Tampereen ammattikorkeakoulu

Ohjelmalohkojen vakiointi ja kehittäminen lavakuljetinjärjestelmässä

Mikko Itäluoma

OPINNÄYTETYÖ
Toukokuu 2024

Sähkö- ja automaatiotekniikan tutkinto-ohjelma
Automaatiotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Sähkö- ja automaatiotekniikan tutkinto-ohjelma
Automaatiotekniikka

ITÄLUOMA, MIKKO:
Ohjelmalohkojen vakiointi ja kehittäminen lavakuljetinjärjestelmässä

Opinnäytetyö 59 sivua, joista liitteitä 7 sivua
Toukokuu 2024

Opinnäytetyössä kehitettiin logiikkaohjelmoinnilla vakioidut ohjelmalohkot toimeksiantajayrityksen lavakuljetinjärjestelmissä käytetyille erilaisille kuljetintyypeille. Työ tehtiin osana asiakasprojektia, jossa toteutettiin lavakuljetinjärjestelmä käyttäen opinnäytetyössä kehitettyjä ohjelmalohkoja. Ohjelmalohkojen toimivuus todennettiin asiakasprojektin yhteydessä tehtyjen testien, sekä asiakasyrityksen valvoman tehdastestauksen pohjalta.

Ohjelmalohkojen vakioinnin tavoitteisiin vastaten luotiin mahdollisimman uudelleenkäytettävät, toimivat ja helposti käyttöönotettavat ohjelmalohkot, jotta vastaavia järjestelmiä pystytään toimeksiantajayrityksessä tekemään nopeammin ja tehokkaammin. Ohjelmalohkojen vakioinnilla pyritään edistämään toimeksiantajayrityksen myymien lavakuljetinjärjestelmien tuotteistamista.

Työssä perehdyttiin logiikkaohjelmoinnilla toteuttavan automaatioprojektin tekemiseen projektin alusta tehdastestaukseen saakka, sekä tehtiin jatkokehitystyötä ohjelmalohkoille asiakasprojektin valmistumisen jälkeen. Työn aikana määritettiin ohjelmalohkojen tarkoituksenmukainen toiminta ja suunniteltiin toteuttavan ohjelmiston rakenne. Lisäksi työssä suunniteltiin kuljetinlohkojen ohjelmat ja ohjelmoitiin kuljetinlohkot, minkä jälkeen aikaansaatu ohjelmisto käyttöönotettiin ja testattiin oikean laitteiston kanssa. Asiakasprojektin päätyttyä kuljetinlohkoille tehtiin jatkokehitystyönä käyttöönottoa helpottavia ja nopeuttavia muutoksia.

Tehty työ mahdollistaa toimeksiantajayrityksen myydä lähes valmista lavakuljetinjärjestelmäratkaisua. Yrityksen on tehdyillä ohjelmalohkoilla mahdollista toteuttaa vastaavia järjestelmiä nopeallakin aikataululla, kunhan käyttöönotossa on mahdollista tehdä sopivia sovelluksia ohjelmaan kohteen vaatimusten mukaan.

Asiasanat: logiikkaohjelmointi, kuljetinjärjestelmä, ohjelmistokehitys, automaatiprojekti, siemens

ABSTRACT

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

ITÄLUOMA, MIKKO:
Development and Standardizing of Program Blocks in a Pallet Conveyor System

Bachelor's thesis 59 pages, appendices 7 pages
May 2024

The purpose of this thesis was to develop standardized program blocks for multiple types of conveyors within a pallet conveyor system. The aim was to create and deploy a system that is not only validated but also quick to commission. The design and development were conducted as part of a customer project for JTA Connection Oy, during which the developed program blocks were implemented and examined. Implementation in a customer project served as a performance test for the developed software.

This thesis explores the design and development processes of the program blocks from the project's inception to factory testing. A discussion is provided on the various sub-phases of software engineering throughout the project. The intended functionality of the program blocks was defined, and the software architecture for the system implementing these blocks was designed. Subsequently, the programs for the blocks were designed and tested during the project. Further development work on the program blocks was carried out after the completion of the customer project, focusing on modifications that provide a more straightforward way to commission the system introduced in the thesis.

The end result was the successful implementation of standardized program blocks for a pallet conveyor system in a customer project, which was accepted by the customer during factory testing. The work accomplished also enables JTA Connection Oy to offer essentially complete pallet conveyor systems. Future systems can be composed rapidly and efficiently, provided that during deployment, appropriate applications are tailored to the program according to the specific site requirements.

Key words: plc programming, conveyor system, software development, automation project, siemens

SISÄLLYS

ERITYISSANASTO

Bufferointikuljetin	Kahden lavapaikan kokoinen kuljetin, joka pystyy välivarastoimaan kemikaalikontteja
FBD	Function Block Diagram, IEC 61131-3:n mukainen graafinen ohjelmointikieli
FB	Function block, ohjelmointielementti
HMI	Human-Machine Interface, käyttöliittymä
IO	Input/Output, tiedon siirtämistä eri komponenttien tai laitteiden välillä
Kemikaalikontti	Kemikaaleille tarkoitettu kuljetuslaatikko, joka vastaa pohjan mitoilta eurolavaa
Kenttäväylä	Teollisuudessa käytettävä tiedonsiirtoväylä
Kuljetinlohko	Tarkoittaa tässä opinnäytetyössä kaikkia lavan siirtämiseen tarkoitettuja ohjelmalohkoja
Kääntöpöytä	Yhden lavapaikan kokoinen kuljetin, joka pystyy kääntymään 90 astetta molempiin suuntiin
PLC	Programmable Logic Controller, Ohjelmoitava logiikka
Poistokuljetin	Yhden lavapaikan kokoinen kuljetin, jolta kemikaalikontit viedään pois järjestelmästä
SCL	Structured Control Language, Siemensin käyttämä IEC 61131-3:n mukainen tekstipohjainen ohjelmointikieli
Solu	Automaattinen tuotantoyksikkö
Tehdastestaus	Laitteiston testaaminen koneenrakentajan tiloissa
TIA Portal	Ohjelmointiympäristö Siemensin logiikoille
UDT	User Data Type, Ohjelmoijan määrittämä tietorakenne
Vastaanottokuljetin	Yhden lavapaikan kokoinen kuljetin, joka vastaanottaa järjestelmään kemikaalikontteja
Verkkotopologia	Tietoverkon perusrakenne liitetyistä laitteista

1 JOHDANTO

Tässä opinnäytetyössä käydään läpi, miten osana asiakastoimitusprojektin ohjelmiston toteutusta ohjelmoitavaan logiikkaan voidaan kehittää uudelleenkäytettäviä ohjelmalohkoja. Työssä kehitetään JTA Connection Oy:lle vakioitu tapa ohjelmoida yrityksen käyttämiä kuljettimia. Ohjelmointilohkojen vakiointi kuljettimille on osa yrityksen sisäistä tuotteistusprojektia, jonka on tarkoitus nopeuttaa kuljetinjärjestelmien käyttöönottoa ja helpottaa järjestelmien myyntiä.

Opinnäytetyön ohjelmointikehitystyö tehdään osana JTA Connection Oy:n asiakasprojektia. Työssä käydään läpi kuljetinlohkojen kehittämisen etenemistä projektin alusta loppuun. Projektiin liittyviä asioita käydään läpi keskittyen ohjelman kannalta tärkeisiin asioihin.

Tavoitteena on, että kehitystyön tuloksena syntyy yritykselle helposti, sekä nopeasti käyttöönotettavat vakioidut ohjelmalohkot neljälle erityyppiselle kuljettimelle ja luodaan kuljetintyyppiseen ohjaukseen pohjaa yrityksen sisällä. Aihe on yrityksessä kiinnostava, sillä kuljettimille ei ole ollut vakiintuneita ohjelmalohkoja tai ohjelmointitapoja, vaan vastaavia järjestelmiä on eri ohjelmoijien toimesta tehty eri tavoin.

Kuljetinlohkojen ohjelmallisena tavoitteena on, että erilaisia kuljetinjärjestelmiä voitaisiin rakentaa ohjelmaan ”ketjuttamalla” kuljettimia ja ne olisivat helppo myös ohjelmassa linkittää toisiinsa tavalla, jolla kuljettimien perustoiminta voitaisiin helposti ja nopeasti saada aikaiseksi.

1.1 JTA Connection Oy

JTA Connection on teollisuuden automaatioon erikoistunut yritys, joka tarjoaa monipuolisia automaatioratkaisuja asiakkailleen. JTA Connection suunnittelee ja toteuttaa itse tarjoamansa automaatioratkaisut ”avaimet käteen” -periaatteella.

JTA Connectionin asiakkaita löytyy monilta eri aloilta ja automaatioratkaisuja on tehty mm. elintarviketeollisuuteen, paperiteollisuuteen, sisälogistiikkojen toteuttamiseen, kokoonpanolinjastojen toteutukseen, sekä moneen muuhunkin.

JTA Connection on perustettu vuonna 1999 yhden henkilön yritykseksi, mutta on sittemmin kasvanut noin 20 miljoonan liikevaihtoa pyörittäväksi yritykseksi, joka toimii globaalisti. JTA Connectionilla on toimipisteitä 3 maassa: Suomessa, Virossa ja Saksassa, ja on noin 120:n eri alojen osaajien työllistäjä. (JTA connection 2024).

2 PROJEKTIN ESITTELY

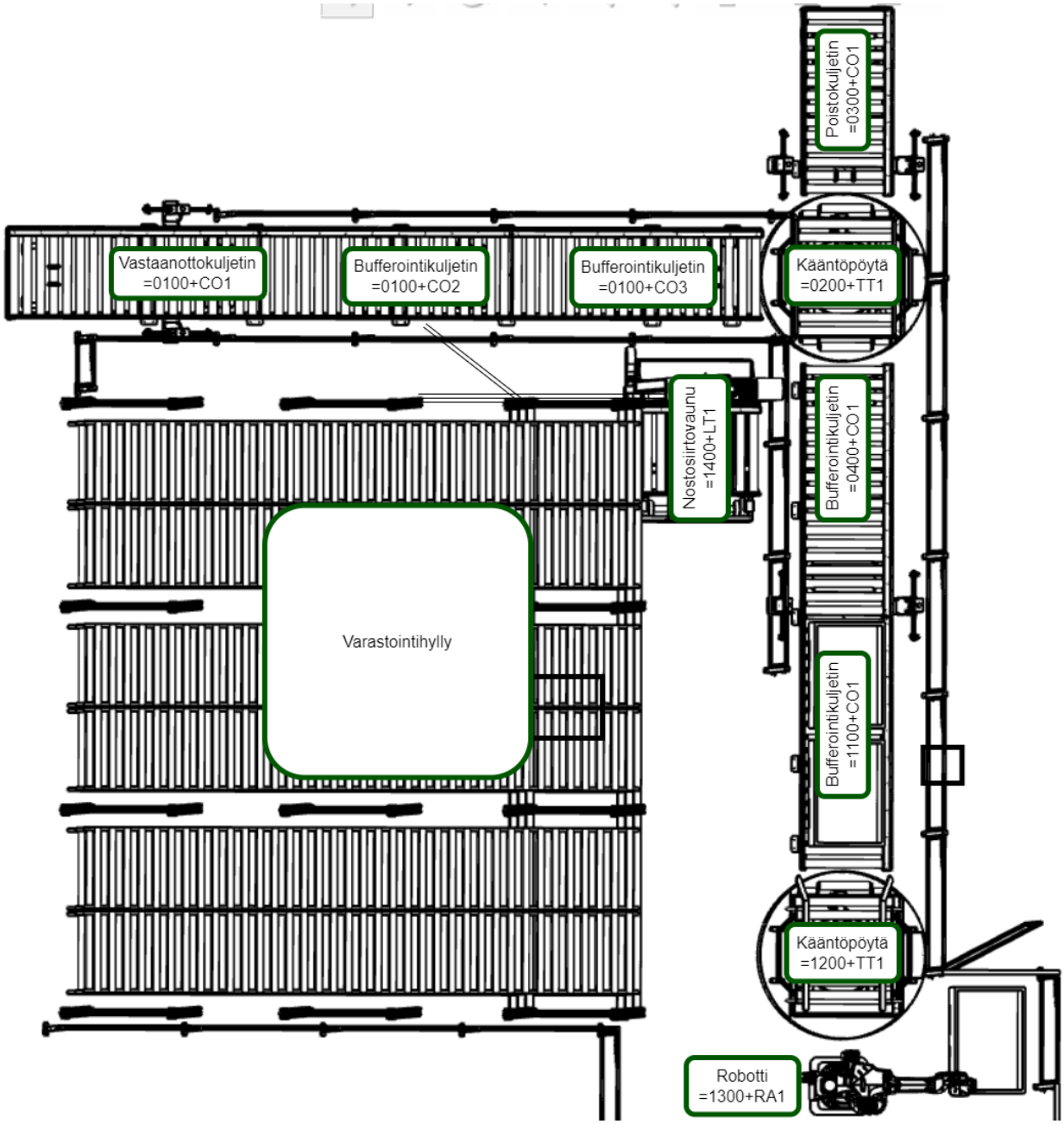
Tätä opinnäytetyötä tehtiin osana JTA Connection Oy:n asiakastoimitusprojektia. Opinnäytetyöntekijä toimi projektissa logiikkaohjelmoijana perussuunnitteluvaiheesta tehdastestaukseen saakka. Osana projektin ohjelmointisuunnittelua ja työtä pyrittiin kehittämään JTA Connectionille vakioitua ohjelmointitapaa ja vakioituja ohjelmointilohkoja JTA Connectionin käyttämiin kuljetinratkaisuihin. Opinnäytetyön aikana siis automatisoitiin koko toimitusprojektin järjestelmä, vaikka tässä opinnäytetyössä keskitytään pääasiallisesti kehitettyihin kuljettimien ohjelmalohkoihin ja kehittämisen eri vaiheisiin.

Tässä luvussa avataan projektikokonaisuutta ja käydään läpi projektissa käytetty laitteisto, sekä avataan järjestelmän tarkoituksenmukainen toiminta. Tämä kappale esittää samalla ensimmäistä työvaihetta automaatioprojektin ohjelmoinnin suunnittelun aloittamisessa, eli automatisoitavan järjestelmän laitteistoon ja toimintaan tutustumisen. Tätä vaihetta toteutetaan usein jo esisuunnitteluvaiheessa, jossa myös ohjelman tekijä suunnittelee automaatiojärjestelmään tarvittavia anturointeja ja toimintalogiikkaa. Tämä vaihe usein suoritetaan kapulanvaihtona esisuunnittelijan ja ohjelmantekijän välillä, jolloin jo tehty dokumentaatio on ohjelmantekijälle tärkeä. Näitä tärkeitä dokumentteja ovat muun muassa toiminnallinen kuvaus, sähköpiirustukset, laiteluettelot ja mahdolliset 3D-mallit järjestelmästä.

2.1 Solun esittely

Projektin solu koostuu kuljettimista, kääntöpöydistä, nostosiirtovaunusta, 3D-kamerasta, robotista ja varastointihyllystä. Näiden laitteiden avulla solu toteuttaa sille tarkoitettua tehtävää, josta kerrotaan tarkemmin toiminnankuvauksessa. Yksinkertaistettuna solun tehtävänä kuitenkin on kuljettaa kemikaalikontteja solussa olevalle robotille täytettäväksi kemikaalipusseilla. Solu toimii myös välivarastona tyhjille kemikaalikonteille.

Seuraavaksi esitellystä kuvan 1 kokoonpanon hahmotelmasta nähdään miten solun laitteet sijoittuvat järjestelmässä.



KUVA 1. Järjestelmän kokoonpano, josta selviää laitteiden positiotunnukset.

Kokoonpanokuvasta nähdään hyvin, kuinka materiaalivirta tulee liikkumaan ohjelmoitavassa järjestelmässä. Mitä kokoonpanokuvasta ei kuitenkaan selviä on yksityiskohtaisempi tieto siitä, että miten materiaalin virtaus käytännössä tulee tapahtumaan. Tässä kohtaa projektia kuitenkin ohjelmoija pyrkii hakemaan kokonaiskuvaa mitä olisi tarkoitus ohjelmoida, joten yksityiskohtainen tieto ei ole vielä tarpeen.

2.2 Toiminnankuvaus

Toiminnankuvaus toimii avaindokumenttina ohjelmoinnin toteutukselle. Se antaa perustiedot, miten materiaalivirta käyttäytyy järjestelmässä ja mitä vaatimuksia toiminnalle on. Toiminnankuvauksen pohjalta lähdettiin rakentamaan ohjelmaa ja siihen pystytään vertaamaan solun toimintaa myöhemmin ohjelmaa testatessa. Seuraavassa kappaleessa on kirjoitettuna toiminnankuvaus solulle.

Solu toimittaa solulle tuotuja tyhjiä kemikaalikontteja täytettäväksi täyttöpisteelle. Solulle tuodut kontit ovat yksilöityjä ja solu seuraa jokaisen kontin paikkatietoa reaaliajassa ja pystyy tarjoamaan paikannustiedot ulkopuoliseen järjestelmään. Solun vastaanottokuljettimelle toimitetut kontit kuljetetaan kohti täyttöpistettä (kääntöpöytää 1200+TT1) niin nopeasti kuin mahdollista. Ennen täyttöpistettä solu tarkastaa kontin tyhjyyden 3D-kameran avulla kääntöpöydällä 0200+TT1. Mikäli kontti ei ole tyhjä, se ajetaan poistopisteelle, josta kontti kulkeutuu tyhjennettäväksi. Kontin ollessa tyhjä, ajetaan se tarkastuspisteeltä kohti täyttöpistettä. Kontin päästyä täyttöpisteelle sen tiedot tarkastetaan lukemalla RFID-tagi ja robotti täyttää tyhjät kontit kemikaaleilla, jonka jälkeen solu varastoi täytetyt kontit läpivirtaushyllyyn.

Toiminnankuvaus voisi pitää sisällään hyvinkin yksityiskohtaista tietoa järjestelmän laitteiden toiminnasta. Tässä projektissa näin ei kuitenkaan ollut ja täten suunnittelulla on vapaammat rajat, jossa toimia parhaan ratkaisun luomiseksi.

2.3 Solun laitteet

Toiminnankuvauksen lisäksi ohjelman suunnittelun tueksi ohjelmoijan on hyvä tuntea laitteet, joita on ohjelmoimassa ja ne laitteet, joita on käytettävissä itse ohjelman suorittamiseen. Tässä luvussa käydään läpi opinnäytetyön kannalta olennaiset laitteet.

Ohjelmoitavat laitteet, kuten tässä projektissa kuljettimet ja kääntöpöydät, asettavat fyysisillä ominaisuuksillaan raamit, joiden pohjalta ohjelmaa voidaan tehdä. Laitteiden ohjaustavat ja anturoinnit vaikuttavat suurelta osin minkälaista ohjelmaa voidaan toteuttaa.

Esimerkkinä tästä voisi antaa, että on hyvin vaikeaa lämmittää vettä haluttuun lämpötilaan pelkän lämpövastuksen avulla ilman lämpötilan mittausta. Tämäkin on mahdollista ainakin teoriassa, mikäli ympäristön olosuhteet tunnetaan hyvin ja tiedetään veden määrä ja lähtölämpötila entuudestaan, sekä tunnetaan perusteellisesti lämpövastuksen veteen tuottama lämpövaikutus. Tämänlaisten tietojen pohjalta voidaan tehdä ohjelma, jonka pitäisi tuottaa haluttu lopputulos. Tämä vaatisi kuitenkin huomattavan paljon laskentaa ohjelmalta ja silti yhden muuttujan, kuten veden lähtölämpötilan, eroavaisuus lähtötietoihin voisi vesittää koko ohjelman. Yhden lämpötila-anturin lisääminen yksinkertaistaisi ja parantaisi prosessin toimivuutta ja luotettavuutta.

Ohjaustavan vaikutuksesta ohjelmaan voisi antaa esimerkin, että mikäli esimerkin lämpövastusta voitaisiin ohjata analogisesti arvoilla 0–100 %, käytettäisiin sitä ainoastaan totuusarvopohjaisesti TOSI (100 %) tai EPÄTOSI (0 %). Analogisella tavalla on mahdollisuus hienovaraisempaan lämpötilan hallintaa, kun taas totuusarvopohjaisesti säätö olisi varmasti sahaavaa.

Ohjelman suorittamiseen käytettävät laitteet taas on hyvä tuntea, jotta tietää minkälaisia ominaisuuksia ohjelmassa on mahdollista käyttää ja minkälainen suorituskyky ohjelmaa suorittavilla laitteilla on. Näitä laitteita projektissa ovat PLC ja HMI. Suorituskykyä ei tosin enää nykyään tarvitse useinkaan epäillä, sillä usein nykyaikaisissa laitteissa on enemmän kuin riittävästi suorituskykyä. Vain erityisissä tapauksissa laitteiden rajat tulevat vastaan (Kalliola, 2022, s. 32). Tästä huolimatta ohjelmaa tehdessä on hyvä pyrkiä ottamaan huomioon laitteiden prosessointikyky. Puhuttaessa ohjelmoitavista logiikoista enemmän prosessointia tarkoittaa pidempiä sykliaikoja, joka taas tarkoittaa ohjelman suorittamisnopeuden hidastumista (Siemens, a).

2.3.1 PLC

Solun ohjauksesta vastaa Siemensin ohjelmoitava logiikka 1515F-2. Tämä PLC on Siemensin tarjoamista ohjaimista edistyneintä ohjainten sarjaa S7-1500. Ohjain on todella tehokas ja sisältää eniten ominaisuuksia verrattuna muihin Sie-

mensin tarjoamiin ohjaimiin. Ohjain suorittaa projektin ohjelmaa ja hoitaa kommunikaation ulkoisiin järjestelmiin. Ohjain toimii myös turvalogiikkana järjestelmälle.

Kaikki ohjainmallit eivät voi toimia turvalogiikkana vaan tämä ominaisuus on erikseen mainittu ohjaimessa, mikäli se on kykeneväinen toimimaan turvalogiikkana. Turvalogiikka eroaa tavallisesta logiikasta siten, että ne ovat suunniteltu erityisesti turvatoimintojen suorittamiseen. Turvalogiikassa voi olla 2 tai 3 prosessoria, jotka suorittavat samaa logiikkaa ja tämän jälkeen ne tarkistavat toisensa, ovatko ne päätyneet samaan lopputulokseen ja mikäli lopputulokset eroavat järjestelmä ajetaan alas turvallisesti. (Bolton, 2015).

2.3.2 Käyttöliittymä

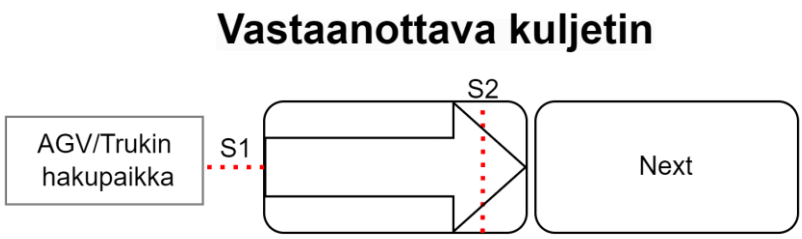
Käyttöliittymänä toimi Siemensin TP1200 Comfort -paneeli, joka on hyvin yleisesti teollisuudessa käytetty paneeli. Paneelin käyttö on integroitu osaksi TIA Portal ympäristöön ja täten se on TIA Portalissa myös helppo käyttöönottaa, sekä rakentaa siihen käyttöliittymää. Käyttöliittymän ja PLC:n välinen kommunikointi onnistuu Profinet-väyläkommunikaation avulla.

2.3.3 Ohjelmoitavat kuljettimet

Solussa konttien liikutteluun käytetyt kuljettimet ovat JTA:n omaa tuotantoa. Ne ovat suunniteltu yrityksessä ja ne valmistetaan itse. Näiden kuljettimien ohjauksia on vuosien varrella tehty usealla eri tyylillä eri ohjelmoijien kesken. Näille kuljettimille opinnäytetyössä on tarkoitus kehittää vakioidut ohjelmalohkot, jotka sopisivat eri käyttökohteisiin vain pienellä paikkakohtaisella ohjelmointityöllä. Kuljettimet ovat tyyliltään rullakuljettimia, joita ohjataan käyttäen oikosulkumoottoria. Niitä on kolmea erityyppistä: vastaanottavaa, bufferoivaa ja poistavaa laatua.

Kuljettimien konttien paikoitukseen tarkoitettu anturointi on integroitu kuljettimeen itseensä. Induktiiviset anturit vaikuttuvat, kun kuljettimiin asennettu mekanismi

vaikuttuu kontin liikkuessa mekanismin kohdalle. Anturoinnit ovat mitoitettu kuljettimille niin, että niillä kulkevat kontit jakaantuvat tasaisesti kuljettimille. Seuraavissa kuvissa hahmotellaan antureiden sijainteja kuljettimilla.

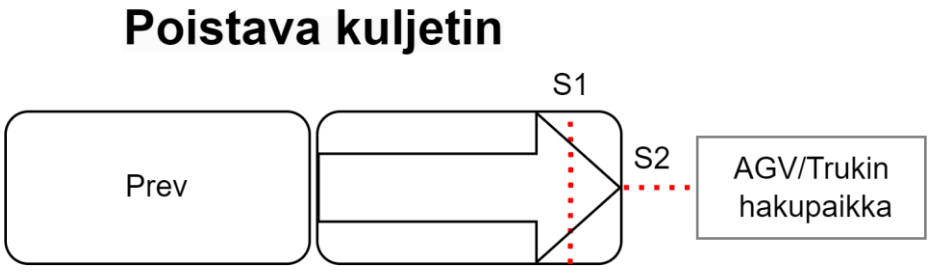


KUVA 2. Vastaanottavan kuljettimen anturointi.

Vastaanottavalla kuljettimella kontin tunnistavia antureita on yksi (S2), sillä vastaanottopaikka on mitoitettu yhden kontin kokoiseksi. Lisäksi kuljettimella on anturointi (S1), joka havaitsee konttia tuovan laitteen läsnäolon etäisyyden perusteella.

KUVA 3. Bufferoivan eli varastoivan kuljettimen anturointi.

Bufferoivalla kuljettimella antureita on kaksi kappaletta, jonka ansiosta pystytään päättelemään, onko kuljettimella yksi vai kaksi konttia.



KUVA 4. Poistavan kuljettimen anturointi.

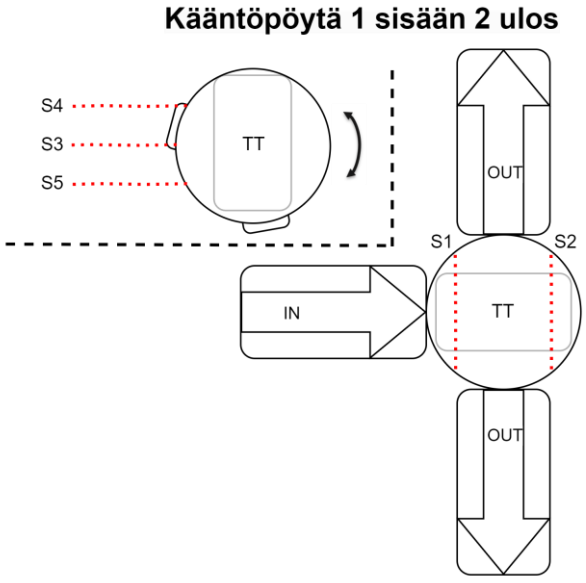
Poistavalla kuljettimella anturointi on pitkälti samanlainen kuin vastaanottavalla kuljettimellakin.

2.3.4 Ohjelmoitavat kääntöpöydät

Kääntöpöydät ovat niin ikään kuljettimien tapaan JTA:n omaa tuotantoa. Myös kääntöpöydät kuuluvat tämä opinnäytetyön aihealueeseen ohjelmalohkojen vakioinnissa.

Kääntöpöydän voi ajatella koostuvan kahdesta eri laitteesta, käännön tuottavasta mekaniikasta ja rullakuljettimesta, joka on rakennettu kääntölaitteen päälle. Näin kääntöpöytää voidaan käyttää saumattomasti rullakuljetinjärjestelmissä. Kääntöliikkeen ja rullakuljettimen ohjaukseen on kummallekin oma siihen funktioon tarkoitettu oikosulkumoottorinsa.

Anturointia kääntöpöydissä on viiden anturin verran. Kolme näistä on tarkoitettu pöydän käännön paikoittamiseen. Tämä kääntöliikkeen anturointi on integroitu kääntöpöytään itseensä. Loput kaksi anturia on tarkoitettu kontin havaitsemiseen pöydällä, mutta toisin kuin kuljettimissa, kääntöpöydillä kontteja havaitaan peilikennoantureiden avulla kontin sivusta. Anturointia on pyritty havainnollistamaan seuraavassa kuvassa.



KUVA 5. Havainnekuva kääntöpöytien anturoinnista.

Kuvassa 5 olevista antureista S1 ja S2 ovat tarkoitettu kontin tunnistamiseen, kun kontti on pöydällä, molemmat anturit ovat vaikuttuneet. S3, S4 ja S5 ovat tarkoitettu kääntöliikkeen hallintaan. Anturit S4 ja S5 tunnistavat käännön lähestyvän ääriasentoa ja anturin S3 tehtävänä on tunnistaa kääntöliikkeen olevan ääriasennossa.

3 OHJELMAN SUUNNITTELU JA TOTEUTUS

Tässä luvussa perehdytään ohjelman suunnitteluun, sekä ohjelman toteutukseen. Tämän vaiheen suunnittelun perustana toimi esisuunniteltu projekti, johon oli suunniteltu jo valmiiksi järjestelmän toiminta, sähköt ja projektissa käytettävät laitteet. Tässä vaiheessa ohjelmisto suunniteltiin lähtötietojen pohjalta rakennetasolla, jonka jälkeen aloitettiin itse ohjelmoiminen. Avaindokumenttina ohjelmiston suunnittelussa toimi toiminnallinen kuvaus.

3.1 TIA Portal ja automaatioprojektin luonti

Projektin kokonaisuuteen tutustumisen jälkeen ensimmäisenä työn vaiheena aloitettiin automaatioprojektin luomisella ohjelmointiympäristöön. Ohjelmointiympäristönä projektissa toimi Siemensin TIA Portal V18 -versio. Ohjelmiston nimen lyhenne TIA tulee sanoista ”Totally Integrated Automation” (Siemens 2024). Suomennettuna tämä tarkoittaa täysin integroitua automaatiota. Ohjelmointiympäristön on siis tarkoitus toimia ympäristönä, josta pystyy toteuttamaan koko automaatiojärjestelmän hallinnan. Kehitysympäristössä pystyy tekemään todella paljon ja tähän projektiin ympäristössä tehtiin mm. laitteiston verkkotopologia, väyläkommunikaation pystytys, laitteiden konfigurointi, käyttöliittymän rakennus ja kaikki ohjelmat pois lukien robotin liikkeiden ohjaus.

Valinta projektin automaation toteuttamisympäristölle tehtiin projektin asiakkaan toivomuksen mukaan. Asiakkaalla itsellään on myös käytössä sama ohjelmointiympäristö, jonka takia on luonnollisinta toteuttaa projekti samassa ympäristössä. Tästä on etua asiakkaalle projektin luovutuksen jälkeen, sillä näin asiakkaalla on mahdollisuus esimerkiksi vikatilanteen ilmetessä takuuajan päätyttyä selvitä omien osaajien avulla.

Ohjelmointiympäristön version valintaan taas vaikutti toimittajan (JTA Connection) halu kokeilla uudempaa versiota. Aikaisemmin käytössä ollut versio olisi myös ollut aikaisempien kokemuksien mukaan hyvä, mutta tämä projekti toimi kokeiluna uudesta versiosta ja sen uusista ominaisuuksia, sekä mahdollisista vioista. PLC ohjelmointia tekevät yritykset ja asiantuntijat ovat usein skeptisiä uusia

versioita kohtaan, sillä erilaisten kokemusten pohjalta epäillään niiden pitävän sisällään ennalta-arvaamattomia vikatilanteita.

Automaatioprojektin luominen tehtiin ohjelmointiympäristössä aloittamalla uusi projekti. Projektin luonnin yhteydessä lisättiin projektiin valikoitunut ohjelmoitava logiikka projektin hardwarekonfiguraatioon.

3.2 Automaatioprojektin pohjatyöt

Automaatioprojektin luomisen jälkeen rakennettiin TIA Portal kehitysympäristöön pohja tulevaa ohjelmistoa varten. Tässä vaiheessa projektia esisuunnittelun pohjalta on jo tehty laitevalinnat ja tehty alustavat sähköpiirustukset, joten automaatioprojektin pohjaa voitiin alkaa työstämään edellä mainittujen dokumenttien pohjalta. Pohjatöiksi luetellaan tässä opinnäytetyössä laitteiston ja ohjelmiston väliseen rajapintaan liittyvät työt kehitysympäristössä. Esisuunnittelun pohjalta voitiin jo sanoa laitteiston rakenteen olevan melko varma, joten voitiin kehitysympäristöön tehdä asioita kuten hardwaren lisääminen, I/O tagien luonti, väylätopologian rakentaminen, väylässä olevien laitteiden nimeäminen ja osoitteiden antaminen.

3.3 Ohjelmistoarkkitehtuurit

Ohjelmistoa suunniteltaessa, suunnitellaan toteutettavalle ohjelmistolle ensin ohjelmistoarkkitehtuuri, jonka mukaan ohjelma rakennetaan ja jonka pohjalta ohjelma toimii. Ohjelmistoarkkitehtuurin suunnitteleminen ja sen noudattaminen selkeyttää ohjelmaa. Näin ohjelman toimintaa on helpompi jakaa osiin ohjelmointityötä varten ja sitä on helpompi ymmärtää myös jälkikäteen ohjelman valmistuttua. Tämä helpottaa vianetsintää tulevaisuudessa ja mahdollistaa nopeampaa jatkokehitystä.

Ohjelmistoja suunniteltaessa voidaan tunnistaa toistuvia rakenteita ja näiden pohjalta voidaan pohtia ohjelmistoarkkitehtuurin tyyppiä. Näitä tyyppejä on useita erilaisia, eivätkä ne ole toisiaan poissulkevia, vaan samasta ohjelmistosta voidaan tunnistaa useita arkkitehtuurityyppejä (Asmala et al. 2005).

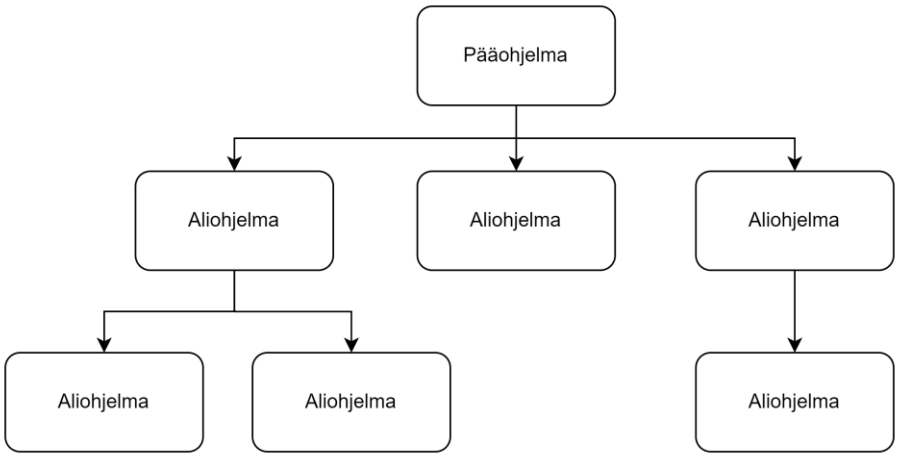
3.3.1 Tietovuoarkkitehtuuri

Tietovuoarkkitehtuureissa järjestelmä hahmotetaan sarjana perättäisiä tietomuunnoksia, jotka kulkevat järjestelmän osaprosessien läpi sisääntulosta aina määrättyyn päämääränsä saakka. Tietovuoarkkitehtuureja on kahta päätyyppiä: ”Batch sequential” ja ”Pipes and filters”. Näistä ensimmäisessä tietoa käsitellään aina kokonaisena jokaisessa osaprosesseissa ja jälkimmäisessä tieto kulkee osaprosessien läpi ikään kuin jatkuvana virtana. (Asmala et al. 2005).

3.3.2 Kutsumekanismiarkkitehtuuri

Kutsumekanismiarkkitehtuureissa ohjelman eri osien suorittaminen on keskiössä. Esimerkiksi ”Main program and subroutine” -mallissa ohjelma on jaettu pääohjelmaan ja aliohjelmiin (Asmala et al. 2005). Tämä malli on hyvin yleinen ja mallia käytetään luonnollisesti ohjelmoitavissa logiikoissa, joiden ohjelmien suoritus on jaettu tähän tapaan IEC 61131-3 -standardin mukaisesti.

Kerrosarkkitehtuurissa tasoja lisätään niin, että aliohjelmilla tai niiden aliohjelmillakin voi olla vielä aliohjelmia. Tässä arkkitehtuurimallissa yleensä kerrosten välille muodostuneita hierarkioita ei saisi rikkoa, eli tietojensiirron ja ohjelmakutsujen pitäisi tapahtua aina kerrosten välillä, eikä kerrosten yli saisi hyppiä. (Asmala et al. 2005).



KUVA 6. Kutsumekanismiarkkitehtuurin havainnekuva.

3.3.3 Riippumattomien komponenttien arkkitehtuuri

Riippumattomien komponenttien arkkitehtuureissa on pyritty rakentamaan ohjelma itsenäisistä prosesseista, jotka kommunikoivat keskenään. Tällä tavalla pyritään liittämään ohjelman osia mahdollisimman kevyesti keskenään. Tähän tapaan kuuluu useita kommunikointimalleja kuten ”broadcast” -kommunikaatio ja ”client-server” -kommunikaatio. (Asmala et al. 2005).

3.3.4 Tietokeskeinen arkkitehtuuri

Tietokeskeisessä arkkitehtuurimallissa järjestelmän eri osat kommunikoivat vain käyttäen määrättyä tietokantaa, johon on määritelty jokaiselle komponentille oma osoitealueensa. Tätä tapaa käyttäessä, voidaan komponentteja lisätä ilman, että se vaikuttaa jo olemassa oleviin komponentteihin. Tietokantaan voi suhtautua kahdella eri tavalla. Järjestelmän komponentit käyvät itse hakemassa komponentin toiminnan kannalta mielenkiintoiset tiedot tietokannasta tai tietokanta ilmoittaa komponenteille, kun niitä kiinnostava tieto muuttuu. Näistä ensimmäistä kutsutaan passiiviseksi tietokannaksi ja jälkimmäistä aktiiviseksi. (Asmala et al. 2005).

3.4 Ohjelman arkkitehtuurin suunnittelu

Solun ohjelmiston suunnittelu aloitettiin suunnittelemalla ohjelmiston arkkitehtuuria. Tässä kohtaa pidettiin mielessä kuljettimien lohkot, joista haluttiin vakioidut versiot. Arkkitehtuurimalliksi koko solun ohjelmiston kannalta toteutettiin kerrosarkkitehtuuri, joka TIA Portal kehitysympäristössä on loogista toteuttaa, sillä kehitysympäristö ohjaa käyttäjää vahvasti tähän suuntaan samoin kuin logiikkaohjelmiin tarkoitettu IEC 61131-3 standardikin.

Korkealaatuiseen ohjelmaan pyrittäessä systemaattinen tapa tehdä ja lähestyä ohjelman luomistyötä parantaa mahdollisuuksia toteuttaa laadukas ohjelma ja voi nopeuttaa sen tekemistä (Bolton, 2015). Arkkitehtuurimallin hahmottaminen ja noudattaminen on yksi tapa systemaattiseen työskentelyyn. Systemaattiseen tapaan kuuluu myös ohjelman eri osien määritteleminen ja pilkkominen hallittaviin

kokonaisuuksiin. Lisäksi ennen itse ohjelman tekemistä, kannattaa näiden jaettujen osien ohjelmointiin käytettävät ratkaisualgoritmit määrittää käyttäen apuna esimerkiksi pseudokoodia tai vuokaavioita.

3.4.1 Ohjelmointikielet

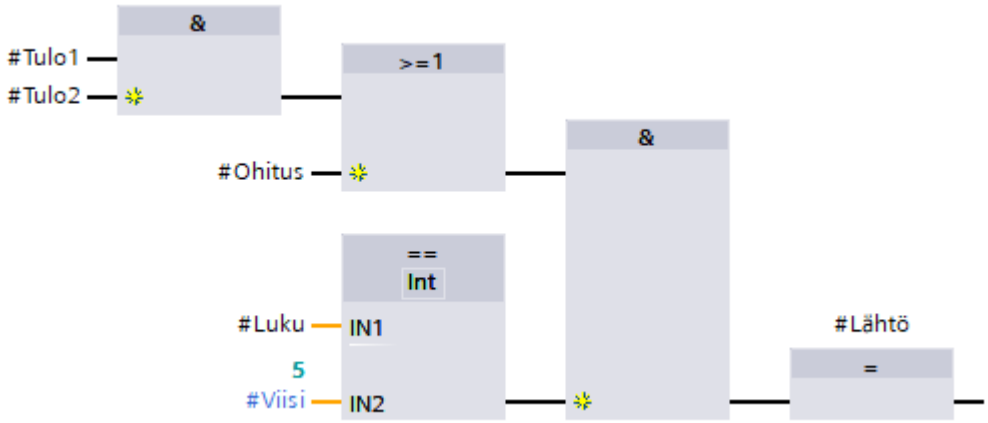
Ohjelmointikielien valinta tähän projektiin oli helppoa tehdä, sillä yrityksen puolelta käytäntönä logiikkaohjelmissa on käyttää standardin IEC 61131-3 mukaista kahta kieltä, graafista FBD:tä (Function Block Diagram) tai tekstipohjaista kieltä, joka Siemensillä tarkoittaa SCL:ää (Structured Control Language). Ohjelman koodaamisessa käytettiin molempia kieliä, sillä kummallakin ohjelmointikielellä on omat etunsa ja niitä yhdistelemällä voi saada aikaan helpommin ymmärrettävän ohjelman.

SCL on tekstipohjainen ohjelmointikieli, joka muistuttaa olemukseltaan huomattavan paljon Pascal-kieltä. SCL-kielen vahvuudet ovat mm. tietojenkäsittelyssä, matematiikassa ja monimutkaisten algoritmien suorituksessa. Haasteita kielessä voi aiheuttaa vianetsintä, sillä kieli ei ole kovin visuaalinen.

```
1  //Muuttujien arvojen määritys
2  #int := 5;
3  #string := 'Esimerkki'; // 9 merkkiä pitkä eli viimeinen indeksi [8]
4  #char := 'l';
5
6  IF #bool THEN // jos muuttuja arvoon true niin suoritetaan seuraava
7      #int := (#int + 10)*2; //Lopputulos 30
8      #string[9] := #char; //Lopputulos 'Esimerkkil'
9  END_IF;
```

KUVA 7. Esimerkki SCL-kielestä.

FBD on graafinen ohjelmointikieli, joka perustuu visuaalisiin lohkoihin. Kieli on helposti luettavaa ja ymmärrettävää visuaalisuutensa ansiosta, sekä sitä on lähtökohtaisesti helppo oppia. FBD:n vahvuuksia ovatkin sen yksinkertaisuus ja helppo luettavuus, joka helpottaa vianetsinnässä. Heikkoutena kielellä on, että järjestelmän monimutkaistuessa ohjelmista voi tulla näytöllä hyvin isokokoisia ja hankalasti luettavia.



KUVA 8. Esimerkki FBD-kielestä.

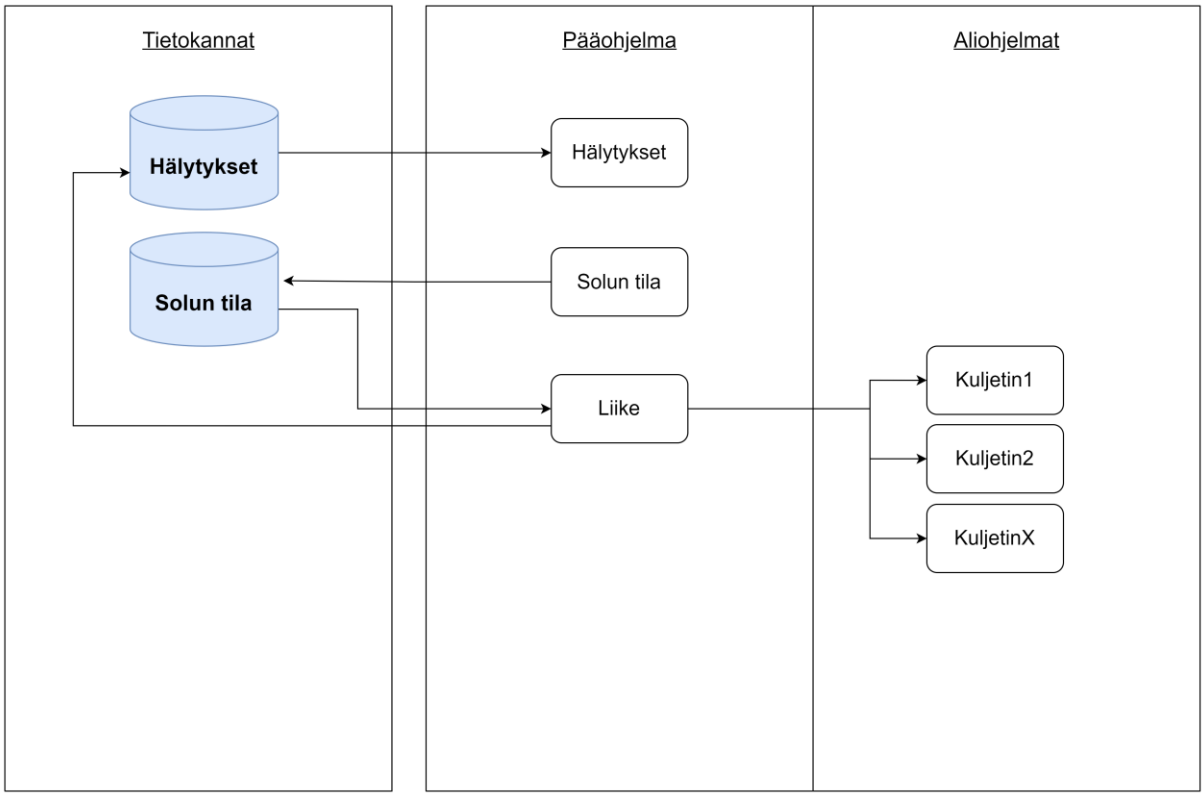
3.4.2 Pääohjelman rakenne

Ohjelmisto jaettiin pääohjelmassa kolmeen osaan, jotka suoritetaan luetellussa järjestyksessä: Hälytykset, Solun tila ja Liike -lohkoihin. Näissä lohkoissa suoritetaan logiikkaa liittyen hiukan eri asioihin.

Hälytykset kutsuaan omassa lohkossaan keskitetysti, josta niitä on helppo hallita ja kun hälytykset luetaan ensimmäisenä ohjelman suorituksessa, pystytään niihin heti reagoimaan muuttamalla solun tilaa ennen liikkeiden ohjauksia. Hälytyksiä generoidaan muusta ohjelmasta ja hälytyslohkossa tehdään päättely vian vakavuudesta ja siitä, miten hälytykseen reagoidaan.

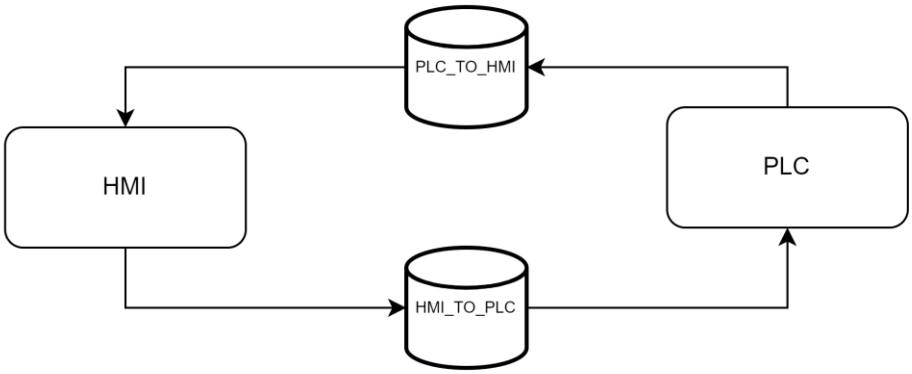
Solun tila -lohkossa hallitaan keskitetysti solun tiloja Auto, Stop ja Service. Solun tila -lohko perustui JTA:lla usein aikaisemminkin käytettyyn ohjelmalohkoon ja tämä ohjaustapa on vakioitunut käytäntö yrityksessä. Solun tilasta data vietiin globaaliin tietokantaan, jonne voitiin tallettaa lohkon palauttamat tiedot, jotta tilojen muutokset ovat helposti myös Liike -lohkon käytettävissä.

Viimeisessä Liike -lohkossa hallitaan keskitetysti laitteita, joilla solun liikkeitä tuottavia laitteita kuten kuljettimia hallitaan. Liike -lohko lukee solun tilaa ja reagoi tarvittaessa siihen ja sen muutoksiin. Lohkossa on omat virheenkäsittelynsä ja mahdolliset virheet kirjoitetaan ylös tietokantaan, josta hälytyksille tarkoitettu lohko käy lukemassa hälytystietoja ja ohjaa hälytykset eteenpäin järjestelmässä oikeisiin paikkoihin.



KUVA 9. Havainnollistus ohjelman rakenteesta.

Pääohjelman suorittamisrakenteen ohella suunniteltiin myös tiedon siirtämistä logiikalta käyttöliittymään ja päinvastoin. Tämä kommunikaatio päätettiin toteuttaa luomalla laitteiden välille kaksi tietokantaa, jotka toimisivat rajapintana. Toista tietokantaa käytettiin logiikan tietojen kirjoittamiseen käyttöliittymää varten ja toista tietokantaa käyttöliittymän tietojen, kuten ohjausten kirjoittamiseen logiikkaa varten. Kuvassa 10 on havainnollistettu PLC:n ja HMI:n välistä tiedonsiirtoa.



KUVA 10. Havainnollistus tiedonvaihdosta käyttöliittymän ja logiikan välillä.

3.5 Kuljetinlohkojen suunnittelu

Kuljettimien ohjelmalohkoja lähdettiin suunnittelemaan ajatuksella, että lohkoja voidaan käyttää onnistuneesti tämä projektin kokoonpanossa ja myös tulevaisuuden projekteissa. Olennainen huomioonotettava asia suunnittelussa oli myös helpottaa ja nopeuttaa kuljetinjärjestelmien käyttöönottoa. Vaatimukset kuljettimien toimintatavalle on määritetty toiminnankuvauksessa ja sen pohjalta kappaleiden ohjauksia ja liikettä kuljettimiin lähdettiin toteuttamaan.

3.5.1 Haasteet

Ajatus kuljettimista on yksinkertainen, ne siirtävät tavaraa eteenpäin ja anturille päästyään pysähtyvät. Täysin automatisoidussa kuljetinjärjestelmässä tilanne ei kuitenkaan ole kokonaisuuden kannalta katsottuna aivan niin yksinkertainen. Kappaleita on kuljettimilla samanaikaisesti monta ja halutaan tietää tarkalleen jokaisen kappaleen sijainti järjestelmässä. Lisäksi halutaan saada virhetilanteet kiinni ja virheistä toipuminenkin halutaan mielellään mahdollisimman automaattiseks. Yksittäisen kuljettimen toiminnan kannalta tilanne kuitenkin on edelleen suhteellisen yksinkertainen. Yksittäisen kuljettimen näkökulmasta sillä on oikeastaan vain 2 tehtävää: vastaanottaa kappaleita tai lähettää niitä eteenpäin.

Ohjelmalohkojen suunnittelua haastoi myös ajatus tulevaisuuden projekteista. Oli otettava huomioon, että tulevaisuuden projekteissa voitaisiin käyttää hyvin erilaisia kuljetinkokoonpanoja ja lohkojen olisi silti tarkoitus toimia oikealla tavalla. Täytyi siis luoda tapa kuljettimien välille, jolla kuljettimet voidaan laittaa kokoonpanoon ihan millä tavalla tahansa ja vain linkittämällä kuljettimet ohjelmassa toisiinsa halutussa järjestyksessä järjestelmä toimisi.

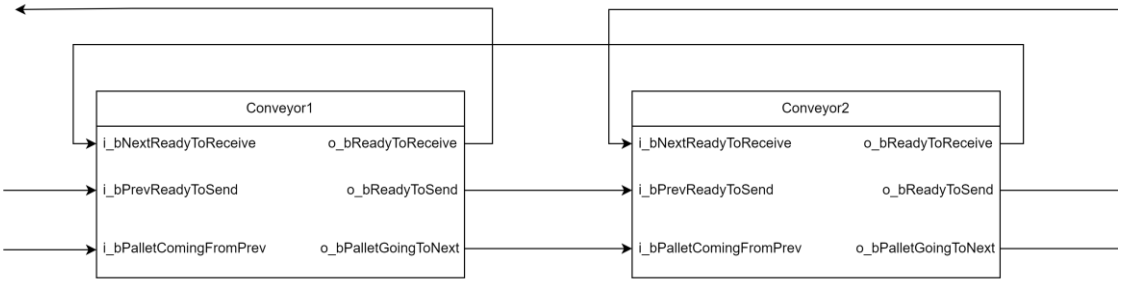
Ohjelmalohkoista haluttiin myös tehdä sellaisia, että aina yhdelle kuljetintyypille on olemassa yksi ohjelmalohko, jota voidaan käyttää ohjelman sisällä uudelleen ja uudelleen, ikään kuin funktiota. Tämä ajatusmalli on vähän samantapainen kuin olio-ohjelmoinnissa, vaikkakin kaikkia olio-ohjelman piirteitä ei IEC 61131-3 määrittelemillä kielillä logiikkaohjelmoinnissa pystykään täysin toteuttamaan. Perusperiaate tuottaa selkeästi strukturoitua uudelleenkäytettävää ohjelmaa kuitenkin pitää.

3.5.2 Lohkojen olemus ja niiden keskinen suhde

Kuljetinlohkoille asetettujen vaatimusten pohjalta lohkojen toimintaa hahmotettiin yksilöinä järjestelmässä, jokainen lohko olisi oma itsenäinen toimija, joka ohjaa omaa kuljetintaan ja pitää huolen omista asioistaan lukien kuitenkin yleisesti järjestelmän tilaa ja reagoimalla sen muutoksiin. Kuljetin toimiakseen osana kuljetinjärjestelmää vaatii kuitenkin tietoa muista kuljettimista ja niiden tiloista, joten kuljettimien välille täytyy olla rakennettu jonkinlaista viestintää. Lohkoista haluttiin yleispätevät jokaiselle kuljetintyypille, joten kuljetinten toimintaa ei voinut suunnitella toimivaksi vain tässä kokoonpanossa vaan täytyi ajatella, että lohkot ovat hyvin itsenäisiä ja toimivat monessa ympäristössä.

Kuljettimien välinen viestintä päätettiin suunnitella joustavaksi eri tilanteisiin ja yhdenmukaiseksi kaikkien kuljetintyyppien kesken. Tätä tarkoitusta varten luotiin siis vakioitu tapa toimia, joka toimisi pohjana kuljettimen interaktiossa muihin kuljettimiin.

Kuljettimien välinen kytkös tehtiin niin, että jokainen kuljetin viesti omaa tilaansa ulospäin muille kuljettimille. Näin ympäröivät kuljettimet voivat reagoida oman toimintansa kannalta tärkeisiin toisiin kuljettimiin. Mikäli siis kahden kuljettimen välissä oleva kuljetin tahtoisi tietää onko edellisellä kuljettimella konttia, se voisi käydä lukemassa edellisen kuljettimen tilan ja sen jälkeen mahdollisesti viestiä, että olisi valmis vastaanottamaan edellisen kuljettimen kontin, joka johtaisi toimenpiteeseen vastaanottaa kontti. Kuvaan 11 on hahmoteltu kuljettimien välinen viestintä.



KUVA 11. Havainnollistus kuinka kuljettimet viestivät keskenään.

Tätä kytköstä voisi kutsua myös kuljettimien väliseksi kättelyksi, vaikka kuljettimet eivät varsinaisesti kättele vaan tarkastelevat vain toistensa tiloja jatkuvasti ohjel-

masyklin mukaan. Ennen kuin kuljetin voi tehdä siirtoa seuraavalle, on sen varmistettava toisen valmius vastaanottaa ja kun varmistus toiselta kuljettimelta on saatu, suoritetaan tehtävä, jonka jälkeen kuljettimet hakevat tilaansa taas uudelleen ja selvittävät mitä tehdä seuraavaksi. Kuljettimet siis ikään kuin leijuttavat omaa statustaan muille nähtäväksi ja lukevat itse itseään kiinnostavia tietoja muilta kuljettimilta, jonka perusteella syntyy päätöksiä tehdä jotain.

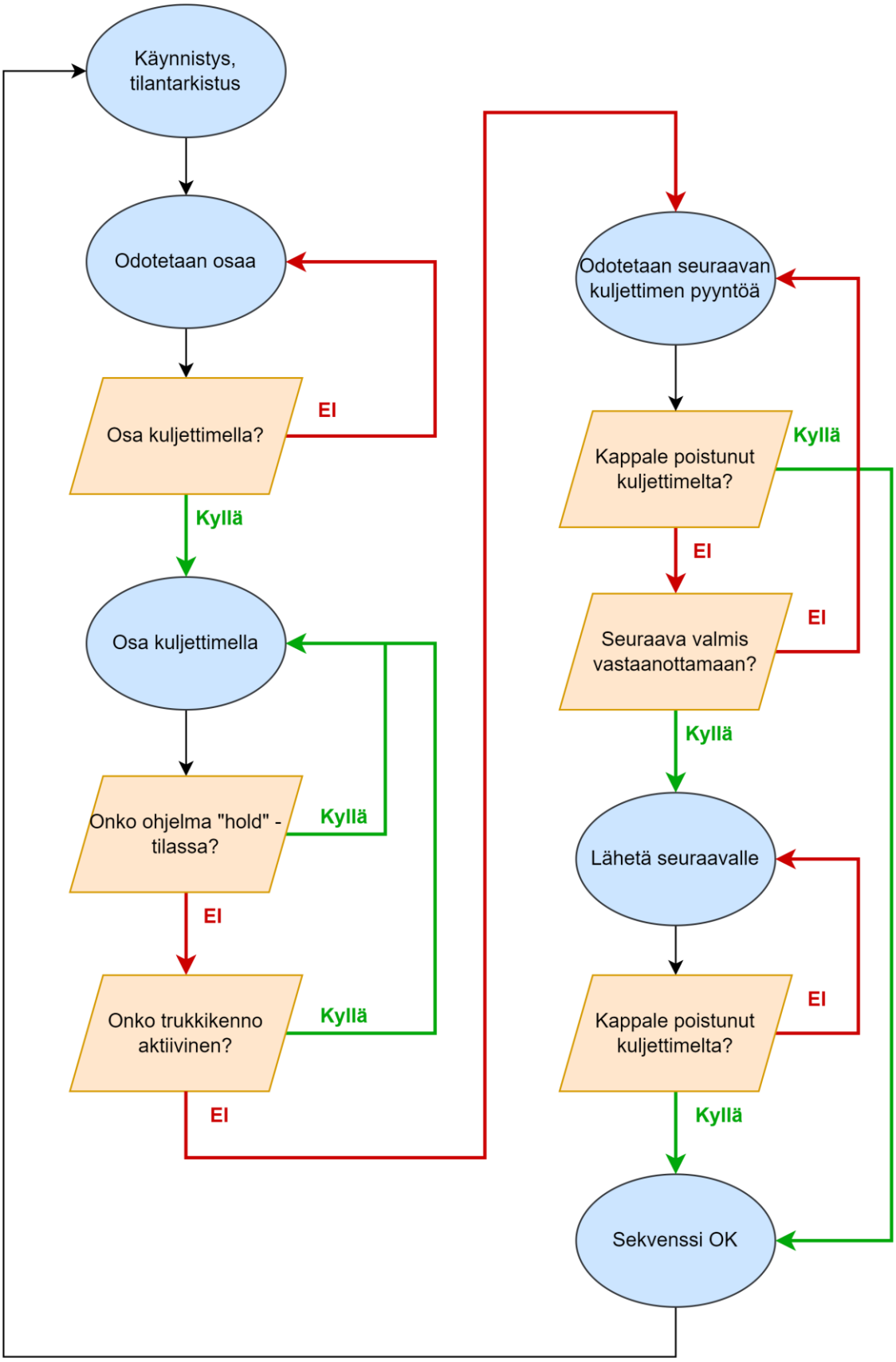
3.5.3 Kuljetintyyppien toiminnan suunnittelu

Kuljettimien toimintaa suunniteltiin vuokaavioiden avulla miettien erilaisia toimintoja ja tilanteita mitä milläkin kuljetintyypillä voi olla. Samaa suunnitteluprosessia käytettiin, jokaista kuljetintyyppiä varten. Tässä opinnäytetyössä ei kuitenkaan käydä yksityiskohtaisesti läpi jokaisen kuljetintyypin suunnitteluprosessia vaan yleisemmin mitä tehtiin suunnittelun eri vaiheissa.

Ensin mietittiin eri skenaariot, mitä suunniteltavalla kuljetintyypillä voi esiintyä. Vastaanottavalla kuljettimella on vain yhden lavan verran tilaa, ja sillä ei ole muuta suuntaa kuin eteenpäin, joten sillä vaihtoehtoisia tilanteita on vähemmän, kuin esimerkiksi keskellä kuljetinlinjaa olevalla kuljettimella. Nämä vastaanottavan kuljettimen eri skenaariot ovat:

- kuljetin on tyhjä
- kuljettimelle tuodaan konttia
- kuljettimella on kontti
- kontti lähetetään eteenpäin kuljettimelta

Vastaanottavan kuljettimen toiminta on suoraviivainen ja ennalta tiedetty. Näissä eri skenaarioissa oikean toiminnan kannalta täytyy kuitenkin miettiä, että miten näissä tilanteissa edetään. Vuokaavion avulla voitiin hahmotella miten ohjelma toimisi kuljettimen osalta. Seuraavalla sivulla kuvassa 12 on esitetty vuokaavion avulla ohjelman toimintaa.



KUVA 12. Vastaanottokuljettimen toiminnan vuokaavioesitys.

Vastaanottokuljetin oli kuljetintyypeistä yksinkertaisin ja suoraviivaisin suunniteltava. Vastaanottavalla kuljettimella on karkeasti 4 skenaariota, joihin varautua, kun taas bufferoivalla kuljetintyypillä erilaisia tilanteita keksittiin 11, johtuen useammasta lavapaikasta ja siitä, että tällä kuljetintyypillä kuljettimen edellä ja takana voi olla kuljetin. Näitä eri skenaarioita kuljettimelle ovat:

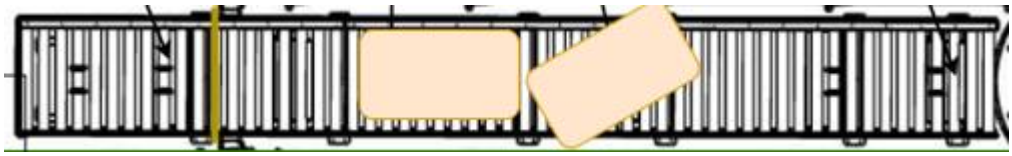
- kuljetin tyhjä
- kuljettimella on kontti ensimmäisellä lavapaikalla
- kuljettimella on kontti toisella lavapaikalla
- kuljetin on täynnä
- vastaanotetaan kuljettimen ollessa tyhjä
- siirretään lava ensimmäiseltä lavapaikalta jälkimmäiselle
- vastaanotetaan kuljettimen ensimmäisen lavapaikan ollessa täytettynä
- vastaanotetaan kuljettimelle samaan aikaan, kun lähetetään konttia eteenpäin jälkimmäiseltä lavapaikalta
- vastaanotetaan kuljettimelle samaan aikaan, kun lähetetään täydeltä kuljettimelta konttia eteenpäin
- lähetetään kontti eteenpäin ensimmäiseltä lavapaikalta
- lähetetään kontti eteenpäin jälkimmäiseltä lavapaikalta

Skenaarioiden määrän lisääntyessä ohjelma monimutkaistuu väistämättä. Työkalujen kuten vuokaavioiden avulla kuitenkin voi huomata jo ongelmatilanteet ennalta ennen itse ohjelmointia, jolloin ohjelman tekohetkellä voi keskittyä itse ohjelman toiminnallisuuteen eikä enää suunnitella ohjelman kulkua.

3.5.4 Virhetilanteisiin varautuminen

Suunniteltaessa kuljetintyyppien toimintalogiikkaa varauduttiin samalla myös ennakkoon erilaisia mahdollisia virhetilanteita varten, jotta ne voidaan myöhemmin ohjelmassa ottaa huomioon. Virhetilanteisiin varautuessa pyrittiin pohtimaan erilaiset virheskenaariot ja miten ne voidaan ohjelmassa havaita. Jokainen kuljetintyyppi käytiin erikseen läpi ajatuksen kanssa, että mitä voi kesken ohjelman ajon tapahtua. Esimerkiksi tyypillinen virhetilanne voisi olla, että ajettu kontti jää syystä tai toisesta jumiin eikä koskaan saavuta raja-anturia. Tällaisessa tilanteessa täytyy olla jonkinlainen aikaikkuna, jonka sisällä vastaanottavan kuljettimen täytyy

saada kontti vastaanotettua, muuten aiheutetaan hälytys. Toinen tyypillinen tilanne voisi olla, että kuljettimella on ollut kontti ja se anturitiedon mukaan yhtäkkiä häviäisi kuljettimelta. Tässäkin tilanteessa olisi aiheutettava hälytys ja tarkistaa järjestelmän oikea tila ja mahdollisesti anturit anturivian varalta.



KUVA 13. Mahdollinen virhetilanne kuljetinlinjalla.

Kaikkia virhetilanteita ei kuitenkaan pysty ennakolta ennustamaan ohjelmassa, sillä ohjelman suorituksen aikana voi syntyä loogisia reittejä, joita ei ole ohjelmaan tarkoitettu, jotka aiheuttavat ennalta-arvaamattoman toiminnan. Tätä varten kuitenkin ohjelmistoa pyritään testaamaan mahdollisimman paljon useassa eri vaiheessa. (Glass, 2001).

3.6 Kuljetinlohkojen ohjelmointi

Aikaisemmin suunniteltujen asioiden ja näissä esiintyneiden vaatimusten pohjalta ruvettiin ohjelmoimaan kuljetinlohkoja niin, että ne täyttäisivät niille asetetut vaatimukset ja toteuttavat suunniteltua toimintaa. Samat työvaiheet koskivat jokaista kuljetintyyppiä.

Ohjelmointityö aloitettiin tekemällä automaatioprojektiin jokaiselle kuljetintyypille oma funktiolohko (FB). Tähän funktiolohkoon alettiin ensitöiksi rakentamaan rajapintaa siitä, että mitä tuloja ja lähtöjä lohkolle halutaan syöttää ja mitä lohkosta halutaan ulos. Rajapinnan rakentamisen jälkeen aloitettiin toiminnallisuuteen liittyvän ohjelmoinnin tekeminen.

3.6.1 Rajapinta ja muuttujien nimeäminen

Rajapinta pyrittiin rakentamaan ensin karkeasti oikeaksi, jotta ohjelmoinnin tekeminen helpottuisi. Ohjelman tekeminen helpottuu, sillä sitten ohjelmointia tehdessä ei ohjelmoijan enää tarvitse miettiä, että minkälaisia signaaleja olikaan käytettävissä.

Muuttujien nimeäminen rajapintaan ja muuallekin ohjelmaan tehtiin laitevalmistaja Beckhoff:n nimeämiskäytäntöjen mukaan. Käytännön mukaan muuttujan nimi aloitetaan kirjoittamalla pienillä kirjaimilla muuttujan datatyyppiä vastaava etuliite, jonka jälkeen kirjoitetaan muuttujan nimi isolla alkukirjaimella. (Beckhoff 2024).

Nimeämiskäytännön lisäksi muuttujille pyrittiin antamaan lyhyet, mutta helposti ymmärrettävät ja kattavat nimet. Muuttujien ymmärrettävästi nimeäminen voi parantaa ohjelman luettavuutta ja vaikuttaa omalta osaltaan ohjelman laatuun.

Rajapinnan teko aloitettiin tuloista. Ensin lisättiin solun ohjauksen moodi, kuljettimella tarvittavat anturitiedot ja kommunikaatiosignaalit muilta kuljettimilta. Lähdöt lisättiin tämän jälkeen. Lähtöihin haluttiin ohjaustieto kuljettimen moottorinohjaukselle, sekä kommunikaatiosignaalit toisille kuljettimille. Signaalit löytyvät taulukosta 1.

TAULUKKO 1. Vastaanottavan kuljettimen tulot ja lähdöt.

Input	Output
i_nMode	o_bRunForward
i_bPartAtConv	o_bRunBackward
i_bHoldSensor	o_bReadyToReceive
i_bMotorSafetySwitch	o_bReadyToSend
i_bNextReadyToReceive	o_bPartGoingToNext

Tässä vaiheessa työtä nämä signaalit eivät ole lopulliset, vaan työn edetessä pystytään lisäämään tarvittavia signaaleja sitä mukaa, kun huomataan, että niitä tarvitaan.

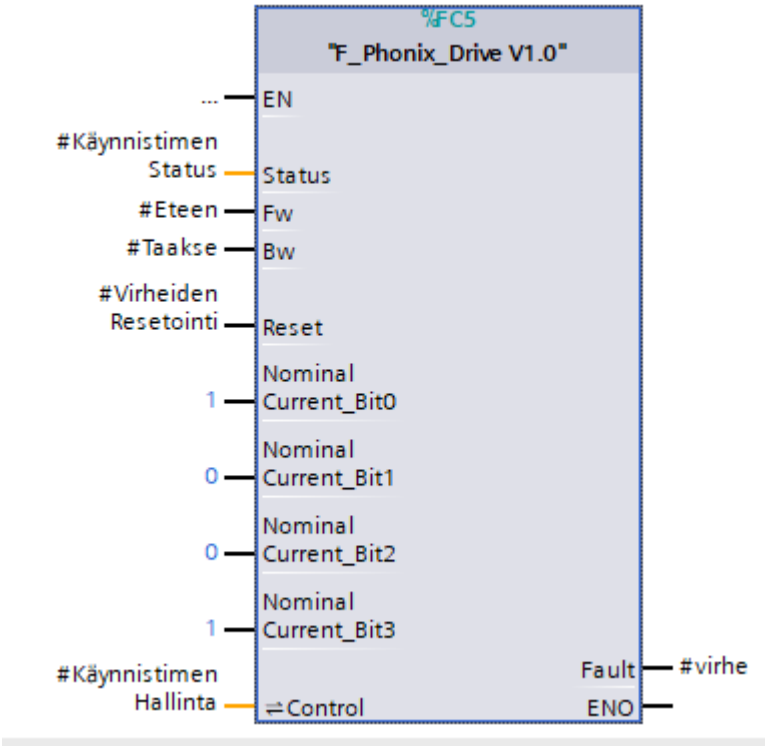
3.6.2 Kuljettimien ohjaustavat

Kuljetinjärjestelmässä on erilaisia kuljettimia ja osa näistä kuljettimista ontässä projektissa kääntöpöytiä. Ohjaukset näille laitteille ovat rullakuljettimien osalta samanlaiset, joko ajetaan eteen- tai taaksepäin.

Projektin lähtötietojen pohjalta tiedettiin, että ohjaus lähes kaikille lavoja siirtävillä rullakuljettimilla tapahtuu väylän yli hybridikäynnistimiä ohjaten. Ainoana poik-

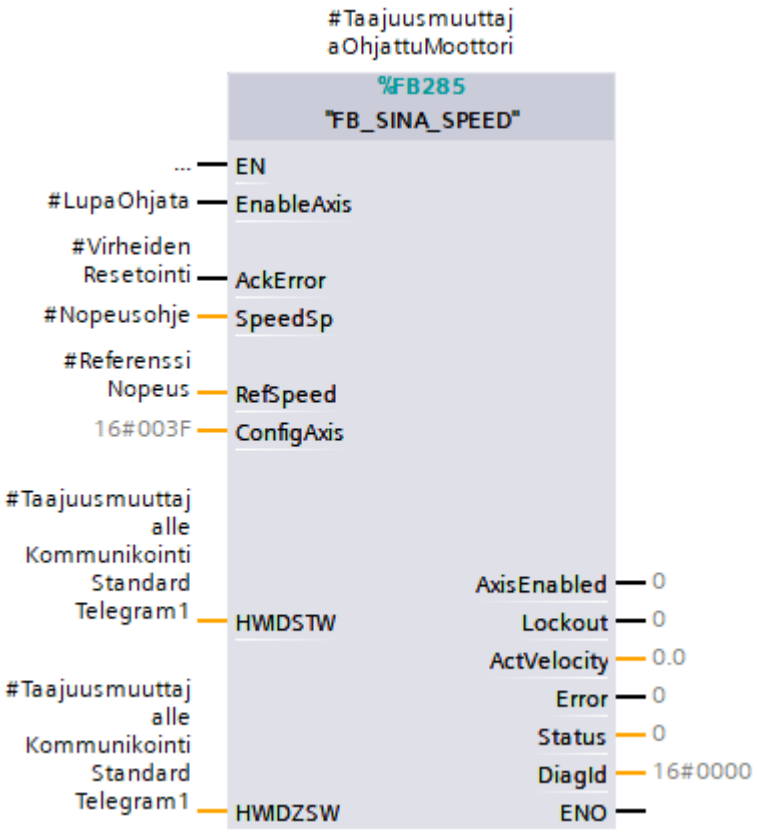
keuksena järjestelmän viimeinen kääntöpöytä, jonka rullakuljettimen ohjaus tapahtui taajuusmuuttajaohjauksella. Tämän rullakuljettimen lisäksi kääntöpöytien kääntöliikkeiden ohjaukset toteutettiin taajuusmuuttajilla.

Ohjaus hybridikäynnistimille tapahtui niiden ohjaukseen tarkoitetulla ohjelmalohkolla, jolla ohjattiin yksilöidyille käynnistimille tarkoitettuja ohjaus ja status signaaleja. Näitä signaaleja käyttämällä kuljettimien ohjaus oli aika yksinkertainen toteuttaa, sillä käynnistimille ei tarvinnut osoittaa muuta kuin ”ohjaus eteen” tai ”ohjaus taakse” -käskyjä. Käytännössä siis päälle/pois -ohjaus. (Phoenix contact 2024).



KUVA 14. Hybridikäynnistimien ohjauslohko.

Taajuusmuuttajien ohjaus ei ollut käytännössä hybridikäynnistimiä monimutkaisempaa ohjelmallisesti. Ohjaukseen oli olemassa Siemensin taajuusmuuttajille tarkoitettu ohjauslohko, jota voitiin hyödyntää. Eroavaisuus ohjauksessa on vain, että se annetaan nopeusohjeena ja suunnanvalinta tapahtuu nopeusohjeen etumerkkiä vaihtamalla. (Siemens, b).



KUVA 15. Taajuusmuuttajan ohjaukseen tarkoitettu SINA_SPEED-lohko.

3.6.3 Case-rakenne

Kuljettimien automaattisen toiminnan ohjelmointitavaksi valikoitui case-rakenne. Valinta tehtiin aikaisemmin tehtyjen vuokaavioesitysten pohjalta. Niitä tehdessä huomattiin, että kuljettimen toimintaa ohjaa erilaiset skenaariot ja case-rakenteella näitä skenaarioita pystytään hallita ja ohjelmaa suorittaa oikeassa järjestyksessä.

Case-rakenteella voidaan toteuttaa valitsin ohjelmaan, jonka perusteella suoritetaan tiettyä ohjelmaosuutta perustuen integer-datatyyppiä olevan valitsinmuuttujan arvoon. Mikäli valitsinmuuttujan arvolle löytyy arvolle määritelty vastine case-rakenteessa, suoritetaan vastineen ohjelmaa. Jos valitsinmuuttujalle ei löydy vastinetta, ei suoriteta mitään case-rakenteesta, paitsi jos rakenteelle on määritetty ELSE-lause, jolloin sen alla olevaa ohjelmaa suoritetaan. (Siemens, c).

```
CASE #Ohjelmanumero OF

    1:
        //Tätä ohjelmaosuutta suoritetaan jos ohjelmanumero = 1
        #Lähtö := true;
        IF #Tulo1 = true THEN
            #Ohjelmanumero := 2;
        END_IF;
    2:
        //Tätä ohjelmaosuutta suoritetaan jos ohjelmanumero = 2
        IF #Tulo2 = true THEN
            #Ohjelmanumero := 3;
        END_IF;
    3:
        //Tätä ohjelmaosuutta suoritetaan jos ohjelmanumero = 3
        IF #Tulo2 = true AND #Tulo1=true THEN
            #Lähtö := false;
            #Ohjelmanumero := 0;
        END_IF;
END_CASE;
```

KUVA 16. Esimerkki case-rakenteesta.

3.6.4 Toiminnallinen ohjelmointi

Kuljettimien koko automaattinen toiminnanohjaus pohjattiin case-rakenteella tehtyyn sekvenssiin, jossa ohjelma kiertää hallitusti alusta loppuun ja uudelleen. Sekvenssi toteutettiin vuokaavioesitysten pohjalta, jotka löytyvät liitteistä 1-5. Sekvenssin eri tiloihin sisällytettiin vain ehtoja seuraavaan sekvenssiin siirtymisestä, sekä virhetilojen havainnoimista.

Case-rakenteella tehty sekvenssiohjaus on yksinkertainen hallittava automattimoodin osalta. Sekvenssiä haluttiin ajaa vain solun ollessa automaatilla, joten pystyttiin lisäämään ehto, että vain automaattitilassa suoritetaan case-lausetta. Näin tehtynä solun tilan muutosten vaikutus automaatioon on suoraviivainen. Lisäksi case-rakenteella tehtyyn sekvenssiin on helppo lisätä ehtoja sekvenssin vaiheiden välillä siirtymisessä, joka mahdollistaa sekvenssin pysäyttämisen haluttuun sekvenssin vaiheeseen esimerkiksi kuljettimen ulkopuolista toimintaa varten, kuten kontin täyttämisen tavaralla.

```
1  //Sequence
2  IF #Mode.Auto THEN
3      CASE #nSeq OF
4              //Init variables
5          #Init:
6              #sStatus := 'Init';
7              #nSeq := #WaitForPart;
8
9              //Wait For Part
10         #WaitForPart:
11             //Statustietoa sekvenssin vaiheesta
12             #sStatus := 'Waiting for part';
13
14             //Ehto seuraavaan sekvenssiin siirtymiseksi
15             IF #bPartOnConveyor AND #DelayAfterArrival.Q THEN
16                 //Seuraavaan sekvenssin vaiheeseen siirtyminen
17                     #nSeq := #PartAtConv;
18                 END_IF;
19
```

KUVA 17. Case-lauseen alku vastaanottavalla kuljettimella.

Sekvenssin sisällä pyrittiin siihen, ettei siellä tarvitse kirjoittaa yhtäkään lähtömuuttujaa. Case-rakenteella tehdyssä sekvenssissä lähtömuuttujat, joita haluttiin kirjoittaa, kirjoitettiin käyttämällä sekvenssinumeroa hyödyksi. Esimerkiksi, sekvenssin vaiheessa, jossa vastaanotetaan lavaa, ohjataan kuljetinta päälle ehdon kautta, että jos sekvenssi on sekvenssinumeron perusteella vaiheessa ”vastaanotto”, niin kuljetinta ohjaava lähtö on päällä. Näin tehdessä muuttujaa ei tarvitse erikseen ohjata pois päältä, vaan kun sekvenssin tila muuttuu, lähtö menee itsestään pois päältä ehdon ansiosta.

```
1  //Run conveyor
2  IF #Mode.Auto AND #nError = 0 AND #i_bEnable THEN
3      #bRunForward :=  #nSeq=#SendPartToNextConv;
4      #bRunBackward := FALSE;
5  ELSE
6      #bRunForward := False;
7      #bRunBackward := False;
8  END_IF;
```

KUVA 18. Vastaanottokuljettimen ohjauksen käskyt automaattitilassa.

Sekvenssien vaiheiden välillä siirtymisten ehtoina toimi pääasiallisesti anturitiedot, sekä tiedot muilta kuljettimilta. Anturitietoja varten oli tarpeellista lisätä pieniä aikaviiveitä anturitiedolle, jottei virheellisen anturitiedon perusteella edetä sekvenssissä. Esimerkiksi vastaanottokuljettimella lisättiin 200ms viive anturitietoon, että voidaan luottaa, että lava todella on kuljettimella, eikä anturia ole vahingossa

esimerkiksi ohikulkijan toimesta vaikutettu. Tämä on tärkeää ottaa huomioon, sillä solun toimintaympäristössä liikkuu ihmisiä ja laitteita.

```
//Timers
#PartPresentOnDelay(IN := #i_bPartAtConv,
                    PT := T#200ms);
```

KUVA 19. Vastaanottokuljettimen lavatiedon viive.

3.6.5 Datansiirto

Kuljetinjärjestelmän yhtenä vaatimuksena asiakkaan puolesta oli, että kuljetinjärjestelmään tuotujen konttien sijaintia järjestelmässä voidaan seurata niiden yksilöllisten tunnusten avulla. Ohjelmalohkoihin tarvitsi siis sisällyttää mahdollisuus datan vastaanottamiselle ja datan siirrolle eteenpäin järjestelmässä.

Datansiirto päätettiin tehdä niin, että syötetty data pidetään ohjelmalohkon muistissa ja se siirretään eteenpäin seuraavalle kuljettimelle, seuraavaan lavapaikkaan kontin siirtyessä järjestelmässä. Datansiirto seuraavalle kuljettimelle tehtiin kättelyn avulla, jolla varmistettiin datan siirtyminen seuraavalle kuljettimelle ja sen muistipaikkaan. Kättely toteutettiin sopivissa sekvenssin vaiheissa, joissa tiedettiin kontin siirtyvän.

Seuranta HMI:lle toteutettiin kirjoittamalla kontin yksilöivää tunnustietoa string-tyyppiseen lähtömuuttujaan, josta se saatiin vietyä HMI näytölle.

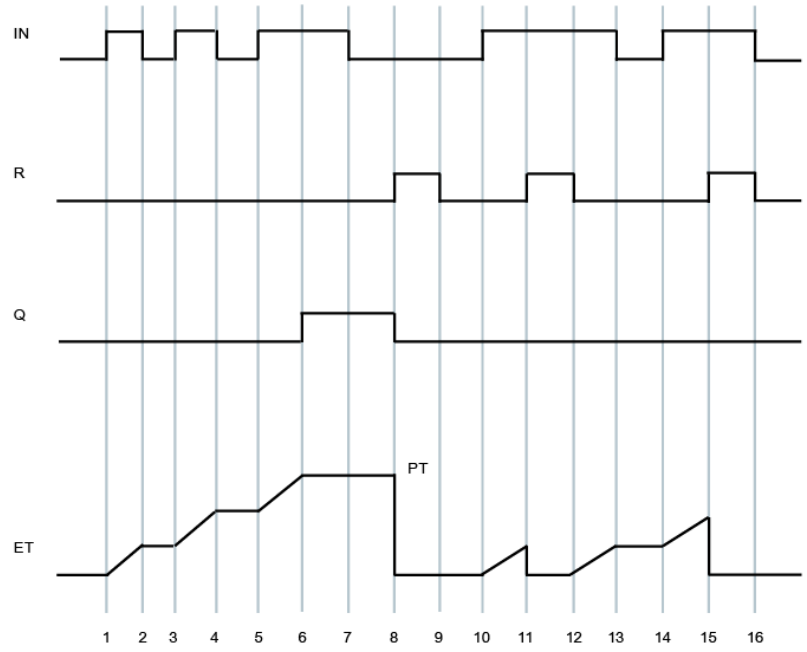
3.6.6 Virheiden käsittely

Sekvenssien eri vaiheissa, voitiin tarkastella kuljettimen tilaa antureiden ja ajastimien avulla. Näistä voitiin tehdä päättelyä siitä, onko kuljetin sekvenssin vaiheen mukaisessa tilassa. Esimerkkinä jos anturi, jolla kontin läsnäoloa mitataan ei ole vaikuttunut ja kuljettimen sekvenssi on vaiheessa ”kontti kuljettimella”, tiedetään sekvenssin olevan virheellisessä tilassa tai anturi saattaa olla hajonnut.

Ajastimia voitiin hyödyntää sekvenssin vaiheissa, joissa kontteja siirretään. Sekvenssin siirtyessä siirtovaiheeseen, voidaan käynnistää ajastin, johon säädetään

maksimiaika, joka sekvenssin vaiheen suorittamiseen saa kulua. Mikäli ajastimen aika kuluu loppuun, aktivoidaan virhe ja pysäytetään sekvenssin suoritus, jotta operaattori voi käydä tarkistamassa, mikä virheen on aiheuttanut.

Siirto-operaatioiden valvontaan käytettiin TONR-tyyppisiä ajastimia, jotka ovat toiminnaltaan samanlaisia kuin TON-ajastimet. TON ja TONR ovat ohjelmallisesti toteutettuja päästöhidastuksia. TONR-ajastin eroaa TON-ajastimesta siten, että se muistaa kuluneen ajan, kunnes ajastin nollataan.



KUVA 20. TONR-ajastimen toimintaperiaate. (Siemens)

Tulona ajastimille käytettiin ehtoa, että seuraavat signaalit ovat tosia: haluttu sekvenssin vaihe, solun tila automaatilla ja kuljetin pyörii. Aikaa otetaan näiden signaalien ollessa tosia ja näin tekemällä vältetään virheellisesti aiheutetut virhetilat.

Ajastimen nollaus pitää tehdä ennen kuin ajanottoa tarvitaan uudelleen, sillä TONR-ajastin muuten aloittaa ajanoton edellisen ajanoton jatkeeksi. Nollaus tehdään ajastimelle aina sekvenssin käynnistyessä alusta, sekä virheenkuittauksessa.

```
#Err_TransferTimeError(IN := (#nSeq = #Receive) AND #Mode.Auto AND #bRunForward,
                       PT := #i_tTransferTimeError_Out,
                       R := #nSeq = #Init  OR (#i_bResetError AND #nError.%X0));
```

KUVA 21. TONR-ajastimella toteutettu valvonta kuljettimen vastaanottovaiheelle.

3.6.7 Ohjelman testaus simulaattorilla

Ohjelmoinnin yhteydessä kuljetinlohkoja testattiin käyttämällä TIA Portalin kanssa toimivaa emulaattoria PLCSIM, joka simuloi logiikkaohjainta. PLCSIM:n avulla ohjelmaa voitiin koekäyttää tietokoneella ja simuloida erilaisia tilanteita luomalla kuljettimille testisignaaleja, joita sitten todellisuudessa saadaan, kun laitteisto on käytössä.

Tämän tyyppinen simulointi mahdollistaa toimintalogiikan testaamisen etukäteen, jolloin suurimmat virheet voidaan etukäteen korjata. Täysin oikeankaltaista simulaatiota kuljettimille on vaikea tässä testiympäristössä kuitenkaan tehdä, sillä kuljettimen toimintaa ohjaavia signaaleja voi olla useita ja testaajan pitää pystyä ohjaamaan signaaleja, sekä pysyä itse kärryillä ohjelmassa tapahtuvista päättelyistä.

Yksittäisen kuljetinlohkon toiminnan simuloiminen oli yksinkertaista, sillä sen toimintaa ohjaavia signaaleja pystyttiin hallitsemaan hyvin. Simuloimisen haasteiksi muodostui useamman kuljettimen samanaikainen toiminta, sillä lohkojen toiminta suunniteltiin niin, että kuljettimet viestivät keskenään ja niiden toiminta riippuu toisistaan. Simuloidessa useampaa kuljetinta samanaikaisesti ohjelman toiminnan ymmärtämisestä muodostui haaste, kun montaa asiaa tapahtui yhtä aikaa.

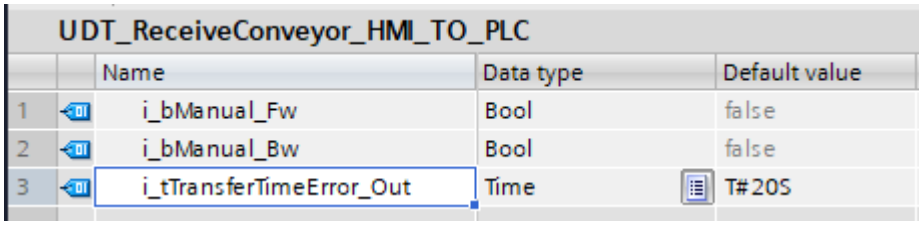
3.6.8 Koodin dokumentointi

Ohjelmakoodia pyrittiin tekemään niin, että muuttujille annetut nimet ovat hyvin kuvaavia, jotta kommentoinnin tarve pienentyisi. Kommentteja laadittiin muuttujille, sekä ohjelmakoodin sekaan, jotta ohjelmakoodia ymmärtää ohjelmoija itse jälkikäteen ja myös muut ohjelmaan perehtyvät henkilöt. Kommentointi ja ohjelman ymmärrettävyys on tärkeä asia yleisesti ohjelmoinnissa ja eritoten tämän projektin ohjelmalohkoissa, sillä ohjelmalohkoja on tarkoitus pystyä kenen tahansa tulevaisuudessa hyödyntämään. Koodin dokumentaationa toimi myös tehdyt vuokaavioesitykset lohkojen toiminnasta.

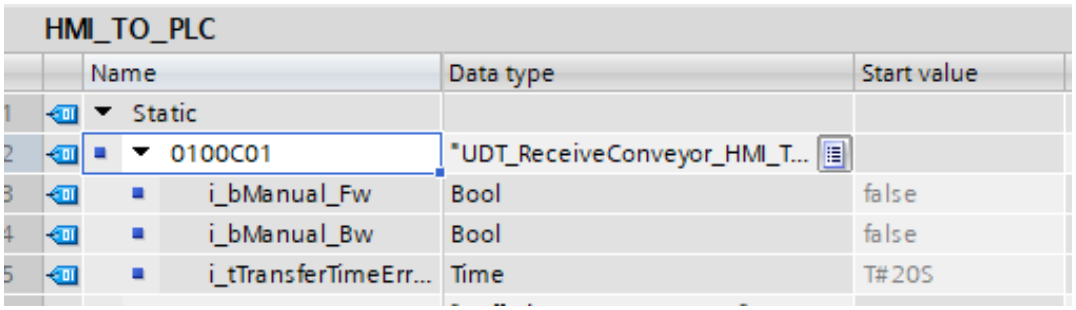
3.7 Käyttöliittymäyhteys kuljetinlohkoista

Kuljetinlohkoista haluttiin saada tietoa välitettyä käyttöliittymälle ja käyttöliittymältä haluttiin välittää tietoa kuljetinlohkoille. Tätä kommunikaatiota varten oli aiemmin suunniteltu jo tapa, jossa käytetään kahta tietokantaa, jolla tämä viestintä toteutetaan.

Tiedon välittämiseksi lohkoille ja lohkoilta käytettiin tietokantoja ”HMI_TO_PLC” ja ”PLC_TO_HMI” rajapintana. Näihin rajapintoihin suunniteltiin UDT-datatyypit jokaiselle kuljetintyypille, jotta signaalien lisääminen rajapintaan tapahtuu nopeammin, eikä jokaista signaalia tarvitse yksitellen lisätä.

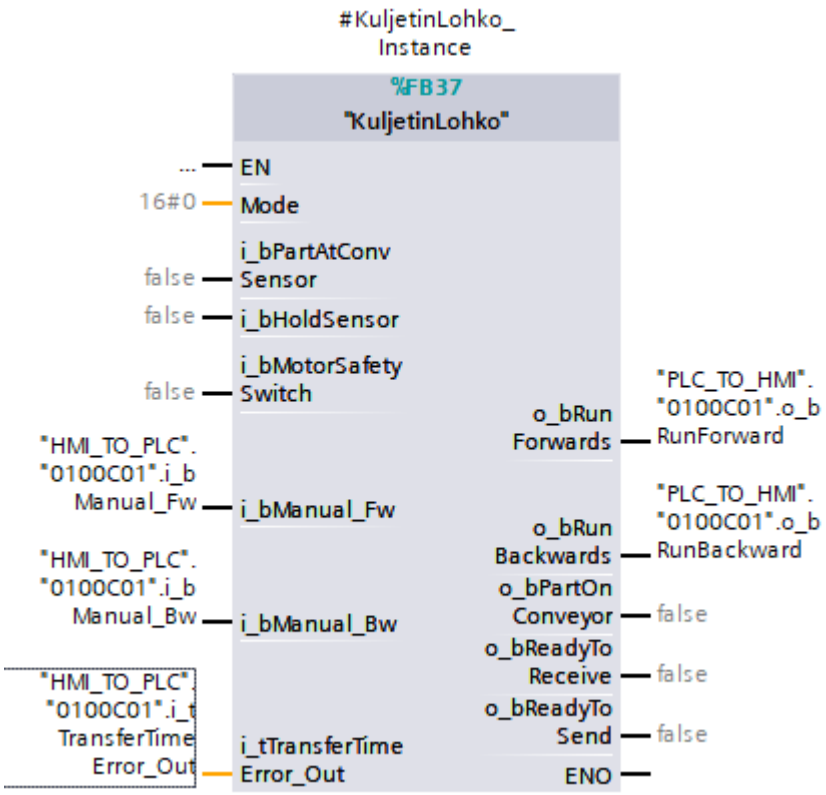


KUVA 22. HMI:n signaalien välitystä varten tehty datatyyppi.



KUVA 23. HMI:ltä PLC:lle välitettävät signaalit lisättynä tietokantaan.

Kumpaankin tietokantaan oli tarpeellista tehdä oma datatyyppi eri kuljetintyypille, HMI:lle välitettävät signaalit ja päinvastoin PLC:lle välitettävät signaalit. Datatyyppien rakentaminen käyttöliittymäyhteyteen tehtiin miettimällä mitä eri signaaleja suuntaan ja toiseen tarvitsee välittää ja ohjelman kehittyessä lisättiin signaaleja tarpeen mukaan.



KUVA 24. HMI:ltä saatavien signaalien ja HMI:lle vietävien signaalien liitos ohjelmalohkoon.

4 KÄYTTÖÖNOTTO JA TESTAUS

Käyttöönoton ja testauksen aikana solun laitteet ja toiminta testattiin toimivan oikein, siten kuten projektin määrittelyssä oli sovittu. Tämän vaiheen aikana ohjelmistoa päästiin koeponnistamaan ensimmäisen kerran oikealla laitteistolla ja ohjelmiston virheitä päästiin korjaamaan, sitä mukaa kun ongelmia havaittiin. Tässä vaiheessa ohjelmistoa kehitettiin ja hiottiin ohjelma oikeasti toimivaksi.

4.1 Laitteiden käyttöönotto

Käyttöönoton ensimmäisenä vaiheena tehtiin solun laitteiden käyttöönotto. Tähän vaiheeseen kuului väyläyhteyksien luominen, laitteiden hardwaren päivitys, laitteiden konfiguroiminen ja laitteiden toimivuuden testaus. Tämän vaiheen tarkoituksena oli varmistaa, että laitteet toimivat oletetulla tavalla.

Ensimmäisenä pystytettiin väyläyhteydet laitteille. Aiemmin suunnitteluvaiheessa automaatioprojektin hardware määrityksessä määritettiin osoitteet laitteistolle ja tässä vaiheessa oikealle laitteistolle annettiin nämä samat osoitteet, jotta väyläyhteys toimii.

Yhteyksien toimittua tehtiin hardwaren päivitys, jossa varmistettiin laitteiden olevan samat kuin automaatioprojektin hardwaren määrityksessä. Tämän lisäksi varmistettiin laiteohjelmistojen ajantasaisuus, jotta ne toimivat oikealla tavalla.

Näiden vaiheiden jälkeen tehtiin konfiguraatiot laitteille, joille se oli tarpeen tehdä. Näitä laitteita olivat taajuusmuuttajat ja moottorikäynnistimet. Näiden laitteiden konfiguraatioissa määritettiin mm. ohjattujen moottoreiden ottamat virrat ja ohjaustapa.

4.2 Ohjelman käyttöönotto

Ohjelman käyttöönotto aloitettiin ensin varmistamalla, että tulot ja lähdöt ovat yhdistettynä oikeisiin tulo- ja lähtökortteihin ja niissä oikeisiin tuloihin ja lähtöihin. Tämän jälkeen ohjelmaa alettiin testaamaan manuaaliohjausten avulla varmistaen mm. kuljettimien pyörimissuunnat.

Laitteistoon liittyvän ohjelman testauksen jälkeen alettiin ohjelmaa koestamaan yksinkertaisilla kokeilla, joissa järjestelmään syötettiin lavoja ja seurattiin ohjelman suoriutumista, sekä korjattiin tarpeen tullen vikakohtia.

4.2.1 I/O testit

Ennen ohjelman testausta tehtiin I/O testit, joissa varmistettiin tulojen ja lähtöjen olevan samoja, mitä ohjelmassa on käytetty. Tulot ja lähdöt määritettiin ohjelmaan sähköpiirustusten mukaan, joten I/O testeillä varmistettiin, että sähköasennukset ovat menneet kuvien mukaisesti ja mahdolliset virhekytkennät saadaan korjattua.

4.2.2 Kuljetinlohkojen testaus yksittäin

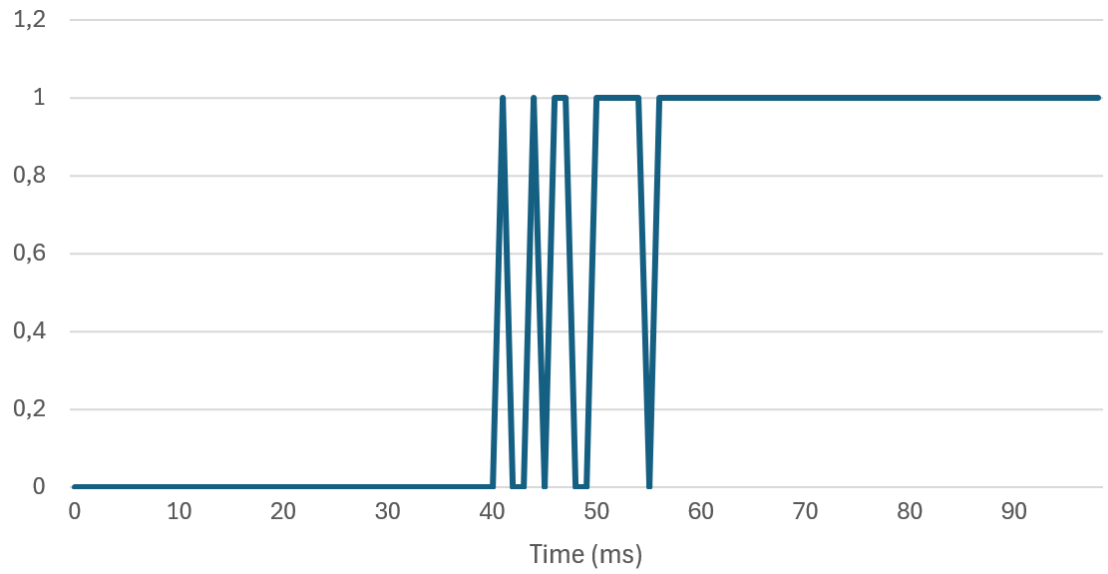
Kuljetinlohkojen ohjelman testausta tehtiin jokaiselle kuljetintyypille ensin erikseen, ilman muiden kuljettimien vaikutusta testattavan kuljettimen toimintaan. Käytännössä tämä yksittäin testaaminen tarkoitti sitä, että testattavalle kuljettimelle asetettiin eurolava haluttuun kohtaan, esimerkiksi kuljettimen alkuun, jonka jälkeen nollattiin kuljettimen sekvenssi ja käynnistettiin ohjelma. Kuljetinsekvenssit tehtiin niin, että sekvenssin nollautuessa sekvenssi hakee kuljettimen nykyisen tilan ja alkaa tehdä sen perusteella päätöksiä.

Käynnistyksen jälkeen seurattiin, että miten kuljetin jatkaa tilanteesta. Ennalta tiedettiin suunnitellun toiminnan pohjalta, että mitä missäkin tilanteessa pitäisi tapahtua. Esimerkiksi jos vastaanottavalla kuljettimella on lava ja sekvenssi nollataan, sen pitäisi hakeutua tilaan, jossa se odottaa seuraavan kuljettimen pyyntöä saada kontti.

Tässä testauksen vaiheessa pyrittiin testaamaan kaikki mahdolliset skenaariot, mitä yksittäisellä kuljettimella voi olla. Lisäksi tässä vaiheessa pystyttiin manipuloimaan signaaleja helposti. Tästä esimerkkinä aiemman esimerkin tilanne vastaanottokuljettimelta, että kun kuljetin odottaa seuraavan kuljettimen pyyntöä siirtää lavaa niin tämä signaali voitiin syöttää kuljetinlohkolle käsin, joka simuloi todellista tilannetta. Näin pystyttiin tarkistamaan reagoiko kuljetin oikealla tavalla.

Tässä vaiheessa testausta huomattiin jo ensimmäisiä ohjelmavirheitä, kun kuljettimien todellinen toiminta ei ollutkaan samanlaista, kuin mitä oli suunniteltu. Useat ohjelmointivirheet, johtuivat virheellisistä ehdoista sekvenssin sisällä, joka esti seuraavaan sekvenssin vaiheeseen siirtymisen.

Sekvenssivirheiden lisäksi huomattiin, että käytetyiltä raja-antureilta saatiin usein epäpuhdas signaali anturin vaikuttuessa. Tämä johti siihen, että ohjelma saattoi saada nousevan reunan jälkeen nopeasti laskevan reunan, jonka pohjalta sekvenssi siirtyi useamman askeleen eteenpäin virheellisesti.



KUVA 25. Raja-anturin signaalin käyttäytyminen.

Tämän ongelman ratkaisemiseksi raja-antureihin lisättiin veto- ja päästöhidastusta, jotta nousevat ja laskevat reunat tunnistetaan vain kerran ja saadaan puhdas 0→1 kulkeva signaali.

4.2.3 Kuljetinlohkojen testaus yhdessä

Yksittäisten kuljetinlohkojen testauksen jälkeen alettiin kokeilemaan kuljetinten keskinäistä toimintaa. Tässä kohdassa tiedettiin jo, että yksittäin kuljettimien toiminta on suunnitellun mukaista. Kuitenkaan tässä kohtaa ei vielä tiedetty, miten kuljettimet toimivat yhdessä. Koko kuljetinjärjestelmän toimiakseen kuljetinten on kommunikoitava keskenään oikein, jotta ei synny tilanteita, joissa kuljettimet reagoivat toisen kuljettimen toimintaan sillä tavoin, että se aiheuttaa virhetilanteita.

Yksittäisten kuljetinlohkojen testauksen aikana varmistettiin kuljetinlohkojen toiminta yksilöinä, mutta tässä vaiheessa kuljetinlohkojen signaalit yhdistettiin toisiinsa ja testattiin, kuinka kuljettimet toimivat keskenään. Tämän vaiheen testaus tehtiin juoksuttamalla yhtä lavaa eteenpäin järjestelmässä. Testaus aloitettiin niin, että kuljetin kerrallaan yhdistettiin järjestelmään lisää kuljettimia sitä mukaa kun edelliset kuljettimet toimivat keskenään ja onnistuivat siirtämään lavaa eteenpäin ilman ongelmia. Näin tehden päästiin kuljetin kuljettimelta eteenpäin järjestelmässä ja lopulta saatiin lava kulkemaan koko systeemin läpi.

Tässä testauksen vaiheessa kuljetinlohkojen kommunikaatiosta löytyi vielä virheitä, joita korjattiin sitä mukaa, kun niitä löytyi. Virheet liittyivät pitkälti virheellisesti lähetettyihin vastaanotto- ja lähetyssignaaleihin.

Näiden virhesignaalien lisäksi oli tarpeen lisätä ohjelmaan tiettyihin ohjelman kohtiin viiveitä, jotta kuljettimet kuljettavat lavaa riittävän pitkälle seuraavalle kuljettimille. Ilman näitä viiveitä kuljetettu lava saattoi jäädä hiukan jumiin edelliselle kuljettimelle.

4.3 Järjestelmän testaus

Onnistuneesti käyttöönotettujen kuljetinlohkojen jälkeen alettiin testaamaan järjestelmän toimivuutta perusteellisesti. Tässä kohtaa voitiin sanoa järjestelmän toimivan periaatteessa, mutta se täytyi vielä todentaa useiden testien kautta. Testausta tehtiin erilaisin testein käyttämällä järjestelmässä eurolavoja. Testaukseen kuului mm. yhden lavan juoksuttaminen järjestelmän läpi ja useiden lavojen samanaikainen juoksuttaminen järjestelmässä. Lisäksi testattiin järjestelmän kykyä huomata virhetilanteita ja virhetilanteista toipumisia. Testausta tehtiin itse, kunnes järjestelmä saatiin toimimaan, jonka jälkeen projektin asiakas totesi tehdastestauksessa järjestelmän toimivan.

4.3.1 Lavojen läpijuoksutus

Onnistuneen lavan läpikuljettamisen jälkeen alettiin testaamaan järjestelmää ajamalla järjestelmän läpi ensin yhtä lavaa useita kertoja, ja sittemmin useampia

lavoja samanaikaisesti, jotta voitiin löytää ohjelmasta vielä olemassa olevat virheet.

Yhden lavan läpijuoksutusta tehtäessä järjestelmässä havaittiin muutamia pieniä virheitä. Yksi näistä löydetyistä virheistä oli, että kääntöpöytä alkoi kääntyä liian aikaisin takaisin vastaanottavaan asemaan luovuttaessaan lavaa seuraavalle kuljettimelle ja lava puristui kääntöpöydän ja kuljettimen väliin. Käytännössä lava oli jo poistunut kääntöpöydältä, joten sen sekvenssi hyppäsi jo palaamaan takaisin vastaanottoasemaan. Tämä virhe korjattiin muuttamalla kääntöpöydän sekvenssiä niin, että seuraavan kuljettimen tilasta luettiin, milloin lava on vastaanotettu, ennen kuin annettiin kääntöpöydän sekvenssin kääntää pöytää.

Testauksen edetessä niin pitkälle, että yhden lavan sai läpi järjestelmästä joka kerralla, alettiin järjestelmää koeponnistamaan ajamalla useita lavoja järjestelmään samanaikaisesti. Tässä vaiheessa järjestelmän toiminta alkoi muistuttaa todellista tilannetta solun tulevassa toimintaympäristössä, joten tämä testausvaihe ennustaa jo hyvin järjestelmän todellista toimivuutta. Lavoja syötettiin käsin järjestelmään alkuun aina heti seuraavan perään ja myöhemmin lavoja syötettiin järjestelmään epäsäännöllisemmin, jotta järjestelmässä toteutuvat tilanteet vaihtelevat.

Useamman lavan ajamisessa järjestelmän läpi havaittiin jälleen ohjelmasta virheitä, joita korjattiin sitä mukaa kun niitä ilmeni. Eniten haasteita aiheutti kuljetinlohkot, jotka ohjasivat bufferoivia kuljettimia, joihin mahtuu kaksi lavaa kerrallaan. Tämä kuljetinlohko oli kaikista kuljetintyypeistä monimutkaisin, joten se ei ollut yllättävää.

Haasteita tämän kuljettimen ohjauksessa olivat erilaiset tilanteet, joita syntyi, kun kuljettimen piti samanaikaisesti lähettää lavaa eteenpäin seuraavalle ja vastaanottaa edelliseltä. Näitä haastavia tilanteita olivat, kun edellä mainittu tilanne tapahtui bufferoivan kuljettimen molempien lavapaikkojen ollessa täynnä, ensimmäisen lavapaikan ollessa täytettynä ja toisen lavapaikan ollessa täytettynä. Virhetilanteet aiheuttivat lavojen siirtymiseen liittyviä ongelmia, kuten esimerkiksi joissain tilanteissa lavan vastaanotto saattoi keskeytyä, sen takia, että lava oli onnistuneesti lähetetty seuraavalle. Pienillä ohjelmamuutoksilla nämäkin ongelmat saatiin selvitettyä ja järjestelmä toimimaan.

4.3.2 Erilaisten skenaarioiden luominen

Useita lavoja järjestelmään syötettäessä havaittiin tiettyjä haasteellisempia tilanteita järjestelmälle ja myös hyvin harvoin toteutuvia tilanteita. Näiden haasteellisten tilanteiden testaamiseksi järjesteltiin erilaisia skenaarioita järjestelmään asettamalla lavat halutuille kuljettimille haluttuihin kohtiin, jonka jälkeen sekvenssit nollattiin ja järjestelmä käynnistettiin. Näin tehdessä voitiin hallitusti toistaa tiettyjä tilanteita, joita järjestelmässä tulee vain harvakseltaan.

Erityisesti tällä tavalla haluttiin testata tilanteita, joissa kuljetin lähettää ja vastaanottaa samanaikaisesti, sillä se havaittiin useassa kohdassa haasteellisimmaksi osaksi ohjelman toimintaa. Tällä tavoin tehtynä pystyttiin myös paremmin seuraamaan tällaisten tilanteiden etenemistä.

Tässä testauksen vaiheessa huomattiin vielä, että vaikka lohkot toimittivat lavoja oikein eteenpäin, niin tiettyjen ehtojen toteutuessa yksittäisen kuljetinlohkon sekvenssi saattoi lukea itsensä väärään tilaan. Esimerkiksi bufferoivat kuljettimet saattoivat lukea oman tilansa liian aikaisin vastaanoton ja lähetyksen aikana, joka aiheutti sen menevän tilaan, jossa se voi vain vastaanottaa. Tämä ei järjestelmän toiminnan kannalta aiheuttanut suurta harmia, mutta se oli silti virhe, joten se korjattiin ohjelmassa lisäämällä ehtoja sekvenssin vaiheista siirtymisiin.

4.3.3 Virhetilanteiden testaus

Viimeisenä testauksen vaiheena toimivan kuljetinjärjestelmän virheenkäsittelyä testattiin aiheuttamalla tahallisesti erilaisia virheitä ja tarkistamalla, että aiheutetut virheet tunnistetaan. Samalla testattiin, miten virhetilanteista toipuminen tapahtuu niin, että järjestelmä voi jatkaa automaattista toimintaansa normaalisti.

Virhetilanteita aiheutettiin fyysisesti jumittamalla lavat paikoilleen siirtovaiheissa niin pitkäksi aikaa, että virheen tapahtumista mittaava ajastin menee päälle ja saadaan virheilmoitus. Jatkotoimenpiteenä virhe nollattiin ja järjestelmä käynnistettiin ja varmistettiin, että järjestelmä edelleen toimii oletetulla tavalla. Virhetilanteet käytiin läpi aiemmin suunnittelussa ennakoitujen tilanteiden pohjalta. Mikäli virheilmoitusta ei saatu, korjattiin ohjelmaa niin, että virhe voitiin havaita.

Lavojen jumittamisen lisäksi testattiin laittaa lavoja peräkkäin järjestelmään niin, ettei niiden välissä ole tyhjää tilaa. Järjestelmän toimivuus on kiinni anturitiedoista, jotka vaativat pienen tilan lavojen väliin, jotta tunnistus uuden lavan saapumisesta voidaan saada. Mikäli tätä väliä ei ole tiedetään ajan perusteella, että vakionopeuksista kuljetinta on ajettu liian pitkään ja saadaan virheilmoitus. Tätä ominaisuutta testattiin eri tilanteissa ja todettiin se toimivaksi.

5 PROJEKTIN JÄLKEINEN JATKOKEHITYS

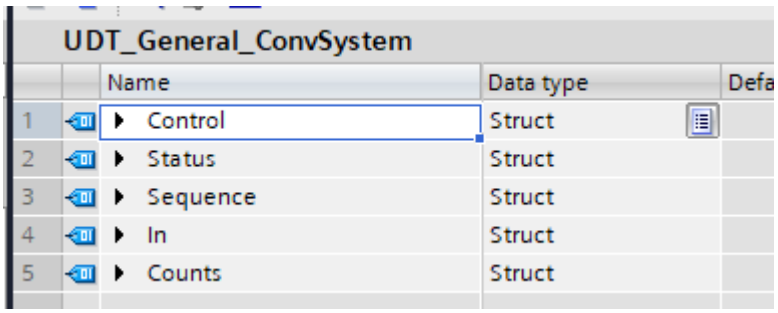
Projektin aikana kuljetinlohkoista muodostui toimiva kokonaisuus, mutta ohjelmassa käytetyn ohjelmointitavan takia ohjelmalohkoista muodostui ohjelmointiympäristössä visuaalisesti isokokoisia ja epäselkeitä. Tämä johtui ohjelmalohkoissa käytetystä rajapinnasta, jossa käytettiin tuloja ja lähtöjä, jonka takia lohkojen visuaalinen esitys ohjelmointiympäristössä kasvoi suureksi. Lohkojen kasvaminen isommiksi alkoi aiheuttamaan käytettävyyden heikkenemistä, joten lohkoille täytyi saada muutos. Projektin jälkeen huomattiin, että ohjelmalohkojen rajapinnan voisi tehdä käyttäen InOut-muuttujaan tehtyjä datatyyppejä, jolloin suurin osa tuloista ja lähdöistä saataisiin pakattua vain muutaman datatyypin sisään.

Jatkokehityksen aikana kuljetinlohkojen rajapinta rakennettiin uusiksi tekemällä yleinen datatyyppi, jota käytetään kaikissa kuljetinlohkoissa InOut-muuttuja rajapinnassa. Tämä muutos vaikutti HMI:n ja PLC:n väliseen signaalien välittämiseen, lohkojen visuaaliseen esitykseen ohjelmointiympäristössä, sekä lohkojen väliseen kommunikaatioon. Muutoksen ansiosta kuljetinlohkojen signaalit saatiin myös helpommin käytettäviksi, minne tahansa muualle ohjelmaan.

5.1 Yleisen datatyypin muodostus

Kuljetinlohkoille haluttiin kehittää yksi oma UDT yleiseksi datatyypiksi, jota käytettäisiin jokaisessa kuljetinlohkossa. Tästä datatyypistä löytyisi siis kaikki lohkon tarvitsemat tulot ja lähdöt, lukuun ottamatta uniikkeja signaaleja, jotka lohko tarvitsee erikseen, kuten anturitiedot ja tietyt aikaviiveet, joita pitää yksittäisille kuljettimille pystyä säätämään erikseen. Tämä haluttiin tehdä, jotta lohkoista saataisiin selkeämmät käyttää ja helpommat käyttöönottaa.

Kuljetinlohkojen kehittäminen selkeämpään suuntaan alkoi määrittämällä, mitkä signaalit ovat yhteisiä kaikille kuljetinlohkoille. Määrityksen jälkeen muodostettiin datatyyppi, jonka sisällä signaalit jaoteltiin eri kategorioihin niiden käyttötarkoituksen mukaan.



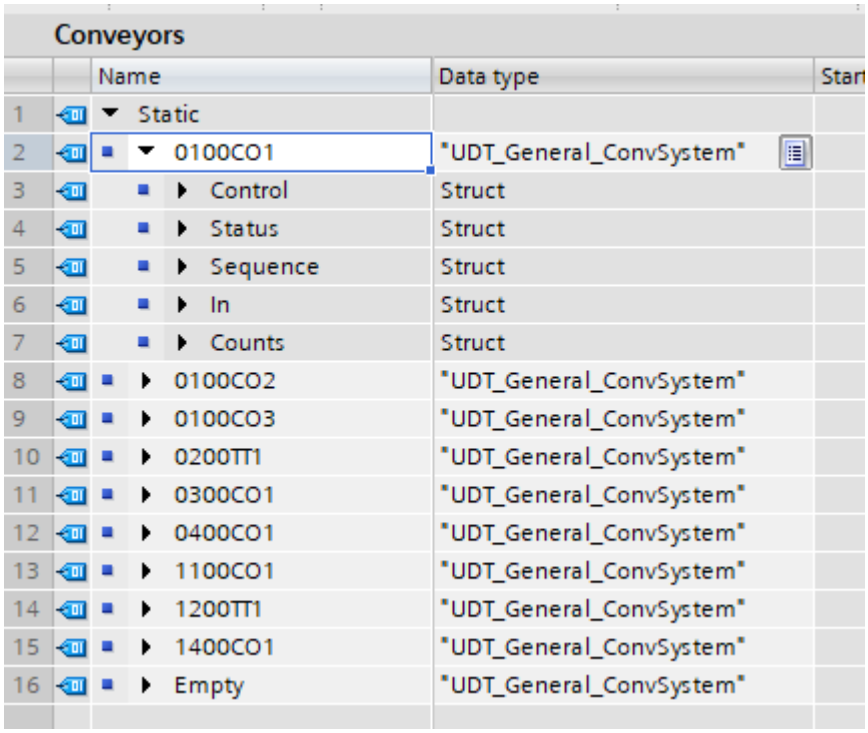
KUVA 26. Kategorisoitu yleinen datatyyppi.

Kategorioiksi muodostui viisi erilaista, jotka pitävät sisällään eri osa-alueiden signaaleja:

- Control – Kuljettimen hallintaan liittyvät signaalit, kuten solun tila, manuaaliajot ja pysäytyspyyntö
- Status – Kuljettimen tilatietoon liittyvät signaalit, kuten ”vastaanottovalmius”, ”lähetysvalmius” ja ”lähettää”
- Sequence – Kuljettimen sekvenssiin liittyvät signaalit, kuten sekvenssin vaihe ja sekvenssin kuvaukset
- In – Tänne kirjoitettiin kuljettimen anturitiedot, jotta ne saatiin datatyypin kautta käyttöön esimerkiksi HMI:lle
- Counts – Kuljettimen laskemia tietoja, kuten kuinka monta konttia kuljettimella on siirretty

5.2 InOut muuttujan käyttö ja tietokanta

Aikaisemmin tulot ja lähdöt olivat kuljetinlohkon omassa instanssissa, mutta siirryttyä käyttämään tehtyä datatyyppiä ja InOut-muuttujaa näitä tietoja varten oli tarpeen tehdä kuljetinlohkosta irrallinen tietokanta. Tässä tietokannassa määritettiin jokaista kuljetinta varten oma alueensa omille signaaleilleen käyttäen tehtyä datatyyppiä. Näin tehtynä, kuljettimien lisäys tapahtuu nopeasti ja kätevästi.



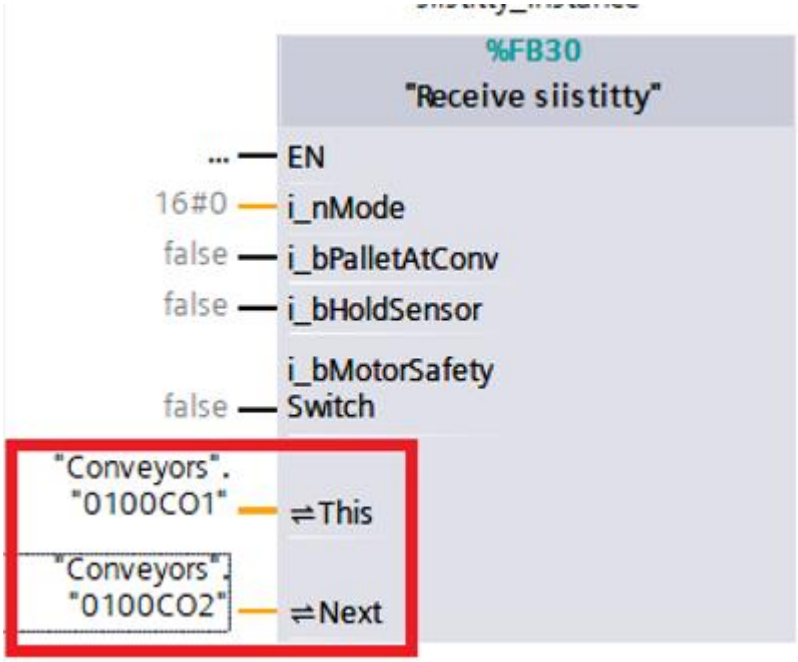
Conveyors

	Name	Data type	Start
1	Static		
2	0100CO1	"UDT_General_ConvSystem"	
3	Control	Struct	
4	Status	Struct	
5	Sequence	Struct	
6	In	Struct	
7	Counts	Struct	
8	0100CO2	"UDT_General_ConvSystem"	
9	0100CO3	"UDT_General_ConvSystem"	
10	0200TT1	"UDT_General_ConvSystem"	
11	0300CO1	"UDT_General_ConvSystem"	
12	0400CO1	"UDT_General_ConvSystem"	
13	1100CO1	"UDT_General_ConvSystem"	
14	1200TT1	"UDT_General_ConvSystem"	
15	1400CO1	"UDT_General_ConvSystem"	
16	Empty	"UDT_General_ConvSystem"	

KUVA 27. Tietokanta kuljettimien signaaleja varten.

Kuljettimelle omistettu alue tietokannasta vietiin oikealle kuljetinlohkolle, johon tehtiin rajapintaan samaa datatyyppiä oleva InOut-muuttuja, josta kuljetinlohko pääsee lukemaan ja kirjoittamaan tietokantaan. Kuljetinlohkoihin jouduttiin tekemään muutoksia rajapinnan muutoksen myötä, mutta tehdyt muutokset olivat vain samojen olemassa olevien muuttujien lukemista InOut-muuttujan kautta.

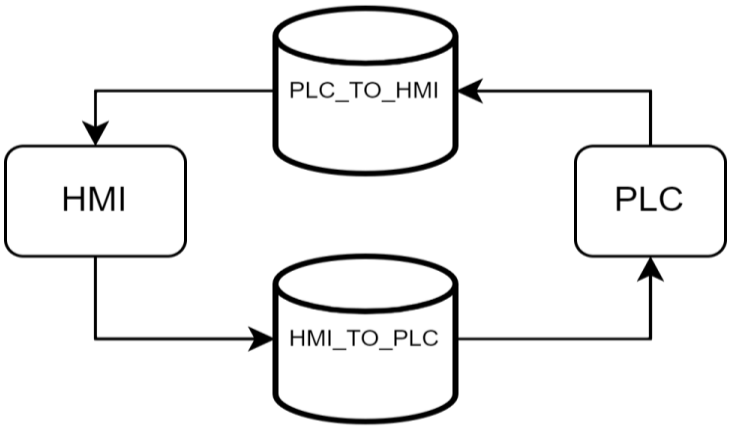
Rajapinnan muutoksen myötä myös kuljettimien välinen kommunikaatio muuttui hiukan. Aiemmin tiedot luettiin edellisen kuljetinlohkon instanssista, mutta nyt tätä kommunikaatiota varten luettiin suoraan kuljetinlohkoon vaikuttavia toisia kuljettimia suoraan niiden ohjaukseen käytettävästä tietokannasta. Tätä varten kuljetinlohkoihin lisättiin taas samaa datatyyppiä oleva InOut-muuttuja, mikä on tarkoitettu seuraavan tai edellisen kuljettimen tilan lukemiseen.



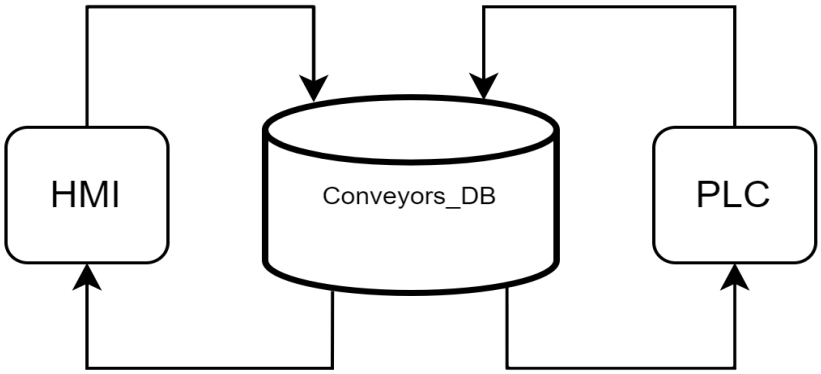
KUVA 28. Vastaanottokuljettimen rajapinta käyttäen InOut-muuttujia ja tehtyä datatyyppiä.

5.3 HMI rajapinnan muutos

Kuljetinlohkojen rajapinnan muutoksen myötä HMI:n ja PLC:n välinen kommunikaatiorajapinta muutettiin, sillä aikaisemmin käytetyt tietokannat kirjoittivat ja lukivat suoraan ohjelmalohkon tuloista ja lähdöistä, jotka käytännössä poistettiin kokonaan. Muutoksen myötä käyttöliittymälle voidaan linkittää uusi kuljettimille tehty tietokanta ja tätä pystytään käyttämään suoraan kommunikaatio rajapintana. Seuraavat kuvat esittävät rajapintamuutoksen vaikutusta signaalien välityksessä.



KUVA 29. Vanha HMI rajapinta



KUVA 30. Uusi HMI rajapinta.

Uuden ja vanhan tavan erona on, että vanhassa rajapinnassa käytettiin kahta tietokantaa, joissa vain jommallakummalla laitteella on kirjoitusoikeus. Uudella tavalla tehtäessä rajapintana on yksi ja sama tietokanta, jossa kummatkin laitteet voivat kirjoittaa ja lukea muuttujia.

Molemmissa tavoissa on omat hyvät puolensa. Vanhalla tavalla rajapinta on hyvin selkeä ja muuttujien tahattomasti päällekirjoittamisen mahdollisuus on pieni ja vianetsintä on yksinkertaisempaa. Uudessa tavassa tämänlainen päällekirjoittamisen mahdollisuus on suurempi, jos tehdään muutoksia. Uudessa tavassa merkittävä etu on kuitenkin, että rajapinta syntyy melkein kuin itsestään, kun kuljettimille tarkoitettu tietokanta luodaan.

6 POHDINTA

Tämä opinnäytetyö keskittyi vakioitujen ohjelmalohkojen kehittämiseen erityyppisiä kuljettimia varten JTA Connection Oy:n käyttöön. Työ tehtiin osana JTA Connection Oy:n asiakasprojektia, jossa kuljetinlohkoja lopulta käytettiin lavakuljetinjärjestelmän ohjaukseen. Työn tavoitteena oli luoda uudelleenkäytettäviä, toimivaksi testattuja ja helposti käyttöönotettavia ohjelmalohkoja, joilla pystytään nopeasti ohjelmoimaan toimiva lavakuljetinjärjestelmä.

Työn aikana ohjelmalohkot suunniteltiin ja ohjelmoitiin käyttäen Siemensin TIA Portal-kehitysympäristöä, ja niiden toimivuus varmistettiin käytännön testauksilla osana asiakasprojektin käyttöönottoa. Tässä työssä tarkasteltiin ohjelmiston kehitysprosessia ohjelmiston suunnittelun ajalta, ohjelman toteutuksen ajalta, sekä testausprosessin ajalta.

Ohjelmalohkoista onnistuttiin kehittämään uudelleenkäytettäviä, toimivia ja helposti käytteenotettavia. Ohjelmalohkot toimivat keskenään saumattomasti ja niitä voi pitää omana ohjelmalohkoperheenä, jolla voi toteuttaa lavakuljetinjärjestelmän ohjelmoinnin kuljettimien osalta. Työn lopputulos mahdollistaa, että vastaavia järjestelmiä voidaan toteuttaa nopeammalla aikataululla, mikä tukee JTA Connection Oy:n strategisia tavoitteita automaatioprojekteissa.

Tulevaisuuden kannalta on mielenkiintoista testata kehitettyjä lohkoja toisessa, mahdollisesti laajemmassa kuljetinjärjestelmässä, sillä vaikka lohkot saatiin toimimaan ja ne testattiin hyvin tämän opinnäytetyön projektissa, niin eri kokoonpanolla voidaan testauksessa törmätä uusiin virhetilanteisiin.

Kuljetinlohkoihin kehitettiin tässä opinnäytetyössä vain yksi ajotapa, jolla lavoja pyritään siirtämään järjestelmässä mahdollisimman nopeasti eteenpäin. Tämän yhden ajotavan lisäksi ohjelmalohkoja voisi jatkokehittää niin, että ne sisältäisivät erilaisia ajotapoja. Jatkokehitystoimena voisi myös kehittää uusia ominaisuuksia, kuten määrätyn alueen kuljettimien täyttämisen ennen alueella olevien lavojen siirtämistä eteenpäin.

LÄHTEET

Asmala H., Koskinen K., Koskela M., Mätäsniemi T., Soini A., Strömman M., Tommilla T. ja Valkonen J. 2005. Automaatiosovellusten ohjelmistokehitys. 1. painos. Helsinki: Painomerkki Oy.

Beckhoff. 2024. Indentifiers for variables and instances. Verkkosivu. Luettu 14.11.2022. https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_plc_intro/12073947403.html&id=

Bolton, W. (2015). Programmable Logic Controllers, 6th Edition. Newnes.

JTA Connection. 2024. JTA Connection. Verkkosivu. Viitattu 27.2.2024. https://www.jtaconnection.fi

Kalliola, J. (2022). Vihreä koodi. Viitattu 12.3. https://www.exove.com/fi/vihrea-koodi/

Phoenix contact. 2024. DB EN ELR H5-IES-..IFS Networkable motor starter (CONTACTRON). Dokumentaatio. Viitattu 5.3.2024. https://www.phoenixcontact.com/en-pc/products/hybrid-motor-starter-elr-h5-ies-pt-500ac-3-ifs-2905139#downloads-link-target

Robert L. Glass. 2001. Frequently Forgotten Fundamental Facts about Software Engineering. Institute of Electrical and Electronics Engineer, Inc. Viitattu 12.4.2024.

Siemens. 2024. Totally Integrated Automation Portal. Verkkosivu. Viitattu 5.3.2024. https://www.siemens.com/global/en/products/automation/industry-software/automation-software/tia-portal.html

Siemens, a. (2014). S7-1500 Cycle and response times. Viitattu 6.4.2024. https://cache.industry.siemens.com/dl/files/558/59193558/att_112303/v1/s71500_cycle_and_reaction_times_function_manual_en-US_en-US.pdf

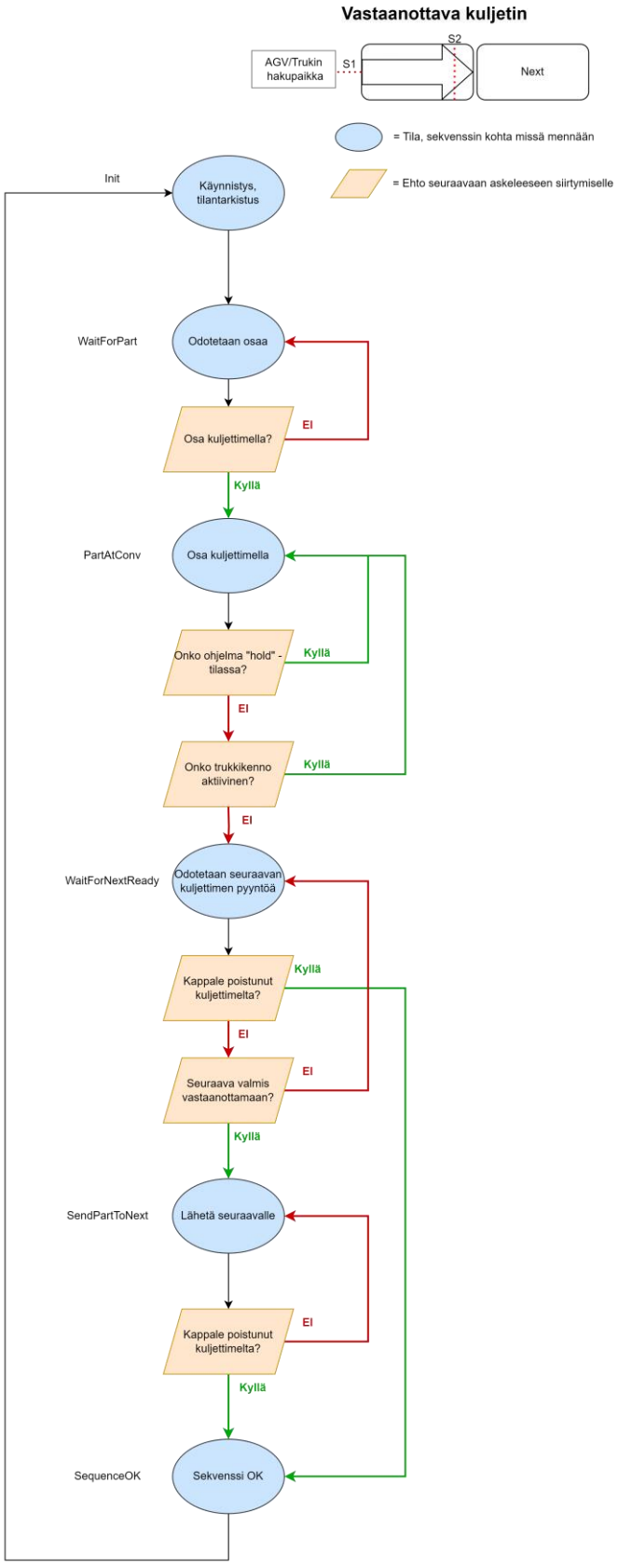
Siemens, b. (2017). SINAMICS G: Controlling a speed axis with the “SINA_SPEED” block. Dokumentaatio. Viitattu 5.3.2024. SINAMICS G: Controlling a speed axis with the “SINA_SPEED” block - ID: 109485727 - Industry Support Siemens

Siemens, c. (1998). Structured Control Language (SCL) for S7-300/S7-400 Programming. Viitattu 20.4.2024. https://cache.industry.siemens.com/dl/files/188/1137188/att_27471/v1/SCLV4_e.pdf

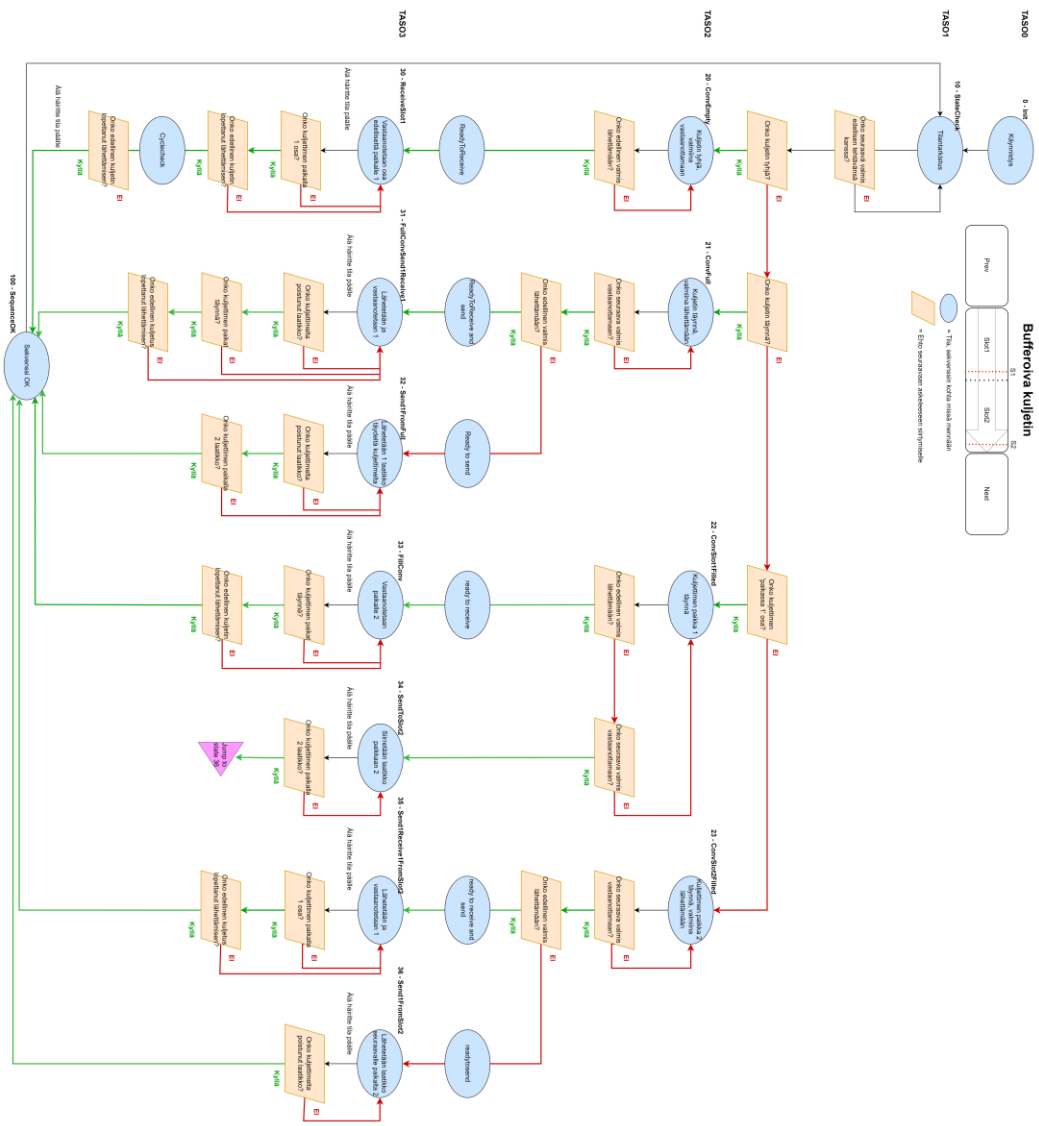
Siemens. (ei pvm). TIA Portal V18 -ohjelman sisäinen Information system.

LIITTEET

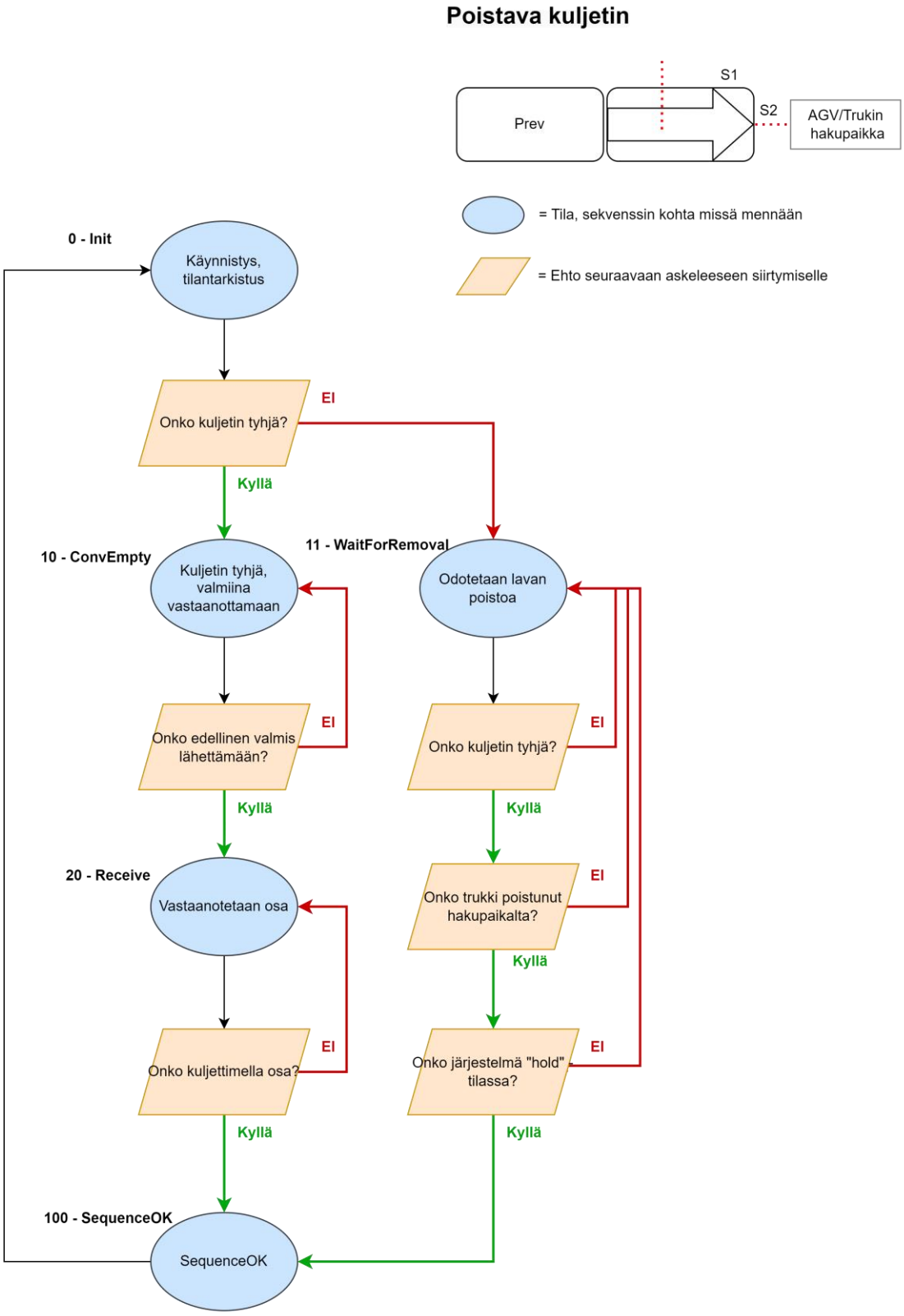
Liite 1. Vastaanottokuljettimen vuokaavio



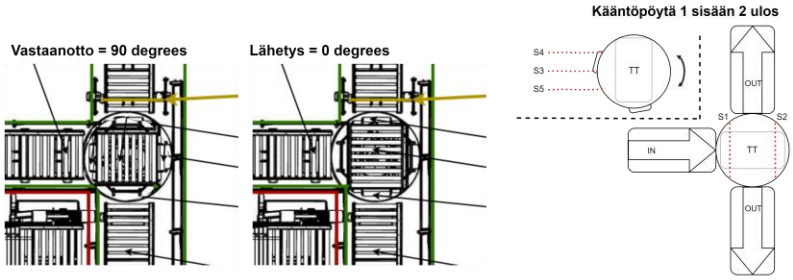
Liite 2. Bufferoivan kuljettimen vuokaavio

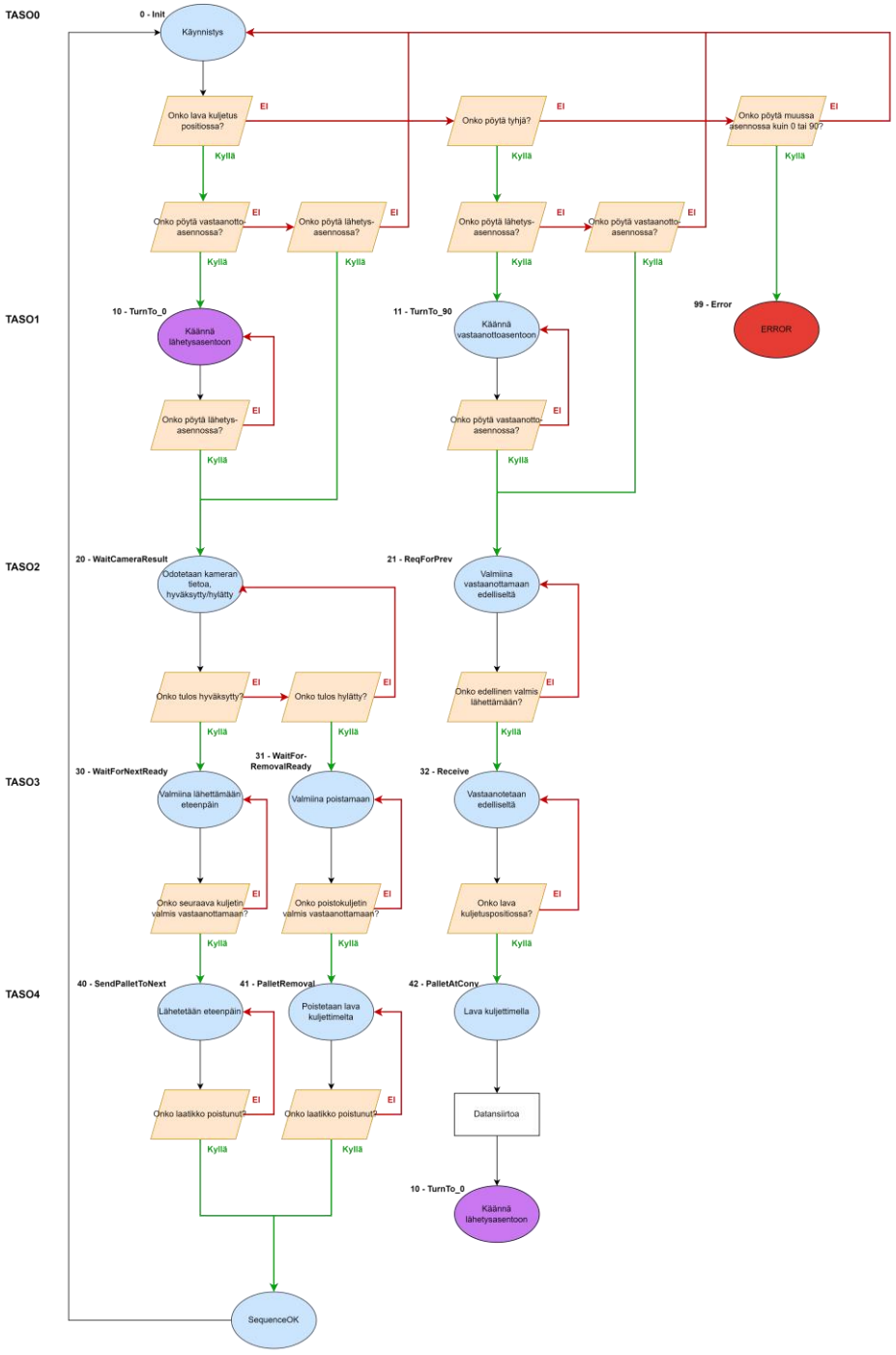


Liite 3. Poistokuljettimen vuokaavio

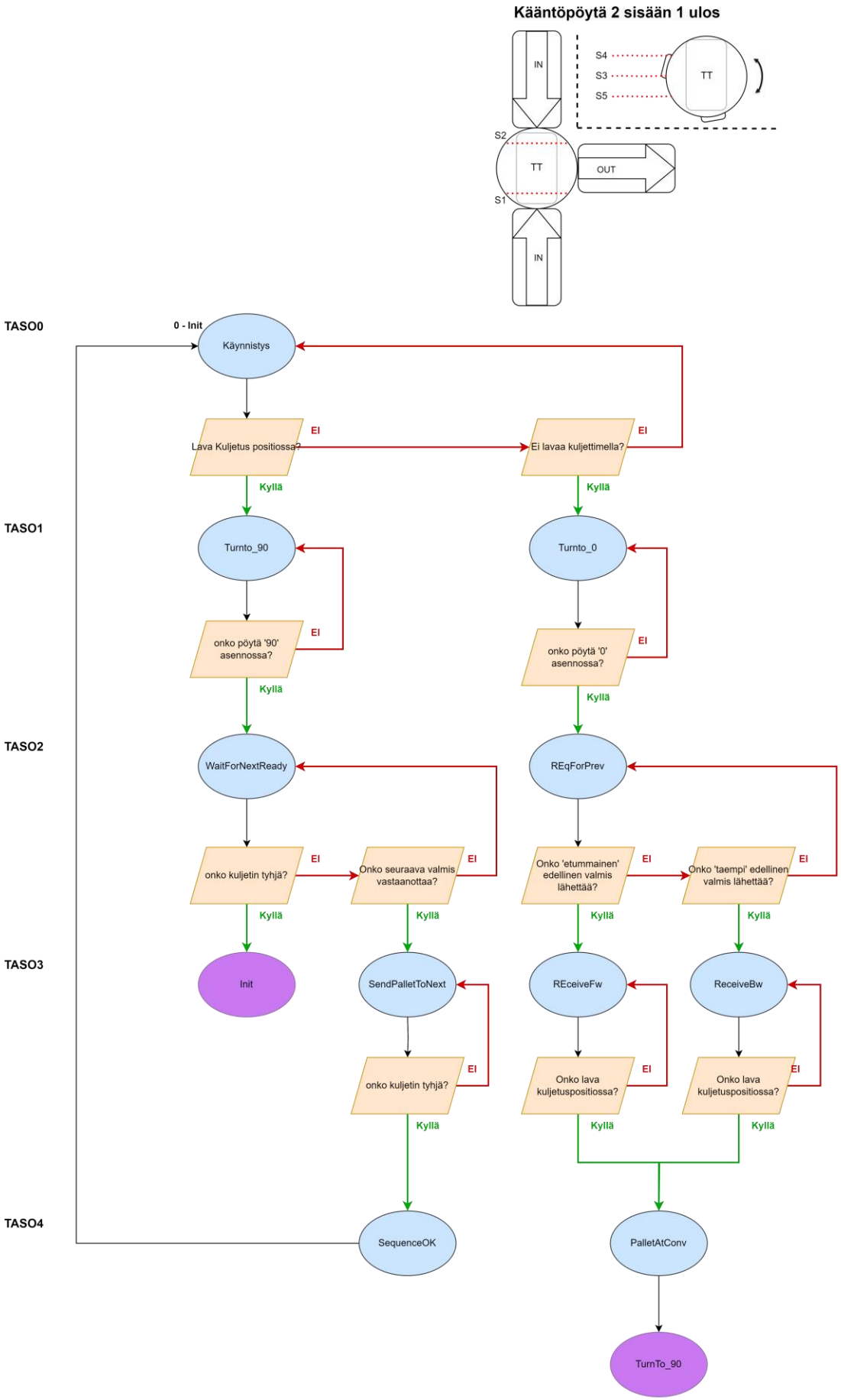


Liite 4. Kääntöpöytä ”1 sisään 2 ulos” vuokaavio

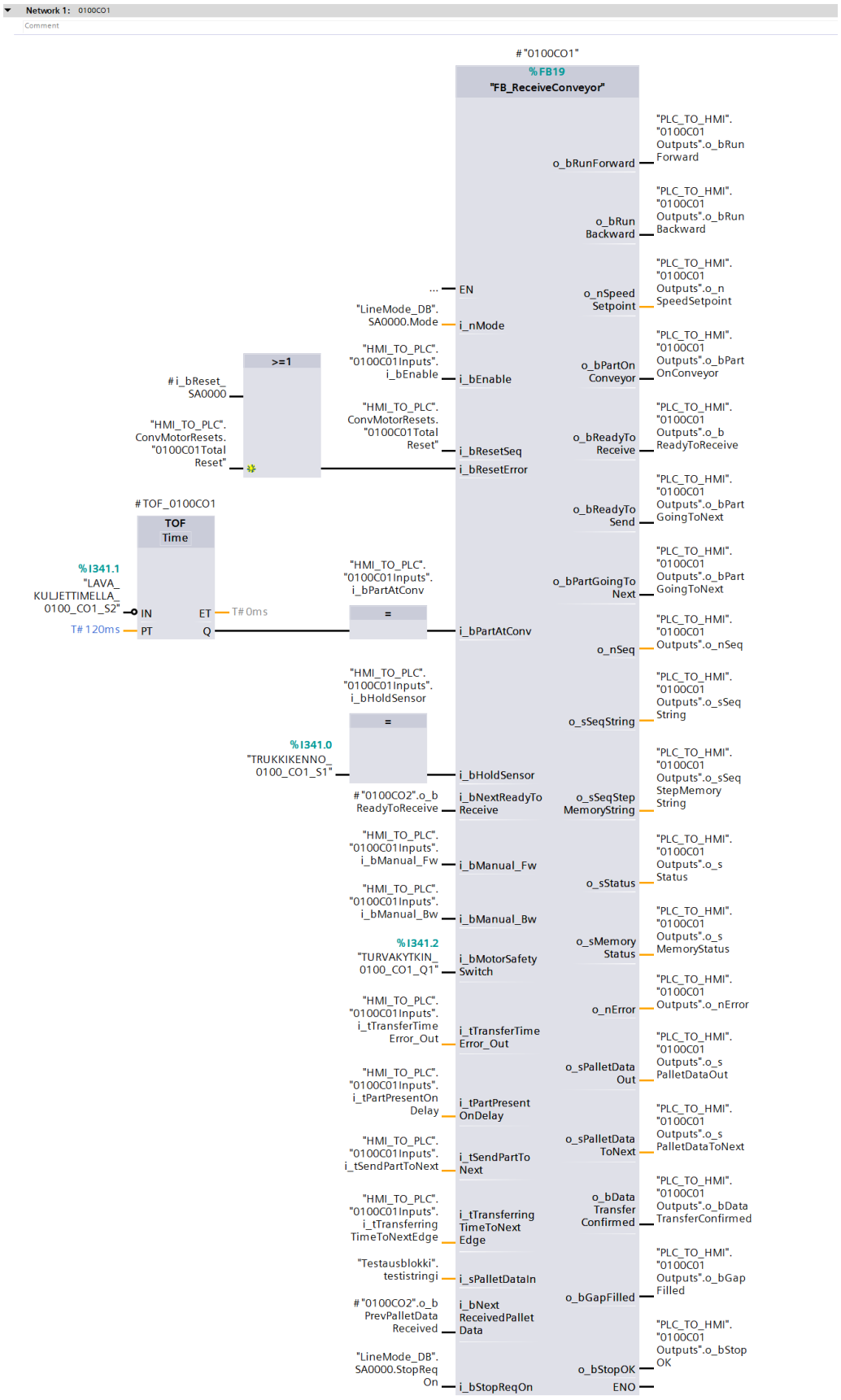




Liite 5. Kääntöpöytä ”2 sisään 1 ulos” vuokaavio



Liite 6. Projektin aikana valmistunut vastaanottokuljettimen ohjelmalohko



Liite 7. Jatkokehityksen jälkeinen vastaanottokuljettimen ohjelmalohko