ISSN 0280–5316
ISRN LUTFD2/TFRT--5493--SE

Implementering av en direkt adaptiv regulator för industrirobotar

Niclas Olsson
Pär Larsson

Institutionen för Reglerteknik
Lunds Tekniska Högskola
December 1993

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
December 1993

Document Number
ISRN LUTFD2/TFRT--5493--SE

Author(s)
Niclas Olsson and Pär Larsson

Supervisor
Rolf Johansson and Klas Nilsson

Sponsoring organisation

Title and subtitle
Implementering av en direkt adaptiv regulator för industrirobotar. (Implementation of a direct adaptive controller for an industrial robot).

Abstract

This report contains results from simulations and experiments for a direct adaptive controller specially developed for robot manipulators.

SIMNON was used for the simulations and the controller was implemented in MOdula-2 and then tested on an ABB IRB L6/2 industrial robot.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280–5316

ISBN

Language
Swedish

Number of pages
63

Recipient's notes

Security classification

Innehållsförteckning

1. Inledning

Grunden för den här rapporten utgörs av ett examensarbete i adaptiv robotreglering, som utfördes vid Institutionen för Reglerteknik i Lund sommaren och hösten 1993.
Inledningsvis, i avsnitt 2, behandlas en del grundläggande teori om kinematik och robotdynamik. I avsnitt 3 tillämpas sedan denna teori på en ASEA IRB L6/2 industrirobot, och i avsnitt 4 presenteras den adaptiva regleralgoritm, som vi har implementerat. Resultaten från våra simuleringar och experiment redovisas därefter i avsnitt 5, och avslutningsvis ges i avsnitt 6 en sammanfattning med de viktigaste slutsatserna.

2. Kinematik och dynamik

I det här avsnittet skall ges en generell beskrivning av kinematik och robotdynamik för en robot med ett godtyckligt antal leder. Roboten antas då bestå av n stycken sammankopplade länkar, vilka i olika stor utsträckning kan röra sig oberoende av varandra, där varje enskild länk betraktas som en stel kropp. Den följande teorin är hämtad från Fowles' och Craigs böcker (se referenser). En beskrivning av ovanstående system på tillståndsform ges av

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau$$

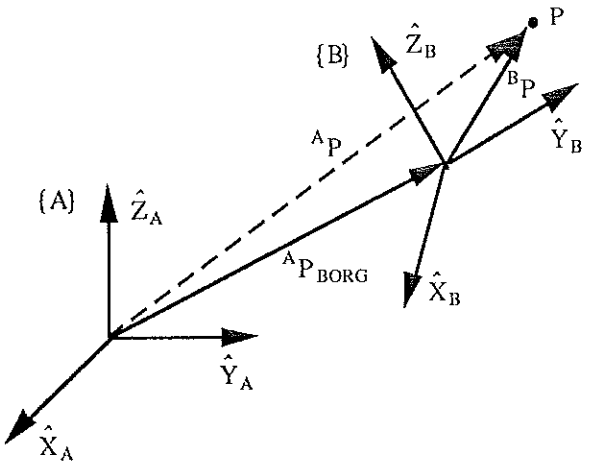
där q, $\dot{q}$ och $\ddot{q}$ är $n \times 1$-vektorer, $M(q)$ är en symmetrisk, positivt definit tröghetsmatris med dimensionerna $n \times n$, $C(q,\dot{q})$ är en $n \times n$-matris, vilken representerar de centrifugal- och Coriolis-krafter som verkar på roboten, $G(q)$ är en $n \times 1$-vektor som motsvarar gravitationskrafterna samt där τ är en $n \times 1$-vektor som representerar moment eller kraft som påläggs roboten.
Denna modell innehåller alltså de okända variablerna, q_k, $k = 1, \ldots, n$, samt deras första- respektive andraderivator. Dessa q_k kallas för generaliserade koordinater och representerar det minimala antalet koordinater som krävs för att specificera konfigurationen för ett givet system. En sådan koordinat kan vara antingen en vinkel eller ett avstånd. Vid härledningen av robotmodellen som följer längre ned motsvarar q_k en vinkel, d.v.s. vinkelutslaget för varje led refererat till en för varje led bestämd referensvinkel. Detta innebär i sin tur att τ_k, $k = 1, \ldots, n$, motsvarar momentet som påläggs varje led.
Det är viktigt att påpeka att ovanstående dynamiska modell för roboten inte omfattar alla krafter som verkar på en robot, utan endast de som kan härledas ur stelkroppsmekanik. En utvidgning av den dynamiska modellen är att även ta med modellering av friktion, vilket leder till

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) + F(q,\dot{q}) = \tau$$

där $F(q,\dot{q})$ är en $n \times 1$-vektor som representerar friktionen. Denna friktionsmodell kan se olika ut beroende på hur komplicerad man vill göra den, men mer om detta i avsnitt 2.2 om framtagande av den dynamiska modellen för en robot med n stycken leder, som följer efter en ganska kortfattad beskrivning av kinematik i avsnitt 2.1. Därefter införs jacobianen och dess användbarhet beskrivs.

2.1. Kinematik

Kinematik är vetenskapen som behandlar rörelse utan hänsyn tagen till krafterna som förorsakar densamma. Krafterna som krävs för att åstadkomma denna rörelse behandlas ju sedan inom dynamiken, vilket är ett stort vetenskapligt fält.
Inom kinematiken ingår alltså problemet att beskriva position och orientering i rymden för robotens sista länk, där arbetsredskapet sitter. Detta kan göras på olika sätt. Beskrivning i det kartesiska rummet, *eng. Cartesian space*, innebär att position beskrivs med tre koordinater, $x\,y\,z$, samt att orienteringen beskrivs med en rotationsmatris, som anger hur ett till robotlänken (sista länken) fast koordinatsystem är roterat i förhållande till ett i rummet fast referenskoordinatsystem. Ett annat sätt är beskrivning i "ledrummet", *eng. joint space*, vilket innebär att man anger vinkelpositionerna för varje robotled. Ett tredje sätt är att ange vinklarna för de motorer som driver robotledernas rörelser, *eng. actuator space*. Man kan lätt visa att det krävs minst sex oberoende variabler för att beskriva en oberoende position och orientering i rymden, vilket ju innebär att det krävs minst sex oberoende länkar hos en robot för att åstadkomma full rörelsefrihet.
För att ändra representationen från ett rum till ett annat kan man använda sig av jacobianen, en omvandlingsmatris som förklaras senare.
Detta är endast en mycket kortfattad genomgång och för utförligare teori hänvisas till Craig. Nedan följer dock introduceringen av begreppet homogen transform och en beskrivning av hur denna kan tas fram, eftersom den är användbar vid härledningen av den dynamiska modellen.



Figur 2.1: En punkt, P, beskriven i två olika koordinatsystem, {A} och {B}.

I figur 2.1 visas hur en punkt, P, kan beskrivas i två olika koordinatsystem (kartesiska rummet), {A} och {B}, med vektorerna ^AP respektive ^BP. Tidigare nämndes att man kunde ange förhållandet mellan olika koordinatsystem m.h.a. en rotationsmatris. Denna kan utnyttjas på följande sätt när man vill ändra beskrivningen av en punkt från ett koordinatsystem till ett annat:

$$^AP = {}^A_BR\,{}^BP + {}^AP_{BORG}$$

där rotationsmatrisen, ${}^A_BR$ (3×3), anger rotationen av koordinatsystem {B} i förhållande till koordinatsystem {A}, samt där $^AP_{BORG}$ anger origo för koordinatsystem {B} uttryckt i {A}:s koordinater, d.v.s. translationen av koordinatsystem {B}.
Genom att införa en ny matris, ${}^A_BT$ (4×4), kan ovanstående transformation utföras endast med en matrisoperation

$$^AP = {}^A_BT\,{}^BP \qquad (*)$$

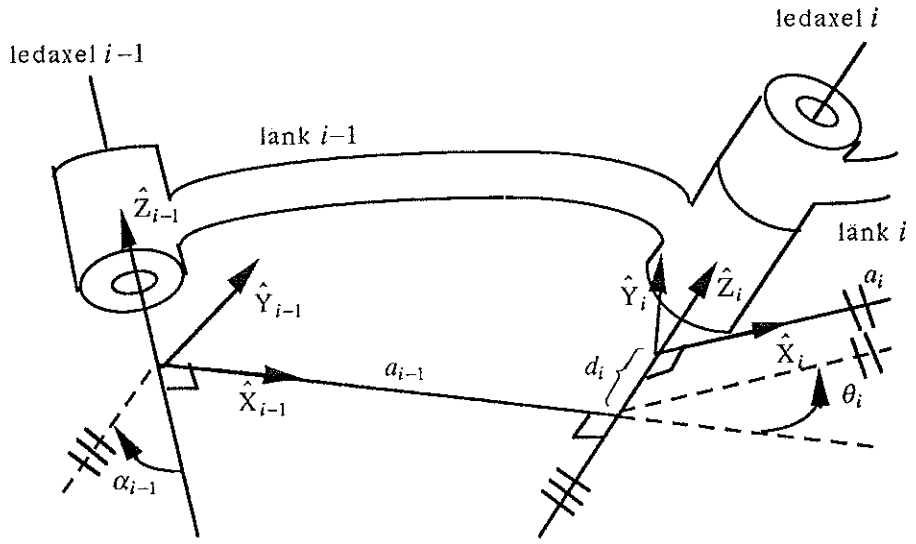
Matrisen, ${}^A_BT$, kallas homogen transform och ser ut enligt följande:

$${}^A_BT = \begin{bmatrix} {}^A_BR & {}^AP_{BORG} \\ 0\ \ 0\ \ 0 & 1 \end{bmatrix}$$

där fjärde raden endast lagts till för att erhålla en kvadratisk matris, som kan inverteras och multipliceras med andra homogena transformer på ett smidigt sätt:

$${}^B_AT = {}^A_BT^{-1},\ {}^A_CT = {}^A_BT\,{}^B_CT$$

där ${}^B_CT$ och ${}^B_AT$ anger transformationen från koordinatsystem {C} till koordinatsystem {B}, respektive från {A} till {B}, (se Craig).
Observera också att man får en etta i fjärde raden i ^AP och ^BP i (*), men den saknar betydelse.
När man skall härleda den dynamiska modellen för roboten är, vilket tidigare nämnts, dessa transformationsmatriser användbara. Man tilldelar då varje robotlänk ett fast koordinatsystem som roterar med robotleden, och translationen och rotationen mellan olika länkar kan då beskrivas med T-matrisen.
Nedan följer en kortfattad beskrivning om hur man väljer dessa koordinatsystem på ett bra sätt. Vidare införs också länkparametrar enligt Denavit-Hartenberg, se figur 2.2, vilka sedan används för att bestämma transformationsmatriserna.



Figur 2.2: Två robotlänkar med ledaxlar, samt införande av länkparametrar enligt Denavit-Hartenberg.

Först inför man då ett referenskoordinatsystem {0}, som man låter vara fixt till robotens bas (första leden). Därefter placerar man nästa koordinatsystem {1} på exakt samma ställe men låter detta rotera med den första leden. Koordinatsystemen väljes så att $\hat{Z}_i$-axeln pekar längs ledaxel i, och $\hat{X}_i$-axeln pekar längs den till $\hat{Z}_i$- och $\hat{Z}_{i+1}$-axlarna gemensamt vinkelräta linjen (se figur 2.2). Skär $\hat{Z}_i$- och $\hat{Z}_{i+1}$-axlarna varandra väljer man $\hat{X}_i$-axeln att vara normal till planet som innehåller dessa båda axlar. Det kan här tilläggas att $\hat{X}_i$-axeln i regel sammanfaller med robotlänk i. $\hat{Y}_i$-axeln väljes sedan så att man erhåller ett koordinatsystem enligt "högerhandsregeln". Man fortsätter sedan att på detta sätt tilldela varje robotlänk ett för länken fixt koordinatsystem. När koordinatsystemen är utplacerade kan man sedan bestämma länkparametrarna som definieras enligt följande:

a_i = avståndet från $\hat{Z}_i$ till $\hat{Z}_{i+1}$ längs $\hat{X}_i$
α_i = vinkeln mellan $\hat{Z}_i$ och $\hat{Z}_{i+1}$ omkring $\hat{X}_i$ ("skruvregeln")
d_i = avståndet från $\hat{X}_{i-1}$ till $\hat{X}_i$ längs $\hat{Z}_i$
θ_i = vinkeln mellan $\hat{X}_{i-1}$ och $\hat{X}_i$ omkring $\hat{Z}_i$ ("skruvregeln"), d.v.s. vinkelutslaget för led i

När dessa parametrar tagits fram kan transformationsmatriserna mellan varje koordinatsystem fås enligt

$$ {}^{i-1}_{i}\mathrm{T} = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & a_{i-1} \\ \sin(\theta_i)\cos(\alpha_{i-1}) & \cos(\theta_i)\cos(\alpha_{i-1}) & -\sin(\alpha_{i-1}) & -\sin(\alpha_{i-1})d_i \\ \sin(\theta_i)\sin(\alpha_{i-1}) & \cos(\theta_i)\sin(\alpha_{i-1}) & \cos(\alpha_{i-1}) & \cos(\alpha_{i-1})d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

För härledning av denna formel, se Craig.

2.2. Dynamisk modell

När det gäller framtagningen av den dynamiska modellen för roboten visas inga härledningar, utan då hänvisas till teorin om stelkroppsmekanik i Fowles' och Craigs böcker samt till Craig vad gäller bestämningen av modellen. Craig redovisar två olika metoder att bestämma modellen då varje robotlänk betraktas som en stel kropp, nämligen Newton-Euler och Lagrange. De skiljer sig åt på det sättet att Newton-Euler-metoden utnyttjar kraftbalans medan Lagrange-metoden är energibaserad. Nedan redovisas de båda algoritmerna under den förutsättningen att alla robotens leder är roterande. Det är här på sin plats att påpeka att siffran uppe till vänster om variabler-

na anger till vilket koordinatsystem de refereras, medan siffran nere till höger anger vilken led det gäller. Dessutom står C för masscentrum.

2.2.1. Newton-Euler-metoden

Newton-Euler-metoden bygger på Newton´s ekvation

$$F = m\dot{v}_C$$

där F är kraften som verkar på en kropps (i det här fallet robotlänkens) masscentrum, m är kroppens totala massa och $\dot{v}_C$ är accelerationen hos kroppens masscentrum, samt på Euler´s ekvation

$$N = {}^{C}I\dot{\omega} + \omega \times {}^{C}I\omega$$

där N är momentet som måste påläggas en kropp för att åstadkomma en roterande rörelse, ω och $\dot{\omega}$ är den roterande kroppens vinkelhastighet respektive vinkelacceleration och ${}^{C}I$ (3×3-matris) är tröghetstensorn för kroppen refererad till koordinatsystem {C}, (se Fowles och Craig). Dessa ekvationer leder så småningom till följande iterativa algoritm:

"Utåtriktade" iterationer: $i: 0 \rightarrow antal\ leder - 1$

$$
\begin{aligned}
{}^{i+1}\omega_{i+1} &= {}^{i+1}_{i}\mathrm{R}\,{}^{i}\omega_i + \dot{\theta}_{i+1}\,{}^{i+1}\hat{\mathrm{Z}}_{i+1} \\
{}^{i+1}\dot{\omega}_{i+1} &= {}^{i+1}_{i}\mathrm{R}\,{}^{i}\dot{\omega}_i + {}^{i+1}_{i}\mathrm{R}\,{}^{i}\omega_i \times \dot{\theta}_{i+1}\,{}^{i+1}\hat{\mathrm{Z}}_{i+1} + \ddot{\theta}_{i+1}\,{}^{i+1}\hat{\mathrm{Z}}_{i+1} \\
{}^{i+1}\dot{v}_{i+1} &= {}^{i+1}_{i}\mathrm{R}({}^{i}\dot{\omega}_i \times {}^{i}\mathrm{P}_{i+1} + {}^{i}\omega_i \times ({}^{i}\omega_i \times {}^{i}\mathrm{P}_{i+1}) + {}^{i}\dot{v}_i) \\
{}^{i+1}\dot{v}_{C_{i+1}} &= {}^{i+1}\dot{\omega}_{i+1} \times {}^{i+1}\mathrm{P}_{C_{i+1}} + {}^{i+1}\omega_{i+1} \times ({}^{i+1}\omega_{i+1} \times {}^{i+1}\mathrm{P}_{C_{i+1}}) + {}^{i+1}\dot{v}_{i+1} \\
{}^{i+1}F_{i+1} &= m_{i+1}\,{}^{i+1}\dot{v}_{C_{i+1}} \\
{}^{i+1}N_{i+1} &= {}^{C_{i+1}}I_{i+1}\,{}^{i+1}\dot{\omega}_{i+1} + {}^{i+1}\omega_{i+1} \times {}^{C_{i+1}}I_{i+1}\,{}^{i+1}\omega_{i+1}
\end{aligned}
$$

"Utåtriktade" iterationer: $i: antal\ leder + 1 \rightarrow 1$

$$
\begin{aligned}
{}^{i}f_i &= {}^{i}_{i+1}\mathrm{R}\,{}^{i+1}f_{i+1} + {}^{i}F_i \\
{}^{i}n_i &= {}^{i}N_i + {}^{i}_{i+1}\mathrm{R}\,{}^{i+1}n_{i+1} + {}^{i}\mathrm{P}_{C_i} \times {}^{i}F_i + {}^{i}\mathrm{P}_{i+1} \times {}^{i}_{i+1}\mathrm{R}\,{}^{i+1}f_{i+1} \\
\tau_i &= {}^{i}n_i^{T}\,{}^{i}\hat{\mathrm{Z}}_i
\end{aligned}
$$

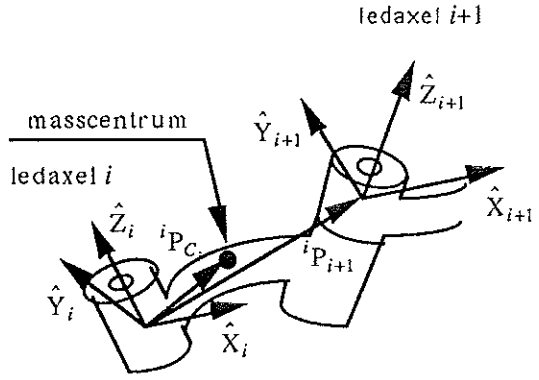
Här erhålles till slut τ_i, d.v.s. momentet som skall påläggas varje led, och ur dessa kan sedan de olika matriserna i den dynamiska modellen identifieras. Detta ger alltså en modell beskriven i *eng. joint space* (under förutsättning att transformationsmatriserna är uttryckta i "ledvinklar"). Några nya beteckningar skall här definieras:

f_i = kraften som länk $i-1$ utövar på länk i
n_i = momentet som länk $i-1$ utövar på länk i

Figur 2.3 visar dessutom definitionen på de i ovanstående algoritm ingående vektorerna ${}^{i}\mathrm{P}_{i+1}$ och ${}^{i}\mathrm{P}_{C_i}$.

För att kunna starta den iterativa algoritmen behövs även några begynnelsevärden:

$${}^{0}\omega_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \quad {}^{0}\dot{\omega}_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$



Figur 2.3: Definition av vektorerna ${}^iP_{i+1}$ och ${}^iP_{C_i}$.

Om roboten rör sig fritt i rummet utan kontakt med omgivningen sätts även

$${}^{k+1}f_{k+1} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \text{ och } {}^{k+1}n_{k+1} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

där k i detta fallet står för antalet leder hos roboten.

Ett enkelt sätt att ta hänsyn till gravitationens inverkan på roboten är att sätta

$${}^0\dot{v}_0 = \mathrm{G}$$

där G är gravitationsvektorn riktad uppåt från robotens bas. Observera här att G:s utseende beror på hur koordinatsystem {0} har definierats.

2.2.2. Lagrange-metoden

En annan metod att bestämma modellen är alltså Lagrange-metoden. Man ställer då upp uttryck för rörelseenergin hos varje led enligt följande:

$$k_i = \frac{1}{2} m_i {}^0v_{C_i}{}^T {}^0v_{C_i} + \frac{1}{2} {}^i\omega_i{}^T {}^{C_i}I_i {}^i\omega_i$$

där k_i här står för rörelseenergin hos led i, ${}^0v_{C_i}$ är den linjära hastigheten för masscentrum hos länk i och beräknas enligt

$${}^0v_{C_i} = {}^0_i\mathrm{R}({}^i_{i-1}\mathrm{R}\,{}^{i-1}v_{C_{i-1}} + {}^i\omega_i \times {}^iP_{C_i})$$

med ${}^0v_{C_0} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$

och där rotationshastigheten, ${}^i\omega_i$, beräknas som förut (se Newton-Euler-metoden). Därefter beräknar man lägesenergin hos varje led:

$$u_i = -m_i {}^0g^T {}^0P_{C_i} + u_{ref}$$

där u_i står för lägesenergin hos led i. ${}^0g$ är gravitationsvektorn, vilken får sitt utseende beroende på hur koordinatsystem {0} är placerat. Observera att g är definierad nedåt. u_{ref} är en konstant som väljs så att minimum av u_i blir noll.

Sedan bildar man den totala rörelse- respektive lägesenergin genom att summera ledernas energier och därefter ställer man upp Lagrange-ekvationen:

$$\mathrm{L}(\theta,\dot{\theta}) = k(\theta,\dot{\theta}) - u(\theta)$$

där $k(\theta,\dot{\theta})$ och $u(\theta)$ är den sammanlagda rörelse- och lägesenergin uttryckt i ledernas vinkelutslag. Genom derivering erhålles sedan momentet för varje robotled som

$$\frac{d}{dt}\frac{\partial \mathrm{L}}{\partial \dot{\theta}_i} - \frac{\partial \mathrm{L}}{\partial \theta_i} = \tau_i$$

eller

$$\frac{d}{dt}\frac{\partial k}{\partial \dot{\theta}_i} - \frac{\partial k}{\partial \theta_i} + \frac{\partial u}{\partial \theta_i} = \tau_i$$

Därefter kan matriserna i den slutgiltiga modellen, som tidigare, identifieras. Man får alltså en modell i *eng. joint space* (under förutsättning att θ_i står för "ledvinklarna"), där de generaliserade koordinaterna motsvarar de olika robotledernas vinkelutslag.

2.2.3. Modellering av friktion

Den framtagna modellen baseras alltså på att varje robotlänk betraktas som en stel kropp. Denna modell kan, vilket även våra experiment senare kommer att visa, modifieras genom att lägga till även en modell av friktionen, $\mathrm{F}(\mathrm{q},\dot{\mathrm{q}})$ enligt ovan. Det kan vara ganska svårt att bestämma en bra sådan modell och de kan dessutom bli ganska komplicerade. Det finns emellertid två enkla friktionsmodeller som kan ge bra resultat och som bara tar hänsyn till robotledernas hastighet, nämligen viskös friktion

$$\tau_{friktion} = v\dot{\theta}$$

där v är en konstant, och Coulomb-friktion

$$\tau_{friktion} = csgn(\dot{\theta})$$

där c är en konstant och alltså annars enbart beror på hastighetens tecken. Dessa modeller kan slås samman till

$$\mathrm{F}(\mathrm{q},\dot{\mathrm{q}}) = \tau_{friktion} = v\dot{\theta} + csgn(\dot{\theta}) = \mathrm{F}(\dot{\mathrm{q}})$$

vilket ger en enkel, men ändå ofta godtagbar, modell av friktionen. Då detta är de enda modeller vi använder i vårt arbete berörs endast dessa i den här rapporten.

2.3. Jacobianen

Det har tidigare nämnts att man kan gå över från beskrivning i ett rum till beskrivning i ett annat m.h.a. jacobianen. Nedan definieras jacobianen och dess funktion demonstreras.

Sätt

$$\mathrm{Y} = \begin{bmatrix} y_1 \\ . \\ . \\ . \\ y_i \end{bmatrix} \quad \text{och} \quad \mathrm{F}(\mathrm{X}) = \begin{bmatrix} f_1(x_1, \ \ldots \ , x_j) \\ . \\ . \\ . \\ f_i(x_1, \ \ldots \ , x_j) \end{bmatrix}$$

Jacobianen definieras då som

$$J(X)=\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_j} \\ \vdots & & \vdots \\ \frac{\partial f_i}{\partial x_1} & \cdots & \frac{\partial f_i}{\partial x_j} \end{bmatrix}$$

och man kan sedan skriva

$$\dot{Y} = J(X)\dot{X} .$$

Jacobianen anger alltså förhållandet mellan hastigheter i olika rum. Det är här mycket viktigt att påpeka att om ena hastigheten är beskriven i det kartesiska rummet måste denna hastighet och jacobianen vara refererad till samma koordinatsystem. Inom robotteorin talar man om jacobianer som relaterar *eng. joint* -hastigheter till kartesiska hastigheter, t.ex.

$${}^0V={}^0J(\theta)\dot{\theta}$$

där θ är en vektor med robotledernas vinkelutslag och V är en vektor med kartesiska hastigheter för robotarmens ände. Man talar också om jacobianer som relaterar *eng. actuator* -hastigheter till *eng. joint* -hastigheter, t.ex.

$$\dot{\theta} = J(S)\dot{S}$$

där S är en vektor med vinkelutslaget hos de motorer som driver robotledernas rörelse. Det är endast detta förhållande vi utnyttjar i vårt arbete eftersom vi aldrig "befinner oss" i det kartesiska rummet. Jacobianen är även mycket användbar för omvandling mellan olika beskrivningar då man befinner sig i kraftplanet. Man kan då använda den till att transformera en dynamisk modell i t.ex. *eng. joint space* till att gälla i *eng. actuator space*, vilket vi utnyttjar, då ju momentet faktiskt tillförs robotledernas motorer. Omvandlingen av modellen sker på följande sätt:

$$\tau_{actuator} = J^T(S)\tau_{joint}$$

och man erhåller då följande dynamiska modell beskriven i *eng. actuator space*:

$$\tau_{actuator} = M^*(S)\ddot{S} + C^*(S,\dot{S})\dot{S} + G^*(S)$$

där

$$M^*(S) = J^T(S)M(\theta(S))J(S)$$
$$C^*(S,\dot{S}) = J^T(S)M(\theta(S))\dot{J}(S) + J^T(S)C(\theta(S),J(S)\dot{S})J(S)$$
$$G^*(S) = J^T(S)G(\theta(S))$$

Vill man även ta med friktionsmodellen erhålles

$$\tau_{actuator} = M^*(S)\ddot{S} + C^*(S,\dot{S})\dot{S} + G^*(S) + F^*(S,\dot{S})$$

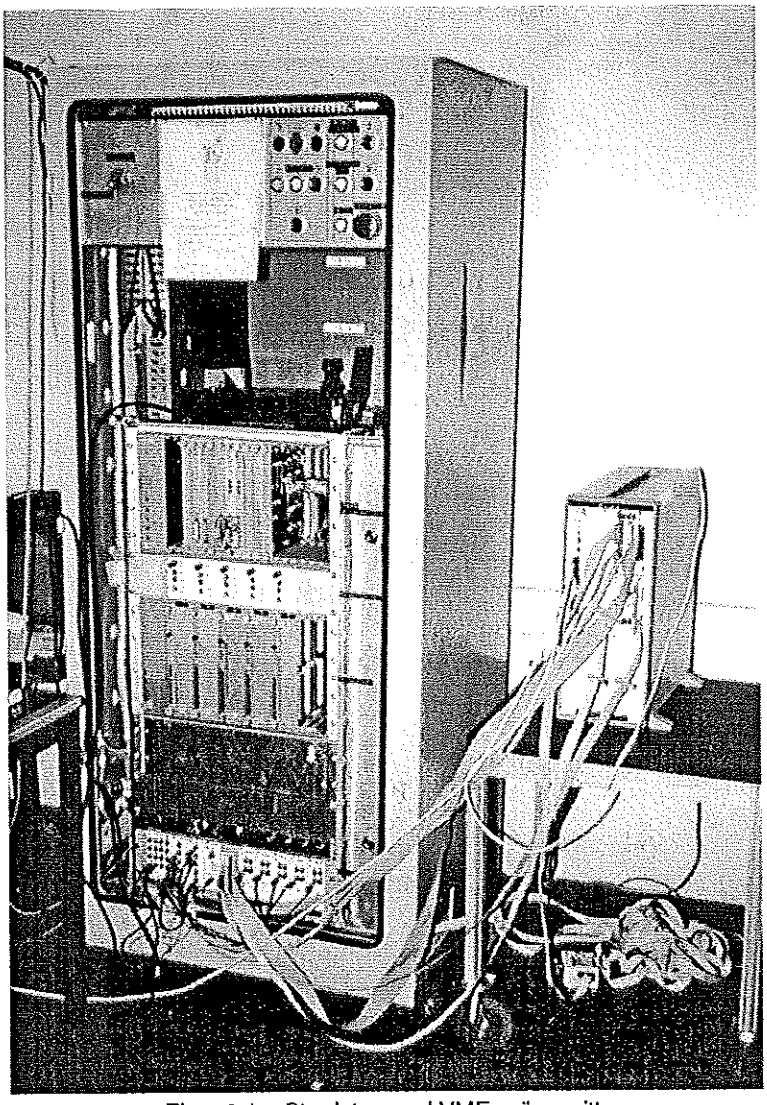
där

$$F^*(S,\dot{S}) = J^T(S)F(\theta(S),J(S)\dot{S})$$

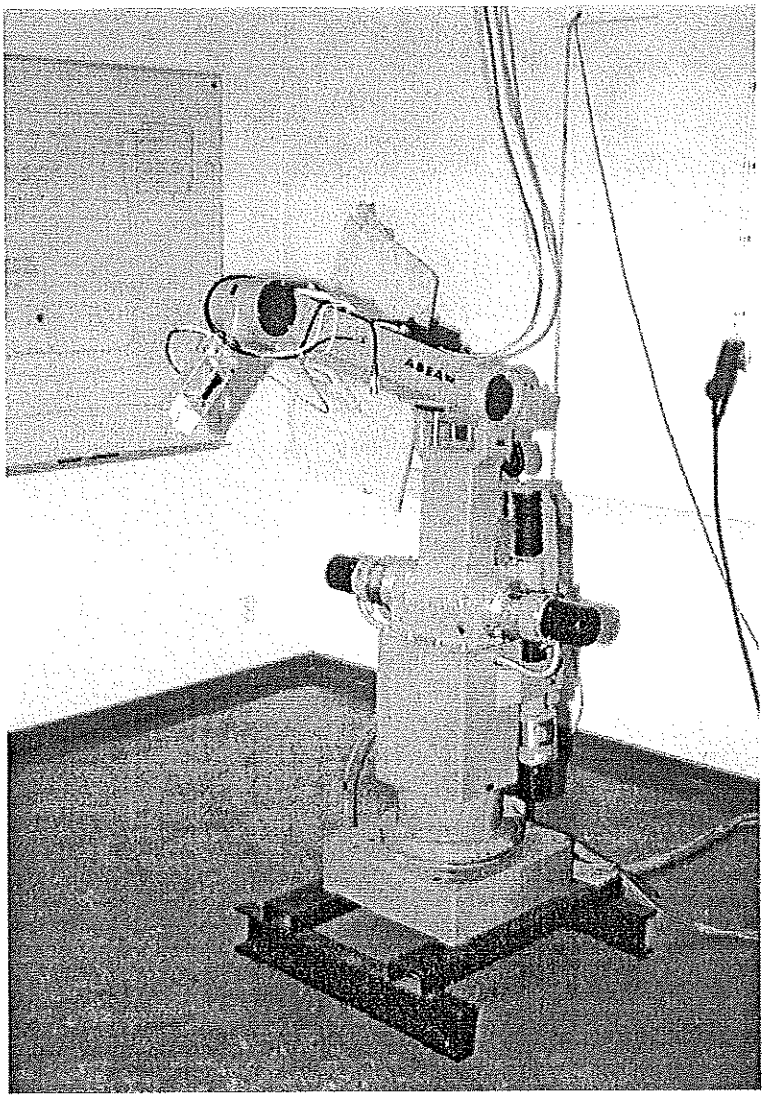
Genom att byta alla S mot q i ekvationerna ovan får man en modell uttryckt i de generaliserade koordinaterna q_k, som här motsvarar vinkelutslaget för robotledernas drivmotorer.

Härmed har de viktigaste bitarna berörts, och för mer ingående teoretisk bakgrund hänvisas till Craig. I avsnitt 5 kommer sedan denna robotteori att appliceras på en industrirobot från ASEA (numera ABB).

3. ASEA IRB L6/2 industrirobot.

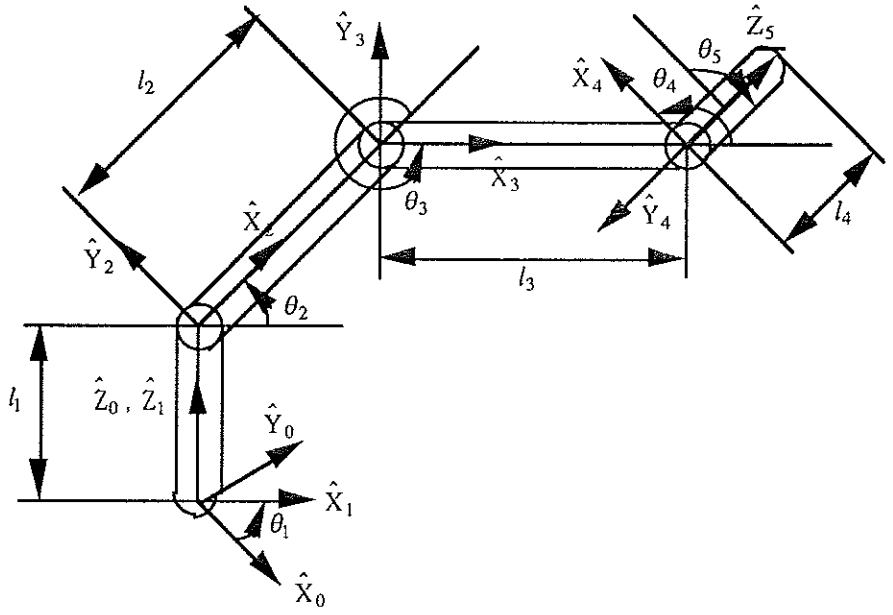


Figur 3.1a: Styrdator med VME-gränssnitt.



Figur 3.1b: ASEA IRB L6/2 industrirobot

I vårt arbete använde vi oss alltså av en industrirobot från ASEA (numera ABB) med beteckningen IRB L6/2. Den har fem frihetsgrader, d.v.s. den består av fem leder, vilka alla är roterande, varför vi kan utnyttja tidigare genomgångna teori för att bestämma den dynamiska modellen. Det nämndes tidigare att man behövde sex frihetsgrader för full rörelsefrihet i rymden. Här uppkommer alltså problemet att bestämma de optimala "ledvinklarna" för minsta möjliga positions- och orienteringsfel i det kartesiska rummet. Detta är emellertid ett komplicerat problem och berörs ej vidare i denna rapport, eftersom vi,vilket tidigare nämnts, aldrig "befinner oss" i detta rummet. Vi kommer nämligen alltid att ge roboten börvärden i *eng. actuator space*, d.v.s. i drivmotorernas vinkelutslag.
I detta avsnitt visas hur T-matriserna kan tas fram, samt hur förhållandet mellan robotledernas vinkelutslag och drivmotorernas vinklar ser ut för robotens fem leder. Även jacobianen och dess tidsderivata tas fram för hela roboten. Däremot bestäms modellen för ett bestämt antal leder i avsnittet om simulering och experiment.
För att kunna ta fram T-matriserna placerade vi först ut koordinatsystem för varje led enligt föregående avsnitt. I figur 3.2. visas en schematisk bild över dessa samt hur "ledvinklarna", θ_i, (i *eng. joint space*) är definierade. När vi i fortsättningen berör robotledernas vinkelutslag kommer denna definition att gälla även om andra, bl.a. ABB, har en annan definition.



$\hat{Y}_1$ riktad in i pappret

$\hat{Z}_2$, $\hat{Z}_3$ och $\hat{Z}_4$ riktade ut ur pappret

$\hat{X}_5$ och $\hat{Y}_5$ ej utritade, roterar med θ_5 runt $\hat{Z}_5$

Figur 3.2.: Schematisk bild över roboten med koordinatsystem och definition av "ledvinklar" utritade.

Därefter bestämde vi de olika länkparametrarna för robotens alla fem leder, se tabell 3.1., och sedan kunde dessa utnyttjas för att beräkna T-matriserna enligt den förut beskrivna formeln. Nedan redovisas också alla dessa transformationsmatriser.

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	90°	0	0	θ_2
3	0	l_2	0	θ_3
4	0	l_3	0	θ_4
5	90°	0	0	θ_5

Tabell 3.1.: Länkparametrarna för robotens fem länkar.

Transformationsmatriserna:

$$ {}^{0}_{1}\mathrm{T} = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

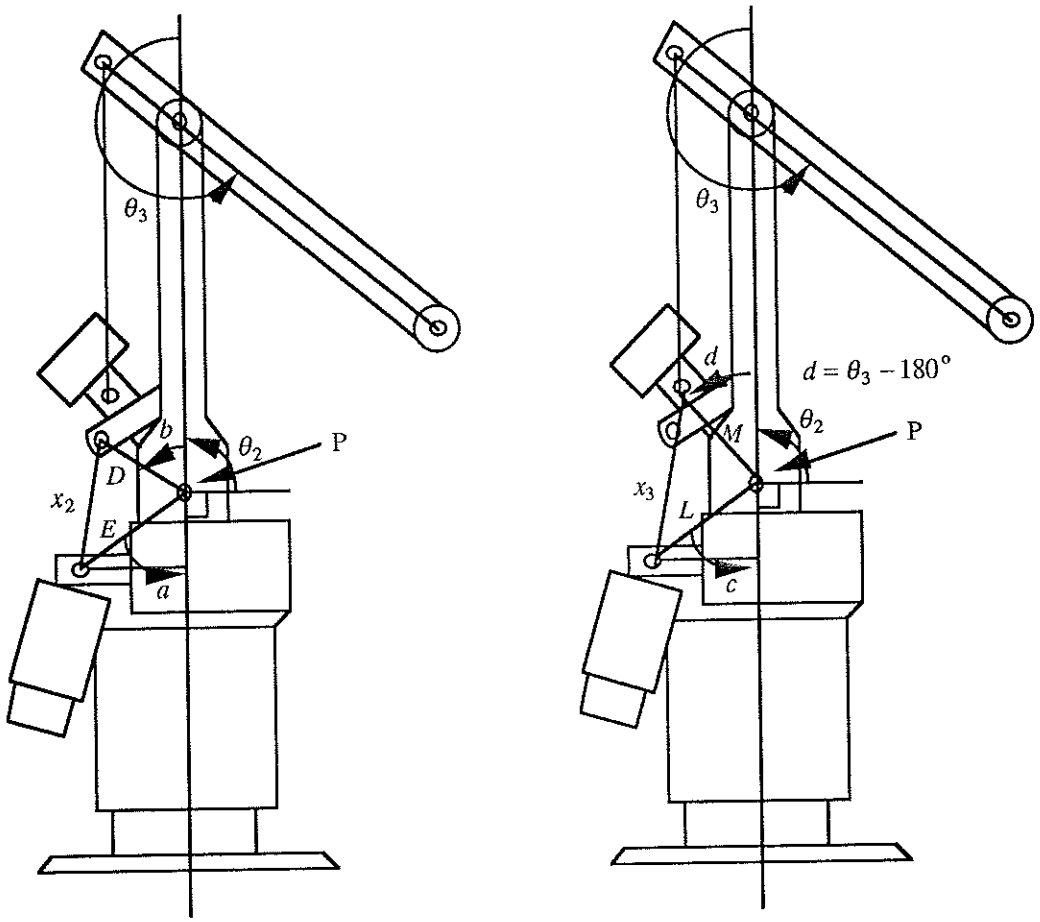
$$ {}^{1}_{2}\mathrm{T} = \begin{bmatrix} \cos(\theta_2) & -\sin(\theta_2) & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \sin(\theta_2) & \cos(\theta_2) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{2}_{3}\mathrm{T} = \begin{bmatrix} \cos(\theta_3) & -\sin(\theta_3) & 0 & l_2 \\ \sin(\theta_3) & \cos(\theta_3) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{3}_{4}\mathrm{T} = \begin{bmatrix} \cos(\theta_4) & -\sin(\theta_4) & 0 & l_3 \\ \sin(\theta_4) & \cos(\theta_4) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{4}_{5}\mathrm{T} = \begin{bmatrix} \cos(\theta_5) & -\sin(\theta_5) & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \sin(\theta_5) & \cos(\theta_5) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

För att kunna gå över till *eng. actuator space* måste man veta förhållandet mellan ledernas vinkelutslag, θ_i, och drivmotorernas vinkelpositioner, s_i. Detta förhållande är ingalunda lineärt för alla leder. Speciellt led 2 och 3 har en ganska komplicerad olineär koppling mellan motorerna och ledernas vinkelutslag. Detta illustreras i figur 3.3. nedan. Det fungerar på så sätt att drivmotorerna styr lederna genom att variera längderna x_2 respektive x_3 vars förhållande till s_2 och s_3 är lineärt. Dessutom beror ju θ_3 på θ_2. Nedan redovisas alla θ_i uttryckta i s_i.



Figur 3.3.: Beskrivning av förhållandet mellan "ledvinkel" och "motorvinkel" för led 2 (vänstra bilden) och led 3 (högra bilden). Led 2 har betecknats med P.

$$\theta_1 = n_1 s_1$$

$$\theta_2 = \frac{3\pi}{2} - a - b - \arccos\left(\frac{D^2 + E^2 - x_2^2}{2DE}\right) = \left[x_2 = \frac{n_2 s_2}{2\pi} + x_{02}\right]$$

$$= \frac{3\pi}{2} - a - b - \arccos\left(\frac{D^2 + E^2 - \left(\frac{n_2 s_2}{2\pi} + x_{02}\right)^2}{2DE}\right)$$

$$\theta_3 = \frac{5\pi}{2} - c - \arccos\left(\frac{L^2 + M^2 - x_3^2}{2LM}\right) - \theta_2 = \left[x_3 = \frac{n_3 s_3}{2\pi} + x_{03}\right]$$

$$= \frac{5\pi}{2} - c - \arccos\left(\frac{L^2 + M^2 - \left(\frac{n_3 s_3}{2\pi} + x_{03}\right)^2}{2LM}\right) - \theta_2$$

$$\theta_4 = n_4 s_4$$

$$\theta_5 = n_5\left(s_5 - \frac{s_4}{n_4}\right)$$

där alla n_i är konstanter.
Uttrycken ovan liksom de tidigare bestämda transformationsmatriserna innehåller alltså en del konstanta robotparametrar, vilka redovisas nedan.

$$n_1 = -\frac{1}{158} \text{ m / varv}$$

$l_1 = 0.700$ m	$D = 0.140$ m	$a = 57°$	$n_2 = -0.005$ m / varv	$x_{02} = 0.222491$ m
$l_2 = 0.450$ m	$E = 0.239$ m	$b = 57°$	$n_3 = -0.005$ m / varv	$x_{03} = 0.178725$ m
$l_3 = 0.670$ m	$L = 0.239$ m			
$l_4 = 0.260$ m	$M = 0.140$ m	$c = 42°$	$n_4 = \frac{1}{128}$ m / varv	

$$n_5 = -\frac{1}{76} \text{ m / varv}$$

När man vet dessa relationer mellan de två olika vinkelrepresentationerna kan man bestämma jacobianen, samt dess tidsderivata, vilka vi sedan behöver för att ta fram en modell i *eng. actuator space*. Jacobianen definieras i detta fallet som

$$J(S) = \begin{bmatrix} \frac{\partial\theta_1}{\partial s_1} & \cdots & \frac{\partial\theta_1}{\partial s_5} \\ \vdots & & \vdots \\ \frac{\partial\theta_5}{\partial s_1} & \cdots & \frac{\partial\theta_5}{\partial s_5} \end{bmatrix}$$

där alla element är noll utom

$$\frac{\partial\theta_1}{\partial s_1} = n_1$$

$$\frac{\partial\theta_2}{\partial s_2} = -\frac{n_2}{2\pi} \frac{(\frac{n_2 s_2}{2\pi} + x_{02})}{DE\sqrt{1 - (\frac{D^2 + E^2 - (\frac{n_2 s_2}{2\pi} + x_{02})^2}{2DE})^2}}$$

$$\frac{\partial\theta_3}{\partial s_2} = \frac{n_2}{2\pi} \frac{(\frac{n_2 s_2}{2\pi} + x_{02})}{DE\sqrt{1 - (\frac{D^2 + E^2 - (\frac{n_2 s_2}{2\pi} + x_{02})^2}{2DE})^2}}$$

$$\frac{\partial\theta_3}{\partial s_3} = -\frac{n_3}{2\pi} \frac{(\frac{n_3 s_3}{2\pi} + x_{03})}{LM\sqrt{1 - (\frac{L^2 + M^2 - (\frac{n_3 s_3}{2\pi} + x_{03})^2}{2LM})^2}}$$

$$\frac{\partial\theta_4}{\partial s_4} = n_4$$

$$\frac{\partial\theta_5}{\partial s_4} = -\frac{n_5}{n_4}$$

$$\frac{\partial\theta_5}{\partial s_5} = n_5$$

Tidsderivatan av jacobianen, $\dot{J}(S)$, fås sedan genom att derivera varje element m.a.p. på tiden. Dessa blir då enligt följande:

$$\frac{d}{dt}\left(\frac{\partial\theta_2}{\partial s_2}\right) = -\frac{1}{DE}\left(\frac{n_2}{2\pi}\right)^2 \frac{\dot{s}_2\left(1 - \dfrac{\dfrac{D^2+E^2-\left(\frac{n_2 s_2}{2\pi}+x_{02}\right)^2}{2DE}}{DE}\left(\frac{n_2 s_2}{2\pi}+x_{02}\right)^2\right)}{\sqrt{1-\left(\dfrac{D^2+E^2-\left(\frac{n_2 s_2}{2\pi}+x_{02}\right)^2}{2DE}\right)^2}}$$

$$\frac{d}{dt}\left(\frac{\partial\theta_3}{\partial s_2}\right) = \frac{1}{DE}\left(\frac{n_2}{2\pi}\right)^2 \frac{\dot{s}_2\left(1 - \dfrac{\dfrac{D^2+E^2-\left(\frac{n_2 s_2}{2\pi}+x_{02}\right)^2}{2DE}}{DE}\left(\frac{n_2 s_2}{2\pi}+x_{02}\right)^2\right)}{\sqrt{1-\left(\dfrac{D^2+E^2-\left(\frac{n_2 s_2}{2\pi}+x_{02}\right)^2}{2DE}\right)^2}}$$

$$\frac{d}{dt}\left(\frac{\partial\theta_3}{\partial s_3}\right) = -\frac{1}{LM}\left(\frac{n_3}{2\pi}\right)^2 \frac{\dot{s}_3\left(1 - \dfrac{\dfrac{L^2+M^2-\left(\frac{n_3 s_3}{2\pi}+x_{03}\right)^2}{2LM}}{LM}\left(\frac{n_3 s_3}{2\pi}+x_{03}\right)^2\right)}{\sqrt{1-\left(\dfrac{L^2+M^2-\left(\frac{n_3 s_3}{2\pi}+x_{03}\right)^2}{2LM}\right)^2}}$$

där resten av elementen blir noll.
När vi sedan ska bestämma en modell med Newton-Euler- eller Lagrange-metoden kommer vi att betrakta varje länk som en stång utan massa med en punktmassa utan volym belägen i änden där då all massa anses finnas. Detta är en approximation men förenklar räkningarna ganska mycket eftersom tröghetstensorn då blir noll.
Därmed har vi tagit fram det vi behöver om roboten för att bestämma denna något approximativa modell.

4. Adaptiv regleralgoritm

Vår uppgift bestod alltså i att testa en adaptiv regleralgoritm, framtagen av Rolf Johansson vid Institutionen för Reglerteknik på LTH, på en industrirobot tillverkad av ASEA (numera ABB). Detta avsnitt kommer att ge en kort genomgång av idéer bakom algoritmen och något om stabilitetsanalys, samt visa algoritmens utseende för ett generellt antal leder hos roboten. För ytterligare information hänvisas till Rolfs artikel (se referenser).
Direkt adaptiv reglering för robotar har studerats tidigare. Craig, Hsu och Sastry tillämpade idéer om modellreferensreglering och utvecklade en regulator och stabilitetsbevis baserade på SPR-system (SPR=*eng. Strictly Positiv Real*, se Adaptive Control av Åström och Wittenmark). Slotine och Li presenterade en lösning påminnande om MIT-regeln. Denna algoritm innefattade en regulator, som till skillnad från den förra, var oberoende av accelerationsmätningar och dessutom var linear i parametrarna. Den tekniska nyheten var utnyttjandet av skevsymmetriska systemmatriser och deras goda, egenskaper, för att slippa mät- och beräkningsproblem. Slotine och Li använde sig av Lyapunov-teori vid sin design, men utnyttjade då inte hela tillståndsvektorn, vilket leder till en ej formellt fullständig Lyaponov-funktion. Koditschek har dock visat Lyapunov-stabilitet för algoritmen.
Rolf Johanssons algoritm (direkt adaptiv reglering) utnyttjar också Lyapunov-teori, men med en fullständig tillståndsvektor, och förutsätter att hastighets- och positionsmätvärden för robotlederna finns tillgängliga. Han utgår här från den välkända robotmodellen (n stycken leder)

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau$$

Eventuell utvidgning med friktionsmodell kan lätt göras till den slutliga regleralgoritmen i efterhand och tas därför ej med här.
Den referenstrajektoria man vill att roboten ska följa specificeras med följande modell:

$$\ddot{q}_r + K_D\dot{q}_r + K_P q_r = Kr$$

där q_r och r ($n \times 1$-vektorer) är generaliserade koordinater, enligt tidigare avsnitt, respektive referenssignal, och där K, K_P och K_D är $n \times n$-matriser som väljes så att en stabil referensmodell erhålles.
För närmare detaljer om Lyapunov-designen hänvisas till Rolfs artikel. Huvudidén är dock den att man bildar en fullständig tillståndsvektor

$$\tilde{x}(t) = (\dot{\tilde{q}}^T(t)\tilde{q}^T(t)\tilde{\theta}^T(t))^T$$

där $\dot{\tilde{q}}(t) = \dot{q}(t) - \dot{q}_r(t)$ är felet i vinkelhastighet, $\tilde{q}(t) = q(t) - q_r(t)$ är positionsfelet och $\tilde{\theta}(t) = \hat{\theta}(t) - \theta(t)$ är felet i skattningen av de parametrar man bestämt sig för att skatta. $\hat{\theta}(t)$ är här skattningen medan $\theta(t)$ är det verkliga värdet på de okända parametrarna. $\tilde{x}(t)$ har dimensionerna $(2n + p) \times 1$, där p är antalet skattade parametrar. Man låter sedan $\tilde{x}(t)$ ingå i en lämpligt vald Lyapunov-funktion, $V(\tilde{x}(t))$. En stabil regleralgoritm ska sedan uppfylla $\dot{V}(\tilde{x}(t)) \le 0$, vilket följande adaptiva algoritm kan visas göra:

$$\dot{\hat{\theta}}(\ddot{q}_r,\dot{q},\dot{q}_r,q,q_r) = -P_{\theta\theta}^{-1}\psi^T(\dot{\tilde{q}} + P_{12}\tilde{q})$$

$$\tau(\ddot{q}_r,\dot{q},\dot{q}_r,q,q_r;\hat{\theta}) = \psi\hat{\theta} + \psi_0 - (D + P_{qq}\Omega^{-1}P_{qq})\dot{\tilde{q}} - DP_{qq}^{-1}\Omega\tilde{q}$$

där

$$\psi\hat{\theta} + \psi_0 = \psi(\ddot{q}_r,\dot{q},\dot{q}_r,q,q_r)\hat{\theta} + \psi_0(\ddot{q}_r,\dot{q},\dot{q}_r,q,q_r)$$
$$= -\frac{1}{2}\dot{M}(\dot{\tilde{q}} + P_{12}\tilde{q}) + M(q)(\ddot{q}_r - P_{12}\dot{\tilde{q}}) + C(q,\dot{q})\dot{q} + G(q)$$

$$P_{12} = P_{qq}^{-1}\Omega$$

samt där de positivt definita $n \times n$-matriserna $P_{qq} = P_{qq}{}^T > 0$, $\Omega = \Omega^T > 0$ och $D = D^T > 0$, och den positivt definita $p \times p$-matrisen $P_{\theta\theta} = P_{\theta\theta}{}^T > 0$ är designparametrar. Styrsignalen, τ, är en $n \times 1$-vektor med momentet som ska påläggas varje led i *eng. joint space* eller varje drivmotor i *eng. actuator space* beroende på i vilket rum man befinner sig.
Detta leder alltså till en kontinuerlig adaptiv "MIMO"-regulator (MIMO=*eng.* M*ulti-I*n*put* M*ulti-Out*-*put*) av PD-typ. En eventuell utvidgning med en friktionsmodell leder till andra ψ eller ψ_0 beroende på om friktionsparametrarna skattas eller ej (se avsnitt 5).
Bevisen för denna algoritmen är alltså gjorda i det kontinuerliga fallet, men eftersom vi ville ha en diskret regulator var vi tvungna att diskretisera den, vilket ger en regulator vars stabilitet ej är strikt bevisad, men med snabb samplingshastighet bör även den diskreta varianten ha goda egenskaper (se avsnitt 5 om simulering och experiment).
I ovanstående regulatoralgoritm minskar Lyapunov-funktionen, $V(\tilde{x}(t))$, så länge det finns ett fel i hastighets- eller positionstillstånden. Däremot är detta ej nödvändigtvis sant för fel i de skattade parametrarna. Detta kan lösas genom att även låta accelerationsmätvärden ingå i algoritmen, men eftersom detta ej är möjligt i praktiken har denna algoritm endast teoretiskt intresse och behandlas ej här.

5. Simuleringar och experiment

I detta avsnitt presenterar vi de resultat som simuleringar och experiment har givit. Alla simuleringar är gjorda med simuleringsprogrammet SIMNON. Experimenten utfördes, vilket tidigare nämnts, på en ASEA IRB L6/2 industrirobot, kopplad till en fristående dator med VME-buss och 68040-processor. Alla implementeringar skrevs i Modula-2. För att underlätta implementeringsarbetet så mycket som möjligt användes programmet SIM2VME, ett program som automatöversätter SIMNON-kod till Modula-2. En del nödvändiga justeringar gjordes i den automatgenererade koden, som därefter kompilerades och länkades ihop med en befintlig programstomme. Denna programstomme innehåller allt som behövs runt regulatorkoden, d.v.s. operatörskommunikation, rutiner för plottning, etc.. Både programmet SIM2VME och programstommen ovan är skrivna av Ola Dahl, Instutitionen för Reglerteknik, LTH. Insamling och plottning av data gjordes med MATLAB.

När man skall reglera en komplicerad process, som i vårt fall en robot, brukar en god regel vara att börja "enkelt". Vi koncentrerade oss därför först på reglering av enbart två leder, nämligen led 2 och 3. Anledningen till att vi har valt just led 2 och 3, är att dessa leder har störst kopplingseffekter, vilket gör dem till ett reglertekniskt svårare problem än de övriga lederna. Om regleralgoritmen fungerar bra för led 2 och 3, finns det goda förutsättningar för att den fungerar bra även för de övriga lederna, då kopplingseffekterna mellan dessa är mindre.

5.1.1. Modell för två leder

Genom att använda metoderna beskrivna i avsnitt 2, kan man härleda nedanstående dynamiska modell för led 2 och 3 i *eng. joint space*. Dock ingår inte modellering av friktion i alla simuleringar och experiment nedan. Observera att båda metoderna, d.v.s. både Newton-Euler och Lagrange, ger samma modell som resultat:

$$\tau_{joint} = \mathrm{M}(\theta)\ddot{\theta} + \mathrm{C}(\theta,\dot{\theta})\dot{\theta} + \mathrm{G}(\theta) + \mathrm{F}(\theta,\dot{\theta})$$

där

$$\mathrm{M}(\theta) = \begin{bmatrix} m_3 l_3^{\,2} + 2m_3 l_2 l_3 \cos(\theta_3) + (m_2 + m_3) l_2^{\,2} & m_3 l_3^{\,2} + m_3 l_2 l_3 \cos(\theta_3) \\ m_3 l_3^{\,2} + m_3 l_2 l_3 \cos(\theta_3) & m_3 l_3^{\,2} \end{bmatrix}$$

$$\mathrm{C}(\theta,\dot{\theta}) = \begin{bmatrix} -2m_3 l_2 l_3 \sin(\theta_3)\dot{\theta}_3 & -m_3 l_2 l_3 \sin(\theta_3)\dot{\theta}_3 \\ m_3 l_2 l_3 \sin(\theta_3)\dot{\theta}_2 & 0 \end{bmatrix}$$

$$\mathrm{G}(\theta) = \begin{bmatrix} m_3 l_3 g \cos(\theta_2 + \theta_3) + (m_2 + m_3) l_2 g \cos(\theta_2) \\ m_3 l_3 g \cos(\theta_2 + \theta_3) \end{bmatrix}$$

samt där

$$\mathrm{F}(\theta,\dot{\theta}) = \begin{bmatrix} v_2\dot{\theta}_2 + c_2 sgn(\dot{\theta}_2) \\ v_3\dot{\theta}_3 + c_3 sgn(\dot{\theta}_3) \end{bmatrix} \quad \text{eller} \quad \mathrm{F}(\theta,\dot{\theta}) = \begin{bmatrix} v_2\dot{\theta}_2 \\ v_3\dot{\theta}_3 \end{bmatrix}$$

beroende på om Coulomb-friktion modelleras eller ej. m_i är här robotarm i:s punktformiga massa, l_i dess längd, g gravitationskonstanten, v_i den viskösa friktionskonstanten, c_i Coulomb-friktionskonstanten, samt där θ_i är vinkelutslaget för robotled i.
Denna modell kan därefter transformeras till *eng. actuator space* m.h.a. jacobianen, vilket leder till följande modell:

$$\tau_{actuator} = \mathrm{M}^*(S)\ddot{S} + \mathrm{C}^*(S,\dot{S})\dot{S} + \mathrm{G}^*(S) + \mathrm{F}^*(S,\dot{S})$$

där

$$M^*(S) = \begin{bmatrix} J_{22}^2(m_2 + m_3)l_2^2 & J_{22}J_{33}m_3l_2l_3\cos(\theta_3) \\ J_{22}J_{33}m_3l_2l_3\cos(\theta_3) & J_{33}^2m_3l_3^2 \end{bmatrix}$$

$$C^*(S,\dot{S}) = \begin{bmatrix} C^*_{22} & C^*_{23} \\ C^*_{32} & C^*_{33} \end{bmatrix}$$

med

$$C^*_{22} = J_{22}\dot{J}_{22}(m_2 + m_3)l_2^2 - J_{22}^2J_{33}m_3l_2l_3\sin(\theta_3)\dot{s}_3$$
$$C^*_{23} = J_{22}m_3l_2l_3(\dot{J}_{33}\cos(\theta_3) + J_{33}\sin(\theta_3)(J_{22}\dot{s}_2 - J_{33}\dot{s}_3))$$
$$C^*_{32} = J_{33}m_3l_2l_3(\dot{J}_{22}\cos(\theta_3) + J_{22}^2\sin(\theta_3)\dot{s}_2)$$
$$C^*_{33} = J_{33}\dot{J}_{33}m_3l_3^2$$

$$G^*(S) = \begin{bmatrix} J_{22}(m_2 + m_3)l_2g\cos(\theta_2) \\ J_{33}m_3l_3g\cos(\theta_2 + \theta_3) \end{bmatrix}$$

samt där

$$F^*(S,\dot{S}) = \begin{bmatrix} J_{22}^2(v_2 + v_3)\dot{s}_2 - J_{22}J_{33}v_3\dot{s}_3 + J_{22}c_2sgn(J_{22}\dot{s}_2) - J_{22}c_3sgn(-J_{22}\dot{s}_2 + J_{33}\dot{s}_3) \\ -J_{22}J_{33}v_3\dot{s}_2 + J_{33}^2v_3\dot{s}_3 + J_{33}c_3sgn(-J_{22}\dot{s}_2 + J_{33}\dot{s}_3) \end{bmatrix}$$

eller

$$F^*(S,\dot{S}) = \begin{bmatrix} J_{22}^2(v_2 + v_3)\dot{s}_2 - J_{22}J_{33}v_3\dot{s}_3 \\ -J_{22}J_{33}v_3\dot{s}_2 + J_{33}^2v_3\dot{s}_3 \end{bmatrix}$$

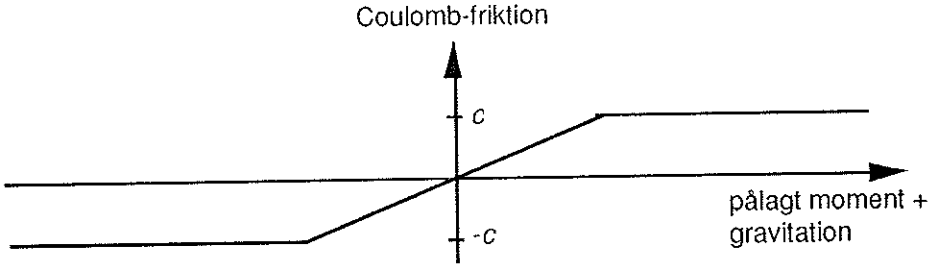
beroende på om Coulomb-friktion modelleras eller ej. J_{ij} och $\dot{J}_{ij}$ är här elementen i jacobianen respektive dess derivata enligt följande:

$$J(S) = \begin{bmatrix} \frac{\partial\theta_2}{\partial s_2} & \frac{\partial\theta_2}{\partial s_3} \\ \frac{\partial\theta_3}{\partial s_2} & \frac{\partial\theta_3}{\partial s_3} \end{bmatrix} = \begin{bmatrix} J_{22} & 0 \\ -J_{22} & J_{33} \end{bmatrix}$$

$$\dot{J}(S) = \begin{bmatrix} \frac{d}{dt}(\frac{\partial\theta_2}{\partial s_2}) & \frac{d}{dt}(\frac{\partial\theta_2}{\partial s_3}) \\ \frac{d}{dt}(\frac{\partial\theta_3}{\partial s_2}) & \frac{d}{dt}(\frac{\partial\theta_3}{\partial s_3}) \end{bmatrix} = \begin{bmatrix} \dot{J}_{22} & 0 \\ -\dot{J}_{22} & \dot{J}_{33} \end{bmatrix}$$

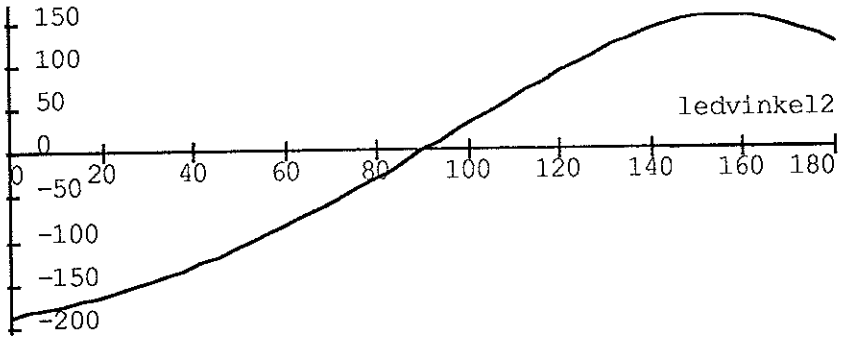
Samtliga J_{ij} och $\dot{J}_{ij}$ samt θ_i finns redan härledda i avsnitt 3 (uttryckta i s_i). s_i motsvarar här motorvinkel i. Vi använder i fortsättningen beteckningen q_i i stället för s_i, eftersom dessa utgör våra generaliserade koordinater.
Man kan alltid diskutera hur avancerad modellen skall vara, d.v.s. hur mycket man skall ta hänsyn till. I vårt fall är vi intesserade av en modell som ger ett bra simuleringsresultat, d.v.s. ett resultat som stämmer väl överens med verkligheten. I modellen ovan är en del approximationer gjorda, bl.a. antas massorna vara punktmassor utan volym vilket leder till att tröghetstensorn blir noll (se även avsnitt 3). Trots approximationerna blev simuleringsresultaten bra. I de simuleringar där vi inkluderade både viskös friktion och Coulomb-friktion i processen var det möjligt att använda regulatorparametrarna från simuleringarna direkt på roboten med godkänd reglering.
Att modellera Coulomb-friktion m.h.a. sgn-funktionen är bekvämt, men inte helt korrekt. Vid hastigheten noll måste summan (med tecken) av pålagt moment, gravitation och Coulomb-friktionen vara lika med noll (i annat fall skulle roboten röra sig). Om vi nu sätter pålagt moment till noll,

måste gravitationen och Coulomb-friktionen vara lika stora men med motsatt tecken. Hastigheten är alltså fortfarande noll men Coulomb-friktionen är inte noll. Detta förhållande syns inte om man använder sgn-funktionen, som alltid ger en Coulomb-friktion lika med noll när hastigheten är noll. Det stämmer däremot bra om hastigheten är skild från noll. Egentligen är Coulomb-friktionen en ramp-funktion (se figur 5.1 nedan) istället för en steg-funktion, där friktionen alltid är lika stor som summan av moment och gravitation (med tecken) tills friktionen når sitt maxvärde (alt. minvärde), d.v.s. då robotarmen börjar röra sig. Vi har därför i våra simuleringar modellerat Coulomb-friktionen i processen med en ramp-funktion istället för en sgn-funktion. För närmare detaljer, se SIMNON-filen PROCESS i appendix B.

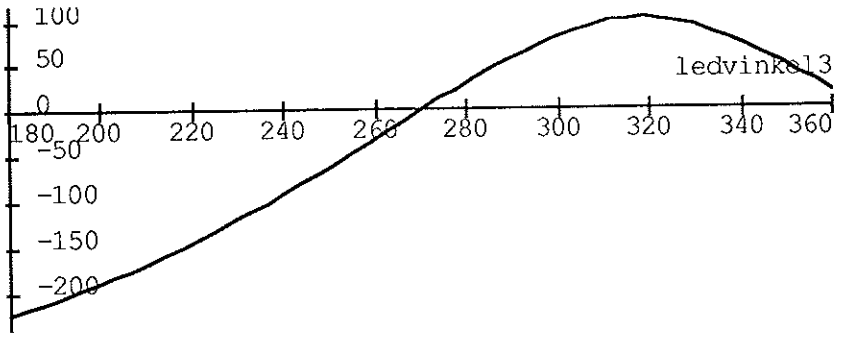


Figur 5.1: Coulomb-friktionen modellerad som en ramp.

Det finns ytterligare ett skäl till varför vi använder oss av en ramp istället för ett steg. Av någon anledning tar det SIMNON cirka 40 gånger längre tid att simulera, när vi i processen modellerar Coulomb-friktion med en sgn-funktion istället för en ramp. Varför vet vi inte i skrivande stund.
Vi har tidigare behandlat hur man transformerar en modell mellan *eng. joint space* och *eng. actuator space* med hjälp av jacobianen. För att denna transformation skall bli så enkel som möjligt, vill man gärna ha ett lineärt samband mellan "ledvinkel" och "motorvinkel" för lederna. Jacobianen blir då synnerligen trivial, en matris med enbart konstanter. För led 1, 4 och 5 är förhållandena lineära. För led 2 och 3 är sambanden betydligt mer komplicerade (se avsnitt 3). Om man plottar dessa samband (se figur 5.2 och 5.3), ser man dock att förhållandena är nästan lineära inom robotens arbetsområde. Vi tror därför att det skulle gå bra att approximera sambanden mellan "ledvinkel" och "motorvinkel" för led 2 och 3 med lineära funktioner och därmed få en enklare transformation. Vi har dock inte själva använt oss av denna approximation.



Figur 5.2: "Motorvinkel 2" (i radianer) som funktion av "ledvinkel 2" (i grader).



Figur 5.3: "Motorvinkel 3" (i radianer) som funktion av "ledvinkel 3" (i grader).

I transformationen ovan används jacobianen också för omvandling av moment mellan *eng. joint space* och *eng. actuator space* (se även avsnitt 2.3). För led 2 och 3 gäller därmed följande samband:

$$\tau_{actuator} = J^T(S)\tau_{jo\,int}$$

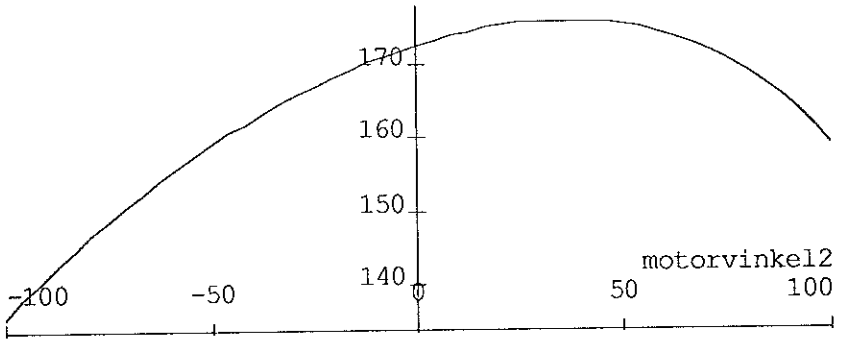
Omvänt gäller också det inversa förhållandet d.v.s.:

$$\tau_{jo\,int} = (J^T(S))^{-1}\tau_{actuator}$$

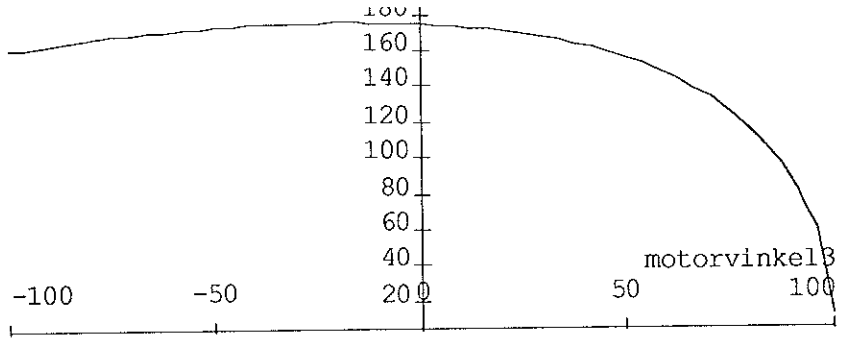
där

$$(J^T(S))^{-1} = \begin{bmatrix} \frac{1}{J_{22}} & \frac{1}{J_{33}} \\ 0 & \frac{1}{J_{33}} \end{bmatrix}$$

Det man skall observera här, är att ett konstant pålagt moment på drivmotorerna (i *eng. actuator space*) ger ett konstant moment på lederna (i *eng. joint space*). Dock är detta moment beroende av i vilket läge roboten står. Hur dessa omvandlingskoefficienter varierar med motorvinklarna visas i figur 5.4 och 5.5 nedan. Detta gäller dock bara för led 2 och 3, där sambanden mellan "ledvinkel" och "motorvinkel" inte är lineära.



Figur 5.4: 1/J22 som funktion av "motorvinkel 2" (i radianer).



Figur 5.5: 1/J33 som funktion av "motorvinkel 3" (i radianer).

5.1.2. Regleralgoritm för två leder

Eftersom uttrycken i den adaptiva regleralgoritmen, ψ och ψ_0 (se avsnitt 4), beror på hur många parametrar som skattas samt på hur modellen ser ut (d.v.s. om vi tar hänsyn till friktion eller ej),

redovisas därför ψ och ψ_0 samt regulatorparametrarna för varje enskild regulator i nästa delavsnitt.

5.1.3. Simuleringar och experiment för två leder

Den adaptiva regleralgoritmen i avsnitt 4 är i sin ursprungsform en kontinuerlig regleralgoritm. För att kunna implementera denna algoritm i Modula-2, är det nödvändigt att diskretisera den. Detta gjorde vi genom att använda bilineär transformation eller Tustins approximation som den också kallas (se Computer-Controlled Systems av Åström och Wittenmark). Tustins approximation har fördelen att den alltid transformerar stabila respektive instabila kontinuerliga system till stabila respektive instabila diskreta system. Den diskreta algoritmen erhåller man genom att ersätta Laplaceoperatorn, s, med följande uttryck:

$$s = \frac{2}{h}\frac{z-1}{z+1}$$

där z är z-transformens operator.
I vårt fall är detta mycket enkelt då det endast är parameteruppdateringslagen som innehåller en Laplaceoperator (p.g.a. att $L(\dot{\hat{\theta}}) = s\hat{\theta}$). Dessutom använder sig ju regleralgoritmen, vilket tidigare beskrivits, av både robotledernas position och vinkelhastighet. Positionen kan vi mäta, men inte vinkelhastigheten. Därför approximeras denna i våra simuleringar och experiment med en differens enligt:

$$\dot{q}(t) = \frac{q(t) - q(t-h)}{h}$$

där h är samplingsperioden. Även modellens vinkelhastighet samt vinkelacceleration används och dessa approximeras på samma sätt (accelerationen beräknas med hastighetsdifferensen istället för positionsdifferensen).
Regleralgoritmen får nu i diskret form följande utseende:

$$\hat{\theta}(t+h) = -\frac{h}{2}(P_{\theta\theta}{}^{-1}\psi^T(t+h)(\frac{\tilde{q}(t+h) - \tilde{q}(t)}{h} + P_{12}\tilde{q}(t+h))$$
$$+ P_{\theta\theta}{}^{-1}\psi^T(t)(\frac{\tilde{q}(t) - \tilde{q}(t-h)}{h} + P_{12}\tilde{q}(t))) + \hat{\theta}(t)$$
$$\tau(t) = \psi(t)\hat{\theta}(t) + \psi_0 - (D + P_{qq}\Omega^{-1}P_{qq})\dot{\tilde{q}}(t) - DP_{qq}{}^{-1}\Omega\tilde{q}(t)$$

där ingående variabler och konstanter har samma innebörd som tidigare, men med den skillnaden att variablerna är tidsdiskreta och uppdateras vid varje sampeltidpunkt.
Den intresserade läsaren finner denna algoritm implementerad i SIMNON-kod i appendix B.
När man använder sig av en diskret regleralgoritm måste man välja en lämplig samplingsperiod. För att avgöra vilken samplingsperiod som var lämplig i vårt fall, gjorde vi jämförelser (m.h.a. simuleringar) mellan en kontinuerlig och en diskret variant av samma regulator. Vi kom fram till att man helst bör ha en samplingsperiod på 5 ms och därunder för att regleringen skall bli riktigt bra. I "nödfall" kan man acceptera samplingsperioder på upp till 10 ms. Vid samplingsperioder på över 10 ms blir dock regleringen påtagligt sämre för den diskreta regulatorn. Om inget annat anges, har vi nedan använt oss av en samplingsperiod på 5 ms. Detta var också den kortaste samplingsperiod som VME-datorn klarade av.
SIMNON-koden är uppdelad i tre filer, en fil med referensmodell och regulator, en fil med processen (d.v.s. process-modellen) och ett connecting system. Det mest logiska hade varit att ha referensmodell och regulator i varsin fil, men eftersom referensmodellen behövs när vi "kör" regulatorn på roboten, är det smidigare att inkludera den i regulatorn. Vi får därmed bara en fil att automatöversätta till Modula-2.
Referensmodellen måste naturligtvis också vara i diskret form (av samma skäl som ovan). För två leder definieras denna i kontinuerlig form enligt följande (se även avsnitt 4):

$$\ddot{q}_r + K_D\dot{q}_r + K_P q_r = Kr$$

där q_r och r är 2×1-vektorer, och K, K_P och K_D är 2×2-matriser. Genom att bara använda diagonalelementen i matriserna K, K_P och K_D, får man två fristående differentialekvationer av andra ordningen (frikopplade från varandra). Polerna till dessa bör man välja så att man inte får någon översläng, annars är det omöjligt att reglera till en position nära ett föremål utan att stöta emot. Ett lämpligt val är därför att placera de två polerna som en dubbelpol, vilket leder till att överföringsfunktionen från referenssignal, r, till önskad position, q_r, får följande principiella utseende:

$$G(s)=\frac{a^2}{(s+a)^2}$$

där a är en lämpligt vald konstant (naturligtvis positiv för att erhålla ett stabilt system). Zero-order-hold-sampling ger följande diskreta överföringsfunktion (se exempelvis tabell 3.1 i Computer-Controlled Systems):

$$H(z)=\frac{b_1z+b_2}{z^2+a_1z+a_2}$$

med

$$b_1=1-e^{-ah}(1+ah)$$
$$b_2=e^{-ah}(e^{-ah}+ah-1)$$
$$a_1=-2e^{-ah}$$
$$a_2=e^{-2ah}$$

där h är samplingsperioden. Denna modell finns implementerad i SIMNON-kod i appendix B.
Alla simuleringar och experiment gjordes i *eng. actuator space* och som referenssignal har vi använt oss av en fyrkantvåg, om inget annat anges. Referensmodellens poler placerades i -2 (i s-planet), d.v.s. $a=2$. I alla simuleringar nedan är massan m_2 3 kg. Massan m_3 ändras vid tidpunkten 20 sekunder från 2 kg till 6 kg.
I experimenten för regulatorerna A till E nedan har vi använt oss av en laststörning på 1.5 kg. Denna inträffar ungefär vid tidpunkten 30 sekunder och upphör ungefär 50 sekunder senare. I experimenten för regulator F har vi däremot använt oss av en laststörning på 2.9 kg, medan vi för regulator G inte har någon laststörning alls.

A. Adaptiv regulator utan friktion

Denna första regulator baserades på modellen ovan, men utan friktionstermerna. Samma modell användes också som process-modell vid simuleringarna. Alla storheter är kända utom massorna m_2 och m_3, vilka skattades. Matriserna ψ och ψ_0 i regulatorn kan därmed härledas till att bli följande:

$$\psi=\begin{bmatrix}\psi_{11} & \psi_{12}\\ \psi_{21} & \psi_{22}\end{bmatrix}$$

$$\psi_0=0$$

där

$$\psi_{11}=J_{22}\dot{J}_{22}l_2^2(\dot{q}_2-y_1)+J_{22}^2l_2^2z_1+J_{22}l_2g\cos(\theta_2)$$
$$\psi_{12}=J_{22}\dot{J}_{22}l_2^2(\dot{q}_2-y_1)-\frac{1}{2}\dot{J}_{22}J_{33}l_2l_3\cos(\theta_3)y_2+J_{22}\dot{J}_{33}l_2l_3\cos(\theta_3)(\dot{q}_3-\frac{1}{2}y_2)$$
$$+J_{22}l_2g\cos(\theta_2)+J_{22}J_{33}^2l_2l_3\sin(\theta_3)\dot{q}_3(\frac{1}{2}y_2-\dot{q}_3)-\frac{1}{2}J_{22}^2J_{33}l_2l_3\sin(\theta_3)\dot{q}_2y_2$$
$$+J_{22}^2l_2^2z_1+J_{22}J_{33}l_2l_3\cos(\theta_3)z_2$$
$$\psi_{21}=0$$

$$\psi_{22} = \dot{J}_{22}J_{33}l_2l_3\cos(\theta_3)(\dot{q}_2 - \frac{1}{2}y_1) - \frac{1}{2}J_{22}\dot{J}_{33}l_2l_3\cos(\theta_3)y_1 + \frac{1}{2}J_{22}J_{33}{}^2l_2l_3\sin(\theta_3)\dot{q}_3y_1$$
$$+ J_{22}{}^2J_{33}l_2l_3\sin(\theta_3)\dot{q}_2(\dot{q}_2 - \frac{1}{2}y_1) + J_{33}\dot{J}_{33}l_3{}^2(\dot{q}_3 - y_2) + J_{33}{}^2l_3{}^2z_2 + J_{22}J_{33}l_2l_3\cos(\theta_3)z_1$$
$$+ J_{33}l_3g\cos(\theta_2 + \theta_3)$$

med

$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \dot{\tilde{q}} + P_{12}\tilde{q}$ och $\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \ddot{q}_r - P_{12}\dot{\tilde{q}}$, där $\tilde{q}$ och P_{12} finns definierade i avsnitt 4.

θ_i i uttrycken ovan avser ledvinkel i (se avsnitt 3).
Skattningsvektorn är

$$\hat{\theta} = \begin{bmatrix} \hat{m}_2 \\ \hat{m}_3 \end{bmatrix}$$

Simuleringsresultaten blev bra med bra följning av referensmodellen, och de skattade parametrarna, $\hat{m}_2$ och $\hat{m}_3$, konvergerade mot rätt värden. Regulatorparametrarna vi använde vid simuleringarna var följande:

$$P_{qq} = \begin{bmatrix} 0.0001 & 0 \\ 0 & 0.0001 \end{bmatrix},\ \Omega = \begin{bmatrix} 0.001 & 0 \\ 0 & 0.001 \end{bmatrix},\ D = \begin{bmatrix} 0.002 & 0 \\ 0 & 0.002 \end{bmatrix},\ P_{\theta\theta} = \begin{bmatrix} 0.03 & 0 \\ 0 & 0.5 \end{bmatrix}$$

Resultaten av experimenten blev däremot inte lika bra. Vi provade regulatorn två gånger, och båda gångerna blev den instabil efter en kortare tids "körning" (cirka en halv minut). Under den tid som regulatorn "fungerade" var regleringen väldigt ryckig, mycket beroende på friktionen hos roboten. Vi beslutade oss därför för att också inkludera friktion i modellen.

B. Adaptiv regulator med skattning av viskös friktion

I denna regulator har vi inkluderat viskös friktion i modellen. Fyra storheter är okända, massorna m_2 och m_3, samt de viskösa friktionskonstanterna ν_2 och ν_3. Dessa skattades. Matriserna ψ och ψ_0 visas nedan:

$$\psi = \begin{bmatrix} \psi_{11} & \psi_{12} & \psi_{13} & \psi_{14} \\ \psi_{21} & \psi_{22} & \psi_{23} & \psi_{24} \end{bmatrix}$$

$$\psi_0 = 0$$

där

$$\psi_{13} = J_{22}{}^2\dot{q}_2$$
$$\psi_{14} = J_{22}{}^2\dot{q}_2 - J_{22}J_{33}\dot{q}_3$$
$$\psi_{23} = 0$$
$$\psi_{24} = -J_{22}J_{33}\dot{q}_2 + J_{33}{}^2\dot{q}_3$$

och övriga element enligt ovan.

Skattningsvektorn är

$$\hat{\theta} = \begin{bmatrix} \hat{m}_2 \\ \hat{m}_3 \\ \hat{\nu}_2 \\ \hat{\nu}_3 \end{bmatrix}$$

Den viskösa friktionen är hastighetsberoende, precis som elementen i Coriolis-matrisen, C. Man kan därför slå ihop Coriolis-matrisen med den viskösa friktionen till en ny C-matris (se SIMNON-kod i appendix B).
Multiplicerar man denna matris med vektorn $\dot{q}$, får man förutom Coriolis-termerna också alla termerna för den viskösa friktionen. Vi behöver därmed inte särbehandla den viskösa friktionen, F(q), utan kan använda de formler som beskrevs i avsnitt 4 för att räkna fram matriserna ψ och ψ_0.
Vid simuleringarna provade vi olika värden på konstanterna v_2 och v_3 i process-modellen (mellan 10 och 200). Tyvärr vet vi inte värdena på dessa parametrar för roboten. Dessa har därför valts dels med hänsyn till vad som kan vara rimligt, och dels med hänsyn till de skattade värden som experimenten på roboten gav.
Även denna regulator gav bra simuleringsresultat med bra följning av referensmodellen och alla skattade parametrar konvergerade mot rätt värden. Regulatorparametrarna vi använde var följande:

$$P_{qq} = \begin{bmatrix} 0.0001 & 0 \\ 0 & 0.0001 \end{bmatrix}, \Omega = \begin{bmatrix} 0.001 & 0 \\ 0 & 0.001 \end{bmatrix}, D = \begin{bmatrix} 0.002 & 0 \\ 0 & 0.002 \end{bmatrix},$$

$$P_{\theta\theta} = \begin{bmatrix} 0.03 & 0 & 0 & 0 \\ 0 & 1.0 & 0 & 0 \\ 0 & 0 & 0.0003 & 0 \\ 0 & 0 & 0 & 0.0003 \end{bmatrix}$$

Experimentresultaten blev klart bättre än för regulator A. Framför allt var regulatorn stabil, men det ryckiga beteendet fanns fortfarande kvar (speciellt i ändlägena), dock med betydligt mindre amplitud än för regulator A.

C. Adaptiv regulator med skattning av viskös friktion och kompensering för Coulomb-friktion

I denna regulator har vi även lagt till kompensering för Coulomb-friktion i modellen. Fyra storheter skattades, m_2 och m_3 samt v_2 och v_3. Matriserna ψ och ψ_0 visas nedan:

$$\psi = \begin{bmatrix} \psi_{11} & \psi_{12} & \psi_{13} & \psi_{14} \\ \psi_{21} & \psi_{22} & \psi_{23} & \psi_{24} \end{bmatrix}$$ (definierade enligt ovan)

$$\psi_0 = \begin{bmatrix} J_{22}c_2 sgn(J_{22}\dot{q}_2) - J_{22}c_3 sgn(-J_{22}\dot{q}_2 + J_{33}\dot{q}_3) \\ J_{33}c_3 sgn(-J_{22}\dot{q}_2 + J_{33}\dot{q}_3) \end{bmatrix}$$

För regulator B ovan kunde den viskösa friktionen ses som en del av C-matrisen, när vi skulle bestämma ψ och ψ_0. På samma sätt kan vi här, och i regulator D nedan, se Coulomb-friktionen som en del av gravitationsmatrisen, G. Därmed är problemet med bestämning av ψ och ψ_0 löst även då modellen innehåller Coulomb-friktion.
Värdena på konstanterna c_2 och c_3 bestämdes experimentellt genom att vi mätte upp de spänningar som fordrades för att respektive led precis skulle börja röra sig. Vid mätningen var robotledernas vinklar (i *eng. joint space)* $\theta_2 = 90°$ och $\theta_3 = 270°$. Mätningen gav följande värden:

Led 2:	Led 3:
$U_+ = 0.80$ V	$U_+ = 2.50$ V
$U_- = 0.44$ V	$U_- = 1.40$ V

Index + respektive - anger att lederna har rört sig så att vinklarna θ_2 och θ_3 har ökat respektive minskat. Maximal spänning för respektive led är 10 V, vilket motsvarar ett moment på 1.3 Nm (i *eng. actuator space*). Omvandlingskoefficienterna för moment mellan *eng. joint space* och *eng. actuator space* är vid de aktuella vinklarna ovan ungefär 173 för led 2 och 175 för led 3 (se figur

5.4 och 5.5 ovan). Därmed kan vi räkna ut ett ungefärligt värde på konstanterna c_2 och c_3 (i *eng. joint space*) enligt följande:

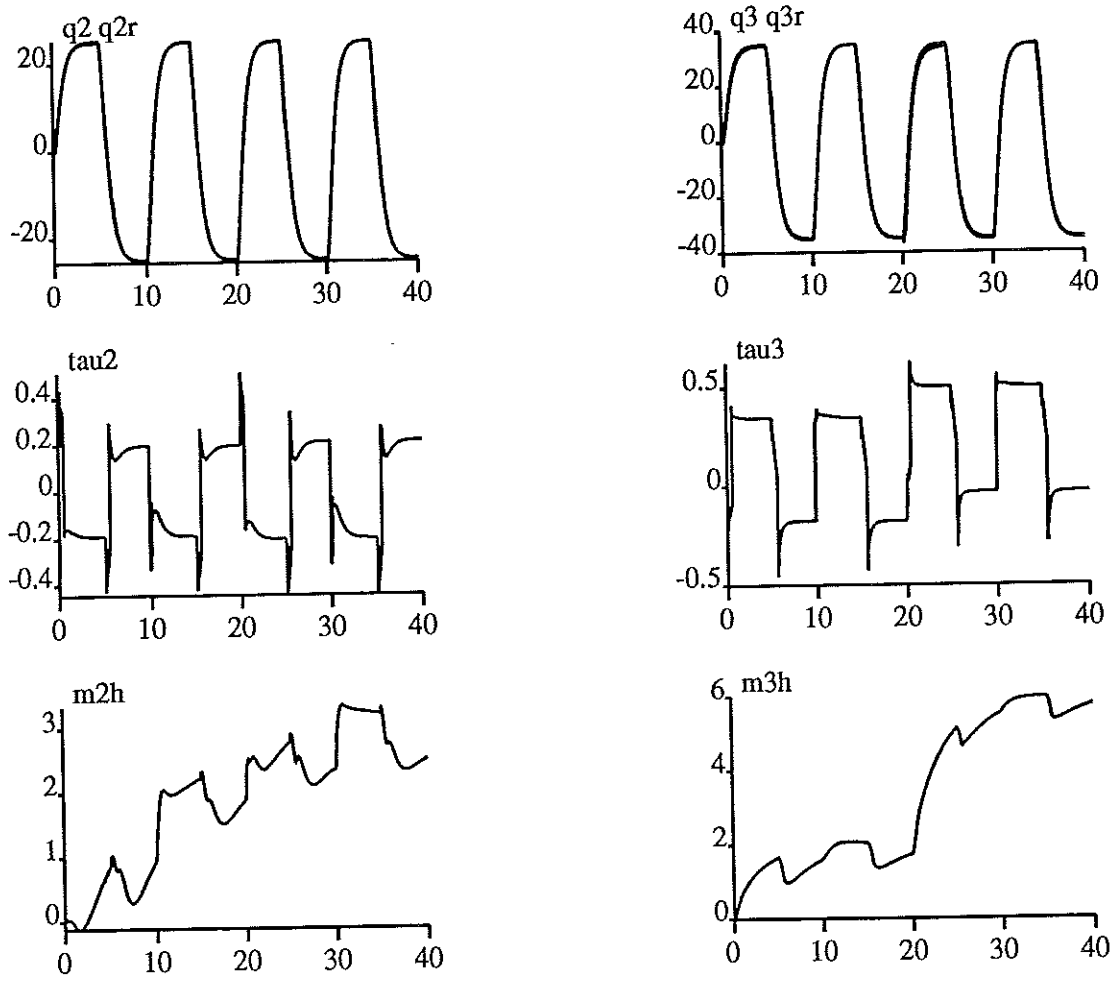
Led 2:

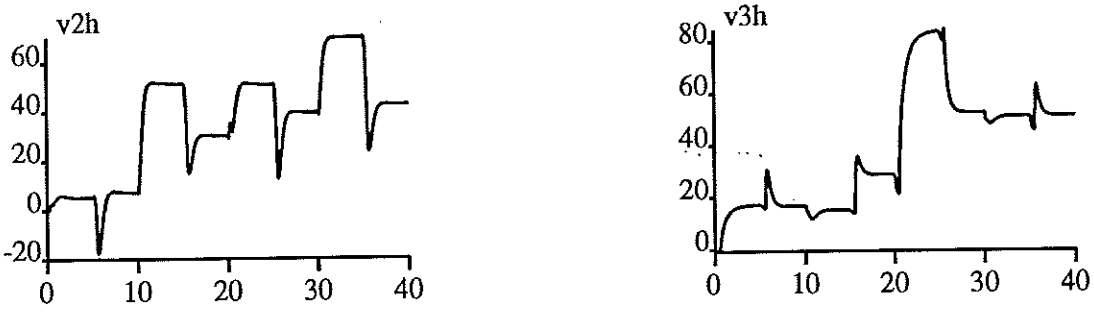
$$c_2 \approx (0.80 + 0.44) / (2 * 10) * 1.3 * 173 \approx 14$$

Led 3:

$$c_3 \approx (2.50 + 1.40) / (2 * 10) * 1.3 * 175 \approx 44$$

Konstanterna c_2 och c_3 antas alltså här vara konstanta i *eng. joint space*. Nu är detta en sanning med modifikation eftersom Coulomb-friktionen ändras då robotlederna värms upp. Värdena ovan är därför inte exakta utan får snarare ses som riktvärden.
Figur 5.6 nedan visar resultatet av simuleringarna. Konstanterna v_2 och v_3 i process-modellen har båda värdet 100. Med viss risk för att bli tjatiga, vill vi dock ännu en gång påpeka att Coulomb-friktionen har modellerats med en ramp i process-modellen, medan vi däremot har använt oss av sgn-funktionen i modellen som ligger till grund för regulatordesignen.





Figur 5.6: Simuleringsresultat för regulator C. Bilderna visar i tur och ordning positionsföljning (referenssignal streckad), styrsignaler, skattning av massor samt skattning av viskös friktion för respektive led.

Regulatorparametrarna vi använde vid simuleringarna ovan var följande:

$$P_{qq} = \begin{bmatrix} 0.0001 & 0 \\ 0 & 0.0001 \end{bmatrix},\ \Omega = \begin{bmatrix} 0.001 & 0 \\ 0 & 0.001 \end{bmatrix},\ D = \begin{bmatrix} 0.004 & 0 \\ 0 & 0.004 \end{bmatrix},$$

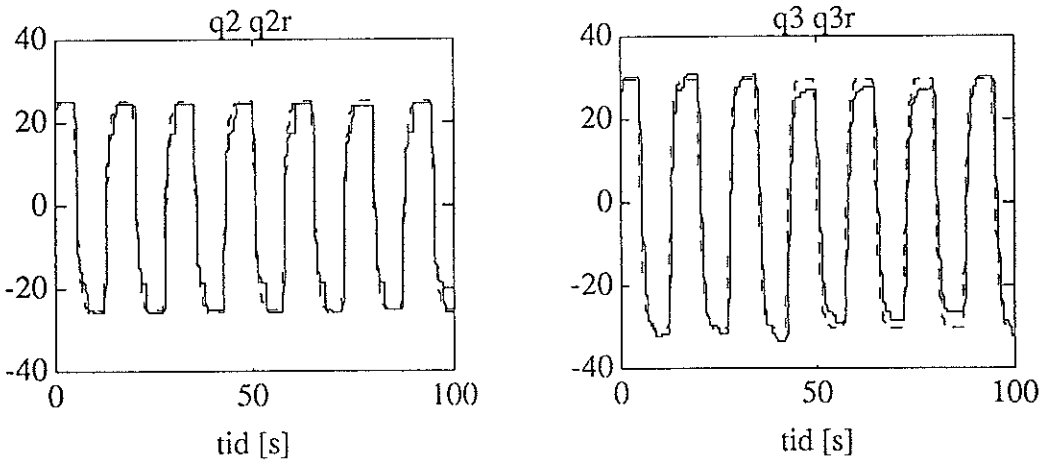
$$P_{\theta\theta} = \begin{bmatrix} 0.03 & 0 & 0 & 0 \\ 0 & 1.3 & 0 & 0 \\ 0 & 0 & 0.0003 & 0 \\ 0 & 0 & 0 & 0.0003 \end{bmatrix}$$

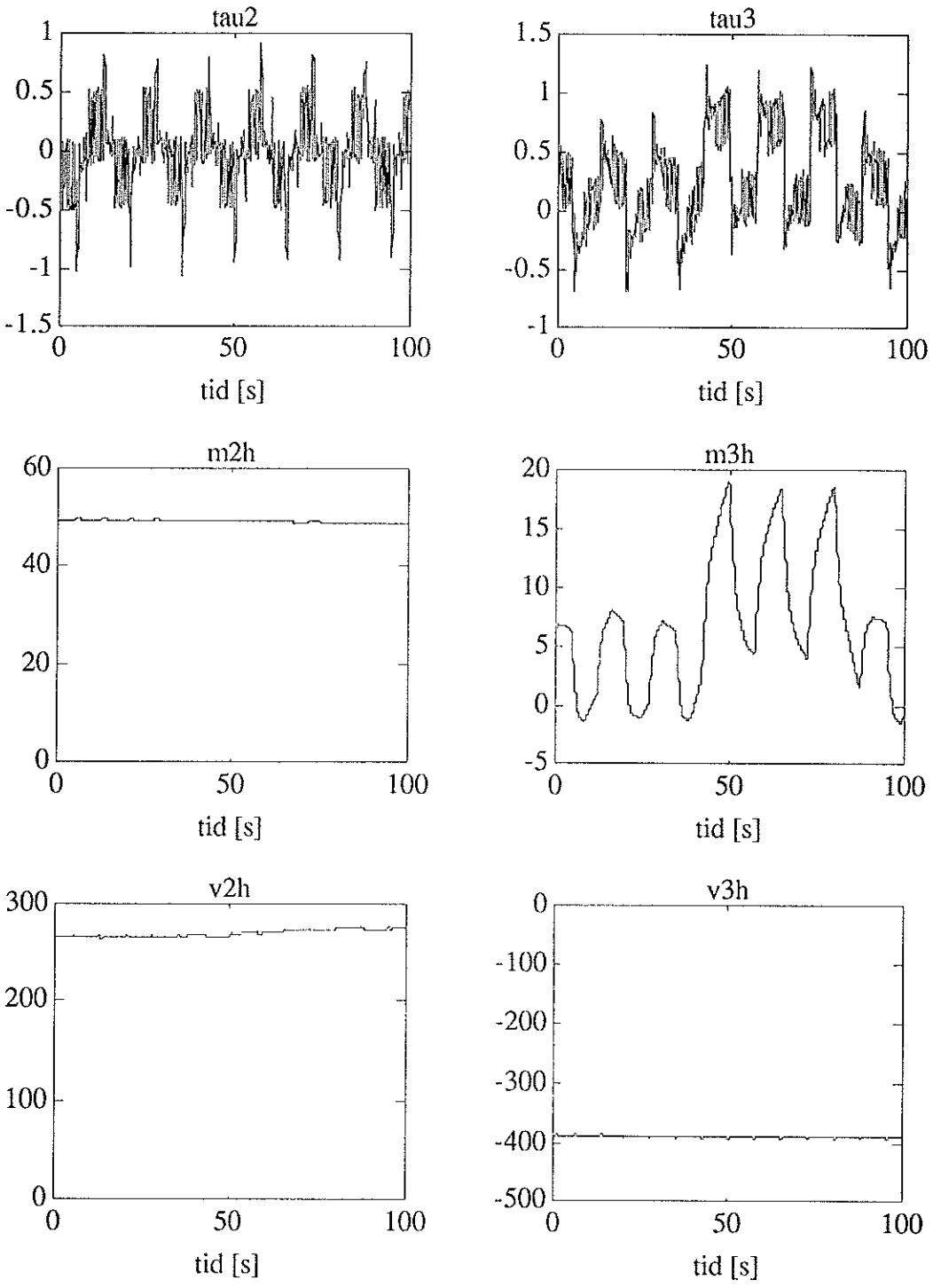
Resultatet av experimenten visas i figur 5.7 nedan. Regulatorparametrarna vi använde var följande:

$$P_{qq} = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix},\ \Omega = \begin{bmatrix} 0.001 & 0 \\ 0 & 0.001 \end{bmatrix},\ D = \begin{bmatrix} 0.1 & 0 \\ 0 & 1.0 \end{bmatrix},\ P_{\theta\theta} = \begin{bmatrix} 1.0 & 0 & 0 & 0 \\ 0 & 1.0 & 0 & 0 \\ 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0.01 \end{bmatrix}$$

Coulomb-friktionens konstanter, som varierades något under experimentens gång (p.g.a. att robotlederna värmdes upp), var i figur 5.7 följande:

$c_2 = 10,\ c_3 = 40$





Figur 5.7: Experimentresultat för regulator C. Referenssignal streckad.

D. Adaptiv regulator med skattning av viskös friktion och skattning av Coulomb-friktion

I denna regulator har vi istället skattat Coulomb-friktionen. Matriserna ψ och ψ_0 visas nedan. Totalt skattades alltså sex stycken parametrar.

$$\psi = \begin{bmatrix} \psi_{11} & \psi_{12} & \psi_{13} & \psi_{14} & \psi_{15} & \psi_{16} \\ \psi_{21} & \psi_{22} & \psi_{23} & \psi_{24} & \psi_{25} & \psi_{26} \end{bmatrix}$$

$\psi_0 = 0$

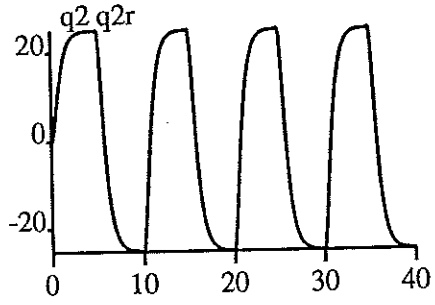
där

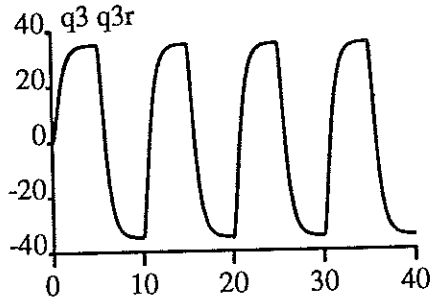
$\psi_{15} = J_{22} sgn(J_{22}\dot{q}_2)$
$\psi_{16} = -J_{22} sgn(-J_{22}\dot{q}_2 + J_{33}\dot{q}_3)$
$\psi_{25} = 0$
$\psi_{26} = J_{33} sgn(-J_{22}\dot{q}_2 + J_{33}\dot{q}_3)$

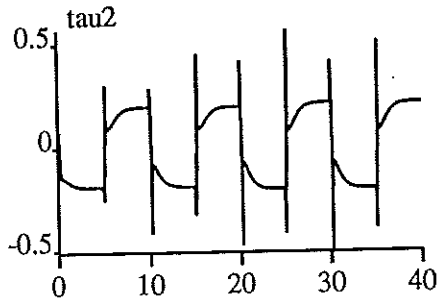
och övriga element enligt ovan.
För denna regulator gäller i övrigt samma förutsättningar som för regulator C ovan. Figur 5.8 nedan visar resultatet av simuleringarna. Regulatorparametrarna vi använde vid simuleringarna var följande:

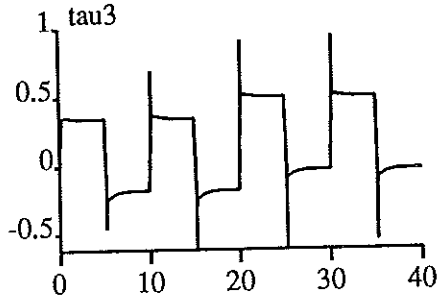
$$P_{qq} = \begin{bmatrix} 0.0001 & 0 \\ 0 & 0.0001 \end{bmatrix}, \Omega = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix}, D = \begin{bmatrix} 0.004 & 0 \\ 0 & 0.004 \end{bmatrix},$$

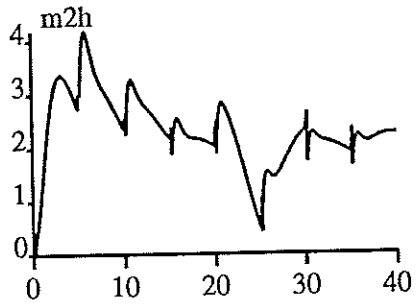
$$P_{\theta\theta} = \begin{bmatrix} 0.02 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.0005 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.001 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.02 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.05 \end{bmatrix}$$

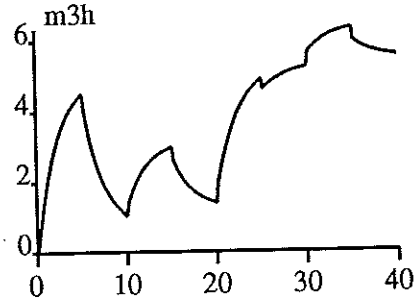


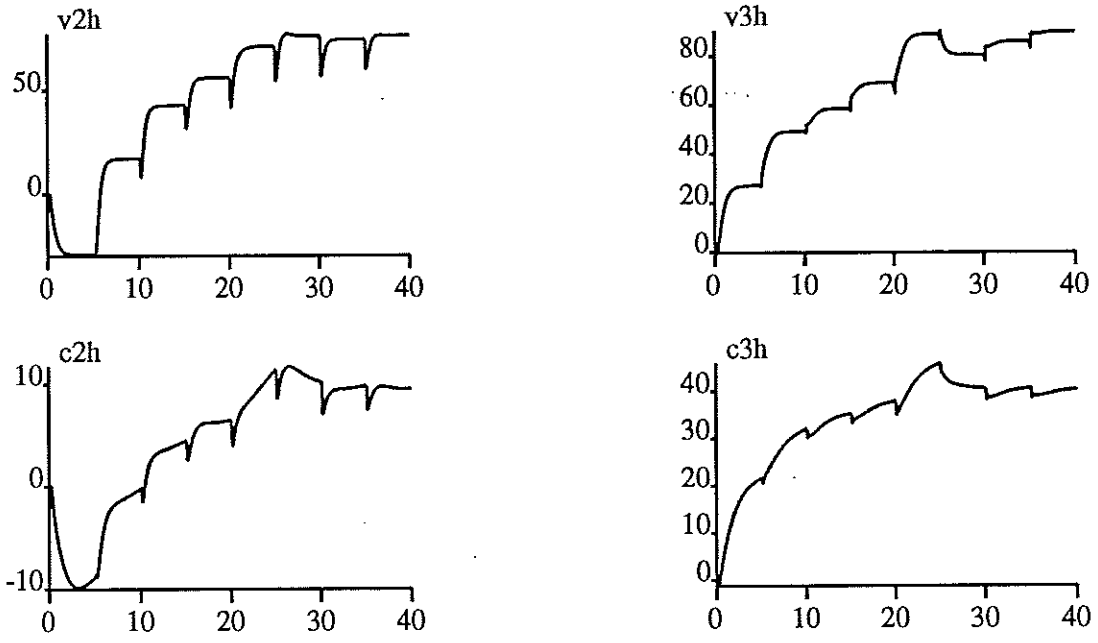








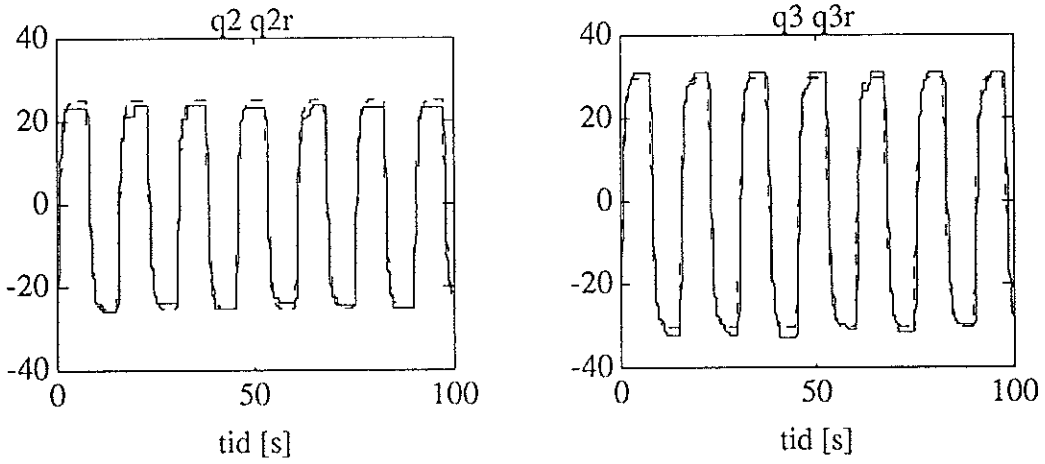


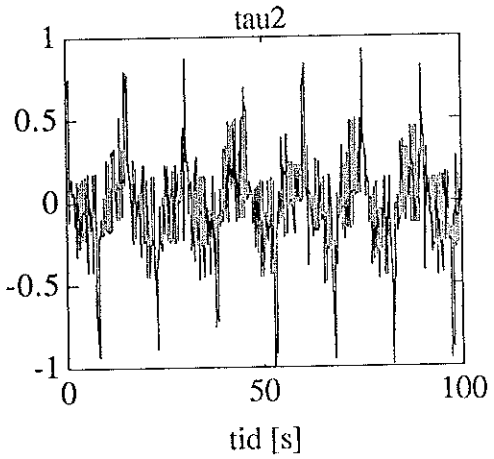


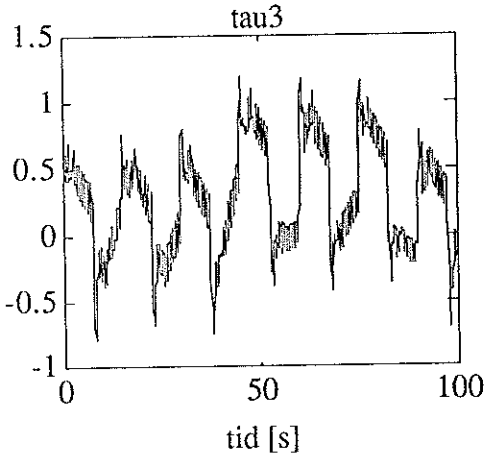
Figur 5.8: Simuleringsresultat för regulator D, där de två nedersta bilderna visar skattningarna av Coulomb-friktionen.

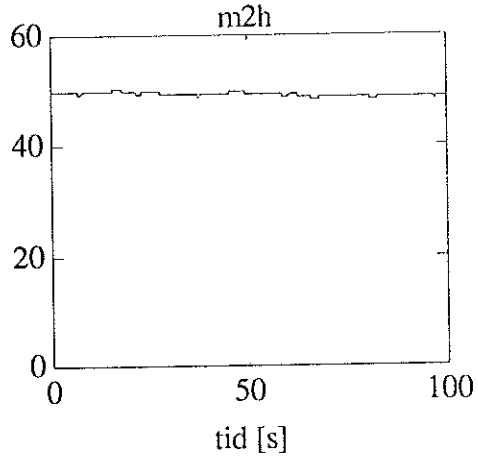
Resultatet av experimenten visas i figur 5.9 nedan. Regulatorparametrarna vi använde vid experimenten var följande:

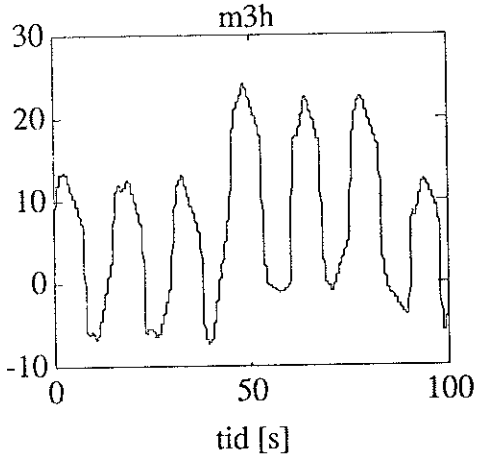
$$P_{qq} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},\ \Omega = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},\ D = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix},\ P_{\theta\theta} = \begin{bmatrix} 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.05 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.05 \end{bmatrix}$$

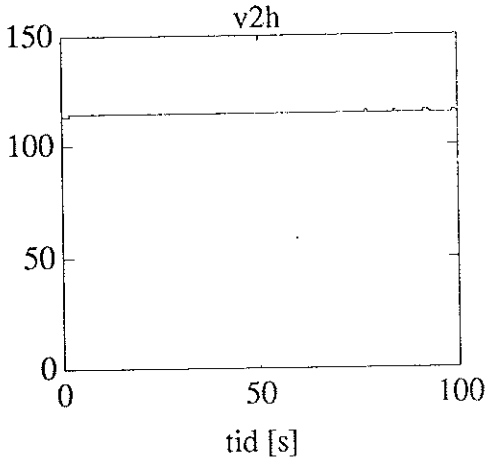


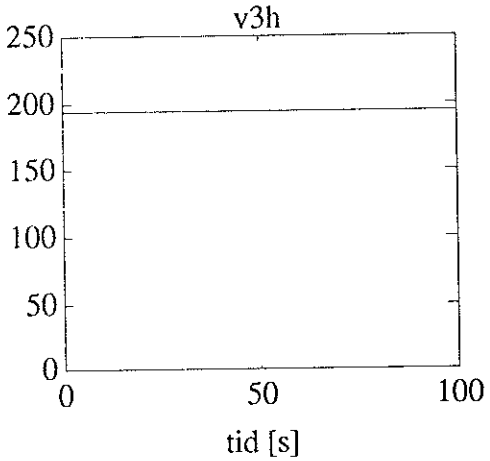

tau2
1
0.5
0
-0.5
-1
0
50
100
tid [s]

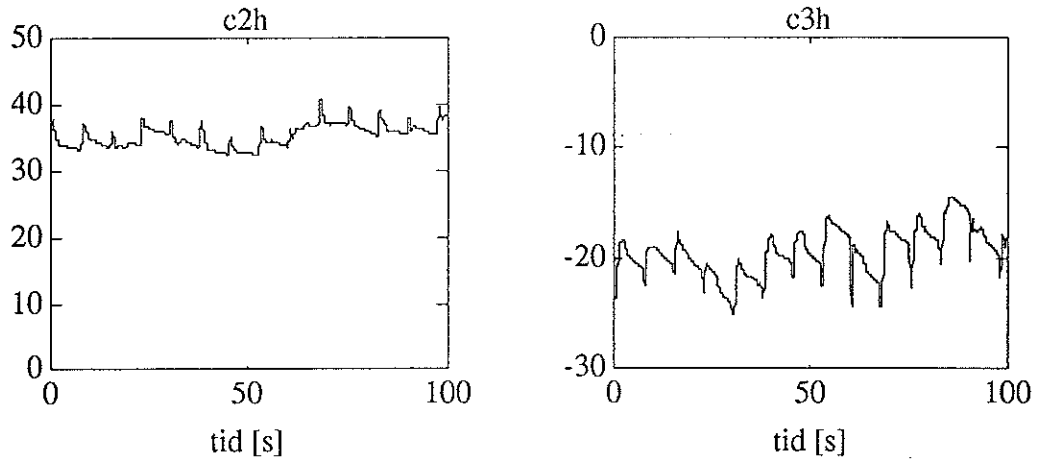

tau3
1.5
1
0.5
0
-0.5
-1
0
50
100
tid [s]


m2h
60
40
20
0
0
50
100
tid [s]


m3h
30
20
10
0
-10
0
50
100
tid [s]


v2h
150
100
50
0
0
50
100
tid [s]


v3h
250
200
150
100
50
0
0
50
100
tid [s]

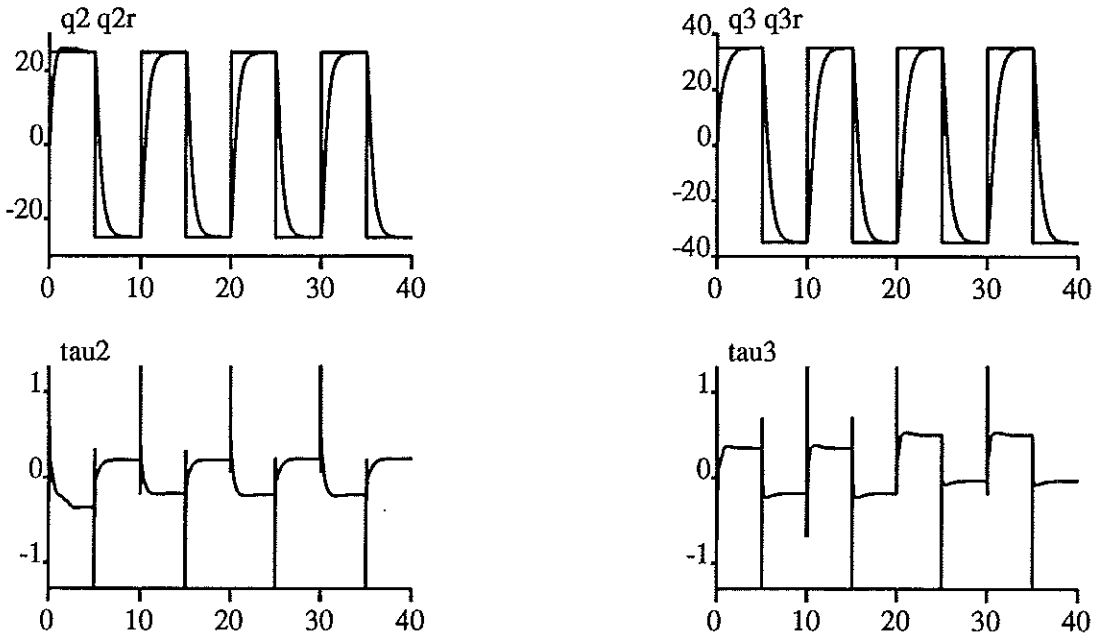


Figur 5.9: Experimentresultat för regulator D. Referenssignal streckad.

E. Diskret PID-regulator.

För att ha någonting att jämföra med, designade vi även en diskret PID-regulator (med anti-wind-up) för roboten. Denna PID-regulator består egentligen av en separat PID-regulator för varje led. Samplingshastigheten var 5 ms, alltså densamma som för de adaptiva regulatorerna ovan. SIMNON-koden för regulatorn finns listad i appendix B.
Figur 5.10 nedan visar resultatet av simuleringarna med PID-regulator. Process-modellen innehåller här både viskös friktion och Coulomb-friktion modellerad med en ramp. Regulatorparametrarna är valda med tanke på att försöka åstadkomma samma snabbhet som för de adaptiva regulatorerna. Vid simuleringarna hade regulatorparametrarna följande värden:

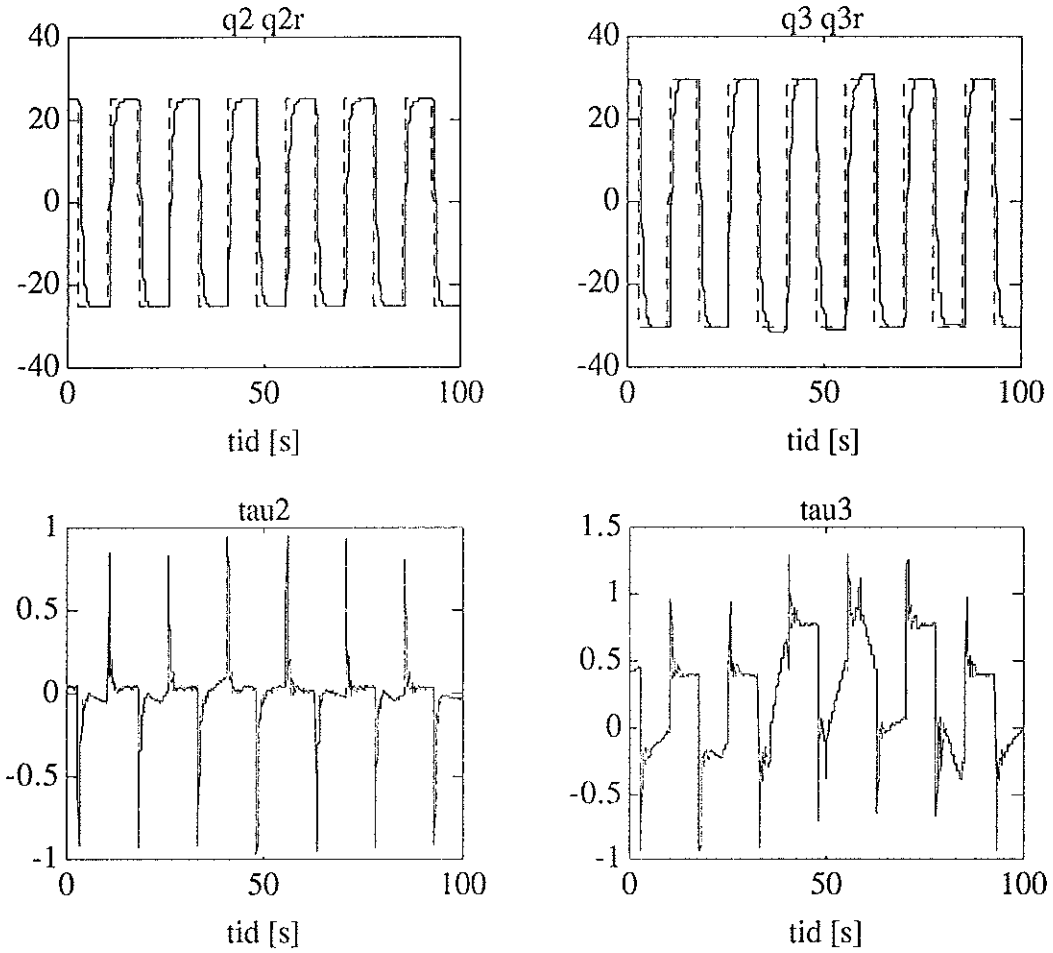
$$K = 0.08,\ T_i = 0.8,\ T_t = 0.8,\ T_d = 0.15,\ N = 10$$



Figur 5.10: Simuleringsresultat för regulator E.

Resultatet av experimenten visas i figur 5.11 nedan. De regulatorparametrar som användes vid experimenten var följande:

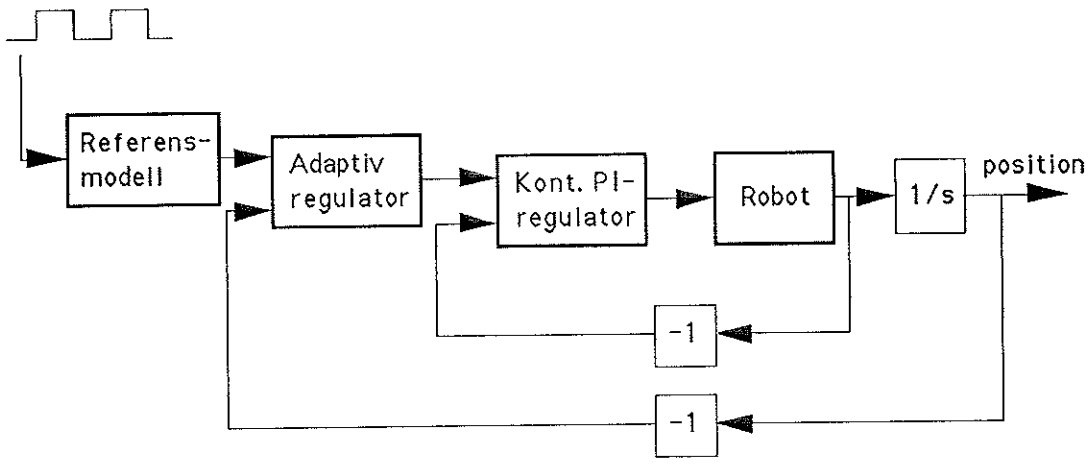
$K = 0.12,\ T_i = 0.8,\ T_t = 0.8,\ T_d = 0.15,\ N = 10$



Figur 5.11: Experimentresultat för regulator E.

F. Adaptiv regulator och PI-regulator i kaskad.

I elektroniken till roboten finns det redan två stycken kontinuerliga regulatorer implementerade, en P-regulator och en PI-regulator med anti-windup. Dessa kan man koppla in och ur med en switch och för alla regulatorer A till E ovan har de varit urkopplade. Av misstag råkade vi dock ha PI-regulatorn inkopplad när vi första gången provade regulator B ovan, och resultatet blev en reglering som var i det närmaste helt perfekt. När vi väl upptäckte vårt "misstag", förstod vi att den fina regleringen inte enbart berodde på den adaptiva regulatorn, utan även på den PI-regulator som nu var kopplad i kaskad med vår adaptiva regulator.
Figur 5.12 nedan visar ett blockdiagram över hur roboten och regulatorerna är ihopkopplade. Det man bör observera, är att den kontinuerliga PI-regulatorn är återkopplad med vinkelhastigheten, och inte positionen, vilket är fallet för de adaptiva regulatorerna och för den diskreta PID-regulatorn ovan. Den adaptiva regulatorn i blockdiagrammet har samma struktur som regulator B.

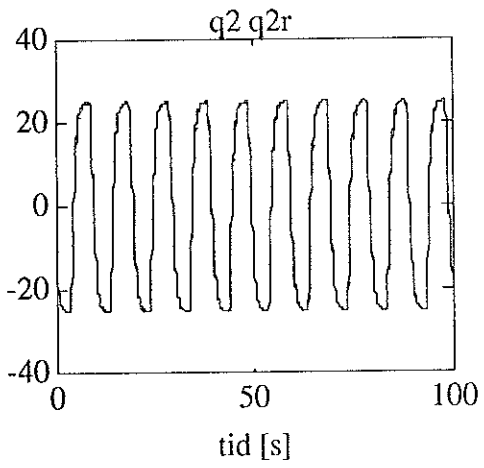


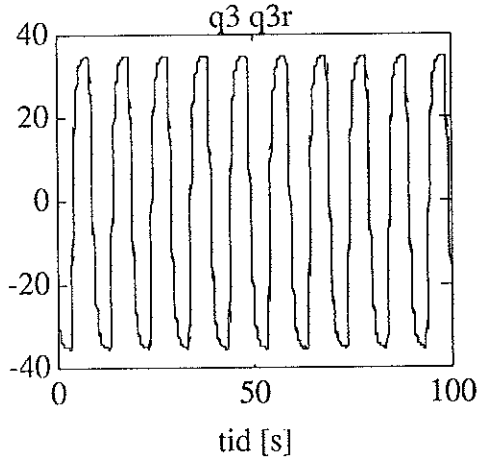
Figur 5.12: Blockdiagram över adaptiv regulator i kaskad med kontinuerlig PI-regulator.

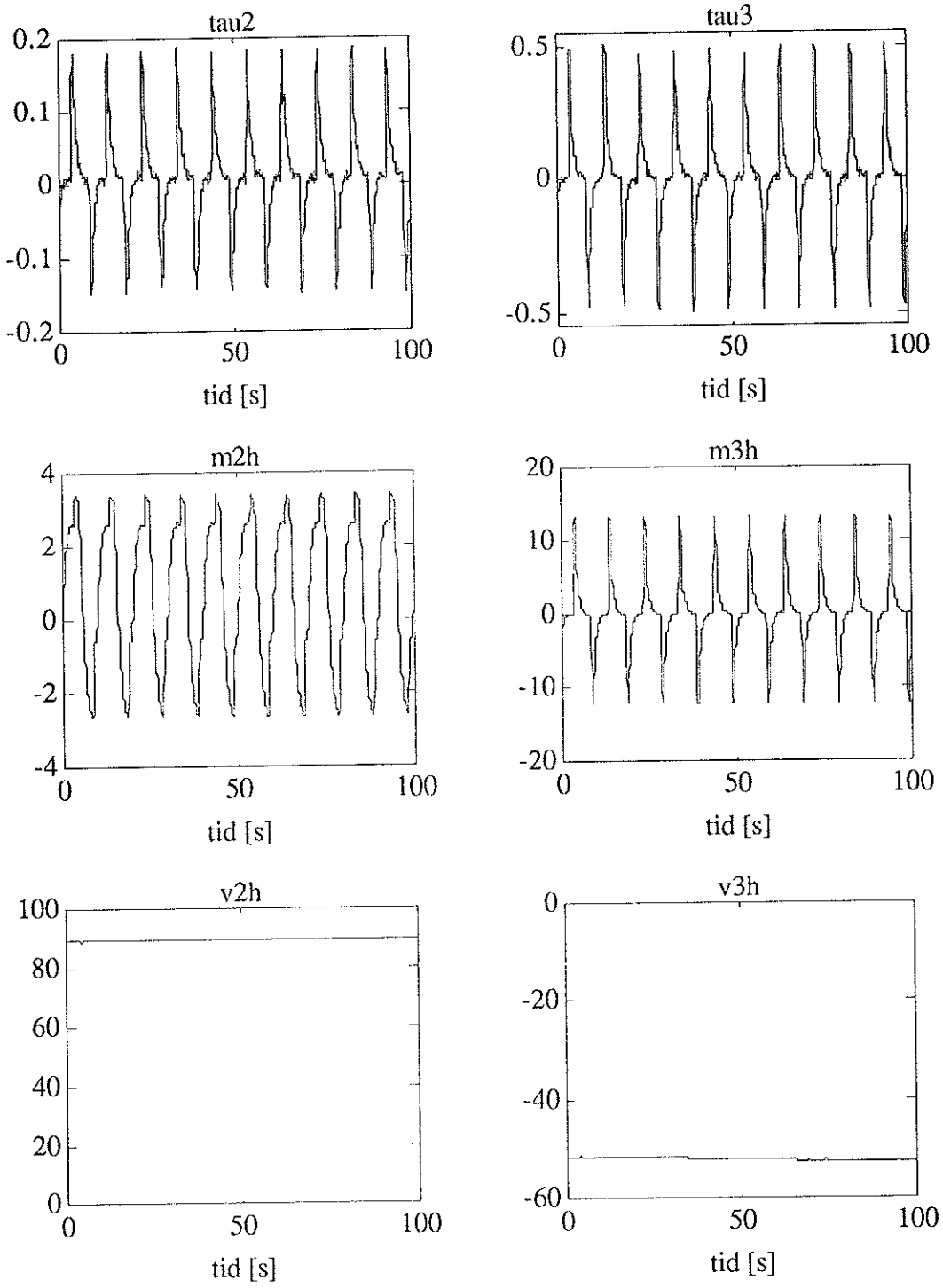
I figur 5.13 nedan visar vi resultatet av regleringen. De styrsignaler som visas (τ_2 och τ_3), är styrsignalerna från den adaptiva regulatorn. Styrsignalerna från PI-regulatorn har vi tyvärr ingen bild på (kunde ej mäta dessa). I den adaptiva regulatorn har nedanstående parametervärden använts. När detta skrivs vet vi tyvärr inte värdena på PI-regulatorns parametrar.

$$P_{qq} = \begin{bmatrix} 0.0001 & 0 \\ 0 & 0.0001 \end{bmatrix},\ \Omega = \begin{bmatrix} 0.001 & 0 \\ 0 & 0.001 \end{bmatrix},\ D = \begin{bmatrix} 0.002 & 0 \\ 0 & 0.002 \end{bmatrix},$$

$$P_{\theta\theta} = \begin{bmatrix} 0.01 & 0 & 0 & 0 \\ 0 & 0.05 & 0 & 0 \\ 0 & 0 & 0.05 & 0 \\ 0 & 0 & 0 & 0.05 \end{bmatrix}$$







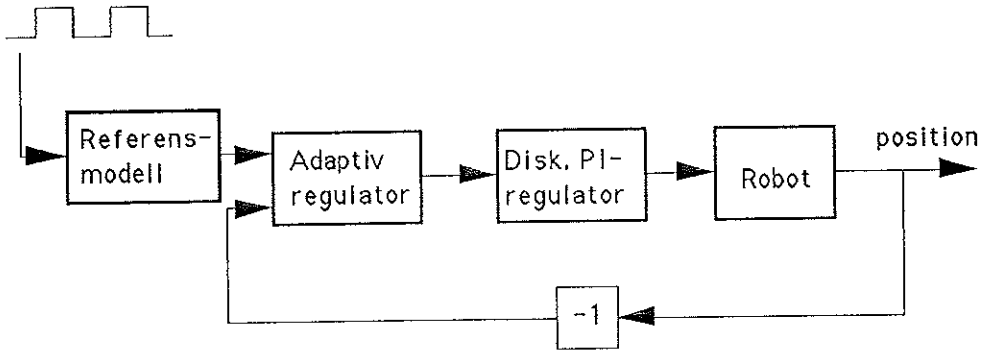
Figur 5.13: Experimentresultat för regulator F. Referenssignalen streckad.

Det kanske mest spektakulära är att det faktiskt inträffar en laststörning på 2.9 kg vid tidpunkten 25 sekunder. Denna ser man ingen inverkan av i bilderna, varken i positionsföljning, styrsignal eller parameterskattning. Detta betyder i sin tur att det helt och hållet är den inre reglerkretsen med den kontinuerliga PI-regulatorn som kompenserar för laststörningen, och så snabbt att vi inte hinner se det i mätningarna. I bilderna ser vi också att regulatorn inte verkar ha några problem med friktionen hos roboten, vilket har varit det stora problemet för de övriga regulatorerna. För att avgöra hur stor betydelse integratorn har i den inre reglerkretsen, provade vi även med den kontinuerliga P-regulatorn istället för PI-regulatorn. Resultatet blev en klart sämre reglering där båda lederna, men framför allt led 2, hade stora problem med att följa referensmodellen.

Den adaptiva regulatorn använder fortfarande robot-modellen som grund för skattningen av de 4 parametrarna ovan. Ur identifieringssynpunkt är dock situationen den, att den adaptiva regulatorn inte identifierar enbart roboten längre, utan en process som består av den inre reglerkretsen (där roboten ingår) "muliplicerat" med en integrator, där den inre reglerkretsen kan sägas vara ungefär lika med 1 upp till en viss gränsfrekvens (ca. 30 Hz). Det är alltså mer tur än skicklighet från vår sida att denna regulator fungerar så bra som den gör (eller egentligen fungerar överhuvudtaget), men trots det är resultatet fortfarande lika intressant och det finns all anledning att studera denna regulatorstruktur närmare.
Man kan dock påpeka att en specialisering av den kontinuerliga PI-regulatorn till en P-regulator skulle medföra att man kan dela upp den hastighetsberoende delen i den adaptiva styrlagen i en diskret och en kontinuerlig del.
Vi kan dessutom nämna att vi av ren nyfikenhet även provade de andra adaptiva regulatorerna ihop med den kontinuerliga PI-regulatorn, men ingen av dem gav så bra resultat som regulator B ovan.

G. Adaptiv regulator med diskret PI-regulator

I samband med att vi provade olika varianter av regulator F och med tanke på att styrsignalerna från de adaptiva regulatorerna emellanåt kan vara ganska brusiga, dök idén upp till den regulatorstruktur som visas i figur 5.14 nedan.

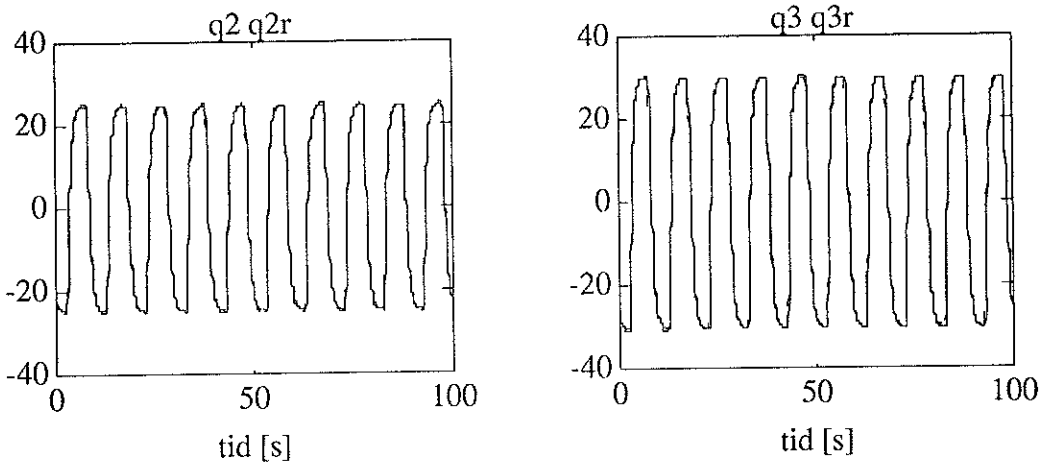


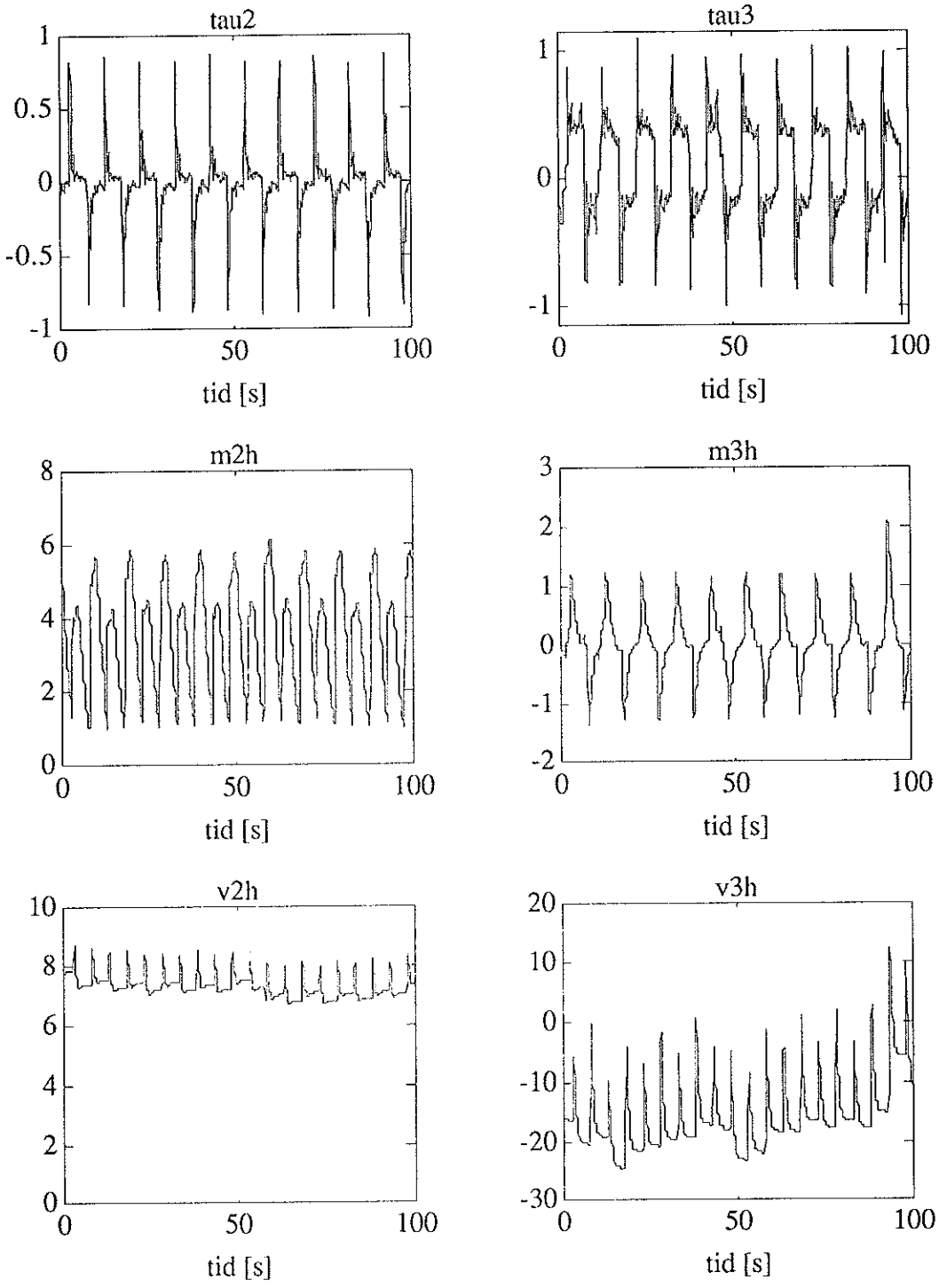
Figur 5.14: Blockdiagram över adaptiv regulator med diskret PI-regulator.

I denna regulator har vi bytt ut den kontinuerliga PI-regulatorn (i regulator F) mot en diskret PI-regulator, och dessutom tagit bort hastighetsåterkopplingen. Regulator G kan därmed ses som ett specialfall av regulator F. De parametrar vi använde i PI-regulatorn var

$$K = 8,\ T_i = 0.65$$

Resultatet av regleringen på roboten visas i figur 5.15 nedan. Styrsignalerna som visas (τ_2 och τ_3) är styrsignalerna efter PI-regulatorn (observera skillnaden jämfört med regulator F).





Figur 5.1.3.10.: Experimentresultat för regulator G.

Parametervärdena för den adaptiva regulatorn redovisas nedan:

$$P_{qq} = \begin{bmatrix} 0.0001 & 0 \\ 0 & 0.0001 \end{bmatrix}, \Omega = \begin{bmatrix} 0.001 & 0 \\ 0 & 0.001 \end{bmatrix}, D = \begin{bmatrix} 0.002 & 0 \\ 0 & 0.002 \end{bmatrix},$$

$$P_{\theta\theta} = \begin{bmatrix} 0.01 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.005 & 0 \\ 0 & 0 & 0 & 0.001 \end{bmatrix}$$

Bilderna ovan innehåller ingen laststörning. Vi kan dock avslöja att regulatorn är något bättre än de övriga regulatorerna ovan på att kompensera för laststörning (naturligtvis med undantag av regulator F).
Ur identifieringssynpunkt är situationen för denna regulator liknande den för regulator F ovan. Den adaptiva regulatorn identifierar en process bestående av den diskreta PI-regulatorn och roboten, med enbart robot-modellen som grund. Relevansen av detta kan man naturligtvis diskutera och man kan säkert göra ett bättre val av modell som grund för skattningen.

5.2. Modell för tre leder

Vi hade egentligen för avsikt att prova den adaptiva regleralgoritmen för fler än två leder, men under tiden som vi gjorde experimenten för två leder kunde vi konstatera att en utvidgning till fler leder skulle öka komplexiteten hos regulatorn i så hög grad att en väsentlig ökning av samplingsperioden hade varit nödvändig. Detta strider emot det vi tidigare kommit fram till om lämpligt val av samplingsperiod (se avsnitt 5.1.3.). Vi redovisar dock här hur den dynamiska robotmodellen för tre leder ser ut (i *eng. joint space*). Denna är framtagen m.h.a. det symboliska matematikprogrammet MAPLE, och de kommandon vi använt finns redovisade i appendix A.

Modell för tre leder:

$$\begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} = \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix} \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \\ \ddot{\theta}_3 \end{bmatrix} + \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} + \begin{bmatrix} G_1 \\ G_2 \\ G_3 \end{bmatrix}$$

där

$$M_{11} = \frac{1}{2} m_3 l_3^2 \cos(2\theta_2 + 2\theta_3) + \frac{1}{2}(m_2 + m_3) l_2^2 \cos(2\theta_2) + m_3 l_2 l_3 \cos(\theta_3) + m_3 l_2 l_3 \cos(2\theta_2 + \theta_3)$$
$$+ \frac{1}{2} m_2 l_2^2 + \frac{1}{2} m_3 l_2^2 + \frac{1}{2} m_3 l_3^2$$
$$M_{12} = M_{13} = M_{21} = 0$$
$$M_{22} = 2 m_3 l_2 l_3 \cos(\theta_3) + m_2 l_2^2 + m_3 l_2^2 + m_3 l_3^2$$
$$M_{23} = m_3 l_2 l_3 \cos(\theta_3) + m_3 l_3^2$$
$$M_{31} = 0$$
$$M_{32} = M_{23}$$
$$M_{33} = m_3 l_3^2$$

$$C_{11} = 0$$
$$C_{12} = -(m_2 + m_3) l_2^2 \sin(2\theta_2)\dot{\theta}_1 - 2 m_3 l_2 l_3 \sin(2\theta_2 + \theta_3)\dot{\theta}_1 - m_3 l_3^2 \sin(2\theta_2 + 2\theta_3)\dot{\theta}_1$$
$$C_{13} = -m_3 l_2 l_3 \sin(\theta_3)\dot{\theta}_1 - m_3 l_3^2 \sin(2\theta_2 + 2\theta_3)\dot{\theta}_1 - m_3 l_2 l_3 \sin(2\theta_2 + \theta_3)\dot{\theta}_1$$
$$C_{21} = \frac{1}{2}(m_2 + m_3) l_2^2 \sin(2\theta_2)\dot{\theta}_1 + m_3 l_2 l_3 \sin(2\theta_2 + \theta_3)\dot{\theta}_1 + \frac{1}{2} m_3 l_3^2 \sin(2\theta_2 + 2\theta_3)\dot{\theta}_1$$
$$C_{22} = -2 m_3 l_2 l_3 \sin(\theta_3)\dot{\theta}_3$$
$$C_{23} = -m_3 l_2 l_3 \sin(\theta_3)\dot{\theta}_3$$
$$C_{31} = \frac{1}{2} m_3 l_2 l_3 \sin(\theta_3)\dot{\theta}_1 + \frac{1}{2} m_3 l_3^2 \sin(2\theta_2 + 2\theta_3)\dot{\theta}_1 + \frac{1}{2} m_3 l_2 l_3 \sin(2\theta_2 + \theta_3)\dot{\theta}_1$$
$$C_{32} = m_3 l_2 l_3 \sin(\theta_3)\dot{\theta}_2$$
$$C_{33} = 0$$

$$G_1 = 0$$
$$G_2 = (m_2 + m_3)l_2 g\cos(\theta_2) + m_3 l_3 g\cos(\theta_2 + \theta_3)$$
$$G_3 = m_3 l_3 g\cos(\theta_2 + \theta_3)$$

6. Sammanfattning och slutsatser

Det finns säkert en del läsare som undrar varför vi inte har modellerat Coulomb-friktionen med en ramp även i regulatorn, då detta borde stämma bättre överens med verkligheten. Anledningen till detta är att modellering av Coulomb-friktionen med en ramp, kräver att man vet storleken på de gravitationskrafter som påverkar lederna. Dessa beror i sin tur på massorna m_2 och m_3, som är okända för regulatorn, och man blir därför tvungen att använda skattningarna av dessa massor för att bestämma storleken på gravitationskrafterna. Läsaren inser därmed säkert vilka problem detta för med sig om skattningarna av massorna inte är bra. I vårt fall är skattningen av massa m_3 alltför orolig för att resultatet skall bli tillfredställande. Om man däremot använder sig av sgn-funktionen istället, undviker man alla problem som skattningen av graviationskrafterna för med sig. I de experiment vi gjorde med regulator C och D med ramp, blev regleringen sämre än för samma regulator med sgn-funktion.

Vi skall nu ge en kort sammanfattning med de viktigaste simulerings- och experimentresultaten för de olika regulatorerna.
Om vi börjar med att kommentera simuleringsresultaten kan vi konstatera att samtliga blev bra (för att inte säga mycket bra). Felet i positionsföljningen blev litet trots laststörningar. Vad gäller styrsignalerna var dessa i det närmaste brusfria. Alla parameterskattningar tenderade att konvergera mot rätt värden, även om konvergensen var långsammare för regulatorerna C och D.
Känsligheten för olika inställningar hos regulatorparametrarna visade sig vara stor och en ändring med en tiopotens kunde medföra att en bra regulator blev så dålig att SIMNON ej "accepterade" den.

När det gäller experimentresultaten kan vi konstatera att en vanlig diskret PID-regulator med samma samplingshastighet, gav en bättre reglering och är bättre på att kompensera för laststörningar än alla de renodlade adaptiva regulatorer vi har testat. Vi kan också konstatera att det inte var möjligt att reglera roboten (med adaptiv regulator) utan att ta hänsyn till friktion (se regulator A). En jämförelse mellan de adaptiva regulatorerna ger att regulator C och D är ungefär lika bra (eller dåliga). Resultatet är dock kanske inte riktigt rättvisande eftersom vi både i simuleringar och experiment endast har använt oss av "diagonalparametrarna". Orsaken till detta är att det annars skulle vara besvärligt att ställa in regulatorn, eftersom man får fler parametrar som samverkar med varandra. Dessutom skulle antalet parametrar att ställa in bli mer än dubbelt så stort. Faktum kvarstår dock att om man hade lyckats trimma den adaptiva regulatorn med utnyttjande av även de andra regulatorparametrarna kanske man hade kunnat åstadkomma en bättre reglering.
Vissa av de skattade parametrarna var oroliga och svängde i takt med perioden hos referenssignalen. En del blev t.o.m. negativa. Kanske beror detta på att friktionen hos roboten inte låter sig modelleras med de enkla modeller vi har använt. Dessutom använde vi oss av en stegfunktion (sgn-funktionen) för att modellera Coulomb-friktion och vi har inga bevis för att skattningarna av parametrarna verkligen konvergerar mot rätt värden då denna används.

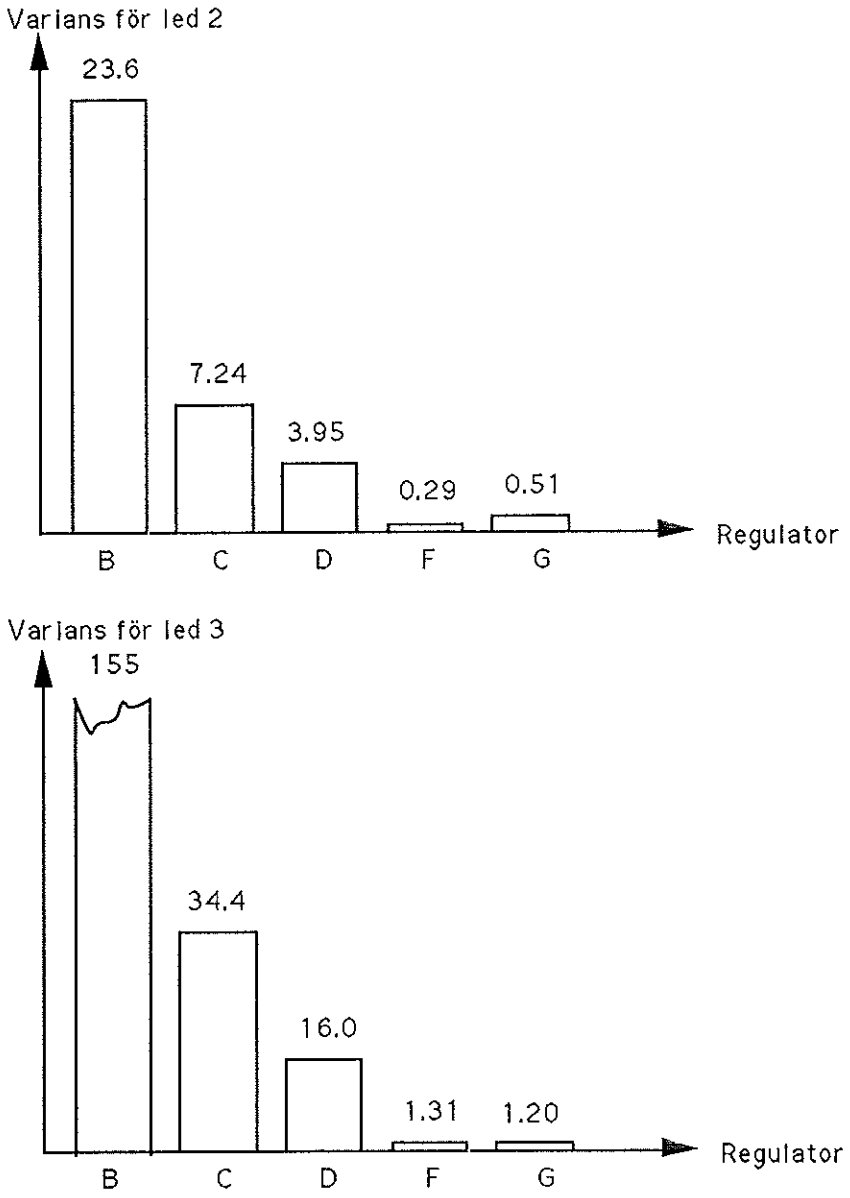
Den uppmärksamme läsaren har säkert lagt märke till att regulatorparametrarna i simuleringar och experiment ibland skiljer sig åt flera tiopotenser, och man kan därför undra om parametrarna från simuleringarna verkligen gick att använda i experimenten (vilket vi kommenterat tidigare). Svaret på denna fråga är ja. Vi har i våra experiment kunnat konstatera att de adaptiva regulatorerna var tämligen okänsliga för inställningen av parametrarna (naturligtvis med undantag av regulator A), och det gick som regel att variera dessa några tiopotenser utan att regleringen påverkades i någon större utsträckning, dock med undantag av $P_{\theta\theta}$. För låga värden på $P_{\theta\theta}$ resulterade nämligen i en ryckig reglering. Det kan även här påpekas att då vi testade regulatorerna på roboten utgick vi hela tiden från de parametrar vi fick fram vid simuleringarna, och trimmade därefter in regulatorerna genom att prova olika parameterinställningar tills vi ansåg att vi ej kunde få en bättre reglering.
Till skillnad från simuleringsresultaten blev styrsignalerna väldigt brusiga vid våra experiment på roboten. Detta visade sig vara väldigt svårt att undvika, och en trimning av regulatorparametrarna för att minska bruset medförde ofta en sämre reglering.

Ett annat problem som kan uppstå vad gäller styrsignalerna, är att dessa blir så stora att de mättar. Detta leder till att skattningarna blir dåliga, eftersom dessa integreras fram (se avsnitt 4). Vi vill däremot påpeka att i de resultat, som redovisas i denna rapport, mättade aldrig styrsignalerna, vilket medförde att vi ej behövde ta hänsyn till detta problem. Vi har dock under arbetets gång erhållit styrsignaler som mättat, och det bästa sättet att undvika integratoruppvridningen visade sig då vara att stänga av adaptionen (parameterskattningen) då styrsignalen mättade, för att se-

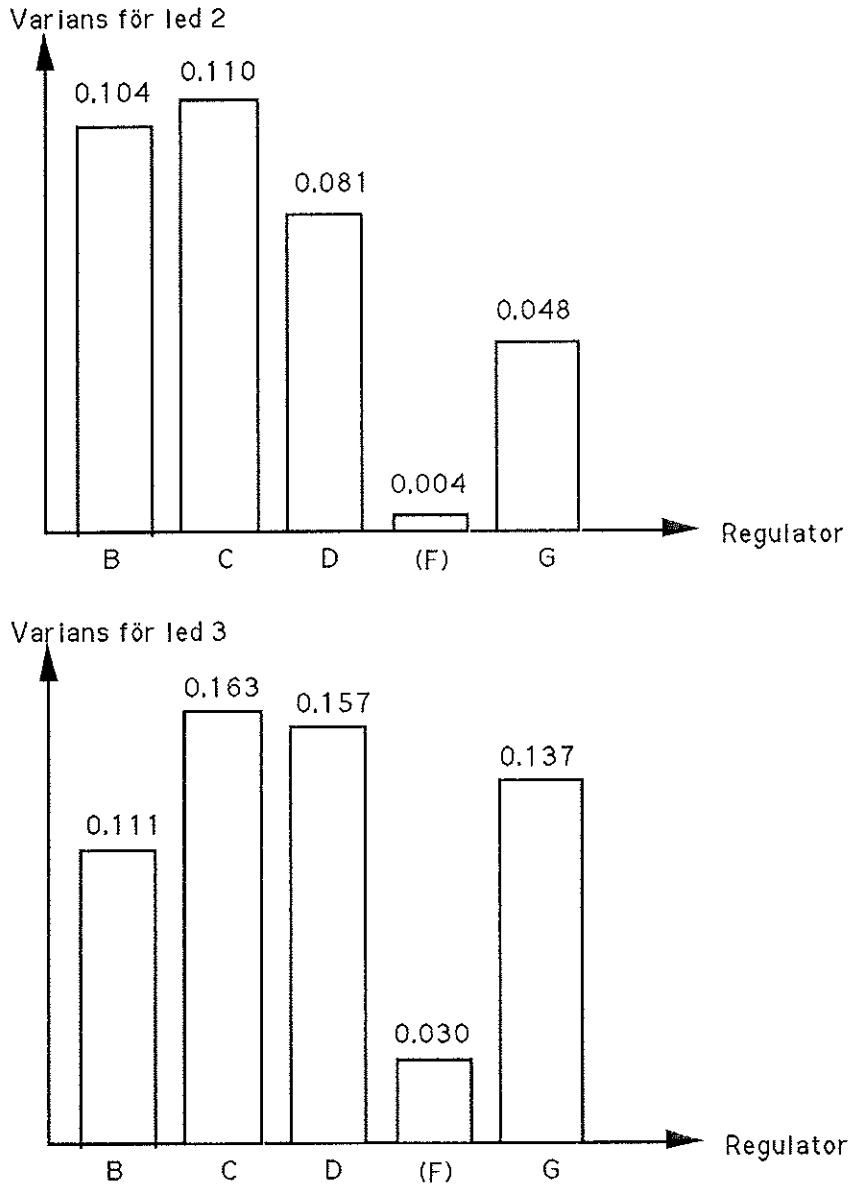
dan hålla adaptionen avstängd till en viss tid efter det att styrsignalen åter kommit in i arbetsområdet.

Vi kunde i våra experiment konstatera att ju kortare samplingsperiod vi använde oss av, desto bättre klarade regulatorn av att hantera friktionsproblem hos robotlederna. Komplexiteten hos regulatorn medförde emellertid, vilket även tidigare påpekats, att vi ej kunde minska samplingsperioden under 5 ms, men det finns alltså tecken som tyder på att en ännu kortare samplingsperiod hade inneburit en ytterligare förbättring av regleringen (jfr. t.ex. med regulator F där det ingår en kontinuerlig PI-regulator).
När det gäller simuleringarna märktes inte problemen med friktion lika tydligt. Detta gäller i första hand positionsföljningen, vilken visserligen försämrades då vi ökade samplingsperioden, men inte i samma utsträckning som vid experimenten på roboten. Vi kunde dock konstatera att styrsignalerna blev brusigare och skattningarna i allmänhet blev sämre vid lägre samplingshastighet.

För att få en något mer objektiv bedömning av de olika regulatorerna visar vi i figur 6.1 och 6.2 variansen hos positionsfelen respektive styrsignalerna för led 2 och 3. Regulator F skiljer sig från de övriga regulatorerna såtillvida att vi inte har kunnat mäta styrsignalerna till robotens motorer (d.v.s. utsignalen från den kontinuerliga PI-regulatorn). Vi visar här istället variansen hos den



Figur 6.1: Variansen hos positionsfelen för led 2 och 3.



Figur 6.2: Variansen hos styrsignalerna för led 2 och 3.

adaptiva regulatorns utsignal. Denna varians är naturligtvis inte direkt jämförbar med de övriga, varför vi i figur 6.2 satt denna regulator inom parentes.

Av alla de regulatorer vi provade på roboten var regulator F den klart bästa. Det verkar därför som om en mycket snabb inre reglerkrets med en integrator är "idealet" för att ta hand om friktionsproblem och laststörningar.

Avslutningsvis kan vi även konstatera att regulator G gav en bättre reglering än de övriga adaptiva regulatorerna (A-D), vad gäller positionsföljning. En PI-regulator, vilken även kan ses som ett filter av lågpasstyp, tillsammans med en adaptiv regulator verkar därför vara positivt för regleringen. Hur detta filter ska se ut och var det ska placeras för att göra störst verkan kan däremot diskuteras, och detta problem kan kanske vara lämpligt som utgångspunkt för ett nytt examensarbete.

Referenser

Bolmsjö, Gunnar S (1989): *Industriell robotteknik.* Studentlitteratur, Lund.

Craig, John J. (1989): *Introduction to Robotics: Mechanics and Control, Second Edition.* Addison-Wesley.

Diaz, Claudio A (1989): *Implementation to Adaptiv Tracking Control for an Industrial Robot.* CODEN: LUTFD2/(TFRT-5395)/1-50/(1989), Lund.

Fowles, Grant R (1986): *Analytical Mechanics. Fourth Edition.* Saunders College Publishing.

Johansson, Rolf (1990): *Adaptive Control of Robot Manipulation Motion.* IEEE Transactions on Robotics and Automation, Vol.6 No 4.

N-Nagy, Francis & Siegler, Andras (1987): *Engineering Foundations of Robotics.* Prentice-Hall.

Åström, Karl J & Wittenmark, Björn (1990): *Computer-Controlled Systems: Theory and Design. Second Edition.* Prentice-Hall.

Åström, Karl J & Wittenmark, Björn (1989): *Adaptive Control.* Addison-Wesley.

Appendix A

Nedan visas de kommandon som använts i Maple vid härledningen av robotmodellen för tre leder. Vi har här använt oss av Newton-Euler-metoden.

• with(linalg);

• T10 := matrix(4,4,[cos(theta1(t)),-sin(theta1(t)),0,0,sin(theta1(t)),
cos(theta1(t)),0,0,0,0,1,0,0,0,0,1]);

• T21 := matrix(4,4,[cos(theta2(t)),-sin(theta2(t)),0,0,0,0,-1,0,
sin(theta2(t)),cos(theta2(t)),0,0,0,0,0,1]);

• T32 := matrix(4,4,[cos(theta3(t)),-sin(theta3(t)),0,l2,
sin(theta3(t)),cos(theta3(t)),0,0,0,0,1,0,0,0,0,1]);

• T43 := matrix(4,4,[cos(theta4(t)),-sin(theta4(t)),0,l3,
sin(theta4(t)),cos(theta4(t)),0,0,0,0,1,0,0,0,0,1]);

Outward Iterations:

• w00 := matrix(3,1,[0,0,0]);

• wp00 := matrix(3,1,[0,0,0]);

• vp00 := matrix(3,1,[0,0,g]);

• w11 :=
evalm(multiply(transpose(submatrix(T10,1..3,1..3)),w00)+diff(theta1(t),t)*matrix(3,1,[0,0,1]));

• wp11 :=
evalm(multiply(transpose(submatrix(T10,1..3,1..3)),wp00)+matrix(3,1,[crossprod(vector([multiply(transpose(submatrix(T10,1..3,1..3)),w00)[1,1],multiply(transpose(submatrix(T10,1..3,1..3)),w00)[2,1],multiply(transpose(submatrix(T10,1..3,1..3)),w00)[3,1]]),vector([diff(theta1(t),t)*matrix(3,1,[0,0,1])[1,1],diff(theta1(t),t)*matrix(3,1,[0,0,1])[2,1],diff(theta1(t),t)*matrix(3,1,[0,0,1])[3,1]]))[1],crossprod(vector([multiply(transpose(submatrix(T10,1..3,1..3)),w00)[1,1],multiply(transpose(submatrix(T10,1..3,1..3)),w00)[2,1],multiply(transpose(submatrix(T10,1..3,1..3)),w00)[3,1]]),vector([diff(theta1(t),t)*matrix(3,1,[0,0,1])[1,1],diff(theta1(t),t)*matrix(3,1,[0,0,1])[2,1],diff(theta1(t),t)*matrix(3,1,[0,0,1])[3,1]]))[2],crossprod(vector([multiply(transpose(submatrix(T10,1..3,1..3)),w00)[1,1],multiply(transpose(submatrix(T10,1..3,1..3)),w00)[2,1],multiply(transpose(submatrix(T10,1..3,1..3)),w00)[3,1]]),vector([diff(theta1(t),t)*matrix(3,1,[0,0,1])[1,1],diff(theta1(t),t)*matrix(3,1,[0,0,1])[2,1],diff(theta1(t),t)*matrix(3,1,[0,0,1])[3,1]]))[3]])+diff(theta1(t),t,t)*matrix(3,1,[0,0,1])));

• vp11temp1 :=
matrix(3,1,[crossprod(vector([wp00[1,1],wp00[2,1],wp00[3,1]]),vector([T10[1,4],T10[2,4],T10[3,4]]))[1],crossprod(vector([wp00[1,1],wp00[2,1],wp00[3,1]]),vector([T10[1,4],T10[2,4],T10[3,4]]))[2],crossprod(vector([wp00[1,1],wp00[2,1],wp00[3,1]]),vector([T10[1,4],T10[2,4],T10[3,4]]))[3]]);

• vp11temp2 :=
matrix(3,1,[crossprod(vector([w00[1,1],w00[2,1],w00[3,1]]),vector([T10[1,4],T10[2,4],T10[3,4]]))[1],crossprod(vector([w00[1,1],w00[2,1],w00[3,1]]),vector([T10[1,4],T10[2,4],T10[3,4]]))[2],crossprod(vector([w00[1,1],w00[2,1],w00[3,1]]),vector([T10[1,4],T10[2,4],T10[3,4]]))[3]]);
• vp11temp3 :=
matrix(3,1,[crossprod(vector([w00[1,1],w00[2,1],w00[3,1]]),vector([vp11temp2[1,1],vp11temp2[2,1],vp11temp2[3,1]]))[1],crossprod(vector([w00[1,1],w00[2,1],w00[3,1]]),vector([vp11temp2[1,1],vp11temp2[2,1],vp11temp2[3,1]]))[2],crossprod(vector([w00[1,1],w00[2,1],w00[3,1]]),vector([vp11temp2[1,1],vp11temp2[2,1],vp11temp2[3,1]]))[3]]);
• vp11 := multiply(transpose(submatrix(T10,1..3,1..3)),vp11temp1+vp11temp3+vp00);

• vcp11temp :=
matrix(3,1,[crossprod(vector([w11[1,1],w11[2,1],w11[3,1]]),vector([T21[1,4],T21[2,4],T21[3,4]]))[1],crossprod(vector([w11[1,1],w11[2,1],w11[3,1]]),vector([T21[1,4],T21[2,4],T21[3,4]]))[2],crossprod(vector([w11[1,1],w11[2,1],w11[3,1]]),vector([T21[1,4],T21[2,4],T21[3,4]]))[3]]);

• vcp11 :=
evalm(matrix(3,1,[crossprod(vector([wp11[1,1],wp11[2,1],wp11[3,1]]),vector([T21[1,4],T21[2,4],T21[3,4]]))[1],crossprod(vector([wp11[1,1],wp11[2,1],wp11[3,1]]),vector([T21[1,4],T21[2,4],T21[3,4]]))[2],crossprod(vector([wp11[1,1],wp11[2,1],wp11[3,1]]),vector([T21[1,4],T21[2,4],T21[3,4]]))[3]])+matrix(3,1,[cro

```
ssprod(vector([w11[1,1],w11[2,1],w11[3,1]]),vector([vcp11temp[1,1],vcp11temp[2,1],vcp11temp[3,1]]))
[1],crossprod(vector([w11[1,1],w11[2,1],w11[3,1]]),vector([vcp11temp[1,1],vcp11temp[2,1],vcp11temp
[3,1]]))[2],crossprod(vector([w11[1,1],w11[2,1],w11[3,1]]),vector([vcp11temp[1,1],vcp11temp[2,1],vcp
11temp[3,1]]))[3]])+matrix(3,1,[vp11[1,1],vp11[2,1],vp11[3,1]]));
```

• F11 := evalm(m1*vcp11);

• N11 := matrix(3,1,[0,0,0]);

• w22 := evalm(multiply(transpose(submatrix(T21,1..3,1..3)),w11)+diff(theta2(t),t)*matrix(3,1,[0,0,1]));

```
• wp22 :=
evalm(multiply(transpose(submatrix(T21,1..3,1..3)),wp11)+matrix(3,1,[crossprod(vector([multiply(transp
ose(submatrix(T21,1..3,1..3)),w11)[1,1],multiply(transpose(submatrix(T21,1..3,1..3)),w11)[2,1],multiply(
transpose(submatrix(T21,1..3,1..3)),w11)[3,1]]),vector([diff(theta2(t),t)*matrix(3,1,[0,0,1])[1,1],diff(thet
a2(t),t)*matrix(3,1,[0,0,1])[2,1],diff(theta2(t),t)*matrix(3,1,[0,0,1])[3,1]]))[1],crossprod(vector([multiply(t
ranspose(submatrix(T21,1..3,1..3)),w11)[1,1],multiply(transpose(submatrix(T21,1..3,1..3)),w11)[2,1],mu
ltiply(transpose(submatrix(T21,1..3,1..3)),w11)[3,1]]),vector([diff(theta2(t),t)*matrix(3,1,[0,0,1])[1,1],diff
(theta2(t),t)*matrix(3,1,[0,0,1])[2,1],diff(theta2(t),t)*matrix(3,1,[0,0,1])[3,1]]))[2],crossprod(vector([multi
ply(transpose(submatrix(T21,1..3,1..3)),w11)[1,1],multiply(transpose(submatrix(T21,1..3,1..3)),w11)[2,1
],multiply(transpose(submatrix(T21,1..3,1..3)),w11)[3,1]]),vector([diff(theta2(t),t)*matrix(3,1,[0,0,1])[1,1
],diff(theta2(t),t)*matrix(3,1,[0,0,1])[2,1],diff(theta2(t),t)*matrix(3,1,[0,0,1])[3,1]]))[3]])+diff(theta2(t),t,t)
*matrix(3,1,[0,0,1]));
```

```
• vp22temp1 :=
matrix(3,1,[crossprod(vector([wp11[1,1],wp11[2,1],wp11[3,1]]),vector([T21[1,4],T21[2,4],T21[3,4]]))[1],
crossprod(vector([wp11[1,1],wp11[2,1],wp11[3,1]]),vector([T21[1,4],T21[2,4],T21[3,4]]))[2],crossprod(v
ector([wp11[1,1],wp11[2,1],wp11[3,1]]),vector([T21[1,4],T21[2,4],T21[3,4]]))[3]]);
```

```
• vp22temp2 :=
matrix(3,1,[crossprod(vector([w11[1,1],w11[2,1],w11[3,1]]),vector([T21[1,4],T21[2,4],T21[3,4]]))[1],cro
ssprod(vector([w11[1,1],w11[2,1],w11[3,1]]),vector([T21[1,4],T21[2,4],T21[3,4]]))[2],crossprod(vector([
w11[1,1],w11[2,1],w11[3,1]]),vector([T21[1,4],T21[2,4],T21[3,4]]))[3]]);
```

```
• vp22temp3 :=
matrix(3,1,[crossprod(vector([w11[1,1],w11[2,1],w11[3,1]]),vector([vp22temp2[1,1],vp22temp2[2,1],vp
22temp2[3,1]]))[1],crossprod(vector([w11[1,1],w11[2,1],w11[3,1]]),vector([vp22temp2[1,1],vp22temp2[
2,1],vp22temp2[3,1]]))[2],crossprod(vector([w11[1,1],w11[2,1],w11[3,1]]),vector([vp22temp2[1,1],vp22
temp2[2,1],vp22temp2[3,1]]))[3]]);
```

• vp22 := multiply(transpose(submatrix(T21,1..3,1..3)),vp22temp1+vp22temp3+vp11);

```
• vcp22temp :=
matrix(3,1,[crossprod(vector([w22[1,1],w22[2,1],w22[3,1]]),vector([T32[1,4],T32[2,4],T32[3,4]]))[1],cro
ssprod(vector([w22[1,1],w22[2,1],w22[3,1]]),vector([T32[1,4],T32[2,4],T32[3,4]]))[2],crossprod(vector([
w22[1,1],w22[2,1],w22[3,1]]),vector([T32[1,4],T32[2,4],T32[3,4]]))[3]]);
```

```
• vcp22 :=
evalm(matrix(3,1,[crossprod(vector([wp22[1,1],wp22[2,1],wp22[3,1]]),vector([T32[1,4],T32[2,4],T32[3,4
]]))[1],crossprod(vector([wp22[1,1],wp22[2,1],wp22[3,1]]),vector([T32[1,4],T32[2,4],T32[3,4]]))[2],cross
prod(vector([wp22[1,1],wp22[2,1],wp22[3,1]]),vector([T32[1,4],T32[2,4],T32[3,4]]))[3]])+matrix(3,1,[cro
ssprod(vector([w22[1,1],w22[2,1],w22[3,1]]),vector([vcp22temp[1,1],vcp22temp[2,1],vcp22temp[3,1]]))
[1],crossprod(vector([w22[1,1],w22[2,1],w22[3,1]]),vector([vcp22temp[1,1],vcp22temp[2,1],vcp22temp
[3,1]]))[2],crossprod(vector([w22[1,1],w22[2,1],w22[3,1]]),vector([vcp22temp[1,1],vcp22temp[2,1],vcp
22temp[3,1]]))[3]])+matrix(3,1,[vp22[1,1],vp22[2,1],vp22[3,1]]));
```

• F22 := evalm(m2*vcp22);

• N22 := matrix(3,1,[0,0,0]);

• w33 := evalm(multiply(transpose(submatrix(T32,1..3,1..3)),w22)+diff(theta3(t),t)*matrix(3,1,[0,0,1]));

```
• wp33 :=
evalm(multiply(transpose(submatrix(T32,1..3,1..3)),wp22)+matrix(3,1,[crossprod(vector([multiply(transp
ose(submatrix(T32,1..3,1..3)),w22)[1,1],multiply(transpose(submatrix(T32,1..3,1..3)),w22)[2,1],multiply(
transpose(submatrix(T32,1..3,1..3)),w22)[3,1]]),vector([diff(theta3(t),t)*matrix(3,1,[0,0,1])[1,1],diff(thet
a3(t),t)*matrix(3,1,[0,0,1])[2,1],diff(theta3(t),t)*matrix(3,1,[0,0,1])[3,1]]))[1],crossprod(vector([multiply(t
ranspose(submatrix(T32,1..3,1..3)),w22)[1,1],multiply(transpose(submatrix(T32,1..3,1..3)),w22)[2,1],mu
ltiply(transpose(submatrix(T32,1..3,1..3)),w22)[3,1]]),vector([diff(theta3(t),t)*matrix(3,1,[0,0,1])[1,1],diff
(theta3(t),t)*matrix(3,1,[0,0,1])[2,1],diff(theta3(t),t)*matrix(3,1,[0,0,1])[3,1]]))[2],crossprod(vector([multi
ply(transpose(submatrix(T32,1..3,1..3)),w22)[1,1],multiply(transpose(submatrix(T32,1..3,1..3)),w22)[2,1
```

],multiply(transpose(submatrix(T32,1..3,1..3)),w22)[3,1]]),vector([diff(theta3(t),t)*matrix(3,1,[0,0,1])[1,1
],diff(theta3(t),t)*matrix(3,1,[0,0,1])[2,1],diff(theta3(t),t)*matrix(3,1,[0,0,1])[3,1]])[3]])+diff(theta3(t),t,t)
*matrix(3,1,[0,0,1]));

• vp33temp1 :=
matrix(3,1,[crossprod(vector([wp22[1,1],wp22[2,1],wp22[3,1]]),vector([T32[1,4],T32[2,4],T32[3,4]]))[1],
crossprod(vector([wp22[1,1],wp22[2,1],wp22[3,1]]),vector([T32[1,4],T32[2,4],T32[3,4]]))[2],crossprod(v
ector([wp22[1,1],wp22[2,1],wp22[3,1]]),vector([T32[1,4],T32[2,4],T32[3,4]]))[3]]);

• vp33temp2 :=
matrix(3,1,[crossprod(vector([w22[1,1],w22[2,1],w22[3,1]]),vector([T32[1,4],T32[2,4],T32[3,4]]))[1],cro
ssprod(vector([w22[1,1],w22[2,1],w22[3,1]]),vector([T32[1,4],T32[2,4],T32[3,4]]))[2],crossprod(vector([
w22[1,1],w22[2,1],w22[3,1]]),vector([T32[1,4],T32[2,4],T32[3,4]]))[3]]);

• vp33temp3 :=
matrix(3,1,[crossprod(vector([w22[1,1],w22[2,1],w22[3,1]]),vector([vp33temp2[1,1],vp33temp2[2,1],vp
33temp2[3,1]]))[1],crossprod(vector([w22[1,1],w22[2,1],w22[3,1]]),vector([vp33temp2[1,1],vp33temp2[
2,1],vp33temp2[3,1]]))[2],crossprod(vector([w22[1,1],w22[2,1],w22[3,1]]),vector([vp33temp2[1,1],vp33
temp2[2,1],vp33temp2[3,1]]))[3]]);
• vp33tmp := multiply(transpose(submatrix(T32,1..3,1..3)),vp33temp1+vp33temp3+vp22);

• vp33 :=
matrix(3,1,[combine(vp33tmp[1,1],trig),combine(vp33tmp[2,1],trig),combine(vp33tmp[3,1],trig)]);

• vcp33temp :=
matrix(3,1,[crossprod(vector([w33[1,1],w33[2,1],w33[3,1]]),vector([T43[1,4],T43[2,4],T43[3,4]]))[1],cro
ssprod(vector([w33[1,1],w33[2,1],w33[3,1]]),vector([T43[1,4],T43[2,4],T43[3,4]]))[2],crossprod(vector([
w33[1,1],w33[2,1],w33[3,1]]),vector([T43[1,4],T43[2,4],T43[3,4]]))[3]]);

• vcp33 :=
evalm(matrix(3,1,[crossprod(vector([wp33[1,1],wp33[2,1],wp33[3,1]]),vector([T43[1,4],T43[2,4],T43[3,4
]]))[1],crossprod(vector([wp33[1,1],wp33[2,1],wp33[3,1]]),vector([T43[1,4],T43[2,4],T43[3,4]]))[2],cross
prod(vector([wp33[1,1],wp33[2,1],wp33[3,1]]),vector([T43[1,4],T43[2,4],T43[3,4]]))[3]])+matrix(3,1,[cro
ssprod(vector([w33[1,1],w33[2,1],w33[3,1]]),vector([vcp33temp[1,1],vcp33temp[2,1],vcp33temp[3,1]]))
[1],crossprod(vector([w33[1,1],w33[2,1],w33[3,1]]),vector([vcp33temp[1,1],vcp33temp[2,1],vcp33temp
[3,1]]))[2],crossprod(vector([w33[1,1],w33[2,1],w33[3,1]]),vector([vcp33temp[1,1],vcp33temp[2,1],vcp
33temp[3,1]]))[3]])+matrix(3,1,[vp33[1,1],vp33[2,1],vp33[3,1]]));

• F33 := evalm(m3*vcp33);

• N33 := matrix(3,1,[0,0,0]);

Inward Iterations:

• f44 := matrix(3,1,[0,0,0]);

• n44 := matrix(3,1,[0,0,0]);

• f33 := evalm(multiply(submatrix(T43,1..3,1..3),f44)+F33);

• n33 :=
evalm(N33+multiply(submatrix(T43,1..3,1..3),n44)+matrix(3,1,[crossprod(vector([T43[1,4],T43[2,4],T43[
3,4]]),vector([F33[1,1],F33[2,1],F33[3,1]]))[1],crossprod(vector([T43[1,4],T43[2,4],T43[3,4]]),vector([F3
3[1,1],F33[2,1],F33[3,1]]))[2],crossprod(vector([T43[1,4],T43[2,4],T43[3,4]]),vector([F33[1,1],F33[2,1],
F33[3,1]]))[3]])+matrix(3,1,[crossprod(vector([T43[1,4],T43[2,4],T43[3,4]]),vector([multiply(submatrix(T
43,1..3,1..3),f44)[1,1],multiply(submatrix(T43,1..3,1..3),f44)[2,1],multiply(submatrix(T43,1..3,1..3),f44)[
3,1]]))[1],crossprod(vector([T43[1,4],T43[2,4],T43[3,4]]),vector([multiply(submatrix(T43,1..3,1..3),f44)[
1,1],multiply(submatrix(T43,1..3,1..3),f44)[2,1],multiply(submatrix(T43,1..3,1..3),f44)[3,1]]))[2],crosspro
d(vector([T43[1,4],T43[2,4],T43[3,4]]),vector([multiply(submatrix(T43,1..3,1..3),f44)[1,1],multiply(subm
atrix(T43,1..3,1..3),f44)[2,1],multiply(submatrix(T43,1..3,1..3),f44)[3,1]]))[3]]));

• tau3 :=
collect(combine(multiply(transpose(n33),matrix(3,1,[0,0,1]))[1,1],trig),[diff(theta1(t),t,t),diff(theta2(t),t,t)
,diff(theta3(t),t,t),diff(theta1(t),t),diff(theta2(t),t),diff(theta3(t),t)]);

• f22 := evalm(multiply(submatrix(T32,1..3,1..3),f33)+F22);
• n22 :=
evalm(N22+multiply(submatrix(T32,1..3,1..3),n33)+matrix(3,1,[crossprod(vector([T32[1,4],T32[2,4],T32[
3,4]]),vector([F22[1,1],F22[2,1],F22[3,1]]))[1],crossprod(vector([T32[1,4],T32[2,4],T32[3,4]]),vector([F2
2[1,1],F22[2,1],F22[3,1]]))[2],crossprod(vector([T32[1,4],T32[2,4],T32[3,4]]),vector([F22[1,1],F22[2,1],

F22[3,1]]))[3]])+matrix(3,1,[crossprod(vector([T32[1,4],T32[2,4],T32[3,4]]),vector([multiply(submatrix(T
32,1..3,1..3),f33)[1,1],multiply(submatrix(T32,1..3,1..3),f33)[2,1],multiply(submatrix(T32,1..3,1..3),f33)[
3,1]]))[1],crossprod(vector([T32[1,4],T32[2,4],T32[3,4]]),vector([multiply(submatrix(T32,1..3,1..3),f33)[
1,1],multiply(submatrix(T32,1..3,1..3),f33)[2,1],multiply(submatrix(T32,1..3,1..3),f33)[3,1]]))[2],crosspro
d(vector([T32[1,4],T32[2,4],T32[3,4]]),vector([multiply(submatrix(T32,1..3,1..3),f33)[1,1],multiply(subm
atrix(T32,1..3,1..3),f33)[2,1],multiply(submatrix(T32,1..3,1..3),f33)[3,1]]))[3]]));

• tau2 :=
collect(combine(multiply(transpose(n22),matrix(3,1,[0,0,1]))[1,1],trig),[diff(theta1(t),t,t),diff(theta2(t),t,t)
,diff(theta3(t),t,t),diff(theta1(t),t),diff(theta2(t),t),diff(theta3(t),t)]);

• f11 := evalm(multiply(submatrix(T21,1..3,1..3),f22)+F11);

• n11 :=
evalm(N11+multiply(submatrix(T21,1..3,1..3),n22)+matrix(3,1,[crossprod(vector([T21[1,4],T21[2,4],T21[
3,4]]),vector([F11[1,1],F11[2,1],F11[3,1]]))[1],crossprod(vector([T21[1,4],T21[2,4],T21[3,4]]),vector([F1
1[1,1],F11[2,1],F11[3,1]]))[2],crossprod(vector([T21[1,4],T21[2,4],T21[3,4]]),vector([F11[1,1],F11[2,1],
F11[3,1]]))[3]])+matrix(3,1,[crossprod(vector([T21[1,4],T21[2,4],T21[3,4]]),vector([multiply(submatrix(T
21,1..3,1..3),f22)[1,1],multiply(submatrix(T21,1..3,1..3),f22)[2,1],multiply(submatrix(T21,1..3,1..3),f22)[
3,1]]))[1],crossprod(vector([T21[1,4],T21[2,4],T21[3,4]]),vector([multiply(submatrix(T21,1..3,1..3),f22)[
1,1],multiply(submatrix(T21,1..3,1..3),f22)[2,1],multiply(submatrix(T21,1..3,1..3),f22)[3,1]]))[2],crosspro
d(vector([T21[1,4],T21[2,4],T21[3,4]]),vector([multiply(submatrix(T21,1..3,1..3),f22)[1,1],multiply(subm
atrix(T21,1..3,1..3),f22)[2,1],multiply(submatrix(T21,1..3,1..3),f22)[3,1]]))[3]]));

• tau1 :=
collect(combine(multiply(transpose(n11),matrix(3,1,[0,0,1]))[1,1],trig),[diff(theta1(t),t,t),diff(theta2(t),t,t)
,diff(theta3(t),t,t),diff(theta1(t),t),diff(theta2(t),t),diff(theta3(t),t)]);

Appendix B

Nedan följer SIMNON-koden för regulatorerna A-E samt process-modellen. Dessutom finns filen Regmod.t, vilken innehåller de procedurer och deklarationer som behöver inkluderas vid genereringen av Modula-2-kod.

```
discrete system controller_A

"SIMNON-code for controller A

"This command is needed for SIM2VME
"% modula-file regmod.t

state mh2 mh3
state mhi2 mhi3
state q2old q3old qr21old qr22old qr31old qr32old
state r21old r22old r31old r32old
new nmh2 nmh3
new nmhi2 nmhi3
new nq2old nq3old nqr21old nqr22old nqr31old nqr32old
new nr21old nr22old nr31old nr32old
input q2 q3
output tau2 tau3
time t
tsamp ts

INITIAL

detPqq=Pqq11*Pqq22-Pqq12*Pqq21
invPqq11=Pqq22/detPqq
invPqq12=-Pqq12/detPqq
invPqq21=-Pqq21/detPqq
invPqq22=Pqq11/detPqq

detomega=omega11*omega22-omega12*omega21
invom11=omega22/detomega
invom12=-omega12/detomega
invom21=-omega21/detomega
invom22=omega11/detomega

P1211=invPqq11*omega11+invPqq12*omega21
P1212=invPqq11*omega12+invPqq12*omega22
P1221=invPqq21*omega11+invPqq22*omega21
P1222=invPqq21*omega12+invPqq22*omega22

Pqqiom11=Pqq11*invom11+Pqq12*invom21
Pqqiom12=Pqq11*invom12+Pqq12*invom22
Pqqiom21=Pqq21*invom11+Pqq22*invom21
Pqqiom22=Pqq21*invom12+Pqq22*invom22

temp111=Pqqiom11*Pqq11+Pqqiom12*Pqq21+D11
temp112=Pqqiom11*Pqq12+Pqqiom12*Pqq22+D12
temp121=Pqqiom21*Pqq11+Pqqiom22*Pqq21+D21
temp122=Pqqiom21*Pqq12+Pqqiom22*Pqq22+D22

DiPqq11=D11*invPqq11+D12*invPqq21
DiPqq12=D11*invPqq12+D12*invPqq22
DiPqq21=D21*invPqq11+D22*invPqq21
DiPqq22=D21*invPqq12+D22*invPqq22

temp211=DiPqq11*omega11+DiPqq12*omega21
temp212=DiPqq11*omega12+DiPqq12*omega22
temp221=DiPqq21*omega11+DiPqq22*omega21
temp222=DiPqq21*omega12+DiPqq22*omega22

SORT

a1=-2*exp(-ak*h)
a2=exp(-2*ak*h)

b1=1-exp(-ak*h)*(1+ak*h)
b2=exp(-ak*h)*(exp(-ak*h)+ak*h-1)

r2=if mod(t,per2)<per2/2 then step2 else -step2
r3=if mod(t,per3)<per3/2 then step3 else -step3

qr2=-a1*qr21old-a2*qr22old+b1*r21old+b2*r22old
qr3=-a1*qr31old-a2*qr32old+b1*r31old+b2*r32old
nqr21old=qr2
nqr31old=qr3
nqr22old=qr21old
nqr32old=qr31old
qr2p=(qr2-qr21old)/h
qr3p=(qr3-qr31old)/h
qr2pp=(qr2-2*qr21old+qr22old)/h
qr3pp=(qr3-2*qr31old+qr32old)/h
nr21old=r2
nr31old=r3
nr22old=r21old
nr32old=r31old

q2p=(q2-q2old)/h
q3p=(q3-q3old)/h

x2=n2*q2/(2*pi)+x02
x3=n3*q3/(2*pi)+x03

th2=(270-b-a)*pi/180-arccos((D*D+E*E-x2*x2)/(2*D*E))
th3=(450-c)*pi/180-th2-arccos((M*M+L*L-x3*x3)/(2*M*L))

J2k=(D*D+E*E-x2*x2)/(2*D*E)
J3k=(M*M+L*L-x3*x3)/(2*M*L)

J22=-n2*x2/(2*pi*D*E*sqrt(1-J2k*J2k))
"J23=0
"J32=-J22
J33=-n3*x3/(2*pi*M*L*sqrt(1-J3k*J3k))

J221p=n2*n2*q2p/(4*pi*pi*D*E*sqrt(1-J2k*J2k))
J222p=J2k*x2*x2/(D*E*(1-J2k*J2k))-1
J22p=J221p*J222p
"J23p=0
"J32p=-J22p
J331p=n3*n3*q3p/(4*pi*pi*M*L*sqrt(1-J3k*J3k))
J332p=J3k*x3*x3/(M*L*(1-J3k*J3k))-1
J33p=J331p*J332p

q2t=q2-qr2
q2pt=q2p-qr2p
q3t=q3-qr3
q3pt=q3p-qr3p

u1=q2pt+P1211*q2t+P1212*q3t
u2=q3pt+P1221*q2t+P1222*q3t

v1=qr2pp-P1211*q2pt-P1212*q3pt
v2=qr3pp-P1221*q2pt-P1222*q3pt

psi11=J22*J22p*l2*l2*(-u1+q2p)+J22*J22*l2*l2*v1+J22*l2*g*cos(th2)
psi121=J22*J22p*l2*l2*(-u1+q2p)-0.5*J22p*J33*l2*l3*cos(th3)*u2
psi122=J22*J33p*l2*l3*cos(th3)*(q3p-0.5*u2)+J22*l2*g*cos(th2)
psi123=J22*J33*J33*l2*l3*sin(th3)*q3p*(0.5*u2-q3p)
psi124=-0.5*J22*J22*J33*l2*l3*sin(th3)*q2p*u2
```

```
psi125=J22*J22*l2*l2*v1+J22*J33*l2*l3*cos(th3)*v2
psi12=psi121+psi122+psi123+psi124+psi125

"psi121=0
psi1221=J22p*J33*l2*l3*cos(th3)*(-0.5*u1+q2p)
psi1222=-0.5*J22*J33p*l2*l3*cos(th3)*u1
psi1223=0.5*J22*J33*J33*l2*l3*sin(th3)*q3p*u1
psi1224=J22*J22*J33*l2*l3*sin(th3)*q2p*(-0.5*u1+q2p)
psi1225=J33*J33p*l3*l3*(-u2+q3p)+J33*J33*l3*l3*v2
psi1226=J22*J33*l2*l3*cos(th3)*v1+J33*l3*g*cos(th2+th3)
psi122=psi1221+psi1222+psi1223+psi1224+psi1225+psi1226

"psi21 zero below
nmhi2=psi111*u1/Ptt11
nmh2=-h/2*(nmhi2+mhi2)+mh2
nmhi3=(psi12*u1+psi22*u2)/Ptt22
nmh3=-h/2*(nmhi3+mhi3)+mh3

tau21t=psi111*mh2+psi12*mh3
tau22t=-temp111*q2pt-temp112*q3pt-temp211*q2t-temp212*q3t
tau2=tau21t+tau22t
"tau2=min(max(tau2t,tau2min),tau2max)

"psi21 zero below
tau31t=psi22*mh3
tau32t=-temp121*q2pt-temp122*q3pt-temp221*q2t-temp222*q3t
tau3=tau31t+tau32t
"tau3=min(max(tau3t,tau3min),tau3max)

nq2old=q2
nq3old=q3

ts=t+h

pi:3.141592654
Pqq11:0.0001
Pqq12:0
Pqq21:0
Pqq22:0.0001
omega11:0.001
omega12:0
omega21:0
omega22:0.001
D11:0.002
D12:0
D21:0
D22:0.002
Ptt11:0.03
Ptt22:0.5
l2:0.45
l3:0.67
g:9.81
D:0.140
E:0.239
M:0.14
L:0.239
a:57
b:57
c:42
n2:-0.005
n3:-0.005
"tau2max:1.3
"tau2min:-1.3
"tau3max:1.3
"tau3min:-1.3
h:0.005
x02:0.222491
x03:0.178725
ak:2
per2:10
per3:10
step2:25
step3:35

"This variable is needed for SIM2VME
"reginit:1.0

end
```

```
discrete system controller_B

"SIMNON-code for controller B

"This command is needed for SIM2VME
"% modula-file regmod.t

state mh2 mh3 fvish2 fvish3
state mhi2 mhi3 fvisi2 fvisi3
state q2old q3old qr21old qr22old qr31old qr32old
state r21old r22old r31old r32old
new nmh2 nmh3 nfvish2 nfvish3
new nmhi2 nmhi3 nfvisi2 nfvisi3
new nq2old nq3old nqr21old nqr22old nqr31old nqr32old
new nr21old nr22old nr31old nr32old
input q2 q3
output tau2 tau3
time t
tsamp ts

INITIAL

detPqq=Pqq11*Pqq22-Pqq12*Pqq21
invPqq11=Pqq22/detPqq
invPqq12=-Pqq12/detPqq
invPqq21=-Pqq21/detPqq
invPqq22=Pqq11/detPqq

detomega=omega11*omega22-omega12*omega21
invom11=omega22/detomega
invom12=-omega12/detomega
invom21=-omega21/detomega
invom22=omega11/detomega

P1211=invPqq11*omega11+invPqq12*omega21
P1212=invPqq11*omega12+invPqq12*omega22
P1221=invPqq21*omega11+invPqq22*omega21
P1222=invPqq21*omega12+invPqq22*omega22

Pqqiom11=Pqq11*invom11+Pqq12*invom21
Pqqiom12=Pqq11*invom12+Pqq12*invom22
Pqqiom21=Pqq21*invom11+Pqq22*invom21
Pqqiom22=Pqq21*invom12+Pqq22*invom22

temp111=Pqqiom11*Pqq11+Pqqiom12*Pqq21+D11
temp112=Pqqiom11*Pqq12+Pqqiom12*Pqq22+D12
temp121=Pqqiom21*Pqq11+Pqqiom22*Pqq21+D21
temp122=Pqqiom21*Pqq12+Pqqiom22*Pqq22+D22

DiPqq11=D11*invPqq11+D12*invPqq21
DiPqq12=D11*invPqq12+D12*invPqq22
DiPqq21=D21*invPqq11+D22*invPqq21
DiPqq22=D21*invPqq12+D22*invPqq22

temp211=DiPqq11*omega11+DiPqq12*omega21
temp212=DiPqq11*omega12+DiPqq12*omega22
temp221=DiPqq21*omega11+DiPqq22*omega21
temp222=DiPqq21*omega12+DiPqq22*omega22

SORT

a1=-2*exp(-ak*h)
a2=exp(-2*ak*h)

b1=1-exp(-ak*h)*(1+ak*h)
b2=exp(-ak*h)*(exp(-ak*h)+ak*h-1)

r2=if mod(t,per2)<per2/2 then step2 else -step2
r3=if mod(t,per3)<per3/2 then step3 else -step3

qr2=-a1*qr21old-a2*qr22old+b1*r21old+b2*r22old
qr3=-a1*qr31old-a2*qr32old+b1*r31old+b2*r32old
nqr21old=qr2
nqr31old=qr3
nqr22old=qr21old
nqr32old=qr31old
qr2p=(qr2-qr21old)/h
qr3p=(qr3-qr31old)/h
qr2pp=(qr2-2*qr21old+qr22old)/h
qr3pp=(qr3-2*qr31old+qr32old)/h
nr21old=r2
nr31old=r3
nr22old=r21old
nr32old=r31old

q2p=(q2-q2old)/h
q3p=(q3-q3old)/h

x2=n2*q2/(2*pi)+x02
x3=n3*q3/(2*pi)+x03

th2=(270-b-a)*pi/180-arccos((D*D+E*E-x2*x2)/(2*D*E))
th3=(450-c)*pi/180-th2-arccos((M*M+L*L-x3*x3)/(2*M*L))

J2k=(D*D+E*E-x2*x2)/(2*D*E)
J3k=(M*M+L*L-x3*x3)/(2*M*L)

J22=-n2*x2/(2*pi*D*E*sqrt(1-J2k*J2k))
"J23=0
"J32=-J22
J33=-n3*x3/(2*pi*M*L*sqrt(1-J3k*J3k))

J221p=n2*n2*q2p/(4*pi*pi*D*E*sqrt(1-J2k*J2k))
J222p=J2k*x2*x2/(D*E*(1-J2k*J2k))-1
J22p=J221p*J222p
"J23p=0
"J32p=-J22p
J331p=n3*n3*q3p/(4*pi*pi*M*L*sqrt(1-J3k*J3k))
J332p=J3k*x3*x3/(M*L*(1-J3k*J3k))-1
J33p=J331p*J332p

q2t=q2-qr2
q2pt=q2p-qr2p
q3t=q3-qr3
q3pt=q3p-qr3p

u1=q2pt+P1211*q2t+P1212*q3t
u2=q3pt+P1221*q2t+P1222*q3t

v1=qr2pp-P1211*q2pt-P1212*q3pt
v2=qr3pp-P1221*q2pt-P1222*q3pt

psi11=J22*J22p*l2*l2*(-u1+q2p)+J22*J22*l2*l2*v1+J22*l2*g*cos(th2)
psi121=J22*J22p*l2*l2*(-u1+q2p)-0.5*J22p*J33*l2*l3*cos(th3)*u2
psi122=J22*J33p*l2*l3*cos(th3)*(q3p-0.5*u2)+J22*l2*g*cos(th2)
psi123=J22*J33*J33*l2*l3*sin(th3)*q3p*(0.5*u2-q3p)
psi124=-0.5*J22*J22*J33*l2*l3*sin(th3)*q2p*u2
```

```
psi125=J22*J22*l2*l2*v1+J22*J33*l2*l3*cos(th3)*v2
psi12=psi121+psi122+psi123+psi124+psi125
"psi13 and psi14 viscous friction
psi13=J22*J22*q2p
psi14=J22*J22*q2p-J22*J33*q3p

"psi21=0
psi221=J22p*J33*l2*l3*cos(th3)*(-0.5*u1+q2p)
psi222=-0.5*J22*J33p*l2*l3*cos(th3)*u1
psi223=0.5*J22*J33*J33*l2*l3*sin(th3)*q3p*u1
psi224=J22*J22*J33*l2*l3*sin(th3)*q2p*(-0.5*u1+q2p)
psi225=J33*J33p*l3*l3*(-u2+q3p)+J33*J33*l3*l3*v2
psi226=J22*J33*l2*l3*cos(th3)*v1+J33*l3*g*cos(th2+th3)
psi22=psi221+psi222+psi223+psi224+psi225+psi226
"psi23 and psi24 viscous friction
"psi23=0
psi24=-J22*J33*q2p+J33*J33*q3p

"psi21 and psi23 zero below
nmhi2=psi11*u1/Ptt11
nmh2=-h/2*(nmhi2+mhi2)+mh2
nmhi3=(psi12*u1+psi22*u2)/Ptt22
nmh3=-h/2*(nmhi3+mhi3)+mh3
"Tustin approximation for viscous friction
nfvisi2=psi13*u1/Ptt33
nfvish2=-h/2*(nfvisi2+fvisi2)+fvish2
nfvisi3=(psi14*u1+psi24*u2)/Ptt44
nfvish3=-h/2*(nfvisi3+fvisi3)+fvish3

tau21t=psi11*mh2+psi12*mh3+psi13*fvish2+psi14*fvish3
tau22t=-temp111*q2pt-temp112*q3pt-temp211*q2t-temp212*q3t
tau2=tau21t+tau22t
"tau2=min(max(tau2t,tau2min),tau2max)

"psi21 and psi23 zero below
tau31t=psi22*mh3+psi24*fvish3
tau32t=-temp121*q2pt-temp122*q3pt-temp221*q2t-temp222*q3t
tau3=tau31t+tau32t
"tau3=min(max(tau3t,tau3min),tau3max)

nq2old=q2
nq3old=q3

ts=t+h

pi:3.141592654
Pqq11:0.0001
Pqq12:0
Pqq21:0
Pqq22:0.0001
omega11:0.001
omega12:0
omega21:0
omega22:0.001
D11:0.002
D12:0
D21:0
D22:0.002
Ptt11:0.03
Ptt22:1.0
Ptt33:0.0003
Ptt44:0.0003
l2:0.45
l3:0.67
g:9.81
D:0.140
E:0.239
M:0.14
L:0.239
a:57
b:57
c:42
n2:-0.005
n3:-0.005
"tau2max:1.3
"tau2min:-1.3
"tau3max:1.3
"tau3min:-1.3
h:0.005
x02:0.222491
x03:0.178725
ak:2
per2:10
per3:10
step2:25
step3:35

"This variable is needed for SIM2VME
"reginit:1.0

end
```

```
discrete system controller_C

"SIMNON-code for controller C

"This command is needed for SIM2VME
"% modula-file regmod.t

state mh2 mh3 fvish2 fvish3
state mhi2 mhi3 fvisi2 fvisi3
state q2old q3old qr21old qr22old qr31old qr32old
state r21old r22old r31old r32old
new nmh2 nmh3 nfvish2 nfvish3
new nmhi2 nmhi3 nfvisi2 nfvisi3
new nq2old nq3old nqr21old nqr22old nqr31old nqr32old
new nr21old nr22old nr31old nr32old
input q2 q3
output tau2 tau3
time t
tsamp ts

INITIAL

detPqq=Pqq11*Pqq22-Pqq12*Pqq21
invPqq11=Pqq22/detPqq
invPqq12=-Pqq12/detPqq
invPqq21=-Pqq21/detPqq
invPqq22=Pqq11/detPqq

detomega=omega11*omega22-omega12*omega21
invom11=omega22/detomega
invom12=-omega12/detomega
invom21=-omega21/detomega
invom22=omega11/detomega

P1211=invPqq11*omega11+invPqq12*omega21
P1212=invPqq11*omega12+invPqq12*omega22
P1221=invPqq21*omega11+invPqq22*omega21
P1222=invPqq21*omega12+invPqq22*omega22

Pqqiom11=Pqq11*invom11+Pqq12*invom21
Pqqiom12=Pqq11*invom12+Pqq12*invom22
Pqqiom21=Pqq21*invom11+Pqq22*invom21
Pqqiom22=Pqq21*invom12+Pqq22*invom22

temp111=Pqqiom11*Pqq11+Pqqiom12*Pqq21+D11
temp112=Pqqiom11*Pqq12+Pqqiom12*Pqq22+D12
temp121=Pqqiom21*Pqq11+Pqqiom22*Pqq21+D21
temp122=Pqqiom21*Pqq12+Pqqiom22*Pqq22+D22

DiPqq11=D11*invPqq11+D12*invPqq21
DiPqq12=D11*invPqq12+D12*invPqq22
DiPqq21=D21*invPqq11+D22*invPqq21
DiPqq22=D21*invPqq12+D22*invPqq22

temp211=DiPqq11*omega11+DiPqq12*omega21
temp212=DiPqq11*omega12+DiPqq12*omega22
temp221=DiPqq21*omega11+DiPqq22*omega21
temp222=DiPqq21*omega12+DiPqq22*omega22

SORT

a1=-2*exp(-ak*h)
a2=exp(-2*ak*h)

b1=1-exp(-ak*h)*(1+ak*h)
b2=exp(-ak*h)*(exp(-ak*h)+ak*h-1)

r2=if mod(t,per2)<per2/2 then step2 else -step2
r3=if mod(t,per3)<per3/2 then step3 else -step3

qr2=-a1*qr21old-a2*qr22old+b1*r21old+b2*r22old
qr3=-a1*qr31old-a2*qr32old+b1*r31old+b2*r32old
nqr21old=qr2
nqr31old=qr3
nqr22old=qr21old
nqr32old=qr31old
qr2p=(qr2-qr21old)/h
qr3p=(qr3-qr31old)/h
qr2pp=(qr2-2*qr21old+qr22old)/h
qr3pp=(qr3-2*qr31old+qr32old)/h
nr21old=r2
nr31old=r3
nr22old=r21old
nr32old=r31old

q2p=(q2-q2old)/h
q3p=(q3-q3old)/h

x2=n2*q2/(2*pi)+x02
x3=n3*q3/(2*pi)+x03

th2=(270-b-a)*pi/180-arccos((D*D+E*E-x2*x2)/(2*D*E))
th3=(450-c)*pi/180-th2-arccos((M*M+L*L-x3*x3)/(2*M*L))

J2k=(D*D+E*E-x2*x2)/(2*D*E)
J3k=(M*M+L*L-x3*x3)/(2*M*L)

J22=-n2*x2/(2*pi*D*E*sqrt(1-J2k*J2k))
"J23=0
"J32=-J22
J33=-n3*x3/(2*pi*M*L*sqrt(1-J3k*J3k))

J221p=n2*n2*q2p/(4*pi*pi*D*E*sqrt(1-J2k*J2k))
J222p=J2k*x2*x2/(D*E*(1-J2k*J2k))-1
J22p=J221p*J222p
"J23p=0
"J32p=-J22p
J331p=n3*n3*q3p/(4*pi*pi*M*L*sqrt(1-J3k*J3k))
J332p=J3k*x3*x3/(M*L*(1-J3k*J3k))-1
J33p=J331p*J332p

q2t=q2-qr2
q2pt=q2p-qr2p
q3t=q3-qr3
q3pt=q3p-qr3p

u1=q2pt+P1211*q2t+P1212*q3t
u2=q3pt+P1221*q2t+P1222*q3t

v1=qr2pp-P1211*q2pt-P1212*q3pt
v2=qr3pp-P1221*q2pt-P1222*q3pt

psi11=J22*J22p*l2*l2*(-u1+q2p)+J22*J22*l2*l2*v1+J22*l2*g*cos(th2)
psi121=J22*J22p*l2*l2*(-u1+q2p)-0.5*J22p*J33*l2*l3*cos(th3)*u2
psi122=J22*J33p*l2*l3*cos(th3)*(q3p-0.5*u2)+J22*l2*g*cos(th2)
psi123=J22*J33*J33*l2*l3*sin(th3)*q3p*(0.5*u2-q3p)
psi124=-0.5*J22*J22*J33*l2*l3*sin(th3)*q2p*u2
```

```
psi125=J22*J22*l2*l2*v1+J22*J33*l2*l3*cos(th3)*v2
psi12=psi121+psi122+psi123+psi124+psi125
"psi13 and psi14 viscous friction
psi13=J22*J22*q2p
psi14=J22*J22*q2p-J22*J33*q3p

"psi21=0
psi221=J22p*J33*l2*l3*cos(th3)*(-0.5*u1+q2p)
psi222=-0.5*J22*J33p*l2*l3*cos(th3)*u1
psi223=0.5*J22*J33*J33*l2*l3*sin(th3)*q3p*u1
psi224=J22*J22*J33*l2*l3*sin(th3)*q2p*(-0.5*u1+q2p)
psi225=J33*J33p*l3*l3*(-u2+q3p)+J33*J33*l3*l3*v2
psi226=J22*J33*l2*l3*cos(th3)*v1+J33*l3*g*cos(th2+th3)
psi22=psi221+psi222+psi223+psi224+psi225+psi226
"psi23 and psi24 viscous friction
"psi23=0
psi24=-J22*J33*q2p+J33*J33*q3p

"Coulomb friction with sign-function
psi01=J22*fcol2*sign(J22*q2p)-J22*fcol3*sign(-J22*q2p+J33*q3p)
psi02=J33*fcol3*sign(-J22*q2p+J33*q3p)

"psi21 and psi23 zero below
nmhi2=psi11*u1/Ptt11
nmh2=-h/2*(nmhi2+mhi2)+mh2
nmhi3=(psi12*u1+psi22*u2)/Ptt22
nmh3=-h/2*(nmhi3+mhi3)+mh3
"Tustin approximation for viscous friction
nfvisi2=psi13*u1/Ptt33
nfvish2=-h/2*(nfvisi2+fvisi2)+fvish2
nfvisi3=(psi14*u1+psi24*u2)/Ptt44
nfvish3=-h/2*(nfvisi3+fvisi3)+fvish3

tau21t=psi11*mh2+psi12*mh3+psi13*fvish2+psi14*fvish3
tau22t=-temp111*q2pt-temp112*q3pt-temp211*q2t-temp212*q3t
tau2=tau21t+tau22t+psi01
"tau2=min(max(tau2t,tau2min),tau2max)

"psi21 and psi23 zero below
tau31t=psi22*mh3+psi24*fvish3
tau32t=-temp121*q2pt-temp122*q3pt-temp221*q2t-temp222*q3t
tau3=tau31t+tau32t+psi02
"tau3=min(max(tau3t,tau3min),tau3max)

nq2old=q2
nq3old=q3

ts=t+h

pi:3.141592654
Pqq11:0.0001
Pqq12:0
Pqq21:0
Pqq22:0.0001
omega11:0.001
omega12:0
omega21:0
omega22:0.001
D11:0.004
D12:0
D21:0
D22:0.004
Ptt11:0.03
Ptt22:1.3
Ptt33:0.0003
Ptt44:0.0003
l2:0.45
l3:0.67
g:9.81
D:0.140
E:0.239
M:0.14
L:0.239
a:57
b:57
c:42
n2:-0.005
n3:-0.005
"tau2max:1.3
"tau2min:-1.3
"tau3max:1.3
"tau3min:-1.3
h:0.005
x02:0.222491
x03:0.178725
ak:2
per2:10
per3:10
step2:25
step3:35
fcol2:14
fcol3:44

"This variable is needed for SIM2VME
"reginit:1.0

end
```

```
discrete system controller_D

"SIMNON-code for controller D

"This command is needed for SIM2VME
"% modula-file regmod.t

state mh2 mh3 fvish2 fvish3 fcolh2 fcolh3
state mhi2 mhi3 fvisi2 fvisi3 fcoli2 fcoli3
state q2old q3old qr21old qr22old qr31old qr32old
state r21old r22old r31old r32old
new nmh2 nmh3 nfvish2 nfvish3 nfcolh2 nfcolh3
new nmhi2 nmhi3 nfvisi2 nfvisi3 nfcoli2 nfcoli3
new nq2old nq3old nqr21old nqr22old nqr31old nqr32old
new nr21old nr22old nr31old nr32old
input q2 q3
output tau2 tau3
time t
tsamp ts

INITIAL

detPqq=Pqq11*Pqq22-Pqq12*Pqq21
invPqq11=Pqq22/detPqq
invPqq12=-Pqq12/detPqq
invPqq21=-Pqq21/detPqq
invPqq22=Pqq11/detPqq

detomega=omega11*omega22-omega12*omega21
invom11=omega22/detomega
invom12=-omega12/detomega
invom21=-omega21/detomega
invom22=omega11/detomega

P1211=invPqq11*omega11+invPqq12*omega21
P1212=invPqq11*omega12+invPqq12*omega22
P1221=invPqq21*omega11+invPqq22*omega21
P1222=invPqq21*omega12+invPqq22*omega22

Pqqiom11=Pqq11*invom11+Pqq12*invom21
Pqqiom12=Pqq11*invom12+Pqq12*invom22
Pqqiom21=Pqq21*invom11+Pqq22*invom21
Pqqiom22=Pqq21*invom12+Pqq22*invom22

temp111=Pqqiom11*Pqq11+Pqqiom12*Pqq21+D11
temp112=Pqqiom11*Pqq12+Pqqiom12*Pqq22+D12
temp121=Pqqiom21*Pqq11+Pqqiom22*Pqq21+D21
temp122=Pqqiom21*Pqq12+Pqqiom22*Pqq22+D22

DiPqq11=D11*invPqq11+D12*invPqq21
DiPqq12=D11*invPqq12+D12*invPqq22
DiPqq21=D21*invPqq11+D22*invPqq21
DiPqq22=D21*invPqq12+D22*invPqq22

temp211=DiPqq11*omega11+DiPqq12*omega21
temp212=DiPqq11*omega12+DiPqq12*omega22
temp221=DiPqq21*omega11+DiPqq22*omega21
temp222=DiPqq21*omega12+DiPqq22*omega22

SORT

a1=-2*exp(-ak*h)
a2=exp(-2*ak*h)

b1=1-exp(-ak*h)*(1+ak*h)
b2=exp(-ak*h)*(exp(-ak*h)+ak*h-1)

r2=if mod(t,per2)<per2/2 then step2 else -step2
r3=if mod(t,per3)<per3/2 then step3 else -step3

qr2=-a1*qr21old-a2*qr22old+b1*r21old+b2*r22old
qr3=-a1*qr31old-a2*qr32old+b1*r31old+b2*r32old
nqr21old=qr2
nqr31old=qr3
nqr22old=qr21old
nqr32old=qr31old
qr2p=(qr2-qr21old)/h
qr3p=(qr3-qr31old)/h
qr2pp=(qr2-2*qr21old+qr22old)/h
qr3pp=(qr3-2*qr31old+qr32old)/h
nr21old=r2
nr31old=r3
nr22old=r21old
nr32old=r31old

q2p=(q2-q2old)/h
q3p=(q3-q3old)/h

x2=n2*q2/(2*pi)+x02
x3=n3*q3/(2*pi)+x03

th2=(270-b-a)*pi/180-arccos((D*D+E*E-x2*x2)/(2*D*E))
th3=(450-c)*pi/180-th2-arccos((M*M+L*L-x3*x3)/(2*M*L))

J2k=(D*D+E*E-x2*x2)/(2*D*E)
J3k=(M*M+L*L-x3*x3)/(2*M*L)

J22=-n2*x2/(2*pi*D*E*sqrt(1-J2k*J2k))
"J23=0
"J32=-J22
J33=-n3*x3/(2*pi*M*L*sqrt(1-J3k*J3k))

J221p=n2*n2*q2p/(4*pi*pi*D*E*sqrt(1-J2k*J2k))
J222p=J2k*x2*x2/(D*E*(1-J2k*J2k))-1
J22p=J221p*J222p
"J23p=0
"J32p=-J22p
J331p=n3*n3*q3p/(4*pi*pi*M*L*sqrt(1-J3k*J3k))
J332p=J3k*x3*x3/(M*L*(1-J3k*J3k))-1
J33p=J331p*J332p

q2t=q2-qr2
q2pt=q2p-qr2p
q3t=q3-qr3
q3pt=q3p-qr3p

u1=q2pt+P1211*q2t+P1212*q3t
u2=q3pt+P1221*q2t+P1222*q3t

v1=qr2pp-P1211*q2pt-P1212*q3pt
v2=qr3pp-P1221*q2pt-P1222*q3pt

psi11=J22*J22p*12*12*(-u1+q2p)+J22*J22*12*12*v1+J22*12*g*cos(th2)
psi121=J22*J22p*12*12*(-u1+q2p)-0.5*J22p*J33*12*13*cos(th3)*u2
psi122=J22*J33p*12*13*cos(th3)*(q3p-0.5*u2)+J22*12*g*cos(th2)
psi123=J22*J33*J33*12*13*sin(th3)*q3p*(0.5*u2-q3p)
psi124=-0.5*J22*J22*J33*12*13*sin(th3)*q2p*u2
```

```
psi125=J22*J22*l2*l2*v1+J22*J33*l2*l3*cos(th3)*v2
psi12=psi121+psi122+psi123+psi124+psi125
"psi13 and psi14 viscous friction
psi13=J22*J22*q2p
psi14=J22*J22*q2p-J22*J33*q3p
"psi15 and psi16 Coulomb friction
psi15=J22*sign(J22*q2p)
psi16=-J22*sign(-J22*q2p+J33*q3p)

"psi21=0
psi221=J22p*J33*l2*l3*cos(th3)*(-0.5*u1+q2p)
psi222=-0.5*J22*J33p*l2*l3*cos(th3)*u1
psi223=0.5*J22*J33*J33*l2*l3*sin(th3)*q3p*u1
psi224=J22*J22*J33*l2*l3*sin(th3)*q2p*(-0.5*u1+q2p)
psi225=J33*J33p*l3*l3*(-u2+q3p)+J33*J33*l3*l3*v2
psi226=J22*J33*l2*l3*cos(th3)*v1+J33*l3*g*cos(th2+th3)
psi22=psi221+psi222+psi223+psi224+psi225+psi226
"psi23 and psi24 viscous friction
"psi23=0
psi24=-J22*J33*q2p+J33*J33*q3p
"psi25 and psi26 Coulomb friction
"psi25=0
psi26=J33*sign(-J22*q2p+J33*q3p)

"psi21, psi23 and psi25 zero below
nmhi2=psi11*u1/Ptt11
nmh2=-h/2*(nmhi2+mhi2)+mh2
nmhi3=(psi12*u1+psi22*u2)/Ptt22
nmh3=-h/2*(nmhi3+mhi3)+mh3
"Tustin approximation for viscous friction
nfvisi2=psi13*u1/Ptt33
nfvish2=-h/2*(nfvisi2+fvisi2)+fvish2
nfvisi3=(psi14*u1+psi24*u2)/Ptt44
nfvish3=-h/2*(nfvisi3+fvisi3)+fvish3
"Tustin approximation for Coulomb friction
nfcoli2=psi15*u1/Ptt55
nfcolh2=-h/2*(nfcoli2+fcoli2)+fcolh2
nfcoli3=(psi16*u1+psi26*u2)/Ptt66
nfcolh3=-h/2*(nfcoli3+fcoli3)+fcolh3

tau211t=psi11*mh2+psi12*mh3+psi13*fvish2+psi14*fvish3
tau212t=psi15*fcolh2+psi16*fcolh3
tau22t=-temp111*q2pt-temp112*q3pt-temp211*q2t-temp212*q3t
tau2=tau211t+tau212t+tau22t
"tau2=min(max(tau2t,tau2min),tau2max)

"psi21, psi23 and psi25 zero below
tau31t=psi22*mh3+psi24*fvish3+psi26*fcolh3
tau32t=-temp121*q2pt-temp122*q3pt-temp221*q2t-temp222*q3t
tau3=tau31t+tau32t
"tau3=min(max(tau3t,tau3min),tau3max)

nq2old=q2
nq3old=q3

ts=t+h

pi:3.141592654
Pqq11:0.0001
Pqq12:0
Pqq21:0
Pqq22:0.0001
omega11:0.01
omega12:0
omega21:0
omega22:0.01
D11:0.004
D12:0
D21:0
D22:0.004
Ptt11:0.02
Ptt22:1.5
Ptt33:0.0005
Ptt44:0.001
Ptt55:0.02
Ptt66:0.05
l2:0.45
l3:0.67
g:9.81
D:0.140
E:0.239
M:0.14
L:0.239
a:57
b:57
c:42
n2:-0.005
n3:-0.005
"tau2max:1.3
"tau2min:-1.3
"tau3max:1.3
"tau3min:-1.3
h:0.005
x02:0.222491
x03:0.178725
ak:2
per2:10
per3:10
step2:25
step3:35

"This variable is needed for SIM2VME
"reginit:1.0

end
```

```
discrete system controller_E

"SIMNON-code for controller E

"This command is needed for SIM2VME
"% modula-file regmod.t

state d2old d3old q2old q3old i2 i3
new d2 d3 nq2old nq3old ni2 ni3
input q2 q3
output tau2 tau3
time t
tsamp ts

SORT

r2=if mod(t,per2)<per2/2 then step2 else -step2
r3=if mod(t,per3)<per3/2 then step3 else -step3

p2=k2*(r2-q2)
p3=k3*(r3-q3)
ni2=i2+k2*h*(r2-q2)/ti2+(tau2t-tau2tt)/Tt2
ni3=i3+k3*h*(r3-q3)/ti3+(tau3t-tau3tt)/Tt3
d2=td2/(td2+N2*h)*d2old-k2*td2*N2/(td2+N2*h)*(q2-q2old)
d3=td3/(td3+N3*h)*d3old-k3*td3*N3/(td3+N3*h)*(q3-q3old)
tau2tt=p2+i2+d2
tau3tt=p3+i3+d3
tau2t=min(max(tau2tt,tau2min),tau2max)
tau3t=min(max(tau3tt,tau3min),tau3max)
tau2=tau2t
tau3=tau3t
nq2old=q2
nq3old=q3

ts=t+h

k2:0.08
k3:0.08
ti2:0.8
ti3:0.8
Tt2:0.8
Tt3:0.8
td2:0.15
td3:0.15
N2:10
N3:10
h:0.005
step2:25
step3:35
per2:10
per3:10
tau2min:-1.3
tau2max:1.3
tau3min:-1.3
tau3max:1.3

"This variable is needed for SIM2VME
"reginit:1.0

end
```

```
continuous system process

"SIMNON-code for process-modell of robot

state q21 q22 q31 q32
der dq21 dq22 dq31 dq32
input tau2 tau3
output q2 q2p q2pp q3 q3p q3pp
time t

SORT

x2=n2*q21/(2*pi)+x02
x3=n3*q31/(2*pi)+x03

th2=(270-b-a)*pi/180-arccos((D*D+E*E-x2*x2)/(2*D*E))
th3=(450-c)*pi/180-th2-arccos((M*M+L*L-x3*x3)/(2*M*L))

J2k=(D*D+E*E-x2*x2)/(2*D*E)
J3k=(M*M+L*L-x3*x3)/(2*M*L)
J22=-n2*x2/(2*pi*D*E*sqrt(1-J2k*J2k))
"J23=0
"J32=-J22
J33=-n3*x3/(2*pi*M*L*sqrt(1-J3k*J3k))

J221p=n2*n2*q22/(4*pi*pi*D*E*sqrt(1-J2k*J2k))
J222p=J2k*x2*x2/(D*E*(1-J2k*J2k))-1
J22p=J221p*J222p
"J23p=0
"J32p=-J22p
J331p=n3*n3*q32/(4*pi*pi*M*L*sqrt(1-J3k*J3k))
J332p=J3k*x3*x3/(M*L*(1-J3k*J3k))-1
J33p=J331p*J332p

M22=J22*J22*(m2+m3)*l2*l2
M23=J22*J33*m3*l2*l3*cos(th3)
M32=M23
M33=J33*J33*m3*l3*l3

detM=M22*M33-M23*M32
invM22=M33/detM
invM23=-M23/detM
invM32=-M32/detM
invM33=M22/detM

"------------------------------

"C-matrix without viscous friction
"C22=J22*J22p*(m2+m3)*l2*l2-J22*J22*J33*m3*l2*l3*sin(th3)*q32
"C23=J22*m3*l2*l3*(J33p*cos(th3)+J33*sin(th3)*(J22*q22-J33*q32))
"C32=J33*m3*l2*l3*(J22p*cos(th3)+J22*J22*sin(th3)*q22)
"C33=J33*J33p*m3*l3*l3

"C-matrix with viscous friction
C22v=J22*J22*fvis2+J22*J22*fvis3
C23v=-J22*J33*fvis3
C32v=-J22*J33*fvis3
C33v=J33*J33*fvis3
C22=J22*J22p*(m2+m3)*l2*l2-J22*J22*J33*m3*l2*l3*sin(th3)*q32+C22v
C23=J22*m3*l2*l3*(J33p*cos(th3)+J33*sin(th3)*(J22*q22-J33*q32))+C23v
C32=J33*m3*l2*l3*(J22p*cos(th3)+J22*J22*sin(th3)*q22)+C32v
C33=J33*J33p*m3*l3*l3+C33v

"------------------------------

"G-matrix in actuator space
G2=J22*(m2+m3)*l2*g*cos(th2)
G3=J33*m3*l3*g*cos(th2+th3)

"G-matrix in joint space
G2j=m3*l3*g*cos(th2+th3)+(m2+m3)*l2*g*cos(th2)
G3j=m3*l3*g*cos(th2+th3)

"------------------------------

"Coulomb friction with sign-function
"F2=J22*fcol2*sign(J22*q22)-J22*fcol3*sign(-J22*q22+J33*q32)
"F3=J33*fcol3*sign(-J22*q22+J33*q32)

"Coulomb friction with ramp and gravitation
"Index j means joint space
tau2j=tau2/J22+tau3/J33
tau3j=tau3/J33
f2abs=min(abs(tau2j-G2j),fcol2)
F2j=if (tau2j-G2j)<0 then -f2abs else f2abs
f3abs=min(abs(tau3j-G3j),fcol3)
F3j=if (tau3j-G3j)<0 then -f3abs else f3abs
F2=J22*F2j-J22*F3j
F3=J33*F3j

"------------------------------

"State update without Coulomb friction
"dq21=q22
"dq31=q32
"dq221=-(invM22*C22+invM23*C32)*q22-(invM22*C23+invM23*C33)*q32
"dq222=invM22*(tau2-G2)+invM23*(tau3-G3)
"dq22=dq221+dq222
"dq321=-(invM32*C22+invM33*C32)*q22-(invM32*C23+invM33*C33)*q32
"dq322=invM32*(tau2-G2)+invM33*(tau3-G3)
"dq32=dq321+dq322

"State update with Coulomb friction
dq21=q22
dq31=q32
dq221=-(invM22*C22+invM23*C32)*q22-(invM22*C23+invM23*C33)*q32
dq222=invM22*(tau2-G2-F2)+invM23*(tau3-G3-F3)
dq22=dq221+dq222
dq321=-(invM32*C22+invM33*C32)*q22-(invM32*C23+invM33*C33)*q32
dq322=invM32*(tau2-G2-F2)+invM33*(tau3-G3-F3)
dq32=dq321+dq322

"------------------------------

q2=q21
q2p=q22
q2pp=dq22
q3=q31
q3p=q32
q3pp=dq32

m3=if t>20 then 6 else 2

pi:3.141592654
m2:3
"m3:2
```

```
l2:0.45
l3:0.67
g:9.81
D:0.140
E:0.239
M:0.140
L:0.239
a:57
b:57
c:42
n2:-0.005
n3:-0.005
x02:0.222491
x03:0.178725
fvis2:100
fvis3:100
fcol2:14
fcol3:44

end
```

```
% import-statement

FROM ServoIO IMPORT absPos, RefOut, RefScale, MaxTorque;
IMPORT MathLib0;

% end

% const-declaration

CONST Pi = 3.14159265359;
      halfPi = 1.5707963268;

% end

% procedure-declaration

PROCEDURE sign(x : LONGREAL) : LONGREAL;
BEGIN
  IF x > 0.0 THEN
    RETURN 1.0
  ELSIF x < 0.0 THEN
    RETURN -1.0
  ELSE
    RETURN 0.0
  END; (* if *)
END sign;

PROCEDURE arctan2(y, x : LONGREAL) : LONGREAL;
(* arctan(y/x) *)
BEGIN
   IF (x > 0.0) AND (y >= 0.0) THEN
      RETURN MathLib0.arctan(y / x)
   ELSIF (x > 0.0) AND (y <= 0.0) THEN
      RETURN MathLib0.arctan(y / x)
   ELSIF (x < 0.0) AND (y <= 0.0) THEN
      RETURN MathLib0.arctan(y / x) -  Pi
   ELSIF (x < 0.0) AND (y >= 0.0) THEN
      RETURN MathLib0.arctan(y / x) +  Pi
   ELSIF (x = 0.0) AND (y >= 0.0) THEN
      RETURN halfPi
   ELSIF (x = 0.0) AND (y <= 0.0) THEN
      RETURN -halfPi
   END;  (* if *)
END arctan2;

PROCEDURE arccos(x : LONGREAL) : LONGREAL;
BEGIN
  IF ABS(x) < 1.0 THEN
    RETURN arctan2(MathLib0.sqrt(1.0 - x*x),x)
  ELSIF x >= 1.0 THEN
    RETURN 0.0
  ELSE
    RETURN Pi
  END;
END arccos;

%end

% update-code

  IF reginit > 0.5 THEN
    RefScale(1,MaxTorque);
    RefScale(2,MaxTorque);
    RefScale(3,MaxTorque);
    RefScale(4,MaxTorque);
    RefScale(5,MaxTorque);
    reginit := 0.0;
  END;

(*  q1 := absPos(1); *)
  q2 := absPos(2);
  q3 := absPos(3);
(*  q4 := absPos(4); *)
(*  q5 := absPos(5); *)

(*  RefOut(1,tau1); *)
  RefOut(2,tau2);
  RefOut(3,tau3);
(*  RefOut(4,tau4); *)
(*  RefOut(5,tau5); *)

% end
```