkt
%mu      - Ekspansjonskoeffisient
%N       - Antall datapunkt i simuleringsrespons
%modell  - Simuleringsmodell
%
%Ke      - Ekspansjonspunkt
%Fe      - Simuleringsresultat av Ke

n=size(Kce,1);

%Utregning og begrensning
Ke = Kce+mu*(Kr-Kce);
for i=1:n
    if Ke(i) < 0
        Ke(i)=0;
    end
end

%Simulering
Fe=Simulation(Ke,modell,N);

end
```

```
function [ Kcc, Fcc ] = NelderMead_ContractInside( Kce, Kn1, lambda, N,
    modell )
%NELDERMEAD_CONTRACTINSIDE Utregning av indre sammentrekningspunkt
%
%Kce     - Tyngdepunkt
%Kn1     - Daarligste punkt(n+1) i simplexen
%lambda  - Sammentrekningskoeffisient
%N       - Antall datapunkt i simuleringsrespons
%modell  - Simuleringsmodell
%
%Kcc     - Indre sammentrekningspunkt
%Fcc     - Simuleringsresultat av Kcc

n=size(Kce,1);

```

```
15 %Utregning og begrensning
16 Kcc = Kce-lambda*(Kce-Kn1);
17 for i=1:n
18     if Kcc(i) < 0
19         Kcc(i)=0;
20     end
21 end
22
23 %Simulering
24 Fcc=Simulation(Kcc,modell,N);
25
26 end
```

```
1  function [ Kc, Fc ] = NelderMead_ContractOutside( Kce, Kr, lambda, N,
       modell )
2  %NELDERMEAD_CONTRACTOUTSIDE Utregning av ytre sammentrekningspunkt
3  %
4  %Kce     - Tyngdepunkt
5  %Kr      - Refleksjonspunkt
6  %lambda  - Sammentrekningskoeffisient
7  %N       - Antall datapunkt i simuleringsrespons
8  %modell  - Simuleringsmodell
9  %
10 %Kc      - Ytre sammentrekningspunkt
11 %Fc      - Simuleringsresultat av Kc
12
13 n=size(Kce,1);
14
15 %Utregning og begrensning
16 Kc = Kce+lambda*(Kr-Kce);
17 for i=1:n
18     if Kc(i) < 0
19         Kc(i)=0;
20     end
21 end
22
23 %Simulering
24 Fc=Simulation(Kc,modell,N);
25
26 end
```

```
1  function [ Kv, Fv ] = NelderMead_Shrink( K, sigma, N, F1, modell )
2  %NELDERMEAD_SHRINK Reduksjon av simplex
3  %
4  %K       - Simplex
5  %sigma   - Reduksjonskoeffisient
6  %N       - Antall datapunkt i simuleringsrespons
7  %F1      - Simuleringsresultat av det beste punktet i simplexen
8  %modell  - Simuleringsmodell
9  %
10 %Kv      - Redusert Simplex
11 %Fv      - Simpuleringsresultat av Kv
12
```

```
13 n=size(K,1);
14 Kv=zeros(n,n+1);
15 Fv=zeros(n+1,1);
16
17 %Plasserer det beste punktet fra den gamle simplexen over i den nye
18 Kv(:,1)=K(:,1);
19 Fv(1)=F1;
20
21 %Regner ut resterende elementer til den nye simplexen
22 for i=2:(n+1)
23     Kv(:,i)=K(:,1) + sigma*(K(:,i)-K(:,1));
24 end
25
26 %Simulerer
27 for i=2:(n+1)
28     Fv(i)=Simulation(K(:,i),modell,N);
29 end
30
31 end
```

```
1  function [ s ] = NelderMead_StopCond( K )
2  %NELDERMEAD_STOPCOND Regner ut konvergens av algoritmen
3
4  n=size(K,1);
5  a=0;
6  s=0;
7  for i=2:(n+1)
8      Ka=K(:,i)-K(:,1);
9      for j=1:n
10         a=a+Ka(j)*Ka(j);
11     end
12     s=s+a;
13     a=0;
14 end
15 s=s/(n+1);
16 s=sqrt(s);
17
18 end
```

```
1  %Nelder-Mead simplex algoritme
2
3  %Initiering
4  global sim_n
5  sim_n=0;                %Til opptelling av antall simuleringer
6  N=1000;                 %Antall datapunkt i simuleringsresponsen
7  modell='Reg';           %Simuleringsmodell
8  n=3;                    %Antall parametere som skal estimeres
9  ro=1;                   %Refleksjonskoeffisient
10 lambda=0.75;            %Sammentrekningskoeffisient
11 mu=2;                   %Ekspansjonskoeffisient
12 sigma=0.99;             %Reduksjonskoeffisient
13 time_out=1500;          %Maksimalt antall iterasjoner
14
```

```
15 Kce=zeros(n+1,1);          %Tyngdepunkt
16 Kr=zeros(n+1,1);           %Refleksjonspunkt
17 Ke=zeros(n+1,1);           %Ekspansjonspunkt
18 Kc=zeros(n+1,1);           %Ytre Sammentreknisngspunkt
19 Kcc=zeros(n+1,1);          %Indre Sammentreknisngspunkt
20 F=zeros(9,1);              %Simuleringsresultater
21
22 %Initiering av simplex
23 K=Kg(:,1:n+1);
24 %Simulering til rangering av parametersett i simplex
25 F = NelderMead_Rangering( K,N,modell );
26
27
28 %Algoritmestruktur
29 for i=1:time_out
30
31     %Sortering av simplexelementer fra best til daarligst
32     [ K, F ]= NelderMead_Sort( K,F );
33
34     %Utregning av tyngdepunkt
35     Kce = NelderMead_Centroid( K );
36
37     %Utregning av refleksjonspunkt
38     [ Kr, Fr ] = NelderMead_Reflection( Kce, K(:,n+1), ro, N, modell );
39
40     %Hvis refleksjonspunktet ikke er bedre eller daarligere enn de n-1
           beste
41     %punkt i simplexen velges Kr til aa erstatte det daarligste punktet
42     if (Fr >=F(1)) && (Fr < F(n))
43         K(:,n+1)=Kr;
44         F(n+1)=Fr;
45     end
46
47     %Hvis refleksjonspunktet er bedre enn det beste punktet i simplexen
48     %forsokes det aa ekspandere simplexen for aa ta storre skritt
49     if Fr < F(1)
50
51         %Utregning av ekspansjonspunkt
52         [ Ke, Fe ] = NelderMead_Expand( Kce, Kr, mu, N, modell );
53
54         %Hvis ekspansjonspunktet er bedre enn refleksjonspunktet
55         %velges Ke til aa erstatte det daarligste punktet i simplexen.
56         %Hvis ikke velges Kr
57         if Fe < Fr
58             K(:,n+1)=Ke;
59             F(n+1)=Fe;
60         else
61             K(:,n+1)=Kr;
62             F(n+1)=Fr;
63         end
64     end
65
66     %Hvis refleksjonspunktet er minst like daarlige som det n beste punktet
           ,
67     %men bedre enn det daarligste forsokes ytre sammentrekning for aa ta
```

```
68      %mindre skritt.
69      if (Fr >=F(n)) && (Fr < F(n+1))
70
71          %Utregning av ytre sammentrekningspunkt
72          [ Kc, Fc ] = NelderMead_ContractOutside( Kce, Kr, lambda, N, modell
                );
73
74          %Dersom sammentrekningspunktet er bedre enn refleksjonspunktet
75          %beholdes sammentrekningspunktet. Hvis ikke krympes hele simplexen
76          if Fc < Fr
77              K(:,n+1)=Kc;
78              F(n+1)=Fc;
79          else
80              [ K, F ] = NelderMead_Shrink( K, sigma, N, F(1), modell );
81          end
82      end
83
84      %Dersom refleksjonspunktet er daarligere enn punktet som blir
            reflektert
85      %forsokes en indre sammentrekning for aa holde sammentrekningspunktet
86      %innenfor simplexen
87      if Fr >= F(n+1)
88
89          %Utregning av indre sammentrekningspunkt
90          [ Kcc, Fcc ] = NelderMead_ContractInside( Kce, K(:,n+1), lambda, N,
                modell );
91
92          %Dersom sammentrekningspunktet er bedre enn refleksjonspunktet
93          %beholdes sammentrekningspunktet. Hvis ikke krympes hele simplexen
94          if Fcc < F(n+1)
95              K(:,n+1)=Kcc;
96              F(n+1)=Fcc;
97          else
98              [ K, F ] = NelderMead_Shrink( K, sigma, N, F(1), modell );
99          end
100     end
101
102     i       %Skriver ut iterasjonsnummeret
103
104     %Algoritmen avsluttes dersom den oppnaar tilstrekkelig konvergens
105     s = NelderMead_StopCond( K )
106     if s <= 0.0001
107         break
108     end
109
110 end
```

Tillegg G

Mergesort

Sorteringsalgoritmen som benyttes i den genetiske algoritmen og Nelder-Mead

```
function [ B ] = MergeSort( A,n )
%MERGESORT Sorteringsalgoritme
%
%Sorterer radene i A med hensyn paa forste kolonne.
%
%A - Matrise som skal sorteres
%n - Antall elementer
%B - Sortert matrise

%Deler A og starter rekurrensen
B = MergeSort_Split(A, 1,  n+1);

end
```

```
function [ B ] = MergeSort_Split( A,iBegin, iEnd )
%SPLITMERGE Oppdelingsbiten av Mergesortalgoritmen
%
%Deler A(iBegin-iEnd) i to. Kjorer rekursivt og setter til slutt sammen de
%sorterte halvdelene
%
%iBegin - inklusiv indeks som markerer starten av listen
%iEnd   - exclusive indeks som markerer slutten av listen

%Finner relative start og slutt indekser
ib=1;
ie=1+iEnd-iBegin;

if(ie - ib < 2)                         % Dersom storrelse==1
    B=A;                                % Anse som sortert
else
    im = floor((ie + ib) / 2);          %Finner midtpunkt

    %Kjorer rekursivt og faar tilbake sorterte halvdeler
    Al = MergeSort_Split(A(ib:(im-1),:), ib,  im);
    Ar = MergeSort_Split(A(im:(ie-1),:), im, ie);
    Am=[Al;Ar];
```

```
23
24      %Sorter de to halvdelene
25      B = MergeSort_Merge(Am, ib, im, ie);
26  end
27
28  end
```

```
1   function [ B ] = MergeSort_Merge( A, iBegin, iMiddle, iEnd)
2   %MERGESORT_MERGE Sammensetningsdelen av algoritmen
3   %
4   %Sorterer de to listene som er markert av iBegin, iMiddle og iEnd
5
6   %Finner de relative punktene
7   i0 = 1;
8   i1 = 1+iMiddle-iBegin;
9   im = 1+iMiddle-iBegin;
10  ie=1+iEnd-iBegin;
11
12  % Setter sammen element for element av listene i sortert rekkefolge
13  for j = 1:(ie-1)
14      if (i0 < im) && (i1 >= ie || A(i0) <= A(i1))
15          B(j,:) = A(i0,:);
16          i0 = i0 + 1;
17      else
18          B(j,:) = A(i1,:);
19          i1 = i1 + 1;
20      end
21  end
22
23  end
```

Bibliografi

[1] Dionysios C. Aliprantis, Scott D. Sudhoff, and Brian T. Kuhn. Genetic algorithm-based parameter identification of a hysteretic brushless exciter model. *IEEE TRANSACTIONS ON ENERGY CONVERSION, VOL. 21, NO. 1*, 2006.

[2] Benjamin Seth Cazzolato and Zebb Prime. On dynamics of the foruta pendulum. *Journal of Control Science and Engineering*, 2010.

[3] Shuang Cong, Guodong Li, and Xianyong Feng. Parameters identification of nonlinear dc motor model using compound evolution algorithms. *Proceedings of the World Congress on Engineering*, 2010.

[4] Sindre Hansen. Forstudie til styring av trippelinvertert pendelsystem, 2014.

[5] Adam Himes. Triple inverted pendulum, 2014.

[6] John Henry Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan press, 1975.

[7] Tolgay Kara and IIlyas Eker. Nonlinear modeling and identification of a dc motor for bidirectional operation with real time experiments. *Energy Conversion and Management*, 2003.

[8] Ruben Kristiansen. Pre-study for upswing and stabilization of a triple inverted pendulum. Master's thesis, Norges Teknisk-Naturvitenskapelig Universitet, 2006.

[9] John A. Nelder and Roger Mead. A simplex method for function minimization. *Computer Journal 7: 308–313*, 1965.

[10] Vegard Åstebøl Larssen. Modeling, control and simulation of inverted pendulum systems. Master's thesis, Norges Teknisk-Naturvitenskapelig Universitet, 2006.

[11] Alan M. Turing. Computing machinery and intelligence. *Oxford Journals. Mind LIX (236)*, 1950.

[12] Leehter Yao and William A. Sethares. Nonlinear parameter estimation via the genetic algorithm. *Proceedings of the World Congress on Engineering*, 1994.

[13] Øyvind Bjørnson-Langen. Modellering og regulering av sirkulært opphengt invertert multipendel. Master's thesis, Norges Teknisk-Naturvitenskapelig Universitet, 2007.