Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Alfons Korhonen

Modulaarinen python-sovellus teollisuusautomaatiossa

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Konetekniikka

Insinöörityö

15.4.2021

Tekijä Otsikko Sivumäärä Aika	Alfons Korhonen Modulaarinen python-sovellus teollisuusautomaatiossa 36 sivua 15.4.2021
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Konetekniikka
Ammatillinen pääaine	Koneautomaatio
Ohjaajat	Lehtori Antti Liljaniemi Toimitusjohtaja Ari Kuisma

Insinöörityössä luotiin modulaarinen python-sovellus, joka toimii soft plc:nä sekä toteuttaa kommunikaatiota erilaisen teollisuusväylien yli. Työssä tehtiin pääohjelmarunko, jonka ympärille voi liittää erilaisia moduuleja riippuen mitä toiminnallisuuksia tarvitaan. Työssä tutkittiin ja luotiin moduulit modbus TCP/RTU/ASCII-protokollalle, opcua-protokollalle sekä moduuli, joka voi tallentaa tietoa siitä, mitä kyseiset väylämoduulit tuottavat. Insinöörityössä tutkittiin myös erilaisia ohjelmistosuunnittelumenetelmiä.

Työn tarjoajana on Vuo Automaatio Oy, joka tarjoaa teollisuusautomaation ohjelmisto- ja suunnittelupalveluita.

Työtä lähdettiin mallintamaan codea and fix -ohjelmointisuunnittelumenetelmällä. Työssä tutkittiin sen toimivuutta menetelmänä ja sitä, oliko se toimiva insinöörityössä tehdylle projektille. Python-sovellukseen mallinnettiin pääohjelma, joka huolehti kommunikaatiosta eri moduulien välillä. Moduuleita lähdettiin suunnittelemaan tutkimalla avoimen lähdekoodin kirjastoja sen osalta, mitä voisi hyödyntää, jotta saataisiin moduulin toiminnallisuus toteutettua. Jos pythonista löytyi valmis kirjasto, jolla kyettiin toteuttamaan moduulin toiminnallisuus, käytettiin aina ensisijaisesti sitä.

Insinöörityöstä valmistui pääohjelmarunko, joka päälle voi muodostaa enemmän moduuleja ja muita toiminnallisuuksia. Moduuleille luotiin tiettyjä sääntöjä, joiden pohjalta uusia moduuleja voitaisiin lisätä projektiin tulevaisuudessa.

Avainsanat	python, ohjelmoinninsuunnittelu, python automaatiossa

Author Title Number of Pages Date	Alfons Korhonen Modular Python Application for Industry Automation 36 pages 15 April 2021
Degree	Bachelor of Engineering
Degree Programme	Mechanical Engineering
Professional Major	Machine Automation
Instructors	Antti Liljaniemi, Senior Lecturer Ari Kuisma, CEO

In the engineering thesis, a modular python application was created, which acts as a soft plc and implements communication across different industry buses. A framework program was made in the thesis, around which different modules can be connected, depending on what functionalities are needed.

The modules for the modbus TCP / RTU / ASCII protocol, the opcua protocol and a module that can store information produced by these bus modules were studied and created. Different software design methods were also studied in the engineering thesis.

The engineering thesis was commissioned by Vuo Automaatio Oy, which offers industrial automation software and design services.

The thesis started by modeling a code and fix programming design method. The work examined its functionality as a method and whether it was functional for this engineering project. A main program was modeled for the Python application, which took care of the communication between the different modules. The modules were designed with the help of open-source libraries that could be utilized to implement the functionality of the module. If a library were found in Python standard library that could implement the functionality of the module, it was always used first.

The engineering thesis completed the main program framework, on top of which can more modules be formed and other functionalities. Certain rules were created for the modules, based on which new modules could be added to the project in the future.

Keywords	python, program design, python in automation

Sisällys

Lyhenteet

PLC	Ohjelmoitava logiikka (programmable logic controller)
CSV	Pilkuilla erotellut arvot
JSON	JavaScript Object Notation.
XML	Extensible Markup Language.
Moduuli	itsenäinen osa tietokoneohjelmassa
Säie	kevyempi tietokoneohjelman itsenäisesti suoritettava osa tai tehtävä.
RTU	Remote terminal unit
ASCII	American Standard Code for Information Interchange
SOAP	Simple Object Access Protocol

1 Johdanto

Työn tarkoituksena on luoda modulaarinen python-projekti, joka sisältää teollisuusautomaation kommunikaatioprotokollia, joita voitaisiin hyödyntää soft plc -ratkaisuna. Python-projekti sisältää myös datan keräämistä sekä muita moduuleja esimerkiksi ftp-protokollalle. Käytännössä python-projektiin voi luoda mitä moduuleja tahansa. Tässä kyseisessä työssä keskitytään niihin moduuleihin, jotka keskustelevat eri teollisuusväyläprotokollien yli. Työssä myös luodaan datankeräysmoduuli sekä pienimuotoinen soft plc. Työn tavoitteena on luoda toimivat modbus TCP/RTU/ASCII, opcua, csv-moduulit sekä soft plc-lisäys. Tässä työssä tutkitaan myös erilaisia ohjelmistosuunnittelumenetelmiä.

Työtä lähdetään mallintamaan hyödyntäen ensimmäistä ohjelmointisuunnittelumenetelmää, jota ohjelmointisuunnittelussa aikanaan käytettiin. Kyseisen menetelmän nimi oli code and fix ja sitä hyödynnettiin ennen kuin ohjelmointiprojektit alkoivat paisua suuremmiksi. Kyseinen menetelmä tarjoaa nopean pääsyn itse ohjelmointiin ja suunnittelu tapahtuu samaan aikaan, kun ohjelma kehittyy.

Suunnittelun alkuvaiheessa tutkitaan myös sitä, miten pääohjelma ja moduulit voivat keskustella toisensa kanssa, millä tavalla data liikkuu ja miten pääohjelma pitää huolen niistä kaikista tuloista ja lähdöistä, joita moduulit lähettävät pääohjelmalle.

Itse moduuleita lähdetään muodostamaan tutkimalla sitä, mitä valmiita, muiden tekemiä kirjastoja on tarjolla. Ennen kuin muiden tekemää kirjastoa muotoillaan moduuliin, tarkistetaan, onko mahdollista käyttää pythonin omia kirjastoja. Pythonista valmiiksi löytyvä kirjasto menee aina muiden tekemien kirjastojen edelle.

Työn tarjoajana on Vuo Automaatio Oy, joka tarjoaa teollisuusautomaation ohjelmistoja suunnittelupalveluita. Kyseinen projekti on täysin firman sisäinen projekti.

2 Ohjelmiston suunnittelu

Ohjelman suunnittelua voi lähestyä monesta eri näkökulmasta. Menetelmä valitaan ohjelmoitavan projektin suuruudesta riippuen asiakkaan tarpeiden mukaan. Ohjelman suunnittelu ja tiettyjen menetelmien noudattaminen antaa ohjelmalle hyvää rakennetta ja yhteiset pelisäännöt kaikille, jotka projektia tekevät. Ilman ohjelmoinnin suunnittelua ohjelmistoala alkoi joutua ongelmiin. [2]

Syntyneitä ongelmia:

- Projektit alkoivat myöhästyä niille asetetuista aikatauluista.
- Ohjelmien budjetit alkoivat venyä ja maksaa liikaa.
- Ohjelmat eivät olleet niin tehokkaita, mitä ne olisivat voineet olla.
- Ohjelmien laatu oli alhainen ja eivät vastannut odotuksia.
- Ohjelmien ylläpito oli vaikeata tai lähes mahdotonta.

Yllä mainittujen ongelmien takia ohjelmointiyhteisössä ryhdyttiin kehittämään erilaisia menetelmiä, joiden avulla saataisiin vähennettyä näiden ongelmien syntyä. [2]

2.1 Code and fix

Code and fix on ensimmäisiä tapoja, joiden avulla ohjelmia tuotettiin. Kyseistä menetelmää käytettiin ennen kuin muita menetelmiä ryhdyttiin pohtimaan. Kyseinen menetelmä on nimensä mukainen, eli ensin ohjelmoidaan ja sen jälkeen korjataan syntyvät bugit. Jos tätä menetelmää hyödynnetään nykypäivänä, tehdään yleensä lyhyt vaatimussuunnitelma ja sen jälkeen ryhdytään rakentamaan projektia. Tämä menetelmä sopii hyvin pienempiin projekteihin, joissa ei ole suurta budjettia. [1]

Hyödyt:

- Pääsee heti ohjelmoimaan, joka säästää aikaa.
- Budjetit pysyvät alhaisina.
- Saa heti tuloksia.
- Soveltuu hyvin, kun tarvitaan nopeasti prototyyppi.

Haitat:

- Suurempi riski siihen, että ohjelma ei täytä asiakkaan vaatimuksia, kun isompaa tai laajempaa etukäteissuunnittelua ei tehty.
- Kommunikaatio-ongelmia, jos ohjelmaa kehitetään isommalla tiimillä.
- Riski siihen, ettei ohjelma toimi enää, jos siihen lisätään myöhemmässä vaiheessa päivityksiä.

2.2 Vesiputous-malli

Vesiputous-mallissa ohjelmoinnin suunnittelu totutetaan putousmaisesti ylhäältä alaspäin tiettyjen vaiheiden kautta.

Vesiputous-mallin vaiheet:

1. Järjestelmänvaatimukset → Mitä kyseinen järjestelmä vaatii, jotta ohjelma voitaisiin ajaa järjestelmällä?
2. Ohjelmistovaatimukset → Mitä ohjelman toiminnallisuuksia vaaditaan / tarvitaan?

3. Analyysi → Pilkotaan ongelmat pienempiin osiin, jotta ne olisi helpompi ratkaista.
4. Ohjelmiston suunnittelu → Suunnitellaan ohjelman rakenne koodin näkökulmasta.
5. Ohjelmointi → Ohjelmoidaan kyseinen kokonaisuus, joka on suunniteltu.
6. Testaus → Testataan lopullisen ohjelman toimivuutta.
7. Käyttöönotto → Laitteiston asennus ja valmistelu, tietoyhteyksien valmistelu, vanhojen tietojen konvertointi ja käyttäjien valmentaminen.

Mallin käyttö vaatii, että jokainen vaihe tehdään valmiiksi ennen kuin seuraavaa aloitetaan. Ideana on se, ettei palattaisi enää takaisin päin sen jälkeen, kun yksi vaihe saadaan valmiiksi. Vesiputous-mallissa panostetaan ohjelmoinnin alkuvaiheen suunnitteluun, mikä voi säästää suurissa projekteissa rahaa ja aikaa. Kun jokaisen vaiheen täytyy olla kokonaan valmis ennen kuin jatketaan seuraavaan vaiheeseen, kaikki vaiheet on dokumentoitu erittäin hyvin. Tällöin kuka tahansa voi tulla projektiin mukaan ja tutustua ensin siihen dokumentointiin, mitä on jo tuotettu, ja saada hyvän kuvan siitä, mitä on tekemässä. Tämä malli tarjoaa myös selkeän ja helposti ymmärrettävän suunnittelumenetelmän.[1]

Tätä mallia pidetään toisaalta huonona, koska ajatellaan, että on mahdotonta toteuttaa mitään suurempaa ohjelmistokokonaisuutta ilman että jouduttaisiin palaamaan takaisin suunnitteluvaiheeseen. Mallia käyttävät suunnittelijat eivät myöskään osaa ennakoida aivan kaikkia ongelmia, mitä projektissa voi syntyä. Myös asiakkaan mielipide voi vaihtua matkan varrella tai silloin, kun asiakas näkee ensimmäisen prototyypin. Tällöin pitäisi palata takaisin suunnittelupöydälle ja toteuttaa kaikki vaiheet uudestaan, mikä vie aikaa.[3]

Huomioitavaa on se, että tätä mallia voi hyödyntää vain suunnittelun pohjana, jolloin ei tarvitse täysin seurata mallia alusta loppuun. Jos ongelmia syntyy, voidaan palata takaisin tiettyyn kohtaan eikä työtä tarvitse aloittaa täysin uudestaan alusta.

Parempi vaihtoehto onkin käyttää ketterän kehityksen malleja.

2.3 Ketterän menetelmän malli

Ketterä ohjelmointikehitys on nykyään se tapa, jolla ohjelmistoprojekteja kehitetään. Näitä menetelmiä on erilaisia ja niitä yhdistää suora viestintä asiakkaan ja muiden kehittäjien kanssa, ohjelman ensisijaisuus sekä nopea reagointi ongelmiin tai ohjelman toimivuuteen. [1]

Muutama esimerkki ketteristä menetelmistä [4]:

- Extreme Programming
- Scrum
- DSDM
- Crystal Methods
- Agile modeling
- Adaptive software development
- Pragmatic Programming
- Feature driven development
- Gilb-EVO

Menetelmien yhteiset piirteet:

Ketterä menetelmä pyrkii pienentämään riskejä jakamalla ohjelmointiprojektin pieniin kiertoihin, jotka kestävät perinteisesti yhdestä neljään viikkoon. Jokainen kierto sisältää kaikki ohjelmointiprojektin vaatimukset ja on itsessään kuin pieni ohjelmointiprojekti. Kun kierto on saatu toteutettua, arvioidaan saadut tulokset ja suunnitellaan seuraavaa kiertoa.[1]

Kierron vaiheet:

1. Projektisuunnittelu
2. Vaatimusanalyysi
3. Ohjelmistosuunnittelu
4. Koodaus
5. Testaus
6. Dokumentointi

Ketterässä menetelmässä pidetään viestintää palaverityyppisesti tai suoraan tiimin henkilöille tärkeämpänä kuin suurta dokumentointia, kuten muissa perinteisissä malleissa esimerkiksi vesiputousmallissa. Ketterässä menetelmässä tiimiin kuuluvat kaikki, joita tarvitaan, jotta projekti saadaan valmiiksi. Yleensä koko tiimi työskentelee samassa tilassa, jotta viestintä olisi helpompaa. Tiimi voi sisältää päälliköitä, käyttöönottajia suunnittelijoita tai testaajia. [1]

2.4 Työn suunnittelu

2.4.1 Vaatimukset ja menetelmän valinta

Tässä insinöörityössä suunnittelumenetelmäksi valikoitui code and fix. Valinta kohdistui tähän menetelmään, koska työssä toteutettu kehitysprojekti ei ollut suuri. Projekti alkoi nollasta ja oli tarvetta päästä suoraan ohjelmoimaan. Projektin alkuvaiheessa tehtiin pienimuotoinen suunnitelma siitä, mitä kaikkea ohjelman tulisi sisältää ja mitä siltä voisi vaatia.

Ohjelman vaatimukset:

1. Pääohjelmalla on tieto kaikesta datasta, mikä liikkuu moduulien välillä.

2. Tulo/lähtö -moduulit pystyvät lukemaan ja kirjoittamaan väylän yli ulkoiselle laitteelle.

3. Kaikkien moduulien kommunikaatio liikkuu pääohjelman kautta.

4. Tulevaisuudessa uuden moduulin lisääminen ei vaikuta muihin moduuleihin.

2.4.2 Työn vaatimuksien toteuttaminen

Aluksi työtä lähdettiin rakentamaan moduulien kautta tutustumalla erilaisiin avoimiin python- moduuleihin. Kun sopivat valmiit kirjastot oli valittu GitHubista, lähdettiin moduuleja mallintamaan ja niitten toiminnallisuuksia testaamaan. Tässä syntyi ohjelmistosuunnittelun ensimmäinen virhe. Ohjelmiston rakenteen suunnittelu olisi pitänyt aloittaa pääohjelmasta ja hahmottaa perusteellisesti se, miten tieto liikkuu ohjelman läpi. Tämän ensimmäisen virheen olisi voinut välttää, jos olisi miettinyt perusteellisemmin, miten ohjelman tulisi toimia sekä miten data liikkuu koko ohjelmassa.

Kun uusi pääohjelman rakenne oli suunniteltu sekä luotu, syntyi uusia ongelmia suunnitteluvaiheessa. Nämä ongelmat liittyivät pääsääntöisesti siihen, miten kyseisiä uusia ratkaisuja lisättäisiin moduuleihin, kun niissä oli jo valmiiksi paljon kirjoitettua koodia. Tässä vaiheessa olisi ollut järkevää ottaa askel taaksepäin ja suunnitella se, miten jokainen moduuli voisi implementoida uudet muutokset. Kun uusia muutoksia pakotettiin moduuleihin, niihin syntyi liian monimutkaisia ja yliajateltuja rakenteita, joita jouduttiin myöhemmässä vaiheessa karsimaan.

Suunnittelun viimeiseen vaiheeseen pääsyn vaatimuksena oli se, että pääohjelma ja moduulit toimivat yhdessä. Kun tämä vaihe oli saavutettu, alkoi koko ohjelman testaus, mikä sijoittuu code and fix -menetelmän mukaisesti fix-osuuteen. Itse testaus on selitetty tarkemmin kappaleessa ”tulokset”.

2.4.3 Code and fix -toimivuus

Menetelmänä code and fix oli projektin alkuvaiheessa oikein sujuva ja toimiva, koska ohjelman koko oli hyvin pieni. Oli myös sekä käytännöllistä että palkitsevaa päästä itse asiaan, eli kirjoittamaan koodia. Suurimmat ongelmat ja kompastuskivet syntyivät, kun itse ohjelma muuttui suuremmaksi. Kuten aiemmin on todettu, code and fix toimii pienissä projekteissa. Vaikka tässä työssä tehty projekti ei ollut hirveän iso, tämä menetelmä ei toiminut kyseisessä projektissa. Aikaa olisi säästynyt, jos olisin panostanut hieman enemmän itse suunnitteluun ja vasta tämän jälkeen ryhtynyt ohjelmoimaan.

3 Hyödynnetyt teknologiat

3.1 Python

Python on tulkittava kieli ja tällä tarkoitettaan sitä, että ohjelmat ovat suoraan valmiiksi ajettavissa eikä niitä tarvitse kääntää ensin kokonaisuudessaan konekieleksi niin kun esimerkiksi C-kielessä täytyy. Tulkittavassa kielessä on toinen ohjelma, joka lukee ja suorittaa yhden komennon kerrallaan, eikä ole suoraan suoritettu kohde koneella. Tämä nopeuttaa ohjelmistojen kehitystä, koska niitä voi testata nopeasti pienissä sykleissä. Haittana on suorittamisnopeus. Python-ohjelma voi olla 100-kertaisesti hitaampi kun sama C-kielellä tehty ohjelma. Koska nykyiset tietokoneet ovat nopeita, itse ohjelmalta ei aina tarvita nopeutta, kunhan ohjelma tekee tarpeeksi nopeasti halutun tehtävän.[5]

Python soveltui tähän työhön hyvin, koska ei ole tarvetta esimerkiksi siirtää tietoa todella nopeasti. Jos esimerkiksi kommunikoidaan modbus-väylän yli, ja vaikka se olisi modbus TCP/IP, vastausnopeudet ovat noin 30ms luokkaa, joka on tietokonemaailmassa todella hidasta.

Python on noussut suosioon, koska se on todella helppo oppia, vaikka ei olisi ennen ohjelmoinut. Pythonilla on todella suuri kirjastokokoelma, mitä voi hyödyntää omissa projekteissa. Python toimii myös suurena tekijänä koneoppimisessa.

3.2 Opc-ua

OPC UA (Open Platform Communications United Architecture) on datanvaihtostandardi, joka on tarkoitettu teollisuudelle. Tämä standardi on riippumaton valmistajan laitteista tai ohjelmista, eikä ole väliä, mitä käyttöliittymää käyttää. OPC UA -standardi on myös vapaa kaikille, jotka sitä käyttävät. OPC UA on jaettu binaariseen sekä verkkopalvelustandardeihin. Binaarinen protokolla antaa paremman suorituskyvyn ja se ei ota hirveästi resursseja, mikä on hyvä asia, jos käyttää sulautettua järjestelmää. Binaarinen protokolla toimii yhdellä valitulla TCP-portilla, joka on yleensä 4840. Web Service (SOAP) -

protokolla on parhaiten tuettu esimerkiksi käyttäen Javaa tai .NET-ympäristöä. Tämä protokolla käyttää tavallista http/https TCP -portteja ja se on palomuuriystävällinen [9].

3.3 Modbus

Modbus on protokolla, joka määrittää sen, miten viesti on rakennettu lähetettäväksi laitteiden välillä. Tämän takia modbus ei määritele sitä, millä tavalla laitteet ovat fyysisesti yhteydessä toisiinsa. Modbusia voi hyödyntää sarjaliikennemallina, mille se on alunperin kehitetty. Tällöin yleensä käytetään RS232, RS485, RS422 -standardeja fyysisenä kerroksena. Modbus-sarjaliikennemallissa on yksi isäntälaite, joka lähettää kyselyjä siihen liitetyille orjille. RS232-mallissa isännässä voi olla kiinni vain yksi laite. Tätä RS232-standardia ei ole enää yleisessä käytössä. Parempi vaihtoehto on RS485, koska tällä standardilla voidaan liittää isäntälaitteeseen 32 orjaa.[7]

Modbusin viestirakennetta voi myös hyödyntää Ethernet TCP/IP -protokollalla. Tällöin laitteet voivat olla yhteydessä toisiinsa Ethernet-verkossa. Tällöin viestit ovat pakettimuodossa ja yhteys toimii client (isäntä)/server (palvelin) -periaatteella. Modbus TCP/IP antaa käyttäjän laiteyhteyksille paljon enemmän joustavuutta. Jos on tarvetta yhdistää sarjaliikenneorjia modbus TCP/IP -verkolle, tämän voi toteuttaa muuntimilla, jotka muuttavat sarjaliikenneviestit Ethernet TCP/IP -muotoon ja toisin päin.

Modbus tarjoaa muutamia eri rekistereitä.

Nämä rekisterit ovat [8]:

- Kela (coil), joka on yhden bitin kokoinen. Tätä voidaan lukea ja kirjoittaa.
- Erotettu input (discrete input), joka on kooltaan yhden bitin. Tätä rekisteriä voi pelkästään lukea.
- Omistusrekisteri (holding register), joka on kooltaan kuusitoista bittiä. Tätä voidaan lukea ja kirjoittaa.
- Input-rekisteri (Input register), joka on kooltaan kuusitoista bittiä. Tätä voi pelkästään lukea.

Perinteisesti jokainen näistä rekistereistä elää omalla muistialueella, jotka ovat kooltaan 0–9999.

Tässä voi olla eroja siinä, miten eri laitteet määrittävät nämä muistialueet. Esimerkiksi ABB pm5650 PLC määrittää vain yleisen alueen välillä 0–65534, johon näitä modbus-rekistereitä voi asettaa. Tähän voi sekoittaa kaikkia rekisterityyppejä.

Modbus-protokolla on suosittu teollisuudessa, koska se on avoin ja lisenssimaksuton standardi. Se on myös suunniteltu teollisuustarpeisiin ja tämän takia se on luotettava. Se siirtää dataa ilman, että mikään laitevalmistaja asettaa sille omia rajoituksia. Tämä tarkoittaa sitä, että eri valmistajan laitteita voidaan yhdistää samalla protokollalla. [7]

Modbus-viestit

Sarjaliikenneviestit voivat olla joko muotoa RTU tai ASCII. Näiden kahden viestityypin ero on se tapa, miten data on kuvailtu viestissä. RTU-viestin tavut muodostuvat kahdesta neljän bitin HEXA desimaaliluvusta, kun taas ACSII-viestissä jokainen tavu on muodostettu kahdesta ascii-kirjaimesta. Jos laitteet käyttävät eri viestintätapaa, ne eivät voi kommunikoida toistensa kanssa [8].

Kun isäntä lähettää viestin väylällä oleville orjalaitteille, tämä viesti alkaa vähintään 28bitin hiljaisuudella, jos viestintä tapa on RTU. ASCII-tavassa viesti alkaa aina → : merkillä [8].

Kun viestin alku on tunnistettu, alkaa itse viesti orjan osoitteella. Tällä orjan osoitteella taataan se, että viesti menee oikealle orjalle. Näitä orjaosoitteita voi olla 1–254 välillä [8].

Viestin seuraavassa osuudessa on funktiokoodi, jolla määritetään se, mitä halutaan tehdä orjalaitteella, esimerkiksi, halutaanko laitteella kirjoittaa tai lukea [8].

Funktio koodin numero	Komennon toiminto
01	Lukea kelan statuksia
02	Lukea inputin statuksia
03	Lukea omistus rekistereitä
04	Lukea omistus input rekistereitä
05	Kirjoittaa yhteen kelaan
06	Kirjoittaa yhteen omistus rekisteriin
15	Kirjoittaa moneen kelaan
16	Kirjoittaa moneen omistus rekisteriin

Taulukko 1 Eri modbus viesti koodeja

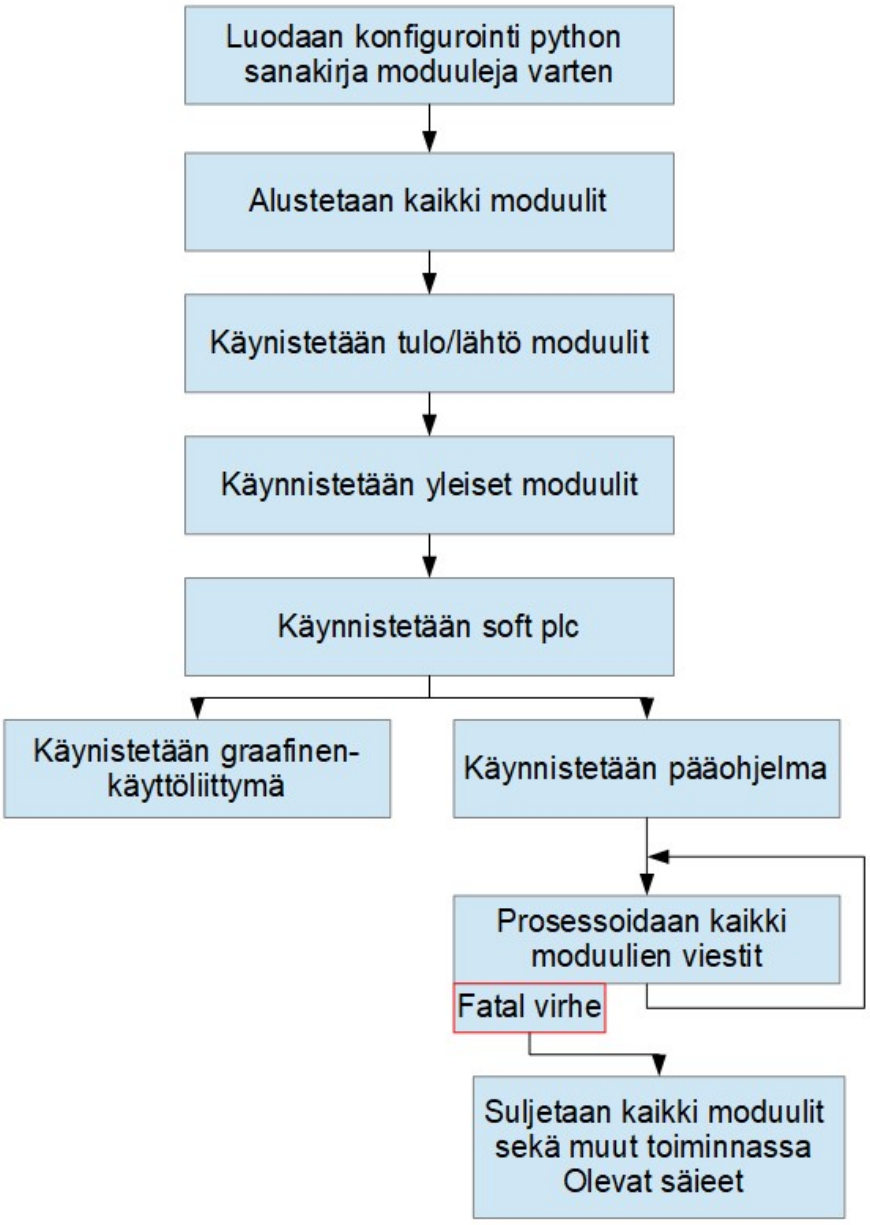
Funktiokoodin jälkeen tulee itse data, joka on laitteelle suunnattu. Tämä osuus sisältää rekisterien arvot. Modbus-viestin maksimikoko on 255 tavua, jos kyseessä on RTU. Tästä 255 tavusta datalle on säästetty 250 tavua, johon mahtuu joko 125 rekisteriarvoa tai 2000 kela-arvoa. Jos laitteet käyttää ASCII-tyyppiä viestimiseen, on viestin maksimi koko 513 tavua. [8]

Datan jälkeen tulee tarkistus, jolla voidaan testata, tuliko viesti laitteelle oikein. RTU viestimalli käyttää CRC (Cyclic redundancy check) millä se tarkistaa viestin, kun taas ASCII käyttää LRC (Longitudinal redundancy check). [8]

Kun käyttää modbus TCP/IP, on tämä kyseinen modbus-viesti upotettu Ethernet-pakettiin (liite ethernet protocollasta). Tällöin itse modbus-viestiosuus rakentuu samalla periaatteelle kuin sarjaliikennemallissa, mutta viestistä jätetään pois tarkistusosuus. [8]

4 Python-sovelluksen periaate

Insinöörityössä kehitetty sovellus toimii erilaisten moduulien ympärillä, joita voidaan ottaa käyttöön hyödyntäen konfigurointitiedostoa. Sovelluksen sydämenä toimii yksi pääohjelma, joka käynnistää halutut moduulit, sekä toimii soft plc:nä. Moduuleilla on kaikilla oma toiminnallisuus ja jokainen moduuli pääsääntöisesti toteuttaa vain yhtä toimintoa.



Kuva 1. Pääohjelman ohjelmakierto

4.1 Pääohjelma

Pääohjelma on jaettu muutamaan osaan. Ensimmäisessä osassa pääohjelma tarkistaa, mitkä moduulit halutaan valita riippuen siitä, mitä konfigurointitiedostossa on asetettu. Konfigurointitiedostona käytetään INI-tiedostoa, joka on tekstipohjainen tiedosto, johon käyttäjä voi asettaa tiettyjen sääntöjen mukaan konfiguraatioasetuksia. Kun kaikki moduulit on alustettu, käynnistetään niiden säikeet tai prosessit, jotka lähtevät toteuttamaan moduulien tehtäviä. Jos jotain moduulia ei onnistuttu käynnistämään, se hylätään ja siitä ilmoitetaan käyttäjälle.

Pääohjelman toisessa osassa sijaitsee pääluuppi, joka on toiminnassa ikuisesti. Pääohjelman säie on säästetty graafiselle käyttöliittymälle, joka käynnistetään, jos käyttäjä on valinnut sen käyttöön. Toiset toiminnot suoritetaan toisessa säikeessä.

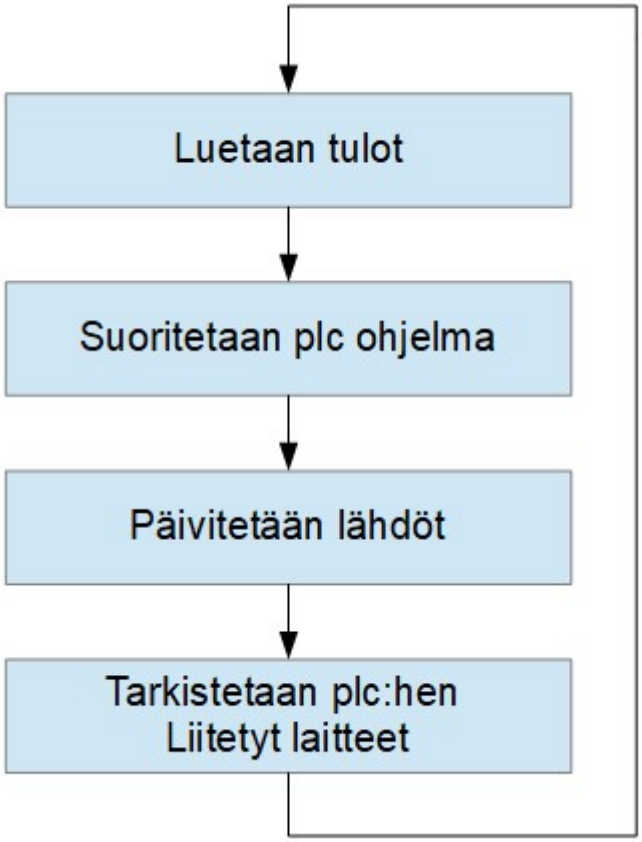
4.1.1 Pääohjelman korvaava säie

Koska pääohjelman säie on säästetty graafiselle käyttöliittymälle, toimii pääluupin korvaava säie ikään kuin toisena pääluuppina tulo/lähtö- sekä muille moduuleille. Tämä luuppi on käynnissä niin kauan, kunnes ohjelma lopetetaan. Tässä osuudessa toteutetaan kommunikointia eri moduulien välillä sekä hallinnoidaan kaikkia niitä tulo-/lähtöarvoja, joita moduulit tuottavat pääohjelmalle.

Pääohjelman rinnalla toimii myös soft plc. Soft plc tarkoittaa sitä, että toteutetaan plc-tyyppinen ratkaisu sulautetulla järjestelmällä. Jotta sulautettu järjestelmä olisi soft plc, sen täytyy sisältää tietyt toiminnallisuudet. Tärkein toiminnallisuus on plc-toimintaketju. Plc-toimintaketjuun kuuluu neljä päärakennetta.

Plc-toimintaketjun rakenteet [6]:

- Tarkistaa, onko plc:hen liitetyt tulot muuttaneet arvoaan.
- Ajetaan plc-ohjelma, jossa voidaan toteuttaa laskutoimituksia tai muita loogisia toiminnallisuuksia.
- Päivitetään kaikki lähdöt sen mukaan, miten ajettu ohjelma on asettanut ne.
- Tarkistetaan kaikki laitteet, jotka on liitetty plc:hen.



Kuva 2. Plc-ohjelman suorituskierto

Tätä toimintaketjua toteutetaan niin kauan, kunnes plc sammutetaan tai tapahtuu joku suurempi virhe.

Pääohjelman soft plc -rakenne sisältää seuraavat neljä vaihetta:

- Soft plc -tulojen päivittäminen pääohjelman kautta

Pääohjelman rinnalla toimii plc-säie, joka toteuttaa plc-tyyppistä kiertoa. Tässä osuudessa plc lähettää GET-viestin pääohjelmalle ja jää odottamaan, että pääohjelma palauttaa kaikki sen tietämät tulot/lähdöt takaisin. Jos pääohjelma saa lähetettyä tietyn aikavälin aikana takaisin viestin, se sisältää komennon IO, joka viittaa siihen, että viesti sisältää tuloja ja lähtöjä. Pääohjelma on jo valmiiksi erotellut tulot ja lähdöt omiin python-sanakirjamuuttujiin.

- Soft plc -ohjelman suorittaminen

Saatuaan pääohjelmalta tulot ja lähdöt, käynnistää plc oman ohjelmansa, missä se voi toteuttaa esimerkiksi laskutoimituksia tai muita loogisia tehtäviä. Tällä ohjelmalla on mahdollisuus muokata sisään tullutta lähtömuuttujaa, jos se tahtoo muokata jonkin ulkoisen laitteen lähtöä. Ohjelma voi myös hyödyntää tuloja ja rakentaa ohjelmaa sen ympärille. Kun ohjelma on suoritettu ja kaikki halutut lähdöt päivitetty, siirtyy ohjelma seuraavaan vaiheeseen.

- Soft plc -lähtöjen päivittäminen

Plc lähettää pääohjelmalle takaisin päivitetyn lähtösanakirjamuuttujan. Tämä onnistuu luomalla PUT-viesti, joka lähetetään. Tämä PUT-viesti sisältää itse komennon PUT sekä päivitetyn lähtösanakirjamuuttujan. Kun pääohjelma saa viestin PUT, se jakaa kaikki lähdöt oikeisiin moduuleihin, joista arvot lähtevät eteenpäin päivitettävälle laitteelle. Pääohjelma osaa tunnistaa, mitkä arvot pitää lähettää millekin moduulille tarkistamalla lähtösanakirjamuuttujan avain-arvo-parit, joiden avainosuudessa on moduulin nimi ja arvoparin osuudessa päivitettävät arvot.

- Soft plc -moduulien tarkistus

Moduulien tarkistus on viimeinen osuus soft plc -pääohjelman kierossa. Tässä osuudessa tarkistetaan moduulien eloisuus. Jos jokin moduuli on kaatunut ohjelmakierron aikana, aloitetaan ohjelman lopettaminen, koska plc-ohjelma ei tule enää toimimaan oikein.

4.1.2 Graafinen käyttöliittymä

Graafinen käyttöliittymä tarjoaa python-ohjelmalle helpomman tavan visualisoida kokonaisuutta. Graafisen ilmeen voi luoda käyttötarkoitusta kohden. Tässä graafisessa käyttöliittymässä voi esimerkiksi olla datan visualisointi kaavioilla. Graafisen ilmeen luominen käyttökohdetta kohden onnistuu, koska koko ohjelmalla on pääsy siihen liikkuvaan dataan, jota tulo-/lähtömoduulit tuottavat.

4.2 Moduulit

Nämä python-moduulit ovat ikään kuin pääohjelman palapelin paloja. Kun niitä lisätään, ne tuovat toiminnallisuutta pääohjelmaan. Kaikki moduulit toimivat samojen sääntöjen mukaisesti. Jos pääohjelmaan lisätään uusi moduuli, se ei vaikuta muiden moduulien toimintaan. Moduulien toiminallisuudet voivat olla joko kommunikointia väylän yli sille ulkoiselle laitteelle, jota halutaan ohjata, datan tallentamista, datan siirtoa tietokantaan tai datan visualisoimista. Kaikki moduulit toimivat omissa langoissa tai prosesseissa ja kommunikoivat pääohjelman kanssa hyödyntämällä pythoniin sisään rakennettua jonomoduulia.

4.2.1 Tulo-/lähtömoduulit

Nämä moduulit toimivat pääsääntöisesti välikappaleina tietyn väylän yli ulkoiselle tulo/lähtölaitteelle. Jokainen tulo-/lähtömoduuli päivittää pääohjelmalle tietyn aikavälin sisällä sille kuuluvat tulot/lähdöt.

Jokaisella tulo-/lähtömoduulilla on PUT-komento, joka on pakollinen, koska pääohjelma käyttää tätä komentoa, kun se haluaa lähettää lähtöjä päivitettäviksi. Itse moduulit voivat lähettää muutamia viestityyppejä pääohjelmaan. Tulevaisuudessa näitä lähteviä viestejä voi lisätä moduuleihin tarpeen mukaan.

Tulo-/lähtömoduulien lähettämät viestit:

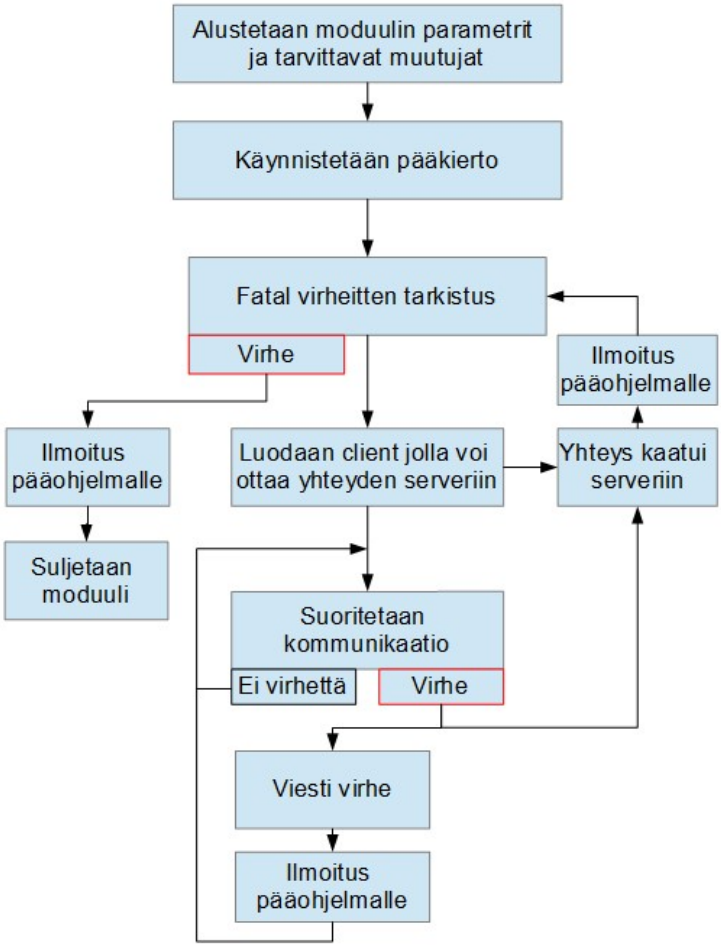
- UPDATE-VAR → Päivitetään tulo-/lähtötiedot pääohjelmaan.

- INFO → Moduulilla on jotain ilmoitettavaa käyttäjälle. Tämä viesti voi sisältää kahta eri tyyppiä info ja warning. Info-viesti ilmoittaa käyttäjälle perustietoja esimerkiksi siitä, että moduuli on käynnistynyt oikein. Warning-viesti kertoo käyttäjälle, että moduulissa on tapahtunut jotain sellaista, mitä ei haluta, mutta se ei ole niin vakavaa, että ohjelma kaatuu.

- ERROR → Moduulissa on tapahtunut virhe, joka liittyy kommunikaatioon väylälaitteelle. Tällöin tästä ilmoitetaan pääohjelmalle ja moduuli pyrkii itse käynnistämään yhteyden uudestaan, jos yhteys on kaatunut. Muita virheitä voivat olla esimerkiksi se, että saatu viesti on virheellinen tai tapahtui aikakatkaisu.

- FATAL → Moduulissa on tapahtunut vakava virhe eikä sitä pystytä pitämään toiminnassa. Kyseisen viestin tullessa kaikki moduulit lopetetaan ja ohjelma pysäytetään.

Jokainen tulo-/lähtömoduulin kommunikaatiokierto on muodostettu samalla periaatteella. Tämä johtuu siitä, että voitaisiin tulevaisuudessa helpommin lisätä uusia tulo-/lähtömoduuleita, jos niitä tarvitaan. Tämä kommunikaatiokierto on hyvin yksinkertainen. Alussa se tarkistaa kaikki viestit, jotka ovat tulleet sisään pääohjelmasta, ja toteuttaa niiden sisältämät tehtävät. Kun nämä viestit on käsitelty, moduuli päivittää kaikki tulot/lähdöt pääohjelmalle, jonka jälkeen se odottaa tietyn ajan, joka on määritetty konfigurointitiedostossa.



Kuva 3. IO-moduulin ohjelma kierto

4.2.1.1 OPC UA -moduuli

Tämä moduuli on rakennettu hyödyntäen avoimen lähdekoodin moduulia nimeltä FreeOpcUa. Tämä FreeOpcUa-moduuli on vapaasti ladattavissa GitHubista. Tämä moduuli antaa pääohjelmalle mahdollisuuden kommunikoida opcua serverin kanssa. Moduulilla voidaan lähettää viestejä serverille sekä vastaanottaa viestejä. Moduulissa on hyödynnetty OPC UA -protokollan tarjoamaa tilaustyyppistä kommunikaatiota. Tämä tarkoittaa sitä, että asiakaskone voi tilata haluttujen solmujen monitorointia ja antaa serverin valvoa näitä kohteita. Serveri ilmoittaa tilaajalle vain, jos solmujen arvot muuttuvat. Tämä vähentää huomattavasti dataliikennettä tilaajan ja serverin välillä [9]. Kun serveri ilmoittaa muutoksesta, muutetaan kyseistä arvoa moduulin omaan datalistaan, mikä palautetaan pääohjelmalle aina kun sitä kysytään GET-komennolla.

Erääseen ongelmaan törmätään, jos käytetään tämän moduulin client-luokkaa luomaan yhteyden serveriin. Ongelma syntyy silloin, kun client on ollut jonkin tietyn aikaa yhteydessä serveriin. Tällöin client tuottaa aikakatkaisuvirheen, joka on bugi kyseisessä moduulissa eikä sitä ole vielä veroissa 0.98.13 päivitetty. Tähän ongelmaan löytyy ratkaisu, joka ei ehkä ole kovin elegantti mutta se toimii. Ideana on luoda clientti ja tehdä muut tarpeelliset konfiguroinnit heti ensimmäisen ikuisen luupin kierossa. Tämän jälkeen tulee uusi ikuinen luuppi, joka tarkastetaan kaikkien virheiden varalta. Kun luuppi kaatuu, se palaa ulkoiselle ikuiselle luupille ja yrittää luoda uuden clientin, millä yhdistyä serverille. Halutessaan voi testata yhteyden luontia monta kertaa. Jos yhteyden luonti ei onnistu, ohjelma lopetetaan tietyn yritysmäärän jälkeen.

Kyseinen moduuli toimii varsin hyvin, jos tahtoo toteuttaa opcua standardin ympärille joko clientin tai serverin. Tarkemmin moduulin tietoihin voi käydä tutustumassa niiden GitHub-sivulla tai niiden dokumentointisivulla. Dokumentointisivulla on kerrottu hyvin toiminnallisuus ja sillä pääsee nopeasti alkuun.

Freeopcua GitHub linkki: https://github.com/FreeOpcUa/python-opcua
Freeopcua dokumentointi linkki: https://python-opcua.readthedocs.io/en/latest/

```
from opcua import client
from time import sleep

reconnects = 0

url='opc.tcp://192.168.0.10:4840'
node = "ns=4;s=|var|AC500 PM56xx-2ETH.Application.opcua_ohjelma.bOpcvar"

ua_client = client.client.Client(url, timeout=1)

while True:
    try:
        ua_client.connect()
        print('Connected')
        test_node = ua_client.get_node(node)
        reconnects = 0
        try:
            while True:
                value = test_node.get_value()
                # Ohjelma tähän while luuppiin
        except KeyboardInterrupt:
            print('KeyboardInterrupt program ends')
            break

    except KeyboardInterrupt:
        print('KeyboardInterrupt program ends')
        break
    except:
        sleep(0.1)
        print("Recconects attempts left >> %d" % (reconnects))
        if reconnects < 10:
            reconnects += 1
        else:
            break
```

Kuva 5. Esimerkki miten kiertää freeopcua aikakatkaisu bugin

4.2.1.2 Modbus TCP/Serial -moduulit

Kummatkin modbus-moduulit hyödyntävät samaa avoimen lähdekoodin moduulia nimeltä pymodbus. Pymodbus on vapaasti ladattavissa GitHubista ja se tarjoaa hyvät työkalut, jotka toteuttavat modbus tcp/rtu -protokollaa. Pymodbus tarjoaa myös modbus ASCII -vaihtoehdon. Koska pymodbus tarjoaa ratkaisuja kaikilla modbus-protokollilla, voidaan moduulit rakentaa samalla tavalla. Vaikka moduulit sisältävät paljon samaa koodia erityisesti siinä, miten dataa luetaan ja kirjotetaan, on moduulit pidetty omissa tiedostoissaan. Näin voidaan erottaa protokollat konfigurointitiedostossa ja eri protokollat sisältävät hieman erilaisia konfigurointimahdollisuuksia. Moduuleille on luotu yhteinen tiedosto, joka sisältää ne funktiot, jolla moduulit luovat niille kuuluvat sanastomuuttujat, joilla voidaan helpottaa modbus-viestien luomista.

Koska modbus-protokolla on yksinkertainen sarjaliikenneprotokolla, se ei anna samoja mukavuuksia, mitä opcua voi tarjota tilaustyyppisellä kommunikaatiolla. Modbus-moduuli sisältää oman ikuisen luupin, joka ajetaan aina tietyin aikavälein. Tämän luupin ensimmäisessä osuudessa se tarkistaa, onko tullitulojen päivitysvaatimusta, ja jos on, se lähettää muutokset laitteelle. Toisessa vaiheessa se toteuttaa koko ajan kaikkien tulojen ja lähtöjen päivittämistä pääohjelmalle. Tämä toistuu tietyin aikavälein, jotka on asetettu konfigurointitiedostoon.

Pymodbus-moduuli tarjoaa paljon valmiiksi rakennettuja funktioita sekä luokkia, joilla niitä voidaan hyödyntää. Moduuli sisältää paljon toiminnallisuuksia ja voi olla haastava ottaa nopeasti käyttöön, kun ottaa huomioon sen, miten yksinkertainen itse modbus-protokolla on. Pymodbus-moduulin suurin vahvuus on se, miten modbus-viestejä voi rakentaa ja harjoittaa eri suuruisiin datatyyppeihin.

Pymodbus GitHub linkki: https://github.com/riptideio/pymodbus
Pymodbus dokumentointi linkki: https://pymodbus.readthedocs.io/en/latest/readme.html

4.2.2 Yleiset moduulit

Yleiset moduulit eivät varsinaisesti vaadi yhtä tarkkaa toteutusta kuin tulo-/lähtömoduulit, koska ne toimivat yleensä täysin eri periaatteella. Moduulit voivat kommunikoida pääohjelmalle. Peruskomennot, jotka löytyvät jo ohjelmasta, ovat informoivia esimerkiksi silloin, jos moduuli tahtoo ilmoittaa virheen log-tiedostoon. Pääohjelmaan voidaan lisätä tarvittaessa uusia komentoja, joita yleiset moduulit voivat hyödyntää. Yleiset moduuleilla on pääsy tulo/lähtö-tietoihin, jos ne tuo kansiosta globaldata-tiedoston data.py. Tämä data.py sisältää python-sanakirjan nimeltä main_data_dict, joka sisältää kaikki tulo/lähtömoduulien arvot. Pääohjelma päivittää tätä python-sanakirjaa.

4.2.2.1 Datan tallentamismoduuli

Datan tallennusmoduulilla on mahdollisuus tallentaa csv, txt-tiedostotyyppeihin. Moduulissa on ikuinen luuppi, joka kerää i/o-data-arvot aina halutuin aikavälein. Moduulilla on

pääsy pääohjelman i/o-dataan, mutta sillä ei ole mahdollisuutta muokata sitä. Data-arvoista luodaan python-sanakirja, missä data järjestetään järkevämpään muotoon tiedostoon tallentamista varten. Tämä python-sanakirja lisätään listaan myöhempää tallentamista varten.

Nämä viestilistat käsitellään moduulissa yksi kerrallaan, jotta saadaan kaikki data tallennettua oikeassa järjestyksessä. Tämä moduuli huolehtii myös siitä, että tiedostot pysyvät tietyn kokoisina. Jos tiedoston koko ylittää annetun konfigurointiarvon, luodaan uusi tiedosto, johon ryhdytään tallentamaan dataa. Moduulille voi myös antaa polun, jonne tiedostot tallennetaan.

Moduuli hyödyntää pandas-nimistä avoimen lähdekoodin moduulia, kun tallennetaan csv -tiedostomuotoon. Pandas-moduulilla voidaan kätevästi luoda datakehys csv-tiedostossa olevasta datasta, tai jos tiedosto on tyhjä, voidaan luoda uusi tyhjä datakehys. Tätä datakehystä voidaan hyödyntää kätevästi, kun halutaan lisätä dataa oikealle sarakkeelle/riville. Kun kaikki haluttu data on lisätty datakehykseen, voidaan se kirjoittaa takaisin csv-tiedostoon hyödyntämällä pandas-moduulia.

Pandas GitHub-linkki: https://github.com/pandas-dev/pandas

Pandas dokumentointilinkki: https://pandas.pydata.org/

4.2.2.2 Ftp-moduuli

Ftp-moduuli antaa tavan, jolla voi ottaa yhteyden ftp-serveriin ja siirtää tiedostoja sinne. Ftp-moduuli on täysin rakennettu pythonista löytyvällä ftplib-moduulilla, joka tulee pythonin mukana, kun python asennetaan koneelle. Moduulin periaate toimii ajastuksella, eli se siirtää esimerkiksi muutaman tunnin tai jopa päivien kuluessa riippuen siitä, mitä konfigurointitiedostossa on asetettu. Kun siirto tapahtuu, se ottaa yhteyden ftp-serveriin ja siirtää kaikki kerätyt datatiedostot ja sulkee yhteyden. Kun kaikki halutut tiedostot on siirretty, se menee takaisin lepotilaan odottamaan seuraavaa sykliä.

4.3 Sovelluksen käyttöalustat

Sovellusta pitää pystyä ajamaan sekä Linux- että Windows-järjestelmässä. Tämän takia sovelluksessa täytyy hyödyntää python-moduuleja, joilla on näiden järjestelmien tuki. Sovellusta voisi esimerkiksi käyttää raspberry pi -piirilevytietokoneella.

5 Käyttöönotto

Ensimmäisessä vaiheessa täytyy varmistaa se, että järjestelmälle on asennettu ne tarvittavat python-moduulit, joita python-sovelluksessa hyödynnetään.

Tarvittavat moduulit:

Python module	Versio	Doc sivusto
pymodbus	2.4.0	https://pymodbus.readthedocs.io/en/latest/
pyserial	3,5	https://pyserial.readthedocs.io/en/latest/
pandas	1.1.5	https://pandas.pydata.org/docs/
opcua	0.98.12	https://python-opcua.readthedocs.io/en/latest/
numpy	1.19.4	https://numpy.org/doc/
Flask	1.1.2	https://flask.palletsprojects.com/en/1.1.x/
Flask-RESTful	0.3.8	https://flask-restful.readthedocs.io/en/latest/

Moduulit asennetaan hyödyntämällä pythonin mukana tullutta pip-asentajaa. Python-sovelluksen kansiostata, missä sijaitsee main.py, löytyy tiedosto nimeltä requirements.txt. Tätä tiedostoa voi käyttää pip-komennolla ja se asentaa kaikki tarvittavat moduulit yhdellä komennolla.

Asennus komento → $pip3\ install\ -r\ requirements.txt$

On huolehdittava siitä, että käyttää komentoa juuri siitä kansiosta, missä tiedosto sijaitsee.

5.1 Konfigurointitiedosto

Konfigurointitiedosto on nimeltään conf_file.ini. Tämä tiedosto sisältää kaikki tarvittavat parametrit moduulien toiminnallisuuteen. Tässä tiedostossa voi esimerkiksi asettaa tarvittavat IP-osoitteet tai valita, haluaako käyttää tiettyä moduulia. Jos tulevaisuudessa luo

uusia moduuleita, tähän tiedostoon lisätään niille tarvittavat parametrit. Tiedostoon täytyy lisätä eri arvot tietyillä etuliitteillä, jotta python-sovellus osaa tulkita eri datatyypit. Lisätietoa parametreistä löytyy konfigurointitiedostosta.

Python-sovellus luo ini-tiedostosta konfigurointi dict-muuttujan, missä kaikki tarvittavat parametrit ovat.

5.1.1 Ini-tiedoston etuliitteet

Jotta python-sovellus osaisi luoda oikealaisen konfigurointi dict-muuttujan, täytyy jokainen datatyyppi kuvailla seuraavilla etuliitteillä.

- b_ → datatyyppi on boolean ja hyväksytyt arvot ovat True/False.
- s_ → datatyyppi on string ja hyväksytyt arvot ovat kaikki unicode merkit.
- i_ → datatyyppi on int ja hyväksyy kokonaislukuja.
- f_ → datatyyppi on float ja hyväksyy reaaliarvoja.
- j_ → polku json-tiedostoon. Tätä käytetään esimerkiksi, kun tuodaan moduuleille parametritietoja modbus-osoitteista.

5.1.2 Monimuotoisemmat muuttujat json-tiedostolla

Jos moduuleihin tarvitaan monimuotoisempia muuttujia, niitä voi muodostaa json-tiedostoon ja asettaa ne variable_files -kansioon. Tältä conf_file.ini tiedosto löytää kyseiset tiedostot, kun tiedostopolun eteen laitetaan → variable_files/. Esimerkiksi jos tiedoston nimi on opcua_variables.json, laitetaan se conf_file.ini -tiedostoon polulla variable_files/ opcua_variables.json. Käytännössä nämä tiedostot voivat sisältää, mitä vain tarvitaan itse moduuliin toimintaan.

5.1.3 Valmiiden moduulien json-tiedostot

Modbus-moduulien:

Kumpaankin modbus-moduuliin tuodaan niiden tulot/lähdöt json-tiedostolla. Nämä muuttujat mallintavat modbus-viestejä, joita moduuli lähettää orjalle. Kentän nimenä voi käyttää mitä vain, joka kuvailee viestiä laitteelle. Arvot, jotka ovat liitettyinä kentän nimeen, tuodaan itse ohjelmaan ja ne prosessoidaan siellä luomalla niistä moduulin sisälle tarvittavat viestit. Jos on tarvetta lisätä enemmän modbus-orjia, niille täytyy lisätä omat modbus json-tiedostot ja linkata ne ini-tiedostossa.

Viestin rakenne:

- slave → Mille orjalle kyseinen viesti halutaan lähettää. Tämä parametri on pakollinen sarjaliikennemoduulille. Jos viestit määritetään modus TCP/IP -moduulille, tämä parametri ei ole pakollinen. Modbus TCP/IP -moduulille voi määrittää yhden orjan ini-tiedostoon.
- function_code → Mitä modbus-funktiokoodia käytetään. Kyseiset moduulit käyttävät vain kela- ja omistusrekisteriä. Hyväksytyt parametrit ovat holding_register, coil.
- Io → Kyseiset muuttujat modbus-muistialueella.

Jokainen Io-muuttuja on muodostettu omasta nimi/arvo-osiosta. Kenttänimeen tulee kyseisen muuttujan nimi, jolla se tunnistetaan ohjelmassa. Arvo-osuus sisältää muutaman parametrin, mitkä helpottavat arvojen tunnistamista ohjelmassa.

Io-muuttujan parametrit:

- addr → Kyseisen muuttujan modbus-muistipaikka.
- data_type → Muuttujan datatyyppi. Tätä parametria käytetään vain, jos kyseessä on omistusrekisteriviesti (holding register). Mahdollisia datatyypejä ovat 16int, 16uint, 32int, 32uint, 32float, 64float ja bool. Huomioitavaa on se, että jos käyttää bool-muuttujaa, se vie kokonaiset kuusitoista bittiä eli yhden ekisterin.

- type → Määrittää onko muuttuja joko tulo tai lähtö. Hyväksytyt arvot ovat input, output.

Valmiit tiedostot:

- modbusTCP_variables.json → Linkitetty conf_file.ini kohtaan modbus-tcp.
- modbusSerial_variables.json → Linkitetty conf_file.ini kohtaan modbus-serial.

```
"modbus_slave3_message1" : {
    "slave" : 3,
    "function_code" : "holding_register",
    "io": {
        "wVar1": {"addr":0,  "data_type" : "16uint",  "type" : "input"},
        "wVar2": {"addr":1,  "data_type" : "16uint",  "type" : "input"},
        "wVar3": {"addr":2,  "data_type" : "16uint",  "type" : "input"},
        "wVar5": {"addr":4,  "data_type" : "16uint",  "type" : "output"},
        "wVar6": {"addr":5,  "data_type" : "16uint",  "type" : "output"},
    }
```

Kuva 6. Modbus holding register viestiesimerkki

Opcua-moduulin:

Opcua-muuttujista ei ole tarvetta muodostaa erillisiä avain-arvoparilohkoja, vaan kaikki opcua-muuttujat ovat omissa avain-arvoparilohkoissa. Avainosaan tulee sen muuttujan nimi, millä se kuvataan ohjelmassa. Projektista löytyy valmis tiedosto nimeltä opcua_variables.json.

Muuttujan rakenne:

- Path → Muuttujan solmun reitti opcua-serverissä.
- Type → Onko muutuja tulo/lähtö hyväksytyt arvot: input/output.

```
{
    "bOpcvar" : {"path" : "ns=4;s=|var|AC500 PM56xx-2ETH.Application.opcua_ohjelma.bOpcvar", "type" : "input"},
    "iOpcvar" : {"path" : "ns=4;s=|var|AC500 PM56xx-2ETH.Application.opcua_ohjelma.iOpcvar", "type" : "output"},
    "rOpcvar" : {"path" : "ns=4;s=|var|AC500 PM56xx-2ETH.Application.opcua_ohjelma.rOpcvar", "type" : "input"}
}
```

Filewriter-moduulin:

Filewriter variables json -tiedosto sisältää tietoa siitä, mitä arvoja tuloja/lähtöjä moduuleista halutaan säästää tiedostoon. Tähän tiedostoon voi määrittää niin monta tiedostoa kuin haluaa. Kentän nimenä voi käyttää mitä vain kuvailevaa nimeä tiedostolle. Projektista löytyy valmis tiedosto nimeltä filewriter_variables.json.

- file-name → Tiedoston nimi, johon arvot tallennetaan.
- file-variables → Niiden tulojen/lähtöjen nimet, jotka halutaan tallentaa tiedostoon. Nämä arvot asetetaan listaan.

```
"file1":{
    "file-name" : "Kokeilu",
    "file-variables" : [
        "wVar3",
        "wVar5",
        "rVar5",
        "rOpcvar"
    ]
}
```

5.2 Yleistä ohjelman käytöstä

Kun konfigurointitiedostossa on asetettu tarvittavat parametrit, ohjelma voidaan yrittää käynnistää. Ohjelmaa on testattu ja sen on todettu toimivan kyseisillä python-versioilla 3.6.4, 3.7.5, 3.9.2. Kyseistä ohjelmaa on testattu myös Windows- sekä linkux-käyttöjärjestelmällä ja todettu toimivaksi.

Ohjelmassa on pyritty huomaamaan kaikki yleisimmät virheet, joita voi tapahtua, ja ilmoitettu tämä käyttäjälle. Syntyneet virheet tallennetaan myös log-tiedostoon, mistä niitä voi käydä tarkistamassa.
Log-tiedosto löytyy kansiosta modules → logfiles.

Suoritettava pääohjelman tiedoston nimi on main,py ja se käynnistetään samoin kuin muut yleiset python skriptit.

6 Tulokset ja ohjelman testaus

Ohjelman testausalustana toimi raspberry pi 3. Tämä raspberry pi 3 toimi soft plc:nä, mihin oli väylälaitteina ABB pm5650 PLC, jossa on modbus TCP -serveri, modbus RTU -orja sekä opcua-serveri. Koska ABB pm5650 PLC ei antanut tukea modbus ASCII:lle, toisena väylälaitteena oli ABB pm573, jossa on modbus ASCII -orja. ABB plc ei sinänsä sisältänyt suurempia ohjelmia vaan pelkästään muuttujia, joilla pystyi testaamaan yhteyttä isäntälaitteelle raspberry pi 3.

Testauksessa tarkistettiin myös datan tallentamismoduulia ja ftp tilaajaa (eng. client), joka siirtää tiedostot ulkoiselle ftp-serverille. Ftp-serverinä toimi Linode-yrityksen tarjoama serveripalvelu. Testausta varten tälle Linode-serverille luotiin pythonilla hyvin yksinkertainen ftp-serveri.

Ensimmäisessä vaiheessa virheitä pyrittiin luomaan tahallaan irrottamalla väylälaite sekä lähettämällä vääriä datatyyppejä. Tällaisella testausmenetelmällä saadaan poistettua turhia virheitä, joista on jo tietoa sekä poistamaan bugeja, joita ohjelman kaatuminen voi aiheuttaa. Bugeina voi esimerkiksi olla, ettei ohjelma sulkeudu oikein ja joku moduuli jää vielä taustalle käyntiin.

Syntyneitä bugeja ensimmäisen vaiheen testauksessa:

- Väylälaitteet eivät sulkeutuneet oikein.
- Tiedostoihin kirjottava moduuli jäi päälle ja jatkoi tiedostoihin kirjoittamista.

Kun kyseiset ongelmat oli saatu ratkaistua, pystyttiin jatkamaan toiseen vaiheeseen.

Toisessa vaiheessa lähdettiin testaamaan koko ohjelman toimivuutta ilman että sille tuotettiin mitään ulkoisia häiriöitä. Ohjelman annettiin suorittaa niin kauan, kunnes siinä tapahtui jokin virhe. Jos virheitä ilmeni, ne korjattiin, jonka jälkeen aloitettiin ohjelman suorittaminen uudestaan. Tavoitteena on saada sellainen ohjelma, joka pystyisi suorittamaan itseänsä toistuvasti ja toimimaan niin kauan, kunnes se lopetetaan manuaalisesti.

Suurimat ongelmat ja bugit sai poistettua ensimmäisessä testivaiheessa. Toisessa testausvaiheessa suurimmat ongelmat tuottivat opcua-moduuli ja sen yhteyden ylläpitäminen. Jos yhteys kaatui ja yhteyttä yritti uudelleen luoda, se ei enää päivittänyt arvoja, vaikka ne serverin puolella muuttuisi. Tämä oli bugi freeopcua-moduulissa ja sen sai kierretty, jos loi kokonaan uuden instanssin freeopcua moduulin client luokasta. Tällöin opcua-moduuli pystyi onnistuneesti luomaan uuden toimivan yhteyden serveriin.

Toista testiä suoritettiin kahden viikon aikajaksoissa. Lopulta kun ohjelma onnistui stabiilisti toimia ilman ongelmia kahden viikon ajan, oli ohjelma kokonaisuus insinöörityö-projektin osalta valmis.

6.1 Lopullinen ohjelma

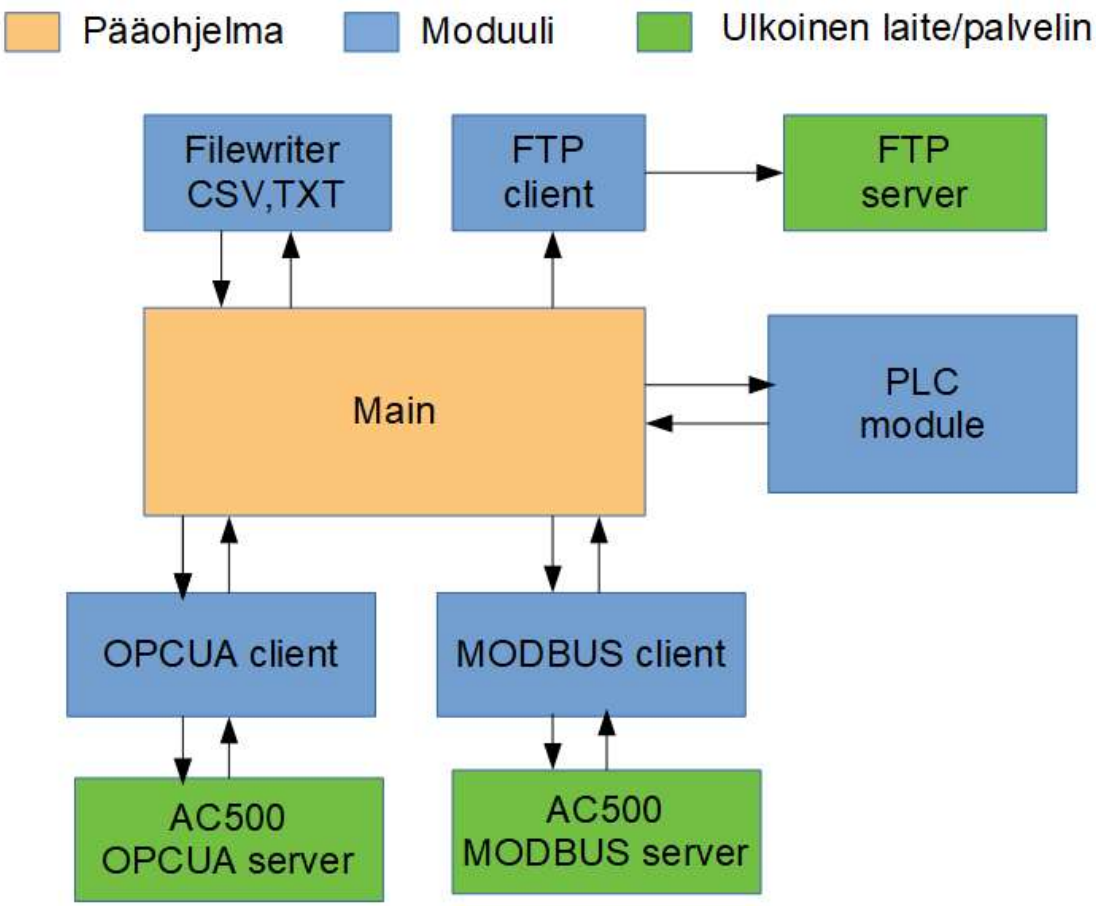
Lopullisessa ohjelmassa toteutettiin runko, jonka päälle voi jatkossa lisätä enemmän toiminnallisuuksia ja toisia moduuleita. Itse pääohjelma jäi yksinkertaiseksi ja toimi vain välikätenä muiden moduulien välillä.

Testauksen jälkeen toimiviksi moduuleiksi todettiin seuraavat:

- Modbus TCP/IP -asiakas
- Modbus Serial RTU/ASCII -asiakas
- OPC UA -asiakas
- Moduuli, joka tallentaa dataa CSV-tiedostoon
- PLC-moduuli

Ohjelmassa luotiin myös soft plc-moduuli, joka imitoi oikean plc-ohjelman kiertoa. Kyseinen plc-moduulin toiminnallisuuksiin kuuluu tulojen/lähtöjen lukeminen pääohjelmalta, kyky suorittaa plc-tyyppisiä ohjelmia sekä lähtöjen muuttamista ja lähettää ne päivitettäväksi tulo-/lähtömoduuleille. Plc-rakenne tarvitsee vielä lisäystä sekä mahdollisesti uudelleen muokkausta paremman toiminallisuuden takaamiseksi. Plc-moduulista jäi puuttumaan myös tavallisimmat funktion blokit, jotka löytyvät normaalista plc-ohjelmointiympäristöstä, kuten esimerkiksi TON, TOF.

Ohjelmalle luotiin myös FTP-asiakasmoduuli, joka voi lähettää tiedostoja FTP-palvelimelle. Moduuli jäi hieman keskeneräiseksi, koska sen avulla ei voida hakea tiedostoja FTP-palvelimelta. Moduuli ei myöskään tarjoa turvallista kommunikaatiota, jonka takia sitä pitäisi parantaa.



Kuva 7. Koko luodun ohjelman hahmotus

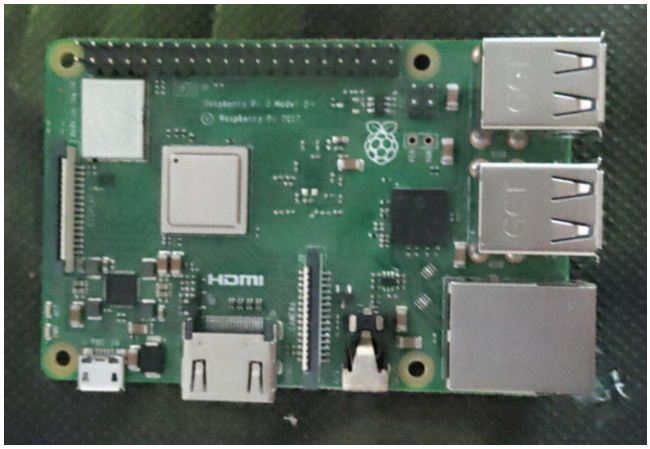
6.2 Testauksessa käytetyt laitteet

6.2.1 Raspberry pi 3 B

Raspberry pi 3 b+ on pieni yhden piirilevyn tietokone, jolla voidaan esimerkiksi käyttää linux-käyttöjärjestelmää. Tämä pieni tietokone on yllättävän tehokas ja sitä voi käyttää kuten normaalia tietokonetta. Raspberry tarjoaa paljon erilaisia teknologioita, joita voi käyttää projektien testaamiseen, kuten tässä kyseisessä insinöörityössä. Kyseisen laitteen hinta on minimaalinen, uutena noin 45 euroa. Kyseisestä laitteesta on tullut markkinoille uudempi versio 4.

Raspberry pi 3 spesifikaatio:

- Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
- 1GB LPDDR2 SDRAM
- 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE
- Gigabit Ethernet
- 40 GPIO pinniä, millä voi ohjata elektroniikkaa
- HDMI-portti
- 4 USB 2.0 -porttia
- Micro SD lakia, millä voidaan tuoda käyttöjärjestelmä raspberrylle
- 5V/2.5A DC -ottoportti



Kuva 8. Kyseinen raspberry pi B+, mitä käytettiin testauksessa

6.2.2 AC500 PM5650

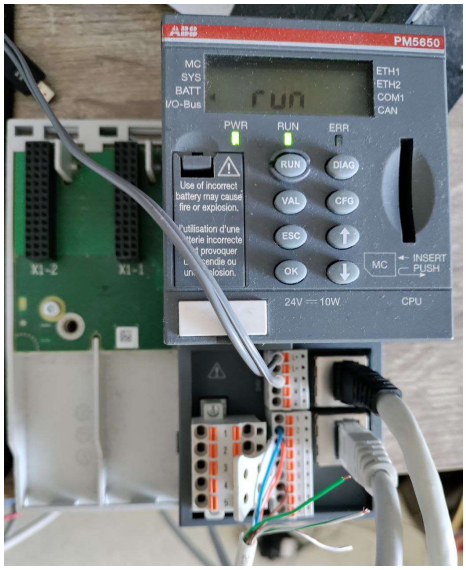
ABB AC500 PM573 V2 PLC ohjelmoidaan ABB:n tarjoamalla Automation Builder ohjelmalla. Kyseiseen ABB PLC -malliin voidaan lisätä kaksi ylimäärästä väyläkorttia, jos siihen on tarvetta. Analogisten tai digitaalisten tulojen/lähtöjen maksimimäärää ei määritetty ABB-sivuilla. Kyseinen prosessori yhdistetään TB5620-2ETH terminaalialustaan.

TB5620-2ETH Linkki: https://new.abb.com/products/1SAP112300R0278/tb5620-2et-hac500-terminal-base-2xslots

PM5650-2ETH linkki: https://new.abb.com/products/1SAP141000R0278/pm5650-2ethac500-cpu-80mb-24vdc-eth

ABB AC500 tarjoamat väylät:

- Kaksi Ethernet TCP/IP, johon voi konfiguroida ModbusTCP/IP clinet/server, FTP server, SNTP client/server, Web server, OPC UA server.
- Sarjaliikenne portti 1 (RS232/RS485), johon voi konfiguroida Modbus RTU isäntä ja orja.
- Can väyläterminaali



Kuva 9. Testauksessa käytetty ABB AC500 PM5650 PLC

6.2.3 AC500 PM573

ABB AC500 PM573 V2 PLC ohjelmoidaan ABB:n tarjoamalla Automation Builder ohjelmalla. Kyseiseen ABB PLC -malliin voidaan lisätä yksi ylimääräinen väyläkortti, jos siihen on tarvetta. Analogisia tuloja/lähtöjä voi kumpiakin olla enintään 160 kappaletta. Digitaalisia tuloja voi olla enintään 320 kappaletta ja digitaalisia lähtöjä 240 kappaletta. Kyseinen prosessori yhdistetään TB511-ETH-terminaalialustaan.

PM573 linkki: https://new.abb.com/products/1SAP130300R0271/pm573-ethac500-prog-logic-contr-512kb

TB511-ETH linkki: https://new.abb.com/products/1SAP111100R0270/tb511-ethac500-terminal-base-1slot

ABB AC500 tarjoamat väylät:

- Ethernet TCP/IP, johon voi konfiguroida ModbusTCP/IP clinet/server, FTP server, SNTP client/server, Web server.
- Sarjaliikenne portti 1 (RS232/RS485), johon voi konfiguroida Modbus RTU/ASCII isäntä ja orja, CS31 isäntä.
- Sarjaliikenne portti 2 (RS232/RS485), johon voi konfiguroida Modbus RTU/ASCII isäntä ja orja.
- FBP portti, johon voi konfiguroida CANopen, DeviceNet, PROFIBUS orjan.

Kuva 10. Testauksessa käytetty ABB AC500 PM573 PLC

7 Yhteenveto

Insinöörityön tarkoituksena on luoda python-moduuleja, joita voidaan yhdistää pääohjelmaan tiettyjen sääntöjen perusteella. Moduulien idea on tuoda tulo/lähtö -toiminallisuutta pääohjelmaan, jotta se voisi myös toimia soft plc:nä. Moduuleina voi toimia myös niin sanottuja yleisiä toimintoja, kuten kerätyn datan siirto tiedostoon.

Työssä hyödynnettiin valmiiksi GitHubista ladattavia python-kirjastoja, jotta saataisiin varsinkin tulo-/lähtömoduulit kehitettyä nopeasti. Työssä pyrittiin myös minimoimaan muiden tekemiä kirjastoja ja aina mahdollisuuksien mukaa hyödyntämään pythonista valmiiksi löytyviä kirjastoja.

Työssä syntyi muutamia toimivia moduuleita sekä pääohjelmaperiaate, jonka perusteella voitaisiin lisätä tulevaisuudessa muita moduuleja.

7.1 Kehitystarpeet

Kehitetyissä tulo-/lähtömoduuleissa saatiin vain kehitettyä itse isäntäosuus. Kyseisistä protokollista täytyisi vielä luoda orjatoiminnallisuus moduuleihin. Projektissa luotiin myös plc-kiertoa muistuttava osuus. Vaikka se toteutui, sen tarvittavat piirteet jäivät hieman yksinkertaisiksi ja siksi sitä pitäisi vielä työstää, jotta se voisi toimia monipuolisissa kohteissa.

Projektiin mietittiin graafista käyttöliittymää. Tätä ei saatu toteutettua itse projektissa, koska aika ei siihen riittänyt. Seuraavassa vaiheessa tästä voisi jatkaa luomalla esimerkiksi djangon avulla verkkoselaintyyppisen graafisen käyttöliittymän.

8 Lähdeluettelo

[1] Luukkainen, M. Ohjelmistotuotanto 2020. Verkkoaineisto.
<https://ohjelmistotuotanto-hy.github.io/osa1/>. 2020.

[2] Softwarecrisis. Verkkoaineisto. Wikipedia.
<https://en.wikipedia.org/wiki/Software_crisis> Päivitetty 7.4.2021.

[3]Vesiputousmalli.Verkkoaineisto.Wikipedia
<https://fi.wikipedia.org/wiki/Vesiputousmalli> Päivitetty 7.11.2020.

[4] Ketterä ohjelmistokehitys. Verkkoaineisto. Wikipedia.
<https://fi.wikipedia.org/wiki/Ketter%C3%A4_ohjelmistokehitys> Päivitetty 6.11.2020.

[5] Dhruvil Karani. How does Python work?. Verkkoaineisto.
<Verkkoaineistohttps://towardsdatascience.com/how-does-python-work-6f21fd197888>
9.1.2020.

[6] Technology transfer, The Basics of PLC Operation. Verkkoaineisto.
<https://www.techtransfer.com/blog/basics-plc-operation/>.

[7] Modbus.Verkkoaineisto. Wikipedia
<https://fi.wikipedia.org/wiki/Modbus> Päivitetty 19.3.2021.

[8] Modbus tools, Protocol Description. Verkkoaineisto. Modbus tools.
<https://www.modbustools.com/modbus.html>.

[9] OPC UA Subscription Consept. Verkkoaineisto. Unified Automation.
<http://documentation.unifiedautomation.com
/uasdkc/1.4.1/html/L2UaSubscription.htm>.

[10] Unified Architecture. Verkkoaineisto. OPC Foundation.
<https://opcfoundation.org/about/opc-technologies/opc-ua/>.

[11] Olivier Roulet. Python OPC-UA Documentation. Verkkoaineisto. <https://python-opcua.readthedocs.io/en/latest/>. 2015.

[12] Sanjay Revision. PyModbus - A Python Modbus Stack. Verkkoaineisto. <https://pymodbus.readthedocs.io/en/latest/readme.html>. 2017.

[13] pandas. Verkkoaineisto. <https://pandas.pydata.org/>.