LAPIN AMK
Lapland University of Applied Sciences

NOSTOLAITTEIDEN ETÄVALVONTAJÄRJESTELMÄN KÄYTTÖÖNOTON TEHOSTAMINEN

Jarkko Hooli

Opinnäytetyö
Insinööri (AMK)
Sähkö- ja automaatiotekniikka

2023

LAPIN AMK
Lapland University of Applied Sciences

Opinnäytetyön tiivistelmä

Sähkö- ja Automaatiotekniikka
Insinööri (AMK)

Tekijä	Jarkko Hooli	**Vuosi**	2023
Ohjaaja	FM Ari Afflekt		
Toimeksiantaja	Konecranes Finland Oy		
Työn nimi	Nostolaitteiden etävalvontajärjestelmän käyttöönoton tehostaminen		
Sivumäärä	33 + 5		

Tässä opinnäytetyössä toteutettiin Konecranes TruConnect -etävalvontajärjestelmän käyttöönottoprosessin tehostaminen. Opinnäytetyön toimeksiantajana toimi Konecranes Finland Oy ja kehitystyö toteutettiin työharjoittelun ohessa Konecranesin globaalissa teknisessä tuessa kesän 2022 aikana yhtiön pääkonttorilla Hyvinkäällä.

Opinnäytetyössä tutkittiin, mitä muutoksia TruConnect PLC-ohjelmaan tarvitaan, jotta etävalvontajärjestelmän käyttöönotto nopeutuu ja millaisia hyötyjä ohjelman kehittämisellä voidaan saavuttaa. Työn tavoitteena oli järjestelmän ohjelmakomponenttien yksinkertaistettu ja nopeutettu käyttöönotto.

Lähdeaineistona käytettiin PLC-ohjelmointia käsittelevää kirjallisuutta ja aiheeseen liittyviä verkkojulkaisuja. Kehitysprosessin onnistumista mitattiin vertailemalla vanhaan ja uuteen käyttöönottoprosessiin kuluvaa aikaa sekä tehtävien viittausten ja määritysten lukumäärää.

Mittaustuloksista voidaan päätellä, että kehitystyö on onnistunut ja työn tuloksena on nopeampi ja yksinkertaisempi TruConnect-käyttöönottoprosessi sekä sen päivitetty käyttöohje.

Avainsanat ohjelmoitavat logiikat, nosturit, etäseuranta

LAPIN AMK
Lapland University of Applied Sciences

Abstract of Thesis

Electrical and Automation Engineering
Bachelor of Engineering

Author Jarkko Hooli **Year** 2023
Supervisor(s) Ari Afflekt, MSc
Commissioned by Konecranes Finland Oy
Title Improving the Commissioning Procedure of a Remote Monitoring System
Number of pages 33 + 5

In this thesis, the commissioning process of the Konecranes TruConnect remote monitoring system was enhanced. The thesis was commissioned by Konecranes Finland Oy, and the development was carried out during the author's internship at Konecranes' global technical support during the summer of 2022 at company headquarters in Hyvinkää.

The thesis examined, what changes are needed in TruConnect's PLC program to speed up the implementation of the remote monitoring system, and what kind of benefits can be achieved by developing the program. The goal of the work was simplified and faster implementation of the system's program components.

Literature on PLC programming and online publications otherwise related to the topic were used as source material. The success of the development process was measured by comparing the time required for the new and old implementation process, as well as the number of references to be made.

From the measurement results, it can be concluded that the development work was successful, and the work resulted in a faster and simpler TruConnect commissioning process, as well as its updated user manual.

Keywords programmable logic controllers, cranes, remote monitoring

SISÄLLYS

KÄYTETYT MERKIT JA LYHENTEET

IPC	Industrial PC (Siemens)
DB	Data block
FB	Function block
FC	Function
RTG	Rubber Tired Gantry
STS	Ship-To-Shore

JOHDANTO

Tässä opinnäytetyössä tutkitaan ja toteutetaan Konecranesin TruConnect-etävalvontajärjestelmän PLC-ohjelman jatkokehitysmahdollisuuksia.

TruConnect-etävalvontajärjestelmä kerää käyttötietoja esimerkiksi nosturin käyttöajasta, moottorien käynnistyksistä, työsykleistä ja hätäpysäytyksistä. Etävalvontajärjestelmä valvoo myös jarrujen ja inverttereiden käyttöä. Tiedot siirtyvät verkkopalvelimelle, jossa ne kootaan ja siirretään yourkonecranes.com-asiakasportaaliin.

Nykytilanteessa järjestelmän käyttöönotossa suoritetaan päällekkäisiä toimenpiteitä, joten aiheen tutkiminen ajansäästön ja yksinkertaistamisen varjolla on perusteltua sekä asiakkaan että toteuttavan yrityksen kannalta.

Työ rajataan järjestelmän PLC-ohjelman kehitykseen, tarkemmin koneistolaskureiden esikonfigurointiin, sillä IPC:llä tapahtuva datankäsittely menee alan ja osaamisen ulkopuolelle.

1 KONECRANES

Konecranes on yksi maailman johtavia nostolaitevalmistajia, jonka tuotteisiin kuuluvat teollisuusnosturit ja niiden osat, työpistenosturit, kontinkäsittelylaitteet ja kunnossapitopalvelut. Yhtiön pääkonttori sijaitsee Hyvinkään kaupungissa Etelä-Suomessa ja vuonna 2021 Konecranes-konsernin liikevaihto oli yhteensä 3,2 miljardia euroa. Konsernilla on noin 16 600 työntekijää noin 50 maassa. Yrityksen liiketoiminta on jaettu kolmeen segmenttiin – Kunnossapitoon, Teollisuuslaitteisiin ja Satamaratkaisuihin (Konecranes 2022a.)

Konecranesin historia ulottuu vuoteen 1908, jolloin Helsingissä perustettiin konepaja nimeltä Tarmo, josta kaksi vuotta myöhemmin muodostettiin osakeyhtiö KONE Oy.

- 1910 – KONE Oy perustetaan
- 1933 – KONE aloittaa nosturituotannon
- 1930- ja 1940-luvut – tuotevalikoimaan sisällytetään nostimet ja satamanosturit
- 1980-luku – kansainvälistymispyrkimykset tuottavat hedelmää yrityksen laajentuessa Yhdysvaltoihin
- 1994 – KCI Konecranes yhtiöitetään KONE Oy:stä erilliseksi osakeyhtiöksi
- 2017 – Konecranes ostaa Terex Corporationin MHPS-liiketoiminnan, mukaan lukien Demag-, Gottwald- ja Noell-tuotemerkit
- 2020 – Konecranes vahvistaa markkina-asemaansa Kaakkois-Aasiassa ostamalla MHE-Demagin (Konecranes 2022b).

2 NOSTURIT

2.1 Määritelmä

Nosturilla tarkoitetaan konekäyttöistä nostolaitetta, jota käytetään kuorman nostamiseen, laskemiseen ja siirtämiseen ja jossa kuorma liikkuu ainoastaan nostoköyden, -ketjun tai vastaavan rakenteen ohjaamana. Nosturina pidetään myös sellaista edellä tarkoitettua nostolaitetta, jossa kuorman heiluntaa rajoitetaan nosturin mukana siirtyvillä laitteilla (Valtioneuvoston päätös työvälineiden turvallisesta käytöstä 1403/1993 §3:6.)

Käytännössä nosturilla tarkoitetaan laitetta, joka nostamisen ja laskemisen lisäksi siirtää kuormaa vaaka-akselilla koneellisesti. Esimerkkinä I-palkissa vapaasti liikkuva yli 500 kg:n nostin muuttuu lain silmissä nosturiksi, jos vaakaliike muutetaan koneelliseksi.

Konecranesin tuotteissa yksittäisiä liikkeitä suorittavia kokonaisuuksia kutsutaan koneistoiksi, ja isoimmissa prosessinostureissa niitä voi olla jopa yli kymmenen. Yksi yleisimmistä siltanosturiyhdistelmistä on kuvion 1 mukainen kolmella akselilla toimiva siltanosturi. Konecranesilla koneistoille on määritelty vakiotunnukset, jotka koostuvat kyseiselle liikkeelle määritetystä kirjaimesta ja tarvittaessa järjestysnumerosta: esimerkiksi päänostokoneiston tunnus on A, vaunukoneiston E ja siltakoneiston H.

Kuvio 1 Siltanosturi (Konecranes 2023a)

2.2 Nosturityypit ja niiden rakenne

Nosturityyppejä on lähes yhtä monia kuin niiden käyttökohteitakin: teollisuusnosturi on yleensä huoltokäyttöön suunniteltu siltanosturi tai jotain tuotantoprosessiapalveleva prosessinosturi, joiden nostokapasiteetit voivat nousta satoihin tonneihin.

Kontinkäsittelylaitteista yleisimpiä ovat esimerkiksi rahtisatamissa näkyvät STS-nosturit (kuvio 2), jotka on suunniteltu purkamaan ja lastaamaan laivat mahdollisimman tehokkaasti, sekä RTG-nosturit, jotka hoitavat rahtiterminaalien konttilogistiikan.

Kuvio 1. STS-nosturi (Konecranes 2023b)

Pääkomponentit ja toimintaperiaatteet näissä tyypeissä ovat suurilta osin samat: koneistot ovat pääasiassa invertteriohjattuja, joilla niiden pyörimisnopeus ja/tai momentti säädetään toivotun suuruisiksi, ja muunnetaan vaihteen avulla käyttöön sopivaksi.

Satama- ja prosessinosturien ohjausjärjestelmät ovat pääasiassa PLC-järjestelmiä, jotka voivat sisältää useita etäasemia ja sadoittain I/O:ta, kun taas yksinkertaisemmissa huoltonostimissa ohjaussignaalit menevät ohjauspaikasta, esimerkiksi riippuohjaimesta mahdollisten lukitusten kautta suoraan inverttereille.

3 PLC-JÄRJESTELMÄT JA NIIDEN KONFIGUROINTI

3.1 PLC –ohjelmoitava logiikka

PLC (Programmable Logic Controller) on mikroprosessoripohjainen ohjausyksikkö, joka voidaan ohjelmoida suorittamaan loogisia, aikaperustaisia ja aritmeettisia funktioita, joiden perusteella ohjaimen lähtösignaaleja ohjataan. PLC:n yleisimpiä käyttökohteita ovat erilaiset teollisuuden koneet ja prosessit. (Bolton 2009, 3.)

Yksinkertaisimmillaan PLC sisältää prosessorin, I/O:n ja kommunikointiliitännät samassa rakenteessa, kun taas suurimmat modulaariset järjestelmät voivat sisältää satoja tuloja ja lähtöjä sekä lukuisia erilaisia laajennuskortteja ja etäasemia. (Bolton 2009, 4.)

3.2 Ohjelmointi

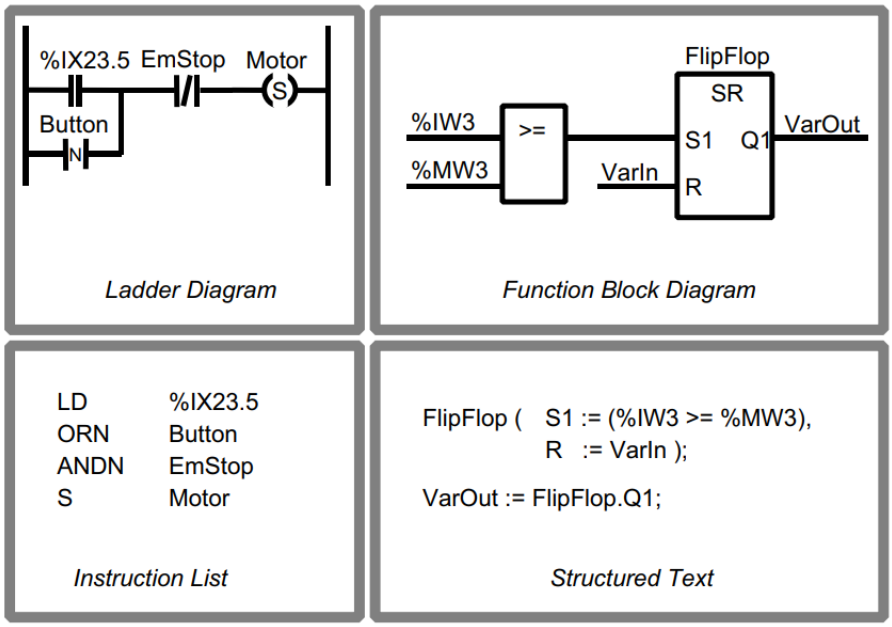
PLC-ohjelmointityökaluissa (nykyään useimmiten PC) käytettävät ohjelmointikielet määritetään IEC 61131-3 standardissa:

1. FBD: Function block diagram (lohkokaavio)
2. LD: Ladder diagram (tikapuukaavio)
3. SFC: Sequential function chart (sekvenssikaavio)
4. ST: Structured text (rakenneteksti)
5. IL: Instruction list (käskylista).

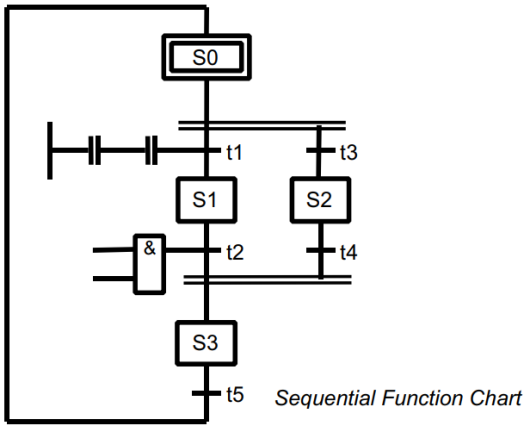
(IEC 61131-3, 14).

Ohjelmointikielistä FBD ja LD ovat esitystavaltaan graafisia, elementtien koostuessa lohkoista ja koskettimista, kun taas tekstimuotoisista ST on korkean tason ohjelmointikieli ja IL matalan tason, lähes konekielinen ohjelmointikieli. SFC (kuvio 4) poikkeaa muista sen toimiessa eräänlaisena graafisena kehyksenä, jolloin se voi pitää sisällään muilla listan ohjelmointikielillä määritettyjä elementtejä (John & Tiegelkamp 2010, 23)

Ohjelmoitaessa käytetty ohjelmointikieli on jossain määrin preferenssikysymys, mutta joitain operaatioita voidaan suorittaa ainoastaan ST ja IL käyttämällä. Pelkästään binaarisia ja aritmeettisia operaatioita ohjelmoitaessa LD ja FBD ajavat asiansa mainiosti, kun taas esimerkiksi osoiterekisteriin vaikuttaminen onnistuu ainoastaan IL:n avulla. ST mahdollistaa korkean tason ohjelmointikielien lauseiden ja silmukoiden käytön (IF, FOR, WHILE) joista on apua esimerkiksi pitkien listojen läpi iteroidessa (kuvio 2).



Kuvio 2. LD-, FBD-, IL- ja ST-esimerkit (John & Tiegelkamp 2010, 24)



Kuvio 3. SFC-esimerkki (John & Tiegelkamp 2010, 25)

3.2.1 STL-käskylistaohjelmointi

STL on Siemensin käyttämä matalan tason koodikieli, joka vastaa (muutamia operaattorieroja lukuun ottamatta) IEC 61131-3 standardissa määriteltyä IL-käskylistakoodikieltä (Siemens 2017).

Käskylista koostuu käskyjen sarjasta. Jokainen käsky alkaa uudelta riviltä ja sisältää operaattorin valinnaisilla muuttujilla, ja tarvittaessa yhden tai useamman operandin pilkuilla erotettuna (IEC 61131-3).

Esimerkkinä kuviossa 5. esitetyssä käskylistassa operaattori (A) vastaa loogista AND-operaattoria, operaattori (AN) vastaa AND-NOT-operaattoria ja operaattori (=) määrittää logiikkaoperaation kohteen. Tulot ja lähdöt (I, Q) ovat tässä esimerkissä operandeja, joita luetaan tai joihin vaikutetaan.

```
A     I    200.0              //Pysäytyspiiri ok
A     I    200.1              //Käynnistyspyyntö ohjauspaikasta
AN    I    200.2              //Sekvenssi kesken
=     Q    200.0              //Käynnistä sekvenssi
```

Kuvio 5. STL-esimerkki

Kuvion 5. koodi kirjoittaa lähtöön (Q 200.0) sitä edeltävän logiikkaoperaation tuloksen, joka on (TRUE) ainoastaan kun tulot (I 200.0 = TRUE) sekä (I 200.1 = TRUE) ja (I 200.2 = FALSE).

3.2.2 Epäsuora osoiteviittaus

Epäsuoralla osoiteviittauksella (eng. indirect addressing) tarkoitetaan PLC-ohjelmoinnissa yksittäiseen dataan tai data-alueeseen viittaamista sen sijainnin perusteella, käyttäjän määrittämän symbolin sijasta. Näin on joissain tapauksissa mahdollista virtaviivaistaa ohjelmointiprosessia, esimerkiksi ohjelmasilmukoita voidaan suorittaa helposti muuttuvilla parametreilla.

Toisaalta epäsuorien viittausten käyttö ohjelmoinnissa vaikuttaa negatiivisesti koodin seurattavuuteen, esimerkiksi tilanteessa, jossa bittiä X kirjoitetaan epäsuorasti, ei kirjoittajaa voida nähdä ohjelmointityökalun ristiviittauksista, joten asianmukaisen dokumentaation ja kommentoinnin rooli korostuu huomattavasti.

Epäsuoraan viittaukseen käytetään erityistä ’Pointer’ -formaattia, joka sisältää alueen, tavun sekä bittiosoitteen, ja data-alueeseen viitatessa lisäksi alueen leveyden (Berger 2012).

STEP 7:ssä on mahdollista käyttää kolmenlaisia pointtereita:

Area-pointer, esim. P# M 10.0, jolloin viitataan dataan, joka sijaitsee muistialueen tavu 10 bitissä 0, DB-pointer, esim. P# DB 20.DBX 10.5, jolloin viitataan dataan, joka sijaitsee datalohko 20, tavu 10, bitissä 5, ja ANY-pointer esim. P# M 16.0 BYTE 8, jolloin viitataan dataan alueessa, joka alkaa muistialueen tavu 16 bitistä 0, ja jonka leveys on 8 tavua.

Tässä työssä keskitytään SFC20 (BLKMOV) -järjestelmäfunktiolla ja dynaamisella ANY-pointterilla toteutettavaan datansiirtoon.

3.2.3 SFC20 ”BLKMOV”

SFC20 on Siemens Simatic 300/400 sarjassa käytettävä järjestelmäfunktio, jolla voidaan kopioida muistialueen sisältö toiseen muistialueeseen ohjelmakierron aikana. Funktio parametroidaan kahdella ANY-pointerilla, joista ensimmäinen osoittaa kopioitavaan muistialueeseen ja toinen kopion kirjoitusalueeseen. Funktio palauttaa heksadesimaalisen koodin suorituksen jälkeen, joka osoittaa onnistuneen kopioinnin tai epäonnistuneen kopioinnin vikakoodin (Siemens 2010.)

Kuvion 6 esimerkissä SFC20 on parametroitu havainnollistamiseksi muuttujien sijasta suoraan pointereilla siten, että datalohko 1 ensimmäinen tavu kopioidaan datalohko 2 toiseen tavuun.

```
CALL "BLKMOV"   SFC20
        SRCBLK  := P# DB1 DBX0.0 BYTE 1         -- Lähde-pointer
        RET_VAL := #ret                         -- Tilakoodi
        DSTBLK  := P# DB2 DBX1.0 BYTE 1         -- Kohde-pointer
```

DB1	
Offset	Value
0.0	0
0.1	1
0.2	1
0.3	1
0.4	0
0.5	1
0.6	1
0.7	1
1.0	1
1.1	1
1.2	0
1.3	0
1.4	1
1.5	1
1.6	1
1.7	0

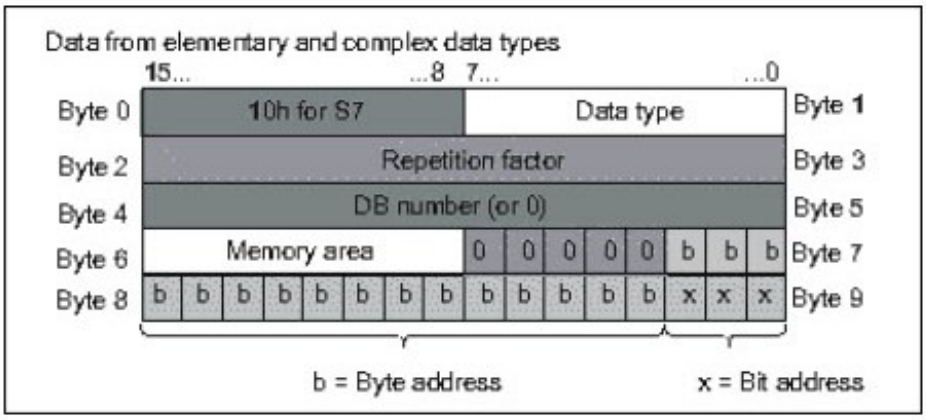
DB2	
Offset	Value
0.0	0
0.1	1
0.2	1
0.3	1
0.4	0
0.5	1
0.6	1
0.7	1
1.0	0
1.1	1
1.2	1
1.3	1
1.4	0
1.5	1
1.6	1
1.7	1

Kuvio 6. Esimerkki SFC20 kutsusta ja toiminnasta

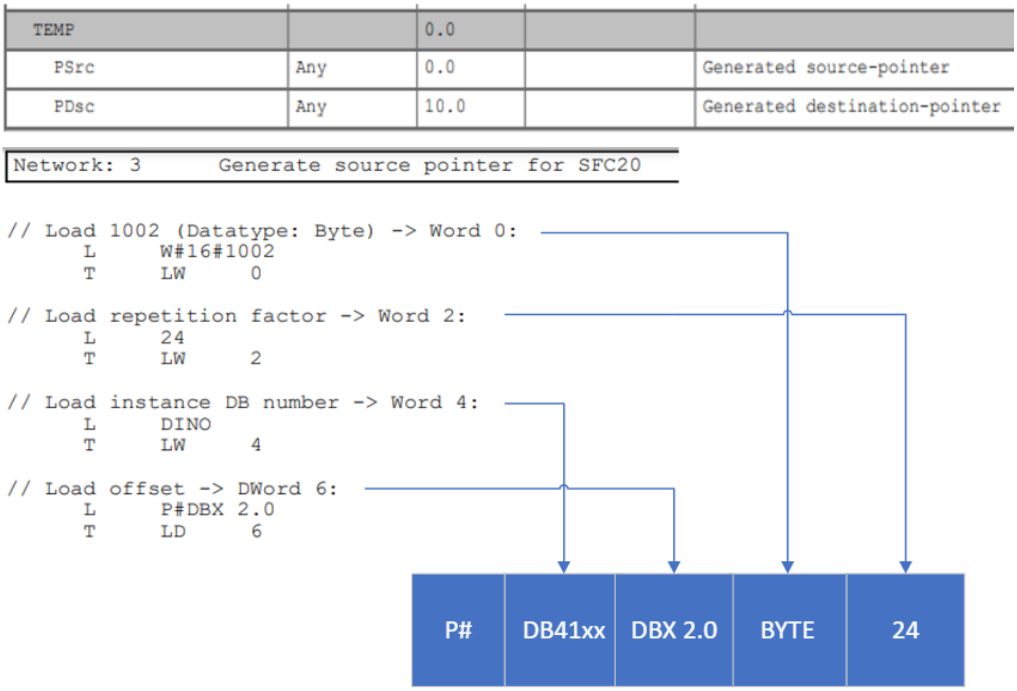
3.2.4 Dynaaminen ANY-pointer

Dynaamisella ANY-pointerilla tarkoitetaan ohjelmakierron aikana määritettyjen ehtojen mukaan muodostettavaa pointer-muuttujaa, jota voidaan käyttää esimerkiksi SFC20 lähde- tai kohdeparametrina. ANY-muuttuja voidaan luoda esimerkiksi aliohjelman väliaikaismuistiin, jossa se muokataan toivottuun muotoon väliaikaismuistiin vaikuttamalla. Kuviossa 7 on esitettynä ANY-datatyypin rakenne

Väliaikaismuistiin kirjoittaminen onnistuu käskylistalla lataamalla haluttu arvo oikeaan kohtaan muistia, tässä tapauksessa ensimmäisen pointerin alkupiste väliaikaismuistissa on 0.0 ja leveys 10 tavua. Kuvia 7 ja 8 vertaamalla nähdään, että esimerkiksi toistokerroin (Integer: 24) ladataan LW2 (Local Word) eli pointerin tavuihin 2 ja 3.



Kuvio 7. ANY-datatyypin rakenne (Siemens 2006, A-58)



Kuvio 8. Lähde-pointerin muodostus

Kuvion 8 koodissa muokataan lähde-pointer-muuttuja datan siirrolle: Tavuun 0 ladataan heksadesimaalina 10 (vakio) ja tavuun 1 ladataan 02, joka määrittää siirrettäväksi dataleveydeksi tavun. Tavuihin 2 ja 3 ladataan toistokerroin, joka määrittää montako peräkkäistä tavua siirretään. Tavuihin 4 ja 5 ladataan lähde-datalohkon numero, DINO palauttaa tähän suoritettavan aliohjelman instanssida-talohkon numeron. Tavuihin 6, 7, 8 ja 9 ladataan siirrettävän datan alkupiste lähde-datalohkossa.

3.3 OPC-rajapinta

OPC (Open Platform Communications) on alustariippumaton yhteensopivuusstandardi, joka mahdollistaa luotettavan ja turvallisen tiedonsiirron automaatiojärjestelmissä useiden laitetoimittajien laitteiden välillä. Standardin kehityksestä ja ylläpidosta vastaa OPC-Foundation, ja sen spesifikaatiot on toteutettu yhteistyössä alan laitetoimittajien, loppukäyttäjien ja ohjelmistokehittäjien kanssa (OPC Foundation 2023b.)

Standardin sisältämät spesifikaatiot määrittävät rajapinnan serverien ja clientien välillä, esimerkiksi reaaliaikaisen monitoroinnin (OPC DA) sekä hälytys-(OPC AE) ja historiadatan(OPC HDA) lukemista varten. Nämä OPC Classic spesifikaatiot ovat Windows-riippuvaisia, ja niiden toiminnallisuudet sisällytettiin 2008 julkaistuun alustariippumattomaan OPC-UA-arkkitehtuuriin (OPC Foundation 2023a.)

OPC-UA (Unified Architecture) suunniteltiin korvaamaan aiemmat OPC Classic -versiot, ja parantamaan entisestään rajapinnan suorituskykyä ja toiminnallisuuksia. Rajapinnan käyttö mahdollistaa tässä käyttötapauksessa helposti konfiguroitavan ja turvallisen tiedonsiirron PLC:n ja IPC:n välillä (OPC Foundation 2023a.)

4 TARKOITUS, TAVOITTEET JA TUTKIMUSONGELMA

Opinnäytetyön tarkoituksena on nopeuttaa ja yksinkertaistaa TruConnect-järjestelmän käyttöönottoa optimoimalla sen PLC-ohjelma. Tavoitteena on esittää tutkimus- ja kehitystyön vaiheet ja tulokset riittävän kattavasti, jotta tuloksia voidaan hyödyntää yleisesti PLC-ohjelmistojen kehityksessä.

Opinnäytetyöstä on hyötyä sekä toimeksiantajayritykselle että loppuasiakkaalle, koska järjestelmän käyttöönotto yksinkertaistuu ja siihen vaadittava aika lyhenee.

4.1 Tutkimuskysymys: Vaadittavat muutokset

Tämän opinnäytetyön pääkysymys on: Mitä muutoksia PLC-ohjelmaan tarvitaan, jotta etävalvontajärjestelmän käyttöönotto nopeutuu?

Ohjelmapaketin käyttöönotosta tulee karsia kaikki päällekkäiset toimenpiteet, esimerkiksi koneiston nopeus syötetään alkuperäisessä versiossa kolmeen eri laskentalohkoon.

Ohjelmapaketti tulee myös esikonfiguroida mahdollisimman pitkälle: yleisimpien koneistojen laskentalohkot määritetään valmiiksi ohjelmaan, ja ohjataan koneistomuuttujat jollain ehdolla oikeaan laskentalohkoon.

Myös nosturin konfiguraatio-datalohko kopioidaan toiseen, vapaaseen datalohkoon ja kaikkien varsinaiseen konfiguraatio-datalohkoon viittaavien elementtien uudelleenohjaus tähän uuteen datalohkoon. Näin saadaan konfiguraatio-datalohkon numerosta vakio, jolloin siihen viittaukset voidaan esikonfiguroida.

4.2 Tutkimuskysymys: Saavutettavat hyödyt

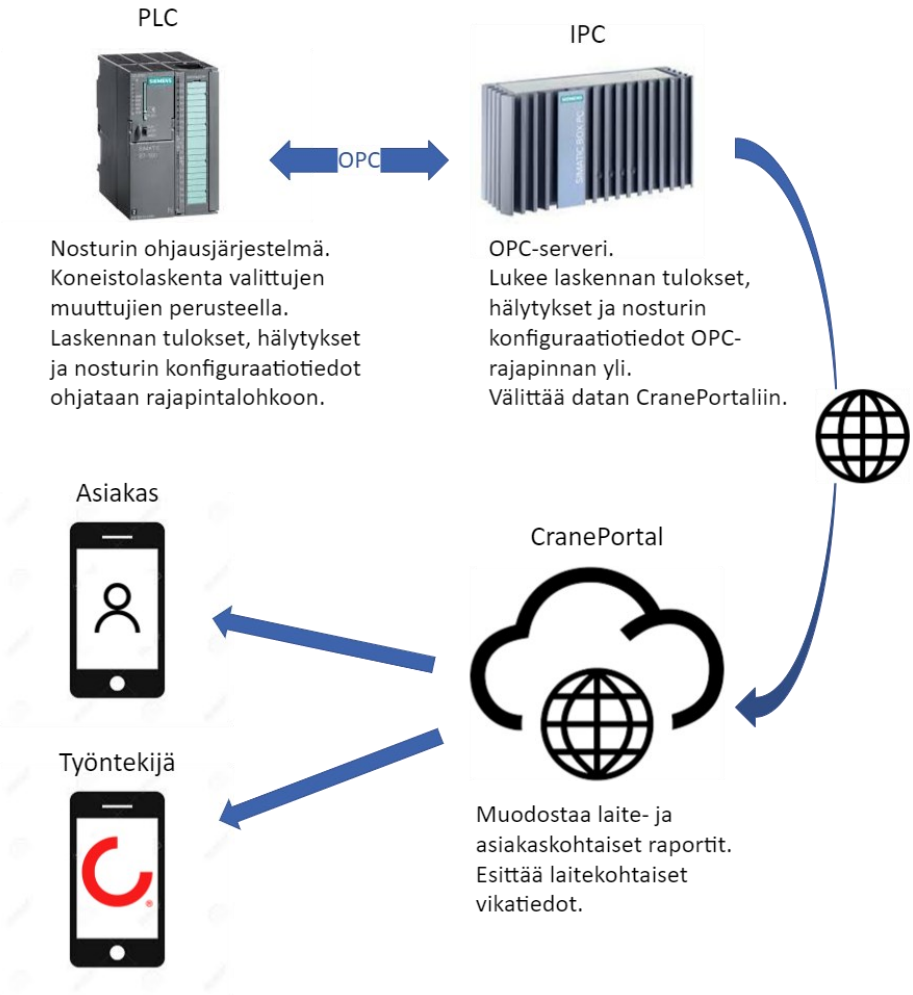
Alikysymys: Millaisia hyötyjä PLC-ohjelman (koneistojen esikonfigurointi ohjelmaan) kehittämisellä voidaan saavuttaa?

Saavutettu hyöty on säästetty aika (ks. mittaustulokset), ei pelkästään siksi että määritettäviä elementtejä on huomattavasti vähemmän, vaan myös siksi että käyttöönoton yksinkertaistumisen ja selkeytymisen vuoksi käyttöohjeen lukemiseen kuluva aika todennäköisesti vähenee, ja käyttöönottoprosessin oppii nopeammin ulkoa.

5 TOTEUTUS

5.1 Opinnäytetyön eteneminen

Ohjelman kehitys aloitettiin Konecranesin globaalissa teknisessä tuessa, jossa työn tarve oli jo aiemmin tunnistettu. Ensimmäinen tehtävä oli käyttöönottaa TruConnect laboratoriolaitteistoon, ja siten tutustua järjestelmään ja sen käyttöönottoprosessiin. Kuviossa 9 on havainnollistettu järjestelmän rakenne.



Kuvio 9. TruConnect havainnekuvio

5.2 Kehitysprosessi

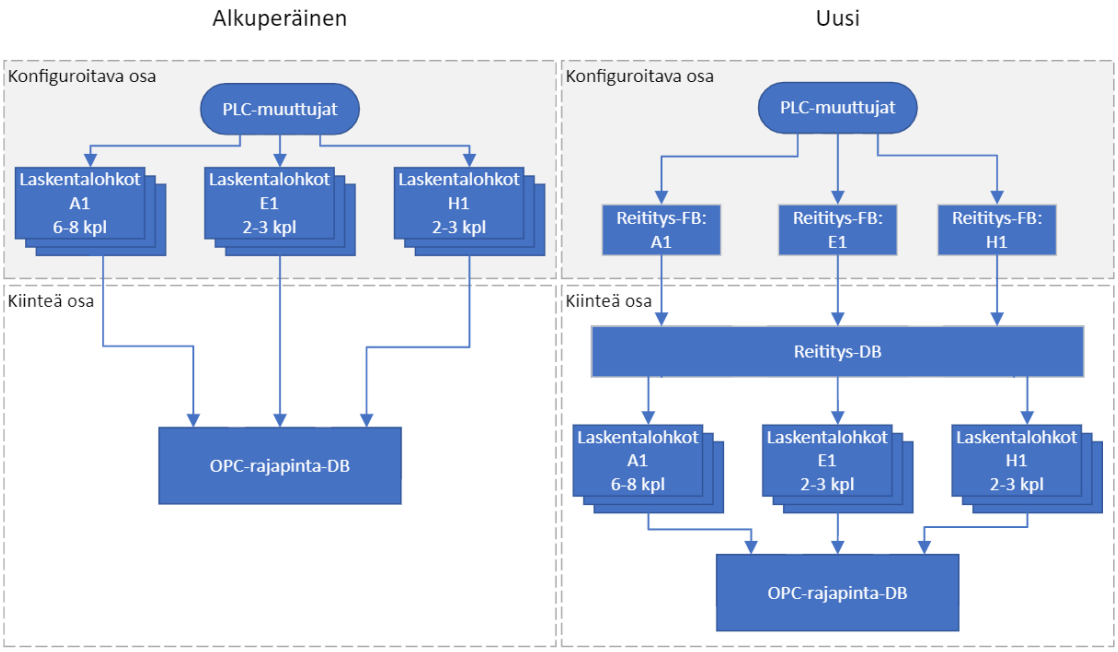
Kehityksestä oli jo visio olemassa: uusi paketti rakennetaan alkuperäisen ohjelmapaketin ympärille. Tämä tarkentui myöhemmin esikonfigurointiin, eli uusi oh-

jelmapaketti sisältää alkuperäisen, johon on ennalta konfiguroitu yleisimmät koneistot, ja muuttujat ohjataan jonkin ehdon mukaan oikeisiin osoitteisiin (kuvio 10).

Seurattavuuden vuoksi reititykseen päädyttiin käyttämään datalohkoja: valittu koneistomuuttuja kirjoittaa reititys-aliohjelman instanssidatalohkoon, josta aliohjelma ohjaa muuttujan datan annetun koneistoparametrin mukaan reititys-datalohkon kautta oikeaan laskentalohkoon.

Tähän datansiirtoon vaihtoehdot olivat järjestelmäfunktiot SFC20 ”BLKMOV” ja SFC81 ”UBLKMOV” joiden ainoa ero on siinä, että SFC81 on keskeyttämätön, joka taas voi vaikuttaa ohjelman keskeytysaikaan ja siten koko nosturin toimintaan. Näistä valittiin ei-kriittiselle valvontajärjestelmälle turvallisempi SFC20.

Aluksi datansiirto suoritettiin järjestelmäfunktion kuudella erillisellä, kiinteästi parametroidulla instanssilla, kunnes opiskelun ja testaamisen tuloksena onnistuttiin muodostamaan pointer-parametri ohjelmakierron aikana, jolloin SFC20-instanssien määrä putosi yhteen.



Kuvio 10. Ohjelmarakenne

Kun ohjelman toiminta oli varmennettu laboratorioympäristössä, oli vuorossa käyttöohjeen päivitys ja mittausten suorittaminen: valittiin kaksi erilaista nosturiohjelmaa, joihin integroitiin vuorotellen sekä vanha että uusi ohjelmapaketti ja mitattiin integrointiin kulunut aika sekä tehtyjen määritysten/viittausten lukumäärä.

5.3 PLC-Ohjelma

PLC-ohjelman kehityksen tuloksena syntyi yksi parametroitava aliohjelma, sekä yksi funktio, joiden rakenne ja toiminta selostetaan seuraavissa kappaleissa.

5.3.1 FB4100 - Koneistomuuttujien reititys

Truconnect PLC-ohjelmaan kehitetty reititys-aliohjelma kohdistaa siihen liitetyt muuttujat koneistoparametrin perusteella oikeaan kohtaan reititys-datalohkoa, josta muuttujat on valmiiksi linkitetty kyseessä olevan koneiston laskentalohkoihin.

Kohdistus tapahtuu aliohjelmassa muodostamalla luvun 3.5 mukaisesti kohdepointer annettua koneistoparametria vastaavalla alkupisteellä, esimerkiksi koneistolla A1 alkupiste on 0.0 kun taas B1 on 28.0 jne.

Ensimmäiseksi aliohjelma käsittelee nosturin yhteisdatan (kuvio 11), eli kaikille koneistoille yhteiset muuttujat, esimerkiksi pääkontaktorin tilatiedon. Nämä ”common data” -muuttujat tulee määrittää vain yhteen FB4100-instanssiin, sillä niiden kohde on aina vakiona reititys-datalohkon lopussa: muuttujien määrittäminen useampaan instanssiin tarkoittaisi sitä, että reititys-datalohkon yhteisdataa kirjoitettaisiin useammasta paikasta, millä voi olla epätoivottuja vaikutuksia.

```
Network: 1      Handle common data

      A     #I_Common_MC_On                #I_Common_MC_On    -- Crane main-contactor on
      =     "TC_Input_table".mc_on         DB4031.DBX224.0

      A     #I_Common_Fault                #I_Common_Fault    -- Crane fault on
      =     "TC_Input_table".common_fault  DB4031.DBX224.1

      A     #I_Common_Alarm                #I_Common_Alarm    -- Crane alarm on
      =     "TC_Input_table".common_alarm  DB4031.DBX224.2

      A     #I_Common_Event                #I_Common_Event    -- Crane event on
      =     "TC_Input_table".common_event  DB4031.DBX224.3
```

Kuvio 11. Yhteisdatan reititys

Seuraavaksi tunnistetaan instanssiin linkitetty koneisto: FB4100 vaatii koneiston tunnistamiseksi kaksi 8-bittistä ”char”-datatyypin ASCII-merkkiä, esimerkiksi ’A’ ja ’1’. Koska ASCII-merkit kääntyvät ohjelmassa suoraan numeroarvoiksi, voidaan koneisto tunnistaa kertomalla merkit keskenään: ’A’ kertaa ’1’ vastaa bittitasolla kertolaskua 65 kertaa 49, jonka tulos on 3185.

Tämän jälkeen kertolaskun tulosta vertaillaan tuetuille koneistoille määrättyihin numeroarvoihin, ja vastaavan arvon löytyessä asetetaan kyseessä olevan koneiston tunnistusbitti todeksi (kuvio 12).

```
Network: 2      Identify machinery

// Generate Mach_ID_int by multiplying input characters:
      L     #Mach_id_char  #Mach_id_char      -- Machinery identification character 1
      L     #Mach_id_num   #Mach_id_num       -- Machinery identification character 2
      *I
      T     #mach_ID_int   #mach_ID_int       -- Generated by multiplying machID chars

//Scan for A1:
      L     #mach_ID_int   #mach_ID_int       -- Generated by multiplying machID chars
      L     3185
      ==I
      =     #Machinery_A1  #Machinery_A1      -- True if machID matches A1

//Scan for B1:
      L     #mach_ID_int   #mach_ID_int       -- Generated by multiplying machID chars
      L     3234
      ==I
      =     #Machinery_B1  #Machinery_B1      -- True if machID matches B1
```

Kuvio 12. Koneiston tunnistaminen

Koneiston tunnistamiseen kokeiltiin useampaa eri vaihtoehtoa: Aluksi ASCII-merkit laskettiin yhteen kertolaskun sijasta, mutta tästä seurasi, että esimerkiksi yhdistelmien A1 ja B2 summa oli sama.

Paras parametrivaihtoehto tunnistamiseen olisi ollut String eli merkkijono, esimerkiksi ”A1”, mutta vaikka Simatic antaa asettaa aliohjelmaa tai funktiota luodessa tuloparametrin String-datatyyppiseksi, työkalu antaa virheilmoituksen vääräntyyppisestä parametrista, vaikka parametrin antaisi oikeassa formaatissa.

Kolmannessa osiossa generoidaan lähde-pointer datan reititystä varten luvun 3.2.4 mukaisesti: FB4100 paikallismuistissa sijaitsevaan pointer-muuttujaan kirjoitetaan arvot siten, että pointer osoittaa suoritettavan FB4100-instanssin datalohkoon ja tarkemmin sen koneistomuuttujiin (kuvio 13).

```
Network: 3      Generate source pointer for SFC20

// Load 1002 (Datatype: Byte) -> Word 0:
      L     W#16#1002
      T     LW     0

// Load repetition factor -> Word 2:
      L     24
      T     LW     2

// Load instance DB number -> Word 4:
      L     DINO
      T     LW     4

// Load offset -> DWord 6:
      L     P#DBX 2.0
      T     LD     6
```

Kuvio 13. Lähde-pointer muodostus

Neljännessä osiossa generoidaan kohde-pointer datansiirrolle, myös luvun 3.2.4 mukaisesti (kuvio 14). Kohde-pointerissa poikkeuksena datalohkon numero on vakio (reititys-datalohko) mutta datan kohdistus muuttuu aiemmin asetetun koneiston tunnistusbitin perusteella, sillä kaikki määritetyt koneistot ovat peräkkäin reititys-datalohkossa: ohjelma tarkistaa koneistobitin, ja mikäli bitti on tosi, ladataan sitä vastaava alkupiste ja hypätään kirjoitukseen (t:). Mikäli bitti ei ole tosi, hypätään tarkistamaan seuraavaa koneistobittiä.

```
Network: 4      Generate destination pointer for SFC20

// Load 1002 (Datatype: Byte) -> Word 0:
      L     W#16#1002
      T     LW    10


// Load repetition factor -> Word 2:
      L     24
      T     LW    12

// Load destination DB number -> Word 4:
      L     4031
      T     LW    14


// Choose machinery offset pointer:
      A     #Machinery_A1      #Machinery_A1      -- True if machID matches A1
      JCN   b1
      L     P#DBX 0.0
      JU    t

b1:   A     #Machinery_B1      #Machinery_B1      -- True if machID matches B1
      JCN   c1
      L     P#DBX 28.0
      JU    t
```

Kuvio 14. Kohde-pointerin generointi

Mikäli yksikään koneistobiteistä ei ollut tosi, ohjelma asettaa vikabitin ”Invalid_MachID” todeksi. Muutoin pointer-muuttujaan ladataan aiemmin tunnistettua koneistoa vastaava alkupiste (kuvio 15).

```
o1:   A     #Machinery_O1      #Machinery_O1      -- True if machID matches O1
      JCN   Err
      L     P#DBX 196.0
      JU    t


// Set errorbit if no match found:
Err:  S     #Q_Invalid_machID  #Q_Invalid_machID  -- True if machID is invalid
      BE


// Load machinery offset pointer -> bytes 6-9:
t:    T     LD    16


// Reset errorbit if offset loaded succesfully:
      R     #Q_Invalid_machID  #Q_Invalid_machID  -- True if machID is invalid
```

Kuvio 15. Kohde-pointerin muodostus ja viankäsittely

Lopuksi aliohjelma kutsuu järjestelmäfunktion SFC20 ”BLKMOV” muodostetuilla pointer-muuttujilla (kuvio 16).

```
Network: 5

CALL  "BLKMOV"          SFC20              -- Copy Variables
 SRCBLK :=#PSrc         #PSrc              -- Generated source-pointer
 RET_VAL:=#Q_scf20_ret  #Q_scf20_ret       -- SFC20 Blkmove return value (0=ok)
 DSTBLK :=#PDsc         #PDsc              -- Generated destination-pointer
```

Kuvio 16. SFC20-kutsu

5.3.2 FC901 - Konfiguraatio-datalohkon reititys

Koneistojen nimellisarvot (esimerkiksi nimelliskuorma ja nopeus) laskentalohkoille tulevat viittaamalla nosturin konfiguraatio-datalohkoon (vakio nro. 900). Joissain tapauksissa lohkonumero 900 on kuitenkin muussa käytössä, jolloin jokainen esikonfiguroitu nimellisarvoviittaus on korjattava ja käyttöönotto hidastuu.

Ongelma ratkaistiin määrittämällä esikonfiguraatiossa datalohkon numeroksi 4200, joka reilusti korkeampana numerona ei ole Konecranes-ohjelmissa juuri koskaan käytössä (normaalisti välillä 1-5000), ja luomalla välittäjäksi vastaavanlainen reititys-aliohjelma kuin koneistomuuttujille. Tämän aliohjelman ainoa tuloparametri on varsinaisen konfiguraatio-datalohkon numero ohjelmassa, jonka sisältö kopioidaan ensimmäisellä ohjelmakierrolla datalohko 4200:n.

Jokainen nimellisarvo siis noudetaan esikonfiguraatiossa oletuksena datalohko 4200:sta, jonne ne on ohjattu varsinaisesta konfiguraatio-datalohkosta, jonka numero annetaan käyttöönotossa reitittäjälle muuttujaan *#DZ_Config_DBnum*, jota käytetään alla kuviossa 17 lähde-pointerin muodostamiseen.

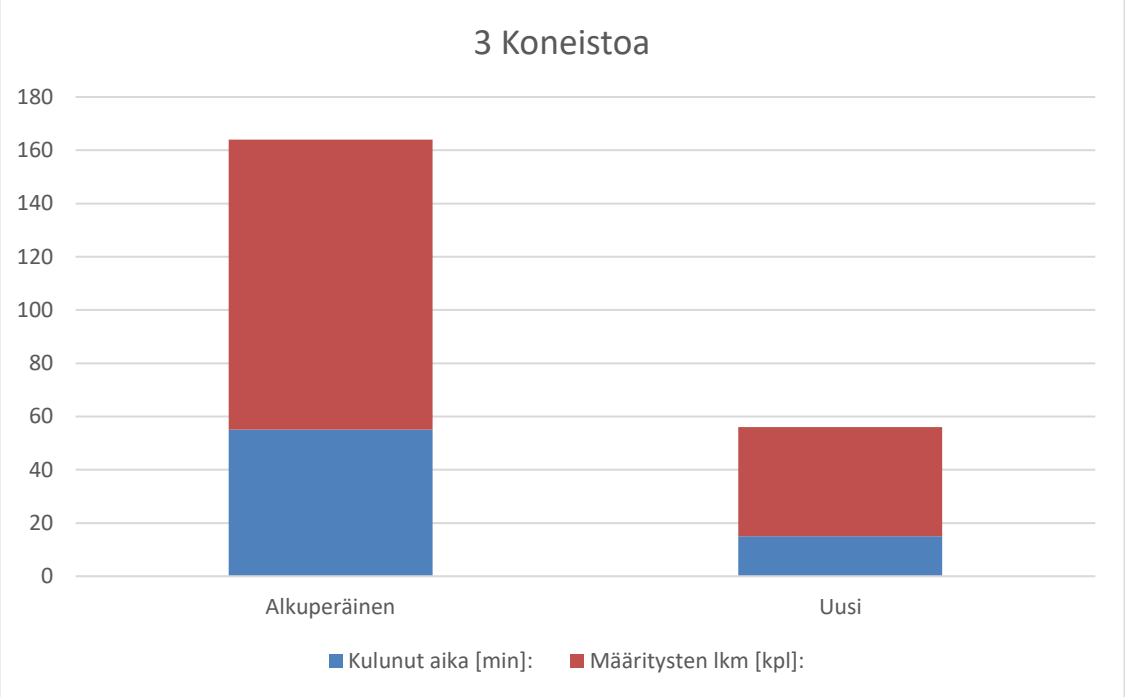
```
Network: 1 Generate pointer

// Load 10 (default) -> Byte 0:
        L B#16#10
        T LB 0

// Load 2 (datatype: Byte) -> Byte 1:
        L B#16#2
        T LB 1

// Load repetition factor -> Bytes 2 & 3:
        L W#16#338
        T LW 2

// Load source DB number -> Bytes 4 & 5:
        L #DZ_Config_DBnum
        T LW 4

// Load offset pointer -> Bytes 6-9:
        L P#DBX 0.0
        T LD 6

Network: 2 Call SFC20

        CALL "BLKMOV" SFC20 -- Copy Variables
        SRCBLK :=#Psrc
        RET_VAL:=#Ret
        DSTBLK :=P#DB4200.DBX0.0 BYTE 338
```

Kuvio 17. FC901 -ohjelmarakenne

6 TULOKSET

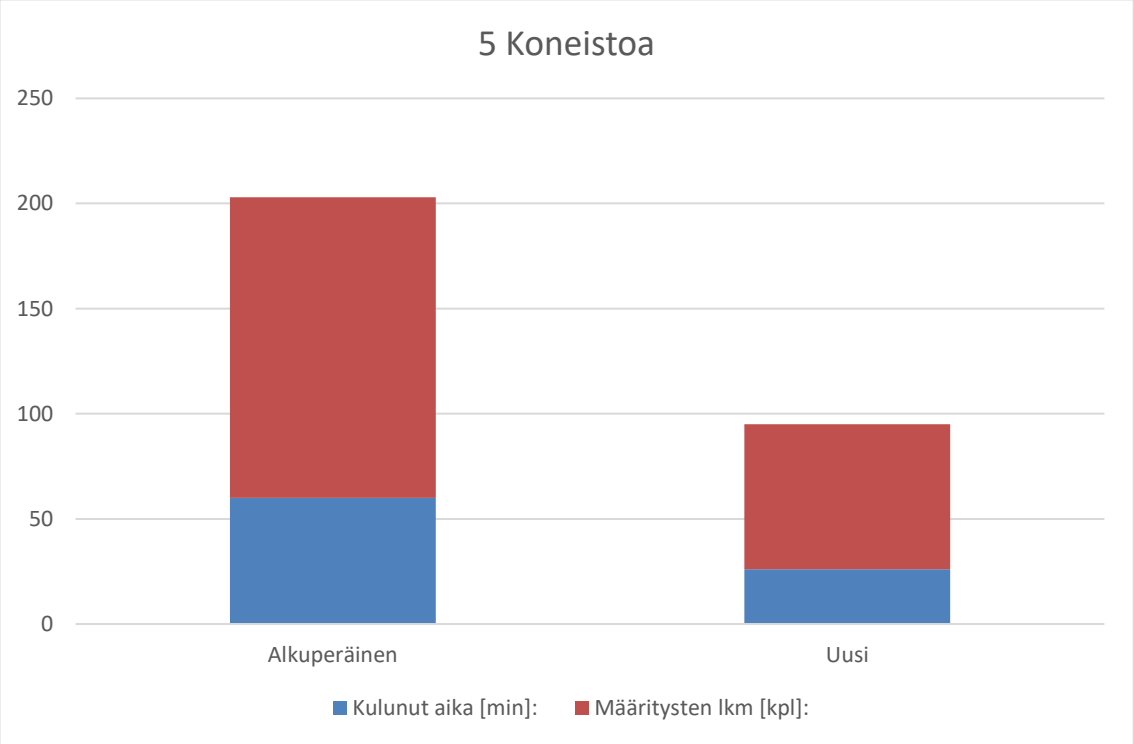
Mittaustuloksista on selkeästi nähtävissä esikonfiguroinnin positiivinen vaikutus PLC-ohjelmapaketin integrointiin, kolmella koneistolla varustetussa nosturissa integrointiin kulunut aika väheni 73 % (kuvio 18), ja viidellä koneistolla vastaavasti 57 % (kuvio 19).

Ohjelmapaketti:	Alkuperäinen	Uusi	Muutos
Kulunut aika:	55 min	15 min	-73 %
Määritysten lkm:	109 kpl	41 kpl	-62 %
PLC-ohjelman koko:	-	+11,4 %	+11,4 %



Kuvio 18 Käyttöönoton vertailu kolmen koneiston nosturissa

Ohjelmapaketti:	Alkuperäinen	Uusi	Muutos
Kulunut aika:	60 min	26 min	-57 %
Määritysten lkm:	143 kpl	69 kpl	-52 %
PLC-ohjelman koko:	-	+11,4 %	+11,4 %



Kuvio 19 Käyttöönoton vertailu viiden koneiston nosturissa

Negatiivisena puolena ohjelmapaketin koko kasvoi 11.4 % alkuperäiseen verrattuna, mutta koska 3-koneistoisen nosturin PLC:n työmuistin (384kB) käyttöaste kasvoi kuitenkin vain 28 % → 31 % uuden ohjelmapaketin myötä, ei tästä käytännössä seuraa ongelmia .

7 JATKOKEHITTÄMINEN

Ohjelmapakettia voidaan jatkokehittää sisällyttämällä koneistokatalogi kokonaisuudessaan reititys-aliohjelmaan, mikäli se ei kasvata ohjelmapaketin kokoa liikaa. Tässä versiossa on esikonfiguroitu kahdeksan yleisintä koneistoa 50:stä mahdollisesta.

Koska kahdeksan koneiston esikonfigurointi kasvatti kyseessä olevan nosturin PLC-muistin käyttöastetta noin 3 %, kasvattaisi kaikkien 50:n esikonfigurointi käyttöastetta noin 19 %, mikä on jo huomattavan paljon. Loppujen lopuksi tämän jatkokehityksen kannattavuus riippuu täysin PLC-muistien ylimitoituksesta suunnittelussa.

Tämän kehityksen voisi toteuttaa esimerkiksi rinnakkaisversiona: tilanteen vaatiessa käytettäisiin ohjelmapaketin 50-koneiston täysversiota, ja vastaavasti kevyempää 8-koneiston versiota sen riittäessä.

8 POHDINTA

Opinnäytetyön suorittaminen oli kaikkiaan aikaa vievä prosessi, sillä pelkästään järjestelmän käyttöönotto edellytti paljon laitteistoon perehtymistä, sekä suuren määrän PLC-ohjelmoinnin opiskelua ja toteutettujen ratkaisujen testaamista. Kehitystyötä aloittaessani STL ohjelmointikielenä oli itselleni täysin vieras, ja muun muassa pointer käsitteenä tuntematon.

Opinnäytetyön hyöty osaamisen kehittymisen kannalta oli erittäin arvokas, sillä kehitystyö opetti käyttämään näitä työkaluja tehokkaasti, ja konfiguroimaan laitteiden välisiä yhteyksiä ja rajapintoja. Vaikka STL onkin ohjelmointikielenä vanhentunut, siihen törmää vanhemmassa laitekannassa vielä usein.

Opinnäytetyön laatua ja raportointia olisi helpottanut, jos opinnäytetyöprosessin olisi käynnistänyt ennen kehitystyön aloittamista, tai edes samanaikaisesti, koska tällöin kehitysprosessin järjestelmällisyyteen olisi todennäköisesti kiinnitetty enemmän huomiota, mutta havahduin kehitystyön potentiaaliin opinnäytetyöksi liian myöhään.

Tästä huolimatta opinnäytetyö onnistui hyvin, ja mittaustuloksista voidaan päätellä, että työn tavoitteet saavutettiin.

LÄHTEET

John, K. & Tiegelkamp, M. 2010. IEC 61131-3: Programming industrial automation systems: concepts and programming languages, requirements for programming systems, decision-making aids (2nd ed.). https://link.springer.com/book/10.1007/978-3-642-12015-2. Springer.

Berger, H. 2012. Automating with STEP 7 in STL and SCL: Programmable controllers SIMATIC S7-300/400 (6th rev. and enlarged ed.). Publicis Pub.

Bolton, W. 2009. Programmable Logic Controllers (5th ed). Newnes.

IEC 61131-3. 2003. Programmable controllers – Part 3: Programming languages. 2.painos. Geneve: International Electrotechnical Commission IEC.

Konecranes 2023a. SMARTON® open winch crane. Viitattu 5.3.2023. https://www.konecranes.com/equipment/overhead-cranes/open-winch-cranes/smartonr-open-winch-crane

Konecranes 2023b. STS Raising. Viitattu 5.3.2023. https://www.konecranes.com/discover/sts-raising

OPC Foundation 2023a. Unified Architecture. Viitattu 11.12.2022. https://opcfoundation.org/about/opc-technologies/opc-ua/

– 2023b. What is OPC?. Viitattu 11.12.2022. https://opcfoundation.org/about/what-is-opc/.

SFS-ISO 4306-1:2007. Nosturit. Sanasto. Osa 1: Yleistä. Helsinki: SFS.

Siemens 2001. Parameterizing the SFC 20 (BLKMOV) with ANY Pointer. https://support.industry.siemens.com/cs/document/2738030/parameterizing-the-sfc-20-(blkmov)-with-any-pointer?dti=0&lc=en-WW

Siemens 2006. SIMATIC Programming with STEP 7. Viitattu 1.2.2023. https://cache.industry.siemens.com/dl/files/056/18652056/att_70829/v1/S7prv54_e.pdf

Siemens 2010. System Software for S7-300/400 System and Standard Functions. Viitattu 1.2.2023. https://cache.industry.siemens.com/dl/files/604/44240604/att_67003/v1/s7sfc_en-EN.pdf

Siemens 2013. How do you program the PLC with STEP 7 (TIA Portal) in compliance with the IEC 61131-3 standard? Viitattu 1.2.2023. https://support.industry.siemens.com/cs/document/50204938/how-do-you-program-the-plc-with-step-7-(tia-portal)-in-compliance-with-the-iec-61131-3-standard-?dti=0&lc=en-WW

Siemens 2017. Statement List (STL) for S7-300 and S7-400 Programming. Viitattu 1.2.2023. https://cache.industry.siemens.com/dl/files/814/109751814/att_933093/v1/STEP_7_-_Statement_List_for_S7-300_and_S7-400.pdf

LIITTEET

Liite 1. FB4100 ”TC_Mach_Inputs” -ohjelmarakenne

Liite 2. FB4100 ”TC_Mach_Inputs” -käyttöohje

Liite 1 1(4) FB4100 ”TC_Mach_Inputs” -ohjelmarakenne

FB4100 - <offline>

"TC_Mach_Inputs" TC_Mach_Inputs
Name: TC_Mach_ **Family:** TC_Mach_
Author: JHo **Version:** 0.1
Block version: 2
Time stamp Code: 10/12/2022 11:16:38 AM
Interface: 10/12/2022 11:16:38 AM
Lengths (block/logic/data): 00650 00434 00028

Name	Data Type	Address	Initial Value	Comment
IN		0.0		
Mach_id_char	Char	0.0	' '	Machinery identification character 1
Mach_id_num	Char	1.0	' '	Machinery identification character 2
ICP_D1	Bool	2.0	FALSE	Machinery direction 1 request
ICP_D2	Bool	2.1	FALSE	Machinery direction 2 request
I_DrivingOn	Bool	2.2	FALSE	Machinery moving
I_FloatFeedback	Bool	2.3	FALSE	Machinery load floating active feedback
I_D1_Feedback	Bool	2.4	FALSE	Machinery running to direction 1 feedback
I_D2_Feedback	Bool	2.5	FALSE	Machinery running to direction 2 feedback
I_r32Load	Real	4.0	0.000000e+000	Current load [t]
I_SpeedFB	Int	8.0	0	Speedfeedback scaled between -100...0...+100 (+-100 % = Max speeds)
I_i16MaxSpeed	Int	10.0	0	Speedfeedback scaled between -100...0...+100 (+-100 % = Max speeds)
I_i16MinSpeed	Int	12.0	0	Speedfeedback scaled between -100...0...+100 (+-100 % = Max speeds)
I_i32DesignedBrkStops	DInt	14.0	L#0	Designed brake stops (Brake life time)
I_boHoistOverload_1	Bool	18.0	FALSE	Hoist overload signal sensor 1
I_r32Position	Real	20.0	0.000000e+000	Current machinery position [m]
I_BrakeOpen	Bool	24.0	FALSE	Machinery brake open
I_LoadOn	Bool	24.1	FALSE	Machinery loaded
I_FaultAct	Bool	24.2	FALSE	Fault Active
I_i16MotorTemp	Int	26.0	0	Current Motor temp in Celcius degrees
I_Common_MC_On	Bool	28.0	FALSE	Crane main-contactor on
I_Common_Fault	Bool	28.1	FALSE	Crane fault on
I_Common_Alarm	Bool	28.2	FALSE	Crane alarm on
I_Common_Event	Bool	28.3	FALSE	Crane event on
OUT		0.0		
Q_Invalid_machID	Bool	30.0	FALSE	True if machID is invalid
Q_scf20_ret	Int	32.0	0	SFC20 Blkmove return value (0=ok)
IN_OUT		0.0		
STAT		0.0		
mach_ID_int	Int	34.0	0	Generated by multiplying machID chars
Machinery_A1	Bool	36.0	FALSE	True if machID matches A1
Machinery_B1	Bool	36.1	FALSE	True if machID matches B1
Machinery_C1	Bool	36.2	FALSE	True if machID matches C1
Machinery_E1	Bool	36.3	FALSE	True if machID matches E1
Machinery_E2	Bool	36.4	FALSE	True if machID matches E2
Machinery_H1	Bool	36.5	FALSE	True if machID matches H1
Machinery_H2	Bool	36.6	FALSE	True if machID matches H2
Machinery_O1	Bool	36.7	FALSE	True if machID matches O1
TEMP		0.0		
PSrc	Any	0.0		Generated source-pointer
PDsc	Any	10.0		Generated destination-pointer

Block: FB4100

Common data inputs need to be set in only one instance of FB

Machinery identification is used to insert mach.data into the correct point in DB4031.

Liite 1 2(4) FB4100 ”TC_Mach_Inputs” -ohjelmarakenne

```
Machinery ID parameters are set with characters (case sensitive) ex;
Mach_id_char: 'A'
Mach_id_num: '1'

Supported combinations:
A1
B1
C1
E1
E2
H1
H2
O1
22.6.2022//JHo
```

Network: 1 Handle common data

```
      A     #I_Common_MC_On                #I_Common_MC_On    -- Crane main-contactor on
      =     "TC_Input_table".mc_on         DB4031.DBX224.0

      A     #I_Common_Fault                #I_Common_Fault    -- Crane fault on
      =     "TC_Input_table".common_fault  DB4031.DBX224.1

      A     #I_Common_Alarm                #I_Common_Alarm    -- Crane alarm on
      =     "TC_Input_table".common_alarm  DB4031.DBX224.2

      A     #I_Common_Event                #I_Common_Event    -- Crane event on
      =     "TC_Input_table".common_event  DB4031.DBX224.3
```

Network: 2 Identify machinery

```
// Generate Mach_ID_int by multiplying input characters:
      L     #Mach_id_char  #Mach_id_char      -- Machinery identification character 1
      L     #Mach_id_num   #Mach_id_num       -- Machinery identification character 2
      *I
      T     #mach_ID_int   #mach_ID_int       -- Generated by multiplying machID chars

//Scan for A1:
      L     #mach_ID_int   #mach_ID_int       -- Generated by multiplying machID chars
      L     3185
      ==I
      =     #Machinery_A1  #Machinery_A1      -- True if machID matches A1

//Scan for B1:
      L     #mach_ID_int   #mach_ID_int       -- Generated by multiplying machID chars
      L     3234
      ==I
      =     #Machinery_B1  #Machinery_B1      -- True if machID matches B1

//Scan for C1:
      L     #mach_ID_int   #mach_ID_int       -- Generated by multiplying machID chars
      L     3283
      ==I
      =     #Machinery_C1  #Machinery_C1      -- True if machID matches C1

//Scan for E1:
      L     #mach_ID_int   #mach_ID_int       -- Generated by multiplying machID chars
      L     3381
      ==I
      =     #Machinery_E1  #Machinery_E1      -- True if machID matches E1

//Scan for E2:
      L     #mach_ID_int   #mach_ID_int       -- Generated by multiplying machID chars
      L     3450
      ==I
      =     #Machinery_E2  #Machinery_E2      -- True if machID matches E2

//Scan for H1:
      L     #mach_ID_int   #mach_ID_int       -- Generated by multiplying machID chars
      L     3528
      ==I
      =     #Machinery_H1  #Machinery_H1      -- True if machID matches H1

//Scan for H2:
```

Liite 1 3(4) FB4100 ”TC_Mach_Inputs” -ohjelmarakenne


```
      L     #mach_ID_int   #mach_ID_int       -- Generated by multiplying machID chars
      L     3600
      ==I
      =     #Machinery_H2  #Machinery_H2      -- True if machID matches H2

//Scan for O1:
      L     #mach_ID_int   #mach_ID_int       -- Generated by multiplying machID chars
      L     3871
      ==I
      =     #Machinery_O1  #Machinery_O1      -- True if machID matches O1
```

Network: 3 Generate source pointer for SFC20

```
// Load 1002 (Datatype: Byte) -> Word 0:
      L     W#16#1002
      T     LW     0

// Load repetition factor -> Word 2:
      L     24
      T     LW     2

// Load instance DB number -> Word 4:
      L     DINO
      T     LW     4

// Load offset -> DWord 6:
      L     P#DBX 2.0
      T     LD     6
```

Network: 4 Generate destination pointer for SFC20

```
// Load 1002 (Datatype: Byte) -> Word 0:
      L     W#16#1002
      T     LW    10


// Load repetition factor -> Word 2:
      L     24
      T     LW    12

// Load destination DB number -> Word 4:
      L     4031
      T     LW    14


// Choose machinery offset pointer:
      A     #Machinery_A1      #Machinery_A1      -- True if machID matches A1
      JCN   b1
      L     P#DBX 0.0
      JU    t

b1:   A     #Machinery_B1      #Machinery_B1      -- True if machID matches B1
      JCN   c1
      L     P#DBX 28.0
      JU    t

c1:   A     #Machinery_C1      #Machinery_C1      -- True if machID matches C1
      JCN   e1
      L     P#DBX 56.0
      JU    t

e1:   A     #Machinery_E1      #Machinery_E1      -- True if machID matches E1
      JCN   e2
      L     P#DBX 84.0
      JU    t

e2:   A     #Machinery_E2      #Machinery_E2      -- True if machID matches E2
      JCN   h1
      L     P#DBX 112.0
      JU    t

h1:   A     #Machinery_H1      #Machinery_H1      -- True if machID matches H1
      JCN   h2
      L     P#DBX 140.0
      JU    t
```

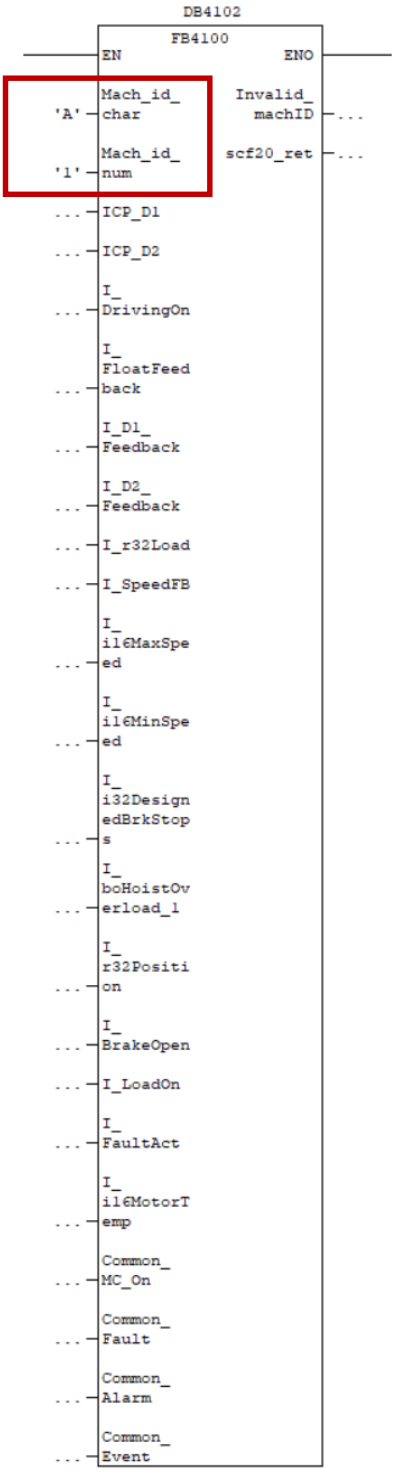
Liite 1 4(4) FB4100 ”TC_Mach_Inputs” -ohjelmarakenne


```
h2:   A     #Machinery_H2      #Machinery_H2      -- True if machID matches H2
      JCN   o1
      L     P#DBX 168.0
      JU    t

o1:   A     #Machinery_O1      #Machinery_O1      -- True if machID matches O1
      JCN   Err
      L     P#DBX 196.0
      JU    t


// Set errorbit if no match found:
Err:  S     #Q_Invalid_machID  #Q_Invalid_machID  -- True if machID is invalid
      BE


// Load machinery offset pointer -> bytes 6-9:
t:    T     LD    16


// Reset errorbit if offset loaded succesfully:
      R     #Q_Invalid_machID  #Q_Invalid_machID  -- True if machID is invalid
```

Network: 5

```
      CALL  "BLKMOV"           SFC20              -- Copy Variables
       SRCBLK :=#PSrc          #PSrc              -- Generated source-pointer
       RET_VAL:=#Q_scf20_ret   #Q_scf20_ret       -- SFC20 Blkmove return value (0=ok)
       DSTBLK :=#PDsc          #PDsc              -- Generated destination-pointer
```

1.1.1 Configuring FB4100 – “TC_Mach_Inputs”



	Call structure: FB4100 should be called in FC4000 and must be called before TC-calculation blocks. Every instance of FB4100 needs to have its own instance-DB generated. The left column depicts which machineries require the input.		
Mach.	**Parameter**	**Data Type**	**Description**
All	Mach_id_char	CHAR	Machinery ID character (case sensitive) ex. ‘A’
All	Mach_id_num	CHAR	Machinery ID number ex. ‘1’
Hoist	ICP_D1	BOOL	Machinery direction 1 request
Hoist	ICP_D2	BOOL	Machinery direction 2 request
All	I_DrivingOn	BOOL	Driving ON
Hoist	I_FloatFeedback	BOOL	Machinery load floating active feedback
Hoist	I_D1_Feedback	BOOL	Machinery running to direction 1 feedback
Hoist	I_D2_Feedback	BOOL	Machinery running to direction 2 feedback
All	I_r32Load	REAL	Current load in tons
All	I_SpeedFB	INT	Speed feedback scaled between -100...0...+100 (+-100 % = Max speeds)
All	I_i16MaxSpeed	INT	Scale maximum value (Positive max value < 0)
All	I_i16MinSpeed	INT	Scale minimum value (Negative min value < 0)
Hoist	I_i32DesignedBrkStops	DINT	Designed brake stops (Brake life time)
Hoist	I_boHoistOverload_1	BOOL	Hoist overload signal sensor 1
All	I_r32Position	REAL	Machinery position in % of maximum value (S1 maximum)
All	I_BrakeOpen	BOOL	Brake open status bit
All	I_LoadOn	BOOL	Load on status bit
All	I_FaultAct	BOOL	Fault active ON
Hoist*	I_i16MotorTemp	INT	Current Motor temp in Celsius degrees
Any**	I_Common_MC_On	BOOL	TRUE when crane is ON (Drives ON, DynAReg is synchronized)
Any	I_Common_Fault	BOOL	Crane fault active
Any	I_Common_Alarm	BOOL	Crane alarm active
Any	I_Common_Event	BOOL	Crane event active
			*If measured **Common data only needed for one instance of FB4100
	OUTPUT		
	Parameter	**Data Type**	**Description**
	Q_Invalid_machID	BOOL	True if mach_ID parameter does not match any of the supported machineries.
	Q_scf20_ret	INT	SFC20 return value, 0 = OK