Technical University of Denmark

Kenneth Marx Hoe

Regulering af ustabilt system baseret på inertialmålinger

Bachelorprojekt, maj 2012

DTU Electrical Engineering
Department of Electrical Engineering

Regulering af ustabilt system baseret på inertialmålinger
Control of Unstable System Based on Inertial Measurements

Rapporten er udarbejdet af:
Kenneth Marx Hoe

Vejleder(e):
Nils Axel Andersen
Ole Ravn

DTU Elektro
Automation and Control
Danmarks Tekniske Universitet
Elektrovej
Bygning 326
2800 Kgs. Lyngby
Danmark
Tlf. : +45 4525 3576
Fax : +45 4588 1295

studieadministration@elektro.dtu.dk

Projektperiode:	01/02-2012 – 31/05-2012
ECTS:	15
Uddannelse:	Bachelor
Retning:	Elektroteknologi
Klasse:	Offentlig
Bemærkninger:	Denne rapport er indleveret som led i opfyldelse af kravene til ovenstående uddannelse på Danmarks Tekniske Universitet.
Rettigheder:	

Abstract

This project deals with the problem of stabilising an inverted pendulum in the form of a tightrope walker model, which is based on inertial calculations. A theoretical model of the system has been made and modelled into MATLAB/Simulink, where it illustrates how a tightrope walker keeps its balance by aid of a balancing beam to control its torque. A controller based on the model has been created by an LQR-design, which afterwards has been implemented in a program running on an RTAI-Linux machine. From measurements with an accelerometer, a gyroscope, and a potentiometer on the system it appears, that the created model illustrates the real tightrope walker system very well. The final controller design accomplished a stabilisation of the tightrope walker setup with less than a ± 0.3 degrees variation and a limit on the starting angle of the pendulum without toppling of around 10 degrees. The project is concluded with the introduction of the theoretical work in the further development of the project, that is designing a self-balancing unicycle.

Resumé

Dette projekt beskæftiger sig med stabilisering af et omvendt pendul i form af en linedansermodel, der er baseret på inertialberegninger. Der er opstillet en teoretisk model for systemet, som er modelleret i MATLAB/Simulink, hvor det illustrerer, hvordan en linedanser opretholder balancen ved hjælp af en balancestang til at kontrollere kraftmomentet. Baseret på modellen er en regulator bestemt ud fra LQR-design, hvorefter den implementeres i et program kørende på en RTAI-Linux maskine. Ud fra målinger på systemet med accelerometer, gyroskop og potentiometer har det vist sig, at den opstillede model illustrerer linedansersystemet meget godt. Det endelige regulatordesign har formået at stabilisere linedanseropstillingen inden for $\pm 0,3$ grader og en vinkelgrænse, som pendulet kan starte fra uden at vælte på ca. 10 grader. Projektet afsluttes ved at dække det indledende teoretiske arbejde i videreudvikling af linedanseropstillingen for at designe en selvbalancerende unicykel.

Indhold

Figurer

Kapitel 1

Indledning

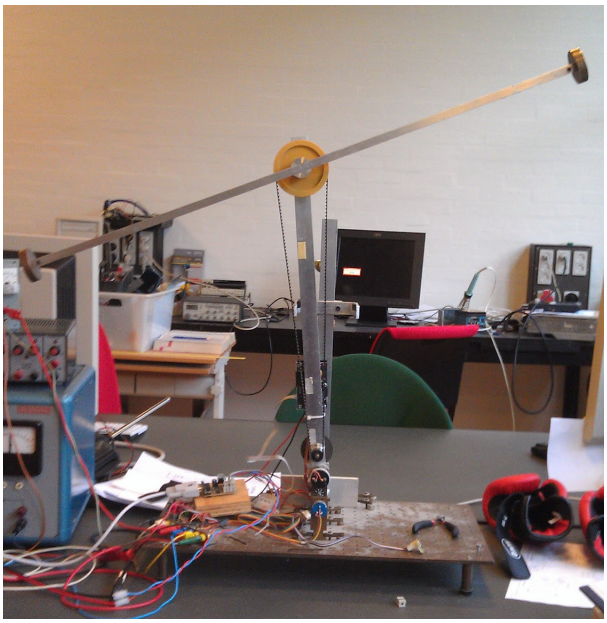
Reguleringsteknik beskæftiger sig ofte med henblik på at stabilisere et umiddelbart ustabilt system. Stabilisering opnås gennem design og implementering af regulatorer, som styrer systemet, så det besidder en ønsket dynamik. Et omvendt pendul er en ofte benyttet opstilling inden for reguleringsteknik, der som udgangspunkt er et ustabilt system. Det omvendte pendul har længe været en opstilling benyttet som eksempel til at give studerende inden for reguleringsteknik et indblik i regulering i praksis. Med de rette regulatorer er det muligt at stabilisere pendulet til at stå lodret ved hjælp af en eller flere aktuatorer. Ofte har opstillingerne en motor som aktuator, der enten giver et kraftmoment direkte til pendulet eller styrer en slæde, der kan køre fra side til side og på den måde frembringe et moment i pendulstangen. Dette projekt retter sig mod stabiliseringen af et sådant system, hvor det kunne være interessant at regulere systemet ved hjælp af en anden metode end de klassiske oplagte.

Tidligere DTU-projekter, gennemført ved "Automation and Control", har arbejdet med en opstilling indeholdende et omvendt pendul, der skulle illustrere balancen af en linedanser, som ved hjælp af en balancestang nemmere kan opretholde balancen (se figur 1.1). Linedanseropstillingen er oprindeligt lavet af Michael Heisel Nielsen i 1982 ved "Servolaboratoriet" ved DTU - nu "Automation and Control". Ved en vinkelacceleration af en fastgjort balancestang bliver der gennem stangens inertimoment oprettet et kraftmoment, som påvirker pendulstangens moment. Med den rette model af inertialberegninger for systemet er det muligt at styre balancestangen, så en ønsket opretholdelse af pendulets balance opnås.

Den ovenstående metode er en interessant måde at regulere systemet, da det er en anderledes metode til stabilisering af et omvendt pendul, idet systemet styres af et kraftmoment, som gives til toppen af pendulstangen modsat de typiske opstillinger, hvor der gives et kraftmoment til bunden af pendulstangen gennem eksempelvis en skubbende vogn.

Dette projekt forsøger at få et sådant pendul til at stabilisere sig. Systemet modelleres først, hvor der ved forskellige LQR-designs (Linear Quadratic Regulator) forsøges at få modellen til at stabilisere sig gennem simuleringer i MATLAB/Simulink. Her er det relevant at designe en regulator, som giver en kort

indsvingningstid for systemet samtidig med et acceptabelt oversving, der ikke er for stort. Derudover er det spændende at se, hvor stor en vinkel pendulstangen kan indsvinge sig fra, uden at systemet bliver ustabilt.



Figur 1.1: Linedanseropstillingen ved "Automation and Control" - DTU

Herfra vil det være en spændende udfordring at videreudvikle opstillingen frem mod en avanceret linedanserrobot, der kan køre på en ethjulet cykel, også kaldet en unicykel, hvor reguleringen af hjulet sørger for, at robotten ikke vælter forover eller bagover. Denne regulering lægger sig mere op ad problemet med det klassiske omvendte pendul, hvor reguleringen foregår ved pendulstangens bund.

Baseret på simuleringer af regulatordesignet ønskes regulatoren implementeret gennem en realtidscomputer med realtidssystemet RTAI-Linux (Real Time Application Inteface) med en interface connector, der kan læse spændinger fra måleenheder og sende spændinger til en motor. Formålet er at finde ud af, om det virkelige system kan reguleres ud fra den fundne model, og hvor tæt systemet opfører sig som det modellerede system.

Kapitel 2

Modellering af linedanseren

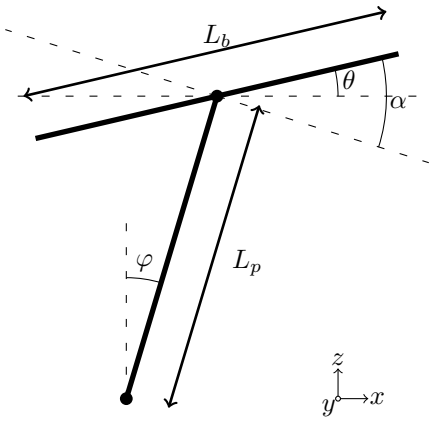
Linedanseropstillingen skal illustrere en linedanser, der ved hjælp af en balancestang skal forsøge at holde balancen på en line, som man ofte ser det i cirkus eller som de lidt mere vovede gadeartister, som det eksempelvis ses i figur 2.1. En linedanser, som balancerer med en balancestang, holder balancen ved at give sig selv et kraftmoment med hjælp fra stangen. Stangen har et inertimoment, der ved en vinkelacceleration giver et kraftmoment, som skal modvirke tyngdekraftens påvirkning af tyngdepunktet. Ændringen af vinklen skal styres sådan, at kraftmomentet opretholder linedanseren.

Figur 2.1: Philippe Petit balancerer i 1974 på en line fra toppen af det ene af tvillingetårnene fra World Trade Center til det andet

http://www.wineandbowties.com/wordpress/wp-content/uploads/2011/11/boldness-Philippe-Petit.jpg

2.1 Modelopstilling

Modellen af linedanseren kan opstilles som et todimensionelt system, som vist i figur 2.2. Den er modeleret som et omvendt pendul med en roterbar stang fastsat i toppen. Længderne er henholdsvis L_p og L_b. Vinklen af pendulet i forhold til dens opretstående tilstand er angivet som φ og vinklen af stangen i forhold til vandret er angivet som θ. Modellen vist i figuren, hvor også et koordinatsystem er indført med retninger x, y og z, hvor y-retningen peger indad i figuren.



Figur 2.2: Model af linedansersystemet

Vinklen θ for balancestangen kan umiddelbart være svær praktisk at bestemme. Derfor indføres vinklen $\alpha = \theta + \varphi$, som er en relativ vinkel i forhold til pendulet. Denne vinkel kan man f. eks. bestemme med et potentiometer, som kan måle vinklen i forhold til pendulets vinkel i stedet for en vinkel i forhold til vandret.

Dette systems stationære tilstand er når $\varphi = 0$. Systemet er ustabilt, så en hvilken som helst afvigelse fra dette, får pendulet til at vælte. Vinklen α er ikke så vigtig for det stillestående system, da den ikke påvirker balancepunktet ved stilstand. Da modellen skal illustrere en linedanser vil det være ønskeligt at kontrollere systemet, så dens stationære tilstand er $\varphi = 0$ og $\alpha = 0$. Det svarer til at linedanseren holder balancen med balancestangen i vandret position.

2.2 Det fysiske system

To tidligere projekter [1] og [2] har beskæftiget sig med linedansersystemet. Der viderearbejdes med opstillingen fra disse projekter. Se figur 1.1.

Opstillingen består af pendulet og balancestangen lavet i aluminium, et vægtlod i hver ende af balancestangen til at give et øget inertimoment til stangen, motor, tachometer, gearing og rem til styring af balancestangen samt potentiometer til måling af vinklerne φ og α. De fysiske størrelser, som længder og masser, kan findes i bilag A, som er et opslag over de symboler, der anvendes gennem rapporten.

2.3 Den teoretiske model for regulering

For at kunne regulere systemet, skal systemets dynamik kendes. Ud fra den opstillede model kan man beregne, hvorledes systemet tilnærmelsesvist opfører sig. Det er kraftmomentet, man ønsker at kontrollere, så der opstilles ligninger for de kraftmomenter, der indgår i systemet.

2.3.1 Beregning af inertimoment

Først bestemmes inertimomentet af balancestangen I_b. Denne kan antages at være inertimomentet for en tynd stang med en punktmasse i hver ende:

$$I_b = \underbrace{\frac{1}{12}M_bL_b^2}_{\text{balancestang}} + \underbrace{2 \cdot \frac{1}{4}M_lL_b^2}_{\text{lodder}} \tag{2.1}$$

M_b er massen af balancestangen, M_l er massen af et lod og L_b er længden af balancestangen. Ligningen forudsætter at pendulstangen har fri drejelighed om sin akse. Udtrykket kan forkortes til

$$I_b = (\frac{1}{2}M_l + \frac{1}{12}M_b)L_b^2 \tag{2.2}$$

For selve pendulstangen, I_p, er inertimomentet

$$I_p = \frac{1}{3}M_pL_p^2 \tag{2.3}$$

, hvor M_p er massen af pendulstangen.

Inertimomentet for hele pendulet, I_a, er lidt vanskeligere at opskrive. Den er en sammensætning af inertimomenter for pendulet selv, for balancestangen monteret i toppen, for motoren monteret på pendulet, for tandhjulet i toppen monteret på balancestangen og for tandhjulet i bunden til gearing fra motoren. Dette opskrives:

$$\begin{aligned} I_a = & \overbrace{\frac{1}{3}M_pL_p^2}^{I_p} + \overbrace{(\frac{1}{2}M_l + \frac{1}{12}M_b)L_b^2}^{I_b} \\ & + \underbrace{M_mL_m^2}_{\text{motor}} + \underbrace{M_aL_p^2}_{\text{top tandhjul}} + \underbrace{M_{a_1}L_{a_1}^2}_{\text{gearingstandhjul}} \end{aligned} \tag{2.4}$$

L_m er længden af pendulstangen, M_m er motorens rotormasse, L_m er positionslængden af motoren på pendulet, M_a er massen af tandhjulet mm. i toppen af balancestangen, M_{a_1} er massen af gearingstandhjulet og L_{a_1} er dets placering på pendulstangen. Dette forkortes til

$$I_a = (2M_l + M_b + \frac{1}{3}M_p + M_a)L_p^2 + M_mL_m^2 + M_{a_1}L_{a_1}^2 \tag{2.5}$$

2.3.2 Beregning af tyngdepunkt

Til opstilling af modellen er det også nødvendigt at inddrage pendulets tyngdepunkt, L_T, som befinder sig et sted langs pendulet. Denne afhænger af alle de dele af systemet, hvor massen har en betydende størrelse. Når pendulet står skævt trækker tyngdekraften i pendulet gennem tyngdepunktet, sådan at pendulet vælter. For at stabilisere systemet ønskes tyngdepunktet placeret i midten ud for omdrejningsaksen, så tyngdekraften trækker parallelt med pendulets retning,

og derfor ingen virkning har. Tyngdepunktet udregnes for de enkelte elementer af linedanseropstillingen i forhold til den samlede masse, M, som produktet af massen og positionen af massemidtpunktet med pendulets omdrejningspunkt som reference:

$$L_T = \frac{\overbrace{\frac{1}{2}M_pL_p}^{\text{pendulstang}} + \overbrace{M_bL_p}^{\text{balancestang}} + \overbrace{M_mL_m}^{\text{motor}} + \overbrace{M_aL_p}^{\text{top tandhjul}} + \overbrace{M_{a_1}L_{a_1}}^{\text{gearingstandhjul}}}{M} \tag{2.6}$$

Dette forkortes til

$$L_T = \frac{\frac{1}{2}M_p + \frac{L_m}{L_p}M_m + M_b + 2M_l + M_a + \frac{L_{a_1}}{L_p}M_{a_1}}{M}L_p \tag{2.7}$$

2.3.3 Beregning af kraftmoment

Til beregning af kraftmomenter anvendes vektornotation, hvor enhedsvektorerne $\mathbf{e_x}$, $\mathbf{e_y}$ og $\mathbf{e_z}$ benyttes som retningerne i henhold til figur 2.2. For balancestangen kan kraftmomentet $\boldsymbol{\tau_b}$ bestemmes ved

$$\boldsymbol{\tau_b} = \tau_b\,\mathbf{e_y} = -I_b\,\ddot{\theta}(t)\,\mathbf{e_y} \tag{2.8}$$

, hvor I_b er inertimomentet for balancestangen, ligning (2.2), og $\ddot{\theta}(t)$ er balancestangens vinkelacceleration. Det negative fortegn skyldes, at kraftmomentet følger højrehåndskonventionen, hvilket for figur 2.2 svarer til at kraftmomentet peger ud af figuren, hvorimod y-aksens retning peger modsat.

Pendulets kraftmoment $\boldsymbol{\tau_p}$ afhænger af pendulets vinkelacceleration $\ddot{\varphi}(t)$ og er

$$\boldsymbol{\tau_p} = \tau_p\,\mathbf{e_y} = I_a\,\ddot{\varphi}(t)\,\mathbf{e_y} \tag{2.9}$$

, hvor I_a er hele pendulets inertimoment, ligning (2.5).

Pendulstangen påvirkes af to kraftmomenter. Det ene er kraftmomentet for balancestangens påvirkning $\boldsymbol{\tau_{pb}}$, som er modsatrettet af $\boldsymbol{\tau_b}$

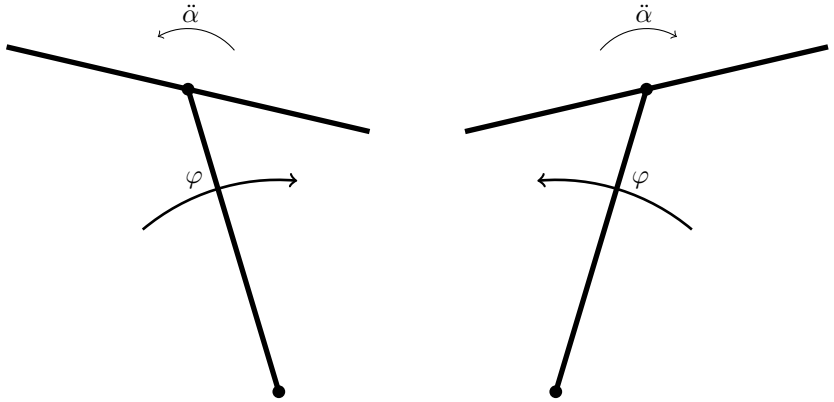
$$\boldsymbol{\tau_{pb}} = \tau_{pb}\,\mathbf{e_y} = -\boldsymbol{\tau_b} = I_b\,\ddot{\theta}(t)\,\mathbf{e_y} \tag{2.10}$$

Den anden påvirkning er tyngdekraften, der giver et kraftmoment $\boldsymbol{\tau_{pg}}$. Det kan udregnes ved

$$\boldsymbol{\tau_{pg}} = (L_T \sin(\varphi(t))\,\mathbf{e_x}) \times (-Mg\,\mathbf{e_z})$$

Dette giver

$$\boldsymbol{\tau_{pg}} = \tau_{pg}\,\mathbf{e_y} = MgL_T \sin(\varphi(t))\,\mathbf{e_y} \tag{2.11}$$



(a) Positiv vinkelacceleration (b) Negativ vinkelacceleration

Figur 2.3: Balancestangens vinkelacceleration $\ddot{\alpha}$'s påvirkning af pendulstangen

Her er g tyngdeaccelerationen og L_T er tyngdepunktet, ligning (2.7).

Der gælder at kraftmomentet $\boldsymbol{\tau_p}$ afhænger af påvirkningen fra kraftmomenterne $\boldsymbol{\tau_{pb}}$ og $\boldsymbol{\tau_{pg}}$, så

$$\boldsymbol{\tau_p} = \boldsymbol{\tau_{pb}} + \boldsymbol{\tau_{pg}} \tag{2.12}$$

Det vil sige, at

$$\tau_p \, \mathbf{e_y} = \tau_{pb} \, \mathbf{e_y} + \tau_{pg} \, \mathbf{e_y} \tag{2.13}$$

Kraftmomentet består udelukkende af en komposant i y-aksens retning. Hvis man ser bort fra den indflydelse, tyngdekraften har, ses det, at en positiv vinkelacceleration for θ vil give anledning til, at pendulet drejer med urets retning om y-aksen, hvis man sammenholder dette med figur 2.2. Det modsatte gør sig gældende for en negativ vinkelacceleration. Dette er illustreret i figur 2.3. Tyngdekraften påvirker dog også pendulets kraftmoment. Denne kan både påvirke positivt eller negativt afhængigt af pendulvinklen. I dette tilfælde, hvor pendulstangen ønskes lodret, vil tyngdekraften dog altid virke negativt i forhold til den ønskede påvirkning, da θ altid vil accelerere for at modvirke tyngdekraften og skubbe pendulet på plads.

Af ligning (2.13) får man følgende sammenhæng:

$$\tau_p = \tau_{pb} + \tau_{pg} \tag{2.14}$$

I det videre udregningsarbejde benyttes dette, da det ikke længere vil være nødvendigt at benytte vektornotation, idet kraftmomentet kun bestemmes ved én komposant.

2.3.4 Den samlede model

Ved at indsætte de forskellige kraftmomenter fra ligning (2.9), (2.10) og (2.11) i ligning (2.14) kan modellen beskrives ved følgende ligning:

$$I_a \, \ddot{\varphi}(t) = I_b \, \ddot{\theta}(t) + MgL_T \sin(\varphi(t)) \tag{2.15}$$

I modellen arbejdes der inden for små værdier af φ, hvorved man kan antage, at $\sin(\varphi(t)) \simeq \varphi(t)$. Den lineariserede ligning kan da opskrives ved

$$I_a \, \ddot{\varphi}(t) = I_b \, \ddot{\theta}(t) + MgL_T \, \varphi(t) \tag{2.16}$$

Da det i praksis er besværligt at måle vinklen θ, anvendes vinklen $\alpha = \theta + \varphi$, som beskrevet tidligere. Det vil sige, at θ i ligning (2.16) erstattes med $\theta = \alpha - \varphi$:

$$I_a \, \ddot{\varphi}(t) = I_b \, (\ddot{\alpha}(t) - \ddot{\varphi}(t)) + MgL_T \, \varphi(t) \tag{2.17}$$

Isoleres de afhængige af φ på venstre side af ligningen, fås udtrykket

$$(I_a + I_b) \, \ddot{\varphi}(t) - MgL_T \, \varphi(t) = I_b \, \ddot{\alpha}(t) \tag{2.18}$$

2.3.5 Laplacetransformering af modellen

For lettest at kunne beskrive modellen til design af regulatoren, overføres ligning (2.18) til frekvensdomænet med Laplacetransformering:

$$(I_a + I_b) \, s^2 \varphi(s) - MgL_T \, \varphi(s) = I_b \, \ddot{\alpha}(s)$$

Isoleres φ får man

$$\varphi(s) = \frac{I_b}{(I_a + I_b) \, s^2 - MgL_T} \, \ddot{\alpha}(s)$$

Dette omskrives til

$$\varphi(s) = \frac{\frac{I_b}{(I_a + I_b)}}{s^2 - \frac{1}{I_a + I_b} MgL_T} \, \ddot{\alpha}(s)$$

Hvis man omskriver dette til

$$\varphi(s) = \frac{1 - \frac{I_a}{(I_a + I_b)}}{s^2 - \frac{I_a}{I_a + I_b} \frac{MgL_T}{I_a}} \, \ddot{\alpha}(s)$$

kan man indføre $c^2 = \frac{I_a}{(I_a + I_b)}$ og $p = \sqrt{\frac{MgL_T}{I_a}}$, sådan at ligningen kan skrives som

$$\varphi(s) = \frac{1 - c^2}{s^2 - (cp)^2} \, \ddot{\alpha}(s) \tag{2.19}$$

Dette vil være modellen benyttet til det videre arbejde med beregninger af blokmodel og design af regulator for linedansersystemet.

Kapitel 3

Regulatordesign til linedanseren

Med teorien til modellen for linedansersystemet på plads fra kapitel 2 er det muligt ved opstilling af modellen i et blokdiagram gennem MATLAB/Simulink at simulere modellen og designe regulatoren.

3.1 Blokdiagram af modellen

Bloksystemet for modellen beskrevet ved ligning (2.19) opstilles som en state-space model ud fra

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{3.1}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \tag{3.2}$$

, hvor state vektoren $\mathbf{x} = [\varphi \quad \dot{\varphi} \quad \alpha \quad \dot{\alpha}]^T$ og inputvektoren $\mathbf{u}$ er et inputsignal, der skal ændre vinkelaccelerationen $\ddot{\alpha}$. Systemmatricen $\mathbf{A}$ og inputmatricen $\mathbf{B}$, som er tidsinvariante, opskrives ud fra ligning (2.19):

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ (cp)^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.3}$$

$$\mathbf{B} = \begin{bmatrix} 0 \\ 1 - c^2 \\ 0 \\ 1 \end{bmatrix} \tag{3.4}$$

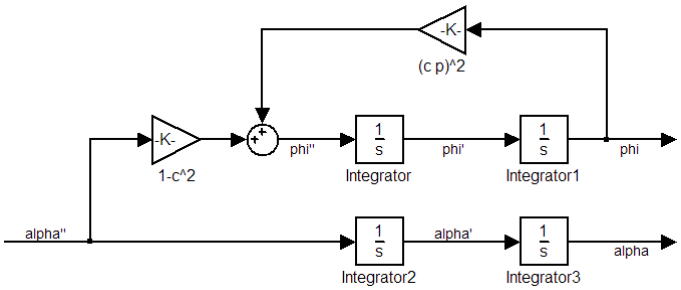
Til at starte med er $\mathbf{C}$ og $\mathbf{D}$ lig med 0. Ved at indsætte værdierne fra bilag A, fås

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 14,26 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.5}$$

$$\mathbf{B} = \begin{bmatrix} 0 \\ 0,3316 \\ 0 \\ 1 \end{bmatrix} \tag{3.6}$$

Denne model har polerne $\mathbf{E} = [3,777 \quad -3,777 \quad 0 \quad 0]$ fundet ved MATLAB's funktion `eig()` til at finde egenværdier. Dette viser tydeligt, at systemet er ustabilt ved dens positive reele pol.

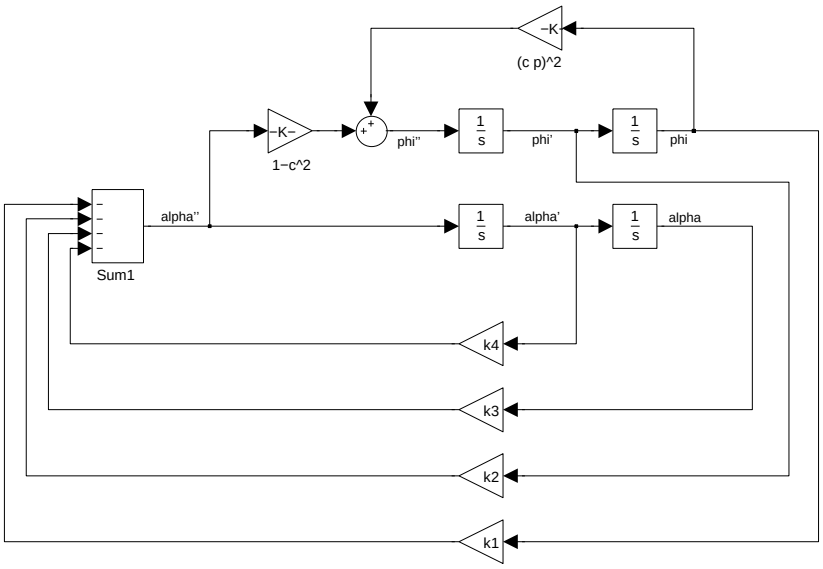
Af ligning (3.3) og (3.4) kan modellen opstilles i et blokdiagram, som er vist i figur 3.1.



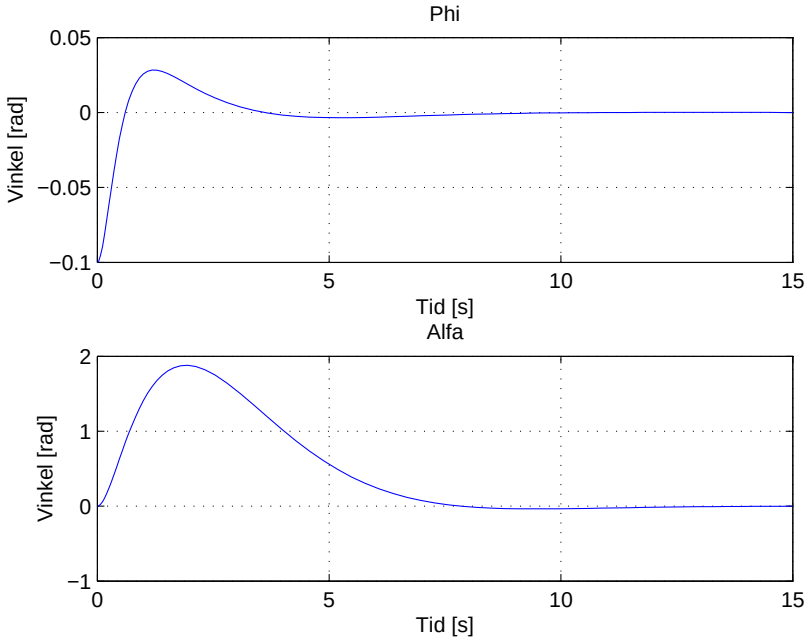
Figur 3.1: Blokdiagram af model

Først opstilles modellen med en ideel servomekanisme, der har tilbagekobling fra de forskellige tilstande, som er vist i figur 3.2. I princippet vil denne regulere systemet mod equilibrium, hvor alle tilstande er 0. Tilbagekoblingskonstanterne k_1, k_2, k_3 og k_4 , samlet $\mathbf{K}$, bestemmes vha. LQR-design (Linear Quadratic Regulator), der bestemmer konstanterne for den optimale regulering ud fra formlen

$$J = \int \dot{\mathbf{x}}\,\mathbf{Q}\,\mathbf{x} + \dot{u}\,\mathbf{R}\,u + 2\,\dot{\mathbf{x}}\,\mathbf{N}\,\mathbf{u}\,dt \tag{3.7}$$



Figur 3.2: Blokdiagram af model med ideel servomekanisme



Figur 3.3: Simulering af det ideelle servosystem

Der benyttes den indbyggede funktion `lqr()` i MATLAB til dette med vægtmatricerne $\mathbf{Q}$, $\mathbf{R}$ og $\mathbf{N}$ valgt til passende størrelser. Som udgangspunkt er valgt

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{R} = 5, \mathbf{N} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.8}$$

, hvilket resulterer i tilbagekoblingskonstanterne og egenværdierne

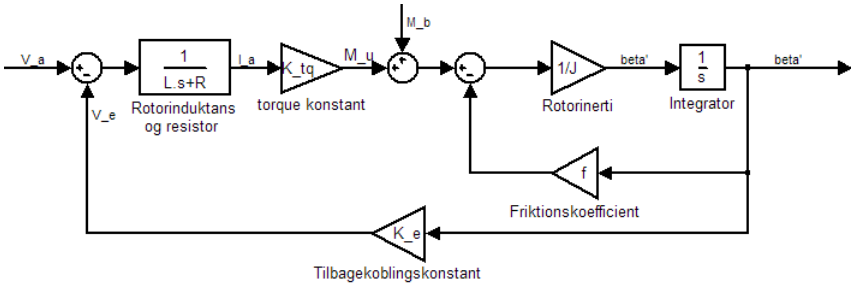
$$\mathbf{K} = \begin{bmatrix} 112,56 & 29,81 & -0,4472 & -1,283 \end{bmatrix} \tag{3.9}$$

$$\mathbf{E} = \begin{bmatrix} -3,849 & -3,705 & -0,523 + 0,417i & -0,523 - 0,417i \end{bmatrix} \tag{3.10}$$

Her er egenværdiernes realdele alle negative, hvilket betyder, at systemet nu er stabilt modsat systemet uden regulering.

En simulering i MATLAB ved disse tilbagekoblingskonstanter er vist for vinklerne φ(phi) og α(alfa) i figur 3.3, hvor der er givet en startvinkel $\varphi_0 = -0,1$ radianer. Dette giver en ide om, hvordan pendulstangen og balancestangen indsvinger sig. Balancestangen er knap så vigtig udmiddelbart, når bare man har styr over, at den ikke har for store udsving i reguleringen. Fremover vises normalt kun indsvingningen for pendulstangen.

Da systemet ikke er et ideelt servosystem, men derimod indeholder et motorsystem med modstand og gearing, skal modellen revideres til at indeholde denne. En blokmodel for en motor er vist i figur 3.4.



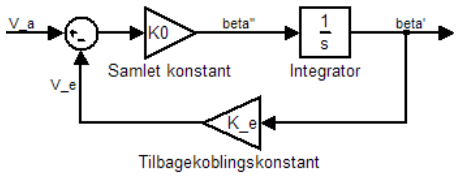
Figur 3.4: Blokdiagram af motor

Det antages i modellen, at der ikke er nogen rotorinduktans ($L = 0$), at motorforstyrrelsen $M_b = 0$ og friktionen $f = 0$. Motormodellen kan derved simplificeres til blokdiagrammet vist i figur 3.5. Motortilbagekoblingskonstanten K_e har her samme værdi som torquekonstanten K_{tq}, og motorens rotorinerti, J, er så lille i forhold til det samlede inertimoment, at den er uden betydning. Det samlede inertimoment på motoraksen kan herved vises at være

$$I_t = \frac{I_a\, I_b}{n^2((1 + \frac{1}{n})I_b + I_a)} \tag{3.11}$$

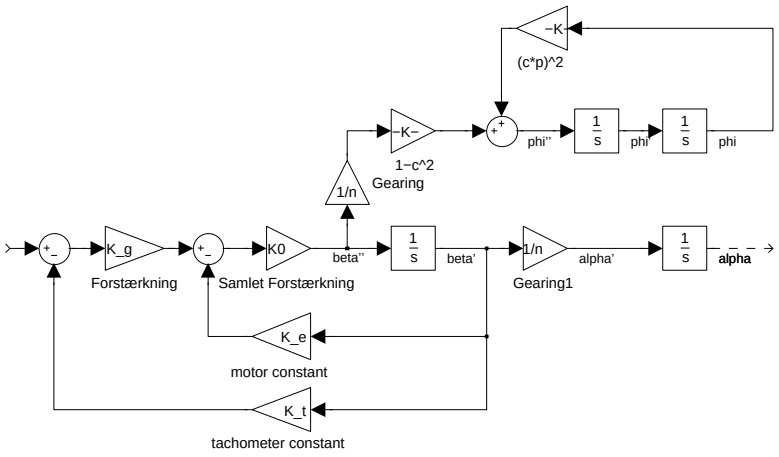
Heraf giver den samlede konstant $K0$ i den simple motor

$$K0 = \frac{K_e}{R\, I_t} \tag{3.12}$$



Figur 3.5: Simplificeret blokdiagram af motor

Som motor benyttes en hastighedsservo, dvs. vinkehastigheden for motorakslen er negativt tilbagekoblet til indgangen på motoren via et tachometer med forstærkningen K_t. Figur 3.6 viser motoren indsat i modellen fra figur 2.2.

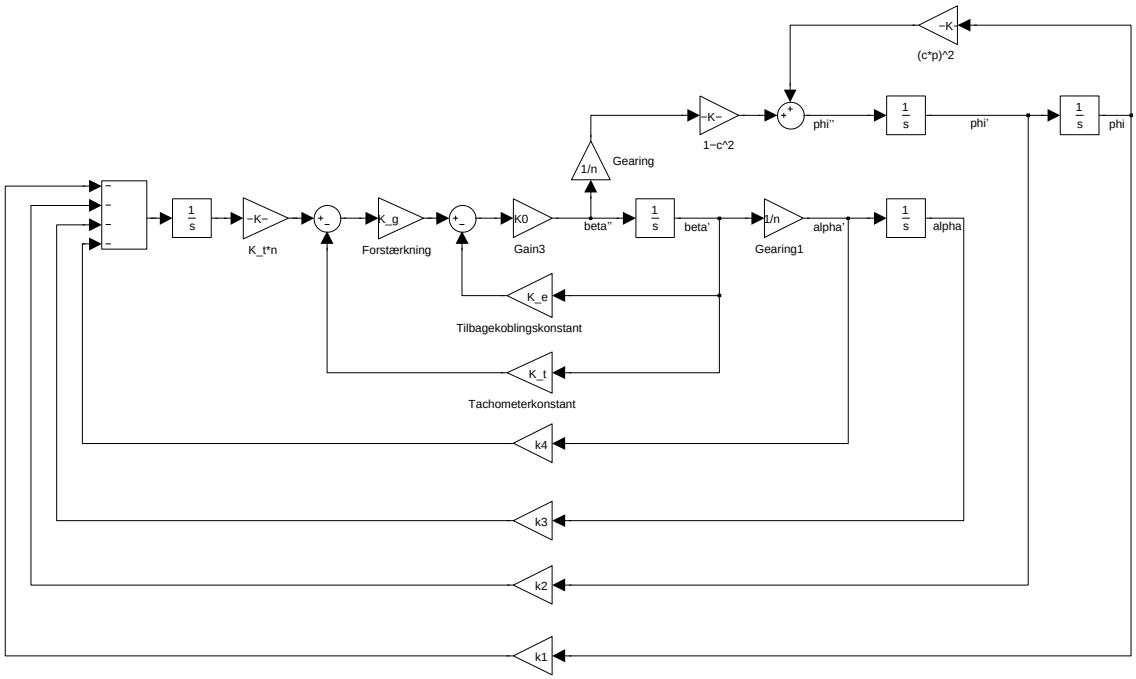


Figur 3.6: Model af linedanser med motorsystem

Den ideelle model for linedanseren er for en accelerationsservo. I modellen for linedanseren med motor indsættes derfor en integrator før motorens indgang, så indgangen på motoren er en hastighed og ikke en acceleration, som returneres fra LQR-tilbagekoblingen. Dette er vist i figur 3.7, hvor også gearing og forstærkning er inkluderet.

Dette giver en ny **A**- og **B**-matrice for modellen svarende til

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ (cp)^2 & 0 & 0 & \frac{(-K_e - K_t K_g) K0 (1-c^2)}{n} \\ 0 & 0 & 0 & \frac{1}{n} \\ 0 & 0 & 0 & (-K_e - K_t K_g) K0 \end{bmatrix} \tag{3.13}$$



Figur 3.7: Det samlede blokdiagram med motor og tilbagekobling

$$\mathbf{B} = \begin{bmatrix} 0 \\ \frac{K_g K0(1-c^2)}{n} \\ 0 \\ K_g K0 \end{bmatrix} \tag{3.14}$$

Med indsatte værdier fås

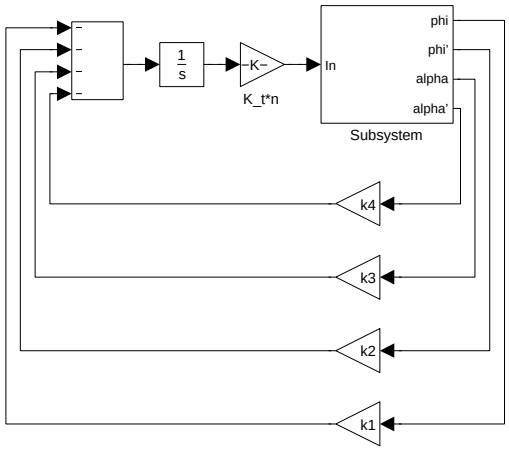
$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 14,2617 & 0 & 0 & -0,1676 \\ 0 & 0 & 0 & 0,04 \\ 0 & 0 & 0 & -12,6381 \end{bmatrix} \tag{3.15}$$

$$\mathbf{B} = \begin{bmatrix} 0 \\ 6,7294 \\ 0 \\ 507,39 \end{bmatrix} \tag{3.16}$$

Ved simulering af modellen med en motor i stedet for en ideel servo kan de samme tilbagekoblingskonstanter fundet for den ideelle servomekanisme benyttes. En LQR af det nye system vil ikke give væsentlige ændringer af konstanterne. Der skal dog tages højde for, at der med en hastighedsservo gives en reference i forhold til motorakslens hastighed $\dot{\beta}$ og ikke $\dot{\alpha}$. Dette gør, at styresignalet til

motoren skal multipliceres med $K_t \cdot n$ for at give en rigtig reference til motoren i forhold til $\dot{\alpha}$. Dette er vist i figur 3.7.

Modellen er efterhånden blevet stor, så blokdiagrammet i figur 3.6 samles til én blok. Herved ser blokdiagrammet fra figur 3.7 ud som i figur 3.8 med tilbagekoblingerne. En simulering af modellen med motoren indsat er vist i figur 3.9, hvor også vinklen α er vist til sammenligning med figur 3.3. Her oversvinger φ lidt mere og har en lidt længere indsvingningstid.



Figur 3.8: Simplificeret blokdiagram

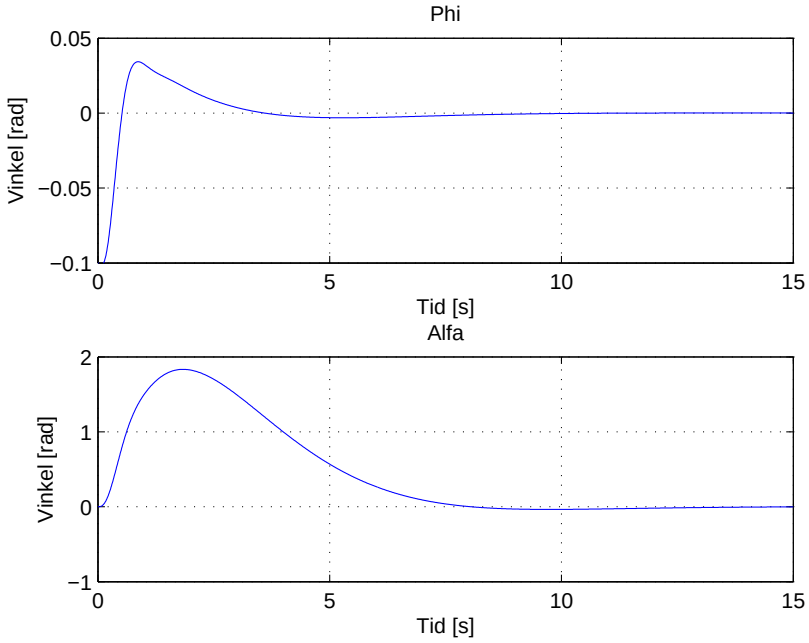
Ved tilføjelsen af integratoren i modellen, kan der opstilles en ny tilstand u, og derved omskrives **A**- og **B**-matricerne endnu engang, så de passer med tilstandsvektoren $\mathbf{x} = [\varphi \quad \dot{\varphi} \quad \alpha \quad \dot{\beta} \quad u]^T$, til

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ (cp)^2 & 0 & 0 & \frac{(-K_e - K_t K_g) K0 (1-c^2)}{n} & \frac{K_g K0 (1-c^2)}{n} \\ 0 & 0 & 0 & \frac{1}{n} & 0 \\ 0 & 0 & 0 & (-K_e - K_t K_g) K0 & K_g K0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.17}$$

$$\mathbf{B} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{3.18}$$

Med indsatte værdier fås

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 14,2617 & 0 & 0 & -0,1676 & 6,7294 \\ 0 & 0 & 0 & 0.04 & 0 \\ 0 & 0 & 0 & -12,6381 & 507,3963 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.19}$$



Figur 3.9: Simulering af det ikke ideelle system

$$\mathbf{B} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{3.20}$$

Det bemærkes, at tilstandsvektoren indeholder $\dot{\beta}$ og ikke $\dot{\alpha}$, som den oprindelige tilstandsvektor gjorde. Ved tilføjelsen af en ekstra tilstand er et nyt LQR-design nødvendigt, da systemet nu får brug for en ekstra tilbagekobling fra den nye tilstand. Ved et nyt LQR-design, hvor $\mathbf{Q}$ nu er en 5×5 diagonalmatrix, $\mathbf{R} = 400$ og $\mathbf{N}$ er en 5-dimensionel vektor indeholdende nuller, fås

$$\mathbf{K} = \begin{bmatrix} 107,10 & 28,360 & -0,0500 & -0,2879 & 9,462 \end{bmatrix} \tag{3.21}$$

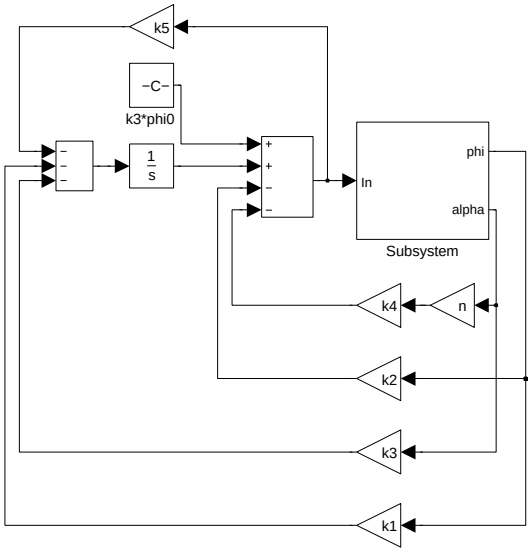
$$\mathbf{E} = \begin{bmatrix} -12,47 & -3,793 & -3,760 & -2,034 & -0,040 \end{bmatrix} \tag{3.22}$$

Denne $\mathbf{R}$ er valgt for at komme nogenlunde i nærheden af værdierne og simuleringen fra tidligere til at sammenligne med.

Med 5 tilstande skal der i modellen indsættes en tilbagekobling yderligere fra integratoren. Da LQR-metoden nu er baseret på modellen med motorsystemet inkluderet, er der taget højde for motorens dynamik, dvs. at signalet efter LQR-tilbagekoblingen ikke skal multipliceres med $K_t \cdot n$. I stedet skal tilbagekoblingen fra $\dot{\alpha}$ multipliceres med n, for at tilbagekoblingen svarer til $\dot{\beta}$, som nu er fjerde

tilstand i tilstandsvektoren i forhold til den ideelle model. Dette er for at kompensere for den gearing, som vinkelhastigheden ændres af.

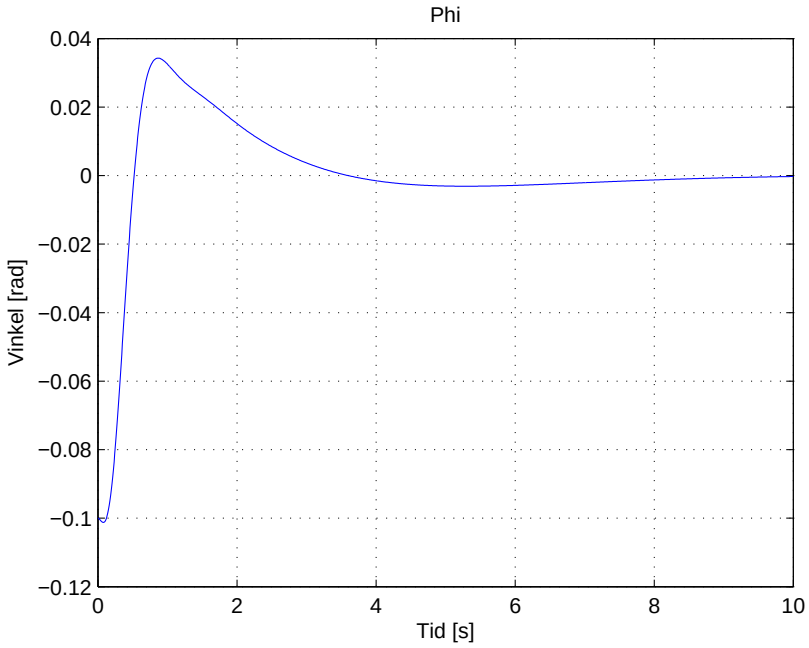
Da der indgår en integrator efter LQR-tilbagekoblingen, kan man undlade at tilbagekoble vinkelhastighederne $\dot{\varphi}$ og $\dot{\theta}$ og i stedet tilbagekoble φ og θ til efter integratoren og før dens tilbagekobling, da disse allerede er blevet integreret fra vinkelhastighederne. Dette er vist i figur 3.10, hvor også den femte tilbagekobling fra den forreste integrator er inddraget. Man kan herved nøjes med at måle vinklerne φ og θ. Når man gør dette, er man nødt til at sørge for, at startvinklerne φ_0 og θ_0 begge er 0, ellers vil en afvigelse fra dette give en tilbagekobling for hastighederne, der ikke er 0 til at starte med. Man kan undgå dette ved at lægge startvinklerne multipliceret med deres henholdsvise tilbagekoblingskonstant, dvs. lægge $k_2 \cdot \varphi_0$ og $k_4 \cdot \alpha_0$, til signalet mellem integratoren og motorindgangen. Ved at glemme dette, kan systemet blive ustabilt, hvis regulatoren tror, at vinkelhastigheden er så stor, at tilbagekoblingen reagerer voldsomt for at forhindre dette, så motoren derved giver et så stort udslag på vinklen, at systemet ikke længere kan holde sig oppe.



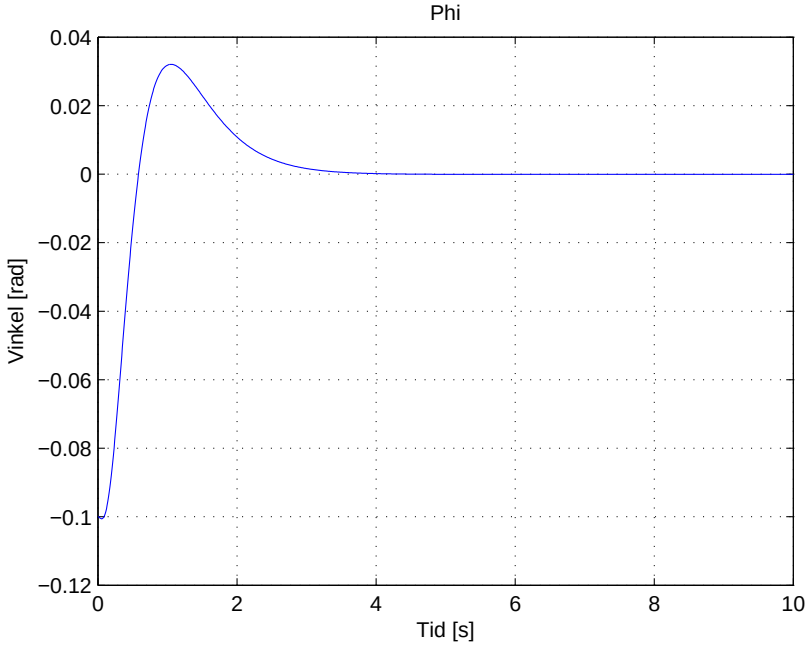
Figur 3.10: Model med ekstra tilbagekobling

En simulering af dette både for 4 tilstande og 5 tilstande med samme startbetingelser som tidligere er vist i figur 3.11 og 3.12.

Af simuleringen i figur 3.11 og 3.12 ser systemet med 5 tilbagekoblinger ud til at have et roligere indsvingningsforløb og en mindre indsvingningstid end systemet med 4 tilstande. Der kigges nærmere på dette i de kommende kapitler, hvor den diskrete regulator designes og implementeres.



Figur 3.11: Simulering med 4 tilstande



Figur 3.12: Simulering med 5 tilstande

3.2 Implementering med accelerometer og gyroskop

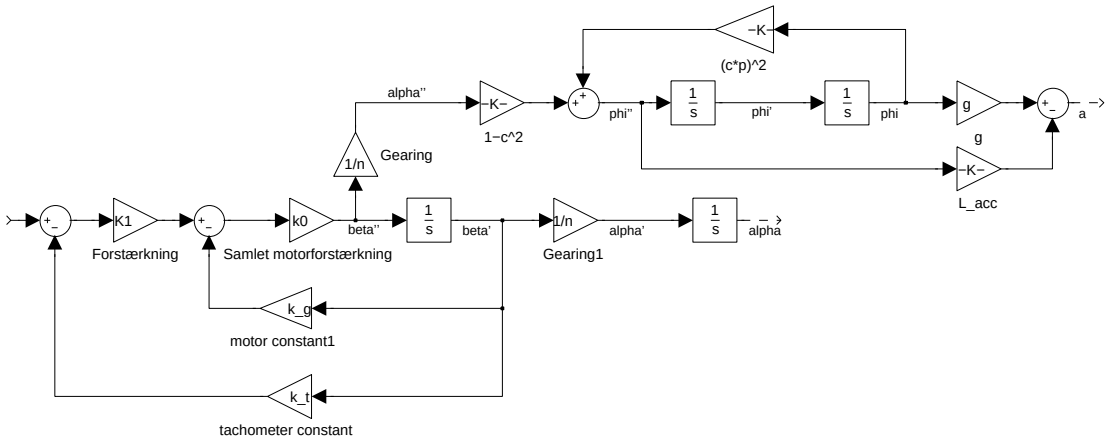
Når "linedanseren" er monteret på sit stativ er det forholdsvist let at bestemme pendulvinklen φ med et potentiometer. Men da systemet ikke skal forblive fastmonteret, er det nødvendigt at bestemme vinklen og for den sags skyld vinkelhastigheden med andre metoder. Her er det relevant at benytte et gyroskop og et accelerometer. Med gyroskopet kan vinkelhastigheden direkte måles. Det er lidt mere besværligt med accelerometeret at bestemme vinkelpositionen, da accelerometeret måler acceleration. Accelerationen af pendulet består af en acceleration fra tyngdeaccelerationen og pendulets tangentielle acceleration givet ud fra vinkelaccelerationen. Dette giver ligningen

$$a = g\sin(\varphi) - L_a\ddot{\varphi} \tag{3.23}$$

, hvor L_a er afstanden fra accelerometeret til pendulets omdrejningspunkt, og g er tyngdeaccelerationen. Dette kan simplificeres ved at antage, at $\sin(\varphi) \simeq \varphi$ for små værdier af φ, så udtrykket bliver

$$a = g\varphi - L_a\ddot{\varphi} \tag{3.24}$$

Dette indsættes i blokdiagrammet for modellen, og den oprindelige model fra figur 3.1 kommer således til at se ud som i figur 3.13.



Figur 3.13: Modellen implementeret som acceleration

Der tages udgangspunkt i den oprindelige model med 4 tilstande. Systemmatricen $\mathbf{A}$ og $\mathbf{B}$ (ligning (3.13) og (3.14)) ændres ikke ud fra figur 3.13, men der kommer herved til at være en $\mathbf{C}$- og $\mathbf{D}$-matrice til systemet, der knytter sig til vektoren $\mathbf{y} = [a \quad \dot{\varphi} \quad \theta]^T$ givet ved ligning (3.2). Disse bliver

$$\mathbf{C} = \begin{bmatrix} g - L_{acc}(cp)^2 & 0 & 0 & \frac{(K_t K_g + K_e) K0 (1-c^2) L_{acc}}{n} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{n} & 0 \end{bmatrix} \tag{3.25}$$

$$\mathbf{D} = \begin{bmatrix} \frac{-K_g K0 (1-c^2) L_{acc}}{n} \\ 0 \\ 0 \end{bmatrix} \tag{3.26}$$

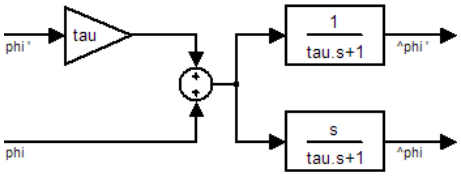
Indsættes værdierne, fås

$$\mathbf{C} = \begin{bmatrix} 8,394 & 0 & 0 & 0,0168 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0,04 & 0 \end{bmatrix} \tag{3.27}$$

$$\mathbf{D} = \begin{bmatrix} -0,673 \\ 0 \\ 0 \end{bmatrix} \tag{3.28}$$

Ud fra accelerometerets målinger kan vinklen φ bestemmes. Det kan antages, at accelerometeret er placeret så tæt på omdrejningsaksen, at den tangentielle acceleration er forsvindende lille. Derved påvirkes accelerometeret kun af tyngdekraften. Tyngdekraftens påvirkning på pendulet er $a = g \sin(\varphi) \simeq g\,\varphi$ for små vinkler. Dvs. vinklen direkte kan bestemmes som $\varphi = \frac{a}{g}$.

For at opnå pæne måleresultater fra accelerometeret og gyroskopet uden for meget støj, filtreres deres signaler. Accelerometeret er lavfrekvent og gyroskopet er højfrekvent. En måde at filtrere signalerne på kan ses ud fra blokdiagrammet i figur 3.14. Dette filter påsættes modellen med en passende værdi af tidskonstanten τ. τ vælges alt efter, hvor knækfrekvensen for begge filtre ønskes. Hvis τ bliver for stor, bliver det filtrerede signal fra accelerometeret for langsomt til at reagere. Samtidig skal den ikke vælges for lille, da støjen fra accelerometeret får stor betydning for det filtrede vinkelhastighedssignal. Det har vist sig passende at benytte $\tau = 0.5$ sekunder.

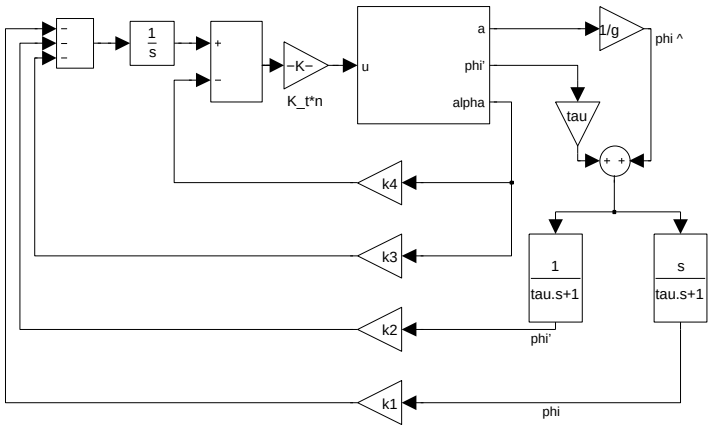


Figur 3.14: Metode til filtrering af signaler

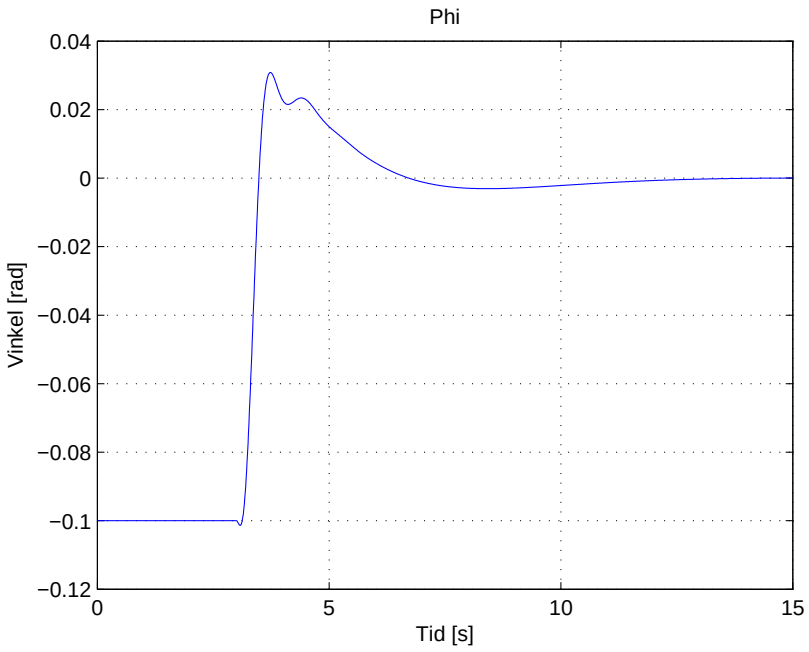
Med dette ser blokmodellen for 4 tilstande ud som i figur 3.15 og ellers tilsvarende som fra figur 3.10 for 5 tilstande.

En simulering både for 4 tilstande og for 5 tilstande med tilbagekoblingsmatricerne henholdsvis ligning (3.9) og (3.21) er vist i figur 3.16 og 3.17. Da vinklen φ

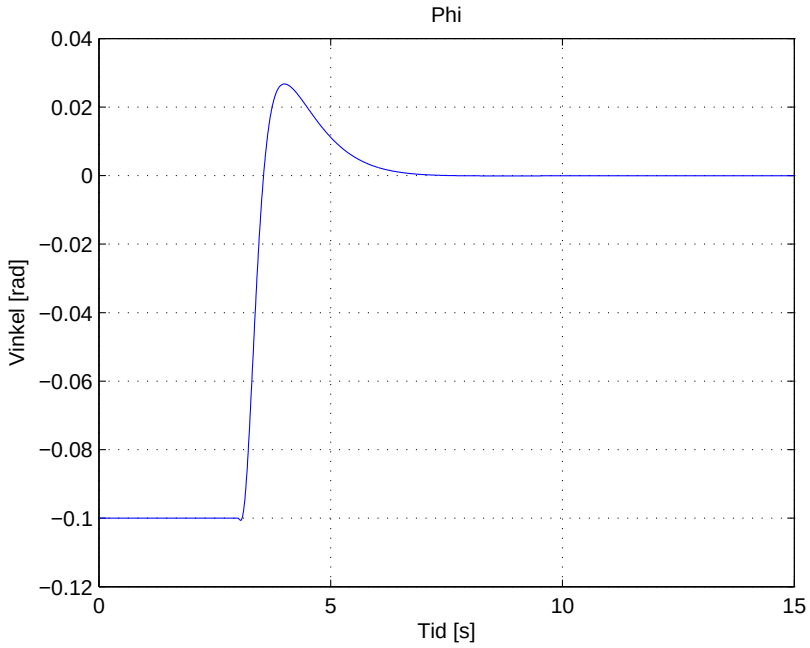
er lavpasfiltreret, tager det tid, inden denne opnår værdien φ_0. Efter tiden $3 \cdot \tau$ dvs. 1.5 sekunder, har filteret nogenlunde opnået sin startværdi. I simuleringen ventes der derfor noget tid, inden systemet starter. Her er valgt 3 sekunder.



Figur 3.15: Model med filter

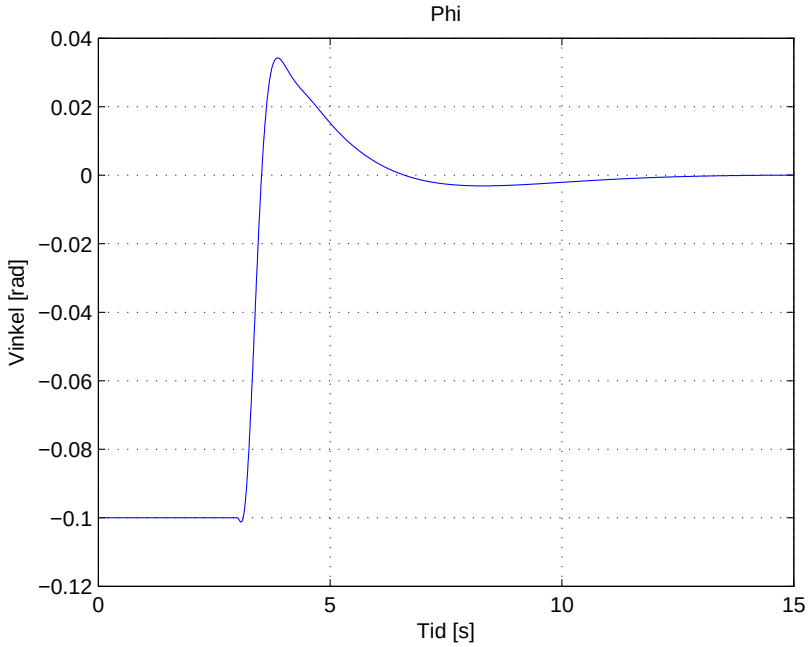


Figur 3.16: Simulering med accelerometer for 4 tilstande

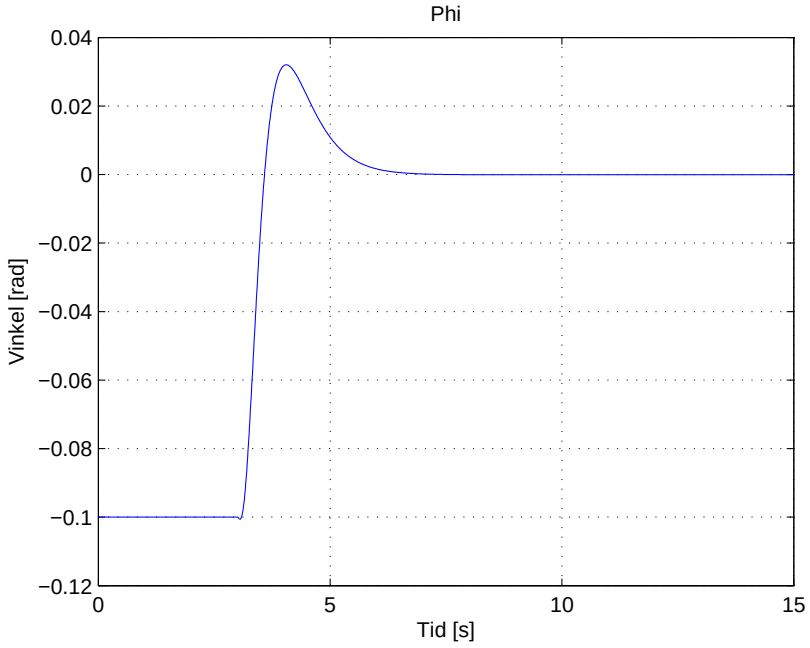


Figur 3.17: Simulering med accelerometer for 5 tilstande

Også her har systemet med 5 tilstande det bedste indsvingningsforløb. Systemet med 4 tilstande oplever nogle problemer i indsvingningen, ved at der kommer svingninger på indsvingningen. Dette skyldes, at accelerometeret ikke er fastmonteret præcist om omdrejningaksen for pendulet, men med en lille afstand derfra. Dette gør, at antagelsen om at accelerometeret kun påvirkes af tyngdekraften er upræcis, da den tangentielle kraft får en lille indflydelse, som vokser jo længere væk fra pendulets omdrejningsakse accelerometeret placeres. Dette gør, at systemet oplever flere og flere oscilleringer ved indsvingning. Dette problem synes modellen med 5 tilstande ikke at lide af, ved at regulatoren med en ekstra tilstand har bedre kendskab til systemet. Dette taler for at anvende en regulator med 5 tilbagekoblinger frem for 4 tilbagekoblinger. I de øvrige kapitler antages, at placeringen af accelerometeret er $L_{acc} = 0$ for bedre at kunne sammenligne de to modeller. Herved ser indsvingningen ud som i figur 3.18 og 3.19.



Figur 3.18: Simulering med accelerometer for 4 tilstande og $L_{acc} = 0$



Figur 3.19: Simulering med accelerometer for 5 tilstande og $L_{acc} = 0$

Kapitel 4

Det diskrete regulatordesign

Med regulatoren på plads i kontinuert tid fra forrige kapitel, kigges der nu på modellen i diskret tid. Da regulatoren skal implementeres som en computerbaseret løsning, er det nødvendigt at designe regulatoren i diskret tid. I diskret tid er modellen af linedanseren ligesom i kontinuert tid, men computeren ser den kun for diskrete tider. Denne sampling sker med en fast tidsforskel givet ved sampling-tiden T_s.

4.1 Den diskrete model

Den diskrete model af systemet, som computeren ser, kan bestemmes ved at gå fra Laplace-domænet til z-domænet, hvor man kigger på tidsforsinkelser. I MATLAB kan man hurtigt og nemt benytte funktionen `c2d()`, som omregner et state-space system fra kontinuert tid til diskret tid for en given sampling tid.

I diskret tid er state-space modellen næsten magen til den kontinuerte. Med de samme tilstandvektorer som for kontinuert tid kan den diskrete state-space model opskrives ved

$$\mathbf{x}(k+1) = \mathbf{F}\mathbf{x}(k) + \mathbf{G}\mathbf{u}(k) \tag{4.1}$$

$$\mathbf{y}(k) = \mathbf{C}\mathbf{x}(k) + \mathbf{D}(k)\mathbf{u}(k) \tag{4.2}$$

Dette er under antagelse af, at det er et tidsinvariant system, hvilket systemet kan siges at være. Fra de kontinuerte systemmatricer $\mathbf{A}$ og $\mathbf{B}$ fra ligning (3.13) og (3.14) med 4 tilstande fås de diskrete matricer vha. MATLAB's `c2d()`-funktion med $T_s = 0,01$ sekunder til

$$\mathbf{F} = \begin{bmatrix} 1,0007 & 0,01 & 0 & 0 \\ 0,1427 & 1,0007 & 0 & 0 \\ 0 & 0 & 1 & 0,01 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

$$\mathbf{G} = \begin{bmatrix} 0 \\ 0,0033 \\ 0,0001 \\ 0,0100 \end{bmatrix} \tag{4.4}$$

Matricerne $\mathbf{C}$ og $\mathbf{D}$ ændres ikke fra det kontinuerte system til det diskrete system.

Ligesom for kontinuert tid kan man til diskret tid benytte en "Discrete Linear Quadratic Regulator", DLQR, der ligesom den kontinuerte LQR kan hjælpe med at bestemme den optimale tilstandstilbagekobling. MATLAB har også en funktion `dlqr()` til DLQR-design. Med samme $\mathbf{Q}$, $\mathbf{R}$ og $\mathbf{N}$ som i LQR-designet giver DLQR-designet tilbagekoblingskonstanterne

$$\mathbf{K} = \begin{bmatrix} 109,90 & 29,105 & -0,4284 & -1,2312 \end{bmatrix} \tag{4.5}$$

$$\mathbf{E} = \begin{bmatrix} 0,962 & 0,9636 & 0,9948 + 0,0041i & 0,9948 - 0,0041i \end{bmatrix} \tag{4.6}$$

For modellen med 5 tilstande fra ligning (3.17) og (3.18) fås matricerne

$$\mathbf{F} = \begin{bmatrix} 1,0007 & 0,01 & 0 & -8,04 \cdot 10^{-6} & 3,23 \cdot 10^{-4} \\ 0,1427 & 1,0007 & 0 & -0,0016 & 0,0632 \\ 0 & 0 & 1 & 3,76 \cdot 10^{-4} & 9,73 \cdot 10^{-4} \\ 0 & 0 & 0 & 0,8813 & 4,7664 \\ 0 & 0 & 0 & 0,8813 & 1 \end{bmatrix} \tag{4.7}$$

$$\mathbf{G} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0,01 \end{bmatrix} \tag{4.8}$$

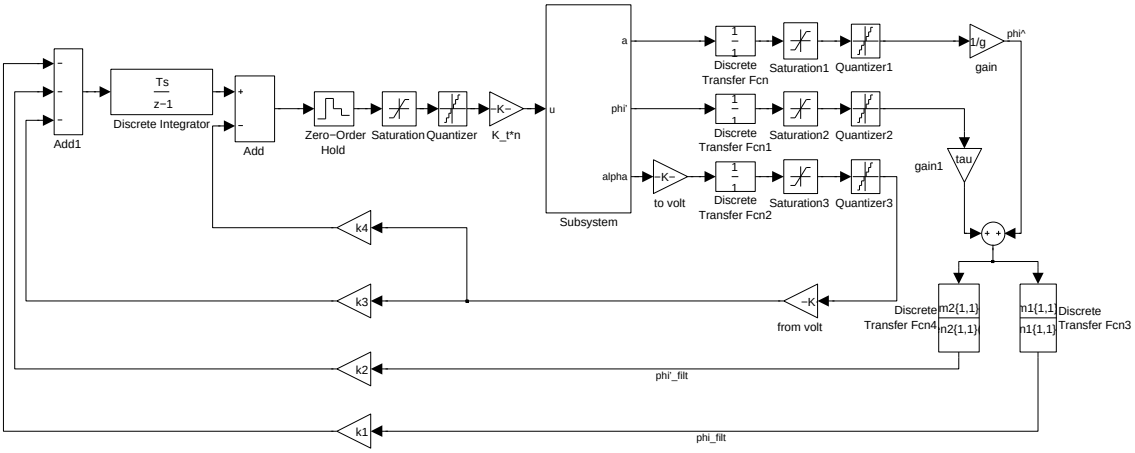
For 5 tilstande giver DLQR-designet

$$\mathbf{K} = \begin{bmatrix} 106,11 & 28,100 & -0,0477 & -0,2855 & 9,4634 \end{bmatrix} \tag{4.9}$$

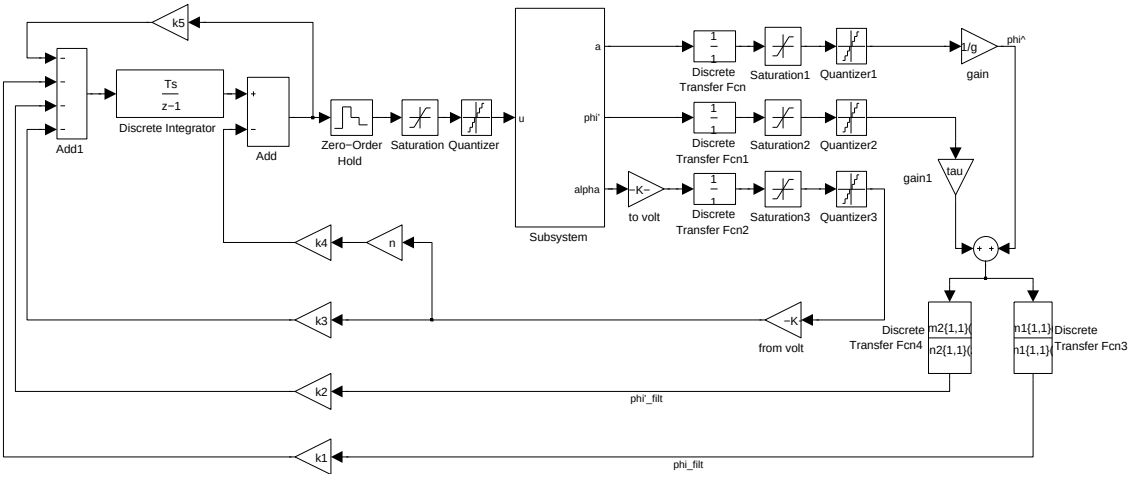
$$\mathbf{E} = \begin{bmatrix} 0,8827 & 0,9628 & 0,9631 & 0,9799 & 0,9996 \end{bmatrix} \tag{4.10}$$

4.2 Den diskrete blokmodel

Blokmodellen for det diskrete system, altså modellen som det implementeres for en computer, indeholder kvantiseringer, selvhold, zero-order-hold og diskrete overføringsfunktioner. Det eneste i modellen, der beholder sin kontinuerte natur, er selve linedansersystemet, dvs. indholdet af blokken fra tidligere figurer, bl. a. figur 3.15, forbliver som før. Den diskrete model for henholdsvis 4 og 5 tilstande er vist i figur 4.1 og 4.2.
Alt uden for blokken indeholdende systemet skal illustrere computerens beregninger. Målesignalerne a, $\dot{\varphi}$ og α bliver alle diskretiseret med hensyn til sampletiden T_s, selvholdt af grænserne for målingerne og kvantiseret i måleenhedens opløsning. Disse er alle begrænsninger, der dukker op i digitale systemer. Selvholdet er den øvre og den nedre grænse for værdien, som er de ekstremer, der kan



Figur 4.1: Den diskrete model med 4 tilstande



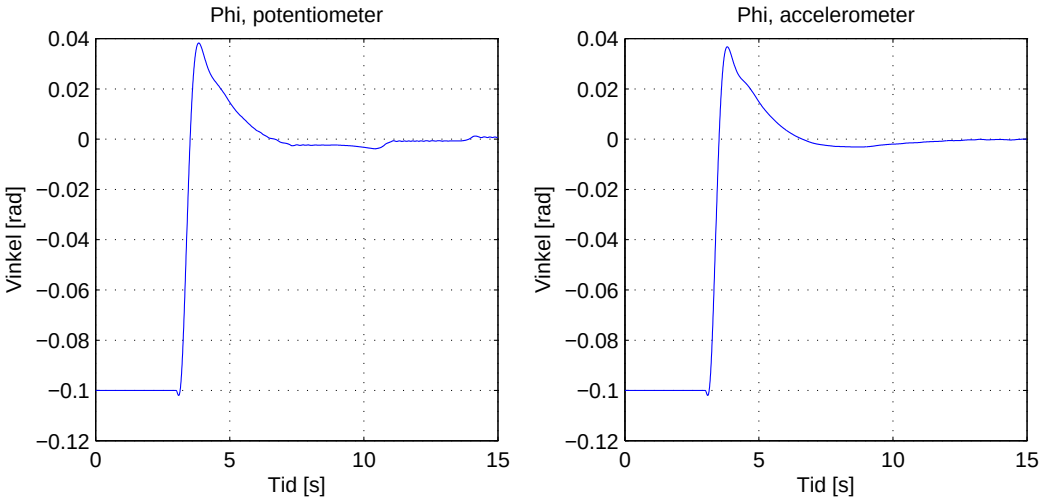
Figur 4.2: Den diskrete model med 5 tilstande

repræsenteres binært af den digitale måleenhed. Kvantiseringen er som nævnt opløsningen for måleenheden, dvs. hvor meget en bit repræsenterer. Dette er kort beskrevet i bilag D for gyroskopet og accelerometeret.

Vinklen α måles med computerens interface connector. Denne har et måleområde på $\pm 10V$. Den benytter 12 bit til målingerne, så kvantiseringen er $20V/4096\text{kvanter} = 0,00488V/\text{kvant}$. For at opnå en simulering tæt på virkeligheden, omregnes vinklen til en spænding, når den kommer ud af blokken. Efter kvantisering og selvhold omregnes der til en vinkel igen. På den måde kan spændingen også følges i simuleringen.
Indgangssignalet, u, til motoren er ligeledes selvholdt og kvantiseret. Dette er med de samme værdier som for målingen af α, som er $\pm 10V$ selvhold og

$0,00488V$/kvant kvantisering. Derudover er der et zero-order-hold for systemet, der gør at inputsignalet er et diskret signal, der holder på værdierne i hver periode af T_s.

Simulering af systemet for 4 tilstande og 5 tilstande er vist i henholdsvis figur 4.3 og 4.4.

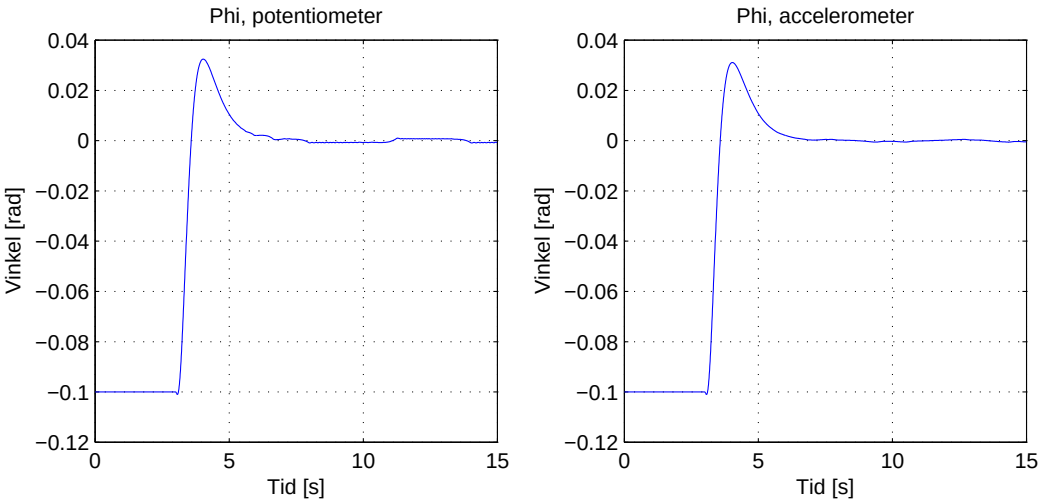


Figur 4.3: Simulering af den diskrete model for 4 tilstande med potentiometer og med accelerometer

4.3 Forbedring ved optimerede vægtmatricer

Vægtmatricen $\mathbf{Q}$ anvendt til LQR-designet og DLQR-designet har indtil nu bare været en identitets matrix. For at få den optimale regulering afhængig af den ønskede dynamik, kan man ændre denne, så den afspejler hvilke af tilstandene, der har størst betydning i forhold til hinanden. En tommelfingerregel for at vælge vægtmatricen er, at man for den enkelte tilstand benytter et mål for, hvor meget systemet i den stationære tilstand skal oscillere. Jo mindre udsving man ønsker, og derved mere præcis regulering, jo højere sættes værdien. Her benytter man værdien $\frac{1}{max(\text{udsving})^2}$. Vægtmatricen $\mathbf{Q}$ for n tilstande kommer derfor til at se ud på følgende måde

$$\mathbf{Q} = \begin{bmatrix} \frac{1}{max(\text{state}_1)^2} & 0 & \dots & 0 & 0 \\ 0 & \frac{1}{max(\text{state}_2)^2} & \dots & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \dots & \frac{1}{max(\text{state}_{n-1})^2} & 0 \\ 0 & 0 & \dots & 0 & \frac{1}{max(\text{state}_n)^2} \end{bmatrix} \tag{4.11}$$



Figur 4.4: Simulering af den diskrete model for 5 tilstande med potentiometer og accelerometer

Ud fra denne kan man variere matrixelementerne og se på simuleringsresultaterne, så man opnår den ønskede dynamik.

Den endeligt anvendte vægtmatrice for 4 tilstande er

$$\mathbf{Q} = \begin{bmatrix} 40000 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 \\ 0 & 0 & 1111 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix} \tag{4.12}$$

, med $\mathbf{R} = 400$ og $\mathbf{N} = [0 \quad 0 \quad 0 \quad 0]^T$, som giver DLQR-konstanterne og egenværdierne for det diskrete system med `dlqr()`-kommandoen i MATLAB:

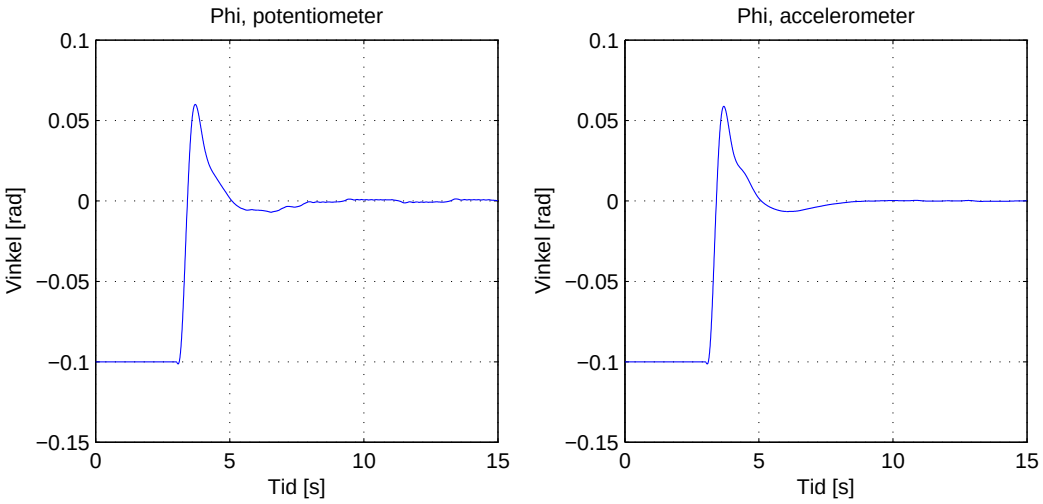
$$\mathbf{K} = \begin{bmatrix} 134,74 & 35,594 & -1,5900 & -2,6060 \end{bmatrix} \tag{4.13}$$

$$\mathbf{E} = \begin{bmatrix} 0,9627 + 0,0041i & 0,9627 - 0,0041i & 0,9909 + 0,0089i & 0,9909 - 0,0089i \end{bmatrix}$$

En simulering af dette både for accelerometer og potentiometer er vist i figur 4.5.

Det samme udregnes for 5 tilstande, hvor $\mathbf{Q}$-matricen er

$$\mathbf{Q} = \begin{bmatrix} 40000 & 0 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 1111 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & ... & 0 & 1 \end{bmatrix} \tag{4.14}$$



Figur 4.5: Simulering af den diskrete model for 4 tilstande potentiometer og med accelerometer

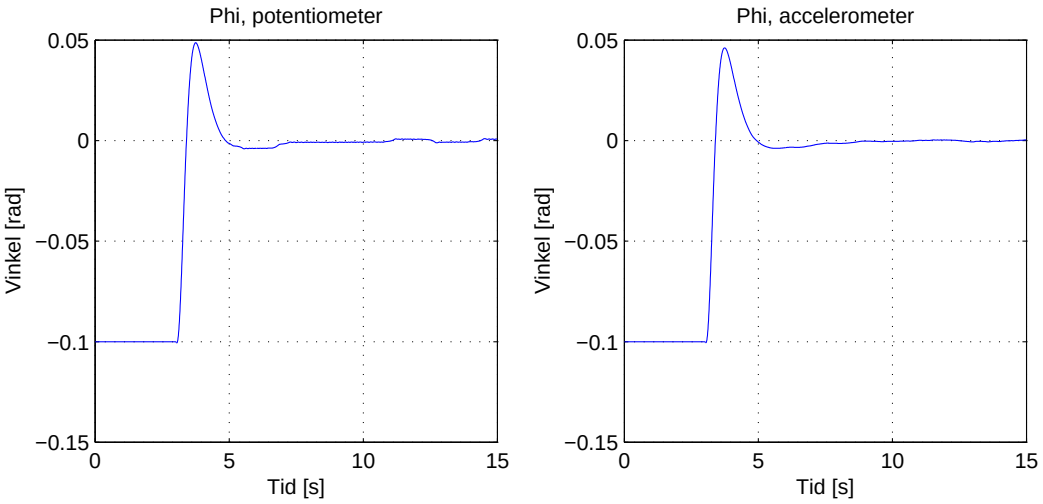
, hvilket for $\mathbf{R} = 400$ og $\mathbf{N} = [0 \quad 0 \quad 0 \quad 0]^T$ ved DLQR-design giver tilbagekoblingskonstanterne og egenværdierne

$$\mathbf{K} = \begin{bmatrix} 162,92 & 43,071 & -1,5718 & -0,4393 & 11,721 \end{bmatrix} \tag{4.15}$$

$$\mathbf{E} = \begin{bmatrix} 0,8877 & 0,9567 + 0,0089i & 0,9567 - 0,0089i & 0,9712 & 0,9932 \end{bmatrix}$$

Simuleringen af dette er vist i figur 4.6. Man kan sikkert optimere disse bedre (også bedre i forhold til hinanden), da der her er valgt samme vægtning for de 4 ens tilstande. Men ud fra dette taler simuleringen for at benytte modellen med 5 tilstande i og med, at oversvinget er mindre, og indsvingningstiden er hurtigere end for 4 tilstande.

Ved at prøve sig frem vil man derimod finde ud af, at startvinklen φ_0, hvormed modellen kan stabilisere sig fra, er væsentlig større for systemet med 4 tilstande end for systemet med 5 tilstande. For 4 tilstande viser simuleringen ved forskellige startvinkler, at systemet kan klare en startvinkel på ca. 1,05 radianer svarende til 60 grader. For 5 tilstande er grænsen på omkring 0,35 radianer svarende til 20 grader. Ved disse vinkler, især 60 grader, er modellen nok ikke særlig repræsentativ for det virkelige system, da modellen er lineariseret for små vinkler, som beskrevet i kapitel 2. Dette betyder, at de virkelige vinkelgrænser ikke er så høje. Dette undersøges i kapitel 6, hvor resultaterne for linedansersystemet er vist. Det er knap så vigtigt, hvor stor en startvinkel systemet kan have, når bare den holder sig inden for 10 grader. Det er mere vigtigt, at indsvingningen forløber pænt og overbevisende.



Figur 4.6: Simulering af den diskrete model for 5 tilstande med potentiometer og med accelerometer

Hermed er designet af regulatoren for systemet bestemt, og næste del af projektet består i at implemetere denne på en computer for så at afprøve den på det virkelige system.

Kapitel 5

Implementering

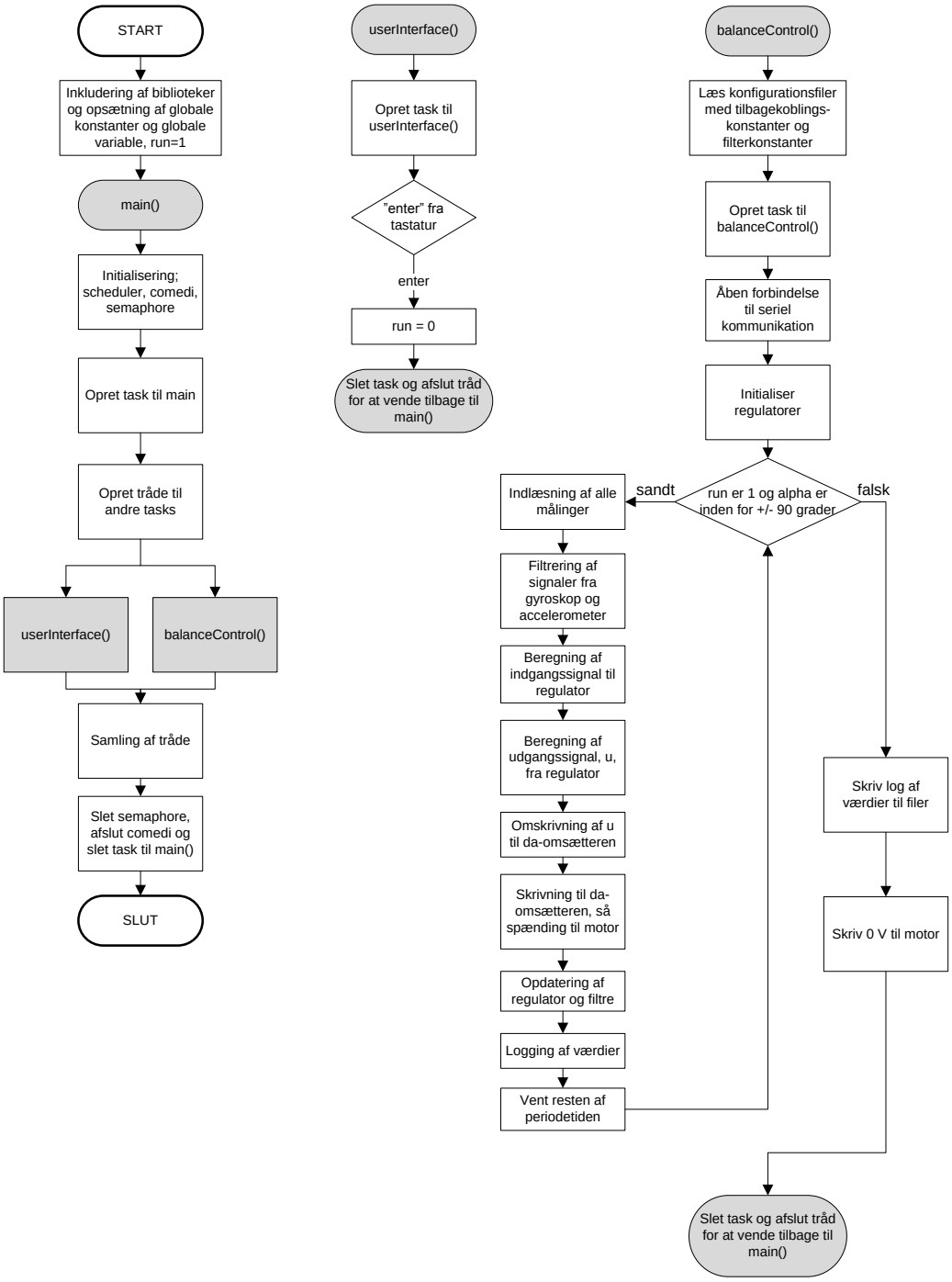
Programkoden til projektet er skrevet i programmeringsproget C og er opsat til at køre på en RTAI-Linux maskine, som er en realtidsmaskine, der indeholder mange værktøjer til realtidsopgaver, som reguleringen tager brug af. Af disse bruges der

- scheduler til at holde styr på, hvilke dele af programmet, der skal køre hvornår
- comedi til at forbinde til AD- og DA-omsætteren, der har forbindelse til motor og målekomponenter
- semaforer til at reservere kørselstid til specifikke opgaver, som andre funktioner, der er afhængige af denne semafor, ikke kan afbryde, før opgaven er færdig og semaforen er frigivet

I følgende afsnit er kodens grundlæggende dele beskrevet. Først er et flowdiagram vist for at skabe et overblik over koden, hvorefter de forskellige elementer i koden gennemgås. Desuden er kildekoden givet i bilag F.

5.1 Flowdiagram af kodestrukturen

I figur 5.1 ses flowdiagrammet for strukturen af koden. De vigtigste elementer indgår i funktionen `balanceControl()`.



Figur 5.1: Flowdiagram af koden

5.2 Delelementer i koden

Koden til implementering af regulatoren består af 3 filer: TRW.c, trwcontroller.c og trwcontroller.h samt en Makefile.

trwcontroller.c og trwcontroller.h indeholder de funktioner, der bruges til regulering og filtrering.

TRW.c består af 3 funktioner; `main()`, `userInterface()` og `balanceControl()`. De er illustreret i flowdiagrammet figur 5.1.

Hovedfunktionen `main()` bruges til at opsætte RTAI-maskinen med en scheduler, comedi til input og output med AD- og DA-omsætteren, en semafor og en task til `main()`-funktionen. Herudover opretter den to tråde (threads); én til hver funktion ud over main(). Derefter gør `main()`-funktionen ikke andet end at vente på, at de to tråde lukker, så den kan afslutte programmet.

Funktionen `userInterface()` er en kort funktion, der venter på, at brugeren trykker "enter" på tastaturet, så den globale variabel `run` sættes til 0, hvilket får programmet til at lukke. Denne funktion var ment til at blive større, når programmet kunne køre med den ethjulede cykel. Her er det senere meningen, at brugeren skal kunne indtaste forskellige kommandoer og parametre til at styre cyklen.

Den sidste funktion `TRW()` er den primære funktion i koden, hvor alt reguleringen foregår. Denne består af initialisering af regulator og filtre samt indlæsning af deres parametre. Resten af funktionen er en periodisk løkke, der kører, så længe `run`-variablen er 1. Derudover stopper programmet også, hvis balancestangen får en vinkel større end 90 grader enten den ene eller den anden vej. Dette er af sikkerhedsmæssige årsager både for opstillingen selv og for omkringstående personer. Efter løkken er kørt igennem, venter programmet på, at periodetiden T_s er gået, så løkken herved kører i perioder af T_s.

I løkken startes med at indlæse målinger fra potentiometer, gyroskop og accelerometer. Herefter filtreres gyroskop og accelerometermålingerne som beskrevet i afsnit 3.2. Herefter reguleres efter disse målinger og udgangen af regulatoren skrives til DA-omsætteren, som sender en spænding til motoren. Til sidst opdateres regulator og filtre, og væsentlige værdier logges.

Semaforen benyttes i koden, når de forskellige data logges til sidst i hver periode. I koden for linedanseren er det knap så nødvendigt i og med, der kun er den ene funktion, som benytter den. Men den kan blive nødvendig, hvis koden videreudvikles med en regulatorfunktion mere, der gør brug af nogle af de samme ressourcer. Dette sørger for, at programmet ikke bruger værdier, der kan være forsinkede eller for tidlige i forhold til andre værdier, der reguleres efter, og at måleresultaterne passer sammen, så man ikke ind i mellem har nogle måleresultater, der er målt til andre tidspunkter, da den anden funktion afbrød den første under nogle beregninger eller logging.

Man kan som sådan ikke implementere en integrator. Man er nødsaget til at

anvende en approksimation af integratoren. Denne og filtrene er beskrevet i de følgende underafsnit.

5.2.1 Implementering af integrator i regulatoren

Den diskrete integrator implementeres med det, der kaldes en "tustin"-approksimation. I MATLAB kan man bestemme en "tustin"-approksimation ud fra den kontinuerte integrator ved kommandoen `c2d()` ved at give den parametren `'tustin'`. Med en sampling-tid $T_s = 0.01$ sekunder, fås "tustin"-approksimationen for integratoren til

$$H_{int_{tustin}}(z) = \frac{0.005z + 0.005}{z - 1} \tag{5.1}$$

Denne kan implementeres, som det er gjort i C-filen trwcontroller.c, hvor koefficienterne indlæses fra nogle txt-filer i starten af programmet.

5.2.2 Implementering af lavpasfilter

Lavpasfiltret bestemmes også ved en "tustin"-approksimation på samme måde som integratoren. Herved fås

$$H_{low_{tustin}}(z) = \frac{0.009901z + 0.009901}{z - 0.9802} \tag{5.2}$$

Som integratoren findes funktionerne til implementering af filteret i filen trwcontroller.c

5.2.3 Implementering af højpasfilter

Højpasfiltret bestemmes ligeledes til

$$H_{high_{tustin}}(z) = \frac{1.98z + 1.98}{z - 0.9802} \tag{5.3}$$

Funktionerne for implementeringen ligger i filen trwcontroller.c ligesom de forrige.

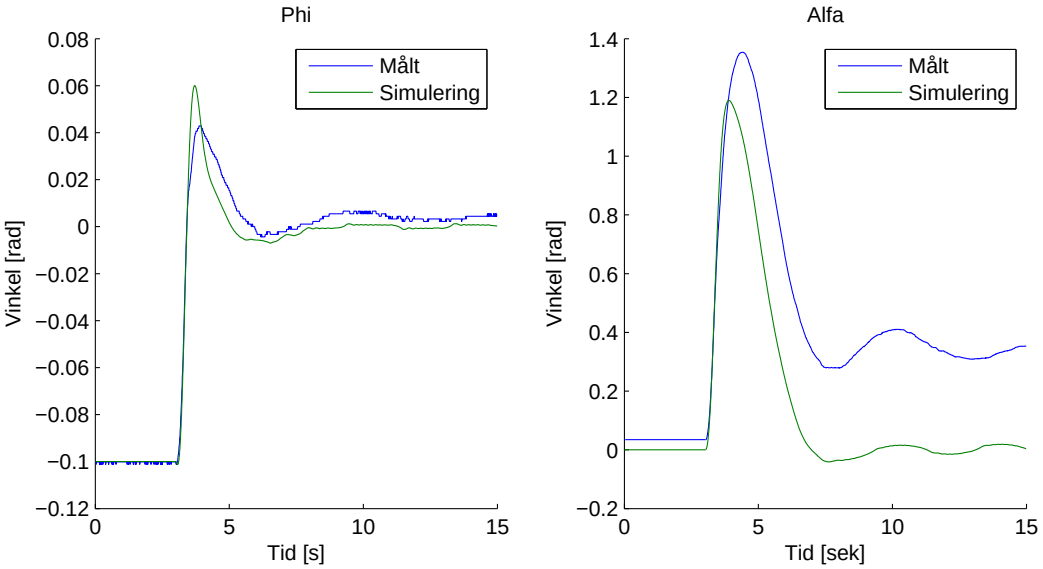
Kapitel 6

Resultater

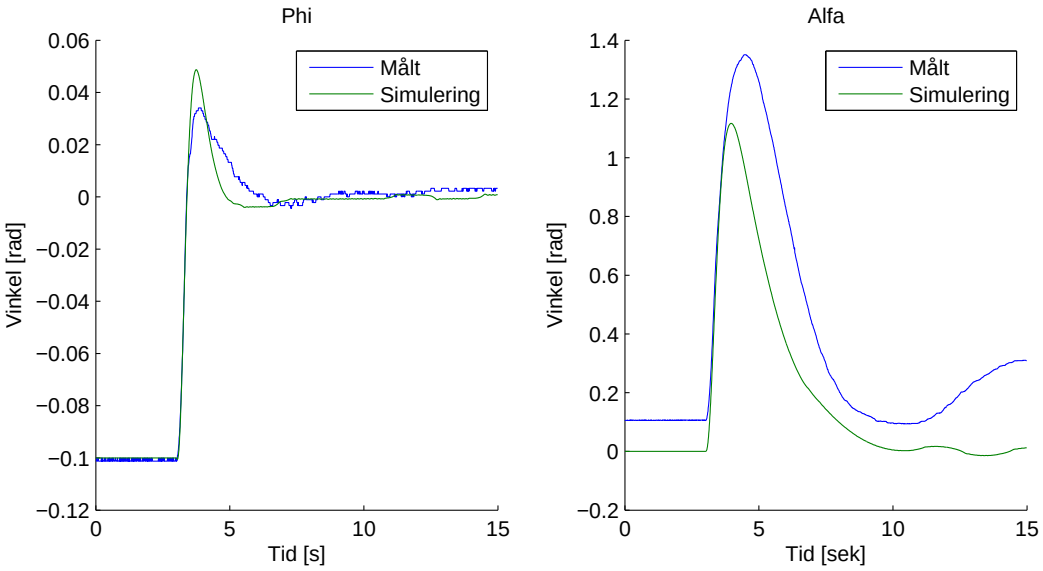
I dette kapitel er måleresultaterne for systemet af linedanseren vist, hvor også de tidligere simuleringer indgår til sammenligning.

6.1 Målinger med potentiometer

De første test er med potentiometeret til at måle vinklen φ og tilbagekoble fra de målinger. Tilbagekoblingen fra α er meget lille i forhold til den fra tilbagekoblingen fra φ. Dette gør, at det er svært at få α til at indsvinge sig omkring 0, hvilket også gør sig klart af følgende figurer, hvor både måleresultater for systemet med 4 tilstande og for 5 tilstande er vist i henholdsvis figur 6.1 og 6.2.



Figur 6.1: Måling af φ og α med potentiometer for 4 tilstande



Figur 6.2: Måling af φ og α med potentiometer for 5 tilstande

Det viser sig også her, at systemet med 5 tilstande har en bedre dynamik end for systemet med 4 tilstande.

6.2 Målinger med accelerometer og gyroskop

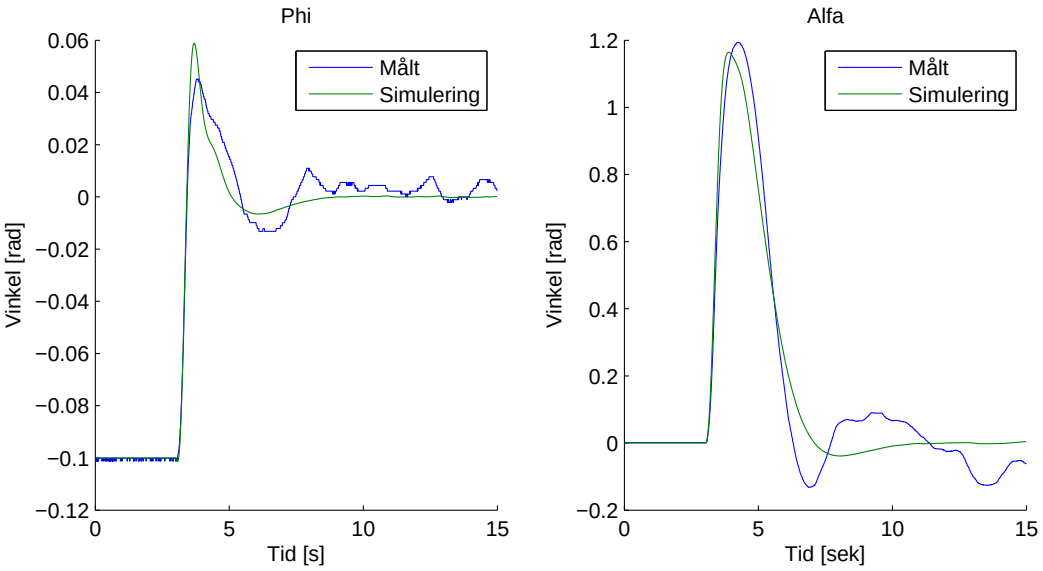
Målingerne er her målt med accelerometeret og gyroskopet, som filteres og tilbagekobles. Målingerne i figurerne 6.3 og 6.4 er dog målinger foretaget med potentiometret på φ, da disse giver mere præcise målinger uden støj.

Overordnet set ser systemt med 5 tilstande ud til at virke bedst i forhold til oversving og indsvingningstid.

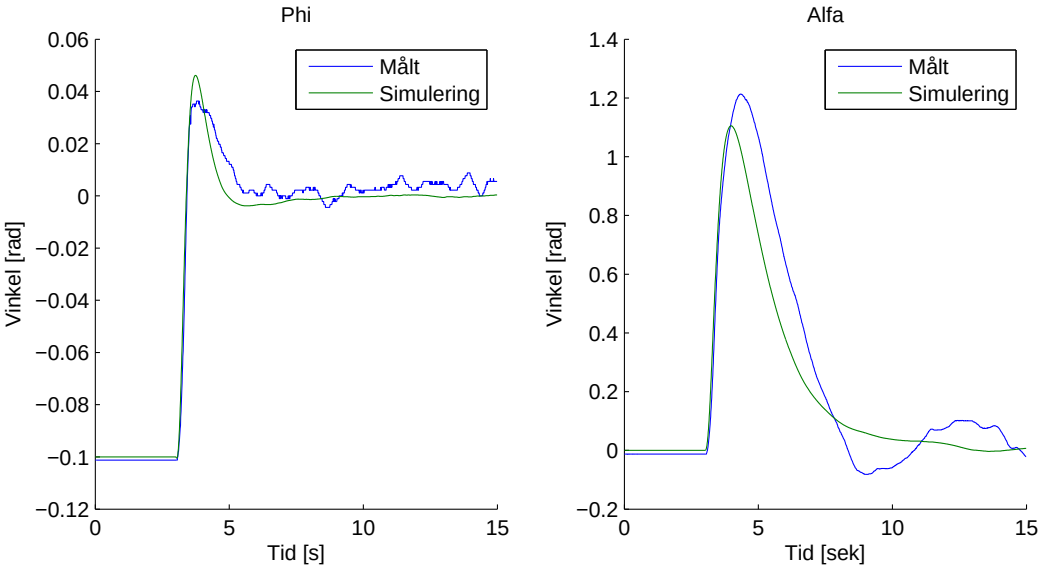
Yderligere laves en test for at se at simulering og målinger stemmer overens, hvor der indgår forstyrrelser på styresignalet til motoren. Dette vises i figur 6.5.

Her ligger kurverne forholdsvist pænt oven i hinanden. α er lidt ved siden af, men som forklaret før, er tilbagekoblingen herfra lille i forhold til for φ. Her er det vigtige, at dynamikken ligner hinanden i simuleringen og i måleresultaterne.

Regulatoren med 5 tilstande har gennem hele designforløbet virket som det mest fornuftige regulatordesign, og for en startvinkel på 0,1 radianer har den en indsvingningstid på omkring 3 sekunder med ca. 40% oversving, hvilket er en anelse bedre end regulatoren med 4 tilstande. Samtidig har svingningerne omkring den stationære tilstand for systemet med 5 tilstande mindre amplitude og står derfor mere stabilt ved stilstand. Yderligere er der gjort antagelser om, at accelerometeret er placeret ved omdrejningsaksen for pendulet, da det gav nogle uønskede svingninger for systemet med 4 tilstande. Regulatoren med 5



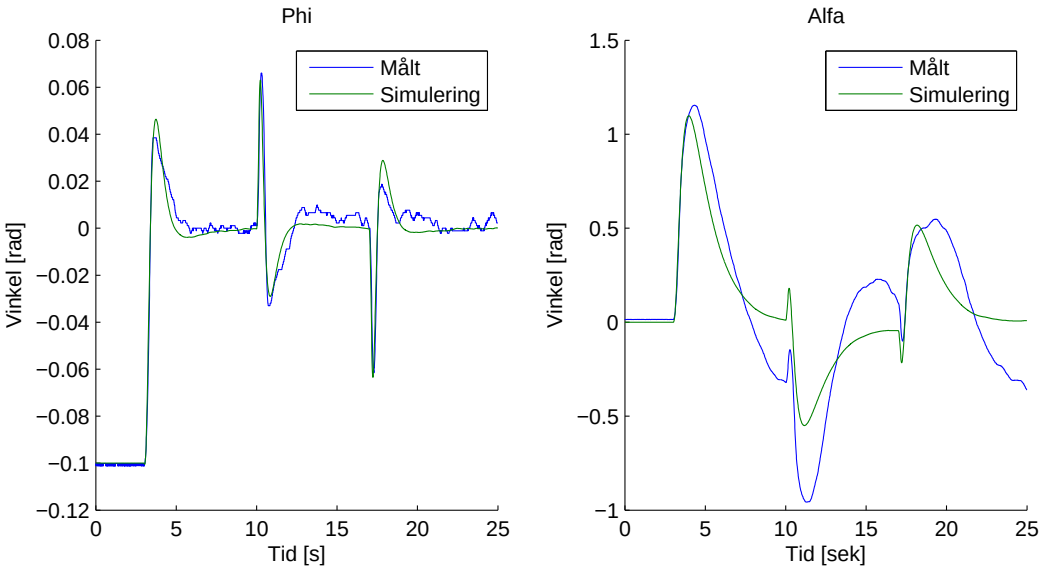
Figur 6.3: Måling af φ og α med accelerometer for 4 tilstande



Figur 6.4: Måling af φ og α med accelerometer for 5 tilstande

tilstande er derfor at foretrække og er valgt som det endelige regulatordesign.

Opstillingen af linedanseren er efterfølgende blevet afmonteret fra sit stativ for at være helt uden støtte. Følgende grafer af forskellige målinger har derfor ikke kunnet måles med et potentiometer, men er målt som det filtrerede signal



Figur 6.5: Måling af φ og α med forstyrrelser på motorsignalet

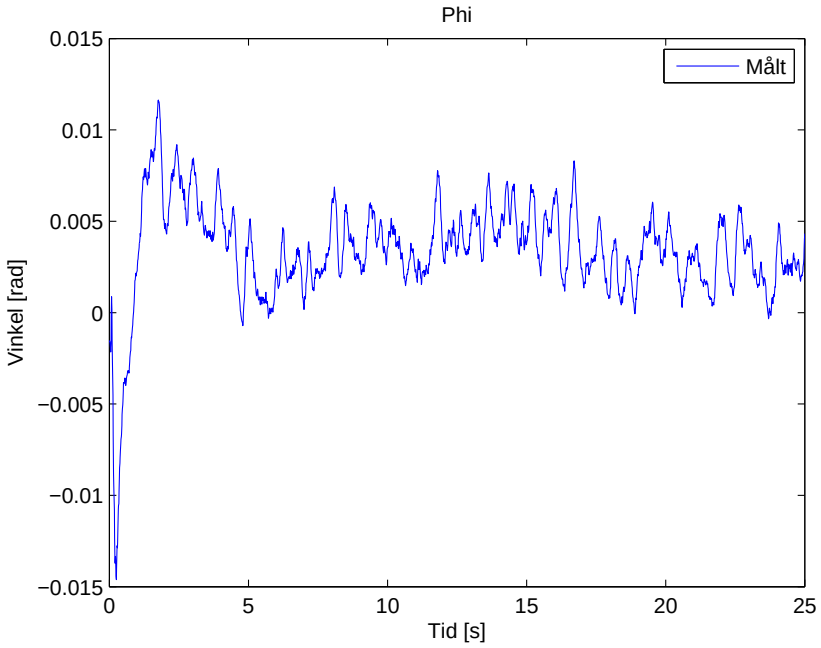
fra accelerometer og gyroskop. Afmontering af potentiometerakslen skulle give et mere præcist billede af det virkelige system, da der sandsynligvis har været noget friktion forbundet med denne aksel. Derudover har et gummibeslag ved fastspænding af potentiometret givet en smule elasticitet til akslen.

En måling af linedanseren, som balancerer på en skinne, er vist i figur 6.6. Her ses det, at systemet i den stationære tilstand svinger inden for $\pm 0,005$ radianer.

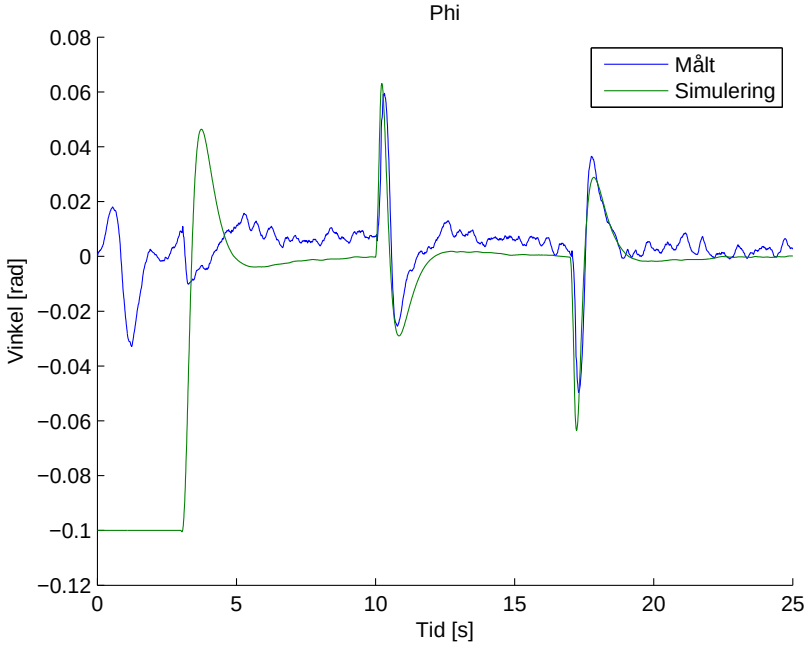
Det samme er vist i figur 6.7, hvor der er givet en forstyrrelse svarende til det, der er vist i figur 6.5. Her er de 3 første sekunder ligegyldige samt det første af indsvingningsforløbet, hvor man i noget af tiden er nødt til at holde den, da den ingen støtte har. Her ses, at systemet følger modellen meget godt ligesom set i figur 6.5 men med et lille offset.

Det har vist sig, at det virkelige system med 5 tilstande i reguleringen kunne klare en startvinkel på ca. 0,175 radianer, dvs. ca. 10 grader, uden at vælte, og designet med 4 tilstande kunne klare en til to grader mere. Som forventet ville det virkelige system ikke klare så store vinkler som simuleringen forudså. Men systemet med 4 tilstande viste sig alligevel at kunne klare en større startvinkel end det andet system. Startvinklen på 10 grader er tilfredsstillende, så selvom systemet med 4 tilstande kan klare en lidt større vinkel, vælges regulatoren med 5 tilbagekoblinger som det endelige design, da dens dynamik ved indsvingning er pænere og linedanseren står mere stabilt i den stationære tilstand.

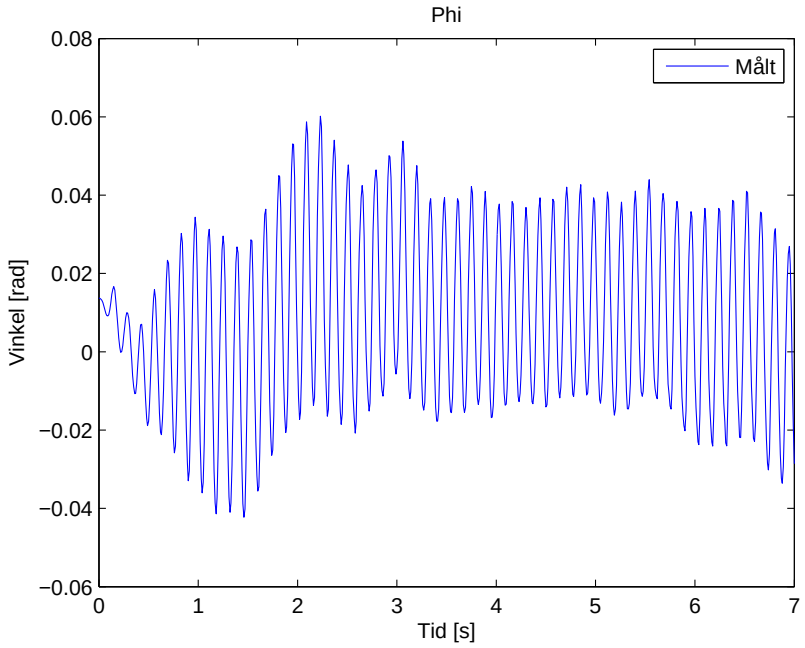
Efterfølgende var det interessant at se, hvorledes linedanseropstillingen var i stand til at balancere på en line, der modsat skinnen er noget elastisk.



Figur 6.6: Måling for linedanseren på en skinne



Figur 6.7: Måling for linedanseren på en skinne med forstyrrelse



Figur 6.8: Måling for linedanseren på en line

På trods af, at den opsatte line ikke var strammet mere, end det umiddelbart kunne lade sig gøre, kunne linedanseren stadig opretholde sin balance med mange rystelser, der kommer af linens elasticitet. Man kan sammenligne dette med selv at prøve at stå på en line, hvor man ville stå og ryste i forsøget på at holde balancen.

Med den designede regulator er det lykkedes at få linedanseren til at have en rimelig stabil balance med pæne indsving fra forstyrrelser. Ud fra målinger, hvor opstillingen var afmonteret fra sin støtte, holdes linedanseren stabil inden for $\pm 0,005$ radianers udsving (se figur 6.6), hvilket er lige under $\pm 0,3$ grader, og det kan klare en startvinkel uden at vælte på ca. 10 grader. Dette må siges at være acceptabelt. Det har vist sig, at den opstillede model repræsenterer det virkelige system meget pænt. Dette åbner nu op for, at få den anden regulator til den ethjulede cykel designet og implementeret. Næste kapitel introducerer teorien for modellen af en sådan unicykel.

6.3 Fejlkilder

Under projektet er noteret en række fejlkilder, som står listet nedenfor:

- Gummibeslaget fra potentiometeret til pendulets aksel i opstillingen har muligvis givet lidt elasticitet og påvirket systemet en smule
- En mulig friktion i pendulets gennemgående aksel i opstillingen giver en lille modstand i stabiliseringen
- Linearisering af modellen er kun en tilnærmelse af det virkelige system
- I modelopstillingen er gjort visse antagelser om delelementernes inertimomenter, som f. eks. at lodderne på balancestangen blev set som punktmasser
- Der indgår sandsynligvis nogle offset-fejl i målingerne fra gyroskop, accelerometer og potentiometer, som kan være svære fuldstændigt at nulstille
- Gyroskopet og accelerometeret har internt nogle driftsfejl, der akkumuleres over tid. Ved lang tids kørsel kan dette give fejlmålinger

Kapitel 7

Modellering af unicyklen

Den ethjulede cykel, unicyklen, er et meget simpelt transportmiddel, men det er yderst komplekst at køre på. Her skal man holde balancen både sidelæns som for linedanseren men også frem- og tilbagerettet. Se f. eks. figur 7.1. Dette kapitel danner grundlag for at kunne designe regulatoren for at stabilisere unicyklen ved at opstille dens model og simulere denne.

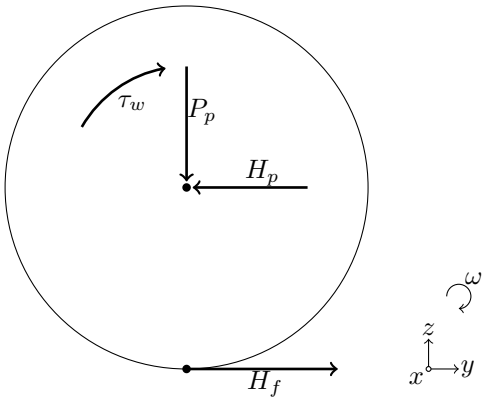
Figur 7.1: Cirkusartist forsøger sig med dette vovede nummer over en line

http://darkroom.baltimoresun.com/2012/04/a-day-at-the-piccadilly-circus/ph-ho-vw-circus-9/

7.1 Modelopstilling

Halvdelen af balancen af unicyklen er på plads med balancereguleringen af linedanseren, som er den sidelæns balance. Nu mangler den frem- og tilbagerettede balance. Modellen for unicyklen bygger på et afgangsprojekt fra The University of Western Australia School of Mechanical Engineering [6], som beskæftiger sig med balance af en tohjulet robot med hjulene placeret parallelt

overfor hinanden som eksempelvis ved en Segway. Teorien er stort set det samme med nogle forsimplinger, hvor kun ét hjul er monteret på det omvendte pendul ved hjulets aksel. En figur af hjulet og de kræfter, der påvirker det, er vist i figur 7.2. Orienteringen af koordinatsystemet er vendt, så det passer med figur 2.2.



Figur 7.2: Model af hjulet og dets kræfter

H_f er friktionskraften fra underlaget på hjulet, H_p er den kraft, pendulet påvirker hjulet med i y-aksens retning, P_p er den kraft, pendulet påvirker hjulet med i z-aksens retning, og τ_w er kraftmomentet, som motoren påvirker hjulet med. Hjulets vinkelomdrejning er ω. Der ses kun på bevægelsen i y-aksens retning. Hjulets masse er M_w og hjulets radius er R. For at følge samme fremgangsmåde som for modelopstillingen af linedanseren ignoreres motorkraften indtil videre, så der kun kigges på systemets dynamik. Ved at benytte Newtons lov om summen af kræfter langs den vandrette akse fås

$$\ddot{y}\, M_w = H_f - H_p \tag{7.1}$$

Det samme kan opstilles for summen af kraftmomenter

$$\ddot{\omega}\, I_w = -H_f R \tag{7.2}$$

Der gælder her at $\ddot{\omega} = \frac{\ddot{y}}{R}$. Ligning (7.2) bliver herved til

$$\ddot{y}\, \frac{I_w}{R} = -H_f R \tag{7.3}$$

Isoleres H_f fra ligning (7.1), fås

$$H_f = \ddot{y}\, M_w + H_p \tag{7.4}$$

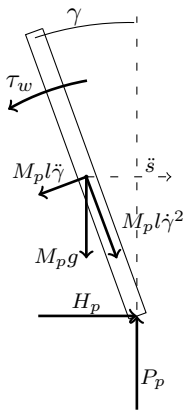
Dette indsættes i 7.3, hvorved man får

$$\ddot{y}\, \frac{I_w}{R} = -\ddot{y}\, M_w R - H_p R \tag{7.5}$$

Dette omskrives til

$$\left(M_w + \frac{I_w}{R^2}\right) \ddot{y} = -H_p \tag{7.6}$$

For selve pendulstangen er vist figur 7.3 med de kræfter, der virker på den.



Figur 7.3: Model af det omvendte pendul og dets kræfter

l er afstanden fra hjulets centrum, hvor pendulstangen er monteret, til pendulstangens massemidtpunkt. Kræfterne $M_p l\ddot{\gamma}$ og $M_p l\dot{\gamma}^2$ kommer af, at det antages, at pendulet foretager en cirkelbevægelse, hvor den tangentielle acceleration er $a_t = l\ddot{\gamma}$ og den radielle acceleration er $a_r = l\dot{\gamma}^2$. Herved kan summen af kræfter i den vandrette retning opskrives

$$M_p\, \ddot{y} = H_p - M_p l\ddot{\gamma}\cos(\gamma) + M_p l\dot{\gamma}^2 \sin(\gamma) \tag{7.7}$$

, hvor M_p er massen af pendulstangen. Her kan H_p isoleres til

$$H_p = M_p\, \ddot{y} + M_p l\ddot{\gamma}\cos(\gamma) - M_p l\dot{\gamma}^2 \sin(\gamma) \tag{7.8}$$

Denne kan nu indsættes i ligning (7.6), hvilket giver

$$\left(M_w + \frac{I_w}{R^2}\right) \ddot{y} = -M_p\, \ddot{s} - M_p l\ddot{\gamma}\cos(\gamma) + M_p l\dot{\gamma}^2 \sin(\gamma) \tag{7.9}$$

Herefter findes summen af kræfter vinkelret på pendulstangen. Det vil sige i modsat retning af kraften $M_p l\ddot{\gamma}$

$$M_p\, \ddot{y}\cos(\gamma) = H_p \cos(\gamma) + P_p \sin(\gamma) - M_p g \sin(\gamma) - M_p l\ddot{\gamma} \tag{7.10}$$

Kraftmomentet omkring massemidtpunktet, hvor den positive omløbsretning er mod urets retning, opstilles

$$I_p\ddot{\gamma} = H_p l\cos(\gamma) + P_p l\sin(\gamma) \tag{7.11}$$

Dette omskrives til

$$H_p l \cos(\gamma) + P_p l \sin(\gamma) = I_p \ddot{\gamma} \tag{7.12}$$

Hvis man ganger ligning (7.10) med l, fås

$$M_p l \, \ddot{y} \cos(\gamma) = H_p l \cos(\gamma) + P_p l \sin(\gamma) - M_p l g \sin(\gamma) - M_p l^2 \ddot{\gamma} \tag{7.13}$$

Nu vil man se, at ligning (7.12) kan indsættes i ligning (7.13), hvoraf H_p og P_p nu helt kan elimineres fra modellen. Her vil man få

$$M_p l \, \ddot{y} \cos(\gamma) = I_p \ddot{\gamma} - M_p l g \sin(\gamma) - M_p l^2 \ddot{\gamma} \tag{7.14}$$

Disse ovenstående ligninger er ulineære. Disse lineariseres derfor med antagelserne $\dot{\gamma}^2 \simeq 0$, samt det indføres at $\gamma = \pi + \delta$, hvor δ er en lille vinkel. Dette gør, at $\cos(\gamma) \simeq -1$ og $\sin(\gamma) \simeq -\delta$.

Ligning (7.9) og (7.14) er efter lineariseringen

$$\left(M_w + \frac{I_w}{R^2} \right) \ddot{y} = -M_p \, \ddot{y} - M_p l \ddot{\delta} \tag{7.15}$$

$$-M_p l \, \ddot{y} = -I_p \ddot{\delta} + M_p l g \, \delta + M_p l^2 \ddot{\delta} \tag{7.16}$$

Ligning (7.15) kan ganges med l, hvilket giver

$$\left(M_w l + \frac{I_w l}{R^2} \right) \ddot{y} = -M_p l \, \ddot{y} - M_p l^2 \ddot{\delta} \tag{7.17}$$

Det ses, at et af leddene $M_p l^2 \ddot{\delta}$ optræder i begge ligninger. Derfor isoleres dette fra ligning (7.17).

$$M_p l^2 \ddot{\delta} = -\left(M_w l + \frac{I_w l}{R^2} \right) \ddot{y} - M_p l \, \ddot{y} \tag{7.18}$$

Dette indsættes efterfølgende i ligning (7.16).

$$-M_p l \, \ddot{y} = -I_p \ddot{\delta} + M_p l g \, \delta - \left(M_w l + \frac{I_w l}{R^2} \right) \ddot{y} - M_p l \, \ddot{y} \tag{7.19}$$

De to variable samles nu på hver sin side, så man får

$$\left(M_w l + \frac{I_w l}{R^2} \right) \ddot{y} = -I_p \ddot{\delta} + M_p l g \, \delta \tag{7.20}$$

Ligning (7.20) er herved modelligningen for systemet.

7.2 Laplace-transformering af modellen

Laplace-transformering af modellen fra ligning (7.20) giver

$$\left(M_w l + \frac{I_w l}{R^2}\right) \ddot{y}(s) = -I_p s^2 \delta(s) + M_p l g \, \delta(s) \tag{7.21}$$

Dette giver, at

$$\delta(s) = \frac{\left(M_w l + \frac{I_w l}{R^2}\right)}{-I_p s^2 + M_p l g} \ddot{y}(s) \tag{7.22}$$

, hvilket kan omskrives til

$$\delta(s) = \frac{-\left(\frac{M_w l}{I_p} + \frac{I_w l}{I_p R^2}\right)}{s^2 - \frac{M_p l g}{I_p}} \ddot{y}(s) \tag{7.23}$$

State-spacemodellen for systemet med $\mathbf{x} = [\delta \quad \dot{\delta} \quad y \quad \dot{y}]^T$ kan ud fra modellen skrives som

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{M_p l g}{I_p} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{7.24}$$

$$\mathbf{B} = \begin{bmatrix} 0 \\ -\left(\frac{M_w l}{I_p} + \frac{I_w l}{I_p R^2}\right) \\ 0 \\ 1 \end{bmatrix} \tag{7.25}$$

7.3 Blokdiagram af modellen

Modellens blokdiagram kommer ud fra ligning (7.24) og (7.25) til se ud som i figur 7.4, hvor også den ideele servomekanisme er implementeret. Servomekanismens tilbagekoblingskonstanter er lavet ud fra LQR-design af systemets state-space model, ligesom det er beskrevet i kapitel 3. Her er konstanterne valgt ud fra tabellen i bilag A.

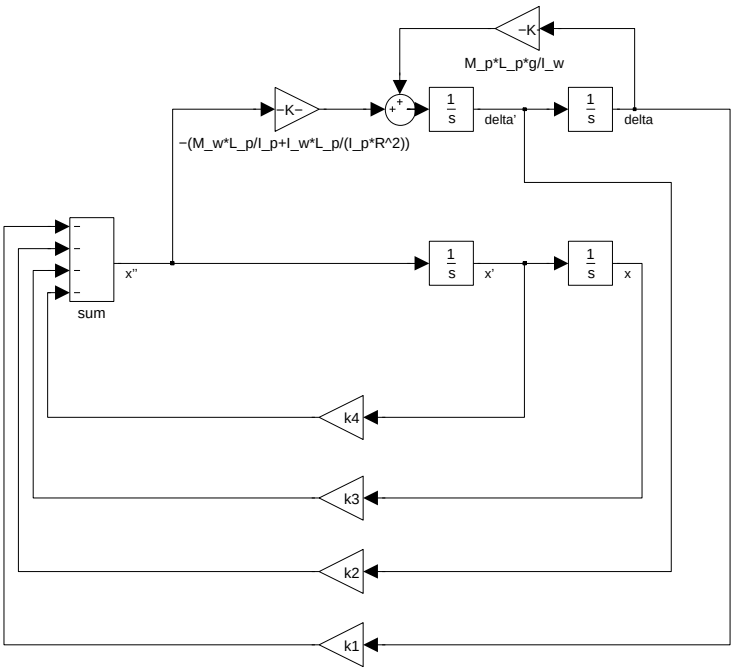
Før tilbagekoblingen var systemets egenværdier

$$\mathbf{E} = \begin{bmatrix} 7,621 & -7.621 & 0 & 0 \end{bmatrix} \tag{7.26}$$

, hvilket giver at systemet er ustabilt.

Ved LQR-designet er benyttet

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{R} = 5, \mathbf{N} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{7.27}$$



Figur 7.4: Blokdiagram af model med ideel servomekanisme

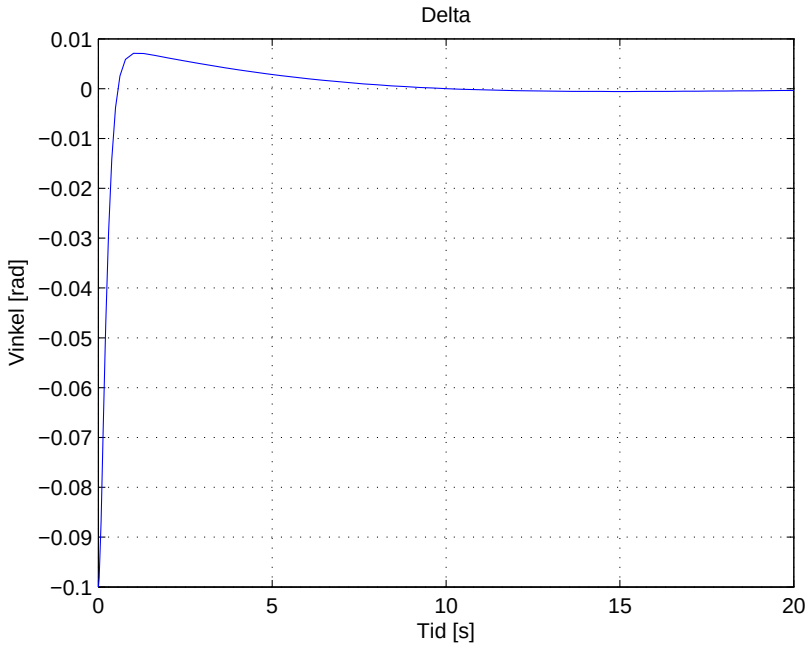
, hvilket resulterer i et LQR-design med tilbagekoblingskonstanterne og egenværdierne

$$\mathbf{K} = \begin{bmatrix} -19,172 & -2,516 & -0,050 & -0,333 \end{bmatrix} \tag{7.28}$$

$$\mathbf{E} = \begin{bmatrix} -7,779 & -7,466 & -0,160 + 0,156i & -0,160 - 0,156i \end{bmatrix} \tag{7.29}$$

Dette giver et stabilt system. En simulering af dette med en startvinkel $\delta_0 = 0,1$ radianer er vist i figur 7.5.

Pga. et større mekanisk arbejde i at designe og bygge unicyklen, der ikke har været tid til, har det ikke været muligt at teste modellen mod det virkelige system. Herfra er vejen lagt til at arbejde videre med projektet, så unicyklen kommer til at stå oprejst.



Figur 7.5: Simulering af det ideelle servosystem for den ethjulede cykel

Kapitel 8

Videreudvikling på projektet

Linedanseropstillingen har vist sig at kunne stabiliseres med den designede regulator. Denne regulator er ikke nødvendigvis den optimale. Ved at arbejde videre på at justere vægtningen i vægtmatricen kan regulatoren sikkert optimeres endnu mere. Heraf vil det også være relevant at afprøve andre regulatordesign end lige LQR-design for at bestemme et bedre design.

Afprøvning af bedre sensorer med bedre opløsning og mindre støj vil uden tvivl forbedre måleresultaterne og derved stabiliseringen. Her vil man nok opleve en systemdynamik, der ligger endnu tættere på modellens.

Som det tidligere kapitel har lagt op til, vil det være interessant at udvikle linedanseropstillingen, så den blev i stand til at balancere på en ethjulet cykel. Herfra kunne man udvikle opstillingen til en unicykelrobot, som både kan køre og dreje samtidig med, at den holder balancen. Dette kan være ved, at den skal modtage og udføre kommandoer givet via en terminal eller programmeres til at udføre forskellige opgaver gennem en række kommandoer, så den er en selvkørende unicykelrobot.
I dette skal der bestemmes en motor, der skal fungere som aktuator på systemet. Heri ligger meget mekanisk arbejde i design af cyklen, så linedanseren kan monteres derpå. Der skal tages højde for bl.a., hvor stor og tung cyklen skal være, og hvorledes motoren skal trække hjulene, uden at cyklen trækker til den ene side under kørsel.
Den valgte motor skal inddrages i modellen, hvorfra der skal designes en regulator ved f. eks. et LQR-design. Dette skal overføres til en diskret regulator, der således kan implementeres som en ny funktion i den allerede etablerede kode for linedanseren. Denne funktion skal køre sideløbende med linedanserdelen for så i samarbejde at balancere cyklen.

Kapitel 9

Konklusion

Ved at modellere, hvorledes en linedanser holder balancen gennem inertialberegninger, er det lykkedes med værktøjer og simuleringer i MATLAB at designe en regulator, der kan få en linedanseropstilling til at balancere. Regulatoren er lavet ved et såkaldt LQR-design, hvor der bestemmes tilbagekoblingskonstanter med forskellige vægtninger fra hver tilstand i modellen. Ved hjælp af en række sensorer som potentiometer, accelerometer og gyroskop har det været muligt at bestemme disse tilstande, som er vinkler og vinkelhastigheder.

Gennem simuleringer først i kontinuert tid og derefter i diskret tid har det vist sig bedst ikke kun at tilbagekoble fra modellens tilstande, men også fra den tilstand, der opstår ved integrering i regulatoren, så man i stedet for 4 tilstande har 5 tilstande i sin state-space model.

Det har følgende vist sig, at den opstillede model for linedansersystemet, repræsenterer det virkelige system meget pænt dynamisk set. Dette viser sig ved sammenligning af resultater fra simuleringen og fra det virkelige system ud fra indsvingningsforløb og dynamik ved givne forstyrrelser i motorsignalet. Med det endelige regulatordesign stabiliserede systemet sig inden for $\pm 0,3$ grader og en maksimal startvinkel på 10 grader uden at vælte. Ved en startvinkel på 0,1 radianer, som svarer til 5,7 grader, blev en indsvingningstid målt til ca. 3 sekunder.

Til sidst har dette projekt beskæftiget sig med teorien omkring modelopstillingen af en unicykel i forsøget på at udvikle projektet mod en ethjulet robot, der med linedanserens opstilling holder den sidelæns balance, mens et monteret hjul sørger for at holde den fremadrettede balance. Opbygningen af en fysisk model ville kræve et stort mekanisk arbejde, hvilket, der i dette projekt, ikke har været tid til. Dette projekt afsluttes ved at pege på en række forslag til forbedringer og en videreudvikling hen i mod en unicykelrobot, der balancerer helt af sig selv.

Litteratur

[1] Nielsen, Michael Heisel, *Linedanserens Stabilitetsproblem*. Servolaboratoriet, Eksamensprojekt, 1982.

[2] Yu, Zhiyin, *Control of Unstable Object*. DTU Electrical Engineering, Master's Thesis, 2011.

[3] Hendricks, Elbert & Jannerup, Ole & Sørensen, Poul Haase, *Linear Systems Control*. Springer, 2008.

[4] Leth, Nils, *Servomekanismer, bind 1*. Servolaboratoriet, 1978.

[5] Bar-Shalom, Yaakov & Li, X. Rong & Kirubarajan, Thiagalingam, *Estimation with Applications to Tracking and Navigation*. John Wiley & Sons, Inc., 2001.

[6] Ooi, Rich Chi, *Balancing a Two-Wheeled Autonomous Robot*. Final Year Thesis, The University of Western Australia School of Mechanical Engineering, 2003.

Bilag A

Symbolliste og værdier

Dette er en liste over de symboler og fysiske værdier benyttet i de to modeller.

A.1 Linedansermodellen

a	Pendulets acceleration	
α	Vinkel af balancestang i forhold til pendul	
$\dot{\alpha}$	Vinkelhastighed af balancestang i forhold til pendul	
$\ddot{\alpha}$	Vinkelacceleration af balancestang i forhold til pendul	
β	Vinkelposition af motorakslen	
$\dot{\beta}$	Vinkelhastighed af motorakslen	
$\ddot{\beta}$	Vinkelacceleration af motorakslen	
c	Indført konstant	0,818
f	Indre modstand i motoren, som antages	$0\ N \cdot m$
g	Tyngdeaccelerationen	$9{,}82\ \frac{\mathrm{m}}{\mathrm{s}^2}$
I_a	Pendulstangens samlede inertimoment	$0{,}313\ \mathrm{kg{\cdot}m}^2$
I_b	Balancestangens inertimoment	$0{,}0865\ \mathrm{kg{\cdot}m}^2$
I_p	Pendulstangens inertimoment alene	$0{,}0133\ \mathrm{kg{\cdot}m}^2$
I_t	Det samlede inertimoment for motoraksen	$9,127 \cdot 10^{-5}\mathrm{kg} \cdot \mathrm{m}^2$
J	Motorens rotorinerti (antages 0 i model)	$9,18 \cdot 10^{-7}\mathrm{kg} \cdot \mathrm{m}^2$
$\mathbf{K}$	Samlet vektor for tilbagekoblingskonstanterne	
$\mathbf{E}$	Samlet vektor for egenværdierne	
$K0$	Samlet motorforstærkning	$5{,}074\ \frac{\mathrm{rad}}{\mathrm{V{\cdot}s}^2}$
K_e	Tilbagekoblingskonstant i motor	$7,766 \cdot 10^{-3}\,\frac{\mathrm{V{\cdot}s}}{\mathrm{rad}}$
K_g	Indgangsforstækning efter tachometertilbagekobling	100
K_t	Tachometerkonstant	$0{,}0248\ \frac{\mathrm{V}}{\mathrm{rad}}$

K_{tq}	Motortorque	$7,766 \cdot 10^{-3} \frac{\mathrm{N \cdot m}}{\mathrm{A}}$
$k1$	Tilbagekoblingskonstant fra φ	
$k2$	Tilbagekoblingskonstant fra $\dot{\varphi}$	
$k3$	Tilbagekoblingskonstant fra α	
$k4$	Tilbagekoblingskonstant fra $\dot{\alpha}$ eller $\dot{\beta}$	
$k5$	Tilbagekoblingskonstant fra u	
L	Motorens rotorinduktans, som antages	0 H
L_{a_1}	Placering af gearingstandhjulet på pendulstangen	0,061 m
L_p	Længde af pendul	0,502 m
L_b	Længde af balancestang	0,864 m
L_m	Placering af motoren i forhold til pendulstangen	0,030 m
L_T	Tyngdepunkt af pendul	0,305 m
M	Opstillingens samlede masse	1,243 kg
M_a	Massen af det øverste tandhjul	0,105 kg
M_{a_1}	Massen af gearingstandhjulet på pendulstangen	0,220 kg
M_b	Massen af balancestangen	0,100 kg
M_l	Massen af lodderne	0,214 kg
M_m	Massen af motorens rotor	0,230 kg
M_p	Massen af pendulstangen	0,158 kg
n	Gearingskonstant	25
p	Indført konstant	4,619
φ	Vinkel af pendul i forhold til lodret	
$\dot{\varphi}$	Vinkelhastighed af pendul i forhold til lodret	
$\ddot{\varphi}$	Vinkelacceleration af pendul i forhold til lodret	
R	Motorens indre modstand	16,77 Ω
T_s	Sampling-tid for det diskrete system	0,01 s
τ	Tidskonstant for filtrene	0,5 s
τ_b	Kraftmomentet af balancestangen	
τ_p	Kraftmomentet af pendulstangen	
τ_{pb}	Kraftmomentet fra balancestang på pendulstang	
τ_{pg}	Kraftmomentet fra tyngdekraften på pendulstang	
θ	Vinkel af balancestang i forhold til vandret	
$\dot{\theta}$	Vinkelhastighed af balancestang i ift. vandret	
$\ddot{\theta}$	Vinkelacceleration af balancestang ift. til vandret	
u	Styresignal til motor	

A.2 Unicykelmodellen

δ	Vinkel af pendulstang	
$\dot{\delta}$	Vinkelhastighed af pendulstang	
$\ddot{\delta}$	Vinkelacceleration af pendulstang	
γ	Vinkel af pendulstang i forhold til lodret	
$\dot{\gamma}$	Vinkelhastighed af pendulstang i forhold til lodret	
$\ddot{\gamma}$	Vinkelacceleration af pendulstang i forhold til lodret	
g	Tyngdeaccelerationen	9,82 $\frac{\mathrm{m}}{\mathrm{s}^2}$
H_f	Friktionskraft fra underlag på hjulet	
H_p	Pendulets kraftpåvirkning på hjulet	
I_p	Pendulets inertimoment	0,3132 kg·m^2
I_w	Hjulets inertimoment	0,1055 kg·m^2
K	Samlet vektor for tilbagekoblingskonstanterne	
$k1$	Tilbagekoblingskonstant fra δ	
$k2$	Tilbagekoblingskonstant fra $\dot{\delta}$	
$k3$	Tilbagekoblingskonstant fra y	
$k4$	Tilbagekoblingskonstant fra $\dot{y}$	
l	Længde fra hjulets centrum til pendulets massemidtpunkt	
M_p	Massen af pendulstangen	1,243 kg
M_w	Massen af hjulet	2,255 kg
ω	Vinkel af hjulet	
$\dot{\omega}$	Vinkelhastighed af hjulet	
$\ddot{\omega}$	Vinkelacceleration af hjulet	
P_p	Pendulets kraftpåvirkning af pendulet	
R	Radius af hjulet	
τ_w	Kraftmomentet fra motor på hjul	
y	Position af cyklen	
$\dot{y}$	Hastighed af cyklen	
$\ddot{y}$	Acceleration af cyklen	

Bilag B

Komponentliste

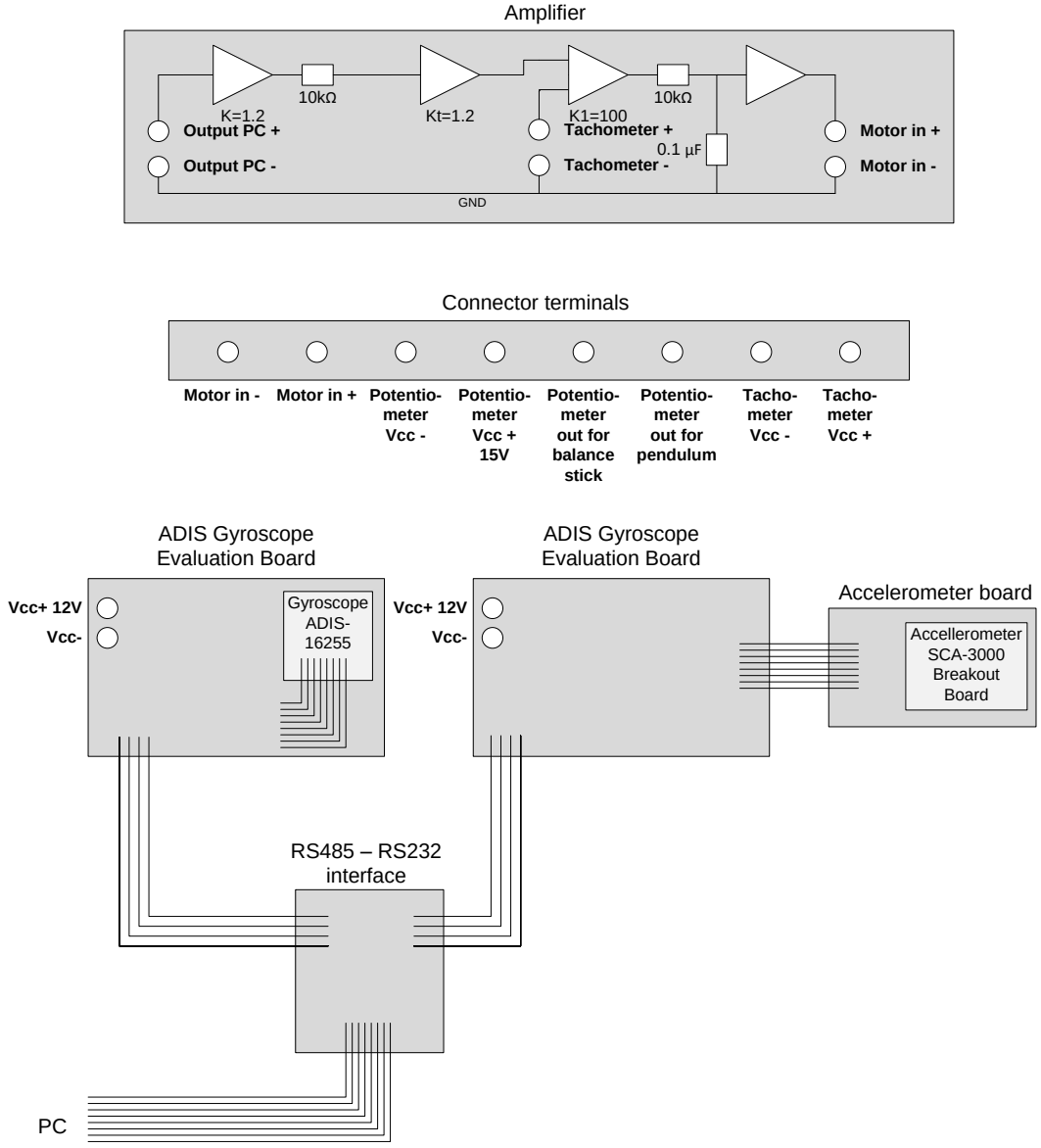
Dette er en liste over de benyttede komponenter

Accelerometer	SCA3000-D01 Breakout Board fra Sparkfun
Forstærker	DTU's servoforstærker SL308A/2
Gyroskop	ADIS-16255 fra Analog Devices
Motor	T-0716A DC torque motor fra Inland Motor Corporation
Potentiometer til vinklen α	6187 R10K L1.0 fra BI Technologies
Potentiometer til vinklen φ	6538S-1-103 fra Bourns Inc.
Tachometer	SA-740A-7 fra Servo Tek

Bilag C

Tilkobling af linedanseropstillingen

Tilkoblingen af linedanseren kan ses af diagramet på næste side.



Figur C.1: Diagram over tilkoblingen af systemet

Bilag D

Anvendelse af gyroskop og accelerometer

Et gyroskop og et accelerometer er benyttet som sensorer til projektet. Begge virker som slave i en master-slaveopsætning med en microcontroller, ATMEGA8, som master. Den kommunikerer med enhederne gennem SPI (Serial Peripheral Interface). Koden til ATMEGA8 microcontrollerne for gyroskopet og for accelerometeret er inkluderet på den vedlagte CD.

D.1 Gyroskop

Gyroskopet benyttet til projektet er et ADIS-16255 fra Analog Devices. Det er et gyroskop med en enkelt akse, der kan indstilles til at måle i intervallet $\pm 80°/s$, $\pm 160°/s$ eller $\pm 320°/s$ med 14 bits digital output. DTU havde mange af disse monteret på et PCB-print med interface til en ATMEGA8 microcontroller, som var beregnet til at sætte i serie med hinanden, for på den måde at få flere akser til rådighed. Disse Triple ADIS Gyroscope Board kører med en ekstern klokke på 14,7456 MHz. De er lavet af Anders Billesø Beck fra DTU i 2008. Der hører også kildekode med, som er benyttet og redigeret til projektet.

D.1.1 Selvhold og kvantisering

Som beskrevet i afsnit 4.2, er det nødvendigt i simuleringen af sin model, at inddrage selvhold og kvantisering som begrænsninger for modellen, da digital databehandling altid har sine begrænsninger.
For gyroskopet er selvholdet ved et signal uden for måleområdet $\pm 320°/s$. Kvantiseringen for gyroskopet udregnes, som den opløsning, der er, for intervallet fra $-320°/s$ til $320°/s$ med 14-bit målinger. Da sensoren i gyroskopet ifølge specifikationerne for den kan måle op til $\pm 600°/s$, skal opløsningen beregnes ud fra dette interval. Opløsningen er derved $1200°/s / 16384\,\text{kvanter} = 0,0732\frac{°/s}{\text{kvant}}$. Dette er også den værdi, der kan opslås i databladet for gyroskopet. Man skal bare være opmærksom på, at i modellen regnes der med rad/s, og man derfor skal omregne til denne enhed.

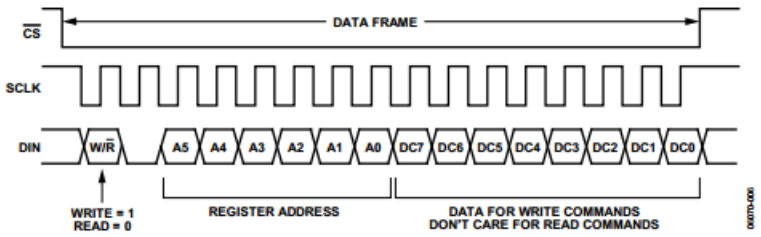
D.1.2 Implementering af gyroskop

Gyroskopet fungerer som slave i en master-slave opsætning med en ATMEGA8 microcontroller som master. Disse er forbundet gennem SPI-interfacet med SCK (source clock), CS (chip select), MISO (master-in, slave-out) og MOSI (master-out, slave-in). Derudover er der også tilkoblet et RESET-signal, et PWR-signal (power) og et GND-signal (ground).

Microcontrolleren opsættes gennem SPCR-registret ("SPI Control Register") til at aktivere SPI med mest betydende bit sendt først, microcontrolleren kørende i master-mode, kloksignalet er højt ved "idle", hvor sampling sker ved den faldende flanke (falling edge), og klokken er præskaleret til 1/16 af systemklokken, dvs. 9,216 MHz. For at opnå dette skrives der til SPCR-registret hexværdien 0x5d. Programmet er opsat til at køre i perioder af 1 ms gennem et overflow interrupt på microcontrollerens "timer 2".

Til projektet er benyttet det højeste måleinterval på $\pm 320°/s$, da dette var en del af den oprindelige kode. Det er dog ikke nødvendigt med så stort et interval, da vinkelhastighederne helst ikke skulle blive så store. Man kan nøjes med $\pm 80°/s$ for at få en større nøjagtighed. Programmet virkede rigtig godt ved afprøvning af det høje interval, så dette blev fastholdt, så der ikke skulle ændres for meget i den originalle kode, når nu det virkede.

SPI-rammen for gyroskopet ADIS-16255 er vist i figur D.1.



Figur D.1: SPI ramme for gyroskopet fra Analog Device's manual

http://www.analog.com/static/imported-files/data_sheets/ADIS16250_16255.pdf

I koden fra Anders Beck er det READRATE-funktionen, der er den væsentlige funktion ved måling af vinkelhastigheden. Denne går ind og læser værdierne for vinkelhastigheden i registrene 0x04 og 0x05. Dette kan ses i figur D.2 over registrene.

D.2 Accelerometer

Det anvendte accelerometer er et SCA3000-D01 Breakout Board fra Sparkfun indeholdende et treakse accelerometer, der har et måleområde på $\pm 2g$. Denne skal have 3,3 volts spænding. Til forsyningsspændingen er en spændingsregulator, der kan regulere spændingen til 3,3 volt ved en indgangsspænding på alt fra 3,3 volt til 10 volt.

Name	Function	Address	Resolution (Bits)	Data Format	Scale Factor (per LSB)
ENDURANCE	Flash Memory Write Counter	0x01, 0x00	16	Binary	1 count
SUPPLY_OUT	Power Supply Data	0x03, 0x02	12	Binary	1.8315 mV
GYRO_OUT	Gyroscope Data	0x05, 0x04	14	Twos Complement	0.07326°/sec[1]
AUX_ADC	Auxiliary Analog Input Data	0x0B, 0x0A	12	Binary	0.6105 mV
TEMP_OUT	Sensor Temperature Data	0x0D, 0x0C	12	Twos Complement	0.1453°C
ANGL_OUT	Angle Output	0x0F, 0x0E	14	Binary	0.03663°

[1] Assumes that the scaling is set to 320°/sec.

Figur D.2: Dataregistre for gyroskopet fra Analog Device's manual

http://www.analog.com/static/imported-files/data_sheets/ADIS16250_16255.pdf

D.2.1 Selvhold og kvantisering

Ligesom for gyroskopet har accelerometeret også begrænsninger i form af selvhold og kvantisering. Selvholdet for accelerationen sættes over $2g$ og under -$2g$, da det anvendte accelerometer har et måleområde mellem -$2g$ og $2g$. Kvantiseringen for accelerometeret beregnes ved at intervallet fra -$2g$ til $2g$, altså $4g$, skal deles med de mulige binære værdier. Accelerometeret måler med 12 bits nøjagtighed, som i decimaltal er 4096. Kvantiseringen bliver derved $4g/4096\text{kvanter} = 0,000977\frac{g}{\text{kvant}}$.

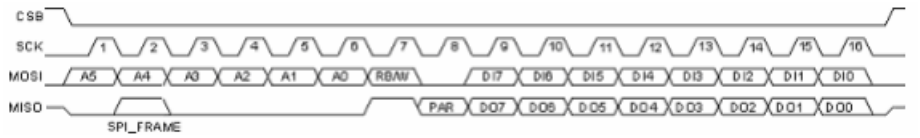
D.2.2 Implementering af accelerometer

Opsætningen af ATMEGA microcontrolleren, som snakker med accelerometeret, er baseret på samme kode som for gyroskopet.

Accelerometeret kommunikerer med en ATMEGA8 microcontroller gennem SPI, der gennem SPCR registret er opsat til at sende data med mest betydende bit først, microcontrolleren er sat som master, kloksignalet er lavt ved idle, hvor signalsampling foregår ved klokkens stigende flanke (rising edge), og til sidst er SPI-frekvensen præskaleret til 1/128 af systemklokken, dvs. 115,2 kHz. For at opnå dette skrives der til SPCR-registret hexværdien 0x57. Programmet er opsat til at køre i perioder af 1 ms gennem et overflow interrupt på microcontrollerens "timer 2".

Accelerometeret er fra fabrikken opsat til at starte i tilstanden "normal measurement mode", så den direkte kan begynde at sende måledata. Måledataen tilgås gennem MOSI'en (Master Out, Slave In) ved at sende adressen for det register, der ønskes læst fra samt en read/write bit. Dette er for den første byte. Ved læsning er anden byte ligegyldig. Denne SPI-ramme kan ses af figur D.3.

Gennem MISO'en (Master In, Slave Out) kan herefter læses data fra det ønskede register. Det er registrene fra 0x04 til 0x09, der indeholder henholdsvis data for X_LSB til Z_MSB. Se figur D.4.



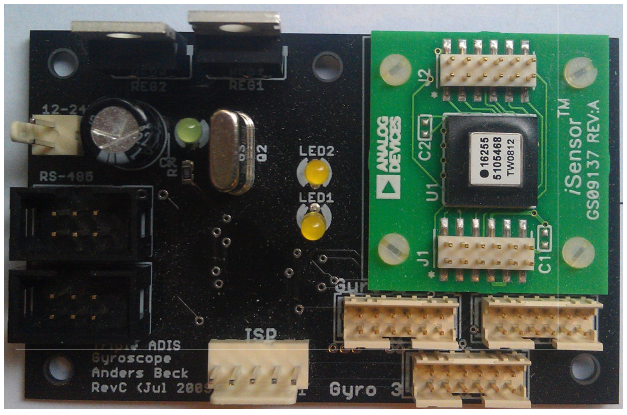
Figur D.3: SPI ramme for accelerometeret fra Sparkfuns manual

http://www.sparkfun.com/datasheets/Sensors/Accelerometer/SCA3000-Manual.pdf

Addr.	Name	Description	Mode (R, W, RW, IA)	Reg. type	Locked
00h	REVID	ASIC revision ID number	R	Conf	
01h		Reserved			-
02h	STATUS	Status register	R	Conf	
03h		Reserved			-
04h	X_LSB	X-axis LSB frame	R	Output	
05h	X_MSB	X-axis MSB frame	R	Output	
06h	Y_LSB	Y-axis LSB frame	R	Output	
07h	Y_MSB	Y-axis MSB frame	R	Output	
08h	Z_LSB	Z-axis LSB frame	R	Output	
09h	Z_MSB	Z-axis MSB frame	R	Output	

Figur D.4: Dataregistre for accelerometer fra Sparkfuns manual

http://www.sparkfun.com/datasheets/Sensors/Accelerometer/SCA3000-Manual.pdf



Figur D.5: Triple ADIS Gyroscope Board af Anders Beck, 2008

Bilag E

Filstruktur for den vedlagte CD

På den vedlagte CD er kildekoden til programmet for projektet, kildekode til programmerne på de to ATMEGA8 microcontrollere, MATLAB-kode, Simulinkmodeller og måleresultater fra hovedprogrammet i følgende mapper:

Kode

I kode-mappen findes kildekoden til de forskellige programmer i undermapperne

Accelerometer

Her findes kildekoden til programmeringen af ATMEGA8 microcontrolleren, der interfacer med accelerometeret. Dette indbefatter en main.c-fil. Derudover er C- og headerfilen acc.c og acc.h, der gennem SPI'en sørger for at sende og modtage de forskellige data, og comm.c og comm.c, der opretter SPI-kommunikationen mellem microcontrolleren og accelerometeret. Desuden er også en Makefile, der sørger for at filerne linkes og bygges rigtigt sammen til programmmet.

Gyroskop

Her findes kildekoden til programmeringen af ATMEGA8 microcontrolleren, der interfacer med gyroskopet. Denne indeholder C-filerne main.c, gyro.c, comm.c og Makefile samt de tilhørende headerfiler, der på tilsvarende vis virker som for accelerometeret.

Gyroskop original

Denne indeholder samme filer som mappen Gyroskop. Dette er det oprindelige program lavet af Anders Billesø Bech fra DTU i 2008.

TRW

Denne mappe indeholder alle C-filerne samt deres headerfiler og Makefile til hovedprogrammet. Dette er de filer, hvor reguleringen af systemet indgår. Disse filers indhold er vist i bilag F. Dette er også mappen, hvor målelogfiler gemmes.

Configuration
Denne undermappe i TRW-mappen indeholder konfigurationsfiler til programmet. Dette er regulatorkonstanter, filterkonstanter og tilbagekoblingskonstanter.

Functions
Denne undermappe i TRW-mappen indeholder filerne globalfunc.c og globalfunc.h beregnet til serial/socket RX/TX kommunikation. Disse hører ligesom den oprindelige kode til gyroskoperne lavet af Anders Billesø Bech i 2010.

Målinger

Denne mappe indeholder en række målinger i logfiler i undermapper, hvor både potentiometer-, accelerometer- og gyroskopmålinger indgår. Der er også logfiler af nogle af de interne variabler, hvor bl.a. indgangssignalet til motoren er gemt.

Matlab

Denne mappe indeholder MATLAB filer. Dette er både m-scripts og data-filer. Her er model.m hovedfilen.

Configuration
Denne undermappe indeholder de anvendte Simulinkmodeller.

Video

Denne mappe indeholder forskellige videoklip, der viser linedanserrobotten.

Bilag F

Kildekode

F.1 TRW.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <math.h>
5
6  #include <rtai_shm.h>
7  #include <sched.h>
8  #include <pthread.h>
9  #include <rtai_sem.h>
10 #include <comedilib.h>
11 #include <unistd.h> //for read and write functions
12
13 #include "trwcontroller.h"
14
15 #include "Functions/globalfunc.h"
16
17 #define ACC_G 9.82
18 #define DEG2RAD 0.017453 //pi/180
19 #define RAD2DEG 57.2958  //180/pi
20
21 #define PHI_CONV 0.0011 //From quant to SI
22 #define ALPHA_CONV 0.00096 //From quant to SI
23 #define ACC_CONV 0.000935//Measured on accelerometer
24 #define GYRO_CONV 0.07326//0.0390625//0.009766
25 #define VOLT2ADC 0.00488 //Voltage to converter
26 #define REF_MATCH 0.6208 //K_t*n
27
28 #define LOG 100000
29
30
31 unsigned int run=1;
32
```

```
33 double logPhi[LOG];
34 double logAlpha[LOG];
35 double logObsPhi[LOG];
36 double logObsDPhi[LOG];
37 double logAcc[LOG];
38 double logGyro[LOG];
39 double logU[LOG];
40
41 int l=0;
42
43 SEM* sem;       //Global semaphore
44 comedi_t* comedi; //Global comedi
45
46 void *userInterface(void *param);
47 void *balanceControl(void *param);
48
49 int main(){
50   int check;
51
52   RT_TASK* task;
53   pthread_t threadUserInterface;
54   pthread_t threadBalanceControl;
55
56   struct sched_param sched;
57
58   rt_allow_nonroot_hrt();
59   mlockall(MCL_CURRENT | MCL_FUTURE);
60
61   /* Setup scheduler */
62   sched.sched_priority=sched_get_priority_max(SCHED_FIFO)
       ;
63   if(sched_setscheduler(0, SCHED_FIFO, &sched) == -1){
64     printf("Error setting scheduler\n");
65   }
66
67   /* Setup comedi used for input/output */
68   comedi = comedi_open("/dev/comedi2");
69   if (comedi == NULL){
70     printf("Error setting comedi\n");
71   }
72
73   /* Setup semaphore */
74   sem = rt_typed_sem_init(nam2num("trwsem"), 1, CNT_SEM);
75
76   /* Create task for main */
77   task = rt_task_init(nam2num("trwmain"), 1, 0, 0);    //
       high priority
78   if (task == NULL){
79     printf("Error with creating task\n");
80   }
```

```
81
82    /* Create thread for userInterface function */
83    check = pthread_create(&threadUserInterface, NULL,
          userInterface,NULL);
84    if (check != 0){
85      printf("Error creating thread\n");
86    }
87
88    /* Create thread for balanceControl function */
89    check = pthread_create(&threadBalanceControl, NULL,
          balanceControl,NULL);
90    if (check != 0){
91      printf("Error creating thread\n");
92    }
93
94
95    /* Now main waits for threads to close and then shuts
          down the program */
96    pthread_join(threadBalanceControl, NULL);// collect the
          threads again
97
98    rt_sem_delete(sem);
99    comedi_close(comedi);
100   rt_task_delete(task);
101
102   return 0;
103 }
104
105 void *userInterface(void *param){
106   /* Create task */
107   RT_TASK* task;
108   task = rt_task_init (nam2num("trwui"), 5, 0, 0);
109   if (task==NULL){
110     printf("Task not correct");
111   }
112   printf("Press enter to quit\n");
113   getchar();
114   run=0;
115
116   rt_task_delete(task);
117   pthread_exit(NULL);
118
119 }
120
121 void *balanceControl(void *param){
122   long long int period = 10000000;
123
124   int t=0;
125   int i;
126
```

```
127     int input;
128     int inPhi=0;
129     int inAcceleration=0;
130     int inDPhi=0;
131     int inAlpha=0;
132     double obsPhi=0;
133     double obsDPhi=0;
134 
135     //int inPhi0=0;
136     double obsPhi0=0;
137 
138     double acceleration=0;
139     double dPhi=0;
140 
141     //int offsetPhi = -11;
142     int offsetAlpha = 60;
143 
144     double e=0;
145     double u=0;
146 
147     double b0 = 0.005;
148     double b1 = 0.005;
149     double a1 = -1;
150 
151     double k1,k2,k3,k4,k5;
152     int n = 25;
153 
154     double tau = 0.5;
155 
156     double b0phi;
157     double b1phi;
158     double a0phi;
159     double a1phi;
160     double b0dphi;
161     double b1dphi;
162     double a0dphi;
163     double a1dphi;
164 
165     double dummy;
166 
167     char regStart=0;
168 
169     lsampl_t ad;
170     lsampl_t da;
171 
172     int fd;
173     ssize_t txLen;
174     unsigned char sendbuf[32];
175     unsigned char recvbuf[32];
176 
```

```
177     /*Read constants from constants.txt file*/
178     FILE *fconst;
179     if((fconst = fopen("Configuration/constants.txt", "r"))
            ==NULL){
180       printf("Error opening \"constants.txt\"\n");
181     }
182     fscanf(fconst, "%lf%lf%lf%lf%lf",&k1,&k2,&k3,&k4,&k5);
183     //fscanf(fconst, "%lf%lf%lf%lf",&k1,&k2,&k3,&k4);
184     fclose(fconst);
185
186     /*Read filter constants from files*/
187     if((fconst = fopen("Configuration/filterphi.txt", "r"))
            ==NULL){
188       printf("Error opening \"filterphi.txt\"\n");
189     }
190     fscanf(fconst, "%lf%lf%lf%lf",&b0phi,&b1phi,&a0phi,&
            a1phi);
191     fclose(fconst);
192
193     if((fconst = fopen("Configuration/filterdphi.txt", "r")
            )==NULL){
194       printf("Error opening \"filterdphi.txt\"\n");
195     }
196     fscanf(fconst, "%lf%lf%lf%lf",&b0dphi,&b1dphi,&a0dphi,&
            a1dphi);
197     fclose(fconst);
198
199     printf("k1=%lf k2=%lf k3=%lf k4=%lf k5=%lf\n",k1,k2,k3,
            k4,k5);
200     printf("b0=%lf b1=%lf a1=%lf\n",b0phi,b1phi,a1phi);
201     printf("b0=%lf b1=%lf a1=%lf\n",b0dphi,b1dphi,a1dphi);
202
203     /* Create task */
204     RT_TASK* task;
205     task = rt_task_init (nam2num("trwbal"), 1, 0, 0);
206     if (task==NULL){
207       printf("Task not correct");
208     }
209
210     /* Connect to Gyroscope and Accelerometer unit with
            baudrate 115200 */
211     fd = open("/dev/ttyS0", O_RDWR);
212     if (fd == -1) {
213       printf("Error connecting to unit\n");
214     }
215     if (set_serial(fd,115200) == -1){
216       printf("Error in set_serial\n");
217     }
218
219     regul_initTRW();
```

```
220   rt_task_make_periodic_relative_ns(task,100,period);
221   /*
222   input=comedi_data_read_delayed(comedi,0,0,0,AREF_GROUND
          ,&ad,50000); // para 3 is channel
223   if (input==0){
224     printf("Input failed\n");
225   }
226
227   inPhi0=-((int)ad-2048)+offsetPhi;
228   */
229
230   while(run && inAlpha < 1700 && inAlpha > -1700){
231     /*
232     input=comedi_data_read_delayed(comedi,0,0,0,
            AREF_GROUND,&ad,50000); // para 3 is channel
233     if (input==0){
234       printf("Input failed\n");
235     }
236
237     inPhi=(-((int)ad-2048)+offsetPhi);
238     //acceleration=inPhi*ACC_G*PHI_CONV;
239     */
240
241     input=comedi_data_read_delayed(comedi,0,1,0,
            AREF_GROUND,&ad,50000); // para 3 is channel
242     if (input==0){
243       printf("Input failed\n");
244     }
245
246     inAlpha=(int)ad-2048+offsetAlpha;
247
248     /* Output of gyro */
249     txLen = 2; //Length of transmitted message
250     sendbuf[0] = 0x01; //Length of message
251     sendbuf[1] = 0x2A; //Get rate (2) on device (A)
252     write(fd, sendbuf, txLen);
253
254     // Check if buffer is full
255     do read(fd, recvbuf, 1);
256     while (recvbuf[0] > 31);
257
258     for (i=0; i< recvbuf[0]; i++) {
259       read(fd, &recvbuf[i+1], 1);
260     }
261     inDPhi = (((signed char)(recvbuf[2])<<8)+(unsigned
            char)recvbuf[3]);
262     dPhi=-inDPhi*GYRO_CONV*DEG2RAD;
263
264     //printf("Buffer: %x %x %2x %2x %2x %2x %2x %2x\n ",
            recvbuf[0], recvbuf[1], recvbuf[2],recvbuf[3],
```

```
        recvbuf[4], recvbuf[5], recvbuf[6], recvbuf[7]);
265     // 0 is ID, 1 is command, 2 is x MSB, 3 is x LSB etc.
266
267
268     // Output of ACC
269     sendbuf[0] = 0x01; //Length of message
270     sendbuf[1] = 0x2B; //Get rate (2) on device (B)
271     write(fd, sendbuf, txLen);
272
273     do read(fd, recvbuf, 1);
274     while (recvbuf[0] > 31);
275
276     for (i=0; i< recvbuf[0]; i++) {
277       read(fd, &recvbuf[i+1], 1);
278     }
279
280     inAcceleration = (((signed char)(recvbuf[4])<<8)+(
          unsigned char)recvbuf[5]);
281     acceleration = (inAcceleration-600)*ACC_CONV;
282
283     dummy = acceleration/ACC_G+tau*dPhi;
284     obsPhi = regul_outPhi(dummy,b0phi,b1phi,a1phi);
285     obsDPhi = regul_outDPhi(dummy,b0dphi,b1dphi,a1dphi);
286
287     /* Calculations for e*/
288     //e=(-k1*inPhi*0.0011-k3*inAlpha*ALPHA_CONV); //for
          pot 4 state
289     //e=(-k1*obsPhi-k2*dPhi-k3*inAlpha*ALPHA_CONV); //for
           acc 4 state
290     //e=(-k1*inPhi*0.0011-k3*inAlpha*ALPHA_CONV-k5*u); //
          for pot 5 state
291     e=(-k1*obsPhi-k2*dPhi-k3*inAlpha*ALPHA_CONV-k5*u); //
          for acc 5 state
292
293
294     if(regStart) {
295       /* Calculations for u */
296       //u=(regul_outTRW(e,b0,b1,a1)-k2*inPhi*0.0011-k4*
            inAlpha*ALPHA_CONV+k2*inPhi0*0.0011)*REF_MATCH;
            //for pot 4 state
297       //u=(regul_outTRW(e,b0,b1,a1)-k4*inAlpha*ALPHA_CONV
            )*REF_MATCH;// for acc 4 state
298       //u=(regul_outTRW(e,b0,b1,a1)-k2*inPhi*0.0011-k4*
            inAlpha*ALPHA_CONV*n+k2*inPhi0*0.0011); //for
            pot 5 state
299       u=(regul_outTRW(e,b0,b1,a1)-k4*inAlpha*ALPHA_CONV*n
            ); //for acc 5 state
300     }
301     //Saturation
302     if(u>2047) u=2047;
```

```
303       if(u<-2047) u=-2047;
304
305       //Disturbance
306       /*
307       if(t>1000 && t<1015){
308         u=-1;
309       }
310       if(t>1700 && t<1715){
311         u=1;
312       }
313       */
314
315
316       da=u/VOLT2ADC+2048; // 0 is 2048
317
318       comedi_data_write(comedi,1,0,0,AREF_GROUND,da);
319
320       regul_updateTRW(e,b0,b1,a1);
321       regul_updatePhi(dummy,b0phi,b1phi,a1phi);
322       regul_updateDPhi(dummy,b0dphi,b1dphi,a1dphi);
323
324       if(!regStart && (t%100)==0) {
325         printf("Time for start: %d\n", 3-(t/100));
326       }
327       if(!regStart && (t/100 >= 3)) {
328         regStart = 1;
329         obsPhi0=obsPhi;
330       }
331       t++;
332
333       /* Data logging */
334       rt_sem_wait(sem);
335
336       if(l<LOG){
337         logPhi[l]=inPhi*PHI_CONV;
338         logObsPhi[l]=obsPhi;
339         logObsDPhi[l]=obsDPhi;
340         logAlpha[l]=inAlpha;
341         logAcc[l]=acceleration;
342         logGyro[l]=dPhi;
343         logU[l]=u;
344         l++;
345       }
346
347       rt_sem_signal(sem);
348
349       rt_task_wait_period();
350     }
351
352     /* Write log to file */
```

```
353     if((fconst = fopen("logPhi.txt", "w"))==NULL){
354       printf("Error opening \"logPhi.txt\"\n");
355     }
356     for(l=0;l<LOG;l++){
357       fprintf(fconst, "%lf\n",logPhi[l]);
358     }
359
360     if((fconst = fopen("logAlpha.txt", "w"))==NULL){
361       printf("Error opening \"logAlpha.txt\"\n");
362     }
363     for(l=0;l<LOG;l++){
364       fprintf(fconst, "%lf\n",logAlpha[l]);
365     }
366
367     if((fconst = fopen("logObsPhi.txt", "w"))==NULL){
368       printf("Error opening \"logObsPhi.txt\"\n");
369     }
370     for(l=0;l<LOG;l++){
371       fprintf(fconst, "%lf\n",logObsPhi[l]);
372     }
373
374     if((fconst = fopen("logObsDPhi.txt", "w"))==NULL){
375       printf("Error opening \"logObsDPhi.txt\"\n");
376     }
377     for(l=0;l<LOG;l++){
378       fprintf(fconst, "%lf\n",logObsDPhi[l]);
379     }
380
381     if((fconst = fopen("logAcc.txt", "w"))==NULL){
382       printf("Error opening \"logAcc.txt\"\n");
383     }
384     for(l=0;l<LOG;l++){
385       fprintf(fconst, "%lf\n",logAcc[l]);
386     }
387
388     if((fconst = fopen("logGyro.txt", "w"))==NULL){
389       printf("Error opening \"logGyro.txt\"\n");
390     }
391     for(l=0;l<LOG;l++){
392       fprintf(fconst, "%lf\n",logGyro[l]);
393     }
394
395     if((fconst = fopen("logU.txt", "w"))==NULL){
396       printf("Error opening \"logU.txt\"\n");
397     }
398     for(l=0;l<LOG;l++){
399       fprintf(fconst, "%lf\n",logU[l]);
400     }
401
```

```
402     comedi_data_write(comedi,1,0,0,AREF_GROUND,2048); //
            Zero
403     rt_task_delete(task);
404     pthread_exit(NULL);
405 }
```

F.2 trwcontroller.h

```
1  /*Controller with servo feedback for TRW*/
2
3  /*Initialize state variables*/
4  void regul_initTRW(void);
5
6  /*Controller output to program*/
7  double regul_outTRW(double e, double b0, double b1,
       double a1);
8
9  /*Update state variables*/
10 void regul_updateTRW(double e, double b0, double b1,
       double a1);
11
12 /*Controller output to program*/
13 double regul_outPhi(double e, double b0, double b1,
       double a1);
14
15 /*Update state variables*/
16 void regul_updatePhi(double e, double b0, double b1,
       double a1);
17
18 /*Controller output to program*/
19 double regul_outDPhi(double e, double b0, double b1,
       double a1);
20
21 /*Update state variables*/
22 void regul_updateDPhi(double e, double b0, double b1,
       double a1);
```

F.3 trwcontroller.c

```
1  #include "trwcontroller.h"
2
3  static struct{
4    double u;
5    double f0;
6    double dummy_a;
7    double dummy_b;
8  }controlStatesTRW;
9
10 static struct{
11   double u;
12   double f0;
13   double dummy_a;
14   double dummy_b;
```

```
15 }controlStatesPhi;
16
17 static struct{
18   double u;
19   double f0;
20   double dummy_a;
21   double dummy_b;
22 }controlStatesDPhi;
23
24 void regul_initTRW(void){
25   controlStatesTRW.u=0;
26   controlStatesTRW.f0=0;
27   controlStatesTRW.dummy_a=0;
28   controlStatesTRW.dummy_b=0;
29   controlStatesPhi.u=0;
30   controlStatesPhi.f0=0;
31   controlStatesPhi.dummy_a=0;
32   controlStatesPhi.dummy_b=0;
33   controlStatesDPhi.u=0;
34   controlStatesDPhi.f0=0;
35   controlStatesDPhi.dummy_a=0;
36   controlStatesDPhi.dummy_b=0;
37 }
38
39 double regul_outTRW(double e, double b0, double b1,
     double a1){
40   controlStatesTRW.f0 = e-controlStatesTRW.dummy_a;
41   controlStatesTRW.u = b0*controlStatesTRW.f0+
       controlStatesTRW.dummy_b;
42   return controlStatesTRW.u;
43 }
44
45 void regul_updateTRW(double e, double b0, double b1,
     double a1){ /*update state variables*/
46   controlStatesTRW.dummy_a = a1*controlStatesTRW.f0;
47   controlStatesTRW.dummy_b = b1*controlStatesTRW.f0;
48 }
49
50 double regul_outPhi(double e, double b0, double b1,
     double a1){ /*calculate output*/
51   controlStatesPhi.f0 = e-controlStatesPhi.dummy_a;
52   controlStatesPhi.u = b0*controlStatesPhi.f0+
       controlStatesPhi.dummy_b;
53   return controlStatesPhi.u;
54 }
55
56 void regul_updatePhi(double e, double b0, double b1,
     double a1){ /*update state variables*/
57   controlStatesPhi.dummy_a = a1*controlStatesPhi.f0;
58   controlStatesPhi.dummy_b = b1*controlStatesPhi.f0;
```

```
59 }
60
61 double regul_outDPhi(double e, double b0, double b1,
       double a1){ /*calculate output*/
62   controlStatesDPhi.f0 = e-controlStatesDPhi.dummy_a;
63   controlStatesDPhi.u = b0*controlStatesDPhi.f0+
         controlStatesDPhi.dummy_b;
64   return controlStatesDPhi.u;
65 }
66
67 void regul_updateDPhi(double e, double b0, double b1,
       double a1){ /*update state variables*/
68   controlStatesDPhi.dummy_a = a1*controlStatesDPhi.f0;
69   controlStatesDPhi.dummy_b = b1*controlStatesDPhi.f0;
70 }
```