VAASAN YLIOPISTO

TEKNILLINEN TIEDEKUNTA

AUTOMAATIOTEKNIIKKA

Mika Hanhila

PID-SÄÄTIMEN OPTIMOINTI DIFFERENTIAALIEVOLUUTIOLLA

Diplomityö, joka on jätetty tarkastettavaksi diplomi-insinöörin tutkintoa varten
Vaasassa 15.05.2015.

Työn valvoja Professori Jarmo Alander

Työn ohjaaja Professori Timo Mantere

VAASAN YLIOPISTO
Teknillinen tiedekunta
Tekijä: Mika Hanhila
Diplomityön nimi: PID-säätimen optimointi differentiaali-evoluutiolla
Valvojan nimi: Professori Jarmo Alander
Ohjaajan nimi: Professori Timo Mantere
Tutkinto: Diplomi-insinööri
Yksikkö: Sähkö- ja energiatekniikan yksikkö
Koulutusohjelma: Sähkö- ja energiatekniikan koulutusohjelma
Suunta: Automaatiotekniikka
Opintojen aloitusvuosi: 2011
Diplomityön valmistumisvuosi: 2015 **Sivumäärä:** 91

TIIVISTELMÄ

Differentiaalievoluutio on uusi optimointimenetelmä, joka soveltuu erinomaisesti PID-säätimen parametrien numeeriseen optimointiin yksinkertaisuutensa ja reaali-aikaisuutensa vuoksi. Tässä tutkimuksessa PID-säädintä optimoidaan FPGA:lla. FPGA:lla ei ole aiemmin toteutettu tällaista differentiaalievoluutiomenetelmään perustuvaa PID-säädintä. Alkuperäistä differentiaalievoluutioalgoritmia parannetaan ranking-perusteisella mutaatio-operaatiolla ja itse-adaptoituvilla mutaatio- ja risteytys-parametreilla. Ranking-perusteisella mutaatio-operaatiolla pystytään parantamaan ratkaisun optimoinnin onnistumistodennäköisyyttä, laatua ja suppenemisnopeutta. Lisäksi itse-adaptoituvien ohjausparametrien ansiosta käyttäjän ei tarvitse arvioida mutaatio- ja risteytysparametrien arvoja.

FPGA:lla optimoidaan kustannusfunktiota, joka koostuu eroarvosta, näytteenottovälistä ja derivointitermistä. Hyvyysarvo lasketaan yritevektorin parametreista sukupolvittain. Hyvyyslaskennan perusteella valitaan parhaat yriteparametrit optimiparametreiksi. PID-säädinohjelma on ensin testattu Modelsimilla ja tämän jälkeen kyseiset testaustulokset on analysoitu Matlabilla. Tulosten perusteella ohjelman eri osa-alueita voidaan tulevaisuudessa kehittää sekä laajentamalla ja monimutkaistamalla satunnaisuutta että kasvattamalla yksilöiden ja sukupolvien määrää. Edellä mainituilla toimenpiteillä vaikutettaisiin satunnaislukujen määrään ja toistuvuuteen ja lisättäisiin mutaatio-laskennan monimuotoisuutta.

AVAINSANAT: Differentiaalievoluutioalgoritmi, FPGA, itse-adaptoituvat ohjaus-parametrit, hyvyysarvo, kustannusfunktio, ranking perusteinen mutaatio-operaatio, PID-säädin, numeerinen optimointi

UNIVERSITY OF VAASA
Faculty of Technology
Author: Mika Hanhila
Topic of the Thesis: Optimization of PID-controller by differential evolution
Supervisor: Professor Jarmo Alander
Instructor: Professor Timo Mantere
Degree: Master of Science in Technology
Degree Programme: Degree Programme in Electrical and Energy Engineering
Department: Department of Electrical Engineering and Energy Technology
Major of Subject: Automation Technology
Year of Entering the University: 2011
Year of Completing the Thesis: 2015 **Pages:** 91

ABSTRACT

Differential evolution algorithm is a new optimization method, which is ideally suited for numerical optimization for the PID controller parameters due to its simplicity and real-time nature. PID controller optimization is implemented in FPGA, for which there is only marginally experience in controller optimizations. The original differential evolution algorithm was improved by a ranking-based mutation operation and self-adaptation of mutation and crossover parameters. Ranking-based mutation operation improves the quality of solution, convergence rate and success of optimization. In addition, due to the self-adaptive control parameters the user doesn't have to estimate the mutation and crossover rate values.

The cost function consisting of the delta value, sampling value and derivation term is the optimized fitness by FPGA. Fitness value is calculated for each generation trial parameters. According to fitness value the best control parameters is selected for PID-controller. PID-controller has been tested using the ModelSim program at first and then the test results have been analyzed with Matlab. Based on the results of different aspects of the program can be developed in the future as well as expanding and complicated by randomness that increasing the number of individuals and generations.

KEYWORDS: Differential evolution algorithm, FPGA, self-adaptive control parameters, fitness value, cost function, ranking-based mutation operation, PID-control, numerical optimization

SISÄLLYSLUETTELO sivu

KUVALUETTELO sivu

TAULUKKOLUETTELO sivu

SYMBOLIT JA LYHENTEET

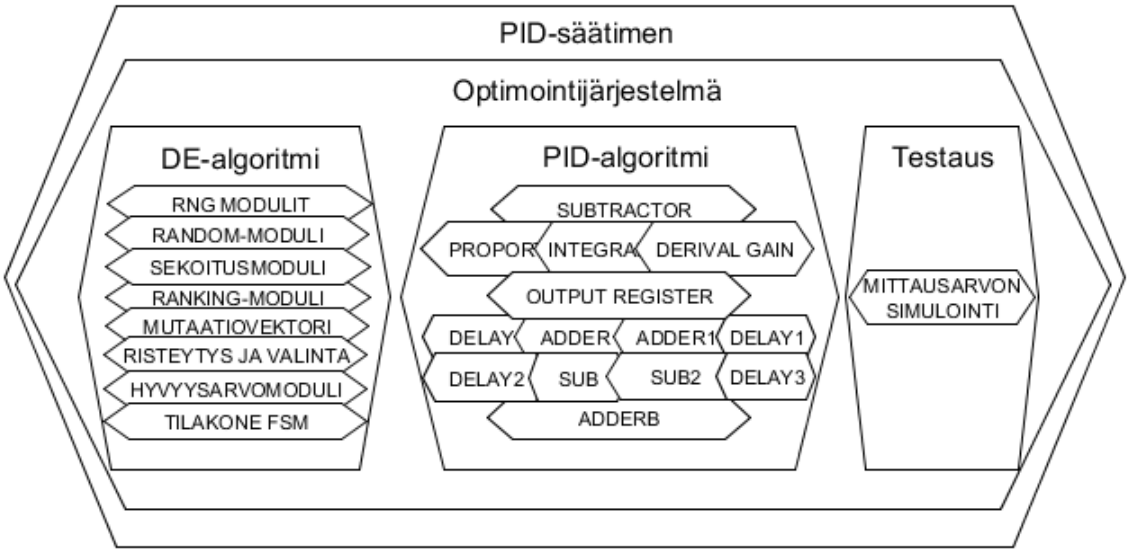
a	Tangentin y-akselin leikkauskohdan parametri askelvastemenetelmässä
a_i	*a:t* ovat ISE-, ISTE- ja IST^{2}E-kriteerien kertoimia
b_i	*b:t* ovat ISE-, ISTE- ja IST^{2}E-kriteerien kertoimia
c	Kerroin, joka ilmaisee variaatio-operaatioiden vaikutuksen populaation monimuotoisuuteen
clk	kellopulssi
CR,Cr	Risteytysvakio (crossover constant)
Cr_ulos	Risteytysvakion siirtoarvo
CRi,G+1	Uusi ohjausparametri risteytysvakiolle
D	Yksilövektorin parametrien eli kromosomien määrä (dimension)
Data	Hyvyysarvon laskennassa käytetty hyvyyssignaalin arvona
DE	Differentiaalievoluutio (Differential evolution)
DEGL	Differential Evolution with Global and Local Neighborhoods
dout	Sekoitusmodulin lakennan välisignaalia
e	identtisten yritevektoreiden prosenttiosuus koko suorituksen aikana generoiduista yritevektoreista
EA	Evoluutioalgoritmi (Evolutionary Algorithm)
ER-arvo	PID-säätimen erotusarvo
F	Mutaatiovakio (mutation constant)
Fi,G+1	Uusi ohjausparametri mutaatiokertoimelle
F_l	Mutaatioalueen minimiarvo
F_μ	Mutaatioalueen maximiarvo
FP	Liukuluku (floating point)
FPGA	Field Programmable Gate Array
FPN	Liukulukuesitys (floating point notation)
FSM	Ohjauksen tilakonemodulit
F_ulos	Mutaatiovakion siirtoarvo
$f(\mathrm{Pop_j})$	Hyvyysfunktio (fitness function)
G	Sukupolvi (generation)
$G_{\max}$	Maksimi sukupolvien määrä (maximum number of generations)
h	PID-säätimen näytteenottoväli
IAE	Integraalin absoluuttinen virhe
IIR	Ääretön impulssivaste (infinite inpulse response)
ISE	Integraalin neliöllinen virhe

ISTE	Ajan ja virheen tulon neliöllinen virhe
ITSE	Aikaintegraalin ja neliöllisen virheen tulo
JADE	Adaptive Differential Evolution
jDE	Self-adaptive Differential Evolution of Random Matching-based
K1,K2	Askelheräte ja -vaste
K_{krit}	Kriittinen vahvistuskerroin
K_{qt}	Neljännes vahvistuskerroin
Kp	PID-säätimen vahvistuskerroin (Propotional gain)
Ki,Ti	PID-säätimen integrointikerroin (Integral gain)
Kd,Td	PID-säätimen derivointikerroin (Derivative gain)
L	Kuollutaika PID-säätimen viiveellisessä järjestelmässä (dead time)
LSFR	Lineaarinen siirtorekisteri (linear feedback shift register)
MEMORY	Ohjelmamuisti
MODULI	Sisältää ohjelman osan
MSE	Keskineliövirhe (mean squared error)
MV	PID-säätimen mittausarvo
NP	Populaation koko (population size)
$N(\mu,\sigma)$	Normaalijakautunut satunnaismuuttuja
o	Identtisten yritteiden määrä nykyisen ja yritepopulaation välillä
p1,p2,p3	parametri yksi, kaksi ja kolme
p1_rank	Ranking-vektorin ensimmäinen parametri
p2_rank	Ranking-vektorin toinen parametri
p3_rank	Ranking-vektorin satunnaisesti valittu kolmas parametri
PID	Propotional Integral Derivative
Pop_{ij}	Populaation kohdevektori
p_s	Onnistuneiden mutaatioiden prosenttiosuus
pr1,pr2	Ranking-arkkitehtuurin valintatodennäköisyyksiä
P-termi	Diskreetti vahvistustermi
I-termi	Diskreetti integrointiaika
D-termi	Diskreetti derivointiaika
q1,…,q12	Sekoitusmodulin sekoitussignaalia
R	Kulmakerroin
rand [0,1]	Tasaisesti jakautunut satunnaismuuttuja välillä [0,1]
rand	Yleisesti tekstissä käytetty lyhenne satunnaisarvosta (random)
Random	Tuottaa alustetun satunnaisparametrin yhdellä kellopulssilla 32:ta satunnaislukugeneraattorilla alustetusta parametrista

rank	Yleisesti tekstissä käytetty lyhenne suuruusjärjestysvertailulle (ranking)
Reset	Ohjelman nollaus
Ref	PID-säätimen ohjearvo
RNG	Satunnaislukugeneraattori (random number generator)
RNG-FSM	RNG-modulien sisäistä toimintaa ohjaava tilakone
r1,r2,r3	Ranking-arkkitehtuurissa laskettujavektori-indeksejä
SaDE	Simulated Annealing Differential Evolution
Sub	Digitaalisen PID-säätimen vähennysmoduli (Subtractor)
T	Aikavakio ja nousuaika
t	Identtisten yritteiden määrä yritepopulaation sisällä
T_{krit}	Värähtelyrajataajuus
temp	Yleisesti ohjelmoinnissa käytetty tilapäinen väliarvo
T_{qt}	Neljännes värähtelyjakson aika
τ_1, τ_2	Vertailuarvot mutaatio- ja risteytysvakion suhteelliseen säätöön
$U_{ij,G+1}$	Yritevektorin *i*:s parametri eli kromosomi (chromosome)
$U_{j,G+1}$	Kohinavektori, jossa vektori on luotu yksilön yksi ja kaksi painotetusta erotusarvosta ja lisäämällä siihen kolmas populaation yksilö
U0,…,U12	Sekoitusmodulin alustussignaalia
u1_mutvek	Mutaatiovektorin parametri yksi mutaatiovektorimodulin ohjelmassa
u2_mutvek	Mutaatiovektorin parametri kaksi mutaatiovektomodulin ohjelmassa
u3_mutvek	Mutaatiovektorin parametri kolme mutaatiovektorimodulin ohjelmassa
valinta_s2	PID-säätimen optimointinopeuden valintakytkin
valinta_s3	PID-säätimen optimointinopeuden valintakytkin
valinta_sw	PID-säätimen optimoinnin käynnistys kytkin
VHDL	Laitteistokuvauskieli (VHSIC Hardware Description Language)
VHSIC	Erittäin nopea integroitu piiri (Very High Speed Integrated Circuit)
X_G	Mutaatiovektori
X_{G+1}	Uuden sukupolven mutaatiovektori
X*i,G*	Sukupolven *G* populaation *i*:s yksilö
$X_{i,}G+1$	Uuden sukupolven G+1 populaation i:s yksilö
$X_{i,j,G}$	Yksilön *i*:s parametri eli kromosomi (chromosome)
Y1,…,Y4	Ranking-arkkitehtuurin vertailtavat yksilöt
Z(-1)	Signaalia viivästetään yhdellä kellopulssilla
μ	Normaalijakauman odotusarvo
σ	Normaalijakauman keskihajonta

1. JOHDANTO

Älykkäitä (PID) säätimiä toteutetaan nykyään myös erilaisilla optimointimenetelmillä. PID-säädin on kehittynyt vaiheittain mekaanispneumaattisesta järjestelmästä mikroprosessori pohjaiseen järjestelmiin ja nykyään ovat yleistyneet digitaaliset Field Programmable Gate Array (FPGA) ohjausjärjestelmät. Säädin on käytössä erilaisissa avaruus, prosessinohjaus, tuotteiden valmistus-, robotiikan ja autoteollisuuden järjestelmissä (Chan, Moallem and Wang 2006: 1).

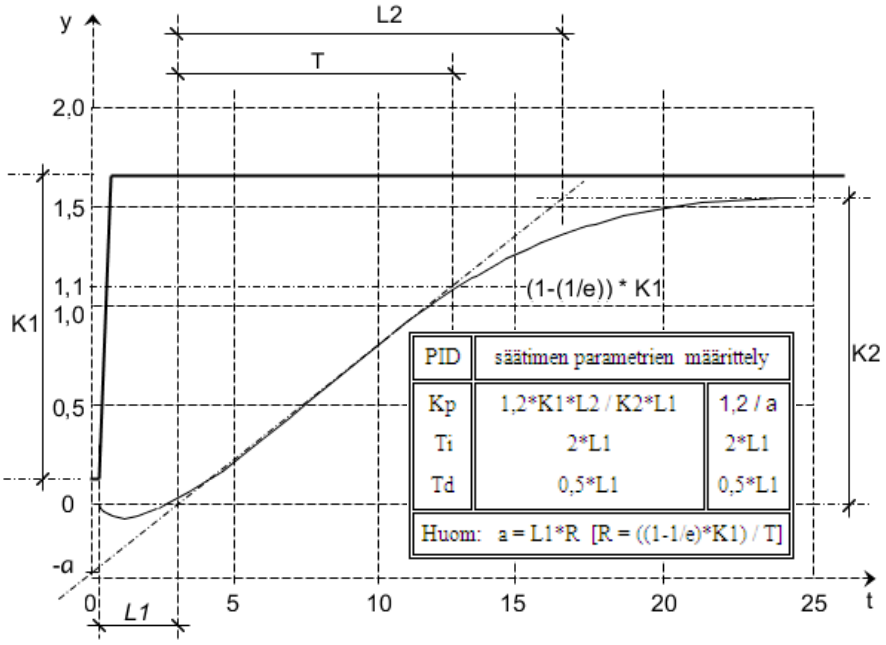


Kuva 1. PID-säätimen optimointijärjestelmän yleiskaavio (mukailtu Grout 2008: 517-520) .

FPGA:lla on tarkoitus toteuttaa evoluutiolaskentaan perustuva älykäs digitaalinen PID-säädin (kuva 1). Differentiaalievoluutio (DE) algoritmi on yksi uusimmista evoluutiota jäljittelevistä menetelmistä (Chakraborty 2008). Differentiaali Evoluutioalgoritmi valittiin PID-säätimen optimointitehtävään, koska se on esitykseltään yksinkertainen ja sen toiminta on reaaliaikaista verrattuna muihin evoluutioalgoritmeihin (EA) (Anumandla, Peesapati, Sabat, Udgata and Abraham 2013: 1). Optimointia on sovellettu onnistuneesti keinotekoisiin ja todellisiin optimointiongelmiin kuten aerodynaamisten muotojen, koneteknisten suunnitelmien optimointiin (Ilonen, Kämäräinen and Lampinen 2003: 93-94), liikkeen ennustamiseen, napojen paikan suunnitteluun äärettömän impulssivasteen suodattimissa (IIR) tulevaisuuden kehittyviin koneisiin (Anumandla ym. 2013: 1). Optimointi soveltuu myös erinomaisesti PID-säätimen numeeristen parametrien tarkkaan minimointiin. FPGA-ohjelmalla pystytään lisäämään PID-säätimen tehokkuutta, monipuolistaa toiminnallisuutta, reaaliaikaisuutta, vakautta

ja minimoimaan tehon kulutusta (Chan ym. 2006: 1). Kiinteän pilkun lukujen ja liukulukujen (FP) käyttö FPGA-ohjelmassa kuitenkin vaikeuttaa suunnittelua, koska monissa sovelluksissa tarvitaan muunnosten tekemiseen erilaisia FPGA-laitteistorakenteita, joiden käyttö vaatii paljon kokemusta (Anumandla ym. 2013). Kiinteän pilkun arkkitehtuuri on edullisempi ja nopeampi kuin liukulukuesitys (FPN), mutta kokonaislukujen ja desimaaliosien laskenta vaatii kuitenkin aikaa kuluttavia toimintoja FPGA:ssa. (Lima, Menotti, Cardoso and Marques 2006: 2.) Lisäksi FPGA:ssa kiinteänpilkunlaskenta voidaan korvata skaalaamalla liukuluvut kokonaisluvuiksi. Skaalaamalla laskeminen on yksinkertaista ja nopeaa yhteen- ja vähennyslaskuissa sekä kerto- ja jakolaskuissa kun kertominen ja jakaminen tapahtuu bittien siirrolla kahden potensseina. (Alander 2015.)

PID-säätimen virityksessä voidaan käyttää sovelluskohtaisia viritysmenetelmiä. Yleensä diskreetti PID-säädin vaatii lyhyen näytteenottovälin, jolloin se vastaa jatkuva-aikaisen PID-säätimen toimintaa. PID-säätimellä näytteenottoväli riippuu derivointiajasta. Näytteenottotaajuuden ollessa matala virityksessä täytyy käyttää prosessimallia (Aalto yliopisto 2011b: 4). Yleisimmät kokeelliset PID-säätimen viritysmenetelmät ovat askelvaste-, värähtelyrajamenetelmä sekä nyrkkisääntöihin ja numeeriseen viritykseen perustuva optimointi (Aalto yliopisto 2011b: 1-5). Lisäksi on käytetty optimiasetuskriteeriä (Dingyu 2007: 205-212) ja ekstrapoloivaa viritysmenetelmää (Niiranen 1999).



Kuva 2. Askelvastekokeen määrittelyparametrit avoimessa säätöpiirissä (Aalto yliopisto 2014: 1-14.)

Askelvastemenetelmässä askelvaste määritellään säätämällä kokeellisesti askelvasteen jyrkimmän kohdan tangetin x-akselin leikkauskohdan kuollut aika *L1* ja y-akselin leikkauskohdan parametri *a* kun säätimen takaisinkytkentä on avoin. Tangetin kulmakerroin *R* saadaan, kun tiedetään askelherätesignaalin *K1* ja aikavakion *T* suuruus. Aikavakio *T* määritetään ajanhetkestä, jossa askelvaste *K2* saavuttaa arvon joka on 63.2 % lopullisesta arvostaan. Y-akselin leikkauskohdan mitoitusparametri *a* lasketaan kuolleen ajan L1 ja tangetin kulmakertoimen R tulosta. Säätimen parametrit sijoitetaan taulukkoon (kuva 2), josta lasketaan PID-säätimen parametrit (Aalto yliopisto 2014: 1-14.)

Värähtelyrajamenetelmä perustuu avoimeen takaisinkytkentään, jossa kasvatetaan kokeellisesti ensin vaiheittain vahvistuskerrointa niin kauan, että PID-säätimen erotusarvo alkaa värähdellä (Aalto yliopisto 2011b: 1-2). PID-säätimen parametrit lasketaan tämän jälkeen kriittisen vahvistuskertoimen (K_{krit}) ja värähtelyrajataajuuden (T_{krit}) avulla taulukosta. Parametrien asetusten jälkeen takaisinkytkentäsilmukka suljetaan. Pelkkä kokeellinen menetelmä vaatii ohjelman tekijältä PID-säätimen parametrien käyttäytymisen tuntemusta. Numeerisella optimoinnilla minimoidaan eroarvosta riippuvaa kustannusarvoa, jotka lasketaan eri tavalla painotetuista kustannusfunktioista (Aalto yliopisto 2011b: 1-5). Kustannusfunktioita ovat integraalin absoluuttinen virhe (IAE), integraalin neliöllinen virhe (ISE), aikaintegraalin ja neliöllisen virheen tulo (ITSE) sekä ajan ja virheen tulon neliöllinen virhe (ISTE). Näillä voi minimoida erilaisia fysikaalisia suureita. Optimointiasetuskriteerissä optimi-teoreettisille ISE-, ISTE, IST^2E kertoimille a_i ja b_i on asetettu arvot sovitettujen käyrien perusteella ja laskettu säädön optimi kuolleenajan *L* ja nousuajan *T* suhteella sekä PID-säätimen ohjearvon *k* avulla (Dingyu 2007: 205-212). Ekstrapoloiva viritysmenetelmä on kehitetty, koska usein ei pystytä ajamaan piiriä värähtelyrajalle ja on koetettu ekstrapoloimalla löytää kriittinen vahvistusarvo ja sen avulla optimoida PID-säädin (Niiranen 1999). Kyseisissä virityksissä PID-säädin viritetään kokeellisesti, kun ei tunneta järjestelmän mallia. Nämä viritysmenetelmät sopivat hyvin evoluutiopohjaiseen säätimen viritykseen. (Aalto yliopisto 2011b: 1-4.)

PID-säätimen parametrien optimoinnin laatu riippuu differentiaalievoluutiolaskennan tarkkuudesta (Rönkkönen 2003: 37-40) ja minkä tyyppisellä tehokkaalla adaptoituvalla differentiaalievoluutio-optimoinnilla päästään parhaaseen mahdolliseen optimiin ja mahdollisimman nopeasti (Rönkkönen 2003: 8-16). Adaptoituvia differentiaali-evoluutiolaskennan tuloksia voidaan verrata standardiin laskentaan. Häviötehojen ja

energian kulutuksen vähentämiseksi sekä laitteistoresurssien säästämiseksi tarvitaan sanan pituuden analysointitekniikoita. Kertojapiirien jakaminen on tarpeellista, jos FPGA-piirillä ei ole riittävästi kertojia pitkille sananpituuksille kun ohjelmoidaan toiminnaltaan laajoja järjestelmiä. Tälläisissä tapauksissa, jossa FPGA-piirin kapasiteetti on lähes kokonaan käytetty voidaan käyttää apuna liukulukujen skaalaamista. (Lima ym. 2006: 6.)

1.1. Tutkimustavoitteet

Tavoitteena on kehittää differentiaalievoluutiolla optimoitava PID-säädin. Ohjelma on tarkoitus toteutetaan Altera DE2 kehitys- ja koulutusalustalla, jossa ohjelmointikielenä käytetään erittäin nopeaa laitteistokuvauskieltä (VHDL). Ohjelmoinnin esitys toteutetaan kiinteän pilkun (8,7) tai (16,15) bitin esityksenä ja skaalaamalla liukuluvut kokonaisluvuiksi (Alander 2015). Esityksen parametrien tarkkuudeksi saadaan näin ollen n. kahden tai neljän desimaalin tarkkuus. PID-säätimen optimoinnin toteutuksessa tarkoituksena on optimoida differentiaalievoluutiolla itse adaptoituvia ohjausparametriä (Self-Adapting Control Parameters in Differential Evolution), jossa mutaatiokerrointa (F) ja risteytysvakiota (CR) säädetään mutaatioalueen minimi (F_l) ja mutaatioalueen maximi (F_μ) arvojen sekä vertailuarvojen (τ_1) ja (τ_2) avulla. Ohjelma ei vaadi suurta muutosta verrattuna alkuperäiseen differentiaalievoluutioon, jossa käyttäjä säätää ohjausparametrit manuaalisesti (Brest, Greiner, Bošković and Mernik 2005: 649).

Differentiaalievoluutioalgoritmi ja PID-säädin on tarkoitus ohjelmassa jakaa kahteen arkkitehtuuriin, jossa ne on yhdistetty toisiinsa FPGA-rakenteessa. Differentiaali-evoluutioalgoritmissa kohdevektorille suunnitellaan aloitusarvo, jota optimoidaan minimoimalla arvoa sukupolvittain ja vertailemalla sitä yritevektoriin (Saad, Sadli, Jamaluddin & Darus 2012: 143-144). Kohdevektori sisältää kolme kappaletta kromosomeja, jotka ovat PID-säätimen vahvistuskerroin (K_p), integraalikerroin (T_i) ja derivointikerroin (T_d) parametrit. Differentiaalievoluutioarkkitehtuurissa optimoidut arvot syötetään PID-säätimen ohjausjärjestelmään populaatioittain, jolloin säätimen parametrit päivittyvät optimoinnin aikana (Saad ym. 2012: 139-144). Optimointi lopetetaan, kun vasteaika PID-säätimen ohjausjärjestelmässä on optimaalinen, jolloin sukupolvien lukumäärä määräytyy ohje-arvon ja olo-arvon erotuksen mukaan. Optimaalisin arvo voidaan laskea myös sukupolvien määrän mukaan, jossa sukupolvien määräksi ohjelmassa asetetaan optimointikertojen määrä.

Viritysmenetelmänä PID-säätimen optimoinnissa käytetään numeerista menetelmää, jossa jokaiselle populaation yksilölle lasketaan hyvyysarvo PID-säätimen erotusarvosta ISE- tai IAE-menetelmällä (Aalto yliopisto 2011b: 1-5). PID-säätimen parametrit valitaan vertailemalla kohdevektorin ja yritevektorin arvoja. Ohjelman testaus on aluksi tarkoitus testata modulikohtaisesti FPGA:n omalla kehitysalustalla, ModelSimilla simuloimalla ja Matlabilla tuloksia analysoimalla.

1.2. Työn rakenne

Kirjallisuustutkimuksessa luvussa 2 tutustutaan optimointiin, säätimen rakenteeseen ja toimintaan sekä FPGA esitykseen kiinteän pilkun luvuilla. Differentiaali-evoluutioalgoritmin selvityksessä käytetyimpiä lähteitä olivat Kenneth V. Prizen ym. kirja ”Differential Evolution A Practical Approach to Global Optimization” (Price ym. 2005), jossa tutustuttiin alkuperäisen differentiaalievoluution toimintaan ja rakenteeseen, Janes Brest:in ym. artikkeli ”Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems” (Brest ym. 2006a), jota on käytetty tämän työn optimoinnin perustana, Wenyin Gong:in artikkeli “Differential Evolution with Ranking-based Mutation Operators” (Gong ym. 2013a), jolla parannetaan alkuperäisen differentiaalievoluutioalgoritmin toimintaa ja nopeutta sekä Janes Brest:in ym. artikkeli ”Performance comparison of self-adaptive and adaptive differential evolution algorithms” (Brest ym. 2006b), jolla pidetään ratkaisuun sidotut yritevektorin rajat käyttökelpoisina ongelmaan nähden. PID-säädin algoritmissa käytetyimpiä artikkeleja olivat Mohd Sadli Saad:in ”PID Controller Tuning Using Evolutionary Algorithms” (Saad ym. 2012), jota on käytetty tämän työn säädön perustana, Aki Penttisen artikkeli ”FPGA:lle sulautetulla mikroprosessorilla toteutettu sähkökäytön säätöjärjestelmä” (Penttinen 2005), jossa on tutkittu kiinteän pilkun esitystä ja Aalto yliopiston artikkeli ”PID-säätimen kokeellinen virittäminen” (Aalto yliopisto 2011b), jota on käytetty säätimen virityksen perustana. Luvussa 3 käsitellään PID-säätimen uusia itse-adaptoituvia ohjausparametreja, säätimen rakennetta ja PID-säätimen optimoinnin kulkua. Luvuissa 4, 5 ja 6 tehdään PID-säätimen toteutusta FPGA:lla ja testataan sen ohjelmaa Modelsimilla ja lopuksi testaustulokset analysoidaan Matlab:lla. Luvussa 7 analysoidaan työn eri vaiheet ja tehdään niistä johtopäätökset.

2. KIRJALLISUUSKATSAUS

Luvussa tutustutaan differentiaalievoluutiolaskennalla toteutettuun PID-säätimen toimintaperiaatteeseen. Teoriaosuudessa käydään läpi erilaisia differentiaali-evoluutiototeutuksia, jotka sopivat PID-säätimen optimointiin. Kirjallisuuden avulla perehdytään alkuperäiseen differentiaalievoluutioalgoritmiin ja sen ohjelmalliseen rakenteeseen sekä erilaisiin parametrien säätömenetelmiin. PID-säätimessä tutustutaan säätimen rakenteeseen, digitaaliseen kokonaislukutoteutukseen kiinteän pilkun luvuilla ja kokonaislukuja skaalaamalla sekä säätimen virittämismenetelmiin.

2.1. Differentiaalievoluutio

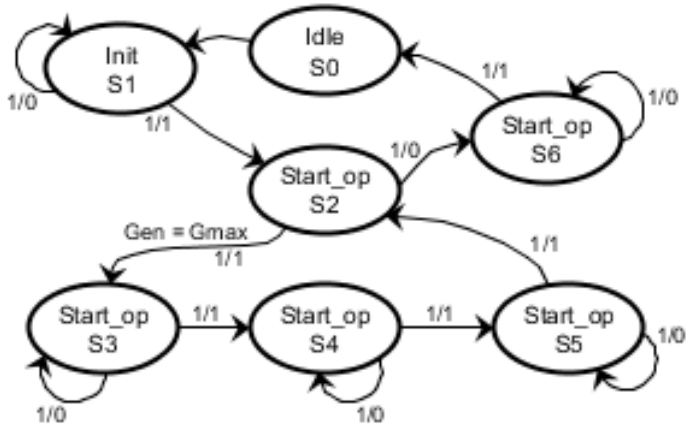
Differentiaalievoluutiolaskenta on yksinkertainen toteuttaa siten, että toteutus vastaa selkeästi FPGA-ohjelman rakennetta. Näennäissatunnaisilla muutoksilla differentiaali-evoluutiolaskenta löytää optimaalisimmat arvot parametreille. Konvergenssin ollessa liian nopeaa voidaan DE-laskennassa juuttua lokaaliin optimiin. Tällöin differentiaali-evoluutioalgoritmin parametreja voidaan optimoida (Rönkkönen 2003: 8-16), jolloin nopeutetaan ja tarkennetaan globaalin optimin löytymistä eli pyritään välttämään juuttuminen paikalliseen optimiin. Seuraavassa alaluvussa tutustutaan lähemmin differentiaalievoluution toimintaan, rakenteeseen ja laskenta-algoritmeihin.

2.1.1. Differentiaalievoluution rakenne

Julkaisussa Kiran Kumar Anumandla ym. (2013) ovat tutkineet skaalattavaa rinnakkais-prosessoria nopeuttamaan differentiaalievoluutiolaskentaa. Differentiaalievoluutio-moduli on tiukasti yhdistetty hyvyysfunktiomoduuliin mikä vähensi viestintä- ja ohjauskustannuksia. Julkaisussa on tutkittu sellaista differentiaalievoluutiolaskennan laitteistoarkkitehtuuria, jossa on seitsemän modulia. Arkkitehtuuriin kuuluu muistin alustus, mutaatio, risteytys, valinta, satunnaislukugeneraattori, hyvyyden evaluointi ja ohjauksen tilakonemodulit (FSM).

FSM-moduli sisältää valmius- (idle state), alustus- (initialization state), toiminta-(operation state), odotus- (wait state) ja lukutilat (reading state) (kuva 3). Valmiustilassa kaikki FSM-modulin tilat ovat resetoituna. FSM-modulilla ohjataan kaikkien modulien eli risteytys-, mutaatio- ja valintamodulien toimintaa algoritmin eri vaiheissa. FSM-

moduli on odotustilassa kunnes sen hetkinen vaihe on suoritettu. Lukutilassa FSM lukee hyvyysarvon ja tallentaa sen rekisteriin.



Kuva 3. DE-algoritmin FSM-diagrammi (Anumandla ym. 2013: 6.)

FSM-modulin ollessa alustustilassa alustusmodulin (Initialization Module) populaation kromosomit on satunnaisesti synnytetty määrätylle alueelle ja talletettu populaatiomuistiin (Population Memory). Populaation muuttujan kromosomien koko on 32 bittiä. NP ja D maksimiarvoiksi on asetettu 32. Populaation kromosomien arvot syötetään hyvyyden arviointimodulille (Fitness Evaluation Module), josta arvot tallennetaan hyvyys muistille (Fitness Memory).

Mutaatiomoduli siirtyy toimintatilaan, kun FSM-modulin tila muuttuu alustustilasta toimintatilaan. Mutaatiovektorin tuotetaan jokaisesta populaation kohdevektorista. Vertailijayksiköllä kertojalta ja laskurilta saatuja arvoja vertaillaan, ja sen mukaan lasketaan kolme erilaista vektori-indeksiä *r1*, *r2*, *r3*. Indeksibittien mukaan annetaan kolmelle populaation kohdevektorille laskentajärjestys mutaatio-operaatiossa, joiden avulla lasketaan mutaatiovektorin arvo. Jokainen populaation kohdevektori syötetään ensin oikeaan laskentarekisteriin indeksibittien mukaisesti, josta kohdevektorit syötetään laskentaan yhtä aikaisesti. Laskennasta saadaan mutaatiovektorin arvo.

Riteytysmodulille syötetään mutaatiovektori ja populaation kohdevektori tilakoneen risteytystilan ollessa toimintatilassa. Risteytysmodulilla pääosin kasvatetaan yritevektorin monimuotoisuutta risteytysvakion (CR) ohjaamana. Monimuotoisuuden kasvaessa paikalliseen optimiin juuttuminen vähenee. Tällä tavalla ainakin yhdeksi yritevektorin kromosomiksi saadaan sukupolvittain mutanttivektori. Risteytys perustuu kahteen vertailijaan, jossa ensimmäisessä vertailijassa vertaillaan populaation

yksilölaskurin ja kertojan arvoja. Kertojalla kerrotaan satunnaisluku (RNG_num) ja populaation kromosomin indeksi. Vertailijalla kaksi verrataan risteytysvakiota (CR) ja rekisterin yksi arvoa (Reg 1). Rekisterille yksi on tallennettuna satunnaisluku. Vertailijoiden bittien yhteisarvon mukaan valitaan multiplekseriyksiköllä joko mutaatiovektori tai populaation kohdevektori yritevektoriin.

Risteytysmodulilta yritevektori lähetetään valintamodulille ja lasketaan valintamodulilla yritteellle hyvyysarvo. Hyvyysarvon laskennasta yritevektori ja sen hyvyysarvo syötetään hyvyysrekisteriin. Valinta perustuu kohdevektorin hyvyysrekisterissä ja yritevektorin hyvyysrekisterissä olevien arvojen vertailuun. Yritevektorin ja kohdevektorin hyvyysarvovertailijan tuloksen mukaan multiplekserilla valitaan joko yritevektori tai kohdevektori parhaaksi yksilöksi jatkamaan sukua. Yritevektori valitaa seuraavaan sukupolveen, jos sen hyvyysrekisterissä oleva hyvyysarvo on pienempi kuin hyvyysrekisterissä ollut kohdevektorin arvo. Muuten valitaan populaation tämän hetkinen kohdevektori parhaaksi populaation jäseneksi. Toiminta jatkuu kunnes on käyty läpi maksimi määrä sukupolvia.

RNG modulilla on suuri merkitys DE-operaatioon. Työssä on käytettu LSFR-siirtorekisteriä synnyttämään satunnaislukuja. LSFR on yksinkertainen ja tuottaa hyviä satunnaislukuja. Moduli tuottaa satunnaislukuja alustus-, valinta-, risteytys- ja mutaatio-moduleille. Arkkitehtuurissa sekvenssin maksimi pituus on 2^{32}-1 bittiä. Moduli synnyttää satunnaislukuja, jossa ei esiinny arvoa nolla.

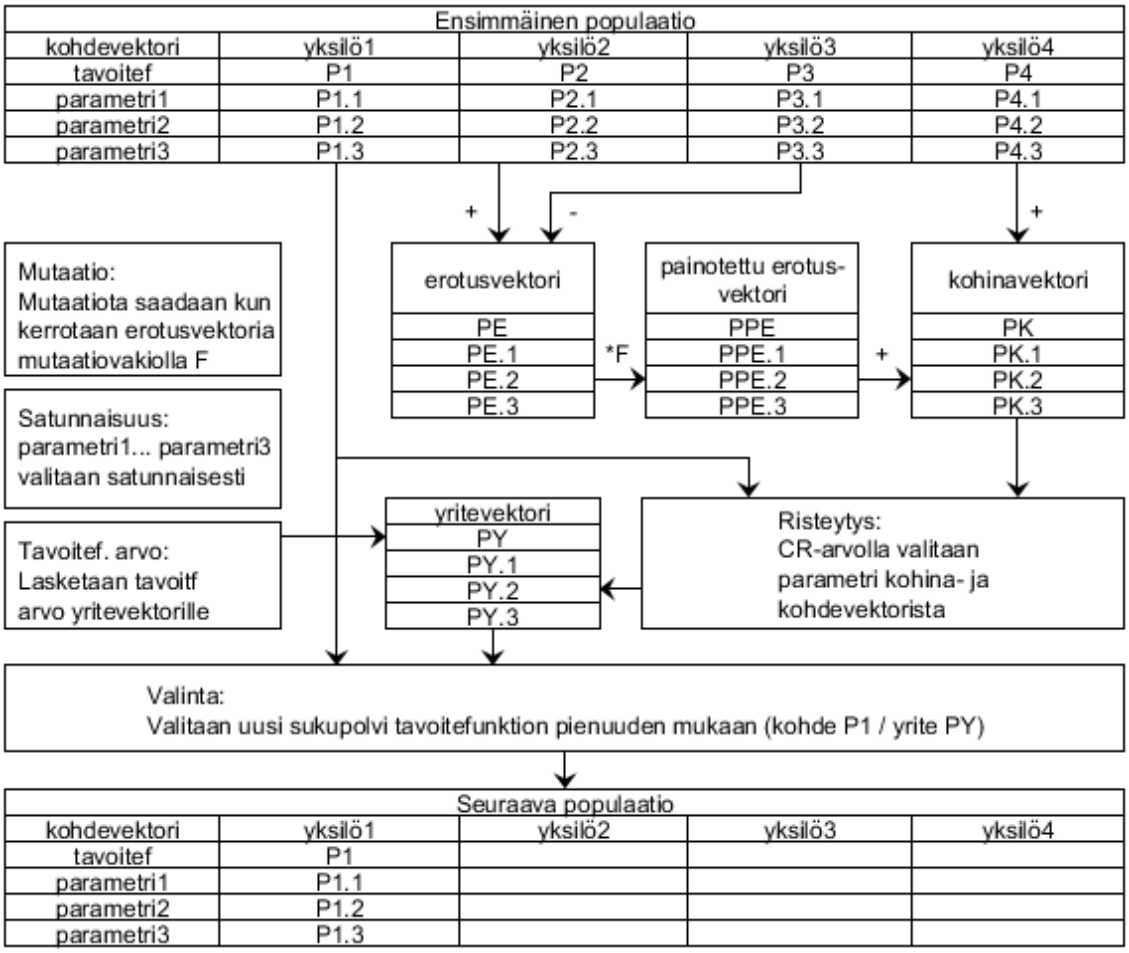
2.1.2. Alkuperäinen DE-toteutus

Kenneth V. Price ym. (2005) ovat tutkineet käytännössä erilaisia optimointi-menetelmiä. Optimointimenetelmiltä vaaditaan yksinkertaista toimintaa, helppoa käyttöä, varmuutta ja reaaliaikaisuutta. DE-optimointia on käytetty korkean teknologian tuotteiden suunnittelussa kuten digitaalisten suodattimien suunnittelussa sekä mikro- ja optoelektroniikan mittauksissa. DE-laskenta on tehokas ja yksinkertainen numeerisen optimoinnin menetelmä.

Alkuperäinen (*DE*/*rand*/*bin*) menetelmä kuuluu perinteisiin suoriin etsintämenetelmiin (kuva 4). Se on populaatioperustainen optimointimenetelmä, jossa optimointi tapahtuu näytteistämällä kohdefunktio usealla eri satunnaisesti valitulla alustuspisteellä. Populaation rakenne tarvitsee kontrolliparametrit populaation koon (*NP*), risteytys-

vakion (*CR*), mutaatiovakion (*F*) ja kromosomien lukumäärän (*D*) sovittamiseksi. Populaatio alustetaan generoimalla satunnaisesti yksilöiden 1,…,*NP* kromosomit tasaisesti välille 1,…,*D* optimointikohteesta riippuen. Yksilölle yksi annetaan kromosomien lähtöarvot manuaalisesti, jonka jälkeen yksilön yksi kromosomien paikalle valitaan kohde- tai yritevektori tavoitefunktion arvon perusteella sukupolvittain. Yritevektoria lähdetään muodostamaan satunnaisesti generoiduista yksilön kaksi ja yksilön kolme kohdevektoreista, joista muodostetaan erotusvektori eli differentiaali. Erotusvektorista muodostetaan painotettu erotusvektori, jossa erotusvektoria kerrotaan mutaatiovakiolla. Painotetun erotusvektorin kromosomit voivat saada myös negatiivisia arvoja. Mutaatiovakiolla vaikutetaan mutaation askelpituuteen. Differentiaalin eli erotusvektorin pienentyessä mutaation askelpituus pienenee populaation pienentyessä. Painotetun erotusvektorin ja yksilön neljä kohdevektorin kromosomit lasketaan parametreittain yhteen, josta syntyy kohinavektori eli mutaatiovektori. Risteytyksessä mutaatiovektorin ja yksilön yksi kohdevektorin kromosomeista valitaan risteytysvakion CR mukaan joko mutaatiovektorin tai kohdevektorin yksi kromosomin yritevektoriksi. Yritevektorin kromosomin ylittäessä risteytysvakion CR arvon säilytetään kohdevektorin yksi kromosomi. Risteytysvakio CR voidaan valita väliltä [0,1]. Mutaatiovektorista valitaan kuitenkin aina vähintään yksi kromosomi. Liian suuri CR arvo johtaa liian nopeaan suppenemiseen eli lokaaliin optimiin. Uuden sukupolven valinta kohdevektoriksi yksi tehdään yritevektorin ja kohdevektorin tavoitefunktion pienuuden perusteella eli kohdevektoriksi valitaan pienemmän tavoitefunktion arvon mukainen vektori. Jos vektorit ovat yhtäsuuria valitaan yritevektori jatkamaan sukua uuteen populaatioon.

Ohjausparametrien valinnassa on ehdotettu hyväksi aloitusvaihtoehdoksi alkuperäistä *DE*/*rand*/1/*bin* DE-optimointia kun optimointi aikaa ja vaativuustasoa ei tiedetä. Alkuperäinen *DE*/*rand*/1/*bin* DE-optimointi on myös todistettu tehokkaaksi kun optimointiparametreilla on vähäinen parametrien riippuvuus. Yleensä alkuperäisessä differentiaalievoluutiossa risteytysvakion oletusarvona on pidetty $CR = 0{,}2\ldots1{,}0$ ja parhaana aloitusarvona mutaatiokertoimelle pidetään $F = 0{,}3\ldots1{,}0$. Risteytysvakion *CR* kasvaessa yleensä kasvatetaan myös mutaatiokerrointa ja populaation kokoa samassa suhteessa. Tällä tavalla ylläpidetään monimuotoisuutta ja estetään ennenaikaista suppenemista. Hyvänä matalana aloitusarvona populaation koolle pidetään yksinkertaisilla funktiolla $NP = 5*D*CR$ ja hyvin monihuippuisilla parametririippuvilla funktioilla $NP = 10*D$. *DE*-algoritmin onnistuminen pienillä populaatioilla ja pienillä *CR*-arvoilla osoittaa testifunktioiden olevan erottuvia.



Kuva 4. Alkuperäinen differentiaalievoluutioalgoritmi (*DE*/*rand*/1/*bin*) (Lampinen 2001.)

2.1.3. Normaalijakautuneet ohjausparametrit

Greenwood ja Zhu (2001) ja Rönkkönen (2003) ovat tutkineet empiirisesti uutta normaalijakaumaan perustuvaa mutaatio-operaatiopohjaista differentiaalievoluutio-algoritmia. Algoritmin oletetaan vähentävän entisestään ennenaikaista suppenemista eli lokaaliin optimiin juuttumista. Normaalijakaumaan perustuva *DE*-algoritmi on teoreettisesti osoitettavissa suppenevaksi. Tutkimuksessa käytetään 1/5-onnistumis-sääntöä, jossa mutaatio on muotoa $X_{G+1} = X_G + N(0, \sigma)\ X_G$. *DE*-algoritmin kontrolliparametrien määrä vähenee, koska mutaatiokerrointa ei enää tarvita. Mutaatiovektorin laskemiseksi tarvitaan normaalijakautunut satunnaismuuttuja $N(0,\sigma)$, odotusarvo μ=0 ja keskihajonta σ sekä kerrointa c. Tutkimuksessa on määritetty keskihajonnalle σ yleiskäyttöinen arvo ja kertoimelle c oli valittu väliltä [1,0;1,5], koska

näiden rajojen ulkopuolella olevat arvot ovat joko harvoin hyödyllisiä tai jopa käyttökelvottomia. Populaation koko ei vaikuta normaalijakaumaan, kun käytetään kertoimen *c* arvona 1,5. Kertoimen arvon ollessa lähellä alarajaa niin, silloin ennenaikaisen suppenemisen riski kasvaa ja kertoimen arvon ollessa lähellä ylärajaa suoritus ei ole optimaalinen. Tässä tapauksessa algoritmi toimii luotettavasti suurella joukolla erilaisia ongelmia. Risteytysvakioksi *CR* oli valittu käyttökokemuksen perusteella 0,9. Uudella normaalijakautumaan perustuvalla mutaatio-operaatiolla vaikutetaan askelpituuteen ja sen suuntaan, joka johtuu erotusvektorin jokaisen kromosomin kertomisesta eri luvulla. Normaalijakaumaan perustuva mutaatiovektori löytää aina optimin, jos optimointiaikaa ja -tarkkuutta on riittävästi.

Populaation koon ollessa 10**D* tai suurempi hyvin multimodaalisilla ja parametri-riippuvilla funktioilla on hyvä olla säädettävissä oleva suppeneminen. Ensin lasketaan mutaation prosenttiosuus (P_s) noin 10*D* funktiolle, jossa käydään läpi koko yrite ja nykyinen populaatio. Mutaatio onnistuu, kun tavoitefunktion arvo on parempi kuin vanhempi tavoitefunktion arvo. Onnistuneiden mutaatioiden prosenttiosuuden ollessa suurempi tai pienempi kuin viidesosa kaikista mutaatioista niin, keskihajonta-arvo jaetaan tai kerrotaan luvulla 0,85. Keskihajonta pysyy samana jos onnistuneiden mutaatioiden prosenttiosuus on viidesosan kaikista mutaatioista. Seuraavassa laskukaava normaalijakauman keskihajonnan σ muutoksen laskemiseksi:

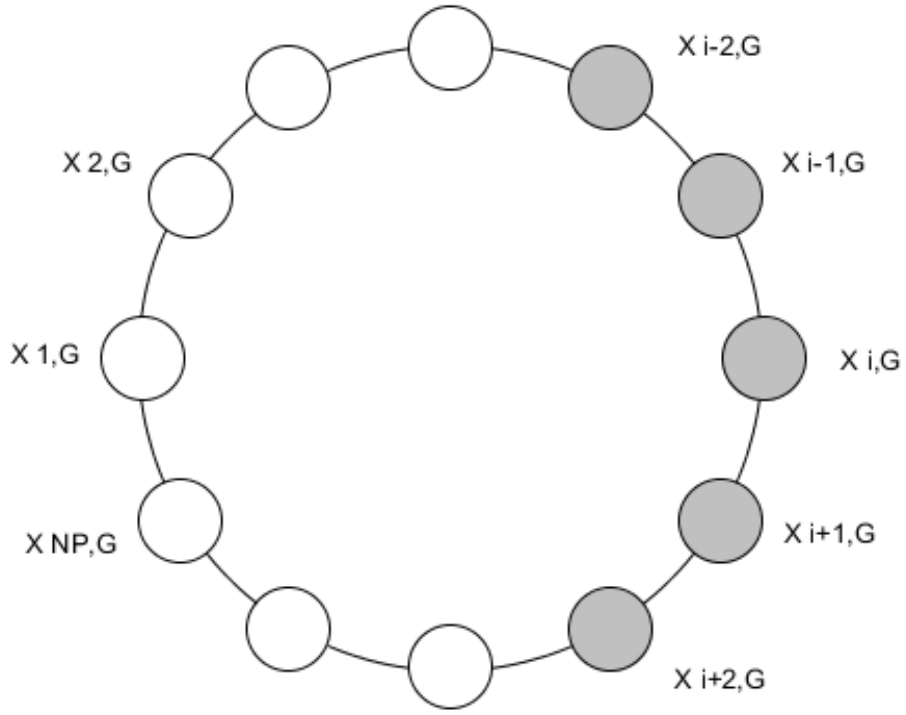
$$\sigma = \begin{cases} \sigma / 0{,}85 & \text{jos } p_s > 1/5 \\ \sigma \times 0{,}85 & \text{jos } p_s < 1/5 \\ \sigma & \text{jos } p_s = 1/5 \end{cases} \quad (1)$$

2.1.4. Paikallinen ja kansallinen mutaatiomalli (DEGL)

Swagatam, Abraham, Chakraborty and Konar (2009) ovat tutkineet DEGL (DE with global and local neigborhoods) perusteisen DE-algoritmin mutaatiota. Järjestelmä tasapainoittaa etsintää ja tehostaa populaatiopohjaista satunnaista DE-algoritmeihin perustuvaa etsintätekniikkaan. DEGL käyttää binomiaalista risteytysjärjestelmää. Tutkimuksen yhteydessä on tutkittu kahdenlaista differentiaalievoluution naapuruusmallia. Toista kutsutaan paikalliseksi naapuruusmalliksi (local neighborhood model) ja toista kansalliseksi naapuruus malliksi (global neighborhood model). Paikallisessa naapuruusmallissa kaikki vektorit on mutatoitu käyttämällä parasta asemaa

sen toiseksi pienimmän naapuruston löytämiseksi osittaisessa eli paikallisessa populaatiossa. Kansallisessa mutaatiomallissa on laskettu kansallisesti paras vektori $X_{best,G}$ sen hetkisen sukupolven koko populaatiosta mutatoimalla populaation jäsenet.

Naapuruusperusteinen DE-populaatio on $P_G = [X_{1,G}, X_{2,G},\ldots,X_{NP,G}]$, jonka jokaisen kohdevektorin kromosomi $X_{i,G}$ ($i = 1,2,\ldots,NP$) on D-mittainen. Alustuksessa vektorin indeksit on järjestelty satunnaisesti, jolloin naapuruuden monimuotoisuus säilyy. Jokaiselle kohdevektorin $X_{i,G}$ vektorille määritetään k säteinen naapuruusarvo, joka on nollasta poikkeava kokonaisluku ($k = 0,1,\ldots,(NP\text{-}1) / 2$). Naapuruusarvon koko pitää olla pienempi kuin populaation koko, esimerkiksi $2k + 1 \leq NP$. Naapuruus sisältää seuraavat kohdevektorit $Xi\text{-}k_{,G},\ldots, X_{i,G},\ldots,X_{i+k,G}$. Kodevektorit on järjestelty rengas-topologiaan (ring topology) indeksien suhteen (kuva 5). Naapuruustopologiassa kohdevektorin indeksit on järjestelty satunnaisesti. DE-algoritmissa populaation koko on yleensä suurempi, jolloin rengastopologia antaa paremman esityksen kuin muut merkittävät naapuruusrakennemallit.



Kuva 5. DE-naapuruuden rengastopologia (Swagatam ym. 2009: 531)

Populaation jäsenistä paikalliseen mutaatiovektoriin $L_{i,G}$ on valittu paras vektori $X_{n\_besti,G}$ naapuruudessa. Seuraavassa mutaatiovektorin kaava:

$$L_{i,G} = X_{i,G} + \alpha \times (X_{n\_besti,G} - X_{i,G}) + \beta \times (X_{p,G} - X_{q,G}), \quad (2)$$

jossa alaindeksi $n_{\_besti}$ kuvaa parasta populaation kohdevektorin $X_{i,G}$ naapuruutta ja p, q alaindeksit kuuluvat joukkoon $[p, q] = [i-k, i+k]$, jossa $p \neq q \neq i$. Kansallinen mutaatio-vektori on luotu vastaavalla kaavalla:

$$g_{i,G} = X_{i,G} + \alpha \times (X_{g\_best,G} - X_{i,G}) + \beta \times (X_{r1,G} - X_{r2,G}), \quad (3)$$

jossa alaindeksi g_best,G kuvaa parasta kohdevektoria koko populaation sukupolvessa ja alaindeksit $r1$, $r2$ kuuluu joukkoon $[1, NP]$, jossa $r1 \neq r2 \neq$ i. Indeksit α ja β ovat skaalauskertoimia, joista termi α on aritmeettinen uudelleenyhdistämisoperaatio ja termi β on differentiaalimutaatio paikallisessa ja kansallisessa mutaatiomallissa. Indeksien α ja β arvona voidaan pitää $\alpha = \beta = F$, joilla on sama rooli kuin mutaatiovakiolla F. Paikallinen ja kansallinen mutaatiomalli ei ole puhdas mutaatio-operaatio. Todellinen mutaatiovektori muodostetaan paikallisesta ja kansallisesta mutaatiomallista sekä skalaarista painotuskertoimesta $\omega \in [0,1]$ lineaarisen interpolaatiokaavan avulla:

$$V_{i,G} = \omega \times g_{i,G} + (1 - \omega) \times L_{i,G}. \quad (4)$$

DEGL-versiossa käytetään neljää uutta parametria, jotka ovat α, β, ω ja naapuruussäde k. Suurin vaikutus optimoinnissa on painotuskertoimella ω, joka ohjaa tasapainoa tutkimis- ja hyväksikäyttöominaisuuksien välillä. Painotuskertoimen ω arvon ollessa yksi ja sen lisäksi $\alpha = \beta = F$ ollessa yhtäsuuria, mutaatiovektorin mutaatio-operaatioksi tulee ”DE/target-to_best/1”. Kyseinen mutaatio-operaatio valitaan yllä olevien parametriarvojen perusteella. Painotuskertoimen ollessa lähellä ykköstä suositaan kansallista muunnosta, joka parantaa DEGL-version hyväksikäyttöä. Edellinen mutaatio-operaatio on erityinen tapaus DEGL-optimoinnissa. Tasapainoisimmassa DEGL-versiossa ω arvon pitäisi olla noin 0,5. Tälläisessä tasapainoitetussa versiossa ei saada erityistapauksissa täyttä hyötyä painokertoimen normaalijakauman ollessa yksihuippuinen tai kupera. Painokerrointa voidaan korjata kohti nollaa tai ykköstä esityksen parantamiseksi.

Painokerroin voidaan valita ja optimoida adaptiivisesti kolmella eri järjestelmällä, joista ensimmäinen on kasvava painokerroin (Increasing Weight Factor). Tässä painokerroin kasvaa nollasta kohti ykköstä algoritmin suorituksen aikana, joko lineaarisesti (linear increment) tai exponenttiaalisesti (exponential increment). Lineaarisesti kasvava

painokerroin kasvaa tasaisesti ja exponenttiaalinen painokerroin kasvaa aluksi hitaasti ja lopuksi nopeammin algoritmin edetessä. Seuraavaksi kasvavien painokertoimien kaavat:

tasaisesti kasvava: $\omega_G = G / G_{\max}$, (5)

exponentiaalisesti kasvava: $\omega_G = \exp(G / G_{\max} \times \ln(2)) - 1$, (6)

jossa G tarkoittaa sukupolvien lukumäärä, $G_{\max}$ on suurin sukupolvien määrä ja ω_G on painokerroin sukupolvessa G.

Satunnaisessa painotuskertoimessa (Random Weight Factor) painokerroin vektorit $\omega_{i,G}$ toteutetaan tasaisesti jakautuvilla satunnaisluvuilla rand[0,1]. Tämä vaihtoehto laskee optimoinnin lähestymisnopeutta ja kasvattaa monimuotoisuutta.

Itse-adaptoituvissa painotuskertoimissa (Self-Adaptive Weight Factor) jokaisella vektorilla on oma kohdevektorin $X_{i,G}$ painokerroin $\omega_{i,G}$. Alustuksessa painokertoimelle $\omega_{i,G}$, $\omega_{r1,G}$ ja $\omega_{r2,G}$ annetaan satunnaisesti arvo väliltä [0,1]. Tämän jälkeen yritepainokerroin $\omega'_{i,G}$ lasketaan seuraavalla lausekkeella:

$$\omega'_{i,G} = \omega_{i,G} + F \times (\omega_{g\_best,G} - \omega_{i,G}) + F \times (\omega_{r1,G} - \omega_{r2,G}), \quad (7)$$

jossa $\omega_{g\_best,G}$ on painokerroin, joka liittyy parhaaseen koko populaation kansalliseen kohdevektoriin $X_{g\_best,G}$. Adaptoitu yritevektorin painokerroin $\omega'_{i,G}$ rajoitetaan ohjelmallisesti alueelle [0,05;0,95]. Adaptoitu painokerroin $\omega'_{i,G}$ on optimoinnin jälkeen sijoitettu uuden mutaatiovektorin $V_{i,G}$ arvoksi kaavaan (4) . Yritevektori $U_{i,G}$ valitaan binomiristeytyksellä, jossa verrataan kohdevektoria $X_{i,G}$ ja mutaatiovektoria $V_{i,G}$ parametrittain ja valitaan vertailun perusteella uusi yritevektori. Lopullinen valinta tehdään yritevektorin ja kohdevektorin välillä hyvyysarvon suuruuden perusteella kuten alkuperäisessä DE-algoritmissa. Äskettäin laskettu painokerroin $\omega'_{i,G}$ vaikuttaa seuraavaan sukupolveen ainoastaan, jos yritevektorin $U_{i,G}$ hyvyysarvo on pienempi tai yhtäsuuri kuin kohdevektorin $X_{i,G}$ hyvyysarvo. Muuten valitaan vanha yritevektorin painokerroin $\omega_{i,G}$.

2.1.5. Itseadaptoituvat ohjausparametrit (jDE)

Brest ym. (2006a) ovat kuvanneet ja tutkineet uusia tehokkaita itseadaptoituvia ohjausparametreja. Menetelmää on käytetty optimoimaan ohjausparametrit, jotka vaativat moninkertaista optimointia. Tässä tutkimuksessa säädetään differentiaali-evoluution avulla ohjausparametreja *F* ja *CR*. Molemmat toimivat yksilötasolla, jossa paras yksilö selviytyy ja levittää parempia parametriarvoja. Vektori sisältää tietyn määrän kromosomeja, jotka on esitetty vektorissa $X_{i,G}$, i = 1,2…,*NP*. Uudet ohjausparametrit on laskettu seuraavissa kaavoissa:

$$F_{\mathrm{i,\,G+1}} = \begin{cases} F_l + \mathrm{rand}_1 \times F_\mu & \text{jos } \mathrm{rand}_2 < \tau_1 \\ F_{\mathrm{i,\,G}}, & \text{muuten} \end{cases} \tag{8}$$

$$CR_{\mathrm{i,\,G+1}} = \begin{cases} \mathrm{rand}_{3,} & \text{jos } \mathrm{rand}_4 < \tau_2 \\ CR_{\mathrm{i,\,G}}, & \text{muuten} \end{cases} \tag{9}$$

Satunnaisluvuissa $\mathrm{rand}_j \in \{1,2,3,4\}$, jossa yhdenmukaiset satunnaisluvut ovat $\in [0,1]$. Mutaatiokerrointa ja risteytysvakiota säädetään mutaatioalueen minimi (F_l) ja maximi (F_μ) arvojen sekä vertailuarvojen (τ_1) ja (τ_2) avulla. τ_1 ja τ_2 arvoille on asetettu kokemuspohjaisesti arvoiksi 0.1. Siksi F_l = 0,1 ja F_μ = 0,9, jolloin uusi *F* arvo valitaan satunnaisesti välillä [0,1;1,0] ja uusi *CR* arvo on välillä [0,1]. Tällöin arvolla F_l = 0,1 estetään ennenaikainen suppeneminen. Uudet $F_{\mathrm{i,G+1}}$ ja $CR_{\mathrm{i,G+1}}$ vaikuttavat uuden vektorin $X_{i,G+1}$ mutaatioon, risteytykseen ja valintaan.

Arvojen τ_1 ja τ_2 avulla valitaan tietyllä todennäköisyydellä paremmat ohjausparametrit uusille parametreille. Arvot τ_1 ja τ_2 valitaan sovelluksen toiminnan mukaan. Tutkimuksen mukaan hyviä *F* ja *CR* arvoja ei tarvitse keksiä. Itse adaptoituvan järjestelmän säännöt ovat yksinkertaiset, jolloin ne eivät kasvata aikavaatimusta alkuperäiseen DE-algoritmiin verrattuna.

2.1.6. jDE-2 algoritmi

Qin ja Suganthan (2005) ovat esitelleet uuden itseadaptoituvan jDE-2 algoritmin, joka käyttää kahta DE-strategiaa. DE-strategiat ovat “DE/rand/1/bin” ja “DE/current to *best*/1/*bin*”. Edellä mainittuja DE-stategioita on sovellettu populaation yksilöihin.

Uudessa jDE-2 algoritmissa molemmille strategioille lasketaan ohjausparametrit. Strategioista ensimmäinen ”rand/1/bin” usein antaa hyvän monimuotoisuuden ja strategioista toinen ”current to best/1/bin” antaa hyvän lähentymisen. JDE-2 algoritmi sijoittaa *k* kappaletta huonoimpia yksilöitä määrätylle sukupolvivälille *l*, jossa parametriarvot on jaettu tasaisesti ala- ja ylärajan välille ilman *k* yksilön arviointia.

2.1.7. Ranking-perusteinen differentiaalievoluutio

Gong ym. (2013a) ovat ehdottaneet ranking-perusteista mutaatio-operaattoria DE-optimointiin. He ovat vertailleet ranking-perusteista DE-algoritmia jDE-algoritmiin. Ranking-perusteista DE-laskentaa voidaan käyttää parantamaan tulevien yksilöiden valintaa sukupolven ranking-järjestyksen mukaan. Ranking-perusteinen DE-algoritmi on yksi mahdollinen tapa parantaa alkuperäisen DE-algoritmin toimintaa ja nopeutta. Sen toiminta perustuu populaation järjestämiseen ja todennäköisyyden laskentaan.

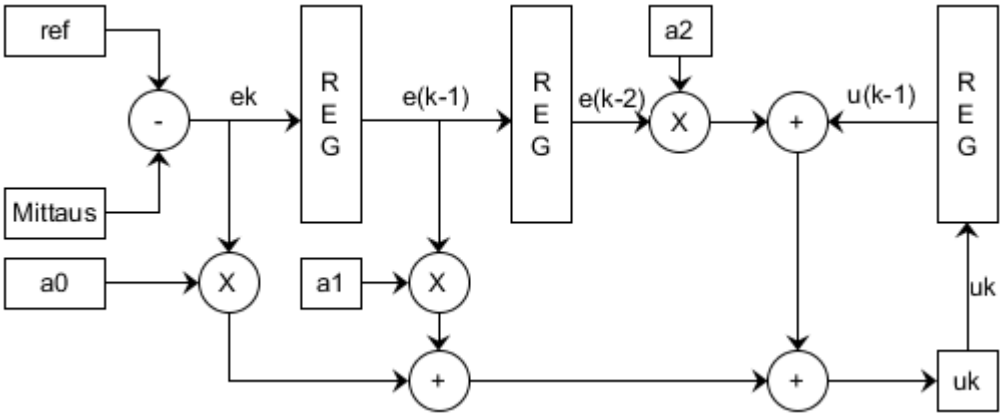
Gong ym. (2013b) empiirisessä tutkimuksessa on tutkittu protoninvaihtokennojen parametrien tunnistusta eri polttokennojen malleissa ranking-perusteinen differentiaali-evoluution avulla, koska ranging-perusteinen DE-algoritmi ei kasvata monimutkaisuutta merkittävästi. Ranking-perusteinen mutaatio-operaattori on yhdistetty viiden erilaisten DE-version kanssa. Nämä DE-versiot jDE, SaDE, JADE, CoDE ja DEGL ratkaisevat parametrien tunnistusongelmat protoninvaihtokennoissa. Verrattuna alkuperäiseen DE-menetelmään ranking-perusteinen DE antaa numeeristen tulosten mukaan paremmat tulokset, lähestymisnopeuden ja onnistumisprosentin. Ranging-perusteisella DE-algoritmilla saadaan pienempi standardipoikkeama kuin sellaisella versiolla, jossa ei ole käytetty ranging-perusteista DE-algoritmia. Edellinen kohta todistaa ranking-perusteisen mutaatio-operaattorin parantavan häiriösietoisuutta alkuperäiseen DE-algoritmiin verrattuna. Tutkimuksen mukaan rank-jDE lähestyy optimia n. 47% nopeammin kuin jDE ja lisäksi sen onnistumisprosentti on korkeampi.

2.2. PID-säädin

Tutustutaan diskreetin PID-säätimen toimintaan, rakenteeseen, kiinteän pilkun esitykseen ja virittämiseen. DE löytää heuristisesti optimaalisimmat PID-säätimen parametrit. PID-säätimen ohjelmalla on oma rakenne, johon tuodaan DE-rakenteesta kolme parametria Kp, Ti ja Td optimoituna. PID-säätimen järjestelmä voidaan optimoida reaaliaikaisille ja viiveellisille prosesseille. PID-säätimen näytteenottoväli voidaan säätää näytteenottovälin (h) ja derivointikertoimen (Td) avulla. Säätimen virittämiseen on erilaisia menetelmiä, joista muutamia käydään läpi tässä luvussa. Seuraavissa alaluvuissa tutustutaan edellä mainittuihin digitaalisen PID-säätimen toimintoihin.

2.2.1. Digitaalisen PID-säätimen arkkitehtuuri

Trimeche ym. (1996) ovat tutkineet digitaalisen FPGA:lla toteutetun PID-säätimen arkkitehtuuria. FPGA:lla toteutettu PID-säädin parantaa selvästi säätimen esitystä, koska ohjelmalla pystytään laskemaan tehokkaasti monimutkaisia reaaliaikaisia prosesseja. Lisäksi FPGA:n ohjelmallinen rakenne minimoi kustannuksia.

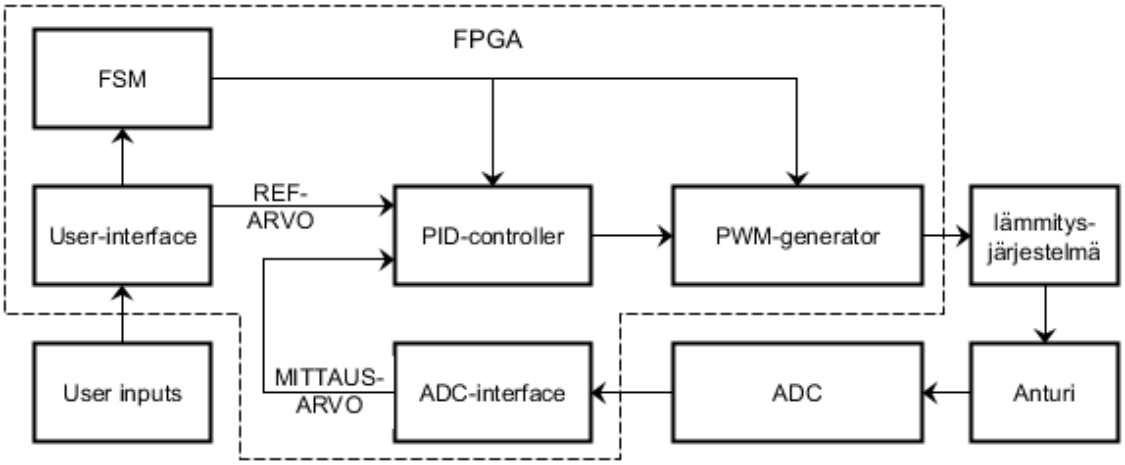


Kuva 6. Digitaalisen PID-säädön arkkitehtuuri (Lima ym. 2006: 2)

PID-säädin arkkitehtuuri sisältää kolme kertojaa, yhden vähentimen, kolme summainta ja kolme rekisteriä (kuva 6). Arkkitehtuurista on luotu differentiaalilauseke, joka on muotoa $u(k) = u(k\text{-}1)+a_0e(k)+a_1e(k\text{-}1)+a_2e(k\text{-}2)$. Kuvan viisi kertoimilla a_0, a_1 ja a_2 kerrotaan erotusarvoja *ek*, *e(k-1)* ja *e(k-2)*. Differentiaalilausekkeen kertoimet a_0, a_1 ja a_2 muodostetaan kaavan (10) mukaan.

2.2.2. PID-säätimen suljetun silmukan moduulitoteutus

Chan ym. (2007) ovat tutkineet moduuleilla toimivaa sulautettua PID-säädintä, joka on toteutettu FPGA:lla. Sovelluksena on ollut lämpötilan ohjausjärjestelmä (kuva 7). Säätimen toiminnot on jaettu moduuleihin, joita voidaan muokata tarvittaessa. Työssä on keskitytty ADC-muunnin, PID-säädin ja PWM-generaattorimoduuliin, joita voidaan käyttää uudelleen tulevissa sovelluksissa.



Kuva 7. PID-säädön suljettu silmukka (Chan ym. 2007: 1903)

FSM-operaatiomoduli vastaa käyttäjän pyynnöistä, joihin kuuluu säätimen valmiustila, optimointi, ylösajo ja alasajo. FSM-operaation voi rakentaa halutunlaiseksi FPGA-ohjelmassa. User-interface moduli tulkitsee käyttäjän antamat sisääntulot, kuten järjestelmän käynnistys ja pysäytys, lämpötilan asetusarvot sekä ohjaa FSM-tilakonetta. ADC-interface moduli lähettää tutkitut signaalit, jotka tulevat vaiheittain ADC-modulilta PID-säätimen laskentaan. PID-controller moduli laskee ohje- ja mittausarvosta P-, I- ja D-termit. PID-säätimen stabiilisuutta säädetään parametrien P, I ja D kertoimilla. Säätimen ulostulo syötetään PWM:n sisääntuloksi. PWM-generaattorimoduli käyttää vertailijaa ja laskuria muuntamaan signaalin sisääntulon säännölliseksi kanttiaalloksi kuormituksen mukaan. Laskurin arvo kasvaa jokaisella kellojaksolla, jossa sen arvoa vertaillaan PID-säätömodulilta tulevaan arvoon. Tämän kyseisen arvon ollessa suurempi kuin laskurin arvo niin, silloin PWM-lähdöt menevät päälle muuten PWM-lähtö menee kiinni. Laskuri nollautuu, kun kellojaksoja on asetusten mukaisesti. PWM:n voi helposti asettaa halutulle kuormitukselle digitaalisessa säädössä sen on/off-toimintaperiaatteen luonteen vuoksi. Tutkimuksessa on käytetty Alteran ja Xilinx:n FPGA piirejä. Järjestelmä on testattu simuloimalla ja kokemuksen perusteella näytetty

toteen hyvän PID-säädinesityksen toiminnan tasaisuus. Moduleita voidaan testata esimerkiksi ModelSim-, Matlab- ja Simulink-ohjelmilla.

2.2.3. Optimointi numeerisessa PID-säätimessä

Saad ym. (2012) ovat tutkineet ja vertailleet PID-säätimen optimointia ja rakennetta differentiaalievoluutiolla ja geneettisillä algoritmeilla (GA). PID-säädintä on viritetty sekä Ziegler-Nichols:n menetelmällä että MSE- ja IAE-menetelmällä. Tutkimuksessa kiinnostaa erityisesti PID-säätimen rakenne, jossa säädin viritetään MSE- ja IAE-menetelmällä. Kyseisten optimointimenetelmien vaikutus optimointinopeuteen verrattuna Ziegler-Nichols:n menetelmään on huomattavan suuri. Optimointiaikaero voi vaihdella useista sekunneista kymmeniin sekunteihin riippuen säädettävästä sovelluksesta. Tutkimuksessa on käyty läpi DE-optimoinnin asetus- ja alustusparametrit arvioimalla ylä- ja alaraja satunnaisesti evaluoimalla. DE-algoritmi sisältää mutaatio-, risteytys- ja valintaoperaatiot sekä kohdefunktion kromosomien hyvyyden laskemisen. Tutkimuksen esityksessä on käytetty apuna matemaattisia kaavoja ja erilaisia kuvia optimoinnin rakenteesta. DE-optimoinnissa vertaillaan PID-säätimeltä saatuja kohde- tai yritevektorin ISE- ja IAE-hyvyysarvoja, jossa kyseisten vektorien arvot saadaan kohde- ja yritevektorin parametrien avulla.

2.2.4. Kiinteän pilkun esitys FPGA pohjaisessa PID-säätimessä

Lima ym. (2006) ovat tutkineet PID-säätimen toimintaa, kun käytetään kiinteän pilkun esitystä FPGA-ohjelmassa. Esitystä on vertailtu liukulukuesitykseen, jossa vertaillaan kiinteän pilkun esityksen vaikutusta eri bittimäärillä järjestelmän stabiilisuuteen. Kiinteän pilkun esitystä on arvioitu sanan pituuden analyysitekniikoilla. Työssä on pyritty löytämään kompromissi oikealle bittitarkkuudelle liuku- ja kiinteänpilkun esityksen välimaastosta. Tutkimuksessa FPGA:lle voidaan helposti ohjelmoida useita PID-säätimiä, jos käytetään kiinteän pilkun esitystä. Kiinteän pilkun esityksellä pystytään tutkimuksen mukaan säästämään merkittävästi FPGA-piirin resursseja kuten tehohäviöitä ja energian kulutusta sekä vähentämään suunnitteluaikaa. Suunnitteluongelmat FPGA:lla johtuvat usein laajan suunnittelutilan hyödyntämisestä, liuku- ja kiinteän pilkun lukujen muuntamisesta ja muunneltavien laitteistojen suunnittelun vaikeudesta.

Liman ym. (2006) empiirisessä tutkimuksessa on vertailtu kolmenlaista FPGA laitteistokuvausta. PID-säädintoteutuksesta on tehty diskreetti systeemi. Toteutuksia on

testattu kolmannen asteen siirtofunktiolla $G(s) = 1/(s + 1)^3$. Kuvauksissa on käytetty Matlab ja Simulink ohjelman liuku- ja kiinteän pilkun esitystä, jota on vertailtu FPGA-pohjaiseen kiinteän pilkun esitykseen. Esityksissä on käytetty differentiaalilauseketta $u(k) = u(k\text{-}1)+a_0e(k)+a_1e(k\text{-}1)+a_2e(k\text{-}2)$ parametrien laskemiseksi. Differentiaalilausekkeen kertoimet on rakennettu differentiaalilausekkeen tulojen a_0, a_1 ja a_2 avulla:

$$a_0 = Kc\left(1+\frac{T_d}{T_s}\right),\quad a_1 = -Kc\left(1+2\frac{T_d}{T_s}-\frac{Ts}{Ti}\right),\quad a_2 = Kc\frac{T_d}{T_s} \tag{10}$$

T_s on näytteenottoväli sekunneissa. K_c, T_i ja T_d ovat optimoitavat PID-säätimen parametrit, jotka automaattisessa aikavariantissa optimoinnissa päivitetään optimoinnin näytteenottovälin mukaan. Yhdessä esimerkeistä on vertailtu liuku- ja kiinteän pilkun luvuilla optimoituja kertoimia a_0, a_1 ja a_2. Liukulukukertoimiksi Matlab:lla on testauksessa saatu 223,1538, -441,4616 ja 218,3344. FPGA:lla kiinteän pilkun kertoimiksi on vertailevassa testauksessa saatu 223,1484, -441,4688 ja 218,3281. Kiinteän pilkun lukujen tarkkuudeksi kertoimella a_0, a_1, a_2 on asetettu ennen testausta (<9,7>), (<10,6>), (<9,6>). Tutkimuksen mukaan kiinteän pilkun lukujen kertoimilla tarvittaisiin 44, 43 ja 45 desimaalia, että päästäisiin liukulukuesityksen tarkkuuteen.

2.2.5. Sähkökäytön säätöjärjestelmän toteutus FPGA:lla

Penttinen ym. (2005) ovat käsitelleet tutkimushankkeessa lukujen esitysmuotoja ja erilaisia FPGA aritmetiikkaesityksiä. Aritmetiikka voidaan toteuttaa liukuluku- tai kokonaislukuaritmetiikalla. Kokonaislukuaritmetiikkaan kuuluu kiinteän pilkun aritmetiikka ja desimaalilukujen laskeminen skaalaamalla tai käyttämällä molempia yhtaikaa. Kokonaislukujen eli kiinteän pilkun aritmetiikan toteutus FPGA:lla on paljon helpompaa kuin liukulukuaritmetiikalla. Kokonaisluvut voidaan esittää etumerkittöminä tai etumerkillisinä. Kokonaislukuaritmetiikalla toimivat prosessorit ovat yleisempiä ja edullisempia säätöjärjestelmissä ja digitaalitekniikassa yksinkertaisen rakenteen ja nopeuden vuoksi.

Kokonaislukuoperaatiot ovat hyvin nopeita, joten ne olisi paras toteuttaa kokonaislukuja käyttäen. Kokonaislukulaskutoimitukset voidaan toteuttaa kiinteän pilkun luvuilla, jolloin luvut ovat aina oikeassa muodossa. Toinen vaihtoehto on skaalata luvut sopivalle kokonaislukualueelle, jolloin laskutoimitukset suoritetaan suoraan kokonaisluvuilla. Kiinteän pilkun esitys vastaa desimaaliesitystä, jossa pilkun oikea puoli vastaa desimaaliosaa ja vasen puoli vastaa kokonaislukuosaa. Kiinteän pilkun luvuilla

desimaaliosan tarkkuus on määritelty desimaaliosan bittien määrällä. Puhuttaessa esim. 16-bittisestä Q15-luvusta esitetyn luvun tarkkuus on tällöin 2^{-15} eli luvulla on n. neljän desimaalin tarkkuus, joka on riittävä monissa sovelluksissa. Reaaliluku X esitetään fraktaali eli murtolukumuodossa seuraavasti:

$$X = \sum_{i=-A}^{B} b_i r^{-1}, \quad 0 \le bi \le (r-1) , \tag{11}$$

Jossa A on kokonaisluvun muodostavien numeroiden määrä ja B fraktaaliosan numeroiden määrä. Binääriesityksessä kantaluku $r = 2$ ja binäärinumerot b_i saavat arvon väliltä [0,1]. Seuraavassa esimerkissä kiinteän pilkun kokonaisluku muunnetaan desimaaliluvuksi $(110.01)_2 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} = 6{,}25$. Vastaavasti desimaaliluku 6,25 muunnetaan kiinteän pilkun kokonaisluvuksi kertomalla se desimaalibittien määrällä $6{,}25 \times 2^2 = (110.01)_2$.

Liukuluvut voidaan myös skaalata kokonaislukualueelle käyttäen skaalausarvona 2:n potenssia. 32-bittisillä muuttujilla kertolaskun bitit on mahdutettava 30-bittiin (30 + 2 merkkibittiä), jolloin maksimi bittisiirros on 15-bittiä. Esimerkkinä yhteenlasku 0,87 + 1,65 = 2,52, jossa desimaaliluvut kerrotaan kertoimella 2^{15} kokonaislukualueelle. Kokonaisluvuiksi skaalattuna summaksi tulee 28508 + 54067 = 82575. Kokonaislukujen skaalaaminen takaisin alkuperäiselle lukualueelle tehdään jakamalla vastaus kertoimella 2^{15}. Tulokseksi saadaan n. 2,52, kun luku katkaistaan kahden desimaalin tarkkuuteen. Lukua ei skaalata takaisin alkuperäiselle lukualueelle, jos laskutoimitukset jatkuvat sopivalla kokonaislukualueella myöhemmin. Kertolaskussa luvut skaalataan vastaavasti kuin yhteenlaskussa. Skaalattaessa vastaus takaisin alkuperäiselle lukualueelle, jaetaan kertolaskun kertoimella tulos 2^{16}. Esimerkki jakolaskussa lausekkeen 1,5/0,5 = 3 luvut skaalataan: $2^{15} \times (2^{15} \times 1{,}5 / 2^{15} \times 0{,}5) = 98304$. Skaalaus takaisin tapahtuu jakamalla tulos 2^{15}.

2.2.6. Sanan pituuden valintamenetelmä kiinteän-pilkun luvuilla

Urriza, Navarro, Artigas and Lucia (2010) ovat tutkineet sanan pituuden valintamenetelmää liukuluvuilla ja kiinteän pilkun luvuilla digitaalisilla FPGA-toteutuksilla. Menetelmät on toteutettu uusilla VHDL-2008 kiinteän pilkun- ja liukulukukirjaston pakkauksilla. Näillä valintamenetelmillä pystytään lyhentämään vaikeiden yksityiskohtien suunnittelukiertoa, helpottamaan aritmeettisia operaatioita ja tekemään VHDL:stä kilpailukykyinen muiden vaihtoehtojen kanssa. Vertailtaessa

liuku- ja kiinteän pilkun esitystä suunnittelijan täytyy itse analysoida kumpi esityksistä on sopivampi kyseiseen sovellukseen. Liukulukuesitys kuluttaa enempi muistia ja on hitaampi. Kiinteän pilkun työkaluina erilaisissa sovelluksissa on käytetty Alteran DSP-builderia ja Xilinx järjestelmä-generaattoria, joka tuottaa VHDL-kieltä suoraan Matlab ja Simulink esityksestä.

Perinteisissä aritmeettisissä operaatioissa alemmalla kuvaustasolla käytetään ”numeric_std” ja ”std_logic” kirjastopakkauksia. Molemmissa kirjastopakkauksissa on käytetty etumerkillisiä (signed) vektoreita, aritmeettisia ja vertailuoperaatioita. Seuraavaksi esimerkki kahden muuttujan laskemisesta kiinteän pilkun luvuilla, jossa täytyy kohdistaa binääripisteet ja lisätä laskelmaan. Summaoperaattori antaa tuloksen, jossa vektorin pituus on yhtä suuri kuin sen pisimmän kohdemuuttujan pituus. Seuraava koodi esittää desimaalibittien siirron vasemmalle, kun kohdistetaan binääripistettä:

```
signal   A:          signed(8 downto 0);          <w=9,nq=8>
signal   B:          signed(7 downto 0);          <8,5>
signal   Y:          signed(11 downto 0);         <12,8>          (12)
-------------------------------------------------------------------------------
Y <= A + (B(7) & B & "000");
```

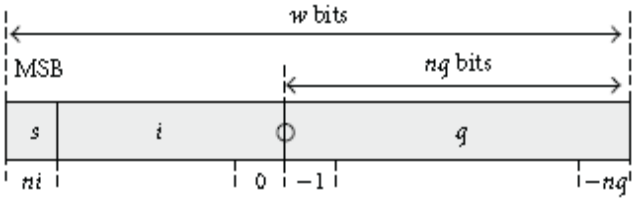
Alla olevassa koodissa kaksi etumerkillistä kohdemuuttujaa kerrotaan keskenään. Tehtävässä on varauduttu lisäämään bittien määrää ja käyttämään tuloksen viipalointia ja indeksointia tietyssä osassa tulosta. Tulos pyöristetään ennen kuin bittijono katkaistaan määrätylle vähiten merkitsevän bitin (*LSB* / 2) tasolle. Seuraavassa koodiesimerkki:

```
signal   C:          signed(7 downto 0);          <8,5>
signal   D:          signed(7 downto 0);          <8,5>
signal   Y:          signed(10 downto 0);         <11,7>
signal   Y_tmp:      signed(15 downto 0);                         (13)
constant LSB2:       signed(3 downto 0);          "0100"
-------------------------------------------------------------------------------
Y_tmp <= (C * D) + LSB2;
Y <= Y_tmp(13 downto 3);
```

Yllä olevassa koodissa (13) tulee ylivuotoa, jos kerrontatulosta ei voida esittää muodossa (<11,7>). Suunnittelija hallitsee laskutoimitusta käyttämällä esim. saturoitua aritmetiikkaa, jossa ylivuotavat lukuarvot asetetaan suurimpaan mahdolliseen arvoon.

VHDL-2008 ”fixed_pkg” pakkausten kiinteän pilkun järjestelmissä tuetaan korkeamman kuvaustason suunnittelua. Tämä täysin syntetisoitava pakkaus on osa VHDL-2008 standardia, jossa voidaan käyttää kiinteän pilkun aritmetiikkaa. Pakkauksessa on esitetty kaksi uudentyyppistä kiinteän pilkun etumerkitöntä (ufixed) ja etumerkillistä (sfixed) esitystä. Ne ovat kokonaislukualueen taulukoita, jossa signaalit ovat muodossa (<*w*,*nq*>). Etumerkillisen kiinteän pilkun luvun signaali muodostetaan seuraavasti:

```
signal   A:          sfixed(ni downto -nq);          (14)
```



Kuva 8. Kiinteän pilkun muoto kahden komplementin esityksessä (Urriza ym. 2010: 2)

Merkitsevimmän bitin (MSB) asema suhteessa binääripisteeseen on $ni = w - nq - 1$ (ohjelma 14, ohjelma 15 ja kuva 8). Kiinteän pilkun esitystarkkuus on 2^{-nq}, jonka alue etumerkillisillä kiinteän pilkun luvuilla esitetään $[-2^{ni}, 2^{ni} - 2^{-nq}]$. Signaalin arvot on koodattu kokonaisuudessaan *w*-bittiin, jossa pisteen jälkeen on esitetty murtoluvut *nq*-bitteinä. Seuraavassa operaatiossa on kohdistettu binääripiste ja esitetty yhteenlaskuoperaation etumerkkimenetelmä:

```
signal   A:          sfixed(0 downto -8);          <9,8>
signal   B:          sfixed(2 downto -5);          <8,5>
signal   Y:          sfixed(3 downto -8);          <12,8>          (15)
--------------------------------------------------------------------------------
Y <= A + B;
```

Kertolaskuoperaatiossa on käytetty VHDL-2008 standardin ”resize” funktiota, jossa on muunnettu kertolaskun vastaus yhdestä kiinteän pilkun tyypistä toiseen (ohjelma 16). Funktio käyttää kvantisointimoodia, joka tarkoittaa pyöristämistä (rounding) tai typistystä (truncate). Kun pyöristysrutiini (fixed_round) on ”true” niin, silloin käynnistetään pyöristys. Pyöristysrutiinin ollessa ”false” numeroarvo typistetään (fixed_truncate). Pyöristys on tehty, jos MSB-bitin loppuosassa on ”1” ja pyöristämättömän LSB-bitin tulos on ”1” tai alemmat bitit loppuosassa sisältävät ykkösiä. Edellisen lauseen numeroarvot pyöristyvät, jos kyseiset bittien arvot ovat voimassa. Ylivuoto (overflow) moodin ollessa ”fixed_saturate” oletusarvossa silloin

ohjelma palauttaa mahdollisimman maksimaalisen arvon tulokseksi, jos numeroarvo on liian suuri esitettäväksi. Seuraava ohjelmaesitys näyttää kertolaskun signaalien *C* ja *D* välillä, vastauksen signaalin *Y* mukaan, maksimaalisen arvon palauttamisen kertolaskun tulokseksi ja tuloksen pyöristämisen:

```
signal   C:          sfixed(2 downto -5);          <8,5>
signal   D:          sfixed(2 downto -5);          <8,5>
signal   Y:          sfixed(3 downto -7);          <11,7>          (16)
-------------------------------------------------------------------------------
Y <= resize(C*D,Y,fixed saturate, fixed round);
```

Mitoitussääntö datan leveydeksi on se, että ei synny ylivuotomahdollisuutta. Mitoitussääntöjen esityksessä on poistettu yli- ja alivuoto bitit (taulukko 1).

Operaatio	Tulosalue
A ± B	max(A'left, B'left) + 1 downto min(A'right, B'right)
A * B	A'left, B'left + 1 downto A'right, B'right

Taulukko 1. Kiinteän pilkun esityksen lisäämissäännöt (Bishop 2008: 2)

2.2.7. PID-säätimen kokeellinen virittäminen

Seuraavissa alaluvuissa on tutustuttu PID-säätimen kokeelliseen virittämiseen. Kokeellista virittämistä käytetään kun prosessin mallia ei tunneta tarkasti. Kokeellinen virittäminen edellyttää yleensä erilaisten kokeellisten menetelmien tuntemusta.

2.2.7.1. Kokeellinen menetelmä

Koskelan työssä (2010) on tutkittu säätöjärjestelmän toimintaa täysin kokeellisessa virityksessä erilaisilla kokeellisilla virityksillä, kunnes vaste täyttää vaatimukset. Kokeellisessa virityksessä täytyy tuntea erilaisia nyrkkisääntöjä säätimen käyttäytymisestä sekä lisäksi tuntea prosessi hyvin. Kun PID-säädintä aloitetaan virittämään kytketään Ti- ja Td-termi pois käytöstä ja asetetaan Kp vahvistuskerroin pieneksi. Tämän jälkeen Kp-termiä kasvatetaan kunnes vaste on kasvanut yli 10% mutta alle 15% ohjearvosta. Kun Kp-termi on saatu oikealle alueelle, säädetään Td-termiä alaspäin kunnes osa ylityksestä poistuu ja asettumisaikaa saadaan pienemmäksi. Lopuksi hienosäädetään Ti-termiä alaspäin kunnes asettumisaika ja vasteen säätövirhe poistuu. Analogisen säädön (2011b) PID-säätimen kokeellisen virittämisen

nyrkkisäännössä neuvotaan säätämään I-termi niin suureksi, että pysyvä poikkeama poistuu. I-termin kasvattaminen lisää värähtelyä, jolloin säätimen epätasapaino lisääntyy. Integroivalla säädinjärjestelmällä D-termi säädetään isoksi kunnes säätimen vaste stabiloituu, nopeutuu ja värähtely vähenee. Nopeutensa vuoksi säädin on herkkä kohinalle ja viiveelle.

2.2.7.2. Zhuangin ja Attertonin optimointikriteeri

Tan ym. (2006) tutkimuksen mukaan FOPDT-malli on paljon käytetty optimointi-menetelmä erilaisissa ohjausprosesseissa. Se on yhdistelmä ensimmäisen asteen prosessista ja kuolleesta ajasta. FOPDT-mallia on usein arvioitu hyväksi myös korkeamman asteen prosesseissa eli toisen asteen ja korkeammilla prosesseilla. Dingyu ym. (2007) ovat tutkineet optimiasetuskriteeriä. Virityssuunnitelman voi esittää FOPDT-mallilla, jossa kolme optimikriteeriä perustuvat ISE-, ISTE- ja IST^{2}E-kriteereihin. Lausekkeisiin on saatu optimiteoreettiset parametrit sovitettujen käyrien avulla. PID-säätimessä sen parametrit ja kertoimet voidaan asettaa seuraavasti:

$$K_p = \frac{a_1}{k}\left(\frac{L}{T}\right)^{b1}, \qquad T_i = \frac{T}{a_2 + b_2(L/T)}, \qquad T_d = a_3 T\left(\frac{L}{T}\right)^{b3}, \tag{17}$$

jossa a_i ja b_i ovat ISE-, ISTE- ja IST^{2}E-kriteerien kertoimia ja L/T on kuolleenajan L ja nousuajan T suhde. Kuollutaika on lyhempi kuin nousuaika, kun L/T-suhde on väliltä [0,1;1,0]. Kun kuollutaika on suurempi kuin nousuaika, silloin L/T-suhde on väliltä [1,1;2,0]. Asetusarvot PID-säätimen parametreille reaaliaikaisessa ja viiveellisessä takaisinkytkennässä löytyvät taulukosta 2 ja taulukosta 3.

L/T-suhde	0.1 − 1			1.1 − 2		
reaaliset kriteerit	ISE	ISTE	IST²E	ISE	ISTE	IST²E
a1	1,048	1,042	0,968	1,154	1,142	1,061
b1	-0,897	-0,897	-0,904	-0,567	-0,579	-0,583
a2	1,195	0,987	0,977	1,047	0,919	0,892
b2	-0,368	-0,238	-0,253	-0,220	-0,172	−0,165
a3	0,489	0,385	0,316	0,490	0,384	0,315
b3	0,888	0,906	0,892	0,708	0,839	0,832

Taulukko 2. Reaaliaikaisen PID-säätimen parametrien asetusarvot (Dingyu ym. 2007: 206)

viiveelliset kriteerit	ISE	ISTE	IST²E	ISE	ISTE	IST²E
a1	1,260	1,053	0,942	1,295	1,120	1,001
b1	-0,887	-0,930	-0,933	-0,619	-0,625	-0,624
a2	0,701	0,736	0,770	0,661	0,720	0,754
b2	-0,147	-0,126	-0,130	-0,110	-0,114	-0,116
a3	0,375	0,349	0,308	0,378	0,350	0,308
b3	0,886	0,907	0,897	0,756	0,811	0,813

Taulukko 3. Viiveellisen PID-säätimen parametrien asetusarvot (Dingyu ym. 2007: 206)

2.2.7.3. Värähtelyrajamenetelmä

Analogisen säädön (2011b) Ziegler-Nicholsin värähtelykoemenetelmässä prosessin Ti- ja Td-termit kytketään pois käytöstä ja Kp-termi asetetaan pieneksi, jonka jälkeen Kp vahvistuskerrointa nostetaan kokeellisesti ylöspäin kunnes askelvaste tai impulssivaste värähtelee harmonisesti stabiilisuusrajalla. Tätä vahvistusta kutsutaan kriittiseksi vahvistukseksi K_{krit} ja kriittisen vahvistuksen jaksonaika on kriittinen jakso T_{krit}. Kriittisen vahvistuksen ja kriittisen jaksonajan selvittyä optimoidaan vahvistuskerrointa. Seuraavaksi parametreille K_{krit} ja T_{krit} löytyy hyvät virityskertoimet PID-säätimen parametrien Kp, Ti ja Td virittämiseksi (taulukko 4 ja 5). Taulukon arvoilla ei välttämättä saada stabiilia tulosta, jolloin PID-säätimen viritystä täytyy jatkaa kokeellisesti hienosäätämällä kokeellisen menetelmän mukaan.

Säädin	Kp	Ki	Kd
P	0.5Ku		
PI	0.4Ku	0.8Tu	
PID	0.6Ku	0.5Tu	0.125Tu

Taulukko 4. Alkuperäiset Ziegler-Nichols asetukset (Hägglund ym. 1995: 5)

Säädin	Kp	Ki	Kd
alkuperäinen	$0,6K_{krit}$	$0,5T_{krit}$	$0,125T_{krit}$
vähän ylitystä	$K_{krit} / 3$	$0,5T_{krit}$	$0,125T_{krit}$
ei ylitystä	$K_{krit} / 5$	$0,5T_{krit}$	$0,125T_{krit}$

Taulukko 5. Ziegler-Nichols asetusten eri versiot (Hägglund ym. 1995: 5)

2.2.7.4. Ekstrapoloiva viritysmenetelmä

Neljännesohje on värähtelyyn perustuva viritysmenetelmä (Niiranen 1999). Virityksessä säädin kytketään avoimeksi, jossa vahvistuskerroin asetaan pieneksi ja aletaan nostamaan kokeellisesti ylöspäin. Vahvistuskerrointa nostetaan niin kauan, että lähtösuureeseen saadaan vaimeneva värähtely. Tässä perättäisten ylitysten amplitudien tulee vaimentua neljännes per periodi, kun vahvistuskerrointa nostetaan. Tämän jälkeen mitataan neljännesvärähtelyjaksonaika T_{qt} ja sijoitetaan vahvistuskerroin K_{qt} ja värähtelyjaksonaika taulukkoon 5.

Säädin	Kp	Ti	Td
P	$0{,}25K_{qt}$		
PI	$0{,}25K_{qt}$	$0{,}67T_{qt}$	
PID	$0{,}25K_{qt}$	$0{,}67T_{qt}$	$0{,}17T_{qt}$

Taulukko 6. Neljännesmenetelmän asetukset (Niiranen 1999)

Taulukkoarvot sijoitetaan PID-säätimen parametreiksi ja suljetaan avoin kierto. Neljännesmenetelmän vaimennus saavuttaa likimäärin I_2-indeksin minimiarvon. Kyseinen likimääräismerkki tarkoittaa, että vahvistuskerrointa aletaan yleensä korjaamaan ylöspäin minimiarvosta. Vahvistuslisäyksien jälkeen on hyvä herättää säätöpiiri muuttamalla asetusarvoa vuorottain ylös- ja alaspäin toimintapisteen säilyttämiseksi.

2.2.7.5. Numeerinen optimointi

Analogisen säädön (2011b) PID-säätimen kokeellisen virittämisen numeerisessa optimoinnissa parametrit PID-säätimelle voidaan virittää numeerisen optimoinnin avulla, jossa minimoidaan PID-säätimen erosuuretta IAE- tai ISE-kustannusfunktiolla. Seuraavassa numeerisen menetelmän painotetut aikaintegraalit:

$$ISE = \int_0^T (e(t))^2\,dt \;\approx\; \sum_{k=0}^{n-1} (e(k))^2 \times T,\;\; ISE : lle \tag{18}$$

$$IAE = \int_0^T \left|e(t)\right|\,dt \;\approx\; \sum_{k=0}^{n-1} e(k) \times T,\;\; IAE : lle \tag{19}$$

Aikaintegraalien ylärajaksi valitaan suurehko *T*:n arvo, jolla referenssin ja todellisen vasteen erotus menee nollaan eli poikkeamaa oikeasta arvosta ei ole. Muuten integraalin arvosta tulee liian suuri tai pieni, jolloin erosuure ei minimoidu vaan kasvaa tai yliminimoituu oikeaan arvoon verrattuna. Integrointiaskeleelle täytyy antaa aikavakio *T*, joka on erotusarvon funktion aikaintegraalin askelpituus. IAE- ja ISE- kriteereillä voidaan integroida fyysistä suuretta, energian kulutusta tai minimoida esim. PID-säätimen erosuuretta.

3. ITSE-ADAPTOITUVAT OHJAUSPARAMETRIT

Brest ym. (2008) ovat tehneet empiirisen tutkimuksen itse-adaptoituvien ohjausparametrien muutoksesta evoluutioprosessin aikana. Yksi itse-adaptoituva ohjausparametrimekanismi on äskettäin esitetty jDE-algoritmi, joka käyttää differentiaalievoluutiota. Se perustuu alkuperäiseen ”DE/rand/1/bin” differentiaali-evoluutioon, jonka esitys on parempi kuin alkuperäisen DE-algoritmin. Lisäksi jDE:tä voidaan laajentaa mukauttamalla kahta mutaatiostrategiaa uudella jDE-2 algoritmilla (Qin ym. 2005). Uudessa jDE-Algoritmissa optimoidaan *F* ja *CR* ohjausparametreja. Uusi *F* valitaan satunnaisesti väliltä [0,1;1,0]. Uusi *CR* arvo on välillä [0,1]. Todennäköisyysjakaumaan perustuvia parametreja jDE-algoritmissa ovat τ_1 ja τ_2 vertailuparametrit. Mutaatiokertoimen skaalausrajaparametrit ovat F_l ja F_μ. Skaalausrajaparametrien ja vertailuparametrien avulla lasketaan $F_{i,G+1}$ ja $CR_{i,G+1}$ arvot, jotka on saatu ennen mutaatiolaskentaa. Ne vaikuttavat mutaatioon, risteytykseen ja uuden vektorin $X_{i,G+1}$ valintaan. Mekanismi pitää kolmannen ohjausparametrin *NP* kiinteänä optimoinnin aikana. Alkuperäisessä DE-algoritmissa kaikki ohjausparametrit ovat kiinteitä. Automaattisella parametrien säädöllä on todettu olevan suurta hyötyä optimoinnissa. Itse säätyviä parametreja on usein käytetty evoluutiolaskennalla toteutetuissa optimointitehtävissä. Modifioitu DE-algoritmi parantaa optimoinnin tehokkuutta ja häiriönsietokykyä. Populaation yksilöitä on laajennettu *F* ja *CR* ohjausparametreja optimoimalla.

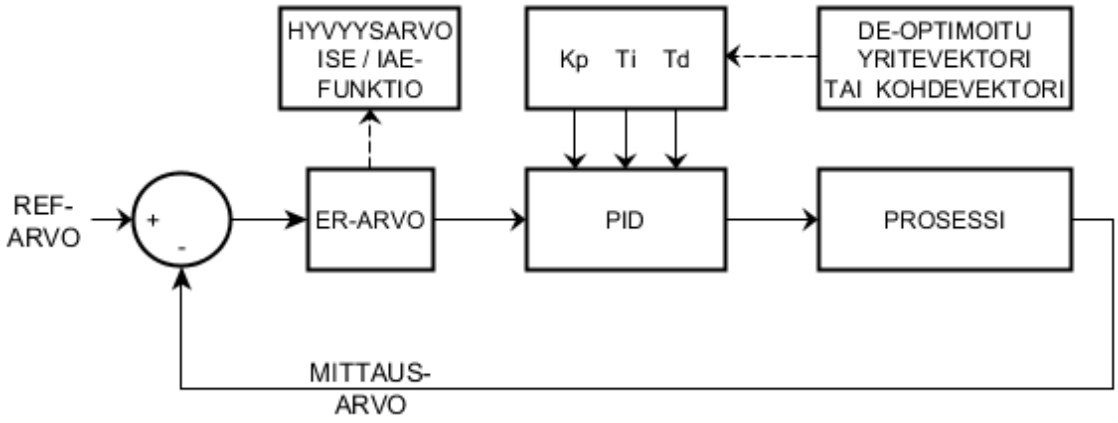
Empiirisessä tutkimuksessa (Brest ym. 2006a; Brest ym. 2006b) on vertailtu erilaisilla funktioilla alkuperäistä DE-algoritmiä, itse-adaptoituvaa differentiaalievoluutio-algoritmia (SADE) sekä itse adaptoituvia ja adaptoituvia ohjausparametrimekanismeja (jDE). Populaation kokona on pidetty (*NP*=100). Tutkimuksessa voidaan nähdä, että itse-adaptoituvissa DE-algoritmeissä suppeneminen eri funktioilla on lähes samanlaista ja tulokset ovat parempia kuin alkuperäisellä DE-algoritmilla. Parametrien automaattinen optimointi parantaa suppenemisnopeutta kohti parasta mahdollista optimiarvoa kokeellisten testausten perusteella. Tutkimuksen perusteella jDE antaa parhaat tulokset parannuskertojen määrässä ja suppenemisessa. jDE-algoritmit perustuvat satunnaissovituspohjaiseen DE-algoritmiin, jonka toiminta on helposti muokattavissa adaptiiviseksi alkuperäisestä DE-algoritmistä.

3.1. PID-säätimen optimointi jDE-algoritmilla

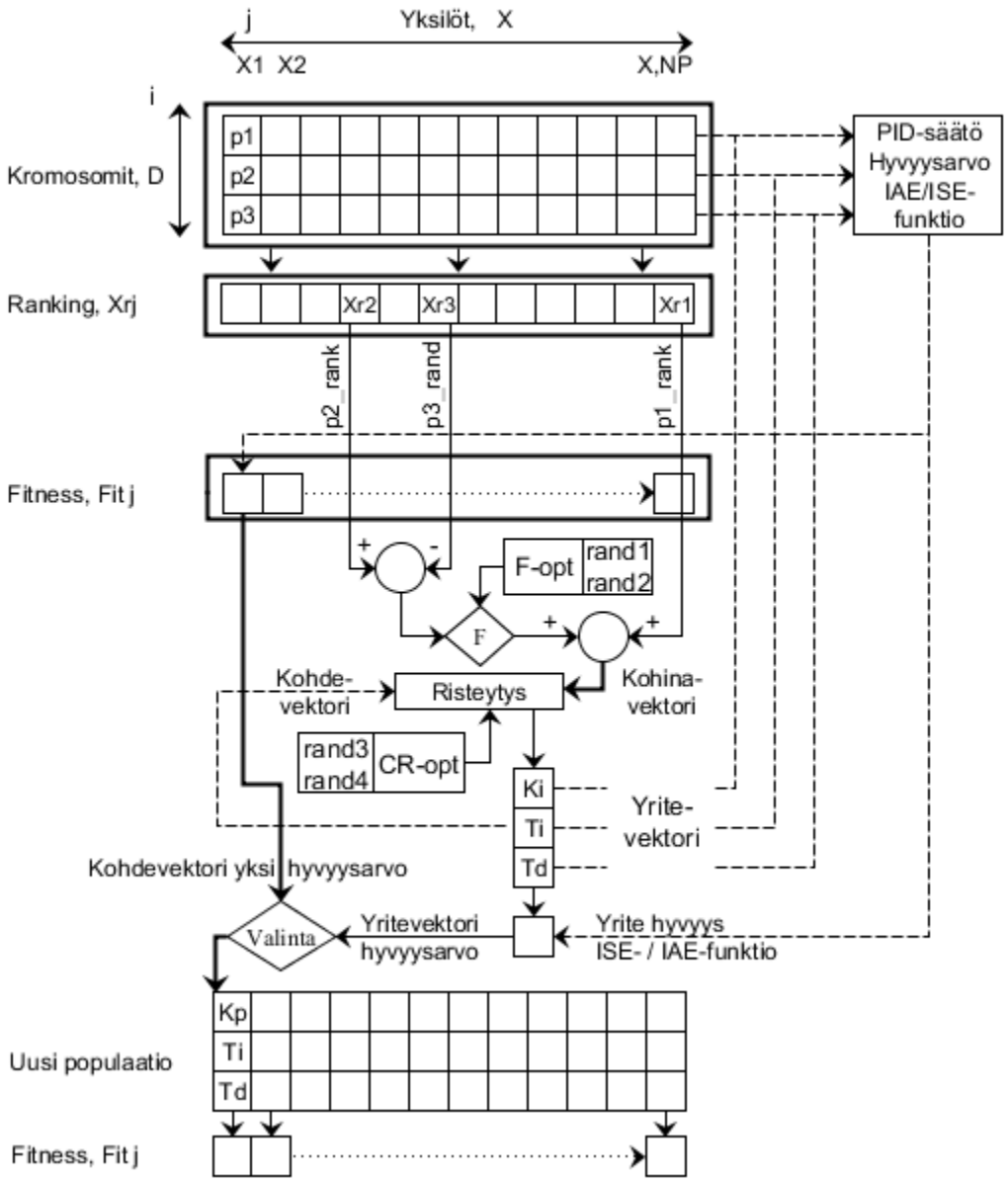
Luvussa 3 suunnitellaan PID-säädin, joka on numeerisesti optimoitu itse-adaptoituvalla jDE-algoritmilla (Brest ym. 2006a; Brest ym. 2006a; Brest ym. 2008;). Lisäksi itse-adaptoituvan jDE-algoritmin parametrit on tarkoitus järjestää suuruusjärjestykseen ranking-perusteisella mutaatio-operaatiolla (Gong ym. 2013a; Gong ym. 2013b) ja parantaa sen esitystä pitämällä ratkaisuun sidotut rajat käyttökelpoisina ongelmaan nähden (kuva 10) (Rönkkönen ym. 2005; Brest ym. 2006b). Alaluvuissa suunnitellaan edellä mainittujen lisäksi säätimen rakennetta ja optimointia.

3.1.1. Säätimen rakenne

Kuvassa 1 ja 4 on kuvattu PID-säätimen optimointirakenteen osat. Optimointirakenne sisältää optimoinnin alustuksen, hyvyysarvon päivityksen PID-säätimellä, mutaatio-, risteytys- ja uuden yksilön valintavaiheen DE-laskennassa. Alkuperäiseen DE-algoritmiin (DE/rand/1/bin) verrattaessa jDE-algoritmilla (jDE/rand/1/bin) on itse-adaptoituvat parametrit. Kyseisen itse-adaptoituvan optimointimenetelmän soveltaminen alkuperäiseen DE-algoritmiin on helposti toteutettavissa alkuperäisen DE-algoritmin mukaisesti (kuva 4). Uusien yksilöiden hyvyysarvon päivitys jDE-algoritmissa (Brest ym. 2008) tapahtuu PID-säätimen algoritmissa (kuva 9 ja 10). Yritevektorin kromosomit syötetään PID-säätimen Kp, Ti ja Td parametreiksi. PID-säätimellä yksilöiden hyvyysarvo päivitetään suoraan ohjaus- ja mittausarvon erotusarvosta, josta ne syötetään hyvyysarvon laskennan jälkeen yrite- ja kohdevektorin vertailuun ja valintaan uudeksi populaation yksilöksi (Saad ym. 2012).



Kuva 9. Hyvyysarvon päivitys PID-säätimellä (mukaillen Saad ym. 2012).



Kuva 10. Populaation uusien yksilöiden päivitys (mukaillen Saad ym. 2012: 8; Brest ym. 2008).

3.1.2. Optimoinnin alustus

Ensin on asetettu DE-optimointiparametrit, jossa populaation koko NP=10…100, risteytysvakio CR=0,9, mutaatiokerroin F=0,6, kromosomien määrä D=3 ja maksimi populaatioiden määrä G_{max}=100. Asetusten jälkeen vektoripopulaatio on alustettu antamalla ylä- ja alaraja arvioimalla sekä satunnaisesti evaluoimalla jokaisen kohdefunktion kromosomin hyvyys. Jos jostain syystä kohdevektorin arvo alittaa tai

ylittää ala- ja ylärajan niin, alustetaan kyseinen kohdevektori Pop_{ij} uudestaan. Alla kaavat kromosomien laskemiseksi: (Saad ym. 2012, 6.)

$$Pop_{ij} = L + (H - L) \times rand_{ij}[0,1], \quad i = 1,\ldots,D, \quad j = 1,\ldots,NP \tag{20}$$

$$\text{if } Pop_i \notin [L, H], \quad Pop_i = L + (H - L) \times rand_i[0,1] \tag{21}$$

Yksilöiden kromosomien esitykset alustuksessa säätimen parametreiksi on laskettu esimerkissä Kp, Ti ja Td:lle seuraavasti. (Saad ym. 2012, 7)

$K_p \in [0,15]$,

$$Pop_{11} = 0 + (15 - 0) \times rand_{11}(0,1) \tag{22}$$
$$Pop_{11} = p_{11}$$

$K_i \in [0,15]$,

$$Pop_{12} = 0 + (15 - 0) \times rand_{12}(0,1) \tag{23}$$
$$Pop_{12} = p_{12}$$

$K_d \in [0,15]$,

$$Pop_{13} = 0 + (15 - 0) \times rand_{13}(0,1) \tag{24}$$
$$Pop_{13} = p_{13}$$

3.1.3. Mutaatiokertoimen optimointi

Mutaatioesityksessä populaatiosta valitaan kaksi yksilöä (kohdevektoria) satunnaisesti erotusvektoriksi. Erotusvektori kerrotaan tämän jälkeen mutaatiokertoimella *F*, jolloin erotusvektori muuttuu painotetuksi erotusvektoriksi. Mutaatiokerrointa säädetään mutaatioalueen minimi (F_l) ja maximi (F_μ) arvojen sekä vertailuarvojen (τ_1) ja satunnaislukujen $rand_j \in \{1,2\}$ avulla, joissa satunnaisluvut ovat $\in [0,1]$. Kohinavektorissa yhdistetään painotettu erotusvektori ja populaatiosta valittu yksilö seuraavasti $V_{j,G+1} = X_{r3,G} + F_{i,G+1} \times (X_{r1,G} - X_{r2,G})$, jossa populaation jäsenet valitaan satunnaisesti alaindekseillä $r1$, $r2$ ja $r3$. Mutaatiokerrointa optimoidaan kaavalla (25):

$$F_{i,G+1} = \begin{cases} F_l + rand_1 \times F_\mu, & \text{jos } rand_2 < \tau_1 \\ F_{i,G}, & \text{muuten.} \end{cases} \tag{25}$$

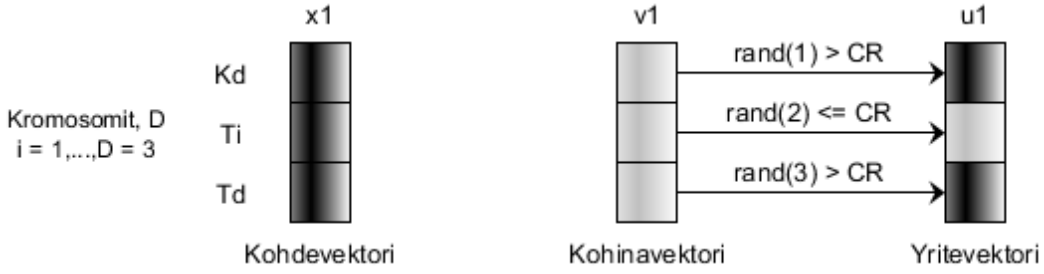
3.1.4. Risteytysvakion optimointi

Risteytysesityksessä on luotu yritevektori $U_{j,G+1} = (U_{1j,G+1}, U_{2j,G+1},\ldots,U_{Dj,G+1})$ kaavassa (27), jossa alaindeksi $i = 1,\ldots,D$ tarkoittaa populaation parametrien määrää. Esityksessä kohdevektorista yksi tai kohinavektorista eli mutaatiovektorista valitaan kromosomi eli parametri yritevektoriin risteytysvakion optimointikaavan (26), risteytyskaavan (27) ja risteytys diagrammin mukaisesti (kuva 11). Risteytysvakiota optimoidaan vertailuarvon (τ_2) ja satunnaislukujen $\text{rand}_j \in \{3,4\}$ avulla, joissa satunnaisluvut ovat $\in [0,1]$.

$$CR_{i,G+1} = \begin{cases} \text{rand}_3, & \text{jos } \text{rand}_4 < \tau_2 \\ CR_{i,G}, & \text{muuten} \end{cases} \tag{26}$$

$$U_{ij,G+1} = \begin{cases} U_{ij,G+1}, & \text{jos } (\text{rand}_i \le CR_{i,G+1}) \vee (\text{rand} = i) \\ X_{ij,G}, & \text{muuten.} \end{cases} \tag{27}$$

Risteytys on tehty PID-säätimen parametrien monimuotoisuuden lisäämiseksi populaation jokaisessa yksilössä risteytysvakiota ja satunnaislukua $\text{rand}_i \in [0,1]$ käyttäen kaavassa (26). Diagrammissa on näytetty risteytyksen toteuttaminen, kun yritevektori on luotu kohde- ja kohinavektoreista (kuva 11).

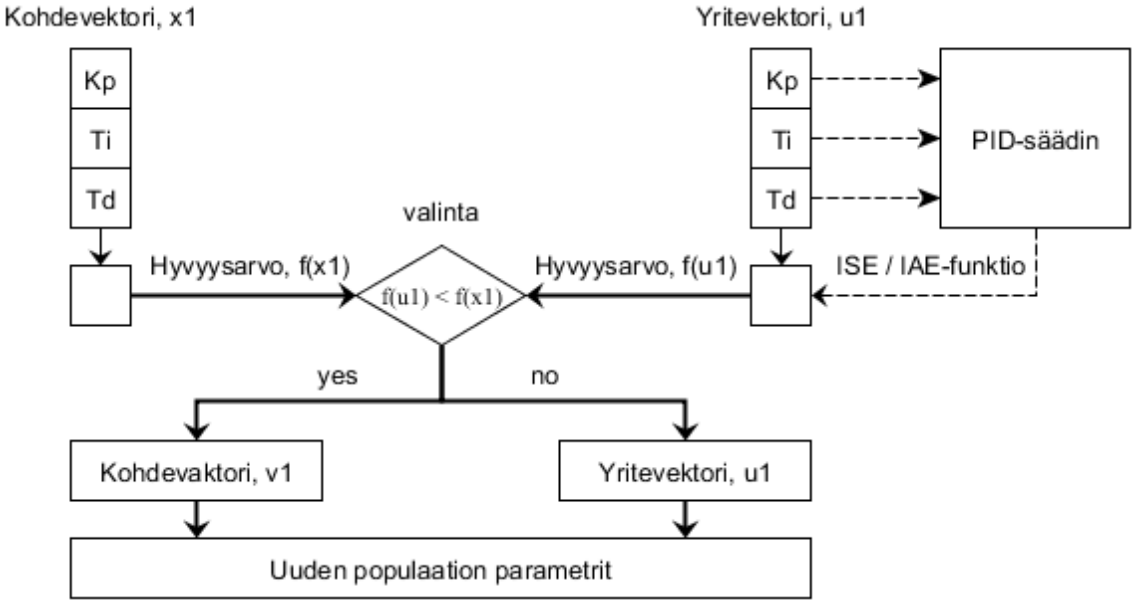


Kuva 11. Vektorien kromosomien risteytysdiagrammi (mukaillen Saad ym. 2012: 9)

3.1.5. PID-säätimen parametrien valinta

Alustusvaiheessa populaation kohdevektorin kromosomosomeille on laskettu arvo, joka perustuu käytännön kokemukseen ja satunnaisuuteen. Näin saadaan optimointialue, jolla parametrien optimointi tapahtuu. Nämä alustetut arvot syötetään PID-säätimen parametreiksi, jossa lasketaan kyseiselle kohdevektorille ISE- ja IAE hyvyysarvo (Aalto yliopisto 2011b: 1-5). Näin saadaan alustetuille kohdevektorin parametreille ISE- ja

IAE-hyvyysarvo. Samalla optimoidaan DE-laskennan kautta yritevektorin parametreille eli kromosomeille arvot, jotka lähetetään PID-säätimen Kp, Ti ja Td parametreiksi, joista lasketaan PID-säätimen erotusarvosta kyseiselle yritevektorille ISE- ja IAE-hyvyysarvo. Alustusvaiheen ja ensimmäisen DE-optimointisukupolven jälkeen optimoinnissa valitaan hyvyysarvon perusteella joko yrite- tai kohdevektorin parametrit uuteen yritepopulaatioon (kuva 12). Optimointia jatketaan kunnes maksimi populaatioden määrä on saavutettu.



Kuva 12. Parametrien valinta (mukaillen Saad ym. 2012: 10)

Hyvyysarvon laskentaan tarvitaan PID-säätimen ohjearvoa ja prosessin mittausarvoa sekä näytteenottoväliä T_h ja derivointitermiä K_d, joiden avulla pystytään laskemaan ISE ja IAE lausekkeen arvot. Näytteiden määrä n asetetaan niin suureksi, että erosuure ehtii minimoitua kustannuslaskennan aikana. ISE- ja IAE-hyvyys on laskettu PID-säätimen erotusarvosta kaavoilla:

$$\mathrm{ISE} = \int_0^T (e(t))^2 \,\mathrm{dt} \approx \sum_{k=0}^{n-1} (e(k))^2 \times T, \ : \mathrm{ISE} \tag{28}$$

$$\mathrm{IAE} = \int_0^T |e(t)| \,\mathrm{dt} \approx \sum_{k=0}^{n-1} e(k) \times T, \ : \mathrm{IAE} \tag{29}$$

3.1.6. Parannuksia jDE-algoritmiin

Rönkkönen ym. 2005 ja Brest ym. 2006 ovat kehittäneet jDE algoritmiin parannuksia, jotka tekevät algoritmista yleisemmän ja parantavat sen esitystä. Kyseisillä parannuksilla pidetään ratkaisuun sidotut yritevektorin rajat käyttökelpoisina ennaltamääriteltyjen rajojen ($X_{j,\text{low}}$) ja ($X_{j,\text{upp}}$) mukaan. Kaavassa (30) yritevektorin arvo palautetaan sidottujen yritevektorin parametrien rajojen rikkoutuessa:

$$U_{ij,G} = \begin{cases} 2 * X_{j,\text{low}} - U_{ji,G} & \text{jos } U_{ji,G} < X_{j,\text{low}}, \\ 2 * X_{j,\text{upp}} - U_{ji,G} & \text{jos } U_{ji,G} > X_{j,\text{upp}}, \end{cases} \tag{30}$$

Seuraavassa on vielä parannettu versio edelliselle jDE-algoritmille yritevektoriin sidottujen rajojen pitämiseksi. Alla olevat kaavat perustuvat satunnaisesti valituun arvoon t = rand(0,1) ja todennäköisyyteen p_0 = 0,5, joita verrataan keskenään. Lisäksi kaavassa (31) verrataan yritetevektorin parametreja ennaltamääriteltyihin rajoihin ($X_{j,\text{low}}$) ja ($X_{j,\text{upp}}$):

$$U_{ij,G} = \begin{cases} X_{j,\text{low}} & \text{jos } (t \le p_0) \wedge (U_{ji,G} < X_{j,\text{low}}), \\ X_{j,\text{upp}} & \text{jos } (t \le p_0) \wedge (U_{ji,G} > X_{j,\text{upp}}), \\ 2 * X_{j,\text{low}} - U_{ji,G} & \text{jos } (t > p_0) \wedge (U_{ji,G} < X_{j,\text{low}}), \\ 2 * X_{j,\text{upp}} - U_{ji,G} & \text{jos } (t > p_0) \wedge (U_{ji,G} > X_{j,\text{upp}}), \end{cases} \tag{31}$$

$X_{j,\text{low}}$ ja $X_{j,\text{upp}}$ ovat ennalta määritetyt populaation yksilöiden ylä- ja alarajat. Luku t on satunnainen arvo ϵ [0,1] ja p_0 on todennäköisyysarvo.

3.1.7. Ranking-perusteinen mutaatio-operaatio

Gong ym. (2013a) tutkimuksessa vektorit valitaan uuteen populaatioon suhteellisesti suuruusjärjestykseen perustuen. Tehtävässä järjestetään populaation jokainen vektori nousevassa järjestyksessä parhaasta huonompaan hyvyysarvojen perusteella. Vektorin ranking R_i valitaan seuraavasti:

$$R_i = Np - i, \qquad i = 1,2,\ldots,Np \tag{32}$$

Edellisen kaavan mukaan paras vektori nykyiseen populaatioon saadaan parhaan sijoituksen perusteella. Valinnan todennäköisyys lasketaan, kun on määritetty kaikkien vektorien suuruusjärjestys. Valintatodennäköisyys p_i on laskettu kohdevektorille X_i seuraavasti:

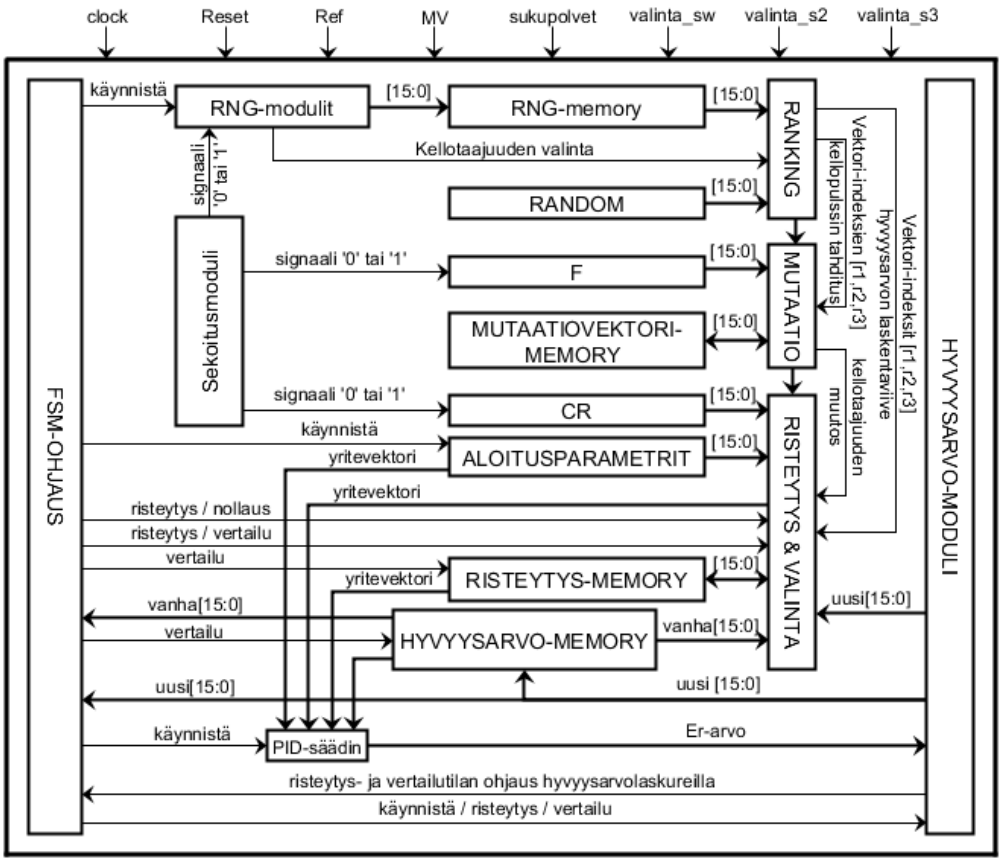
$$p_i = (R_i / Np)^2, \qquad i = 1,2,\ldots,Np \tag{33}$$

Vektorin valinnassa on ensin laskettu valintatodennäköisyys p_i jokaiselle vektorille. Mutaatio-operaattorissa vektorit pitäisi valita todennäköisyyden p_i mukaan. Ensin valitaan perusvektori Xr_1 ja differenssivektori Xr_2 ranking-perusteisesti, jossa päätepisteen valinta perustuu todennäköisyyteen. Toinen differenssivektorin päätepiste Xr_3 on valittu satunnaisesti kuten alkuperäisessä DE-algoritmissa. Ranking-perusteisessa jDE-algoritmissä ainoastaan toinen differentiaalivektorin päätepisteistä valitaan suuruusjärjestyksen perusteella, koska etsintäaskeleen suuruus differenssivektorissa voisi laskea liian nopeasti aiheuttaen ennenaikaista lähentymistä niiden vektori suuruuden ollessa lähes samanlaisia Gong ym. (2013b). Mutaatio-operaattorit erottaa DE-algoritmista ”DE/a/b” merkintätavalla, jossa kirjain *a* tarkoittaa vektorien olevan mutatoitu ja *b* tarkoittaa käytettyjen differenssivektorien määrää. Itse-adaptoituvan jDE-algoritmin mutaatio-operaattori on lausekkeessa (34).

jDE/rand/1/bin: $$V_\mathrm{i} = X_{r1} + F \times (X_{r2} - X_{r3}) \tag{34}$$

4. SÄÄTIMEN TOTEUTUS

4.1. DE-algoritmin ohjelmarakenne



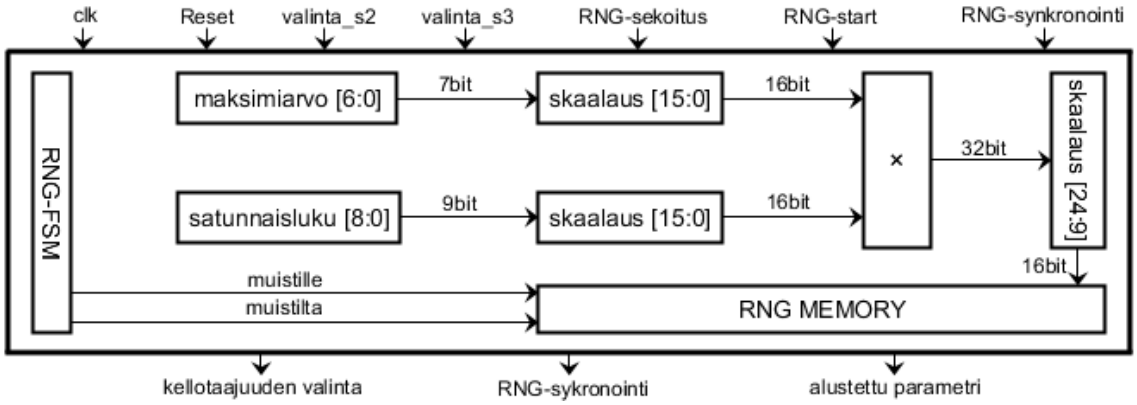
Kuva 13. DE-arkkitehtuurin ohjelmamodulit ja niiden väliset kytkennät

Ohjelma on toteutettu Altera DE2-35 Cyclone II tuoteperheen FPGA kehitysalustalle. Ohjelmassa on käytetty VHDL-kieltä ja toteutettu kiinteän pilkun kokonaislukuaritmetiikkaa ja lukujen skaalaamista oikeaan desimaalitarkkuuteen pääsemiseksi.

RNG-moduleita (RNG-modulit) on kaksitoista kappaletta, jossa jokainen satunnais-lukugeneraattori sisältää yhden yksilön parametrin. Sekoitusmoduli sekoittaa satunnaisarvoja satunnaislukugeneraattoreilla. RNG-modulit tuottavat kaikkiaan neljä yksilöä, jotka asetetaan suuruusjärjestykseen ranking-modulilla (ranking). Ranking-modulilla valitaan kaksi suurinta arvoa mutaatiolaskentaan (mutaatio) vertailemalla ja kolmas parametri valitaan mutaatiolaskentaan satunnaisesti random-modulilla (random). Random-moduli tuottaa kolmekymmentäkaksi parametria, joista yhden se syöttää

satunnaisen parametrivalinnan päätteeksi ranking-vektorin kolmanneksi parametriksi. Mutaatiovektorimodulilla (mutaatio) lasketaan ranking-vektorin parametrien ja mutaatiovakion (F) avulla mutaatiovektorin arvo. Mutaatiovektorimuistia (mutaatio-memory) tarvitaan apumuistina, kun mutaatiolaskennasta sarjasiirrolla lasketut mutaatioparametrien arvot muunnetaan rinnakkain mutaatiovektoriksi. Sarjasiirrosta rinnakkaismuotoiseen siirtoon muuntaminen hidastaa optimointia kahdella kellopulssilla. Risteytyksessä risteytysvakiota (CR) verrataan vertailijan satunnaisarvoon. Risteytysvertailun perusteella valitaan mutaatioparametri tai kohdevektorin parametri uudeksi yriteparametriksi (risteytys ja valinta). Risteytysmuistissa (risteytys-memory) säilytetään risteytettyjä parametreja, jotka ajetaan PID-säätimen parametreiksi (yritevektori) ja risteytykseen (kohdevektori). Hyvyysarvomuistissa (hyvyysarvo-memory) säilytetään risteytetyillä yriteparametreilla PID-säätimellä ja hyvyysarvolaskennassa (hyvyysarvo-moduli) optimoituja optimiparametreja, jotka on valittu hyvyysarvojen vertailun perusteella. Lopullinen valinta tehdään, kun kaikki sukupolvet on optimoitu. Tilakoneen ohjauksia (FSM-ohjaus) voi verrata muihin tilakoneesta tehtyihin arkkitehtuureihin (kuva 16, 20 ja 21).

4.1.1. RNG- ja RANDOM-moduli



Kuva 14. RNG-arkkitehtuuri alustaa satunnaisarvot oikealle optimoinnin edellyttämälle kokonaislukualueelle ennen niiden yhdenaikaista syöttämistä laskentaan.

Populaation suuruudeksi on asetettu neljä yksilöä. Jokaisella yksilöllä on kolme parametria. Parametrien lukumäärä vastaa PID-säätimen Kp-, Ti- ja Td-parametreja. RNG-moduleiden sisäistä toimintaa ohjaavat tilakoneet (RNG–FSM) käynnistetään tilakoneelta FSM yhtäaikaisesti optimoinnin käynnistyessä (kuva 16). Tilakone FSM käynnistää kaikki RNG-modulit RNG-start-signaalilla (kuva 14). Tilakoneen RNG–FSM ollessa muistille -tilassa muisti (RNG-memory) tallentaa kerrotut ja skaalatut

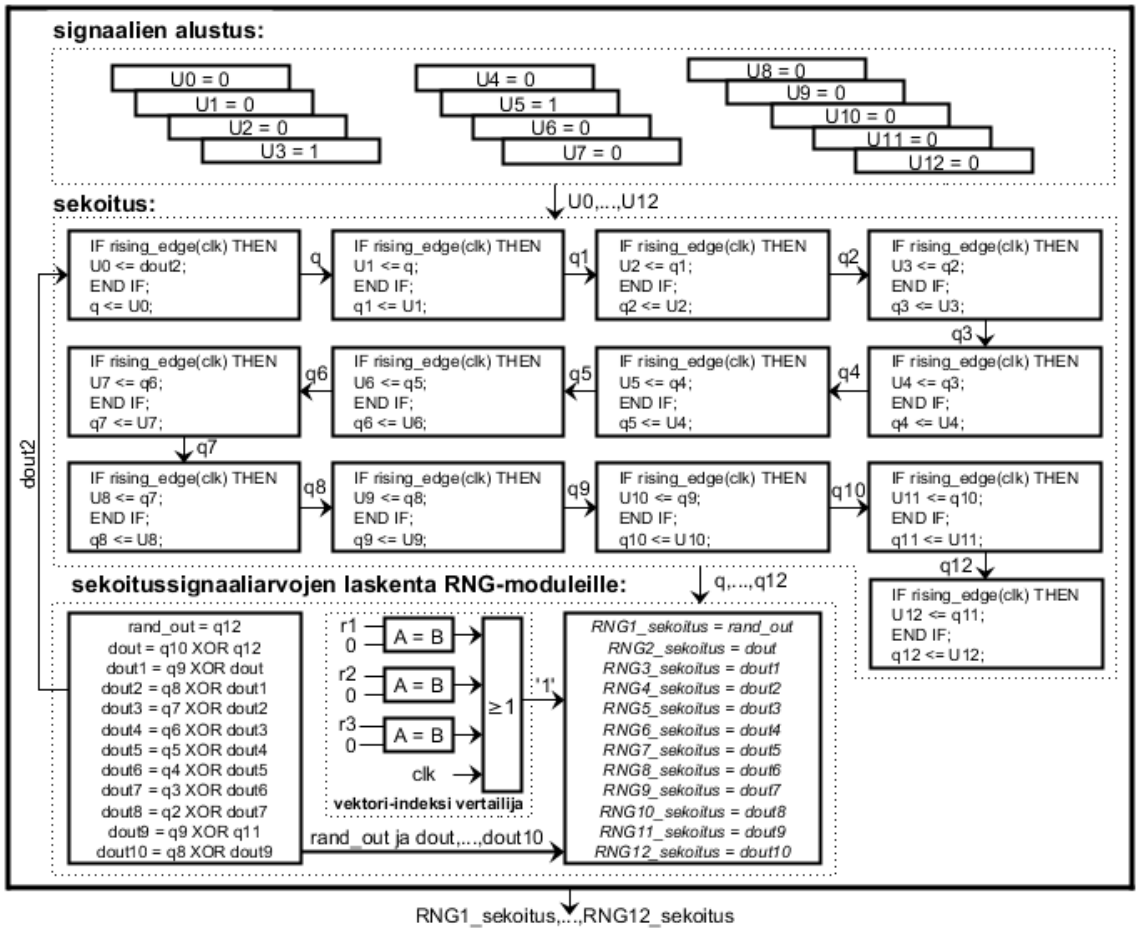
satunnaisparametrit muistiin. Tilakoneen RNG–FSM ollessa muistilta -tilassa se lähettää muistissa (RNG-memory) olevat satunnaisparametrit (alustettu parametri) seuraavaan laskentavaiheeseen. Alustetut parametrit on skaalattu 16 bittisiksi. Kaikkien satunnaisgeneraattoreiden (kuva 14) yhtäaikainen toiminta on tahdistettu samanaikaiseksi RNG-synkronointisignaalilla RNG-modulien tilakoneiden RNG–FSM välillä. Tahdistuksen avulla varmistetaan, että alustetut parametrit lähetetään samanaikaisesti satunnaislukugeneraattorien muistilta (RNG-memory) seuraavaan laskentavaiheeseen. RNG-arkkitehtuurin (kuva 14) satunnaislukujen luominen ja kertojien toiminta on estetty satunnaislukusekoituksen (sekoitusmoduli) ollessa päällä. RNG-sekoitussignaali pysäyttää RNG-modulit arvolla yksi.

RNG-moduleilla voidaan myös hidastaa optimointiprosessia valitsemalla hitaampi kellotaajuus valintakytkimien (valinta_s2 ja valinta_s3) avulla. DE-optimoinnin kellotaajuudeksi voidaan valita valintakytkimillä kolme eri taajuutta. Valintakytkinten ollessa poissa päältä optimointi toimii 50 % nopeudella. Optimointi toimii 33 %:in nopeudella valintakytkimen (valinta_s2) ollessa päällä. Kun valintakytkin (valinta_s3) on valittuna optimointi toimii 25 %:in nopeudella.

Alustetut etumerkittömät (unsigned) parametrit on skaalattu sopivalle kokonaislukualueelle (16bit) kiinteän pilkun aritmetiikalla. Sopivana desimaalien määränä ohjelmassa on käytetty yhdeksää desimaalibittiä. RNG-arkkitehtuurissa on ensin annettu maksimiarvo (”0001111”) yksilön parametrille, joka on skaalattu lisäämällä esimerkissä desimaalien määrä bittejä (”0001111” & ”000000000”) maksimiarvon jälkeen. Seuraavaksi yhdeksän bittinen satunnaisluku on skaalattu sopivalle kokonaislukualueelle lisäämällä esimerkissä satunnaisluvun eteen seitsemän bittiä (”0000000 & ”010010101”). Lopuksi skaalattu maksimiarvo ja skaalattu satunnaisluku on kerrottu keskenään, jonka jälkeen niiden tulos on skaalattu takaisin kokonaislukualueelle (skaalaus [24:9]) (kuva 14).

Alustetut satunnaisluvut toteutetaan random-modulilla vastaavasti kuin RNG-modulilla (kuva 14). Random-moduli ei vain sisällä tilakonetta ja muistia kuten RNG-moduli. Moduli sisältää kuitenkin multiplekserin (CASE-lauseke), jolla voidaan valita satunnaisesti jokin kolmestakymmenestäkahdesta satunnaislukugeneraattorilla alustetusta parametrista seuraavalle modulille. CASE-lauseke on valittu juuri tämän vuoksi ohjelman parametrien valintalausekkeeksi.

4.1.2. Satunnaisarvojen sekoitusmoduli



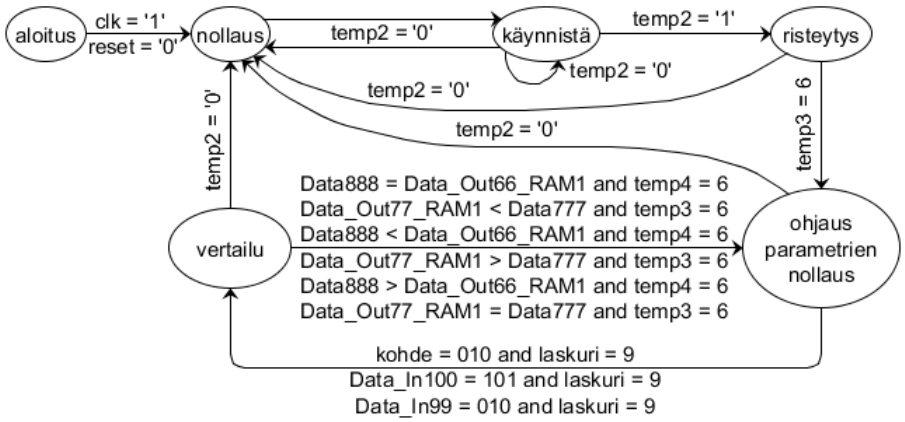
Kuva 15. Sekoitusmoduli sekoittaa kellotahdissa sekoitusosan lähtöarvoja (q,…,q12) sekoitussignaaliarvojen laskennasta tulevalla väliarvolla (dout2) ja lähettää sekoitusosan lähtöarvot takaisin laskentaan. Vektori-indeksivertailija ohjaa sekoitusmodulin toimintaa vektori-indeksiarvojen r1, r2 ja r3 mukaan jonkun indeksiarvoista ollessa nolla. Vertailija syöttää eri satunnaislukugeneraattoreiden sekoitussignaaliarvot laskennasta yhtäaikaisesti RNG-moduleille. Sekoitus alkaa, kun ohjelma syöttää alustetut signaalit (U0,…,U12) sekoitusosaan.

Sekoitusmodulille (kuva 15) on luotu sekoitin, jonka sekoitussignaaliarvoilla (RNG1_sekoitus,…,RNG12_sekoitus) ohjataan jokaisen RNG-modulin pysäytystä ja käynnistystä. Sekoitusmodulin toiminta perustuu satunnaislukugeneraattorien pysäyttämiseen eri ajanhetkellä, kun säätimen optimointiprosessi on poissa toiminnasta. Satunnaislukuarvojen sekoitus RNG-moduleilla toteutetaan sekoitusmodulin sekoitussignaaliarvojen ollessa yksi. RNG-modulin toiminta pysäytetään, kun kyseisen RNG-modulin sekoitussignaaliarvo on yksi muuten kyseinen RNG-moduli toimii normaalisti. Tällä tavalla voidaan sekoittaa RNG-modulien alustettuja parametriarvoja,

jotka muuten olisivat samanlaisia. Satunnaislukujen sekoitin toimii ja optimointiprosessi on pysähtynyt, kun vektori-indeksien r1, r2 ja r3 arvoista yksi tai useampi arvo on nolla. Satunnaislukujen sekoituksen ollessa toiminnassa kaikkien RNG-modulien toimintaa voidaan sekoittaa niiden oman sekoitussignaaliarvon mukaan (kuva 15).

Sekoitusmodulin rakenne (kuva 15) on jaettu kolmeen osaan. Sekoitusmodulin osat ovat signaalien alustus, sekoitus ja sekoitussignaaliarvojen laskenta RNG-moduleille. Signaalien alustuksessa annetaan halutut aloitusarvot sekoitusosan signaaleille. Aloitusarvoja pystytään muuttamaan eri tilanteissa. Sekoitus alkaa toimia, kun alustetut signaalit syötetään sekoitusosaan (sekoitus). Sekoitusosassa sekoitetaan sen lähtöarvoja (q,…,q12) laskentaosasta tulevalla sekoitussignaalin väliarvolla (dout2). Sekoitusosan lähtöarvot syötetään sekoitussignaaliarvojen laskentaan, jossa niitä sekoitetaan sekoitussignaalin väliarvojen kanssa lisää poissulkevia XOR-portteja käyttäen. Sekoitussignaaliarvojen laskennassa vektori-indeksien vertailijan ohjaussignaalin ollessa arvossa yksi sekoitusmoduli käynnistää sekoitussignaaliarvojen (RNG1_sekoitus,…,RNG12_sekoitus) syötön sekoitussignaaliarvojen mukaisille RNG-moduleille (RNG1,…,RNG12). Sekoitussignaaliarvot muuttuvat kellopulsseittain.

4.1.3. Tilakone FSM



Kuva 16. Tilakone FSM-arkkitehtuuri ohjaa optimoinnin kulkua. FSM nollaa ohjaukset ennen optimoinnin alkua, käynnistää optimoinnin ja risteytyksen, nollaa risteytyksen ja hyvyyslaskennan ohjausarvot ja tekee sukupolvittaisen hyvyysarvojen vertailun vaiheittain oikeassa järjestyksessä. Optimointi käynnistyy ja pysähtyy ohjauksen (temp2) mukaan.

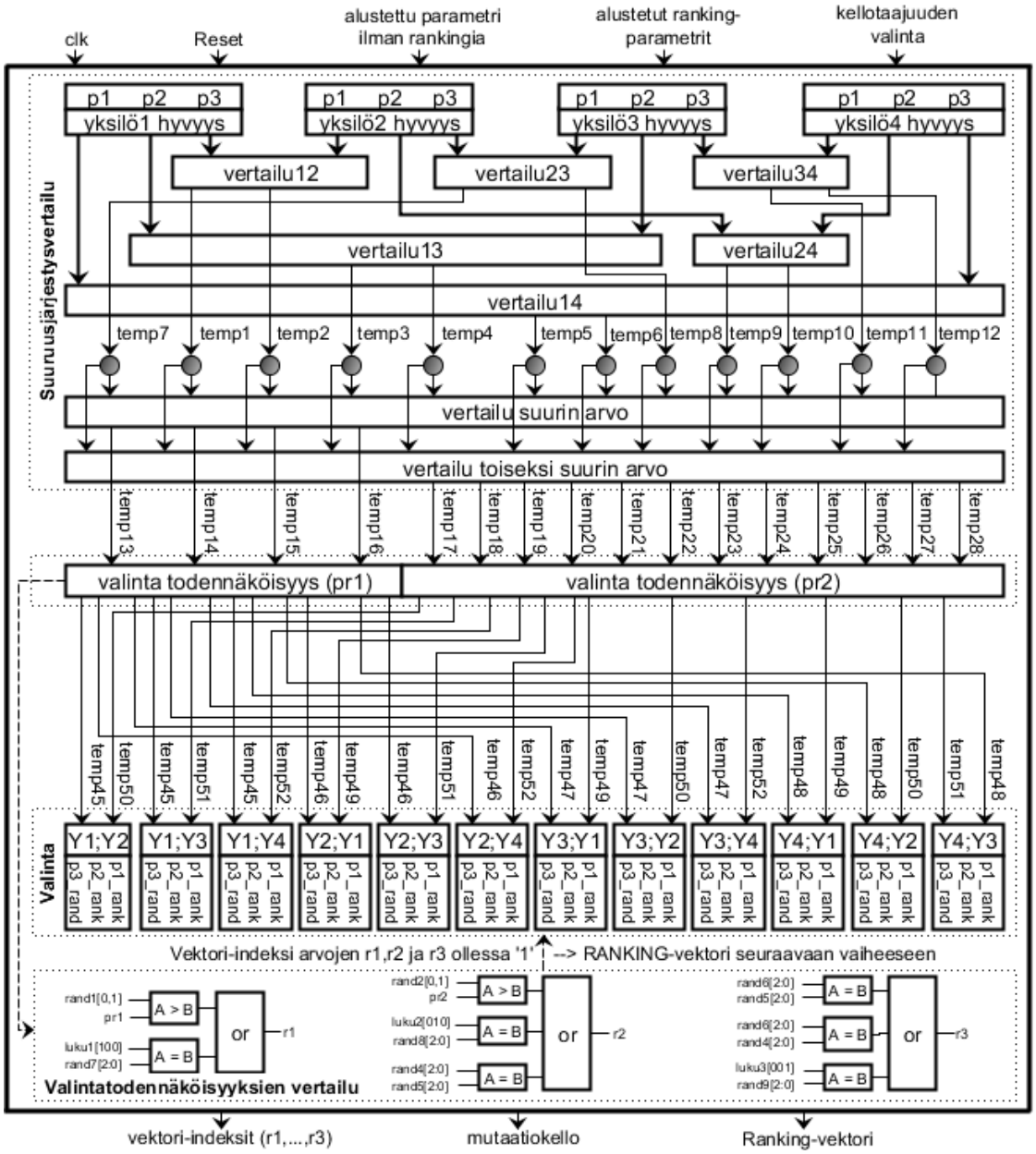
Tilakoneella ohjataan ohjelman kulkua optimoinnin käynnistymisestä siihen hetkeen kun sukupolvia on haluttu määrä (kuva 16). Nollaustilassa kaikki ohjaussignaalit nollataan. Käynnistä-tilassa käynnistetään kaikki RNG-modulit yhtä aikaa, syötetään

aloitusparametrit PID-säätimelle, estetään risteytys aloitusparametreilla ajettaessa ja käynnistetään hyvyysarvon laskenta. Risteytystilassa aloitusparametrit suljetaan ja käynnistetään risteytys sekä syötetään ensimmäiset yriteparametrit PID-säätimelle ja aloitetaan ensimmäisillä yriteparametreilla hyvyysarvon laskenta. Ohjausparametrien nollaustilassa nollataan hyvyyslaskennan ohjausparametrit. Vertailu tapahtuu hyvyyslaskennan mukaan. Vertailutilassa vertaillaan edellisen ja nykyisen hyvyyslaskennan arvoa. Hyvyyslaskennassa integroidaan kumulatiivisesti PID-säätimen erotusarvoa sukupolvittain. Hyvyysarvo sukupolvi sisältää tietyn määrän mittausnäytteitä. Tilakone suorittaa vertailun, näytteiden määrän ollessa täynnä. Tilakoneen nollaustila ja vertailutila toistuvat sukupolvien määrän mukaan. Sukupolvien määrän täyttyessa tilakone palaa nollaustilaan ja optimointi käynnistyy esimerkiksi ohjearvon muutoksesta.

4.1.4. Ranking-moduli

Ranking-arkkitehtuuri (kuva 17) parantaa DE-algoritmin toimintaa ja optimointi-nopeutta järjestämällä satunnaisesti luodut yksilöt suuruusjärjestykseen. Valitsemalla suurimmat yksilöt seuraavaan vaiheeseen pyritään välttämään liian nopeaan paikalliseen optimiin joutumista. Aluksi muodostetaan RNG-moduleilta tulevista alustetuista parametreista (alustetut ranking-parametrit) neljä yksilöä, jonka jälkeen niiden suuruuksia vertaillaan keskenään. Kahdesta suurimmasta yksilöstä valitaan parametrit Ranking-vektorin kahdeksi ensimmäiseksi parametriksi. Ranking-vektorin kolmas parametri valitaan random-modulilla (alustettu parametri ilman rankingia). Vertailun jälkeen kahdelle suurimman yksilön parametrille tehdään valintatodennäköisyyden vertailu, jonka perusteella valitaan ranking-vektori mutataatiolaskentaan.

Valintatodennäköisyyksien vertailussa selvitetään vektori-indeksiarvot (r1, r2 ja r3). Suuruusjärjestysvertailusta kaksi suurinta arvoa otetaan valintatodennäköisyyksien vertailuun, jossa molemmille yksilöille lasketaan valintatodennäköisyydet (pr1) ja (pr2). Tämän jälkeen valintatodennäköisyyksiä (pr1) ja (pr2) vertaillaan satunnaisarvoihin (rand1) ja (rand2) vektori-indeksiarvojen (r1) ja (r2) valinnassa. Lisäksi vektori-indeksiarvojen (r1) ja (r2) ja (r3) valinta sisältää yhtäsuuruusvertailuja eri satunnaisluvuilla ja binääriluvuilla. Satunnaisluvuille ja binääriluvuille on vertailussa annettu kolme bittinen arvo. Lopullinen ranking-vektorin valinta tehdään vektori-indeksiarvojen ja suuruusjärjestysvertailun mukaan. Mutaatikellolla tahdistetaan ranking-arkkitehtuurin ja mutaatiolaskenta-arkkitehtuurin toiminta.



Kuva 17. Ranking-arkkitehtuurin suuruusjärjestysvertailu valitsee Ranking-parametrit yksilöiden (Y1,…,Y4) parametreista (p1,…,p3) signaalien (temp1,…,temp52) avulla. Valinta-hetki toteutuu valintatodennäköisyysvertailun vektori-indeksiarvojen r1, r2 ja r3 mukaan.

Ranking-arkkitehtuurissa (kuva 17) vertaillaan jokaisen yksilön hyvyysarvoa. Hyvyysarvo saadaan laskemalla kaikki yksilön parametrit yhteen. Jokaisesta vertailusta valitaan suurin arvo ja toiseksi suurin arvo esimerkiksi (vertailu12). Suurimman arvon vertailuja on aina kolme per yksilö, jonka perusteella valitaan suurin yksilö valintatodennäköisyyden (pr1) laskentaan. Toiseksi suurimman arvon vertailussa

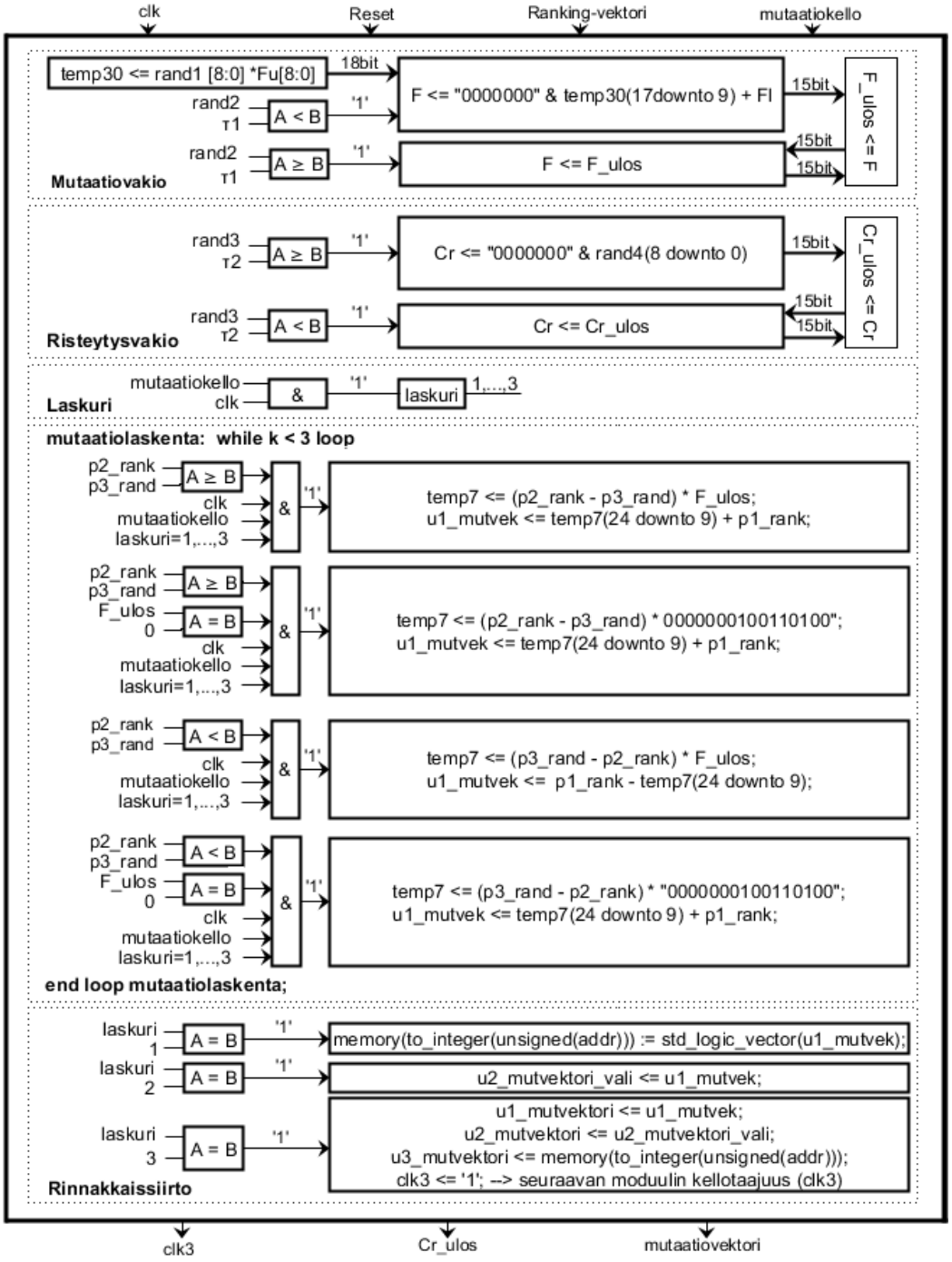
jokaisella yksilöllä on kolme vertailua toiseksi suurimmaksi arvoksi eli kaikkiaan kaksitoista eri vertailua, joiden perusteella yksi valitaan valintatodennäköisyyden (pr2) laskentaan. Suuruusvertailussa pitää olla aina selvillä mikä yksilö (Y1…Y4) on suurin ja toiseksi suurin. Vertailussa on esimerkiksi päädytty seuraavaan suuruusjärjestykseen kahdestatoista vaihtoehdosta (Y4;Y1) (kuva 17), näiden perusteella valitaan (valinta) differenssivektorin perusvektori (p1_rank) sekä ensimmäinen päätepiste (p2_rank) että toinen päätepiste (p3_rand). Differenssivektorin toinen päätepiste (p3_rand) valitaan aina satunnaisesti ilman suuruusvertailua.

Ranking-vertailun suurimmalle ja toiseksi suurimmalle yksilölle lasketaan valintatodennäköisyydet $pr_j \in \{1,2\}$, joiden arvot kuuluvat laskentakaavojen (32) ja (33) mukaan $\in [0,1]$. Tämän jälkeen valintatodennäköisyyksiä vertaillaan satunnaislukujen $rand_j \in \{1,2\}$ avulla, joissa satunnaisluvut ovat $\in [0,1]$. Tämän lisäksi vertaillaan kolmebittisien binäärilukujen yhtäsuuruuksia $luku_j \in \{1,2,3\}$, joiden arvot ovat $\in \{1,2,4\}$ kolmebittisiin satunnaislukuihin $rand_j \in \{7,8,9\}$, joiden arvot ovat $\in [0,7]$. Lisäksi vertaillaan seuraavien satunnaislukujen $rand_j \in \{4,5,6\}$ yhtäsuuruuksia, jossa satunnaislukujen arvot ovat $\in [0,7]$. Vertailtavien lukujen suuruus $luku_j \in \{1,2,3\}$ ja $rand_j \in \{4,5,6,7,8,9\}$ perustuu yksilöiden määrään. Kyseisien vertailujen avulla saadaan vektori-indeksien $r1 \in \{0,1\}$, $r2 \in \{0,1\}$ ja $r3 \in \{0,1\}$ arvo selville valintatodennäköisyysvertailussa. Valitaan vektori-indeksien ja suuruusjärjestys-vertailun (valinta) perusteella ranking-vektoreille (kuva 17) perusvektori (p1_rank) ja ensimmäinen päätepiste (p2_rank) sekä toinen päätepiste (p3_rand). Vektori-indeksien jonkun arvon ollessa nolla pysähtyy koko optimointiprosessi kunnes kaikkien vektori-indeksien arvo on yksi.

4.1.5. Mutaatiovektorimoduli

Mutaatiovektoriarkkitehtuurilla (kuva 17) toteutetaan itse-adaptoituvan mutaatiovakion (F), risteytysvakion (CR), mutaatiolaskennan ja mutaatiolaskennasta sarjassa laskettujen parametrien rinnakkaissiirto risteytysmodulille (kuva 18). Mutaatiovakio lasketaan kertomalla ensin mutaatioalueen maksimiarvo $F_\mu = 0,9$ satunnaisarvolla (rand1), joka jälkeen väliarvon (Fv) tulos skaalataan takaisin kokonaislukualueelle. Tämän jälkeen väliarvoon lisätään mutaatioalueen minimiarvo $F_l = 0,1$ ja skaalataan sopivalle kokonaislukualueelle lisäämällä tuloksen eteen nollia. Itse-adaptoituvan mutaatiovakion laskentatulos perustuu vertailuun, jossa satunnaislukua (rand2) ja vertailuarvoa $\tau_1 = 0,1$ verrataan. Jos vertailussa $rand2 < \tau_1$ päivitetään mutaatiovakio muuten

mutaatiolaskennassa käytetään vanhaa mutaatiovakion arvoa. Itse-adaptoituvan mutaatiovakion laskennassa käytetään laskentavaiheiden välillä arvoa F_ulos, kun mutaatiovakion laskettu arvo syötetään mutaatiolaskentaan (kuva 18).



Kuva 18. Mutaatiolaskenta-arkkitehtuurissa lasketaan itse-adaptoituva mutaatio- ja risteytysvakio. Mutaatioparametria (F) käytetään mutaatiovektorin satunnaisessa laskennassa, jossa mutaatiokellon, laskurin ja ranking-parametrien (p2_rank) ja (p3_rand) vertailun avulla luodaan positiiviset arvot mutaatioparametreille, jotka syötetään rinnakkain risteytykseen.

Risteytysvakio lasketaan skaalaamalla satunnaisarvon (rand4) tulos sopivalle kokonaislukualueelle lisäämällä tuloksen etupuolelle nollia. Itse-adaptoituvan risteytysvakion laskentatulos perustuu satunnaisluvun (rand3) ja vertailuarvon $\tau_2 = 0{,}1$ vertailuun. Jos vertailussa rand3 $\geq \tau_2$ päivitetään risteytysvakio muuten käytetään vanhaa risteytysvakion arvoa. Itse-adaptoituvan risteytysvakion laskennassa käytetään arvoa Cr_ulos, kun se syötetään risteytykseen (kuva 18).

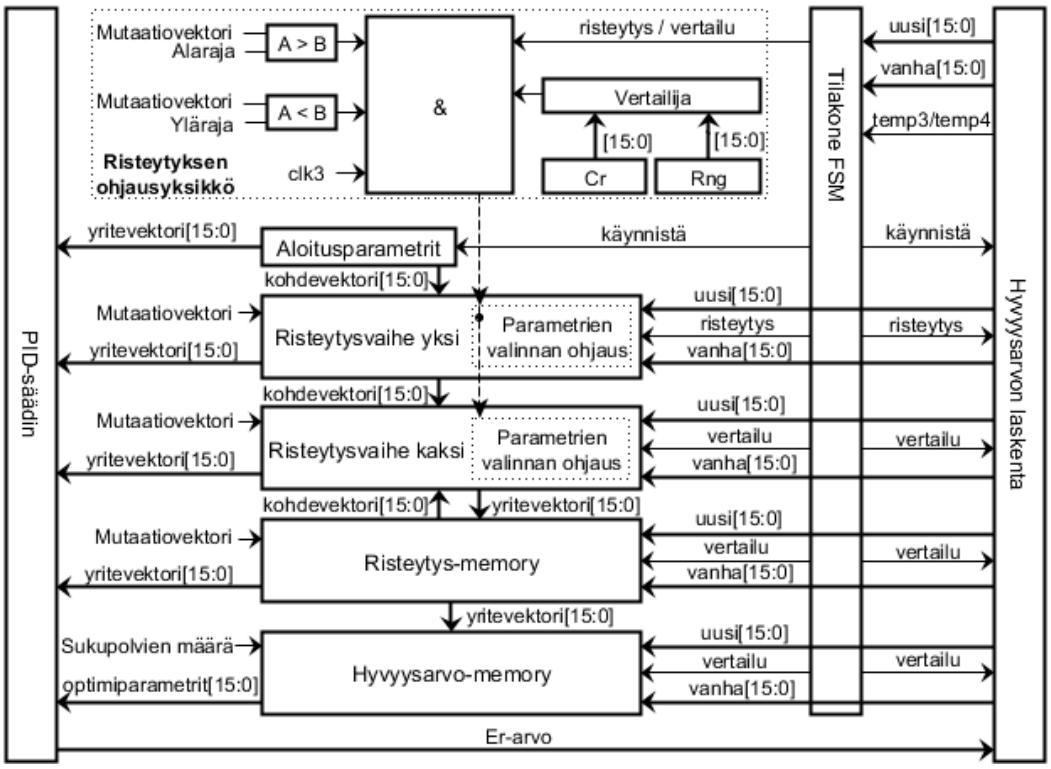
Tahdistuslaskuri (Laskuri) pitää mutaatiolaskennan sarjassa lasketut parametrit ja mutaatioparametrien rinnakkaissiirron tahdissa ranking-arkkitehtuurista tulevien ranking-vektorien parametrien kanssa. Laskuri pysyy laskentatahdissa mutaatiokellon ja kellopulssin avulla (clk). Rinnakkaissiirto perustuu laskurin toimintaan. Sen arvoalue on asetettu mutaatiovektorin parametrien määrän mukaan (kuva 18). Laskuri ohjaa ja lähettää mutaatiovektorin mutaatiolaskentaan laskurin ollessa arvossa kolme.

Mutaatiolaskennassa (mutaatiolaskenta) on käytetty while-silmukkaa laskemaan mutaatiovektorille kolme parametria silmukoiden lukumäärän mukaan. Ranking-vektorin (differenssivektori) kaikki parametrit syötetään yhtäaikaa mutaatiolaskentaan. Mutaatiolaskenta laskee yhden kellopulssin (clk) ja mutaatiokellosignaalin aikana yhden mutaatiovektorin parametrin. Mutaatiokellon arvon ollessa yksi syötetään ranking-vektorin arvo mutaatiolaskentaan. Mutaatiovektoriin vaaditaan kolme ranking-vektoria, joiden määrä lasketaan laskurin avulla (kuva 18). Peräkkäin sarjasiirrolla lasketut parametrit lähetetään rinnakkain (rinnakkaissiirto) yhtäaikaisesti risteytykseen (kuva 18). Mutaatiolaskenta (kuva 18) perustuu vertailuun, jota ohjataan mutaatiokellolla, kellopulssilla, laskurilla, mutaatiovakion vertailulla ja ranking-vektorin (differenssivektori) ensimmäisen päätepisteen (p2_rank) ja satunnaisesti muodostetun päätepisteen (p3_rand) suuruusvertailulla. Ranking-vektorin ensimmäisen päätepisteen (p2_rank) ja satunnaisesti muodostetun päätepisteen (p3_rand) vertailulla pystytään pitämään niiden välinen erotus positiivisena laskutoimituksessa. Näin negatiivisilta mutaatioparametrien arvoilta vältytään laskennassa. Lisäksi mutaatiovakion väliarvon (F_ulos) mennessä nollaan annetaan laskennassa mutaatiovakiolle positiivinen vähän nollan yläpuolella oleva arvo, joka on skaalattu sopivalle kokonaislukualueelle (kuva 18). Mutaatiovektorin parametrin laskennassa on käytetty yhtä välivaihetta (temp7) (mutaatiolaskenta). Välivaiheessa ranking-vektorin ensimmäisestä päätepisteestä on vähennetty satunnaisesti muodostettu toinen päätepiste tai toisin päin vertailun mukaan ja kerrottu erotusarvon tulos mutaatiovakiolla. Välivaihe (temp7) on tämän jälkeen skaalattu sopivalle kokonaislukualueelle ja lisätty

tulokseen ranking-vektorin perusvektori. Tällä tavalla on laskettu mutaatiovektorin parametrille arvo (u1_mutvek) (mutaatiolaskenta).

Rinnakkaissiirrossa muodostetaan kolme parametria sisältävä mutaatiovektori. Sen kolmas parametri on ensin laskettu muistiin laskentasilmukan ja laskurin arvon ollessa (laskuri = 1 ja k = 0). Mutaatiovektorin toinen parametri on laskettu väliarvona laskurin ja laskentasilmukan arvon ollessa (laskuri = 2 ja k = 1). Mutaatiovektorin ensimmäinen parametri on laskettu laskurin ja laskentasilmukan arvon ollessa (laskuri = 3 ja k = 2). Samalla hetkellä kun laskurin arvo on tullut arvoon kolme syötetään kaikki edellä lasketut parametrit rinnakkain risteytysvaiheeseen (rinnakkaissiirto) (kuva 18). Mutaatiolaskennan laskurin arvolla (laskuri = 3) tahdistetaan myös risteytys- ja valinta-arkkitehtuurin kellotaajuus (clk3) mutaatiolaskenta-arkkitehtuurin kanssa.

4.1.6. Risteytys ja valinta



Kuva 19. Risteytys- ja valinta-arkkitehtuuri vertailee parametreittain risteytyksen ohjausyksiköllä risteytysvakiota (Cr) satunnaisarvoa (Rng), joilla ohjataan yriteparametrien valintaa risteytysvaiheissa yksi ja kaksi. Risteytysvaiheen kaksi arvo tallennetaan risteytysmuistiin, josta parametrit palautetaan takaisin risteytysvaiheeseen kaksi risteytettäviksi (kohdevektori) ja PID-säätimen yritevektoriksi. Parhaat yritteet talletetaan hyvyysarvomuistiin, josta sukupolvien määrän täytyttyä syötetään PID-säätimen optimiparametreiksi.

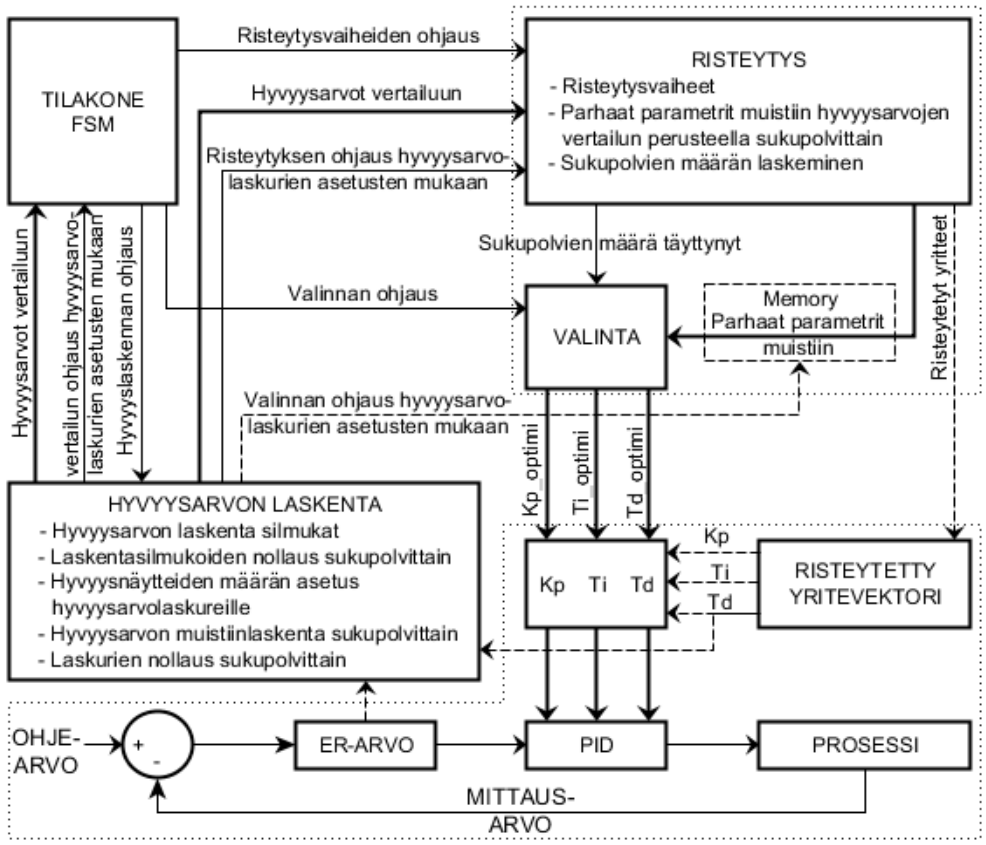
Risteytysvaiheessa valitaan joko mutaatiovektori tai kohdevektori PID-säätimen uusiksi yriteparametreiksi (kuva 19). Risteytystapahtumaa ohjataan tilakoneella (FSM) hyvyysarvolaskurien temp3 ja temp4 asetuksen mukaan (kuva 16, 19 ja 20). Risteytys aloitetaan (risteytyksen ohjausyksikkö) vertailemalla (vertailija) risteytysvakion CR arvoa vertailevaan satunnaislukugeneraattorin RNG-arvoon. Itse-adaptoituvan risteytysvakion CR ja vertaileva satunnaisarvo ovat välillä ϵ [0,1]. Mutaatiovektori valitaan uuteen sukupolveen, kun RNG-arvo $\leq$ CR. Muutoin valitaan kohdevektori uuteen populaatioon. Risteytys ei onnistui, jos mutaatiovektorin parametrien yläraja tai alaraja ylittyy. Risteytyksen ohjausyksiköltä ohjataan valintasignaalien mukaan parametrien valinnan ohjausta (kuva 19) parametrikohtaisesti kellopulssin tahdissa (clk3). Valintasignaaleilla valitaan mutaatio- tai kohdevektorin parametri risteytysvaiheessa yksi ja kaksi.

Tilakone (FSM) menee käynnistä-tilaan (kuva 16), jossa tilakoneen ohjaus (käynnistä) käynnistää PID-säätimen ja hyvyysarvon laskennan sekä ohjaa aloitusparametrien syöttön PID-säätimelle ja risteytysvaiheeseen yksi. Tilakoneen ohjaus (risteytys) käynnistää risteytysvaiheen yksi ja uudestaan hyvyysarvon laskennan, kun ensimmäisen sukupolven hyvyysarvo on laskettu aloitusparametreilla (kuva 16 ja 19). Risteytysvaiheessa yksi aloitusparametrit (kohdevektori) risteytetään mutaatiovektorin parametreilla parametrien valinnan ohjauksen mukaan. Tilakoneen ohjaus (vertailu) käynnistää risteytysvaiheen kaksi ja uudestaan hyvyysarvon laskennan, kun risteytysvaiheen yksi hyvyysarvosukupolvi on laskettu ja uusi ja vanha hyvyysarvo on vertailtu tilakoneella (FSM). Risteytysvaiheessa kaksi suoritetaan risteytys risteytysvaiheen yksi parametreille (kohdevektori), jossa ne risteytetään mutaatiovektorin kanssa parametrien valinnan ohjauksen mukaan. Risteytysvaiheen kaksi jälkeen yritevektori otetaan muistiin (risteytys-memory) uuden ja vanhan hyvyysarvon sukupolvittaisen vertailun perusteella (kuva 16 ja 19). Samalla hetkellä hyvyysarvolaskurien (temp3) tai (temp4) ollessa asetetussa arvossa (kuva 16 ja 21) parametrit (yritevektori) syötetään PID-säätimen parametreiksi. Tämän jälkeen muistissa olevat parametrit (kohdevektori) syötetään takaisin risteytysvaiheeseen kaksi, jossa ne risteytetään mutaatiovektorin parametrien kanssa parametrien valinnan ohjauksen mukaan. Risteytyksen suoritus toistuu risteytysvaiheen kaksi ja risteytysmuistin (risteytys-memory) välillä kunnes kaikki sukupolvet on optimoitu. Hyvyysarvomuistiin (hyvyysarvo-memory) talletetaan aina sukupolvittaisen hyvyysarvovertailun perusteella parhaat parametrit (yritevektori). Paras hyvyys saavutetaan uuden hyvyysarvon ollessa vanhaa pienempi. Kun sukupolvet on laskettu,

syötetään parhaan hyvyysarvon omaavat parametrit (optimiparametrit) PID-säätimelle. Lopuksi ohjaus (vertailu) pysäyttää optimoinnin (kuva 16, 19 ja 20).

Parametrien valinnan ohjauksella valitaan joko kohde- tai yriteparametri jatkamaan sukua seuraavan sukupolven yritteeseen (yritevektori) risteytysohjauksen vertailun perusteella (Risteytyksen ohjausyksikkö) (kuva 19). Parametrien valinnan ohjausvertailu tarvitaan, mikäli risteytyksen ohjausyksiköltä tulevat valintasignaalit valitsevat yksilön jokaiseksi parametreiksi kohdevektorin parametrin. Tälläisessä tapauksessa pakotetaan parametrien valinnan ohjauksella yhdeksi yriteparametriksi mutaatiovektorin parametri. Muutoin yriteparametrien valinta tapahtuu risteytyksen ohjausyksikön verailun perusteella (kuva 19).

4.1.7. Risteytettyjen parametrien hyvyysarvon laskeminen



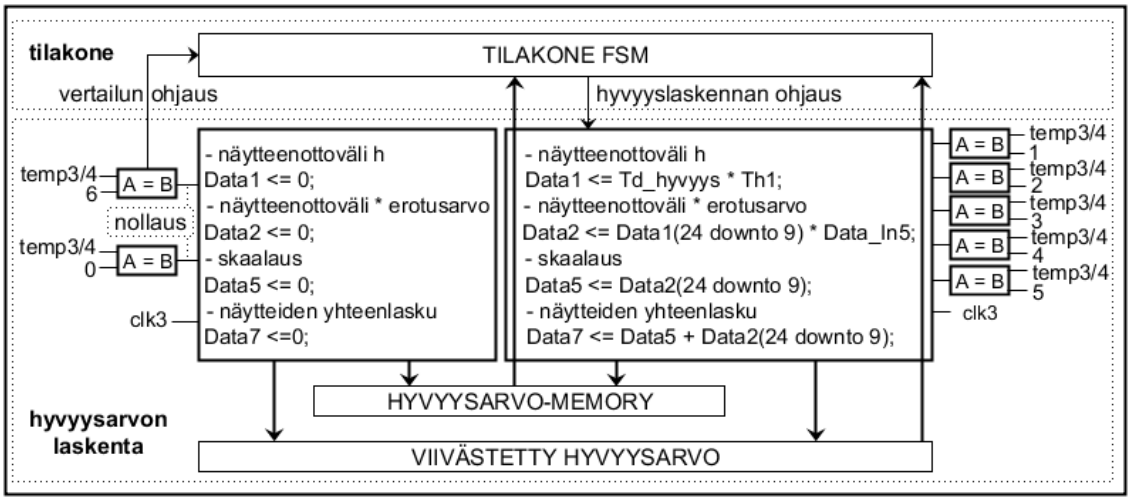
Kuva 20. Hyvyysarvo- ja risteytysarkkitehtuuri esittää miten tilakone FSM ohjaa risteytystä ja valintaa hyvyysarvolaskennan mukaan. Optimointi tapahtuu risteytetyillä yritteillä (Kp, Ti ja Td) sukupolvittain. Hyvyysarvo lasketaan Td-parametrista, er-arvosta ja näytteenottoajasta. Parhaat parametrit talletetaan muistiin risteytettyjen yritteiden hyvyyden perusteella. Optimiparametrit (Kp_optimi, Ti_optimi ja Td_optimi) valitaan sukupolvien määrän täytyttyä.

Hyvyysarvolaskennan etenemistä seurataan laskureilla (kuva 16, 20 ja 21). Ohjelmassa apuna käytetään myös muisteja, joihin hyvyysarvot tallennetaan hyvyysarvolaskureiden asetusmäärän mukaan sukupolvittain. Ohjelmassa hyvyysarvo lasketaan kahdella silmukalla, koska se selkeyttää hyvyysarvojen ja risteytyksen laskentaa. Hyvyyslaskennan arvot (Data1, Data2, Data5 ja Data7) nollataan ennen, kuin uudet yriteparametrit syötetään hyvyyslaskentaan hyvyysarvolaskurien arvon mukaan (kuva 21). Ohjelman hyvyysarvolaskuri nollautuu tilakonevertailussa, jos sitä ei valita vertailun perusteella seuraavan sukupolven hyvyysarvolaskuriksi. Muuten hyvyysarvolaskuri alkaa laskea arvosta yksi ylöspäin (kuva 21). Hyvyyslaskenta päättyy hyvyysarvolaskurin arvon ollessa viisi. Tilakone (FSM) vertailee vanhoja (hyvyysarvo-memory) ja uusia (viivästetty hyvyysarvo) hyvyysarvolaskurin arvon ollessa arvossa kuusi (kuva 21). Uuden sukupolven hyvyysarvon laskenta alkaa vertailun jälkeen.

Hyvyysarvon laskenta perustuu ohje- ja mittausarvosta laskettuun erotusarvoon (er-arvo), näytteenottoaikaan T ja yritevektorin derivointiparametrin Td laskentaan (ohjelma 29). Hyvyysarvon laskentaa ohjaa tilakone (FSM), jossa hyvyysarvojen vertailua ohjataan hyvyysarvolaskurien asetusarvon mukaan (kuva 16, 19 ja 20). Tilakone (FSM) ohjaa myös risteytyksen toimintaa, jossa eri risteytysvaiheet (kuva 19 ja 20) tapahtuvat samanaikaisesti hyvyysarvolaskurien asetusmäärien mukaisesti. Risteytyksen yriteparametrit syötetään optimoinnin aikana PID-säätimelle sukupolvien määrän mukaan. Optimoinnin lopuksi tilakone ohjaa optimiparametrien valintaa. Optimoinnissa sukupolvien määrä lasketaan hyvyysarvolaskurien asetusten mukaan, jossa hyvyysarvolaskurien (temp3) tai (temp4) tullessa vertailuvaiheen asetusarvoon kuusi (kuva 16) lasketaan nousevalla reunalla sukupolvien määrää optimoinnin aikana. Optimoinnin aikana parhaat parametrit talletetaan muistiin (Memory) vertailemalla vanhoja ja uusia hyvyysarvoja toisiinsa sukupolvittain (kuva 16 ja 20). Muistiin tallentaminen perustuu hyvyysarvolaskurien toimintaan. Optimiparametrien valinta tapahtuu ja optimointi päättyy, kun sukupolvien määrä on täyttynyt ja ohjausparametrit on nollattu. Optimoinnin päätyttyä optimiparametrit syötetään PID-säätimelle.

Tilakone (FSM) vertailussa vertaillaan edellistä hyvyysarvoa uuteen hyvyysarvoon (kuva 16). Hyvyysarvon laskennan loputtua laskettu hyvyysarvo ajetaan sekä hyvyys-arvomuistiin (hyvyysarvo-memory) että tilakoneen vertailuun viivästämällä hyvyysarvoa (viivästetty hyvyysarvo). Hyvyysarvon viivästyksessä hyvyysarvon tulosta siirretään yhden kellopulssin eteenpäin niin, että vanhaa muistilta tulevaa ja uutta laskennasta tulevaa hyvyysarvoa voidaan vertailla. Hyvyysarvolaskurien (temp3) ja

(temp4) avulla lasketaan sukupolvikohtainen laskentakertojen määrä. Tällöin hyvyysarvolaskurien sukupolvittainen laskentakertojen määrä on asetettu haluttuun arvoon (kuva 20 ja 21). Hyvyysarvolaskurien arvo on asetettu niin suureksi, että erosuure ehtii minimoitua hyvyyslaskennan aikana. Optimoinnin hyvyyslaskennassa hyvyysarvolaskurien (temp3) ja (temp4) laskentaväli on asetettu välille yksi ja viisi (kuva 21). Hyvyysarvon laskenta tapahtuu laskurien arvon ollessa laskentavälillä. Laskurien laskennan aloitus on määritelty alkavaksi arvosta yksi. Tilakoneen vertailua ohjataan hyvyysarvolaskureiden (temp3) ja (temp4) arvon mukaan (kuva 16 ja 21). Hyvyysarvon laskentaparametrien nollausta ohjataan suoraan hyvyysarvomodulilta laskureiden temp3 ja temp4 laskenta-arvon mukaan (kuva 21).



Kuva 21. Hyvyysarvon laskentasilmukat laskevat tilakoneen FSM vertailun ohjaamana sukupolvittaisen hyvyysarvon (Data7), minkä ne tallentavat sekä suoraan hyvyysarvomuistiin laskennan päätyttyä että viivästävät hyvyysarvoa ennen tilakoneella tapahtuvaa vertailua. Vertailu tapahtuu edellisen sukupolven hyvyysarvon ja nykyisen viivästetyn hyvyysarvon kesken. Laskenta tehdään Hyvyysarvolaskureille (temp3) ja (temp4) asetettujen laskentakertojen määrän mukaan. Hyvyysarvolaskennan parametrit nollataan (nollaus) laskennan päätyttyä sukupolvittain hyvyysarvolaskurien mukaan.

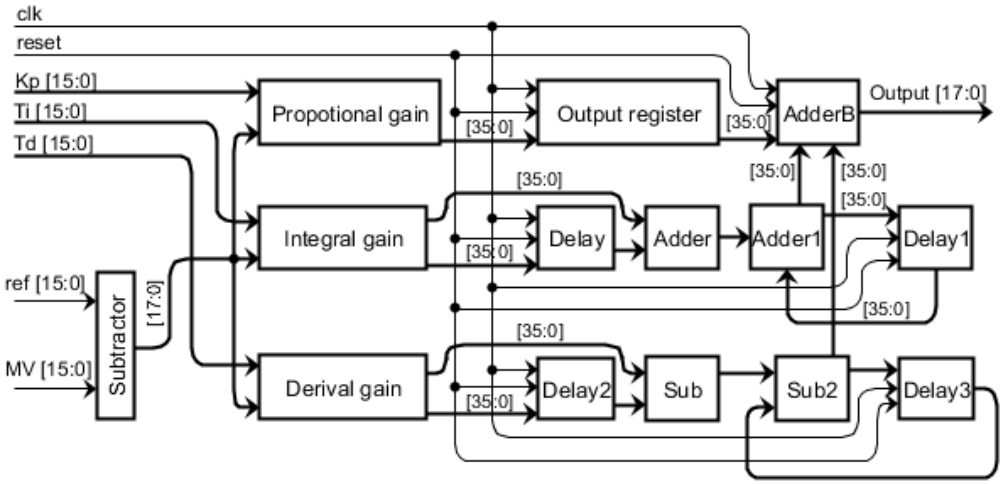
PID-säätimen parametrille lasketaan hyvyysarvolaskennassa sukupolvien määrän mukainen hyvyysarvo sukupolvittain (kuva 21 ja 22). PID-säätimen hyvyyslaskenta toimii 50MHz kellotaajuudella, jossa näytteenottoväli (T) on asetettu n. 2 ms:iin. PID-säätimen laskenta tapahtuu kellopulssin tahdissa (clk) sukupolvittaisessa optimoinnissa. PID-säädin ottaa erotusarvonäytteitä noin 25 kertaa sukupolvessa ja laskee näytteiden perusteella jokaiselle näytteelle hyvyysarvon. Hyvyysarvon laskennassa kaikkien näytteiden hyvyysarvot lasketaan yhteen (kuva 21).

4.2. PID-säätimen ohjelmarakenne

Ian Grout (2008) on suunnitellut PID-säätimen, joka on toteutettu digitaalisesti sulautetulla järjestelmällä FPGA:ssa. Laitteistopohjaiseen järjestelmään ohjelmoitua PID-säädintä voidaan optimoida sovelluksessa. Työ on toteutettu siten, että parametrit, ohje- ja mittausarvo annetaan PID-säätimelle digitaalisessa muodossa. Analoginen signaali täytyy muuntaa digitaaliseen muotoon, jos ei ole käytetty digitaalisia antureita.

4.2.1. PID-säätimen toteutus

Optimoidut parametrit tuodaan DE-optimoinnista PID-säätimelle digitaalisesti etumerkittömillä 16 bittisillä unsigned arvoilla. Proportional, integral ja derival kertoimien jälkeen signaalin arvo muutetaan etumerkilliseksi (signed), koska signaalin arvo voi vaihdella positiivisesta arvosta negatiiviseen arvoon. Laskennan bittitarkkuudeksi säätimellä on skaalattu 36 bittiä. Virtauskaavio PID-säätimen arkkitehtuurista on esitetty kuvassa 22.



Kuva 22. PID-säädinrakenne modulitasolla (Grout 2008: 517-522) laskee säätimen P-, I- ja D-termin. Laskenta tapahtuu ohjelmamoduleissa, joiden arvoja summaamalla, vähentämällä ja viivästämällä saadaan PID-säätimelle laskettua ulostuloarvo (output [17:0]) kellopulsseittain.

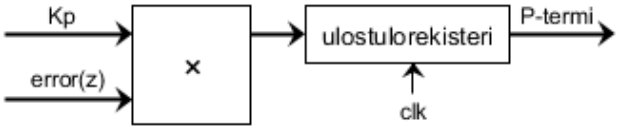
Kp-parametri tuodaan suhteelliselle vahvistimelle (proportional gain), jossa se kerrotaan erotusarvolla. Suhteellinen arvo siirretään rekisteriin odottamaan siirtoa summalaskimelle (AdderB). Laskennan ulostulorekisteri toimii kellopulssin tahdissa. Ti-parametri tuodaan integrointivahvistimelle (integral gain), jossa se kerrotaan erotusarvolla. Integraalin arvo siirretään summalaskimelle (Adder) ja viivemodulille (Delay), jossa integraalin arvoa viivytetään yhden kellojakson ajan. Summalaskimella

integraalin viivästetty ja viivästämätön arvo lasketaan yhteen. Summalaskimelta integraalin arvo siirretään summalaskimelle yksi (Adder1). Summalaskimella yksi yhdistetään summalaskimen ja summalaskimen yksi viivästetty arvo. Summalaskimelta yksi arvo viedään summalaskimelle (AdderB) kellopulsseittain.

Td-parametri tuodaan derivointivahvistimelle (derival gain), jossa se kerrotaan ohje- ja mittausarvolla ja skaalataan laskenta-alueelle. Derivointikertoimen ulostuloarvo syötetään viivemodulille kaksi (Delay2) ja erotusmodulille (Sub). Erotus tapahtuu Sub-modulilla, jossa viivästetty ja viivästämätön ulostuloarvo erotetaan. Sub-modulin ulostuloarvosta erotetaan Sub2-modulilla sen viivästetty arvo. Sub2-modulin erotusarvo syötetään Summalaskimelle (AdderB) kellopulsseittain. Summalaskimella (AdderB) säätimen P-, I- ja D-termi lasketaan yhteen ja skaalataan ulostuloarvo oikealle alueelle. Summalaskin (AdderB) toimii kellopulssin tahdissa.

4.2.2. Diskreettiaikainen laskenta-algoritmi

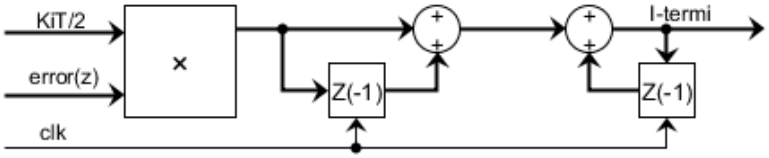
Diskreetti laskentatapa PID-säätimen vahvistustermin laskemiseksi digitaalisessa muodossa on esitetty kuvassa 23. Digitaalisen vahvistuksen laskennassa on käytetty hyödyksi kertolaskua, signaaliviivettä Z(-1) ja kellopulssia (clk). Vahvistusparametri Kp kerrotaan ensin ohje- ja mittausarvon erotusarvolla error(z). Kertolaskimen tulos ajetaan tämän jälkeen ulostulorekisteriin. Diskreetin vahvistustermin tulos (P-termi) syötetään PID-säätimen ohjaustermien summalaskimelle (AdderB) kellopulsseittain.



Kuva 23. Diskreetti vahvistustermi (Grout 2008: 517) saadaan kertolaskua ja ulostulorekisteriä käyttäen laskettua kellopusseittain. P-, I- ja D-termi yhdistetään laskennan päätteeksi.

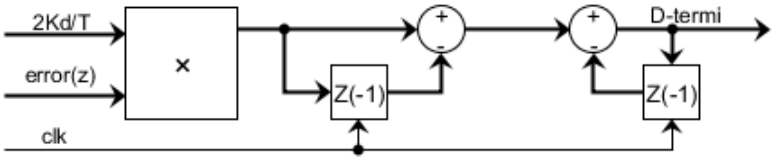
Diskreetti laskentatapa PID-säätimen integrointiajan laskemiseksi digitaalisessa muodossa on esitetty kuvassa 24. Digitaalisen integrointiajan laskennassa on käytetty kertolaskua, signaaliviivettä Z(-1), yhteenlaskua, bittien siirtoa ja kellopulssia (clk). Diskreettiä integrointiaikaa laskiessa ensin kerrotaan integrointiparametri Ki näytteenottoajalla T ja jaetaan kertolaskun tulos kahdella. Jakolaskussa bittiä siirretään yksi askel oikealle. KiT/2-arvo kerrotaan tämän jälkeen ohje- ja mittausarvon erotusarvolla error(z). Kertolaskimen tulos syötetään ensimmäiselle summalaskimelle, jossa siihen summataan viivästetty kertolaskimen tulos Z(-1). Ensimmäisen

summalaskimen tulos syötetään seuraavalle summalaskimelle, jossa ensimmäisen summalaskimen tulokseen lisätään toisen summalaskimen viivästetty tulos Z(-1). Diskreetin integrointiajan tulos (I-termi) syötetään PID-säätimen ohjaustermien summalaskimelle kellopulsseittain.



Kuva 24. Diskreetti integrointiaika (Grout 2008: 520) saadaan summaamalla ja viivästämällä kertolaskun tulosta kellopulsseittain. P-, I- ja D-termi yhdistetään laskennan päätteeksi.

Diskreetti laskentatapa PID-säätimen derivointiajan laskemiseksi digitaalisessa muodossa on esitetty kuvassa 25. Digitaalisen derivointiajan laskennassa on käytetty kertolaskua, signaaliviivettä Z(-1), vähennyslaskua, bittien siirtoa ja kellopulssia (clk). Diskreettiä derivointiaikaa laskiessa ensin jaetaan derivointiparametri Kd näytteenottoajalla T ja kerrotaan jakolaskun tulos kahdella. Kertolaskussa bittiä siirretään yksi askel vasemmalle. Edellinen 2Kd/T-arvo kerrotaan tämän jälkeen ohje- ja mittausarvon erotusarvolla error(z). Kertolaskimen tulos syötetään ensimmäiselle erotuslaskimelle, jossa siitä erotetaan viivästetty kertolaskimen tulos Z(-1). Ensimmäisen erotuslaskimen tulos syötetään tämän jälkeen seuraavalle erotuslaskimelle, jossa ensimmäisen erotuslaskimen tuloksesta vähennetään toisen erotuslaskimen viivästetty tulos Z(-1). Diskreetin derivointiajan tulos (D-termi) syötetään PID-säätimen ohjaustermien summalaskimelle kellopulsseittain.



Kuva 25. Diskreetti derivointiaika (Grout 2008: 522) saadaan vähentämällä ja viivästämällä kertolaskun tulosta kellopulsseittain. P-, I- ja D-termi yhdistetään laskennan päätteeksi.

Diskreetin integrointi- ja derivointiajan laskennassa on käytetty Tustinin approksimaatiota. Lausekkeista (35) ja (36) nähdään integrointi- ja derivointiajan laskentakaavat.

$$Y(z) = [(KiT/2)(X(z) + X(z)z^{-1})] + Y(z)z^{-1} \tag{35}$$

$$Y(z) = [(2Kd/T)(X(z) - X(z)z^{-1})] - Y(z)z^{-1} \tag{36}$$

5. SÄÄTIMEN TESTAUS

Testaus on tärkeä osa ohjelman toteutusta. On myös tärkeä jakaa ohjelma sopivan kokoisiin moduleihin, että testaaminen onnistuisi mahdollisimman helposti testausohjelmalla. Lopuksi, kun ohjelmamodulit on testattu yhdistetään ne toisiinsa ja testataan kokonaisuutena. Seuraavien lukujen testaustaulukot on saatu keräämällä otoksina ModelSim:tä optimaalisimmat sukupolvet taulukoihin.

5.1. Satunnaisesti luodut yksilöiden parametrit

sukupolvi 19-21	yksilö1			yksilö2			yksilö3			yksilö4		
ajokerrat	p1	p2	p3	p1	p2	p3	p1	p2	p3	p1	p2	p3
1	210	210	1395	1725	2805	105	105	6600	5625	2820	5520	5625
2	5775	6720	6105	105	4530	3360	5775	2805	1395	1305	3570	4530
3	5775	6720	6105	105	4530	3360	5775	2805	1395	1305	3570	4530
4	1935	2400	7035	4800	7515	4800	1935	5040	3750	3225	5100	6390
5	2520	5100	3750	5040	5715	2400	2400	2550	5265	2715	5115	7515
6	2520	5100	3750	5040	5715	2400	2400	2550	5265	2715	5115	7515
sukupolvi 20-21	yksilö1			yksilö2			yksilö3			yksilö4		
ajokerrat	p1	p2	p3	p1	p2	p3	p1	p2	p3	p1	p2	p3
7	2520	2550	5715	5100	2850	2400	5040	6390	6465	2715	5115	3750
8	2520	2550	5715	5100	2850	2400	5040	6390	6465	2715	5115	3750
9	2550	6390	5265	5115	6465	2520	5100	7350	5445	675	7035	2850
10	2550	6390	5265	5115	6465	2520	5100	7350	5445	675	7035	2850
11	6390	7350	3225	7035	5445	2550	5115	3750	1350	165	7515	6465
12	6390	7350	3225	7035	5445	2550	5115	3750	1350	165	7515	6465
sukupolvi 21	yksilö1			yksilö2			yksilö3			yksilö4		
ajokerrat	p1	p2	p3	p1	p2	p3	p1	p2	p3	p1	p2	p3
13	3750	2850	675	5715	330	7350	7515	6465	3915	5280	5265	1350
14	5715	5265	330	2850	165	3750	7515	5445	3915	7080	6465	675
15	5265	3225	3915	6465	3915	2850	3750	1350	5790	3540	5445	165
16	5265	3225	3915	6465	3915	2850	3750	1350	5790	3540	5445	165
17	2715	675	5280	1350	5280	5445	5265	165	6480	5490	675	5280
18	2715	675	5280	1350	5280	5445	5265	165	6480	5490	675	5280
sukupolvi 21-22	yksilö1			yksilö2			yksilö3			yksilö4		
ajokerrat	p1	p2	p3	p1	p2	p3	p1	p2	p3	p1	p2	p3
19	165	2895	3540	3915	5610	675	1350	6480	5610	5205	2895	5610
20	3915	6480	5610	5790	3315	330	1350	6480	5610	5205	6480	5610
21	2895	5610	5490	6480	5475	2895	3915	5610	6585	1605	5610	5490
22	5280	6645	6585	7080	5205	2895	5790	3315	3285	795	6645	6585
23	7080	5490	5475	3540	3210	6480	2895	5490	5475	390	5490	3285
24	5610	6585	2730	6645	795	3540	6480	3285	2730	3930	6585	2730

Taulukko 7. Yksilöiden satunnaisparametrit p1, p2 ja p3 ajokertojen mukaan

Ohjelmalla luotiin satunnaisesti neljä yksilöä, jossa jokaiselle yksilölle on annettu kolme parametria per ajokerta. Otoksen parametrit on kerätty sukupolvien 19 ja 22 väliltä ModelSimin testausohjelmasta. Ajokertoja on kaikkiaan 24 kappaletta, jossa jokainen ajokerta sisältää kaksitoista satunnaisparametria. Satunnaisparametreista osa on valittu mutaatio- ja hyvyysajojen perusteella, koska silloin on saatu satunnaisarvot ja parhaat hyvyysarvot täsmäämään keskenään testausprosessissa. Loput parametreista on otettu järjestyksessä simulointitaulukosta. Ohjelmassa pyrittiin saamaan yksilöiden satunnaisparametrien välille vaihtelua, jossa onnistuttiin melko hyvin (taulukko 7). Optimoinnin käynnistystilanteessa ennen satunnaissekoittimen (sekoitusmoduli) käynnistymistä satunnaisarvoihin saatiin vaihtelua käynnistämällä satunnaisluku-generaattorit laskurien avulla eri aikaan. Todellinen satunnaisarvojen sekoitin alkoi toimia optimointiohjelman käynnistyttyä. Sekoitin toimi, kun optimointiprosessi pysähtyi jonkin vektori-indeksiarvoista (r1), (r2) tai (r3) ollessa nolla. Parametrit, jotka on luotu ajokertojen 1 ja 18 välillä on osa ajokerroista ajettu vertailuprosessissa kahteen kertaan populaation ranking-modulilla (taulukko 7 ja 8). Virheellisyys johtuu populaation suuruusjärjestyslausekkeen ohjelmakoodista, josta puuttuu Ranking-modulilta annettu mutaatiokello, joka synkronoi ranking-modulin ja mutaatiolaskennan samaan rytmiin. Tämän virheen vuoksi ohjelma valitsee yhden ajokerran kaksi kertaa, kun vektori-indeksiarvot ovat arvossa yksi useita kellopulsseja peräkkäin. Ajokertojen 19 ja 24 välillä luodut satunnaisparametrit ovat onnistuneet hyvin, koska vektori-indeksiarvo on ollut arvossa yksi yhden kellopulssin kerrallaan suuruusarvovertailun aikana. Testauksesta voidaan päätellä satunnaislukusekoittimen vaikuttavan satunnaislukujen toistuvuuteen vähentävästi ja lisäävän satunnaislukuarvojen vaihtelua eri ajokertojen välillä.

5.2. Yksilöiden järjestely

Satunnaisesti luodut yksilöt ajetaan ranking-modulille, jossa ensimmäisenä lasketaan yksilöiden hyvyysarvo summaamalla yksilöiden parametrit yksilöittäin. Tämän jälkeen yksilöille suoritetaan suuruusvertailu, jossa selvitetään suurin ja toiseksi suurin yksilö ajokerroittain (taulukko 8). Vertailussa syntyy virhettä juuri niissä ajokerroissa, jotka on vertailtu populaation suuruusjärjestyslausekkeessa kahteen kertaan (taulukko 7 ja 8). Virheellisissä ajokerroissa ohjelma on valinnut parametrin (p1_rank) ja (p2_rank) yksilöiden yksi ja neljä väliltä satunnaisesti ilman suuruusvertailua. Yksilöiden suuruusjärjestyksellä ei virheellisessä tapauksessa ole kuitenkaan suurta merkitystä

Ranking-vektorin parametrien yksi (p1_rank) ja kaksi (p2_rank) valintaan. Ranking-vektorin kolmannessa parametrissa (p3_rank) ei ole virhettä, koska satunnaisarvo luodaan jokaisella kellojaksolla. Satunnaisparametrin arvoa voidaan kuitenkin parantaa satunnaislukugeneraattorin arvoaluetta laajentamalla. Toimivilla ajokerroilla valinnat onnistuvat suunnitellun mukaisesti. Suuruusjärjestys ei kuitenkaan tarkoita parametrien arvojen olevan suuruusjärjestyksessä. Parametrit valitaan suurimmasta ja toiseksi suurimmasta yksilöstä niiden parametrien suuruuden mukaan.

sukupolvi 19-21	yksilö1 hyvyys	yksilö2 hyvyys	yksilö3 hyvyys	yksilö4 hyvyys	rank		rand	virhe
ajokerrat					p1	p2	p3	
1	1815	4635	12330	13965	2820	6600	4905	o
2	18600	7995	9975	9405	1305	6720	5400	v
3	18600	7995	9975	9405	5755	2805	6540	o
4	11370	17115	10725	14715	4800	5100	915	o
5	11370	13155	10215	15345	2715	5100	3945	v
6	11370	13155	10215	15345	2715	5715	5805	o
sukupolvi 20-21	yksilö1 hyvyys	yksilö2 hyvyys	yksilö3 hyvyys	yksilö4 hyvyys	rank		rand	virhe
ajokerrat					p1	p2	p3	
7	10785	10350	17895	11580	2715	2850	6735	v
8	10785	10350	17895	11580	5040	5115	7200	o
9	14205	14100	17895	10560	5100	7035	7440	v
10	14205	14100	17895	10560	5100	6390	7560	o
11	16965	15030	10215	14145	5115	7350	7620	v
12	16965	15030	10215	14145	6390	5445	7650	o
sukupolvi 21	yksilö1 hyvyys	yksilö2 hyvyys	yksilö3 hyvyys	yksilö4 hyvyys	rank		rand	virhe
ajokerrat					p1	p2	p3	
13	7275	13395	17895	11895	7515	330	945	o
14	11310	6765	16875	14220	7515	6465	4065	o
15	12405	13230	10890	9150	3750	5445	5865	v
16	12405	13230	10890	9150	6465	3225	6765	o
17	8670	12075	11910	11445	1350	675	5640	o
18	8670	12075	11910	11445	1350	165	6660	o
sukupolvi 21-22	yksilö1 hyvyys	yksilö2 hyvyys	yksilö3 hyvyys	yksilö4 hyvyys	rank		rand	virhe
ajokerrat					p1	p2	p3	
19	6600	10200	13440	13710	5205	6480	4050	o
20	16005	9435	13440	17295	5205	6480	4845	o
21	13995	14850	16110	12705	3915	5475	4620	o
22	18510	15180	12390	14025	5790	5205	6150	v
23	18045	13230	13860	9165	7080	3210	5370	v
24	14925	10980	12495	13245	5610	6585	5460	o

Taulukko 8. Ranking-parametrit p1, p2 ja p3 ajokerroittain yksilöiden suuruusvertailussa

Populaation suuruusjärjestely sekoittaa parametreja ajokertojen sisällä lisäämällä satunnaisuutta. Yhtäsuuria arvoja ei myöskään ole päässyt syntymään parametrien joukkoon. Ajokertojen ranking-parametreista näkee parametrien sekoittuneen erinomaisesti. Populaation suuruusjärjestely parantaa optimoinnin lopputulosta.

5.3. Mutaatio ja risteytys

sukupolvi 19-21	rank		rand	mutaatio-vakio	mutaatio-vektori			risteytys-vakio	yrite-vektori			hy-vyys
ajokerrat	p1	p2	p3	F	p1	p2	p3	CR	Kp	Ki	Kd	
1	2820	6600	4905	118								
2	1305	6720	5400	118	6635	1609	3210	83	6635	1609	3210	*462
3	5775	2805	6540	118								
4	4800	5100	915	91								
5	2715	5100	3945	77	2728	2888	5543	480	2728	1609	3210	570
6	2715	5715	5805	77								
sukupolvi 20-21	rank		rand	mutaatio-vakio	mutaatio-vektori			risteytys-vakio	yrite-vektori			hy-vyys
ajokerrat	p1	p2	p3	F	p1	p2	p3	CR	Kp	Ki	Kd	
7	2715	2850	6735	77								
8	5040	5115	7200	77	5160	5353	3299	508				
9	5100	7035	7440	77								
10	5100	6390	7560	77								
11	5115	7350	7620	77	6721	5155	5275	127				
12	6390	5445	7650	77								
sukupolvi 21	rank		rand	mutaatio-vakio	mutaatio-vektori			risteytys-vakio	yrite-vektori			hy-vyys
ajokerrat	p1	p2	p3	F	p1	p2	p3	CR	Kp	Ki	Kd	
13	7515	330	945	91								
14	7515	6465	4065	91	3824	7941	7624	481				
15	3750	5445	5865	91								
16	6465	3225	6765	91								
17	1350	675	5640	91	2504	2232	7094	135				
18	1350	165	6660	91								
sukupolvi 21-22	rank		rand	mutaatio-vakio	mutaatio-vektori			risteytys-vakio	yrite-vektori			hy-vyys
ajokerrat	p1	p2	p3	F	p1	p2	p3	CR	Kp	Ki	Kd	
19	5205	6480	4050	90								
20	5205	6480	4845	90	4065	5492	5632	205				
21	3915	5475	4620	90								
22	5790	5205	6150	90								
23	7080	3210	5370	90	5807	7459	5956	475	5807	7459	5956	1401
24	5610	6585	5460	90								

Taulukko 9. Mutaatio- ja risteytystulokset ajokerroittain Ranking-parametrien p1, p2, ja p3 arvoilla ja optimoinnin eri ajokertojen mutaatio- ja risteytysparametrien laskentatulokset.

Mutaatiovektori on luotu ranking-parametreista itse-adaptoituvan mutaatioparametrin F avulla. Mutaatioparametri on ajotilanteessa vaihdellut [0,13;0,9] välillä eli arvot ovat skaalattuna [64,461] välillä. Itse-adaptoituvan mutaatioparametrin F arvot vaihtelevat otoksessa [0,15;0,23] välillä ja skaalattuna arvot ovat [77,118] välillä. Ranking-modulilta tulleiden differenssivektorin ensimmäinen (p2_rank) ja differenssivektorin päätepisteen (p3_rand) erotusarvo (p2_rank - p3_rand) kerrotaan mutaatioparametrilla F, jonka jälkeen arvo lisätään perusvektoriin (p1_rank) ajokerroittain. Mutaatiovektorissa on kolme ajokertaa, jossa yhden ajokerran ranking-parametrit muodostavat yhden mutaatiovektorin parametrin (taulukko 9). Laskennassa ensimmäisen ajokerran ranking-parametrit muodostavat mutaatiovektorin kolmannen parametrin, toinen ajokerta muodostaa toisen parametrin ja kolmas ajokerta muodostaa ensimmäisen parametrin. Itse-adaptoituvan mutaatioparametrin matala arvo otoksessa vaikuttaa mutaatiovektorin parametrien suuruuteen. Mutaatiovektorin kolmannen parametrin alhaisella arvolla on vaikutusta risteytyessään hyvyysarvon suuruuteen, jonka perusteella PID-säätimen optimiparametrien valinta tapahtuu (taulukko 9). Sukupolvi, jonka Hyvyysarvon suuruus on optimointiajon matalin valitaan PID-säätimen uusiksi ohjausparametreiksi. Mutaatiovektorien parametrien arvo vaihtelee itse-adaptoituvan mutaatioparametrin F ja ranking-populaation parametrien mukaan. Mutaativektorin arvojen vaihtelu otoksessa on onnistunut ja niistä voidaan päätellä parhaiden mutaatioparametrien onnistuvan pienillä mutaatioparametrin F arvoilla.

Yritevektori luodaan mutaatiovektorista risteyttämällä se itse-adaptoituvan risteytysparametrin CR avulla (taulukko 9). Risteytys tapahtuu parametreittain vertailemalla risteytysparametrin CR arvoa satunnaislukuun RNG. Mutaatiovektorin parametrit risteytyy, kun RNG $\leq$ CR. Itse-adaptoituvan risteytysparametrin CR arvo on otoksen risteytyksessä vaihdellut [0,16;0,93] välillä eli skaalattuna arvot ovat [83,480] välillä. Paras hyvyysarvo 462 on saatu pienimmällä risteytysparametrin 83 arvolla, jossa yritevektoriin on valittu kaikki mutaatiovektorin parametrit (p1, p2 ja p3). Sukupolvista pienimmän hyvyysarvon omaavaa yritevektoria kutsutaan PID-säätimen optimiparametreiksi. Yriteparametreilla, joilla on ajettu toiseksi paras hyvyysarvo 570 ja risteytysvakio on 480. Sen yritevektoriin on valittu ensimmäinen mutaatiovektorin parametri (p1). Otoksen viimeiset yritteet (yritevektori) on ajettu sukupolven 22 kuluessa ja sen hyvyysarvo on 1401 ja risteytysvakio on 475. Sen yritevektoriin on valittu kaikki mutaatiovektorin parametrit. Hyvyysarvojen suuruuteen vaikuttaa yritevektorin kolmas parametri eli derivointitermi Kd ja PID-säätimen erotusarvo. Mutaatio- ja risteytystulokset ovat onnistuneet suunnitelman mukaisesti.

5.4. Mutaatio- ja risteytysvakio

sukupolvi 20	mutaatio-vakio	rand1	rand2	väli-arvo	min. / maks. arvo	τ_1	risteytys-vakio	rand3	rand4	τ_2
ajokerrat	F=F_ulos			Fv	Fl / Fμ		CR			
1	298	260	9	298			37	130	18	
2		130	260	181			18	65	9	
3		65	130	122			9	288	260	
4		288	65	323			260	400	130	
5		400	288	424	64 / 461	64	130	200	65	64
6		200	400	244			65	100	288	
7	315	279	46	315			185	395	92	
8		395	279	419			92	453	46	
9		453	395	471			46	482	279	
sukup 20-21	mutaatio-vakio	rand1	rand2	väli-arvo	min / max arvo		risteytys-vakio	rand3	rand4	
ajokerrat	F=F_ulos			Fv	Fl / Fμ		CR			
10	300	263	15	300			61	387	30	
11		387	263	412			30	449	15	
12		449	387	468			15	480	263	
13		480	449	496			263	496	387	
14	92	32	32	92	64 / 461	64	64	32	32	64
15		272	272	308			64	272	272	
16		136	136	186			272	136	136	
17		68	68	125			136	68	68	
18	94	34	34	94			68	34	34	

Taulukko 10. Mutaatio- ja risteytysvakioiden laskentavaiheet erillisessä testausotoksessa, jossa mutaatiovakion ja risteytysvakion tulokset on laskettu satunnaisarvojen (rand1, rand2, rand3 ja rand4), mutaatioalueen minimin (Fl) ja maksimin (Fμ), vertailuarvojen τ_1 ja τ_2 avulla.

Testistä on otettu mutaatioparametrista F ja risteytysparametrista CR kahdeksantoista ajon otos (taulukko 10). Mutaatio- ja risteytysparametrin laskenta-arvot ovat onnistuneet testiajojen yksi ja kolmetoista välillä, mutta testiajossa on virheitä ajojen neljätoista ja kahdeksantoista välillä. Virheet aiheutuvat ohjelman sekoitinlaskurien nollaushäiriöstä. Mutaatiovektorimodulin mutaatio- ja risteytysparametrin ohjelmaan oli rakennettu satunnaislukugeneraattoreille sekoitinlaskurit, jotka käynnistävät satunnaislukugeneraattorit eri aikaan. Virhe testiajossa aiheutui laskurien nollauksen puuttumisesta. Ensimmäisessä optimointiajossa mutaatio- ja risteytysvakion arvot onnistuivat, mutta nollauksen jälkeen laskurit eivät nollautuneet. Tämän vuoksi satunnaislukugeneraattorit toimivat samassa kellotahdissa ja mutaatiparametri F antoi

vain alhaisia arvoja ajojen 14 ja 18 välillä (taulukko 10). Virheet pystytään korjaamaan erillisellä satunnaislukugeneraattorien sekoittimella, jolla satunnaslukujen (rand1, rand2, rand3 ja rand4) välille saadaan enempi vaihtelua. Tämänhetkinen laskurisekoitin ei riitä tuottamaan riittävää vaihtelua satunnaislukujen välille. Lisäksi risteytysvakion CR ohjelmasta löytyi ylimääräinen kellopulssi, joka aiheutti satunnaislukuarvoon viivettä (taulukko 10). Puutteiden korjauksella mutaatio- ja risteytysparametrin arvo saadaan korjatuksi.

5.5. Hyvyysarvo

sukupolvi 20	ohje-arvo	näytteenotto-väli	derivointi-termi	erotus-arvo	hyvyys-arvo	kokonais-hyvyys
	ref	h	Kd	error		
20	2000	2ms	3210	2039 1987 1961 2012 1974 2019 1977 2020 1978 2021 2042 2053 2058 2061 2062 2063 1999 1967 1951 1943	23 23 22 23 23 23 23 23 23 23 23 24 24 24 24 24 23 23 22 22	*462
21	2000	2ms	3210	1945-2056	k.a. ~23	570
22	2000	2ms	5956	1941-2059	k.a. ~42	1401

Taulukko 11. Hyvyyslaskennan optimitulokset PID-säätimen optimointiajossa

Hyvyysarvotestitaulukossa PID-säätimen hyvyysarvot lasketaan derivointitermin Kd, näytteenottovälin ja erotusarvon mukaan. Hyvyysarvolaskennan aikana näytteenottoväli ja derivointitermi pysyvät vakiona, mutta erotusarvo muuttuu kellopulssin välein. Hyvyysarvo lasketaan kaikille erotusarvoille sukupolvittain, jossa kaikki lasketut hyvyysarvot lasketaan lopuksi yhteen (taulukko 11). Yläpuolella olevassa taulukossa näkyy sukupolven 20 aikana laskettu paras hyvyysarvo. Optimaalisimmat PID-säätimen

parametrit on valittu hyvyysarvon suuruuden perusteella. Sukupolven 20 aikana tuotetut risteytysparametrit on valittu uusiksi PID-säätimen parametreiksi (taulukko 9) ja (taulukko 12). Modulin hyvyyslaskimen testauksessa mittausarvoa MV simuloimalla (kuva 26) luotiin erotusarvo kellopulsseittain hyvyysarvolaskimelle. Hyvyyslaskenta toimii hyvyysarvolaskurien (temp3) ja (temp4) mukaan (kuva 21) niiden arvon ollessa välillä [1,5]. Sukupolven 20 aikana hyvyysarvosilmukka laski kaksikymmentä hyvyysarvoa (taulukko 11), jolloin sukupolven 20 kokonaishyvyys oli 462. Hyvyysarvolaskennan tulosta vääristi kuitenkin hyvyysarvosilmukoiden eri sukupolvien välinen laskettu hyvyysarvojen määrä, joka vaihteli eri sukupolvien välillä. Hyvyysarvon oikea tulos saadaan korjattua jakamalla kokonaishyvyys laskentakertojen määrällä. Ohjelma korjataan lisäämällä laskuri hyvyysarvosilmukkaan, joka laskee laskettujen hyvyysarvojen määrän sukupolvittaisen hyvyysarvolaskennan aikana. Laskurin arvolla jaetaan kokonaishyvyysarvo sukupolvittaisen laskennan loputtua.

5.6. PID-säätimen testaus optimiparametreilla

sukupolvi 20	ohje-arvo	optimi-parametrit			erotus-arvo	P termi	I termi	D termi	yhteis-arvo	ohjaus-arvo	kokonais-hyvyys
	ref	Kp	Ki	Kd	error						
20	2000	6635	1609	3210	39	181	-470	-498	-787	1213	462
					-13	505	-470	444	479	2479	
					-39	168	-471	-471	-774	1226	
					12	-505	-472	525	-453	1547	
					-26	156	-472	-564	-881	1119	
					19	-337	-472	611	-198	1802	
					-23	246	-472	-655	-881	1119	
					20	-298	-472	700	-70	1930	
					-22	259	-472	-744	-957	1043	
					21	-285	-472	789	32	2032	
					42	272	-471	-767	-966	1034	
					53	544	-469	778	853	2853	
					58	687	-467	-773	-554	1446	
					61	752	-465	776	1063	3063	
					62	790	-463	-775	-448	1552	
					63	803	-461	776	1119	3119	
					-1	816	-460	-843	-487	1513	
					-33	-13	-461	810	336	2336	
					-49	-428	-462	-826	-1716	284	
					-57	-635	-464	818	-281	1719	

Taulukko 12. PID-säätimen laskentatulokset eri erotusarvoilla ja optimiparametreilla

PID-säädintestissä on ajettu säädintä optimiparametreilla, joiden kokonaishyvyydeksi on saatu 462 (taulukko 12). Testi on ajettu sukupolven 20 optimoinnin aikana. Yllä olevasta taulukosta nähdään säätimen optimiparametrit (Kp), (Ki) ja (Kd), ohjearvo (ref), erotusarvot (error) sekä säätimen P-, I- ja D-termien laskentatulokset ajetuilla erotusarvoilla. Säätimen ohjausarvo on saatu, kun PID-säätimen P-, I- ja D-termien yhteisarvo on vähennetty ohjearvosta. PID-säädin antaa taulukon mukaan osittain vääriä ohjausarvoja, koska derivointiosan (D-termi) ohjelmasta puuttui kerroin kaksi joka aiheuttaa pientä virhettä.

5.7. FPGA-piirin resurssit

PID-säätimen optimoinnin toteutus on toteutettu kokonaislukuoperaatioilla yhdelle FPGA-piirille ja sen toiminta ja nopeus on testattu ModelSimilla. Yhden mikropiirin toteutuksessa etuna on nopeuden kasvaminen ja häiriöiden väheneminen. Lisäksi säätöjärjestelmän rakenteen muuttaminen ja uusien ominaisuuksien lisääminen ohjelmaan on helppoa. Suorituskyky säädintoteutuksessa on mitattu kellojaksojen määrää mittaamalla eri kokoisista sukupolvista (taulukko 13). Suorituskykyä on testattu 25, 50 ja 100 sukupolven suuruisissa optimoinneissa. Suoritusaika näkyy suoraan mikrosekunteina (µs) tai sekunteina (s) simulointiohjelman tuloksessa. Suoritusaika riippuu myös näytteenottoajasta (taulukko 13). PID säätimen testauksessa käytettiin 0,02 µs näytteenottoaikaa, jolloin suoritusaika eri sukupolvimäärillä oli välillä [27,96;120,20] µs. Näytteenottoajan ollessa 2ms suoritusaika eri sukupolvimäärillä olisi välillä [2,796;12,02] s. Kellojaksojen määrä kaikkiaan eri sukupolvimäärillä oli molemmilla näytteenottoajoilla väliltä [1398,6010].

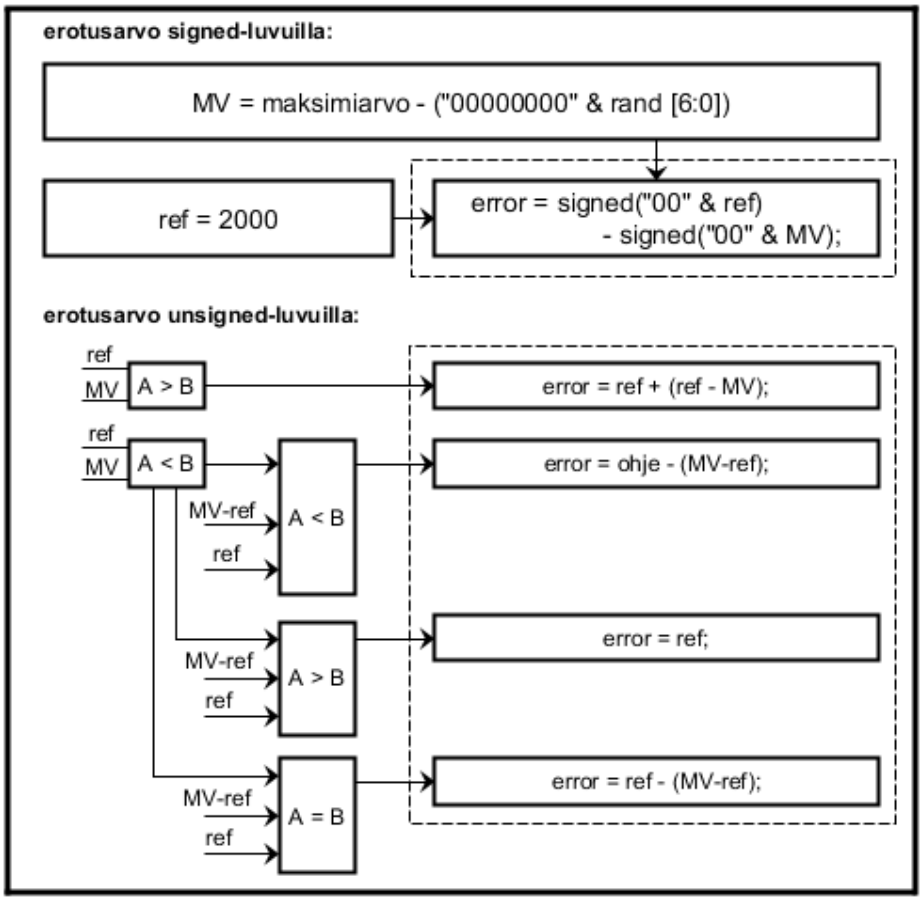
sukupolvien määrä	**kellojaksojen määrä**			**näytteenottoaika**		**suoritusaika**
25	1398					27,96 µs
50	3281			0,02 µs		65,62 µs
100	6010					120,20 µs
25	1398					2,796 s
50	3281			2 ms		6,562 s
100	6010					12,02 s
logiikkaelementit	**LUT**	**rekisterit**	**I/O-pinnit**	**kertolaskimet**		**sisäiset kytkennät**
5383	4713	1622	79	16 (9bit)	8 (18bit)	20049

Taulukko 13. Optimoitavan PID-säätimen FPGA-piirin kokonaissuorituskyky ja laskentaresurssit.

FPGA-ohjelman resurssit on laskettu logiikkaelementtien, logiikkaelementtien LUT-tulojen käytön, rekisterien, I/O-tulojen ja -lähtöjen ja kertolaskimien määrän mukaan. Koko järjestelmän resurssien kulutukseksi saadaan 11821 logiikkasolua (taulukko 13). Lisäksi taulukosta nähdään kaikki ohjelman sisäiset kytkennät, joita on 20049 kpl.

Alteran Cyclone II EP2C35 PowerPlay -tehoanalysaattori antoi optimoitavan PID-säätimen keskeiseksi pysyväksi tehonkulutukseksi 79,95 mW, tulojen ja lähtöjen tehonkulutukseksi 39,18 mW ja kokonaistehonkulutukseksi 119,13 mW 50 MHz kellotaajuudella. PID-säätimen tehonkulutus ylittää vähän Cyclone III EP3C16 tuoteperheen tehonkulutuksen, jossa kymmenentuhannen logiikkayksikön keskeiseksi pysyväksi tehonkulutukseksi oli annettu 42 mW ja kokonaistehonkulutukseksi 161 mW 100 MHz kellotaajuudella (Actel Corporation 2008: 3).

5.8. Erotusarvon tuottaminen säätimen simulointitestaukseen

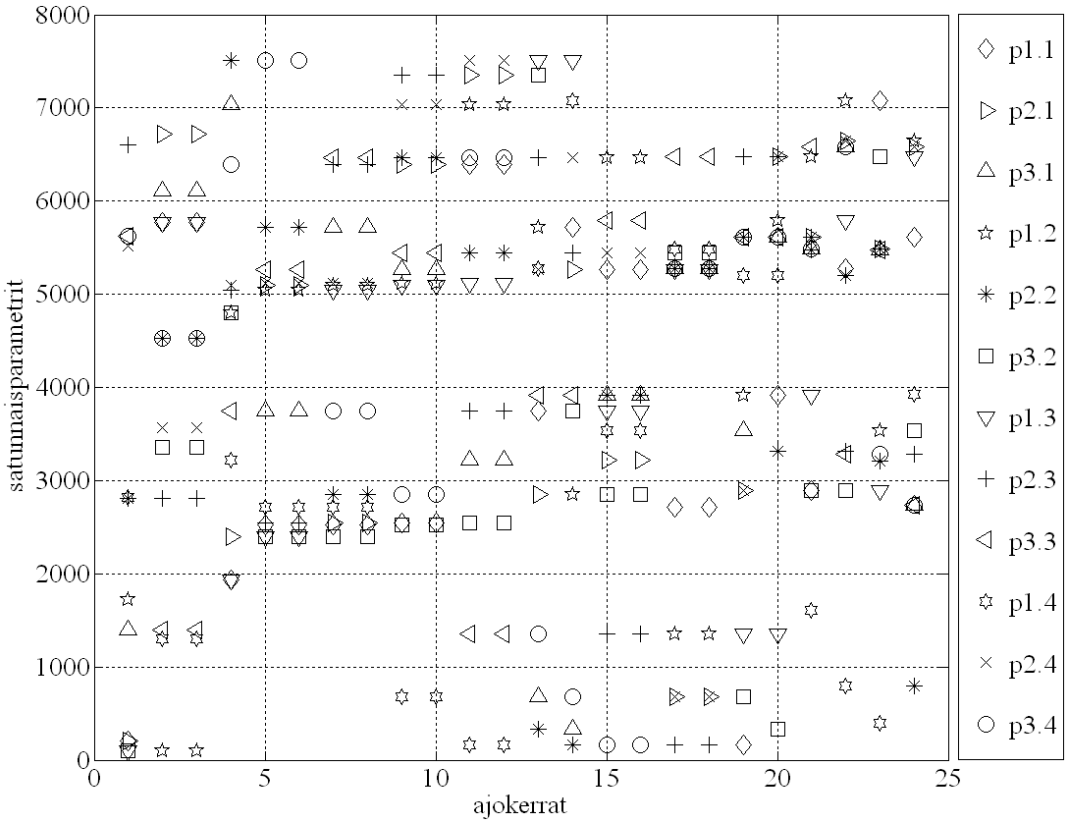


Kuva 26. Etumerkillisen ja -merkittömän erotusarvon luominen PID-säätimen ja hyvyys-laskennan ohjelmamoduleille.

Mittausarvo on luotu ohjelmallisesti käyttämällä apuna satunnaisarvoa (rand). Mittausarvo luodaan vähentämällä maksimiarvosta 2064 sopivalle kokonaislukualueelle skaalattu satunnaisarvo (kuva 26). Erotusarvo (error) on laskettu vähentämällä mittausarvo ohjearvosta. Erotusarvon suuruus on skaalattu välille [1936,2064]. Hyvyysarvomodulille lähetetään etumerkitön erotusarvo (erotusarvo unsigned-luvuilla**)** (kuva 26), jonka pitää hyvyyslaskennassa olla positiivinen arvo (taulukko 11). Erotusarvo lasketaan tässä tapauksessa vertailemalla ohje- ja mittausarvoa (ref ja MV). Ohje- ja mittausarvo lähetetään erikseen PID-säätimen Subtractor-modulille, jossa niiden arvot muunnetaan etumerkillisiksi (erotusarvo signed-luvuilla). Subtractor-modulilla PID-säätimen erotusarvo lasketaan vähentämällä ohjearvo mittausarvosta ilman vertailua. PID-säädin algoritmin laskenta on toteutettu etumerkillisillä luvuilla säädintestissä (taulukko 12). Mittausarvoa simuloimalla oli mahdollista toteuttaa PID-säätimen kokonaistestaus (kuva 26).

6. TULOSTEN ANALYSOINTI

6.1. Differentiaalievoluutiolaskenta

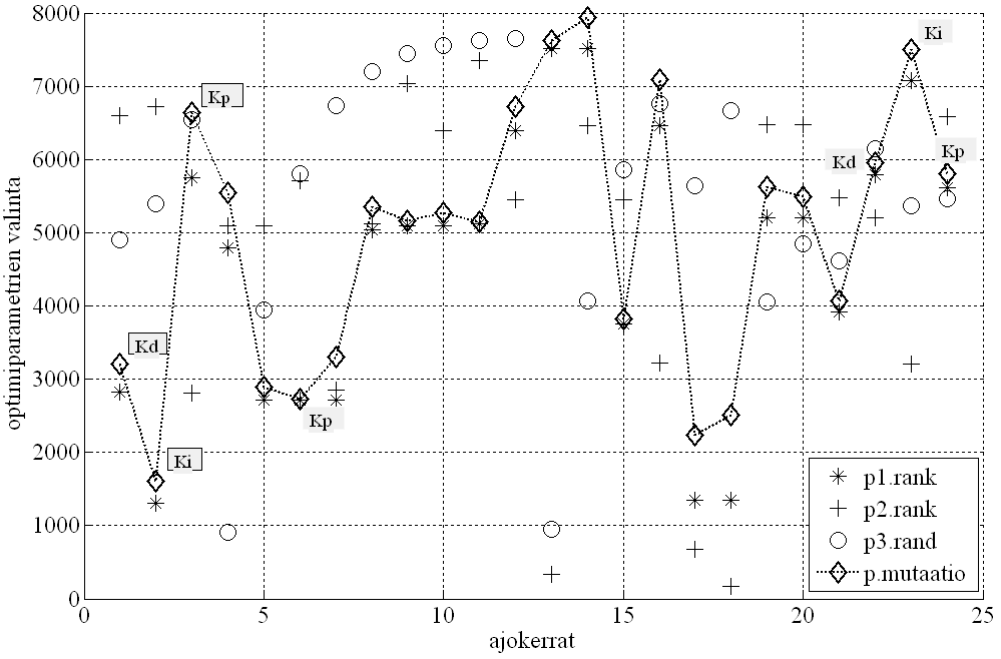


Kuva 27. Satunnaisparametrien sirontakuvaajassa on esitettynä kaikkien otoksen yksilöiden parametrien (p1.1,…, p3.1; p1.2,…,p3.2; p1.3,…,p3.3; p1.4,…,p3.4) hajonta.

Ensimmäisenä analysoitiin Matlab:lla kaikkien satunnaisparametrien (p1.2,…,p3.4) hajontaa (kuva 27). Sirontakuvaajassa olevat satunnaisparametrit ovat valittuja satunnaisarvoja, joista valitaan parametrit yksilön suuruuden mukaan tulevaan ranking-vertailuun ja sen jälkeen mutaatiolaskentaan. Satunnaisparametrit ovat jakautuneet tasaisesti välille [0,8000] lähes jokaisella ajokerralla. Satunnaisparametrejä on 12 kappaletta ajokerrassa ja kolme kappaletta per yksilö. Ohjelman DE-optimoinnissa on käytetty neljää yksilöä. Ajokerrat 5 ja 6, 7 ja 8, 9 ja 10, 11 ja 12, 15 ja 16 sekä 17 ja 18 sisältävät samat satunnaisparametrit. Virheellisyys johtuu ranking-modulin ja mutaatiolaskennan vääränlaisesta ohjelmasta. Ajokertojen satunnaisarvot välillä [19,24] ovat sekoittuneet paremmin. Kuvaajasta voi huomata satunnaisparametrien olevan suppealla alueella, koska samat arvot toistuvat eri ajokerroilla (kuva 27).

Satunnaislukuarvojen aluetta voidaan parantaa ajokerroittain suurentamalla satunnaislukugenaraattorin parametrialueen laajuutta satunnaislukugeneraattoreilla.

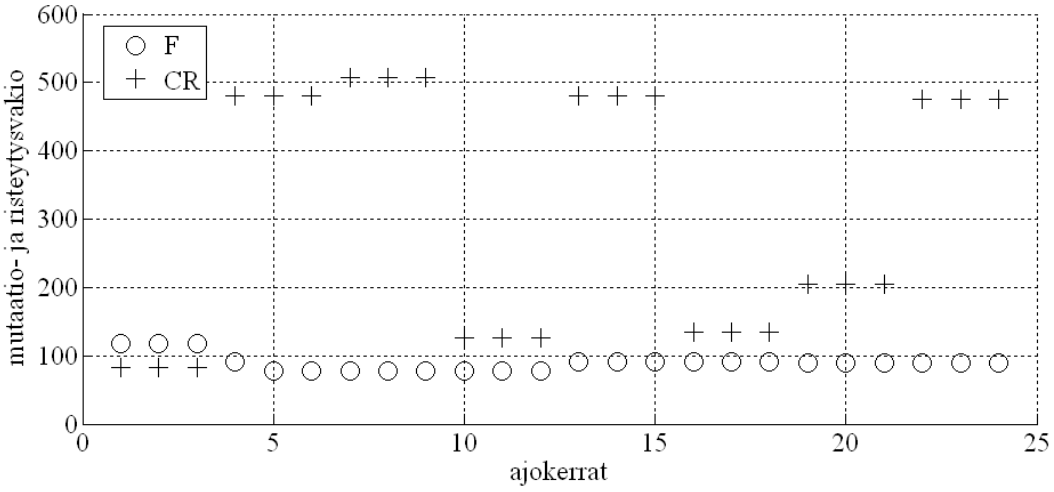
Jokainen ajokerta sisältää Ranking-modulilla yksilöiden suuruusjärjestyksen mukaan valitut parametrit (kuva 28). Ranking-parametrit eivät ole aina suuruusjärjestyksessä (p1.rank ja p2.rank), koska suuruusjärjestyksen vertailu on tehty yksilöiden hyvyyden mukaan. Kolmas parametri (p3.rand) on valittu satunnaisesti. Suuruusjärjestysvertailu on vaikuttanut niin, että pienet satunnaisarvot ovat karsiutuneet pois. Ranking-arvot ovat jakautuneet tasaisesti koko alueelle ajokertojen 1 ja 7 välillä, josta optimaalisimmat mutaatioparametrit on valittu. Ajokerroilla 8:sta 14:sta sekä 19:sta 24:ään arvot ovat jakautuneet arvojoukon yläpäähän, jolloin mutaatiovektorin arvot ovat kasvaneet liian suuriksi. Tämän vuoksi myös optimoinnin hyvyysarvo on kasvanut liian suureksi Kd-parametrin arvon kasvaessa liikaa ajokerroilla 22, 23 ja 24. D-termillä (Kd) on suurin vaikutus hyvyysarvon suuruuteen ja tämän vuoksi se vaikuttaa paljon optimiparametrien valintaan. Kuvasta voi päätellä, että parametrien satunnaisuus on optimoinnin onnistumisen kannalta tärkeää. Esimerkiksi, kun ranking-vektorin (differenssivektorin) ensimmäinen päätepiste (p2.rank) ja toinen päätepiste (p3.rank) ovat liian lähellä toisiaan ajokertojen 8 ja 16 välillä ajokerrasta toiseen. Tämä vaikuttaa suoraan yriteparametrien suuruuteen.



Kuva 28. Optimiparametrien valinta lasketuista mutaatiovektorin parametreista (p.mutaatio) risteytyksen ja hyvyyslaskennan perusteella. Mutaatiovektorin parametrit on laskettu ranking-parametreista (p1.rank, p2.rank ja p3.rand) ajokerroittain.

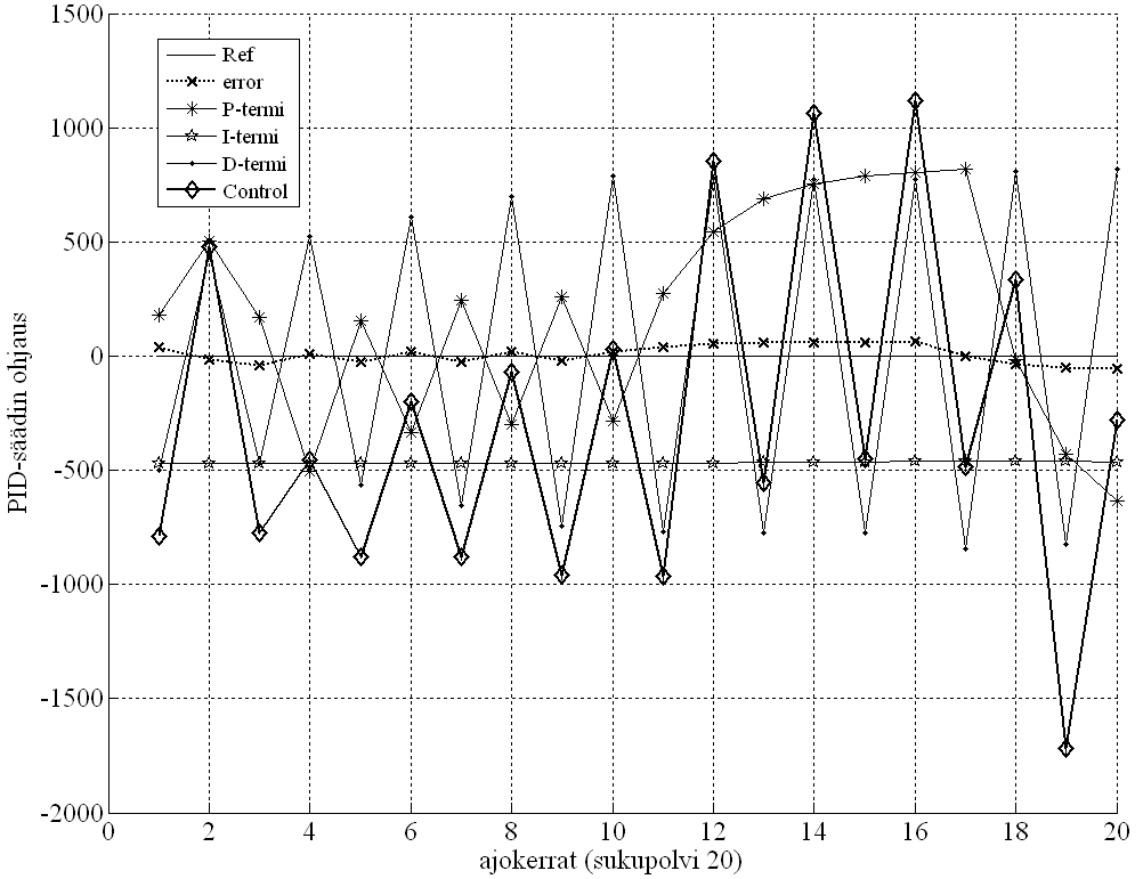
Mutaatiovektorin (p.mutaatio) parametrit on valittu aina kolmesta perättäisestä ajokerrasta. Mutaatiovektorin laskennassa on käytetty apuna mutaatiovakiota (kuva 29). Optimointiajossa mutaatiovakion F arvot olivat pieniä. Arvojen suuruuden syynä oli ohjelmavirhe, jossa mutaatiovakion F satunnaisarvot eivät sekoittuneet vaan sen kaikki satunnaislukugeneraattorit antoivat samaa arvoa. Tästä syystä mutaatiovakion arvot painuivat alas (kuva 29). Pienellä mutaatiovakion arvolla oli myös positiivista vaikutusta mutaatioparametrien suuruuteen (kuva 28). Liian suuri mutaatiovakion arvo kasvattaa mutaatioparametrien arvoa liian suureksi. Parhaimman mutatointituloksen antaa kuitenkin mutaatiovakion arvot, jotka vaihtelevat tasaisesti sen optimointialueella. Mutaatiovakion satunnaisgeneraattoreiden (rand1 ja rand2) toimintaa parannetaan ohjelman satunnaislukusekoittimella.

Risteytyksessä risteytysvakion CR arvot ovat olleet joko ylä- tai alapäässä risteytysvakion optimointialueella (kuva 29). Risteytysvakion CR optimoinnissa on tapahtunut ohjelmavirhe, jollainen oli myös mutaatiovakion optimoinnissa. Optimiarvon tasaisuus risteytysparametrin optimialueella saadaan lisäämällä risteytysparametrin ohjelmaan satunnaislukujen (rand3) ja (rand4) satunnaislukusekoitin. Risteytys-vaiheessa säätimen yriteparametreiksi on valittu merkityt parametrit Kp, Ki, Kd (kuva 28). Kuvaajasta nähdään, että sukupolven 21 ajokerroilla 4, 5 ja 6 (kuva 28) on risteytetty vain ensimmäinen parametri (Kp). Optimiparametreiksi on valittu sukupolven 20 parametrit ajokerroilta 1, 2 ja 3, jossa parametrien ympärillä on tummennettu reunaviiva (kuva 28). Mutaatioparametrien sopiva hajonta ajokertojen yksi ja kolme välillä on vaikuttanut hyvään optimoinnin lopputulokseen.



Kuva 29. Mutaatio- ja risteytysvakioiden arvot optimiajossa ajokerroittain. Mutaatiovakion (F) ja risteytysvakion (CR) arvot ovat itse-adaptoituvia.

6.2. PID-säädin



Kuva 30. PID-säätimen testaus optimiarvoilla. PID-säätimelle on annettu ohjearvo (Ref), joka on asetettu nollakohdaksi kuvaajalla. Ohjearvon ja mittausarvon erotusarvoa on kuvattu arvolla (error). Suhdeosa (P-termi), integroiva osa (I-termi) ja derivoiva osa (D-termi) ovat erosuureeseen verrannollisia termejä. Säädön ulostulo (Control) vastaa edellisten termien yhteisarvoa.

PID-säädintä on testattu yriteparametreilla sukupolvittain. Optimoinnin lopuksi on valittu optimiparametrit sukupolven 20 yriteparametreista (kuva 28 ja kuva 30). Säätimen erotusarvo (error) on luotu satunnaisesti simuloimalla (erotusarvo signed-luvuilla) (kuva 26). Kuvasta 30 ja taulukosta 11 voidaan nähdä, että ohjaus (control) seuraa voimakkaasti vahvistusarvoa (P-termi). Integrointiarvo (I-termi) pysyy vakiona testausajon aikana ja derivointiarvo (D-termi) vahvistuu vähän optimointiajon loppua kohden. Derivointiarvo vahvistuu, koska erotusarvo (error) kasvaa optimoinnin loppua kohden. Erotusarvo kasvaa, koska se on luotu satunnaisesti simuloimalla eikä vastaa oikeaa käytännöllistä arvoa. Ohjausarvo (control) vastaa P-termin, I-termin ja D-termin yhteisarvoa.

7. JOHTOPÄÄTÖKSET

Diplomityössä suunniteltiin PID-säätimen parametrien optimointi differentiaalievoluutiomenetelmällä ja digitaalisella Field Programmable Gate Array (FPGA) ohjelmalla Alteran DE2 kehitys- ja koulutusalustalla käyttäen (VHDL) laitteistokuvauskieltä. Ohjelman testaus tapahtui ModelSimilla ja tulosten analysointi Matlabilla. Tavoitteena oli kehittää reaaliaikainen PID-säätimen optimointi juuri FPGA:lla, jonka toiminta on digitaalista, reaaliaikaista, vakaata ja tehonkulutukseltaan pientä. Differentiaali-evoluutioalgoritmi valittiin optimointitehtävään sen yksinkertaisuuden ja laajan sopivuuden ansiosta erilaisissa tarkkoissa optimointitehtävissä kuten PID-säädin on. Säätimen ohjelma suunniteltiin toteuttavaksi kokonaisluvuilla kiinteän pilkun (<16,9>) bitin esityksenä, jolla on kahden-kolmen desimaalin tarkkuus. Optimointiin menevä suoritusaika mitattiin ModelSimilla PID-säätimen ohjelmaa testaten. Optimoinnissa suunniteltiin käytettäväksi 500 Hz näytteenottonopeutta. Testauksen aikana käytettäessä 50 MHz näytteenottonopeutta (maksiminopeus) viidenkymmenen sukupolven optimointisuoritukseen kului 65,62 µs. Kun käytettiin suunnitelman mukaista 500 Hz:n näytteenottonopeutta samaan tehtävään kului 6,562 s. Testauksen perusteella suorituksen nopeuteen vaikutti näytteenottoaika ja sukupolvien määrä. Optimointisuorituksessa kellojaksojen määrä viidenkymmenen sukupolven aikana oli 3821 kellojaksoa, joka pysyi vakiona sukupolvien määrän pysyessä samana.

Kirjallisuuskatsaukseen etsittiin tietoa erilaisista differentiaalievoluutio-, PID-säädin ja FPGA julkaisuista. Julkaisut sisälsivät tietoa erilaisien differentiaalialgoritmien ja PID-säädinten rakenteesta sekä FPGA-ohjelmien toteutuksista kiinteän pilkun kokonaisluvuilla. Lisäksi löytyi paljon tietoa erilaisista PID-säätimen kokeellisesti optimoitavista viritysmenetelmistä. Edellä saatujen tietojen perusteella lähdettiin toteuttamaan FPGA:lla toteutettavaa ja DE-menetelmillä optimoitavaa PID-säädintä.

Työssä kehitettiin alkuperäistä differentiaalievoluutiolaskentaa lisäämällä siihen yksilöiden suuruusjärjestysvertailu ja itse-adaptoituvat mutaatioparametri F ja risteytysparametri CR. Edellä mainittua differentiaalievoluutioalgoritmia kutsutaan jDE-algoritmiksi (jDE/rand/1/bin), johon on lisätty ranking-perusteinen mutaatio-operaatio. Yksilöiden suuruusvertailulla parannettiin satunnaisparametrien laatua ennen mutaatiolaskentaa, hidastettiin liiallista suppenemisnopeutta ja parannettiin optimoinnin laatua. Itse-adaptoituvien ohjausparametrien ansiosta käyttäjän ei tarvitse arvailla hyviä

mutaatio- ja risteytysvakioiden arvoja. Itse-adaptoituvilla ohjausparametreilla löydetään myös oikea alue mutaatio- ja risteytysparametrille säätösovelluksesta riippuen.

PID-säätimen virityksessä oli useita erilaisia vaihtoehtoja kuten askelvaste-, värähtelyrajamenetelmä ja optimiasetuskriteeri sekä numeerinen ja ekstrapoloiva viritysmenetelmä. Edellä olevat viritysmenetelmät ovat yleisimpiä kokeellisia PID-säätimen viritysmenetelmiä. Käyttämällä differentiaalievoluutiolaskentaa voidaan PID-säätimelle kokeellisesti ja reaaliaikaisesti virittää oikeat säätimen ohjausparametrien optimiarvot. Virityksessä päädyttiin numeeriseen optimointiin, jossa minimoidaan erotusarvosta ja derivointitermistä riippuvaa kustannusarvoa. Kustannusarvo voidaan laskea eri tavalla painotetuista kustannusfunktioista, joita ovat integraalin absoluuttinen virhe (IAE), integraalin neliöllinen virhe (ISE), aikaintegraalin ja neliöllisen virheen tulo (ITSE) ja ajan ja virheen tulon neliöllinen virhe (ISTE). Yllä mainittujen kustannusfunktioiden avulla voidaan optimoida kokeellisesti koeteltua ja matemaattisessa muodossa olevaa mittaustietoa. Hyvyysarvon eli kustannusarvon laskennassa kustannusfunktiona käytettiin integraalin absoluuttista virhettä (IAE), jossa erotusarvosta lasketaan tietyn näytemäärän mukainen hyvyysarvo sukupolvittain. Lopuksi eri sukupolvien hyvyysarvoja vertailemalla valitaan pienimmän hyvyysarvon sisältämä sukupolvi PID-säätimen optimiparametreiksi.

Toteutuksessa PID-säätimen ohjelma luotiin FPGA-ohjelman rakennemallilla, jossa DE-algoritmin arkkitehtuuri luotiin säätimen päärakenteeksi. DE arkkitehtuurin alarakenteeseen sijoitettiin PID-säätimen arkkitehtuuri. Järjestelmään kului resursseja kaikkiaan 11821 logiikkasolua. FPGA-ohjelman resurssit on laskettu logiikka-elementtien ja niiden LUT-tulojen käytön, rekisterien, I/O-tulojen ja -lähtöjen ja kertolaskimien määrän mukaan. Kokonaistehonkulutukseksi käytetyillä resursseilla Alteran PowerPlay -tehoanalysaattori antoi 119 mW, joka ylittää vähän Cyclone III EP3C16 tuoteperheen kymmenentuhannen logiikkayksikön 161 mW tehonkulutuksen. DE-arkkitehtuurin sisällä tilakoneella FSM ohjataan differentiaalievoluutio-optimoinnin ja PID-säätimen toiminnan kulkua optimoinnin aikana. Satunnaislukugeneraattoreilla (RNG) ja (random) luodaan DE-optimoinnissa tarvittavat yksilöt, joista muodostetaan optimoinnin aikana säätimen optimaaliset ohjausparametrit. Satunnaisparametrien sekoitusmodulilla (RNG-sekoitus) sekoitetaan satunnaisparametreja niiden arvojen parantamiseksi, jotka on ohjelmassa luotu näennäissatunnaisesti. Ranking-modulilla järjestetään satunnaisesti luodut yksilöt suuruusjärjestykseen. Valitaan suurimman yksilön ensimmäinen parametri mutaatio-operaation ensimmäiseksi parametreiksi ja

toiseksi suurimman yksilön toinen parametri mutaatio-operaation toiseksi parametriksi. Mutaatio-operaation kolmas parametri valitaan satunnaisesti. Mutaatiovektorimodulilla etsitään uusia parametreja joilla parannetaan optimointilaatua. Itse-adaptoituvalla mutaatiovakion F arvolla pyritään vaikuttamaan uusien optimaalisten parametrien syntyyn. Hyvyysarvomodulilla etsitään PID-säätimen ohjausparametreille optimiarvoa kustannusfunktion hyvyysarvon perusteella. Risteytys- ja valintamodulilla muodostetaan uusia yksilöitä, jota ohjelma yrittää parantaa sukupolvittain etenevässä optimoinnissa. Parhaan yksilön valinta tapahtuu optimoinnin päätteeksi hyvyysarvon perusteella. Ranking-pohjaisen jDE-algoritmin arkkitehtuuri on onnistunut helpottamaan ja nopeuttamaan suurien populaatioiden laskentaa, koska ranking-modulilla voidaan käsitellä useita populaation yksilöitä samanaikaisesti. Ranking-modulille syötetään sisään yhtäaikaisesti kaikki populaation yksilöt ja ohjataan ulos suuruusjärjestykseen rankattu yksilö. Edellä mainittu yksilö ajetaan tämän jälkeen mutaatiolaskentaan. Mutaatiovektori luodaan aina kolmesta rankatusta yksilöstä. Mutaatiovektori risteytetään kohdevektorin kanssa, joka sisältää vanhan yksilön parametrit. Kun on risteytetty sukupolvia riittävästi populaatio lopulta konvergoituu ja löydetään jokin minimi hyvyysarvo optimoiduille ohjausparametreille kohdefunktion avulla.

Differentiaalievoluutioarkkitehtuurin toteutuksessa keskusmodulina toimi tilakone FSM, jolla käynnistettiin ohjelmamodulit oikeassa järjestyksessä. Tilakoneen valmiustilassa (nollaus) nollattiin kaikkien modulien ohjaukset. Tilakoneen käynnistätilassa (käynnistä) optimointi aloitettiin, satunnaislukugeneraattorit käynnistettiin, syötettiin aloitusparametrit PID-säätimelle, estetään risteytys ajettaessa aloitusparametreilla ja käynnistettiin ensimmäisen sukupolven hyvyyslaskenta. Tilakoneen risteytystila (risteytys) käynnistyy kun aloitusparametreilla tehty hyvyyslaskenta on päättynyt. Risteytystilassa käynnistetään risteytys ja aloitetaan toisen sukupolven hyvyyslaskenta. Tilakoneen vaihe kolme (ohjausparametrien nollaus) toimii risteytyksen ja hyvyyslaskennan ohjausparametrien nollausvaiheena. Tilakoneen vertailutilassa (vertailu) suoritetaan sukupolvittainen hyvyysarvovertailu ja käynnistetään risteytys ja uuden sukupolven hyvyyslaskenta. Tilakoneen ohjausparametrien nollauksen ja vertailuvaiheiden välillä on kierto kunnes sukupolvien määrä täyttyy tai optimointi pysäytetään.

PID-säätimen arkkitehtuuri on moduliaarinen, koostuen kolmesta laskentavaiheesta. Ensimmäisessä vaiheessa lasketaan suhdeosan (Propotional gain) vahvistus. Toisessa

vaiheessa lasketaan integroivan osan (Integral gain) vahvistus. Kolmannessa vaiheessa lasketaan derivoivan osan (Derival gain) vahvistus. Säätimen ohjelmassa on käytetty diskreettiä laskentatapaa suhdeosan sekä integroivan ja derivoivan osan laskemisessa. Säätimen toteutuksessa on hyödynnetty kerto-, yhteen- ja vähennyslaskua. Lisäksi eri laskentavaiheissa on viivästetty signaalia digitaalisesti kellopulssin avulla PID-säätimen ohjausarvon laskemiseksi.

Säätimen ohjelma testattiin Modelsimilla. Ohjelman simuloinnissa annettiin ohjearvo (Ref) suuruudeksi 2000, asetettiin satunnaisgeneraattorien syöttönopeudeksi 25 MHz:a ja sukupolvien määräksi 24. Tämän jälkeen ajettiin simulointiohjelmalla kaksi optimointiajoa, joista viimeisestä testiajosta valittiin sukupolvet 20, 21 ja 22 testiotokseen. Simuloinnin tulokset kerättiin taulukoihin. Taulukoihin kerättiin satunnaisesti luodut yksilöiden parametrit suuruusjärjestysvertailun ja ranking-parametrien valintavertailun perusteella, mutaatio- ja risteytysarvojen laskenta-arvot tuloksineen, PID-säätimellä ajetut hyvyysarvot ja PID-säätimen testaustulokset ja erikseen ajettu mutaatio ja risteytys vakioiden arvot. Taulukot sisältävät kaikkien modulien yhden optimointiotoksen testausvaiheet alusta loppuun. Taulukkojen tiedot siirrettiin Matlabiin, jossa taulukkojen arvoista muodostettiin kuvaajat ja analysoitiin tuloksia niiden perusteella. Kuvaajista nähdään optimoinnin eteneminen ajokerroittain. Satunnaisparametrien vaihtelu oli onnistunut satunnaislukusekoittimen avulla hyvin. Satunnaisarvojen vaihtelualue voisi kuitenkin olla vieläkin laajempi kuin nykyinen satunnaislukusekoittimeen perustuva versio. Satunnaislukugeneraattoreiden vaihtelualuetta voisi parantaa esimerkiksi muuttamalla Fibonacci toteutuksen LFSR-sekvenssin kokoa. Ohjaamalla satunnaisesti multiplexeria kellopulsseittain $2^{(2,3,\ldots,n)}$ välillä käyttämällä oletuspolynomiarvoja. Testausotoksessa osa ajokertojen satunnaisparametreista ajettiin ranking-vertailussa kahteen kertaan. Virhe aiheutti suuruusjärjestysvertailussa virheellisyyttä kun vektori-indeksiarvo oli arvossa yksi kaksi tai yli kaksi kellopulssia kerrallaan. Ohjelma korjattiin synkronoimalla ranking-moduli ja mutaatiolaskenta mutaatiokellon avulla samaan rytmiin.

Mutaatio- ja risteytysmodulilla tarvittavien itse-adaptoituvien ohjausparametrien vaihtelu jäi optimointiajossa vähäiseksi. Arvojen vaihtelun vähäisyys johtui näennäissatunnaisgeneraattorien samasta kellotahdista, jolloin vertailtavien satunnaislukugeneraattorien arvot olivat samanlaisia. Tämä aiheutti mutaatiovakioon matalia arvoja ja risteytysvakioon matalien ja korkeiden arvojen vaihtelua. Korjaus

voidaan tehdä satunnaislukusekoittimella ja -generaattorin arvoaluetta laajentamalla. Mutatointi ja risteys onnistuivat virheistä huolimatta odotetusti.

PID-säädin testauksessa saatiin kuvaaja, josta voidaan vertailla säätimen eri osa-alueita. Erotusarvo on luotu satunnaisesti matkimalla oikeaa säätötilannetta n. 5 %:iin ohjearvosta, joka oli asetettu arvoon 2000. Kuvaajassa säätimen termejä voidaan verrata erotusarvon suuruuteen. Testausajossa erotusarvo kasvoi optimoinnin loppua kohden. Suhdeosa (P-termi) vaikuttaa voimakkaasti säätimen nopeaan reaktioon ohjauksessa (Control). Integrointiosa (I-termi) pysyy vakiona ja derivointiosa (D-termi) kasvaa erotusarvon kasvaessa. Ohjelmavirheenä säätimen derivointiosasta puuttui kerroin kaksi. Kertoimen lisäys kasvattaisi derivointiosan ja säätimen ohjauksen suuruutta, mutta samalla ohjausarvo muuttuisi vakaammaksi.

Tulevaisuudessa ohjelmaa ja sen eri osa-alueita voitaisiin kehittää. Satunnaisluku-generaattorin toimintaa voitaisiin parantaa monimutkaistamalla satunnaisuutta, laajentamalla satunnaislukualuetta vähentämällä samalla satunnaislukujen toistuvuutta. Lisäksi kasvattamalla populaation kokoa, jolloin myös satunnaisparametrien määrä kasvaisi ja ranking-pohjaiseen mutaatiolaskentaan saataisiin lisää monimuotoisuutta.

LÄHDELUETTELO

Aalto yliopisto, Automaatio- ja systeemitekniikan laitos(2014). *PID-säätö* [verkkojulkaisu]. [siteerattu 25.01.2015]. Saatavissa: ftp://noppa.aalto.fi/noppa/kurssi/as-0.2230/materiaali/AS-0_2230_tyo_11__tyoohje_2.pdf

Aalto yliopisto, Digitaalisen säädön verkkokurssi (2011a). *Diskreetti PID-säädin* [verkkojulkaisu]. [siteerattu 25.03.2014]. Saatavissa: ftp://autsys.aalto.fi/pub/control.tkk.fi/Kurssit/Verkkokurssit/AS-74.2112/oppitunti8/diskreettipid.html.

Aalto yliopisto, Analogisen säädön verkkokurssi (2011b). *PID-säätimen kokeellinen virittäminen* [verkkojulkaisu]. [siteerattu 25.03.2014]. Saatavissa: ftp://autsys.aalto.fi/pub/control.tkk.fi/Kurssit/Verkkokurssit/AS-74.2111/kehittyneet/oppitunti12/kokeellinen.html.

Actel Corporation (2008). Zero-Power or NOT Zero-Power: That Is the Question. *Actel eZone* [verkkojulkaisu] 2008:1 [Siteerattu 05.05.2015], 3. Saatavissa http://www.actel.com/eZone/Q108/p3.html

Alander, Jarmo, Digitaalitekniikan perusteet (2015). *Lukujärjestelmät ja koodit* [verkkojulkaisu]. [siteerattu 12.01.2015]. Saatavissa: ftp://lipas.uwasa.fi/~TAU/AUTO1010/slides.php? File=0200Bin.txt.

Anumandla, Kiran Kumar, Rangababu Peesapati, Samrath L. Sabat, Siba K. Udgata & Ajith Abraham (2013). Field programmable gate arrays-based differential evolution coprocessor: a case study of spectrum allocation in cognitive radio network. *IET Computers & Digital Techniques 7:*5, 221-234. doi:10.1049/iet-cdt.2012.0109.

Bishop, David(2008). *Fixed point package user's guide* [verkkojulkaisu]. [siteerattu 25.05.2014]. Saatavissa: ftp://www.vhdl.org/fphdl/Fixed_ug.pdf.

Brest, Janez, Aleš Zamuda, Borko Bošković, Sašo Greiner & Viljem Žumer (2008). An analysis of the control parameters adaptation in DE. *Advances in Differential Evolution, Springer Berlin Heiderberg* 143, 89-110. doi:10.1007/978-3-540-68830-3_3.

Brest, Janez, Sašo Greiner, Borko Bošković, Marjan Mernik & Viljem Žumer (2006a). Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *Evolutionary computation, IEEE Transactions on 6, 646-657*.

Brest, Janez, Borko Bošković, Sašo Greiner, Viljem Žumer & Mirjam Sepesy Maučec (2006b). Performance comparison of self-adaptive and adaptive differential evolution algorithms. *Soft Computing* 11:7, 617-629. doi:10.1007/s00500-006-0124-0.

Brest, Janez & Mirjam Sepesy Maučec (2010). Self-adaptive Differential Evolution Algorithm using population size reduction and three strategies. *Soft Computing* 15:11, 2157-2174. doi:10.1007/s00500-010-0644-5.

Chakraborty Uday K. (2008). *Advances in Differential Evolution,* Studies in Computational Intelligence. Springer-Verlag Berlin Heidelberg, 339.

Chan, Yuen Fong, M. Moallem & Wei Wang (2007). Design and implementation of modular FPGA-based PID controllers. *Industrial Electronics, IEEE Transactions on* 4, 1898-1906.

Das Swagatam, Ajith Abraham, Uday K. Chakraborty & Amit Konar (2009). Differential evolution using a neighborhood-based mutation operator. *Evolutionary Computation, IEEE Transactions on 3, 526553*.

Dingyu, Xue, YangQuan Chen & Derek P. Atherton (2007). *Advances in Design and Control. Linear feedback control: Analysis and design with MATLAB, 183-235* [verkkojulkaisu]. [siteerattu 15.04. 2014]. Saatavissa: ftp://bookstore.siam.org/dc14.

Gong, Wenyin & Zhihua Cai (2013a). Differential evolution with ranking-based mutation operators. *Cybernetics, IEEE Transactions on 6, 2066-2081*.

Gong, Wenyin & Zhihua Cai (2013b). *Accelerating parameter identification of proton exchange membrane fuel cell model with ranging-based differential evolution, 356-364. School of Computer Science, China University of Geosciences* [verkkojulkaisu]. [siteerattu 13.05.2014]. Saatavissa: ftp://cs.cug.edu.cn/teacherweb/gwy/Publication/Energy-2013.pdf.

Gordon, Robert (1998). A calculated look at fixed-point arithmetic. *EE Times-India, April 1998, 3.*

Greenwood, Garrison W. and Qiji J. Zhu (2001). Convergence in evolutionary programs with self-adaption. *Evolutionary Computation 2, 147-157*.

Grout, Ian (2008). Digital Systems Design with FPGAs and CPLDs. *Newnes*, 784.

Ilonen, Jarmo, Joni Kristian Kämäräinen & Jouni Lampinen (2003). Differential evolution training algorithm for feed-forward neutral networks. *Neutral Processing Letters* 1, 93-105.

Koskela, Tero (2010). Kuormamekaniikan vaikutus sähkökäytön viritykseen. *Tampereen teknillinen yliopisto, Sähkötekniikan laitos*.

Lampinen, Jouni (2001). Global Optimization by Differential Evolution. *Lappeenrannan teknillinen yliopisto, Tuotantotalouden tiedekunta. Tietotekniikka* [verkkojulkaisu]. [siteerattu 15.03. 2014]. saatavissa: ftp://www.it.lut.fi/kurssit/01-02/010778000/DE.pdf.

Lima, João, Ricardo Menotti, João M. P. Cardoso & Eduardo Marques (2006). A methodology to design FPGA-based PID controllers. *Systems, Man and Cybernetics, IEEE Transactions on*, 2577-2583.

Mushrat, Ali, Millie Pant & Ajith Abraham (2009). Simplex Differential Evolution. *Acta Polytechnica Hungarica* 5, 95-115.

Niiranen, J. (1999). *Sähkömoottorikäytön digitaalinen ohjaus.* Otatieto, Helsinki.

Oberstar, Erick L. (2007). Fixed-Point Representation & Fractional Math. *Report Oberstar Consulting*, 1-18 [verkkojulkaisu]. [siteerattu 03.04. 2014]. Saatavissa: ftp://www.superkits.net/whitepapers/Fixed%20Point%20Representation%20&%20Fractional%20Math.pdf.

Penttinen, Aki (2005). FPGA:lle sulautetulla mikroprosessorilla toteutettu sähkökäytön säätöjärjestelmä. *Lappeenrannan teknillinen yliopisto, Sähkötekniikan osasto Teollisuuselektroniikan laitos*.

Price, Kenneth V, Rainer M. Storn (1997). Differential Evolution - A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization* 4, 341-359.

Price, Kenneth V, Rainer M. Storn & Jouni A. Lampinen (2005). Differential Evolution A practical approach to global optimization. *Natural Computing Series*,1-539.

Qin, A. K & P.N. Suganthan (2005). Self-adaptive differential evolution algorithm for numerical optimization. *Evolutionary Computation, IEEE Congress on, 2005* 2, 1785-1791. doi:10.1109/CEC.2005.1554904.

Rönkkönen, Jani (2009). Continuous multimodal global optimization with differential evolution-based method. *Lappeenrannan teknillinen yliopisto, Teknistaloudellinen tiedekunta Tietotekniikka, Konenäön ja hahmontunnistuksen laboratorio. ISBN*: 978-952-214-852-0.

Rönkkönen, Jani (2003). Normaalijakaumaan perustuva mutaatio-operaatio differentiaalievoluutioalgoritmiin. *Lappeenrannan teknillinen yliopisto, Tietotekniikan osasto*.

Rönkkönen, Jani, Saku Kukkonen & Kenneth V. Price (2005). Real parameter optimization with differential evolution. *Evolutionary Computation, IEEE Congress on, 2005 Edinburg* 1, 506-513. doi:10.1109/CEC.2005.1554725.

Saad, Mohd Sadli, Hishamuddin Jamaluddin & Intal Zaurah Mat Darus (2012). PID controller tuning using evolutionary algorithms. *WSEAS Transactions on Systems and Control* 4, 139-149.

Simon, Dan (2013). *Evolutionary Optimization Algorithms: Biologically Inspired and Population-based Approaches to Computer Intelligence.* Wiley, 293-310.

Swagatam, Das, Ajith Abraham, Uday K. Chakraborty & Amit Konar (2009). Differential Evolution Using a Neighborhood-Based Mutation Operator. *IEEE Transactions on Evolutionary Combutation* 13:3, 526-553.

Tan, Wen, Jizhen Liu, Tongwen Chen & Horacio J. Marquez (2006). Comparison of some well-known PID tuning formulas. *Computers & Chemical Engineering* 30:9, 1416-1423. doi:10.1016/j.compchemeng.2006.04.001.

Trimeche, Abdesselem, Anis Sakly, Abdelatif Mtibaa and Mohamed Benrejeb (1996). PID controller using FPGA technology. *Advances in PID Control. InTech,* 259-274. doi:10.5772/18295.

Urriza, L.A, D. Navarro, J.I. Artigas & O. Lucia (2010). Word length selection method for controller implementation on FPGAs using the VHDL-2008 fixed-point and floating-point packages. *EURASIP Journal on Embedded Systems* 2010:593264, 1-11. doi:10.1155/2010/593264.

Yuen Fong Chan, M. Moallem & Wei Wang (2007). Design and implementation of modular FPGA-based PID controllers. *Industrial Electronics, IEEE Transactions on* 54:4, 1898-1906.

Zhang, Jingqiao & Arthur C. Sanderson (2009a). Adaptive differential evolution: a robust approach to multimodal problem optimization. *Springer Science & Business Media*, 1-79.

Zhang, Jingqiao & Arthur C. Sanderson (2009b). JADE:Adaptive Differential Evolution with optional external archive. *Evolutionary Computation, IEEE Transactions on* 13:5, 945-958. doi:10.1109/TEVC.2009. 2014613.

Åström K. & T. Hägglund (1995). *PID Controllers: Theory, Design and Tuning.* International Society for measurement and Control, Durham.