Tampereen yliopisto

Pyry Vehmas

Modbus TCP/IP-pohjainen moottorinohjaussovellus Raspberry Pi:llä

Kandidaatintyö
Tekniikan ja luonnontieteiden tiedekunta
Tarkastaja: Mikko Salmenperä
Huhtikuu 2021

TIIVISTELMÄ

Pyry Vehmas: Modbus TCP/IP-pohjainen moottorinohjaussovellus Raspberry Pi:llä
Kandidaatintyö
Tampereen yliopisto
Teknisten tieteiden kandidaatin tutkinto-ohjelma
Opinnäytetyö 31 sivua + 6 liitesivua
Huhtikuu 2021

Tässä kandidaatintyössä perehdytään Modbus TCP/IP-protokollan toimintaan, ja selvitetään Raspberry Pi:n käyttömahdollisuutta toimia Modbus TCP/IP-verkon asiakaslaitteena. Työn sisällön voi jakaa kahteen erilliseen osaan: luvut 2-4 tarjoavat työn teoriaosan ja luku viisi tarjoaa selvityksen työn käytännönosuudesta, jossa toteutettiin Modbus TCP/IP-pohajainen moottorinohjaussovellus Raspberry Pi:llä.

Luvussa kaksi perehdytään automaatiojärjestelmien peruselementteihin, automaation luokittelutapoihin, sekä fyysisen kerroksen kahteen eri tiedonsiirtotapaan: kenttäväyliin ja teolliseen Ethernettiin. Luku kolme keskittyy tiedonsiirtoprotokolliin, sekä protokollien kerrosmaleihin. Luvussa esitellään kaksi tunnettua kerrosmallia: OSI- ja TCP/IP-viitemallit. Kerrosmallien lisäksi luvussa käydään läpi myös IP- ja TCP-protokollien perustoimintaperiaatteita. Luku neljä jatkaa protokollateemasta, mutta keskittyy työn kannalta oleellisimpaan protokollaan, eli Modbus-protokollaan. Luvun alussa käydään läpi protokollan yleinen toimintaperiaate, sekä Modbus-viestin rakenne. Tämän jälkeen keskitytään protokollan tuoreimpaan, sekä tässäkin työssä käytettyyn Modbus TCP/IP-versioon, ja selvitetään sen eroavaisuutta kahteen aiempaan, eli Modbus ASCII- ja RTU-versioihin.

Luvussa viisi käydään läpi työn käytännönosuudessa toteutettua Modbus TCP/IP-sovellusta. Luvun alussa tutustutaan sovelluksessa käytettyihin laitteisiin, sekä ohjelmistoihin. Laitteistoselvityksen jälkeen alkaa osuus, missä käydään läpi itse sovellusohjelman toteuttamista ja sen dokumentaatiota. Dokumentaatiossa apuna on käytetty erilaisia toimintaa havainnollistavia UML-kaaviota, kuten: luokka-, käyttötapaus-, tila- ja sekvenssikaaviota. Työn loppuun on vielä lisättynä liite, joka sisältää sovellusohjelman lähdekoodin.

Avainsanat: Modbus, Modbus TCP/IP, Raspberry Pi, Python, Pymodbus

SISÄLLYSLUETTELO

KUVALUETTELO

LYHENTEET JA MERKINNÄT

.

ADU	Application Data Unit
ASCII	American Standard Code for Information Interchange
CRC	Cyclic Redundancy Check
DNS	Domain Name System
ERP	Enterprise Resource Planning
FTP	File Transfer Protocol
HDMI	High-Definition Multimedia Interface
HTTP	Hypertext Transfer Protocol
HVAC	Heating, ventilation, and air conditioning
IEC	International Electrotechnical Commission
IP	Internet Protocol
ISO	International Organization for Standardization
LAN	Local Area Network
LRC	Longitudinal Redundancy Check
MBAP	Modbus Application Protocol
MES	Manufacturing Execution System
MTU	Maximum Transmission Unit
OSI	Open Systems Interconnection
PDU	Protocol Data Unit
PLC	Programmable Logic Controller
RS	Recommended Standard
RTU	Remote Terminal Unit
SCADA	Supervisory Control and Data Acquisition
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol and Internet Protocol
UDP	User Datagram Protocol
USB	Universal Serial Bus

1. JOHDANTO

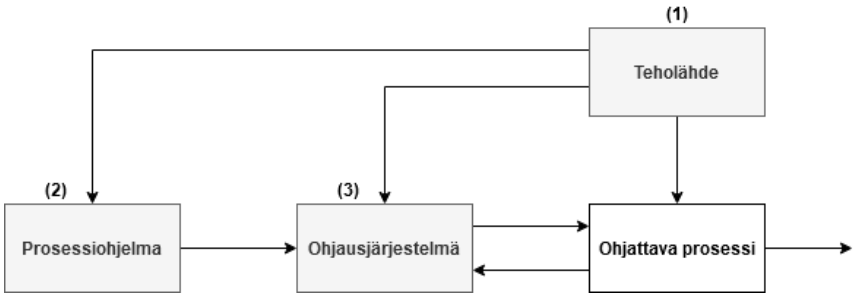
Jatkuva automaatioasteen kasvu, sekä järjestelmien kehittyminen luo paineen uusien tekniikoiden ja sovellusten kehitykselle. Yrityksillä, sekä muilla tutkimusta tekevillä tahoilla on tarve löytää kehitystyöhön mahdollisimman kustannustehokkaita ratkaisuja. Viimeisen kymmenen vuoden aikana kuluttajamarkkinoille on tullut runsaasti erilaisia yhden piirilevyn mikrokontrolleri- ja tietokoneratkaisuja. Näistä mahdollisesti tunnetuimpia ovat Arduino sekä Raspberry Pi. Vaikka kyseiset laitteet eivät välttämättä olisikaan soveltuvimpia ratkaisuja ankariin teollisuusolosuhteisiin, ovat ne kuitenkin herättäneet kiinnostusta eräänlaisina prototyyppi- ja kokeilualustaratkaisuina. Laitteiden helppo saatavuus ja erityisesti alhainen hinta madaltaa uuden sovelluksen kehittämiseen liittyvää riskiä.

Tässä kandidaatintyössä perehdytään Modbus TCP/IP-protokollan toimintaan, ja selvitetään Raspberry Pi:n käyttömahdollisuutta toimia Modbus TCP/IP-verkon asiakaslaitteena. Käyttömahdollisuuden selvitys tapahtuu viidennessä luvussa, eli työn käytännön osuudessa, jossa toteutetaan kahden Vacon 100 HVAC taajuusmuuntajan kanssa Modbus TCP/IP-kommunikaatiota käyttävä moottorinohjaussovellus Raspberry Pi:lle. Itse sovellusohjelman toteuttamiseen käytetään Python-ohjelmointikieltä, sekä Python-ohjelmille tarkoitettua Pymodbus-ohjelmakirjastoa.

Työn käytännönosuudessa pyritään näin ollen siis kuvaamaan Raspberry Pi:lle toteutettu ohjelmointiprojekti, joka sisältää Modbus TCP/IP-asiakaskomponentin lisäksi myös käyttöliittymätoiminnallisuuden, jolla sovellusta voidaan käyttää. Työn kirjoittajalla oli aiempaa kokemusta Python-ohjelmointikielestä, sekä hieman Raspberry Pi-laitteista, mutta ei aiempaa kokemusta Modbus TCP/IP-protokollasta, tai sen toimintaperiaatteesta ennen kyseistä työtä. Kuitenkin ennen käytännönosuutta työ jatkuu teoriaosuudella.

2. AUTOMAATIOTEKNIIKAT

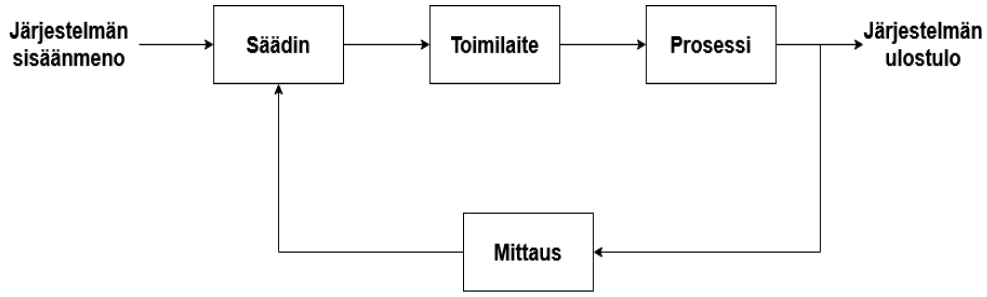
Tarkasteltaessa automaatiojärjestelmiä ne sisältävät yleisesti kolme peruselementtiä. Näitä ovat teholähde, prosessin suorittava ohjelma sekä itse ohjausjärjestelmä, joka suorittaa kyseisen ohjelman. [1, s. 78]



Kuva 2. Automaation peruselementit (1) teholähde, (2) prosessiohjelma ja (3) ohjausjärjestelmä. Mukailen lähdettä [1, s. 78]

Ohjausjärjestelmät jaetaan käytännössä kahteen eri ryhmään. Avoimiin säätöpiireihin ja suljettuihin säätöpiireihin. Avoin säätöpiiri ohjaa prosessia suoraan ilman takaisinkytkentää, eikä sen tuottama ohjaus ole riippuvainen prosessin ulostulosta. Avoimeen piiriin perustuvan järjestelmän valinta voi tulla kyseeseen silloin, kun esimerkiksi prosessihäiriöt ovat erittäin harvinaisia. Tällöin yksikertaisempi ratkaisu voi olla parempi. Esimerkiksi tämän kyseisen työn käytännön osuudessa taajuusmuuntaja ohjaa sähkömoottoria juuri avoimen säätöpiirin avulla ilman arvoja mittaavia antureita.

Toisin kuin avoin järjestelmä, suljettu järjestelmä hyödyntää prosessista saatua mittausarvoa ohjaukseen. Järjestelmä muodostaa negatiivisen takaisinkytkentäsilmukan. Toimilaitetta ohjaavaa säädin vastaanottaa niin sanotun erosuureen, joka on asetusarvon ja mittausarvon välinen erotus. Valtaosa teollisuuden prosessiohjauksista edustaa kyseistä ratkaisua. [2, s. 5-8]



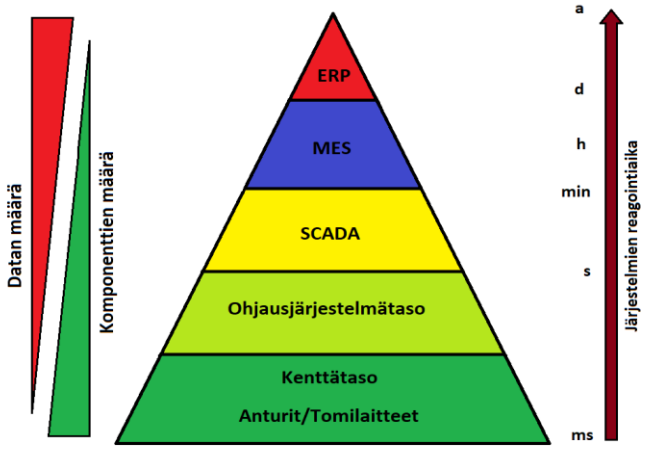
Kuva 2. Mitatun arvon hyödyntämiseen perustuva suljettu säätöpiiri. Tämän työn käytännön osuudessa taajuusmuuntaja säätää moottoria ilman kyseistä takaisinkytkentäsilmukkaa säätimelle. [1, s. 84]

2.1 Automaation tasot

Suuret automaatiojärjestelmät voivat sisältää satoja, ellei tuhansia automaatiolaitteita, jotka voivat olla hajautettuna suurelle alueelle. Laitteiden tuottamaa datavirtaa ja automaation eri hierarkiatasoja perinteisesti kuvataan käsitteellä nimeltään automaatiopyramidi. Pyramidin pohjalla erilaisten automaatiokomponenttien määrä on suurin, niiden vähentyessä aina ylöspäin mentäessä kohti huipulla olevaa toiminnanohjaustasoa. Sen sijaan datan määrä ja järjestelmien reagointiajat käyttäytyvät käänteisesti, niiden kasvaessa ylöspäin mentäessä kohti pyramidin huippua. [3, s. 1]

Pyramidin alin taso eli kenttätaso sisältää prosessia ohjaavat toimilaitteet, sekä sitä mittaavat anturit. Kenttätason laitteiden ohjaus tapahtuu ylemmällä tasolla, eli ohjausjärjestelmätasolla**.** Yleisesti ohjausjärjestelmätason laitteet, kuten PLC-ohjaimet pyörittävät ohjelmia mittausinformaatiota hyödyntäen. Näin ne ohjaavat kenttätason laitteita suorittamaan prosessia niille asetettuun arvoon. Kommunikaatio kahden alimman tason välillä on tyypillisesti toteutettu esimerkiksi erilaisilla kenttäväyläratkaisuilla. Ohjausjärjestelmätason yläpuolella on valvontataso, tai niin sanottu SCADA-järjestelmä (Supervisory Control and Data Acquisition). Järjestelmän tehtävä on valvoa, sekä ohjata suurempaa kokonaisuutta teollisuusympäristössä. Se on yleisesti PC-pohjainen ratkaisu, johon on toteutettu datanvisualisointi- ja säilöönottosovelluksia. [3, s. 1-6]

Valvontatasoa ylempänä on kaksi ylintä tasoa: tuotannon- ja toiminnanohjaustasot (MES ja ERP). Näiden luonteeseen tyypillisesti kuuluu se, että ne sijaitsevat fyysisesti toisaalla itse tuotannosta. Tuotannonohjaustasolla tuotannonohjausjärjestelmä valvoo tuotantoa. Esimerkiksi tuotannon optimointi ja aikataulutus erilaisia tuotantoeriä varten tapahtuu tuotannonohjausjärjestelmässä. Ylin taso, eli toiminnanohjaustaso sisältää yrityksen johdolle tarkoitettuja työkaluja, kuten liiketoiminnan ja logistiikan suunnittelutoiminnot. [4]



Kuva 3. *Automaation eri tasoja kuvaava automaatiopyramidi. [3, s. 1]*

2.2 Kenttäväylät

Ensimmäiset kenttäväyläratkaisut otettiin käyttöön 1980-luvulla. Tekniikka yksinkertaistaa vanhoja ratkaisuja, joissa tiedonsiirto hoidettiin laitteiston ja ohjausjärjestelmien välillä kaapeloimalla jokainen komponentti erikseen. Uusi tekniikka mahdollisti, että yhteen kaapeliin eli väylään kytkettiin useampia laitteita samanaikaisesti. Digitalisoituneet kenttäväylät myös mahdollistivat uusien ratkaisujen ja aiempaan älykkäämpien laitteiden käytön, sekä molemminpuolisen kommunikaation laitteilta ohjaimille.

Vaikka ensimmäisten kenttäväylien kehityksestä on kulunut jo lähes 40 vuotta, ei vielä nykyaikanakaan ole täysin yksiselitteistä, mitä termi kenttäväylä tarkalleen pitää sisällään. Tämä selittynee eri valmistajien eriävistä käsityksistä ja ratkaisuista. Fieldbus Foundationin selitys termille on seuraava: ”Kenttäväylä on digitaalinen, kaksisuuntainen, monihaarainen kommunikaatiolinkki älykkäiden mittaus- ja ohjauslaitteiden välillä. Se tarjoaa paikallisverkko (LAN) ratkaisun kehittyneelle prosessi ohjaukselle, etäohjatulle I/O-liikenteelle ja nopeille tehdasautomaatiosovelluksille”. [5, s. 30-34]

2.3 Ethernet ja teollinen Ethernet

Ethernet on Xerox PARC -yhtiön 1970-luvulla kehittämä lähiverkkotekniikka. Myöhemmin Ethernet-tekniikoita varten on muodostettu IEEE 802.3 standardi. [6] Ethernet on yleinen lähiverkkoratkaisu, jolla toteutetaan tiedonsiirron fyysinen siirtoyhteys. Siitä on kehittynyt niin sanottu ”de facto” -standardi IT-sektorilla. Onkin siis varsin helppoa ymmärtää, miksi Ethernet-tekniikka on siirtynyt koti- ja toimistoympäristöistä palvelemaan myös teollisuuden tarpeita. Teollinen Ethernet soveltaa Ethernetin standardeja teolliseen käyttötarpeeseen. Se tarkoittaa koti- ja toimistoympäristöä vaikeampia olosuhteita, joten fyysisen laitteiston pitää kestää esimerkiksi korkeita lämpötiloja, kosteutta tai tärinää. [7, s. 1-3]

Teollisen Ethernetin käyttömuotoja on useita, mutta ne voidaan jakaa karkeasti neljään eri kategoriaan. [7, s. 4]

1. Täysi Ethernet, joka käyttää täsmälleen IEEE 802.3 standardin ratkaisuja.
2. Ethernet-yhteensopivat ratkaisut.
3. Erottavan portin kautta Ethernet-verkkoon liitetyt laitteet.
4. Ethernet-linkkejä käyttävät kenttäväylät.

3. TIETOLIIKENNEPROTOKOLLAT

Protokolla on säännöistä koostuva standardi, jota tietokoneet käyttävät lähettäessään tietoa edestakaisin. Tiedonsiirrossa on tärkeää, että siihen osallistuvan lähettäjän, ja vastaanottajan on noudatettava sekä tunnistettava samoja sääntöjä, eli protokollaa. [8, s. 14]

Tiedonsiirtoa käsittelevät mallit yleensä pyritään selitettään käyttäen erilaisia kerrosmalleja, jotka rakentuvat useammista erillistä protokollista. Kerrosmallien käyttäminen vähentää selvästi asioihin liittyvää turhaa monimutkaisuutta. [9, s. 1] Näistä kerrosmalleista ehkäpä tunnetuimpia ovat OSI-viitemalli, sekä internetin pohjana toimiva TCP/IP-viitemalli.

3.1 OSI-viitemalli

OSI-viitemalli (Open Systems Interconnection Reference Model) on kansainvälisen standardijärjestö ISO:n (International Organization for Standardization) luoma käsitteellinen malli. Malli kuvaa tietoliikenneverkon arkkitehtuuria, joka mahdollistaa tiedon siirtämisen tietokonejärjestelmien välillä. Vaikka malli on vain käsitteellinen malli, sen tarkoituksen ja toiminnan ymmärtäminen voi auttaa ymmärtämään eri protokollien sekä verkkoarkkitehtuurien toimintaa käytännössä. [10, s. 134-135]

OSI-viitemalli koostuu seitsemästä eri kerroksesta, joista jokaisella on tietty toiminto. Kukin kerroksista käyttää yhtä alempana olevan kerroksen tarjoamia palveluita, sekä tarjoaa palvelunsa seuraavalle kerrokselle. Malli rakentuu alhaalta ylöspäin seuraavassa järjestyksessä: fyysinen kerros, siirtoyhteyskerros, verkkokerros, kuljetuskerros, istuntokerros, esitystapakerros ja sovelluskerros. Fyysinen kerros luokitellaan kerrokseksi yksi, ja sovelluskerros seitsemänneksi kerrokseksi. [11, s. 2-8] Seuraavaksi OSI-viitemallin seitsemän kerrosta selitettynä lyhyesti, jatkuen seuraavalle sivulle.

1. **Fyysinen kerros**

   OSI-viitemallin ensimmäinen kerros, eli fyysinen kerros määrittelee tietoliikenteen mahdollistavat fyysiset tekijät, kuten laitteistot ja verkkotopologian. Laitteisto-käsite pitää sisällään esimerkiksi millaisia tietoliikennekaapeleita käytetään, tai käytettävien liittimien tyypit. [10, s. 135]

2. Siirtoyhteyskerros

Siirtoyhteyskerros on fyysisen kerroksen ja ylempien kerrosten välissä oleva välikerros, jossa ylempien kerrosten tietoliikennepaketit kehystetään fyysistä kerrosta varten. Kerroksen tehtäviä ovat myös virheiden tarkastus, niiden korjaaminen sekä laitteiston osoittaminen. [10, s. 136]

3. Verkkokerros

Verkkokerroksen ensisijainen tehtävä on pakettien reitittäminen. Tarjottava toiminnallisuus saadaan reititysprotokollien avulla, jotka ovat toiminallisia ohjelmistokomponentteja. Reitittämisessä käytetään apuna alemman kerroksen ohjelmistokonfiguroituja osoitteita. [10, s.136]

4. Kuljetuskerros

Kuljetuskerros asettaa kommunikaatiolle päittäiset päätepisteet. Tämä tarkoittaa sitä, että dataa ei reititetä enempää toisille laitteille vaan se siirretään ylemmille kerroksille. [5, s. 40] Datan kuljettamiseksi verkossa datalohkot ovat hajotettava paketeiksi, joiden kokoa on mahdollista hallita alemmilla kerroksilla. Pakettien hajottaminen sekä uudelleen kokoaminen on kuljetuskerroksen vastuulla. Kerros varmistaa, että paketit toimitetaan ylemmille kerroksille oikeassa järjestyksessä. [9, s. 4]

5. Istuntokerros

Kerroksen tehtävä on synkronoida tiedonsiirto erillisten laitteiden ja sovellusten välillä toimivaksi. [10, s. 138]

6. Esitystapakerros

Esitystapakerroksen perustoiminto on muuntaa sovelluskerroksesta saatua tai sinne tarkoitettua dataa toiseen muotoon. Data muuntaminen toiseen muotoon on välttämätön toimenpide, jotta sen kuljettaminen onnistuisi verkon läpi. Kerroksen toinen merkittävä tehtävä on datan salaaminen, sekä salauksen purkaminen. [10, s. 138]

7. Sovelluskerros

Sovelluskerros määrittelee prosessit, jotka mahdollistavat sovelluksen käyttäjän käyttää verkkopalveluita. Yksinkertaistettuna kerroksen tarkoitus on ottaa käyttäjältä vastaan pyyntöjä tai dataa, ja välittää niitä OSI-viitemallin alemmille kerroksille. [10, s. 138-139]

3.2 TCP/IP-viitemalli

Toisin kuin OSI-viitemalli, TCP/IP-mallia ei kehitetty aluksi pelkäksi malliksi, vaan se syntyi jo käytössä olevista protokollista [9, s. 5] TCP/IP-viitemallissa on neljä kerrosta, joista kolmella on lähes suora vastine OSI-viitemallissa. Malli rakentuu alhaalta ylöspäin seuraavasti: peruskerros, verkkokerros, kuljetuskerros ja sovelluskerros. [8, s. 24] Vastinkerroksista huolimatta malleilla on myös selvät eronsa. Esimerkiksi OSI-viitemalli erottelee selvästi palvelut, rajapinnat ja protokollat toisistaan. TCP/IP-malli ei pyri tähän niin selvästi. [9, s. 6]. Seuraavaksi TCP/IP-viitemallin kerrokset lyhyesti selitettynä.

1. **Peruskerros**

   TCP/IP pinon alimman kerroksen, eli peruskerroksen pääasiallinen tehtävä on määritellä, kuinka tietokonejärjestelmä muodostaa yhteyden verkkoon. Peruskerros ei kuitenkaan ota kantaa lainkaan verkon tyyppiin, jota järjestelmä käyttää, mutta sanelee ehdot rajapinnan ajurille. Järjestelmän verkkorakenne voi kuitenkin olla lähes minkälainen tahansa. [8, s. 28]

2. **Verkkokerros**

   Verkkokerroksella määritellään virallinen pakettiformaatti, sekä otetaan käyttöön Internet-protokolla (eng. Internet Protocol, IP). [9, s. 5] IP-protokollan toiminta vastaa periaatteeltaan melko hyvin OSI-viitemallin verkkokerroksen toimintaa. IP-protokollaa käytetään IP-datagrammien lähettämiseen tietokoneiden välillä. Jokaisen IP-paketin otsikko sisältää kohdeosoitteen, joka on täydellinen reititystieto paketin kohteeseen. [12, s. 34] IP-protokollaa tarkastellaan lisää osiossa 3.3 IP-protokolla.

3. **Kuljetuskerros**

   Samoin kuten OSI-viitemallin tapauksessa kuljetuskerroksen tehtävä on asettaa kommunikaatiolle päittäinen päätepiste. Kerroksen toiminnasta on vastuussa kaksi protokollaa, joko TCP tai UDP. [12, s. 34]

   TCP (Transmission Control Protocol) on yhteyskeskeinen protokolla, joka mahdollistaa yhdestä laitteesta peräisin olevan tavuvirran toimittamisen ilman virheitä. TCP-protokollaa tarkastellaan lisää osiossa 3.4 TCP-protokolla. UDP (User Datagram Protocol) on yhteydetön protokolla sovelluksille, jotka eivät tarvitse TCP:n tarjoamaa datavirran hallintaa. Sen minkä UDP-protokolla häviää luotettavuudessa, se voittaa TCP-protokollan nopeudessa. [9, s. 6]

4. Sovelluskerros

Sovelluskerroksen protokollat vastaavat useita OSI-viitemallin kerroksia. OSI-viitemallin Istunto-, esitystapa- ja sovelluskerros pelkistetään yhdeksi TCP/IP-sovelluskerrokseksi. [12, s. 35] Sovelluskerros on rakennettu suoraan kuljetuskerroksen päälle, ja se sisältää useampia korkeamman tason protokollia käyttäjä rajapinnan toteuttamiseksi. Näitä ovat esimerkiksi FTP-, DNS - tai HTTP- protokollat. [9, s. 6]

Kuva 4. *OSI- ja TCP/IP-viitemallien kuvaajat vierekkäin. Kuten kuvasta huomataan TCP/IP-viitemallin kerrokset vastaavat yleensä useita OSI-viitemallin kerroksia. [11, s. 9]*

3.3 IP-protokolla

IP-protokolla on verkkokerroksella toimiva protokolla. Se sisältää tarvittavan reititys- ja ohjaustiedon pakettien kuljettamista varten. Protokolla on kaksi ensisijaista tehtävää tiedonsiirtoprosessissa. Sen tehtävä on kuljettaa IP-paketit internetin läpi kohteeseen parhaalla mahdollisella reitillä, sekä pakettien pilkkominen ja uudelleen kokoaminen erilaisia enimmäislähetyskokoja varten. [13, s. 666] IP-protokollasta on olemassa kaksi eri versiota. Tällä hetkellä vielä suuremmassa käytössä oleva IPv4, sekä uudempi ja kehittyneempi IPv6. Yksi IPv6:sen merkittävä lisäys oli sen osoitekentän koon suurentaminen 32 bitistä 128 bittiseksi. [14, s 1]

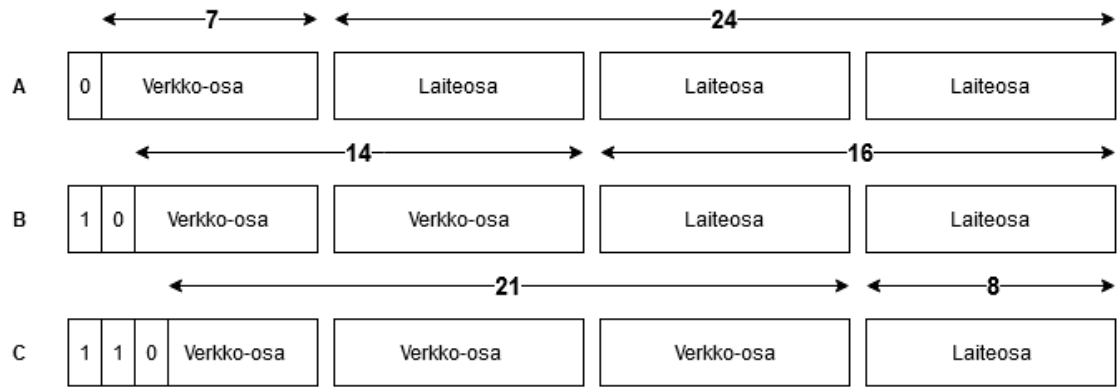
Jokaisella internettiin liitetyllä laitteella on oltava sen yksilöivä globaali IP-osoite. IP-osoite IPv4 tapauksessa on 32 bittinen numerosarja, joka jakautuu neljään

kahdeksan bitin arvoon. [15, s. 68-69] Esimerkiksi 192.168.140.179. Osoite jaetaan lisäksi vielä kahteen erilliseen osaan: verkon yksilöivään ja verkossa olevan laitteen yksilöivään osaan.

Erilaisia verkko-osa luokkia on viisi kappaletta: A, B, C, D ja E. Näistä kaksi viimeistä ovat vähemmän käytettyjä poikkeustapauksiin liittyviä luokkia. Kolmea ensimmäistä luokkaa erottaa verkon koon suuruus. IP-osoite alkaa tunnisteluvulla, joka kertoo mihin näistä luokista se kuuluu. [13, s. 668-669] Edellä mainittu osoite kuuluu C-luokkaan, joka edustaa verkon kooltaan pienintä luokkaa. Sen verkko-osa on 192.168.140, ja laiteosa muodostuu luvusta 179. Kyseinen osoite kuuluu työn käytännön osuudessa käytetylle toiselle taajuusmuuntajista.

***Taulukko 1**. Eri IP-osoiteluokat taulukoituna. Taulukon formaatti sarakkeessa N tarkoittaa verkko-osaa ja H laiteosaa. Mukailen lähdettä [13, s. 668]*

Luokka	Formaatti	Alkubitit	Osoitealue	Bittimäärät Verkko-osa/Laiteosa	Konemäärä
A	N.H.H.H	0	1.0.0.0-126.0.0.0	7/24	16777214
B	N.N.H.H	1,0	128.1-191.254.0.0	14/16	65543
C	N.H.H.H	1,1,0	192.0.1.0-223.255.254.0	22/8	245
D	-	1,1,1,0	224.0.0.0-239.255.255.255	-	-
E	-	1,1,1,1	240.0.0.0-254.255.255.255	-	-



***Kuva 5.** IP-luokkien A, B ja C rakenteiden kuvaajat. [13, s.669]*

IP-paketin reititys perustuu juuri siinä olevaan IP-osoitteeseen. Paketti etenee internetin läpi yhden hypyn kerrallaan reitittimeltä reitittimelle. Paketin reitti ei ole tiedossa matkan alussa, vaan sen sijaan jokaisessa pysähdyskohdassa reitti seuraavaa määränpää lasketaan kyseisen reitittimen reititystaulun arvoja vertailemalla paketissa olevaan kohdeosoitteeseen. [13, s. 675]

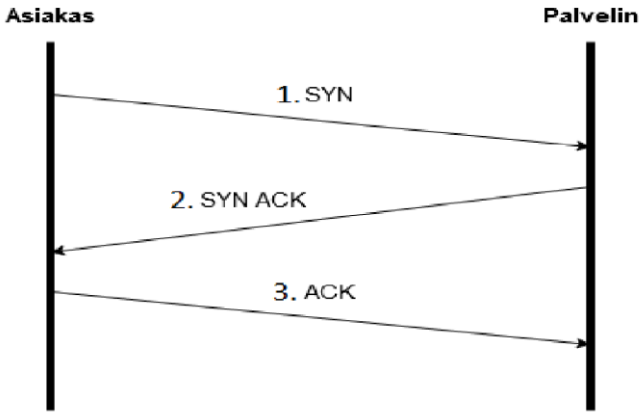
MTU (Maximum Transmission Unit) on arvo mikä kertoo suurimman datayksikön kokoluokan, mitä kukin verkkoyhteys voi kuljettaa. IP-protokolla on siis oltava ominaisuus, millä se voi hallita pakettikokoansa erilaisissa verkoissa. Ongelma ratkaistaan IP-fragmentaatiolla, jossa IP-paketti pilkotaan reitittimessä, joka toimii porttina pienemmän MTU-arvon verkkoon. Reititin pilkkoo paketit pienemmiksi osiksi, joita kutsutaan fragmenteiksi. Jokaisella fragmentilla on oma IP-otsikko, joka on alkuperäisen paketin otsikon kopio. Siten jokaisella fragmentilla on sama tunniste, lähde- ja kohdeosoite kuin alkuperäisellä paketilla. Katkelmien erottamiseksi ja uudelleen kokoamiseksi kukin fragmentti sisältää myös tiedon siitä mistä kohtaa fragmentti oli pilkottu alkuperäisestä paketista. [16, s. 1]

3.4 TCP-protokolla

TCP on kuljetuskerroksen yhteydellinen protokolla, joka käyttää IP-protokollaa kuljetinprotokollanaan. TCP-yhteys tarkoittaa sitä, että kommunikaatiota käyvät laitteet vastavuoroisesti luovat tunnustetun istunnon ennen kommunikaation alkamista. Yksi TCP-protokollan tarkoituksista on lisätä IP-kommunikaation luotettavuutta. Luotettavuuden takaamiseksi TCP-protokolla sisältää ominaisuuksia kuten tietovirran hallinnan (eng. flow control), virheiden havainnoinnin, sekä niiden korjaamisen. TCP- ja IP-protokollan välisen toiminnan voi kiteyttää niin, että TCP-protokolla huolehtii tiedonsiirtämisen sovellustasolle turvallisesti ja luotettavasti, kun taas IP-protokolla on vastuussa itse datan reitittämisestä sekä sen kuljettamisesta laitteelta laitteelle.

TCP-yhteyden muodostus on kolmiosainen kättelyprosessi. (eng. three-way handshake) [10, s. 142-143] Kättely noudattaa seuraavaa järjestystä:

1. Kättelyn aloittava kone lähettää kohde koneelle **SYN** viestin.
2. Kohde kone avaa pyyntöä koskevan yhteyden vastaamalla takaisin viestiin **SYN ACK** viestillä.
3. Keskustelun aloittanut kone vielä vastaa tähän vietiin viestillä **ACK**, jotta kohde kone havaitsee, että viesti on tullut perille.



Kuva 6. *TCP-protokollan kolmivaiheinen kättelyprosessi yhteyden muodostamiseksi. [9, s. 40]*

TCP-protokolla noudattaa asiakas-palvelin-viestintämallia (eng. Client-Server). Aloitteen tekevää laitetta kutsutaan TCP-asiakkaaksi. Vastaavasti laitetta johon yhteys muodostetaan, kutsutaan TCP-palvelimeksi. Mallin mukaisesti palvelin ei voi aloittaa kommunikaatiota, vaan se odottaa TCP-asiakkaan ottavan siihen yhteyttä tarjoten tälle palveluitansa.

TCP-protokollan tarkoitus on olla luotettava protokolla, mutta se ei kuitenkaan vaadi luotettavuutta alemmilta kerroksilta. Täten luotettavan tiedonsiirron vastuu jää ainoastaan TCP-protokollalle. Jos lähetetty data katoaa matkan aikana, se on lähetettävä uudelleen. Tämä tarkoittaa eksplisiittistä tiedonsiirtoa, jota yleensä käytetään, kun tieto ei ole aikakriittistä, mutta silti välttämätöntä. Lähetetyt paketit tarkistetaan tarkistussummalla. Pakettien vastaanottaja käyttää tarkistussumaa varmistaakseen, että tiedot on vastaanotettu oikein. [17, s. 29] Monet sovellukset tarvitsevat toimiakseen luotettavuutta, jota TCP-protokolla pystyy niille tarjoamaan.

4. MODBUS-PROTOKOLLA

Modbus-protokollan juuret sijaitsevat yhdysvaltalaisessa yrityksessä nimeltä Modicon. Protokolla kehitettiin vuonna 1979 alun perin tiedonsiirtoprotokollaksi yhtiön omille ohjelmoitaville logiikoille. Protokollasta on kuitenkin myöhemmin kehittynyt yksi teollisuuden käytetyimmistä tiedonsiirtoprotokollista, jota käytetään laajasti eri ohjaus- ja valvontajärjestelmissä. [17, s. 3]

Nykyisin Modbus on avoin sekä julkinen protokolla, jonka hallinta on siirtynyt Modbus-organisaatiolle. Organisaation jäseniä ovat automaatioalan lukuisat eri yritykset. Tämän vuoksi on selvää, että protokollalla on myös vahva asema tulevaisuudessa. Protokollasta on kehitetty kolme eri versiota: Modbus ASCII, Modbus RTU sekä Modbus TCP/IP. Näistä kaksi ensimmäistä käyttää sarjaliikennetekniikkaa, ja ne hyödyntävät fyysisellä tiedonsiirtokerroksella yleensä RS-232- tai RS-485- porttistandardeja. Viimeisin, eli TCP/IP-versio on IP-verkkopohjainen toteutus protokollasta, joka hyödyntää Ethernet-tekniikkaa. [18]

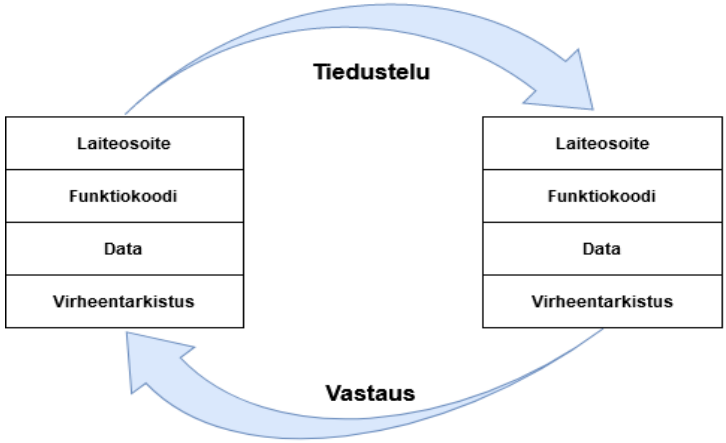
4.1 Modbus-protokollan yleinen toimintaperiaate

Modbus-protokolla itsessään on OSI-viitemallin seitsemännen, eli sovelluskerroksen protokolla, joka hyödyntää alemmilla kerroksilla eri protokollia riippuen käytettävästä Modbus-versiosta. Protokollan perusidea on varsin yksinkertainen. Isäntäkone (eng. master device) aloittaa kommunikaation käskemällä orjakonetta (eng. slave device), tai pyytämällä tältä arvoja viestipyyntönä, johon se vastaa pyydetyillä arvoilla. Huomattavaa on se, että isäntä-orja-mallin (eng. Master-Slave) mukaisesti vain isäntänä toimiva kone voi aloittaa koneiden välisen keskustelun. Isäntäkoneen lähettämää viestiä kutsutaan tiedusteluksi (eng. Query) ja orjakoneen viestiä vastaukseksi (eng. Response). Yhdessä nämä muodostavat tiedustelu-vastaus-syklin (eng. Query-Response Cycle). Isäntäkoneen on saatava vastaus edelliseen tiedusteluunsa, ja näin päätettyä sykli, ennen kuin se voi lähettää uuden tiedustelun. Isäntäkoneella on myös mahdollisuus lähettää laajempi yhteisviesti kaikille verkossa toimiville koneille (eng. Broadcast), johon sen ei tarvitse jäädä odottamaan vastauksia. [19, s. 4-5]

Modbus-viestirakenne koostuu neljästä erillisestä viestikentästä. Ensimmäinen kenttä eli laiteosoitekenttä sisältää viestin kohteena olevan koneen osoitteen. Kyseinen kenttä sisältää ainoastaan orjakoneiden osoitteita. Modbus-verkko rakentuukin niin, että vain orjakoneet tarvitsevat tämän osoitteen, sillä verkko sisältää yhden käskyttävän isäntäkoneen. Mikäli viesti on orjakoneelta tuleva vastaus, se liittää osoitteeksi oman osoitteensa, jonka avulla verkon isäntäkone pystyy tunnistamaan vastauksen lähettäneen orjakoneen. [19, s. 8-12]

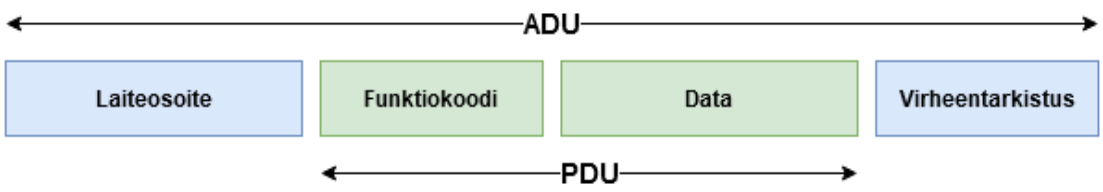
Seuraava kenttä eli funktiokoodikenttä sisältää yhden tavun verran tietoa suoritettavasta toiminnosta. Esimerkiksi funktiokoodi arvolla 01 antaa viestin kohteelle seuraavan toimintakäskyn ”Read Coil Status”. Käskyllä tarkoitetaan diskreetin *ON/OFF*-arvon lukemista laitteelta. Funktiokoodikenttää seuraa datakenttä, joka sisältää kaiken oleellisen lisätiedon funktiokoodissa olevan toiminnon suorittamiseksi. Virhetilanteiden ilmaantuessa datakenttä ilmoittaa poikkeuskoodin, jolla isäntäkone saa tiedon virheestä, ja pystyy näin reagoimaan siihen. [17, s 14-16]

Viimeinen kenttä on virheentarkistuskenttä. Virheentarkistusalgoritmeja on käytössä kaksi eri kappaletta riippuen käytettävästä Modbus-versiosta, LRC (Longitudinal Redundancy Check) on käytössä ASCII-versiossa ja CRC (Cyclic Redundancy Check) RTU-versiossa. Molempien tarkistusalgoritmien toiminta perustuu lähetyspäässä laskettavaan arvoon viestikentästä, jota vastaanottopäässä verrataan saapuvan viestinkentän oikeaan arvoon. [20, s. 14-19]



Kuva 7. Modbus-protokolalla kommunikoivien laitteiden muodostama tiedustelu-vastaus-sykli. [19, s. 5]

Sen lisäksi, että Modbus-viestirakenne koostu neljästä erillisestä kentästä, viestirakenne jaetaan myös kahteen erilliseen viestikehykseen. Funktiokoodi- ja datakentät muodostavat yhdessä *Protocol Data Unit*-kehyksen (PDU), jonka rakenne on yhteinen kaikille Modbus-versioille. PDU:sta laajennettu kehys *Application Data Unit* (ADU) ottaa mukaan laiteosoite- ja virheentarkistuskentät, ja näin ollen eri Modbus-versiot kantavat erilaisia ADU-kehyksiä. [22, s. 4]



Kuva 8. *Modbus-viestirakenne. [21, s. 4]*

4.2 Modbus-datamalli ja funktiot

Modbus-protokollan datamalli sisältää tietotyypin nimeltään rekisteri (eng. register). Yksi rekisteri sisältää 16 bitin verran tietoa. Rekisterit jaetaan kahteen eri kategoriaan. *Input Registers,* jotka ovat tyypiltään rekistereitä, joilla on vain lukuoikeus. *Holding Registers* ovat sen sijaan myös kirjoitettavissa sekä luettavissa olevia rekistereitä. Rekistereiden lisäksi Modbus-datamalliin kuuluu myös kaksi erilaista yksi bittistä tietotyyppiä *Discrete Inputs* ja *Coils*. Yhden bitin tietotyypit ovat niin ikään jaettu luku- ja kirjoitusoikeuksien mukaan. *Discrete Inputs*-tietotyyppi sisältää vain lukuoikeuden, kun taas *Coils*-tietotyyppi sisältää sekä luku- että kirjoitusoikeudet. [21, s. 6-8]

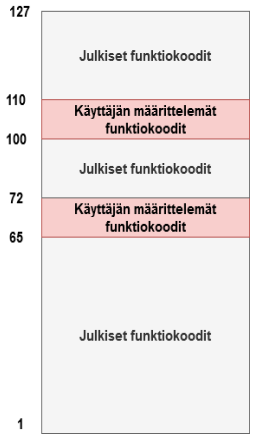
Taulukko 2. Modbus-tietotyypit taulukoituna. Mukailen lähdettä [21, s. 6]

Modbus tietotyyppi	**Rakenne**	**Luku-/kirjoitusoikeus**	**Kuvaus**
Discretes Input	1 bitti	Lukuoikeus	Data on peräisin I/O-järjestelmältä.
Coils	1 bitti	Luku- ja kirjoitusoikeudet	Dataa voidaan käsitellä sovellusohjelmalla.
Input Registers	16 bittinen sana	Lukuoikeus	Data on peräisin I/O-järjestelmältä.
Holding Registers	16 bittinen sana	Luku- ja kirjoitusoikeudet	Dataa voidaan käsitellä sovellusohjelmalla.

Modbus-viestirakenteessa oleva funktiokoodi kertoo Modbus-verkossa olevalle laitteelle mihin muistilohkoon ja toimintoon viestin funktio liittyy. Se määrittelee laitteelle toiminnon datatyypin, sekä onko kyseessä luku- vai kirjoitustoiminto. Modbus-funktiokoodit jaetaan kolmeen eri kategoriaan:

- **Julkiset funktiokoodit** (eng. Public Function Codes) ovat Modbus-organisaation valmiiksi määrittelemiä avoimesti sekä hyvin dokumentoituja uniikkeja funktiokoodeja.
- **Käyttäjän määrittelemät funktiokoodit** (eng. User-Defined Function Codes) ovat käyttäjän itse määrittelemiä funktiokoodeja, hänen omien tarpeidensa mukaan.
- **Varatut funktiokoodit** (eng. Reserved Function Codes) ovat yritysten luomia ja hallinnassa olevia funktiokoodeja, jotka eivät ole julkisessa käytössä.

Julkisille sekä käyttäjän määrittelemille funktioille on varattu yhteensä 127 muistipaikkaa. Kaksi muistilohkoa 65-72, sekä 100-110 ovat varattu käyttäjän määrittelemille funktioille. Jäljelle jäävät loput muistipaikat ovat varattu julkisille funktioille. [21, s. 10-11] Kuvassa 9 on esitetty funktiokoodeille varatut muistipaikat.



Kuva 9. *Kuvaus funktiokoodeille varatuista muistilohkopaikaista. [21, s. 11]*

4.3 Modbus TCP/IP

Modbus TCP/IP on tuorein versio Modbus-protokollasta. Versiossa Modbus RTU-version PDU-viestikehys on kapseloitu uudelleen käyttämään alemmilla tiedonsiirtokerroksilla TCP/IP-pinoon pohjautuvaa tiedonsiirtoa. Tämän vuoksi esimerkiksi isäntä-orja-nimityksen sijaan protokolla käyttää asiakas-palvelin-nimitystä (eng. client-server) käytettävästä tiedonsiirtomallista. Asiakaskoneen lähettämää viestiä kutsutaan pyynnöksi (eng. Request) ja palvelimen viestiä vastaukseksi (eng. Response). [22, s. 2-4]

Kuva 10. *Modbus asiakas-palvelin-mallin muodostama pyyntö/vastaus viestiyhteys pari. [22, s. 2]*

Merkittävä muutos aiempiin sarjaliikenne-pohjaisiin Modbus-versioihin nähden on, se että IP-verkossa on mahdollista toimia samanaikaisesti useita asiakaskoneita, eikä malli rajoitu Modbus RTU:n ja Modbus ASCII:n tapaan yhteen isäntäkoneeseen. Sen sijaan, että yksi isäntäkone ohjaisi rajattua määrää sen alaisuudessa toimivia orjia, voi IP-verkossa toimia periaatteessa mikä tahansa määrä asiakkaita käyttäen mitä tahansa määrää palvelimia. Kuitenkin liian suuri määrä verkossa toimivia asiakkaita voi aiheuttaa ristiriitaisia pyyntöjä palvelimille. Protokollaan on kehitetty erilaisia mekanismeja näiden ongelmien kitkemistä varten. Esimerkiksi palvelimelta välitettyjen viestien maksimi lukumäärää rajoitetaan konfiguraatio parametrilla *NumberMaxOfTransaction* [22, s. 20-29]

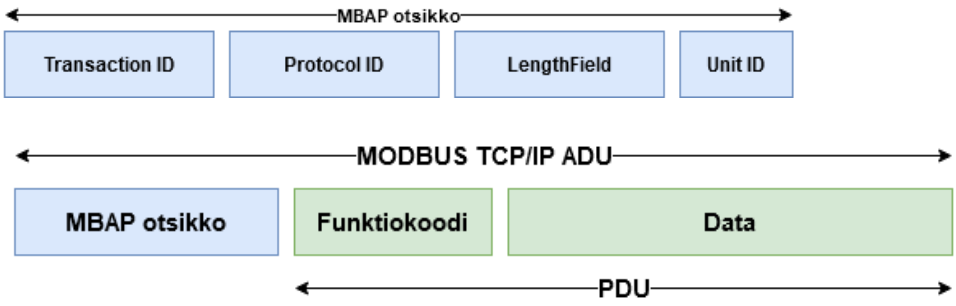
Tämän lisäksi Modbus TCP/IP-protokolalla on toinenkin merkittävä rajoite. Protokollan hyödyntämä Ethernet-tekniikka ei täytä reaaliaikaisuus vaatimuksia kaikkien sovellusten kohdalla riittävän hyvin, joka voi rajoittaa sen käyttöä. Protokollaan on kuitenkin olemassa tätä varten lisäys Real-Time Publisher Subscriber protocol. [23, s. 12]

Modbus TCP/IP kommunikaatio tarvitsee toimiakseen TCP-yhteyden. Toimivan yhteyden muodostamiseen on olemassa kaksi eri vaihtoehtoa: eksplisiittinen tai automaattinen yhteyden hallinta. Ensimmäisessä tapauksessa käyttäjäsovellukselle on tarjottava ohjelmointirajapinta, jotta yhteyttä voitaisiin hallita sen kautta kokonaisvaltaisesti. Toisessa tapauksessa yhteyden hallinta on piilotettu kokonaan pois käyttäjäsovelluksen näkyvistä, ja yhteyden hallinta on automaattista. Tämä vaihtoehto toimii myös oletusarvona, mikäli eksplisiittistä yhteyden hallintaa ei erikseen valita. Automaattinen yhteyden hallinta pyrkii pitämään laitteiden välisen TCP-yhteyden auki ilman, että sitä suljettaisiin, jokaisen viestipyynnön välissä. Yhteyden auki pitäminen toimii myös suosituksena, mikäli käytetään eksplisiittistä yhteyden hallintaa. [22, s. 9-11]

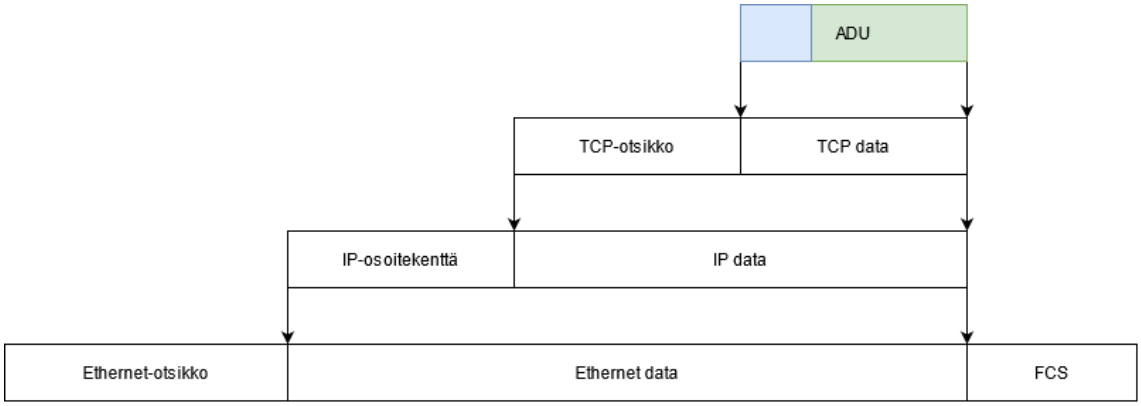
4.4 MBAP-otsikko

Modbus TCP/IP sisältää muuttuneen Application Data Unit (ADU) kehysrakenteen aiempiin Modbus-versioihin nähden. Funktiokoodin ja datakentän sisältävä PDU-rakenne on pysynyt ennallaan, mutta sitä edeltävä laiteosoitekenttä on korvattu MBAP-otsikkokentällä (eng. MBAP header), sekä PDU:n perässä oleva virheentarkistuskenttä on poistunut käytöstä kokonaan. MBAP-otsikko on yhteensä seitsemän tavun mittainen lisä viestiin ja se koostu neljästä erillisestä osasta:

- **Transaction Identifier** on kahden tavun pituinen osa, jota asiakaskone käyttää seuratakseen lähettämiään pyyntöjä palvelimelle. Palvelin vastaa asiakaskoneelle lähettäen identtisen Transaction Identifier-osan takaisin.
- **Protocol Identifier** on kahden tavun pituinen osa, jota käytetään protokollien tunnistamista varten järjestelmissä. Useat eri protokollat käyttävät kyseistä kenttää, ja täten Modbus-protokollan tunnisteeksi on varattu arvo nolla.
- **Length Field** on myös pituudeltaan kaksi tavua. Se kertoo jäljellä olevien kenttien pituudet mukaan lukien PDU:n kaikki kentät.
- **Unit Identifier** on pituudeltaan yhden tavun pituinen osa. Se tarjoaa osoitteen Modbus-protokollan sarjaliikenne-versioiden laitteille, mikäli niitä on liitetty verkkoon erottavan portin kautta. Muuten Modbus TCP/IP käyttää reitittämiseen alempana olevaa IP-protokollaa. [22, s. 4-6]



Kuva 11. *Modbus TCP/IP-viestirakenne, sekä MBAP-otsikon osat [22, s. 4]*



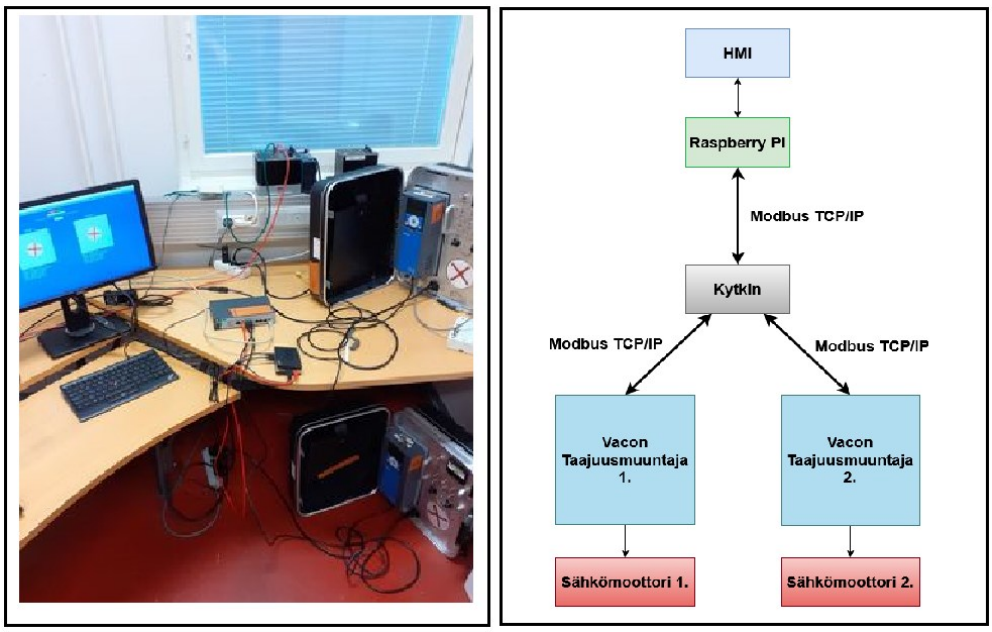
Kuva 12*. Modbus TCP/IP tiedonsiirron muodostama kokonainen viestirakenne. Kuvasta nähdään, että ylemmän kerroksen viestiä kuljetetaan aina yhtä kerrosta alempana olevan protokollan hyötykuormana. [22, s. 15]*

5. KÄYTÄNNÖN OSUUS

Tämä kandidaatintyö sisältää myös käytännön osuuden, jossa on toteutettu kahdelle erilliselle Vaconin taajuusmuuntajalle moottorinohjaussovellus Python-ohjelmointikielellä. Lähtökohtana sovelluksen ja muuntajien väliselle kommunikaatiolle oli käyttää Modbus TCP/IP-protokollaa. Tässä osiossa paneudutaan työn toteutukseen, sekä siinä käytettyyn laitteistoon ja tekniikoihin.

5.1 Laitteisto

Modbus TCP/IP-protokollan mukainen Asiakas-Palvelin lähiverkkoratkaisu toteutettiin kahdella palvelin- ja yhdellä asiakaslaitteella. Palvelinpuolta sovelluksessa edustavat kaksi Vaconin taajuusmuuntajaa, jotka muodostavat avoimet säätöpiirit ohjattavien moottoreiden kanssa. Verkon asiakaslaitteena toimii Raspberry Pi-niminen yhden piirilevyn pienoistietokone, johon sovellusohjelma on toteutettu Python-ohjelmointikielellä. Palvelimien sekä asiakaslaiteen muodostama verkko on toteutettu Ethernet-teollisuuskytkimen avulla.



Kuva 13. *Vasemmassa kuvassa laitteisto, ja oikealla puolella kaavio laitteiston muodostamasta lähiverkosta.*

Käytetyt laite- ja ohjelmistoversiot:

- *Raspberry Pi 4 Model B 1GB RAM, jossa Linux Desbian-pohjainen Raspberry Pi OS käyttöjärjestelmä.*
- *HP J9802A Ethernet-kytkin.*
- *2 kpl Vacon 100 HVAC taajuusmuuntajia.*
- *Python 2.7 versio.*

5.2 Vacon 100 taajuusmuuntajat

Työssä ohjattavat kaksi Vacon 100 HVAC taajuusmuuntajaa ovat osa Vacon 100 tuoteperhettä. Sarjan HVAC- muuntajat ovat suunniteltu silmällä pitäen erityisesti lämmitykseen, ilmanvaihtoon ja ilmastointiin käytettäviin sovelluksiin. [24]

Muita Vacon 100 tuoteperheen malleja ovat Vacon 100 Industrial, sekä Vacon 100 Flow. Industrial -sarjan taajuusmuuntajat ovat niin sanottuja teollisuuden yleismuuntajia, jotka soveltuvat laajalle skaalalle erinäköisiin teollisuussovelluksiin. Ne tukevat myös yleisesti käytössä olevaa IEC 61131-3 logiikkaohjelmointistandardia, jolla käyttäjä pystyy räätälöimään omia funktioita muuntajaansa. Flow-sarjan muuntajat ovat puolestaan suunniteltu erilaisia virtauksen säätö- ja pumppaussovelluksia silmällä pitäen. [25]

5.3 Raspberry Pi

Raspberry Pi-säätiö julkaisi ensimmäisen version Raspberry Pi:stä vuonna 2012. Ensimmäinen erä oli noin parin tuhannen kappaleen testierä, joka sai jatkoa suuren menestyksen ja kysynnän vuoksi. Raspberry Pi on yhden piirilevyn kokoinen tietokone, joka suunniteltiin alun alkaen lähinnä opetustarkoituksiin. Jälkeenpäin se on löytänyt tiensä myös muuhun käyttöön, jopa teollisuussovelluksiin asti. Yksi pääideoista alusta lähtien Raspberry Pi:n kohdalla on ollut se, että se kykenee siihen mihin tavallinen tietokonekin, mutta ei välttämättä yhtä nopeasti. Se sisältää kaikki tietokoneelle ominaiset peruselementit, kuten RAM-muistiyksikön, keskus- ja grafiikkaprosessorit. Tallennuskapasiteetti massamuistille hoidetaan muistikortin avulla, jolla on paikka Raspberry Pi:ssä. Tämän lisäksi Raspberry Pi sisältää lukuisia erilaisia portteja kommunikoinnille kuten Ethernet-, USB- ja HDMI-portit. [26, s. 8-18]

Kuva 14. *Yhden piirilevyn tietokone Raspberry Pi. Lähde [26, s. 10]*

5.4 Python ja PyModbus

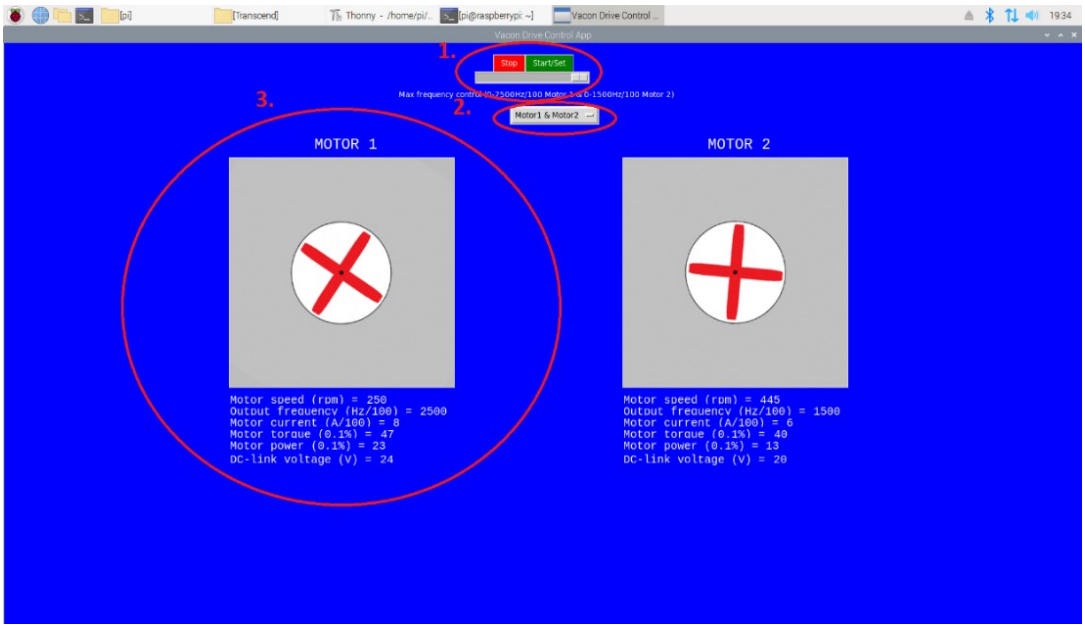
Python on tulkattava ohjelmointikieli. Se sisältää korkean tason tietorakenteita sekä yksinkertaisen, mutta tehokaan lähestymistavan olio-ohjelmointiin. Kieli mahdollistaa ohjelmien kirjoittamisen kompaktisti ja luettavasti. Sillä kirjoitetut ohjelmat ovat tyypillisesti lyhyempiä kuin vastaavat C-, C++- ja Java-ohjelmat. Tämä selittynee esimerkiksi sillä, että Pythonin korkean tason tietotyypeillä voi ilmaista monimutkaisia toimintoja yhdellä käskyllä, ja muuttujan tai argumentin määritystä ei tarvita. Myös sillä on merkitystä, että lauseiden ryhmittely tapahtuu sisennyksillä alku- ja loppusulkeiden sijaan. Python sisältää lukuisia erilaisia kirjastoja erilaisten projektien toteuttamista varten, kuten graafisten käyttöliittymien, tietokantojen hallintaan tai yksinkertaisten pelien kehittämistä varten. [27, s. 1-5]

Pymodbus on Modbus-sovelluksen mahdollistava Python-kirjasto, jonka avulla protokolla saadaan käyttöön Python-pohjaisille ohjelmille. Sitä voidaan käyttää suoraan Pythonissa, ilman että tarvitsisi lisätä kolmansia osapuolia ohjelmaan. Pymodbus toimii kaikilla Python-versioilla 2.7 versiosta eteenpäin. Kirjaston avulla asiakkaana toimiva Python-ohjelma pystyy lukemaan tai kirjoittamaan Modbus-palvelimelle rekistereitä sekä yksittäisiä arvoja. Se tukee kaikkia käytettäviä Modbus-versioita: ASCII, RTU- ja TCP/IP-versiota. [28, s. 1-2]

5.5 Käyttöliittymä kuvaus

Sovelluksen käyttöliittymä on rakennettu Tkinter-nimisen käyttöliittymäkirjaston avulla. Tarkoituksena on ollut, että operointi sillä olisi mahdollisimman selkeää ja yksinkertaista. Tärkeimpiä sovelluksenhallintakomponentteja ovat: *ajoasetusvalikko*, *liukusäädin sekä Start- ja Stop-painikkeet.* Tämän lisäksi käyttöliittymä sisältää kuvake- sekä tekstikenttiä moottoreilta luettavien arvojen seurantaan. Kuvakekenttä sisältää gif-animaation avulla toteutetun elementin, joka alkaa pyörimään, kun moottorilla on kierrosnopeutta.

Käyttäjällä on mahdollisuus asettaa moottorin maksimitaajuus liukusäätimellä, sekä valita ajoasetusvalikosta neljä erilaista ajoasetusta. Molempia moottoreita on mahdollista ajaa erikseen erillisillä asetuksilla. Sekä näiden lisäksi sovelluksessa on kaksi moottoreiden yhteisajoa, joista ensimmäisellä moottorit käynnistyvät yhtäaikaisesti, ja toisella moottorit vuorottelevat keskenään minuutin mittaisilla ajosykleillä.



Kuva 15. Näkymä sovelluksen käyttöliittymästä.

Yläpuolella olevaan käyttöliittymänäkymään on merkattu punaisella värillä tärkeimpiä käyttöliittymäkomponentteja. Alue 1: sisältää *Start*- ja *Stop*-painikkeet, sekä liukusäätimen. Alue 2: ajoasetusvalikon neljälle eri ajoasetukselle. Alue 3: animaatiokuvakkeen, joka havainnollistaa moottorin pyörimisliikettä, sekä alapuolella olevat tekstikentät, joihin päivittyvät seuraavat moottorilta luettavat arvot: moottorin nopeus (rpm), taajuus (Hz/100), ulostulo virta (A/100), vääntöprosentti, tehoprosentti ja tasavirtalinkin jännite (V).

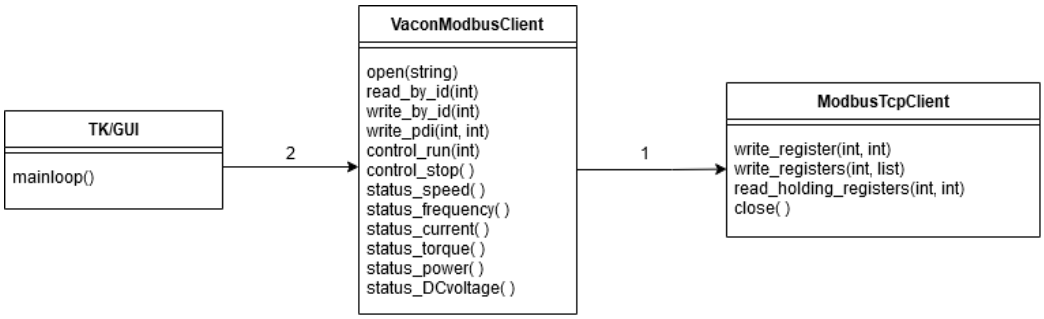
Seuraavaksi esimerkki käyttöliittymällä operoimisesta, jossa käyttäjä haluaa ajaa molempia moottoreita yhtäaikaisesti. Tällöin hän toimii seuraavasti:

(1) Käyttäjä valitsee ajoasetuksen *Motor1 & Motor2*.
(2) Käyttäjä asettaa haluamansa maksimitaajuuden liukusäätimellä.
(3) Käyttäjä painaa *Start/Set*-painiketta käynnistäen molemmat moottorit.
(4) Käyttäjä lopettaa ajon painamalla *Stop*-painiketta.

Myös muiden ajoasetusten kohdalla toimitaan esimerkin omaisesti. Turvallisuustekijöiden vuoksi sovellus pysähtyy aina, jos käyttäjä painaa Stop-painiketta, tai sulkee käyttöliittymän painamalla ikkunan yläreunan raksia.

5.6 Luokkakaavio

Luokkakaavio kuvaa sovellusohjelman rakennetta. Sovellusohjelma sisältää kolme erillistä luokkaa, joista yksi on Pymodbus-kirjaston luokka, joka toimii Modbus TCP/IP-asiakkaana. Toinen luokka on ohjausmetodeja varten rakennettu luokka, ja kolmas pitää sisällään käyttöliittymän toiminnot rakentavan luokan.



***Kuva 16**. Sovelluksen rakennetta kuvaava luokkakaavio.*

ModbusTcpClient

ModbusTcpClient-luokka on Pymodbus-kirjaston valmiiksi toteutettu kapseloitu luokka, jonka rajapintaa pystyy kirjaston avulla käyttämään. Luokalla toteutetaan Modbus TCP/IP-kommunikaatio asiakaslaitteen puolelta, ja se toimii täten Modbus-kommunikaation rajapintana. Luokan käyttäjä saa käyttöönsä sen tarjoamat palvelut ilman syvempää tietämystä siitä, miten luokka on toteutettu.

VaconModbusClient

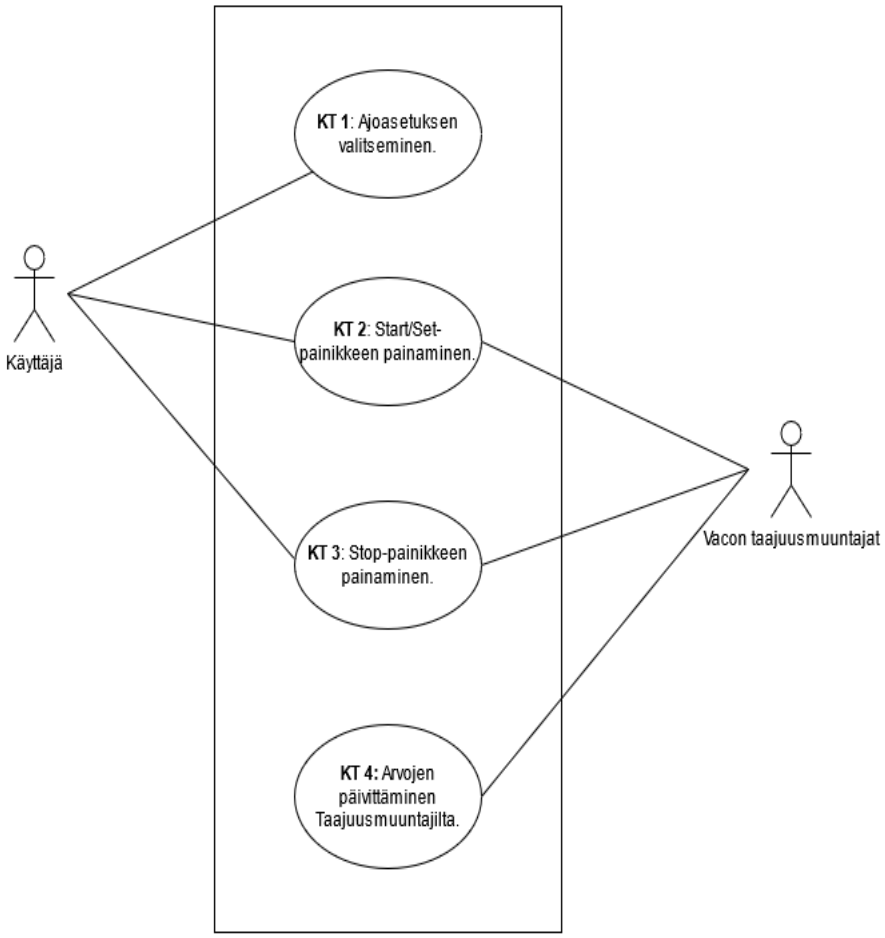
VaconModbusClient-luokka on eräänlainen välikappale käyttöliittymän ja ModbusTcpClient-luokan välissä. Luokassa käytetään ModbusTcpClient-luokan oliota. Luokka rakentaa olion käyttöön sovelluksen kannalta tarpeellisia ohjausmetodeja. Pääfunktiossa luokalle on luotu kaksi oliota, joista kumpikin antaa ohjausmetodinsa eri taajuusmuuntajan käyttöön.

TK/GUI

Sovelluksen graafinen käyttöliittymä on toteutettu Tkinter-käyttöliittymäkirjaston avulla. Kyseinen valmiiksi toteutettu kirjastoluokka sisältää erilaisia käyttöliittymämetodeja käyttöliittymän rakentamiseen. Esimerkiksi käyttöliittymän pääikkuna saadaan käyttöön luomalla luokan olio: *root = Tk(),* ja tämän jälkeen käyttämällä pääsilmukkametodia *root.mainloop().*

5.7 Käyttötapaukset

Käyttötapauskaaviossa kuvataan sovelluksen eri toimijoiden tekemiä toimenpiteitä sovellusohjelmalle. Eri toimijoiksi nähdään sovelluksen käyttäjä, sekä luettavia arvoja ohjelmalle päivittävät taajuusmuuntajat.



Kuva 17. *Sovelluksen käyttötapauskaavio.*

KT 1: Ajoasetusten valitseminen

Suorittaja: Käyttäjä.
Esiehdot: Raspberry Pi:llä, sekä taajuusmuuntajilla on toimiva Modbus TCP/IP-yhteys.
Kuvaus: Käyttäjä valitsee ajoasetusvalikosta haluamansa ajoasetuksen.
Lopputulos: Sovellusohjelma asettaa parametrikseen valitun ajoasetuksen.

KT 2: Start/Set painike. Ajon käynistys ja maksimitaajuuden asettaminen.

Suorittaja: Käyttäjä.
Esiehdot: Raspberry Pi:llä sekä taajuusmuuntajilla on toimiva Modbus TCP/IP-yhteys.
Kuvaus: Maksimitaajuuden asettamisen ja ajon käynnistys Start/Set-painikkeesta voidaan käytännössä katsoa olevan samaa käyttötapausta. Käyttäjä asettaa liukusäätimen avulla haluamansa maksimitaajuuden, ja tämän jälkeen painaa Start/Set-painiketta.
Lopputulos: Järjestelmän ajo käynnistyy, ja järjestelmä kiihdyttää asetettuun maksimitaajuuteen. Mikäli järjestelmä on jo valmiiksi ajossa, tällöin painallus asettaa uuden maksimitaajuuden, johon järjestelmä kiihdyttää.

KT 3: Stop painike. Ajon pysäytys.

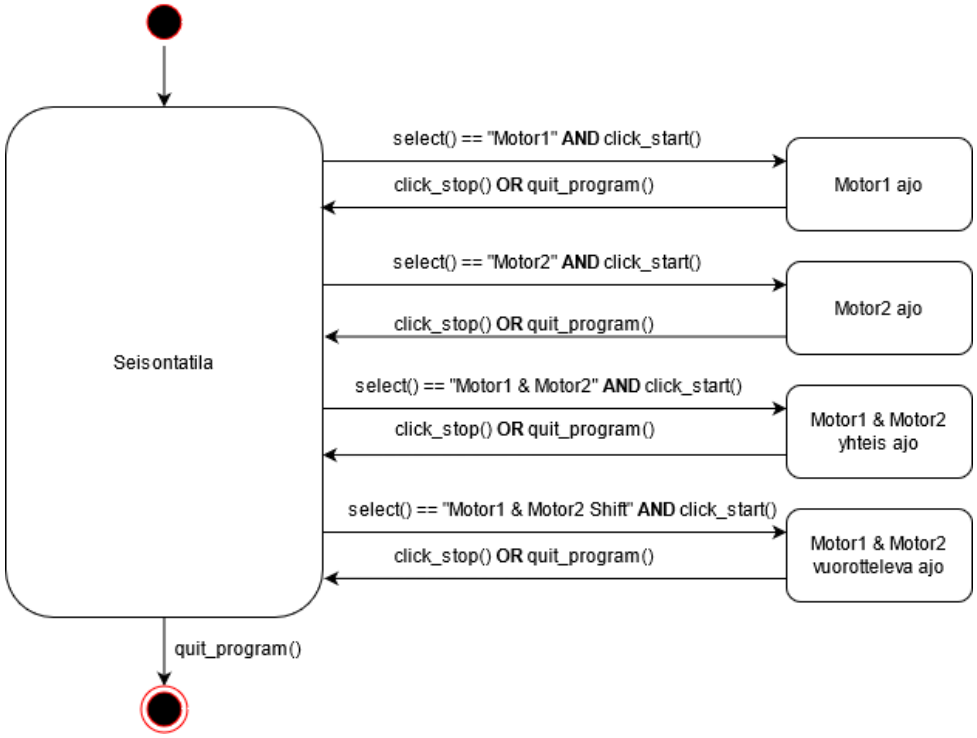
Suorittaja: Käyttäjä.
Esiehdot: Järjestelmän ajo on käynnissä.
Kuvaus: Käyttäjä painaa Stop-painiketta.
Lopputulos: Järjestelmän ajo pysähtyy.

KT 4: Luettavien arvojen päivittyminen taajuusmuuntajilta.

Suorittajat: Taajuusmuuntajat 1 ja 2.
Esiehdot: Raspberry Pi:llä sekä taajuusmuuntajilla on toimiva Modbus TCP/IP-yhteys.
Kuvaus: Taajuusmuuntajat päivittävät arvonsa sovellusohjelmalle 1000 millisekunnin välein.
Lopputulos: Uusi arvo päivittyy edellisen arvon tilalle.

5.8 Sovelluksen tilat

Sovelluksen eri tiloja ja tilasiirtymisiä kuvataan tilakaavion avulla. Kyseisestä sovelluksesta tiloja voidaan tunnistaa viisi eri kappaletta. Sovelluksen käynnistyessä päädytään alkutilasta seisontatilaan, jossa mikään neljästä eri ajoasetuksesta ei ole päällä. Eri ajoasetuksien voidaan katsoa olevan sovelluksen eri ajotiloja.

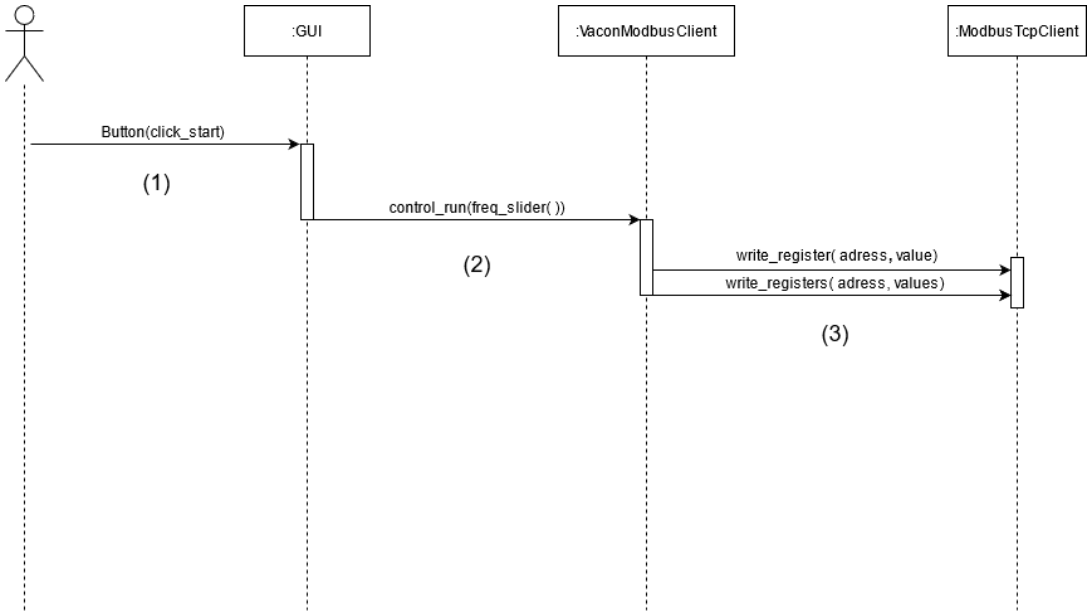


Kuva 18. Sovelluksen tilakaavio.

Seisontatilasta päästään johonkin neljästä ajotilasta valitsemalla ajoasetusvalikosta kyseinen asetus ja painamalla *Start/Set*-painiketta. Takaisin seisontatilaan ajotilasta päästään painamalla *Stop*-painiketta. Lopputilaan sovelluksesta päästä painamalla yläreunan raksia. Huomattavaa on kuitenkin, se että mikäli kesken ajotilan painetaan raksia, niin tällöin lopputilaan siirrytään vasta seisontatilan kautta. Tilakaavion tilasiirtymiin on merkattu käyttäjän toiminnoista seuraavat sovellusohjelmakoodin funktio- ja metodisuoritukset, sekä näistä muodostuvat loogiset ehdot, jotka vaihtavat sovelluksen tilaa.

5.9 Sekvenssikaaviot

Sekvenssikaaviot kuvaavat luokkien instanssien, eli olioiden välisiä vuorovaikutussuhteita. Osiossa esitetään kolmelle aiemmin esitetylle käyttötapaukselle sekvenssikaaviot. Kyseiset käyttötapaukset ovat: KT 2, KT 3 ja KT 4.



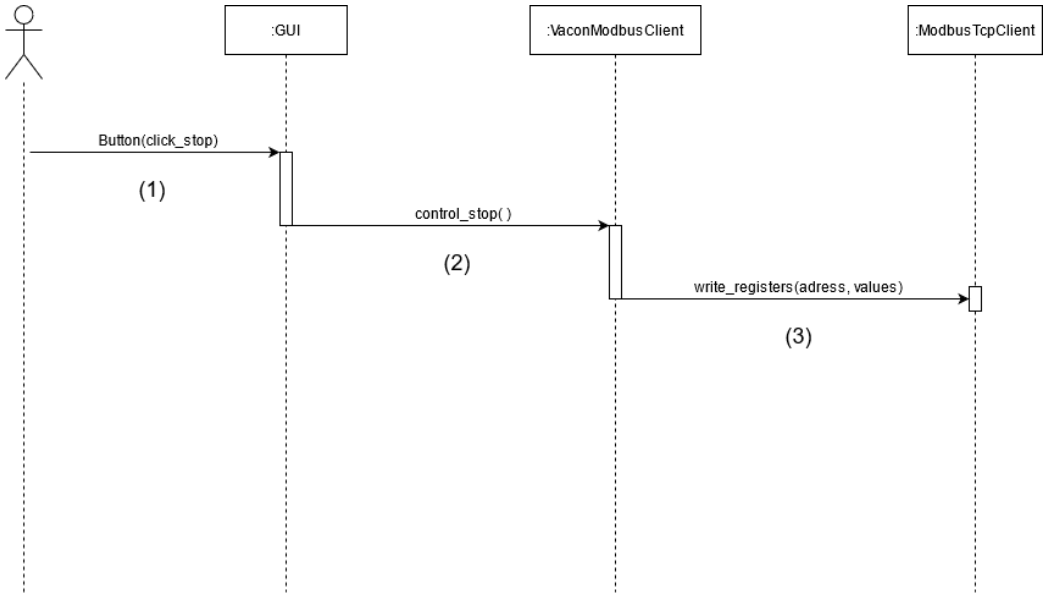
Kuva 19. Sekvenssikaavio 1. kuvaa KT 2:n välisiä vuorovaikutuksia.

Ajon käynnistyminen Start/Set-painike.

(1) Sekvenssi käynnistyy KT 2:n mukaisesti, kun käyttäjä painaa *Start/Set*-painiketta käyttöliittymästä. Painallus kutsuu GUI-kirjaston Button-metodia, jonka käskyksi on asetettu *click_start*-funktion suorittaminen.

(2) Kyseisessä funktiossa käytettään *VaconModbusClient*-luokan oliota, jonka *control_run*-metodiin on asetettu parametriksi liukusäädinkomponentin asettama maksimitaajuus arvo. Metodissa *control_run* käytetään kahta *VaconModbusClient*-luokan apumetodia luokan sisäisesti. Apumetodi *write_by_id* syöttää käytettävän maksimitaajuuden arvon eteenpäin sille annetun parametrin avulla. Toisella apumetodilla *write_pdi* syötetään eteenpäin sille parametrina annettu lista, joka sisältää taajuusmuuntajan kiihdytysarvoja.

(3) Molemmissa apumetodeissa *write_by_id* ja *write_pdi* käytetään *ModbusTcpClient*-luokan olioita. Maksimitaajuuden arvo syötetään Modbus-palvelimelle, eli taajuusmuuntajalle *ModbusTcpClient*-luokan *write_register*-metodilla, ja kiihdytys arvojen syöttämiseen palvelimelle käytetään *write_registers*-metodia. Molemmissa metodeissa parametreina ovat osoite sekä kirjoitettava arvo/arvot.



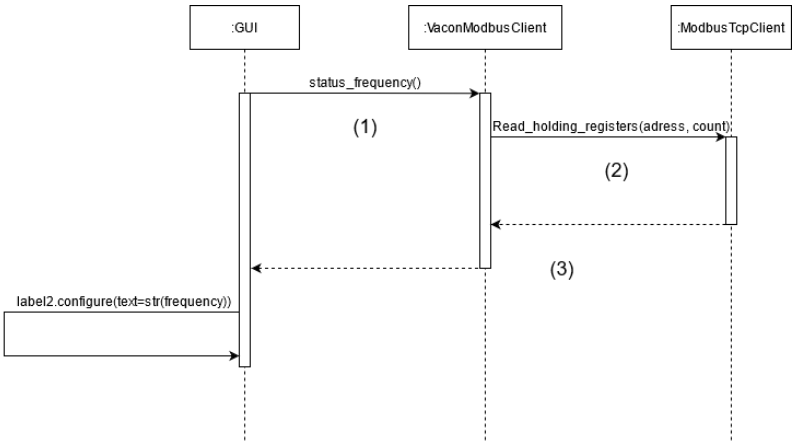
Kuva 20. Sekvenssikaavio 2. kuvaa KT 3:n välisiä vuorovaikutuksia.

Ajon pysäytys Stop-painike.

(1) Sekvenssi käynnistyy KT 3:n mukaisesti, kun käyttäjä painaa *Stop*-painiketta. Painike kutsuu *GUI-kirjaston Button*-metodia, jonka käskyksi on asetettu *click_stop*-funktion suorittaminen.

(2) Funktio käyttää *VaconModbusClient*-luokan oliota, jonka metodi *control_stop* käyttää saman luokan apumetodia write_pdi luokan sisäisesti. Apumetodille on annettu parametriksi lista, joka sisältää taajuusmuuntajalle kirjoitettavia pysäytysarvoja.

(3) Apumetodissa write_pdi käytetään *ModbusTcpClient*-luokan oliota. Tämä olio puolestaan käyttää *write_registers-metodia*, joka syöttää Modbus-palvelimelle, eli taajuusmuuntajalle jarrutusarvoja käyttäen apumetodille parametrina annettua listaa, sekä toisena parametrina annettua osoitetta.



Kuva 21. Sekvenssikaavio 3. kuvaa KT 4:n välisiä vuorovaikutuksia yhden luettavan arvon osalta.

Luettavien arvojen päivittyminen.

Kaikki taajuusmuuntajilta luettavat kuusi arvoa noudattavat samanlaista sekvenssirakennetta, joten kyseiseen sekvenssikaavion esimerkkiin on valittu luettavista arvoista taajuus. Eroavaisuus eri luettavien arvojen välillä ohjelman puolella on lähinnä millä tunnistearvolla niitä luetaan taajuusmuuntajalta. Esimerkiksi taajuuden arvo luetaan tunnistearvolla 2104, kun taas kierrosnopeus arvolla 2105.

(1) Sekvenssi käynnistyy käyttöliittymän *read1*-funktiossa, joka kutsuu itseään takaisin aina 1000 millisekunnin syklein. Funktion sisällä käytetään *VaconModbusClient*-luokan oliota, joka kutsuu *status_frequency*-metodia jokaisen syklin aikana.

(2) Metodissa käytettään saman luokan apumetodia *read_by_id, jolle on parametrina annettu tunnistearvo (2104).* Apumetodi käyttää ModbusTcpClient-luokan oliota, jonka metodilla *read_holding_registers* luetaan rekisteriarvoja Modbus-palvelimelta, eli taajuusmuuntajalta. Parametri *adress*, josta metodi *read_holding_registers* lukee rekisteriarvoja noudattaa seuraavaa kaava: ((tunnistearvo – 1) * 2) + 20000. Parametri *count* tarkoittaa luettavien rekistereiden määrän.

(3) Metodi *read_holding_registers* palauttaa *VaconModbusClient*-luokan oliolle luettavan rekisterin arvon, josta se vuorostaan palautetaan *GUI*-luokan *root*-olion käyttöön. Funktion *read1* sisällä käytetään *GUI*-luokan *configure*-metodia, joka päivittää uuden arvon käyttöliittymään yhden kerran jokaisen syklin aikana.

Taulukko 3. Taajuusmuuntajalta luettavien arvojen id-tunnisteet.

Luettava arvo	Tunnistearvo
Ulostulotaajuus (Hz)	2104
Moottorin kierrosnopeus (rpm)	2105
Virta (A)	2106
Vääntö (%)	2107
Teho (%)	2108
DC-jännite (V)	2110

6. YHTEENVETO

Työn käytännön osuuden tavoitteena oli toteuttaa Raspberry Pi:lle sovellus, jolla pystyisi ohjaamaan Vacon 100 taajuusmuuntajia Modbus TCP/IP-protokollaa hyödyntäen. Tässä onnistuttiin, ja Modbus TCP/IP-yhteys saatiin toimimaan molempiin suuntiin. Sovelluksella on sekä mahdollista lukea, että kirjoittaa arvoja taajuusmuuntajille. Sovelluksen toteutus ei kuitenkaan onnistunut täysin ongelmitta, ja aluksi arvojen kirjoitustoiminto muuntajalle ei meinannut onnistua. Kirjoitustoiminto saatiin kuitenkin toimimaan etäpalaverin kautta työnohjaajan kanssa.

Toteutetun sovellusohjelman toiminnallisuuden voi jakaa kahteen osaan: Sovelluksen käyttöliittymäosaan, sekä taustalla vaikuttavaan osaan, joka vastaa Modbus TCP/IP-kommunikaatiosta sovelluksen asiakaslaiteen päässä. Työn päätarkoituksena oli perehtyä Modbus TCP/IP-protokollaan, sekä selvittää Raspberry Pi:n käyttömahdollisuutta toimia protokollaa hyödyntävänä asiakaslaitteena. Sovelluksessa Modbus TCP/IP-kommunikaatio toimii asiakkaalta palvelimelle sille halutulla tavalla, ja tästä voidaan päätellä, että Raspberry Pi:n hyödyntäminen ainakin pienen mittakaavan Modbus TCP/IP-sovelluksissa on täysin mahdollista käyttäen apuna Pythonin Pymodbus-kirjastoa. Sovellusohjelman käyttöliittymäpuolella esiintyy kuitenkin pieniä ongelmia. Esimerkiksi käyttöliittymässä moottorin pyörimistä kuvaavat GIF-animaatiokuvakkeet toimivat välillä järjestelmää hidastaen ja sitä jumittaen. Käyttöliittymäohjelmoinnin ei kuitenkaan katsottu olevan työn päätarkoitus, ja tähän ongelmaan ei jääty jumiin.

Mielestäni työn toteuttamisen mielenkiintoisin vaihe oli itse käytännön osuudessa oleva sovellusohjelman toteutus. Tämän vaiheen voidaan katsoa myös eniten aikaa vieneeksi osuudeksi. Työn tekemistä helpotti kuitenkin se, että Pymodbus-kirjaston dokumentointi oli riittävän selkeää, ja internetistä löytyi kattavasti esimerkkejä aiheesta.

Lopuksi vielä kiitokset työnohjaajalle Mikko Salmenperälle, sekä Jari Seppälälle, joka luovutti toisen Vaconin taajuusmuuntajista kotikäyttööni COVID-19-pandemiasta seuranneen yliopiston tilojen sulkeutumisen ajaksi.

LÄHTEET

[1] Groover. M. P, Automation, Production Systems, and Computer-Integrated Manufacturing, 4rd ed, Pearson, 2015, 815 p.

[2] Bolton. W, Control Systems, Newnes Oxford, 2002, 180 p.

[3] Marcin Bajer, DATAFLOW IN MODERN INDUSTRIAL AUTOMATION SYSTEMS. THEORY AND PRACTICE, ABB Corporate Research Kraków, 11 p. Saatavissa (viitattu 6.7.2020): https://www.researchgate.net/publication/320187633

[4] M. F. Körner, D. Bauer, R. Keller, M. Rösch, A. Schlereth, P. Simon, T. Bauernhansl, G. Fridgen, G. Reinhart, Extending the Automation Pyramid for Industrial Demand Response, Fim-rc, 7 p. Saatavissa (viitattu 7.7.2020): https://orbilu.uni.lu/bitstream/10993/

[5] Richard Zurawaski, Industrial Communication Technology Handbook, 2nd edition, Taylor & Francis Group, 2015, 1757 p.

[6] The Extension A technical Supplement to control NETWORK, Introduction to Ethernet. ccontrols, 1999, 6 p. Saatavissa (viitattu 20.7.2020): https://www.ccontrols.com/pdf/ExtV1N3.pdf

[7] Bogdan M. Wilamowski, J. David Irwin, The Industrial Electronics Handbook 2nd edition INDUSTRIAL COMMUNICATION SYSTEMS, CRC Press, 2016, 962 p.

[8] Blank Andrew G, TCP/IP foundations, Neil Edde, 2004, 284 p.

[9] Srinivas Jonnalagadda, Ph.D., Introduction to TCP/IP Protocol Suite, Siri Technologies, 2003, 53 p.

[10] Mike Harwood, CompTIA Network+ N10-004 Exam Cram, 3rd Edition. Pearson, 2009, 628 p.

[11] Paul Simoneau, The OSI Model: Understanding the Seven Layers of Computer Networks, Global Knowledge Training LLC, 2006, 11 p. Saatavissa (viitattu 1.8.2020): https://faculty.sfcc.spokane.edu/Rudlock/files/WP_Simoneau_OSIModel.pdf

[12] Libor Dostálek, Alena Kabelová, Understanding TCP/IP: A Clear and Comprehensive Guide, Packt Publishing, 2006, 598 p.

[13] Internetworking Technology Handbook Fourth edition, Cisco Systems Press, 2003, 1128 p.

[14] Learn About Differences in Addressing Between IPv4 and IPv6, Juniper Networks, Inc, 2014, 11 p.

[15] L. Parziale D. T. Britt, C. Davis J. Forrester, W. Liu, C. Matthews, N. Rosselot, TCP/IP Tutorial and Technical Overview, IBM Redbooks, 2006, 974 p.

[16] Colleen Shannon, David Moore, K Claffy, Characteristics of Fragmented IP Traffic on Internet Links, Reseachgate, 2001, 15 p. Saatavissa (viitattu 3.8.2020): https://www.researchgate.net/publication/2412218

[17] INTRODUCTION TO MODBUS TCP/IP, Acromag, 2005, 42 p. Saatavissa (viitattu 28.8.2020): https://www.academia.edu/21786114/

[18] Dieter Wimbeger, Understanding the Modbus Protocol, 7 p. Saatavissa (viitattu 10.9.2020): http://docshare02.docshare.tips/files/12123/121231097.pdf

[19] Modicon Modbus Protocol Reference Guide, Modiconc Inc, 1996, 115 p. Saatavissa (viitattu 11.9.2020): https://modbus.org/docs/PI_MBUS_300.pdf

[20] MODBUS over Serial Line Specification & Implementation guide, Modbus Organization, 2002, 44 p. Saatavissa (viitattu 17.9.2020): https://modbus.org/docs/Modbus_over_serial_line_V1.pdf

[21] MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1b, Modbus Organization, 2006, 51 p. Saatavissa (viitattu 21.9.2020): https://modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf

[22] MODBUS MESSAGING ON TCP/IP IMPLEMENTATION GUIDE V1.0b, Modbus Organization, 2006, 46 p. Saatavissa (viitattu 26.9.2020): https://modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf

[23] Dr. Wolfgang Kastner, Advantages of Industrial Ethernet-Comparison of Modbus over TCP/IP and PROFINET, 2011, 27 p. Saatavissa (viitattu 28.3.2021): https://www.auto.tuwien.ac.at/bib/pdf_TR/TR0153.pdf

[24] VACON®100 HVAC FOR INDOOR CLIMATE CONTROL, VACON Inc, 8 p. Saatavissa (viitattu 28.3.2021): https://www.lsk.fi/globalassets/tuotteet/taajuusmuuttajat/ip54/vacon/100hvac.pdf

[25] VACON 100 selection guide, VACON® 100 – versatile AC drives designed to save energy and improve process control, Danfoss Drives, 2016, 20 p. Saatavissa (viitattu 15.12.2020): https://files.danfoss.com/download/Drives/DKDDPB906A602_VACON_100_LR.pdf

[26] Gareth Halfacree, THE OFFICIAL Raspberry Pi Beginner's Guide How to use your new computer Python Tutorial *Release 3.7.0*, Raspberry Pi Trading Ltd, 2018, 241 p.

[27] Guido van Rossum and the Python development team, Python Tutorial *Release 3.7.0, Python Software Foundation, 2018, 149 p.* Saatavissa (viitattu 4.1.2021): https://bugs.python.org/file47781/Tutorial_EDIT.pdf

[28] Sanjay, PyModbus Documentation Release 2.4.0, 2020, 252 p. Saatavissa (viitattu 4.1.2021): https://buildmedia.readthedocs.org/media/pdf/pymodbus/latest/pymodbus

LIITE

Liite sisältää sovellusohjelman Python-lähdekoodin.

```
"""
Modbus interface to Vacon Drive.
"""

import pymodbus
from tkinter import *
from PIL import Image, ImageTk, ImageSequence
from pymodbus.client.sync import ModbusTcpClient as ModbusClient


class VaconModbusClient:
    def __init__(self, client):
        self.client_module = client
        self.client = None
        self.id_cache = {}


    def open(self, ip, port=502):
        self.client = self.client_module(ip, port)
        return True


    def read_by_id(self, id, extended=False):

        if extended:
            ret = self.client.read_holding_registers(((id - 1) * 2) + 20000, 2)
            if isinstance(ret, pymodbus.register_read_message.ReadHoldingRegistersResponse):
                hi, lo = ret.registers
                value = (True, hi * 65536 + lo)
            else:
                value = (False, 0)
        else:
            ret = self.client.read_holding_registers(id - 1, 1)
            if isinstance(ret, pymodbus.register_read_message.ReadHoldingRegistersResponse):
                value = (True, ret.registers[0])
                self.id_cache[id] = value
            else:
                value = (False, 0)
        return value


    def write_by_id(self, id, value, extended=False):
        ret = self.client.write_register(id - 1, value)
        if isinstance(ret, pymodbus.register_write_message.WriteSingleRegisterResponse):
            value = True
            self.id_cache[id] = value
        else:
            value = False
        return value


    def write_pdi(self, values, extended=False):

        if extended:
            reg_16 = []
            if len(values) > 11:
                values = values[:11]
            for value in values:
                reg_16.append(value / 65536)
                reg_16.append(value % 65536)

            ret = self.client.write_registers(2050, reg_16[:10])
            if isinstance(ret, pymodbus.register_write_message.WriteMultipleRegistersResponse):
```

```
                ret = self.client.write_registers(2060, reg_16[10:])
                if isinstance(ret, pymodbus.register_write_message.WriteMultipleRegistersRe
                              sponse):
                    value = (True, tuple(values))
                else:
                    value = (False, tuple())
            else:
                value = (False, tuple())

        else:
            if len(values) > 11:
                values = values[:11]
            ret = self.client.write_registers(2000, values)
            if isinstance(ret, pymodbus.register_write_message.WriteMultipleRegistersResponse):
                value = (True, tuple(values))
            else:
                value = (False, tuple())

        return value


    def control_run(self, value):

        self.write_by_id(102, value)
        while True:
            max_percentage = 100
            for i in range(0, max_percentage, 3):
                self.write_pdi([1, 0, i * 100, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000])

            break


    def control_stop(self):

        self.write_pdi([0, 0, 0, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000])


    def status_speed(self):

        ret_s, SpeedOut = self.read_by_id(2105)
        return SpeedOut


    def status_frequency(self):

        ret, FreqOut = self.read_by_id(2104)
        return FreqOut


    def status_current(self):

        ret_c, CurrentOut = self.read_by_id(2106)
        return CurrentOut


    def status_torque(self):

        ret_t, TorqueOut = self.read_by_id(2107)
        return TorqueOut


    def status_power(self):

        ret_p, PowerOut = self.read_by_id(2108)
        return PowerOut


    def status_DCvoltage(self):

        ret_dc, DCOut = self.read_by_id(2110)
        return DCOut


if __name__ == '__main__':
```

```
drive1 = VaconModbusClient(ModbusClient)
ip1 = '192.168.140.179'
drive1.open(ip1, port=502)

drive2 = VaconModbusClient(ModbusClient)
ip2 = '192.168.140.178'
drive2.open(ip2, port=502)

root = Tk()
root.geometry("2100x1100")
root.title("Vacon Drive Control App")
root.configure(background="blue")


options = ["Motor1", "Motor2", "Motor1 & Motor2", "Motor1 & Motor2 Shift"]
clicked = StringVar()
clicked.set(options[0])

var = 0
cond = True


def gif():
    canvas = Canvas(root, width=400, height=400)
    canvas.place(x=400,y=200)
    sequence = [ImageTk.PhotoImage(img)
                        for img in ImageSequence.Iterator(
                                Image.open(
                                "/home/pi/SpinningGif1.gif"))]
    image = canvas.create_image(200,200, image=sequence[1])

    canvas2 = Canvas(root, width=400, height=400)
    canvas2.place(x=1100,y=200)
    sequence2 = [ImageTk.PhotoImage(img)
                        for img in ImageSequence.Iterator(
                                Image.open(
                                "/home/pi/SpinningGif1.gif"))]
    image2 = canvas2.create_image(200,200, image=sequence2[1])

    animate(canvas, image, sequence, 1)
    animate2(canvas2, image2, sequence2, 1)


def animate(canvas, image, sequence, counter):
    canvas.itemconfig(image, image=sequence[counter])
    if drive1.status_speed() > 0 and select() == "Motor1":
        root.after(3000//drive1.status_speed(),
        lambda: animate(canvas, image, sequence, (counter+1) % len(sequence)))
    if drive1.status_speed() == 0 or select() == "Motor2":
        root.after(100, lambda: animate(canvas, image, sequence, (counter) % len(sequence)))
    if drive1.status_speed() > 0 and select() == "Motor1 & Motor2":
        root.after(4000//drive1.status_speed(),
        lambda: animate(canvas, image, sequence, (counter+1) % len(sequence)))
    if drive1.status_speed() > 0 and select() == "Motor1 & Motor2 Shift":
        root.after(3000//drive1.status_speed(),
        lambda: animate(canvas, image, sequence, (counter+1) % len(sequence)))

def animate2(canvas2, image2, sequence2, counter2):
    canvas2.itemconfig(image2, image=sequence2[counter2])
    if drive2.status_speed() > 0 and select() == "Motor2":
        root.after(7000//drive2.status_speed(),
        lambda: animate2(canvas2, image2, sequence2, (counter2+1) % len(sequence2)))
    if drive2.status_speed() == 0 or select() == "Motor1":
        root.after(100, lambda: animate2(canvas2, image2, sequence2, (counter2) % len(se
                    quence2)))
    if drive2.status_speed() > 0 and select() == "Motor1 & Motor2":
        root.after(12000//drive2.status_speed(),
        lambda: animate2(canvas2, image2, sequence2, (counter2+1) % len(sequence2)))
    if drive2.status_speed() > 0 and select() == "Motor1 & Motor2 Shift":
        root.after(12000//drive2.status_speed(),
        lambda: animate2(canvas2, image2, sequence2, (counter2+1) % len(sequence2)))
def read1():
```

```
    speed = drive1.status_speed()
    frequency = drive1.status_frequency()
    current = drive1.status_current()
    DCvoltage = drive1.status_DCvoltage()
    torque = drive1.status_torque()
    power = drive1.status_power()
    label1.configure(text=str(speed))
    label2.configure(text=str(frequency))
    label3.configure(text=str(current))
    label4.configure(text=str(DCvoltage))
    label5.configure(text=str(torque))
    label6.configure(text=str(power))
    root.after(1000, read1)

def read2():
    speed2 = drive2.status_speed()
    frequency2 = drive2.status_frequency()
    current2 = drive2.status_current()
    DCvoltage2 = drive2.status_DCvoltage()
    torque2 = drive2.status_torque()
    power2 = drive2.status_power()
    label1_2.configure(text=str(speed2))
    label2_2.configure(text=str(frequency2))
    label3_2.configure(text=str(current2))
    label4_2.configure(text=str(DCvoltage2))
    label5_2.configure(text=str(torque2))
    label6_2.configure(text=str(power2))
    root.after(1000, read2)

def click_start():
    if select() == "Motor1":
        drive1.control_run(freq_slider())
        drive2.control_stop()
        read1()
    elif select() == "Motor2":
        drive2.control_run(freq_slider())
        drive1.control_stop()
        read2()
    elif select() == "Motor1 & Motor2":
        drive1.control_run(freq_slider())
        drive2.control_run(freq_slider())
        read1()
        read2()
    elif select() == "Motor1 & Motor2 Shift":
        read1()
        read2()
        shift_run()

def shift_run():
    global var
    global cond

    if var == 0:
        drive1.control_run(freq_slider())

    if drive1.status_frequency() > (freq_slider() - 27) or var > 0 and var < 26:
        drive1.control_stop()
        drive2.control_run(freq_slider())
        var = var + 1

    if var == 26:
        drive2.control_stop()
        var = 0

    if cond == False or select() != "Motor1 & Motor2 Shift":
        drive1.control_stop()
        drive2.control_stop()
        cond = True
        var = 0
        stop()

    if var < 26:
        root.after(2000, shift_run)
def click_stop():
```

```
    drive1.control_stop()
    drive2.control_stop()
    global cond
    cond = False

def quit_program():
    drive1.control_stop()
    drive2.control_stop()
    root.destroy()

def freq_slider():
    value = slider.get()
    return value

def select():
    value = clicked.get()
    if value == "Motor1" or value == "Motor1 & Motor2 Shift" and var == 0:
        drive2.control_stop()
    if value == "Motor2" or value == "Motor1 & Motor2 Shift" and  var != 0:
        drive1.control_stop()
    return value

label1 = Label(root, text="", bg="blue", fg="white")
label_s = Label(root, text="Motor speed (rpm) = ", bg="blue", fg="white")
label1.place(x=643,y=610)
label_s.place(x=400,y=610)
label1.config(font=("Courier", 15))
label_s.config(font=("Courier", 15))

label2 = Label(root, text="", bg="blue", fg="white")
label_f = Label(root, text="Output frequency (Hz/100) = ", bg="blue", fg="white")
label2.place(x=738,y=630)
label_f.place(x=400,y=630)
label2.config(font=("Courier", 15))
label_f.config(font=("Courier", 15))

label3 = Label(root, text="", bg="blue", fg="white")
label_c = Label(root, text="Motor current (A/100) = ", bg="blue", fg="white")
label3.place(x=690,y=650)
label_c.place(x=400,y=650)
label3.config(font=("Courier", 15))
label_c.config(font=("Courier", 15))

label4 = Label(root, text="", bg="blue", fg="white")
label_DC = Label(root, text="DC-link voltage (V) = ", bg="blue", fg="white")
label4.place(x=655,y=714)
label_DC.place(x=400,y=714)
label4.config(font=("Courier", 15))
label_DC.config(font=("Courier", 15))

label5 = Label(root, text="", bg="blue", fg="white")
label_t = Label(root, text="Motor torque (0.1%) = ", bg="blue", fg="white")
label5.place(x=668,y=670)
label_t.place(x=400,y=670)
label5.config(font=("Courier", 15))
label_t.config(font=("Courier", 15))

label6 = Label(root, text="", bg="blue", fg="white")
label_p = Label(root, text="Motor power (0.1%) = ", bg="blue", fg="white")
label6.place(x=653,y=690)
label_p.place(x=400,y=690)
label6.config(font=("Courier", 15))
label_p.config(font=("Courier", 15))

label1_2 = Label(root, text="", bg="blue", fg="white")
label_s_2 = Label(root, text="Motor speed (rpm) = ", bg="blue", fg="white")
label1_2.place(x=1343,y=610)
label_s_2.place(x=1100,y=610)
label1_2.config(font=("Courier", 15))
label_s_2.config(font=("Courier", 15))

label2_2 = Label(root, text="", bg="blue", fg="white")
label_f_2 = Label(root, text="Output frequency (Hz/100) = ", bg="blue", fg="white")
label2_2.place(x=1438,y=630)
```

```
label_f_2.place(x=1100,y=630)
label2_2.config(font=("Courier", 15))
label_f_2.config(font=("Courier", 15))

label3_2 = Label(root, text="", bg="blue", fg="white")
label_c_2 = Label(root, text="Motor current (A/100) = ", bg="blue", fg="white")
label3_2.place(x=1390,y=650)
label_c_2.place(x=1100,y=650)
label3_2.config(font=("Courier", 15))
label_c_2.config(font=("Courier", 15))

label4_2 = Label(root, text="", bg="blue", fg="white")
label_DC_2 = Label(root, text="DC-link voltage (V) = ", bg="blue", fg="white")
label4_2.place(x=1355,y=714)
label_DC_2.place(x=1100,y=714)
label4_2.config(font=("Courier", 15))
label_DC_2.config(font=("Courier", 15))

label5_2 = Label(root, text="", bg="blue", fg="white")
label_t_2 = Label(root, text="Motor torque (0.1%) = ", bg="blue", fg="white")
label5_2.place(x=1368,y=670)
label_t_2.place(x=1100,y=670)
label5_2.config(font=("Courier", 15))
label_t_2.config(font=("Courier", 15))

label6_2 = Label(root, text="", bg="blue", fg="white")
label_p_2 = Label(root, text="Motor power (0.1%) = ", bg="blue", fg="white")
label6_2.place(x=1353,y=690)
label_p_2.place(x=1100,y=690)
label6_2.config(font=("Courier", 15))
label_p_2.config(font=("Courier", 15))

label7 = Label(root, text=
"Max frequency control (0-2500Hz/100 Motor 1 & 0-1500Hz/100 Motor 2)",
bg="blue", fg="white")
label7.place(x=700,y=80)

motor1_label = Label(root, text="MOTOR 1", bg="blue", fg="white")

motor1_label.config(font=("Courier", 20))
motor1_label.place(x=550,y=160)

motor1_label_2 = Label(root, text="MOTOR 2", bg="blue", fg="white")
motor1_label_2.config(font=("Courier", 20))
motor1_label_2.place(x=1250,y=160)

slider = Scale(root, from_=0, to=2526, orient=HORIZONTAL, showvalue=0, length=200)
button_start = Button(root, text= "Start/Set", command = click_start, fg="white", bg="green")
button_stop = Button(root, text= "Stop", command = click_stop, fg="white", bg="red")
drop_menu = OptionMenu(root, clicked, *options)

slider.place(x=838,y=50)
button_start.place(x=927,y=20)
button_stop.place(x=870,y=20)
drop_menu.place(x=900, y=110)

gif()

root.protocol("WM_DELETE_WINDOW", quit_program)
root.mainloop()
```