Lappeenrannan teknillinen korkeakoulu
Konetekniikan osasto
Konstruktiotekniikan laitos

Mekatronisten koneiden reaaliaikainen simulointi Linux-ympäristössä

Diplomityön aihe on hyväksytty konetekniikan osaston osastoneuvostossa 2.10.2002

Työn tarkastajina ovat toimineet professori Aki Mikkola ja tekniikan tohtori Asko Rouvinen

Lappeenrannassa 24.1.2003

Mikko Asikainen
Teknologiapuistonkatu 2 A 2
53850 Lappeenranta
+358 5 6212465

TIIVISTELMÄ

Tekijä: Mikko Asikainen

Nimi: **Mekatronisten koneiden reaaliaikainen simulointi Linux-ympäristössä**

Osasto: Konetekniikan osasto

Paikka: Lappeenranta

Vuosi: 2002

Diplomityö. Lappeenrannan teknillinen korkeakoulu

50 sivua, 17 kuvaa, 2 taulukkoa, 3 liitettä

Tarkastajat: Professori Aki Mikkola

TkT Asko Rouvinen

Hakusanat: reaaliaika-Linux, monikappaledynamiikka, mekatroniikka, reaaliaikainen simulointi

Työn tavoitteena oli selvittää mahdollisuuksia käyttää Linux-ympäristöä mekatronisten koneiden reaaliaikaisessa simuloinnissa. Työssä tutkittiin C-kielellä tehdyn reaaliaikaisen simulointimallin ratkaisua Linux-käyttöjärjestelmässä RTLinux-reaaliaikalaajennuksen avulla.

Reaaliaikainen simulointi onnistui RTLinuxin avulla tehokkaasti ja mallinnusmenetelmien rajoissa tarkasti. I/O-toimintojen lisäämistä erillisten I/O-korttien avulla ei tarkasteltu tässä työssä.

Reaaliaikaista Linuxia ei ole aikaisemmin käytetty mekatronisten koneiden simulointiin. Tämän vuoksi valmiita työkaluja ei ole olemassa. Linux-ympäristö ei näin ollen sovellu kovin hyvin yleiseen koneensuunnitteluun mallintamisen työläyden vuoksi.

ABSTRACT

Author: Mikko Asikainen

Title: **Real time simulation of mechatronic machines in Linux-environment**

Department: Mechanical Engineering

Place: Lappeenranta

Year: 2002

Master's thesis. Lappeenranta University of Technology

50 pages, 17 figures, 2 tables, 3 appendices

Supervisors: Professor Aki Mikkola

Dr. tech Asko Rouvinen

Keywords: realtime-Linux, multi-body dynamics, mechatronics, realtime simulation

The goal of this study was to explore possibilities to use Linux-environment in the realtime simulation of mechatronic machines. A realtime simulation model, written in C-language, was run in Linux- operating system using RTLinux- real time environment.

The realtime simulation using RTLinux was found to be efficient and accurate. Adding I/O-functionality using additional I/O-boards was not considered in this study.

The real time Linux is not widely used in simulation of mechatronic machines. Therefore, there are no applications available. As a result, Linux-environment is not suitable for general machine design process.

ALKUSANAT

Diplomityö tehtiin Lappeenrannan teknillisen korkeakoulun Mekatroniikan ja virtuaalisuunnittelun laboratoriossa. Työn tarkastajina ovat toimineet professori Aki Mikkola ja tekniikan tohtori Asko Rouvinen, joita haluan kiittää mahdollisuudesta tehdä työ mielenkiintoisesta aiheesta sekä neuvoista ja ohjeista. Pasi Korkealaakson tutkimustyö on ollut myös suureksi avuksi.

Itseäni haluan kiittää ajoittaisesta ahkeruudesta työn parissa. Kiitoksia myös kaikille muille työn edistymistä auttaneille ystäville ja terveisiä lopuille.

Lappeenrannassa 15.11.2002

Mikko Asikainen

KÄYTETYT MERKINNÄT

a_{ij}	Kerroinmatriisin A alkio
$\mathbf{A}$	Kerroinmatriisi
A_i	Pinta-ala i
$\mathbf{A}^i$	Kappaleen i kiertomatriisi
$\mathbf{b}$	Vektori
B_e	Tehollinen puristuskerroin
b_i	Vektorin $\mathbf{b}$ komponentti
C_n	Rajoiteyhtälö
$\mathbf{C}_{\boldsymbol{q}}$	Jacobin matriisi
C_v	Puoliempiirinen tilavuusvirtavakio
f	Funktio
F_s	Sylinterivoima
F_μ	Kitkavoima
k_i	Runge-Kutta –menetelmän kerroin
$\mathbf{m}^i_{RR}$	Kappaleen i massamatriisin komponentti
$\mathbf{m}^i_{R\theta}$	Kappaleen i massamatriisin komponentti
$\mathbf{m}^i_{\theta R}$	Kappaleen i massamatriisin komponentti
$m^i_{\theta\theta}$	Kappaleen i massamatriisin komponentti
$\mathbf{M}$	Järjestelmän massamatriisi
$\mathbf{M}^i$	Kappaleen i massamatriisi
n	Yhtälöiden lukumäärä, kappaleiden lukumäärä
p_i	Paine i
O	Virhetermi
$\mathbf{q}$	Yleistettyjen koordinaattien vektori
$\dot{\mathbf{q}}$	Yleistettyjen koordinaattien vektorin aikaderivaatta
$\ddot{\mathbf{q}}$	Yleistettyjen koordinaattien vektorin toinen aikaderivaatta
$\mathbf{Q}_c$	Rajoitteet huomioiva vektori
$\mathbf{Q}_e$	Yleistetty voimavektori

Q	Tilavuusvirta
Q_i	Tilavuusvirta sisään
Q_o	Tilavuusvirta ulos
$\mathbf{Q}_V^i$	Kappaleen *i* neliöllinen nopeusvektori
$\mathbf{r}_P^i$	Kappaleen *i* pisteen *P* paikkavektori globaalissa koordinaatistossa
r_{1P}^i	Vektorin $\mathbf{r}_P^i$ *X*-komponentti
r_{2P}^i	Vektorin $\mathbf{r}_P^i$ *Y*-komponentti
$\mathbf{R}^i$	Kappaleen *i* lokaalin koordinaatiston paikkavektori
R_1^i	Vektorin $\mathbf{R}^i$ *X*-komponentti
R_2^i	Vektorin $\mathbf{R}^i$ *Y*-komponentti
t	Aika
U	Jännite
$\overline{\mathbf{u}}_P^i$	Kappaleen *i* pisteen *P* paikkavektori lokaalissa koordinaatistossa
$\overline{u}_{1P}^i$	Vektorin $\overline{\mathbf{u}}_P^i$ $\overline{X}^i$-komponentti
$\overline{u}_{2P}^i$	Vektorin $\overline{\mathbf{u}}_P^i$ $\overline{Y}^i$-komponentti
V	Tilavuus
$\mathbf{x}$	Vektori
x_i	Yleinen muuttuja, vektorin $\mathbf{x}$ komponentti
x_0	Askelen alkupiste
x_1	Askelen loppupiste
X	Globaalin koordinaatiston koordinaattiakseli
$\overline{X}^i$	Kappaleen *i* lokaalin koordinaatiston koordinaattiakseli
y	Funktio
y'	*y*:n derivaatta
y_0	Alkuarvo
y_1	Loppuarvo
y^0	Alkuarvotehtävän alkuehto
Y	Globaalin koordinaatiston koordinaattiakseli
$\overline{Y}^i$	Kappaleen *i* lokaalin koordinaatiston koordinaattiakseli

Kreikkalaiset aakkoset:

θ^i	Kappaleen i lokaalin koordinaatiston kiertymä globaalissa koordinaatistossa
$\boldsymbol{\lambda}$	Lagrangen kertoimien vektori
η	Kitkakerroin
ξ	Sylinterikohtainen funktio

SISÄLLYSLUETTELO

1 JOHDANTO

1.1 Yleistä

Reaaliaikasimulointia voidaan hyödyntää monin tavoin mekatronisten koneiden suunnitteluprosessissa. Käyttäjä saadaan reaaliaikaisen simulointimallin avulla mukaan suunnittelutyöhön jo ennen prototyypin rakentamista. Käyttäjän antaman vasteen avulla työkierrosta saadaan todenmukainen, joten myös koneeseen kohdistuvia rasituksia voidaan tarkastella todellisissa käyttöolosuhteissa. On myös mahdollista liittää simulointimalli osaksi todellista konejärjestelmää. Tällöin riittää, että mallinetaan se osa konejärjestelmää, jonka muutosten vaikutusta koneen toimintaan halutaan tarkastella sekä rajapinta simulointimallin ja todellisen konejärjestelmän välille.

Nykyisin mekatronisten järjestelmien simuloinnissa käytettävät reaaliaikajärjestelmät perustuvat erillisiin prosessorikortteihin tai erikoisohjelmistoihin. Tällaisten järjestelmien hinnat ovat moninkertaiset tavanomaisiin työasemiin verrattuna. Lisäksi laitealustojen sekä käytettävien ohjelmistojen suhteen ei ole paljoa valinnanvaraa. On olemassa myös reaaliaikaisuuteen pystyviä laajennuksia CAD-ohjelmiin, mutta näiden avulla saavutettava reaaliaikaisuus ja mallinnustarkkuus ei täytä monien simulointisovellusten tarkkuusvaatimuksia.

PC-tietokoneiden suorituskyvyn kehityksen myötä tavallinen pöytätietokonekin riittää prosessoriteholtaan monissa sovelluksissa reaaliaikasimulointiin. Itseasiassa erilliset reaaliaikaiset prosessorikortit ovat laskentateholtaan usein jopa huonompia kuin nykyaikaiset PC-tietokoneet. Reaaliaikainen simulointi vaatii kuitenkin laskentatehon lisäksi käyttöjärjestelmältä tiettyjä ominaisuuksia kuten prosessien tarkkaa ajoitusta, priorisointia ja I/O-toimintoja. Nämä puuttuvat yleisimmistä PC-tietokoneiden käyttöjärjestelmistä.

1.2 Reaaliaikaisuus

Simulointi on reaaliaikainen, kun simulointimalli reagoi ulkoisiin satunnaisiin herätteisiin määrätyssä ajassa ennakoitavalla tavalla. Simuloinnin tulokset riippuvat annettujen syötteiden lisäksi ajasta. Tämän vuoksi reaaliaikajärjestelmän on pystyttävä toimimaan asetetun aikataulun mukaisesti. /1, s. 5/.

Reaaliaikaisuus voidaan jakaa kahteen päätyyppiin, pehmeään (engl. soft) ja kovaan (engl. hard). Pehmeässä reaaliaikaisuudessa simuloinnin aikarajat saavutetaan useimmiten, mutta aikarajan ylittäminen ei johda katastrofiin. Esimerkiksi tietokoneen musiikinsoitto-ohjelma suoritetaan pehmeässä reaaliajassa. Tässä tapauksessa ei tapahdu suurta vahinkoa, vaikka musiikki pätkisikin prosessorin kuormituksen ollessa suuri. Ohjelman ei myöskään tarvitse reagoida aina tietyssä lyhyessä ajassa käyttäjän käskyihin eli latenssiajalle ei ole ehdottomia vaatimuksia. Latenssiaika tarkoittaa aikaa, joka kuluu signaalin vastaanottamisesta siihen reagoimiseen.

Kovassa reaaliajassa reaaliaikaiset prosessit suoritetaan varmuudella tietyssä ajassa ja kaikki muut vähemmän kriittiset prosessit saavat odottaa. Esimerkiksi koneiden ohjausjärjestelmien on toimittava varmasti kaikissa tilanteissa. Myös simulointimallin suoritus voi epäonnistua tai tarkkuus kärsiä, mikäli laskentaa ei voida suorittaa jokaisella aika-askeleella. Prosessien priorisoinnin lisäksi välttämätön edellytys kovalle reaaliaikaisuudelle on riittävän pieni ja ennustettavissa oleva latenssiaika. Pieni latenssiaika on tärkeä, koska reaaliaikaohjelman tulee pystyä vastaamaan satunnaisiin herätteisiin tietyssä ajassa.

On myös olemassa erilaisia välimuotoja kovan ja pehmeän reaaliaikaisuuden välillä. Nämä sisältävät muutamia varsinaisten reaaliaikakäyttöjärjestelmien ominaisuuksia kuten tarkka prosessien ajastus ja/tai pieni latenssiaika. Tällaisia ratkaisuja sanotaan lujaksi (engl. firm) reaaliaikaisuudeksi.

Mekatronisen koneen simulointimallin vaatimukset reaaliaikaisuuden toteutuksesta riippuvat itse mallista, varsinkin käytetyistä numeerisista menetelmistä, ja

sovelluksesta. Joissakin sovelluksissa aika-askelten väliin jääminen ei aiheuta ongelmia kun taas toisissa sovelluksissa latenssiajalle ei ole tiukkoja rajoituksia. Reaaliaikaisuuden toteutustapa on siis valittava käyttökohteen mukaan.

Tässä työssä keskityttiin tarkastelemaan sovellusta, jossa vaaditaan korkeatasoista reaaliaikaisuutta. Tämä yhdistettynä simulointimalliin, joka kuvaa tarkasti simuloitavaa kohdetta mahdollistaa todenmukaisen simuloinnin.

1.3 Muut menetelmät

1.3.1 Yleistä

Markkinoilla on useita kaupallisia ratkaisuja reaaliaikaisen simuloinnin toteuttamiseksi. Vaihtoehtoina on sekä erilliseen prosessorikorttiin että tietokoneen omaan prosessoriin perustuvia ratkaisuja. Tässä on esitelty joitakin PC-tietokoneille soveltuvia vaihtoehtoja, joita voidaan soveltaa mekatronisten koneiden reaaliaikaiseen simulointiin.

1.3.2 dSPACE

dSPACE tarjoaa erillisen prosessorikortin avulla toteutetun reaaliaikaympäristön. Prosessorikortin lisäksi tarjolla on erilaisia I/O-liityntäpintoja ja tapoja yhdistää prosessorikortteja. dSPACE:n laitteet ovat modulaarisia ja niitä voidaan liittää toisiinsa monin tavoin. Simulointimallit voidaan tehdä Matlab/Simulink-ympäristössä ja kääntää prosessorikortin ymmärtämälle C-kielelle. Muita tapoja mallin luomiseen ei tueta /2/.

1.3.3 Opal-RT RT-Lab

RT-Lab suorittaa ohjelmia reaaliaikaisena tietokoneen omalla prosessorilla. Simulointia hallitaan yhdellä Windows-käyttöjärjestelmällä varustetulla tietokoneella johon on liitetty yksi tai useampi PC-tietokone, joissa on QNX-reaaliaikakäyttöjärjestelmä. Malli luodaan käyttäen Matlab/Simulink- tai SystemBuild-ohjelmistoa ja ladataan sieltä laskentaan käytettäville tietokoneille /3/.

1.3.4 Venturcom Real Time Extension RTX

Venturcom RTX on reaaliaikalaajennus Microsoft Windows NT, Windows NT Embedded, Windows 2000, Windows XP ja Windows XP Embedded –käyttöjärjestelmille. Sen avulla on mahdollista suorittaa reaaliaikaisia prosesseja parhaimmillaan kovassa reaaliajassa prosessien kehittyneen priorisoinnin ja ajoituksen ansiosta. Reaaliaikaohjelmat luodaan käyttämällä Windows-ympäristön ohjelmointityökaluja. Mekatronisten koneiden mallintamiseen suunnattuja työkaluja ei ole /4/.

1.3.5 Lumeo Motion

Lumeo Motion on rivisuunnittelijoille suunnattu Pro/ENGINEER-CAD-ohjelmiston pehmeään reaaliaikaiseen simulointiin kykenevä laajennus. Simulointimallin suoritus tapahtuu normaalin käyttöjärjestelmän alaisuudessa. Simulointimallin luonti tapahtuu CAD-ohjelman avulla. Sekä reaaliaikaisuuden luonteen että fysikaalisten yksinkertaistusten vuoksi Lumeo Motion ei sovellu vaativien mekatronisten koneiden simulointiongelmien ratkaisuun /5/.

1.4 Työn tavoitteet ja rajaus

Reaaliaikaista Linuxia käytetään pääosin erilaisten koneiden ohjausjärjestelmissä ja sulautetuissa järjestelmissä. Mekatronisten koneiden reaaliaikaiseen simulointiin ei Linuxia ole juuri käytetty. Työn tavoitteena oli selvittää Linux-käyttöjärjestelmän mahdollisuuksia tällaisissa sovelluksissa muodostamalla reaaliaikainen simulointimalli, jonka toimintaa testattiin ja tuloksia verrattiin kaupallisella monikappaledynamiikkaohjelmistolla saatuihin tuloksiin.

Työssä oletettiin kappaleiden olevan jäykkiä ja liikkeiden tapahtuvan tasossa. Hydrauliikka idealisoitiin keskittyneiden paineiden menetelmällä. I/0-toimintoja ja simuloinnin visualisointia ei tarkasteltu.

2 KÄYTETYT MENETELMÄT

2.1 Reaaliaikaisuus Linux-ympäristössä

2.1.1 Linux-ympäristö

Linux on GNU-lisenssin alla kehitetty UNIX-tyyppinen käyttöjärjestelmä. GNU-lisenssin mukaisesti sen lähdekoodi on vapaasti saatavilla. Monipuolisuutensa, luotettavuutensa ja suorituskykynsä ansiosta se on suosittu monissa vaativissa sovelluksissa, kuten erilaisissa palvelimissa. Käyttö työasemissa ja kotitietokoneissa on myös yleistynyt viime vuosina laajentuneen sovellusvalikoiman ansiosta.

2.1.2 Erilaisia vaihtoehtoja reaaliaikasimulointiin

Perus-Linux Ilman erillisiä laajennuksia Linux ei sovellu hyvin reaaliaikaohjelmille koska se on moniajoon perustuva käyttöjärjestelmä, eli prosessoriaika jaetaan kaikkien prosessien kesken. Näinollen raskaasti kuormitetulla koneella ei voida ratkaista monimutkaista simulointimallia reaaliajassa, koska reaaliaikaohjelma ei saa tarvittavaa prosessoriaikaa käyttöönsä. Suurin este Linuxin käytölle reaaliaikasimuloinnissa onkin juuri puutteellinen prosessien priorisointi.

Myös prosessien ajoituksen tulee olla riittävän tarkka reaaliaikaisuuden toteutumiseksi. Linuxissa ajoitusresoluutio on useimmilla laitearkkitehtuureilla 10 ms, mikä ei ole riittävän lyhyt reaaliaikaohjelmille. Lisäksi saavutettavat latenssiajat ovat liian pitkiä useimmille reaaliaikasovelluksille.

Kuitenkin joissakin erikoistapauksissa perus-Linuxin mahdollisuudet ovat riittävät myös reaaliaikasimulointiin. Tällöin voidaan käyttää tavallista käyttöjärjestelmää ja ohjelmoida simulointimalli kuten mikä tahansa ohjelma.

RTLinux RTLinux perustuu erilliseen reaaliaikaytimeen. Linuxin normaali ydin toimii reaaliaikaytimen yhtenä prosessina siten, että kun reaaliaikaprosessit eivät tarvitse prosessoriaikaa, annetaan sitä Linuxille. Linuxilla voidaan suorittaa tavallisia ohjelmia, joten esimerkiksi visualisointi voidaan toteuttaa niin, ettei itse simulointimallin reaaliaikainen suoritus kärsi. Tarkempi kuvaus toiminnasta on kappaleessa 2.1.3.

RTLinuxin avulla saavutetaan erittäin hyvä reaaliaikaisuus. Normaalilla PC:llä keskeytyskutsujen latenssiajat ovat monissa sovelluksissa suurimmillaan 15 μs ja usein pienempiä, ei juuri laitteiston asettamaa rajoitusta huonompia. RTLinuxia on jo vuosia käytetty erilaisten koneiden ohjausjärjestelmissä. Nopeisiin ohjauskiertoihin (<100 ms) RTLinux, kuten muutkaan Linux-pohjaiset järjestelmät, ei kuitenkaan sovellu /6, s. 3/.

RTAI RTAI (Realtime Application Interface) perustuu RTLinuxiin. Suurelta osin nämä kaksi ovat samantyyppisiä, suurin ero on reaaliaikaosan ominaisuuksissa. RTLinuxissa reaaliaikaosa on mahdollisimman yksinkertainen ja ominaisuuksiltaan riisuttu kun taas RTAI tarjoaa enemmän ominaisuuksia.

Ylimääräiset ominaisuudet huonontavat suorituskykyä ja osittain kokeellisina saattavat aiheuttaa epävakautta. Tietyissä sovelluksissa lisäominaisuudet saattavat kuitenkin olla tarpeen. RTAI ei ole aivan niin laajasti käytössä kuin RTLinux /7, 8/.

KURT KURT (KU Real-Time Linux) pystyy tarjoamaan lujan reaaliaikaisuuden. Prosessit jaetaan normaaleihin ja reaaliaikaisiin vaiheisiin. Prosessien ajastusta on tarkennettu perus-Linuxiin verrattuna ja ajoitus voidaan suorittaa kymmenien mikrosekuntien tarkkuudella. Ajoitusta varten KURTissa on oma algoritmi. KURT tarjoaa riittävät ominaisuudet esimerkiksi useimpiin mekatronisten koneiden reaaliaikaisiin simulointeihin, mutta vaativissa ohjausjärjestelmissä ominaisuudet eivät välttämättä ole riittävät /9, s. 8-15/.

RED-Linux RED-Linux (Real-time and Embedded Linux) perustuu Linuxin omaan ytimeen tarjoten pehmeän reaaliaikaisuuden. Linuxin ytimeen lisätään tarkempi prosessien ajastin, ajoitusjärjestelmä ja ohjelmallinen keskeytysemulaattori.

Ajoitusta varten on useita eri ajoitusalgoritmeja. Näitä on kolmea eri perustyyppiä:

1. Jakoon perustuva, jossa ohjelmat saavat prosessoriaikaa tasapuolisesti (normaali Linuxin tapa)
2. Prioriteettiin perustuva, jossa tärkein prosessi suoritetaan seuraavaksi
3. Aikaan perustuva, jossa ennaltamäärätyn aikataulun mukaan määrätään suoritusjärjestys

/10, s. 2-3/.

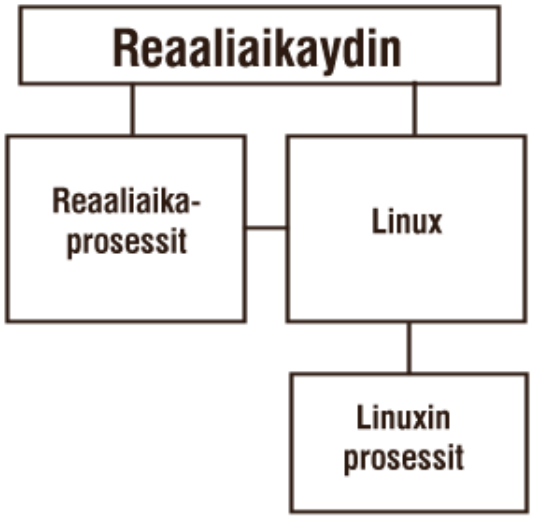
Linux/RK Linux/RK (Resource Kernel) muistuttaa toiminnaltaa RTLinuxia. Reaaliaikaprosessit toimivat erillisessä reaaliaikakernelissä, joka mahdollistaa suorat laitteistokutsut /11, s. 1-2/.

TimeSys Linux GPL Timesys Linux GPL perustuu erillisen resurssiytimen (Linux/RK) käyttöön. Reaaliaikaosat eristetään omaksi osakseen ytimestä ja tietokoneen yhteiset resurssit varataan etukäteen. Timesys Linux GPL tarjoaa prosessien priorisoinnin 2048 eri tasolle. Timesys Linux GPL on suunnattu erityisesti sulautettuihin järjestelmiin /12/.

2.1.3 RTLinuxin toiminta

Kuvassa 1 on kaaviokuva RTLinuxin toiminnasta. Reaaliaikaydin on suoraan yhteydessä tietokoneeseen laitteistotasolla ja reaaliaikaprosessit sekä Linux-

käyttöjärjestelmä käsittelevät laitteistoa sen kautta. Normaalisti Linuxin oma ydin on suoraan yhteydessä laitteistoon. Hallitsemattomat keskeytyskutsut ovat suurin este reaaliaikaisuudelle normaalissa työasemakäyttöjärjestelmässä. Nyt reaaliaikaydin keskeyttää kaikki käyttöjärjestelmän laitteistokutsut ja suorittaa ne vasta, kun reaaliaikaydin on vapaa eli toisin sanoen silloin, kun reaaliaikaprosessit eivät tarvitse prosessoriaikaa /13, s. 1-2/.



Kuva 1. *RTLinux-järjestelmän rakenne.*

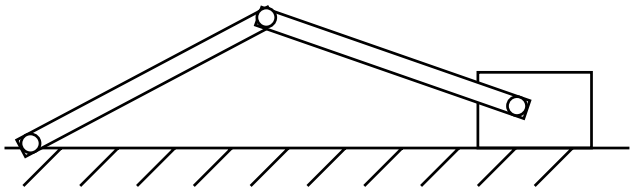
Dynaaminen muisti saattaa johtaa tilanteisiin, joissa alemman prioriteetin prosessi estää korkeamman prioriteetin prosessin muistinhallinnan. Näinollen korkean tason prosessien häiriötön ja ennakoitava suoritus ei ole turvattu. Tämän vuoksi useimmissa RTLinux-järjestelmissä reaaliaikaprosessit eroavat prioriteetin lisäksi tavallisista prosesseista siinä, etteivät ne käytä dynaamista muistia.

RTLinuxin reaaliaikaisuudelle kriittiset osat ovat siis mahdollisimman yksinkertaisia. Järjestelmän osat ovat kuitenkin modulaarisia ja toimintoja on mahdollista lisätä ja muuttaa, esimerkiksi ajoitusalgoritmeja on saatavilla useita erilaisia.

2.2 Monikappaledynamiikan mallintaminen

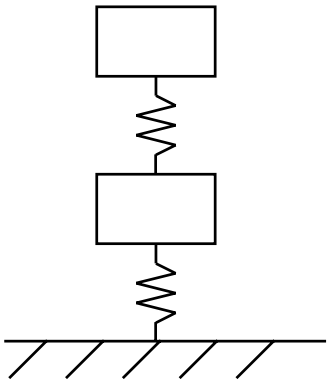
2.2.1 Monikappalejärjestelmä

Monikappaledynamiikan keinoin voidaan käsitellä järjestelmiä, jotka koostuvat nivelten välityksellä toisiinsa vuorovaikutuksessa olevista kappaleista. Kappaleiden kiertymät voivat olla suuria. Suurista kiertymistä syntyvän epälineaarisuuden vuoksi monikappaledynamiikan laskenta on tehtävä numeerisesti. Kuvassa 2 esitetty järjestelmä on monikappalejärjestelmä, koska sen kappaleet ovat nivelten avulla vuorovaikutuksessa toisiinsa.



Kuva 2. *Monikappalejärjestelmä.*

Kuvan 3 järjestelmä sensijaan ei täytä monikappalejärjestelmän ehtoja, koska siinä kappaleet ovat vuorovaikutuksessa pelkästään voimien välityksellä eikä järjestelmällä ole nivelrajoitteita. Tällaisen järjestelmän tarkastelu tosin onnistuu monikappaledynamiikan keinoin, mutta tätä varten on olemassa muitakin työkaluja.

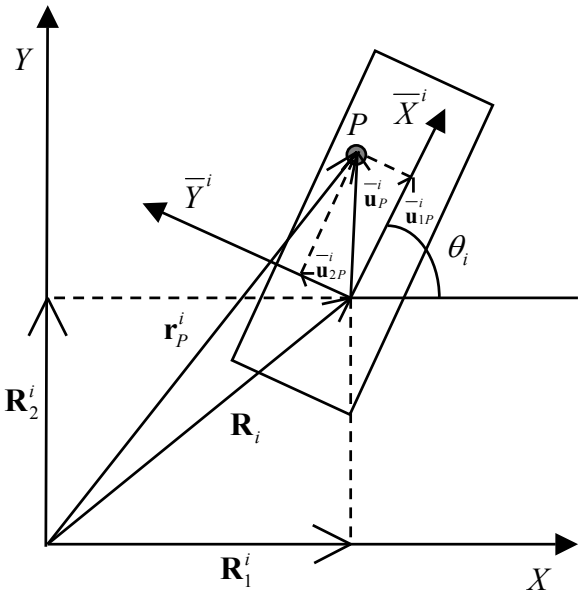


Kuva 3. *Kahden kappaleen järjestelmä.*

Monikappaledynamiikassa kappaleiden välillä voi luonnollisesti olla myös voimavaikutuksia jousien, vaimentimien ja muiden, vaikkapa hydraulisylintereiden, aiheuttamien voimien muodossa.

2.2.2 Tasokinematiikka

Monikappalejärjestelmä mallinnetaan yleensä niin, että kullakin järjestelmän kappaleella on oma lokaali eli paikallinen koordinaatisto. Nämä koordinaatistot liikkuvat kappaleiden mukana. Jos joustoja ei oteta huomioon ovat kappaleen kaikkien partikkeleiden kuvaukset vakioita lokaalissa koordinaatistossa. Kuvassa 4 partikkelin P lokaali asema ilmoitetaan vektoreilla $\overline{\mathbf{u}}_{1P}^{i}$ ja $\overline{\mathbf{u}}_{2P}^{i}$.



Kuva 4. *Jäykän kappaleen i yleistetyt koordinaatit tasotapauksessa.*

Kappaleiden globaali asema määritellään yleistettyjen koordinaattien avulla. Kuvan 4 mukaisessa tasotapauksessa yleistettyihin koordinaatteihin kuuluu kappaleen asema kunkin koordinaattiakselin suhteen ja kiertymä tasossa. Vektoreilla $\mathbf{R}_1^i$ ja $\mathbf{R}_2^i$ ja kulmalla θ^i määrätään kuvan 4 tasotapauksessa kappaleen globaali asema ja kiertymä. Yleistettyjen koordinaattien vektori on tässä tapauksessa siis seuraavanlainen:

$$\mathbf{q}^i = \begin{bmatrix} \mathbf{R}_1^i & \mathbf{R}_2^i & \theta^i \end{bmatrix}^T \tag{1}$$

Kappaleessa olevan partikkelin asema globaalissa koordinaatistossa voidaan määrittää, kun tiedetään partikkelin lokaalit koordinaatit ja kappaleen yleistetyt koordinaatit. Tasotapauksessa lokaalin ja globaalin koordinaatiston välistä kiertymää voidaan kuvata kiertomatriisilla $\mathbf{A}^i$, joka voidaan lausua muodossa:

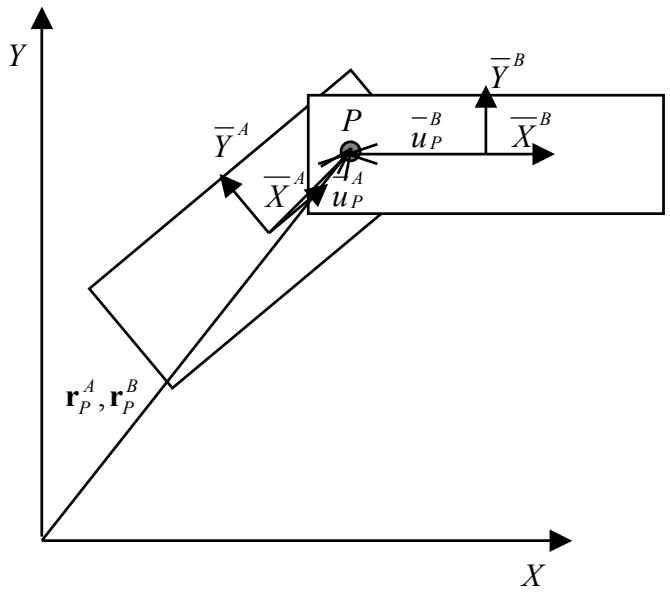
$$\mathbf{A}^i = \begin{bmatrix} \cos\theta^i & -\sin\theta^i \\ \sin\theta^i & \cos\theta^i \end{bmatrix} \tag{2}$$

Kuvassa 4 olevan partikkelin *P* globaali asema voidaan määrittää seuraavasti:

$$\mathbf{r}_P^i = \mathbf{R}^i + \mathbf{A}^i \overline{\mathbf{u}}_P^i \tag{3}$$

Monikappalejärjestelmän määritelmän mukaisesti kappaleiden välillä on rajoitteita. Nivelten perusteella saatavien rajoiteyhtälöiden avulla määritellään niiden vuorovaikutus toisiinsa ja ympäristöön. Tässä työssä tarkasteltavassa järjestelmässä käytetään kierto- ja liukuniveliä. Rajoiteyhtälöiden muodostaminen kiertonivelelle on esitetty seuraavassa.

Kahden kappaleen järjestelmässä kappaleita *A* ja *B* yhdistää pisteessä *P* oleva kiertonivel kuvan 5 mukaisesti. Järjestelmässä ei ole muita rajoitteita.



Kuva 5. *Kahden kappaleen välinen rajoite.*

Nivelpisteen globaalien asemien ($\mathbf{r}_P^A, \mathbf{r}_P^B$) on oltava samat kummassakin kappaleessa. Tämän perusteella voidaan muodostaa nivelen rajoiteyhtälöt. Tasotapauksessa asemavektorit koostuvat kahdesta komponentista. Kiertonivelen tapauksessa saadaan siis kaksi rajoiteyhtälöä:

$$\mathbf{r}_P^A = \mathbf{r}_P^B \tag{4}$$

Yhtälön 3 avulla yhtälö 4 saadaan muotoon:

$$\mathbf{R}^A + \mathbf{A}^A \overline{\mathbf{u}}_P^A = \mathbf{R}^B + \mathbf{A}^B \overline{\mathbf{u}}_P^B \tag{5}$$

Kun yhtälö 5 kirjoitetaan auki, saadaan:

$$\begin{bmatrix} R_1^A \\ R_2^A \end{bmatrix} + \begin{bmatrix} \cos\theta^A & -\sin\theta^A \\ \sin\theta^A & \cos\theta^A \end{bmatrix} \begin{bmatrix} \overline{u}_{1P}^A \\ \overline{u}_{2P}^A \end{bmatrix} = \begin{bmatrix} R_1^B \\ R_2^B \end{bmatrix} + \begin{bmatrix} \cos\theta^B & -\sin\theta^B \\ \sin\theta^B & \cos\theta^B \end{bmatrix} \begin{bmatrix} \overline{u}_{1P}^B \\ \overline{u}_{2P}^B \end{bmatrix} \tag{6}$$

Tästä saadaan kaksi rajoiteyhtälöä:

$$C_1 : \; R_1^A + \overline{u}_{1P}^A \cos\theta^A - \overline{u}_{2P}^A \sin\theta^A - R_1^B - \overline{u}_{1P}^B \cos\theta^B + \overline{u}_{2P}^B \sin\theta^B = 0 \tag{7}$$

$$C_2 : \; R_2^A + \overline{u}_{1P}^A \sin\theta^A + \overline{u}_{2P}^A \cos\theta^A - R_2^B - \overline{u}_{1P}^B \sin\theta^B - \overline{u}_{2P}^B \cos\theta^B = 0 \tag{8}$$

Kun rajoiteyhtälöistä otetaan osittaisderivaatat yleistettyjen koordinaattien suhteen, saadaan Jacobin matriisi $\mathbf{C}_q$ jota tarvitaan liikeyhtälössä. Jacobin matriisissa kullakin rivillä on yhden rajoiteyhtälön osittaisderivaatat joten matriisissa on yhtä monta riviä kuin järjestelmällä on rajoiteyhtälöitä ja yhtä monta saraketta kuin järjestelmällä on yleistettyjä koordinaatteja /14, s. 7-39/.

Kuvan 5 järjestelmän Jacobin matriisi saadaan yhtälöistä 7 ja 8:

$$\boldsymbol{C}_q = \begin{bmatrix} 1 & 0 & -\overline{u}_{1P}^A \sin\theta^A - \overline{u}_{2P}^A \cos\theta^A & -1 & 0 & \overline{u}_{1P}^B \sin\theta^B + \overline{u}_{2P}^B \cos\theta^B \\ 0 & 1 & \overline{u}_{1P}^A \cos\theta^A - \overline{u}_{2P}^A \sin\theta^A & 0 & -1 & -\overline{u}_{1P}^B \cos\theta^B + \overline{u}_{2P}^B \sin\theta^B \end{bmatrix} \tag{9}$$

2.2.3 Tasodynamiikka

Kun kappaleeseen kuuluvien partikkeleiden nopeudet ja tiheydet sijoitetaan kineettisen energian lauseeseen, saadaan massamatriisi $\mathbf{M}^i$. Sen avulla kuvataan kappaleen massa, hitausmomentit ja hitaustulo yhtälön 10 mukaisesti. Hitaustulo kuvaa massakeskipisteen asemaa lokaalissa koordinaatostossa ja se on nolla, mikäli massakeskipiste on lokaalin koordinaatiston origossa.

$$\mathbf{M}^i = \begin{bmatrix} \mathbf{m}^i_{RR} & \mathbf{m}^i_{R\theta} \\ \mathbf{m}^i_{\theta R} & m^i_{\theta\theta} \end{bmatrix} \tag{10}$$

Koko järjestelmän massamatriisi saadaan koottua yksittäisten kappaleiden massamatriiseista. Kun kappaleiden lukumäärä on *n*, järjestelmän massamatriisi voidaan lausua muodossa:

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}^1 & & & 0 \\ & \mathbf{M}^2 & & \\ & & \ddots & \\ 0 & & & \mathbf{M}^n \end{bmatrix} \tag{11}$$

missä $\mathbf{M}^i$ = kappaleen *i* massamatriisi

Kappaleen pyörimisestä aiheutuu keskipakovaikutus kappaleen lokaaliin koordinaatistoon mikäli lokaalin koordinaatiston origo ei ole massakeskipisteessä. Tämä keskipakovaikutus huomioidaan neliöllisen nopeusvektorin avulla. Tässä työssä keskitytään kuitenkin tapauksiin joissa lokaalin koordinaatiston origo on kappaleen massakeskipisteessä. Tällöin neliöllinen nopeusvektori $\mathbf{Q}_v$ on nolla.

Yleistetyn voimavektorin $\mathbf{Q}_e$ avulla määritetään kuhunkin kappaleeseen vaikuttavat yleistetyt voimat ja vääntömomentit. Yleistetyt voimat ovat kappaleisiin vaikuttavat voimat siirrettynä lokaaliin koordinatistoon. Kukin alkio liittyy yhteen yleistettyyn koordinaattiin.

Järjestelmän rajoitteet voidaan huomioida liikeyhtälöitä muodostettaessa kahdella eri tavalla. Rajoiteyhtälöt voidaan lisätä liikeyhtälöihin Lagrangen kertoimien $\boldsymbol{\lambda}$ avulla, jolloin järjestelmän dynamiikan kuvaamiseen käytetään yhtä monta differentiaaliyhtälöä kuin järjestelmällä on yleistettyjä koordinaatteja sekä rajoiteyhtälöiden verran algebraliyhtälöitä. Rajoitevoimat saadaan nyt suoraan Lagrangen kertoimien avulla. Saadut algebraliyhtälöt ovat yleensä melko yksinkertaisia ja siten nopeita ratkaista ja muodostaa.

Toinen tapa rajoitteiden huomioimiseen on rajoiteyhtälöiden sijoittaminen suoraan liikeyhtälöihin. Näin saadaan järjestelmän vapausasteiden mukainen määrä differentiaaliyhtälöitä. Tässä työssä käytettiin Lagrangen kertoimien menetelmää. Tällöin liikeyhtälöiden saattamiseksi integroitavaan muotoon täytyy niihin lisätä rajoitteiden muutosta kuvaava vektori $\mathbf{Q}_c$.

Järjestelmän dynamiikka voidaan ratkaista yhtälöstä:

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{C}_\mathbf{q}^T\boldsymbol{\lambda} = \mathbf{Q}_e + \mathbf{Q}_v \tag{12}$$

Sijoittamalla yhtälöön 12 vektori $\mathbf{Q}_c$ saadaan se muotoon:

$$\begin{bmatrix} \mathbf{M} & \mathbf{C}_\mathbf{q}^T \\ \mathbf{C}_\mathbf{q} & 0 \end{bmatrix}\begin{bmatrix} \ddot{\mathbf{q}} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_e + \mathbf{Q}_v \\ \mathbf{Q}_c \end{bmatrix} \tag{13}$$

Ratkaisemalla yhtälö 13 kiihtyvyyksien ja Lagrangen kertoimien suhteen saadaan:

$$\begin{bmatrix} \ddot{\mathbf{q}} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{M} & \mathbf{C}_\mathbf{q}^T \\ \mathbf{C}_\mathbf{q} & 0 \end{bmatrix}^{-1}\begin{bmatrix} \mathbf{Q}_e + \mathbf{Q}_v \\ \mathbf{Q}_c \end{bmatrix} \tag{14}$$

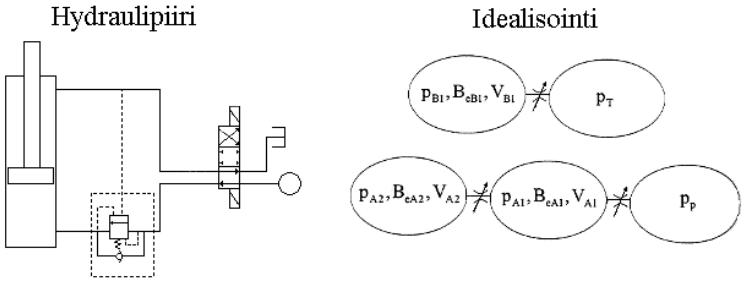
Yhtälöstä 14 saadaan laskettua järjestelmän yleistettyjen koordinaattien kiihtyvyydet ja Lagrangen kertoimet. Kun kiihtyvyyksiä integroidaan ajan suhteen, saadaan nopeudet ja uudelleen integroimalla asemat /14, s. 39-48/.

2.3 Hydrauliikan mallintaminen keskittyneiden paineiden teorian mukaan

Keskittyneiden paineiden teoriassa mallinnettava hydraulipiiri jaetaan tilavuuksiin, joissa paine oletetaan tasan jakautuneeksi. Todellisessa hydraulipiirissä vaikuttavat erilaiset virtausvastukset kuten venttiilit kuvataan tilavuuksien välisillä kuristimilla. Kullekin tilavuudelle muodostetaan differentiaaliyhtälöt paineiden laskemiseksi. Paineiden avulla saadaan laskettua tilavuusvirrat kuristimien läpi.

2.3.1 Hydraulipiirin idealisointi

Kuvassa 6 on esitetty yksinkertainen hydraulipiiri idealisoituna keskittyneiden paineiden menetelmällä. Hydraulipiirin venttiilit on kuvattu kuristimilla ja letkut on yhdistetty samaan tilavuuteen sylinterin kanssa.



Kuva 6. *Hydraulipiirin idealisointi./14, s. 68/*

Virtauksen jatkuvuusyhtälön perusteella saadaan paineen *p* aikaderivaatta laskettua yhtälön 15 mukaisesti /14, s. 68-70/.

$$\frac{dp}{dt} = \frac{B_e}{V}\left(Q_i - Q_o - \frac{dV}{dt}\right) \tag{15}$$

missä B_e = tutkittavan tilavuuden tehollinen puristuskerroin

Q_i = tilavuusvirta tilavuuteen

Q_o = tilavuusvirta tilavuudesta

V = tilavuus

2.3.2 Venttiilien mallinnus

Venttiilien mallinnukseen on olemassa kolme tapaa: empiirinen, puoliempiirinen ja analyyttinen. Tässä työssä on käytetty puoliempiiristä tapaa. Menetelmän periaatteena on se, että venttiilin toimintaa kuvaavat analyyttiset yhtälöt muokataan sellaiseen muotoon, jossa venttiilin ominaisuudet voidaan kuvata helposti saatavilla olevilla parametreilla. Tämä menetelmä on yleensä sopivin mekatronisten koneiden hydrauliikkaa mallinnettaessa koska tarvittavat parametrit ovat saatavilla komponenttivalmistajien luetteloista. Turbulenttinen tilavuusvirta venttiilin läpi puoliempiirisen mallinnustavan mukaisesti on: /14, s. 75-78/.

$$Q = C_v U \sqrt{\Delta p} \tag{16}$$

missä Q = tilavuusvirta

C_v = puoliempiirinen tilavuusvirtavakio

U = venttiilin karan asemaa kuvaava takaisinkytkentäjännite

Δp = paine-ero venttiilin yli

2.3.3 Hydraulisylinterin mallinnus

Toimilaitteet muuttavat hydraulisen paineen monikappalejärjestelmään vaikuttaviksi voimiksi. Tässä työssä tarkastellaan hydraulisylinteriä, jonka tuottama voima saadaan laskettua seuraavasti:

$$F_s = A_1 p_1 - A_2 p_2 - \Sigma F_\mu \tag{17}$$

missä F_s = sylinterin tuottama voima

A_1, A_2 = sylinterin männän eri puolien pinta-alat

p_1, p_2 = paineet sylinterin eri puolilla

ΣF_μ = sylinterin kitkavoimat

Kitkavoimat syntyvät metallin ja tiivisteiden välisestä kosketuksesta. Tämä kitka riippuu epälineaarisesti männän liikenopeudesta. Eräs approksimaatio kitkavoimalle on: /14, s. 82-84/.

$$F_\mu = \xi(\dot{x})(A_1 p_1 - A_2 p_2)(1 - \eta) \tag{18}$$

missä F_μ = kitkavoima

$\dot{x}$ = sylinterin liikenopeus

$\xi(\dot{x})$ = sylinterikohtainen nopeuden funktio

η = sylinterin hyötysuhde

2.4 Yhtälöryhmän ratkaisu

Tässä työssä järjestelmän liikeyhtälömatriisi 13 ratkaistiin yleistettyjen koordinaattien kiihtyvyyksien suhteen numeerisesti käyttäen Gaussin eliminointia, joka on tehokas menetelmä lineaaristen yhtälöryhmien ratkaisuun.

Ratkaistava yhtälöryhmä on muotoa:

$$\mathbf{Ax} = \mathbf{b} \tag{19}$$

missä **A** = kerroinmatriisi
x = vektori
b = vektori

Kun järjestelmässä on *n* kappaletta tuntemattomia ja yhtälöitä, yhtälöryhmä on muotoa:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{20}$$

Eliminointivaiheessa yhtälöryhmä muutetaan rivioperaatioiden (vaihdetaan yhtälöiden paikkaa, kerrotaan yhtälöitä nollasta poikkeavalla luvulla sekä lisätään yhtälöitä toisiinsa) avulla yläkolmiomuotoon:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n = b_1 \\ a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + \ldots + a_{2n}^{(1)}x_n = b_2^{(1)} \\ \vdots \\ a_{nn}^{(n-1)}x_n = b_n^{(n-1)} \end{cases} \tag{21}$$

Kunkin yhtälön ensimmäistä kerrointa sanotaan pivot-alkioksi. Työssä on käytetty ns. osittaista pivotointia. Tällöin pivot-alkioksi otetaan $n - i + 1$:n sarakkeen suurin x_i:n kerroin.

Tämän jälkeen yhtälöryhmä voidaan ratkaista laskemalla x_n viimeisestä yhtälöstä ja sijoittamalla se seuraavaan yhtälöön, josta saadaan ratkaistua x_{n-1}. Näin jatkaen saadaan kaikki tuntemattomat ratkaistua /15, s. 367-368/.

2.5 Numeerinen integrointi

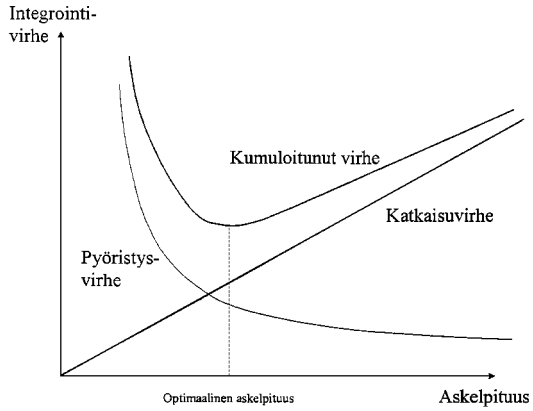
Järjestelmän liikeyhtälömatriisista 13 Gaussin eliminaatiolla ratkaistut kiihtyvyydet voidaan integroida numeerisesti kolmella eri tavalla:

1. Integroidaan suoraan yleistettyjä koordinaatteja. Tällöin integroinnin on oltava riittävän tarkka, ettei rikota rajoitteita. Työssä käytettiin tätä menetelmää reaaliaikamallin ratkaisuun.
2. Yleistetyt koordinaatit jaetaan riippuviin ja riippumattomiin koordinaatteihin. Vain riippumattomia koordinaatteja integroidaan. Loput koordinaatit ratkaistaan rajoiteyhtälöistä. Riippumattomien koordinaattien integrointitarkkuus vaikuttaa koko järjestelmän tulosten tarkkuuteen.
3. Kaikki yleistetyt koordinaatit integroidaan. Riippuvien koordinaattien tulosta tarkennetaan rajoiteyhtälöiden avulla.

Koska numeerisessa integroinnissa approksimoidaan oikeata tulosta, syntyy aina virhettä. Erilaisia virhelähteitä on useita:

- Katkaisuvirhe muodostuu toleranssista, joka määrittelee iteraatioprosessin tarkkuuden.
- Pyöristysvirhe aiheuttaa järjestelmässä luonnotonta vaimennusta. Mikäli yhtälöiden ratkaisussa ei ole riitävää tarkkuusvaraa, menetetään lukujen pyöristysvaiheessa tietoa.
- Numeerinen epästabiilius muodostuu differentiaaliyhtälöiden, askelpituuden sekä käytetyn integrointialgoritmin yhteisvaikutuksesta. Epästabiili laskenta voi aiheuttaa suuria virheitä tuloksissa ja myös laskennan epäonnistumisen.

Integrointivirhe muodostuu katkaisu- ja pyöristysvirheen summasta kuten kuvassa 7 on esitetty. Pienin integrointivirhe saavutetaan, jos virhelähteet voidaan määrittää ja käytetään askelta, jolla virheiden summa on pienin.



Kuva 7. *Numeerisen integroinnin virhelähteitä /14, s. 53/*

2.5.1 Eulerin menetelmä

Ensimmäisen kertaluvun yhden tuntemattoman tavallinen differentiaaliyhtälö ajan funktiona on muotoa:

$$y'(t) = f(t, y(t)) \tag{22}$$

Yhtälöllä 22 on yleensä parvi ratkaisuja. Jotta ratkaisu olisi yksikäsitteinen, annetaan ratkaisulle alkuehto $y(t_0) = y^0$. Differentiaaliyhtälöllä tarkoitetaan jatkossa alkuarvotehtävää ensimmäisen kertaluvun differentiaaliyhtälösysteemille:

$$\begin{cases} y'(t) = f(t, y(t)) \\ y(t_0) = y^0 \end{cases} \tag{23}$$

missä $y=(y_1, y_2, \ldots, y_n)$

Taylorin sarjakehitelmän avulla yhtälöstä 22 saadaan ajanhetkellä $t+\Delta t$:

$$\begin{aligned} y(t+\Delta t) &= y(t)+\Delta t y'(t)+O(\Delta t^2) \\ &= y(t)+\Delta t f(t,y)+O(\Delta t^2) \end{aligned} \tag{24}$$

missä O = virhetermi

Δt = aika-askel

Eulerin integrointimenetelmä on muotoa:

$$y_0 = y^0 \tag{25}$$

$$y_{n+1} = y_n + \Delta t f(t_n, y_n) \tag{26}$$

missä $n = 0,1,2,\ldots$

$t_n = t_0 + n\Delta t$

$y_n = y(t_n)$

Yhtälöstä 26 saadaan integraalin arvo ajanhetkellä n+1. Menetelmä on hyvin yksinkertainen koska kullakin aika-askelella lasketaan funktiolle vain yksi arvo, jonka avulla approksimoidaan integraalin arvo. Menetelmän tarkkuus on kuitenkin niin huono, että edes hyvin pienillä aika-askelilla menetelmä ei sovellu mekatronisten koneiden simulointiin /16, s. 97-98/.

2.5.2 Neljännen kertaluvun Runge-Kutta -menetelmä

Koska numeerisen integroinnin tarkkuus vaikuttaa merkittävästi simuloinnin tulosten tarkkuuteen, on kehitetty joukko erilaisia tarkempia integrointimenetelmiä. Runge-Kutta –menetelmissä lasketaan funktion f arvo useammalla ajanhetkellä. Tunnetuin ja käyttökelpoisin on neljännen kertaluvun Runge-Kutta –menetelmä, jossa funktion arvo

määritetään neljä kertaa yhden askeleen aikana. Näiden arvojen laskenta on esitetty yhtälöissä 27-30.

$$k_1 = \Delta t f\left(t_n, y_n\right) \tag{27}$$

$$k_2 = \Delta t f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{k_1}{2}\right) \tag{28}$$

$$k_3 = \Delta t f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{k_2}{2}\right) \tag{29}$$

$$k_4 = \Delta t f\left(t_n + \Delta t, y_n + k_3\right) \tag{30}$$

Kun funktion arvot eri pisteissä on laskettu, voidaan niiden avulla laskea funktion arvo pisteessä n+1 seuraavasti:

$$y_{n+1} = y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O\left(\Delta t^5\right). \tag{31}$$

Neljännen kertaluvun Runge-Kutta –mentelmä soveltuu hyvin simulointiin, koska se on helppo ohjelmoida tietokoneelle ja saavutettava laskentatarkkuus on hyvä. Juuri hyvän tarkkuuden vuoksi laskenta on myös tehokasta, koska näin voidaan käyttää suurta aika-askelta /16, s. 98-99/.

2.6 Käytetyt työkalut

2.6.1 Tietokonelaitteisto

Työssä käytettiin kahta PC-standardin mukaista tietokonetta. Tietokone, johon RTLinux-reaaliaikajärjestelmä asennettiin oli varustettu Asus P28 –emolevyllä, Intel Pentium II 450 MHz –prosessorilla ja 384 MB:lla 100 MHz SDRAM –muistia. Tulosten vertailuun käytettiin tietokonetta, jossa oli Asus P2-99 –emolevy, Pentium III 600 MHz –prosessori ja 512 MB 100 MHz SDRAM –muistia.

2.6.2 Ohjelmistot

Käytetty Linux-levitysversio on Mandrake Linux 8.2 ja reaaliaikalaajennus RTLinux 3.0. Linuxissa työskenneltiin graafisessa KDE-työpöytäympäristössä. Tulosten vertailuun käytettiin Adams 10.0 –ohjelmistoa Microsoft Windows 2000 –käyttöjärjestelmässä. Tulosten visualisointiin käytettiin Matlab R12 –ohjelmistoa.

3 SOVELLUS

3.1 RTLinux-järjestelmä

3.1.1 RTLinuxin asentaminen

Aluksi tietokoneeseen asennetaan normaali Linux-käyttöjärjestelmä huomioiden muutamat RTLinuxin asettamat vaatimukset. Nyt käytetyn Mandrake Linux 8.2 – levitysversion ytimen versio on 2.4.18 pienin muutoksin. Koska 2.4.4 on uusin RTLinuxin tukemista ytimistä, ei järjestelmässä voida käyttää kaikkia 2.4.18-versioon sisältyviä ominaisuuksia, kuten journaloivaa tiedostojärjestelmää ext3. Ohjelmistokehitykseen liittyvät työkalut on asennettava. Lisäksi asennettiin Patch-ohjelma, jota tarvitaan, kun reaaliaikalaajennus asennetaan ytimeen. Mandraken ja useimpien muidenkin levitysversioiden asennusohjelma osaa yleensä asettaa tietokonelaitteiston asetukset ja asentaa tarvittavat laiteohjaimet.

Kun tietokoneeseen on asennettu Linux, voidaan aloittaa itse RTLinux-reaaliaikalaajennuksen asennus. Aluksi asennetaan ytimen 2.4.4 puhdas versio (niin kutsuttu vanilla-kernel), jossa ei ole ylimääräisiä levitysversioiden muutoksia. Uuteen ytimeen asennetaan RTLinuxin ydinpäivitys Patch-ohjelman avulla.

Ennen ytimen kääntämistä sen asetukset on asetettava kohdalleen paitsi laitteiston, myös reaaliaikaisuuden osalta. Virranhallintaominaisuudet on kytkettävä pois päältä, koska niillä voi olla arvaamattomia vaikutuksia reaaliaikaohjelmien suoritukseen. Kun asetukset on tehty, käännetään ydin ja siihen liittyvät modulit. Kääntämisen jälkeen modulit asennetaan.

Viimeisenä vaiheena asennuksessa on kopioida RTLinux-laajennettu ydin käynnistyshakemistoon ja lisätä vastaava vaihtoehto käynnistysvalikkoon. Näin tietokoneen käynnistysvaiheessa voidaan valita, halutaanko toimia tavallisessa Linuxissa vai RTLinuxissa. RTLinuxin asennus on esitetty käskytasolla liitteessä 1.

Reaaliaikaisten ohjelmien suorittamiseksi tietokone on siis käynnistettävä reaaliaikaytimeen. Sen jälkeen on vielä käynnistettävä reaaliaikapalvelut toteuttavat modulit.

3.1.2 Reaaliaikamallin suorittaminen

Aluksi C-kielinen ohjelma on käännettävä objektitiedostoksi, joka voidaan ladata reaaliaikaytimeen moduliksi. RTLinux-ohjelmien kääntäminen ei onnistu kaikilla kääntäjillä, kuten Mandrake Linuxin mukana tulleella gcc:n versiolla 2.96. Työssä on käytetty kääntäjänä gcc 2.95.3-versiota, joka on yhteensopiva RTLinux-ohjelmien kanssa.

Kääntämisen jälkeen moduli ladataan reaaliaikaytimeen. Listaamalla suoritettavat modulit voidaan tarkistaa latauksen onnistuminen. Käynnissä on oltava oman modulin lisäksi RTLinuxin reaaliaikamodulit.

Reaaliaikamodulin suoritus ei näy ulospäin mitenkään, ellei ohjelmassa ole esimerkiksi tulostuskäskyjä. Tulosteiden näkyvyys käyttäjälle riippuu Linux-ympäristön asetuksista. Mikäli tulosteet eivät tule suoraan näkyviin, voidaan ne saada näytölle erillisellä komennolla.

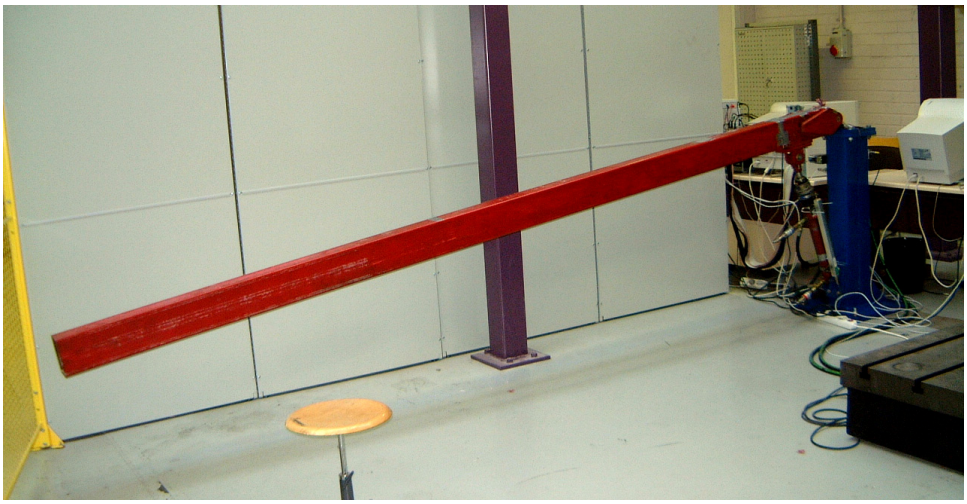
3.1.3 Reaaliaikamallin pysäyttäminen

Reaaliaikamodulista voidaan tehdä sellainen, ettei suorituksella ole varsinaista loppua. Mikäli ohjelmassa ei ole omaa mekanismia suorituksen pysäyttämiseksi, on moduli poistettava käsin. Tässä työssä tarkastellussa simulointimallissa ohjelman suoritus lopetaan asetetun simulointiajan täytyttyä.

3.2 Tarkasteltava kohde

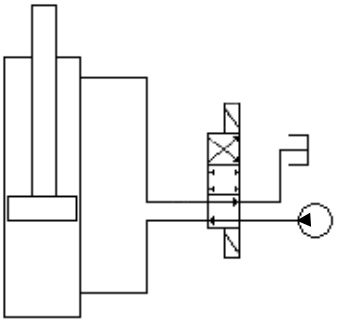
3.2.1 Hydraulikäyttöinen puomi

Työssä tarkasteltu hydraulikäyttöinen puomi sijaitsee Mekatroniikan ja virtuaalisuunnittelun laboratoriossa. Siinä on neljä kappaletta (puomi, runko ja hydraulisylinterin osat) ja yksi vapausaste, jota ohjataan hydraulisylinterillä. Mekaniikka on esitelty tarkemmin lähteessä 17 sivulla 51. Puomi on esitetty kuvassa 8.



Kuva 8. *Tarkasteltava hydraulikäyttöinen puomi.*

Hydraulipiiri on esitetty kuvassa 9. Keskittyneiden paineiden teorian mukaisesti idealisoituna piirissä on kaksi tilavuutta, joissa paine oletetaan tasan jakautuneeksi. Tilavuus A käsittää sylinterin männänpuolen tilavuuden ja hydrauliletkun venttiilille, tilavuus B vastaavasti sylinterin männänvarren puolisen tilavuuden ja letkun venttiilille. Syöttö- ja tankkipaineet oletetaan vakioiksi. Hydraulipiirin simulointiparametrit on esitetty taulukossa 1. Simuloinnin alkuarvot on esitetty työkierron yhteydessä kappaleessa 3.2.2.



Kuva 9. *Tarkasteltava hydraulipiiri.*

Taulukko 1. *Hydraulijärjestelmän parametrit.*

Syöttöpaine	$1{,}5 \cdot 10^7\ Pa$
Tankkipaine	$10^5\ Pa$
Suuntaventtiilin puoliempiirinen tilavuusvirtavakio	$1{,}069 \cdot 10^{-8}\ \frac{m^3}{sV\sqrt{Pa}}$
Sylinterin halkaisija	$0{,}05\ m$
Sylinterin männänvarren halkaisija	$0{,}03\ m$
Keskittyneiden paineiden tilavuudet	$1{,}0632 \cdot 10^{-4}\ m^3$
Sylinterin hyötysuhde	0,9
Öljyn puristuskerroin	$1{,}5 \cdot 10^9\ Pa$
Sylinterin joustoa kuvaava puristukerroin	$21 \cdot 10^9\ Pa$

3.2.2 Simuloitava työkierto

Tässä työssä ei syvennytty RTLinuxin I/O-toimintoihin, joten työkiertoa ei voitu tehdä oikealla ohjaimella käyttäjän ohjaamana. Sensijaan työkierto tehtiin step-funktioiden avulla. Step-funktion kutsu on muotoa:

$$y = STEP(x, x_0, y_0, x_1, y_1) \tag{32}$$

missä y = funktion arvo

x = suure, jonka funktiona arvo lasketaan

x_0 = askelen alkupiste

y_0 = funktion arvo askelen alussa

x_1 = askelen loppupiste

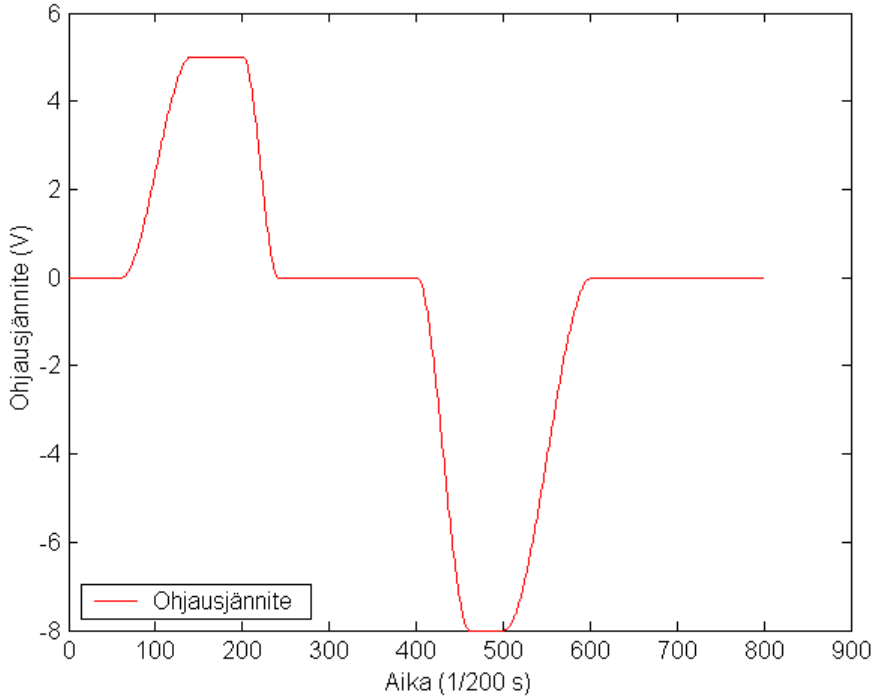
y_1 = funktion arvo askelen lopussa

Kun x on pienempi kuin askelen alkupiste, funktion arvo on y_0 ja vastaavasti askelen loppupistettä suuremmilla x:n arvoilla funktion arvo on y_1. Kun ollaan askelen alku- ja loppupisteiden välillä, lasketaan funktion arvo yhtälön 33 mukaisesti:

$$y = y_0 + (y_1 - y_0)\left(\frac{x - x_0}{x_1 - x_0}\right)^2\left(3 - 2\left(\frac{x - x_0}{x_1 - x_0}\right)\right) \tag{33}$$

Työssä tarkastellussa työkierrossa hydraulijärjestelmän ohjausjännite on yhtälön 34 mukainen. Kuvassa 10 on esitetty näin saatu ohjausjännite ajan funktiona. Työkierron pituus on neljä sekuntia.

$$\begin{aligned} U = {} & STEP(time,0.3,0,0.7,5) + STEP(time,1,0,1.2,-5) + \\ & STEP(time,2,2.3,-8) + STEP(time,2.5,0,3,8)) \end{aligned} \tag{34}$$



Kuva 10. *Ohjausjännite.*

Simuloinnin alussa puomi on vaaka-asennossa. Alkutilanteen parametrit on esitetty taulukossa 2.

Taulukko 2. *Järjestelmän alkutila.*

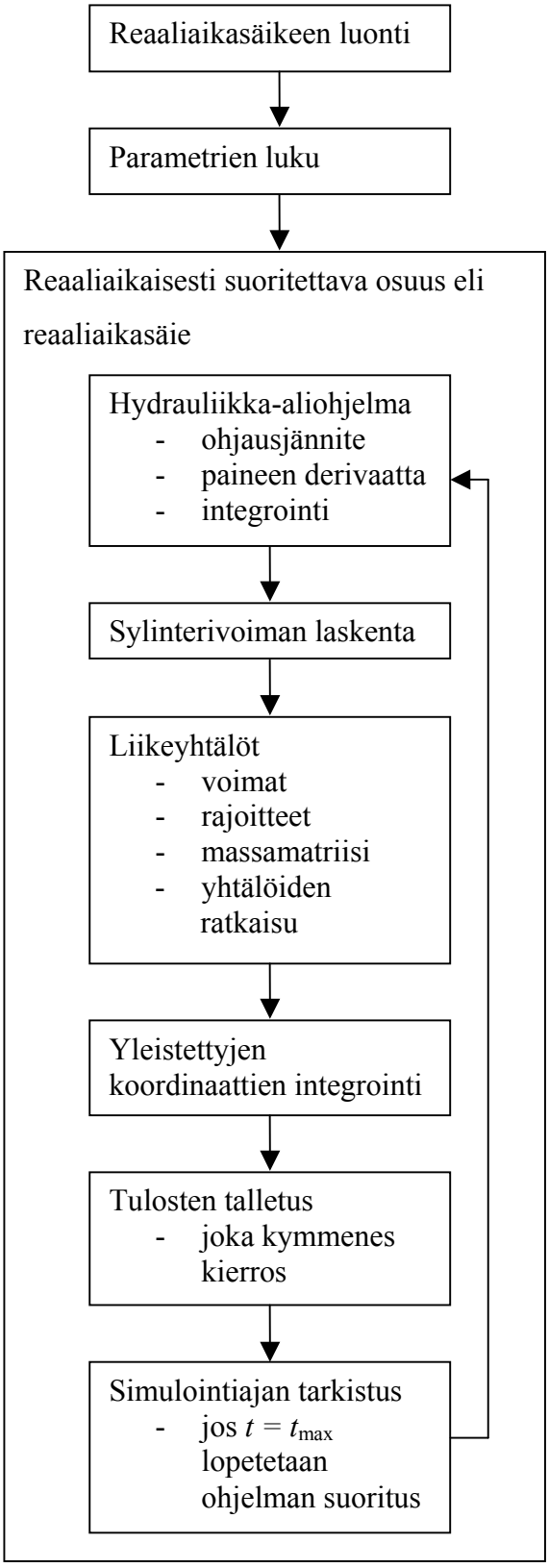
Puomin kiertymä vaakatasosta	0 *rad*
Paine portissa A	$9{,}2 \cdot 10^6$ *Pa*
Paine portissa B	$1{,}1 \cdot 10^7$ *Pa*
Männän asema sylinterissä	0,32 *m*

3.3 Reaaliaikainen simulointimalli

3.3.1 Yleistä

Simulointimalli tehtiin C-kielellä ja se pohjautuu lähteessä 17 esitettyyn simulointimallin rakenteeseen. Kuvassa 11 on kaavio, jossa on esitetty ohjelman toiminta. Malli on jaoteltu aliohjelmiin, joita kutsutaan vuorollaan reaaliaikasäikeestä. Reaaliaikasäikeen suoritusta ohjaa reaaliaika-ajastin. Jokaisella mallin suorituskerralla odotetaan ennen laskennan aloittamista niin kauan, että edellisen kierroksen suorituksesta on kulunut asetettu aika. Näin simulointi tapahtuu reaaliaikaisena huolimatta mallin ratkaisemiseen kuluvasta ajasta.

Ensimmäisellä suorituskerralla yleistettyjen koordinaattien ratkaisussa käytetään annettuja alkuarvoja asemille ja nopeuksille. Liikeyhtälöt ratkaistaan kiihtyvyyksien suhteen ja kiihtyvyydet integroimalla saadaan nopeudet ja asemat. Seuraavalla kierroksella käytetään saatuja arvoja. Ohjelmaa suoritetaan reaaliaika-ajastimen ohjaamana niin monta kertaa, että asetettu simuloinnin loppuaika saavutetaan.



Kuva 11. *Simulointimallin ratkaisu.*

3.3.2 Simuloitavan järjestelmän parametrit

Simuloitavan järjestelmän parametrit kirjoitetaan ohjelmakoodiin. Ohjelman konekielelle kääntämisen jälkeen malliin ei siis voida tehdä muutoksia. Järjestelmän kullekin kappaleelle annetaan inertiatiedot, alkutilanteen asemat ja nopeudet. Järjestelmän rajoitteet määritellään siihen liittyvien kappaleiden lokaaleissa koordinaatistoissa. Hydrauliikkajärjestelmän parametrit (taulukko 2) annetaan myös ohjelmakoodissa.

3.3.3 Hydrauliikka-aliohjelma ja sylinterivoima

Aluksi lasketaan työkierron mukainen ohjausjännite kappaleen 3.2.2 mukaisesti. Kappaleen 2.3 mukaisesti lasketaan paineiden aikaderivaatat, josta integroimalla saadaan keskittyneet paineet. Integrointi tapahtuu käyttäen kappaleessa 2.4.2 esitettyä neljännen kertaluvun Runge-Kutta –menetelmää.

Hydraulijärjestelmän paineiden avulla saadaan laskettua sylinterin voima. Sylinterivoima lasketaan kappaleessa 2.3.3 esitettyjen kaavojen mukaisesti.

3.3.4 Liikeyhtälöt

Liikeyhtälömatriisi 13 kootaan järjestelmän tilan mukaiseksi. Hydraulisylinterin voima lisätään yleistettyyn voimavektoriin. Liikeyhtälömatriisista ratkaistaan kiihtyvyydet ja Lagrangen kertoimet Gaussin eliminoinnin avulla. Ratkaisua varten yhtälön oikeanpuoleiset termit siirretään vasemmalla sijaitsevan matriisin pystysarakkeeksi siten, että matriisi saadaan muotoon:

$$\begin{bmatrix} \mathbf{M} & \mathbf{C}_q^T & \mathbf{Q}_e \\ \mathbf{C}_q & 0 & \mathbf{Q}_c \end{bmatrix} \tag{35}$$

Soveltamalla tähän kappaleessa 2.4 esiteltyä Gaussin eliminaatiota saadaan kiihtyvyysvektori $\ddot{\mathbf{q}}$ ja lagrangen kertoimet $\boldsymbol{\lambda}$ määritettyä.

3.3.5 Yleistettyjen koordinaattien integrointi

Kuten hydraulijärjestelmän paineiden, myös yleistettyjen koordinaattien integrointiin käytetään neljännen kertaluvun Runge-Kutta –integrointialgoritmia. Gaussin eliminaation avulla saaduista kiihtyvyyksistä intergoidaan nopeudet ja asemat. Toisen asteen differentiaaliyhtälöt ratkaistaan muutamalla ne ensimmäisen asteen differentiaaliyhtälöiksi sijoittamalla nopeus- ja kiihtyvyysvektorit samaan vektoriin: /17, s. 41-42/.

$$\dot{\mathbf{y}}^{(t)} = \begin{bmatrix} \dot{\mathbf{q}} \\ \ddot{\mathbf{q}} \end{bmatrix} \tag{36}$$

Integroimalla yhtälö 36 ajan suhteen saadaan vektori, joka sisältää asemat ja nopeudet:

$$\mathbf{y}^{(t+\Delta t)} = \begin{bmatrix} \mathbf{q} \\ \dot{\mathbf{q}} \end{bmatrix} \tag{37}$$

3.3.6 Simulointitulosten tallentaminen

Simulointitulosten talteenotto on toteutettu lisäämällä tulostuskäsky simulointimalliin. Tulosten tarkastelua varten ei ole tarpeen ottaa talteen tulosta jokaisella aika-askelella koska käytetty aika-askel on useista kertaluokkia pienempi kuin tarkasteltavan järjestelmän ominaisvärähtelyn jaksonaika. Tästä syystä tulokset tallennettiin joka kymmenennellä aika-askelella.

Simulointimallin antamat tulokset ovat liukulukuja. Koska RTLinuxin reaaliaikaohjemissa käytettävä tulostuskäsky ei kykene liukulukujen tulostukseen, oli tulostamista varten tehtävä erillinen aliohjelma. Aliohjelman avulla voidaan tulostaa mikä tahansa suure järjestelmässä halutulla tarkkuudella. Tuloste menee ytimen rengaspuskuriin (kernel ring buffer), joka simulointiajon päätyttyä voidaan tallentaa suoraan tekstimuotoiseksi tiedostoksi jatkotoimenpiteitä varten.

3.3.7 Simulointiasetukset

Simuloinnissa käytettiin 0,001 sekunnin aika-askelta. Aika-askel voidaan määrittää erikseen molemmille integraattoreille ja reaaliaika-ajastimelle. Integraattoreiden aika-askel vaikuttaa tulosten tarkkuuteen ja ajastimen aika-askel laskennan nopeuteen. Kun aika-askeleet ovat yhtä suuret, tapahtuu laskenta reaaliajassa.

3.4 Adams-malli

Simulointimalli tehtiin edellä esitettyjen parametrien mukaisesti sekä mekaniikan että hydrauliikan osalta samanlaiseksi kuin reaaliaikamalli. Mekaniikka ja nivelrajoitteet on mallinnettu käyttäen ohjelman omia työkaluja. Hydrauliikka on mallinnettu käyttäen Adamsin sisäisiä funktioita. Mallintaminen tapahtui kappaleessa 2.3 esitettyjen yhtälöiden mukaisesti. Simuloinnissa käytettiin Adamsin Gear-integraattoria aika-askelella 0,0005 sekuntia. Jacobin matriisi laskettiin joka neljännellä kierroksella.

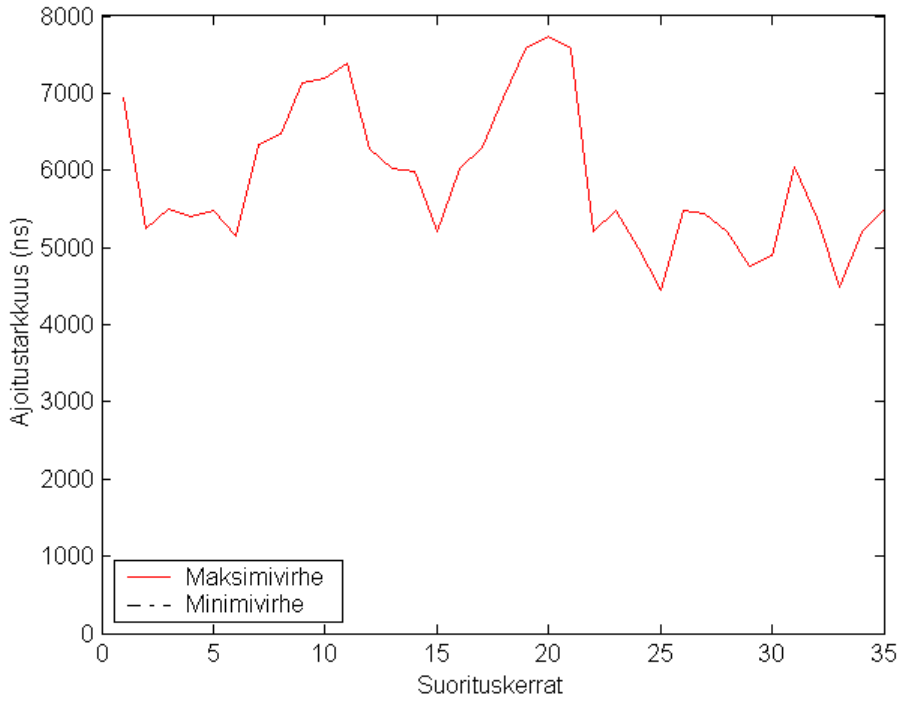
4 TULOKSET JA NIIDEN TARKASTELU

4.1 RTLinuxin toiminta

RTLinuxin toimintaa testattiin ohjelmistopaketin mukana tulevilla mittaustyökaluilla. Tulosten perusteella voidaan tarkastella reaaliaikaisuuden toteutumista. Testit suoritettiin käyttäen graafista käyttöliittymää.

4.1.1 Ajoitustarkkuus

RTLinuxin reaaliaikaprosessit on suoritettava tarkasti oikeaan aikaan. Ajoituksen tarkkuuden mittaamista varten on olemassa ohjelma, joka ajaa reaaliaikasäiettä ja mittaa sen suoritushetken. Ohjelman C-kielinen lähdekoodi on esitetty liitteessä 2. Tuloksena saadaan minimi- ja maksimipoikkeamat odotetusta suoritushetkestä. Kuvassa 12 on esitetty ajoitustarkkuus 35 eri reaaliaikaisen prosessin suorituskerralla.

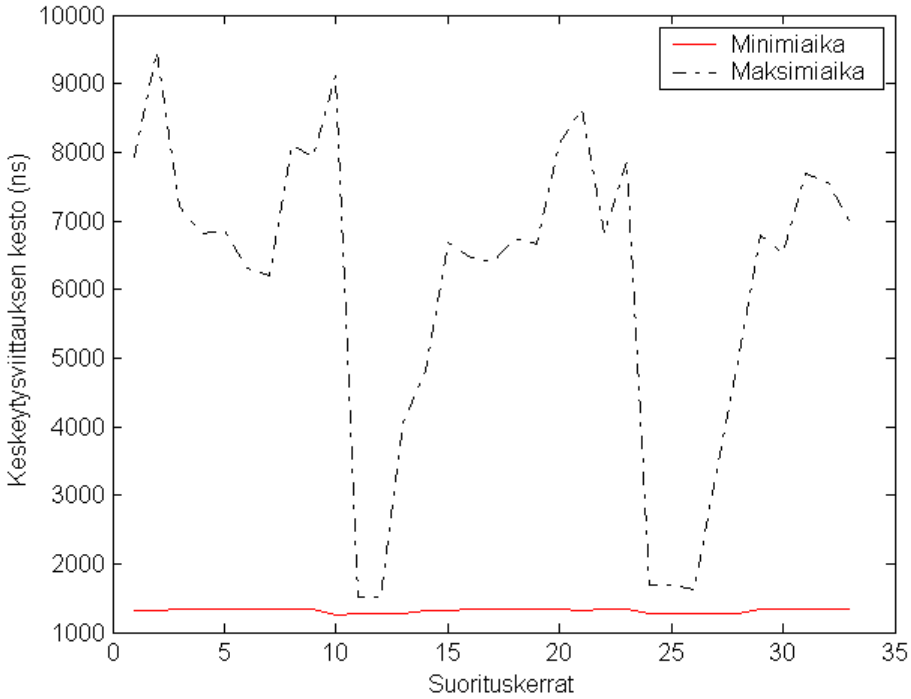


Kuva 12. *Ajoitustarkkuus.*

Kuten huomataan, jokaisella suorituskerralla pienin virhe on ollut nanosekunnin tarkkuudella nolla. Suurin virhe on aina pysynyt alle 8000 nanosekunnin eli 8 mikrosekunnin. Mekatronisten koneiden reaaliaikaisessa simuloinnissa tämän kokoluokan virhe on täysin merkityksetön. Ajoitustarkkuus on riittävän hyvä työssä käsiteltävään sovellukseen.

4.1.2 Keskeytysviittaukset

Toinen testi, jolla tutkittiin reaaliaikaisuutta, on keskeytysviittausten nopeus. Testaukseen käytetyn ohjelman C-kielinen lähdekoodi on esitetty liitteessä 3. Kuvassa 13 on esitetty maksimi- ja minimiaika viittauksesta sen perille menoon 35 testauskerralla.



Kuva 13. *Keskeytysviittauksen kesto.*

Keskeytysviittausten kestoja tarkkailemalla huomataan, että laitteiston asettama raja on hiukan yli 1000 nanosekuntia. Tätä nopeammin viitausta ei pystytä toimittamaan missään vaiheessa. Suurimmillaan kesto on kaikissa tapauksissa alle 10 mikrosekuntia. Reaaliaikaisessa simuloinnissa käyttäjä ei havaitse tämän kokoluokan viivettä toiminnassa joten keskeytysviittaukset ovat riittävän nopeita.

4.1.3 Laskentateho

Simulointimallissa voidaan määrittää aika-askel erikseen numeriselle integroinnille ja reaaliaikaiselle suoritukselle. Näin mallia voidaan suorittaa myös nopeammin tai hitaammin kuin reaaliajassa. Tuloksiin tämä ei kuitenkaan vaikuta, jos integraattoreiden aika-askelta ei muuteta.

Laskentatehoa testattiin niin, että hydraulikäyttöisen puomin simulointimallin reaaliaika-ajastimen aika-askelta pienennettiin kunnes suoritus ei enää onnistunut. Pienin aika-askel, jolla suoritus onnistui, oli työssä muutenkin käytetty 0,001 s. Tätä pienemmällä aika-askelella laskentateho ei riitä ja tietokone jumiutuu hetkeksi reaaliaikaprosessin viedessä kaiken prosessoriajan. Aivan tarkasti pienintä mahdollista aika-askelta ei tällä menetelmällä pysty määrittämään. Voidaan kuitenkin todeta, että nyt esitellyssä sovelluksessa laskentateho riittää simulointimallin suorittamiseen reaaliaikaisena riittävän pienellä aika-askelella. Tarvittaessa hydrauliikan ja dynamiikan integrointiaskelet voivat olla myös eri suuruiset. Näin laskentaa voidaan optimoida paremmin täyttämään tarkkuusvaatimukset.

4.1.4 Reaaliaikaisuus

Reaaliaikaisuuden mittaamiseen ei ole olemassa testiohjelmaa, koska testiohjelman pitäisi ottaa vertailuaika samasta tietokoneesta jolla simulointimallia suoritetaan. Nyt tyydyttiin tarkkailemaan simuloinnin suoritusta ulkopuolisen kellon avulla. Simuloinnin havaittiin silmämääräisesti tarkkaillen toimivan oikea-aikaisesti.

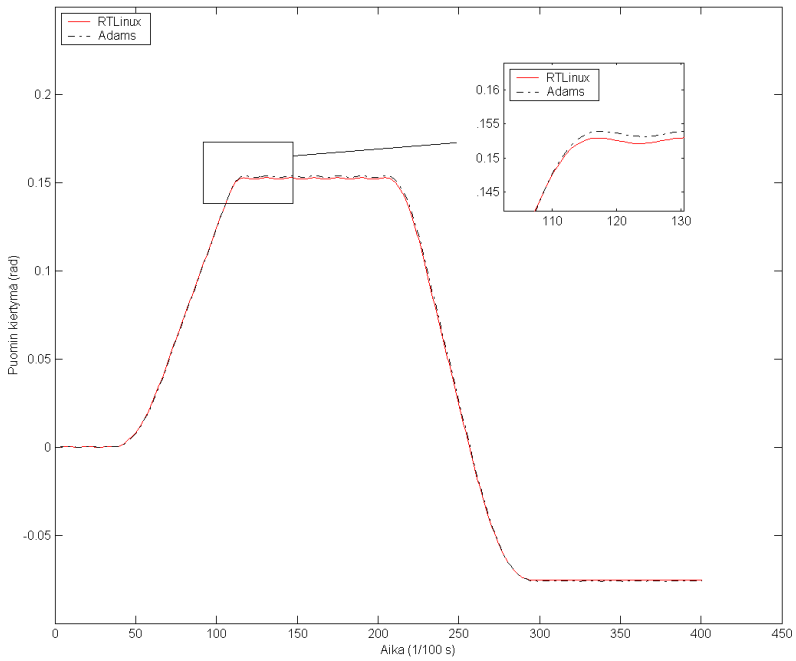
4.2 Hydraulikäyttöisen puomin simulointi

4.2.1 Simuloinnin kesto

RTLinux-simulointimallin ratkaisukykyä hydraulikäyttöisen puomin simuloinnissa tarkasteltiin jo kohdassa 4.1.3. Adams-mallin työkierron laskemiseen meni noin 20 sekuntia kun käytettiin 0,0005 sekunnin aika-askelta. Laskenta-aika on mitattu silmämääräisesti ulkopuolisesta kellosta. Pitkä laskenta-aika johtuu käytetystä muuttuva-askelisesta integrattorista, joka pienentää aika-askelta tarvittaessa riittävän tarkkuuden saavuttamiseksi.

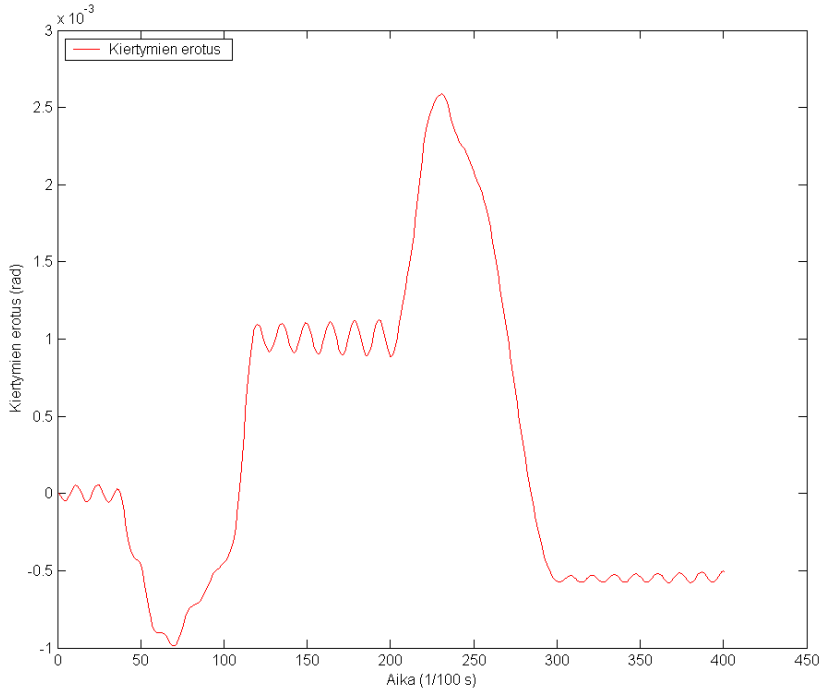
4.2.2 Puomin kiertymä

Mekanismin liikettä tarkasteltiin puomin kiertymän avulla. Kuvassa 14 on esitetty RTLinuxilla ja Adamsilla saadut kiertymät ajan funktiona. Kuvasta huomataan, että RTLinuxin reaaliaikamallin ja Adams-mallin tulokset ovat käytännössä erittäin lähellä toisiaan. Osasuurennoksesta on kiertymä vielä tarkempana suurennoksena noin yhden sekunnin kohdalta, kun puomi on ylimmässä asennossaan. Ominaisvärähtely tapahtuu samalla taajuudella ja samassa vaiheessa.



Kuva 14. *Puomin kiertymä ajan funktiona.*

Kuvassa 15 on esitetty reaaliaikaisen simulointimallista ja Adams-mallista saatujen kiertymien erotus työkierron aikana. Reaaliaikamallin antamasta kiertymästä on vähennetty Adamsin antama. Kuten huomataan, ero on suurimmillaan noin 2,5 tuhannesosaradiaania. Ero mallien välillä muuttuu lähinnä puomin liikkeiden aikana. Puomin pysyessä paikallaan erotus pysyy pientä värähtelyä lukuunottamatta paikallaan. Erotuksen värähtely johtuu simulointimallien toimimisesta hiukan eri vaiheessa. Huomataan myös, että tällöin erotuksen taso on verrannollinen puomin kiertymään.

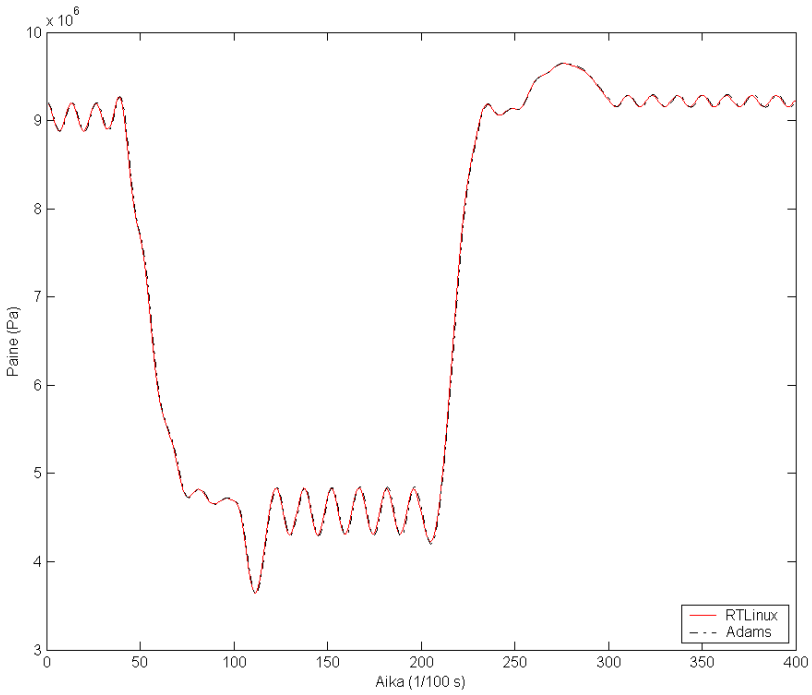


Kuva 15. *Puomin kiertymien erotus.*

4.2.3 Hydraulijärjestelmän paineet

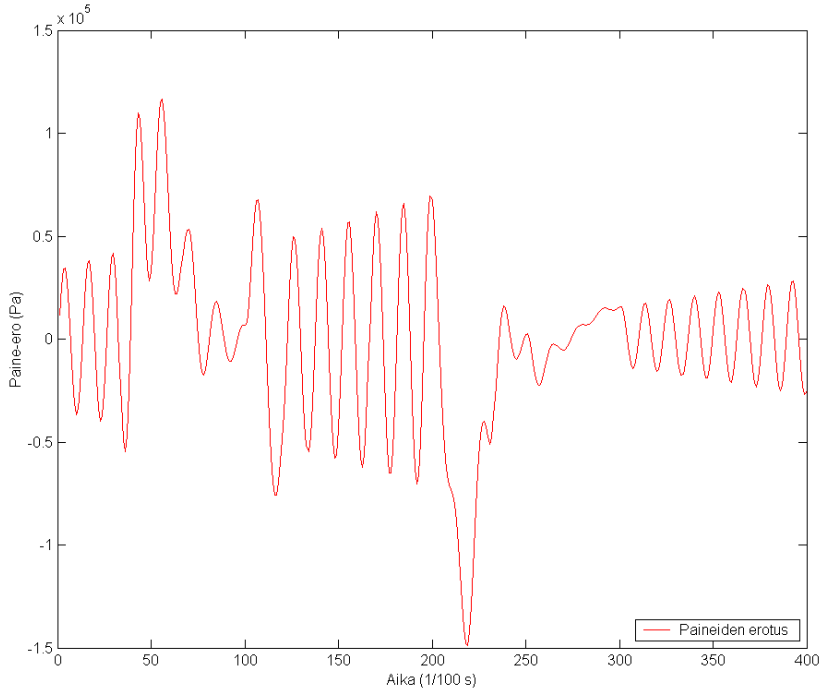
Kuvassa 16 on esitetty simulointimalleista ratkaistut hydraulipaineet keskittyneiden paineiden teorian mukaisessa tilavuudessa A. Kuvassa 17 on esitetty simulointimallien paineiden välinen erotus. Adams-mallin tuloksesta on vähennetty reaaliaikamallin tulos.

Kuten huomataan, hydraulipaineet ovat lähellä toisiaan koko simuloinnin ajan. Tämä on luonnollista, koska hydraulipiiri on mallinnettu samalla tavalla kummassakin simulointimallissa.



Kuva 16. *Paine tilavuudessa A.*

Kuvan 17 hydraulipaineiden erotusta tarkastelemalla huomataan, että paineiden välinen ero on suurimmillaan noin 150000 *Pa*. Erotuksen taso pysyy samana huolimatta paineen suuruudesta. Erotus johtuu pääasiassa vaihe-erosta simulointimallien välillä.



Kuva 17. *Paineiden erotus tilavuudessa A.*

Vaihe-ero mallien välillä ei vaikuta tulosten oikeellisuuteen. Kuten kuvasta 16 huomataan, tulokset ovat kuitenkin käytännössä identtiset. Simulointimallien välinen ero on merkityksetön tarkastellussa sovelluksessa.

5 JOHTOPÄÄTÖKSET

Työssä tutkittiin Linux-ympäristön käyttömahdollisuuksia mekatronisten koneiden reaaliaikaisessa simuloinnissa vaihtoehtona kaupallisille ratkaisuille. Linux-käyttöjärjestelmään asennettiin RTLinux-reaaliaikalaajennus. RTLinuxin kykyä toteuttaa kovaa reaaliaikaisuutta testattiin apuohjelmien avulla. Lisäksi ratkaistiin reaaliaikaisesti hydraulikäyttöisestä puomimekanismista tehtyä C-kielistä simulointimallia ja verrattiin saatuja tuloksia Adams-ohjelmistolla tehtyyn vastaavaan malliin.

Reaaliaikajärjestelmän asentaminen huomattiin melko monimutkaiseksi ja –vaiheiseksi prosessiksi. Linux-käyttöjärjestelmän laite- ja ohjelmistotuki on osin puutteellista. RTLinux-laajennuksella on myös omat vaatimuksensa sekä laitteistolle että ohjelmistolle. Asennusdokumentaatio on sekavaa ja hajanaista. Monet erilaiset ohjelmistojen versiot aiheuttivat ongelmia asennusvaiheessa. Asennuksessa vaaditaan Linux-tuntemusta.

Reaaliaikaista Linux-ympäristöä ei käytetä yleisesti mekatronisten koneiden simulointiin. Näin ollen valmiita mallinnustyökaluja ei ollut olemassa. Simulointimalli tehtiin käsin ohjelmoimalla muokkaamalla Pasi Korkealaakson diplomityössään (lähde 17) luoman simulointimallin rakenne RTLinuxille sopivaksi, lisäämällä reaaliaikakäskyt ja muuttamalla ohjaus- ja tulostustoimintoja. Simulointimallin teko RTLinuxille ei eroa tavallisesta C-kielellä tehtävästä ohjelmoinnista kuin muutamien reaaliaikaisuuteen liittyvien komentojen osalta. Koska ohjelma on suoraan yhteydessä laitteistoon aiheuttaa pienikin virhe koko tietokoneen kaatumisen. Tästä syystä ohjelman kehitys on aikaa vievää.

Simulointimallissa mekaniikka mallinnettiin tasossa ja kappaleet oletettiin jäykiksi. Hydraulipiiri mallinnettiin keskittyneiden paineiden menetelmällä. Kun simulointimalli on luotu käsin, siitä saadaan numeerisesti tehokas, koska mukana on vain tarvittavat

toiminnot. Tämä havaittiin vertaamalla saatuja tuloksia Adams-ohjelmistolla tehdystä simulointimallista saatuihin tuloksiin ja laskenta-aikoihin.

RTLinuxin avulla simulointimalli saatiin toimimaan reaaliajassa ja aika-askelesta riippuen nopeamminkin tavallisella, hiukan vanhentuneella PC-tietokoneella. Adamsilla puolestaan mallin laskemiseen kului 2-4 –kertainen aika simuloinnin pituuteen verrattuna. Tässä tulee ilmi paitsi reaaliaikaisen mallin numeerinen tehokkuus, myös RTLinuxin tehokas kyky priorisoida reaaliaikaiset prosessit muiden edelle.

Koska työssä saadut tulokset simulointimallien välillä ovat hyvin samanlaiset, voidaan todeta, että käytetyillä aika-askelilla reaaliaikamallin laskentatarkkuus on riittävä. Pienempi aika-askel ei tuo enää olennaista muutosta tuloksiin. Näinollen RTLinuxia käyttämällä Pentium II 450 MHz –prosessorin teho riittää tarkastellun koneen reaaliaikaiseen simulointiin.

RTLinux-ympäristön havaittiin olevan hyvä ja edullinen vaihtoehto kaupallisille ympäristöille. Erilaisissa simulaattoreissa ja muissa sovelluksissa, joissa simulointimallia ei ole tarvetta muokata usein sekä lujaa reaaliaikaisuutta vaativissa suunnittelutehtävissä RTLinux on vakavasti otettava vaihtoehto.

Reaaliaikaista simulointiympäristöä voidaan kehittää lisäämällä I/O-toiminnot. Tämän voi tehdä käyttämällä hyväksi joko tietokoneen omia I/O-portteja, kuten COM-porttia, tai erillistä I/O-korttia. Simulointimalliin voidaan myös liittää visualisointiohjelma, joka lisäisi havainnolisuutta huomattavasti. Kolmiulotteisten ja/tai joustavien mekaniikkamallien käyttöön RTLinuxissa ei ole esteitä. Simulointimallien monipuolisuuden suhteen ainoa rajoitus on tietokoneen prosessoriteho. Mekatronisten koneiden mallintamiseen voidaan kehittää työkaluja, jolloin simulointimallia ei välttämättä tarvitsisi ohjelmoida suoraan C-kielellä.

LÄHTEET

/1/ Henricson, J. Real-Time Linux. Automaation tietotekniikan seminaari. 2001. TKK. 16 s.

/2/ Anon. dSPACE [verkkosivu]. [viitattu 11.11.2002]. Saatavissa: http://www.dspace.de

/3/ Anon. Opal-RT [verkkosivu]. [viitattu 11.11.2002]. Saatavissa: http://www.opal-rt.com

/4/ Anon. Venturcom [verkkosivu]. [viitattu 11.11.2002]. Saatavissa: http://www.vci.com

/5/ Anon. Lumeo Motion [verkkosivu]. [viitattu 11.11.2002]. Saatavissa: http://www.lumeo.com

/6/ Hilton, E. Getting Started with RTLinux [verkkosivu]. [viitattu 10.7.2002]. Saatavissa: http://fsmlabs.com/developers/docs/txt/GettingStarted.txt

/7/ Knox, B. Realtimelinux.org [verkkosivu]. [viitattu 1.7.2002]. Saatavissa: http://www.realtimelinux.org/links.html

/8/ RTAI Development Team. DIAPM RTAI [verkkosivu]. [viitattu 12.11.2002]. Saatavissa: http://www.aero.polimi.it/~rtai/

/9/ Dinkel, W., Niehaus, D., Michael, F., Woltersdorf, J. KURT-Linux User Manual. University of Kansas. 2002. 54 s

/10/ Wang, Y., Lin, K. Implementating a General Real-Time Scheduling Framework in RED-Linux. University of California. 1999. 10 s.

/11/ Rajkumar, R., Juvva, K., Molano, A., Oikawa, S. Resource Kernels: A Resource Centric Approach to Real-Time and Multimedia Systems. Carnegie Mellon University. 1998. 18 s.

/12/ Anon. Timesys Linux GPL [verkkosivu]. [viitattu 11.11.2002]. Saatavissa: http://www.timesys.com/index.cfm?hdr=linux_header.cfm&bdy=linux_bdy.cfm

/13/ Barabanov, M., Yodaiken, V. Real-Time Linux. New Mexico Institute of Minig and Technology. 1996. 9 s.

/14/ Mikkola, A. Mekatronisen koneen simulointi. Luentomoniste. LTKK Lappeenranta. 2000. 96 s.

/15/ Råde, L., Westergren, B. Mathematics Handbook for Science and Engineering. 3. painos. Ruotsi. 1995. Studentlitteratur. 539 s.

/16/ Mäkinen, R. Numeeriset menetelmät. Luentomoniste 1. Jyväskylä. 1999. Jyväskylän yliopisto. 107 s.

/17/ Korkealaakso, P. Reaaliaikainen monikappaledynamiikan simulointi. Diplomityö. LTKK Lappeenranta. 2002. 62 s.

LIITTEET

Liite 1. RTLinuxin asennus

Liite 2. Lähdekoodi ohjelmaan, jolla mitataan ajoitustarkkuutta

Liite 3. Lähdekoodi ohjelmaan, jolla mitataan keskeytysviittausten nopeutta

Liite 1. RTLinuxin asennus.

1. Perus-Linuxin asennus

- Mandrake 8.2
- patch asennettava
- ei ext3-osioita
- graafinen käyttöliittymä KDE asennetaan

2. Tarvittavat tiedostot internetistä

- Ydin osoitteesta http://ftp.kernel.org, joko 2.2.19 ta 2.4.4
- RTLinux 3.1 osoitteesta http://www.fsmlabs.com
- RTLinux kannattaa polttaa cdrom-levylle

3. Root-oikeudet

Root-oikeudet saa komennolla:

```
su
```

4. Linux Kernel Makefile -muutos

```
emacs /usr/src/linux/Makefile
```

Muuta kohta:

```
CC              = $(CROSS_COMPILE)gcc
```

Muotoon:

```
CC              = kgcc
```

5. RTLinux-cd sisään

CD:n hakemisto on /mnt/cdrom, Mandraken asennukseen sisältyy ohjelma, joka osaa automaattisesti asentaa (mount) sisäänlaitetun CD:n.

6. Puretaan kernel

Poistetaan mahdolliset vanhat versiot:

```
rm -rf /usr/src/rtlinux
```

(jatkuu)

Tehdään hakemisto rtlinuxille ja mennään sinne:

```
mkdir /usr/src/rtlinux
cd /usr/src/rtlinux
```

Puretaan kernel:

```
tar xzf /mnt/cdrom/Linux/linux-2.4.4.tar.gz
```

Nyt rtlinux-hakemistoon ilmestyy linux-niminen hakemisto.

7. Puretaan RTLinux

Tapahtuu komennolla:

```
tar xzf /mnt/cdrom/Linux/rtlinux-3.1.tar.gz
```

rtlinux-hakemistoon ilmestyy rtlinux-3.1-niminen hakemisto.

8. Patchataan kernel

Tässä vaiheessa pitää patch-paketti olla asennettuna.

```
cd linux
patch -p1 < ../rtlinux-3.1/kernel_patch-2.4.4
```

Pitäisi tulla näkyviin lista seuraavantyyppisiä ilmoituksia:

```
patching file scripts/mkdep.c
```

9. Ytimen asetukset

Tähän on kolme erilaista käyttöliittymää. Ihan puhdas rivi riviltä –liittymä aukeaa komennolla:

```
make config
```

Menupohjainen liittymä:

```
make menuconfig
```

Graafinen liittymä:

```
make xconfig
```

Kaikilla saa tehtyä samat asiat. Graafinen liittymä on varmaankin helppokäyttöisin mutta käyttäminen vaatii X-Windowsin. Seuraavat asetukset ovat tärkeimmät muistaa:

- Prosessorityyppi oikein
- APM-tuki pois päältä, saattaa häiritä reaaliaikaisuutta

Muita asetuksia:

- code maturity y
- PCMCIA n
- SCSI n
- Verkkokorttiasetukset
- Piirisarja ja näytönohjain
- merkistöt
- äänikortti n
- USB n

10. Ytimen ja modulien kääntäminen

Tapahtuu käskyillä:

```
make bzImage
make modules
```

Ensimmäisessä menee aika kauan. Ruudulla näkyy melkoinen lista varoituksia jotka eivät ole vaarallisia.

Asennetaan Linux-kernel ja modulit:

```
make modules_install
```

11. Koneen käynnistys RTLinuxiin

Kopioidaan tiedosto:

```
cp arch/i386/boot/bzImage /boot/rtzImage
```

Laitetaan LILO:on vaihtoehto rtlinux seuraavasti. Avataan asetustiedosto:

```
emacs /etc/lilo.conf
```

Lisätään seuraavat rivit normaalin Linuxin jälkeen:

```
image=/boot/rtzImage
        label=rtlinux
        root=/dev/hda1
        append="devfs=mount"
        read-only
```

Asennetaan LILO:

```
/sbin/lilo
```

Käynnistetään kone uudestaan:

```
/sbin/shutdown –r now
```

Valitaan rtlinux valikosta.

12. RTLinuxin asetukset ja kääntäminen

Tämä tapahtuu samalla tavalla, kuin edellä:

`make config` tai `make menuconfig` tai `make xconfig`

Kääntäminen:

```
make
make Devices
```

12. Reaaliaikamodulien käynnistys

Reaaliaikamodulit käynnistetään seuraavasti:

```
cd /usr/src/rtlinux/rtlinux-3.1
scripts/insrtl
```

Liite 2. Lähdekoodi ohjelmaan, jolla mitataan ajoitustarkkuutta.

```
/*
 * RTLinux scheduling accuracy measuring example
 *
 * (C) Michael Barabanov, 1997
 *  (C) FSMLabs  1999. baraban@fsmlabs.com
 *  Released under the GNU GENERAL PUBLIC LICENSE Version 2, June 1991
 *  Any use of this code must include this notice.
 */

#include <rtl.h>
#include <rtl_fifo.h>
#include <time.h>
#include <rtl_sched.h>
#include <rtl_sync.h>
#include <pthread.h>
#include <unistd.h>
#include <rtl_debug.h>
#include <errno.h>
#include "common.h"

int ntests=500;
int period=1000000;
int bperiod=3100000;
int mode=0;
int absolute=0;
int fifo_size=4000;
int advance=0;

MODULE_PARM(period,"i");
MODULE_PARM(bperiod,"i");
MODULE_PARM(ntests,"i");
MODULE_PARM(mode,"i");
MODULE_PARM(absolute,"i");
MODULE_PARM(advance,"i");

pthread_t thread;
int fd_fifo;


void *thread_code(void *param)
{
        hrtime_t expected;
        hrtime_t diff;
        hrtime_t now;
        hrtime_t last_time = 0;
        hrtime_t min_diff;
        hrtime_t max_diff;
        struct sample samp;
        int i;
        int cnt = 0;
        int cpu_id = rtl_getcpuid();

        rtl_printf ("Measurement task starts on CPU %d\n", cpu_id);
        if (mode) {
                int  ret  =  rtl_setclockmode  (CLOCK_REALTIME,  RTL_CLOCK_MODE_PERIODIC,
period);

                if (ret != 0) {
                        conpr("Setting periodic mode failed\n");
                        mode = 0;
                }
        } else {

                rtl_setclockmode (CLOCK_REALTIME, RTL_CLOCK_MODE_ONESHOT, 0);
        }
```

(jatkuu)

```
        expected = clock_gethrtime(CLOCK_REALTIME) + 2 * (hrtime_t) period;


        fd_fifo = open("/dev/rtf0", O_NONBLOCK);
        if (fd_fifo < 0) {
                rtl_printf("/dev/rtf0 open returned %d\n", fd_fifo);
                return (void *) -1;
        }

        if (advance) {
                rtl_stop_interrupts();  /* Be careful with this! The task won't be
preempted by anything else. This is probably only appropriate for small high-priority
tasks. */
        }

        /* first cycle */
        clock_nanosleep  (CLOCK_REALTIME,  TIMER_ABSTIME,  hrt2ts(expected  -  advance),
NULL);
        expected += period;
        now = clock_gethrtime(CLOCK_MONOTONIC);
        last_time = now;

        do {
                min_diff = 2000000000;
                max_diff = -2000000000;

                for (i = 0; i < ntests; i++) {
                        ++cnt;
                        clock_nanosleep (CLOCK_REALTIME, TIMER_ABSTIME, hrt2ts(expected -
advance), NULL);

                        now = clock_gethrtime(CLOCK_MONOTONIC);
                        if (absolute && advance && !mode) {
                                if (now < expected) {
                                        rtl_delay (expected - now);
                                }
                                now = clock_gethrtime(CLOCK_MONOTONIC);
                        }
                        if (absolute) {
                                diff = now - expected;
                        } else {
                                diff = now - last_time - period;
                                if (diff < 0) {
                                        diff = -diff;
                                }
                        }
                        if (diff < min_diff) {
                                min_diff = diff;
                        }
                        if (diff > max_diff) {
                                max_diff = diff;
                        }

                        expected += period;
                        last_time = now;
                }

                samp.min = min_diff;
                samp.max = max_diff;
                write (fd_fifo, &samp, sizeof(samp));
        } while (1);
        return 0;
}

pthread_t background_threadid;

void *background_thread(void *param)
{
        hrtime_t next = clock_gethrtime(CLOCK_REALTIME);
```

```
        while (1) {
                hrtime_t t = gethrtime ();
                next += bperiod;
                /* the measurement task should preempt the following loop */
                while (gethrtime() < t + bperiod * 2 / 3);
                clock_nanosleep (CLOCK_REALTIME, TIMER_ABSTIME, hrt2ts(next), NULL);
        }
}


int init_module(void)
{
        pthread_attr_t attr;
        struct sched_param sched_param;
        int thread_status;
         int fifo_status;

        rtf_destroy(0);
        fifo_status = rtf_create(0, fifo_size);
        if (fifo_status) {
                rtl_printf("RTLinux           measurement           test           fail.
fifo_status=%d\n",fifo_status);
                return -1;
        }


        rtl_printf("RTLinux measurement module on CPU %d\n",rtl_getcpuid());
        pthread_attr_init (&attr);
        if (rtl_cpu_exists(1)) {
                pthread_attr_setcpu_np(&attr, 1);
        }
        sched_param.sched_priority = 1;
        pthread_attr_setschedparam (&attr, &sched_param);
        rtl_printf("About to thread create\n");
        thread_status = pthread_create (&thread,  &attr, thread_code, (void *)1);
        if (thread_status != 0) {
                rtl_printf("failed to create RT-thread: %d\n", thread_status);
                return -1;
        } else {
                rtl_printf("created RT-thread\n");
        }

        if (bperiod) {
                pthread_create (&background_threadid,  NULL, background_thread, NULL);
        }
        return 0;
}


void cleanup_module(void)
{
         rtl_printf ("Removing module on CPU %d\n", rtl_getcpuid());
        pthread_cancel (thread);
        pthread_join (thread, NULL);
        close(fd_fifo);
        rtf_destroy(0);
        if (bperiod) {
                pthread_cancel (background_threadid);
                pthread_join (background_threadid, NULL);
        }
}
```

Liite 3. Lähdekoodi ohjelmaan, jolla mitataan keskeytysviittausten nopeutta.

```
/*
 * RTLinux irq test
 *
 * measures semaphore wakeup operation performance
 *
 *  (C) FSMLabs  2000. Michael Barabanov <baraban@fsmlabs.com>
 *  Released under the GNU GENERAL PUBLIC LICENSE Version 2, June 1991
 *  Any use of this code must include this notice.
 */

#include <rtl.h>
#include <rtl_fifo.h>
#include <time.h>
#include <rtl_sched.h>
#include <rtl_sync.h>
#include <pthread.h>
#include <unistd.h>
#include <rtl_debug.h>
#include <errno.h>
#include <semaphore.h>
#include "common.h"

int irq=14;
int ntests=100;
int setfocus=0; /* 0 or 1 */

MODULE_PARM(irq,"i");
MODULE_PARM(ntests,"i");
MODULE_PARM(setfocus,"i");

pthread_t thread;
int fd_fifo;
hrtime_t irqtime;
sem_t irqsem;

unsigned int intr_handler(unsigned int irq, struct pt_regs *regs)
{
/*      rtl_printf("%d", rtl_getcpuid()); */
        irqtime = gethrtime();
        rtl_global_pend_irq (irq);
        sem_post(&irqsem);
        return 0;
}


void *thread_code(void *param) {

        hrtime_t diff;
        hrtime_t now;
        hrtime_t min_diff;
        hrtime_t max_diff;
        struct sample samp;
        int i;
        int cpu_id = rtl_getcpuid();

        rtl_printf ("Measurement task starts on CPU %d\n", cpu_id);

        fd_fifo = open("/dev/rtf0", O_NONBLOCK);
        if (fd_fifo < 0) {
                rtl_printf("/dev/rtf0 open returned %d\n", fd_fifo);
                return (void *) -1;
        }

        do {
                min_diff = 2000000000;
                max_diff = -2000000000;
```

(jatkuu)

```
                for (i = 0; i < ntests; i++) {
                        sem_wait (&irqsem);
                        now = gethrtime();

                        diff = now - irqtime;
                        if (diff < min_diff) {
                                min_diff = diff;
                        }
                        if (diff > max_diff) {
                                max_diff = diff;
                        }
                }

                samp.min = min_diff;
                samp.max = max_diff;
                write (fd_fifo, &samp, sizeof(samp));
        } while (1);
        return 0;
}


unsigned long oldaffinity;

int init_module(void)
{
        int ret;
        unsigned long affinity = 1 << rtl_getcpuid();

        rtf_destroy(0);
        rtf_create(0, 4000);

        ret = rtl_request_irq (irq, intr_handler);
        if (ret) {
                rtl_printf("failed to get irq%d: %d\n", irq, ret);
        } else {
                rtl_printf("got irq%d\n", irq);
        }

        if (setfocus) {
                rtl_printf("setting irq%d distribution mask to %x\n", irq, affinity);
                rtl_irq_set_affinity (irq, &affinity, &oldaffinity);
        }
        sem_init (&irqsem, 1, 0);
        rtl_printf("RTLinux measurement module on CPU %d\n",rtl_getcpuid());
        pthread_create (&thread,  NULL, thread_code, (void *)1);

        return 0;
}


void cleanup_module(void)
{
         rtl_printf ("Removing module on CPU %d\n", rtl_getcpuid());

        rtl_free_irq (irq);
        if (setfocus) {
                rtl_irq_set_affinity (irq, &oldaffinity, NULL);
        }
        rtl_printf("freed irq%d\n", irq);

        pthread_cancel (thread);
        pthread_join (thread, NULL);
        sem_destroy (&irqsem);
        close(fd_fifo);
        rtf_destroy(0);
}
```