Tampereen ammattikorkeakoulu

Laboratoriotyön toteutus PID-säätimen ohjelmoinnista

Mikko Hulkkonen

OPINNÄYTETYÖ
Toukokuu 2022
Sähkö- ja automaatiotekniikan tutkinto-ohjelma
Automaatiotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Sähkö- ja automaatiotekniikan tutkinto-ohjelma
Automaatiotekniikka

HULKKONEN, MIKKO:
Laboratoriotyön toteutus PID-säätimen ohjelmoinnista

Opinnäytetyö 59 sivua, joista liitteitä 9 sivua
Toukokuu 2022

Opinnäytetyönä tuotettiin Tampereen ammattikorkeakouluun laboratoriotyö PID-säätimen ohjelmoinnista sähkö- ja automaatiotekniikan opetuskäyttöön. Aihe valittiin, koska kyseisestä aiheesta ei ollut aiemmin tehty laboratoriotyötä. Tavoitteena oli, että opiskelija oppii PID-säätimen sisäistä toimintaa ja ohjelmointia laboratoriotyön avulla.

Laboratoriotyössä PID-säädin ohjelmoitiin käyttäen Python-ohjelmointikieltä. Säätimen testaamisessa käytetiin laboratorion analogista prosessisimulaattoria. Tietokoneen ja prosessin rajapintana toimi Measurement Computingin USB-1408FS-Plus -tiedonkeruukortti. Säätimen toimintaa testattiin simuloiduilla askelvastekokeilla ja tuloksia verrattiin Matlab Simulinkillä tehtyyn säätimeen. Tulosten perusteella voidaan todeta, että käytetyllä laitteistolla voidaan toteuttaa toimiva PID-säädin. Tulosten pohjalta luotiin laboratoriotyö, jonka avulla automaatiotekniikan opetuksessa on mahdollisuus opettaa PID-säätimen ohjelmointia käytännössä.

Laboratoriotyö vaatii vielä testaamista käytännössä, jotta nähdään, onnistuuko se tavoitteessaan. Se ei ole sidottu opinnäytetyössä käytettyyn ohjelmointikieleen tai laitteistoon, joten opettaja voi muuttaa järjestelmää tarvittaessa. Laboratoriotyötä voi vielä laajentaa ottamalla mukaan lisätestejä mutta laajennus riippuu laboratoriotyötä hyödyntävän kurssin toteutustavasta.

Asiasanat: säätötekniikka, pid-säädin, ohjelmointi, python

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Electrical and Automation Engineering
Automation Engineering

HULKKONEN, MIKKO:
The Creation of a Laboratory Exercise about Programming of a PID controller

Bachelor's thesis 59 pages, of which appendices 9 pages
April 2022

The subject of this thesis was the creation of a laboratory exercise about programming a PID controller. The purpose was to produce a laboratory exercise for teaching purposes for the Degree Programme in Electrical and Automation Engineering. The objective was that with this laboratory exercise a student would learn how a PID controller works and how to program one.

The Python programming language was used to program the PID controller. An analog process simulator was used to test the PID controller in the laboratory. A USB-1408FS-Plus data logger was used to do the conversions between analog and digital signals. A step response test was conducted to verify that the controller works properly. The step response was then compared to a PID controller made with Matlab Simulink to see how they compare. Based on the results, a working PID controller can be made with Python and the laboratory equipment. A laboratory exercise was produced that can be used to teach the programming of a PID controller in practice.

Laboratory exercise still requires additional testing with real students to see if it works in practice. Fortunately, the exercise is not dependent on the programming language, or the equipment used so the teacher can change the setup if necessary. The plan for the exercise is subject to change over time when areas of improvement are discovered. The laboratory exercise can be expanded with additional tests but that largely depends on the course where the exercise is implemented.

Key words: control system engineering, pid-controller, programming, python

SISÄLLYS

1 JOHDANTO 6
2 PID-SÄÄDIN 8
2.1 Säädin osana säätöpiiriä 9
2.2 Prosessien mallintaminen 10
2.3 PID-säätimen algoritmi 13
2.4 Algoritmin toteutus tietokoneella 14
2.4.1 Suhteellinen termi 17
2.4.2 Integroiva termi 18
2.4.3 Derivoiva termi 21
2.4.4 Vaihtaminen manuaali- ja automaattitilan välillä 24
2.4.5 Säätimen parametrien muuttaminen 26
3 PYTHON-OHJELMOINTIKIELI 27
3.1 Syntaksi 27
3.2 Olio-ohjelmointi 34
4 LABORATORIOTYÖN LAITTEISTO 37
4.1 USB-1408FS-Plus I/O-kortti 37
4.2 Prosessisimulaattori 43
4.3 Laitteiston kytkentä 44
5 SÄÄTIMEN TOTEUTUS PYTHONILLA 46
5.1 Mcculw-paketti 46
5.2 Säätimen koodi 47
5.3 Säätimen testaaminen 51
6 LABORATORIOTYÖN TOTEUTUSSUUNNITELMA 53
7 YHTEENVETO JA JOHTOPÄÄTÖKSET 56
LÄHTEET 58
8 LIITTEET 59

Key words: control system engineering, pid-controller, python, programming

LYHENTEET JA TERMIT

A/D	Analogisesta digitaaliseksi (Analog to Digital)
API	Ohjelmointirajapinta (Application Programming Interface)
D/A	Digitaalisesta analogiseksi (Digital to Analog)
F(s)	Siirtofunktio
K	Siirtofunktion vahvistus
K_P	PID-säätimen vahvistuskerroin
OOP	Olio-ohjelmointi (Object-Oriented Programming)
PID	Suhteellinen-integroiva-derivoiva (Proportional-Integral-Derivative)
PSF	Pythonin lähdekoodin omistava järjestö (Python Software Foundation)
Python	Ohjelmointikieli
TAMK	Tampereen Ammattikorkeakoulu
T_i	PID-säätimen integrointiaika
T_d	PID-säätimen derivointiaika
T_{aw}	Kerimisen estämisen kerroin
UL	Ohjelmointikirjasto Measurement Computingin® laitteille (The Universal Library)
VDC	Tasajännite Voltit (Voltage Direct Current)
τ	Siirtofunktion aikavakio
θ	Siirtofunktion viive

1 JOHDANTO

Opinnäytetyön aiheena on laboratoriotyön tuottaminen PID-säätimen ohjelmoinnista. Tarkoituksena oli luoda Tampereen Ammattikorkeakoulun (TAMK) sähkö- ja automaatiotekniikan tutkinto-ohjelmalle mahdollisuus opettaa PID-säätimen ohjelmointia käytännössä. Tavoitteena on, että laboratoriotyön avulla opiskelija oppii PID-säätimen sisäisestä toiminnasta ja osaa ohjelmoida sellaisen itse. Työssä käydään läpi PID-säätimen toiminta, Python-ohjelmointikielen perusteet, säätimen ohjelmankoodi, säätimen laitteisto sekä laboratoriotyön suunnittelu.

PID-säädin on olennainen osa automaatiotekniikkaa ja siksi tulevien automaatioinsinöörien tulisi ymmärtää sen toimintaa. TAMK:ssa automaatiotekniikan opetuksessa säätötekniikan osalta käydään läpi prosessien mallinnuksen perusteet, PID-säätimen algoritmi pääpiirteittäin, säätimen oikeaoppinen käyttäminen sekä säätimen virittäminen. Lisäksi tutustutaan hieman kehittyneempiin säätöratkaisuihin kuten kaskadisäätöön. Teoriakursseilla opetettuja asioita päästään harjoittelemaan käytännössä laboratoriossa.

Vaikka PID-säätimen toteuttaminen tietokoneella käydään läpi pääpiirteittäin säätötekniikan teoriakursseilla, säätimen ohjelmointia tai sisäistä toimintaa ei käsitellä sen jälkeen uudestaan missään yhteydessä. Osaaminen jää pintapuoleiseksi, jos aiheesta käydään vain teoriaosuus. Minkä tahansa uuden asian oppiminen vaatii teorian lukemisen lisäksi käytännön harjoittelua ja toistoa. Laboratorioharjoituksen avulla automaatiotekniikassa olisi mahdollisuus opettaa PID-säätimen algoritmin ohjelmointia ja opiskelija saa käsityksen algoritmista sekä osaa ohjelmoida sellaisen itse.

Aluksi käydään läpi PID-säätimen algoritmin toiminnan perusteet ja miten säädin voidaan toteuttaa digitaalisesti eli tietokoneella. Samalla opitaan säätimen toimintaan ja käyttöön liittyvistä ongelmista sekä miten ne voidaan ratkaista. Ennen säätimen toteutukseen siirtymistä tutustutaan Python-ohjelmointikieleen ja laboratoriotyössä käytettävään laitteistoon. Lopussa esitellään säätimen toteutus Pythonilla. Lopuksi käydään laboratorioharjoituksen suunnittelu ja siihen liittyvät

haasteet. Laboratoriotyö tehdään Tampereen Ammattikorkeakoulun automaatiotekniikan laboratorion laitteistolla. Työn toteuttamiseksi vaaditaan tietokoneeseen yhdistyvä analoginen tulo- ja lähtökortti, tietokone ja prosessisimulaattori.

2 PID-SÄÄDIN

PID-säädin on algoritmi, jota käytetään automaatiotekniikassa säädettävän suureen automaattiseen hallintaan. Se on yleisin teollisuudessa käytetty säädintyyppi (Åström & Murray 2020, 375). Säätimen toiminta perustuu säädettävän suureen toistuvaan mittaamiseen ja toimilaitteen, kuten venttiin, asennon muuttamiseen siten, että säädettävä suure asettuu ajan myötä haluttuun arvoon. Ohjattavan toimilaiteen asentoa pitää voida muuttaa liukuvalla alueella lineaarisesti eli on/off-tyyppiset toimilaitteet eivät sovellu tähän tarkoitukseen. Säätimen nimi tulee sen algoritmin osista Proportional-Integral-Derivative eli suhteellinen-integroiva-derivoiva. (Åström & Murray 2020, 31–32)

Bennettin (1996, 19) mukaan PID-säätimen idea on keksitty jo 1922. Teknologia ei ollut kuitenkaan vielä tässä vaiheessa tarpeeksi pitkälle kehittynyt, että säädin olisi voitu toteuttaa. Teknologian kehittyessä 1930-luvulla markkinoille alkoi tulemaan ensimmäisiä pneumaattisesti toimivia säätimiä (Bennett 1996, 19). 1950-luvun puolessa välissä automaattiset säätimet olivat käytössä jo monilla eri teollisuuden aloilla (Bennett 2000, 7).

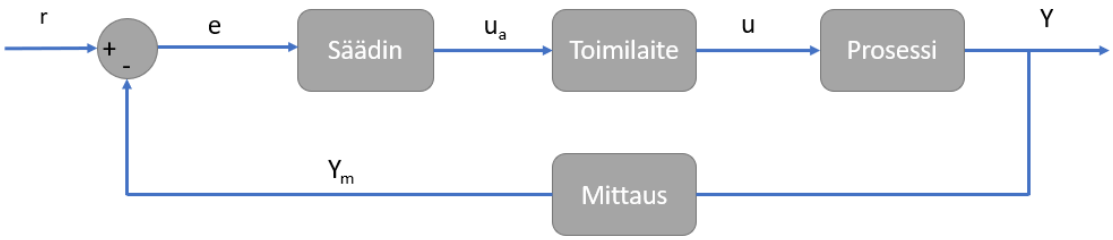
Pneumaattisia säätimiä pidettiin kehittyneimpinä ja luotettavampina verrattuna elektronisiin säätimiin. Kuitenkin sähkötekniikan kehittyessä elektroniset säätimet alkoivat haastamaan pneumaattisia säätimiä. Elektroniset säätimet olivat tässä vaiheessa vielä analogisilla virtapiireillä toteutettuja. 1950-luvun loppupuolella elektroniset PID-säätimet pystyivät kaikkeen mihin pneumaattiset PID-säätimet pystyivät ja vielä enemmän. Niillä pystyttiin tekemään esimerkiksi summausta ja monia muita matemaattisia operaatioita. (Bennett 2000, 7)

Tähän aikaan digitaalisten tietokoneiden käyttäminen säätötekniikassa oli vielä alkutekijöissään. Ensimmäinen teollisuuteen digitaalisella tietokoneella toteutettu suljetun kierron PID-säädin tehtiin vuonna 1959. 1960-luvun aikana digitaaliset säätösysteemit alkoivat yleistymään. Kuitenkin vasta 1970-luvulla alettiin puhumaan analogisten säädinten asteittaisesta vaihtamisesta digitaalisiin säätimiin. (Bennett 2000, 8)

Tietokoneella toteutetulla säätimellä on monia etuja analogisiin verrattuna. Analogisten säätimien tarkkuus huononee ajan myötä komponenttien kuluessa. Tietokoneet sen sijaan suorittavat samat operaatiot samalla tarkkuudella päivästä toiseen. Tietokoneet pystyvät suorittamaan myös huomattavasti monimutkaisempia laskutoimituksia. Logiikan lisääminen hälytysten, käynnistysten ja lopetusten toteuttamiseksi säätöön on myös helpompaa. Lisäksi graafisen käyttöliittymien tärkeyttä ei voi unohtaa. (Wittenmark, Årzén & Åström 2002, 4)

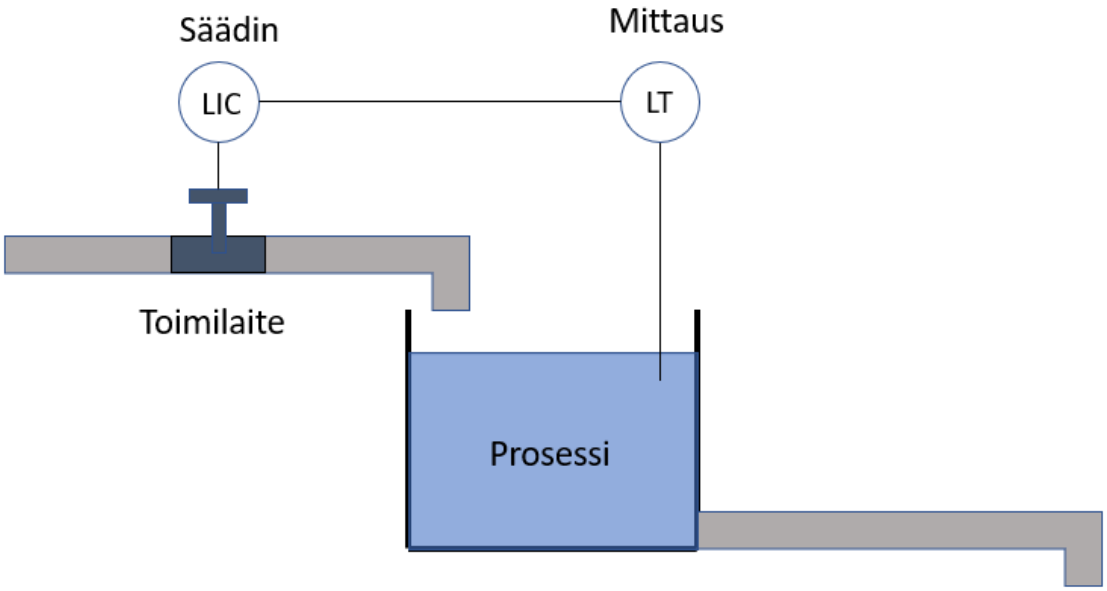
2.1 Säädin osana säätöpiiriä

Kuviossa 1 on esitetty tyypillinen säätöpiiri lohkokaaviona. Säätimelle annetaan säädettävän suureen tavoitearvo eli asetusarvo *r*. Tämä arvo voi tulla säätimen käyttöliittymän kautta käyttäjän antamana, toiselta säätimeltä tai muun laskennan tuloksena. Erosuure *e* muodostetaan vähentämällä asetusarvosta säädettävän suureen nykyinen arvo eli mittaus Y_m. Erosuure viedään säätimelle ja säädin laskee sen avulla ohjaussuureen *p* eli toimilaitteen halutun asennon. Ohjaussuure voi olla esimerkiksi venttiilin avauma tai moottorin nopeus. Toimilaite muuttaa ohjaussuureen toimisuureeksi *u*. Toimisuure voi olla esimerkiksi säiliöön menevä virtaus. Toimisuure aiheuttaa muutoksen prosessin säädettävässä suureessa *Y*. Sen jälkeen suoritetaan jälleen mittaus ja kierros alkaa alusta. (Åström & Murray 2020, 18)



KUVIO 1. Säätöpiiri esitettynä lohkokaaviona (Åström & Murray 2020, 18, muokattu)

Esimerkkinä oikeasta säätöpiiristä kuviossa 2 on esitetty altaan veden pinnankorkeuden hallinnan säätöpiiri. Altaan pinnankorkeutta hallitaan säätämällä altaaseen tulevan veden virtausta venttiilin avulla. Pinnankorkeutta mitataan jatkuvasti ja mittaustieto syötetään säätimelle. Säädin säätää toimilaitteen eli venttiilin asentoa sillä tavalla, että pinnankorkeus pystyy vakiona.



KUVIO 2. Altaan pinnankorkeuden hallinnan säätöpiiri

2.2 Prosessien mallintaminen

Laplace-muunnos

Säätöpiiriä suunnitellessa pitää luonnollisesti tuntea säädettävän prosessin luonne eli käyttäytyminen. Prosessin käyttäytymistä mallinnettaessa matemaattisesti päädytään usein lineaarisiin differentiaaliyhtälöihin. Prosessien käsittely lineaarisina differentiaaliyhtälöinä on kuitenkin usein turhan hankalaa. Käsittelyssä auttaa Laplace-muunnos. Laplace-muunnoksen avulla differentiaaliyhtälö voidaan muuntaa lineaariseksi algebralliseksi yhtälöksi, jolloin ratkaiseminen helpottuu huomattavasti. (Baher 2012, 19–23)

Laplace-muunnos määritellään integraalilla

$$F(s) = L[f(t)] = \int_0^\infty f(t)e^{-st}dt\,, \tag{1}$$

jossa s on kompleksinen taajuusmuuttuja. Laplace-muunnoksia ei tarvitse kuitenkaan laskea integraalin kautta, vaan muunnoksista on olemassa valmiit taulukot, jotka yksinkertaistavat muunnoksen tekemistä melkoisesti. Esimerkiksi funktion $f(t) = \frac{2}{5}e^{-\frac{t}{5}}$ Laplace-muunnos $F(s)$ on

$$F(s) = L[f(t)] = \frac{2}{5s+1}. \quad (2)$$

(Baher 2012, 19–23)

Laplace-muunnoksen avulla prosesseja voidaan mallintaa prosessin lähtösignaalin ja tulosignaalin suhteena (kuvio 3).

KUVIO 3. Prosessin lohkokaavioesitys

Prosessi $F(s)$ voidaan esittää signaalien suhteena kaavalla

$$F(s) = \frac{Y(s)}{U(s)}, \quad (3)$$

jossa U(s) on tulosignaali ja Y(s) on lähtösignaali. Viiveellisen ensimmäisen kertaluvun prosessin siirtofunktion yleinen muoto on

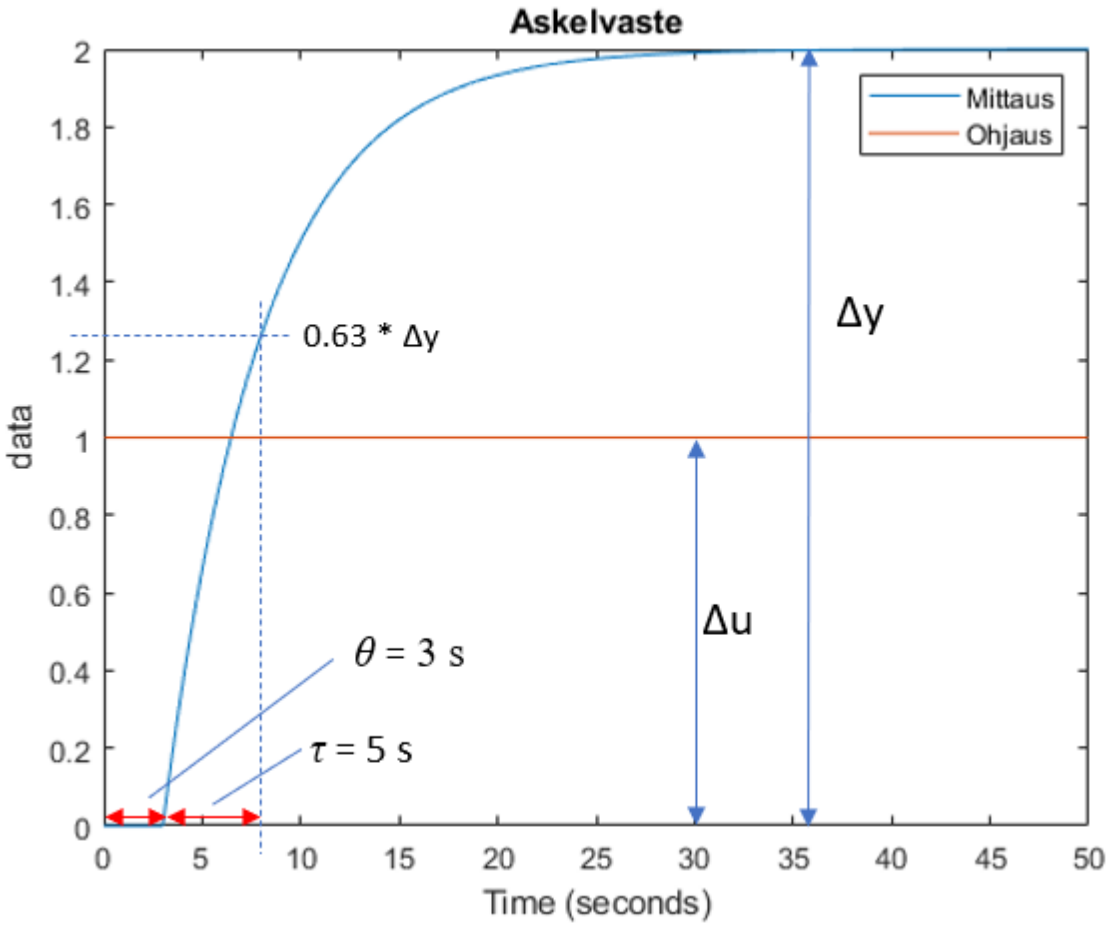
$$F(s) = \frac{K}{\tau s+1} \cdot e^{-\theta s}, \quad (4)$$

jossa K on prosessin vahvistus, τ prosessin aikavakio ja θ viive. Viiveellä tarkoitetaan tulon muutoksen ja lähdön muutoksen välistä aikaa. Viive syntyy silloin, kun mittaus ei saa lähdön muutoksen tietoa välittömästi, vaan myöhässä. (Åström & Murray 2020, 386)

Askelvaste

Askelvasteella tarkoitetaan systeemin reaktiota tulon askelmaiseen muutokseen. Sitä voidaan käyttää apuna tuntemattoman prosessin mallintamiseen. Askelvastekoe tehdään muuttamalla prosessin tuloa askelmaisesti ja mittaamalla prosessin lähtö. Kokeen tuloksena saadaan askelvaste. Askelvasteesta voidaan määrittää prosessin parametrit ja luoda siitä malli. Askelvastekoetta tehtäessä

tulee odottaa, että prosessi saavuttaa tasapainotilan. Kun prosessin $F(s) = \frac{2}{5s+1} \cdot e^{-3s}$ tuloa muutetaan askelmaisesti arvosta 0 arvoon 1, saadaan kuvion 4 mukainen kuvaaja. (Åström & Murray 2020, 53–54)



KUVIO 4. Askelvaste

Askelvasteesta voidaan määrittää prosessin vahvistus *K*, aikavakio τ sekä viive *θ*. Vahvistus saadaan määritettyä prosessin lähdön (mittauksen) suhteesta tulon (ohjauksen) muutokseen:

$$K = \frac{\Delta y}{\Delta u}, \tag{5}$$

jossa Δy on lähdön muutos ja Δu tulon muutos. Prosessin aikavakio saadaan hetkestä, kun mittaus saavuttaa 63 % tasapainotilan arvostaan. Viive mitataan tulon muutoksen ja lähdön muutoksen välisenä aikana. Vahvistuksen, aikavakion ja viiveen määritys kuvaajasta on havainnollistettu kuviossa 4. (Åström & Murray 2020, 387)

2.3 PID-säätimen algoritmi

PID-säätimen toiminta perustuu ohjattavan toimilaitteen asennon säätämiseen siten, että prosessin suure asettuu haluttuun arvoon. Toimilaitteen asennon laskeminen tapahtuu algoritmin avulla. Algoritmin tulo on erosuure eli poikkeama säädettävän suureen asetusarvon ja mitatun arvon välillä. (Åström & Murray 2020, 17)

Erosuure lasketaan kaavalla

$$e = r - y_m, \tag{6}$$

jossa r on säädettävän suureen asetusarvo ja y_m on mitattu säädettävä suure (Åström & Murray 2020, 376, muokattu).

Åströmin ja Murrayn (2020, 376) mukaan PID-säätimen algoritmi ohjaussignaalin laskemiselle on

$$u(t) = K_\mathrm{p} e(t) + \frac{K_\mathrm{p}}{T_i}\int_0^t e(\tau)d\tau + K_\mathrm{p} T_d \frac{de(t)}{dt} = P(t) + I(t) + D(t), \tag{7}$$

jossa *u* on ohjaussignaali, *e* on erosuure, K_p on vahvistuskerroin, T_i on integrointiaika ja T_d on derivointiaika. Ohjaussignaali *u(t)* koostuu siis kolmen eri termin summasta: suhteellinen (Proportional) $P(t)$, integroiva (Integral) $I(t)$ ja derivoiva (Derivative) $D(t)$. Tästä syystä PID-säätimiä kutsuttiin alkuperäisesti kolmen termin säätimiksi. (Åström & Murray 2020, 376)

Säätimen algoritmi voidaan esittää myös Laplace-muodossa kaavalla

$$u(s) = K_\mathrm{p} e(s) + \frac{K_\mathrm{p}}{T_i s} e(s) + K_\mathrm{p} T_d s e(s), \tag{8}$$

josta voidaan muokata säätimen siirtofunktio $C(s)$ohjauksen ja erosuureen suhteena

$$C(s) = \frac{u(s)}{e(s)} = K_\mathrm{p} + \frac{K_\mathrm{p}}{T_i s} + K_\mathrm{p} T_d s. \tag{9}$$

(Åström & Murray 2020, 378)

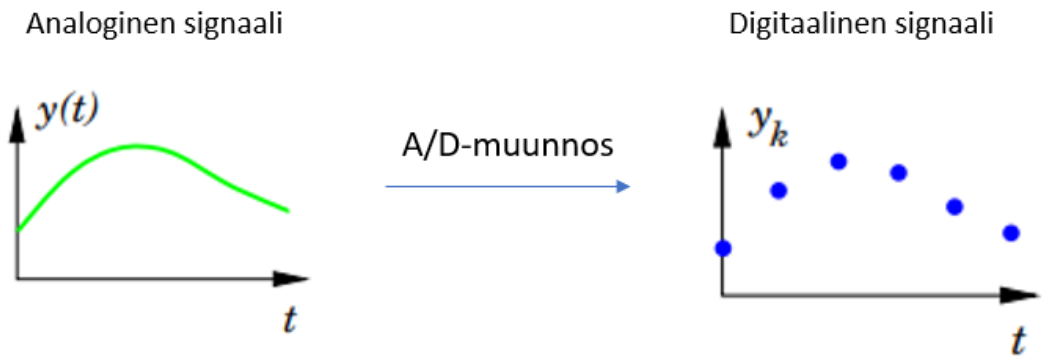
Mitä suuremmaksi integrointi aika valitaan, sitä pienempi vaikutus ohjaukseen integroinnilla on. Jos integrointiajaksi valitaan ääretön, integroiva termi poistuu kokonaan. Silloin kyseessä olisi pelkkä PD-säädin. Jos derivointiaika asetetaan nollaksi, derivoiva termi katoaa kokonaan. Silloin kyseessä on PI-säädin. Åströmin ja Murrayn (2020, 375) mukaan 94.4 % kaikista teollisuudessa käytetyistä säätimistä on PI-säätimiä. (Åström & Murray 2020, 375)

2.4 Algoritmin toteutus tietokoneella

PID-säädin (7) on jatkuva-aikainen dynaaminen systeemi (Åström & Murray 2002, 396). Prosessista mitattava signaali tuottaa tietokoneille ongelmia. Tietokoneet eivät pysty käsittelemään analogisia eli jatkuva-aikaisia signaaleja, koska ne suorittavat operaatioita oman sisäisen kellonsa tahtiin eivätkä jatkuva-aikaisesti. Signaalin arvo voidaan tietää vain niillä ajanhetkillä, kun tietokone suorittaa signaalin mittaamisen. Tämän takia säätimen algoritmi joudutaan toteuttamaan likimääräisesti eli approksimoiden. (Wittenmark, Årzén & Åström 2002, 4)

Signaalien muuntaminen

Signaalin muuntaminen digitaaliseksi tapahtuu A/D-muuntimen (analog-to-digital) avulla. A/D-muuntimet muuttavat analogisen signaalin, kuten jännitteen hetkellisen arvon, digitaaliseksi numeroksi rajallisella tarkkuudella. Muunnoksen tarkkuus riippuu muuntimessa käytetyistä bittien tai tasojen määrästä. Signaalin muuntaminen digitaaliseksi tehdään ottamalla mittauksia eli ”näytteitä” oikeasta signaalista A/D-muuntimella tasavälein, jolloin tuloksena saadaan karkeasti oikeaa signaalia muistuttava paloissa muodostettu digitaalinen signaali. Digitaalisen signaalin tarkkuus paranee ottamalla näytteitä pienemmällä aikavälillä. Oikean signaalin tarkkuutta ei kuitenkaan pystytä saavuttamaan, vaan kyseessä on aina approksimaatio. Kuvio 5 havainnollistaa analogisen signaalin muodostamista. (Wittenmark ym. 2002, 4)



KUVIO 5. Analogisen signaalin muuntaminen digitaaliseksi (Wittenmark ym. 2002, 5, muokattu)

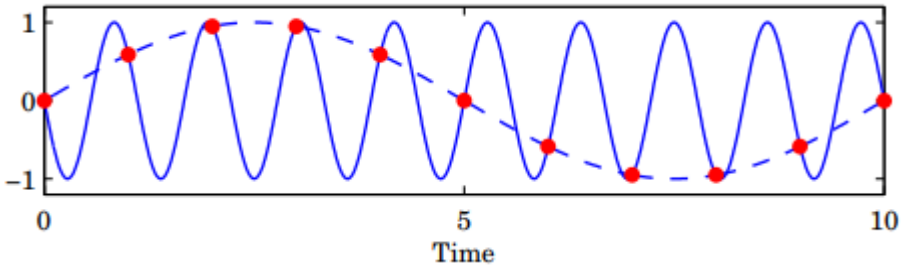
Vastavuoroisesti D/A-muunnin (digital-to-analog) muuntaa digitaalisen luvun takaisin jatkuva-aikaiseksi signaaliksi. Tämä merkitsee, että D/A-muuntimessa on muunninyksikkö sekä pitoyksikkö, mitkä kääntävät numeron fyysiseksi suureeksi, joka menee prosessille. Pitoyksikkö pitää yllä signaalin arvoa, kunnes seuraava näyte lähetetään. Kuva 6 havainnollistaa digitaalisen signaalin muodostamista (Wittenmark ym. 2002, 4)

KUVIO 6. Digitaalisen signaalin muuntaminen analogiseksi (Wittenmark ym. 2002, 5, muokattu)

Laskostuminen

Jos signaalin näytteenottoväliksi valitaan liian suuri väli, aletaan menettämään alkuperäisen signaalin sisältämää tietoa. Liian pienellä näytteenottotaajuudella muodostetun digitaalisen signaalin taajuus on eri kuin alkuperäisen signaalin. Tätä ilmiötä kutsutaan laskostumiseksi. Kuvio 7 havainnollistaa laskostumista.

Sininen yhtenäinen viiva on alkuperäinen signaali ja katkoviiva on laskostunut digitaalinen signaali. (Wittenmark ym. 2002, 6)



KUVIO 7. Laskostumisen tapahtuminen liian pienellä näytteenottotaajuudella (Wittenmark ym. 2002, 6)

Kuviossa 4 alkuperäisen signaalin taajuus on 0.9 Hz ja laskostuneen signaalin taajuus on 0.1 Hz. Näytteenottoväli h on 1 sekunti. Alkuperäisestä signaalista saadaan siis vääristynyt kuva, koska taajuus on muuttunut täysin. Laskostumiselta vältytään silloin, kun näytteenottotaajuus on kaksin kertainen mitattavaan signaalin taajuuteen verrattuna. (Wittenmark ym. 2002, 6)

Algoritmin ohjelmakierto

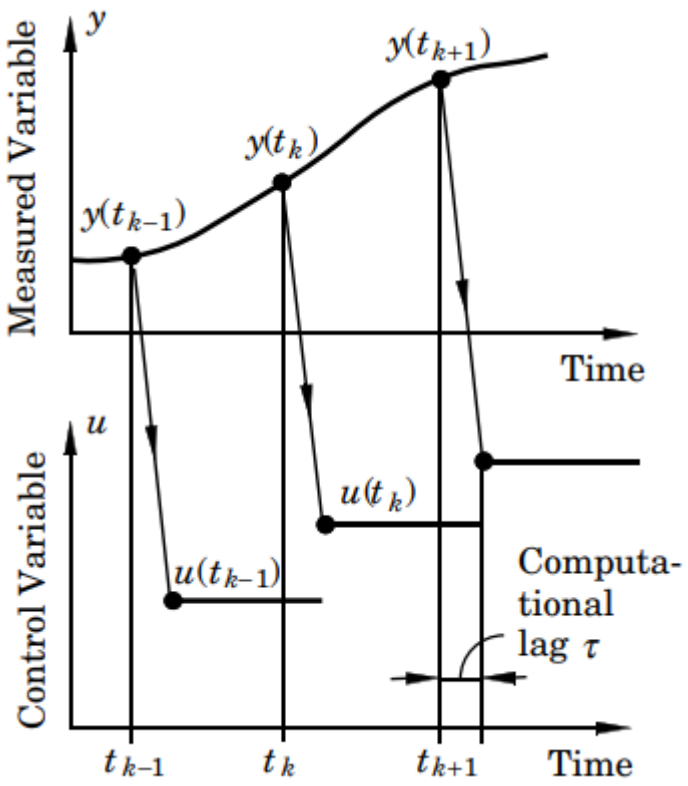
Åström ja Murray (2020, 396) kuvailevat PID-säätimen diskreettiaikaisen toteutuksen eli tietokonetoteutuksen toimivan jaksottaisesti. Koska mittaussignaalin arvoa ei voida tietää kaikkina hetkinä, kun mittaus ja algoritmi suoritetaan tasaväliajoin. Jokaisella jaksolla tehdään sama sarja operaatioita:

1. Odota uutta kierrosta
2. Mittauksen lukeminen (A/D-muunnos)
3. Ohjauksen laskeminen
4. Ohjauksen lähettäminen toimilaitteelle (D/A-muunnos)
5. Säätimen tilan päivittäminen
6. Toista sarja.

Tavoitteena on, että säädin suorittaisi jokaisen jakson mahdollisimman vakiolla suoritusvälillä. Vaikka tietokoneet suorittavat matemaattisia operaatioita todella

nopeasti, mittauksen ottamisen ja ohjauksen välillä esiintyy pientä laskennan aiheuttamaa viivettä. Viive on pyritty minimoimaan tekemällä ohjauksen laskeminen mahdollisimman nopeaksi ja suorittamalla säätimen tilan päivitykset vasta ohjauksen lähettämisen jälkeen. (Åström & Murray 2020, 396)

Kuvio 8 havainnollistaa säädettävän suureen jaksottaisen mittaamisen ja ohjauksen laskemisen välistä viivettä. Käyrä $y(t_k)$ on mitattu säädettävä suure ja $u(t_k)$ on laskettu ohjaus. Merkintä t_k tarkoittaa nykyisellä ajanhetkellä suoritettua laskentakierrosta, t_{k-1} edellistä kierrosta ja t_{k+1} tulevaa kierrosta. Alaindeksi k saa siis kokonaisluku arvoja ja kertoo, viitataanko menneeseen vai tulevaan kierrokseen. Laskennallinen viive on merkattu kuvassa merkillä τ. (Wittenmark ym. 2002, 66)



KUVIO 8. Mittaamisen ja ohjauksen lähettämisen välinen viive (Wittenmark ym. 2002, 66, muokattu)

2.4.1 Suhteellinen termi

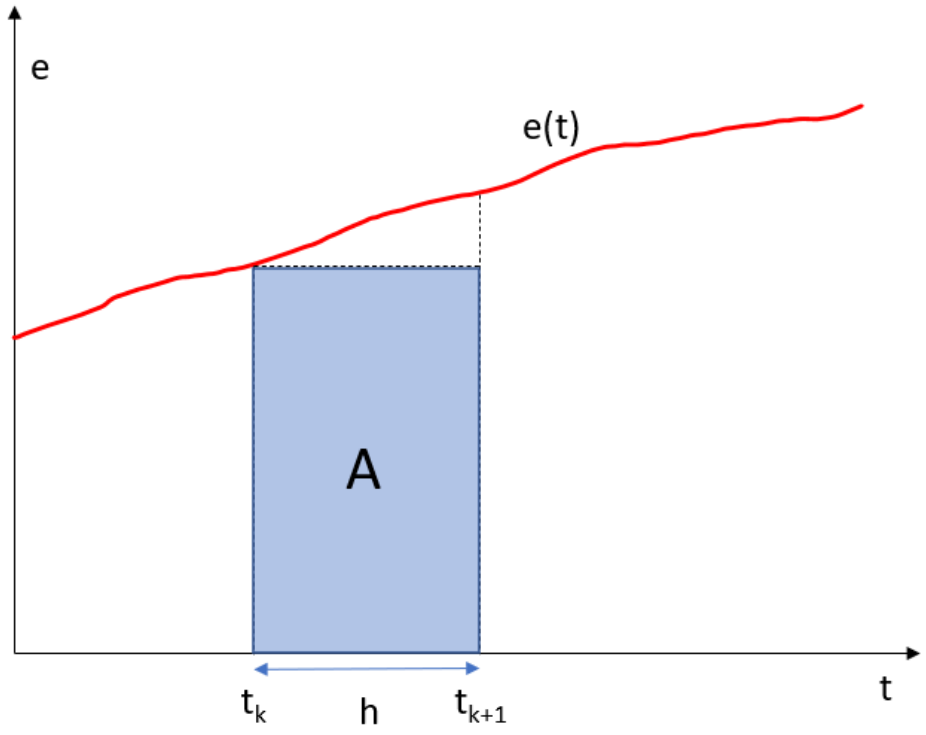
PID-säätimen tietokonetoteutuksen yksinkertaisin osuus on P-termi. Åströmin ja Murrayn (2020, 396) mukaan P-termi toteutetaan korvaamalla jatkuva-aikaisen toteutuksen osat näytteistetyillä versioilla

$$P(t_k) = k_p\big(\beta r(t_k) - y(t_k)\big), \tag{10}$$

jossa β on asetusarvon painokerroin. Sen avulla voidaan suodattaa asetusarvon muutoksen vaikutusta ohjaukseen. Painokerroin saa arvoja väliltä 0–1. (Åström & Murray 2020, 396)

2.4.2 Integroiva termi

Integroiva termi saadaan laskemalla erosuureen pinta-ala suorakaiteena likimääräisesti käyttämällä Euler-approksimaatiota eteenpäin (Wittenmark ym. 2002, 34). Kuvio 9 havainnollistaa integraalin approksimoinnissa tapahtuvaa erosuureen pinta-alan laskemista. Käyrä e(t) on erosuureen kehittyminen ajan suhteen.



KUVIO 9. Erosuureen integraali Euler-approksimaatiolla eteenpäin suorakaiteen muodossa (Wittenmark ym. 2002, 34, muokattu)

Erosuureen käyrän alle jäävän suorakaiteen pinta-ala eli integraali saadaan approksimoitua kaavalla

$$A = e(t_k) \cdot h, \tag{11}$$

jossa $e(t_k)$ on erosuureen arvo näytteenottohetkellä ja *h* on laskentaväli. Mitä pienemmän arvon *h* saa, sitä tarkempi approksimaatio integraalille saadaan. (Wittenmark ym. 2002, 34)

Joka kierroksella saadaan uusi integroiva termi, kun vanhaan integraaliin summataan uusi erosuureen approksimoitu pinta-ala (kuvio 9). Åströmin ja Murrayn (2020, 396) mukaan algoritmin suoritusväliä merkataan kirjaimella h, jolloin uusi kierros on $t_{k+1} = t_k + h$. Integraali saadaan kaavalla

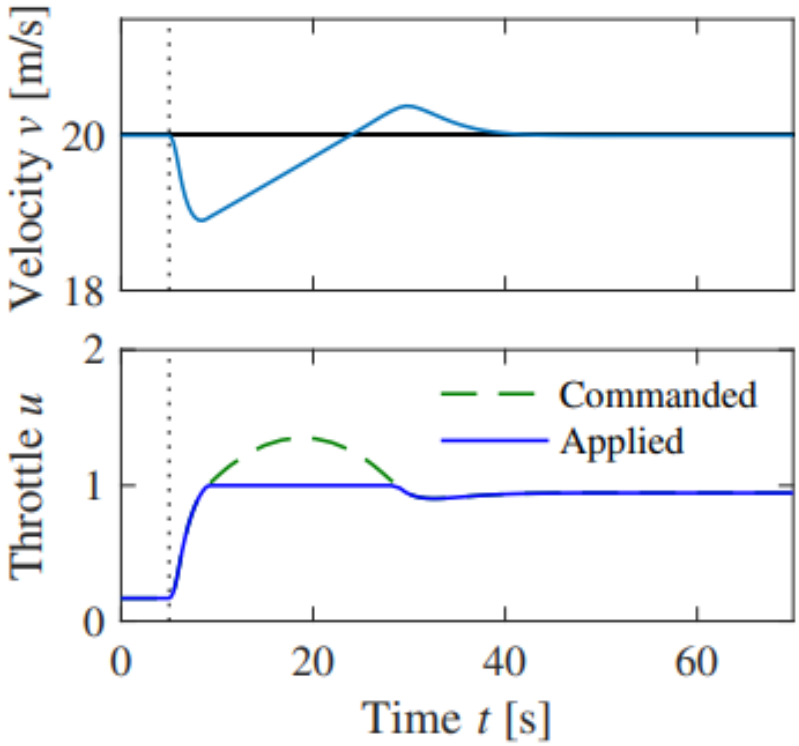
$$I(t_{k+1}) = I(t_k) + \frac{K_p}{T_i} e(t_k)h \tag{12}$$

(Åström & Murray 2020, 396).

Integraalin kerimisen estäminen

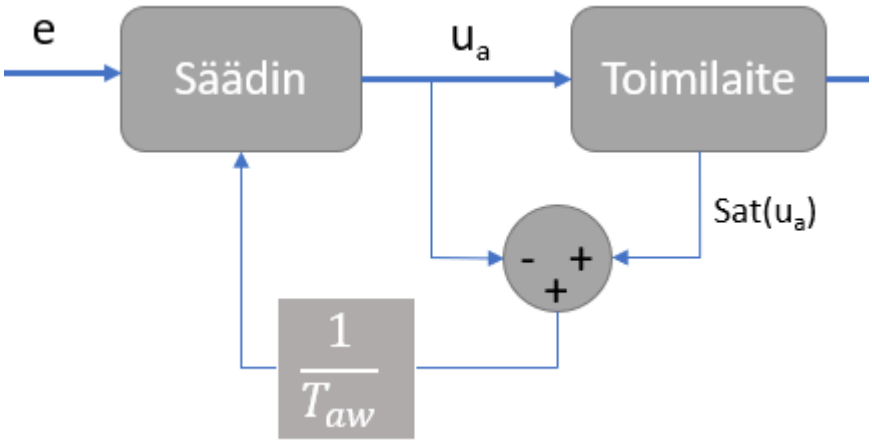
Åströmin ja Murrayn (2020, 389) mukaan toimilaitteilla on tyypillisesti rajat: moottorilla on maksiminopeus ja venttiili voi olla vain täysin auki tai suljettu. Tilannetta, jossa toimilaite saavuttaa toimialueensa rajan, kutsutaan toimilaitteen kyllästymiseksi (saturation). Kyllästymisen tapahtuessa integroiva termi voi kasvaa hyvin suureksi, koska erosuuretta integroidaan jatkuvasti lisää ohjaukseen, vaikka toimilaite on jo saavuttanut rajansa. Tätä ilmiötä kutsutaan integraalin kerimiseksi (integrator windup). Vaikka erosuure muuttuisi, integroiva termi on kasvanut niin suureksi, että sillä kestää kauan palata toimilaitteen toimialueen rajojen sisäpuolelle. (Åström & Murray 2020, 389)

Kuviossa 10 näkyy visualisoi integraalin kerimisen ongelman. Yläpuolella näkyy säädettävä nopeus ja alhaalla ohjaus. Nopeuden noustessa hitaasti toimilaite ajautuu toimialueensa reunalle nopeasti. Asetusarvon ja mittauksen välillä on kuitenkin erosuuretta, joten integraali termi kasvaa lisää. Ohjauksesta nähdään, että laskettu ohjaus (vihreä katkoviiva) on todellista toimilaitteen asentoa (sininen viiva) suurempi. Toimilaite ei kuitenkaan pysty nousemaan rajaansa korkeammalle ja jää arvoon 1. Nopeus saavuttaa lopulta asetusarvon mutta menee sitten sen yli, koska integraali on ehtinyt kasvaa liian suureksi. Vähitellen ohjaus palaa toimialueelle ja nopeus palaa asetusarvoon. (Åström & Murray 2020, 390)



KUVIO 10. Integraalin kerimisen vaikutus säädettävään suureeseen (Åström & Murray 2020, 390, muokattu)

Kerimisen estämiseen (anti-windup) on kehitetty monia eri tapoja. Åströmin ja Murrayn (2020, 390) tarjoama metodi on lisätä säätimeen yksi takaisinkytkentä lisää, joka tulee toimilaitteen todellisesta asennosta joko mittaamalla asento tai mallintamalla toimilaitetta (kuvio 11). Säätimen lasketun ohjauksen ja toimilaitemallin arvosta lasketaan erotus, joka jaetaan kertoimella T_{aw} ja summataan integroivaan termiin. Parametrin T_{aw} suositellaan valittavaksi pienempi luku kuin säätimen integrointiaika. Kun erosuure on nolla, ohjausten erotus on myös nolla, jolloin sillä ei ole vaikutusta säätöpiirin toimintaan. (Åström & Murray 2020, 390)



KUVIO 11. Kerimisen esto kuvattuna lohkokaaviona (Åström & Murray 2020, 391, muokattu)

Lisäämällä kerimisen esto integraalin approksimaation kaavaan (12) saadaan integroivalle termille kaava

$$I(t_{k+1}) = I(t_k) + \frac{K_p}{T_i} e(t_k)h + \frac{h}{T_{aw}}(sat(u_a) - u_a) \tag{13}$$

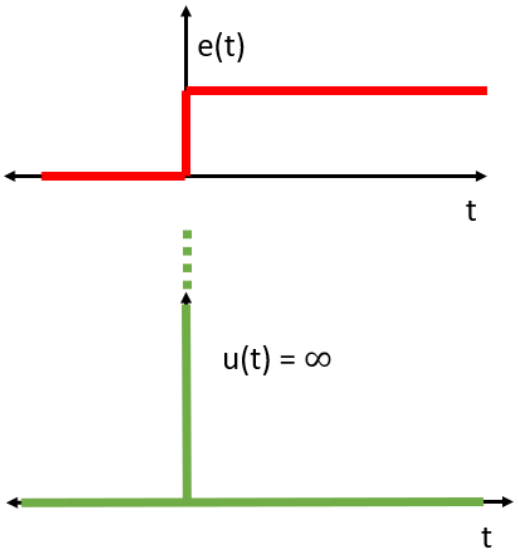
(Åström & Murray 2020, 396).

2.4.3 Derivoiva termi

Vaikka PID-säätimen derivoiva termi on määritelty kaavassa 2 erosuuren derivoimiseksi ajan suhteen, Wittenmarkin ym. (2002, 45) mukaan se ei ole käytännössä mahdollista toteuttaa eikä kannattaisikaan. Erosuureen derivoiminen johtaisi kahteen ongelmaan: mittauksen kohinan vahvistamiseen sekä suuriin piikkeihin ohjauksessa, kun asetusarvoa muutetaan.

Säädinten asetusarvoja muutetaan usein askelmaisesti, mikä johtaa erosuureen askelmaiseen muutokseen. Askelmainen muutos erosuureessa johtaa siihen, että derivoiva termi on ääretön, koska erosuureen muutos on pystysuora eli kulmakerroin on ääretön. Kuviossa 12 näkyy, miten D-termiin tulee äärettömän korkea ’’piikki’’, kun erosuure muuttuu. Piikki derivoivassa termissä, johtaisi säätimen ohjauksen ajautumisen maksimiarvoon pienistäkin muutoksista erosuureessa, mikä huonontaa säätöä. Ongelmaan esitetty ratkaisu on olla derivoimatta

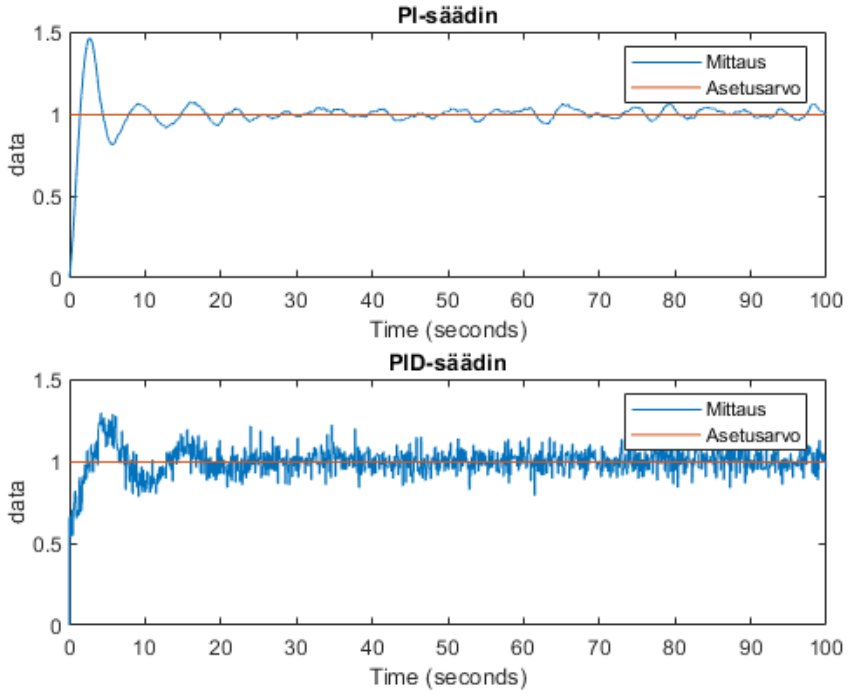
erosuuretta, vaan sen sijaan käytetäisiinkin mittauksen *y* derivaattaa (Wittenmark ym. 2002, 45).



KUVIO 12. Derivoivan termin askelvaste erosuureen äkillisestä muutoksesta

Derivaatan suodatus

Erosuureen derivoimisen toinen ongelma on kohinan vahvistaminen. Kuviossa 13 on suoritettu askelvastekokeet PI- ja PID-säätimelle tilanteessa, jossa prosessin mittauksessa esiintyy kohinaa. Kuviosta nähdään, miten derivointi vahvistaa kohinaa.



KUVIO 13. PI- ja PID-säädinten askelvasteet, kun mittauksessa esiintyy kohinaa.

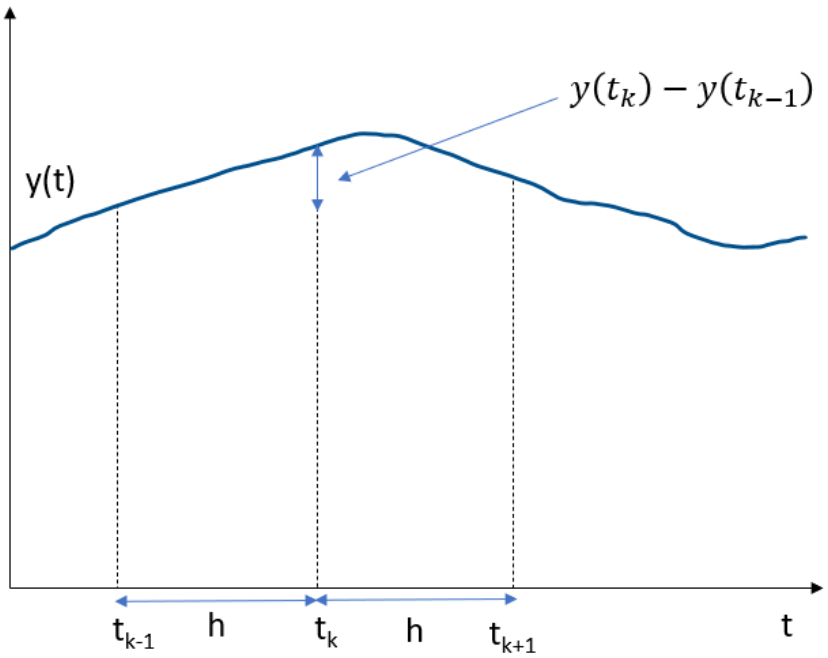
Derivoinnin vahvistusta pitää siis rajata jotenkin. Ratkaisuna on suodattaa derivaatta alipäästösuotimen läpi. Alipäästösuodin suodattaa signaalista korkeat taajuudet mutta jättää matalat taajuudet. Alipäästösuodinta voi mallintaa ensimmäisen kertaluvun prosessin siirtofunktiolla. Suodatettu derivaatta saadaan kertomalla derivointiaika alipäästösuotimen siirtofunktiolla

$$T_d s \approx \frac{1}{\tau_f s + 1} \cdot T_d s, \tag{14}$$

jossa τ_f on suotimen aikavakio. Derivaatan suotimen aikavakioksi valitaan tyypillisesti T_d/N, jossa N saa arvoja väliltä 3–20. Aikavakion τ_f saa olla korkeintaan 1/3 derivointiajasta T_d. Suodatus ei kokonaan poista kohinan vahvistuksen ongelmaa, joten derivoinnin mukaan ottamisen kanssa pitää olla tarkkana. (Wittenmark ym. 2002, 45).

Derivaatan approksimoiminen

Kuten integraalia approksimoidessa, mittauksen derivaatta saadaan Euler-approksimaation avulla mutta tällä kertaa taaksepäin. Kuviossa 14 näkyy, miten derivaatta saadaan laskettua kahden laskentakierroksen mittauksen välillä.



KUVIO 14. Derivaatan Euler-approksimaatio taaksepäin (Wittenmark ym. 2002, 45)

Mittauksen *y* derivaatta Euler-approksimaatiolla taaksepäin saadaan laskettua kaavalla

$$\frac{dy}{dt} \approx \frac{y(t_k) - y(t_{k-1})}{h}, \tag{15}$$

jossa $y(t_k)$ on nykyisen laskentakierroksen mittaus, $y(t_{k-1})$ on edellisen laskentakierroksen mittaus ja *h* on laskentakierrosten välinen aika (Wittenmark ym. 2002, 45).

Åström ja Murray (2020, 396) antavat suodatetun derivoivan termin D:n differentiaaliyhtälöksi

$$T_f \frac{dD}{dt} + D = -K_p T_d \frac{dy}{dt}. \tag{16}$$

Åströmin ja Murrayn (2020, 397) mukaan approksimoimalla derivaatta nykyisen ja edellisen laskentakierroksen mittauksen välillä derivoivalle termille saadaan kaava

$$T_f \frac{D(t_k) - D(t_{k-1})}{h} + D(t_k) = -K_p T_d \frac{y(t_k) - y(t_{k-1})}{h}, \tag{17}$$

joka voidaan kirjoittaa muodossa

$$D(t_k) = \frac{T_\mathrm{f}}{T_\mathrm{f} + h} D(t_{k-1}) - \frac{K_p T_d}{T_\mathrm{f} + h} \left(y(t_k) - y(t_{k-1})\right) \tag{18}$$

(Åström & Murray 2020, 397).

2.4.4 Vaihtaminen manuaali- ja automaattitilan välillä

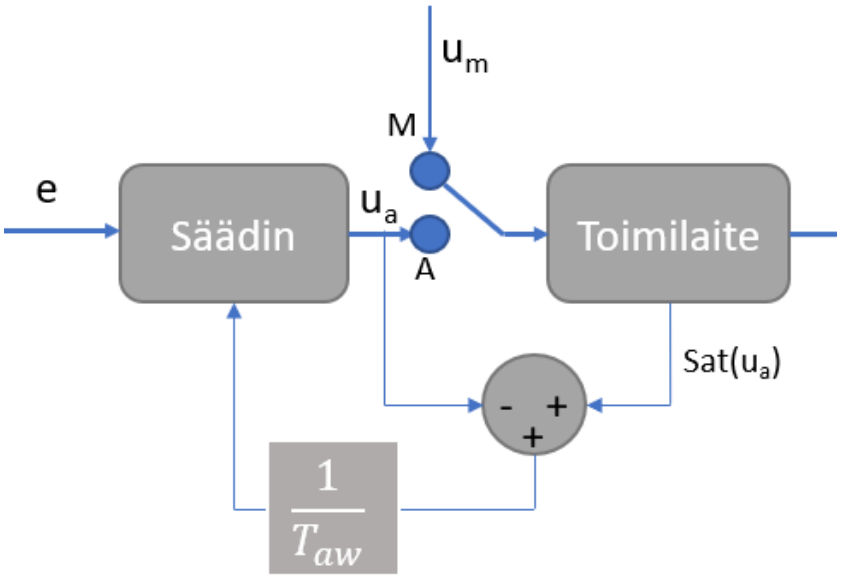
PID-säätimissä on tyypillisesti kaksi eri tilaa ohjauksen asettamiseksi. Automaattitilassa säätimen algoritmi laskee ohjauksen arvon erosuureen avulla ja muuttaa ohjausta tarvittaessa automaattisesti. Manuaalitilassa käyttäjä asettaa ohjauksen arvon itse. Ohjauksen arvoa päivitetä muutoin kuin käyttäjän toimesta. Säätimen

algoritmi kuitenkin pyörii taustalla, vaikka sen laskemaa ohjausta ei lähetettäisikään toimilaitteelle manuaalitilassa. (Åström & Murray 2020, 392)

Säätimen tilojen välillä on kyettävä vaihtamaan tarpeen vaatiessa. Tilojen välillä vaihtamisessa on kuitenkin riskinsä. Manuaalitilassa asetettu ohjaus ei välttämättä ole sama kuin automaattitilassa laskettu ohjaus. Vaihtaminen tilojen välillä voi johtaa isoon ”hyppyyn” ohjauksessa, mikä vastaavasti voi aiheuttaa nopean muutoksen prosessissa. Sama voi tapahtua myös vaihtaessa automaatilta manuaalille. Nopeat heilahdukset prosessissa voivat pilata valmistettavan tuotteen tai aiheuttaa jopa vaaratilanteen, esimerkiksi putkistossa voi syntyä paineaalto venttiilin sulkeutuessa äkillisesti. (Åström & Murray 2020, 392)

Tilan vaihtamisen yhteydessä tapahtuvan hypyn välttämiseen tai vähintään minimoimiseen on erilaisia tapoja. Åström ja Murray (2020, 392) esittävät ongelmaan ratkaisuksi integroivan termin päivittämisen siten, että automaattiohjaus seuraa käyttäjän asettamaa ohjauksen arvoa säätimen ollessa manuaalitilassa. Tällä tavalla automaattiohjaus ja manuaaliohjaus ovat samat kaikilla hetkillä, ja vaihdon yhteydessä ei siten tapahdu hyppyä. (Åström & Murray 2020, 392)

Integroivan termi päivittäminen manuaalitilassa tapahtuu samalla takaisinkytkennällä kuin mitä integraalin kerimisen estossa käytetään (kuvio 11). Takaisinkytkentä varmistaa, ettei algoritmin laskema automaattiohjaus kasva suuremmaksi kuin manuaaliohjaus päivittämällä integraalitermin arvoa. Tämän toiminnallisuuden lisäämiseksi säätimeen ei vaadi muita muutoksia kuin sen, että manuaalimoodissa algoritmi suorittaa laskentaa vapaasti. Kuviota 11 voidaan päivittää lisäämällä siihen kytkin tilojen vaihtamista varten (kuvio 15). (Åström & Murray 2020, 392)



KUVIO 15. Kaavio tilanvaihdossa tapahtuvan hypyn estämiseksi (Åström & Murray 2020, 391, muokattu)

2.4.5 Säätimen parametrien muuttaminen

Samaan tapaan kuin säätimen tilaa vaihdettaessa, myös säätimen parametreja vaihtaessa voi tapahtua hyppy ohjauksessa. Hyppyä ei tapahdu, jos asetusarvon suodatusparametri *b* on 1, koska prosessin ollessa tasapainotilassa erosuure on nolla ja täten P-termikin on nolla. Jos *b* ei ole yksi, P-termi ei ole tasapainotilassa nolla, jolloin hyppy tapahtuu parametreja muutettaessa. Hypyn välttämiseksi integroivaan termiin lisätään parametrien vaihdon aiheuttama muutos P-termissä. Hypytön vaihto saadaan aikaan suorittamalla vaihdon jälkeen kaava

$$I_{uusi} = I_{vanha} + K_{P_{vanha}}(b_{vanha}r - y) - K_{P_{uusi}}(b_{uusi}r - y), \qquad (19)$$

jossa $K_{P_{vanha}}$ on vanha vahvistuskerroin, b_{vanha} on vanha asetusarvon suodatus parametri, $K_{P_{uusi}}$ on uusi vahvistuskerroin ja b_{uusi} on uusi asetusarvon suodatus parametri. Kaavan koodi ajetaan vaihdon jälkeen laskentakierroksen alussa. Koodi vaatii uusien parametrien vaihdon huomaamista jotenkin. (Wittenmark ym. 2002, 49)

3 PYTHON-OHJELMOINTIKIELI

Laboratoriotyö tullaan toteuttamaan Pythonilla. Python on ohjelmointikieli, jonka kehitti Guido van Rossum 1990-luvun alkupuolella ja tällä hetkellä sitä ylläpitää useista ohjelmoijista koostuva yhteisö (Donaldson 2008, 2). Pythonin lähdekoodin omistaa voittoa tavoittelematon Python Software Foundation -järjestö (PSF) (Python Software Foundation 2002). Python on tullut suosituksi liiketoiminnallisissa sekä akateemisissa sovelluksissa (Gaddis 2015, 35).

Chun (2007, 33) kuvailee Pythonia elegantiksi ja robustiksi kieleksi, joka tarjoaa tehokkuutta sekä tulkatuille kielille tyypillistä yleiskäytettävyyttä, helppokäyttöisyyttä unohtamatta. Tulkatulla kielellä tarkoitetaan ohjelmointikieltä, jossa jokainen käsky tulkataan yksitellen konekieleksi ja suoritetaan välittömästi. Prosessi suoritetaan ohjelman jokaiselle käskylle. (Gaddis 2015, 37)

3.1 Syntaksi

Davidsonin (2008, 2) sanoo Pythonin suunnitelluksi syntaksiltaan helposti luettavaksi ja opittavaksi. Python-ohjelmat näyttävät muihin kieliin verrattuna selkeiltä ja siisteiltä. Kielessä ei ole turhia symboleita ja avainsanoiksi on valittu selkeitä englanninkielisiä sanoja. (Davidson 2008, 2)

Muuttujat

Muuttujat ovat olennainen osa ohjelmointia, sillä niihin tallennetaan ohjelmalle tarvittava tieto. Pythonin perusmuuttujatyypit ovat kokonaisluvut, desimaaliluvut eli liukuluvut, merkkijonot. Pythonista löytyy myös totuusarvomuuttujat. Muuttujan tietotyyppiä ei tarvitse määrittää muuttujaa luodessa, koska Python hoitaa sen automaattisesti annetun arvon mukaan. Muuttujalle pitää antaa vain nimi. Pitää kuitenkin muistaa, että muuttujaa käsitellään sen tietotyypin mukaan. Kokonaislukumuuttujien varaaman muistin kokoa ei tarvitse miettiä, koska Python varaa kulissien takana automaattisesti lisää muistia tarvittaessa. (Donaldson 2008, 9)

TAULUKKO 1. Yleisimmät muuttujatyypit (Donaldson 2008, 9)

Nimi	Arvo
Kokonaisluku	1
Desimaaliluku	2.0
Merkkijono	”Omena”
Totuusarvo	True

Matemaattiset operaatiot

Taulukossa 2 on listattu Pythonin matemaattisten operaattorit ja esimerkki niiden käytöstä. Operaatioihin pätevät matematiikan suoritusjärjestyksen säännöt. Kaarisulkeiden avulla voidaan muuttaa suoritusjärjestystä (Donaldson 2008, 12)

TAULUKKO 2. Matemaattiset operaattorit (Donaldson 2008, 12, muokattu)

Nimi	Operaattori	Esimerkki
Summaus	+	>>>3 + 4 7
Vähennys	–	>>>5 – 3 2
Kertominen	*	>>>2 * 3 6
Jakaminen	/	>>>3 / 2 1.5
Kokonaislukujakaminen	//	>>>3 // 2 1
Jakojäännös	%	>>>25 % 3 1
Potenssiin korotus	**	>>> 3 ** 3 27

Listat

Matthesin (2015, 33) mukaan lista on kokoelma asioita tietyssä järjestyksessä. Listan elementtien ei tarvitse liittyä toisiinsa mitenkään. Ne voivat olla numeroita tai vaikka merkkijonoja. Lista määritellään Pythonissa hakasulkeilla. Hakasulkeiden sisään kirjoitetaan listan elementit erotettuna pilkulla. Yksittäiseen listan elementtiin päästään käsiksi kirjoittamalla listan nimi ja perään hakasulkeissa elementin indeksi. Listan elementtien indeksi alkaa tietokoneille tyypilliseen tapaan nollasta. (Matthes 2015, 33)

```
# Tyhjä lista
lista = []

# Merkkijonolista
lista2 = ["foo", "bar"]

# Tulosta listan ensimmäinen elementti
print(lista2[0])
```

Output:

```
foo
```

Listan pituutta ei tarvitse määritellä sitä luodessa, vaan muistia varataan lisää, kun sitä tarvitaan. Listan loppuun voi lisätä elementin *append*-metodilla ja poistaa *pop*-metodilla (Matthes 2015, 37, 39).

```
# luo kolmen kokonaisluvun lista
lista = [1, 2, 3]

# Lisää listan loppuun numero 4
lista.append(4)

# Poista listan viimeinen elementti
lista.pop()

# Tulosta lista konsolille
print(lista)
```

Output:

```
[1, 2, 3]
```

Sisennykset

Pythonissa ohjelmalohkot kuten ehdollisen rakenteet, funktiot ja luokat erotetaan muusta koodista sisennyksen (*indentation*) avulla. Sisennys tehdään rivin alkuun neljän välilyönnin avulla tai *tab*-painikkeella. Jos sisennys puuttuu, Python antaa virheilmoituksen. Myös turha sisennys aiheuttaa virheilmoituksen. (Matthes 2015, 399)

Ehdolliset rakenteet

Matthesin (2015, 71) mukaan ohjelmoinnissa tutkitaan usein kokoelmia ehtoja ja tehdään ehtojen mukaan päätöksiä. *If*-lausekkeet mahdollistavat ohjelman tilan tutkimisen ja vastata tilaan oikealla tavalla. Pythonissa *if*-lausekkeet luodaan avainsanoilla *if*, *else* ja *elif*. Jos ensimmäinen *if*-lausekkeen ehto ei toteudu, ohjelma yrittää toista *elif*-lauseketta. *Elif*-lausekkeita voi olla useampi määrä. Jos mikään ehdoista ei toteudu, ajetaan viimeinen else-lausekkeen lohkon koodi.

```
numero = 3

if numero == 3:
    print("Numero on 3") # Muista sisentää neljällä välilyönnillä
elif numero != 3:
    print("Numero ei ole 3")
else:
    print("Ei kumpikaan")
```

Output:

```
Numero on 3
```

Silmukkarakenteet

Kun halutaan toistaa käskyjä useaan kertaan, on hyvä käyttää apuna silmukoita. Pythonissa on for- ja while-silmukoita. For-silmukoita käytetään silloin, kun halutaan suorittaa listan jokaiselle jäsenelle jokin operaation. Esimerkiksi, jos halutaan tulostaa kaikki listan jäsenet. (Matthes 2015, 49)

```
# Lista kolmella jäsenellä
lista = ["jäsen 1", "jäsen 2", "jäsen 3"]

# Tulosta listan jäsenet
for jasen in lista:
    print(jasen)
```

Output:

```
jäsen 1
jäsen 2
jäsen 3
```

For-silmukkaa voi käyttää myös silloin, kun ei haluta käsitellä listaa mutta halutaan suorittaa ennalta tiedetty määrä toistoja. Tässä tehtävässä auttaa range-luokka, jolle annetaan parametriksi toistojen määrä. Range-luokan avulla voidaan luoda numerosarjoja, joiden alku- ja loppukohta tiedetään. Range-luokalle annetaan parametrina haluttujen elementtien määrä tai aloitus- ja lopetuspisteet. (Matthes 2015, 57)

```
# For-silmukka listanelementtien tulostamiseksi
for elementti in range(3):
    print(elementti)
```

Output:

```
0
1
2
```

Jos suoritettujen toistojen määrää ei tiedetä ennalta, while-silmukka voi olla parempi vaihtoehto. While-silmukkaa toistetaan niin kauan kuin annettu ehto on totta. Pitää kuitenkin olla tarkkana, ettei vahingossa luo pysähtymätöntä silmukkaa. Silmukan sisään voi laittaa myös tarvittaessa *break*-käskyn, jolloin silmukka keskeytyy. (Matthes 2015, 118–121)

```
# Summataan lukuun 1, niin kauan kuin luku on alle 5.
luku = 0
while luku < 5:
    print(luku)
    luku = luku + 1
```

Output:

```
0
1
2
3
4
```

Funktiot

Matthes (2015, 129) kertoo funktioiden olevan koodilohkoja, jotka on suunniteltu yhtä tehtävää varten. Kun halutaan suorittaa tehtävä, jolle on määritetty funktio, kutsutaan funktiota sen nimen avulla. Funktioista on apua erityisesti, kun halutaan toistaa samaa tehtävää useaa kertaan. (Matthes 2015, 129)

Pythonissa funktiot määritellään *def*-avainsanalla. Sen jälkeen annetaan funktion nimi. Nimen jälkeisiin sulkeisiin tulee tarvittaessa funktion parametrit eli funktion suorittamiseen tarvittavat tiedot. Funktion määritelmä loppuu kaksoispisteeseen. Tästä alkaa sisennyksellä eroteltuna funktion runko eli itse funktiota kutsuttaessa suoritettavat käskyt. Tarvittaessa funktion palauttama tulos annetaan *return*-avainsanan avulla. (Matthes 2015, 130)

```
# Funktio kahden numeron summaamiseksi
def summausFunktio(numero1, numero2):
    tulos = numero1 + numero2
    return tulos

luku1 = 1
luku2 = 2

# Kutsutaan summausfunktiota
summa = summausFunktio(luku1, luku2)

# Tulosta summa
print(summa)
```

Output:

```
3
```

Moduulit

Pythonin standardikirjastossa on melko paljon hyödyllisiä funktioita valmiina käytettäväksi. Donaldsonin (2015, 2) mukaan Pythonin kutsutaan tulevan ”patterit mukana”, koska siinä on niin laaja valmis kirjasto (Donaldson 2015, 2). Joskus kuitenkin tarvitaan lisätoiminnallisuuksia apukirjastoista. Pythonissa kirjastoja kutsutaan moduuleiksi (*module*). Moduulit ovat Python-tiedostoja (.py-pääte), jotka sisältävät koodia, joka halutaan lisätä ohjelmaan (Matthes 2015, 150).

Moduulien lisääminen ohjelmaan tapahtuu *import*-avainsanan avulla. Sen jälkeen kerrotaan halutun moduulin nimi. Python lisää moduulin sisällön automaattisesti ohjelmaan. Moduulissa olevia funktioita tai luokkia kutsutaan moduulin nimen kautta: *moduuli.funktio*. Lisättävät moduulit laitetaan ohjelman alkuun.
(Matthes 2015, 150)

```
import math # Matematiikka moduuli

tulos = math.sqrt(4) # neliöjuuri-funktio

print(tulos)
```

Output:
2.0

Koko moduulia ei tarvitse lisätä ohjelmaa, jos tarvitsee vain tiettyjä funktioita. Yksittäisten funktioiden lisääminen tapahtuu avainsanalla *from* ja nimeämällä moduulin nimi. Funktioita voi lisätä pilkulla erotettuna niin paljon kuin haluaa. (Matthes 2015, 152)

```
from math import sqrt, pow # Lisätään math-moduulista sqrt-funktio ja pow-funktio

neliojuuri = sqrt(4) # neliöjuuri-funktio

toinen_potenssi = pow(neliojuuri, 2) # potenssi-funktio

print(toinen_potenssi)
```

Output:
4.0

Jos moduulista lisättävän funktion nimi on sama kuin ohjelmassa jo olevan funktion nimi, tai jos nimi on liian pitkä, voidaan funktiolle antaa ”peitenimi” *as*-avainsanan avulla. Myös moduuleille voidaan tarvittaessa antaa samalla tapaa uusi nimi. Tämän jälkeen funktiota kutsutaan peitenimeä käyttäen. (Matthes 2015, 152)

```
# Nimetään pow-funktio uudelleen peitenimen avulla
from math import pow as potenssi

# Korotetaan 4 2. potenssiin
tulos = potenssi(4, 2)

# Tulosta laskun tulos
print(tulos)
```

Output:
16.0

Paketit

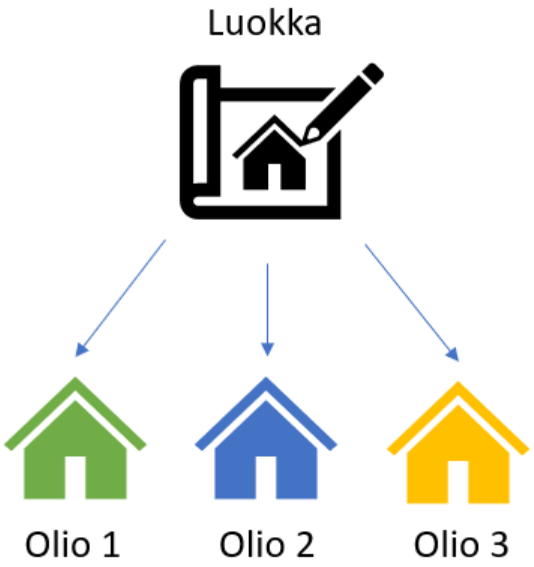
Chunin (2007, 530) mukaan tiedostoa, joka sisältää useita moduuleja kutsutaan Pythonissa paketiksi (*package*). Paketit mahdollistavat useiden moduulien niputtamisen yhteen helpottaen niiden jakamista. Paketit voivat sisältää myös alemman tason paketteja (*subpackage*). Paketin rakenne on hierarkkinen ja vaatii mukaan tiedoston, jonka nimi on *__init__.py*, mikä kertoo Python-tulkille tiedoston olevan paketti. Init-tiedoston ei tarvitse sisältää koodia mutta se pitää olla mukana. Paketteja kutsutaan usein myös kirjastoiksi. Paketin lisääminen tiedostoon tapahtuu myös *import*-komennolla. (Chun 2007, 530–531)

3.2 Olio-ohjelmointi

Matthesin (2015, 157) mukaan olio-ohjelmointi (*object-oriented programming*) on yksi tehokkaimmista tavoista kirjoittaa tietokoneohjelmia. Olio-ohjelmoinnissa luodaan luokkia (*class*), jotka edustavat oikean maailman asioita ja tilanteita. Luokkista luodaan olioita (*object*), jotka jakavat samat ominaisuudet (Matthes 2015, 150).

Luokat

Luokkia voisi kuvailla talon pohjapiirustuksiksi ja niistä luotuja olioita rakennetuiksi taloiksi (kuvio 2). Samasta pohjapiirustuksesta voi rakentaa useita samanlaisia taloja mutta jokainen talo on kuitenkin erillinen yksilö erilaisine ominaisuuksineen (*attribute*). Luokat mahdollistavat useiden ominaisuuksien (muuttujien) ja käytöksen (funktioiden) niputtamisen yhteen, jolloin asioita voidaan mallintaa ja monistaa helposti. Esimerkiksi taloissa olevien ikkunoiden ja ovien määrä, seinien väri tai huoneiden pinta-ala ovat ominaisuuksia, joita taloluokka voisi sisältää. (Matthes 2015, 157)



KUVIO 16. Olio-ohjelmoinnin periaate

Pythonissa luokat määritetään *class*-avainsanalla, jonka jälkeen tulee luokan nimi. Luokan määrittely tehdään sisennyksen avulla, kuten funktioita luodessa. Tässä määritetään luokan ominaisuudet, sisäiset muuttujat sekä metodit eli luokkaan liittyvät funktiot. Metodeihin pätevät samat asiat kuin normaaleihin funktioihin paitsi, että kutsutaan olion kautta: *olio.metodi()*. (Matthes 2015, 157–158)

Erikoismetodia __init__() kutsutaan aina, kun luodaan luokasta uusi olio. *Self*-avainsana vaaditaan metodin määritykseen ensimmäiselle parametrin paikalle. Se viittaa olioon itseensä ja sen kautta päästään käsiksi luokan ominaisuuksiin ja

metodeihin. Esimerkiksi *self.nimi = uusi_nimi* muuttaa olion nimiattribuutin. Metodille voi antaa tarvittaessa muita parametrejä samalla tavalla kuin funktioille. (Matthes 2015, 159)

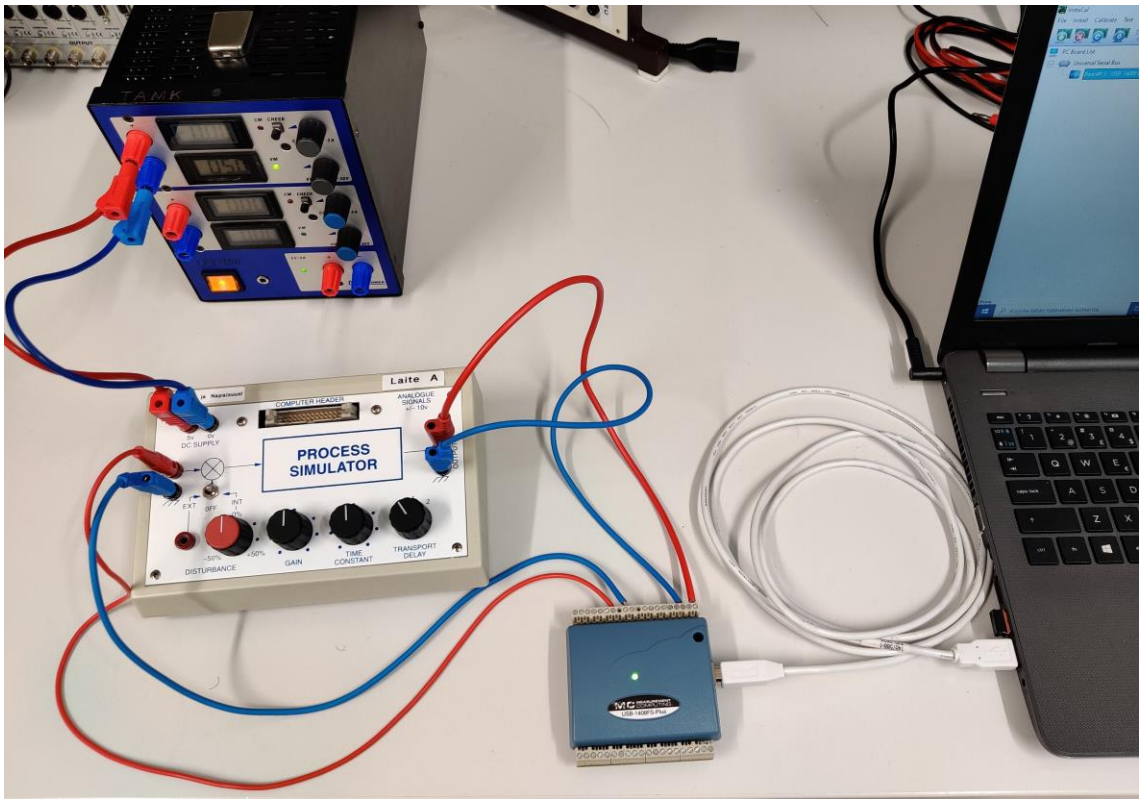
```
# Taloluokan määritys
class talo:
    def __init__(self, ikkuna_lkm, ovi_lkm, vari, pinta_ala):
        self.ikkunat = ikkuna_lkm
        self.ovet = ovi_lkm
        self.seinien_vari = vari
        self.pinta_ala = pinta_ala # m^2

    def TalonTiedot(self):
        print("Ikkunat: ", self.ikkunat)
        print("Ovet: ", self.ovet)
        print("Seinien väri: ", self.seinien_vari)
        print("Pinta-ala: ", self.pinta_ala)

# Luodaan talo-olio, jossa on 5 ikkunaa, 2 ovea, vihreät seinät ja pinta-ala 100
m2.
talo1 = talo(5, 2, "vihreä", 100)

# Tulostetaan talon tiedot metodin avulla
talo1.TalonTiedot()
```

Output:

```
Ikkunat: 5
Ovet: 2
Seinien väri: vihreä
Pinta-ala: 100
```

4 LABORATORIOTYÖN LAITTEISTO

PID-säätimen toteuttamiseksi tarvitaan tietokone sekä I/O-kortti A/D- ja D/A-muunnosten tekemiseksi. Jotta säätimen toimintaa voitaisiin sen toteuttamisen ohessa testata, tarvitaan jokin testiprosessi. Prosessisimulaattori sopii tähän tarkoitukseen hyvin. Siitä kerrotaan lisää myöhemmin tässä luvussa. Kuvassa 2 näkyy laboratoriotyöhön tarvittava laitteisto kokonaisuudessaan. Laboratoriotyö on suunniteltu toteutettavaksi laboratorion pöydällä.



KUVA 1. PID-säätimen laitteisto

4.1 USB-1408FS-Plus I/O-kortti

PID-säätimen tulo- ja lähtökorttina mittaussignaalin lukemiseksi sekä ohjaussignaalin lähettämiseksi käytetään Measurement Computingin USB-1408FS-Plus-tiedonkeruukorttia (kuva 2). Sen voi kuvitella olevan tietynlainen rajapinta digitaalisen ja analogisen maailman välillä. Kortti toimii samaan aikaan tiedon välittäjänä prosessin ja tietokoneen välillä sekä toimilaitteena, jolla ohjataan prosessia.

KUVA 2. USB-1408FS-Plus I/O-kortti

Näytteistys

Käyttäjäohjeen (2014, 9) mukaan kortilla voidaan ottaa analogisia näytteitä kahdessa eri tilassa. Ohjelman tahtiin käyvässä tilassa (*software paced*) suoritetaan A/D-muunnos, kun tietokoneen ohjelma lähettää käskyn. Analoginen mittaus muutetaan digitaaliseen muotoon ja lähetetään tietokoneelle USB-protokollan avulla. Näytteistystaajuus riippuu käytetystä ohjelmasta. Toinen tila on laitteiston tahtiin käyvä (*hardware paced*), jossa kortti suorittaa 32 mittausta kahdeksasta kanavasta samanaikaisesti ja lähettää näytteet tietokoneelle puskuriin. Tässä tilassa skannataan kanavia niin kauan, kunnes käyttäjä lopettaa mittauksen. Näytteenottotaajuus riippuu skannattavien kanavien määrästä (taulukko 3). (USB-1408FS-Plus käyttäjänohje 2014, 9). Koska tarkoituksena on suorittaa mittauksia PID-säätimen tahtiin, ensimmäinen tila käy paremmin tähän käyttötarkoitukseen.

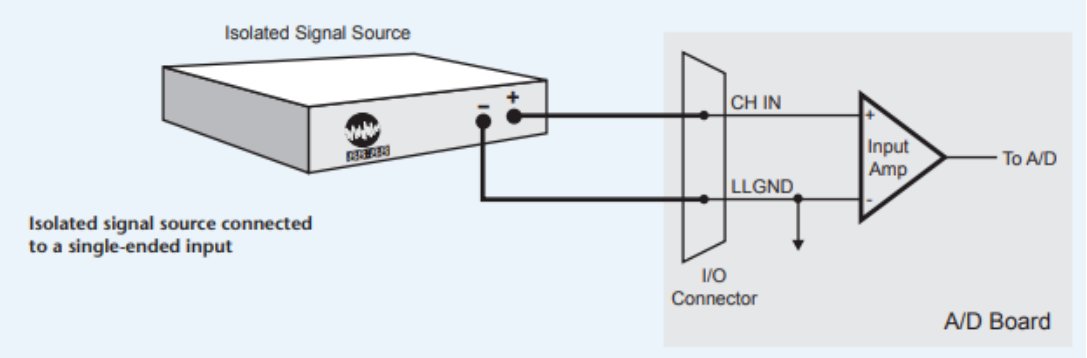
TAULUKKO 3. Näytteistystaajuuden riippuvuus käytetyistä kanavista (USB-1408FS-Plus käyttäjänohje 2014, 9. muokattu).

Kanavien määrä	Näytteenottotaajuus (kS/s)
1	48
2	24
3	16
4	12
5	9.60
6	8
7	6.85
8	6

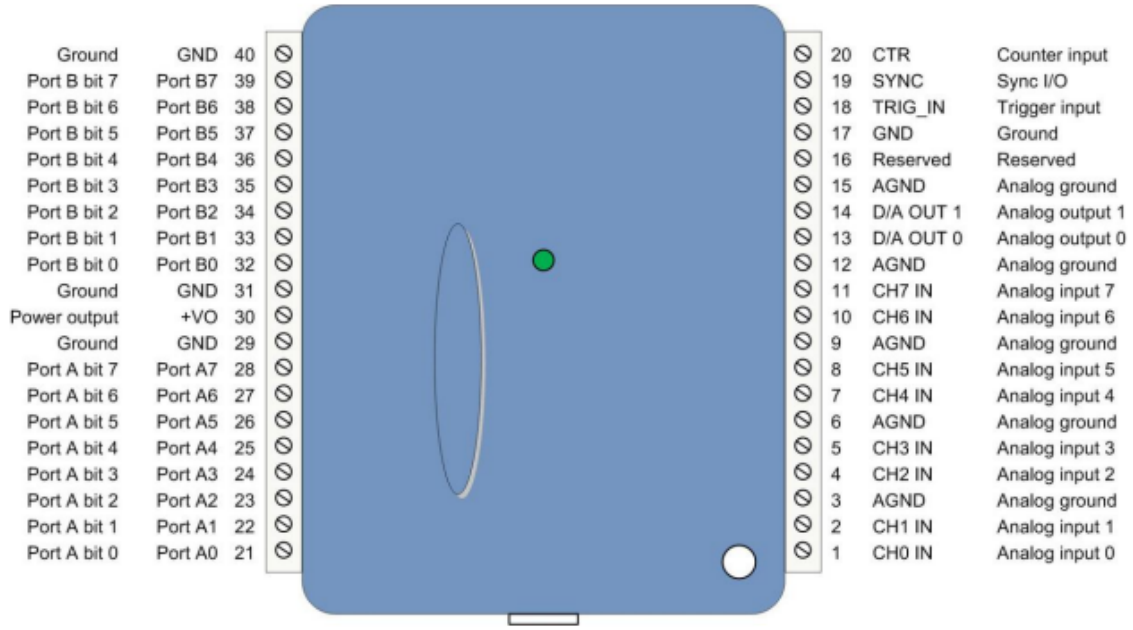
Analogiset tulokanavat

Kortissa on 8 analogista tulokanavaa ja 2 analogista lähtökanavaa. Lisäksi kortissa on 16 digitaalista tulo- ja lähtökanavaa. Analogisten tulokanavien jännitealue on maksimissaan ± 20 V ja analogisten lähtökanavien jännitealue on 0–5 V. Analogiset kanavat ovat pinneissä 1–20. (USB-1408FS-Plus käyttäjänohje 2014, 10–14).

A/D-muunnoksen suorittamiseksi analoginen tulokanava voidaan kytkeä mittauskohteeseen joko kahden johdon kytkennällä (*single-ended-configuration*) tai kolmen johdon kytkennällä (*differential-configuration*). Kahden johdon kytkennässä mitattava signaali kytketään tulokanavaan ja signaalin maa kytketään analogiseen maan pinniin (GND) (kuva 4). Tässä konfiguraatiossa tulosignaalin jännitealue on ± 10 V ja A/D-muunnoksen resoluutio on 13 bittiä. Koska tarvittavia johtoja per kytkentä on vain kaksi, tämä mahdollistaa kahdeksan samanaikaisen mittauksen. (USB-1408FS-Plus käyttäjänohje 2014, 12).

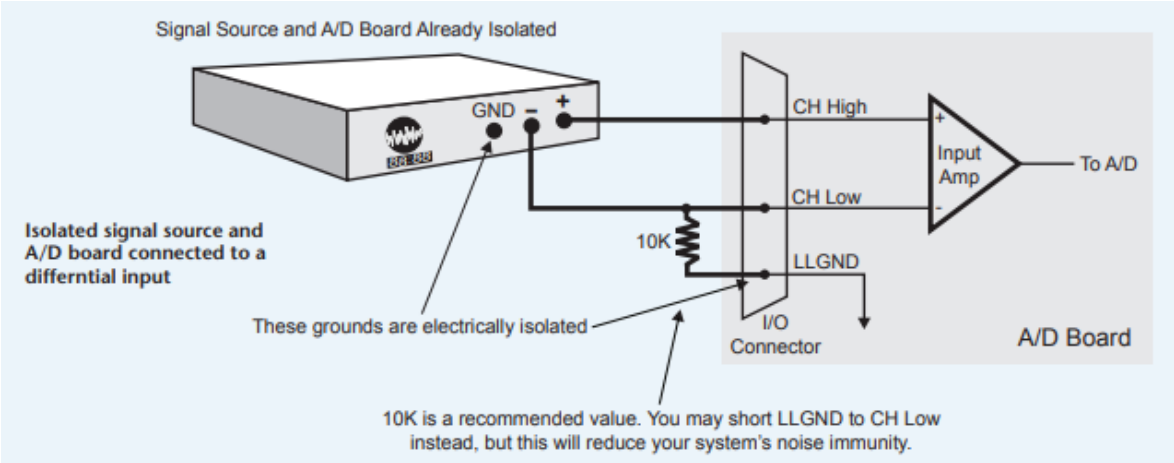


KUVA 3. Single-ended-kytkentä (Guide to DAQ Signal Connections 2012, 11)

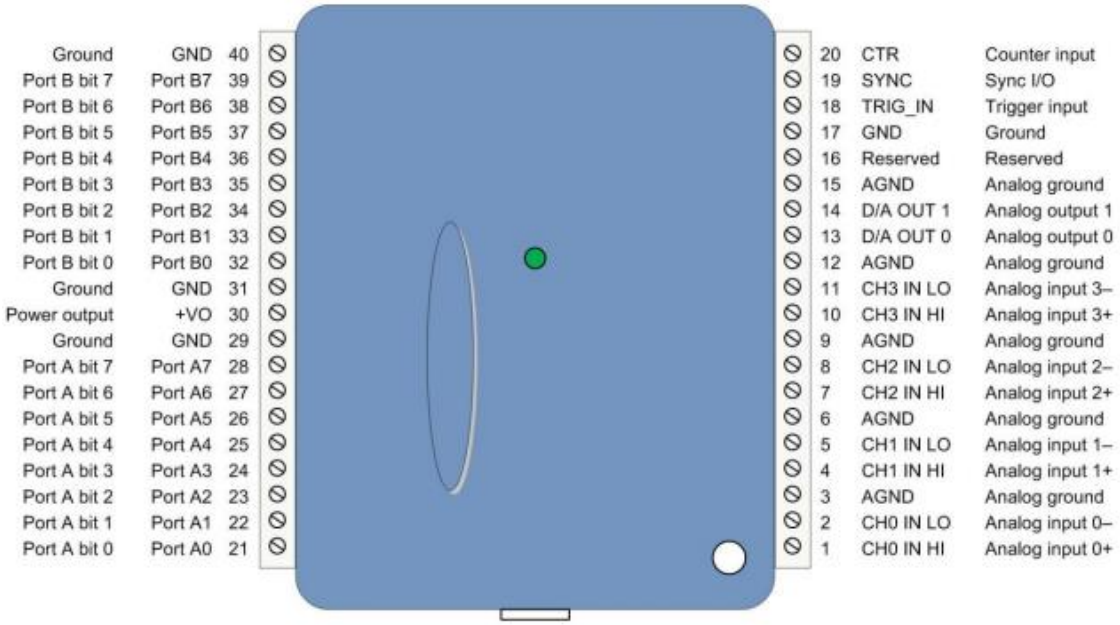


KUVA 4. Kahden johdon kytkennän pinnit (USB-1408FS-Plus käyttäjänohje 2014, 11)

Kolmen johdon kytkennässä mitattavaa signaalia verrataan referenssi signaaliin. Kuvasta 5 nähdään, että mitattava signaali kytketään HI-kanavaan, referenssi-signaali LO-kanavaan ja signaalin maa GND-pinniin (Guide to DAQ Signal Connections 2012, 11). Tällä kytkennällä mitattavat jännitealueet voivat olla ±20 V, ±10 V, ±5 V, ±4 V, ±2.5 V, ±2 V, ±1.25 V ja ±1 V signaaleista riippuen ja muunnoksen resoluutio on 14 bittiä. Yksi lisäjohto kuitenkin tarkoittaa, että tällä kytkennällä samanaikaisia mittauksia voi olla maksimissaan neljä. (USB-1408FS-Plus käyttäjänohje 2014, 12).



KUVA 5. Differential-kytkentä (Guide to DAQ Signal Connections 2012, 11).



KUVA 6. Kolmen johdon kytkennän pinnit (USB-1408FS-Plus käyttäjänohje 2014, 11).

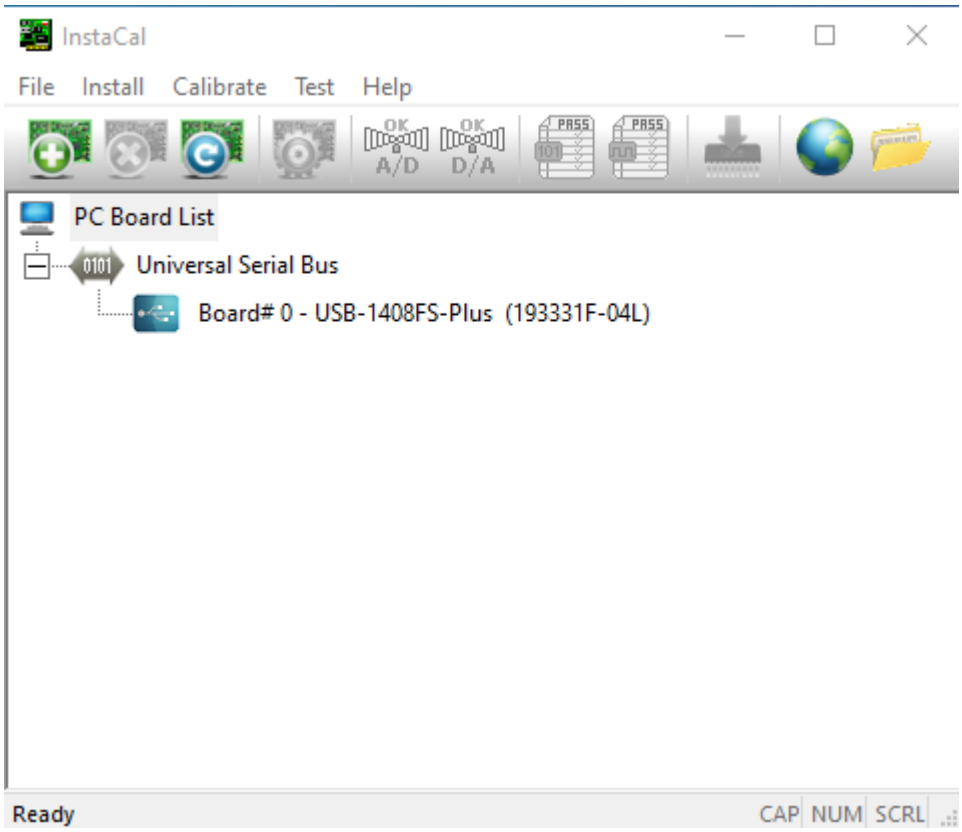
Analogiset lähtökanavat

Kortissa on kaksi analogista lähtökanavaa pinneissä 13 ja 14 (kuva 4; kuva 6). Niiden jännitealue on 0–5 V ja resoluutio 12 bittiä. Molemmilla kanavilla voidaan kirjoittaa 50 tuhatta näytettä sekunnissa. Analogiset lähtökanavat kytketään kahden johdon avulla. (USB-1408FS-Plus käyttäjänohje 2014, 14).

Kortin testaaminen

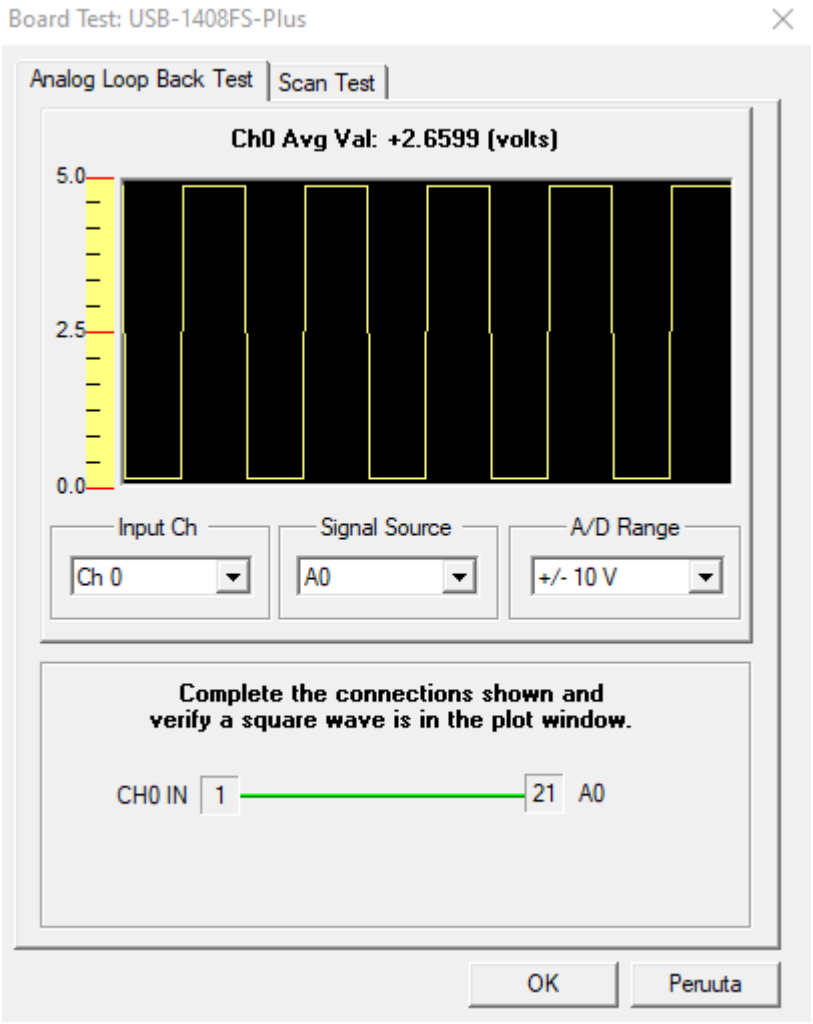
I/O-kortin toiminnan tarkistamiseksi tarvittiin Measurement Computingin InstaCal-ohjelmaa. InstaCal on tiedonkeruulaitteiden asennukseen, konfigurointiin ja testaamiseen tarkoitettu ohjelma (Measurement Computing n.d.). Ohjelman avulla voidaan testata kortin tulo- ja lähtökanavien toiminta sekä hallita eri laitteiden asetuksia.

InstaCal löytää automaattisesti tietokoneeseen kytketyt tiedonkeruulaitteet ja listaa ne käyttäjän nähtäväksi (kuva 7). Löydetyt laitteet numeroidaan omilla numeroilla, jotta ne erottuisivat toisistaan. Laitteen numero on tärkeä tietää, jotta luetaan tieto oikeasta laitteesta.



KUVA 7. InstaCalin laitteiden listaus

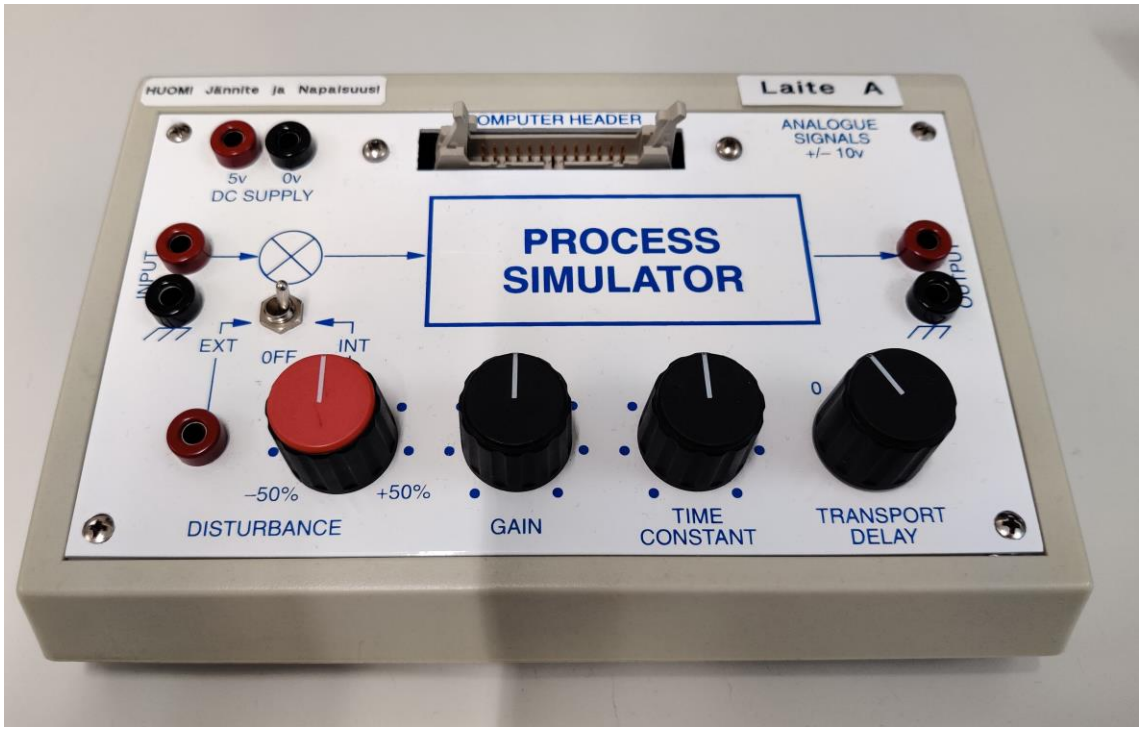
InstaCalilla voidaan myös testata tiedonkeruulaitteen analogisten tulokanavien toiminta. Kytkemällä kortin tulokanava 1 kanavaan 21 ja lähettämällä toisesta kanavasta kanttiaaltoa, voidaan varmistaa, että kanava toimii (kuva 8). Testaus on hyvä suorittaa aina ennen kortin käyttöä. Samalla varmistutaan myös siitä, että on valittu oikea kanava.



KUVA 8. Kortin tulokanavan toiminnan testaus

4.2 Prosessisimulaattori

Säätimen testaamiseksi käytetään analogista prosessisimulaattoria (kuva 9). Simulaattorilla pystytään simuloimaan ensimmäisen kertaluvun viiveellistä prosessia (4). Prosessin ominaisuuksia, kuten vahvistusta K, aikavakiota τ ja viivettä θ, pystytään muuttamaan laitteen paneelissa olevista nupeista. Lisäksi prosessiin voi aiheuttaa kuormahäiriön punaisesta nupista. Laite vaatii toimiakseen 5 VDC käyttöjännitteen.

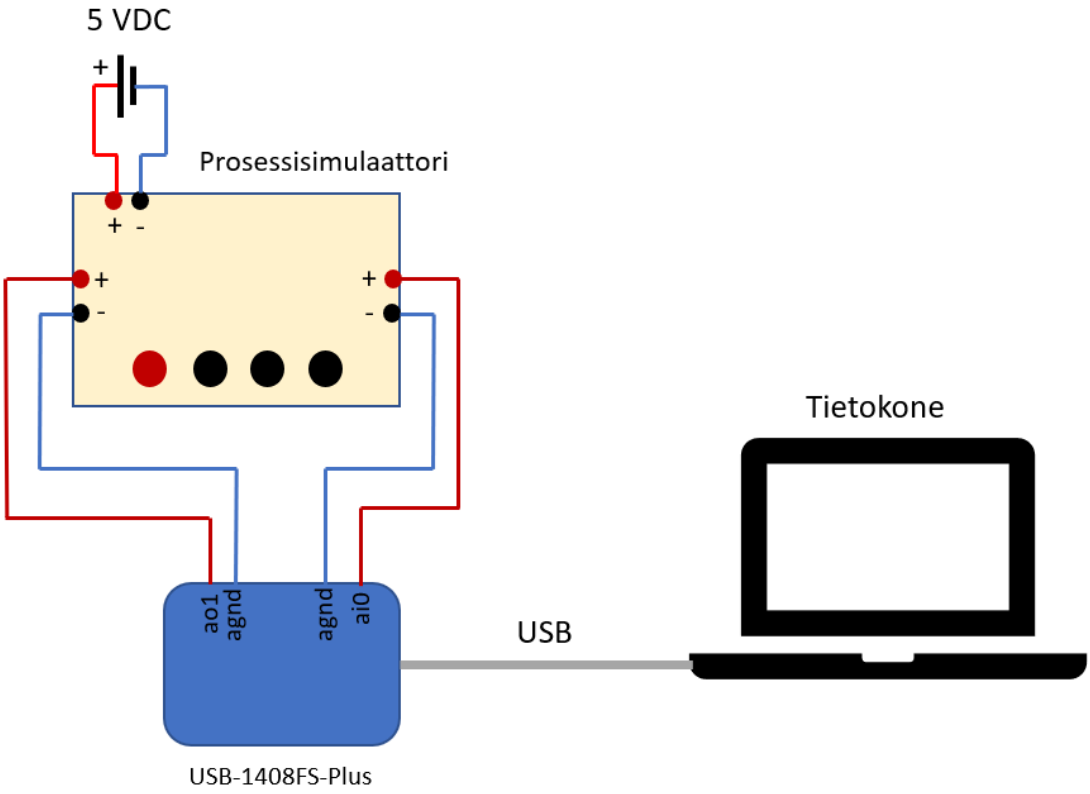


KUVA 9. Prosessisimulaattori

4.3 Laitteiston kytkentä

Tietokoneen, I/O-kortin ja prosessisimulaattorin avulla saadaan luotua kuvion 1 (luku 2.1, s. 9) esittämä säätöpiiri. Tietokoneella pyörivä ohjelma toimii säätimenä, I/O-kortti on toimilaite ja prosessia simuloi prosessisimulaattori. Oikeassa säätöpiirissä I/O-kortin jännitesignaali menisi toimilaitteelle, joka puolestaan aiheuttaisi muutoksen prosessissa. Laboratoriotyössä riittää, että I/O-kortin jännitesignaalin muutos aiheuttaa prosessiin muutoksen. Laitteiston kytkentä on melko yksinkertainen ja sopii siten hyvin laboratoriossa tunnin aikana tehtäväksi.

I/O-kortti kytketään USB-kaapelilla tietokoneeseen (kuvio 17). I/O-kortin analoginen lähtökanava pinni 14 (analog output 1) kytketään prosessisimulaattorin tulokanavaan. Tulokanavan maa kytketään I/O-kortin analogisen maan pinniin 12 (agnd). Prosessisimulaattorin positiivinen lähtösignaali kytketään I/O-kortin tulokanavaan 0 (analog input) ja negatiivinen signaali I/O-kortin maahan pinniin 3 (agnd).



KUVIO 17. Laboratoriotyön laitteiston kytkentä

5 SÄÄTIMEN TOTEUTUS PYTHONILLA

PID-säätimen toteutus tehdään Pythonilla edellisessä kappaleessa esitellyllä laitteistolla. Lopullisessa laboratoriotyössä opiskelija luo PID-säätimen koodin vaihe vaiheelta itse. Opiskelijan ei tarvitse osata kaikkia Pythonin ominaisuuksia saadakseen laboratoriotyön tehtyä. Ohjelmoinnin perusteiden ymmärtämisestä on kuitenkin luonnollisesti apua. Esitelty toteutus toimii vain esimerkkinä. Toteutustapoja on totta kai monia muitakin.

5.1 Mcculw-paketti

Universal Library (UL) on Measurement Computingin ohjelmistokirjasto tiedonkeruutuotteilleen. Kirjasto mahdollistaa ohjelmien vuorovaikutuksen tiedonkeruulaitteiden kanssa kuten A/D-muunnoksen arvon lukemisen. Kirjastolle on tehty Pythonille API eli ohjelmointirajapinta, mikä mahdollistaa Python-ohjelmalla tiedon hakemisen laitteelta tai sen lähettämisen laitteelle. Rajapinta saadaan käyttöön asentamalla koneelle *mcculw*-paketti. Toimiakseen rajapinta vaatii myös InstaCal-ohjelman asentamisen tietokoneelle. (Measurement Computing 2020)

Paketti sisältää lukusia muita funktioita tiedonkeruulaitteen kanssa työskentelemiseen. Laitetta konfiguroidessa InstaCal-ohjelmaa ei tarvitse edes välttämättä avata, Python rajapinnan kautta voi tehdä useimmat toiminnot. Laboratoriotyön kannalta kuitenkin vain A/D- ja D/A-muunnokset ovat oleellisia.

A/D-muunnoksen arvon lukeminen I/O-kortilta

I/O-kortilla voidaan suorittaa A/D-muunnos ja lukea muunnoksen ohjelmaan UL-olion v_in()-metodin avulla. Metodi muuttaa digitaalisen luvun automaattisesti volteiksi, jotta sitä on helpompi ymmärtää. Metodi tarvitsee parametreikseen käyttäjän InstaCalissa määrittämän I/O-kortin numeron, analogisen tulokanavan sekä tulokanavan käytetyn jännitealueen. Parametrit tulee antaa tässä järjestyksessä. Jännitealue annetaan Universal libraryn ULRange-luokan attribuutin *ULRange.BIP10VOLTS* avulla, koska jännitealue on ±10 Volttia. (Measurement Computing 2020)

```
from mcculw import ul # UL-olio I/O-kortin käsittelemiseksi
from mcculw.enums import ULRange # I/O-kortin jännitealueet

board_num = 0 # I/O-kortin järjestysnumero
channel = 1 # analoginen tulokanava
ai_range = ULRange.BIP10VOLTS # analogisen tulokanavan jännitealue

# Luetaan I/O-kortilta analogisen signaalin jännitteen arvo
ai_arvo = ul.v_in(board_num, channel, ai_range)

# Tulostetaan arvo konsolille
print(ai_arvo)
```

D/A-muunnoksen tekeminen I/O-kortilla

Jännitearvon kirjoittaminen ja D/A-muunnoksen tekeminen I/O-kortilla tapahtuu hyvin samanlaisella koodilla kuin A/D-muunnos. Erona on vain se, että tällä kertaa lähetetään tietoa. Lähettäminen tapahtuu ul-olion v_out-metodin avulla. Metodi tarvitsee parametreikseen käyttäjän InstaCalissa määrittämän I/O-kortin numeron, analogisen tulokanavan, tulokanavan käytetyn jännitealueen sekä lähetettävän jännitteen arvon. Parametrit pitää antaa tässä järjestyksessä. Koska käytetyn I/O-kortin analogisen lähtökanavan jännitealue on 0–5 volttia, jännitealueeksi laitetaan *ULRange.UNI5VOLTS*. (Measurement Computing 2020)

```
from mcculw import ul
from mcculw.enums import ULRange # I/O-kortin jännitealueet

board_num = 0 # I/O-kortin järjestysnumero
channel = 1 # analoginen lähtökanava
ao_range = ULRange.UNI5VOLTS # analogisen lähtökanavan jännitealue

# kolme volttia
data_value = 3

# Kirjoitetaan jännitteen arvo I/O-kortille ja suoritetaan D/A-muunnos
ul.v_out(board_num, channel, ao_range, data_value)
```

5.2 Säätimen koodi

PID-säädin on helppo toteuttaa luomalla sille oma luokkansa, jonka pohjalta voidaan luoda sille oma olionsa. Olion avulla on helppo pitää kirjaa PID-säätimen

sisäisten muuttujien tilasta, joita tarvitaan algoritmin toimintaa. PID-luokan _init_-metodilla alustetaan kaikki säätimen sisäiset muuttujat.

```
from mcculw import ul
from mcculw.enums import ULRange # I/O-kortin jännitealueet
import time

class PID:
    # Alustus funktio
    def __init__(self):
        # Säädin-olion kriittisten parametrien alustus
        self.board_num = 0 # I/O-kortin järjestysnumero
        self.ai_range = ULRange.BIP10VOLTS # analogisen tulokanavan jännitealue
        self.ao_range = ULRange.UNI5VOLTS # analogisen lähtökanavan jännitealue
        self.Kp = 2 # Vahvistuskerroin
        self.Ti = 5 # Integrointiaika
        self.Td = 0 # Derivointiaika
        self.bias = 0 # p-säädön vakio "manual reset"
        self.h = 0.1 # näytteistysaika
        self.b = 1 # asetusarvon suodatusparametri
        self.last_y = self.ai_read(ch=0) # Viime kierroksen mittaus
        self.I_term = 0 # Integroiva termi
        self.D_term = 0 # Derivoiva termi
        self.U_High = 5 # Toimilaitteen toimialueen yläarvo
        self.U_Low = 0 # Toimilaitteen toimialueen ala-arvo
        self.N = 5 # vakio N derivaatan suodatukselle
        self.Tf = self.Td / self.N  # derivaatan suodattimen aikavakio
        self.Taw = sqrt(self.Ti) # antiwindupin jakaja
        self.New_Kp = self.Kp # Uusi vahvistuskerroin
        self.New_b = self.b # Uusi asetusarvon suodatusparametri
        self.firstCall = True # onko kyseessä ensimmäinen metodin kutsuminen?
        self.NextCall = 0 # ajanhetki seuraavalle kutsumiskerralle
```

Mittauksen lukeminen I/O-kortilta ja ohjauksen kirjoittaminen laitetaan omiin metodeihinsa. Tämä ei ole tarpeen säätimen toiminnan kannalta mutta se selkeyttää säätimen algoritmin koodia hieman. Molemmat metodit kutsuvat UL-luokalla I/O-kortin rajapintaa.

```
    # Mittauksen lukeminen
    def ai_read(self, ch):
        mittaus = ul.v_in(self.board_num, ch, self.ai_range)
        return mittaus

    # Ohjauksen kirjoittaminen
    def ao_write(self, ch, value):
        ul.v_out(self.board_num, ch, self.ao_range, value)
```

Jokaisen laskentakierroksen välissä odotetaan näytteistysvälin pituinen aika. Näytteistysvälin tulisi pysyä melko vakiona riippumatta siitä, kuinka kauan algoritmin suorittamaan laskentaa kuluu aikaa. Jos ohjauksen laskentaan kuluvaa aikaa ei oteta huomioon, näytteistysväli on aina pitempi kuin *h*. Näytteistysvälin vakiona pitämiseen on monia eri toteutustapoja ja niillä on hyvät sekä huonot puolensa. (Wittenmark ym. 2002, 78–79).

Time-moduuli tarjoaa erilaisia työkaluja ja palveluja ajan käsittelemiseen Pyhton-ohjelmissa. Time-luokan time()-metodi palauttaa sen hetkisen ajan sekunteina. Sleep()-metodi tauottaa ohjelman suorituksen hetkeksi.

Tähän säätimen toteutukseen on valittu tapa, jossa ajoittaiset ylipitkät näytteistysvälit ovat sallittuja. Ylipitkän näytteistysvälin sattuessa säädin jatkaa eteenpäin yrittäen suorittaa algoritmi halutulla näytteistysvälillä. Tämä toiminta on saatu aikaan säätimen metodilla *sleepUntil*. Metodille annetaan ajanhetki parametrilla *nextCall*, mihin asti säädin odottaa ennen kuin se aloittaa uuden laskentakierroksen. Jos laskenta pitkittyy ja ajanhetki on jo mennyt, säädin ottaa sen hetkisen ajan talteen ja jatkaa algoritmin suorittamista normaalilla näytteistysvälillä *h*. Säädin yrittää siis parhaansa mukaan pysyä näytteistysintervallin tahdissa mutta ajoittaiset myöhästymiset sallitaan. Jos näytteistyksen toteutus jää epäselväksi, aihe on käsitelty laajemmin kirjallisuudessa. (Wittenmark ym. 2002, 79).

```
def sleepUntil(self, nextCall):
    # jos kutsumiskerran aika on jo mennyt, jatka eteenpäin ja alusta nextCall
    if (nextCall - time.time()) < 0:
        self.nextCall = time.time()
    else:
        # Muussa tapauksessa odota seuraavaa kutsumiskertaa
        time.sleep(nextCall - time.time())
```

Säätimen algoritmi laitetaan omaan metodiinsa. Aina, kun metodia kutsutaan, PID-säätimen algoritmi suoritetaan kerran. Metodille annetaan asetusarvo parametrilla r.

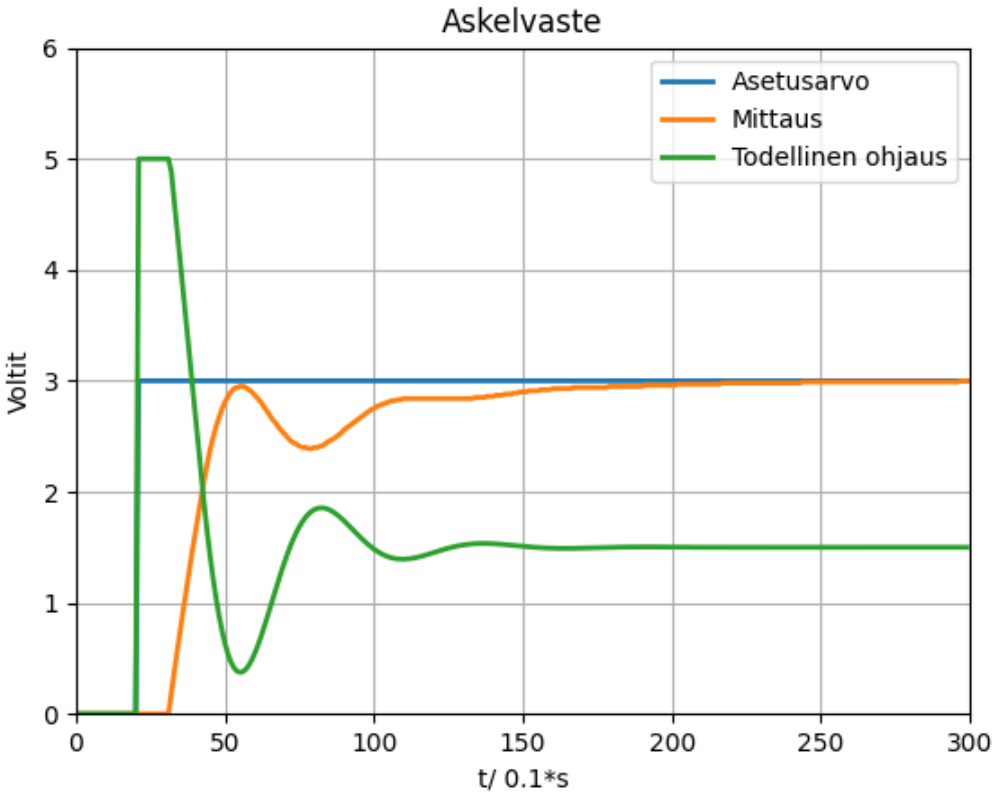
```
def LaskeOhjaus(self, r):
    # Luetaan mittaus I/O-kortilta
    y = self.ai_read(ch=0)

    # Parametrien vaihdon koodi
    if self.Kp != self.New_Kp:
       # Päivitetään I-termi parametrien muutoksella
       self.I_term = (self. I_term
                      + self.Kp*(self.b*r - y)
                      - self.New_Kp*(self.New_b*r - y))

       # Päivitetään parametrit uusiin
       self.Kp = self.New_Kp
       self.b = self.New_b

       # Lasketaan P-termi
       P_term = self.Kp*(self.b*r - y)

       # Lasketaan Derivoiva termi
       self.D_term = (self.Tf / (self.Tf + self.h) * self.D_term
                   - ((self.Kp*self.Td) / (self.Tf + self.h)) * (y - self.last_y))

       # Summataan P-, I- ja D-termit yhteen
       ua = P_term + self.I_term + self.D_term

       # Simuloidaan toimilaitteen saturaatio
       U = max(min(ua, self.U_High), self.U_Low)

       # Lähetetään ohjauksen arvo I/O-laitteelle
       self.ao_write(ch=1, value=U)

       # Lasketaan integroiva-termi
       self.I_term = (self.I_term
                      + (self.Kp / self.Ti) * (r - y) * self.h
                      + (1 / self.Taw) * (U - ua))

       # Otetaan mittaus talteen seuraava laskentakierrosta varten
       self.last_y = y

       # Metodin kutsunnan ajanhetken taltiointi vain ensimmäisellä kierroksella
       if self.firstCall == True:
           self.nextCall = time.time() # Otetaan aika talteen
           self.firstCall = False

       self.nextCall += 0.1 # lisätään näytteistysväli 0.1 s
       self.sleepUntil(self.nextCall) # Odotetaan seuraavaa kierrosta
```

Säätimen olio nimeltään *saadin* luodaan kutsumalla PID-luokan nimellä. Tämä suorittaa luokan __init__-metodin ja alustaa säätimelle muuttujat. Säätimen algoritmia kutsutaan while-silmukassa toistuvasti. Jokaisen kierroksen lopussa odotetaan 0.1 sekuntia ennen seuraavaa laskentakierrosta.

```
# Luodaan säätimen olio
saadin = PID()

# Silmukka PID-säätimen suorittamiseksi toistuvasti
while True:
    saadin.LaskeOhjaus(r=2) # Kutsutaan metodia
```

5.3 Säätimen testaaminen

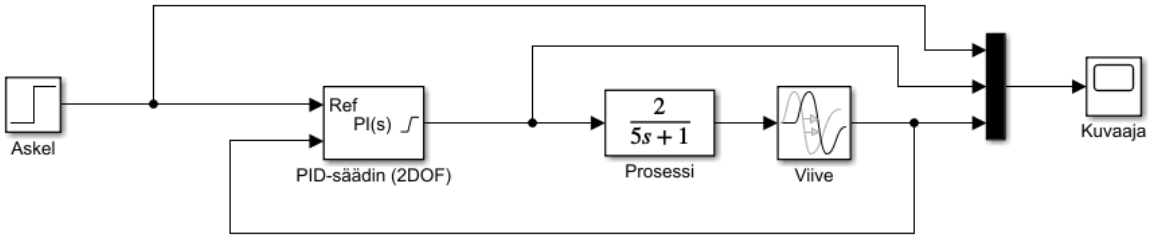
Säätimen toimintaa voidaan tarkastella askelvastekokeen avulla. Asetusarvoa on muutettu askelmaisesti nollasta kolmeen. Kuviosta 18 nähdään, että mittaus asettuu asetusarvoon kuten pitäisikin. Testi on tehty PI-säätimellä, jonka parametreinä oli K_p = 2 ja T_i = 5. Askelvaste on tehty matplotlib-moduulin avulla. Matplotlib on datan visualisointiin tehty kirjasto, jonka avulla voidaan piirtää monenlaisia kuvaajia.



KUVIO 18. Säätöpiirin askelvastekoe

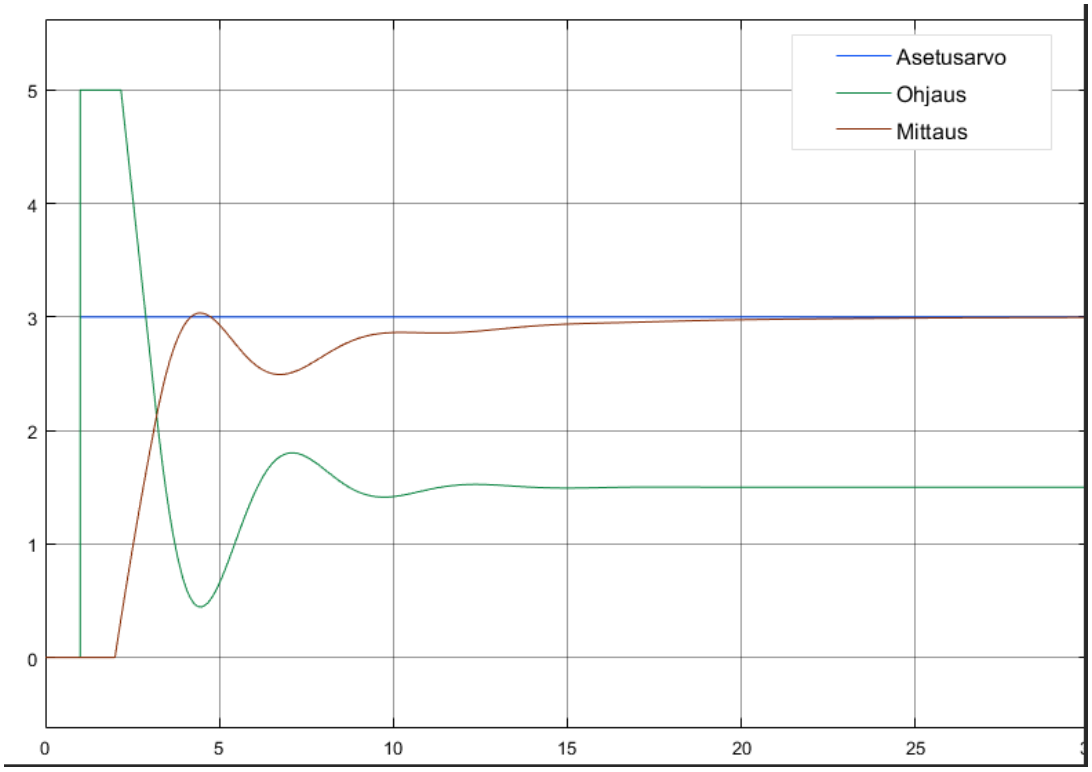
Jotta säätimen toiminta voitaisiin varmistaa, sitä pitää voida verrata johonkin toiseen säätimeen. Matlab Simulinkillä on simuloitu sama säätöpiiri (kuva 10). Säätöpiirissä käytettiin kahden vapausasteen PID-säädinlohkoa PI-säätimenä. Ohjaus oli rajoitettu alueelle 0–5 ja kerimiseneston kerroin T_{aw} oli 2,23. Prosessin siirtofunktio oli

$$F(s) = \frac{2}{5s+1} \cdot e^{-s}. \tag{20}$$



KUVA 10. Simulink-säätöpiiri

Kuviossa 19 näkyy Simulinkillä toteutetun säätöpiirin askelvaste. Pythonin ja Simulinkin askelvasteita verratessa nähdään, että ne ovat melkein identtisiä. Voidaan siis todeta, että Pythonilla luotu PID-säädin toimii niin kuin pitäisikin.



KUVIO 19. Matlab Simulink simulointi

6 LABORATORIOTYÖN TOTEUTUSSUUNNITELMA

Laboratoriotyön tavoitteena on, että opiskelija oppisi PID-säätimen sisäisestä toiminnasta ja ymmärtäisi digitaalisia säätimiä paremmin. Laboratoriotyö on suunniteltu sopivan osaksi automaatiotekniikan opetussuunnitelmaan kuuluvia laboratoriokursseja. Suunniteltua työtä ei ole testattu käytännössä, joten se tulee varmasti vielä muuttumaan hieman. Laboratoriotyölle on tehty alustava ohje (liite 1).

Ennakkotietovaatimukset

Laboratoriotyötä ei voi suunnitella ilman, että ottaa huomioon opiskelijan ennakkotiedot laboratoriotyön aiheesta. Minimivaatimuksena opiskelijan tulee olla käynyt säätötekniikan perusteet -kurssi sekä mittaustekniikan laboratoriotyöt -kurssi, jotta opiskelijalla olisi perusymmärrys säätötekniikasta ja laboratoriotyöskentelystä. Paljon muuta ei voi etukäteen vaatia. Ohjelmoinnin osaamisesta on luonnollisesti hyötyä.

Ennakkotehtävät

Laboratoriokursseilla on tapana antaa opiskelijalle ennakkotehtäviä aiheeseen liittyen ennen laboratorioon tulemista. Ennakkotehtävien avulla voidaan tutustuttaa opiskelija joihinkin työn aiheisiin tai tekniikkaan. Laboratorioaikaa on rajallisesti, joten se on parasta käyttää itse työhön. Esimerkiksi Pythonin syntaksiin tutustuminen on yksi ennakkotehtävä, jolloin työn aloittamiseen on matalampi kynnys. Jos opiskelija haluaa tehdä työn omalla tietokoneellaan, yksi ennakkotehtävä voisi olla InstaCalin ja Pythonin asentaminen sekä niihin tutustuminen. Lisäksi integraalin ja derivaatan määritelmien kertaaminen on yksi ennakkotehtävä, jotta säätimen termien kaavojen tulkinta helpottuisi.

Laboratorioharjoituksen vaiheet

Laboratoriotyön alussa tutustutaan ensiksi käytettävään laitteistoon ja ohjelmistoon. Opiskelija saa itse kytkeä laitteet ohjeiden mukaan ja sen jälkeen testata niiden toiminta. I/O-kortti voidaan testata ja kalibroida InstaCal-ohjelmalla. Samalla on hyvä varmistaa, että kortti on numeroitu oikein, jotta sitä voidaan käyttää Pythonilla rajapinnan kautta. Kun laitteiston toiminnasta on varmistuttu, on aika siirtyä ohjelmointiin.

Ensimmäisenä tehtävänä on luoda ohjelma, joka lukee jännitteen arvon A/D-muunnoksen avulla I/O-kortilta. Jännitelähde asetetaan pariin Volttiin ja kortti kytketään lukemaan sen arvon. Jännitteen arvo tulostetaan konsolille tai vaihtoehtoisesti piirretään kuvaajalle. Jännitteen arvoa voidaan lukea myös jatkuvasti silmukassa, jolloin säädettävän jännitelähteen arvoa voidaan myös muuttaa ja muutos näkyy reaaliajassa tietokoneella.

Toinen tehtävä on jännitteen lähettäminen I/O-kortille Pythonilla. Jotta kirjoitetun jännitteen arvon oikeellisuudesta voitaisiin olla varmoja, jännite mitataan varmuuden vuoksi yleismittarilla. Jännitteen arvoa voidaan muuttaa myös reaaliaikaisesti silmukassa.

Kun I/O-kortin kanssa kommunikoiminen onnistuu opiskelijalta, voidaan lisätä kytkentään prosessisimulaattori ja aloittaa säätöpiirin tekeminen. Kolmas tehtävä on prosessi simulaattorin askelvasteen mittaaminen. I/O-kortilta voidaan kirjoittaa askelmainen muutos tulojännitteeseen ja mitata samalla simulaattorin lähtöjännite. Jos tulo- ja lähtösignaalit piirretään kuvaajalle, saadaan aikaan prosessin askelvaste. Askelvasteesta voi loppuraportissa määrittää prosessin parametrit.

Opiskelija osaa nyt lukea mittauksen arvon ja kirjoittaa ohjauksen. Nyt jäljelle jää enää itse säätimen algoritmin kirjoittaminen. On hyvä aloittaa yksinkertaisesta P-säädöstä. Säädön toimintaa voidaan testata eri vahvistuskertoimilla ja vakiotermillä.

Kun P-säätö on saatu toimimaan, lisätään säätimeen integroiva-termi. Integrointia tehtäessä on hyvä varmistaa, että kaava on oikein, koska siinä tapahtuu helposti virheitä. Integroinnin toiminta voidaan jälleen varmistaa askelvaste kokeella. Vasteesta nähdään helposti, toimiiko integrointi vai ei. Tässä vaiheessa on hyvä testata tilanne, jossa asetusarvo nostetaan tasolle, jota prosessi ei voi saavuttaa. Tässä tilanteessa toimilaite kyllästyy ja integroiva termi alkaa kerimään eli kasvamaan liian suureksi. Tällä tavalla opiskelija oppii kerimisen ongelmasta. Nyt integroivaan termiin voidaan lisätä kerimisen esto ja suorittaa sama koe uudestaan.

Jäljellä on enää derivoinnin lisääminen säätimeen. Derivoinnissa käytetään mittauksen derivaattaa. Säätimeen pitää lisätä muuttuja, joka pitää kirjaa edellisen laskentakierroksen mittauksesta. Tämän vaiheen ymmärtämiseksi on hyvä, jos opiskelija kertaa derivointia (ja integrointi) ennen laboratorioon tulemista. Säädön toiminta on testattava jälleen askelvastekokeella.

Lopuksi, jos aikaa laboratorioaikaa jää, säätimeen lisätään automaattitilan lisäksi manuaalitila. Nyt opiskelija testaa säätimen tilojen vaihtamista sen ollessa käynnissä ja siten näkee, mitä tapahtuu, kun vaihdetaan automaatista manuaalille ja toisinpäin. Sen jälkeen säätimeen lisätään tilan vaihtamissa tapahtuvan ohjauksen hypyn estävä koodi. Vaihtoehtoisesti manuaaliohjaus voidaan tehdä jo ennen P-säädön toteuttamista.

Laboratoriotyön raportointi

Laboratoriotyön raportoinnissa noudatetaan TAMK:n normaaleja raportointikäytäntöjä. Raportissa käydään läpi laboratoriotyön tehtävät ja vastataan lisäkysymyksiin niihin liittyen. Siinä on tyypillistä dokumentoida kaikki työhön liittyvä laitteisto ja ohjelmisto, joten opiskelijan kannattaa ottaa kuvia ollessaan labrassa. Samoin jokaisen askelvastekokeen kuvaajat tulee hyvä ottaa talteen. Raportteihin sisältyy myös usein jonkun verran omaa pohdintaa työn sisällöstä. Se antaa opiskelijalle mahdollisuuden pohtia ja analysoida tekemäänsä.

7 YHTEENVETO JA JOHTOPÄÄTÖKSET

Opinnäytetyön tavoitteena oli luoda laboratoriotyö PID-säätimen ohjelmallisesta toteutuksesta Tampereen Ammattikorkeakoululle sähkö- ja automaatiotekniikan tutkinto-ohjelman opetuskäyttöön. Työn tarkoituksena oli luoda automaatiotekniikan opetukseen mahdollisuus opettaa PID-säätimen ohjelmointia käytännössä.

Teoriaosuudessa perehdyttiin PID-säätimen toimintaan sekä miten säätimen algoritmi toteutetaan tietokoneella. Sen jälkeen käytiin läpi Pythonin syntaksi, säätimen toteutuksen vaatima laitteisto sekä esimerkki säätimen Python-koodi. Lopuksi esiteltiin laboratoriotyön alustava toteutussuunnitelma.

Esitetty laboratoriotyön suunnitelma ei ole sen viimeinen versio, vaan ainoastaan lähtökohta harjoituksen kehittämiselle. Laboratoriotöillä on tyypillisesti tapana muuttua ajan myötä. Jokaisen läpiviennin myötä opitaan jotain uutta harjoituksesta ja löydetään kehityskohteita. Esimerkiksi alkuperäinen ohjeistus saattaa olla liian epäselvä ja sitä voidaan joutua täydentämään. Myös laitteisto, jolla laboratoriotyö on toteutettu, saattaa muuttua, jos löydetään parempia vaihtoehtoja. Työ ei ole onneksi sidottu käytettyyn laitteistoon. Sen voisi toteuttaa myös muun valmistajan I/O-kortilla tai vaihtamalla Pythonin toiseen ohjelmointikieleen. Opiskelijoiden palaute on tärkeää opetuksen kehittämisessä, sillä laboratoriotyön luoja on usein sokea virheilleen.

Laboratoriotyötä jätettiin pedagoginen puoli pois, jottei työn pituus kasvaisi liian suureksi. Opetuksen pohdinta perustuu enemmänkin omiin kokemuksiin opiskelijana laboratoriotöistä. Aiheen opetuksen pohdinta ja eteenpäin vieminen jää opettajien tehtäväksi. Opettajien tulee seuraavaksi tutustua laboratoriotyön ohjeeseen ja kokeilla tehdä työ itse. Lisäksi tarvittavien ohjelmien asentaminen ja toimimaan saaminen on haaste itsessänsä ainakin ensikertalaiselle.

Teoria lukujen haasteena oli laajan aihealueen tiivistäminen siten, että mitään olennaista ei jäisi puuttumaan ja kokonaisuus olisi kuitenkin aiheeseen perehtymättömälle lukijalle ymmärrettävä. Pelkistä PID-säätimistäkin on jo kirjoitettu pitkiä kirjoja. Aihe on melko matemaattinen, mikä tekee joidenkin säätimen toimintojen selittämisestä lukijalle ymmärrettävällä ja intuitiivisella tavalla haastavaa.

Käyttöliittymän puute säädintä testatessa oli yksi päänvaivaa aiheuttanut asia laboratoriotyötä suunnitellessa. Toimivan säätimen ohjelmoiminen ei ole ongelma mutta askelvastekokeiden ja säätimen tilan vaihtamisen tai parametrien muuttamisen vaikutuksen testaaminen ilman käyttöliittymää on hieman hankalaa. Testien tekeminen on mahdollista, jos on kokenut Python-ohjelmoija, mutta se tuotti haasteita laboratoriontyötä suunnitellessa, sillä opiskelija on ensikertaa Pythonin kanssa tekemisissä. Ratkaisuna ongelmaan voisi olla valmiin käyttöliittymän luominen laboratoriotyötä varten. Tässä ratkaisussa on se ongelma, että opiskelijan tekemän säätimen ja valmiin käyttöliittymän koodin liittäminen yhteen on melko hankalaa. Sitä varten jouduttaisiin tekemään säätimeen kasapäin lisäfunktioita, jotta säätimen sisäisiin muuttujiin päästäisiin käsiksi käyttöliittymältä. Niiden testaamiseen kuluisi liikaa laboratorioaikaa. Siksi päädyttiin laboratoriotyö tekemään ilman käyttöliittymää. Käyttöliittymän tekeminen saa jäädä opiskelijan omaksi projektiksi.

LÄHTEET

Åström, K. & Murray R. 2020. Feedback Systems. An Introduction for Scientists and Engineers. 2. Painos. Princeton: Princeton University Press.

Gaddis, T. 2015. Starting Out with Python. Third edition, Global edition. Boston: Pearson, 2015. Print.

Wittenmark, B., Årzén, K-E., & Åström, K. J. (2002). Computer Control: An Overview. (IFAC PROFESSIONAL BRIEF). International Federation of Automatic Control.

Donaldson, T. Python. 2nd ed. Berkeley, Calif: Peachpit Press, 2008. Print.

Chun, W. Core Python Programming. 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2007. Print.

Matthes, E. Python Crash Course: A Hands-On, Project-Based Introduction to Programming. San Francisco: No Starch Press, Incorporated, 2015. Print.

Measurement Computing®. USB-1408FS-Plus User's Guide. 2014. https://www.mccdaq.com/pdfs/manuals/USB-1408FS-Plus.pdf

InstaCal. Luettu 17.02.2022. https://www.mccdaq.com/daq-software/instacal.aspx

Guide to DAQ signal connections. 2012. Measurement Computing. https://www.mccdaq.com/pdfs/DAQ-Signal-Connections.pdf

Bennett, S. A brief history of automatic control. *IEEE Control Systems Magazine*. vol. 16. no. 3. pp. 17-25, June 1996.

Bennett, S. The Past of PID Controllers. IFAC Proceedings Volumes. Volume 33. Issue 4. 2000. ISSN 1474-6670.

Glyn, J. Modern Engineering Mathematics, Pearson Education Limited, 2013. ProQuest Ebook Central.

Python. Python Software Foundation Mission Statement. 2002. Luettu 28.2.2022. https://www.python.org/psf/mission/

Measurement Computing®. Universal Library Help. 2020. Luettu 1.3.2022. https://www.mccdaq.com/PDFs/Manuals/Mcculw_WebHelp/ULStart.htm

Baher, H. Signal Processing and Integrated Circuits. 1st edition. Hoboken, N.J: Wiley, 2012. Print.

8 LIITTEET

Liite 1. Laboratoriotyön ohje

Laboratoriotyö: PID-säätimen ohjelmointi Pythonilla

Tavoitteet

- Opiskelija ymmärtää säätimen algoritmin toiminnan
- Opiskelija osaa Python ohjelmoinnin perusteet

Ennakkotehtävät

Ennakkotehtävissä olisi hyvä käydä läpi PID-säätimen algoritmin termien diskretointi, jotta laboratoriossa voidaan keskittyä ohjelmoimiseen ja testeihin. Sen lisäksi olisi hyvä kerrata mittausalueen skaalaaminen. Jotta Python olisi tuttu ennen laboratorioon menoa, olisi hyvä käydä läpi perusasiat ohjelmointikielestä.

Laboratoriojärjestelmä

Välineet:

- Tietokone (oma tai laboratorion)
- USB-1408FS-Plus-tiedonkeruukortti
- prosessisimulaattori
- säädettävä jännitelähde
- Fluke-yleismittari

Mcculw-manuaali:
https://www.measurementsystems.co.uk/docs/mc/Universal-Library-Help.pdf

Laboratoriotehtävät

1. **Valmistelevat toimenpiteet**

USB-1408FS-Plus-tiedonkeruukortin löytäminen InstaCalilla

Kytke kortti tietokoneeseen. Käynnistä InstaCal-ohjelma ja paina install-valikosta *refresh*, jolloin InstaCalin pitäisi löytää kytketyt laitteet. Ruudulle pitäisi ilmestyä tiedonkeruukortti (I/O-kortti) numerolla 0. Kortti toimii rajapintana digitaalisen ja analogisen maailman välillä.

(jatkuu)

Analogisen tulokanavan testaus

Testataan analogiatulojen toiminta. Analogiatulojen sijainti löytyy kortin manuaalista. Valitse kortti ja sen jälkeen valitse *test*-valikosta *analog* ja kytke haluamasi analoginen tuloportti johdonpätkällä porttiin 21. Valitse *input ch* -valikosta käyttämäsi kanava. Jos tuloportti toimii, ruudulla pitäisi näkyä tuloportin saama signaali.

VS Code

Avaa VS Code -ohjelma. Luo uusi tiedosto file-valikosta. Valitse tiedoston kieli klikkaamalla tiedoston alusta kohtaa *select a language.* Hae hakukentästä *Python*. Tallenna tiedosto samaan kansioon laboratoriotyön apukoodin kanssa. Nyt voidaan siirtyä itse säätimen ohjelmointiin.

2. Jännitteen arvon lähettäminen tiedonkeruukortille

Kytkentä

Ennen prosessisimulaattorin kytkemistä jännitelähteeseen, varmista ettei jännitelähteen jännite ole yli 5 volttia. Kytke prosessisimulaattori jännitelähteeseen ja aseta jännite viiteen volttiin.

Kytke kortin analoginen lähtökanava prosessisimulaattorin tulokanavaan. Muista myös virran paluu reitti analogisen maan pinniin (AGND). Mittaa yleismittarilla lähtökanavan jännite varmistaaksesi, että jännite on oikea.

Ohjelma

Kommunikoidaksesi tiedonkeruukortin kanssa, tuo tiedostoon luokka *ul* paketista mcculw käskyllä:

from mcculw import ul

Lisäksi mcculw-paketin enums-paketista pitää tuoda ULRange-luokka, jotta kortin jännitealueet voidaan asettaa.

from mcculw.enums import ULRange

ULRange-luokassa on kaikki measurement computingin laitteiden jännitealueet. Laita kaikki import-käskyt aina koodin alkuun.

Ul on luokka, mikä mahdollistaa tiedonkeruukortin analogisten tulokanavien arvon lukemisen ja jännitteen lähettämisen analogisille lähtökanaville.

Ul-luokassa on metodi *v_out*, jolla voidaan lähettää jännitteen arvon. Metodia ottaa parametreikseen:

- **kortin numeron** (sama kuin InstaCalissa etusivulla),
- **analogisen tulokanavan numeron** (0 tai 1),
- **analogisen tulokanavan mittausalueen** (ULRange.UNI5VOLTS) sekä
- **jännitteen arvon (0–5 VDC).**

```
ul.v_out(board_num, ao_channel, ao_range, voltage_value)
```

Aseta jännitteeksi arvo väliltä 0–5. Tarkista yleismittarin näytöltä, että jännite on sama, mikä ohjelmaa on kirjoitettu.

Esimerkki:

```
from mcculw import ul
from mcculw.enums import ULRange

board_num = 0   # Kortin numero
ao_channel = 1  # Analoginen lähtökanavan numero
ao_range = ULRange.UNI5VOLTS    # Analogisen lähtökanavan jännitealue
voltage_value = 2 # Lähetettävän jännitteen arvo

# Lähetetään jännite kortille
ul.v_out(board_num, ao_channel, ao_range, voltage_value)
```

3. Mittaus I/O-kortilta

Kytke analoginen tulokanava prosessisimulaattorin lähtökanavaan.

Tulokanavien arvot voi lukea ul-luokan metodilla v_in(). Metodi ottaa parametreinään:

- **kortin numeron** (sama kuin InstaCalissa etusivulla),
- **analogisen lähtökanavan numeron** sekä
- **analogisen lähtökanavan mittausalueen** (ULRange.BIP10VOLTS)**.**

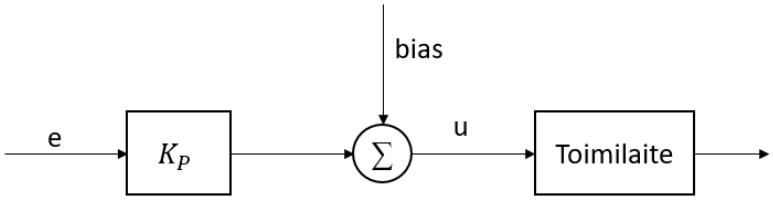
Metodi palauttaa tulokanavan jännitteen arvon. Ota jännitteen arvo talteen muuttujaan ja tulosta muuttujan arvo konsolille print()-funktiolla.

Esimerkki:

```
ai_channel = 0 # Tulokanavan numero
ai_range = ULRange.BIP10VOLTS   # Tulokanavan jännitealue

# Luetaan jännitteen arvo
mittaus = ul.v_in(board_num, ai_channel, ai_range)

# Tulosta mittauksen arvo konsolille
print(mittaus)
```

4. P-säätimen luominen

P-säädin lohkoina:



P-säätimen kaava:

$$P(t_k) = k_p\big(\beta r(t_k) - y(t_k)\big) + bias,$$

jossa k_p on vahvistuskerroin, β on asetusarvon suodatus, *r* on asetusarvo, *y* on mittaus ja *bias* on vakiotermi.

Ohjelma

Säätimen ohjelma pyörii silmukassa ja suorittaa joka kierroksella alla esitetyt neljä vaihetta. Jokaisen kierroksen lopussa odotetaan seuraava kierrosta lyhyt aika, jonka käyttäjä saa päättää.

Säätimen ohjelmakierto:

1. Mittauksen lukeminen (A/D-muunnos)
2. Ohjauksen laskeminen
3. Ohjauksen lähettäminen toimilaitteelle (D/A-muunnos)
4. Odota uutta kierrosta

Tehtävä: Muuta ohjelmakierto koodiksi. Säädin on helpoin luoda omana luokkanaan, josta tehdään olio. Alusta säädin luokan init-metodissa tarvittavat muuttujat P-säätimen luomiseksi. Muista self-avainsana metodin määrittelyssä ja muuttujia

määritellessä. Luo säädinluokalle metodi (funktio), joka ottaa parametrinaan asetusarvon ja suorittaa yllä mainitut vaiheet.

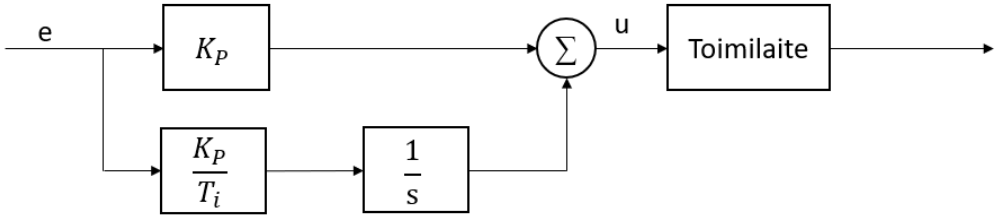
Kuvaajan piirtämiseksi on tehty valmiit luokat nimeltä *kuvaaja_luokka* ja *rajapinta_luokka*. Ne ovat osa laboratorio_luokat-pakettia. Ohjeet luokkien käyttöön löytyy tämän dokumentin lopusta omasta osiostaan.

Tee säätöpiirille askelvastekoe. Raportoi tulokset raporttiin.

Tehtävä: Mitä parametrin β muuttaminen saa aikaan? Tee askelvastekoe parametrin arvoilla 0, 0.5 ja 1. Mitä hyötyä parametrista on? Pohdi raportissa. Esitä myös testien tulokset raportissa.

5. PI-säätö

Lisätään edellä tehtyyn P-säätimeen integrointi:



Integroivan termin kaava on

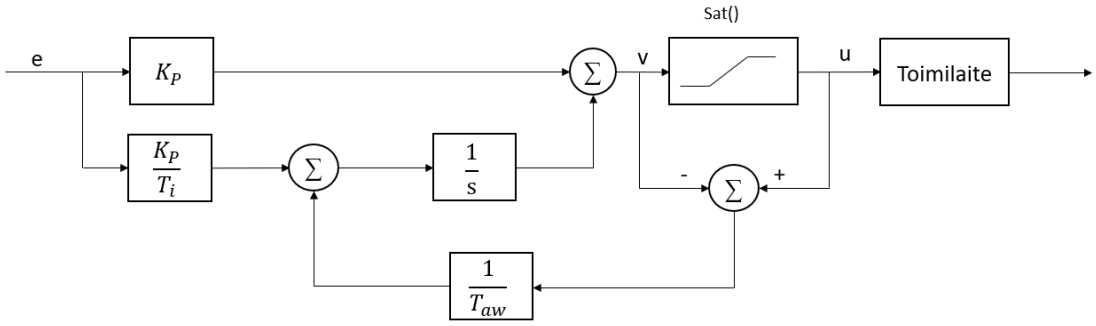
$$I(t_{k+1}) = I(t_k) + \frac{K_p}{T_i} e(t_k)h,$$

jossa, $I(t_k)$ on nykyinen integroivan termin arvo, T_i on integrointiaika, $e(t_k)$ on erosuure ja h on laskentakierrosten välinen aika.

Tehtävä: Tee PI-säätimelle askelvastekoe ja raportoi tulokset raporttiin.

Tee uusi askelvastekoe ja aseta säätimen asetusarvoksi 11 ja tulosta integroiva termi konsolille. Mitä tapahtuu integroivalle termille ja ohjaukselle?

Integraalin kerimisen estäminen



Integraalin kerimisen estäminen saadaan aikaan lisäämällä integroivaan termiin toimilaitteen rajan ylittänyt osuus ohjauksesta jaettuna parametrilla T_{aw}.

$$I(t_{k+1}) = I(t_k) + \frac{K_p}{T_i} e(t_k) h + \frac{1}{T_{aw}} (sat(u_a) - u_a),$$

jossa T_{aw} on kerimisen estämisen nopeutta säätävä jakajaparametri ja u_a on säätimen laskettu ohjaus. Sat-funktio simuloi toimilaitteen toimintaa. Se käytännössä palauttaa lasketun ohjauksen toteuttamiskelpoisen osuuden.

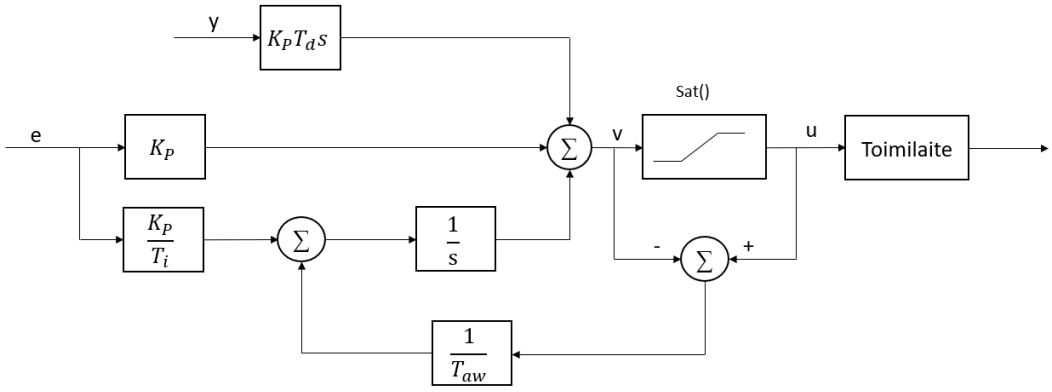
Kaavassa näkyvä funktio sat() on pythonilla toteutettuna:

```
U_yläraja = 5
U_alaraja = 0
v = P + I + D

U = max(min(v, U_yläraja), U_alaraja)
```

Miten funktio toimii? Vinkki: Testaa, mitä arvoja funktio palauttaa, kun ohjaus on yli tai ali jommankumman rajan.

6. PID-säätö

Erosuureen derivointia ei voida käytännössä toteuttaa. Sen sijaan säätimissä derivoidaan vain mittausta. Asetusarvoa muutetaan usein askelmaisesti, mikä johtaa ohjauksessa äkillisiin piikkeihin derivoinnin takia, jos derivoivaa termiä ei ole suodatettu.



Derivoivan termin kaava on:

$$D(t_k) = \frac{T_\mathrm{f}}{T_\mathrm{f} + h} D(t_{k-1}) - \frac{K_p T_d}{T_\mathrm{f} + h} \left(y(t_k) - y(t_{k-1})\right),$$

jossa T_f on derivaatan suodatuksen aikavakio, $D(t_{k-1})$ on edellisen laskentakierroksen derivaatta, T_d on derivointiaika ja $y(t_{k-1})$ on viime laskentakierroksen mittaus.

Kaava vaatii siis mittauksen säilyttämistä seuraavalle kierrokselle. Tee tätä varten oma muuttujansa. $D(t_{k-1})$ muuttujana voi käyttää yhtä ja samaa d-termin muuttujaa, koska muuttujan arvo on sama kuin viimekierroksella. Siihen, mistä derivoivan termin kaava tulee, ei mennä tässä laboratoriotyössä.

Tee säätimelle askelvastekoe ja raportoi tulokset.

7. A/M-vaihto

Lisätään PID-säätimeen manuaalitila, jossa ohjauksen voi asettaa suoraan. Vaikka ohjaus tulee suoraan käyttäjältä, PID-algoritmi suoritetaan silti joka kierroksella.

- vaihtaminen ilman trackingiä ja sen kanssa
- mittauksen seuranta
- asetusarvon seuranta

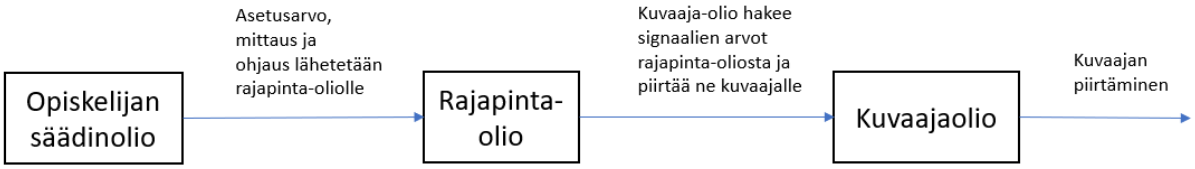
8. **Mittauksen ja ohjauksen skaalaus**

   Lisää säätimeen algoritmiin metodi/funktio, joka skaalaa muuttujan halutulle alueelle. Skaalaa asetusarvo ja mittaus samalle alueelle, esimerkiksi 0–100 %. Miten tätä vaikuttaa PID-säätimen laskentaan?

Kuvaaja- ja rajapintaluokan käyttäminen säätimen luomisessa

Laboratoriotyön avuksi on tehty kaksi luokkaa auttamaan opiskelijan luoman pid-säätimen signaalien piirtämisessä kuvaajalle. Luokkien koodia voi tarkastella niiden lähdekoodin tiedostosta mutta se ei ole pakollista.

Luokkien käytön periaate:



1. vaihe: luokkien lisääminen ohjelmaan
Tuo luokat ohjelmaasi laboratorio_luokat-paketista komennolla:

from laboratorio_luokat import kuvaaja_luokka, rajapinta_luokka

Laita komento ohjelmasi alkuun muiden import-komentojen kanssa.

2. vaihe: säätimen muuttujien siirtäminen rajapinnalle
Rajapintaluokan käyttämiseksi säädinluokkaa pitää muokata hieman. Lisää säädinluokkasi init-metodiin yksi parametri lisää nimeltä rajapinta. Lisää myös yksi muuttuja lisää, johon tallennat rajapinnan. Esimerkki:

```
class PID:

    def __init__(self, rajapinta):
        self._rajapinta = rajapinta
```

Tämä mahdollistaa rajapintaolion palvelujen käyttämisen säädinluokan sisällä. Seuraavaksi lisää säädinluokan ohjauksen laskennan suorittavan metodin loppuun komento:
self._rajapinta.paivita_kuvaajan_arvot(ysp, y, u)

Paivita_kuvaajan_arvot-metodi lähettää asetusarvon, mittauksen ja ohjauksen arvot kuvaajalle. Esimerkki:

```
class PID:

    def __init__(self, rajapinta):
        self._rajapinta = rajapinta


    def LaskeOhjaus(self, asetusarvo):

        # Säätimen algoritmin koodi…
        # …
        # …

        self._rajapinta.paivita_kuvaajan_arvot(asetusarvo, mittaus, ohjaus)
```

3. vaihe: olioiden luominen

Ensiksi luodaan rajapintaolio, jonka parametreiksi annetaan x- ja y-akselin pituudet sekä tieto siitä mitkä kuvaajat piirretään. Seuraavaksi luodaan oma pid-olio. Anna pid-oliolle parametrinä rajapintaolio, kuten vaiheessa 2 on ohjeistettu. Lopuksi luo kuvaajaolio kuvaajaluokasta ja anna kuvaajaoliolle parametrina sama rajapintaolio sekä pid-olion ohjauksen laskeva metodin nimi. **Älä laita sulkeita nimen perään, vaan pelkkä metodin nimi**. Anna kuvaaja luokalle myös askelvasteen alku- ja loppuarvo start- ja end-parametrien avuilla. Kuvaajaolio kutsuu pid-säätimen algoritmia 0.1 s eli pid-säätimelle ei tarvitse luoda omaa silmukkaa.

Kun oliot on luotu, ohjelmasi pitäisi suorittaa askelvaste ja piirtää kuvaaja.

```
# Luodaan rajapintaolio
rajapinta = rajapinta_luokka(x_akseli_pituus=300, y_akseli_pituus=20,
asetusarvo=True, mittaus=True, ohjaus=True)

# Luodaan säädinolio
saadin = PID(rajapinta)

# Luodaan Kuvaajaolio
kuvaaja = kuvaaja_luokka(rajapinta, saadin.LaskeOhjaus, start=0, end=3)
```