LUT-YLIOPISTO
LUT School of Energy Systems
LUT Kone
BK10A0402 Kandidaatintyö

VIRTUAALISEN TYÖKONEEN OHJAUS CODESYS-OHJAIMELLA

CONTROL OF VIRTUAL WORKING MACHINE BY USING CODESYS CONTROLLER

Lappeenrannassa 27.8.2019
Niko Montén
Tarkastaja TkT Kimmo Kerkkänen
Ohjaajat TkT Lauri Luostarinen, TkT Kimmo Kerkkänen

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
LUT Energiajärjestelmät
LUT Kone

Niko Montén

Virtuaalisen työkoneen ohjaus Codesys-ohjaimella

Kandidaatintyö

2019

36 sivua ja 9 kuvaa

Tarkastaja: TkT Kimmo Kerkkänen

Ohjaajat: TkT Lauri Luostarinen
TkT Kimmo Kerkkänen

Hakusanat: työkoneen simulointi, virtuaalinen ohjaus, virtuaalinen testaus, työkoneiden ohjausjärjestelmät, reaaliaikainen simulaatio, hydrauliikan ohjaus, Codesys ohjaus

Tässä kandidaatintyössä päätavoitteena oli löytää yhteysratkaisu reaaliaikaiseen tiedonsiirtoon MeVEAn ja Codesyksen välille. Tämä mahdollisti MeVEAn simulaatiomallin ohjauksen Codesyksen monipuolisen visuaalisen käyttöliittymän ohjainten avulla reaaliaikaisesti. Lisäksi tehtiin yleinen kirjallisuuskatsaus työkoneiden voimansiirrosta, ohjausjärjestelmistä sekä näiden simulaatiosta. Näin luotiin yleiskuva tarvittavasta tiedosta työkoneiden ja niiden ohjausjärjestelmien simulaatioon liittyen.

Reaaliaikainen tiedonsiirto onnistui MeVEAn ja Codesyksen välillä käyttäen C++-ohjelmaa, joka prosessoi ja välitti datan eteenpäin MeVEAlta Codesykselle ja päinvastoin. Näin tiedonsiirto onnistuttiin toteuttamaan reaaliaikaisesti molempiin suuntiin ja suoritettiin testiajo, jossa MeVEAn työkoneen simulaatiomallia ohjataan Codesyksen visuaalisen käyttöliittymän avulla reaaliaikaisesti. Tiedonsiirto Codesyksen ja C++-ohjelman välillä toteutettiin jaetun muistin menetelmällä, ja tiedonsiirto MeVEAn ja C++-ohjelman välillä toteutettiin MeVEAn tarjoaman Socket-yhteysratkaisun avulla. Muodostettua yhteysratkaisua on helppo muokata käytettäväksi monipuolisesti minkä tahansa MeVEAn simulaatiomallin ohjaukseen Codesyksestä käsin.

Dataa onnistuttiin myös lähettämään UDP-yhteydellä Codesykseltä C++-ohjelmalle, muttei vastakkaiseen suuntaan. MeVEAn simulaatiomallin ohjaus toimisi myös käyttäen UDP-yhteyttä, mutta vain avoimen silmukan kaltaisesti, koska tiedonsiirtoa ei onnistuttu lähettämään takaisin C++-ohjelmalta Codesykselle.

ABSTRACT

Lappeenranta University of Technology
LUT School of Energy Systems
LUT Mechanical Engineering

Niko Montén

Control of virtual working machine by using Codesys controller

Bachelor's thesis

2019

36 pages and 9 figures

Examiner: D. Sc. (Tech.) Kimmo Kerkkänen

Supervisors: D. Sc. (Tech.) Lauri Luostarinen, D. Sc. (Tech.) Kimmo Kerkkänen

Keywords: simulation of working machine, virtual control, virtual testing, control systems of working machines, real-time simulation, control of hydraulics, Codesys control

The primary objective of this bachelor's thesis was to find a solution for real-time communication between MeVEA and Codesys. This allowed the control of MeVEA simulation model through the versatile visualization interface of Codesys in real-time. In addition, a general literature review was conducted about power transmission and control systems in working machines, as well as simulations of them. This created a generalized picture of the needed knowledge for simulation of working machines and their control systems.

The real-time communication between MeVEA and Codesys was achieved by using a C++ program, which processed and forwarded the data from MeVEA to Codesys and vice versa. This way, the communication was achieved towards both directions and a test run was executed, where the simulation model from MeVEA was controlled through visualization interface in Codesys in real-time. The communication between Codesys and C++ program was done by using shared memory, and the communication between MeVEA and C++ program was carried out by a Socket communication method provided by MeVEA. The found communication solution can be easily modified to diversely control of any MeVEA simulation model through Codesys.

The data was also successfully transmitted from Codesys to C++ program over UDP connection, but not in the opposite direction. This would also allow the control of MeVEA simulation model from Codesys but only in an open loop way, because the data transmission from C++ program to Codesys was not successful.

SISÄLLYSLUETTELO

SYMBOLI- JA LYHENNELUETTELO

A	State-Space-mallin tilamatriisi
B	State-Space-mallin sisääntulomatriisi
b	Kyseiseen massaan kohdistuvan vaimennuskerroin
C	State-Space-mallin ulostulomatriisi
C_1	Sylinterikammio 1:en menevä virtaus
C_2	Sylinterikammio 2:en menevä virtaus
D	State-Space-mallin läpivientimatriisi (feedthrough matrix)
d	Ulkoisten häiriöiden vaikutus lohkokaavioon
e	Lohkokaavion erotussignaali
K	Säädin lohkokaaviossa
K_D	PID-säätimen derivoivan osan suuruus
K_I	PID-säätimen integroivan osan suuruus
K_P	PID-säätimen vahvistuksen suuruus
K(s)	PID-säätimen siirtofunktio Laplace-muunnoksella
n	Mittauskohinan vaikutus lohkokaavioon
P	Prosessi/koneisto lohkokaaviossa
q	State-Space-mallin tilavektori
R	Virtaus tankkiin
r	Lohkokaavion viitearvo/sisääntuloarvo
S	Syöttöpaine
s	Siirtofunktion kompleksimuuttuja
T*(s)*	Siirtofunktio
U*(s)*	Siirtofunktion sisääntulon Laplace-muunnos
u	State-Space-mallin sisääntulovektori
u	Ohjaussignaali lohkokaaviossa
x_{valve}	Venttiilin karan liikesuunta
Y*(s)*	Siirtofunktion ulostulon Laplace-muunnos
y_0	Lohkokaavion ulostuloarvo
y	State-Space-mallin ulostulovektori
y	Mittausarvo lohkokaaviossa takaisinkytketyssä ohjausjärjestelmässä

CAN	Controller Area Network
DUT	Data Unit Type
GVL	Global Variable List
NVL	Network Variable List
PID	Proportional-Integral-Derivative
PLC	Programmable Logic Controller
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

1 JOHDANTO

Tämä työ jakaantuu kirjallisuuskatsaukseen ja käytännön toteutukseen, jossa reaaliaikaisesti simuloitua työkonetta ohjataan Codesyksen graafisesta käyttöliittymästä. Kirjallisuuskatsauksessa, kappaleessa 2, tutkitaan työkoneita, niiden voimansiirtoa, ohjausjärjestelmiä ja simulaatioita. Koska työkoneissa useimmiten voimansiirtona käytetään hydrauliikkaa, myös kirjallisuuskatsauksessa tutkitaan työkoneiden hydrauliikkaa sekä sen toimintaperiaatteita. Tavoitteena on luoda yleinen kuva tarvittavasta tiedosta, jota tarvitaan työkoneiden ja niiden ohjausjärjestelmien simulaatioon.

Käytännön toteutuksessa, kappaleessa 3 ja 4, etsittiin ratkaisua itse työn pääongelmaan. Pääongelmana oli löytää menetelmä, jossa MeVEA-ohjelman reaaliaikaista simulaatiomallia voitaisi ohjata Codesys-ohjelmalla. Pyrittiin yhdistämään Codesyksen hyvät virtuaaliset ohjausmahdollisuudet MeVEAn simulaattoriin. Ongelmana on siis löytää toimiva ratkaisu reaaliaikaiseen tiedonsiirtoon MeVEAn ja Codesyksen välille. Codesyksen puolelta olisi helpompaa toteuttaa simulaatiomallin reaaliaikainen ohjaus ja mahdollisesti voitaisiin luoda simulaatiomallille monimutkainenkin ohjausjärjestelmä. Tällöin voitaisiin simulaatiolla testata työkoneen dynamiikan lisäksi koko ohjausjärjestelmää osana työkonetta. Alla oleva kuva 1 havainnollistaa tavoiteltavaa ratkaisua.

Kuva 1. MeVEAn ja Codesyksen yhteysratkaisu.

Codesyksen visualisaatio-osioon voitaisiin liittää esimerkiksi näyttöjä, mittareita, nappuloita, vipuja tai valoja, joiden avulla ohjattaisiin MeVEAn simulaatiomallia. Yhteysratkaisu halutaan olevan joustava niin, että sitä voitaisiin käyttää myös minkä tahansa simulaatiomallin kanssa. Näin ollen tiedonsiirron olisi hyvä toimia molempiin suuntiin.

2 KIRJALLISUUSKATSAUS

Tässä osiossa tutkitaan työkoneita, niiden voimansiirtoa ja sen toimintaperiaatetta. Tämän lisäksi tutkitaan työkoneiden ohjausjärjestelmiä sekä työkoneiden ja niiden ohjausjärjestelmien simulointia.

2.1 Työkoneet

Erilaiset työkoneet kuten esimerkiksi nosturit, metsä- ja kaivostyökoneet sekä erilaiset kuormaajat ja maansiirtokoneet ovat laajassa käytössä kuljetustehtävissä sekä eri tavoilla tehostamassa ympäristön muokkausta (Backas et al. 2011, s. 161; Immonen 2013, s. 17). Työkoneet ovat raskaita ja niiden liikuttamiseen tarvittava voima on tyypillisesti suuri. Niissä käytetään useimmiten voimansiirrossa hydrauliikkaa. Hydrauliikalla saadaan aikaan suuri tehotiheys ja tehonsiirrossa saadaan nopeasti ulos tarvittava suuri teho. (Backas et al. 2011, s. 149)

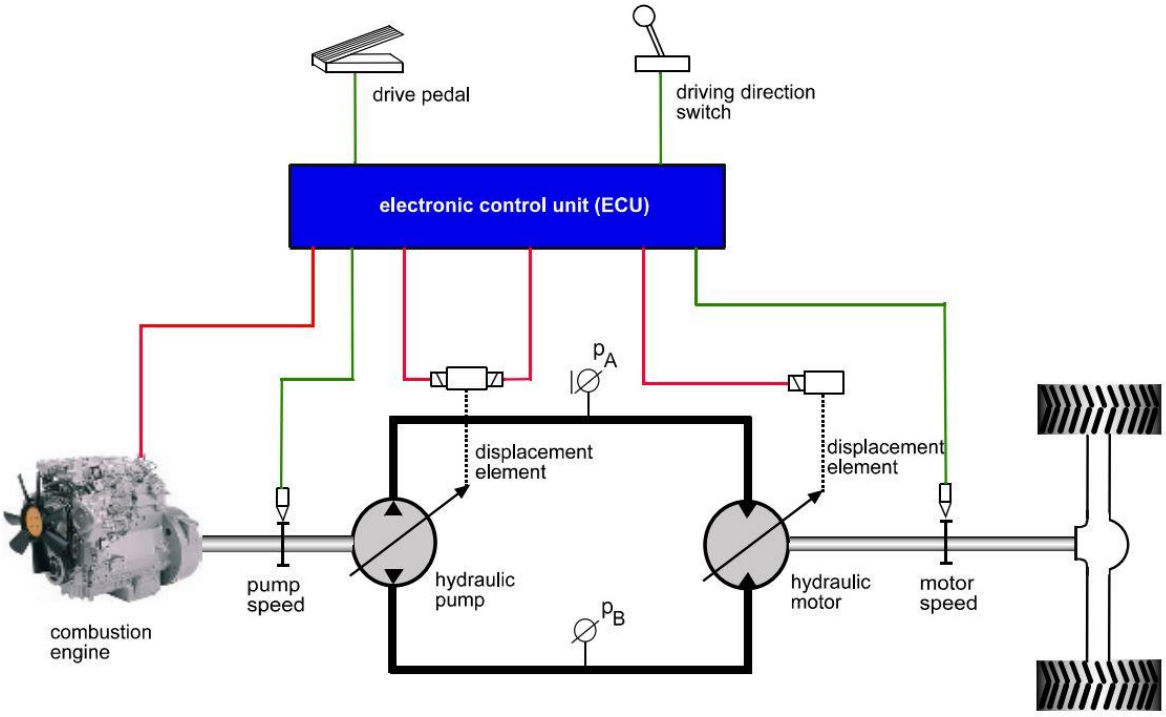
Huomioitavaa on myös, että suurten ja raskaiden työkoneiden käytössä, työturvallisuuteen on käytettävä erityistä huomiota kaikissa työkoneisiin liittyvissä toimenpiteissä jo suunnittelusta alkaen. Työkoneisiin liittyy monia aluekohtaisia turvallisuusmääräyksiä ja säädöksiä, joita tulee noudattaa. (Dadhich et al. 2016, s. 215; Forrai 2013, s. 21.)

2.1.1 Työkoneiden hydrauliikka

Pumput tuovat voiman hydraulisiin järjestelmiin (Manring 2005, s. 256). Korkean paineenalaisen hydraulisen nesteen virtaus siirtää voiman itse toimilaitteille (Manring 2005, s. 3). Hydraulisia järjestelmiä kuvataan usein ”jäykäksi” systeemin pienen hitausvastuksen ja ylityksen (overshoot) vuoksi (Manring 2005, s. 102). Hydrauliset toimilaitteet voidaan jakaa lineaarisiin ja pyöriviin toimilaitteisiin. Lineaarisista toimilaitteista käytetään yleensä nimitystä hydraulinen sylinteri, ja pyöriviä toimilaitteita kutsutaan hydraulisiksi moottoreiksi. (Manring 2005, s. 305.)

Kuvassa 2 nähdään tyypillinen hydraulisen voimansiirron periaatekuva, kun toimilaitteena on työkoneen pyörät. Hydraulinen pumppu on yhteydessä hydrauliseen moottoriin tai

sylinteriin, josta voima siirtyy toimilaitteelle. Kuvan 2 tapauksessa dieselmoottori pyörittää hydraulista pumppua. Ohjausyksikkö koostuu takaisinkytketystä järjestelmästä ja se saa hydraulisen pumpun ja moottorin antureilta tarkan tiedon niiden kaltevuuskulmasta. Renkaiden vääntömomentin ja nopeuden säätö tapahtuu säätelemällä dieselmoottorin nopeutta sekä hydraulisten säätyvätilavuuksisten (variable displacement pump/motor) pumpun ja moottorin kaltevuuskulmia. Säätyvätilavuuksisen pumpun ja moottorin kaltevuuskulmaa muokkaamalla säädetään läpi menevän virtauksen määrää yhdellä kierroksella. Hydraulisia pumppuja ja moottoreita on myös kiinteätilavuusksia (fixed displacement pump/motor), joissa tilavuusvirran määrä on yhdellä kierroksella aina sama (Manring 2005, s. 256–264). Työkoneen kuski säätää ohjauspolkimella työkoneen nopeuden ja ajosuunnan vivulla. Elektroninen ohjausyksikkö saa tätä kautta tiedon halutusta nopeudesta ja kulkusuunnasta. Ohjausyksikkö säätää tiedon perusteella dieselmoottorin, hydraulisen pumpun ja hydraulisen moottorin arvoja. (Ritzke et al. 2011, s. 49–50.)



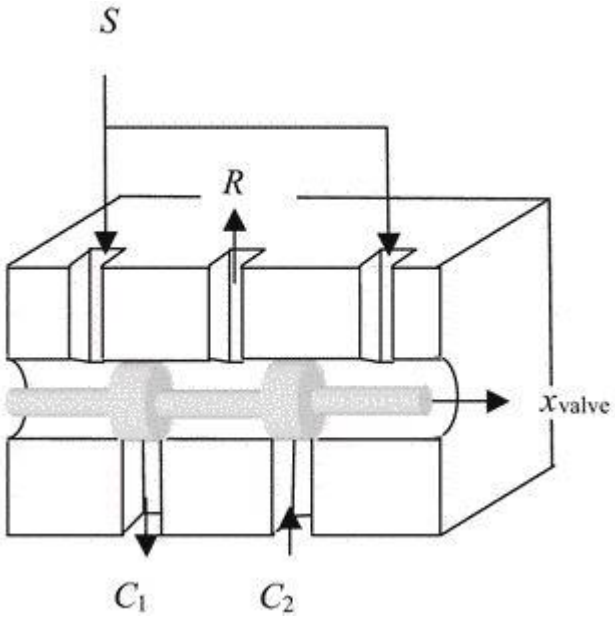
Kuva 2. Hydrostaattinen voimansiirto (Ritzke et al. 2011, s. 50).

Hydraulinen järjestelmä pystyy kontrolloimaan useampaa hydraulista toimilaitetta käyttämällä tarvittaessa useampia hydraulisia pumppuja, hydraulisia moottoreita ja

sylintereitä yhden dieselmoottorin voimin (Backas et al. 2011, s. 162–165.). Venttiilit ovat erittäin tärkeässä roolissa hydraulisia järjestelmiä. Niillä hallitaan hydraulisen järjestelmän virtauksen ja näin ollen voimansiirron suuntaa. Yksinkertaisimmat hydrauliset suuntaventtiilit säätyvät joko auki tai kiinni. Tarkemmissa hydraulisissa järjestelmissä voidaan käyttää proportionaali- tai servoventtiileitä. Nämä venttiilit voivat suunnan ohjauksen lisäksi säätää portaattomasti venttiilin avautumisen astetta ja näin ollen läpivirtauksen määrää. Proportionaaliventtiileitä käytetään usein avoimissa järjestelmissä, ne käyttävät enemmän energiaa ja ne omaavat korkeamman hystereesin sekä hitaamman reaktioajan verrattuna servoventtiileihin. Servoventtiilit toimivat lähinnä suljetuissa järjestelmissä, joissa käytetään takaisinkytkentää ja ne ovat myös paljon tarkempia ja kestävämpiä, mutta myös kalliimpia kuin proportionaaliventtiilit. Avoimista ja suljetuista järjestelmistä kerrotaan tarkemmin kappaleissa 2.3 ja 2.3.1. (Kulakowski 2007, s. 228; Digital Hydraulics 2016, s. 1–13; Manring 2005, s. 169.)

Joskus servojärjestelmä voi koostua useammasta nopeasta ja yksinkertaisesta on/off-suuntaventtiilistä, joita voi olla useampi kytkettynä yhteen samaan toimilaitteeseen. Näitä useampia venttiileitä säätämällä samanaikaisesti auki tai kiinni voidaan hallita portaallisesti toimilaitteelle menevän virtauksen määrää. Mitä enemmän on/off-venttiileitä käytetään, sitä tarkemmin virtauksen määrä voidaan säätää. Tällaista hydraulista servo-ohjausta kutsutaan nimellä digitaalinen hydrauliikka. Digitaalinen hydrauliikka on vasta lähivuosina alettu ottaa käyttöön joissakin hydraulisissa järjestelmissä, mutta sillä on ainakin proportionaalisiin venttiiliratkaisuihin nähden onnistuttu säästämään paljon kustannuksissa sekä parantamaan järjestelmän luotettavuutta. (Digital Hydraulics 2016, s. 1–13.)

Kuvassa 3 alla on esitetty yksinkertaistettu kuva tyypillisestä 4/3-suuntaventtiilin rakenteesta. (Kulakowski 2007, s. 228.)



Kuva 3. 4/3-suuntaventtiilin poikkileikkaus (Kulakowski 2007, s. 229).

4/3-suuntaventtiilissä on neljä eri suuntaa, jonne/josta virtaus voi mennä ja sinä on kolme eri suunta-asetusta. Kuvassa 3 *S* esittää syöttöpaineen tulosuuntaa, *R* kuvaa tankkiin menevää virtausta, C_1 kuvaa sylinterikammio 1:en menevää virtausta ja C_2 sylinterikammioon 2 menevää virtausta. Kuvan 3 venttiilissä mahdolliset suunta-asetukset ovat (Kulakowski 2007, s. 228–229):

1. Ei virtausta
2. Virtaus *S*:stä C_1:en ja C_2:sta *R*:ään
3. Virtaus *S*:stä C_2:en ja C_1:stä *R*:ään

Kuvan 3 venttiili hallitsee virtauksen suuntaa liikuttamalla karaa kuvan asemasta joko hieman x_{valve}-nuolen osoittamaan positiiviseen tai negatiiviseen suuntaan. Itse venttiilin karan liikettä voidaan ohjata sähkövirran avulla esimerkiksi solenoideilla (proportionaaliventtiileissä) tai pienellä vääntömoottorilla (servoventtiileissä), jotka liikuttavat karaa haluttuun asemaan. Venttiilien karan asemaa voidaan ohjata myös esimerkiksi pneumaattisesti tai hydraulisesti. (Kulakowski 2007, s. 228–229; Manring 2005, s. 240–242.)

2.2 Työkoneiden simulaatio

Simulaatiolla tarkoitetaan prosessia, jolla pyritään jäljittelemään todellista tapahtumaa. Simulaatioita ei pysty usein analyyttisin keinoin ratkaisemaan, jolloin voidaan käyttää tietokonesimulaatiota. Tietokonesimulaatiota käytetään laajasti tekniikassa kuten myös monilla muilla eri aloilla. Tekniikassa usein kiinnostavaa on systeemin tai prosessin dynaaminen käyttäytyminen. (Hartmann 1996, s. 1–6) Työkoneissa voidaan simuloida mekaanisia, sähköisiä ja hydraulisia systeemeitä sekä systeemien lämpökäyttäytymistä. Näiden lisäksi voidaan simuloida ohjelmistojen tai säätimien logiikan toimintaa muiden systeemien yhteydessä. (Kulakowski 2007, s. 249; Kelloniemi 2016, s. 53-57) Simulaation avulla voidaan haluta tutkia esimerkiksi rakenteen rasituksia, värähtelyjä, muodonmuutoksia, energiatehokkuutta, tuottavuutta, ohjausjärjestelmän toimivuutta tai muuta dynaamista käyttäytymistä. (Frank et al. 2018, s. 1–6) Simulaatiolla voidaan malliin tehdä helposti muutoksia ilman, että tarvitsee valmistaa kalliita prototyyppejä ja suorittaa niillä tarvittavia testejä. Simulaatiomalli tulisi kuitenkin pystyä validoimaan luotettavasti, sillä yksikin virhe mallissa voi dramaattisesti väärentää simulaatiotuloksia. (Kulakowski 2007, s. 152.)

Jokaisen systeemin simulaation muodostamiseen on omat menetelmänsä, mutta usein työkoneiden simulaatiossa tarvitsee yhdistää kaksi tai useampi systeemityyppiä. Tällöin tarvitaan myös toimiva menetelmä signaalien tai energioiden yhdistämiseen, jotta eri systeemityypit voidaan simuloida yhtenä yhdistettynä systeeminä. Työkoneissa tyypillinen yhdistetty systeemi on hydraulinen systeemi, johon yhdistetään mekaanisten liikkuvien osien muodostuma systeemi. Hydraulisen moottorin tai sylinterin antama voima tai vääntö yhdistyy mekaaniseen malliin liitoskohdassa. (Kulakowski 2007, s. 249; Ritzke et al. 2011, s. 51.)

Hydraulisia systeemeissä vallitsevat paineet ovat tyypillisesti suuria, joten niitä simuloidessa on otettava huomioon mm. hydraulisen nesteen kokoonpuristuvuus sekä hydraulisen nesteen säiliöiden, putkien ja letkujen kimmoisuus. Näiden lisäksi tulee hydraulisen järjestelmän simuloinnissa tyypillisesti laskea mm. nesteen virtaukset, paineet, painehäviöt, sylinterien kitka, vuodot sekä venttiilien dynamiikka. (Kulakowski 2007, s. 218–235; Manring 2005, s. 3) Hydraulisten järjestelmien simulaatio on melko monimutkaista. Tässä työssä ei esitellä

tarkemmin hydraulisen järjestelmän laskukaavoja ja simulaatiomenetelmiä niiden laajuuden vuoksi.

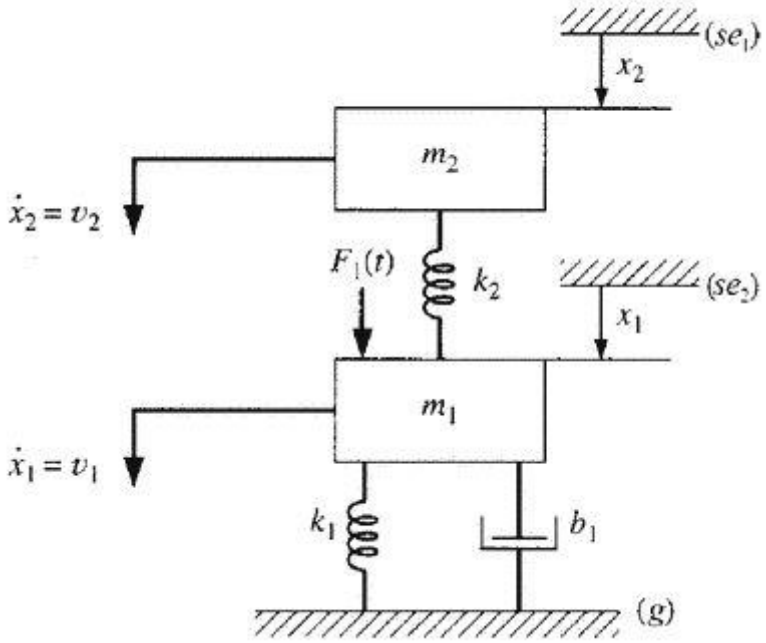
Useissa monimutkaisissa simulaatiotapahtumissa tapauksissa on dynaaminen ohjelmointi havaittu tehokkaimmaksi tavaksi toteuttaa simulaatio. Tällöin monimutkainen ongelma jaetaan useampaan yksinkertaisempaan aliongelmaan, jotka ratkaistaan ja testataan toimiviksi. Pääongelma ratkaistaan rekursiivisesti aliongelmien ratkaisuista. (Kamien 1991, s. 261) Simuloitavat systeemit voivat usein muodostua hyvin monimutkaisiksi. On mahdollista, että yksi ohjelma ei riitä koko systeemin simulointiin ja pitää käyttää useampia ohjelmia, joiden pitää toimia yhdessä. (Franka et al. 2018, s. 1–2; Dadhich et al. 2016, s. 216.)

Joskus simulaatio on erittäin vaikeaa toteuttaa realistisin tuloksin. Esimerkiksi monien maa-ainesta siirtävien työkoneiden simulaatiossa on usein suuria haasteita. Näiden työkoneiden simulaatiossa tulisi mallintaa myös maa-aines. Ongelmana on, että soran, lumen, puulastujen tai muun maa-aineksen poistamis- tai siirtoprosessin realistinen simulaatio useissa tapauksissa on hyvin haasteellista toteuttaa. (Dadhich et al. 2016, s. 213.)

2.2.1 Simulaation matemaattinen mallinnus

Simulaatiossa täytyy luoda matemaattinen malli, kuvaamaan haluttua työkonetta. Dynaamisien matemaattisten mallien luomiseen on esimerkiksi kaksi yleistä menetelmää: State-Space- ja Input-Output-menetelmä. (Kulakowski 2007, s. 54) Tässä kappaleessa on esitelty lyhyesti nämä menetelmät menemättä syvälle itse teoriaan ja niiden muodostustavasta.

Input-Output-menetelmällä tavoitteena on kuvata systeemin liikeyhtälöä yksittäisellä differentiaaliyhtälöllä. Kuvassa 4 on esitetty yksinkertainen kahden vapausasteen mekaaninen systeemi, jossa m kuvaa kappaleen massaa, x kuvaa kyseisen massan sijaintia, k kuvaa jousivakiota, b vaimentimen vaimennuskerrointa ja $F_1(t)$ massaan m_1 vaikuttavaa voimaa. (Kulakowski 2007, s. 54–57.)



Kuva 4. Mekaaninen systeemi (Kulakowski 2007, s. 57).

Massojen liikeyhtälöt luotua ja ne eliminaatiolla yhdistämällä yhdeksi differentiaaliyhtälöksi saadaan yksittäiseksi differentiaaliseksi liikeyhtälöksi (Kulakowski 2007, s. 54–57)

$$\frac{d^4x_1}{dt^4} + \left(\frac{b_1}{m_1}\right)\frac{d^3x_1}{dt^3} + \left(\frac{k_2}{m_2} + \frac{k_1}{m_1} + \frac{k_2}{m_1}\right)\frac{d^2x_1}{dt^2} + \left(\frac{b_1k_2}{m_1m_2}\right)\frac{dx_1}{dt} + \left(\frac{k_1k_2}{m_1m_2}\right)x_1 = \left(\frac{1}{m_1}\right)\frac{d^2F_1}{dt^2} + \left(\frac{k_2}{m_1m_2}\right)F_1(t). \quad (1)$$

State-Space mallinnuksessa tavoitteena on ensin muodostaa systeemin liikeyhtälöistä korkeintaan yhden asteen differentiaaliyhtälöitä. Nämä liikeyhtälöt pyritään järjestämään seuraavaan kaltaiseen muotoon (Kulakowski 2007, s. 54–67):

$$\mathbf{q} = \mathbb{A}\mathbf{q} + \mathbb{B}\mathbf{u} \quad (2)$$

$$\mathbf{y} = \mathbb{C}\mathbf{q} + \mathbb{D}\mathbf{u} \quad (3)$$

Yhtälöissä (2) ja (3) **A** on tilamatriisi, **B** on sisääntulomatriisi, **C** on ulostulomatriisi, **D** on läpivientimatriisi (feedthrough matrix), ***q*** on tilavektori, ***u*** on sisääntulovektori ja ***y*** on ulostulovektori. Läpivientimatriisi **D** on useimmissa tapauksissa nolla. Kuvan 4 kaltainen mekaaninen systeemi State-Space-muodossa on:

$$\begin{bmatrix} \dot{F}_{k1} \\ \dot{v}_1 \\ \dot{F}_{k2} \\ \dot{v}_2 \end{bmatrix} = \begin{bmatrix} 0 & k_1 & 0 & 0 \\ -1/m_1 & -b_1/m_1 & 1/m_1 & 0 \\ 0 & -k_2 & 0 & k_2 \\ 0 & 0 & -1/m_2 & 0 \end{bmatrix} \begin{bmatrix} F_{k1} \\ v_1 \\ F_{k2} \\ v_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1/m_1 \\ 0 \\ 0 \end{bmatrix} F_1(t) \quad (4)$$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1/k_1 & 0 & 0 & 0 \\ 1/k_1 & 0 & 1/k_2 & 0 \end{bmatrix} \begin{bmatrix} F_{k1} \\ v_1 \\ F_{k2} \\ v_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} F_1(t) \quad (5)$$

(Kulakowski 2007, s. 61–67.)

Monilla tietokoneohjelmilla on State-Space-mallista helppoa laskea simulaatiotulokset ja se soveltuu myös epälineaarisiin malleihin toisin kuin Input-Output-menetelmä. Tämän lisäksi State-Space-menetelmä soveltuu monimutkaisempien systeemien simulaatioon. (Kulakowski 2007, s. 81.)

Kahden yllä esitetyn matemaattisen mallinnuksen lisäksi on mahdollista luoda simuloitavasta systeemistä siirtofunktio, mikäli systeemi käyttäytyy lineaarisesti. Siirtofunktiossa luodaan yhtälöön kompleksimuuttuja *s*. Siirtofunktion voi muodostaa esimerkiksi käyttämällä Laplace-muunnosta systeemin dynaamisesta yhtälöstä tai lohkokaaviosta yhdistämällä useammat lohkot lohkokaavioalgebran mukaisesti. Tyypillisellä Laplace-muunnoksella siirtofunktio on systeemin ulostulon ja sisääntulon Laplace-muunnosten osamäärä. Systeemin siirtofunktion perusmuoto on

$$\mathbf{T}(s) = \frac{\mathbf{Y}(s)}{\mathbf{U}(s)}, \quad (6)$$

jossa **T**(*s*) on siirtofunktio, **Y**(*s*) on ulostulon kompleksimuuttujan *s* polynomifunktio ja **U**(*s*) on sisääntulon kompleksimuuttujan *s* polynomifunktio. Luomalla siirtofunktio saatetaan välttää hankalien differentiaaliyhtälöiden muodostaminen ja ratkominen. (Kulakowski 2007, s. 273–299) Siirtofunktion taajuusvasteen ratkaisulla on erityisen tehokasta tutkia systeemin stabiiliutta tai automatisoitujen systeemien dynaamista käyttäytymistä (Kulakowski 2007, s. 323).

2.2.2 Tietokonesimulaatio/numeeriset menetelmät

Vaikka nykyiset tietokoneet ovat erittäin tehokkaita, silti monimutkaiset simulaatiomallit vaativat numeeristen menetelmien käyttämistä approksimoimaan tuloksia. Numeerisia menetelmiä, joita käytetään tietokonesimulaatioon, on esimerkiksi Eulerin menetelmä, parannettu Eulerin menetelmä ja Runge-Kutta-menetelmä. Parannettu Eulerin menetelmä ja Runge-Kutta-menetelmä ovat tarkempia kuin perinteinen Eulerin menetelmä. MATLAB/Simulink-ohjelmalla voidaan käyttää kyseisiä menetelmiä simulaation laskemiseen. Simulaatiomallin voi luoda myös graafisesti lohkokaaviona ja käyttäjä voi valita haluamansa numeerisen menetelmän, jota MATLAB/Simulink käyttää simulaation ratkaisemiseen. (Kulakowski 2007, s. 120–121) MATLAB-ohjelmalla voidaan myös simulaatiomalli ratkaista, jos tiedetään tai lasketaan simulaatiomallin siirtofunktio tai kaavojen (2) ja (3) kaltaiset State-Space-mallin matriisit **A**, **B**, **C** ja **D** (Kulakowski 2007, s. 150–151).

Tietokonesimulaatiota tarvitaan koska, analyyttisillä menetelmillä on alla lueteltuja rajoitteita. Analyyttiset menetelmät soveltuvat lähinnä vain lineaarisiin malleihin, mutta tyypillisesti oikeat simuloitavat systeemit ovat useimmiten epälineaarisia. Vaikka simulaatiomalli olisi lineaarinen, jos siinä on yli kolmannen asteen yhtälöitä, on sen ratkaiseminen hankalaa. Jos halutaan valita sisääntulojen muuttujiksi mielivaltaisia muuttujia, ovat analyyttiset menetelmät usein paljon rajoittuneempia sisääntulomuuttujien valinnan suhteen. Kaikissa tietokonesimulaatioissa simulaation toteutukseen tarvitaan seuraavat osat:

- matemaattinen malli, joka tyypillisesti koostuu useammasta differentiaaliyhtälöstä
- matemaattisen mallin parametrien arvot
- alkuarvot integroitaville muuttujille

- sisääntuloarvot
- ulostuloarvot
- simuloinnin asetusarvot, joita ovat simulaation askelpituus, ulostulon aikaväli, virhetoleranssi ja käytetty integraatioalgoritmi (esimerkiksi parannettu Eulerin menetelmä tai Runge-Kutta-menetelmä).

(Kulakowski 2007, s. 141–142.)

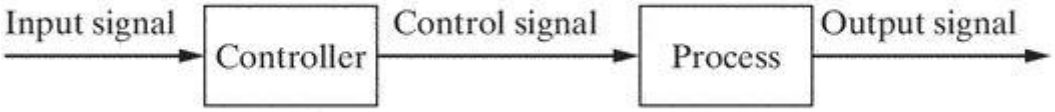
Tyypillisesti ensimmäinen askel tietokonesimuloinnissa on systeemin tapahtumien mallinnus matemaattisilla kaavoilla. Systeemin mallinnuksen voi luoda esimerkiksi aiemmin esitetyillä Input-Output- tai State-Space-menetelmällä. (Kulakowski 2007, s. 157–158) Monimutkaisissa järjestelmissä, kuten esimerkiksi hydraulisissa järjestelmissä, simulaatio on usein helpompaa rakentaa MATLAB/Simulink-ohjelmalla lohkokaaviolla (Kulakowski 2006, s. 231–233).

2.3 Työkoneiden ohjausjärjestelmät

Työkoneissa käytetyn digitaalilaitteiden ja niiden tiedonsiirron määrä on ollut jatkuvassa kasvussa. Työkoneissa on usein monia erilaisia älykkäitä yksiköitä. Nämä yhdessä muodostavat työkoneeseen hajautetun ohjausjärjestelmän. Älykkäät yksiköt kommunikoivat keskenään reaaliajassa. Yksi useasti käytetty reaaliaikaisen tiedonsiirron verkosto on CAN-väylä (Controller Area Network). CAN-väylä on parikaapeli, jota käytetään ohjausjärjestelmän tiedonsiirtoon eri yksiköiden välillä. Sillä voidaan yhdistää esimerkiksi moottorin, voimansiirron, jarrujen, työlaitteen ja käyttöliittymän ohjausyksiköt. CAN-väylä on alun perin autoteollisuudelle suunnattu, mutta sitä käytetään myös esimerkiksi kallionporauslaitteiden, hissien ja metsäkoneiden ohjausjärjestelmissä. (Hietikko 1996, s. 2.)

Työkoneissa ohjausjärjestelmässä voi olla myös paljon muita käyttäjää avustavia toimintoja, kuten esimerkiksi varoitus törmäyksestä, huonosta käyttötavasta tai muusta vaarasta. Joissain tapauksissa ohjausjärjestelmä voi tilanteen sattuessa ottaa ohjauksen omaan haltuun joko avustamalla tai estämällä ongelmatilanne. Mitä monimutkaisemmaksi työkoneen ohjausjärjestelmä muodostuu, sitä hankalampaa on myös itse ohjausjärjestelmä toteuttaa ja säätää. (Dadhich et al. 2016, s. 214.)

Ohjaus voidaan toteuttaa avoimella tai suljetulla silmukalla. Avoimessa silmukassa ohjausjärjestelmä ohjaa prosessia saamatta mitään tietoa itse prosessin ulostulosta. Kuvassa 5 alla on kuvattu avoimen silmukan kaltaisen ohjausjärjestelmän lohkokaavio. (Kulakowski 2007, s. 356.)



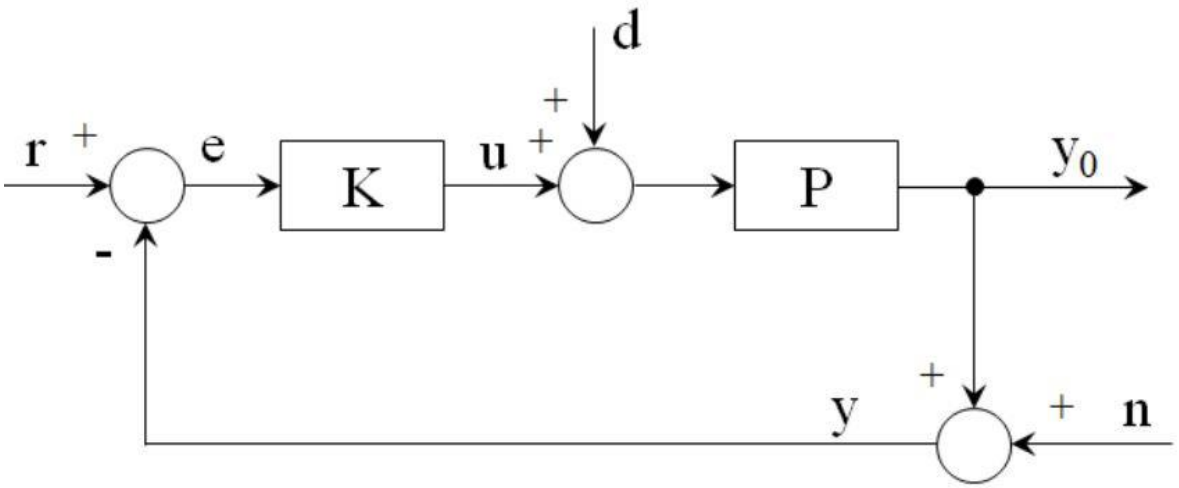
Kuva 5. Avoimen silmukan ohjaus (Kulakowski 2007, s. 357).

Avoimet järjestelmät ovat yksinkertaisempia ja halvempia, mutta ne eivät ole niin tarkkoja, koska ulostuloarvoa ei verrata haluttuun sisääntuloarvoon. Avoimen silmukan ohjausta voidaan käyttää usein prosesseissa, jotka eivät ole kovin alttiita ulkoisille häirinnöille, jotka ovat hyvin toistuvia ja joissa ei vaadita suurta ulostulon tarkkuuden säätöä. Työkoneissa kuitenkin tarvitaan tarkkuutta vaativia ohjausjärjestelmiä sekä ulkoisten häiriöiden vaikutus voi olla hyvin vaihteleva ja näin ollen käytetään ohjausjärjestelmissä tyypillisemmin suljettuja silmukoita. Suljetussa silmukassa prosessin ulostulosignaali tyypillisesti lisätään tai vähennetään takaisin sisääntuloarvoon. Jos ulostuloarvo lisätään sisääntulosignaaliin, kyseessä on positiivinen takaisinkytkentä ja jos taas ulostuloarvo vähennetään sisääntuloarvosta, on kyseessä negatiivinen takaisinkytkentä. (Kulakowski 2007, s. 356–357.)

2.3.1 Takaisinkytkentä ja sen säätöjärjestelmä

Tyypillisesti, kun työkoneissa halutaan suurta tarkkuutta vaativia systeemeitä, käytetään ohjauksessa negatiivista takaisinkytkentää. Koska ulostuloarvoa verrataan haluttuun sisääntuloarvoon, saadaan negatiivisella takaisinkytkennällä eliminoitua ulkoisten häiriöiden aiheuttamat virheet. Hydraulisessa voimansiirrossa käytetään tyypillisimmin negatiivisesti takaisinkytkettyjä ohjausjärjestelmiä. Takaisinkytketyt systeemit ovat kalliimpia ja yhtenä suurimpana ongelmana on, että tällöin myös systeemistä tulee epästabiilimpi. Tämän lisäksi takaisinkytkentä kasvattaa mahdollisesti systeemin viivettä (Kulakowski et al. 2007, s. 424 s.). (Kulakowski 2007, s. 351) Koska työkoneissa käytetään tyypillisesti hydrauliikkaa, löytyy lähes kaikista työkoneista negatiivisella

takaisinkytkennällä varustettuja ohjausjärjestelmiä. Negatiivisen takaisinkytkennän lohkokaavio on esitetty kuvassa 6 alla. (Kulakowski 2007, s. 351–356)



Kuva 6. Negatiivisen takaisinkytketyn ohjausjärjestelmän lohkokaavio (Forrai 2013, s. 41).

Kuvassa 6 *K* on säädin, *P* kuvaa prosessia/koneistoa, *r* kuvaa haluttua viitearvoa/sisääntulosignaalia, y_0 kuvaa ulostuloarvoa, *n* kuvaa mittauskohinaa, *y* kuvaa mittausarvoa, *e* kuvaa erotussignaalia, *u* kuvaa ohjaussignaalia ja *d* kuvaa ulkoista häiriötä. Takaisinkytkennässä ulostuloarvon y_0 haluttaisiin olevan tyypillisesti sama kuin viitearvo *r*. Mittausarvo *y* vähennetään viitearvosta *r*, jolloin säädin *K* saa erotussignaalin *e*. Säädin käsittelee ja tyypillisesti vahvistaa erotussignaalia ennen kuin itse toimilaitteita ohjaava koneisto *P* saa ohjaussignaalin ja liikuttaa toimilaitteita sen mukaan. Tästä muodostuu loputon silmukka, joka kiertää jatkuvasti reaaliajassa siirtäen aina toimilaitteita, jos niiden mitattu *y* arvo ei ole tarpeeksi lähellä viitearvoa *r*. (Forrai 2013, s. 40–41) Prosessina *P* voi olla esimerkiksi pyöräkuormaajan kauhan kaltevuuden säätö ja viitearvona *r* kauhan kaltevuuskulma. Säätimenä voidaan käyttää esimerkiksi PID-säädintä (Proportional-Integral-Derivative). Teollisuudessa käytetyistä digitaalisista säätimistä yli 90% on PID-säätimiä. (Kulakowski et al. 2007, s. 421.)

PID-säädin koostuu kolmesta tekijästä: Suhdeosasta (P), integroivasta osasta (I) ja derivoivasta osasta (D). PID-säätimen siirtofunktio Laplace-muunnoksen avulla on

$$K(s) = K_P + \frac{K_I}{s} + K_D s, \quad (5)$$

jossa *K*:t viittaavat vahvistuksen suuruuteen P-, I, ja D-arvoille. Kyseisten vahvistusten suuruus kasvattaa itse kyseistä P-, I- tai D-arvoa. P-arvon suuruus vaikuttaa prosessin nopeuteen eli siihen, kuinka nopeasti sisääntulosignaalin referenssiarvo saavutetaan (eli vähentää nousuaikaa). Tämän lisäksi P-arvo vähentää pysyvän tilan poikkeamaa. Suuri P-arvo taas kasvattaa signaalin ylitystä, heikentää systeemin stabiiliutta ja kasvattaa hieman asettumisaikaa. D-arvo taas vähentää ylitystä, mutta kasvattaa asettumisaikaa. I-arvo vähentää nousuaikaa, kasvattaa ylitystä ja asettumisaikaa sekä poistaa pysyvän tilan poikkeuman. Tavoitteena PID-säätimen virityksessä on löytää ohjausjärjestelmälle sopiva tasapaino P-, D- ja I-arvojen kesken, jotta systeemin ohjaus toteutuu halutun mukaisesti ja systeemi pysyy stabiilina kaikissa tilanteissa. Jotkin D- tai I-arvoista voivat myös olla nollia, jolloin saatetaan puhua myös P-, PI- tai PD-säätimistä. PID-säätimen arvojen säädön voi toteuttaa simulaatiolla eri arvoja testaten. On olemassa myös ohjemenetelmiä PID-säätimen virittämiseen, kuten Ziegler-Nichols-, Chien-Hrones-Reswick- ja kerroinkaaviomenetelmä (coefficient diagram method). (Kulakowski 2007, s. 365–370; Forrai 2013, s. 115–121.)

Ohjausjärjestelmien säätimet ovat voivat olla analogisia tai digitaalisia. Myös PID-säädin voi olla digitaalinen tai analoginen. Digitaaliset säätimet ovat kehittyneet paljon viime vuosina, ne ovat paljon joustavampia ja niillä voidaan luoda myös hyvin monikutkaisia säätöalgoritmeja. Nykyiset säätimet ovat lähinnä digitaalisia. Analogiset säätimet ovat usein kalliimpia ja rajoittuneempia. Digitaaliset säätimet voivat olla ohjelmoituina esimerkiksi mikrotietokonesiruun tai muihin erillisiin digitaalisiin laitteisiin. Säädinalgoritmin voi koodata usein vain muutamalla rivillä BASIC- tai C-ohjelmointikieleltä. Säädinalgoritmia on myös digitaalisessa säätimessä helppoa muokata tarvittaessa. (Kulakowski, 2007, s. 410–424.)

2.3.2 Automatisoidut ja etäohjatut työkoneet

Työkoneita voidaan tyypillisen manuaalisen ohjauksen lisäksi myös ohjata etänä kauko-ohjauksella tai ne voivat toimia osittain tai täysin automatisoidusti (Backas et al. 2011, s. 162–165; Dadhich et al. 2016, s. 213–214). Työkoneiden automaatiota on kehitetty ja

tutkittu on viimeiset kolme vuosikymmentä laajasti. Tekniikan kehitys etenkin elektroniikka-, tietokone ja informaatioteknologian alalla on avannut uusia kehitysmahdollisuuksia työkoneillekin. Työkoneissa kehitystä on tapahtunut etenkin automaatiossa ja etäohjauksessa. (Backas et al. s. 162.)

Joidenkin työkoneiden, kuten pyöräkuormaajien joukossa, täysin automaattiset kaivoskoneet eivät ole vielä päässyt tuottavuudeltaan samalle tasolle kuin manuaaliset työkoneet. Vaikka työkoneissa automatiikkaa on paljon tutkittu, vain harva yritys käyttää etäohjattuja tai puoliautomaattisia pyöräkuormaajia. Toisaalta kaivoksissa maa-aineksen kuljetukseen erikoistuneita työkoneita, kuten kippiautoja, on jo markkinoilla myös täysin automaattisia versioita useammalta yritykseltä (esimerkiksi Caterpillar, Atlas Copco ja Sandvik). Automaattiset ja etäohjatut työkoneet tarvitsevat tiedonsiirtoyhteyden käyttäjään tai muuhun yhteyspisteeseen. tiedonsiirto voidaan toteuttaa esimerkiksi WLAN-, 3G- tai 4G-yhteydellä. (Dadhich et al. 2016, s. 212–218.)

Vaikka teoriassa automaatiolla päästään vähintään yhtä hyvään tulokseen kuin manuaalisella ohjauksella, haasteena on hyvin monimutkaisten algoritmien luonti ohjausjärjestelmään. Nykyiset ohjausjärjestelmät vaatisivat vielä kehitystä, tutkimusta ja yhteistyötä laitteiden valmistajien kanssa, jotta voitaisiin tuoda markkinoille muitakin täysin automaattisia työkoneita kuin kippiautoja. (Dadhich et al. 2016, s. 212–221.)

2.3.3 Ohjelmistojen testaaminen

Moderneista työkoneista löytyy yhä monimutkaisempia ja kehittyneempiä ohjelmistoja. Ohjausjärjestelmien ohjelmistot ovat hyvin sidottuja niitä käyttäviin koneisiin ja laitteistoihin. Tyypillisesti työkoneen ohjausjärjestelmän ohjelmistoa voidaan testata vain kyseisellä työkoneella tai koneella, jonka mekaaninen rakenne on identtinen. (Huang 2007, s. 1–2) Osittain automatisoiduissa ohjauksissa voi itse ohjelmisto suorittaa suurenkin osan työkoneen ohjauksesta. Näin ollen on tärkeää myös, että ohjelmisto ja sen toimivuus yhdessä koko ohjausjärjestelmän kanssa testataan huolellisesti. Ohjelmistojen testaamiseen metodologisesti kaksi tyypillistä menetelmää ovat Black box -ja White box -menetelmät. (Limaye 2009, s. 107.)

Black box -menetelmässä huomio kiinnitetään ohjelmiston sisään- ja ulostuloarvoihin. Testaajalla ei ole ohjelman koodin tai sen sisällöstä tietoa, ja näin ollen testimenetelmä on ”riippumaton” ohjelman sisältämästä koodista. Tavoitteena antaa ohjelmalle tositilanteessa mahdollisia sekä potentiaalisesti virhealttiita sisääntuloarvoja ja tarkastaa, että ulostuloarvot saavat oikeita arvoja täyttävät vaaditut kriteerit. Ongelmana on, että koska testaaja ei tiedä koodin sisällöstä mitään, hän saattaa jättää tarkastelematta joitakin koodin rakenteeseen ja sisältöön liittyviä tyypillisiä virhetilanteita. (Limaye 2009, s. 107–109.)

White box -menetelmä toimii muuten samalla periaatteella kuin Black box -menetelmä, mutta siinä testaaja tuntee myös ohjelmiston koodin sisällön ja tarkastelee sen toimivuutta. Tällöin testataan tyypillisesti, että kaikki ohjelman koodin rakenteet toimivat halutulla tavalla. (Limaye 2009, s. 107–110.)

2.4 Työkoneiden ohjausjärjestelmien virtuaalinen testaus

Ohjausjärjestelmien virtuaalinen testaus tuo monia etuja tavanomaiseen testaukseen. Useammat vaihtoehtoiset järjestelmät voidaan simuloida rinnakkain ja simulaatio ei ole ajasta riippuvainen. Itse simuloitavan koneen tai prosessin ja sen ohjausjärjestelmän mallinnus voidaan toteuttaa ohjelmointiympäristöillä kuten MATLAB, Dymola, SIMPACK tai MeVEA. Samoilla ohjelmilla voidaan luoda myös prosessin/koneen dynaaminen simulaatiomalli, joka liitetään ohjausjärjestelmän suljettuun silmukkaan. Dymola, SIMPACK ja MeVEA antavat myös tarvittaessa graafisen esityksen simulaatiomallin käyttäytymisestä. (Huang 2017, s. 2; MeVEA 2012) Ongelmana saattaa olla ohjausjärjestelmien ohjelmistojen mallinnus, ellei ohjausjärjestelmän koodia luoda itse. Työkoneiden ohjelmistojen koodit omistavat, ainakin osittain, työkoneiden valmistajat. Ohjausjärjestelmien koodit tulisi saada valmistajalta, jotta voitaisiin luoda mahdollisimman tarkka virtuaalinen ohjausjärjestelmän testiympäristö. Valmistajat eivät yleensä halua jakaa näitä ohjelmistoja. Jos virtuaalinen ohjausjärjestelmien testaus on mahdollista, sillä voidaan päästä pienempiin testauskustannuksiin. Samalla voidaan ajaa monta eri testiasetusta samanaikaisesti ja simulaatioaika määräytyy tietokoneen prosessointitehon mukaan. Näin ollen testitapahtuma voidaan toteuttaa nopeammin. (Huang 2017, s. 1–2.)

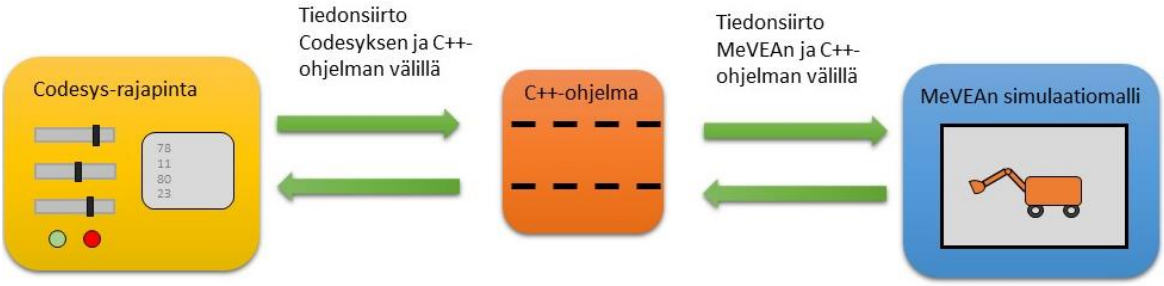
2.4.1 Ohjausjärjestelmän virtuaalinen testaus MATLAB-ohjelmalla

Ohjausjärjestelmän simulaatio voidaan toteuttaa esimerkiksi MATLAB-ohjelmalla, joko suoraan tai usein monimutkaisissa tapauksissa MATLABin sisältämällä Simulink-liitännäisohjelmalla lohkokaaviolla. Tässä kappaleessa on kuvattu yksinkertaistettu menettelytapa MATLAB/Simulink-ohjelmalla muodostetun ohjausjärjestelmän virtuaaliseen testaukseen lohkokaaviolla. (Kulakowski 2007, s. 365–378.)

Ensin muodostetaan itse koneen/prosessin dynaaminen matemaattinen malli, jonka jälkeen se liitetään takaisinkytkettyyn ohjausjärjestelmään (kun ohjauksessa käytetään tyypillistä suljettua silmukkaa), johon liitetään myös PID-säädin. Näin muodostetaan kuvan 6 kaltainen lohkokaavio ohjausjärjestelmästä. Lohkokaavion säädin *K* ja prosessi/koneisto *P* voivat olla alimalleja (sub-model), jossa *K* sisältää PID-säätimen ja *P* koko systeemin/prosessin dynaamisen simulaatiomallin. Tämän jälkeen suoritetaan PID-säätimen viritys ja tutkitaan systeemin käyttäytymistä halutuilla sisääntulosignaaleilla. PID-säätimen alimallissa muuttujina ovat kappaleessa 2.3.1 esitetyt säätimen K_P-, K_i-, ja K_D-arvot, joihin yritetään löytää optimaaliset arvot. (Kulakowski 2007, s. 365–378.)

3 TULOKSET

Kirjallisuuskatsauksen lisäksi tässä työssä pyrittiin löytämään työkoneen reaaliaikaiseen simulaatioon kommunikointiyhteys MeVEAn ja Codesyksen välillä. Haluttiin yhdistää Codesyksen monipuolinen ja helposti muokattava visuaalinen käyttöliittymän rajapinta sekä yksinkertaisesti ja helposti tuotettu koneen ohjaus MeVEAn reaaliaikaiseen simulaatioon. Ohjelmien välisen tiedonsiirron tulisi pystyä kulkemaan molempiin suuntiin. Avuksi kehitettiin C++-ohjelma MeVEAn ja Codesyksen tiedonsiirron väliin. C++-ohjelma käsittelee ja vastaanottaa Codesyksen lähettämän datan ja lähettää sen edelleen MeVEAlle ja samalla vastaanottaa ja käsittelee MeVEAn lähettämän datan ja lähettää sen edelleen Codesykselle. Alla oleva kuva 7 havainnollistaa tiedonsiirtoa MeVEAn, Codesyksen ja C++-ohjelman välillä.



Kuva 7. Tiedonsiirto Codesyksen ja MeVEAn välillä.

Kommunikointiyhteyttä Codesyksen ja C++-ohjelman välillä yritettiin kahdella eri menetelmällä. Käyttämällä verkkoliitännän UDP-yhteyttä ja jaettua muistia. Tiedonsiirto MeVEAn ja C++-ohjelman välillä toteutettiin Socket-yhteydellä. Kappaleissa 3.1, 3.2 ja 3.3 on selitetty tarkemmin käytetyistä tiedonsiirtomenetelmistä.

Tiedonsiirrossa Codesyksen ja C++-ohjelman välillä, molemmille menetelmille (UDP ja jaettu muisti), on pitänyt luoda omat erilliset C++- ja Codesys-ohjelmat. Ohjelmista ajetaan joko UDP:n tai jaetun muistin menetelmän ohjelmat riippuen kumpaa menetelmää halutaan käyttää. Molempia menetelmiä testattaessa kaikki ohjelmat on ajettu samalla tietokoneella. Codesyksellä ohjelmat on kirjoitettu IEC 61131-3 standardin mukaisella Structured text -

ohjelmointikielellä (rakenteinen teksti). Codesyksellä ohjelma ladattiin tietokoneen virtuaaliselle Win x64 PLC:lle (programmable logic controller), jotta koodi toimii Codesyksen puolelta. Codesys-ohjelmaa luotaessa laitteeksi tulee valita CODESYS Control Win V3 x64 (3S – Smart Software Solutions GmbH), jotta ohjelma voidaan ladata virtuaaliselle PLC:lle. MeVEAn ja Codesyksen tiedonsiirron välillä oleva ohjelma on tehty Visual Studio -ohjelmalla C++-ohjelmointikielellä. Ohjelmaversiot ovat Visual Studio 2017 15.6.7, CODESYS V3.5 SP12 64-bit ja CODESYS Control Win v3 – x64 Version 3.5.12.0. MeVEAn ohjelmistoversiot ovat: MeVEA Modeller v2.3.889, Open Scene Graph v3.4.1, wxWidgets 3.0.3 ja MeVEA Solver library 7.70.2916.

Codesyksen ja C++-ohjelman (MeVEAan) yhteys onnistuttiin toteuttamaan jaetun muistin menetelmällä ja rajoitetusti UDP-yhteydellä. Sekä UDP-yhteydellä että jaetun muistin menetelmällä siirrettiin seuraavia datatyyppejä: totuusarvoja (bool), merkkejä (char), kokonaislukuja (int) ja tarkempia liukulukuja (double).

Datan siirto tapahtui reaaliaikaisesti 0,2 sekunnin välein molempiin suuntiin. Tämä toteutettiin asettamalla ohjelmat odottamaan 0,2 sekuntia jokaisen datansiirtokierroksen jälkeen. Jos näin ei tehtäisi, ohjelmat pyrkisivät lähettämään dataa niin nopeasti kuin mahdollista ja tämä käyttäisi kaikki tietokoneen prosessointitehon. Ohjelmien 0,2 sekunnin odotusaikaa tosin voidaan laskea pienemmäksi, jos halutaan nopeampaa tiedonsiirtoa. Odotusaikaa voidaan myös kasvattaa, jos vapauttaa enemmän tietokoneen prosessointitehoa muihin tarkoituksiin.

3.1 UDP-yhteys Codesyksen ja C++-ohjelman välillä

Datan siirtoon verkkoliitännän avulla yleisimmät protokollat ovat UDP (user datagram protocol) ja TCP (transmission control protocol). UDP-yhteys soveltuu paremmin reaaliaikaisen datan siirtoon kuin TCP. Tämä johtuu siitä, että TCP-protokolla käyttää osan yhteydestä datan verifioimiseen ja lähettää mahdollisesti korruptoituneen datan uudestaan. Reaaliaikaisessa datan siirrossa datan varmennukselle ei ole tarvetta, koska dataa lähetetään jatkuvasti kuvaamaan reaaliaikaista tilannetta. (Dadhich et al. 2016, s. 218; Alani 2014, s. 19–34) Näin ollen yhteys MeVEAn ja Codesyksen välille haettiin UDP-yhteydellä, koska se soveltuu paremmin reaaliaikaiseen tiedonsiirtoon kuin TCP. UDP-yhteyttä käyttäen

voitaisiin haluttaessa käyttää erillistä tietokonetta Codesykselle ja MeVEAlle, kun molemmat tietokoneet ovat liitettynä samaan verkkoon (esim. Internettiin tai lähiverkkoon).

Codesyksellä käytettiin Network Variable List (NVL) -objektia lähettämään dataa UDP-sockettiin. Myös vastaanotettavalle datalle luotiin oma NVL-objekti. Tämän lisäksi luotiin visualisaatio, josta nähdään lähetetyt ja vastaanotetut muuttujat. Testeissä lähettiin dataa 0,2 sekunnin välein.

Lähettävään NVL-objektiin lisätään kaikki halutut lähetettävät muuttujat ja alustetaan niiden datatyypit. Lähettävä NVL-objekti luo GVL-tiedoston (Global Variable List). Tyypillisesti NVL-objektit lähettävät dataa ja vastaanottavat dataa toisilta NVL-objekteilta. Mutta tässä työssä dataa halutaan siirtää NVL-objektien C++-ohjelman välillä. Vastaanottava NVL-objekti tyypillisesti avaisi tämän GVL-tiedoston ja saisi sen kautta tiedon, mitä dataa lähetetään ja millä tavalla. Lähettävä NVL-objekti lisää jokaisen datapaketin eteen merkkijonon, joka antaa tiedon näistä NVL-objektin sisältämistä asetuksista (asetuksista, jotka ovat myös tallennettuna kyseiseen GVL-tiedostoon). Jotta Codesyksen vastaanottava NVL-objekti pystyy lukemaan vastaanotetun datan ohjelman käyttöön, on tämän merkkijonon oltava juuri sellainen, kun vastaanottava NVL-objekti odottaa. Muuten koko datapakettia ei voida lukea. Näin ollen lähettävän NVL-objektin luoma GVL-tiedosto halutaan kopioida ja muokata sellaiseksi, että vastaanottava NVL-objekti tunnistaa sen ja osaa sen perusteella lukea vastaanotetun datan. GVL-tiedosto on yksinkertaisen tekstitiedoston kaltainen tiedosto, joka sisältää tietoa mm. lähetettyjen muuttujien määrästä, niiden nimistä ja datatyypeistä sekä tiedon käytetystä portista, lähettäjän IP-osoitteesta, List Identifier -arvosta ja muista NVL-objektin asetuksista. Kopioidusta GVL-tiedostosta tulee muuttaa ainakin muuttujien nimet (ettei käytetä samoja muuttujanimiä sekä lähettävässä että vastaanottavassa NVL-objektissa) ja List Identifier -arvo. Näiden lisäksi GVL-tiedosto tulee muokata sellaiseksi, että sen sisältämien muuttujien määrä ja datatyypit vastaavat C++-ohjelman lähettämiä muuttujien nimiä ja datatyyppejä.

Codesys ei tarjoa informaatiota siitä, mitä lähettävän NVL-objektin datapaketin alussa oleva merkkijono sisältää, mitä datatyyppiä merkit ovat tai kuinka monta merkkiä merkkijonoon kuuluu. Tässä työssä UDP-yhteydellä olisi siis tavoitteena saada C++-ohjelma lukemaan

Codesyksen lähettävän NVL-objektin datapaketin ja samalla saada Codesyksen vastaanottava NVL-objekti lukemaan C++-ohjelman lähettämiä datapaketteja. Näin ollen tulisi myös selvittää NVL-objektien alussa olevien merkkijonojen tarkempaa sisältöä, jotta UDP-yhteys muodostuisi onnistuneesti.

C++-ohjelmalla tavoitteena oli ensin pystyä lukemaan Codesyksen NVL-objektin lähettämä datapaketti ja erottaa sekä tulkita datapaketin alussa olevan merkkijonon sisältö. Merkkijonon sisällön selvittyä yritettiin kopioida tämä merkkijono ja liittää se C++-ohjelman lähetettävän datapaketin eteen ja lähettää datapaketti Codesyksen vastaanottavalle NVL-objektille. C++-ohjelman tulisi siis myös luoda jokaisen datapaketin eteen merkkijono, jonka Codesyksen vastaanottava NVL-objekti ymmärtää ja joka vastaa vastaanottavan NVL-objektin GVL-tiedoston asetuksia.

C++-ohjelma tallentaa vastaanotetun datan ensin yhteen kokonaislukujonoon. Ensin tästä datapaketista erotetaan Codesyksen lähettävän NVL-objektin datapaketin alussa oleva merkkijono. Tämän jälkeen kokonaislukujonosta luetaan dataa järjestyksessä, kun tiedetään, mitä datatyyppejä ja missä järjestyksessä Codesys-ohjelma niitä lähettää. Merkkijonosta siis valitaan aina tietyn pituinen pätkä dataa, jota luetaan sen mukaisena datatyyppinä. Luettu Datatyyppi tallennetaan muuttujaan tai muuttujajonoon, josta se myöhemmin lähetettäisiin C++-ohjelmalta MeVEAlle.

UDP-yhteydellä onnistuttiin datan lähettäminen Codesykseltä C++-ohjelmalle, mutta vastakkaiseen suuntaan tiedonsiirto ei onnistunut tällä menetelmällä. Menetelmällä onnistuttiin lähettämään Codesyksestä C++-ohjelmalle kaikkia testattuja datatyyppejä (bool, char, int ja double) myös sekaisin (useita eri datatyyppejä yhdessä lähetetyssä datapaketissa). Codesyksen lähettävän NVL-objektin jokaisen datapaketin alussa oleva asetuksiin viittaava merkkijono koostui 20:stä yhden tavun kokoisesta merkistä (char). Jokainen merkki voi siis saada 256 eri arvoa (eli kokonaisluvut väliltä 0 – 255 tai -128 – 127 riippuen onko kyseessä signed char vai unsigned char).

Testeissä merkit luettiin etumerkillisinä merkkeinä (signed char). NVL-objektien alussa olevasta 20:a merkistä saatiin seuraavat tiedot:

- 9:s luku oli NVL-objektin list identifier -arvo.
- 11:s luku oli 1, jos NVL-objektin pack variables -asetus oli päällä. Vastaavasti jos pack variables -asetus oli pois päältä, 11:a luku sai arvon 0.
- 13:s luku ilmoitti, kuinka monta muuttujaa datapaketissa siirrettiin.
- 15:s luku kertoi koko datapaketin koon tavuina.
- Luvut 17 – 18 ilmaisivat, kuinka mones viesti on kyseessä seuraavalla tavalla: Alussa 18:a luku on 0. 17:a luku alkaa arvosta 1 ja kasvaa yhdellä aina seuraavalla datapaketilla. Kun 17:a luku oli 256, seuraavalla datapaketilla tämä luku sai taas arvon 0 ja 18:a luku kasvoi yhdellä. Näin ollen NVL-objektit pystyivät laskemaan, kuinka mones datapaketti oli kyseessä arvoon 65 536 asti (256 * 256 = 65 536).
- Kun transmit checksum -asetus oli päällä 19:s luku sai arvon 2 ja 20:s luku arvon -31. Kun taas transmit checksum -asetus ei ollut päällä, molemmat 19:s ja 20:s luku saivat arvon 0.

Seuraavien NVL-objektin asetusten ei nähty vaikuttavan 20:n alussa olevaan merkkiin: Cyclic interval ja transmit on change -asetukset sekä sillä, mikä valinta on task-valikosta valittuna. Lisäksi jos acknowledgement -asetuksen pisti päälle, datan lähettäminen ei onnistunut.

Codesyksen NVL-objekti vastaanotettavalle datalle, ei tunnistanut C++-ohjelman lähettämiä datapaketteja. UDP-yhteyden testaaminen jäi kesken, koska jaetun muistin menetelmällä onnistuttiin löytämään toimiva yhteysratkaisu.

3.2 Jaetun muistin menetelmä Codesyksen ja C++-ohjelman välillä

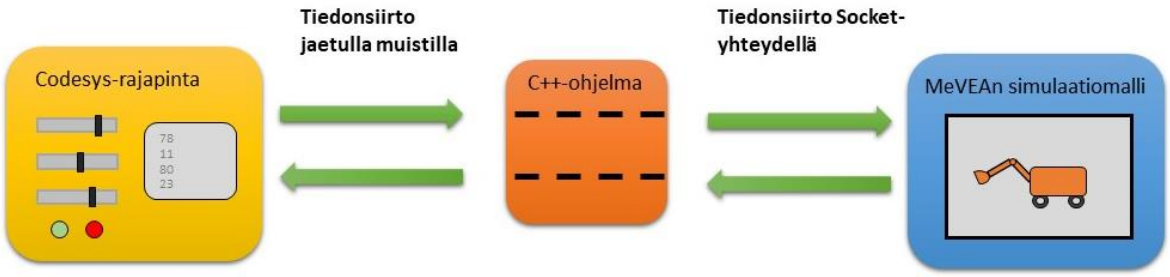
Jaetun muistin menetelmällä osa tietokoneen muistista merkataan jaetuksi, jolloin eri ohjelmat voivat käyttää tietokoneen samaa muistiosiota. Tämä menetelmä on myös UDP-menetelmää suorituskykyisempi, mutta myös rajoitetumpi. Viestintä tapahtuu jaetun muistin välityksellä nopeammin, mutta rajoitteena on käytettävä yhtä tietokonetta, jossa on käynnissä kaikki tarvittavat ohjelmat. (El-Rewini 2004, s. 77–80.)

Codesyksessä jaetun muistin menetelmässä käytettiin pääohjelman lisäksi visualisaatiota ja NVL-objektien tilalla kaksi DUT (data unit type) -objektia. Apuna käytettiin Codesyksen

tarjoamaa ilmaista Shared Memory Communication -esimerkkiprojektia (Codesys 2019). Kahteen Structure-tyyppiseen DUT-objektiin lisätään haluttavat muuttujat, jotka jaetaan jaetun muistin menetelmällä (yhteen DUT:in vastaanotettavat muuttujat ja toiseen DUT:in lähetettävät muuttujat). Visualisaatiota käytettiin graafisen käyttöliittymän luomiseen, josta halutaan ohjata MeVEAssa simuloitua työkonetta. Graafisen käyttöliittymän napeilla, vivuilla ja muilla ohjaimilla halutaan siis ohjata MeVEAn simulaatiomallia.

3.3 Tiedonsiirto MeVEAn ja Codesyksen välillä käyttäen jaettua muistia ja Socket-yhteyttä

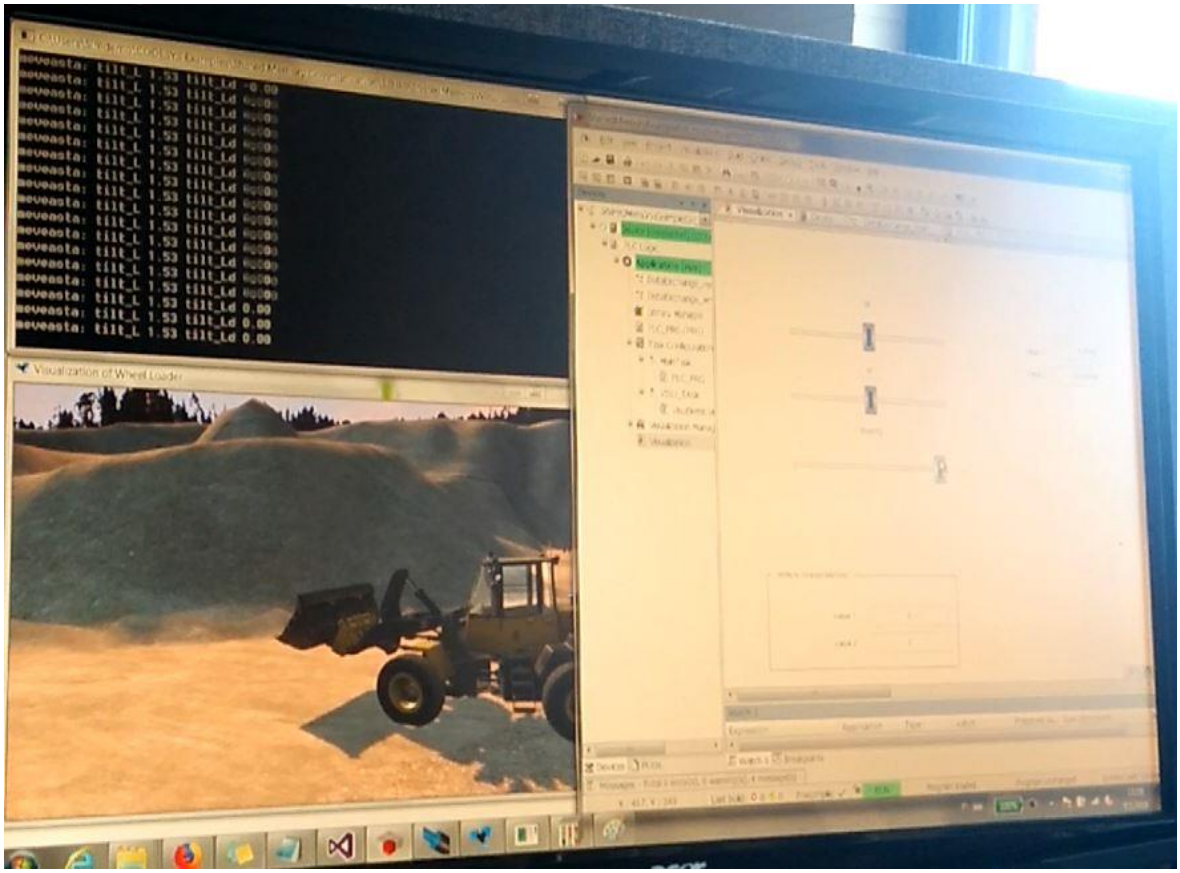
MeVEA tarjoaa esimerkkiohjelman (External Interface -example), jossa C++-ohjelma muodostaa Socket-yhteyden MeVEAan (MeVEA 2012). Tätä C++-ohjelmaa muokattiin niin, että se muodosti samalla kappaleessa 3.2 esitetyn jaetun muistin yhteyden Codesyksen välille. Näin saatiin aikaan reaaliaikainen yhteys Codesyksen ja MeVEAn välille. Toteutettuja tiedonsiirtomenetelmiä havainnollistetaan kuvassa 8 alhaalla.



Kuva 8. Tiedonsiirron menetelmät.

C++-ohjelma siis samanaikaisesti luo jaetun muistin yhteyden Codesyksen kanssa ja Socket-yhteyden MeVEAn kanssa. Samalla käytetään MeVEAssa simulaatiomallina pyöräkuormaajaa, joka ei ole osana MeVEAn tarjoamaa esimerkkiohjelmaa. Dataa lähetettiin tässäkin tapauksessa 0,2 sekunnin välein molempiin suuntiin.

Jaetun muistin menetelmällä ja Socket-yhteydellä onnistuttiin luomaan toimiva yhteysratkaisu, jossa data saatiin kulkemaan molempiin suuntiin. Alhaalla kuva 9 havainnollistaa testitapahtumaa, jossa MeVEAn simulaatiomallia ohjataan Codesyksen visuaalisesta käyttöliittymästä reaaliajassa.



Kuva 9. Testiajo jaetun muistin menetelmällä.

Kuvassa 9 vasemmalla alhaalla on simuloitu pyöräkuormaaja MeVEAssa. Vasemmalla ylhäällä pyörii C++-ohjelma, joka käsittelee tiedonsiirtoa MeVEAn ja Codesyksen välillä. Oikealla näkyy Codesyksen graafinen käyttöliittymä. Tässä testissä näkyvät kolme liukusäädinvipua ohjaavat työkoneen kauhan korkeuden, kauhan nostokulman kaltevuuden ja etupyörien kääntökulman muutosnopeutta. Nämä muutosnopeudet voivat olla positiivisia tai negatiivisia ja muutosnopeuden ollessa 0 valittu osa pysyy paikallaan. Mitä suurempi positiivinen tai negatiivinen arvo on sitä suurempi muutosnopeus. Positiivinen arvo tuottaa liikkeen yhteen suuntaan ja negatiivinen arvo vastakkaiseen suuntaan. Testissä siirretyt datatyypit olivat tarkkuudeltaan kahdeksan tavun kokoisia tarkempia liukulukuja (double). Nämä liukuluvut voivat saada arvoja n. $-1{,}8 * 10^{308}$ ja $1{,}8 * 10^{308}$ väliltä 15:n desimaalin tarkkuudella. Codesyksen puolelta datatyyppi oli LREAL ja C++-ohjelmassa double. Kaikki muutokset simulaatiomallissa tapahtuvat reaaliajassa ilman huomattavan suurta viivettä.

4 TULOSTEN ANALYSOINTI

Codesyksen ja C++-ohjelman välillä siirrettiin jaetun muistin menetelmällä ja UDP-yhteydellä onnistuneesta kaikkia testattuja datatyyppejä (bool, char, int ja double). Muitakin datatyyppejä voidaan todennäköisemmin siirtää helposti, kun vain asetetaan datatyypin lähetys- ja vastaanottotyypit vastaamaan toisiaan sekä datatyyppien koot oikean suuruisiksi. Alla on kerrottu tarkempia tulosten analyysejä kappaleen 3 tuloksista.

4.1 C++-ohjelman ja Codesyksen välillä käytetty UDP-yhteys

Tiedonsiirto C++-ohjelmalta Codesykselle UDP-yhteydellä ei todennäköisimmin onnistunut siksi, että viestin alussa olevista 20 luvusta, jonkun luvun tai joidenkin lukujen arvo oli virheellinen niin, ettei Codesyksen NVL-objekti tunnistanut viestiä. Hyvin todennäköisesti ainakin 20 luvun viestin 9:s luku (list identifier), tulisi muuttaa olemaan eri luku, kuin Codesyksen lähettävän NVL-objektin list idintifier -arvo on. Huomioitavaa on myös, että kun muokkaa C++-ohjelman Codesykselle lähettämän datapaketin alussa olevan merkkijonon arvoja, on myös Codesyksen vastaanottavan NVL-objektin GVL-tiedostoa muutettava vastaamaan muutoksia. Eli esimerkiksi, jos C++-ohjelman lähettävän merkkijonon 9:n merkin (list identifier -arvoon viittaava merkki) muuttaa 1:stä arvoon 2, on myös GVL-tiedostosta muutettava list identifier -arvo 1:stä arvoon 2.

UDP-yhteydellä voitaisiin tietoa myös siirtää eri tietokoneiden välillä, kun tietokoneet on kytketty samaan verkkoon. Näin voitaisiin käyttää toista tietokonetta, jossa olisi käynnissä Codesyksen visualisaatio simulaatiomallin ohjaimena sekä toista tietokonetta, jossa olisi MeVEAn simulaatiomalli, jota ohjataan. Tiedonsiirto voisi tapahtua, joko langattomasti tai langallisesti tietokoneiden välillä.

4.2 Yhteys MeVEAn ja Codesyksen välillä jaetun muistin menetelmällä ja Socket-yhteydellä

Tehdyt C++- ja Codesys-ohjelmat on helppo muokata käytettäväksi mihin tahansa MeVEAn simulaatiokoneeseen. Halutut input- ja output-arvot, jotka siirtyvät ohjelmilta toisille on helposti muokattavissa ja datatyypit tarvittaessa muutettavissa sekä input- ja output-arvoja

voidaan luoda rajattomasti. Tämän lisäksi Codesyksen visualisaatioon (kuvassa 9 oikealla) on helppo lisätä tai muuttaa nappuloita, vipuja ja näyttöjä tarpeen mukaan, Nämä ohjaimet saadaan helposti suoraan liitettyä MeVEAsta tuleviin tai MeVEAlle lähteviin arvoihin.

4.3 Kehitys- ja laajennusmahdollisuuksia

Työssä käytettiin C++-ohjelmaa Codesyksen ja MeVEAn välillä datansiirron käsittelyyn. Teoriassa olisi mahdollista myös luoda yhteys suoraan Codesyksestä MeVEAan ilman tätä C++-ohjelmaa. Tällöin ei käytettäisi kahta erillistä tiedonsiirtomenetelmää samanaikaisesti C++-ohjelman ja Codesyksen sekä C++-ohjelman ja MeVEAn välillä. Tällöin tulisi Codesyksen puolelta osata mallintaa ohjelma, joka osaisi yhdistää suoraan MeVEAn luomaan Socket-yhteyteen. Tämä olisi haastavampaa, koska Codesyksellä tulisi luoda yhteys käyttäen juuri samaa Socket-yhteysmenetelmää, kun MeVEAn esimerkkiratkaisu käyttää. Tähän tulisi löytää oikeat kirjastot Codesykseen ja ohjeistusta tämänkaltaisen yhteyden muodostamiseen on niukasti tarjolla. Suora yhteys MeVEAn ja Codesyksen välillä ilman C++-ohjelmaa olisi hieman suorituskykyisempi. Toisaalta vaikka Codesyksen ja MeVEAn väliin luodaan erillinen C++-ohjelma, tämä ohjelma ei todennäköisesti ole raskas eikä aiheuta huomattavaa viivettä.

Codesyksen graafinen käyttöliittymä voisi olla MeVEAn lisäksi mahdollista liittää muihin reaaliaikaisiin simulaatio-ohjelmiin pienillä muutoksilla C++-koodiin. Codesyksen ja C++-ohjelman välinen tiedonsiirto toteutuisi aivan samalla tavalla kuin MeVEAnkin kanssa, mutta tällöin tulisi vain saada reaaliaikainen yhteys toimimaan myös C++-ohjelman ja halutun simulaatio-ohjelman välillä.

5 JOHTOPÄÄTÖKSET JA YHTEENVETO

Kirjallisuusosiossa tutkittiin työkoneiden voimansiirtoa ja simulaatiota. Esitettiin lyhyesti muutama menetelmä simulaation matemaattiseen mallinnukseen sekä tietokonesimulaation periaatteita. Tämän lisäksi tutkittiin työkoneissa käytettäviä ohjausjärjestelmiä, niiden säätöön käytettävää PID-säädintä ja ohjausjärjestelmän simulaatiota. Työssä tuotiin myös esiin kasvava automaation ja etäohjauksen määrä työkoneissa. Tämä tekee myös voi usein hankaloittaa ohjausjärjestelmien suunnittelua tekemällä niistä entistä monimutkaisempia. Varsinkinkin automatisoiduissa työkoneissa myös itse ohjelmistojen toimivuudella on kriittinen rooli työkoneen oikeanlaiseen ja turvalliseen toimintaan. Näin myös esiteltiin lyhyesti muutama testimenetelmä ohjelmistopuolen testaamiseen. Esitettyjen White Box- ja Black Box-menetelmien ideologiaa voitaisiin periaatteessa soveltaa myös virtuaalisten simulointimallien ja ohjausjärjestelmien simulointimallien testaamiseen joissakin tapauksissa.

Työssä onnistuttiin muodostamaan reaaliaikainen tiedonsiirto Codesyksen ja MeVEAn välille C++-apuohjelman välityksellä. Tällöin C++-ohjelman välillä tiedonsiirto Codesykselle tapahtui jaetun muistin menetelmällä ja MeVEAlle käyttäen Socket-yhteyttä. Koska tiedonsiirto onnistuttiin lähettämään molempiin suuntiin kyseisellä menetelmällä, olisi myös mahdollista muodostaa suljetun silmukan kaltainen takaisinkytketty ohjausjärjestelmä, jota ohjataan Codesyksestä käsin reaaliajassa. Yhteysratkaisua pystytään luomaan joustavasti minkä tahansa MeVEAn reaaliaikaisen simulaatiomallin ohjaukseen ja tiedonsiirtoon.

Käyttäen UDP-yhteyttä Codesyksen ja MeVEAn välillä, onnistuttiin vain siirtämään dataa Codesyksestä C++-ohjelmalle. Todennäköisesti datan siirto Socket-yhteydellä C++-ohjelmalta MeVEAlle olisi toiminut samalla tavalla kuin jaetun muistin menetelmässä, vaikka dataa siirrettäisiinkin vain toiseen suuntaan. Tärkeintä on, että tiedonsiirto nimenomaan tapahtuu Codesyksestä MeVEAan, jotta ohjaus Codesyksestä käsin on mahdollinen. Näin ollen myös UDP-yhteydellä olisi ollut todennäköisesti helppoa luoda yksinkertainen reaaliaikainen ohjausjärjestelmä saavutetuilla tuloksilla. Tällöin

ohjausjärjestelmä olisi vain rajoittunut avoimen silmukan kaltaiseen ohjaukseen ilman takaisinkytkentää.

LÄHTEET

Alani, M. 2014. Guide to OSI and TCP/IP Models. SpringerBriefs in Computer Science. Springer International Publishing. 50 s.

Backas, J., Ahopelto, M., Huova, M., Vuohijoki, A., Karhu, O., Ghabcheloo, R. & Huhtala, K. 2011. IHA-Machine: A Future Mobile Machine. Proceedings of the 12th Scandinavian International Conference on Fluid Power. Tampere. Tampere University of Technology. 18-20.5.2011. S. 161–176.

Codesys. 2019. Shared Memory Communication [esimerkkiohjelmapaketti]. [viitattu 5.4.2019]. Saatavissa: https://store.codesys.com/shared-memory-communication.html

Dadhich, S., Bodin, U. & Andersson, U. 2016. Key challenges in automation of earth-moving machines. Automation in Construction 68. s. 212–222.

Digital Hydraulics. 2016. S. 1–13. [Valmetin www-sivuilla]. [Viitattu 15.8.2019]. Saatavissa: https://www.valmet.com/globalassets/media/downloads/white-papers/process-improvements-and-parts/wpp_digihydraulics.pdf

El-Rewini, H. & Abd-El-Barr, M. 2004. Shared Memory Architecture. Advanced Computer Architecture and Parallel Processing. John Wiley & Sons. s. 77–102.

Forrai, A. 2013. Embedded Control System Design: A Model Based Approach. Springer-Verlag Berlin Heidelber. 264 s.

Frank, B., Kleinert, J. & Filla, R. 2018. Optimal control of wheel loader actuators in gravel applications. Automation in Construction 91. s. 1–14.

Hartmann, S. 1996. The World as a Process. In: Heglesmann, R., Muller, U. & Troitzsch, K.G. Modelling and Simulation in the Social Sciences from the Philosophy of Science Point of View. Theory and Decision Library. Vol 23, s. 77–100.

Huang, H., Hartmann, M., Awad, H.F. & Knetsch, D. 2018. Virtual Functional Testing of a Mechatronic Active Roll Control. 7 s.

Immonen, P. 2013. Energy Efficiency of a Diesel-Electric Mobile Working Machine. 138 s.

Kamien, M. I. & Schwartz, N. L. 1991. Dynamic Optimization: The Calculus of Variations and Optimal Control in Economics and Management. Elsevier Science, 2nd edition. 396 s.

Kelloniemi, T. 2016. Istutuspään sähköjärjestelmän suunnittelu ja toteutus. Diplomityö. 87 s.

Kulakowski, B., Gardner, J., & Shearer, L. 2007. Dynamic Modeling and Control of Engineering Systems (3rd Edition). Cambridge University Press. 502 s.

Limaye, M.G. 2009. Software Testing: Principles, Techniques and Tools. Tata McGraw-Hill Education. 523 s.

Manring, N. D. 2005. Hydraulic Control Systems. John Wiley & Sons. 426 s.

MeVEA. 2012. Eternal Interface -example. 7 s.

Ritzke, J. & Aschemann, H. 2011. Design and Experimental Validation of Nonlinear Trajectory Control of a Drive Chain with Hydrostatic Transmission. Proceedings of the 12th Scandinavian International Conference on Fluid Power. Tampere. Tampere University of Technology. 18–20.5.2011. S. 49–63.