Tampereen yliopisto

Noora Kukkonen

SIMC-VIRITYSTYÖKALU MATLABILLE

Tekniikan ja luonnontieteiden tiedekunta (ENS)
Kandidaatintyö
Huhtikuu 2020

TIIVISTELMÄ

Noora Kukkonen: SIMC-viritystyökalu Matlabille
Kandidaatintyö
Tampereen yliopisto
Automaatiotekniikka
Huhtikuu 2020

PID-säädin on säätötekniikassa yksi yleisimmin käytetyistä säätöalgoritmeista. Kuitenkin nämä säätimet ovat usein huonosti viritettyjä ja näin ollen aiheuttavat järjestelmässä esimerkiksi epätoivottua ylitystä askelvasteeseen tai heikkoa häiriövastetta. Tässä työssä tarkastellaan PID-säätimelle kehitettyä SIMC-viritysmenetelmää ja toteutetaan kyseiselle menetelmälle viritystyökalu Matlab-ohjelmistolla.

Ennen kuin työssä käsitellään SIMC-viritysmenetelmään, tutkitaan PID-säädintä. Työssä käydään läpi säätimen rakennetta ja sitä, miten eri säätimen osat vaikuttavat järjestelmän vasteeseen. Lisäksi käydään läpi erilaisia esitystapoja PID-säätimelle. PID-säätimen viritystä käydään myös yleisesti läpi. Tämän jälkeen esitetään viritysmetodista esimerkkinä Ziegler– Nichols -viritysmenetelmä, joka on esimerkiksi prosessiteollisuudessa yleisesti käytetty viritysmenetelmä sen yksinkertaisuutensa vuoksi.

PID-säätimen teoriaosuuden jälkeen käsitellään SIMC-viritysmenetelmää. Sigurd Skogestadin kehitti vuonna 2001 viritysmenetelmän, joka nimettiin SIMC-metodiksi. SIMC-viritysmetodi on kaksiosainen. Ensimmäisessä osassa tehdään järjestelmästä approksimaatiomalli ja toisessa osassa lasketaan PID-säätimelle viritysparametrit. Järjestelmän approksimaation voi tehdä joko kokeellisesti tai järjestelmän siirtofunktiosta. Siirtofunktiosta tehtävälle approksimaatiolle on omat kaavansa ja sille on tiettyjä rajoitteita, joihin perehdytään työssä tarkemmin. Kokeellisesti approksimaatio voidaan tehdä järjestelmän avoimesta- tai suljetusta vasteesta. Approksimaatioksi saadaan ensimmäisen- tai toisen kertaluokan siirtofunktio, josta voidaan laskea PID-säätimelle viritysparametrit työssä esiteltävien kaavojen avulla. Parametrit lasketaan PID-säätimen kaskadiesitysmuodolle, joka pitää ottaa huomioon PID-säätimen parametrejä käytettäessä.

Tässä työssä on tehty teorian tarkastelun lisäksi Matlab-ohjelmalla työkalu, joka laskee viritysparametrit PID-säätimelle automaattisesti SIMC-viritysmenetelmän muokaisesti. Työkalua ohjelmoitaessa on tehty valinta niin, että SIMC-virityksen ensimmäinen osa eli approksimaatio tehdään vain järjestelmän siirtofunktiosta eikä kokeellisesti vasteesta laskettavaa approksimaatiota voida tällä työkalulla laskea ollenkaan. Järjestelmän siirtofunktion parametrit syötetään työkalulle komentokehotteen kautta, joka toimii myös työkalun käyttöliittymänä. Työssä myös esitetään kaksi esimerkkitapausta työkalun käytöstä.

Avainsanat: PID-säädin, SIMC, Matlab

ALKUSANAT

Alunperin suunnittelin, että tekisin kandin vuoden 2019 kesällä. Loppujen lopuksi kesä venähti melkein vuodeksi, mutta kyllä tämän lopulta sain valmiiksi sopivasti viimeisten kandivaiheen kurssien kanssa samaan aikaan. Kiitokset ohjaajalleni Veli-Pekka Pyrhöselle, joka kommentoi kandini versioita koko vuoden ajan. Kiitokset myös vanhemmilleni sekä kavereilleni, jotka jaksoivat kuunnella ajatuksiani kandityön kirjoittamisesta.

Tampereella, 29. huhtikuuta 2020

Noora Kukkonen

SISÄLLYSLUETTELO

1 JOHDANTO

Säätötekniikassa yksi yleisimpiä säätimiä, joita käytetään takaisinkytketyssä säädössä, on PID-säädin. Säätimen virittäminen on kuitenkin haastavaa huolimatta sen vain kolmesta viritysparametrista. Tähän on kehitetty useita valmiita viritystapoja, mutta monissa niistä on ongelmia. Esimerkiksi Zielger–Nichols-viritys tuottaa usein ylitystä askelvasteeseen, vaikka antaakin integroiville prosesseille hyviä häiriövasteita.

Yhden PID-säätimen valmiista viritysmetodista kehitti Sigurd Skogestad vuonna 2001, joka nimettiin Skogestad Internal Model Control -metodiksi eli SIMC-metodiksi. Koska monet PID-säätimet olivat huonosti viritetty, halusi Skogestad kehittää yksinkertaisen ja helpon prosessimalliin perustuvan virityssäännön. Virityssääntö on luotu ensimmäisen- tai toisen kertaluokan siirtofunktiomalleille, joten ennen kuin virityssääntöä voidaan käyttää, tarvitsee järjestelmämalli approksimoida 1. kertaluokkaan tai 2. kertaluokkaan. Approksimaatio voidaan tehdä joko avoimen- tai suljetun silmukan askelvasteesta sekä järjestelmän korkeamman asteen siirtofunktiosta. [1]

Tässä työssä toteutetaan menetelmään sopivia Matlab-funktioita. Viritystyökalu on luotu niin, että pääfunktio kutsuu muita funktioita, missä syötettyjen parametrien avulla tehdään laskuoperaatiot. Työkalu laskee approksimaatiot sekä PI- tai PID-säätimen parametreille järjestelmän siirtofunktion parametreistä. Viritystyökalun lähdekoodi on liitteessä A.

Tässä työssä PID-säätimestä yleisesti kerrotaan luvussa 2. Luvussa 3 puolestaan syvennytään SIMC-viritysmetodiin ja siihen liittyvään mallien approksimaatioihin. Viimeisenä luvussa 4 kerrotaan työssä tehdystä viritystyökalusta ja sen esimerkkikäytöstä.

2 PID-SÄÄDIN

PID-säädin on yksi yleisimmin käytetyistä takaisinkytketyn säädön algoritmeista. Takaisinkytkennän lohkokaaviokuva on kuvassa 2.1, jossa $C(s)$ ja $P(s)$ ovat säätimen ja prosessin siirtofunktiomalleja. PID-säädin on kaikkein yleisin säädin prosessiteollisuuden sovelluksissa, mutta myös muilla prosessin säätöä vaativilla järjestelmillä säädintä käytetään laajasti. PID-säätimestä löytyy paljon kirjallisuutta, ja siitä tiedon hakeminen on helppoa. Tämän vuoksi PID-säätimen viritykseen löytyy myös paljon erilaisia metodeita, mikä helpottaa viritystä ja oikean metodin löydyttyä antaa prosessille haluttuja tuloksia. [2]

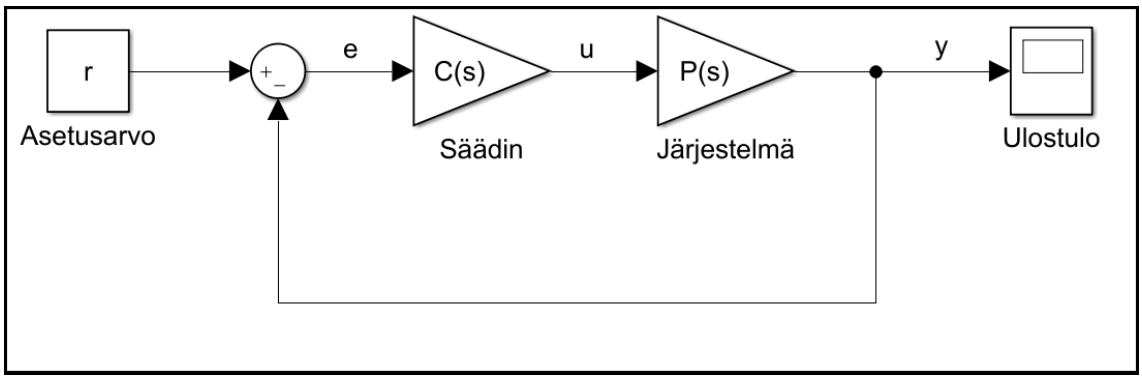
PID-säätimen ohjausfunktion kaava on

$$u(t) = K_p \left(e(t) + \frac{1}{T_i} \int_0^t e(\pi) d\pi + T_d \frac{de(t)}{dt} \right), \tag{2.1}$$

jossa $u(t)$ kuvaa ohjaussignaalia eli PID-säätimen lähtöä, K_p proportionaalivahvistusta, T_i integrointiaikaa ja T_d derivointiaikaa sekä $e(t)$ merkitsee erosuuretta eli asetusarvon ja koko järjestelmän lähdön $y(t)$ (katso kuva 2.1) erotusta [3]. PID-säätimen eräs toinen esitystapa on sen siirtofunktio. Yleisimmät esitysmuodot tästä ovat rinnankytkentämuoto

$$C(s) = K_p + K_i \frac{1}{s} + K_d s, \tag{2.2}$$

jossa K_i on integrointivahvistus ja K_d on derivointivahvistus sekä sarjaankytkentämuoto



Kuva 2.1. *Takaisinkytketty säätö*

(kirjallisuudessa myös englanniksi ideal form) [4] [5]

$$C(s) = K_p \left(1 + \frac{1}{T_i s} + T_d s\right). \tag{2.3}$$

Eri virityssäännöissä voidaan laskea eri muotojen viritysparametreja. Esimerkiksi, jos virityssääntö laskee parametrit K_p, K_i ja K_d PID-säätimen muodolle (2.2) viritysparametrit, ei niitä voi käyttää sellaisenaan säätimelle, jonka muoto on (2.3). Viritystä tehtäessä täytyy siis ottaa huomioon, missä muodossa säätimelle pitää antaa parametrit.

Luvussa 3 käsiteltävän SIMC-virityksen parametrit annetaan PID-säätimen kaskadimuodolle (kirjallisuudessa esiintyy myös sarjamuoto, integoivamuoto),

$$C(s) = K_c \left(\frac{\tau_I s + 1}{\tau_I s}\right)(\tau_D s + 1), \tag{2.4}$$

jossa K_c on säädön vahvistus, τ_I on integrointiaika ja τ_D on derivointiaika. Jos verrataan kaskadimuodon vahvistusta K_c ja sarjaankytkentämuodon proportionaalivahvistusta K_p, saadaan niiden välille yhteys

$$K_c = K_p \left(1 + \frac{T_d}{T_i}\right), \tag{2.5}$$

josta nähdään, etteivät vahvistukset vertaudu suoraan keskenään [6]. Säätimen muotoa (2.4) käytetään SIMC-virityksessä siksi, että derivointiosan käsittely on helpompaa tässä muodossa, koska se on erotettu vahvistuksesta ja integrointiosasta. [1] [7]

2.1 PID-säätimen osat

PID-säätimen nimi tulee proportionaali-, integrointi- ja derivointiosasta, joista säädin koostuu. Propertionaaliosan

$$u(t) = K_p e(t) + u_b \tag{2.6}$$

on tarkoitus saada järjestelmän lähtö $y(t)$ lähestymään asetusarvoa. Proportionaaliosaa kuvaa kaava (2.6), jossa u_b tarkoittaa bias-arvoa. Arvo u_b on ohjauksen suuruus silloin, kun erosuure $e(t)$ on nolla. Yleensä u_b määritellään $(u_{min} + u_{max})/2$, missä u_{min} on säädettävän järjestelmän minimiraja ja u_{max} maksimiraja. Kuitenkin u_b voidaan asettaa myös manuaalisesti niin, että tasapainotilan erosuureen suuruus on nolla. [3] [8]

Integrointiosan tarkoitus on hienosäätää järjestelmän lähtö täsmälleen vastaamaan asetusarvoa tasapainotilassa. Toisin sanoen integrointiosa osaa asettaa kaavan (2.6) biasin u_b arvon tarkaksi automaattisesti niin, että vakaan tilan virhe on nolla. Kun integrointihaara on rinnankytkettynä P-osan kanssa, kyseessä on PI-säädin. Tämä esitetään kaavalla

$$u(t) = K_p(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau), \tag{2.7}$$

jossa T_i on integrointiosan integrointiaika. [2][3]

Kolmas PID-säätimen osa on derivointiosa. Derivointiosan tehtävä säätimessä on lisätä suljetun silmukan stabiiliutta. PD-säätimen rakenne on

$$u(t) = K_p \left(e(t) + T_d \frac{de(t)}{dt} \right), \tag{2.8}$$

missä T_d kuvaa derinointiosan derivointiaikaa. [2][3]

2.2 PID-säätimen viritys

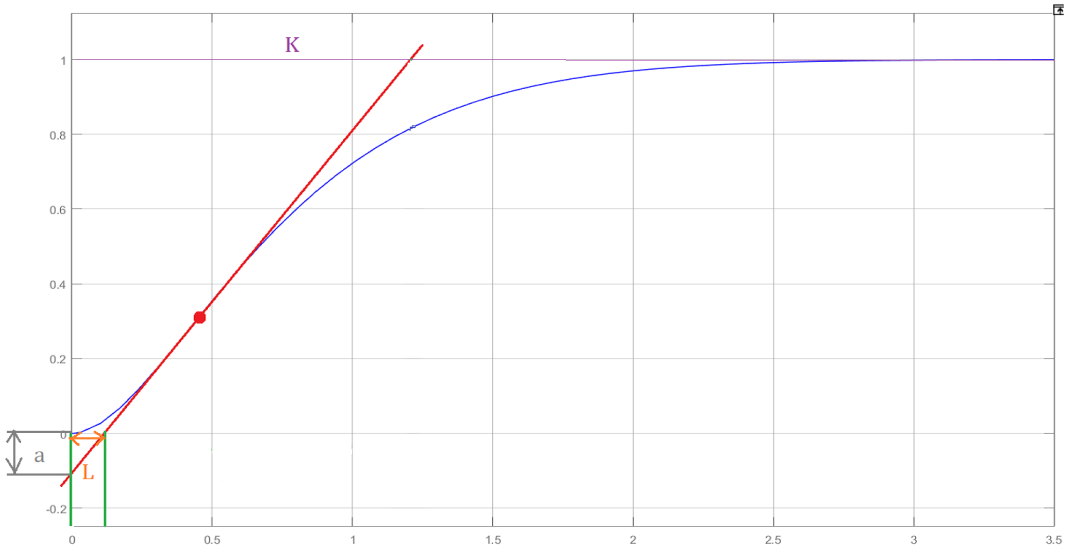
PID-säätimelle löytyy useita eri viritystapoja. Tässä alaluvussa käydään läpi yleisesti PID-säätimen viritykseen liittyviä asioita sekä esimerkkinä Ziegler–Nichols-menetelmää, joka on yleisesti käytetty virityssääntö.

Prosessien monimuotoisuuden ja niiden käyttötarkoituksien vaihtelun takia eivät kaikki säännöt sovi kaikkiin järjestelmiin ja säätöinsinööreiltä vaaditaan kriittisyyttä tämän suhteen. Virityssäännöt voivat olla joko kokeellisia tai malliin perustuvia. Kokeellisuus tarkoittaa sitä, ettei järjestelmän mallia, joka voi olla esimerkiksi siirtofunktio, tarvitse tuntea. Kokeellinen parametrien hakeminen käyttää työkaluinaan esimerkiksi iteroimalla viritys parametreja tai simulaatioita parametrien haussa. Iteroinnilla tarkoitetaan parametrien hakua niitä haarukoimalla, kunnes järjestelmä toimii riittävän tarkasti halutulla tavalla. Iteroimisessa tarvitaan tietoa sekä järjestelmästä että parametrien vaikutuksesta järjestelmään, sillä järjestelmän toiminnan testaaminen voi olla hidasta.

Malliin perustuva viritys tarvitsee matemaattista mallia tai sen approksimaatiota järjestelmästä, jonka avulla viritysparametrit lasketaan. Ziegler–Nichols-viritysmenetelmä, jota tarkastellaan tässä luvussa, on kokeellinen, ja myöhemmin luvussa 3 esiteltävä SIMC-virityssääntö on malliin perustuva. Ziegler–Nichols-menetelmä on valittu tarkasteltavaksi kahdesta eri syystä. Ensimmäinen syy on se, että se on laajasti tunnettu ja käytetty menetelmä, ja toinen syy on se, että Skogestad on käyttänyt tätä apuna kehittäessään omaa SIMC-viritystapaansa, josta kerrotaan tarkemmin luvussa 3.

Ziegler–Nichols-menetelmä on helppo viritysalgoritmi, joka kehitettiin vuonna 1942. Ensimmäinen tapa on käyttää prosessin avoimen silmukan askelvastetta ja toinen prosessin taajuusvastetta. Taulukossa 2.1 on molemmilla tavoilla säätimen parametrien laskentatavat. Ziegler–Nichols-viritysmenetelmä antaa viritysparametrit PID-säätimen (2.3) esitystapaan. Askelvastemetodilla täytyy hakea askelvasteesta parametri a sekä parametri L. Parametrien saamiseksi täytyy askelvastekuvaan piirtää taitospisteeseen tangentti. Taitospisteellä tarkoitetaan pistettä, johon piirretty tangentti on jyrkin mahdollinen käyrälle. Parametri a kuvaa y-akselin ja tangentin leikkauspisteen itseisarvoa. Parametri L on puolestaan x-akselin ja tangentin leikkauspisteen arvo siinä tapauksessa, kun järjestelmään sisääntulon askel on nostettu ajanhetkellä nolla. Kuvasta 2.2 voidaan nähdä graafisesti, mitä parametreilla tarkoitetaan. Tämän jälkeen voi parametrit laskea taulukosta 2.1. [9]

Taajuusvastemetodilla järjestelmä on suljetussa silmukassa. Kyseisellä metodilla ensim-



Kuva 2.2. *Ziegler–Nichols askelvastemetodi*

Taulukko 2.1. *Ziegler-Nichols-metodin parametritaulukot*

Säätimen tyyppi	**Askelvastemetodin viritysparametrit**			**Taajuusvastemetodin viritysparametrit**		
	K_p	T_i	T_d	K_p	T_i	T_d
P	$1/a$			$0.5K_c$		
PI	$0.9/a$	$3L$		$0.4K_c$	$0.8T_c$	
PID	$1.2/a$	$2L$	$L/2$	$0.6K_c$	$0.5T_c$	$0.12T_c$

mäinen tehtävä on asettaa PID-säädin proportionaalimuotoon eli asettaa integrointiaika T_i äärettömäksi ja derivointiaika T_d nollaksi. Tämän jälkeen etsitään kriittinen vahvistus K_c, jolla säätöpiirin askelvaste oskilloi vaimenematta. Tällöin saavutetaan marginaalinen stabiilius. Marginaalisen stabiilisuuden saavuttamisen jälkeen voidaan kokeellisesti mitata vastaavan värähtelyn jaksonaika T_c. Näillä kahdella saadulla kriittisellä arvolla lasketaan säätimen parametrit taulukon 2.1 kaavojen avulla. [9] [3]

Ziegler–Nichols-virityssäännöillä askelvasteesta saadaan nopea, mutta ylitystä tulee paljon. Tämän takia viritysmenetelmä vaatii yleensä parametrien virittämistä vielä jälkeenpäin. Ziegler–Nichols-virityksen oli alun perin tarkoitus antaa hyvä kuormistushäiriövaste. Tämä pitää ottaa huomioon menetelmää käytettäessä. [3]

3 SIMC-VIRITYS

Kuten jo aiemmassa luvussa kerrottiin, PID-säätimen viritys ei ole niin yksinkertaista kuin voisi sen vain kolmesta viritysparametrista ajatella. Tässä luvussa esiteltävä SIMC-viritystapa pohjautuu kahteen virityksen, Ziegler–Nichols-menetelmään ja Internal Model Control -virityssääntöön (IMC). IMC-virityssääntö tunnetaan huonosta häiriövasteesta, mutta antaa hyviä tuloksia asetusarvomuutoksiin. SIMC-viritys on Sigurd Skogestadin esittämä viritysmetodi, joka koostuu kahdesta osasta, järjestelmän mallin approksimaatiosta sekä approksimaation mukaan laskettavista viritysparametreista PID-säätimelle, joka on muodossa (2.4). [1] [7]

3.1 Järjestelmän mallin approksimointi

SIMC-viritystavan ensimmäinen vaihe on approksimoida malli joko ensimmäisen tai toisen kertaluokan viiveelliseeen malliin, jotka ovat

$$g(s) = \frac{k}{\tau_1 s + 1} \exp\left(-\theta s\right) \tag{3.1}$$

sekä

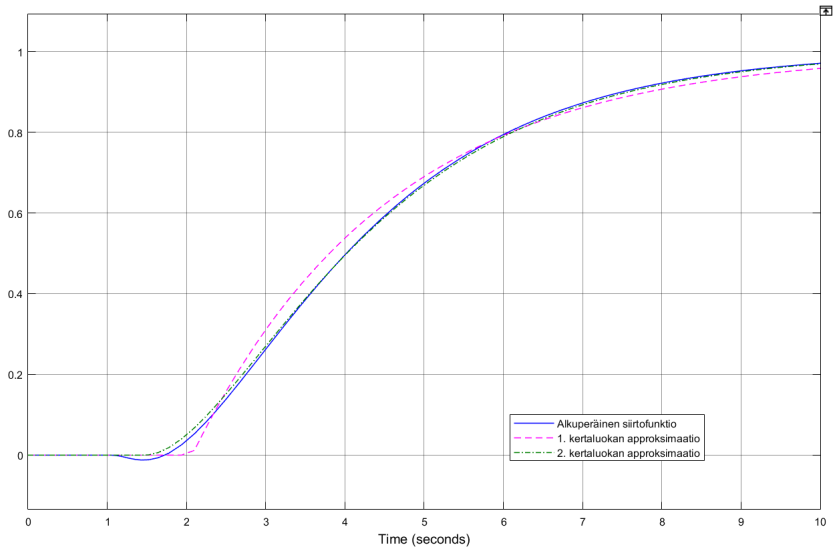
$$g(s) = \frac{k}{\left(\tau_1 s + 1\right)\left(\tau_2 s + 1\right)} \exp\left(-\theta s\right). \tag{3.2}$$

Siirtofunktioissa (3.1) ja (3.2) θ kuvaa viivettä, τ_1 ja τ_2 ovat aikavakioita. Parametrien rajoituksina ovat $k > 0$, $\tau_1 > 0$, $\tau_2 > 0$ sekä $\theta > 0$. Järjestelmän approksimaatiomallia (3.1) käytetään tilanteissa, joissa halutaan käyttää PI-säädintä ja approksimaatiomallia (3.2) silloin, kun halutaan myös derivointiosa mukaan. Tässä luvussa esitetään approksimaatio kolmella eri tavalla; avoimen silmukan vasteella, suljetun silmukan vasteella P-säätimen kanssa ja järjestelmän siirtofunktiosta puolitussäännöllä, joka esitellään alaluvussa 3.1.3. [7]

Kuvassa 3.1 on esitetty siirtofunktiosta

$$P(s) = \frac{\left(-0.3s + 1\right)\left(0.08s + 1\right)}{\left(2s + 1\right)\left(1s + 1\right)\left(0.4s + 1\right)\left(0.2s + 1\right)\left(0.05s + 1\right)^3} \tag{3.3}$$

tehtyjä approksimaatioita puolitussäännöillä. Kuten kuvasta 3.1 voidaan huomata niin toisen kertaluvun approksimaatiolla saadaan tarkempi approksimaatio kuin ensimmäisen kertaluvun approksimaatiolla. Approksimaatiotavan valinta riippuu siitä, mitä tietoa jär-



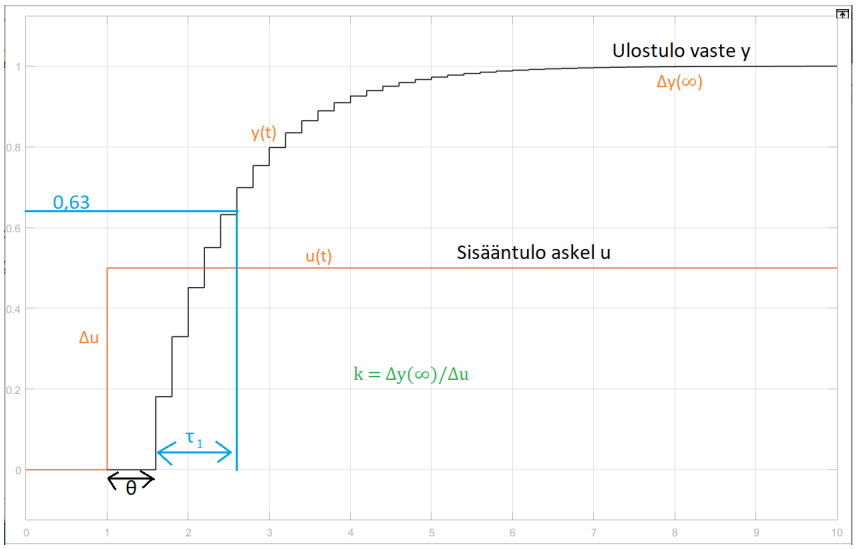
Kuva 3.1. *Seitsemännen kertaluokan siirtofunktiosta (3.3) tehtyjen approksimaatioiden askelvasteet*

jestelmästä on saatavilla ja viritetäänkö PI- vai PID-säädintä. Esimerkiksi, jos järjestelmästä on saatavilla siirtofunktio, saadaan lähimpänä alkuperäistä järjestelmämallia oleva approksimaatio siirtofunktion avulla. Järjestelmästä ei kuitenkaan aina voida saada siirtofunktiota, jota voitaisiin approksimoida mallipohjaisesti. Tällöin käydään läpi kokeellisen approksimaation vaihtoehto. Kokeellisessa approksimaatiossa on kaksi vaihtoehtoa; suljettu- ja avoin vaste. Edellä mainittujen esimerkkien lisäksi käydään siirtofunktiosta tehdyssä approksimaatiossa puolitussäännön lisäksi läpi muutamia poikkeustapauksia.

3.1.1 Avoimen silmukan vaste

Avoimen silmukan vasteesta saa vain PI-säätimelle viritysparametrit, sillä prosessimalli, joka sillä muodostetaan on ensimmäistä kertaluokkaa. Saadulla mallilla parametrit voidaan laskea alaluvussa 3.2 annetun laskukaavan (3.12) avulla. [7]

Haluttu ensimmäisen kertaluvun siirtofunktion kaava (3.1) esiteltiin jo edellisessä kappaleessa. Kuvasta 3.2 voidaan havaita, miten tarvittavat termit saadaan vasteesta. Kuvassa 3.2 on yksikköaskelvaste sellaiselle tapaukselle, jossa alkuarvot ovat nollia. Viive θ saadaan ottamalla erotus asetusarvon muutoksesta ja ulostulon muutoksen alkamisesta. Aikavakio τ_1 puolestaan saadaan laskemalla, kuinka kauan ulostulolla kestää saavuttaa 63% ulostulon loppuarvosta viiveen jälkeen. Viimeisenä DC-vahvistus k saadaan laskemalla vakaan tilan vahvistus jakamalla ulostulon loppuarvo ohjausarvon muutoksella. Näillä tiedoilla saadaan muodostettua siirtofunktio, jonka avulla voidaan laskea SIMC-viritysparametrit. [7]



Kuva 3.2. *Avoimen vasteen approksimaation termien tunnistus*

3.1.2 Suljetun silmukan vaste

Kaikissa tapauksissa ei kuitenkaan saada avoimen systeemin vastetta järjestelmästä vaan vain suljetun systeemin vaste ja tämän käsitteleminen voi olla myös tehokkaampaa. Toisin kuin Ziegler-Nichols-menetelmässä, missä etsitään P-säätimelle vahvistus, jolla systeemi saadaan oskilloimaan. Skogestad esitti Shamsuzzohan kanssa uudemmman tavan muodostaa approksimaatio suljetusta vasteesta, mikä kuitenkin pohjaa Ziegler-Nichols-menetelmään. Tällä uudella tavalla saadaan parempia approksimaatioita laajemmin erilaisille prosesseille kuin alkuperäisellä menetelmällä. Esimerkiksi se toimii hyvin myös viiveen dominoivilla prosesseilla. Ainoa rajoite kyseisen menetelmän käyttämiselle on se, että P-säätimen sisältävän järjestelmän on annettava asetusarvon ylitystä. [7] [10]

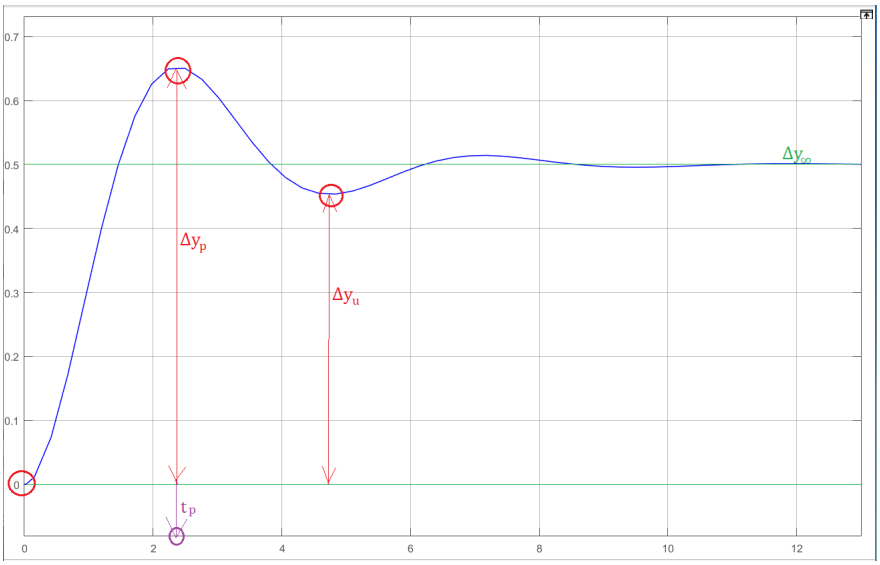
Approksimaatio lähtee siitä, että etsitään virittämällä P-säädintä piste, jolla asetusarvon ylitys (D) on noin 30%. P-säätimen vahvistuksen K_{c0} pitäisi tässä tilanteessa olla noin puolet Ziegler–Nichols-menetelmän vahvistuksesta. Ennen asetusarvomuutosta järjestelmän pitää olla vakaassa tilassa. Asetusarvomuutoksen jälkeen tarvitaan vasteesta parametreja, mitkä näkyvät kuvassa 3.3. Säätimessä käytetty vahvistus K_{c0}, asetusarvomuutos Δy_s, aika vasteen ensimmäiseen huippuun t_p, vasteen ulostulon maksimi arvo Δy_p ja ulostulon ensimmäisen värähdyksen minimiarvo Δy_u. [7]

Maksimi- ja minimivärähdysarvoa voidaan käyttää määrittelemään ulostulon vakaa arvo kaavalla

$$\Delta y_\infty = 0,45(\Delta y_p + \Delta y_u). \tag{3.4}$$

Jos kuitenkin on aikaa odottaa vasteen tasoittumista, voi Δy_∞ katsoa myös suoraa vasteesta. Tällöin arvoa Δy_u ei tarvitse ollenkaan. [7]

Näillä arvoilla voidaan määrittää tarkasti jo aiemmin arvioitu vasteen prosentuaalinen yli-



Kuva 3.3. *Esimerkki suljetun silmukan termien tunnistamisesta*

tys $D = \frac{\Delta y_p - \Delta y_\infty}{\Delta y_\infty}$ ja vakaan tilan prosentuaalinen virhe $B = \left|\frac{\Delta y_s - \Delta y_\infty}{\Delta y_\infty}\right|$. Tämän jälkeen pitää vielä laskea parametri h, jotta saadaan laskettua itse ensimmäisen kertaluokan parametrit k, θ ja τ_1. Kaava h:n laskemiseen on

$$h = \frac{2\left(1,152D^2 - 1,607D + 1\right)}{B}, \tag{3.5}$$

jonka jälkeen voidaan laskea

$$\begin{aligned} k &= 1/(K_{c0}B) \\ \theta &= t_p\left(0,309 + 0,209e^{-0,61h}\right), \\ \tau_1 &= h\theta \end{aligned} \tag{3.6}$$

millä puolestaan saadaan muodostettua haluttu ensimmäisen kertaluokan siirtofunktio (3.1). [11]

3.1.3 Siirtofunktion approksimaatio puolitussäännöllä

Jos prosessista on saatavilla siirtofunktio, silloin on mahdollista tehdä approksimaatio niin kutsutulla puolitussäännöllä. Prosessin siirtofunktio tulee approksimaatiota tehdessä olla muodossa, missä siirtofunktio sisältää napoja, nollia, viiveen ja vahvistuksen. Tämän voi ilmoittaa siirtofunktiolla

$$g_0(s) = ke^{-\theta_0}\frac{\Pi_{j=1}^{m}(T_{j0}s + 1)}{\Pi_{j=1}^{n}(\tau_{j0}s + 1)}, \tag{3.7}$$

missä τ_{j0} ovat napojen aikavakioita, T_{j0} ovat nollien aikavakioita, θ_0 kuvaa viivettä siirtofunktiossa ja k vahvistusta. Parametrien myös täytyy täyttää rajoitukset $T_{j0} \neq 0$, $\tau_{j0} < 0$ sekä $\theta < 0$. [1] [7]

Approksimaatio suoritetaan kolmessa vaiheessa, jotka ovat napojen aikavakioiden τ_{j0} approksimointi, negatiivisten nollien aikavakioiden T_{j0} approksimointi sekä positiivisten nollien aikavakioiden T_{j0} approksimointi. Vahvistus pysyy approksimaation jälkeen samana kuin ennen approksimaatiotakin.

Ensimmäisessä vaiheessa approksimoidaan napojen aikavakiot. Hitain napa luo pohjan τ_1. Toiseksi hitain napa luo puolestaan pohjan τ_2, jos kyseessä on toisen kertaluokan approksimaatio. Tämän jälkeen seuraavaksi hitain napa jaetaan puoliksi joko τ_1:lle ja viiveelle θ tai τ_2:lle ja viiveelle, mistä approksimaation nimi "puolitussääntö"myös tulee. Loput napojen aikavakiot lisätään viiveeseen.

Yhtälö ensimmäiselle kertaluokalle, eli $\tau_2 = 0$

$$\tau_1 = \tau_{10} + \frac{\tau_{20}}{2}; \theta = \theta_0 + \frac{\tau_{20}}{2} + \sum_{i \geq 3} \tau_{i0} \tag{3.8}$$

ja toiselle kertaluokalle

$$\tau_1 = \tau_{10}; \tau_2 = \tau_{20} + \frac{\tau_{30}}{2}; \theta = \theta_0 + \frac{\tau_{30}}{2} + \sum_{i \geq 4} \tau_{i0}. \tag{3.9}$$

Seuraavassa vaiheessa lisätään saatuun viiveeseen, kuvataan tässä θ_1, negatiiviset nollien aikavakiot

$$\theta = \theta_1 + \sum_j |T_{j0}|. \tag{3.10}$$

Viimeisessä vaiheessa approksimoidaan jäljelle jääneet positiivisten nollien aikavakiot. Jos nollan aikavakio on suurempi kuin tähän mennessä lasketun viiveen puolikas $T_{j0} > \frac{\theta_1}{2}$, se vähennetään suurimmasta navan aikavakiosta τ_{10}. Jos kuitenkin $T_{j0} < \frac{\theta_1}{2}$, silloin se vähennetään saadusta viiveestä

$$\theta = \theta_1 - \sum_j T_{j0}. \tag{3.11}$$

Tämän jälkeen approksimaatio on valmis ja voidaan siirtyä viritysparametrien laskemiseen. Vaikka prosessimalli ei täsmäisi suoraan edellä mainittuun tilanteeseen 3.12, muutamalle poikkeustapaukselle on olemassa omat virityssääntönsä, jotka käydään läpi luvussa 3.3.

3.2 Viritysparametrien laskeminen

Sen jälkeen kun ollaan tehty approksimaatio, tilanteesta riippuen ensimmäiseen tai toiseen kertaluokkaan, ollaan valmiita siirtymään toiseen vaiheeseen eli itse viritykseen.

Virityssäännöllä yleisesti

$$\begin{aligned} K_c &= \frac{1}{k}\frac{\tau_1}{\tau_c + \theta} \\ \tau_I &= min\left\{\tau_1, 4(\tau_c + \theta)\right\}, \\ \tau_D &= \tau_2 \end{aligned} \tag{3.12}$$

mistä saadaan vahvistus K_c, τ_I ja tarvittaessa, jos käytetään toisen kertaluokan approksimaatiota aikavakio derivointiosalle τ_D. Ainoaksi itse viritettäväksi parametriksi jää vakio τ_c, joka voi olla mikä tahansa välillä $-\theta < \tau_c < \infty$.

Eräällä viritystavalla, jolla saadaan nopea vaste ja hyvä robustisuus, asetetaan muuttuja τ_c yhtä suureksi approksimaation viiveen kanssa. Tällä laskemalla, virityskaavoiksi saadaan

$$\begin{aligned} K_c &= \frac{0,5}{k}\frac{\tau_1}{\theta} \\ \tau_I &= min\left\{\tau_1, 8\theta\right\}\cdot \\ \tau_D &= \tau_2 \end{aligned} \tag{3.13}$$

Kaavan (3.13) helppouden ja yleisesti hyvän vasteen vuoksi käytetään tätä $\tau_c = \theta$ ohjelman oletusarvona myös Matlab-työkalussa, mistä lisää luvussa 4.

3.3 Erikoistapaukset

Tässä luvussa käydään läpi muutamia erikoistapauksia, joille viritys tulee olemaan erilainen kuin perustapauksessa, jossa mallin approksimaatiossa on vain napoja ja nollia. Jos järjestelmällä on siirtofunktiomalli alaluvussa 3.1.3 esitetyn yleisen tapauksen sekä tässä luvussa esiteltävien erikoistapausten ulkopuolella, ei järjestelmämallille voida laskea ilman muutoksia viritysparametreja kyseisellä viritysmenetelmällä. Vaihtoehtoina kyseisessä tapauksessa olisivat joko järjestelmämallin muokkaus ennen approksimaatiota tai avoimen- tai suljetun vasteen approksimaation käyttäminen, mistä enemmän alaluvuissa 3.1.1 sekä 3.1.2.

3.3.1 Puhdas viivemalli

Puhtaan viiveen ja vahvistuksen sarjaankytkenä on muotoa

$$g(s) = ke^{-\theta s}. \tag{3.14}$$

Tälle prosessille ei voi käyttää ainoastaan P-säädintä, sillä se tuottaisi parhaassakin tapauksessa 50% poikkeaman askelvasteessa asetusarvoon. Tämän vuoksi säätimeen tarvitaan integroiva osa, vaikka prosessimalli on nollatta kertaluokkaa. [1]

Koska $\tau_1 = 0$ puhtaassa viivemallissa, saataisiin ilman muokkauksia $K_c = 0$ ja $\tau_I = 0$, joka tuottaisi PI-säätimen algoritmiksi puhtaan 0. Tästä syystä lähdetään käyttämään integroivaa säädintä

$$C(s) = \frac{K_I}{s}, \tag{3.15}$$

missä $K_I = \frac{K_c}{\tau_I} = \frac{1}{k}\frac{\tau_1}{\tau_c+\theta}\frac{1}{\tau_1}$. Tällöin τ_1 supistuu pois ja jos käytetään $\tau_c = \theta$ saadaan säätimelle muoto

$$C(s) = \frac{0,5}{k\theta}\frac{1}{s}, \tag{3.16}$$

joka on jo edellä mainittu puhdas integroiva säädin. [7][1]

Edellä kuvatun säätimen saa approksimoitua normaalilla PI-säätimellä parametrien K_c valinnalla $K_c = \frac{0,5}{k}\frac{\tau_I}{\theta}$ ja τ_I pieneksi esimerkiksi $\tau < 0,1\theta$. [1]

3.3.2 Integroiva prosessi

Otetaan käyttöön $k' = \frac{k}{\tau_1}$ ja mallinnetaan tämän avulla integroiva prosessi

$$g(s) = k'\frac{e^{-\theta s}}{s}. \tag{3.17}$$

Tässä erikoistapauksessa $\tau_1 \to \infty$, jolloin 3.12 kaavoilla saadaan suoraan viritysparametrit

$$\begin{aligned} K_c &= \frac{1}{k'}\frac{1}{\tau_c+\theta} \\ \tau_I &= 4(\tau_c+\theta) \end{aligned}. \tag{3.18}$$

PID-säätimen viritys ei ole mahdollista tälle erikoistapaukselle, sillä prosessi on ensimmäistä kertaluokkaa. [1]

3.3.3 Tuplaintegroiva prosessi

Tuplaintegroiva prosessi

$$g(s) = k''\frac{e^{-\theta s}}{s^2}, \tag{3.19}$$

missä $k'' = k'/\tau_2$. Tuplaintegroivalle prosessille PID-säädin saa parametrit

$$\begin{aligned} Kc &= \frac{1}{k''}\frac{1}{4(\tau_c+\theta)^2} \\ \tau_I &= 4(\tau_c+\theta) \\ \tau_D &= 4(\tau_c+\theta) \end{aligned}. \tag{3.20}$$

Tuplaintegroiva prosessin PID-säätimen τ_I jäisi nollaksi ilman lisäviritystä. Tällöin säädin ei kuitenkaan pystyisi palautumaan ulkoisista häiriöistä aiheutuvista poikkeamista, jonka

vuoksi τ_I viritetään samoin kuin integroivan prosessin tapauksessa 3.22. [1]

3.3.4 Integroiva prosessi viiveen ja navan kanssa

Viimeisenä erikoistapauksena tarkastetaan integroivaa prosessia, missä on viive ja napa eli

$$g(s) = k' \frac{e^{-\theta s}}{s(\tau_2 s + 1)}. \tag{3.21}$$

Tämä viritetään muuten samoin kuin integroiva prosessi (3.17), mutta lisätään toisesta kertaluokasta aiheutuva τ_D termi, joka viritetään (3.12) mukaan. Tästä saadaan viritys-säännöt [1]

$$\begin{aligned} K_c &= \frac{1}{k'} \frac{1}{\tau_c + \theta} \\ \tau_I &= 4(\tau_c + \theta) \\ \tau_D &= \tau_2 \end{aligned} \tag{3.22}$$

4 SIMC-VIRITYSTYÖKALUN MATLAB-OHJELMA

Tässä työssä on tehty työkalu SIMC-viritykselle Matlab ohjelmistolla. SIMC-virityksen teoriaa tarkasteltiin luvussa 3. Matlab-työkalun käyttäminen toimii Matlab-ohjelmiston niin oletuskirjastoilla eikä tarvitse CST-lisäkirjastoa ("Control System Toolbox"). Matlab-viritystyökalun lähdekoodi on liitteessä A.

Työkalun käyttö on pyritty tekemään mahdollisimman käyttäjäystävälliseksi. Käytettävä kieli on suomi ja ohjelma kertoo joka vaiheessa, mitä käyttäjän pitää tehdä ja miten, esimerkiksi, missä muodossa työkalu vaatii käyttäjän syötteet. SIMC-työkalulla ei ole erillistä käyttöliittymää vaan käyttö tapahtuu Matlabin komentokehotteen kautta ("Command Window"). Komennolla "help viritystyokalu"saadaan lyhyet ohjeet ohjelman käyttöön.

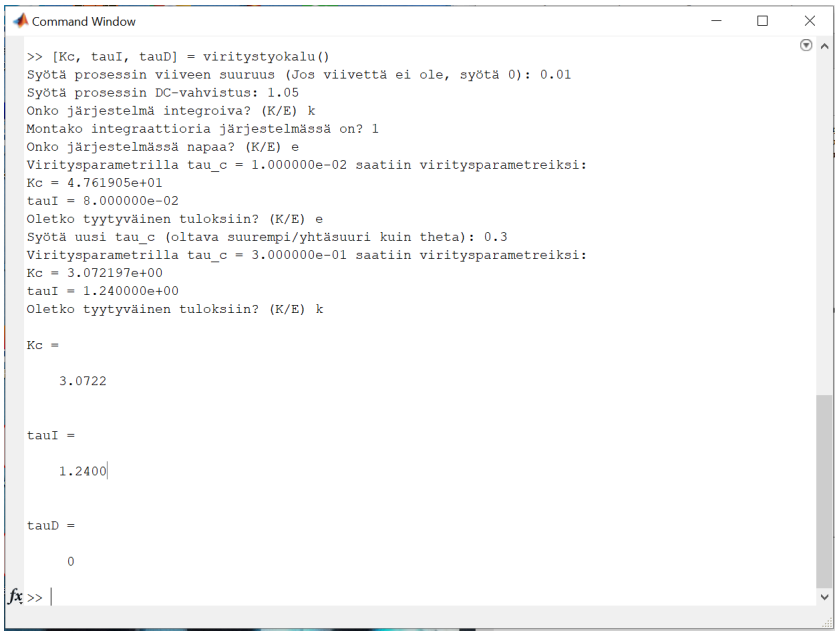
Työkalun käyttö alkaa komennolla "[Kc, tauI, tauD] = viritystyokalu", jossa K_c, tau_I ja tau_D kuvaavat käyttäjän nimitettäviä parametreja. Työkalun käytön jälkeen viritysparametrit tallentuvat käyttäjän ilmoittamiin kyseisiin muuttujiin.

Kuten kuvasta 4.1 ja 4.2 voidaan nähdä, työkalu kysyy käyttäjältä ensin järjestelmän viiveen ja vahvistuksen suuruudet. Ensimmäinen esimerkkitapaus, mikä käydään työkalulla läpi, on integroiva järjestelmä. Approksimaatio toimii kahteen integraattoriin asti, mutta jos integraattoreita ilmoitetaan olevan enemmän, ohjelma heittää virhetulostuksen siitä, ettei kyseiselle järjestelmälle voida laskea approksimaatiota. Integroivan järjestelmän parametrien lasku voidaan nähdä kuvasta 4.1.

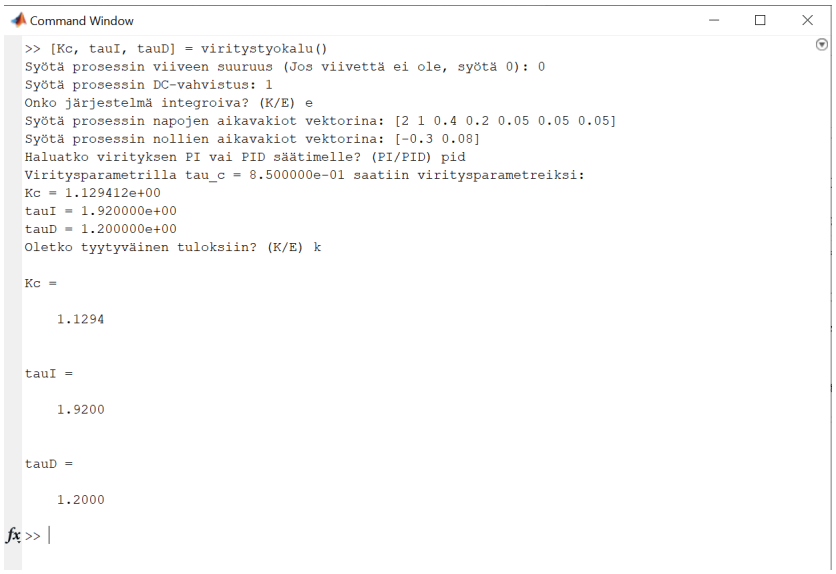
Puolitussäännöllä approksimaation laskun yleisen säännön käyttö puolestaan näkyy kuvassa 4.2. Kuten kuvastakin voidaan huomata, jos järjestelmässä ei ole integraattoreita, saa järjestelmään syöttää aikavakiot vektoreina. Jos haluaisi laskea parametrit puhtaalle viivemallille, voisi vektorit jättää syöttämättä tai tyhjiksi, milloin työkalu laskisi tuon kyseisen erikoistapauksen parametrit.

Ohjelmassa on otettu huomioon järjestelmään liittyviä rajoituksia. Esimerkiksi aikavakioiden syöttämisen jälkeen ohjelma tarkistaa, että napojen aikavakiot täyttävät ehdon $\tau_{j0} > 0$ ja $T_{j0} \neq 0$. Jos joku approksimaation järjestelmään liittyvä esiehto ei täyty, mukaan lukien järjestelmän vahvasti aitouteen liittyvä ehto, ohjelma lopettaa toiminnan ja tulostaa kyseisen virheilmoituksen.

Tulosten saamisen jälkeen käyttäjä saa nähtäville sen hetkiset työkalun ehdottamat PID-säätimen parametrit. Kuitenkin, jos käyttäjä ei ole tyytyväinen näihin, voi hän muuttaa tau_c parametria, miten haluaa, kunhan se on suurempi kuin θ.



Kuva 4.1. *Esimerkkikäyttö integroivan järjestelmän tapauksessa*



Kuva 4.2. *Esimerkkikäyttö järjestelmässä, missä on napoja ja nollia*

Ohjelman rakenteessa on yritetty pyrkiä järkevään ohjelmarakenteeseen ja erottaa mahdollisia osatapahtumia omiksi funktioiksi, kuten eri tapauksien laskemiset ja ohjelmassa toistuvat käyttäjäkyselyt.

5 YHTEENVETO

PID-säätimen viritys voi olla hankalaa. Säätimet ovat usein huonosti viritettyjä ja näin ollen aiheuttavat järjestelmässä esimerkiksi epätoivottua ylitystä askelvasteeseen tai heikkoa häiriövastetta. Järjestelmä tulee siis viritettäessä tuntea hyvin ja tiedettävä, minkälaista häiriökäyttäytymistä se sallii. Työssä ollaan myös käyty läpi PID-säätimen rakennetta, PID-säätimen viritykseen liittyviä asioita sekä Ziegler–Nichols-viritysmetodia PID-säätimen esimerkkiviritystavasta.

Tässä työssä on esitelty perusteellisesti yksi viritysmetodi, SIMC, jonka hyviä puolia ovat hyvä häiriövaste sekä toimii hyvin laajalle joukolle erilaisia järjestelmiä. Myös helppo malliapproksimaatio ja yksinkertaiset säätimen parametrien laskukaavat ovat Skogestadin esittämiä ominaisuuksia työssä tutkittavalle viritysmetodille.

Helpottaakseen viritystä, teoriakatsauksen lisäksi työssä on luotu työkalu, joka laskee automaattisesti järjestelmälle SIMC-viritysmetodin mukaiset viritysparametrit PID-säätimelle. Viritystyökalu toimii antamalla sille järjestelmän siirtofunktiomallin parametrit ja se laskee ensin $\tau_c = \theta$ parametrilla viritysparametrit. Tämän jälkeen jos ei ole tuloksiin tyytyväinen, voi τ_c:n arvoa muuttaa haluamallaan tavalla, kunhan se on yhtäsuuri tai suurempi kuin θ.

LÄHDELUETTELO

[1] S. Skogestad. Probably the best simple PID tuning rules in the world. *Journal of Process Control*, 2001.

[2] A. Visioli. *Practical PID Control*. English. London: Springer, 2006.

[3] K. J. Åström ja T. Hägglund. *Advanced PID control*. English. Research Triangle Park, NC: ISA, 2006.

[4] K. H. Ang, G. Chong ja Y. Li. PID control system analysis, design, and technology. English. *IEEE Transactions on Control Systems Technology* 13.4, 2005, 559–576.

[5] A. O'Dwyer. An Overview of Tuning Rules for the PI and PID Continuous-Time Control of Time-Delayed Single-Input, Single-Output (SISO) Processes. eng. *PID Control in the Third Millennium: Lessons Learned and New Approaches*. 2012. painos. Advances in Industrial Control. London: Springer London, 2012, 3, 44. ISBN: 9781447124245.

[6] A. O'Dwyer ja W. S. (Firm). *Handbook of PI and PID controller tuning rules*. English. 3rd. London;Singapore; Imperial College Press, 2009.

[7] S. Skogestad ja C. Grimholt. The SIMC Method for Smooth PID Controller Tuning. eng. *PID Control in the Third Millennium: Lessons Learned and New Approaches*. 2012. painos. Advances in Industrial Control. London: Springer London, 2012, 147, 175.

[8] I. Tejado, B. Vinagre, J. Traver, J. Prieto-Arranz ja C. Nuevo-Gallardo. Back to Basics: Meaning of the Parameters of Fractional Order PID Controllers. English. *Mathematics* 7.6, 2019, 530.

[9] K. Ogata. *Modern control engineering*. eng. 5th ed. Upper Saddle River, New Jersey: Pearson, 2010. ISBN: 978-0-13-713337-6.

[10] M. Shamsuzzoha ja S. Skogestad. The setpoint overshoot method: A simple and fast closed-loop approach for PID tuning. eng. *Journal of Process Control* 20.10, 2010, 1220, 1234. ISSN: 0959-1524.

[11] R. Vilanova ja A. Visioli. *PID Control in the Third Millennium: Lessons Learned and New Approaches*. English. 1. Aufl. Springer Verlag London Limited, 2012.

A LÄHDEKOODI

```
function [Kc, tauI, tauD] = viritystyokalu()
% VIRITYSTYOKALU Laskee PI/PID- säätimelle viritysparametreja
% SIMC-viritysmetodin mukaan
%
%   Parametrit annetaan säätimen kaskadinotaatiolle
%
%   Ohjelma antaa ohjeita käytön aikana
%   Viritystyökalulle ei tarvitse antaa parametreja
%   Palauttaa kolme parametria;
%       Vahvistuksen Kc
%       Integraattorin aikavakion Ti
%       Derivaattorin aikavakion Td
%
%   Oletuksena viritysparametri tau_c = theta. Voidaan vaihtaa,
%   jos tulokset eivät ole hyviä oletusasetuksella

    % Viritysparametrit mitä lasketaan
    Kc = 0;
    tauI = 0;
    tauD = 0;

    % Tarkistusarvot
    tarkistus2 = false;

    kyssari = 'Syötä prosessin viiveen suuruus (Jos viivettä ei ole,
                                                syötä 0): ';
    theta0 = input(kyssari);

    kyssari = 'Syötä prosessin DC-vahvistus: ';
    k = input(kyssari);

    kyssari = 'Onko järjestelmä integroiva? (K/E) ';
    int = input(kyssari, 's');
    int = upper(int);
```

```
if int == 'K'
    kyssari = 'Montako integraattioria järjestelmässä on? ';
    int_lkm = input(kyssari);

    if int_lkm == 1
        kyssari = 'Onko järjestelmässä napaa? (K/E) ';
        intnapa = input(kyssari, 's');
        intnapa = upper(intnapa);

        if intnapa == 'K'
            tauc = theta0;

            while tarkistus2 == false
                kyssari = 'Syötä navan aikavakio: ';
                tau2 = input(kyssari);

                if tau2 <= 0
                   error('Siirtofunktio epästabiili, ei voida laskea
                            viritysparametreja');
                end
                [Kc, tauI, tauD] = intnapasimc(theta0, tau2, k, tauc);
                fprintf('Viritysparametrilla tau_c = %d saatiin
                        viritysparametreiksi: \n', tauc);
                fprintf('Kc = %d \n', Kc);
                fprintf('tauI = %d \n', tauI);
                fprintf('tauD = %d \n', tauD);

                [tauc, tarkistus2] = tarkistus(tauc, theta0);
            end
            return;
        else
            tauc = theta0;

            while tarkistus2 == false
                [Kc, tauI] = intsimc(theta0, k, tauc);
                fprintf('Viritysparametrilla tau_c = %d saatiin
                        viritysparametreiksi: \n', tauc);
                fprintf('Kc = %d \n', Kc);
                fprintf('tauI = %d \n', tauI);

                [tauc, tarkistus2] = tarkistus(tauc, theta0);
```

```
            end
            return;
        end

    elseif int_lkm == 2
        tauc = theta0;

        while tarkistus2 == false
            [Kc, tauI, tauD] = doupintsimc(theta0, k, tauc);
            fprintf('Viritysparametrilla tau_c = %d saatiin
                    viritysparametreiksi: \n', tauc);
            fprintf('Kc = %d \n', Kc);
            fprintf('tauI = %d \n', tauI);
            fprintf('tauD = %d \n', tauD);

            [tauc, tarkistus2] = tarkistus(tauc, theta0);
        end
        return;
    else
        disp('Valitettavasti tälle mallille ei voida laskea
                viritysparametreja');
        return;
    end
end

% Ohjelma kysyy käyttäjältä prosessin navat ja nollat
kyssari = 'Syötä prosessin napojen aikavakiot vektorina: ';
navat = input(kyssari);

for aikavakio = navat
    if aikavakio <= 0
        error('Siirtofunktio epästabiili, viritysparametreja ei voida
                laskea');
    end
end

kyssari = 'Syötä prosessin nollien aikavakiot vektorina: ';
nollat = input(kyssari);

tarkistus1 = false;
tarkistus2 = false;
```

```
for aikavakio = nollat
    if aikavakio == 0
        error('Nollan aikavakio ei voi olla arvo nolla');
    end
end

% Tarkastus sille, onko kyseessä aito tai vahvasti aito siirtofunktio
% (onko napoja enemmän
% tai yhtäpaljon kuin nollia)
if length(navat) < length(nollat)
    error('Siirtofunktio on epäaito, viritysparametrejä ei voida laskea');
end

% Järjestää vektorit suurimmasta pienimpään
navat = sort(navat, 'descend');
nollat = sort(nollat, 'descend');

% Sievennys!
% Etsii yhteiset tekijät vectoreista
C = intersect(navat, nollat);

% Poistetaan nämä tekijät
for sievennettava = C
    navat(navat == sievennettava) = [];
    nollat(nollat == sievennettava) = [];
end
% Navat ja nollat ovat nyt sievennetty

% Onko puhdas viive eli onko napoja ja nollia ollenkaan
if isempty(nollat) && isempty(navat)
     [Kc, tauI] = cleardelay(theta0, k);
     fprintf('SIMC-virityksell viritysparametreiksi saatiin: ');
     fprintf('Kc = %d \n', Kc);
     fprintf('tauI = %d \n', tauI);
     return;
end
while tarkistus1 == false
    kyssari = 'Haluatko virityksen PI vai PID säätimelle? (PI/PID) ';
    saadin = input(kyssari, 's');
    saadin = upper(saadin);

    if strcmp('PI', saadin) == 1
```

```
    % PI viritys
    % Approksimaatio ensimmäiseen kertaluokkaan
    [tau1, theta] = piapp(navat, nollat, theta0);

    % Valitaan oletusarvo viritysparametrille tauc
    tauc = theta;

    while tarkistus2 == false
        Kc = 1/k*tau1/(tauc+theta);
        tauI = min(tau1, 4*(tauc+theta));
        tauD = 0;

        fprintf('Viritysparametrilla tau_c = %d saatiin
                viritysparametreiksi: \n', tauc);
        fprintf('Kc = %d \n', Kc);
        fprintf('tauI = %d \n', tauI);

        [tauc, tarkistus2] = tarkistus(tauc, theta0);
    end
    tarkistus1 = true;

elseif strcmp('PID', saadin) == 1
    % PID viritys
    [tau1, tau2, theta] = pidapp(navat, nollat, theta0);
    tauc = theta;

    while tarkistus2 == false
        Kc = 1/k*tau1/(tauc+theta);
        tauI = min(tau1, 4*(tauc+theta));
        tauD = tau2;

        fprintf('Viritysparametrilla tau_c = %d saatiin
                viritysparametreiksi: \n', tauc);
        fprintf('Kc = %d \n', Kc);
        fprintf('tauI = %d \n', tauI);
        fprintf('tauD = %d \n', tauD);

        [tauc, tarkistus2] = tarkistus(tauc, theta0);
    end
    tarkistus1 = true;

else
```

```
            disp('Syöttäisitkö jommankumman tarjotuista vaihtoehdoista?
                    (PI/PID) ');
        end
    end
end

function [tauc, tarkistus1] = tarkistus(tauc, theta)

    kyssari = ('Oletko tyytyväinen tuloksiin? (K/E) ');
    vast = input(kyssari, 's');
    vast = upper(vast);
    if vast == 'K'
        tarkistus1 = true;

    elseif vast == 'E'
        kyssari = ('Syötä uusi tau_c (oltava suurempi/yhtäsuuri kuin
                    theta): ');
        tauc_new = input(kyssari);
        while tauc_new < theta
            kyssari = ('Syötä uusi tau_c (oltava suurempi/yhtäsuuri kuin
                            theta): ');
            tauc_new = input(kyssari);
        end
        tauc = tauc_new;
        tarkistus1 = false;
    end

end


function [tau1, theta] = piapp(navat, nollat, theta0)

    for i = 1:length(navat)
        % mitä navoille tehdään
        if i == 1
            tau1 = navat(i);
            theta = theta0;
        elseif i == 2
            tau1 = tau1 + navat(i)/2;
            theta = theta + navat(i)/2;
        else
            theta = theta + navat(i);
```

```
        end
    end

    for i = 1:length(nollat)
        % mitä nollille tehdään
        if nollat(i) < 0
            % negatiiviset nollat
            theta = theta + abs(nollat(i));

        elseif nollat(i) > 0
            % positiiviset nollat

            if nollat(i) < theta/2
                tau1 = tau1 - nollat(i);
            else
                theta = theta - nollat(i);
            end

        else
            error('Nollat eivät voi olla arvoa nolla, viritysparametreja
                    ei voida laskea');

        end
    end

end

function [tau1, tau2, theta] = pidapp(navat, nollat, theta0)
    for i = 1:length(navat)
        % mitä navoille tehdään
        if i == 1
            tau1 = navat(i);
            theta = theta0;
        elseif i == 2
            tau2 = navat(i);
        elseif i == 3
            tau2 = tau2 + navat(i)/2;
            theta = theta + navat(i)/2;
        else
            theta = theta + navat(i);
        end
    end
```

```
    for i = 1:length(nollat)
        % mitä nollille tehdään
        if nollat(i) < 0
            % negatiiviset nollat
            theta = theta + abs(nollat(i));

        elseif nollat(i) > 0
            % positiiviset nollat

            if nollat(i) < theta/2
                tau1 = tau1 - nollat(i);
            else
                theta = theta - nollat(i);
            end

        else
            error('Nollat eivät voi olla arvoa nolla, viritysparametreja
                    ei voida laskea');
        end
    end
end

function[Kc, tauI] = cleardelay(theta, k)
    if theta == 0
        error('Valitettavasti tälle yhdistelmälle ei voida laskea viritysparametreja')
    end
    tauI = 0.08*theta;
    Kc = 0.5/k*tauI/theta;
end

function [Kc, tauI] = intsimc(theta, k, tauc)
    Kc = 1/k*1/(tauc + theta);
    tauI = 4*(tauc + theta);
end

function [Kc, tauI, tauD] = doupintsimc(theta, k, tauc)
    Kc = 1/k*1/(4*(tauc + theta)^2);
    tauI = 4*(tauc + theta);
    tauD = 4*(tauc + theta);
end
```

```
function [Kc, tauI, tauD] = intnapasimc(theta, k, tau2, tauc)
    Kc = 1/k*1/(tauc + theta);
    tauI = 4*(tauc + theta);
    tauD = tau2;
end
```