Tampereen yliopisto

Jarno Leppänen

HYBRIDIPYÖRÄKUORMAAJAN OHJAUSARKKITEHTUURIN SUUNNITTELU JA TOTEUTUS

Diplomityö
Tekniikan ja luonnontieteiden tiedekunta
Tarkastajat: Yliopistonlehtori Petteri Multanen
Emeritusprofessori Kalevi Huhtala
Joulukuu 2023

TIIVISTELMÄ

Jarno Leppänen: Hybridipyöräkuormaajan ohjausarkkitehtuurin suunnittelu ja toteutus
Diplomityö
Tampereen yliopisto
Konetekniikka
Joulukuu 2023

Päästöjen vähentäminen tulee olemaan yksi työkoneteollisuutta ohjaavista tekijöistä tulevaisuudessa. Sähköistäminen, hybridisaatio sekä vaihtoehtoiset polttoaineet nähdään helposti toteutettavina ratkaisuina päästöjen vähentämiseksi. Tampereen yliopistossa on valmisteilla sähkö–diesel-sarjahybridi hybridityökoneiden tutkimuksen edistämiseksi. Tässä työssä osallistutaan tämän demonstraatiotyökoneen kehittämiseen suunnittelemalla ja toteuttamalla koneen ohjausjärjestelmän arkkitehtuuri. Työn tavoitteita ovat fyysisen arkkitehtuurin muodostaminen ja ohjelmistomoduulien määrittäminen työssä valittaville ohjelmoitaville logiikoille sekä niiden kommunikaatiorajapintojen määrittäminen. Lisäksi työssä on tarkoitus dokumentoida menetelmä ohjausalgoritmien mallipohjaiselle kehittämiselle kohdejärjestelmällä. Työssä ei keskitytä ohjausalgoritmien kehittämiseen tai testaamiseen. Työ jakaantuu kahteen osaan. Kirjallisuustutkimusosa käsittelee systeemi- ja ohjausjärjestelmäsuunnittelun kehitysprosesseja, ohjausjärjestelmän arkkitehtuuriin valintaan vaikuttavia tekijöitä sekä CAN-väylää ja sen kahta ylemmän tason standardia: J1939 ja CANopen. Tämän jälkeen toteutusosassa toteutetaan työn tavoitteet ja esitetään lopuksi jatkokehitystarpeet.

Systeemisuunnittelun näkökulmasta työssä esitellään mekatronisten järjestelmien kehitysprosessina VDI 2206 -viitemalli. Arkkitehtuurin suunnittelussa pyritään noudattamaan viitemallin käytäntöjä esimerkiksi tekemällä vaatimusten määrittelyprosessi mahdollisimman perusteellisesti. Suurimmaksi haasteeksi muodostui vaatimusten yksityiskohtainen määrittäminen, johon viitemallin mukainen päätöksenteko nojaa. Arkkitehtuuriin liittyen varsinkin ohjelmoitavien logiikkojen laskentatehoille vaatimusten asettaminen oli haastavaa, koska järjestelmällä halutaan testata erilaisia ohjausalgoritmeja. Ohjausjärjestelmäsuunnittelun näkökulmasta työssä esitetään mallipohjaisen työkierron prosessi. Prosessimallin SIL-, HIL- ja IIL- testausmenetelmistä koettiin olevan merkittävä hyöty kommunikaatiorajapintojen ohjelmoinnissa.

Ohjausjärjestelmän arkkitehtuurin valintaa eniten ohjaaviksi tekijöiksi löydettiin laskentatehovaatimus ja ohjausjärjestelmään kytkettävien laitteiden määrä, jotka riippuvat puolestaan järjestelmältä halutuista toiminnallisuuksista. Myös modulaarisuus, pilviliitännäisyys sekä toiminnalliseen turvallisuuteen liittyvä lainsäädäntö tunnistettiin valintaan vaikuttaviksi tekijöiksi. Perustavanlaatuiseksi valinnaksi arkkitehtuurisuunnittelussa todettiin hajautusasteen määrittäminen. Verrattuna ajoneuvoteollisuuden käyttämiin Ethernet-pohjaisiin tiedonsiirtoratkaisuihin, työkoneiden ohjausjärjestelmän arkkitehtuurivaihtoehtoja rajaa lisäksi CAN-väylän rajallinen baudinopeus. CAN-väylää käytettäessä hajauttamista kannattaakin lähestyä hajauttamalla sellaisia toimintokokonaisuuksia, joiden välillä ei tarvita merkittävästi tiedonsiirtoa.

Demonstraatiotyökoneeseen valittu ohjausarkkitehtuuri ei edusta hajautusasteen kumpaakaan ääripäätä. Tutkimuksen kannalta oleelliset toiminnallisuudet päätettiin keskittää ja loput hajauttaa. Päätöstä ohjasi ajatus siitä, että tutkimuksenaikainen ohjelmistokehitys voidaan toteuttaa yhdelle logiikalle, eikä kommunikointirajapintoja tai muita logiikoita tarvitse muuttaa. Tällöin muutosten tekeminen on nopeaa, eivätkä muutokset myöskään riko yhtä helposti muita toiminnallisuuksia. Koska laskentateholle ei voitu asettaa vaatimusta, pyrittiin arkkitehtuurista lisäksi tekemään mahdollisimman modulaarinen. Arkkitehtuurissa CAN-väyläliitännäiset komponentit yhdistettiin ohjausjärjestelmään käyttäen kuutta eri CAN-väylää. Väylien määrään vaikutti erityisesti hybriditopologian inventterien vaatima kaista, mutta myös joidenkin komponenttien baudinopeudet, joita käyttäjä ei voinut valita. Mikäli baudinopeudet olisi voitu asettaa kaikille laitteille samaksi, ei laitteiden ylempi CAN-standardi olisi rajoittanut komponenttien sijoittelua samaan väylään työssä esitettyjä muutamia huomioita lukuun ottamatta. Demonstraatiotyökoneen valmistuttua arkkitehtuurin validointi tulee viedä loppuun muun muassa tarkistamalla väylien kuormitustasot.

Avainsanat: Ohjausjärjestelmäarkkitehtuuri, hybridityökone, VDI 2206, CANopen, J1939

ABSTRACT

Jarno Leppänen: Control architecture design and implementation for a hybrid wheel loader
Master's Thesis
Tampere University
Mechanical engineering
December 2023

In the future, reducing emissions will be one of the guiding principles in the off-road working machinery industry. Electrification, hybridization, and alternative fuels are seen as the most feasible solutions for reducing emissions. To contribute to the research of hybrid machines, an electric-diesel series hybrid is being built at the University of Tampere. In this thesis, control system architecture is defined for this demonstration machine. In detail, the objectives are to form a physical architecture and define software modules for PLCs which are also selected. In addition, PLC communication interfaces are defined and model-based development process for developing control algorithms for PLCs is documented. However, development of control algorithms is not focused. This thesis is divided into two parts. The literature review forms a base for the development process by investigating models for control system design but also for mechatronics system design. In addition, factors affecting the architecture are investigated along with the CAN bus and its two higher-level standards: J1939 and CANopen. After this, the objectives are reached in the implementation part and finally, the needs for further development are presented.

From the perspective of mechatronics system design, VDI 2206 reference model is presented. In accordance with the ideology of this model, the architecture design process aims to fundamentally follow the practices of the model in defining requirements. The biggest challenge was a precise definition of requirements, on which decision-making according to the reference model relies. Especially in relation to the architecture, it was challenging to define a requirement for the computational resources of PLCs when differing control algorithms are to be tested with the machine. From the perspective of control system design, this study presents a process model for the model-based design. In the communication interfaces programing phase, the SIL, HIL and IIL testing methods of model-based design were found to bring significant benefits for the process.

The selection of the control system architecture was found to be most influenced by the computational requirements and the number of devices connected to the control system, which in turn depend on the desired system functionalities. Modularity, cloud connectivity and legislation related to functional safety were also identified as factors affecting the selection. The fundamental choice in architecture design was found to be the determination of the degree of distribution. Compared to Ethernet-based data transfer solutions used in the automotive industry, the limited baud rate of the CAN bus limits the architectural options of the control system of off-road working machinery. When using the CAN bus, decentralization should be approached by decentralizing such functional entities which have no need for significant data transfer from one to another.

The control architecture selected for the demonstration machine does not represent either extreme of distribution degree. The essential functionalities for the research were implemented as centralized and the rest as distributed. The decision was based on the idea that software development during the research can be carried out for one PLC and changes are not needed for communication interfaces or other PLCs. Therefore, changes can be made quickly, and changes do not break other functionalities as easily. In addition, since a requirement for computational resources could not be set, the architecture was made as modular as possible. In the architecture, components were connected to the control system using six different CAN buses. The number of buses was particularly affected by the bandwidth required by the inverters of the hybrid topology, but also by the baud rates of some components, which the user could not choose. If the baud rates could have been set to be similar for all devices, the higher CAN standard of the devices would not limit the placement of components on the bus topology, excluding a few observations presented in the study. After the demonstration machine is built, the architecture must be validated by checking the load levels of the buses, among other things presented in the study.

Keywords: Control system architecture, hybrid working machine, VDI 2206, CANopen, J1939

ALKUSANAT

Tämä diplomityö on tehty osana Business Finlandin Clean Propulsion Technologies Co-Innovation -yhteishanketta, jonka tähtäimenä on löytää uusia energiatehokkaampia ja päästöttömämpiä ratkaisuja niin meriliikenteeseen kuin liikkuviin työkoneisiin samalla säilyttäen Suomen asema voimansiirtoratkaisuihin liittyvillä markkinoilla. Projektissa on mukana Teknologian tutkimuskeskus VTT:n lisäksi yrityksiä ja yliopistoja. Tarkemmin tämä työ liittyy projektin neljänteen ja viidenteen työpakettiin edesauttamalla ohjausjärjestelmiin liittyvää tutkimusta sekä hybriditeknologiaa hyödyntävän demonstrointityökoneen valmistumista Tampereen yliopistolla.

Haluan kiittää työn tarkastamisesta sekä mahdollisuudesta osallistua projektiin Petteri Multasta sekä Kalevi Huhtalaa. Kiitos kuuluu myös Jyrki Tammistolle isosta panoksesta demonstraatiotyökoneen mekaniikka- ja järjestelmäsuunnitteluun sekä tuesta ohjausjärjestelmän arkkitehtuurisuunnitteluun. Lopuksi haluan kiittää IHA-tutkimusryhmää, perhettäni sekä opiskelutovereitani kannustuksesta opintojeni ja diplomityön aikana.

Tampereella, 20.12.2023

Jarno Leppänen

SISÄLLYSLUETTELO

KUVALUETTELO

LYHENTEET JA MERKINNÄT

ACK	engl. Acknowledge, CAN-viestikehyksen parametri, jolla viesti kuitataan lähteneen väylälle, jonkin väylällä olevan laitteen toimesta
ASCII	engl. American Standard Code for Information Interchange, Merkistö, jolla voidaan esittää amerikanenglannissa käytetyt kirjaimet, numerot, välimerkit ja eräät erikoismerkit binäärilukujen avulla
BMS	engl. Battery management system, Useampi kennoisen akkumoduulin sisältä löytyvä piirilevy, joka suojaa ja tasapainottaa kennoja.
CAN-väylä	engl. controller area network, tyypillisesti ajoneuvoissa ja työkoneissa käytetty automaatioväyläratkaisu
CAN FD	engl. controller area network with Flexible Data-Rate, versio CAN-väylästä joka sallii muuttuvan baudinopeuden ja suuremman hyötykuorman
CAN XL	engl. controller area network with extra large dataframes, työn kirjoitushetkellä uusin versio CAN-väylästä, joka mahdollistaa CAN FD suuremman datansiirtonopeuden
COB-ID	engl. Communication object identifier, CANopen viestin identifiointikenttä
CiA	engl. Can in automation, CAN-väylän käyttäjäorganisaatio
CIL	engl. customer in the loop, testausmenetelmä, jossa asiakas otetaan mukaan kenttätesteihin
CRC	engl. Cyclic redundancy check, viestinnässä käytetty menetelmä vastaanotettavan datan oikeellisuuden varmentamiseksi vertaamalla viestin mukana tullutta arvoa laskettuun arvoon
DA	engl. Destination address, J1939-standardin ykköstyypin viestissä käytettävä parametri, joka määrittää viestin vastaanottavan laitteen solmunumeron
DLC	engl. Data Length Code, CAN-viestikehyksessä oleva parametri, jolla määritetään, kuinka monta tavua dataa viesti sisältää
DP	engl. Data page, Yksi J1939-standardin parametriryhmänumeroa määrittävä parametri, jonka tarkoitus on laajentaa parametriryhmänumeroavaruutta
EDS	engl. electronic data sheet, CANopen standardiin liittyvä tiedosto, joka pitää sisällään laitteen objektikirjaston ja laitteen toimintaan liittyvät viestirakenteet
EIL	engl. Enviroment in loop, testausmenetelmä, jossa valmiin laitteen ohjausjärjestelmää testaan sen oikeassa toimintaympäristössä

EMCY	engl. Emergency-object, hätätilanneobjekti, jolla viestitiään viallisesta toiminnasta CANopen standardissa
GE	engl. Group Extension, J1939-standardin kakkostyypin viestin identifiointiosassa käytettävä parametri laajentamaan parametriryhmänumeroavaruutta
GECMS	engl. global equivalent consumption minimization strategy, hybridivoimalinjan ohjausstrategia, jossa mallin avulla määritetään tilanteeseen pienimmän polttoaineenkulutuksen mahdollistava ohjaus.
HIL	engl. hardware in the loop, ohjausjärjestelmän alustava testausmentelmä, jossa simulaatioon lisätään todellisia komponentteja
HVIL	engl. high voltage interlock loop, virtapiiri laitteen korkeajännitekomponenttien läpi, jonka jatkuvuudella varmistetaan korkeajänniteliittimien paikallaolo ennen jännitteen kytkemistä
IDE	engl. identifier extension, CAN-viestikehyksen parametri, jolla määritetään, onko viestikehys normaali vai jatkettu
IFS	engl. Interframe spacing, kolme väistyvää bittiä sisältävä välimerkki kahden CAN-väylälle peräkkäin lähetettävän viestin välillä
IIL	engl. iron in loop, ohjauskoodin testausmenetelmä, jossa ohjauskoodi käännetään ohjainlaitteille ja toimintaa testataan testijärjestelyssä, jossa on jo suurin osa järjestelmään tulevista komponenteista
IO-lisälaite	engl. Input/Output board, ohjelmoitavaan logiikkaan – usein tiedonsiirtoratkaisun avulla – kytkettävä komponentti, johon voidaan kytkeä analogisia ja digitaalisia lähtöjä ja tuloja vaativia komponentteja
ISO	engl. International Organization for Standardization, kansainvälinen standardisoimisjärjestö
LFSR	engl. Linear Feedback Shift Register, tietokone yhteensopiva laskentamenetelmä, jota voidaan käyttää esimerkiksi CRC-laskentaan
MBD	engl. Model based design, (ohjausjärjestelmän) mallipohjainen suunnittelu
MIL	engl. Model in the loop, ohjausalgoritmin alustava testausmenetelmä, joka perustuu järjestelmästä tehdyn mallin simuloimiseen
NMT	engl. Network management service, CANopen standardin palvelu, jolla monitoroidaan ja ylläpidetään laitteiden tilakonetta
NRZ	(engl. non-return-to-zero) Linjakoodauksen muoto tiedonsiirrossa, jossa ei ole kolmatta niin sanottua lepotilaa bitin tilojen lisäksi
OBD	engl. On board diagnostic, standardoitu diagnostiikka rajapinta ajoneuvoissa ja työkoneissa
OD	engl. object dictionary, CANopen laitteen parametrikirjasto

OSI-viitemalli	engl. Open Systems Interconnection model, yksi tiedonsiirtoprotokollien määrittämiseen käytetty viitemalli
PDO	engl, Process data object, CANopen standardissa prosessiin liittyvän datan lähettämiseen tarkoitettu viestityyppi
PDU	engl. power distribution unit, komponentti, joka yhdistää korkeajännitekomponentit akustoon ja valvoo järjestelmän sähköturvallisuutta tai engl. Protocol data unit, PDU, jolla viitataan J1939 CAN-standardin viestikehykseen
PF	engl. Protocol data unit format, J1939-standardin parametriryhmännumeroa määrittävä osa, jolla esitetään viestin tyyppi
PFCS	engl. Power follower control strategy, Hybridivoimalinjan ohjausstrategia, jossa dieselmoottorilla tuotetaan suuret tehot ja akustolla pienemmät
PG	engl. Parameter group, J1939 standardissa parametri kokonaisuus, joka on aseteltu yhden tai useamman viestikehyksen datakenttiin tietyllä tavalla
PGN	engl. Parameter group number, J1939-viestin identifiointikenttään tuleva parametriryhmän tunniste, joka koostuu viestikehyksen tyypistä ja sen määrittelemästä tyyppikohtaisesta parametrista.
PIL	engl. processor in loop, Ohjauskoodin testausmenetelmä, jossa ohjauskoodi käännetään virtuaaliselle ohjainlaitteelle ja sen toimintaa simuloidaan osana järjestelmää
PLC	engl. Programmable logic controller, ohjelmoitava logiikka
PS	engl, Protocol data unit specific, J1939-standardin parametriryhmännumeroa määrittävä osa, jolla esitetään vastaanottavan laitteen solmunumero tai parametriryhmän laajennusnumero riippuen viestin tyypistä (ks. DA ja GE)
PWM	engl. Pulse width modulation, modulointimenetelmä signaalin tehon säätämiseen, joka perustuu signaalin potentiaalin jatkuvaan muuttamiseen 0V ja maksimiarvon välillä
RTI	engl. Real-Time Interface, rajapinta käyttäjällä reaaliaikasimulaatiossa
RTR	engl. remote transmission request, CAN-viestikehyksen parametri, jolla määritetään, onko viesti datapyyntö vai -lähetys
SA	engl. Source Address, J1939-viestikehyksen identifiointikentän osa, josta käy ilmi viestin lähettävän laitteen solmunumero
SCS	engl. Supervisory control system, Ohjausjärjestelmä, joka perustuu alemmalla tasolla olevien ohjaimien valvontaan ja ohjaamiseen
SDO	engl. Service data object, CANopen laitteen parametrikirjaston parametrien arvojen vaihtamiseen tarkoitettu viestityyppi

SIL	engl. Softaware on loop, ohjauskoodin testausmenetelmä, joka perustuu koodin kääntämiseen ja simulointii kehitysympäristössä
SOC	engl. state of charge, sähköisen energianvarastointiratkaisun varaus prosenteina
SPN	engl. Suspect parameter number, SPN, J1939-standardin standardoitujen viestirakenteiden parametrien tunnistamiseen käytettävä numero
SSR	engl. Substitute extension, CAN-standardin jatketussa viestikehyksessä oleva merkityksetön bitti, jolla kohdistetaan RTR-kenttä normaalin ja jatketun viestikehyksen välillä.
SYNC	engl. Synchronization object, CANopen standardissa esimerkiksi PDO-viestien tahdistamiseen käytetty viestityyppi
TCP/IP-pino	engl. Internet protocol suite, internet pohjaisessa tiedonsiirrossa käytettävien protokollien kokoelma
TCS	engl. Thermostat control strategy, Hybridivoimalinjan ohjausstrategia, jossa dieselmoottori sammutetaan, mikäli sitä ei voi käytetä vain yhdessä – optimaalisimmassa – toimipisteessä.
TP	engl. Transport protocol, J1939 standardin siirtoprotokolla, joka mahdollistaa parametriryhmien lähettämisen, jotka eivät mahdu yhteen viestiin. Siirtoprotokolla sisältää TP.CM ja TP.DT viestit
TP.CM	engl. Transport protocol Connection management, viesti, joka avaa J1939 standardissa useammalla viestillä siirrettävän parametriryhmän lähetyksen
TP.DT	engl. Transport protocol Data transfer, TP.CM viestiä seuraavat datapakettien lähettämiseen käytetyt viestit J1939-standardissa
VDI	saksaksi Verein Deutscher Ingenieure, Saksan insinöörien liitto

i	virta
L	induktanssi
R	resistanssi
t	aika
u	jännite
SOC_L	PFCS algoritmissa akuston varaustason minimiarvo
SOC_M	PFCS algoritmissa akuston varaustason kynnysarvo
SOC_U	PFCS algoritmissa akuston varaustason maksimiarvo
τ	aikavakio

1. JOHDANTO

Tietoisuus polttomoottorien ympäristövaikutuksista ja sen pohjalta säädetty päästölainsäädäntö ovat ajaneet liikkuvaa kalustoa valmistavaa teollisuutta kohti päästöttömämpiä ratkaisuja jo parinkymmenen vuoden ajan. Tänä päivänä esimerkiksi autoteollisuuden valmistajat, kuten Ford ja Volvo, ovat asennoituneet korvaamaan Euroopan alueelle valmistettavien henkilöautojen polttomoottorit täysin päästöttömillä – usein sähköisillä – ratkaisuilla vuoteen 2035 mennessä siitä huolimatta, ettei ajoneuvojen päästölainsäädännöstä ole päästy sopuun (Transport & Environment 2022). Asenteiden muutos ja autoteollisuuden näyttämä tie luo painetta myös työkonevalmistajille päästöjen vähentämisestä (Clean Propulsion Technologies 2023).

Toisin kuin autoteollisuudessa, työkoneteollisuudessa täysin sähköiset ratkaisut eivät nykyisellä teknologialla näytä soveltuvan kuin harvoihin sovelluksiin. Tällaisia ovat esimerkiksi rakennetulla alueella kerrallaan suhteellisen lyhyitä aikoja toimivat pienet maanrakennuskoneet sekä rajatulla alueella jatkuvasti toimivat koneet, jotka voivat vaihtaa akuston työsyklin aikana (ks. Volvo 2022 & Sandvik 2022). Haasteita aiheutuu nykyisten sähköenergian varastointiratkaisuiden, kuten litiumioniakkujen, huonosta energiatiheydestä verrattuna polttoaineeseen ja korkeasta hinnasta, huonosta lataussyklien kestosta sekä pitkistä latausajoista (Koohi-Fayegh & Rosen 2020). Uusia sähköenergian varastointiratkaisuja on kuitenkin kehitteillä (ks. Shaibani 2020), jotka varmasti parantavat tulevaisuudessa täysin sähköisten voimansiirtolinjojen soveltuvuutta työkoneisiin. On kuitenkin mahdollista, ettei ainoana tehonlähteenä sähköenergianvarastointiratkaisuun nojautuva voimansiirtolinja ole ikinä järkevä ratkaisu sellaisiin työkoneisiin, jotka toimivat sähkönjakeluverkon ulottumattomissa.

Lyhyellä aikavälillä vaihtoehtoiset polttoaineet polttomoottoreille ja dieselmoottorista sekä sähköenergianvarastointiratkaisusta koostuva hybridivoimalinja nähdään järkevimpinä ratkaisuna työkoneiden päästöjen vähentämisessä (Clean propulsion Technologies 2023). Tiivistettynä tällaisten sähkö–dieselhybridien hyöty verrattuna perinteiseen toteutukseen perustuu kykyyn ottaa energiaa talteen sähköenergianvarastointiratkaisuun esimerkiksi jarruttamisen tai kuormanlaskemisen aikana sekä mahdollisuuteen optimoida dieselmoottorin käyttö paremman hyötysuhteen alueelle, kun hetkellisiin pyörimisnopeuden muuttumiseen tehopiikin myötä ei tarvitse varautua. (Nokka 2018) Tällaisista työkoneista ja niillä saavutetuista hyödyistä hyvinä esimerkkeinä ovat Ponssen (2022) EV1

kuormatraktorikonsepti sekä Volvon (2017) ja Hitachin ZW220HYB-5B pyöräkuormaajakonseptit (Ishida & Higurashi 2015).

Koska hybridityökoneet ovat kuitenkin merkittävästi kalliimpia kuin perinteiset työkoneet, täytyy niillä olla suhteellisen lyhyt takaisinmaksuaika, jotta niiden hankinta on loppukäyttäjälle kannattavaa ja markkinavetoista. Yhtenä tekijänä takaisinmaksuaikaan vaikuttaa polttoainekuluissa säästäminen järjestelmän energiatehokkuuden kautta, johon puolestaan voidaan vaikuttaa hybridikomponenttien mitoittamisella, mutta myös säätöalgoritmien kehittämisellä. (Immonen 2013; Tupitsina et. al 2019) Tällä hetkellä yleisenä haasteena alan tutkimuskirjallisuudessa on, että harva mitoitustyökalu ja säätöalgoritmi on pystytty verifioimaan todellisella työkoneella (ks. esim. Immonen 2013; Tupitsina et. al 2019).

Tässä diplomityössä osallistutaan uuden sWille-tutkimusalustan (myöhemmin demonstraatiotyökone) kehittämiseen, jolla on tarkoitus verifioida aikaisempia tutkimuksia ja edesauttaa uusien ohjausalgoritmien tutkimusta. Tavoitteena on luoda demonstraatiotyökoneen ohjausjärjestelmän fyysinen ja ohjelmallinen arkkitehtuuri, ohjelmoida kommunikaatiorajapinnat ohjelmoitaville logiikoille sekä dokumentoida menetelmä ohjausalgoritmien mallipohjaiselle kehittämiselle kohdejärjestelmällä. Tarkemmin fyysinen arkkitehtuuri pitää sisällään ohjelmoitavien logiikkojen valinnan ja niiden integroinnin ohjausjärjestelmään. Ohjelmallinen arkkitehtuuri puolestaan pitää sisällään ohjelmistomoduulien rajapintojen määrittämisen ja niiden allokoinnin ohjelmoitaville logiikoille. Työn ulkopuolelle rajataan ohjausalgoritmien kehitys ja testaus. Työn päätutkimuskysymys on ”Millainen ohjausjärjestelmäarkkitehtuuri demonstraatiotyökoneeseen tulisi valita?”

Työssä käytetään konstruktiivista tutkimusotetta. Lukan (2001) mukaan konstruktiivisessa tutkimusotteessa pyritään luomaan konkreettinen lopputulos soveltaen aihealueen teoriaa. Tutkimusotteelle tyypillisesti tutkimuksen tavoite on selvä, mutta selvää tapaa sen saavuttamiseksi ei ole. Lopputulos on usein myös iteroituva ja vaatii kohdejärjestelmän testaamista, jotta lopputulos voidaan validoida. (Lukka 2001) Työ koostuu kuudesta luvusta, joista johdannon jälkeen kolme seuraavaa lukua käsittelevät teoriaa implementoinnin tueksi. Luvussa kaksi käsitellään kehitysprosessia, niin järjestelmäsuunnittelun kuin ohjausjärjestelmä osalta. Luvussa kolme puolestaan tutkitaan, millaisia ohjausjärjestelmän arkkitehtuureita on olemassa, ja mitkä tekijät vaikuttavat ohjausarkkitehtuurin valintaan. Luvussa neljä perehdytään CAN-väylän (engl. controller area network, CAN-bus) toimintaan tiedonsiirtoratkaisuna, sillä ymmärrystä sen toiminnasta tarvitaan kommunikaatiorajapintojen toteuttamiseksi.

Teoriaosuudessa pyritään vastaamaan apututkimuskysymyksiin:

1. Kuinka demonstraatiotyökoneen ohjausjärjestelmän kehitystä tulisi lähestyä ja viedä eteenpäin, niin järjestelmäsuunnittelun kuin ohjausjärjestelmäsuunnittelun kannalta?
2. Mitkä tekijät vaikuttavat ohjausjärjestelmän arkkitehtuurin valintaan?
3. Mitä rajoituksia CAN-väylä asettaa ohjausarkkitehtuurille ja voidaanko eri CAN-pohjaisia tiedonsiirtoratkaisuja käyttäviä laitteita laittaa samaan tiedonsiirto-väylään?

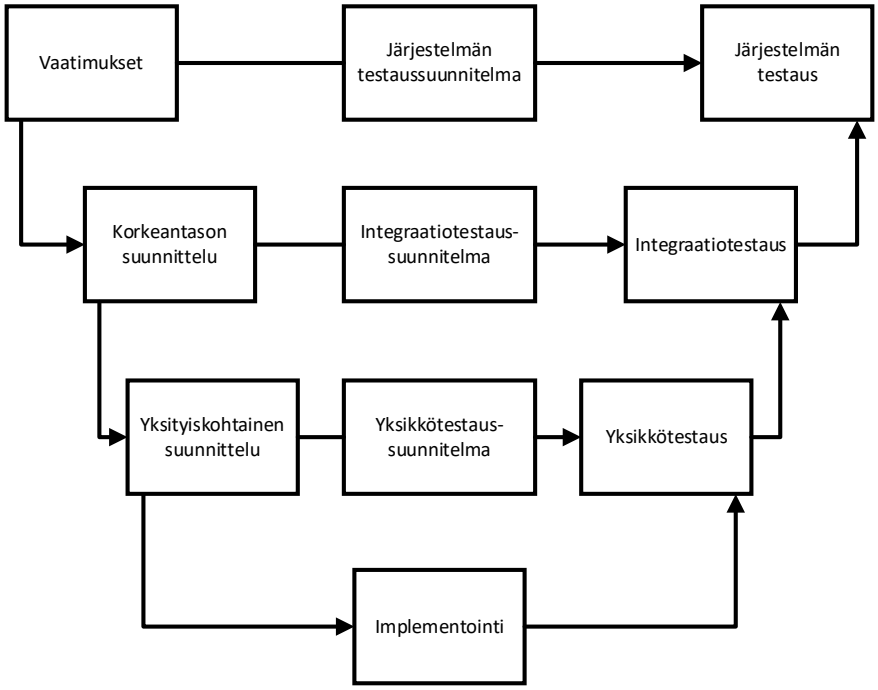
Teoriaosuuden jälkeen viidennessä luvussa toteutetaan työn tavoitteet, jota seuraavat työn johtopäätökset luvussa kuusi.

Ensimmäiseen ja toiseen apututkimuskysymykseen vastaamiseksi aineistona käytetään pääsääntöisesti tieteellisiä julkaisuja, mutta myös standardeja ja komponentti- sekä työkonevalmistajien internetlähteitä. Kolmanteen apukysymykseen vastaamiseksi käytetään puolestaan CAN-väylään liittyvää oppikirjaa ja aihealueen standardeja. Viimeiseksi työn implementaatiovaiheessa lähtökohtana käytetään pääsääntöisesti demonstraatiokoneeseen hankittujen komponenttien teknisiä manuaaleja, aikaisempaa suunnittelua ja ohjelmoitavien logiikoiden ohjelmointiin liittyvää dokumentaatiota.

2. KEHITYSPROSESSI

Ajansaatossa kehitysprosesseja on alettu optimoida, ja lopputuloksena on syntynyt erilaisia prosessimalleja. Prosessimalleilla yritetään tyypillisesti kuvata, mitä tulisi tehdä, jotta riski projektin epäonnistumiselle pienenee. Prosessimalleja on luotu useita erilaisia ja niitä on muokattu erilaisiin kehitysprojekteihin, käyttötapauksiin sekä vastaamaan ajankohtaisiin ongelmiin. Usein ne kuvaavat prosessia tietystä näkökulmasta, jonka vuoksi niiden käyttäminen ilman systeeminsuunnittelun perusteiden ymmärtämistä voi olla rajoittunutta. (Blanchard & Blyler 2016, s. 44–49)

Yksi usein käytetty prosessimalli on muotonsa mukaan nimetty ja kuvassa 1 esitetty V-malli. Mallin perusperiaate on lähteä liikkeelle V-kirjaimen vasemman sakaran yläreunasta projektin määrittelyvaiheiden läpi kohti implementointivaihetta ja tämän jälkeen päätyä verifiointi- ja testausvaiheiden kautta mallin oikean sakaran yläreunaan. Mallissa testaussuunnitelmat määritetään projektin määrittelyvaiheiden aikana, jolloin itse testausvaiheilla ainoastaan toteutetaan testaussuunnitelma. Koska mallissa vaiheet tulee suorittaa aina valmiiksi ennen seuraavaan siirtymistä, voidaan malli nähdä systemaattisena ja helposti omaksuttavana. (Munassar & Govardhan 2010) Tarkoituksena on siis määrittää koko projekti täysin ennen kuin implementointi aloitetaan. Tämän taustalla on muun muassa idea siitä, että tuotteen koko elinkaarikustannuksista suurin osa määritetään erittäin aikaisessa vaiheessa kehitysprosessia, jolloin projektin määrittelyyn ja varsinkin vaatimuksien muodostamiseen tulisi käyttää myös suhteellisen iso osa ajasta (Blanchard & Blyler 2016, s.29–33, 38).



Kuva 1. *Perinteinen V-malli (Munassar & Govardhan 2010, muokattu)*

Tällaisen perinteisen sekventiaalisen lähestymistavan yksi haasteista on se, ettei se tarjoa mahdollisuutta tehdä aikaisia prototyyppejä. Aikaiset prototyypit voivat olla hyödyllisiä implementoitavaan järjestelmään liittyvässä tiedonhankinnassa eli esimerkiksi vaatimuksien tarkentamisessa. Lisäksi V-malli ei tarjoa yleensä selvää tapaa ratkaista tilanteita, joissa projektin testausvaiheissa huomataan määritettyjen vaatimusten olevan puutteelliset tai tilanteita, joissa määritetyt vaatimukset muuttuvat projektin aikana. Haasteiden vuoksi mallia on helpompi soveltaa pienemmille ja lyhyemmille projekteille. (Munassar & Govardhan 2010)

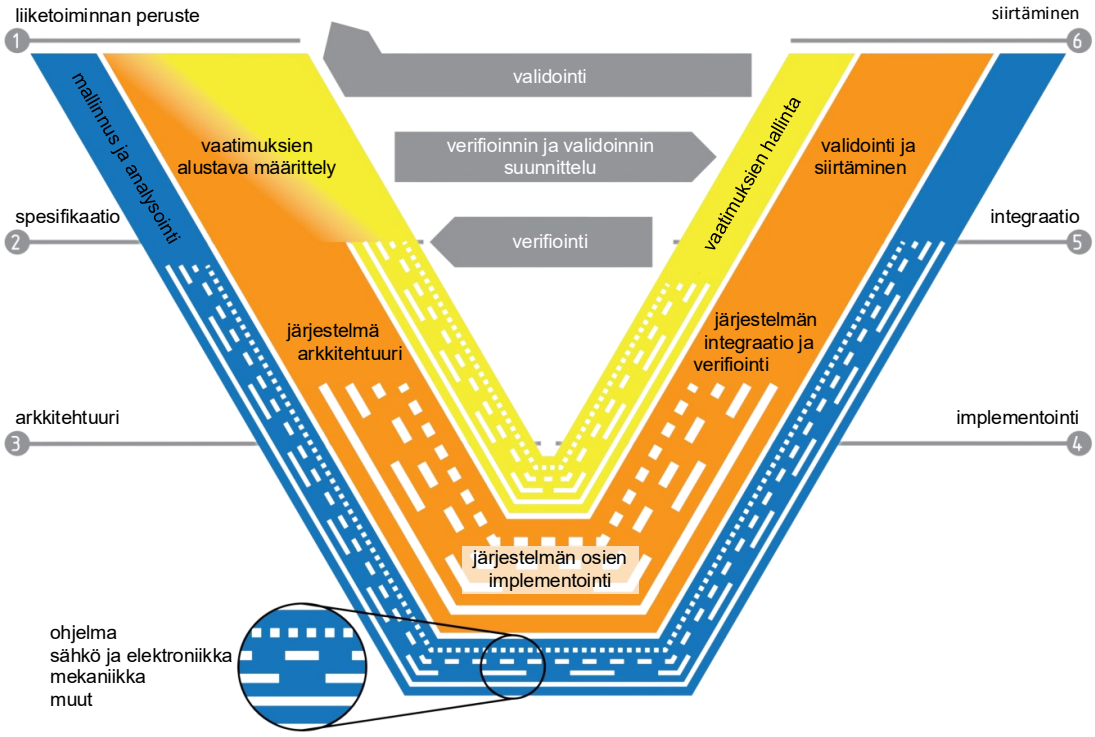
Munassar ja Govardhan (2010) esittelevät julkaisussaan myös muita malleja, kuten vesiputous-, iteraatio- ja spiraalimallin, mutta malleista V-malli lienee olevan parhaiten sovellettavissa ohjausjärjestelmäsuunnitteluun, sillä sitä on sovellettu aihealueen julkaisuissa, kuten esimerkiksi Ahopellon (2019, s. 21) liikkuvien työkoneiden ohjausjärjestelmiin liittyvässä väitöskirjassa tai Skjetnen ja Egelandin (2006) laivojen ohjausjärjestelmiä käsittelevässä julkaisussa. Myös esimerkiksi Forrai (2013, s. 28–30) esittelee sen avulla kehitysprosessin sulautettujen ohjausjärjestelmien suunnittelua koskevassa oppikirjassa sekä saksan insinööriliitto VDI (saksaksi Verein Deutscher Ingenieure, 2021) on luonut siitä viitemallin ja sitä käsittelevän julkaisun 2206:2021 mekatroniikan kehitysprojekteihin, johon myös ohjausjärjestelmän suunnittelu luonnollisesti kuuluu.

Näistä malleista VDI:n 2206:2021 viitemalli antaa laajemman kuvan tuotekehitysprosessista ja muut mallit keskittyvät enemmän ohjausjärjestelmäsuunnitteluun. Koska nykyai-

kaisissa ja varsinkin pienemmissä projektiorganisaatioissa ohjausjärjestelmäsuunnittelusta vastaava voi joutua enenevissä määrin osallistumaan systeemisuunnitteluun, kokonaiskuvan eli muun muassa myös talousnäkökulman sekä vaatimuksien määrittely- ja hallintaprosessin ymmärtäminen perinteisen kehitysprosessin lisäksi on tärkeää. Edellä mainitusta syystä tässä työssä kuvataan ensin V-mallin mukainen kehitysprosessi pohjautuen VDI:n 2206:2021 viitemalliin. Tämän jälkeen luvussa 2.2 keskitytään ohjausjärjestelmäsuunnittelun ominaispiirteisiin eri kehitysprosessin vaiheissa.

2.1 Systeemisuunnittelun näkökulma mekatroniikan projektiin

Kuvassa 2 esitetyn VDI:n (2206:2021) viitemallin periaate on sama kuin edellä esitetyn perinteisen V-mallin, mutta sitä on kehitetty vastaamaan paremmin nykyaikaista kehitysprojektia. Viitemalli on luotu analysoimalla aikaisempien V-mallien vahvuuksia ja heikkouksia ja pyrkimällä etsimään ratkaisuja aikaisemman VDI:n 2206:2004 viitemallin saamaan kritiikkiin (Graessler et al. 2018; Graessler et al 2020). Merkittäviä muutoksia ovat esimerkiksi pyrkimys esittää vaatimuksien hallinnointi jatkuvana läpi koko projektin sekä projektin todellisen iteratiivisen luonteen esille tuominen. Myös mallinnus- ja analysointimenetelmien hyödyntämistä sekä rinnakkaissuunnittelun eli samaan aikaan tapahtuvaa ohjausjärjestelmä-, sähkö- ja mekaniikkasuunnittelun tärkeyttä on yritetty korostaa.



Kuva 2. *VDI:n V-malli (VDI 2206:2021, muokattu)*

VDI:n 2206:2021 viitemallin mukainen suunnitteluprosessi lähtee liikkeelle liiketoiminnan perusteista pohjautuen organisaation sisäiseen tai ulkoiseen tilaukseen. Tarkoituksena on perustella tarve projektille, sillä luonnollisesti kannattamatonta projektia ei ole järkevää toteuttaa. Tähän kuuluu muun muassa projektin toteaminen taloudellisesti kannattavaksi sekä projektin riskien selvitys riskianalyysin kautta. Tärkeää on myös tutkia, tukeeko projekti organisaation pitkän aikavälin tavoitteita esimerkiksi liiketoimintamallin kautta. Jotta näitä kohtia voidaan ylipäätänsä pohtia, tarvitsee projektin lähtökohdat kuitenkin sopia ja dokumentoida toimittajan ja tilaajaan välillä, mitä voidaan pitää yhtenä onnistuneen tuotekehitysprojektin edellytyksenä. Dokumentoinnin tarkoituksena on tuoda esille muun muassa mikä on projektin haluttu lopputulos, millainen projektin aikataulu on, miten sidosryhmien vastuut jakautuvat ja miten niiden välinen kommunikointi hoidetaan. (VDI 2206:2021) Huomioitavaa on, että toimittajan ja tilaajan ei välttämättä tarvitse olla eri organisaatioissa, vaan projekti, johon viitemallia sovelletaan voi olla myös organisaation sisäinen.

Seuraavassa vaiheessa aloitetaan vaatimusten määrittelyprosessi, jonka lähtökohtana ovat tilaajan halut ja toiveet. Kaikki vaatimukset eivät välttämättä muodostu suoraan tilaajan haluista ja toiveista vaan niihin voi vaikuttaa esimerkiksi lainsäädäntö ja projektin realiteetit, kuten aikataulu tai budjetti. Lisäksi useinkaan tilaajan halut ja toiveet eivät ole sellaisenaan täydellisiä, ja ne voivat jopa muuttua prosessin aikana. Vaatimuksien määrittelyprosessi nähdäänkin tyypillisesti interaktiivisena tilaajan ja toimittajan välillä kuin myös iteratiivisena ja läpi koko kehitysprosessin kestävänä. Tämä on kuvattu VDI:n 2206:2021 viitemalliin jatkumona vaatimuksien alustavasta määrittelyvaiheesta vaatimuksien hallintavaiheeseen. Näistä vaatimuksien alustavassa määrittelyvaiheessa vaatimuksien määrittelyprosessi on tarkoitus saada sellaiseen vaiheeseen, että järjestelmän arkkitehtuuri voidaan luoda ja hallintaprosessissa puolestaan vaatimuksia on tarkoitus muokata ja tarkentaa arkkitehtuurin ja siihen mennessä tehdyn suunnittelun mukaan.

Hyvä vaatimus on itsestään tunnistettu tarve, joka on yksiselitteiselitteinen, realistinen, kvantitatiivinen ja sidottu konkreettiseen asiaan, kuten laitteen osaan tai alijärjestelmään. Vaatimus voi olla tyypiltään joko toiminnallinen tai ei-toiminnallinen, joista jälkimmäiset kuvaavat laitteen toiminnallisuuksiin sitoutumattomia ominaisuuksia, jotka voivat liittyä esimerkiksi luotettavuuteen tai turvallisuuteen. Molempien vaatimusten toteutuminen tulee kuitenkin olla todettavissa projektin lopussa esimerkiksi mittaamalla, eivätkä ne saa olla ristiriidassa muiden vaatimusten kanssa. (VDI 2206:2021)

Vaatimukset on kuitenkin tarkoitus pitää ratkaisusta riippumattomina ennen arkkitehtuurisuunnittelun valmistumista (VDI 2206:2021). Näin varmistutaan siitä, etteivät ensim-

mäiseksi mieleen tulleet suunnitteluratkaisut rajoita arkkitehtuuria. Prosessia onkin todennäköisesti kannattavaa lähestyä ensin muodostamalla ylemmän tason vaatimuksia eli järjestelmää koskevia vaatimuksia, joista voidaan myöhemmässä vaiheessa iteroida tarkempia alijärjestelmä kohtaisia vaatimuksia. VDI 2206:2021 viitemalli mukaan on kuitenkin tärkeää täyttää kaikki vaatimusten ominaisuudet niitä luodessa. Jos parempaa käsitystä ei ole, pitäisi ominaisuuksia arvioida.

Alustavien vaatimusten määrittelyprosessin lopputuloksena on spesifikaatio eli vaatimusten avulla esitetty määritelmä projektin lopputuloksesta. Tämän jälkeen arkkitehtuurivaiheessa on tarkoitus määrittää spesifikaation pohjalta laiteen yleinen rakenne eli millaisia alijärjestelmiä tuotteeseen on tulossa ja millaiset niiden väliset rajapinnat, kuten sähköiset, hydrauliset ja mekaaniset rajapinnat, ovat (VDI 2206:2021). VDI:n 2006:2021 viitemallin ehdottama keino arkkitehtuurin valitsemiselle on eri arkkitehtuurivaihtoehtojen luominen ja näistä yhden valitseminen eri suunnitteluryhmien kanssa yhdessä määritellyillä kriteereillä.

Perusperiaate arkkitehtuurin suunnittelulle on muodostettujen funktionaalisten vaatimusten avulla erilaisten pääpiirteisen toteutustapojen miettiminen ja niiden allokointi loogisiin kokonaisuuksiin, joista pystyttäisiin muodostamaan alijärjestelmiä eli moduuleja. Arkkitehtuurin muodostamisessa kannattaa lisäksi tutkia, voitaisiinko aikaisempiin projekteihin tehtyjä moduuleja hyödyntää pienin muutoksin. Moduuleissa itsessään on puolestaan järkevää pyrkiä hyödyntämään valmiita komponentteja tai joissain tapauksessa suunnittelemaan uusia, jotka pystyvät toteuttamaan useampia vaatimuksia. Tämä johtaa siihen, että moduuleissa on vähemmän mahdollisesti hajoavia osia, jolloin luotettavuus kasvaa. Lisäksi moduulien asentaminen vie usein vähemmän tilaa ja aikaa. Huomioitavaa kuitenkin on, että tällaisten integroitujen komponenttien käyttäminen tai valmistaminen vaatii tyypillisesti enemmän teknillistä osaamista ja työtä verrattuna yksinkertaisten komponenttien käyttämiseen, ja että tulevaisuudessa syntyneiden rajapintojen muuttaminen voi olla haastavaa ja kallista. (VDI 2206:2021)

Kun arkkitehtuuri on saatu valmiiksi, sen luomat rajoitukset on tarkoitus kuvata vaatimuksiin päivittämällä jo olemassa olevia vaatimuksia, jotta ne vastaavat hyvän ominaisuuden kriteerejä, mutta myös iteroida uusia alemman tason vaatimuksia referenssiksi yksityiskohtaiseen suunnitteluun. Tästä eteenpäin projektin onnistumisen kannalta onkin tärkeää hallita vaatimuksia eli pitää ne ajan tasalla. Mikäli aikaisemmin määritettyihin vaatimuksiin ei syystä tai toisesta päästä, tulee suunnitteluristiriitojen välttämiseksi tarvittavien muutoksien vaikutusta arkkitehtuuriin ja jo tehtyyn suunnitteluun tutkia vaikutusanalyysin kautta.

Arkkitehtuurin valmistumisen jälkeen yksityiskohtaisessa suunnitteluvaiheessa on tarkoitus jatkaa siitä, mihin arkkitehtuurivaiheessa jäätiin. VDI:n 2206:2021 julkaisun mukaan ideana on edistää arkkitehtuurin mukaista suunnittelua rinnakkaissuunnitteluna huomioiden kuitenkin tiimien välinen yhteistyö sekä informaation vaihtaminen. Informaation välittäminen korostuu varsinkin niissä tilanteissa, joissa arkkitehtuurista joudutaan syystä tai toisesta poikkeamaan. Tällöin muutoksien vaikutuksen pohtiminen yhdessä ja niistä tiedottaminen kaikille sidosryhmille on tärkeää. Jotta todennäköisyys muutostarpeille saadaan pidettyä jo lähtökohtaisesti matalana, on suunnittelua tarkoitus verifioida sitä tehtäessä mahdollisuuksien mukaisesti mallintamisen ja simuloinnin keinoin. (VDI 2206:2021) Mallintamista ja simulointia käyttäessä täytyy kuitenkin huomioida, että kyseiset menetelmät ovat alttiita virheille ja niillä on aina tietty pätevyysalue ja tarkkuus – luonnollisesti kaikkea ei voida mallintaa. Vaiheen lopputuloksena järjestelmän osat tulisivat olla suunniteltuina siten, että kaikki vaatimukset mukaan lukien lainsäädännön asettamat vaatimukset olisi huomioitu. Myös fyysiset komponentit tulisivat olla hankittuina (VDI 2206:2021).

Integrointi- ja verifiointivaihe puolestaan alkaa järjestelmän kokoonpanolla. Koska muutoksien tekeminen on helpompaa ja halvempaa mitä aikaisemmassa vaiheessa muutostarpeet huomataan, on jokaisen alijärjestelmän valmistumisen yhteydessä tarkoitus verifioida jo siihen kohdistuva vaatimukset. Lopuksi, kun järjestelmä on kasassa, voidaan koko järjestelmää koskevat vaatimukset verifioida. Koska vaatimukset tulisivat olla määritetty siten, että ne ovat todennettavissa esimerkiksi tarkistamalla tai mittaamalla, pitäisi vaiheen olla suoraviivainen. (VDI 2206:2021)

VDI:n 2206:2021 mukaisen viitemallin viimeinen vaihe on validointi ja siirtäminen. Vaiheen päätarkoitus on todeta tilaajan kanssa lopputuloksen olevan halutunlainen ja luovuttaa lopputulos tilaajalle. Validoinnissa ei niinkään todeta vastaako lopputulos luotua spesifikaatiota vaan tarkoitus on todeta tilaajaan asettamien vaatimuksien ja halujen toteutuminen. Mikäli lopputulos on vastaa tilaajan odotuksia, voidaan lopputulos siirtää tilaajalle. Joidenkin tuotteiden tapauksessa vaiheeseen voi kuulua lainsäädäntöön liittyvä tyyppihyväksyntä tai tuotteen osoittaminen standardin mukaiseksi testillä. Siirtämisen yhteydessä näihin liittyvä dokumentaatio yhdessä muiden lopputulokseen liittyvien dokumenttien kanssa luovutetaan tilaajalle. Usein myös tilaajaan tai tilaajan edustajan kouluttaminen lopputuloksen käyttämisestä kuuluu siirtoprosessiin. (VDI 2206:2021)

Siirtämisen yhteydessä on syytä sopia myös, miten vastuu jakautuu siirtämisen jälkeen toimittajan ja tilaajaan kesken, mikäli niitä ei aikaisemmin ole sovittu. Jossain tapauksissa toimittaja voi esimerkiksi vastata tuotteen varaosista, ylläpidosta, päivittämisestä

ja lopulta hävittämisestä. (VDI 2206:2021) Koska muut elinkaaren ylläpitoon liittyvät toimet vaativat toimittajalta resursseja myös jatkossa, kannattaa näistä sopia todennäköisesti heti prosessin alussa, jotta tarvittaviin toimiin voidaan varautua tarpeeksi ajoissa.

2.2 Ohjausjärjestelmäsuunnittelun näkökulma

Eri alijärjestelmien kehitysprosesseissa on tyypillisesti ominaispiirteitä, jotka eivät tule esille VDI 2206:2021 kaltaisesta kokonaisvaltaisesta järjestelmän kehitysprosessia kuvaavasta mallista tai jotka on vaihtoehtoisesti kuvattu geneerisesti, mikä näkyy varsinkin mallinnus- ja analysointimenetelmien sekä suunnittelutyökalujen kuvaamisessa. Alijärjestelmistä ohjausjärjestelmän koostuu niin ohjelmistokomponenteista kuin myös fyysistä komponenteista. Jotta VDI:n 2206:2021 viitemallin mukaisessa verifiointi- ja integrointivaiheessa ohjausjärjestelmä olisi fyysistä asennusta vaille valmis, tulisi luonnollisesti kaikki ohjausjärjestelmän komponentit olla hankittuina, mutta myös ohjauskoodi implementoituna, siirrettynä ohjainlaiteelle sekä todettu toimivaksi.

Perustavanlaatuinen ongelma ohjausjärjestelmä suunnittelussa muodostuukin nykypäivänä siitä, että lopputuotetta pitäisi pystyä ohjaamaan halutulla tavalla ja lähes virheettömästi heti, kun laite valmistuu. Markkinoilla tapahtuvan kilpailun kiristämän aikatauluvaatimuksen takia ohjausjärjestelmän kehittämiselle ja ohjauskoodissa esiintyvien vikojen korjaamiselle ei haluta jättää suhteettoman paljon aikaa enää kokoonpanon valmistumisen jälkeen. On arvioitu, että vikojen löytäminen ja korjaaminen kenttätestien yhteydessä on noin 30 kertaa kalliimpaa verrattuna niiden löytämiseen suunnitteluvaiheessa. Tilannetta ei myöskään helpota se, että samaan aikaan ohjausjärjestelmät muuttuvat monimutkaisemmiksi teknologian mahdollistaessa yhä kehittyneempiä ominaisuuksia, kuten ajoneuvojen itseohjautuvuuden, mikä tekee ohjausjärjestelmästä yhä monimutkaisemman. Monimutkaisuus puolestaan näkyy usein ohjauskoodin pituutena, jolloin myös virheiden määrä ohjauskoodissa kasvaa. (Socci 2015)

Ongelman kokoluokka käy ilmi The Economist (2010) lehden julkaisusta, jonka mukaan nykyaikaiset sulautetut järjestelmät, kuten autot, voivat sisältää miljoonia rivejä ohjauskoodia ja parhaimmissakin tilanteissa laajojen sisäisten testien jälkeen niissä esiintyy 0,5 virhettä tuhatta koodiriviä kohden. Teollisuudessa organisaation sisäisessä tai toiselle organisaatiolle tehdyissä ohjelmissa, joka ei ole käynyt läpi samanlaista äärimmäisyyksiin vietyä testausprosessia, arvioidaan olevan 5–50 virhettä tuhatta koodiriviä kohden. Tarkempi luku riippuu kuitenkin muun muassa käytetystä ohjelmointikielestä ja kehitysympäristöstä sekä ohjauskoodin tekemiseen käytetystä ajasta ja siitä, kuinka paljon asiakas on valmis maksamaan lopputuloksesta (The Economist 2010).

2.2.1 Mallipohjainen suunnittelu osana kehitysprosessia

Alalla jo lähes vakiintunutta ideologiaa kehitysprosessin kulusta, joilla ohjausjärjestelmästä ja sen ohjelmistosta yritetään saada laadukkaampi eli valmiimpi ja virheettömämpi ennen laitteen kokoonpanoa ja kenttätestejä, kutsutaan ohjausjärjestelmän mallipohjaiseksi kehitysprosessiksi tai mallipohjaisen suunnittelun (engl. model based design, MBD) työkierroksi. Ohjausjärjestelmien mallipohjainen kehitysprosessi esitetään usein V-mallin avulla, jossa päävaiheet ovat käytännössä samat kuin perinteisissä V-malleissa. Edellä esitettyjen nykyaikaisen ohjausjärjestelmän kehityksen haasteisiin vastataan kuitenkin kehitysprosessin nimen mukaisesti mallinnuksen avulla. Ideana on luoda kehitysprosessin alussa tekeillä olevasta järjestelmästä tarpeeksi tarkka tietokonepohjainen malli simulointiympäristöön, jota voidaan hyödyntää suuntaa antavaan verifiointiin kehitysprosessin edetessä eri tavoin.

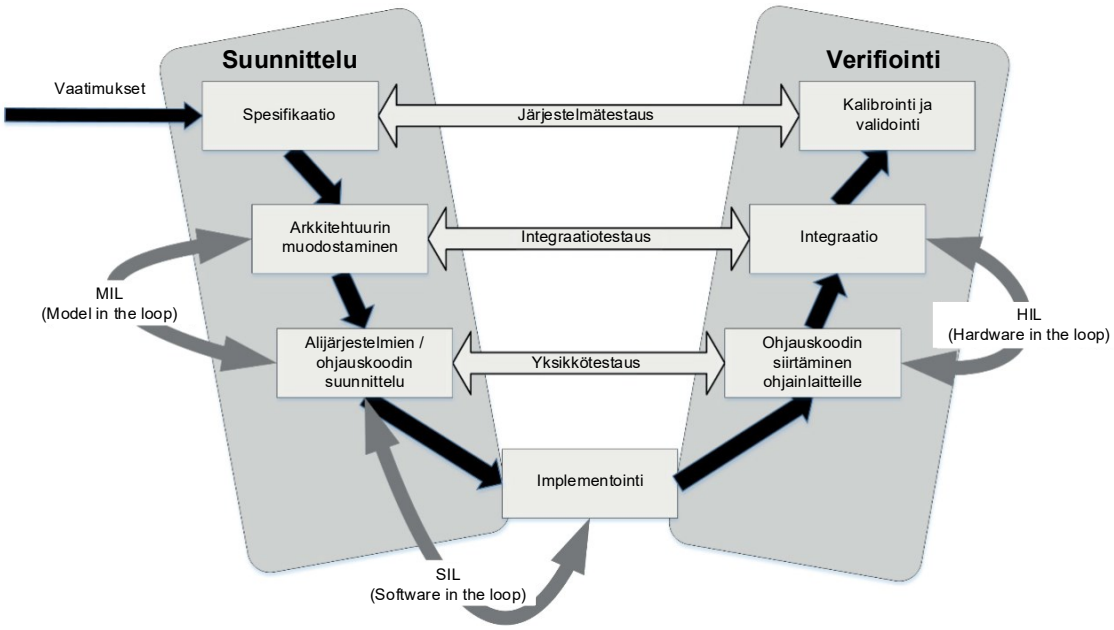
Mallipohjaisen kehitysprosessin tärkein hyöty saavutetaan kuitenkin jo kehitysprosessin alussa, jolloin simuloinnin avulla voidaan verrata useita erilaisia ohjausalgoritmeja erilaisilla ohjausparametreilla. Tällöin on mahdollista saavuttaa laadukkaamman ohjauskoodin lisäksi esimerkiksi suorituskykyisempi ja luotettavampi ohjaus. Kirjallisuudessa tähän viitataan joskus käsitteellä pikamallinnus (engl. rapid prototyping). Pikamallinnuksen käyttäminen vastaakin esimerkiksi Munassarin ja Govardhanin (2010) esittämään kritiikkiin siitä, ettei perinteisen V-mallin mukainen kehitysprosessi mahdollista aikaisia prototyyppejä.

Pikamallinuksesta ei välttämättä kuitenkaan mainita esiteltäessä mallipohjaista kehitysprosessia vaan puhutaan laajemmin mallin hyödyntämisestä osana kehitysprosessia. Riippuen lähteestä menetelmän testivaiheet jaetaan eri tavalla, mutta kokonaisuutena ne pitävät sisällään usein samat asiat. Soccin (2015) ja Ahopellon (2019, s. 20–22) esittämät vaiheet on esitetty taulukossa 1.

Taulukko 1. *Ohjausjärjestelmän mallipohjaisen kehitysprosessin testausvaiheet*

Testausvaihe (Ahopelto 2019)	**Testausvaihe (Socci 2015)**	**Selite**
Model in the loop, MIL	Model in the loop, MIL	Mallin simulointia käytetään apuna vaatimusten tarkentamiseen ja ohjausalgoritmikonseptin valitsemiseen ja verifiointiin
Software in the loop, SIL	Software in the loop, SIL	Ohjausalgoritmin pohjalta tehtyä ohjauskoodia testataan simulointiympäristössä
	Processor in the loop, PIL	Ohjauskoodia testataan simuloimalla se pyörimään virtuaalisella ohjainlaitteella ohjainlaitekohtaisessa kehitysympäristössä.
Hardware in the loop, HIL	Hardware in the loop, HIL	Ohjauskoodia testataan fyysisellä ohjainlaitteella simuloimalla muu järjestelmä
	Iron in the loop, IIL	Muita fyysisiä komponentteja lisätään simulaatioon, kuten dieselmoottori ohjainlaitteiden lisäksi
-	Environment in the loop, EIL	Laitteen valmistumisen jälkeen sen toimintaa testataan kenttätesteillä laitteelle ominaisessa ympäristössä
-	Customer in the loop, CIL	Asiakas ja loppukäyttäjä otetaan mukaan kenttätesteihin. Tarkoituksena saada aikaista palautetta validointia varten.

Kuten taulukosta huomataan, Soccin (2015) esittämistä vaiheista EIL (engl. environment in loop) ja CIL (engl. customer in loop) ei ole sisällytetty millään tavalla Ahopellon (2019) esittämään prosessimalliin. Soccin (2015) mukaan kyseiset vaiheet ovat jopa ratkaisevia todentamisen ja validoinnin vaiheita, jotta laitteesta ja sen ohjausjärjestelmästä tulee asiakkaan ja loppukäyttäjään vaatimuksien mukainen. Samaa ideologiaa tukee myös VDI:n 2206:2021 viitemalli. Ahopelto (2019) on kuitenkin onnistunut esittämään prosessin vaiheet ohjausjärjestelmäsuunnittelun kannalta konkreettisemmalla tasolla, minkä vuoksi se on esitetty kuvassa 3.



Kuva 3. *V-mallin avulla esitetty ohjausjärjestelmän mallipohjainen kehitysprosessi (Ahopelto 2019, s. 21, muokattu)*

Verrattaessa kuvan 3 esittämää ohjausjärjestelmän mallipohjaista kehitysprosessia kuvassa 2 esitettyyn koko mekatronisen järjestelmän kehitystä kuvaavaan VDI:n 2206:2021 viitemalliin on huomattava, että mallien aikajanat eroavat toisistaan. Ohjausjärjestelmän ohjauslaitteiden testaamista sekä ohjausjärjestelmän integrointia ja sen testaamista on tarkoitus tehdä juuri HIL-menetelmällä ennen kuin VDI:n 2206:2021 viitemallissa siirrytään järjestelmän osien implementointivaiheesta järjestelmän integrointi- ja verifiointivaiheeseen. Tällöin mallipohjaisen kehitysprosessin päätavoitteeseen päästään eli ohjauskoodin laatua saadaan parannettua ennen fyysisen laitteen kokoonpanon aloittamista. Tyypillisesti kuitenkin osa ohjausjärjestelmän integraatiovaiheesta joudutaan tekemään fyysisen kokoonpanon eli järjestelmän integraation yhteydessä, sillä usein osa komponenteista on jouduttu simuloimaan HIL-vaiheessa (Ahopelto 2019 s. 20–22). Luonnollisesti myös validointi ja iso osa komponenttien, kuten antureiden, kalibroinnista joudutaan tekemään vasta laitteen fyysisen kokoonpanon valmistumisen jälkeen.

2.2.2 Kehitykseen käytettävät työkalut

Mallipohjaisen kehitysprosessin hyödyntämiseen tarvitaan kolmea kehitysympäristöä, jotka ovat mallinnus- ja simulointiympäristö (engl. rapid prototyping unit), ohjauskoodin ohjelmointi- ja simulointiympäristö (engl. desktop test enviroment) sekä järjestelmän integraatioalusta (engl. system integration platform). Näistä mallinnus- ja simulointiympä-

ristössä (jatkossa mallinnusympäristössä) muodostetaan järjestelmämalli ja ohjausalgoritmivaihtoehtoja, jotta voidaan hyödyntää taulukon 1 menetelmistä MIL-menetelmää. (Socci 2015)

Seuraavaksi ohjauskoodin ohjelmointi- ja simulointiympäristössä (jatkossa ohjelmointiympäristössä) muodostetaan ohjauskoodi ja verifioidaan se SIL- ja PIL-menetelmillä. Lisäksi ohjelmointiympäristössä ohjauskoodi voidaan usein siirtää myös yhdelle tai useammalle ohjainlaitteelle, jolloin verifiointia voidaan tehdä tältä osin HIL-menetelmällä. Lopuksi ohjausjärjestelmä on tarkoitus koota laboratoriossa käyttäen fyysisiä komponentteja. Mikäli kaikkia komponentteja ei voida lisätä tässä vaiheessa järjestelmään, voidaan loput simuloida järjestelmän integraatioalustalla eli toisin sanoen tarkoitus on hyödyntää HIL- ja IIL-menetelmiä. (Socci 2015)

Tyypillisiä kaupallisia mallinnusympäristöjä ovat Mathworksin (2022a) Simulink tai Siemensin (2022) Simcenter Amesim, jotka molemmat mahdollistavat graafisen lohkopohjaisen mallinnuksen ja sisältävät paljon funktiokirjastoja. Näistä ainakin Simulink mahdollistaa lisäksi automaattisen ohjauskoodin generoimisen C- ja C++ -ohjelmointikielille tai IEC-61131-3 mukaiselle PLC-koodille (engl. programmable logic controller, PLC) (Simulink 2022a). Automaattista koodin generointia käytettäessä kehitysprosessi suoraviivaistuu merkittävästi, kun esimerkiksi simulaation avulla toimivaksi todettuja säätimiä voidaan automaattisesti kääntää suoraan ohjainlaitteen tukemaksi koodiksi. Menetelmä ei kuitenkaan täysin poista tarvetta ohjelmointiympäristölle, sillä fyysisen ohjauslaitteen sisäänmenot ja ulostulot tulee alustaa oikeanlaisiksi, ja niihin tulee osoittaa generoidun ohjauskoodin vastaavat. Myös tiedonsiirtorajapintaan liittyvät alustukset ja sen avulla vastaanotettavat ja lähetettävät parametrit määritetään usein ohjelmointiympäristössä. Näiden lisäksi voi olla mielekkäämpää ja nopeampaa tehdä joitain ominaisuuksia, kuten esimerkiksi yksinkertaisia turvatoimintoja, suoraan ohjelmointiympäristössä.

Itse ohjelmointiympäristön valinta riippuu puolestaan valitusta ohjainlaitteesta, sillä ohjainlaitteen tulee olla tuettu, jotta ohjauskoodi voidaan kääntää sen avulla ohjainlaitteelle yhteensopivaan muotoon. Useiden kaupallisten työkonekäyttöön soveltuvien ohjainlaitteiden valmistajat, kuten esimerkiksi Dasa (2022), EPEC (2022), HAWE Hydraulik (2022) ja HYDACin omistama Technion (2022), käyttävät tuotteissaan ohjelmointiympäristönä Codesysiä (2022). Bosch Rexroth (2022) on puolestaan toteuttanut Codesys-tuen lisäksi C-ohjelmointikielelle rajapinnan, jonka avulla ohjauskoodi voidaan tehdä myös C-pohjaisissa ohjelmointiympäristöillä, kuten Microsoftin Visual studiolla (Microsoft 2022). Haluttaessa hyödyntää automaattista koodin generointia näissä kehitysympäristöistä Codesys tukee IEC-61131-3 mukaista PLC-koodia ja luonnollisesti C-pohjaiset kehitysympäristöt C-koodia.

Järjestelmän integraatioalustan voidaan puolestaan ajatella koostuvan ohjausjärjestelmän laitteiden ja niiden välisen tiedonsiirtoratkaisun – usein liikkuvien työkoneiden kehityksen tapauksessa CAN-väylän – konfigurointi-, monitorointi- ja diagnosointisovelluksista sekä tietokoneesta, joilla voidaan muodostaa puuttuvat komponentit simuloituna. Tarkemmin ideana on käyttää komponenteista tehtyjen simulointimallien sisääntuloina todellisilta komponenteilta tulevia signaaleja ja lähettää muille komponenteille mallien ulostuloina saatuja signaaleita. Tätä varten käytettyjen tietokoneiden tulee olla tarpeeksi nopeita, jotta simulaatiomalli toimii reaaliajassa, ja niissä tulee olla tarpeellinen määrä sisäänmenoja ja ulostuloja. (Socci 2015)

Tarvittavat laitteiden konfigurointisovellukset ja -adapterit riippuvat täysin valituista laitteistosta. Kaupallisia simuloitujen komponenttien muodostamiseen soveltuvia tietokoneita puolestaan valmistaa esimerkiksi dSpace (2022), mutta myös kuluttajamarkkinoille tehtyjä pöytätietokoneita voidaan hyödyntää esimerkiksi Mathworksin (2022b) Simulinkin ja sen tukeman I/O-lisälaitteen (engl. Input/output board, IO-board) sekä tiedonsiirtoratkaisuun, kuten CAN-väylään, liittymiseen tarvittavan sovittimen avulla. Etuna Simulinkin kaltaisen sovelluksen käytössä virtuaalisista komponenttien muodostamiseen on, ettei simulointimallia tarvitse kääntää erillisen tietokoneen tukemalle ohjelmointikielelle, vaan se voidaan ajaa reaaliaikaisesti. (Mathworks 2022b)

Nykypäivänä kehittyneimmät ohjausjärjestelmän kehitykseen soveltuvat ohjelmistot, kuten MathWorksin (2022c) Simulink Real-time-lisäosineen, voivat pitää sisällään kaikki edellä mainitut kehitysympäristöt. Tällaisten kehitysympäristöjen käyttö alusta loppuun voi nopeuttaa merkittävästi kehitysprosessia, mutta rajoittaa ohjausjärjestelmään valittavia ohjainlaitteita, sillä ohjainlaitekohtainen tuki vaaditaan. Simulinkin tapauksessa suoraan tuettuja ohjainlaitteita ovat tällä hetkellä ainoastaan Speedgoatin (2022) valmistamat ohjausjärjestelmän kehitykseen tarkoitetut ohjaustietokoneet, mutta esimerkiksi myös dSpacen (2022) valmistamia ohjaustietokoneita voidaan hyödyntää dSpacen itse tarjoamalla rajapinnalla (engl. Real-Time Interface, RTI). Tehtäessä ensimmäistä fyysistä prototyyppiä esimerkiksi sarjatuotettavasta työkoneesta, on tällaisten käytöstä suuri hyöty. Esimerkiksi INCOVA onnistui nopeuttamaan raskaiden työkoneiden hydrauliventtiilistöjen ohjausjärjestelmien suunnittelua noin puolella hyödyntämällä Simulink Real-time -ohjelmistoa yhdessä Speedgoatin ohjaustietokoneiden kanssa verrattuna heidän aikaisemmin käyttämäänsä prosessiin (Mathworks 2022d).

Hintansa ja usein kokonsa sekä säänkestonsa puolesta kehitykseen tarkoitetut ohjaintietokoneet eivät kuitenkaan sovellu käytettäväksi sarjatuotettavassa työkoneessa ja pienemmissä organisaatiossa näiden käyttäminen jopa ensimmäisessä prototyypissä voi olla liian kallista yhdessä kokonaisvaltaisen kehitysympäristön lisenssimaksujen kanssa.

3. OHJAUSJÄRJESTELMÄN ARKKITEHTUURI

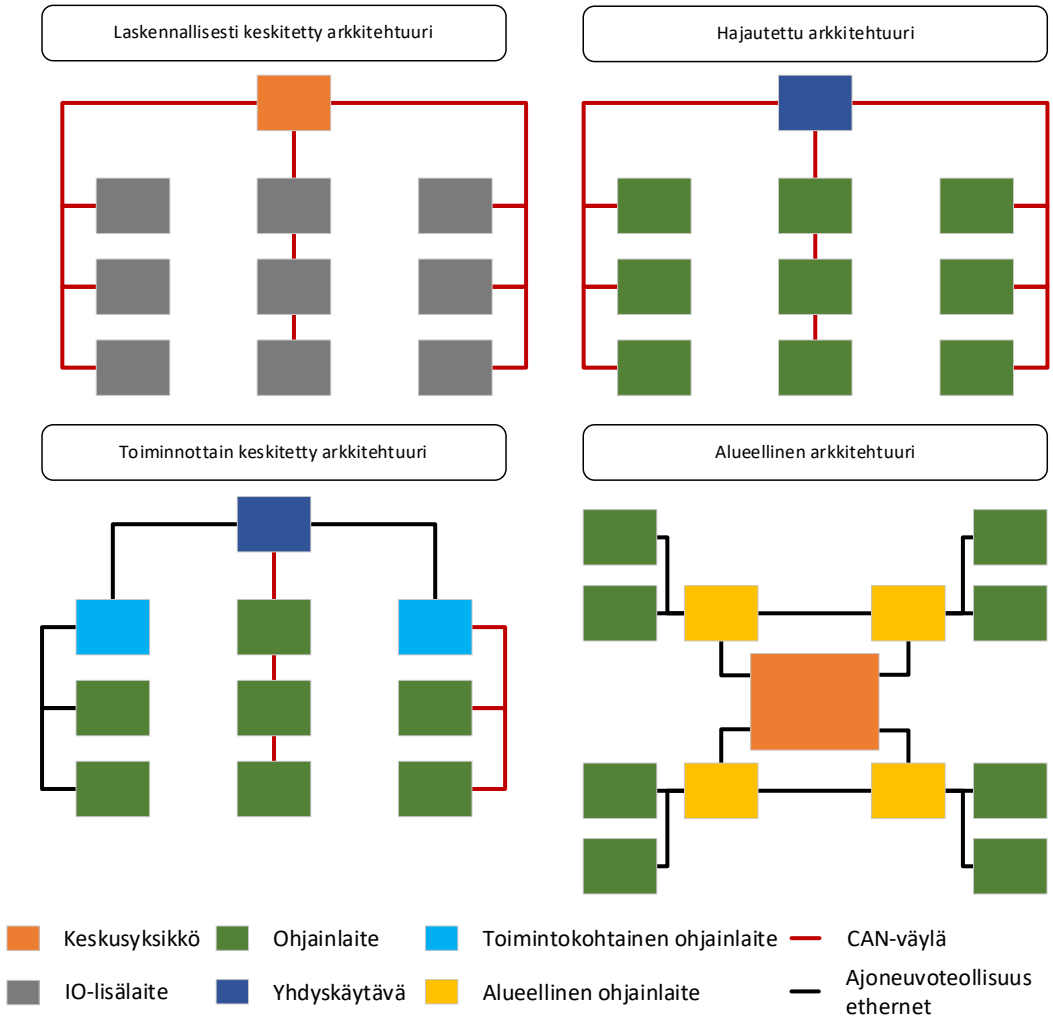
Ehkä perustavinlaatuisin ohjausjärjestelmän arkkitehtuurissa määritettävä asia on hajautusaste, jonka ääripäät ovat täysin keskitetty järjestelmä ja täysin hajautettu järjestelmä. Ideologisesti täysin keskitetyssä järjestelmässä kaikki laskenta ja toimilaitteiden ohjaus tehdään yhdellä ohjainlaitteella. Koska useinkaan – ja varsinkin nykyaikaisten liikkuvien työkoneiden tapauksessa – tämä ei ole järkevää, sillä esimerkiksi kaapeloinnin määrä toimilaitteiden sijaitessa ympäri järjestelmää kasvaisi järjettömästi, puhutaan keskittämisen yhteydessä nimenomaan laskennallisesta keskittämisestä. Tällaisessa arkkitehtuurissa, johon usein viitataan isäntä–orja-arkkitehtuurina, laskenta hoidetaan keskitetysti keskusyksiköllä ja toimilaitteet ja anturit yhdistetään ohjausjärjestelmään tiedonsiirtoratkaisun ja erillisten ympäri työkonetta sijaitsevien logiikkaa sisältämättömien I/O-lisälaitteiden avulla. Puhuttaessa taas täysin laskennallisesti hajautetusta järjestelmästä tarkoitetaan järjestelmää, jossa jokaisen toimilaitteen ohjaus hoidetaan erillisellä ohjainlaitteella. Mikäli informaatiota ohjainlaitteelta toiselle tarvitsee siirtää, hoidetaan tämä käyttäen tiedonsiirtoratkaisua.

Käytännössä kuitenkin kumpikin arkkitehtuurin ääripää on liikkuvissa työkoneissa erittäin harvinainen. Esimerkiksi täysin keskitetty järjestelmä on haastava toteuttaa, sillä työkoneissa on usein laitteita, kuten dieselmoottori, jonka ohjainlaite on integroituna itse toimilaitteeseen. Toisaalta täysin hajautettu järjestelmä vaatii merkittävän määrän ohjainlaitteita, mikä voi näkyä merkittävästi valmistuskustannuksissa. Tyypillinen lähestyminen arkkitehtuurisuunnittelulle sarjatuotettavissa työkoneissa on kuitenkin enemmän hajautettu kuin keskitetty, sillä se tukee paremmin konfiguroitavia modulaarisia tuotteita ja niiden tuotekehitystä, jollaiseen liikkuvan kaluston koneenrakennuksessa usein pyritään (Ahopelto 2019, s. 22–25 & VDI 2206:2021).

Tässä lähestymistavassa ohjainlaiteet sijoitetaan moduuleittain, jolloin myös eri varustelutason valinta tai perustavanlaatuiset muutokset moduuleihin onnistuvat ilman, että moduulin ulkopuolelle tai rajapintoihin tarvitsee tehdä muutoksia. On kuitenkin huomionarvoista, että modulaarisen ohjausjärjestelmän, kuten muunkin modulaarisen rakenteen, implementointi vaatii hyvin suunnitellut rajapinnat, jotta niihin ei tarvitse tehdä muutoksia tuotepäivitysten yhteydessä (Ahopelto 2019, s. 22).

Arkkitehtuurin ja sen hajautusasteen määrittelyyn liittyy modulaarisuustavoitteiden lisäksi keskeisesti laskentatehon tarve. Luonnollisesti hajauttamalla eli lisäämällä järjes-

telmään useampi ohjainlaite käytettävissä olevaa laskentatehoa voidaan kasvattaa. Hajauttaminen voidaan tehdä korvaamalla yksi toimilaitteita ohjaava ohjainlaite useammalla rinnakkaisella ohjainlaiteella tai jakamalla ohjausjärjestelmän toiminnot aikakriittisyyden mukaan useampaan hierarkkiseen tasoon, jolloin toimilaitteiden ohjaamista vähemmän aikakriittiset toiminnot voidaan siirtää toiselle niin sanotulle ylemmän tason ohjauslaitteelle. Varsinkin autoteollisuudessa kovaa vauhtia kasvanut laskentatehon tarve esimerkiksi autonomisen ajamisen myötä on ajanut kehitystä tähän suuntaan. Autoteollisuudessa hierarkkista hajautusta on toteutettu aluksi lisäämällä yhteinen ylemmän tason ohjainlaite kaikille saman toimintakokonaisuuden ohjainlaiteille ja aivan uusimmissa järjestelmissä lisäämällä toimintakokonaisuuskohtaisten ohjainlaitteiden yläpuolelle vielä yksi yhteinen tehokas keskusyksikkö (Askaripoor et al. 2022). Näistä ensimmäistä arkkitehtuuria voidaan kutsua toiminnoittain keskitetyksi järjestelmäksi ja jälkimmäistä alueelliseksi arkkitehtuuriksi. Kuvassa 4 on esitetty nämä arkkitehtuurit yhdessä perinteisen hajautetun ja keskitetyn järjestelmän arkkitehtuurien lisäksi.



Kuva 4. *Tyypillisiä ohjausarkkitehtuureja (mukailen osaksi Askaripoor et al. 2022)*

Hajautetussa arkkitehtuurissa ja toiminnoittain keskitetyssä arkkitehtuurissa tummansininisillä kuvatuilla yhdyskäytävillä voidaan kääntää viestejä eri ohjainlaitteilta toisille, mikäli ne eivät ole suoraan kiinni toisissa esimerkiksi kaapelien reitityksen tai eri tiedonsiirtostandardin takia. Yhdyskäytävällä toteutetaan myös keskitetysti vikakoodien haku eri ohjainlaitteilta ja määritetään varustelutaso eli toisin sanoen määritetään, mitkä ohjainlaitteet pitäisi löytyä järjestelmästä, jotta ohjainlaitteet ymmärtävät siirtää dataa keskenään toteuttaakseen tietyn varustelutason toiminallisuudet. Alueellisessa arkkitehtuurissa nämä hoidetaan oranssilla kuvatun keskusyksikön avulla. Laskennallisesti keskitetyssä arkkitehtuurissa yhdyskäytävälle ei ole tarvetta, sillä I/O-lisälaitteissa ei ole logiikkaa eikä niiden välillä täten voida vaihtaa informaatiota. Kuten alueellisessa arkkitehtuurissa, myös laskennallisesti keskitetyssä arkkitehtuurissa vikadiagnostiikkaliityntä voidaan toteuttaa suoraan keskusyksikköön.

Laskentatehoresurssien kasvamisen lisäksi selkeä muutos on näkynyt tiedonsiirtoratkaisun avulla siirrettävän datan määrässä, mikä johtuu ohjausjärjestelmään yleisesti liitettävien laitteiden määrän kasvusta, mutta varsinkin autonomisten toimintojen toteuttamista varten tarvittujen tutka- ja kamerajärjestelmien lisäämisestä. Tätä varten on otettu käyttöön ajoneuvoteollisuuteen optimoidut Ethernet-pohjaiset tiedonsiirtoratkaisut. CAN-väylään verrattuna sen on osoitettu myös vähentävän kaapeloinnin määrää kompleksissa arkkitehtuureissa. Ethernet-pohjaisten tiedonsiirtoratkaisujen käyttäminen on kuitenkin vasta yleistymässä autoteollisuudessa 2020-luvulle tultaessa, koska aikaisemmin kaistan jakamisen hallinnassa on ollut puutteita, eikä sen avulla ole voinut synkronisoida laiteiden reaaliaikakelloja. Tiedonsiirtoratkaisu ei myöskään ole täyttänyt valmistajien elektromagneettisen häiriön sietoon kohdistuvia vaatimuksia ja ennen kaikkia tiedonsiirron viive on ollut liian suuri, ja siinä on ollut merkittävää epävarmuutta riippuen tiedonsiirron kuormituksesta. (Ixia 2014)

Nykyisin suuri haaste CAN-väylän korvaamiselle ja Ethernetin adaptoinnille työkoneteollisuudessa on Ethernet-tuen puuttuminen hyllykomponenteissa. Kehitystä hidastaa myös yleisesti hyväksytyn koko tiedonsiirtoratkaisua kuvaavan standardin puuttuminen (Ixia 2014). Autoteollisuudessa tämä on kierretty tekemällä autoihin omat valmistajakohtaiset ohjainlaiteet omine kommunikaatioprotokollineen, mikä voidaan nähdä usein kannattamattomana työkoneteollisuudessa merkittävästi pienemmän markkinakoon takia. Siitä huolimatta, että Ethernet ei ole yleistynyt ohjauksessa, ovat muutamat työkoneiden ohjainlaitevalmistajat toteuttaneet ohjelmointi- ja diagnosointirajapinnan perinteisen TCP/IP-pinon (engl. Internet protocol suite) avulla. Näin esimerkiksi ohjelman lataaminen ohjainlaitteelle ja toiminnan monitorointi onnistuu suoraan tietokoneilta jopa internetin välityksellä (EPEC 2023a).

Ohjausarkkitehtuuriin ja tarkemmin laitevalintoihin voi vaikuttaakin tarve pilvipalvelinyhteydelle. Esimerkiksi Ponssen (2023) metsäkoneiden toiminnasta kerättyä dataa lähetetään reaaliajassa hallinnointijärjestelmään, jonka avulla koneiden omistajat voivat antaa ohjeita yksittäisen koneen ajajalle, kerätä tietoa koneen toiminnasta, muodostaa hakkuuraportteja sekä suunnitella huoltoja ja kuljetuksia koneille. Ohjausjärjestelmän laitevalintoihin voi lisäksi vaikuttaa alueellinen toiminnallisen turvallisuuden lainsäädäntö. Esimerkiksi Euroopan unionnin alueella voimassa oleva konedirektiivin uusin versio 2006/42/EC (2019) määrittää, ettei ohjausjärjestelmän laitteisto- tai ohjelmistovika saa aiheuta vaaratilanteita. Tämän toteuttamiseksi voi olla perusteltua hankkia ohjausjärjestelmään valmiiksi turvallisuussertifioituja ohjainlaitteita, joissa laskenta on kahdennettu ja joiden mukana toimitetaan usein turvallisuuslähtöisiä ohjelmointikirjastoja.

4. CAN-VÄYLÄ

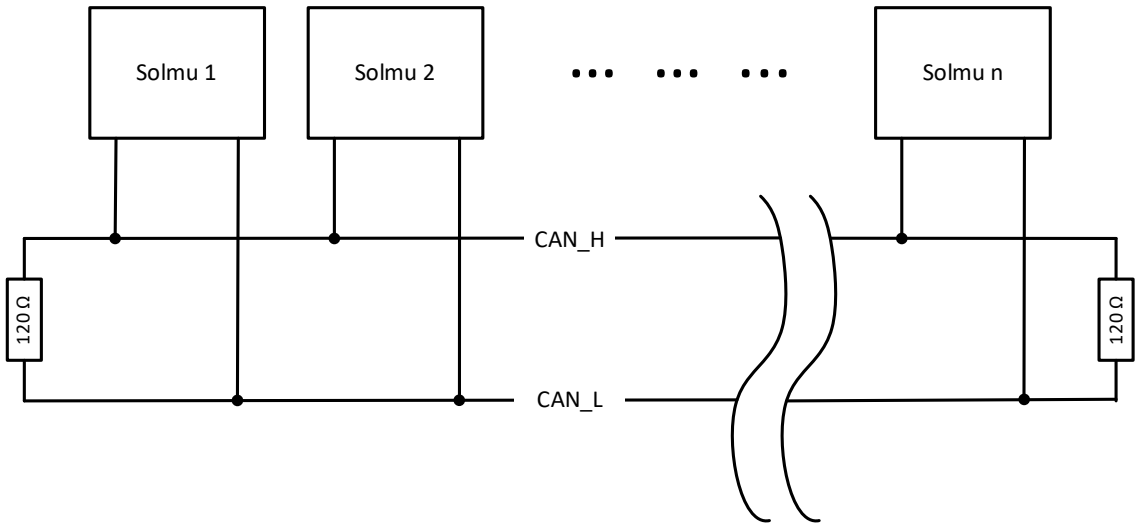
Alun perin CAN-väylä on Boschin vuonna 1986 julkaisema digitaalinen tiedonsiirtoratkaisu, joka luotiin yksinkertaistamaan analogisen tiedonsiirron monimutkaisuutta ajoneuvoissa. Myöhemmin kansainvälisen standardisoimisjärjestön (engl. International Organization for Standardization, ISO) 11898 standardiin määritelty CAN-väylä ei itsessään ole täydellinen tiedonsiirtoratkaisu vaan määrittää vain ISO/IEC 7498-1 (1994) julkaisussa esitetyn OSI-mallin (engl. Open Systems Interconnection Reference Model) kerroksista fyysisen ja siirtokerroksen. Lisäksi siitä on olemassa useita käytössä olevia versiota, kuten CAN2.0, CAN FD (engl. CAN with Flexible Data-Rate) sekä valmisteilla oleva CAN XL (engl. CAN with extra-large dataframes).

Tässä työssä keskitytään versioon CAN2.0 eli klassiseen CAN-väylään ja tarkemmin sen nopeampaan versioon, johon suurin osa työkoneteollisuudessa tällä hetkellä käytettävistä CAN-pohjaisista tiedonsiirtoratkaisuista, kuten CANopen ja SAE (engl. Society of Automotive Engineers) J1939 sekä maatalouskoneisiin SAE J1939 väyläratkaisusta jatkokehitetty ISOBUS, perustuu. Nopeampien ja enemmän dataa kerrallaan siirtävien FD ja XL CAN-väylien käyttö työkoneteollisuudessa ei ole yleistynyt merkittävästi, sillä klassisen CAN-väylän tiedonsiirtokapasiteetti on usein koettu riittäväksi. Saman kenttäväyläratkaisun käyttäminen vuosikymmenien ajan on tuonut markkinoille laajan yhteensopivan CAN-väylää tukevan laitekannan, mikä voidaan nähdä valmistajien halukkuutena pidättäytyä klassiseen CAN-väylään pohjautuvissa tiedonsiirtoratkaisuissa.

Seuraavissa aliluvuissa käsitellään klassisen nopean CAN-väylän alimmat kerrokset OSI-mallin mukaisesti, jonka jälkeen esitetään pääpiirteittäin työn kannalta oleellisten CANopen ja SAE J1939 -standardien tuomat lisämäärittelyt. Huomioitavaa on, ettei ISO 11898 standardin ja ylempien standardien rajapinta ole täydellisesti OSI-viitemallin siirto- ja verkkokerroksen välissä, vaan ISO 11898 -standardi ottaa kantaa myös joihinkin ylempien kerroksien määritettäviin asioihin. Selkeyden vuoksi usein puhutaankin alemman tason standardista, jolla viitataan ISO 11898 -standardin ensimmäiseen ja toiseen osaan, ja ylemmän tason standardista, joka määrittää loput oleelliset osat mahdollistaakseen tiedonsiirtoratkaisun implementoinnin eri valmistajan laitteiden kesken. Huomioitavaa on myös, että ylempiinkään standardeihin ei usein ole määritetty aivan kaikkia OSI-mallin mukaisia toiminnallisuuksia, kuten viestien reitittämistä useamman eri verkon läpi, sillä perusluonteeltaan CAN-väylää käytetään suhteellisen yksinkertaisen informaation lähettämiseen yksinkertaisessa topologiassa toisin kuin esimerkiksi Ethernetia.

4.1 Fyysinen kerros

Klassinen nopea CAN-väylä (jatkossa CAN-väylä) on topologialtaan aina väylä, jossa laitteet eli solmut liitetään lyhyillä haaroituksilla parikaapeliin perustuvaan fyysiseen mediaan, kuten kuvassa 5 on esitetty. Parikaapelin impedanssin tulee olla 120 Ω ja se voidaan suojata vapaaehtoisesti elektromagneettista häiriötä vastaan sen ympärille tulevalla suojavaipalla. Heijastuksien estämiseksi väylän molempiin päihin tulee asentaa lisäksi erilliset 120 Ω vastukset parikaapelin johtimien välille. (Di Natale et al. 2012 s. 1–12)



Kuva 5. *CAN-väylän topologia*

Parikaapelin johtimet on puolestaan nimetty niiden jännitetason mukaan CAN_Low ja CAN_High -johtimiksi, ja digitaalisen informaation lähettäminen perustuu näiden johtimien väliseen jännite-eroon. Jännite-eroon perustuvassa informaation lähettämisessä pyritään välttämään sähkömagneettisen häiriön vaikutus, koska häiriö vaikuttaa lähes yhtä voimakkaasti kahteen toisiaan lähellä olevaan johtimeen, jolloin jännite-ero pysyy muuttumattomana. ISO 11898 määrittelyn lisäksi kiertämällä kaapelit toistensa ympärille voidaan häiriön tasaista indusoitumista parantaa entisestään. Kierrettäessä monisäikeiset johtimet toistensa ympärille tulee kierroksia olla noin 40 metriä kohden. Häiriöiden indusointia väylään voidaan estää lisäksi valitsemalla suojattu kaapeli, joka tulee maadoittaa ainoastaan toisesta päästä maadoitussilmukan välttämiseksi.

Koska CAN-väylässä käytetään NRZ-linjakoodausta (engl. non-return-to-zero line coding), ei kolmatta erillistä niin sanottua lepotilaa ole. Toisin sanoen jokainen solmu lähettää väylälle joko bitin arvon nolla tai yksi. Kun lähetetään bitti arvoltaan nolla, CAN_High-johtimen ja maan välinen potentiaali on 3,5 V ja CAN_Low-johtimen ja maan välinen

potentiaali 1,5 V. Kun lähetetään bitti arvoltaan yksi, ovat molempien johtimien potentiaalit puolestaan 2,5 V. Jännite-eron avulla esitettäessä bitin arvo on nolla, kun CAN-johtimien välillä on 2,0 V ja bitin arvo yksi, kun johtimien välillä on 0 V. Jännite-eron mukaan bitin tilaa nolla kutsutaan dominoivaksi ja bitin tilaa yksi väistyväksi. Kun väylälle ei lähetetä mitään, vastaavat väyläjännitteet väistyvän bitin arvoa. (Di Natale et al. 2012 s. 1–12) Signaalien arvot toleransseineen on esitetty tarkemmin taulukossa 2. Mikäli jännitteet ovat annettujen toleranssien ulkopuolella, se johtaa bitin sisältävän viestin hylkäämiseen.

Taulukko 2. *CAN-johtimien jännitteet*

Signaali	Väistyvä tila [1]			Dominoiva tila [0]		
	Minimi	Nimellinen	Maksimi	Minimi	Nimellinen	Maksimi
CAN-High	2,0 V	2,5 V	3,0 V	2,75 V	3,5 V	4,5 V
CAN-Low	2,0 V	2,5 V	3,0 V	0,5 V	1,5 V	2,25 V

Koska CAN-väylä on tahdistettu tiedonsiirtoratkaisu, bitin lähetykselle ja vastaanottamiselle on varattu tietty aikaikkuna. Tahdistaminen tapahtuu aina, kun bitin arvo muuttuu arvosta yksi arvoon nolla, jolloin säädetään aikaikkunan aloituskohtaa jännite-eron nousuajan avulla eikä erillistä kellosignaalia käytetä tahdistamiseen. Jotta tiedonsiirtoratkaisu saadaan pidettyä tahdissa myös silloin, kun väylälle ei lähetetä mitään tai viestissä on useampi saman arvoinen bitti peräkkäin, käytetään menetelmää, jota kutsutaan englanniksi termillä bit stuffing. Menetelmässä lähetetään jokaisen viiden saman bitin jälkeen vastakkainen bitti. Näiden bittien lisääminen lähettävässä päässä ja poistaminen vastaanottavassa päässä tapahtuu automaattisesti CAN-lähettimen ja -vastaanottimen toimesta. (Di Natale et al. 2012 s. 1–12)

Aikaikkunan pituuden nimellisarvoa kutsutaan bittiajaksi, joka puolestaan on sidottu väylän tiedonsiirtonopeuteen. Jos bitin tila halutaan vaihtaa vastakkaiseksi, tulee väylän johtimien jännitteiden muuttua tässä ajassa. Toisaalta todellinen aika, joka bitiltä kuluu siirryttäessä laitteelta toiselle, riippuu väylän pituudesta. Näiden pohjalta väylän teoreettinen maksimipituus on mahdollista määrittää tietylle tiedonsiirtonopeudelle. Todellisuudessa näihin väylänpituuksiin ei päästä johtuen ainakin sähkömagneettisista heijastuksista ja häiriöistä sekä väyläjohtimissa tapahtuvista häviöistä, joita aiheuttavat epäjatkuvuudet, kuten liittimet ja piirilevyillä esiintyvät juotokset. (Di Natale et al. 2012 s. 1–12)

Heijastuksien tulee olla vaimentuneena ennen seuraavan bitin tilan lähettämistä, jotta bitin tila rekisteröityy oikein. Heijastuksiin vaikutetaan ennen kaikkia asentamalla resistanssiltaan oikeansuuruiset päätevastukset väylien päähän ja käyttämällä oikean impedanssin kaapelointia, mutta myös minimoimalla väylän pituus ja ennen kaikkea laitteille vievien haaroituksien pituudet. Käyttämällä lyhyempää väylää ja lyhyempiä haaroituksia

on heijastuneella signaalilla aina lyhyempi matka solmusta päätevastukseen, jolloin heijastuksien vaimenemiseen kuluu vähemmän aikaa.

Jotta tiedonsiirtoratkaisu toimii, tulee tiedonsiirtonopeus ja sen määräämät bittien ajoittamiseen liittyvät parametrit, kuten nimellisbittiaika, olla valittu. ISO 11898 -standardi ei kuitenkaan määritä näitä, vaan rajaa tiedonsiirtonopeuden ylärajaksi yhden megabitin sekunnissa. Väylän käyttäminen maksiminopeudella ei ole useinkaan tarpeellista ja sen haittapuolena on tiedonsiirtoratkaisun robustisuuden heikentyminen ja väylän maksimipituuden pieneneminen. Tämän vuoksi ylemmän tason standardit määrittävät joko maksiminopeudesta pienemmän nopeuden tai antavat useamman eri väylänopeuden, joista käyttäjä voi valita tilanteeseen sopivan. Ylemmän tason standardille jäävät myös määritettäväksi käytettävät liittimet, sillä ISO 11898 ei määritä niitäkään.

4.2 Siirtokerros

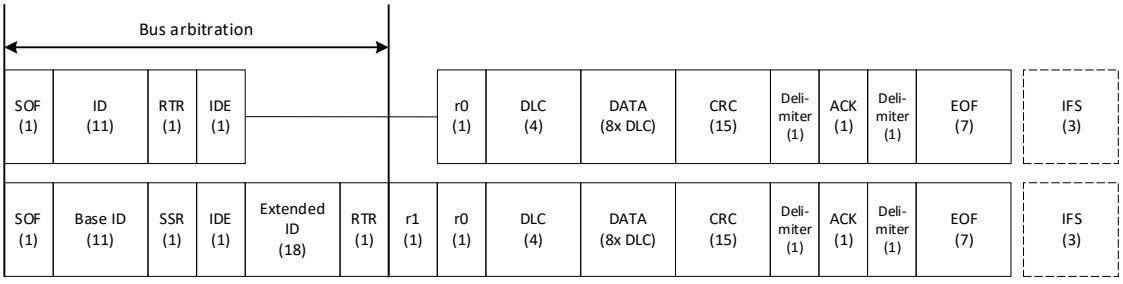
ISO 11898-1 määrittää kaksi erilaista viestikehystä datan lähettämiseksi, jotka kulkevat nimillä normaalikehys (engl. standard frame) ja jatkettu kehys (engl. extended frame). Näiden lisäksi on olemassa myös

- viestikehys datapyyntöä varten (engl. remote frame),
- ylikuormituskehys (engl. overload frame), jolla pakotetaan tietylle solmulle viivettä viestien lähetykseen, mikäli toisen solmun vastaanottopuskuri täyttyy eikä se kykene prosessoimaan viestejä tarpeeksi nopeasti sekä
- vikakehys (engl. error frame), joka aiheutuu signaloinnissa tapahtuneesta virheestä ja siihen liittyvästä toipumisprosessista. (Di Natale et al. 2012 s. 13)

Lähettimet ja vastaanottimet, jotka tukevat ainoastaan datan siirron normaalikehystä, esitetään CAN2.0A-yhteensopivina ja ne, jotka tukevat molempia esitetään CAN2.0B-yhteensopivina. Mikäli käytössä on CAN2.0B-yhteensopivat solmut, voidaan väylällä käyttää molempia viestikehyksiä samaan aikaan. Mikäli väylässä on molempia CAN2.0A- ja CAN2.0B-solmuja, voidaan molempia viestikehyksiä käyttää, mutta CAN2.0A-laitteet eivät voi vastaanottaa jatketun kehyksen mukaisia viestejä. Toisin sanoen CAN2.0A-solmujen tulee sietää jatketun kehyksen mukaisia viestejä väylällä, vaikka niitä ei voida vastaanottaa. Muut edellä mainitut kehykset ovat tuettuja niin A- kuin B-versiossa. (Di Natale et al. 2012 s. 14)

4.2.1 Datan lähettämiseen käytetyt kehykset

Kuten edellä on mainittu, datan lähettäminen tapahtuu joko normaalikehyksellä tai jatketulla kehyksellä. Molempien viestikehyksien avulla voidaan lähettää yhtä paljon dataa ja ne ovat rakenteeltaan hyvin samanlaiset. Ero viestikehyksien välillä muodostuu kuitenkin identifiointikentän pituudessa, joka normaalikehyksessä on 11 bittiä ja jatketussa 29 bittiä. CAN-viestissä identifiointikentällä muodostetaan jokaiselle viestille ainutlaatuinen osoite. Täten normaalikehys mahdollistaa 2048 ainutlaatuista viestiä ja jatkettu 536 870 912. Identifiointikentän jatkaminen on toteutettu erilliseen osaan viestistä, jota kutsutaan jatko-osaksi. Tämä mahdollistaa molempien viestikehyksien alkamisen samalla tavalla, mitä voidaan hyödyntää viestien priorisointiin tilanteessa, jossa väylälle lähetetään molemman viestikehyksen mukaisia viestejä. Viestikehyksien erottaminen toisistaan tapahtuu normaalikehyksen identifiointikentän ja jatketun viestikehyksen identifiointikentän perusosan jälkeisellä IDE-bitillä (engl. identifier extension bit). Viestien tarkempi rakenne on esitetty kuvassa 6, jossa esitettyjen kenttien pituus biteissä on ilmoitettu suluissa. (Di Natale et al. 2012 s. 14–16)



Lyhenne	Englanniksi	Selite
SOF	Start of Frame	Aloittaa viestin (aina dominoiva [0])
ID	Identifier	Viestin tunniste
RTR	Remote transmission request	Datan lähetyspyyntö (Dominoiva [0] dataa lähettettäessä, väistyvä [1] dataa pyydettäessä)
SSR	Substitute remote request	IDE:n kohdentamiseen jatketun ja normaalin viestikehyksen välillä (aina väistyvä [1])
IDE	Identifier extension bit	Määrittää viestikehyksen (dominoiva [0] normaalille kehykselle ja väistyvä [1] jatketulle kehykselle)
r0 – r1	Reserved bits	Varattu bitti myöhempää käyttöä varten. (aina väistyvä [1], mutta kumpikin arvo hyväksytään)
DLC	Data length code	Kertoo kuinka monta viestin datakentää viestissä on eli monta tavua dataa viesti sisältää.
DATA	Data field	DLC:n verran datatavuja
CRC	Cyclic redundancy check	Tarkistussumma, jota käytetään viestin oikeellisuuden tarkistamiseen vastaanottavassa päässä
-	Delimiter	Erotin (aina väistyvä [1])
ACK	Acknowledge	Asetetaan lähtevään väistyväksi [0]. Mahdollinen vastaanottaja ylikirjoittaa bitin dominoivaksi [1]
-	Delimiter	Erotin (aina väistyvä [1])
EOF	End of frame	Lopettaa viestin (aina seitsemän peräkäistä väistyvää [1] bittiä)
IFS	Interframe spacing	Viestikehyksien välinen tauko (aina kolme väistyvää [1] bittiä)

Kuva 6. *CAN-väylän normaalikehys ja jatkettu kehys*

Kun väylälle ei lähetetä viestejä, vastaa väylän jännite-ero väistyvää bittiä. Tämän vuoksi viestikehykset alkavat aina lähettämällä väylälle dominoiva bitti, jolla nimenomaan tunnistetaan viesti alkaneeksi. Tätä seuraa normaalikehyksen identifiointikenttä tai jatketun kehyksen identifiointikentän perusosa. Tilanteessa, jossa useampi laite olisi lähettämässä viestejä samaan aikaan, siirtyy se ensimmäisenä kuuntelutilaan, joka lähettää identifiointikentässä ensimmäisenä väistyvän bitin. Tätä kutsutaan CAN-väylän priori-

sointimenetelmäksi (engl. bus arbitration), joka on esitetty kuvaan 7. Priorisointimenetelmään otetaan mukaan identifiointikentän lisäksi RTR-bitti (engl. remote transmission request bit), jolla määritetään, halutaanko tehdä datapyyntö vai lähettää dataa, sekä yllä kuvattu IDE-bitti. RTR-bitin tilalla jatketussa kehyksessä on SSR-bitti (engl. substitute extension bit), jonka idea on kohdentaa normaalin ja jatketun kehyksen IDE-kentät. Kohdentamisen tarve johtuu siitä, että jatketussa kehyksessä RTR-bitti tulee vasta identifiointikentän jatko-osan jälkeen. (Di Natale et al. 2012 s. 13–21)

	SOF	ID10	ID09	ID08	ID07	ID06	ID05	ID04	ID03	ID02	ID01	ID00	RTR/SSR	IDE	.	.	.
Viesti 1	0	1	0	0	1	0	1	1	0	0	0	1	0	0	.	.	.
Viesti 2	0	1	0	1	Siirtyy kuuntelemaan												
Viesti 3	0	1	0	0	1	0	1	1	0	0	0	1	1	Siirtyy kuuntelemaan			
CAN-Väylä	0	1	0	0	1	0	1	1	0	0	0	1	0	0	.	.	.

Kuva 7. *CAN-väylän viestien priorisointimenetelmä*

Priorisointimenetelmän mukaan korkein prioriteetti tulee normaalikehyksen mukaisessa viestissä identifiointikentästä ja jatketun kehyksen mukaisessa viestissä sen perusosasta. Toiseksi tärkein prioriteetti on datapyynnöllä, sillä dataa lähtevässä viestissä RTR-bitti on väistyvä ja datapyynnössä puolestaan dominoiva. Mikäli molemmat edellä mainituista olisivat samat normaalia ja jatkettua kehystä käyttäville viesteille, priorisoitaisiin normaalikehys, sillä IDE-bitin tila jatketulle kehykselle on väistyvä ja normaalille taas dominoiva. Luonnollisesti kahden jatkettua kehystä käyttävän viestin välillä prioriteettijärjestys voi määräytyä myös identifiointikentän jatko-osan mukaan.

Kuten kuvasta 7 näkyy, järjestetään identifiointikentän bitit CAN-viestissä nousevaan tavujärjestykseen (engl. little endian). Samaa ideaa käytetään myös muissa kentissä. Huomion arvoista on, että jatketussa kehyksessä identifiointikentän jatko-osan viimeinen bitti vastaa koko identifiointiosan ensimmäistä bittiä ja ensimmäinen bitti bittiä numero 17. Perusosan viimeinen bitti jatkaa tästä bitin numerolla 18 ja päättyy bittiin numero 28. Toisin sanoen jatketun kehyksen perusosassa on identifiointikentän bitit väliltä 28–18 ja vastaavassa kohdassa normaalikehyksessä on bitit väliltä 10–0. Viestien priorisointi normaalikehyksen ja jatketun kehyksen välillä tapahtuu siis näiden bittien avulla.

Priorisointiin käytettävien bittien jälkeen normaalikehyksen mukainen viesti sisältää varatun bitin ja jatkettu kaksi varattua bittiä, joille ei ole esitetty CAN2.0-standardissa käyttöä. Nämä bitit tulee asettaa väistyviksi, mutta solmujen tulee pystyä vastaanottamaan näissä kentissä myös dominoivat bitit. Varattuja bittejä seuraa neljästä bitistä koostuva DLC-kenttä (engl. data length code), jolla ilmoitetaan DLC-kenttää seuraavan datakentän datatavujen määrä. Datakentässä voi olla datatavuja 1–8 kappaletta eli bitteinä ilmoi-

tettuna 8–64 bittiä. (Di Natale et al. 2012 s. 13–21) Ylimääräisten datatavujen poistaminen viestistä nopeuttaa marginaalisesti viestin prosessointia ja ennen kaikkea vähentää väyläkuormitusta.

Datakentän jälkeen viestissä on 15 bittiä pitkä tarkastuslaskentakenttä. Yleisesti tarkastuslaskennan ideana on lähettää viestin mukana viestin alkuosasta tietyllä algoritmilla laskettava tulos, joka voidaan laskea myös vastaanottavassa päässä ja verrata tuloksia keskenään. Mikäli vastaanottavassa päässä päädytään samaan tulokseen kuin viestin mukana lähetetyssä, todetaan datan säilyneen lähetyksen aikana muuttumattomana. CAN2.0-standardissa tarkastuslaskenta toteutetaan CRC (engl. cyclic redundancy check) algoritmin avulla siten, että viestin alkuosa, jonka perään on asetettu 15 nollaa, jaetaan kaavassa 1 esitetyllä polynomimuodossa esitetyllä binääriluvulla ja saatu jakojäännös kertoo tarkistuslaskennan tuloksen. Koska perinteinen jakaminen ei ole mahdollista binääriluvuilla käytetään XOR-operaattoria sen sijaan. (Di Natale et al. 2012 s. 13–21)

$$G(x) = x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1 \quad (1)$$

Käytetyn tarkastuslaskentamenetelmän toteuttaminen ohjainlaitteelle toteutetaan tyypillisesti LFSR:llä (engl. Linear Feedback Shift Register), jolloin menetelmä on analoginen jakokulmassa laskemisen kanssa (GHS Infotronic 2023 & Microchip 2000). Huomioitavaa kuitenkin on, että CAN-viestin alkuosaa yhdistettynä viiteentoista nollaan voidaan harvoin käsitellä yhtenä binäärilukuna, sillä sen pituus ylittää usein tuettujen datatyyppien pituuden. Esimerkiksi Codesys-kehitysympäristön tukema binääriluvun maksimipituus on 32 bittiä tai 64 bittiä riippuen käytetystä ohjainlaitteesta. Pisimmillään viestin alkuosa yhdistettynä 15 nollaan voi olla täyspitkässä jatketussa kehyksessä 118 bittiä. Keino kiertää edellä mainittu haaste on käyttää esimerkiksi taulukkolaskentaa, jossa jokaisella alkiolla merkitään binääriluvun yhtä bittiä.

CAN-väylässä käytetyn tarkastuslaskenta-algoritmin Hammingin etäisyys on kuusi, mikä mahdollistaa viiden väärän bitin tunnistamisen viestin alusta CRC-kenttään asti. Mikäli virheet ovat peräkkäin voidaan maksimissaan 15 bittiä pitkä purskevirhe tunnistaa. Analyyttisesti tästä voidaan johtaa CAN-standardin mukainen $3x10^{-5}$ todennäköisyys sille, että täyspitkässä viestissä on enemmän kuin viisi virhettä ja virheellistä viestiä ei tunnisteta. Huomioitavaa kuitenkin on, että viesteissä, joihin on jouduttu lisäämään bittejä käytössä olevan bit stuffing -menetelmän vuoksi, voi huonoimmassa tapauksessa toteutunut Hammingin etäisyys olla ainoastaan kaksi. Tällöin vain yksi bittivirhe on mahdollista 100 % varmuudella tunnistaa. (Di Natale et al. 2012 s. 20, 121–132)

Koska bit stuffing -menetelmän takia raakaviestin pituus muuttuu, analyyttisen todennäköisyyden johtaminen virheellisen viestin hyväksymiselle on haastavaa. Simulaatiotutkimuksella on kuitenkin tietyin rajoituksin osoitettu, että todennäköisyys virheellisen viestin hyväksymiselle voi jäädä jopa reilun dekadin verran pienemmäksi verrattuna CAN-standardin ilmoittamaan. Tämä johtuu siitä, että tarkastuslaskennalla huomattu virhe ei ole ainoa syy viestin hylkäämiselle. Raakaviestin tulee noudattaa muun muassa viestikehyksen formaattia sekä oikeata bit stuffing -käytäntöä. (Di Natale et al. 2012 s.121–132)

Tarkastuslaskentakentän jälkeen viestikehyksessä tulee aina väistyvien erotinbittien välissä ACK-bitti (engl. Acknowledge bit). ACK-bitti asetetaan väistyväksi lähtevään viestiin, mutta mikäli vastaanottava solmu toteaa tarkastuslaskennan olevan oikein, ylikirjoittaa vastaanottava solmu bitin tilan dominoivaksi. ACK-bitin tarkoitus onkin viestiä lähettävälle solmulle, että viesti on ylipäätänsä vastaanotettu ja siinä ei ole virheitä. (Di Natale et al. 2012 s. 13–21) ACK-bittiin liittyen huomioitavaa on, että tilanteessa, jossa vastaanottavia solmuja on useampi, ei voida tietää saivatko kaikki vastaanottavat solmut viestin vastaan vai vain yksi vastaanottavista solmuista, sillä nimensä mukaisesti dominoivan bitin tila ylikirjoittaa väistyvän bitin tilan.

Lopuksi erottimien välisen ACK-bitin jälkeen viestissä tulee seitsemän peräkkäistä väistyvää bittiä, joilla määritellään viesti päättyneeksi. Mikäli väylään lähetetään heti uusi viesti perään, tulee lähetyksessä olla lyhyt tauko. Tämä näkyy väylällä vähintään kolmena väistyvänä IFS-bittinä (engl. interframe spacing bit), jota myöhemmin kutsutaan tekstissä välimerkiksi. (Di Natale et al. 2012 s. 13–21)

4.2.2 Muut kehykset

Kuten aikaisemmin on mainittu, datan lähettämiseen käytettävien viestikehyksien lisäksi väylästandardissa on määritetty kehys datapyyntöön sekä ylikuormitus- ja virhetilanteille. Nimensä mukaisesti datapyyntökehyksellä voidaan kysyä dataa toiselta solmulta. Se noudattaa datan lähettämiseen käytettyä viestikehystä, mutta

- viestin identifiointikenttään asetetaan sen viestin tunniste, joka halutaan vastaanottaa,
- datapyynnöstä kertova RTR-bitti asetetaan dominoivaksi,
- datakentän pituudesta kertovaan DLC-kenttään asetetaan tieto siitä, kuinka pitkä vastaanotettava viesti on, ja
- itse datakenttä asetetaan tyhjäksi. (Di Natale et al. 2012 s. 17)

Datapyynnön lähettämisen jälkeen solmu, jonka on määrä lähettää kyseisen tunnisteen viesti, lähettää erillisen viestin datan lähettämiseen käytettyjen viestikehysten mukaisesti.

Ylikuormituskehyksellä puolestaan viestitään toisille solmuille CAN-lähetin–vastaanottimen ylikuormituksesta eli tilanteessa, jossa vastaanotettavia viestejä ei ehditä prosessoida samaa vauhtia kuin ne saapuvat. Ylikuormituksesta ilmoittaminen väylälle tapahtuu lähettämällä 6 peräkkäistä dominoivaa bittiä ensimmäisen tai kahden ensimmäisen IFS-bitin jälkeen ilman bit stuffing -menetelmää ja tämän perään 8 väistyvää bittiä pitkä ylikuormituskehyksen erotin. Kun ylikuormituskehys otetaan vastaan, siirtyy solmu hetkeksi kuuntelutilaan. Tyypillisesti nykyaikaiset CAN-lähetin–vastaanottimet ovat niin kehittyneitä, että niiden viestien prosessointinopeus on nopeampi kuin väylän tiedonsiirtonopeus, minkä vuoksi ylikuormituskehystä ei tarvitsee lähettää. Merkittävää kuitenkin on ylikuormituskehyksen ymmärtäminen tilanteissa, joissa väylällä on myös vanhempia CAN-laitteita. (Di Natale et al. 2012 s. 17)

Vikakehys taas ei käytännössä ole oikea viestikehys vaan pikemminkin seuraus virheellisestä toiminnasta. Vikakehyksen lähettäminen alkaa välittömästi, kun virhe huomataan, jokaisen virheen huomaavan solmun toimesta, ja se sisältää joko kuusi dominoivaa tai väistyvää bittiä pitkän vikamerkin ja sen perään kahdeksan väistyvää bittiä pitkän erottimen. Huomattavia virheitä voivat olla:

- viestikehyksestä poikkeaminen,
- bit stuffing -menetelmän virheellinen käyttö
- ACK-virhe, kun viestiä ei vastaanoteta ja ACK-bitti jää viestiin väistyväksi,
- bittivirhe, kun lähettävä laite toteaa lähettäneessä virheellisen bitin väylälle
- CRC-virhe, kun CRC-laskennan tulos poikkeaa viestin mukana tulleesta. (Di Natale et al. 2012 s. 21–22)

Kun vikamerkki lähetetään väylälle, huomaavat muut väylän laitteet sen bit stuffing -menetelmän virheenä tai viestikehyksestä poikkeamisena ja lähettävät täten oman vikakehyksensä väylälle heti alkuperäisen vikamerkin jälkeen. Täten vika näkyy väylällä kahtena peräkkäisenä vikamerkkinä ja sitä seuraavana erottimena. Tämän jälkeen kaikki solmut viivästyttävät viestin lähettämistä toisella kahdeksalla bitillä ja sitä seuraa välimerkki. Virheelliseksi merkattu viesti lähetetään uudestaan automaattisesti alkuperäisen lähettäjän toimesta normaalin viestikehyksen mukaisesti. (Di Natale et al. 2012 s. 22–23)

Vikamerkki itsessään riippuu laitteen tilasta. Vikamerkin ollessa kuusi peräkkäistä dominoivaa bittiä puhutaan aktiivisestä vikamerkistä, ja se lähetetään väylälle, kun laite huomaa virheen, mutta ei epäile omassa toiminnassaan olevan vikaa eli toisin sanoen se on aktiivisessa tilassa. Kun se on puolestaan 6 väistyvää bittiä, puhutaan passiivisesta vikamerkistä ja se lähetään väylälle vian huomaamisen yhteydessä, kun oman toiminnan epäillään olevan virheellistä eli laitteen tila on passiivinen. Kummassakaan tilassa solmun toimintaa ei kuitenkaan ole rajoitettu. (Di Natale et al. 2012 s. 22–23)

Laitteen tila puolestaan määritetään lähetys- ja vastaanottovirheitä laskevilla CAN-lähetin–vastaanottimen sisäisillä laskureilla. Laskureiden arvoa kasvatetaan useimmissa virhetapauksissa kahdeksalla, kun virhe huomataan, ja arvoista vähennetään tietyin poikkeuksin yksi, kun solmu onnistuu vastaanottaman tai lähettämään viestin onnistuneesti. Mikäli toisen laskurin arvo kasvaa yli 128, siirrytään aktiivisesta virhemerkistä passiiviseen virhemerkkiin. Lopulta, jos toisen laskurin arvo kasvaa yli 256, siirtyy solmu pois päältä. (Di Natale et al. 2012 s. 22–23)

4.3 Ylemmät kerrokset

Kuten luvuista 4.1 ja 4.2 voi huomata, ISO 11898-standardin ulkopuolelle jää huomattava määrä asioita, jotka täytyy määrittää, jotta eri valmistajien laitteiden välinen tiedonsiirto toimisi sellaisenaan eikä laitekohtaisia ratkaisuja tarvittaisi. Jotta tiedonsiirtoratkaisusta saadaan toimiva, pitäisi ISO 11898-standardin lisäksi määrittää ainakin:

- väylänopeuden ja bittien ajoitukseen liittyvät parametrit,
- käytetyt liittimet,
- viestikehyksen tunnistekentän käyttäminen,
- datatyypit ja suurempien kuin 8 bittiä sisältävien parametrien koodaaminen ja
- istunnon ja verkon hallinta sekä monitorointi.

Näiden lisäksi tiedonsiirtoratkaisun käyttöä ja käyttöönottoa voitaisiin helpottaa lukuisilla toiminnoilla, kuten palveluilla esimerkiksi ajan synkronoitiin sekä ennalta määritetyillä viestirakenteilla tyypillisille parametreille, joita halutaan lähettää tai kysyä. Koska nämä usein riippuvat sovelluskohteesta, on ylempiä standardeja syntynyt useampia.

4.3.1 SAE J1939

Ylemmän tason standardeista Society of Automotive Engineers – organisaation standardoima SAE J1939 (2023) on suunniteltu ajoneuvojen ja työkoneiden sekä niiden oheislaitteiden kuten trailerien ja generaattorien kenttäväyläksi, mutta sen suosio on kasvanut

myös esimerkiksi tehdasautomaatio- ja meriteollisuuden keskuudessa. Standardin tavoitteena on ohjata tiettyä komponenttia, kuten dieselmoottoria, standardoiduilla viesteillä, jolloin komponentin ohjaamisen toimintatavat eivät riipu komponentin valmistajasta. Näin ollen tietyn komponentin käyttöönotto tai vaihtaminen toisen valmistajan vastaavaan yksinkertaistuu. Lisäksi standardissa määritetään diagnostiikkarajapinta ja -käytännöt, joiden tavoitteena on mahdollistaa ajoneuvojen ja työkoneiden diagnosointi riippumatta sen valmistajasta. Haasteena standardin käytölle voidaan nähdä olevan sen maksullisuus.

Itse standardi on erittäin laaja ja koostuu tällä hetkellä ylemmän tason dokumentista ja sen 24 osasta. Osa standardin osista on päällekkäisiä ja niistä oleellisten valitseminen tapahtuu sovelluskohtaisten dokumenttien avulla. Tällä hetkellä sovelluskohtaisia dokumentteja on maantiekulkuiselle raskaalle kalustolle (J1939-1), maatalous-, metsä- sekä maastotyökoneille (J1939-2) sekä meriteollisuuden tarpeisiin (J1939-5). Koska maatalous-, metsä- sekä maastotyökoneiden tiedonsiirtoratkaisuissa käytetään yhä enenevissä määrin ISOBUS-pohjaisia ratkaisuja, johon myös J1939-2 osa viittaa, esitellään kuvassa 8 tämän työn kannalta oleellisemman J1939-1 mukaiset standardin osat. (J1939 2023)

Diagnostiikkan toteutus	J1939-3
Diagnostiikka rajapinnan (OBD) yhteensopivuus testit	J1939-84
Verkon hallinta	J1939-81
7. Sovelluskerros	J1939-71, 73, 76
6. Esitystapakerros	–
5. Istuntokerros	–
4. Kuljetuskerros	–
3. Verkkokerros	J1939-31
2. Siirtokerros	J1939-21
1. Fyysinen kerros	(J1939-11 TAI 14 TAI 15), 13, 16
Digitaalinen liite viestirakenteista	J1939DA

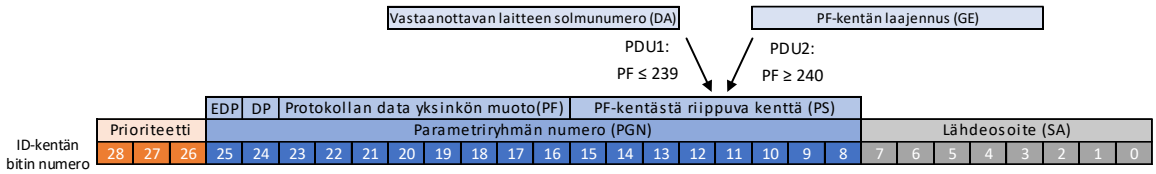
OSI-mallin (ISO/IEC 7498-1) kerrokset

Kuva 8. *Tarvittavat standardin osat SAE J1939-1 (2021) spesifikaation mukaan*

OSI-mallin kerroksista J1939-1 spesifioi kolme eri fyysistä kerrosta kuvaavaa standardin osaa perustuen ISO 11898 mukaan määritettyyn fyysiseen kerrokseen. Näistä J1939-11 spesifioi fyysisen kerroksen 250 kbit/s baudinopeudelle käyttäen suojattua ja kierrettyä parikaapelia ja osa 15 spesifioi baudinopeudeksi saman 250 kbit/s, mutta fyysiseksi me-

diaksi suojaamattoman kierretyn parikaapelin. Osa 14 spesifioi fyysisen kerroksen baudinopeudeksi 500 kbit/s, mutta ei ota kantaa kaapelin suojaukseen vaan esittää noudatettavaksi valinnan mukaan osissa 11 ja 15 esitettyjä kriteereitä. Fyysisen median tyypin sekä väylänopeuden ja siihen liittyvien parametrien, kuten nimellisbittiajan, lisäksi osissa esitetään vaatimukset väylän ja sen haaroituksien maksimipituuksille, solmujen eli laitteiden määrälle, liittimille, kaapeleille sekä muille ISO 11898-standardissa auki jätetyille kohdille. Osa 13 puolestaan spesifioi diagnostiikkaliittimen ja osa 16 harvemmin laitteisiin toteutetun automaattisen baudinopeuden etsimisen. (J1939-1 2021)

Siirtokerrosta kuvaava osa J1939-21 puolestaan spesifioi käytettäväksi normaalissa viestinnässä ISO 11898 mukaista jatkettua viestikehystä ja tarkentaa kuinka CAN-viestin identifiointikenttä muodostetaan. J1939-standardi uudelleennimeää jatketun viestikehyksen viestin protokollan datayksiköksi (engl. protocol data unit, PDU) ja sen datakentän parametriryhmäksi (engl. parameter group, PG). Standardin perusidea on käyttää standardoituja parametriryhmiä, jotka määrittävät tietyt parametrit tietyllä tavalla viestin datakenttään. Näillä parametriryhmillä on ainutlaatuinen järjestysnumero (engl. parameter group number, PGN), joka löytyy viestin identifiointikentästä, viestin prioriteetin ja lähdeosoitteen (engl. source address, SA) eli lähettävän laitteen solmunumeron lisäksi, kuten kuvassa 9 on esitetty. (J1939-21 2022)



Kuva 9. *J1939 viestikehyksen identifikaatiokenttä*

Parametriryhmänumero jakautuu vielä datasivuun (engl. data page, DP) ja sen jatkoon (engl. extended data page) sekä protokollan datayksikön muodon ilmaisevaan kenttään (engl. protocol data unit format, PF) ja protokollan datayksiköstä riippuvaan kenttään (engl. protocol data unit specific, PS). Protokollan datayksikön muotoja on kaksi. Mikäli PF-kenttä saa arvon väliltä 0–239, ilmoitetaan PS-kentällä vastaanottavan laitteen solmunumero (engl. destination address, DA) ja mikäli PF-kentän arvo on välillä 240–255, ilmoitetaan PS-kentällä PF-kentän laajennus (engl. group extension, GE). Toisella PDU-tyypillä lisätään mahdollisten parametriryhmien määrää, mikä on myös datasivun tarkoitus. Tulevaisuudessa parametriryhmien määrää voidaan kasvattaa myös jatketulla datasivulla, joka tällä hetkellä asetetaan aina väistyväksi, kun normaaleja J1939-viestejä lähetetään. (J1939-21 2022)

Merkittävin ero PDU1- ja PDU2-tyyppien välillä on, että PDU1-tyypin viestit voivat olla laitteelta-laitteelle tai yleislähetyksiä eli laitteelta kaikille, mutta PDU2-tyypin viestit ovat

aina tyypiltään yleislähetyksiä. PDU1-tyypin viesti on tarkoitettu kaikille, kun vastaanotettavan laitteen solmunumeroksi asetetaan 255. PDU1-tyypin yleislähetys tulee kuitata kaikkien väylällä olevien laitteiden toimesta, kun puolestaan PDU2-tyypin viestissä samaa vaatimusta ei ole. Molemman tyyppisiä viestejä voidaan kuitenkin lähettää tai pyytää viestikehyksen RTR-bitin avulla. (J1939-21 2022)

Verkkokerrosta kuvaava osa 31 puolestaan määrittää, kuinka CAN-väylä segmentoidaan eli jaetaan tarpeen mukaan useampaan kerrokseen ja esittää siihen liittyvien laitteiden, kuten reitittimien, toistimien ja siltaimien spesifikaation ja toimintojen, kuten tunneloinnin, viestien suodattamisen ja estämisen, toiminnan. Osa 31 määrittää myös, kuinka tiedonsiirtoratkaisulla lähetetään parametriryhmiä, jotka eivät mahdu yhteen CAN-viestiin. Tiivistettynä J1939-standardin siirtoprotokolla (engl. transport protocol, TP) mahdollistaa maksimissaan 1785 tavua pitkän parametriryhmän lähettämisen useammalla viestillä. Kun yli 8 tavua pitkä parametriryhmä lähetetään, lähtee väylälle aluksi automaattisesti TP.CM (engl. transport protocol connection management) -viesti, jossa käy ilmi, kuinka monta viestiä lähetykseen kuuluu. TP.CM-viestin jälkeen dataa sisältävät TP.DT (engl. transport protocol data transfer) -viestit lähetetään perään. (J1939-31 2023) Koska yksinkertaisissa sovelluksissa CAN-väylän segmentointia ei tarvita, ei tässä työssä käsitellä tarkemmin verkkokerroksen määrittelyjä.

OSI-mallin kuljetus-, istunto- ja esitystapakerroksiin J1939-standardissa ei vielä ole julkaistu osaa tämän työn kirjoitushetkellä. Näitä seuraava sovelluskerros koostuu useammasta osasta. Yleiset normaaliin tiedonsiirtoon liittyvään asiat on spesifioitu osassa 71, diagnostiikkaan liittyvät osassa 73 ja toiminalliseen turvallisuuteen liittyvät osassa 76. (J1939 2023). Alun perin osa 71 listasi kaikki parametriryhmät ja antoi niissä esiintyville parametreille omat tunnisteet (engl. suspect parameter number, SPN). Nykyisin listaukset löytyvät digitaalisesta J1939-DA-liitteestä, ja osassa 71 käsitellään yleisellä tasolla periaatteita datan asettelulle viestin datakenttään ja erityyppisten datamuotojen, kuten ASCII (engl. American Standard Code for Information Interchange), käyttöä (J1939-71 2022).

Identifiointikenttään tehdyin määrittelyin J1939-standardi mahdollistaa tällä hetkellä 8672 uniikkia parametriryhmää. Huomioitavaa on, että kaikki parametriryhmän numerot eivät ole varattu ja niitä on tarkoituksella jätetty laitevalmistajien laitekohtaisille parametriryhmille. Kuvassa 10 on esitetty esimerkki yhdestä standardoidusta parametriryhmästä. (J1939-DA 2023)

Parametriryhmän numero: 64982
Parametriryhmän nimi (lyhenne) : Yksinkertaisen ohjainsauvan viesti 1 (BJM1)
Vakio lähetysväli: 100 ms
Vakio prioriteetti: 3
Datasivu: 0
Data yksikön muodon kenttä, PF: FD
PF-kentästä riippuva kenttä, PS: D6

***** 00b = Nappi ei ole painettu
01b = Nappi on painettu
10b = Vika, 11b = Ei saatavilla

SPN	Kuvaus	Tyyppi [yksikkö]	Skaalaus (offset) / tilat	Aloittava bitti viestissä (pituus bitteinä)
2675	X-akseli keskiasennossa	ENUM	00b = Keskiasennossa 01b = Poikeutettu keskiasennosta 10b = Vika, 11b = Ei saatavilla	0 (2)
2670	X-akseli poikkeutettu negatiiviseen suuntaan (vasemmalle)	ENUM	00b = Ei poiketettu negatiiviseen suuntaan 01b = Poikeutettu negatiiviseen suuntaan 10b = Vika, 11b = Ei saatavilla	2 (2)
2665	X-akselin poikkeutettu positiiviseen suuntaan (oikealle)	ENUM	00b = Ei poikeutettu positiiviseen suuntaan 01b = Poikeutettu positiiviseen suuntaan 10b = Vika, 11b = Ei saatavilla	4 (2)
2660	X-akselin asema	Parametri [%]	Lineaarisesti skaalattu välille 0–1000, 0,1 % per bitti, ei offsettia (0 = keskiasema, 1000 = ääriasema, 1022 = virhe)	6 (10)
2676	Y-akseli keskiasennossa	ENUM	00b = Keskiasennossa 01b = Poikeutettu keskiasennosta 10b = Vika, 11b = Ei saatavilla	16 (2)
2671	Y-akseli poikkeutettu negatiiviseen suuntaan (taakse)	ENUM	00b = Ei poiketettu negatiiviseen suuntaan 01b = Poikeutettu negatiiviseen suuntaan 10b = Vika, 11b = Ei saatavilla	18 (2)
2666	Y-akselin poikkeutettu positiiviseen suuntaan (eteen)	ENUM	00b = Ei poikeutettu positiiviseen suuntaan 01b = Poikeutettu positiiviseen suuntaan 10b = Vika, 11b = Ei saatavilla	20 (2)
2661	Y-akselin asema	Parametri [%]	Lineaarisesti skaalattu välille 0–1000, 0,1 % per bitti, ei offsettia (0 = keskiasema, 1000 = ääriasema, 1022 = virhe)	22 (10)
2681	Y-akseli poiketettu ja lukittu	ENUM	00b = Ei ole lukittu 01b = Lukittu 10b = Vika, 11b = Ei saatavilla	36 (2)
2680	X-akseli poiketettu ja lukittu	ENUM	00b = Ei ole lukittu 01b = Lukittu 10b = Vika, 11b = Ei saatavilla	38 (2)
2688	Nappi 4 painettu	ENUM	*	40 (2)
2687	Nappi 3 painettu	ENUM	*	42 (2)
2686	Nappi 2 painettu	ENUM	*	44 (2)
2685	Nappi 1 painettu	ENUM	*	46 (2)
2692	Nappi 8 painettu	ENUM	*	48 (2)
2691	Nappi 7 painettu	ENUM	*	50 (2)
2690	Nappi 6 painettu	ENUM	*	52 (2)
2689	Nappi 5 painettu	ENUM	*	54 (2)
2696	Nappi 12 painettu	ENUM	*	56 (2)
2695	Nappi 11 painettu	ENUM	*	58 (2)
2694	Nappi 10 painettu	ENUM	*	60 (2)
2693	Nappi 9 painettu	ENUM	*	62 (2)

Kuva 10. *Esimerkki J1939-standardin parametriryhmästä*

OSI-mallin kerroksien lisäksi standardissa on osa 81, joka spesifioi uniikin 8 tavuisen valmistaja- ja sovelluskohtaisen nimen identifiointitarkoitusta varten. Kyseistä nimeä ei käytetä tiedonsiirtoon suoranaisesti, vaan sen avulla laite tunnistetaan ja sille määritetään solmunumero ja sen lähettämät ja vastaanottamat parametriryhmät. Toisin sanoen standardin osa 81 standardoi tietyn tyyppisen komponentin ohjaamisen riippumatta sen

valmistajasta, mikä esimerkiksi mahdollistaa eri valmistajien trailerien kytkemisen vetoautoon. Laitteiden tunnistaminen tapahtuu väylän alustuksessa ja sitä kutsutaan solmunumeroiden varaamiseksi (engl. address claiming). (J1939-81 2017) Mikäli kaksi samantyyppistä laitetta kytketään väylälle, on niillä vakiona sama solmunumero, joka aiheuttaa alustuksen epäonnistumisen. Standardin osa määrittää kuitenkin keinon kyseisen ristiriidan ratkaisemiseksi muokkaamalla laitteiden solmunumeroa automaattisesti (J1939-81 2017). Kyseinen ominaisuus on merkittävä varsinkin tilanteissa, joissa väylällä olevien laitteiden määrä muuttuu normaalissa toiminnassa. Tällainen tilanne saattaisi tulla vastaan esimerkiksi, kun vetoautoon kytketään useampi perävaunu. On kuitenkin huomioitavaa, että dynaaminen uuden solmunumeron saaminen on valinnainen ominaisuus eikä sitä välttämättä löydy jokaisesta J1939-yhteensopivasta laiteesta.

Näiden lisäksi standardista löytyy J1939-3 osa, jossa esitellään ohjeistus diagnostiikkaan käytetyn OBD-protokollan (engl. on board diagnostic) toteuttamiseen ja osa J1939-84 diagnostiikka rajapinnan testaamiseksi (J1939-1 2021). Nämä rajataan myös tämän työn ulkopuolelle, sillä tutkimusalustaan ei ole tarkoituksen mukaista implementoida standardin mukaista diagnostiikkarajapintaa. Myytävien laitteiden tapauksessa standardoitu diagnostiikkarajapinta on järkevä toteuttaa, sillä se vähentää tarvetta laitekohtaiselle tuntemukselle ja mahdollistaa samojen diagnostiikkalaitteiden käyttämisen riippumatta laitteen valmistajasta.

4.3.2 CANopen

CANopen puolestaan on nimensä mukaisesti avoin ylemmän tason standardi, jota ylläpitää CiA (engl. CAN in automation) eli maailmanlaajuinen CAN-väylän käyttäjien organisaatio (CiA 2023a). CANopen ei rajaudu lähtökohtaisesti pelkästään ajoneuvo- ja työkonesovelluksiin vaan se on suunniteltu käytettäväksi kaikissa sovelluskohteissa, joissa CAN-väylää voidaan hyödyntää. Koska sovelluskohtaista rajausta ei ole, on standardiin määritelty valmiit viestit vain laitteiden väyläkonfigurointiin sekä laiteprofiilit vain yleisesti kaikenlaisissa järjestelmissä käytetyille komponenteille, kuten esimerkiksi enkoodereille. Tämän vuoksi sovelluskohtaisten laitteiden käyttöönotto tai sen vaihtaminen toisen valmistajan vastaavaan voi vaatia enemmän aikaa verrattuna J1939-yhteensopiviin laitteisiin. Yleisesti CANopen-laitteiden käyttöönottoa on kuitenkin pyritty helpottamaan määrittelemällä EDS-tiedostoformaatti (engl. electronic data sheet) laitteen mukana toimitettavaksi. Se pitää sisällään laitteelle lähetettävät ja siltä vastaanotettavat viestit sekä laitteen parametrilistauksen (CiA-306 2005). EDS-tiedosto voidaan tyypillisesti avata suoraan väyläjärjestelmien kehitysympäristössä.

Toisin kuin J1939-standardissa erillistä CANopen-standardin osat kokoavaa ylemmän tason dokumenttia ei ole julkaistu. CiA:n yli 200 julkaistusta CANopen-standardin keskiöissä on CiA-301 julkaisu, joka käytännössä määrittää kokonaisuudessaan tiedonsiirtoratkaisun OSI-mallin ylempien kerroksien osalta sekä täydentää ISO 11898-standardin mukaista siirtokerrosta. ISO 11898 mukaisen fyysisen kerroksen täydennykset on puolestaan esitetty dokumenteissa CiA-102, -106 ja -303–1. Muita merkittäviä dokumentteja ovat muun muassa CiA-302 ja sen osat, jotka määrittävät CANopen-standardin vapaaehtoisesti toteutettavat ominaisuudet, CiA-306, joka määrittää EDS-tiedoston käytön ja CiA-4** -alkuiset dokumentit, joilla määritetään laiteprofiileja yleisesti kaikenlaisissa järjestelmissä käytetyille komponenteille. Näiden lisäksi julkaisuja löytyy esimerkiksi testaamiseen, diagnosointiin, turvallisuuteen liittyen sekä spesifikaatioita tiettyihin sovelluskohteisiin. (CiA 2023b)

Fyysisen kerrokseen liittyvistä dokumenteista CiA-102 (1994) määrittää väylän baudinopeuden ja sen perusteella bittiajoitukseen liittyvät parametrit sekä esimerkiksi väylän maksimipituuksien suositukset. CANopen-standardissa käyttäjä voi itse valita tilanteeseen sopivan baudinopeuden taulukossa 3 esitetyistä vaihtoehdoista, jolloin on mahdollista optimoida väylän nopeus perustuen todelliseen väyläkuormitukseen ja esimerkiksi mahdollistaa pidempi väylä, jos pienempää baudinopeutta voidaan käyttää. Huomioitavaa kuitenkin on, että standardin mukaan CANopen-laitteissa baudinopeuksista ainoastaan 20 kbit/s tulee olla tuettuna ja muiden nopeuksien tukeminen on valinnaista.

Taulukko 3. *CiA-102 (1994) mukaiset suositukset väylän pituudelle eri siirtonopeuksilla*

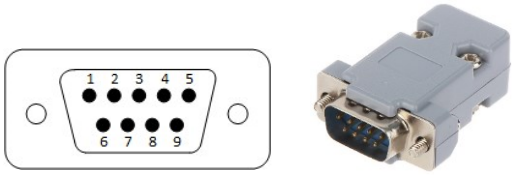
Baudinopeus	Nimellisbittiaika	Väylän pituus
10 kbit/s	100 µs	< 5000 m
20 kbit/s	50 µs	< 2500 m
50 kbit/s	20 µs	< 1000 m
125 kbit/s	8 µs	< 500 m
250 kbit/s	4 µs	< 250 m
500 kbit/s	2 µs	< 100 m
800 kbit/s	1,25 µs	< 50 m
1000 kbit/s	1 µs	< 25 m

CiA-102 määrittää 1000 kbit/s nopeudella toimivan väylän maksimipituudeksi 25 m, joka on merkittävästi ISO 11898 määrittämää teoreettista 40 m maksipituutta lyhyempi. Todellisuudessa saavutettava väylän maksimipituus riippuu tilannekohtaisesti monesta tekijästä, kuten liitoksien laadusta ja indusoituvan häiriön määrästä, kuten 4.1 luvussa on esitetty, eikä maksimipituudelle voida määrittää yhtä arvoa. Verrattaessa CANopen-standardia J1939-standardiin voidaan J1939-standardin todeta olevan vielä konservatii-

visempi, sillä J1939-11 (2016) määrittää 250 kbit/s nopeudella toimivan väylän maksimipituudeksi 40 m ja J1939-14 (2022) osa 500 kbit/s nopeudella toimivan väylän maksimipituudeksi 40–56 m riippuen solmujen määrästä. Konservatiivisuudella todennäköisesti pyritään vaikuttamaan J1939-tiedonsiirron robustisuuteen. J1939-standardin konservatiivisuus näkyy myös muissa ISO 11898-standardin fyysiseen kerrokseen tehtävissä täydennyksissä, sillä niissä spesifioidaan toleranssit hieman tiukemmin.

CiA-106 (2023) puolestaan spesifioi suuren määrän erilaisia liittimiä käytettäväksi CAN-väylässä. Standardoiduista liittimistä ehkä kaksi käytetyintä on kuvassa 11 esitetyt DB9 ja A-koodattu 5-napainen M12-liitin. Näistä DB9-liitintä käytetään usein tietokoneen CAN-väyläsovittimissa ja antureissa puolestaan tyypillinen ratkaisu on DB9-liitintä kompaktimpi A-koodattu 5-napainen M12-liitin. Ohjainlaitteissa ja muissa CAN-liitännäisissä laitteissa ei ole puolestaan yleistynyt mikään liitintyyppi. Useimmissa tapauksissa laitevalmistajat ovat valinneet tuoda CAN-väylään liittyvät johtimet muiden laitteeseen tulevien johtimien, kuten tehonsyötön ja IO-johtimien, kanssa saman liittimen avulla.

DB9-liitin



Napa	Signaali
1	Varattu
2	CAN_L
3	Maa, 0 V
4	Varattu
5	(Kaapelin suojavaippa)
6	(Maa, 0 V)
7	CAN_H
8	Varattu
9	(Syöttöjännite)

5-napainen M12-liitin

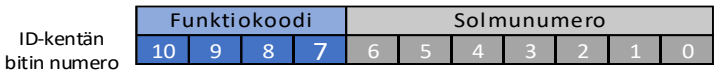
Napa	Signaali
1	(Kaapelin suojavaippa)
2	(Syöttöjännite)
3	Maa, 0 V
4	CAN_H
5	CAN_L

Kuva 11. *Kaksi esimerkkiä CAN-väylässä käytettävissä olevista liitimistä.*

Molempiin kuvassa 11 esitettyihin liittimiin on varattu napa maalle, jolla pyritään tasoittamaan mahdollisia maadoituseroja eri laitteiden välillä, jotta väyläjohtimien jännitetasot rekisteröityisivät oikein jokaisessa solmussa. Maadoituksen lisäksi molempien liittimien avulla voidaan vapaaehtoisesti välittää syöttöjännite ja jatkaa kaapeloinnin suojavaipan jatkuvuutta liittimien yli. Kaapelointiin tarkemmin liittyvä julkaisu CiA-303–1 (2023) spesifioi, että vapaaehtoisen syöttöjännitteen tulee olla 18–30 V välillä, jotta laite toimii 24 V tehonlähteillä. Muita CiA-303–1 (2023) julkaisun spesifioimia kohtia ovat muun muassa kaapeloinnin johtimien poikkipinta-alat ja resistanssit metriä kohden ja haaroituksien

maksimipituudet. Verrattaessa näitä J1939-standardin määrityksiin, voidaan jälleen huomata J1939-standardin olevan konservatiivisempi. Esimerkiksi J1939-11 (2016) edellyttää väyläkaapeloinnin johtimien poikkipinta-alaksi 0,75 mm^2 väylän maksimipituuden ollessa 40 m. CiA-303–1 (2023) edellyttää maksimissaan 40 m pitkälle väylälle käytettävän minimissään 0,25 mm^2 poikkipinta-alaltaan olevia johtimia.

OSI-mallin fyysistä kerrosta lukuun ottamatta muut kerrokset CANopen-standardissa käytännössä määrittää CiA-301, kuten aikaisemmin on mainittu. CiA-301 spesifioi CANopen-standardin käyttämään normaalia viestikehystä, mikä on ehkä merkittävin ero verrattuna J1939-standardiin. Sen 11 bitin identifiointikenttä, jota kutsutaan standardissa termillä kommunikaatio-objektin tunniste (engl. communication object identifier, COB-ID), jakautuu neljäbittiseen funktiokoodiin ja seitsemänbittiseen solmunumeroon, kuten kuvassa 12 on esitetty. Näistä funktiokoodi määrittää kommunikaatio-objektin tyypin ja solmunumerolla ilmoitetaan kommunikaatio-objektin tyypin mukaan joko lähettävän tai vastaanottavan laitteen solmunumero. Huomioitavaa on, että seitsemän bittiä rajaa CANopen-standardin tukemaan 127 eri solmua, sillä solmunumero nolla ei ole sallittu. (CiA-301 2011)



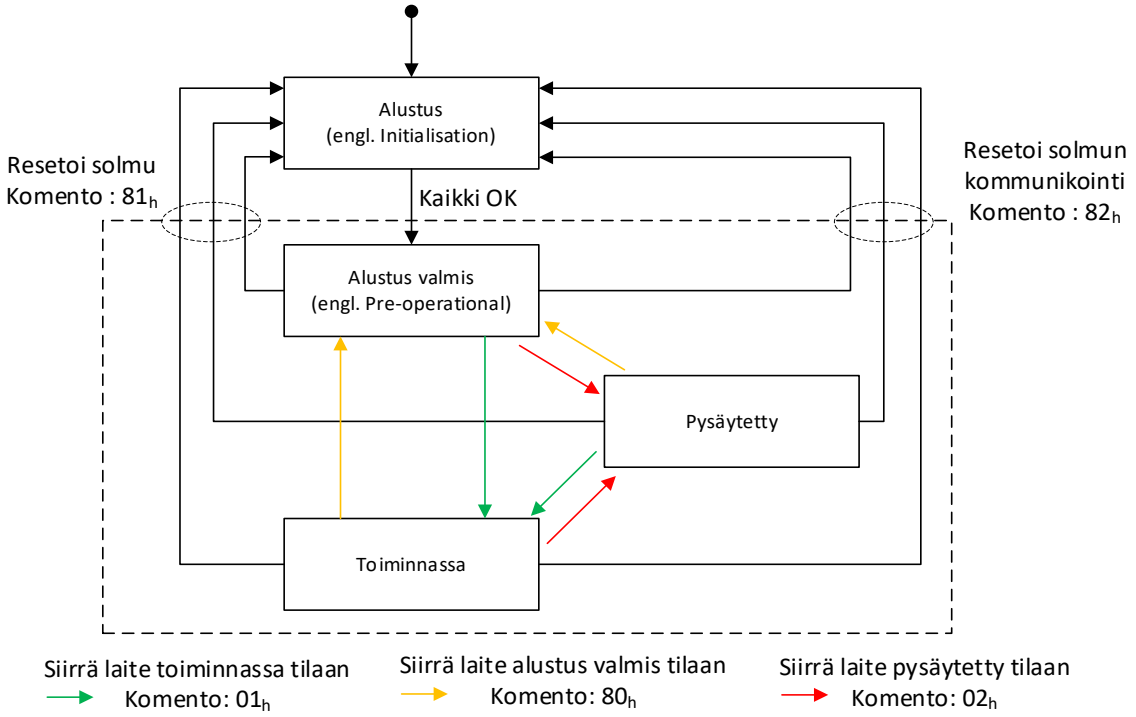
Kuva 12. *CANopen-standardin kommunikaatio-objektin tunniste*

Funktiokoodin määrittämä kommunikaatio-objekti pitää sisällään tietyn tyyppisen viestin datakentän allokaation ja ne on listattu funktionumeroineen taulukkoon 4. Huomioitavaa on, että kommunikaatio-objektien funktiokoodi määrittää ensisijaisesti viestin prioriteetin luvussa 4.2.1 esitetyn CAN-väylän priorisointimenetelmän mukaisesti. Taulukossa 4 kommunikaatio-objektit on lajiteltu prioriteetin mukaan laskevaan järjestykseen. Lisäksi huomioitavaa on, että tilanteessa, jossa kaksi solmua lähettää saman kommunikaatio-objektin, priorisoituu sen solmun viesti, jonka solmunumero on pienempi. (CiA-301 2011)

Taulukko 4. *CANopen identifiointikentän funktiokoodit*

Kommunikaatio objekti	Funktiokoodi	COB-ID
NMT	0000_b	0_h
SYNC	0001_b	80_h
EMCY	0001_b	80_h + solmunumero
TIME	0010_b	100_h
PDO1 (tx)	0011_b	180_h + solmunumero
PDO1 (rx)	0100_b	200_h + solmunumero
PDO2 (tx)	0101_b	280_h + solmunumero
PDO2 (rx)	0110_b	300_h + solmunumero
PDO3 (tx)	0111_b	380_h + solmunumero
PDO3 (rx)	1000_b	400_h + solmunumero
PDO4 (tx)	1001_b	480_h + solmunumero
PDO4 (rx)	1010_b	500_h + solmunumero
SDO (tx)	1011_b	580_h + solmunumero
SDO (rx)	1100_b	600_h + solmunumero
NMT error control	1110_b	700_h + solmunumero

Taulukossa 4 esitetyistä kommunikaatio-objekteista kaksi on varattu NMT-palvelulle (engl. network management service). NMT-palvelun idea on monitoroida ja ylläpitää CAN-väylän laitteiden tilaa isännäksi määritetyn laitteen toimesta. Tätä varten jokaisella CANopen-laitteella on sen tilan määrittävä kuvassa 13 esitetty tilakone, jonka tilaa isäntä voi vaihtaa NMT-kommunikaatio-objektilla. (CiA-301 2011)



Kuva 13. *CANopen-laitteen tilakone*

Laitteen ollessa päällä raportoi jokainen laite olemassaolostaan sydämenlyönniksi kutsutulla viestillä, joka lähetetään väylälle NMT error control -kommunikaatio-objektilla.

Isännän tehtävä on saattaa väylän alustuksen yhteydessä kaikki väylän laitteet toiminnassa-tilaan ja monitoroida muita laitteita. Mikäli tietyn laiteen sydämenlyöntiä ei löydy väylältä tai se ei esiinny tarpeeksi useasti, voi isäntä esimerkiksi pysäyttää kaikki väylän laitteet tai yrittää resetoida kyseisen laitteen. Toteutettava toiminnallisuus tilanteessa, jossa huomataan jokin laitteen poissaolo tai viallinen viesti, jää laitevalmistajan päätettäväksi. Sydämenlyönnin voi myös asettaa pois päältä tarvittaessa. (CiA-301 2011) Molempien NMT kommunikaatio-objektien rakenne on esitetty kuvassa 14.

0	Komento	Solmunumero (0h kaikille)
COB-ID	Datatavu 0	Datatavu 1

700h + solmunumero	Varattu (0h)	Solmunumero	Sydämmen lyönnin aikaväli	
COB-ID	Datatavu 0	Datatavu 1	Datatavu 2	Datatavu 3

Kuva 14. *NMT-kommunikaatio-objektien viestirakenne*

Tahdistusobjektin (engl. synchronization object, SYNC) idea on sovittaa solmujen toiminta yhteen. Sitä voidaan käyttää esimerkiksi anturidatan lähetyshetken optimointiin, jotta se saapuu ohjainlaitteelle juuri uuden ohjelmasyklin alkuun. Tahdistusobjektin toiminta ei perustu globaaliin kellonaikaan, vaan sitä varten on erillinen aikaobjekti (engl. time object). Aikaobjektilla lähetetään väylälle aika 48 bittisenä kokonaislukuna, jossa arvon muutos vastaa millisekuntia ja arvo nolla päivämäärän 1.1.1984 kellonaikaa 00.00. Aikaobjektia tarvitaan tilanteissa, joissa laitteiden toiminta on sidottu aikaan ja sillä estetään laitteiden omien kellojen vaeltaminen. Molemmat edellä mainitut objektit lähetetään yhden vapaasti valittavan solmun toimesta ja niiden käyttö on vapaaehtoista. (CiA-301 2011)

Niin sanottu hätätilanneobjekti (engl. emergency-object, EMCY) lähetetään väylälle, kun solmun sisäisessä toiminnassa tapahtuu virhe. Se lähetetään aina vain kerran yhtä vikaa kohden ja objekti sisältää virhekoodin, jollaisia standardi listaan vikakoodikirjastoon. Jokainen väylän laite voi olla hätätilanneobjektin tuottaja ja vastaanottaja. Jälleen tämänkin kommunikaatio-objektin käyttäminen on vapaaehtoista ja myös toiminnallisuus, kun vastaanottajaksi määritetty laite vastaanottaa kyseisen objektin, on valmistajan valittavana. Siitä huolimatta, että hätätilanneobjektia ei käytetä, tulee laitteen tukea vähintään kahta vikakoodia, joista toinen kuvaa yleistä virhettä ja toinen tilannetta, jossa vikakoodi ei ole aktiivinen tai se halutaan resetoida. Vikakoodit tai niiden nollaaminen onnistuu myös käyttäen seuraavaksi esiteltävää huoltodataobjektia. (CiA-301 2011)

Huoltodataobjekteilla (engl. service data object, SDO) puolestaan muutetaan laitteen parametreja tai niiden nykyistä arvoa kysytään. SDO-protokolla sisältää kaksi viestiä, joista

lähtevä sisältää muutospyynnön tai parametrin arvon kyselyn ja vastaanotettava kuittauksen muutokseen tai vastauksen parametrikyselyyn. Kumpikin viesteistä voi olla segmentoitu, mikäli lähetettävä data ei mahdu yhteen viestikehykseen. Lisäksi SDO-protokolla pitää sisällään virheviestin, joka lähetetään väylälle esimerkiksi laittoman kyselyn tai parametrimuutospyynnön tapauksessa tai esimerkiksi silloin kuin segmentoitu lähetys halutaan keskeyttää. (CiA-301 2011)

SDO-viestin COB-ID-kenttään tulee molemmissa viesteissä kohdelaitteen solmunumero eli sen solmun numero, jolle kysely tai muutospyyntö on tarkoitettu. Lisäksi molemmissa viesteissä RTR-bitti on dominoiva ja viesti sisältää 8 datatavua. SDO-viestien rakenne on esitetty kuvassa 15. Huomioitavaa on, että SDO-viesteissä käytetään nousevaa tavujärjestystä. (CiA-301 2011) Esimerkiksi indeksi 2010_h laitettaisiin viestiin siten, että datatavu 1 saisi arvon 10_h ja datatavu 2 arvon 20_h.

580h tai 600h + solmunumero	Komennon tunniste	Indeksi		Ala-indeksi	Data			
COB-ID	Datatavu 0	Datatavu 1	Datatavu 2	Datatavu 3	Datatavu 4	Datatavu 5	Datatavu 6	Datatavu 7

Kuva 15. *SDO-viestin rakenne*

SDO-viestirakenteen ensimmäinen datatavu esittää komentotunnisteen. Yleisesti ottaen komentotunnisteella määritetään esimerkiksi, onko viesti segmentoitu, jatkuuko lähetys seuraavaan viestiin ja kuinka moni datatavua on merkitsevä, mutta sen tarkempi rakenne riippuu SDO-viestistä. Erilaisia komentotunnisteen rakenteita on lähtevälle, vastaanotettavalle, niiden segmentoiduille versioille sekä SDO-vikaviestille. (CiA-301 2011)

Seuraavaksi SDO-viestissä esiintyvät indeksi ja alaindeksi viittaavat CANopen-standardin objektikirjastoon (engl. object dictionary, OD). Objektikirjaston idea on ryhmitellä parametreja indeksien alle ja antaa sen sisäisesti parametreille uniikki alaindeksi, mikäli yhden indeksin alle halutaan useampi parametri. Objektikirjasto indeksiavaruus on jaettu taulukon 5 esittämällä tavalla alueisiin. Osa alueista on laitevalmistajan vapaasti käytössä, mutta osa on standardisoitu, mitä laitevalmistajien tulee tukea. (CiA-301 2011)

Taulukko 5. *CANopen-standardin objektikirjasto*

Indeksialue	**objekti**
0000_h	Ei käytössä
0001_h – $001F_h$	Staattiset datatyypit
0020_h – $003F_h$	Kompleksiset datatyypit
0040_h – $005F_h$	Valmistajakohtaiset kompleksiset datatyypit
0060_h – $025F_h$	Laiteprofiilikohtaiset datatyypit
0260_h – $0FFF_h$	Varattu
1000_h – $1FFF_h$	Kommunikaatioprofiilin alue
2000_h – $5FFF_h$	Valmistajakohtaisen profiilin alue
6000_h – $9FFF_h$	Standardoitu laiteprofiilialue
$A000_h$ – $AFFF_h$	Standardoitu verkon muuttujien alue
$B000_h$ – $BFFF_h$	Standardoitu järjestelmän muuttujien alue
$C000_h$ – $FFFF_h$	Varattu

Objektikirjaston standardoiduista alueista datatyyppejä kuvaavat alueet määrittävät tuetut datatyypit ja niiden avulla määritetään muilla alueilla käytettyjen parametrien datatyyppi. Datatyyppi määrittää kuinka parametri määritetään bittien avuilla. Riippuen datatyypistä tietty jono bittejä tarkoittaa eri arvoa, jonka vuoksi datatyyppiin on syytä kiinnittää huomiota. Tieto parametrin datatyypistä on sidottu indeksiin, mikä tekee sen, että jokaisen mahdollisen indeksin tulee käyttää samaa datatyyppiä. (CiA-301 2011)

Kommunikaatioprofiilin alue määrittää nimensä mukaisesti CAN-väyläkonfiguroinnin eli mahdollistaa esimerkiksi solmunumeron ja väylänopeuden muuttamisen, mutta sieltä löytyy myös objekteja laiteen tunnistamista varten ja esimerkiksi aikaisemmin mainittu vikaobjekti indeksiltä 1001_h. Standardoidulta laiteprofiilialueelta löytyy puolestaan CiA-4** -alkuisten julkaisujen mukaiset parametrilistaukset sellaisille laitteille, joille laiteprofiili on määritelty, kuten esimerkiksi enkooderille tai proportionaaliselle hydrauliventtiilille. Standardoidut verkko- ja järjestelmämuuttujien alueet on varattu tulevaisuuden käyttöön, eikä nykyinen CiA-301 versio ota niihin kantaa. (CiA-301 2011)

Viimeisenä kommunikaatio-objekteista on prosessidataobjekti (engl. process data object, PDO), jolla on tarkoitus hoitaa itse toimintaan liittyvä kommunikaatio eli esimerkiksi mittauksien tai toimilaitekomentojen lähettäminen. Siinä missä SDO-protokollaan kuuluu aina kaksi viestiä, ei PDO-protokollassa käytetä kuin yhtä viestiä. Tästä poikkeuksena on datakyselyviestikehyksen käyttäminen RTR-bitin avulla, jonka PDO-protokolla mahdollistaa. (CiA-301 2011) Yhden viestin käyttämisen haittapuolena on, ettei viestin perille menemisestä saada varmuutta, mutta toisaalta protokollan vaatima kaista väylästä vähenee merkittävästi.

Taulukon 4 mukaisesti prosessidataobjekteja on määritetty jokaiselle solmunumerolle kahdeksan eli neljä vastaanotettavaa ja neljä lähetettävää prosessidataobjektia. Mikäli prosessiobjekteja tarvittaisiin enemmän, voidaan toinen solmunumero varata laitteen käyttöön. Huomioitavaa on, että laite voi ottaa vastaan toisen laitteen lähteväksi määritetyn prosessidataobjektin tai se voi lähettää toisen laitteen vastaanotettavaksi määritetyn prosessidataobjektin. COB-ID:n määrittämisessä käytetään funktiokoodin lisäksi aina sen laitteen solmunumeroa, jonka prosessidataobjekti on kyseessä. (CiA-301 2011)

Toisin kuin huoltodataobjektissa, prosessidataobjektin datakentän pituus on vapaasti valittavissa ja kaikki datakentän tavut ovat käytössä datan lähettämiseksi. Mikäli laitetyypille on määritetty standardoitu laiteprofiili määrittää se käytetyt prosessidataobjektit ja niiden datakentän käytön. (CiA-301 2011) Verrattuna J1939-standardiin merkittävä ero on, että CANopen vaatii prosessidatan latomisen viestiin siten, että ne varaavat aina

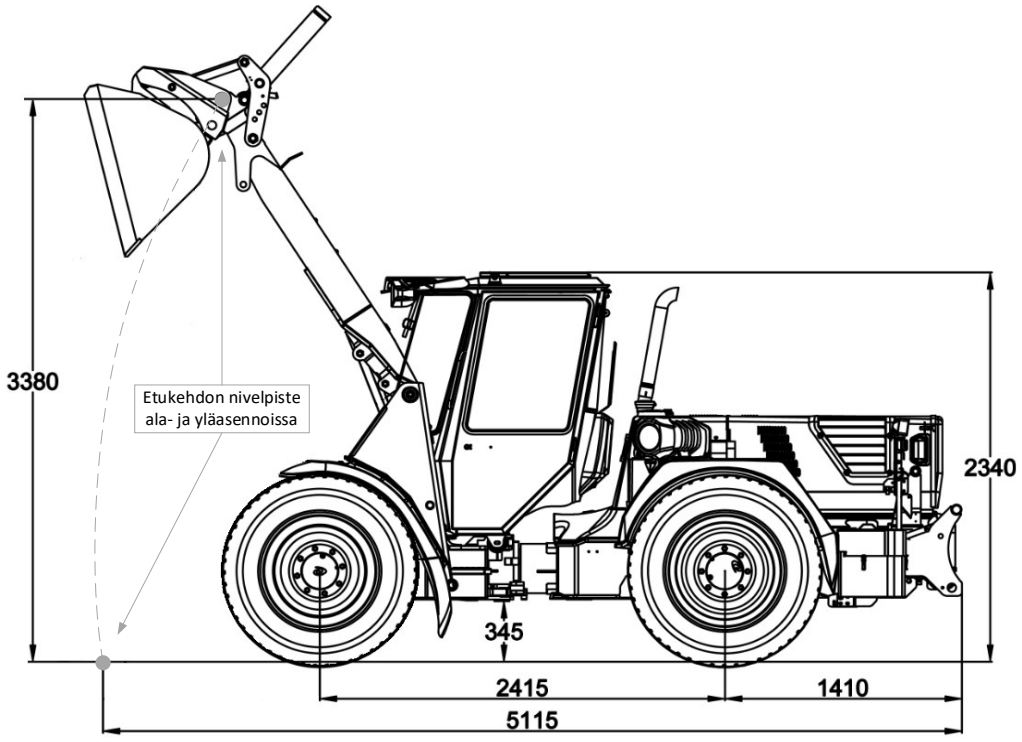
kokonaan parametrin käyttämät datatavut. Esimerkiksi kuvassa 10 esitetty J193- standardin parametriryhmän SPN 2666 alkaa bitistä 20 ja päättyy 21. Siitä seuraava SPN 2661 alkaa bitistä 22 ja päättyy bittiin 31. Lähetettäessä nämä parametrit käyttäen CANopen-standardin prosessidataobjektia varaisi kahdella bitillä määritettävä parametri kokonaisen datatavun ja kymmenellä bitillä määritettävä parametri siitä seuraavat kaksi datatavua. Toisin sanoen J1939-standardi mahdollistaa parametrien latomisen viestirakenteeseen tehokkaammin kuin CANopen.

Prosessidatan lähettäminen puolestaan voidaan toteuttaa useammalla tavalla. Lähetys voi olla tahdistettu, tahdistamaton tai tapahtumapohjainen. Tahdistettu lähetys perustuu tahdistusobjektiin, jolloin viesti lähetään, kun tahdistusobjekteja huomataan väylältä haluttu määrä. Tahdistamaton lähetys tapahtuu taas laitteelle ennalta määritetyin aikavälein ja tapahtumapohjainen tietystä laukaisevasta tekijästä. Laukaiseva tekijä voi olla esimerkiksi prosessidataobjektin sisältämien parametrien arvojen muuttuminen. Tyypillisesti tapahtumapohjaisessa lähettämisessä määritetään kahden peräkkäisen viestin lähettämisvälin minimiaikaa, jotta prosessidataobjekti ei lähde tarpeettoman useasti väylälle. Prosessidataobjektin vastaanottavassa päässä voidaan myös määrittää viestille aikakatkaisu, jolla voidaan esimerkiksi lopettaa toiminta, jos viesti ei saavu tietyin väliajoin. (CiA-301 2011) Huomioitavaa on myös, että CANopen-standardi mahdollistaa myös prosessidataobjekteihin ladottujen parametrien yhdistämisen objektikirjastoon, jolloin näitäkin parametrejä on mahdollista kysyä käyttäen SDO-protokollaa.

Koska CANopen on joustava, avoin ja sitä tukevia komponentteja on paljon markkinoilla, on se saavuttanut suosiota erityisesti liikkuvien työkoneiden valmistajien keskuudessa. Tästä huolimatta CANopen ei ole syrjäyttänyt J1939-standardia dieselmoottorin ja muiden voimalinjan komponenttien ohjauksessa, mihin lienee syyksi J1939-standardin tarjoamat tarkemmat rajapinnat näiden komponenttien ohjaamiseen. Usein laitevalmistajat haluaisivatkin hyödyntää molempien standardien hyviä puolia. Koska CANopen ja J1939 ovat yhteensopivia fyysisesti, ovat SAE ja CiA yhteistyössä tehneet yleisimmille työkoneissa käytettäville CANopen-komponenteille, kuten enkoodereille, laiteprofiilin, joka mahdollistaa CANopen-laitteen kytkemisen osaksi J1939-standardin mukaista väylää. Tarkemmin nämä laiteprofiilit määrittävät yhteyden J1939-standardin parametriryhmille ja CANopen-standardin prosessidataobjekteille. Huomioitavaa on, että CANopen-laitteen prosessidataobjektit käyttävät normaalia viestikehystä ja konfigurointi tapahtuu silti CANopen-standardin huoltodataobjekteilla. Koska J1939-standardi tukee normaaliaviestikehystä vain tietyin poikkeuksin, on ainoastaan tahdistamattoman prosessidatan lähetys mahdollista.

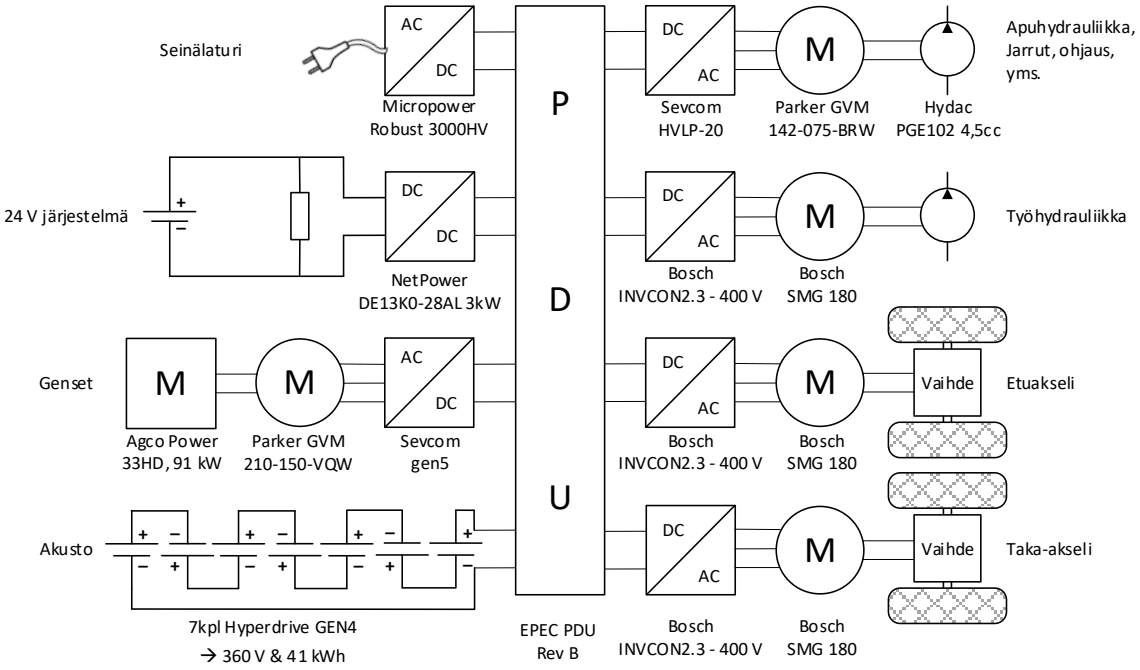
5. DEMONSTRAATIOKONE JA SEN OHJAUSJÄRJESTELMÄN TOTEUTUS

Demonstraatiotyökoneen valmistaminen tutkimuskäyttöön alusta alkaen ei ole kustannustehokasta, minkä vuoksi hybridityökone valmistetaan muokkaamalla sarjatuotettavan työkoneen voimansiirtoa, työhydrauliikkaa ja näiden ohjausjärjestelmää. Tällöin huomattava määrä komponentteja ja moduuleita voidaan hyödyntää alkuperäisestä työkoneesta eikä resursseja kulu niin paljoa tutkimuksen kannalta merkityksettömien komponenttien suunnitteluun ja valmistamiseen. Demonstraatiotyökoneelle lähtökohdaksi valikoitui dieselmoottorilla toimiva ja hydrauliikkaa ajovoimansiirrossa ja työhydrauliikassa käyttävä Wille 665 -pyöräkuormaaja, sillä kyseiseen työkoneeseen pohjautuvia demonstraatioalustoja on käytetty aikaisemmissa tutkimuksissa, kuten esimerkiksi Huovan et al. (2010 & 2018), Ahopellon (2019) ja Backaksen (2018) tutkimuksissa. Näin ollen konetyyppistä löytyy valmista mittausdataa, jota voidaan hyödyntää hybridisoinnin hyödyllisyyden pohtimiseen. Työkoneen kokoluokan hahmottamisen vuoksi alkuperäisen Wille 665:n massa on 6000 kg ja sen perusmitat on esitetty kuvassa 16 pois lukien leveys, joka on reilu 1,9 metriä (Wille Machines 2018). Tarkemmin työkoneen alkuperäinen ajovoimansiirto ja työhydrauliikka on esitetty esimerkiksi Huovan et al. (2018) julkaisussa.



Kuva 16. *Wille 665 pyöräkuormaaja (Wille Machines 2018, muokattu)*

Demonstraatiotyökoneen voimalinjaksi valikoitui puolestaan kuvassa 17 esitetty sarjatopologia, jonka valintaa projektiin tehty alustava simulaatiotutkimus puolsi, vaikka erot sarja- ja rinnakkaishybridin välillä eivät olleet suuria (Tupitsina et al. 2023). Merkittävin valintaan vaikuttava tekijä oli kuitenkin ajatus siitä, että tulevaisuudessa työkone on kyseisellä topologiaratkaisulla helpommin muutettavissa täysin sähkökäyttöiseksi poistamalla dieselmoottori generaattoreineen ja lisäämällä vapautuneeseen tilaan järeämpi sähköenergian varastointiratkaisu. Huomioitavaa on myös, että sähköisten työkoneiden tutkintaa voidaan jo aloittaa sarjahybridillä, kun dieselmoottoria ei käytetä.

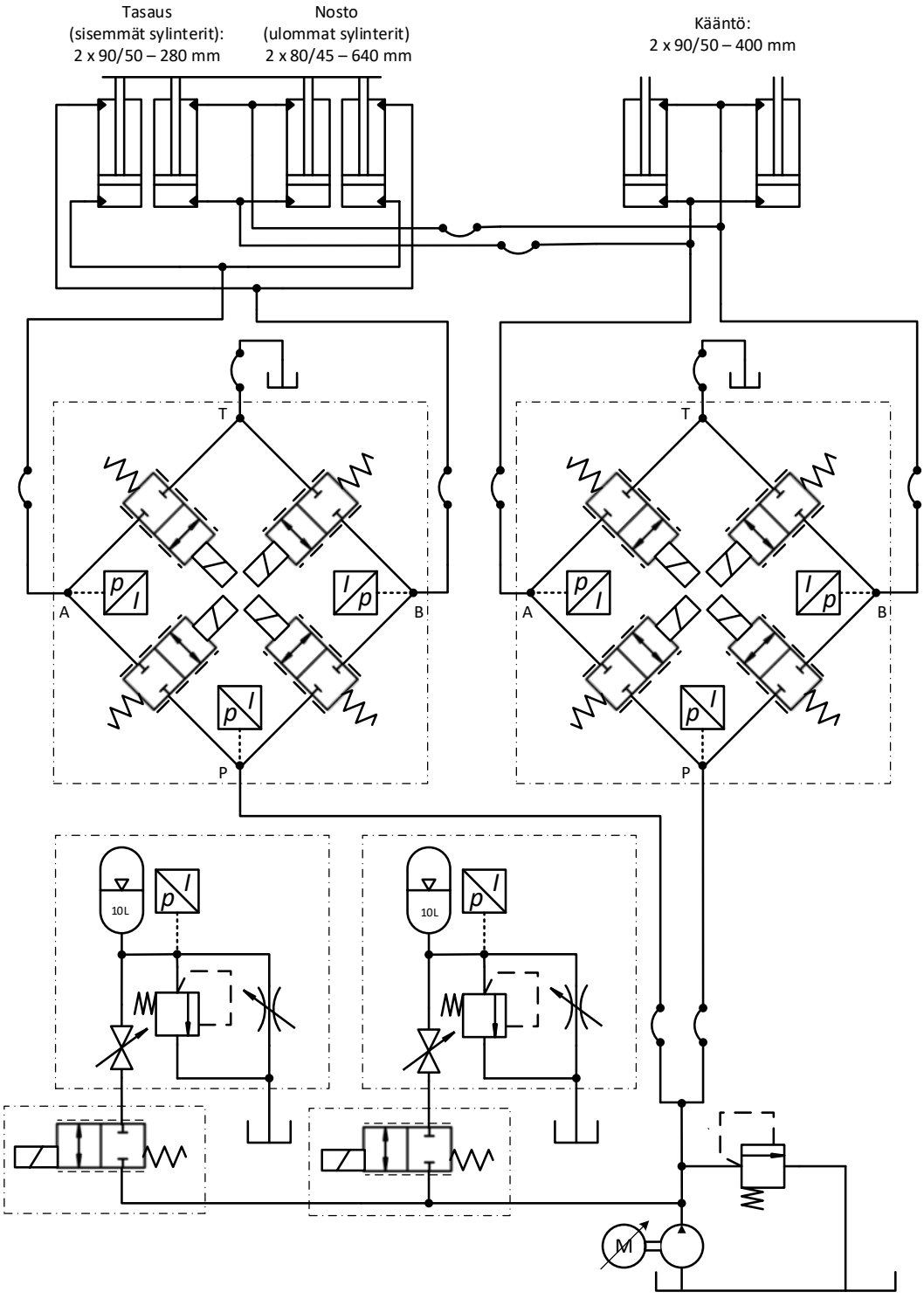


Kuva 17. *Demonstraatiokoneen hybriditopologia*

Kuvassa 17 esitetyt komponentit eivät ole mitoituksen kannalta täysin optimaaliset, sillä on esitetty, että komponenttien tilaushetkellä markkinatilanne työkoneisiin soveltuvien komponenttien osalta oli nykyistä heikompi ja tuolloin valitseva pandemia asetti omat haasteensa komponenttien toimitusajoille. Komponenttivalintoihin osakseen vaikuttivat myös yhteistyökumppanien asettamat rajoitukset kuin myös rajallinen tila Wille 665:n runkorakenteessa. Suurimpien haasteiden voidaan nähdä olevan generaattorin 25 kW nimellisteho suhteessa dieselmoottorin 91 kW nimellistehoon sekä vaihelaatikon puuttuminen työhydrauliikan pumpun ja sähkömoottorin välistä. Haasteeksi voi myös muodostua akuston jäähdytyskierron puuttuminen, mikä todennäköisesti asettaa rajoituksia pitkien yhtäjaksoisten mittauksien tekemiselle. Valitulla topologialla pystytään kuitenkin loistavasti edesauttamaan tutkimusta, jonka lyhyen aikavälin tavoitteena on verifioida projektissa tekeillä oleva simulaatiomalli ja mahdollistaa erilaisten hybridivoimalinjan ohjausstrategioiden tutkiminen.

Topologiassa päädyttiin akseleittain asennettaviin ajomoottoreihin, koska on esitetty, että yhden suuren moottorin tai napamoottorien löytäminen oli haasteellista. Akseleittain asennettavat moottorit mahdollistivat myös alkuperäisten akseleiden ja niihin asennettujen jarrusylintereiden hyödyntämisen, mikä jälleen yksinkertaisti demonstraatiokoneen suunnittelua ja implementointia. Tilankäytön kannalta todennäköisesti napakohtaisilla moottoreilla olisi päästy parempaan lopputulokseen, jolloin tilaa olisi vapautunut esimerkiksi työhydrauliikan pumpun ja sen sähkömoottorin väliselle vaihdelaatikolle.

Kuten aikaisemmin on mainittu, voimansiirtolinjan lisäksi työkoneen työhydrauliikkaan tehdään muutoksia. Alkuperäisestä järjestelmästä puomiston hydraulisylinterit hyödynnetään, mutta niitä ohjaavat kaksi 4/3-proportionaaliventtiiliä korvataan vastaavilla digitaalisilla virtauksensäätöyksiköillä. Tiivistettynä digitaalisten virtauksensäätöyksiköiden idea on käyttää useampaa erikseen päälle tai pois ohjattua istukkaventtiiliä muodostamaan erillisreunaohjaus, joka mahdollistaa virtaaman ohjaamisen vapaasti reunalta toiselle ilman luistirakenteen tuomia rajoituksia (Linjama 2011). Työhydrauliikan piiriin lisätään myös kaksi 10 litran paineakkua ja niiden ohjaukseen digitaaliset 2/2-virtauksensäätöyksiköt. Tilavuusvirta tuotetaan puolestaan pyörimisnopeusohjatulla kiinteän kierrostilavuuden pumpulla. Työhydrauliikan osalta hydraulijärjestelmä on esitetty kuvassa 18, jossa digitaaliset virtauksensäätöyksiköt on esitetty virtausreunakohtaisesti tiivistetyillä piirrosmerkeillä. Jokaisessa digitaalisessa virtauksensäätöyksikössä on viisi samankokoista solenoidiventtiiliä virtausreunaa kohden, joiden läpivirtaus on säädetty venttiilien eteen asennettavilla samankokoisilla kuristimilla.



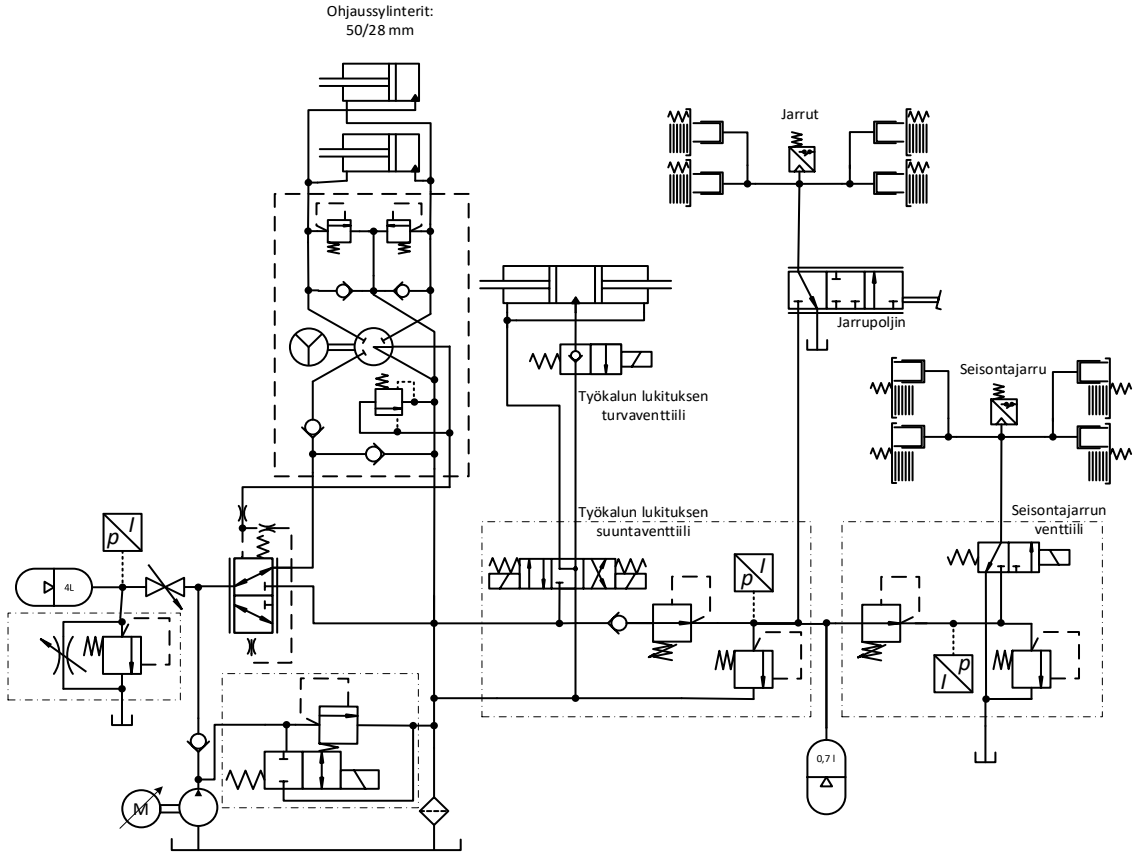
Kuva 18. *Demonstraatiokoneen työhydrauliikka*

Vasemmalla kuvan 18 digitaalisella virtauksensäätöyksiköllä ohjataan puomin nostoliikettä ja oikeanpuoleisella työkalun, kuten kauhan, kääntöliikettä. Kääntöliikkeen yhteyteen on myös toteutettu mekaanisesti kahdella hydraulisylinterillä stabilointitoiminnallisuus, jonka idea on pitää työkalu samassa asennossa suhteessa maahan riippumatta puomin asennosta. Kyseinen toiminallisuus voitaisiin toteuttaa myös ohjaamalla aktiivisesti kääntösylintereitä noston liikkeen aikana. Aikaisemmassa tutkimuksessa

Huova et al. (2010) ovat kuitenkin todenneet samalla puomistolla mekaanisen stabiloinnin energiatehokkaammaksi vaihtoehdoksi, minkä vuoksi sitä käytetään myös tässä demonstraatiokoneessa. Puomiston takaisinkytketyn asemasäädön toteuttamiseksi nosto- ja kääntösylintereiden asemaa mitataan potentiometreillä.

Poiketen aikaisemmista tutkimuksista yhdistetään demonstraatiokoneen työhydrauliikasta nähtävästi ensimmäistä kertaa työkonesovelluksessa digitaalisilla virtauksensäätöyksiköillä toteutettu erillisreunaohjaus sähkömoottorilla pyöritettyyn vakiokierrostilavuuspumppuun. Tällaisen järjestelmän suurin hyöty on mahdollisuus pysäyttää pumppu, kun sen tuottoa ei tarvita eikä pumppu aiheuta tällöin edes viskoosihäviöitä, kuten kierrostilavuussäädettävä pumppu. Pyörimisnopeusohjatun pumpun dynamiikka on kuitenkin kierrostilavuussäädettävää pumppua hitaampi, minkä vaikutus puomiston käyttökokemukseen on yksi demonstraatiokoneella tehtävän tutkimuksen tavoitteista. Järjestelmästä löytyy myös paineakut, joiden avulla voidaan tutkia energian talteenottoa.

Demonstraatiokoneen apuhydrauliikan piiri on puolestaan esitetty kuvassa 19, ja se koostuu hydraulisesta ohjauksentehostuksesta, käyttö- ja seisontajarruista sekä puomiston työkalun lukituksen turva- ja suuntaventtiileistä. Piiristä löytyy myös kaksi paineakkua, jotka mahdollistavat jarrujen ja ohjauksen turvallisen käytön mahdollisessa vikatilanteessa ja pumpun sammuttamisen, kun järjestelmän paine on riittävä. Edellä mainitut toiminallisuudet toteutettiin hyödyntämällä komponentteja alkuperäisestä työkoneesta säästäen demonstraatiokoneen suunnitteluun ja valmistamiseen tarvittavia resursseja.



Kuva 19. *Demonstraatiokoneen apuhydrauliikka*

Koska työkoneen alkuperäinen sähköjärjestelmän nimellisjännite oli 12 V, vaihdettiin se 24 V. Korkeammasta jännitetasosta on siirtohäviöiden minimoimisen lisäksi hyötyä varsinkin digitaalisten virtauksensäätöyksiköiden venttiilien dynamiikan parantamisessa. Koska alkuperäisestä työkoneessa ei säilytetä lähes mitään ohjausjärjestelmään liittyen, valittavaksi tässä työssä jää mittaristona toimiva näyttö ja ohjelmoitavat logiikat, joilla voidaan ohjata hydrauliikan venttiilejä sekä alijärjestelmiä, kuten sähkömoottoreiden inverttereitä ja dieselmoottoria. Näiden komponenttien ja fyysisen arkkitehtuurin valitsemisen jälkeen ohjelmoitavien logiikoiden ohjelmistomoduulit rajapintoineen suunnitellaan ja toimilaitteiden sekä ohjainlaitteiden välinen kommunikointirajapinta muodostetaan. Lopuksi dokumentoidaan myös, kuinka ohjausalgoritmien siirtäminen ohjelmoitaville logiikoille tapahtuu sekä esitetään ohjausjärjestelmän jatkokehityskohteet.

5.1 Vaatimukset ja tavoitteet

Alustavat ohjausjärjestelmää koskevat vaatimukset toimeksiannon yhteydessä olivat lähes olemattomat, mikä voidaan nähdä tyypilliseksi haasteeksi organisaation sisäisessä tutkimusalustojen kehitysprojektissa. Käytännössä vaatimuksina oli, että ohjausjärjestelmään hankittavien ohjelmoitavien logiikkojen tulee olla kaupallisia – suoraan hyllystä

saatavia – ja niiden tulee soveltua kohteen työkoneeseen. Lisäksi ohjainlaitteiden vaadittiin tukevan Matlab & Simulink -ympäristöllä tehtävää mallipohjaista ohjausalgoritmin kehitystä eikä logiikoiden ohjelmointi saa vaatia maksullista lisenssiä. Työkoneen tarkoituksen eli erilaisten ohjausalgoritmien kehittämisen takia tiettyä spesifikaatiota laskentateholle ei pystytty määrittämään. Tavoitteeksi asetettiinkin mahdollisimman modulaarinen rakenne, jotta ohjausjärjestelmän laskentatehoa pystytään nostamaan myöhemmässä vaiheessa mahdollisimman helposti esimerkiksi vaihtamalla ohjainlaitteita laskennallisesti tehokkaampiin tai lisäämällä niiden määrää.

Kuten demonstraatiokoneen esittelystä voidaan huomata, määrittää aikaisemmin tehty suunnittelu jo merkittävästi ohjausjärjestelmään kohdistuvia vaatimuksia. Käytännössä kaikki alijärjestelmät ja toimilaitteet kuin myös anturit on valittu. Näiden komponenttien datalehtien pohjalta voidaan muodostaa vaatimukset logiikkojen sisäänmenoille ja ulostuloille sekä spesifioida ne tiedonsiirtoratkaisut, joiden tulee olla vähintään tuettuna. Tästä poikkeuksena ovat kuitenkin järjestelmään tulevat digitaaliset virtauksensäätöyksiköt, jotka ovat omavalmisteisia eikä niille ei ole datalehteä.

Seuraavissa aliluvuissa tarkennetaan ohjausjärjestelmään kohdistuvia vaatimuksia ja tavoitteita. Aluksi luvussa 5.1.1 selvitetään hybridivoimalinjan ohjauksen keskiössä olevan tehonhallinta-algoritmin toiminnan peruste ohjelmistomoduulien määrittämisen tueksi. Luvussa 5.1.2 perehdytään puolestaan digitaalisten virtauksensäätöyksiköiden ohjauselektroniikkaan ja säätöön kohdistuviin vaatimuksiin, koska venttiileitä on tarkoitus ohjata tässä työssä valittavilla ohjainlaitteilla eikä erillistä ohjauselektroniikkaa ole käytettävissä. Tämän jälkeen luvussa 5.1.3 kootaan yhteen kaikki toimilaitteet, anturit ja syöttölaitteet, jotka on tarkoitus kytkeä ohjelmoitavien logiikoiden sisäänmenoihin ja ulostuloihin ja tutkitaan millaisia vaatimuksia nämä asettavat logiikalle. Lopuksi luvussa 5.1.4 tutkitaan, mitä tiedonsiirtoratkaisuja ohjelmoitavien logiikoiden tulee tukea ja mitä vaatimuksia tiedonsiirtoratkaisun avulla ohjattavat komponentit asettavat ohjausjärjestelmälle.

5.1.1 Hybridivoimalinjan tehonhallinta

Koska hybridivoimalinjassa on useampi tehonlähde ja toimilaite, on hybridivoimalinjan säädön keskiössä tehonhallinta eli miten käyttöteho tuotetaan ja miten se jaetaan tehoa kuluttaville toimilaitteille. Lisäksi demonstraatiotyökoneen kaltaisen sähkö–dieselsarjahybridin säädössä tulee huomioida, että jotkin toimilaitteet voivat toimia hetkellisesti myös tehonlähteenä ja että korkeajänniteakuston varaustasoa (engl. state of charge, SOC) pitää ylläpitää dieselgeneraattorin avulla.

Yksinkertaisimmillaan säädin voi perustua sääntöihin, joilla ohjataan toimintaa perustuen toimitilan reaaliaikavalvontaan (engl. supervisory control system, SCS). Aihealueen julkaisuissa termostaattiohjausstrategia (engl. thermostat control strategy (TCS) ja tehon seurauksen ohjausstrategia (engl. power follower control strategy, PFCS) ovat osoittautuneet hyviksi lähtökohdiksi ja vertailukohteeksi uudempien ohjausstrategioiden kehittämiselle. Termostaattiohjausstrategian keskiössä on pyrkimys käyttää polttomoottoria sen optimaalisimmassa toimintapisteessä, ja sen toteuttaminen on suhteellisen yksinkertaista. Ideana on tuottaa dieselmoottorilla vakiotehoa ja ottaa loput akustosta. Mikäli dieselmoottori tuottaa liikaa tehoa kulutukseen nähden, ladataan akustoa tai sammutetaan moottori, jos akuston varaus on tarpeeksi korkea. Jos puolestaan kulutus on liian suuri tuottoon nähden, rajoitettaan kulutusta kohteeseen soveltuvalla tavalla, kuten priorisoimalla. (Shabbir & Evangelou 2016)

Tehon seurauksen ohjausstrategiassa perusideana on tuottaa dieselmoottorilla tarvittu teho pitkällä aikavälillä eli toisin sanoen seurata tehon kulutusta. Kun akusto on tarpeeksi täynnä ja kulutus tarpeeksi pieni, voidaan teho ottaa täysin akustosta. Dieselmoottorin rooli tehon tuotosta kasvaa, kun akuston varaus alkaa tippumaan tai luonnollisesti, kun kuormitus ylittää akuston maksimiulosantotehon. Kuitenkin, kun akuston varaustaso tippuu määritettyyn kynnysarvoon SOC_M, pyritään dieselmoottorilla kattamaan koko kulutus. Jos varaustaso tippuu entisestää alarajaan SOC_L, pyrkii diesel puolestaan nostamaan akuston varaustason ylärajaan SOC_U, jonka jälkeen akusto siirtyy jälleen pääsääntöiseksi tehonlähteeksi. Huomioitavaa on, että kyseisiä ylä- ja alarajoja ei ole useinkaan rajattu akuston käytön laajimmalle mahdolliselle käyttöalueelle. (Shabbir & Evangelou 2016)

Käyttörajojen määrittämiseen vaikuttaa useampi tekijä, joita ovat esimerkiksi akuston käyttöiän optimointi, sovelluskohde ja akuston kapasiteetti ja molempien tehonlähteiden tuotto verrattuna tarvittuun maksimitehoon. Jos sovelluskohde on sellainen, että sähkönjakeluverkosta otettua energiaa voidaan käyttää suurimman osan ajasta, lienee järkevää asettaa algoritmin rajat suhteellisen alas. Vaihtoehtoisesti, jos tarvittu maksimiteho on pieni suhteessa tehonlähteiden maksimitehoon, mutta myös akuston kapasiteetti on suuri, kannattaa todennäköisesti raja-arvot asettaa laajalle, jolloin latauksen jälkeen polttomoottori voidaan sammuttaa.

Sääntöpohjaisten ohjausstrategioiden lisäksi laskentatehon hinnan tultaessa alas on mallipohjaisia säätimiä kehitetty. Tunnetuin on ehkä globaalin ekvivalentin polttoaineen kulutuksen minimoinnin strategia (engl. global equivalent consumption minimization strategy, GECMS). Menetelmässä jokaisella laskentakierroksella menetelmän parametri-

soitu malli ekvivalentille polttoaineenkulutukselle pyritään minimoimaan Pontryagin minimointiperiaatteella (Shabbir & Evangelou 2016). Laskenta on raskas, mutta nähtävästi mahdollinen tehdä reaaliajassa. Sen on esitetty antavan lähes identtisen polttoaineenkulutuksen verrattuna niin sanottuun pienimpään mahdolliseen globaaliin polttoaineen kulutukseen, joka voidaan laskea ei-reaaliaikaisesti simulaatioympäristössä käyttäen työkoneesta mitattua dataa ja dynaamista ohjelmointia (Serrao et al. 2011). Simulaatioympäristössä lasketun pienimmän mahdollisen polttoaineen kulutuksen määrittäminen vaatii tarkkaa validoitua mallia koko työkoneesta, josta on myös hyötyä globaalin ekvivalentin polttoaineen kulutuksen minimoinnin strategiamallin parametrien määrittämisessä. Näiden parametrien määrittämiseksi malli ei ole pakollinen ja ne voidaan määrittää yritys–erehdysmenetelmän avulla, mutta tarkka polttoaineenkulutuksen kartta kuitenkin vaaditaan. (Shabbir & Evangelou 2016) Muita kirjallisuudessa esitettyjä ohjausstrategioita on käsitellyt esimerkiksi Ehsani et al. (2021) kirjallisuuskatsauksessaan.

Huomioitavaa yleisesti ohjausstrategioista on, että polttoaineen kulutuksen minimointi ei tulisi olla ainoa ohjausstrategian tavoite, vaan ohjauksen suunnittelussa tulisi pyrkiä maksimoimaan komponenttien, kuten akuston ja polttomoottorin käyttöikää, mutta myös minimoimaan päästöjä. Varsinkin hiukkaspäästöjen määrää kasvattaa dieselmoottorin transienttikuormitus, mutta myös moottorin käyttö sen optimaalisen käyttölämpötilan alapuolella. Toisin sanoen polttomoottoria tulisi käyttää mahdollisimman tasaisella kuormituksella ja moottorin jatkuvaa sammuttelua tulisi välttää. Polttomoottorin sammuttamisessa tulee myös huomioida, ettei sitä voida sammuttaa liian kuumana – suoraan optimaalisimmasta toimipisteestä – sillä se voi aiheuttaa moottorin osien ennenaikaista hajoamista ja öljyn ennenaikaista vanhentumista.

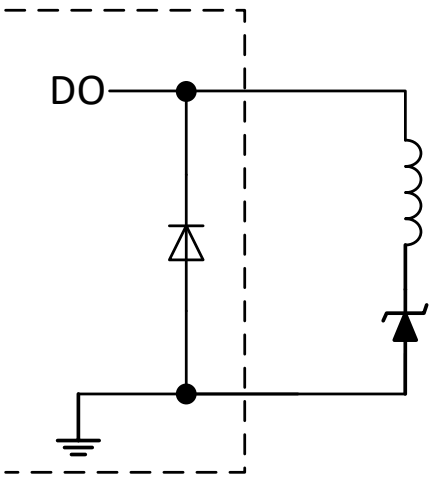
5.1.2 Digitaalisten virtauksensäätöyksiköiden ohjaus

Kuten aikaisemmin on mainittu, digitaalisten virtauksensäätöyksiköiden idea on käyttää useampaa erikseen päälle tai pois ohjattua istukkaventtiiliä muodostamaan erillisreuna ohjaus. Tarkemmin liikkeensäädössä digitaalisen virtauksensäätöyksikön läpivirtaus haluttuun virtausreunaan pyritään pitämään ulkoisessa referenssissä kompensoiden virtausreunojen painevaihtelu. Painekompensointia varten digitaalisessa virtauksensäätöyksiköstä löytyykin venttiilien lisäksi virtausreunojen paineanturit. Jotta ratkaisussa muun muassa energian kierrättämistä voidaan hyödyntää aina kun mahdollista, on nopea venttiilidynamiikka keskiössä. Nopea venttiilidynamiikka parantaa mahdollisuutta myös painekompensointiin eli työnpaineen häiriöiden minimointiin, joka on usein haluttu ominaisuus. Istukkaventtiili on jo itsessään esimerkiksi tyypillistä luistirakenteista 4/3-proportionaaliventtiiliä nopeampi, mutta sen dynamiikkaa voidaan parantaa entisestään ohjauselektroniikalla.

Tiivistettynä venttiilien avautumista pyritään nopeuttamaan digitaalisessa virtauksensäätöyksikössä kasvattamalla solenoidin läpi menevää virtaa venttiilin avautumisen aikana. Tutkimusalustassa tämä on tarkoitus toteuttaa PWM-ohjauksella (engl. pulse width modulation, PWM) ja käyttämällä venttiileissä 12 V nimellisjännitteen solenoideja 24 V järjestelmässä. Ratkaisussa venttiilin avautumisen aikana solenoidin läpi kulkevaa virtaa kasvatetaan asettamalla PWM-suhde yhteen. Kun venttiili on avautunut, PWM-suhde tiputetaan sopivaan arvoon, jotta solenoidin nimellisteho ei ylity. Huomioitavaa on, että nimellistehon ylittyminen pitkällä aikavälillä voi nostaa solenoidin lämpötilaa liikaa, mikä voi aiheuttaa oikosulun lakan sulaessa solenoidin päältä.

Nimellistehoa vastaava PWM-suhde voidaan helposti laskea jakamalla solenoidin nimellisjännite käyttöjännitteellä. Virran arvo halutaan kuitenkin usein minimoida, jotta järjestelmän energiatehokkuus paranee ja solenoidin magneettikenttään ei sitoudu ylimääräistä energiaa, joka hidastaisi puolestaan venttiilin sulkeutumista. Solenoidin läpikulkeva virta on kuitenkin solenoidin resistanssista riippuva, joka puolestaan riippuu lämpötilasta. Koska virta ei saa laskea liikaa, mikä aiheuttaisi venttiilin kiinni menemisen, säädetään PWM-suhdetta usein läpikulkevan virran mittauksen takaisinkytkennän avulla. Virranmittauksella olevat PWM-lähdöt asetettiin myös demonstraatiokoneen ohjelmoitavien logiikoiden vaatimukseksi.

Jotta puolestaan sulkeutumista voidaan nopeuttaa, tulee solenoidin magneettikenttään sitoutuneesta energiasta päästä mahdollisimman nopeasti eroon. Ohjelmoitavissa logiikoissa tyypillinen ratkaisu on rinnan kytkeä vapaakiertodiodi ulostuloon, jolloin magneettikenttään sitoutunut energia muuttuu hitaasti lämmöksi diodin kynnysjännitteestä. Digitaalisen virtauksensäätöyksikön venttiilien ohjauksessa tämä ei useinkaan riitä ja sulkeutumista halutaan nopeuttaa entisestään. Erilaisia keinoja on olemassa useita, jotka perustuvat niin passiivisiin komponentteihin kuin myös aktiivisesti ohjattuja komponentteja sisältäviin piirilevyratkaisuihin. (Tiainen 2014, s. 22–23) Näistä aktiiviset ratkaisut vaatisivat omien piirilevyjen tekemistä, minkä vuoksi demonstraatiokoneessa päädyttiin yksinkertaiseen ratkaisuun, jossa zenerdiodi kytketään sarjaan venttiilin solenoidin kanssa kuvan 20 esittämällä tavalla.



Kuva 20. *Digitaalisen virtauksensäätöyksikön venttiilin kytkeminen logiikkaan*

Zenerdiodin kytkeminen kuvan 20 esittämällä tavalla tiputtaa venttiilin avaamiseen käytettävissä olevaa efektiivistä jännitettä ja tätä kautta hidastaa venttiilinavautumista verrattaessa järjestelmään, jossa zenerdiodia ei ole. Zenerdiodi ei kuitenkaan aiheuta samanlaista tehohäviötä kuin sen korvaaminen esimerkiksi vastuksella. Järkeväksi zenerdiodin kynnysjännitteeksi on esitetty 5,6 V, josta suuremmalla kynnysjännitteellä oleva ei merkittävästi enää nopeuta venttiilin sulkeutumista (Tiainen 2013 s. 57). Samat zenerdiodit valittiin myös demonstraatiokoneeseen, jolloin avaamisen yhteydessä efektiiviseksi jännitteeksi 100 % PWM-signaalilla jää 23,2 V, jos oletetaan 24 V nimellisjännitteen järjestelmän tuottavan työkoneille tyypillinen 28,8 V käyttöjännite.

Lähdön tyypin lisäksi merkittävä tekijä ohjelmoitavia logiikkoja valittaessa digitaalisille virtauksensäätöyksiköille on niiden lähtöjen virrankesto ja logiikan tehonlähteen riittävyys. Yhden venttiilin solenoidin läpi menevä virta voidaan laskea ajan *t* funktiona kaavalla

$$i(t) = \frac{U}{R}\left(1 - e^{-\frac{t}{\tau}}\right), \tag{2}$$

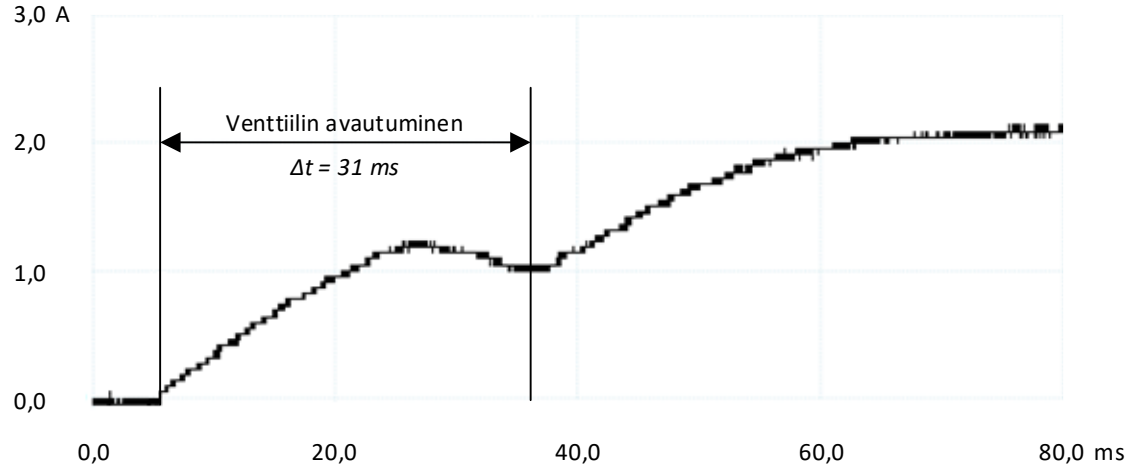
jossa *U* on jännite, *R* resistanssi ja *τ* aikavakio. Aikavakio voidaan puolestaan laskea solenoidin induktanssin *L* ja resistanssin suhteena kaavalla

$$\tau = \frac{L}{R} \tag{3}$$

(Tiainen 2013, s.18–19).

Koska demonstraatiokoneen digitaalisiin virtauksensäätöyksiköihin tulevien venttiilien solenoidien induktanssi on vaikea määrittää, koska se muuttuu venttiilin nopeuden, aseman ja solenoidin läpikulkevan virran funktiona, päätettiin virran käyttäytyminen mitata

suoraan Fluke i30 -virtapihtien ja Picoscope 2205A oskilloskoopin avulla. Mittausjärjestelyissä venttiili ja sen kanssa sarjassa oleva 5,6 V zenerdiodi kytkettiin suoraan valmiiksi päällä olevaan jännitelähteeseen, jonka ulostulo oli asetettu 28,8 V. Mittausjärjestelmä käytännössä vastaa tilannetta, jossa PWM-suhdesäädettävä ulostulo säädetään suoraan nollasta prosentista sataan prosenttiin. Huomioitavaa kuitenkin on, ettei venttiili ollut hydraulijärjestelmässä mittauksen aikana, minkä vuoksi avautuminen voi todellisessa järjestelmässä poiketa hieman mitatusta. Tulos on esitetty kuvassa 21.



Kuva 21. *Digitaaliventtiilistön istukkaventtiilin avausvirta ajan funktiona*

Kuvasta 21 voidaan huomata venttiilin avautuminen virran arvon hetkellisestä notkahtamisesta, mikä johtuu venttiilin kelan back-EMF -ilmiöstä (Tiainen 2013, s. 24) Siitä huolimatta, että käytössä olleen oskilloskoopin mittausalue ja resoluutio olisivat voineet olla tarkempia, nähdään virran nousevan avauksen aikana noin 1,3 A, jonka jälkeen virta tasautuu reiluun 2 A. Kuten aikaisemmin on mainittu, venttiilien solenoidien ollessa suunniteltu 12 V nimellisjännitteelle, ne eivät todennäköisesti kestä jatkuvasti 23,2 V teholliselle jännitteellä saavutettua 2 A virtaa, jonka vuoksi virtaa tulee rajoittaa PWM-suhteella venttiilin avautumisen jälkeen.

Koska venttiilin avautumisen jälkeen neula asettuu solenoidin suhteen siten, että niiden välinen ilmarako pienenee, tuottaa solenoidi suuremman voiman samalla virralla (Tiainen 2013, s. 21). Tämän vuoksi virtaa voidaan laskea avautumisen jälkeen entisestään ja voidaan olettaa minimipitovirran jäävän alle 1 A. Minimipitovirran määrittäminen tarkemmin tulee kuitenkin tehdä valmiista järjestelmästä, sillä järjestelmän painetaso vaikuttaa venttiilin auki pitämiseen vaadittuun voimaan. Mittauksien pohjalta alustava venttiilin avaussekvenssi voisi logiikan näkökulmasta olla:

1. Säädetään PWM-suhde 0 prosentista 100 prosenttiin
2. Odotetaan 50 ms tai, että virta nousee 1,5 A
3. Tiputetaan virta 1 A pitovirtaan.

Avaussekvenssiä varten lähdöiltä, joihin digitaalisen virtauksensäätöyksikön venttiilit kytketään, vaaditaan virrankestoksi vähintään 1,5 A.

Ohjauksen kannalta venttiilin dynamiikan lisäksi koko digitaalisen virtauksensäätöyksikön dynamiikkaan vaikuttaa, kuinka usein yksittäinen venttiilin avaus- tai sulkusekvenssi voidaan aloittaa. Vaatimukseksi tälle asetettiin 2 ms tutkimusryhmän aikaisemman kokemuksen pohjalta. Koska digitaalisen virtauksensäätöyksikön avauman muuttaminen perustuu nopealla aikavälillä aina virtausreunojen paineeseen eikä ulkoiseen, kuten ohjaussauvalta tulevaan referenssiin, tulee painetieto kyetä vastaanottamaan ja analysoimaan samassa aikamääreessä. Digitaalista virtauksensäätöyksikköä ohjaavan logiikan kannalta tämä asettaa vaatimuksen 2 ms kiertoajalle, mikäli venttiilit ja paineanturit ovat suoraan kiinni logiikan ulostuloissa. Huomioitavaa on, että jos tietoa välitetään tiedonsiirtoratkaisun avulla, tulee kokonaisviive ottaa huomioon.

5.1.3 Ohjausjärjestelmän sisäänmenot ja ulostulot

Digitaalisten virtauksensäätöyksikköjen asettamien vaatimuksien selvittämisen jälkeen muiden järjestelmään tulevien komponenttien datalehdet käytiin läpi. Selvityksen pohjalta koostettiin taulukko 6, jossa on esitetty ohjausjärjestelmältä vaadittujen ulostulojen ja sisäänmenojen määrä ja tyyppi. Tarkemmin vaatimukset on esitetty taulukon jälkeen.

Taulukko 6. *Demonstraatiokoneen ohjelmoitavilta logiikoilta vaaditut sisäänmenot ja ulostulot*

Ulostulot	**Tyyppi (virrankesto)**	**Määrä**
Työhydrauliikka:		
Virtauksensäätöyksiköiden venttiilit	PWM-lähtö (1,5 A) virtamittauksella	50
Puomin asema-antureiden referenssi	Reguloitu 5 V -jännite	2
Ohjaus ja jarrutus:		
Vapaakiertoventtiili	Digitaalinen (1,5 A)	1
Työkalun lukituksen turvaventtiili	Digitaalinen (1,5 A)	1
Työkalun lukituksen suuntaventtiili	Digitaalinen (1,5 A)	2
Pysäköintijarrun venttiili	Digitaalinen (1,5 A)	1
Jäähdytyspiiri:		
HV-komponenttien jäähdytyspumpun syöttö	Digitaalinen (17,5 A)	1
HV-komponenttien jäähdytyspumpun nopeus	PWM (–)	1
Puhaltimien ohjaus	Digitaalinen (20 A)	3
Työkoneen käyttäjän rajapinta:		
Kaasupolkimen referenssi	Reguloitu 5 V- jännite	2
Sisääntulot	**Tyyppi (mittausalue)**	**Määrä**
Työhydrauliikka:		
Työliikkeiden painelinjojen paineanturit	Virtaviesti (4-20mA)	6
Paineakkujen paineanturit	Virtaviesti (4-20mA)	2
Puomin asema-anturit	Jänniteviesti (0–10 V)	2
Ohjaus ja jarrutus:		
Puomin työkalun lukituslohkon paineanturi	Virtaviesti (4-20mA)	1
Jarrulohkon paineanturi	Virtaviesti (4-20mA)	1
Apuhydrauliikan syöttöpaineen paineanturi	Virtaviesti (4-20mA)	1
Paluusuodattimen tukkeutumisindikaattori	Digitaalinen	1
Jäähdytyspiiri:		
Lämpötila-anturi	Virtaviesti (4-20mA)	1
Jäähdytysjärjestelmän pumpun nopeussignaali	Pulssilaskuri (12 ppr)	1
Käyttäjän ja työkoneen välinen rajapinta:		
Kaasupolkimen asema-anturit	Jänniteviesti (0–4,5 V)	2
Seisontajarrun painokytkin	Digitaalinen	1
Työkalun lukituksenavaus painokytkin	Digitaalinen	1

Digitaalisten virtauksensäätöyksiköiden venttiilien lisäksi järjestelmässä on myös muita istukkaventtiileitä, joissa on yhteensä 5 solenoidia, kuten kuvien 18 ja 19 hydraulikaavioista nähdään. Nämä solenoidit on suunniteltu 24 V nimellisjännitteelle ja tehonkestoltaan ne ovat 17–22 W, jolloin nimellisjännitteellä ne vaativat ulostulolta 0,75–0,92 A virran. Koska 24 V nimellisjännitettä käyttävässä työkoneessa käyttöjännite on noin 28,8 V, kasvaa venttiilien solenoidien läpi kulkeva virta hieman nimellisjännitteestä lasketusta. Tästä syystä näidenkin venttiilien ulostuloille asetettiin vaatimukseksi varmuuden vuoksi 1,5 A virran kesto. Koska näitä venttiileitä ei käytetä liikkeen ohjaamiseen eikä niillä ole merkittävää roolia tutkimuksen kannalta, riittää niille digitaalinen eli päälle ja pois -ohjattu ulostulo. Energiatehokkuuden kannalta PWM-lähdöstä voisi olla hyötyä, jos venttiili pysyy auki pienemmällä virralla. PWM-lähtöä ei kuitenkaan vaadita näille venttiileille.

Venttiilien lisäksi ulostuloja tarvitaan kolmen jäähdytinjärjestelmän puhaltimen ja korkeajännitekomponenttien jäähdytyspumpun ohjaamiseen. Näistä puhaltimet ovat yksinopeuksisia ja tarvitsevat ainoastaan tehonsyötön katkaisevan ohjauksen. Pumppupuolestaan on muuttuvanopeuksinen ja tarvitsee tehonsyötön katkaisevan ohjauksen lisäksi nopeusreferenssin PWM-signaalista. Näistä molemmat tarvitaan, koska pumppu pyörii 2500 rpm miniminopeudella ilman PWM-signaalia eikä sitä voida tämän vuoksi kytkeä suoraan käyttöjännitteeseen. Koska puhaltimet tarvitsevat toimiakseen 20 A virran ja pumppu 17,5 A, ei niitä todennäköisesti voida ohjata suoraan logiikan avulla, vaan logiikalla joudutaan ohjaamaan erillisiä releitä, jotka kytkevät ja irrottavat kyseiset komponentit käyttöjännitteestä. Tämä johtuu siitä, että tyypillisen työkoneisiin soveltuvan logiikan ulostulo kestää vain muutaman ampeerin virtoja. Releiden käytölle ei sovelluskohteessa ole esteitä, mutta komponenttien määrän minimoimiseksi suorakytkentä olisi tietenkin tavoiteltavaa.

Ulostuloja vaativien toimilaitteiden lisäksi järjestelmässä on useita antureita ja muutama syöttölaite käyttäjää varten, mitkä vaativat logiikoilta sisäänmenoja. Anturit, joiden mittauslukema välitetään anturin yliolevan jännite-eron avulla eli jänniteviestinä, tarvitsevat tyypillisesti jännitettä mittaavan sisäänmenon lisäksi reguloidun – usein käyttöjännitettä matalamman – ulostulon tehonsyöttöä varten. Demonstraatiotyökoneen tapauksessa kaasupoljin ja puomin asemaa mittaavat potentiometrit tarvitsevat yhteensä neljä 5 V reguloitua ulostuloa ja neljä välillä 0–5 V mittavaa sisäänmenoa. Tarvittaessa 5 V reguloitu tehonsyöttö voidaan toteuttaa näille antureille yhdellä ulostulolla, sillä yksittäinen anturi ei vaadi ulostulolta kuin muutaman mikroampeerin tehon keston.

Muut järjestelmän anturit ja syöttölaitteet käyttäjää varten eivät vaadi reguloitua syöttöjännitettä ja niiden tehonsyöttö voidaan toteuttaa suoraan käyttöjännitteestä. Näistä antureista kahdentoista paineanturin ja yhden lämpötila-anturin mittauslukema perustuu 4–

20 mA virtaviestiin ja aikaisemmin mainitun korkeajännitekomponenttien jäähdytysjärjestelmän pumpun pyörimisnopeuden mittauslukema pulssiviestiin. Pulssiviestissä on 12 pulssia kierrosta kohden, jolloin pumpun pyörimisnopeus alueella pulssitaajuus on 500–1050 Hz. Jotta pyörimisnopeus saadaan luettua oikein, tulee ohjelmoitavan logiikan pystyä tunnistamaan pulsseja kyseiseltä väliltä.

Käyttäjän ja työkoneen välinen rajapinta puolestaan haluttiin pitää mahdollisimman yksinkertaisena ohjelmoitavien logiikkojen ulostulojen ja sisäänmenojen kannalta. Ideana oli toteuttaa kaikki varoitukset ja ilmoitukset graafiseen käyttöliittymään kuin myös sellaiset painokytkimet, joita ei tarvitse ajettaessa työkoneella. Lähtökohta rajapinnalle oli alkuperäisen työkoneen toteutus, jota karsittiin edellä mainituin kriteerein poistaen esimerkiksi kaikki varoitusvalot ja työkoneen liikennekelpoisuuteen liittyvät komponentit. Prosessin lopputulokseksi jäi aikaisemmin mainitun kaasupolkimen lisäksi ainoastaan painokytkimet seisontajarrulle ja työkalun lukitusventtiilille. Nämä painokytkimet vaativat molemmat digitaaliset sisäänmenot.

Alkuperäisestä käyttäjän ja työkoneen välisestä rajapinnasta säilytettiin myös ohjaussauva puomin liikuttamista varten. Ohjaussauva sisältää sen akselien lisäksi 4 painokytkintä, joista kahdella vaihdetaan työkoneen ajosuunta ja toisella kahdella liikutetaan työkalunlukituksen suuntaventtiiliä, kuten alkuperäisessäkin työkoneessa. Ohjaussauva ei vaadi kuitenkaan ohjelmoitavalta logiikalta sisäänmenoja tai ulostuloja, sillä se liitetään ohjausjärjestelmään tiedonsiirtoratkaisulla.

5.1.4 Tuettavat tiedonsiirtoratkaisut

Ohjelmoitavilta logiikoilta vaadittujen sisäänmenojen ja ulostulojen lisäksi niiden tulee mahdollistaa tiedonsiirtoratkaisuiden avulla kytkettävien laitteiden liittäminen ohjausjärjestelmään. Demonstraatiokoneen ohjausjärjestelmään tiedonsiirtoratkaisulla kytkettävien komponenttien datalehdet käytiin läpi ja selvityksen pohjalta koostettiin taulukko 7.

Taulukko 7. *Demonstraatiokoneessa käytetyt tiedonsiirtoratkaisut ja komponenttien määrä*

CAN-laitteet	Tyyppi (baudinopeus [kbit/s])	Määrä
INVCOM2.3 -invertteri	Bosch CAN (500)	3
Ohjauksen kulma-anturi	CANopen (50, 125, 250, 500,1000)	1
Pinnankorkeusanturi	CANopen (10, 20, 50, 125, 250, 500, 800, 1000)	2
Tehonjakoyksikkö	CANopen / J1939 (50, 125, 250, 500, 800, 1000)	1
Korkeajännitelaturi	CANopen (20, 50, 125, 250, 500, 800, 1000)	1
Ohjaussauva	J1939 (250)	1
Dieselmoottori	J1939 (250)	1
DC/DC-muunnin	J1939 (125, 250, 500, 1000)	1
SEVCOM-invertteri	J1939 (10, 20, 50,100, 125, 250, 500, 1000)	2
Korkeajänniteakkumoduuli	J1939 (50, 125, 250, 500, 1000)	7

Kuten taulukosta 7 huomataan, käyttävät demonstraatiokoneen tiedonsiirtoratkaisun avulla kytkettävät komponentit kolmea eri CAN-pohjaista ratkaisua. Näistä INVCOM2.3-invertterien käyttämä Boschin oma toteutus CAN-väylästä perustuu myös ISO 11898-standardiin, kuten CANopen- ja J1939-tiedonsiirtoratkaisut, mutta ylempi taso tiedonsiirtoratkaisusta on valmistajan itse määrittelemä. Tiivistettynä Bosch implementaatio CAN-väylästä käyttää kiinteää 500 kbit/s baudinopeutta ja normaalia viestikehystä. Se ennalta määrittää viidelle INVCOM2.3-invertterille niiden lähettämät ja vastaanottamat viestit. Näissä viesteissä kulkevat kaikki toimintaan liittyvät parametrit, eikä erillistä säätöön liittyvää konfiguraatiota tarvita. Ainoastaan invertterien järjestysnumero jää valittavaksi valmistajan omalla konfigurointisovelluksella.

Jotta INVCOM2.3-inverttereitä voidaan ohjata logiikalla, tulee logiikalla pystyä lähettämään ja vastaanottamaan normaaleja viestikehyksiä riippumatta ylemmän tason standardista. Koska J1939- ja CANopen-standardeihin perustuvien tiedonsiirtoratkaisujen toiminta on hieman monimutkaisempaa kuin Boschin implementaation, on tavoiteltavaa löytää ohjainlaitteet, joiden ohjelmointiympäristö tarjoaa suoraan ohjelmakirjastot helpottamaan näiden standardien käyttöä.

Taulukon 7 laitteista CANopen-laiteiden mukana toimitettiin EDS-tiedostot ja niiden kommunikaatioon liittyvä konfigurointi tapahtuu CANopen-standardin mukaisesti huoltodata-objekteilla. Koska J1939-standardissa vastaavaa protokollaa ei ole, tulee konfigurointi tehdä komponenttivalmistajan ohjelmistolla, kuten INVCOM2.3-inverttereidenkin tapauksessa. J1939-komponenteista ohjelmistot konfigurointia varten saatiin DC/DC-muuntimelle, Sevcomin valmistamille invertterille sekä korkeajänniteakustolle. Ohjaussauvan ja dieselmoottorin osalta konfigurointi ei ole mahdollista käyttäjän toimesta, jonka vuoksi alustavasti ohjaussauva ja dieselmoottori tulisi kytkeä 250 kbit/s baudinopeudella toimivaan väylään.

Koska kaikki ohjausjärjestelmän CAN-laitteet käyttävät ISO 11898 mukaisia CAN2.0A tai CAN2.0B lähetinvastaanottimia, on teoriassa mahdollista kytkeä kaikki laitteet samaan väylään huolimatta siitä, mitä ylemmän tason standardia käytetään. Koska taulukossa 7 esitettyjen laitteiden ei tarvitse vaihtaa informaatiota keskenään, vaan ainoastaan valittavien ohjelmoitavien logiikkojen kanssa, riittää että ohjelmoitava logiikka itsessään kykenee lähettämään ja vastaanottamaan useamman eri ylemmän standardin viestejä samaan aikaan.

Luvun 4 selvityksen mukaan huomioitavaa ainakin on, että:

- kaikkien lähetettävien viestien identifiointikenttien tulee olla ainutlaatuisia,
- normaalin viestikehyksen mukaiset viestit priorisoituvat jatkettua kehystä käyttävien viestien edelle,
- baudinopeus tulee olla kaikille laitteille asetettu samaksi ja
- kaistaa tulee olla riittävästi.

Koska demonstraatiokoneessa kaikkien laitteiden baudinopeutta ei voida valita samaksi eikä todennäköisesti suurinkaan CAN-väylän baudinopeus mahdollistaisi tarpeeksi suurta kaistaa tällöin väylällä liikkuvalle datamäärälle, tarvitaan demonstraatiokoneeseen useampi väylä. Lisäksi, koska Boschin implementaatiossa identifiointikenttään ei vaikuta laitteen solmunumero, tulisi kaikki samalla väylällä liikkuvat – myös normaalia viestikehystä käyttävät – CANopen-viestit tarkistaa, etteivät CANopen-laitteiden solmunumerot luo sellaisia viestejä, joiden identifiointikenttä on sama kuin INVCOM2.3-invertereiden lähettämissä tai vastaanottamissa viesteissä. Koska tämän voidaan nähdä aiheuttavan ylimääräistä kompleksisuutta varsinkin tilanteessa, jossa myöhemmässä vaiheessa komponentteja vaihdetaan, vaaditaan CANopen-laiteet sijoitettavaksi eri väylään kuin INVCOM2.3-invertterit. Komponenttien sijoittamisessa tulee myös huomioida, että kaikki korkeajänniteakuston moduulit tulee asettaa samaan väylään, sillä akkumoduulit vaihtavat keskenään tietoa esimerkiksi kennojen jännitteistä kennojen tasapainottamista varten.

Edellä esitettyjen rajoitusten lisäksi komponenttien vaatima kaista CAN-väylältä vaikuttaa komponenttien sijoittamiseen eri väyliin. Koska komponenttien manuaalit esittävät laitteiden lähettämät ja vastaanottamat viestit, voidaan niiden avulla arvioida väyläkuormitus laitetta kohden. Tämä tapahtuu laskemalla yhden bitin lähettämiseen vaadittu aika eli nimellisbittiaika ottamalla baudinopeudesta käänteisluku. Tämän jälkeen nimellisbittiaika kerrotaan viestin pituudella bitteinä ja saadaan yhden viestikehyksen lähettämiseen vaadittu aika. Jakamalla tämän jälkeen vaadittu aika viestin lähetysvälillä saadaan viestin vaatima kaista. (Vector 2023) Laskutoimitus voidaan esittää kaavana muodossa

$$viestin\ vaatima\ kaista\ = \frac{\frac{1}{baudinopeus} * viestikehyksen\ pituus}{lähetysväli}. \qquad (4)$$

Laskutoimituksessa vaadittu viestikehyksen pituus saadaan laskemalla kuvassa 6 esitettyjen viestikehyksien bitit yhteen. Koska viestikehyksen pituus riippuu osaksi datatavujen määrästä, voidaan normaalille kehyksen pituudelle johtaa kaava 47 + 8 * DLC ja jatketulle 67 + 8 * DLC. Tämän lisäksi bit stuffing -menetelmä tulee huomioida. Kuten

luvussa 4 on esitetty bit stuffing -menetelmässä joka viiden saman tilan bitin jälkeen viestiin lisätään yksi ylimääräinen vastakkaisen tilan bitti.

Koska lisättävien bittien maksimimäärä riippuu viestin pituudesta ja tiettyyn viestiin lisätyt bitit viestin sisällöstä, on bit stuffing -menetelmän aiheuttamaa väyläkuormaa vaikea arvioida. Pahimmillaan normaalin kehyksen viestissä voi olla 18 ylimääräistä bittiä, kun viesti on täyspitkä (Vectror 2023). Tässä työssä laskentaa yksinkertaistetaan olettamalla jokaisessa normaalin kehyksen viestissä olevan 10 lisättyä bittiä ja jatketussa kehyksessä 13. Siitä huolimatta, että bit stuffing -menetelmän lisäämät bitit voitaisiin määrittää tarkasti, on menetelmä joka tapauksessa vain suuntaa antava, sillä todellisuudessa väylä ei toimi deterministisesti ja riippuen käytetystä standardista voi väylällä liikkua jonkin verran tapahtumapohjaisia viestejä.

Koska demonstraatiokoneen laitteet lähettävät ja vastaanottavat pääsääntöisesti useita viestejä kirjoitettiin Matlab ohjelma 1 kaavan 4 pohjalta, jolla voidaan laskea yhden laitteen vaatima kaista CAN-väylältä. Ohjelmassa hyödynnetään vektoreita, joissa laskutoimitukset tehdään alkioittain ja lopuksi laitteen kaistavaatimus muodostetaan summaamalla yksittäisten viestien kaistavaatimukset. Ohjelmassa on käytetty esimerkkinä Sevcom-invertterien vastaanottamia ja lähettämiä viestejä.

```
baudinopeus          = 500 * 10^3; % [b/s]
lahetysvali          = [ 5, 10, 50, 100, 1000] * 10^-3; % [s]
viestien_maara       = [ 6,  3,  1,   1,    2];
DLC                  = [ 8,  8,  8,   8,    8]; % datatavujen määrä

kehyksen_bittimaara  = (80 + 8 * DLC);


bittiaika = 1 / baudinopeus; % [s/b]

viestiaika = kehyksen_bittimaara .* bittiaika;

kaista_per_viesti = viestiaika ./ lahetysvali * 100; % [%]

kaistavaatimus = sum(kaista_per_viesti .* viestien_maara) % [%]
```

Ohjelma 1. *Matlab-ohjelmakoodi CAN-laiteen vaatiman kaistan määrittämiseksi*

Laskenta toistettiin kaikille taulukon 7 komponenteille ja tulokset on esitetty taulukossa 8 pyöristettynä ylöspäin puolen prosentin tarkkuuteen. Laskennassa käytettiin pääsääntöisesti 500 kbit/s baudinopeutta niiden laitteiden kohdalla, jotka tukevat sitä. Huomioitavaa on, että tulokset ovat lineaarisesti skaalattavissa muille baudinopeuksille.

Taulukko 8. *CAN-laitteiden vaatima kaista laskennallisesti*

CAN-laitteet	baudinopeus [kbit/s]	Kaistavaatimus
INVCOM2.3 -invertteri	500	11,5 %
Ohjauksen kulma-anturi	500	0,5 % *
Pinnankorkeusanturi	500	2,5 % *
Korkeajännitelaturi	500	0,25 % *
Tehonjakoyksikkö	500	0,5 % *
Ohjaussauva	250	3,0 %
Dieselmoottori	250	13,5 %
DC/DC-muunnin	500	0,50 %
SEVCOM-invertteri	500	44,5 % *
Koko korkeajänniteakusto	500	13,0 %

* Kaistavaatimus riippuu konfiguraatiosta. Tulokset laskettu käyttäen vakioasetuksia.

CiA (2023c) ohjeistaa, ettei CAN-väylän keskimääräinen kuormitusaste tulisi ylittää 50 % saatavilla olevasta kaistasta. Tämän taustalla lienee tarve jättää kaistaa tapahtumapohjaisille viesteille sekä muun muassa CAN-standardin vikakehyksille. Kokemuksen mukaan yhden laitteen viallinen toiminta voi nostaa merkittävästi väyläkuormitusta ja pahimmassa tapauksessa nostaa kuormituksen sataan prosenttiin, jolloin vain osa viesteistä lähtee väylälle ja näin ollen yhden viallisen laitteen toiminta vaikuttaa herkimmin kaikkiin väylällä oleviin laitteisiin.

Taulukossa 8 esitetyistä kaistavaatimuksista huomataan, että yhden Sevcomin valmistaman invertterin kanssa kommunikointiin tarvittaisiin vakioasetuksilla käytännössä oma väylä. Huomioitavaa kuitenkin on, että kyseisten invertterien tapauksessa väyläkuormitusta voidaan laskea esimerkiksi muuttamalla viestien lähetys- ja vastaanottovälejä valmistajan toimittaman sovelluksen avulla. Invertterit lähettävät ja vastaanottavat vakioasetuksilla useita viestejä 5 ms välein, joiden lähetysvälin nostaminen 10 ms tiputtaisi merkittävästi laitteen vaatimaa kaistaa. Huomioitavaa on, että mikään muukaan järjestelmän laite ei käytä vakioasetuksilla alle 10 ms lähetysväliä.

Yleisesti ottaen pohdittaessa laitteiden sijoittamista eri väyliin, ei väylien määrän minimointi ole erityisemmin tavoiteltavaa varsinkaan hybridityökoneessa, jossa useat korkeajännitekomponentit aiheuttavat sähkömagneettista häiriöitä, vaikka se vähentääkin tarvittua kaapelointia. Tämä johtuu sitä, että häiriösietoisuus ja samalla myös luotettavuus paranee, mitä pienempi väylän baudinopeus on, mitä lyhempiä väylän runko ja sen haaroitukset ovat sekä mitä vähemmän väylään kytkettyjä laitteita on. Lähtökohtaisesti tärkeämpi tavoite on laittaa sellaiset komponentit samaan väylään, jotka kommunikoivat keskenään, jotta ohjelmoitavien logiikoiden ei tarvitse siirtää viestejä väylästä toiseen.

5.2 Arkkitehtuuri

Kuten luvussa 3 on mainittu, yksi perusvanlaatuisimmista arkkitehtuuriin määritettävistä asioista on hajautusaste. Ääripäistä mahdollisimman keskitetty ratkaisu mahdollistaisi erilaisten ohjausalgoritmien testaamisen nopeammin, kun tarvittavat muutokset voidaan tehdä aina yhdelle laitteelle eikä kommunikointirajapintoja tarvitse muokata prosessissa. Toisaalta hajauttamisella saavutetaan modulaarisempi ratkaisu, josta on esimerkiksi päivittämisen yhteydessä helpommin löydettävissä koodivirheet. Tämä johtuu siitä, että virheellisen toiminnan aiheuttava tekijä on usein helposti kohdistettavissa yhteen ohjainlaitteeseen, kun tiedetään mihin toimintokokonaisuuteen virhe liittyy ja minkä ohjainlaiteen ohjauskoodia viimeksi päivitettiin.

Merkittävä etu hajauttamiselle on myös, ettei virheellinen ohjauskoodi ja siitä mahdollisesti aiheutuva virheellinen toiminta vaikuta yhtä helposti muuhun toimintaan. Lisäksi varsinkin työkoneisiin soveltuvissa ohjelmoitavissa logiikoissa laskentatehoa on yhtä ohjainlaitetta kohden suhteellisen rajallisesti saatavilla, jolloin hajauttamisesta voi olla hyötyä myös uusien ohjausalgoritmien laskentatehovaatimuksien saavuttamisessa. Selkeä laskentatehoon vaikuttava tekijä on usein moniydinlaskentatuen puuttuminen työkoneisiin soveltuvissa logiikoissa.

Hajauttamisen hyötyjen ja haittojen sekä luvussa 5.1 esitettyjen vaatimuksien ja tavoitteiden pohjalta keskeiseksi hajauttamisen lähtökohdaksi muodostui toiminnon oleellisuus tutkimuksen kannalta. Ideassa tutkimuksen kannalta oleelliset toimintakokonaisuudet on tarkoitus toteuttaa keskitetysti yhdellä ohjainlaitteella, jolloin tutkimuskäytössä muutoksien tekeminen tapahtuu yhdelle laitteelle eikä kommunikaatiorajapintoja tarvitse päivittää jatkuvasti. Tutkimuksen kannalta ei niin oleelliset toiminnot, mutta turvallisuuden ja työkoneen liikkumisen kannalta kuitenkin vaaditut toiminnot, on puolestaan tarkoitus toteuttaa tarpeellisella määrällä erillisiä ohjainlaitteita. Tällöin näitä toimintoja ohjaavien ohjainlaitteiden ohjauskoodeihin ei tarvitse tehdä lähtökohtaisesti muutoksia, jolloin erilaisten ohjausalgoritmien testaaminen ei lähtökohtaisesti riko muita toiminnallisuuksia. Tutkimuksen kannalta oleelliset toiminnallisuudet tunnistettiin liitteen A käyttötapauskaaviosta, joka esitellään tarkemmin luvussa 5.2.2 ohjelmallisen arkkitehtuurin käsittelyn yhteydessä. Oleellisia toiminnallisuuksia ovat tehonhallinta, puomin hallinta ja ajovoimansiirronhallinta.

Koska lähtökohtaisesti tavoitteeksi asetettiin ohjausjärjestelmän modulaarisuus, jotta laskentatehoa voidaan nostaa tarvittaessa myöhemmin, koettiin, että tutkimuksen kannalta oleellisten toimintojen keskittämisellä päästään lähtökohtaisesti parempaan loppu-

tulokseen kuin niiden lähtökohtaisella hajauttamisella. Toiminnot on kuitenkin syytä toteuttaa ohjelmallisesti omina kokonaisuuksinaan, jotta ne voidaan siirtää tarpeen mukaan sellaisinaan esimerkiksi tulevaisuudessa hankittaville ohjainlaitteille. Tarve hajauttamiselle selviää vasta monitoroimalla ohjainlaitteen kiertoaikaa, kun se suorittaa aktiivisesti ohjelmaa.

5.2.1 Fyysinen arkkitehtuuri ja laitevalinnat

Fyysisen arkkitehtuurin muodostaminen aloitettiin pohtimalla niin sanottuun keskusyksikön valintaan vaikuttavia tekijöitä, millä on tarkoitus toteuttaa juuri tutkimuksen kannalta oleellisten kokonaisuuksien ohjaus. Niiden toteuttamiseksi keskusyksikön tulee pystyä kommunikoimaan CAN-väylän avulla taulukossa 7 esitettyjen voimalinjan komponenttien sekä ohjaussauvan kanssa. Lisäksi keskusyksikön tulee kyetä ohjaamaan ja vastaanottamaan tietoa kaasupolkimelta ja työhydrauliikkaan liittyviltä I/O-komponenteilta.

Alkuvaiheessa edellä mainittujen I/O-komponenttien kytkemiseksi ajateltiin käytettävän I/O-lisälaitteita, mutta 50 venttiilin virtaohjauksen sekä kuuden paineanturin ja kahden lineaaripotentiometrin mittauksen lähettäminen todettiin lähes mahdottomaksi käyttäen klassista CAN-väylää. Jotta kokonaisviive venttiilin avaamiseksi saadaan pidettyä alle vaaditun kahden millisekunnin, tulisi keskusyksikön kiertoajan, CAN-viestin lähetysvälin ja I/O-lisälaitteen reagointiajan summa olla alle kahden millisekunnin. Kokemuksen mukaan logiikat itsessään pystyvät yleensä reilusti alle millisekunnin kiertoaikoihin, mutta vaaditun datan lähettäminen CAN-väylän avulla on haasteellista edes kahden millisekunnin välein.

Tämä johtuu siitä, että virran, paineen ja asennon arvot voidaan lähettää väylälle tarpeeksi hyvällä resoluutiolla 16 bittisellä kokonaisluvulla, jolloin edellä mainittujen tietojen lähettäminen väylälle vaatisi vähimmillään 15 kokonaista viestiä. Ohjelman 1 avulla laskettuna näiden siirtäminen 2 ms välein aiheuttaisi 1 Mbit/s baudinopeudella 91 % kuorman käytettäessä normaalikehystä ja 108 % kuorman käytettäessä jatkettua kehystä. Käyttäen 50 % väyläkuormituksen mitoitusperiaatetta tarvittaisiin 2–3 väylää pelkästään näiden IO-komponenttien kytkemiseksi. Huomioitavaa on, että lähetysväli pitäisi olla todellisuudessa vieläkin tiheämpi, jotta 2 ms kokonaisviiveeseen päästäisiin.

Tämän pohjalta heräsi ajatus siitä, että digitaalisten virransäätöyksiköiden venttiilikohtainen ohjauslogiikka voitaisiin eriyttää puominohjauksessa, jolloin ohjaukseen riittäisi yksi väylä. Ratkaisussa puomin ohjausalgoritmi ohjaisi digitaalisiin virransäätöyksiköihin kiinnitettyjä ohjelmoitavia logiikoita CAN-liitännäisten proportionaaliventtiilien tavalla eli yhdellä 16 tai 32 bittisellä kokonaisluvulla. Tämän jälkeen tämä alemmalla tasolla oleva logiikka valitsisi ylemmän tason ohjauksen ja painelinjojen paineiden avulla tilanteeseen

sopivan avauksen ja hoitaisi muun muassa venttiilien avauksen nopeuttamisen. Alemman tason logiikka voisi valita puomin liikkeiden ohjaamiseen käytettyjen digitaalisten virransäätöyksiköiden tapauksessa esimerkiksi differentiaalikytkennän itse, mikäli se sopii tilanteeseen.

Idean osoittautuessa toimivaksi voisi se mahdollisesti luoda tuotekonseptin teollisuuteen helposti adaptoitavissa olevasta digitaalisesta virtauksensäätöyksiköstä, joka ei vaadi ohjausjärjestelmäsuunnittelijalta tarkempaa ymmärrystä digitaalihydrauliikasta tai erillisreunaohjauksesta. Lisäksi ratkaisu tiputtaisi myös keskusyksikön ohjelman kiertoaikavaatimusta kommunikaation suhteen kahdesta millisekunnista invertterien vaatimaan kymmeneen tai viiteen, mikäli Sevcomin valmistamien inventterien viestien lähetysväliä ei tiputeta. Kuten luvussa 5.1.1 mainittiin, vaadittiin kahden millisekunnin vaste venttiilien avauksien aloittamiseksi perustuen paineen muutokseen, eikä ylemmän tason referenssin muutokseen.

Idean toteuttamiseksi päätettiin valita kolme ohjelmoitavaa logiikkaa: molemmille puomin liikkeitä ohjaaville digitaalisille virtauksensäätöyksiköille oma ja paineakkujen varaamiseen käytetyille kahdelle 2/2-virtauksensäätöyksiköille yhteinen. Koska paineakkujen varaamiseen käytettyjä venttiileitä on yhteensä vain 10, päätettiin alustavasti kolmanteen logiikkaan kytkeä loputkin demonstraatiokoneen I/O-kytkentäiset komponenteista, mutta toteuttaa myös tutkimuksen kannalta ei niin tärkeiden kokonaisuuksien ohjaus. Näiltä ohjelmoitavilta logiikoilta vaadittu I/O on esitetty taulukossa 9.

Taulukko 9. *Ohjelmoitavilta logiikoilta vaadittu I/O*

Liikkeenohjauksen virransäätöyksiköt	**Muu I/O**
20 kpl 1,5 A virtaohjattuja ulostuloa	10 kpl 1,5 A virtaohjattua ulostuloa
3 kpl 4–20 mA sisääntuloa	5 kpl 1,5A Digitaalista ulostuloa
	4 kpl n. 20 A Digitaalista ulostuloa
	1 kpl PWM-ohjattua ulostuloa
	6 kpl 4–20 mA sisääntuloa
	2 kpl 0–10 V sisääntuloa
	2 kpl 0–4,5 V sisääntuloa
	1 kpl Pulssilaskureita
	3 kpl Digitaalisia sisääntuloa
	4 kpl reguloituja 5 V -referenssejä

Idean myötä myös keskusyksikköön tarvittavia väyläliitäntöjen määrää pohdittiin ottaen huomioon myös muut luvussa 5.1.4 esitetyt rajoitukset laitteiden sijoittamiselle eri väyliin. Alustavasti todettiin, että tutkimuksen kannalta oleellisten kokonaisuuksien ohjaukseen liittyvät komponentit voitaisiin kytkeä neljällä väylällä keskusyksikköön:

1. väylä Sevcom-inverttereille,
2. INVCOM2.3-inverttereille,
3. dieselmoottorille, ohjaussauvalle, tehonjakoyksikölle, DC/DC-muuntimelle, korkeajänniteakustolle ja
4. näytölle ja puomin ohjaamiseen tarvituille alemman tason logiikoille.

Vähintään neljän väyläliitännän lisäksi keskusyksikön valinnan kriteerinä – kuten muillekin demonstraatiokoneeseen valittaville ohjelmoitaville logiikoille – on aikaisemmin 5.1 luvussa asetukset vaatimukset siitä, että laitteen tulee olla kaupallinen helposti saatava tuote ja soveltua työkoneisiin sekä tukea mallipohjaista ohjelmistokehitystä Matlab-Simulink ympäristössä eikä sen ohjelmointi saa vaatia erillistä maksullista lisenssiä. Lisäksi laitteesta tulisi löytyä suhteellisen tehokas laskentaydin tukemaan tulevaisuuden ohjausalgoritmien kehittämistä.

Ohjainlaitteet päätettiin valita tutkimusryhmän aikaisemman hyvän kokemuksen pohjalta EPECin (2023b) tuotevalikoimasta. EPECin ohjainlaitteet ohjelmoidaan IEC 61131–3 mukaisella PLC-koodilla käyttäen Codesys-kehitysympäristöä ja näin ollen mallipohjainen ohjelmistokehitys onnistuu Matlab–Simulink -ympäristössä Simulinkistä löytyvän PLC-koodin generointilisäosan ansiosta. Päätöstä puolsi myös Ethernetin välityksellä tehtävä ohjainlaitteiden ohjelmointi sekä kommunikaatiorajapintojen luomisen käytetty graafinen Multitool-sovellus. Tiivistettynä Multitool luo ja ylläpitää ohjainlaitteiden kehi-

tysympäristön koodipohjaa, joka parametrisoi valmiiksi käyttöliittymän avulla määritettyjen J1939- ja CANopen-viestien parametrit sekä pitää sisällään käyttöliittymän avulla määritetyt väylien alustukseen ja IO-pinneihin liittyvät asetukset. EPECin ohjelmointikirjastot mahdollistavat myös raakaviestien lukemisen ja lähettämisen väylälle, joka vaaditaan INVCOM2.3-invertterien kommunikaatiorajapinnan luomiseksi.

Tuotevalikoimasta digitaalisille virtauksensäätöyksiköille ja muulle I/O-komponenteille valittiin yhteensä kolme 5050 ohjainlaitetta, joissa on neljä väylää ja 160 MHz yksiytiminen prosessori. I/O-liitännäisyyden osalta 5050 ohjainlaitteista löytyy 65 liitäntää, joista kaksikymmentä ulostuloa voidaan asettaa virtasäädettäviksi riittäen näin niukasti puomin liikkeen ohjaamiseen käytettyjen digitaalisien virtauksensäätöyksiköiden venttiileille. Se tarjoaa myös tarpeeksi liitännällisyyttä muille taulukkoon 9 listatuille I/O-komponenteille, kun korkeimmat 20 A lähdöt toteutetaan releiden avulla.

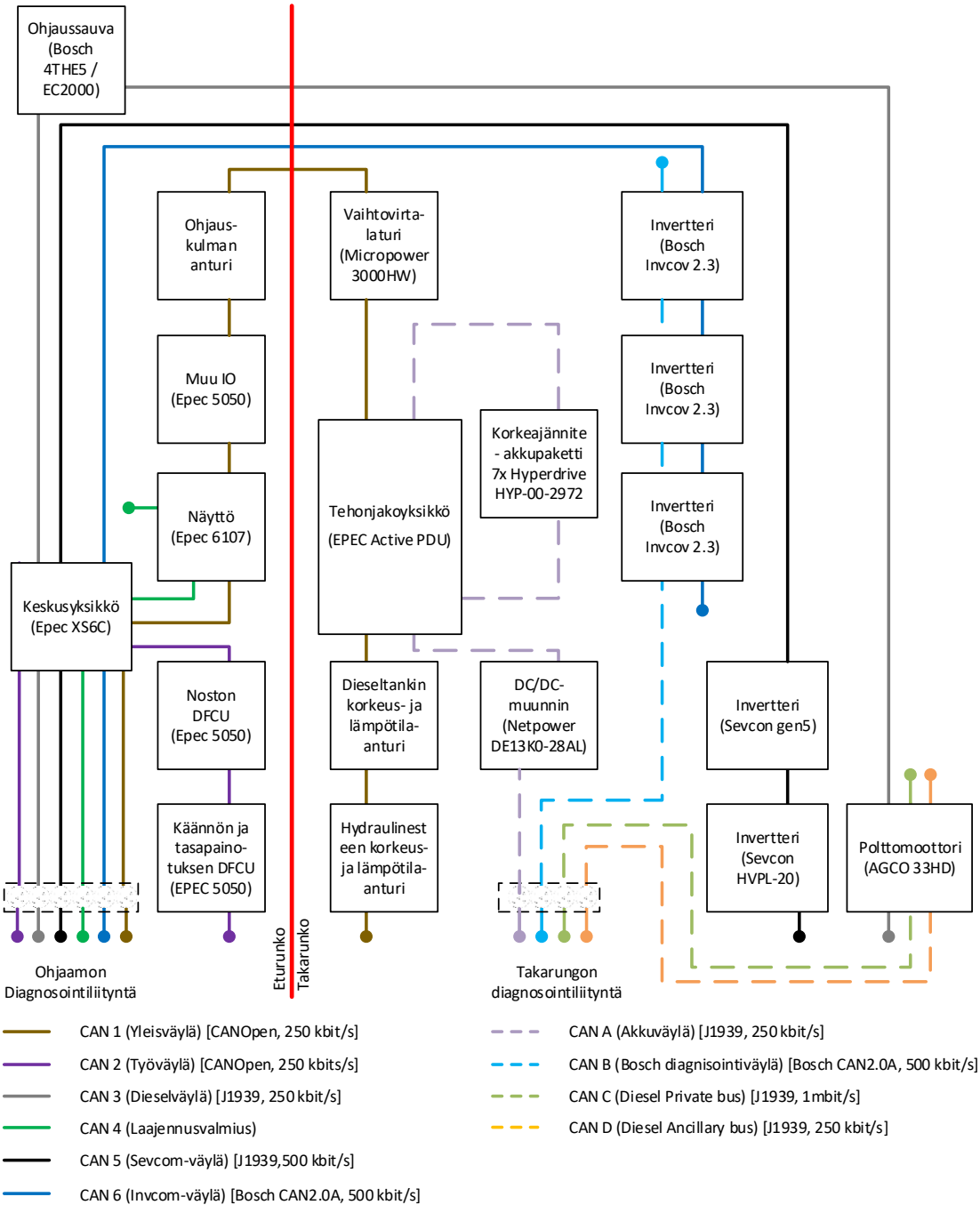
Digitaalisille virtauksensäätöyksiköille valittavien ohjelmoitavien logiikkojen valinnan yhdeksi haasteeksi muodostui logiikkojen virtalähteiden riittävyys. Valituissa 5050 ohjainlaitteissa on tuotekatalogin tehokkaimmat virtalähteet, jotka tarjoavat kaksi 14 A virtalähdettä. Nämä on kuitenkin jaettu virtasäädettävien lähtöjen suhteen siten, että ensimmäisessä virtalähteessä on kiinni 4 lähtöä ja toisessa 16. Mikäli toisen virtalähteen alla olevat 16 venttiiliä olisivat samaan aikaan auki, olisi virrankulutus reilusti yli sallitun. Venttiilit onnistuttiin kuitenkin jakamaan virtalähteiden alle siten, etteivät virtalähteet rajoita digitaalisen virtauksensäätöyksikön normaalia toimintaa. Jako tehtiin siten, että kaksi venttiiliä painelinjasta molempiin työlinjoihin allokoitiin ensimmäisen virtalähteen alle ja loput toisen virtalähteen alle. Täten normaalissa toiminnassa toisen virtalähteen alla on maksimissaan 8 venttiiliä auki luoden 12 A kuorman virtalähteelle, kuten taulukko 10 osoittaa. Taulukossa venttiilit on nimetty niiden yhdistämien virtausreunojen lyhenteiden mukaan, jotka on esitetty kuvassa 18.

Taulukko 10. *Venttiilien allokointi 5050-ohjainlaitteiteen virtalähteille*

Virtalähde 1				
Pinnin numero	Venttiili	Plus-liike	Diff. kytkentä	Miinus-liike
1	PA1	x	x	
2	PA2	x	x	
3	PB1		x	x
4	PB2		x	x
venttiilien lkm.		2	4	2
Virtalähde 2				
1	PA3	x	x	
2	PA4	x	x	
3	PA5	x	x	
4	PB3		x	x
5	PB4		x	x
6	PB5		x	x
7	AT1			x
8	AT2			x
9	AT3			x
10	AT4			x
11	AT5			x
12	BT1	x		
13	BT2	x		
14	BT3	x		
15	BT4	x		
16	BT5	x		
venttiilien lkm.		8	6	8

Keskusyksiköksi puolestaan valittiin Linux-pohjainen XS6C, jossa on kuusi väylää ja neljäytiminen Cortex A9 0,8 GHz ARM prosessori. Lisäksi valittavana oli näyttö, joksi valittiin kosketusnäytöllinen 6107. Kyseinen näyttö on myös Linux-pohjainen ja siitä löytyy kaksiytiminen versio myös XS6C-yksikössä käytetystä prosessorista. Osa näytön laskentatehosta menee luonnollisesti graafisen käyttöliittymän pyörittämiseen, mutta hyvin todennäköisesti näyttöä voidaan hyödyntää projektin myöhäisemmässä vaiheessa tutkimuksen kannalta oleellisten kokonaisuuksien ohjaamisen hajauttamiseksi, mikäli keskusyksikön laskentateho ei yksinään riitä. Lähtökohtaisesti näyttö ei kuitenkaan sovellu korvaamaan keskusyksikköä, sillä siinä ei ole kuin kaksi väyläliitäntää. Kummankin laitteen tapauksessa Linux-käyttöjärjestelmällä käytetään Codesys-rajapintaa (engl. Codesys runtime), mutta käyttöjärjestelmällä on mahdollisuus ajaa myös muita sovelluksia. Tällä hetkellä kummankaan laitteen ohjelmistossa (engl. firmware) löytyvä Codesys-rajapinta ei tue moniydinlaskentaa, mutta sen päivittäminen uudempaan moniydintuelliseen rajapintaan on suunnitteilla.

Laitteiden hankinnan jälkeen arkkitehtuuri iteroitui kuvassa 22 esitettyyn muotoon. Komponenttien sijoittamiseen eri väyliin vaikutti komponenttien fyysinen sijainti demonstraatiokoneessa ja halu minimoida väylien kokonaismäärä. Väylien haluttiin myös päättyvän ohjaamoon, jotta väylillä liikkuvia viestejä voidaan nauhoittaa työsyklin aikana tutkimustarkoitusta varten. Nauhoitus halutaan lähtökohtaisesti tehdä yhdellä useampikanavaisella CAN-analysointilaitteella, jotta eri väylien viestiliikennettä ei tarvitse kohdistaa ajallisesti helpottaen näin kerätyn datan analysointia.



Kuva 22. *Demonstraatiokoneen ohjausarkkitehtuuri*

Poiketen alkuperäisestä ideasta keskusyksikön hankinnan kriteerinä olleesta neljästä väylästä keskusyksikön viidennen väylän käyttöönotto mahdollisti kaikkien laitteiden lukuun ottamatta korkeajänniteakuston ja DC/DC-muuntimen kytkemisen samoihin väyliin. Tällöin myös apuinformaation siirtäminen, kuten polttoaineen pinnankorkeusanturin mittaustuloksen välittämiseen näytölle, onnistuu ilman muiden ohjainlaitteiden erillisiä väyliä, vähentäen juuri väylien kokonaismäärää. Valitussa arkkitehtuurissa korkeajänniteakuston ja DC/DC-muuntimen käyttöön oleellisesti liittyvät parametrit reititetään akkuväylästä yleisväylään. Ratkaisun taustalla on akuston vaatima kaista, jonka takia moduuleita ei voi suoraan laittaa kuvassa 22 esitettyyn yleisväylään, mutta myös verkkovirtalatauksen toteuttaminen.

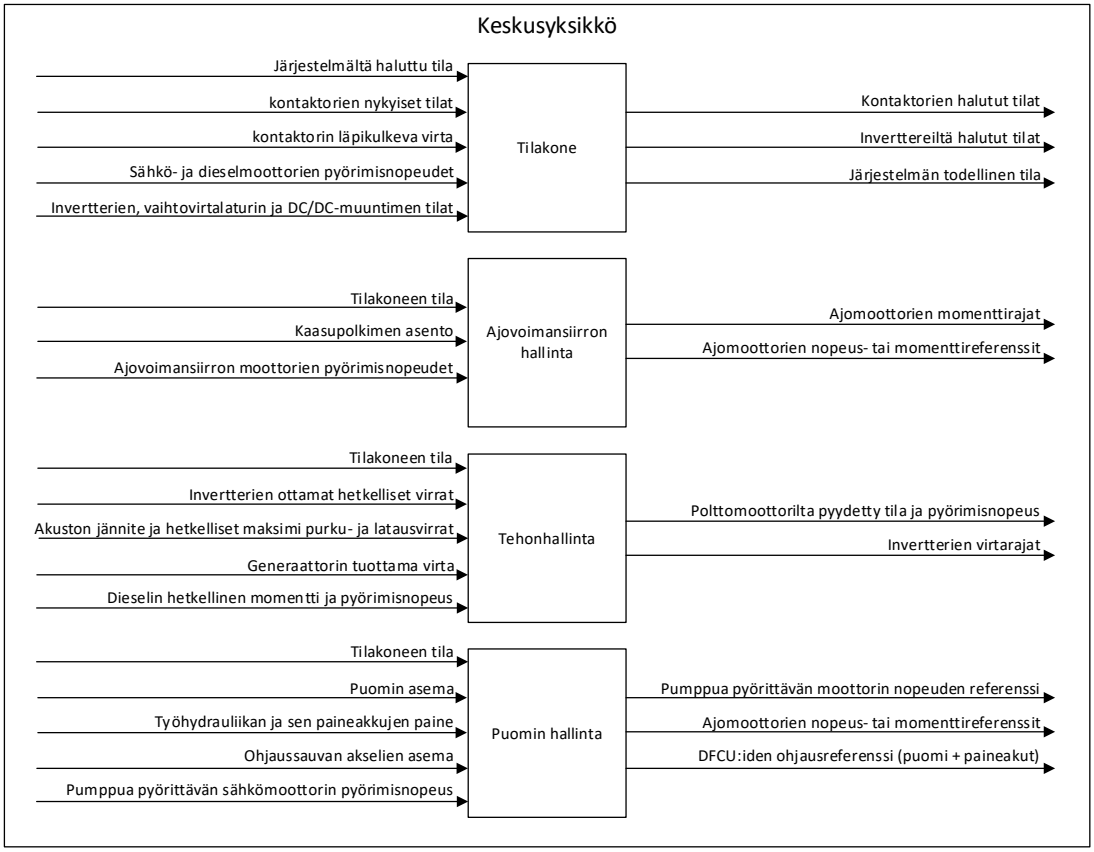
Jotta verkkovirtalataus voidaan suorittaa, tarvitsee vaihtovirtalaturin lisäksi tehonjakoyksikön olla päällä kontaktorien sulkemista varten, mutta myös korkeajänniteakuston hallintayksikön (engl. battery management system, BMS). Tehonjakoyksikkö ja akuston hallintayksikkö toimivat 24 V -järjestelmän avulla, minkä vuoksi myös DC/DC-muuntimen tulee olla päällä latauksen aikana 24 V akuston tyhjenemisen välttämiseksi. Koska tehonjakoyksikkö (engl. power distribution unit, PDU) hankittiin demonstraatiokoneeseen räätälöitynä, onnistui sen sisäiseen ohjelmoitavaan logiikkaan sisällyttää omaa ohjauskoodia. Tehonjakoyksiköllä päätettiin toteuttaa sen alkuperäisen käyttötarkoituksen eli sähköturvallisuuden valvonnan lisäksi myös korkeajänniteakuston valvonta sekä vaihtovirtalaturinlaturin ja DC/DC-muuntimen ohjaus. Ratkaisu mahdollistaa edellä mainittuja komponentteja lukuun ottamatta muun järjestelmän sammuttamisen verkkovirtalatauksen aikana eikä Muu IO -ohjainlaitteen tarvitse olla päällä hallitsemassa latausta, kuten alun perin suunniteltiin.

Laskettaessa taulukossa 8 esitettyjen laitteiden kaistavaatimuksien mukaan ja ottaen huomioon tarve siirtää informaatiota ohjelmoitavien logiikoiden välillä, pitäisi kaikkien väylien kuormituksen jäädä alle 50 %. Mikäli ohjelmoitavien logiikkojen välillä siirretään dataa suunniteltua enemmän, voidaan ensimmäisen ja toisen väylän baudinopeutta nostaa. Komponenteista keskusyksikön ja näytön välille lisättiin valmiiksi myös ylimääräinen väylä, mikäli myöhemmässä vaiheessa laskentaa halutaan hajauttaa keskusyksiköltä näytölle ja näin ollen kaistaa vaaditaan enemmän. Lähtökohtaisesti väylien baudinopeudeksi valittiin 250 kbit/s, sillä lähtökohtaisesti matala baudinopeus parantaa ratkaisun robustisuutta. Baudinopeudeksi määritettiin 500 kbit/s ainoastaan inverttereiden ohjaamiseen käytettyihin väyliin. Huomioitavaa on, että Sevcomin valmistamien invertterien väyläkuormitusta tulee laskea tiputtamalla ohjaamiseen käytettyjen viestien lähetysväliä 5 ms esimerkiksi 10 ms, jotta yhden 500 kbit/s väylän kaista riittää kahdelle invertterille. Vaihtoehtoisesti väylänopeus voidaan nostaa 1 Mbit/s.

5.2.2 Ohjelmallinen arkkitehtuuri ja ohjelmistomoduulit

Ohjelmallisen arkkitehtuurin muodostaminen tapahtui prosessissa samaan aikaan fyysisen arkkitehtuurin määrittämisen kanssa. Ohjelmallinen arkkitehtuuri koostuu ohjelmistomoduuleiden eli ohjelmiston itsenäisten osien, joilla on selkää tarkoitus ja rajapinta, määrittämisestä sekä niiden allokoinnista eri ohjainlaitteille. Ohjelmistomoduulien määrittämisen apuna muodostettiin liitteen A käyttötapauskaavio, jossa pyrittiin juuri löytämään toisistaan riippumattomia ohjattavia kokonaisuuksia eli sellaisia kokonaisuuksia, jotka voidaan helposti toteuttaa erillisillä ohjainlaitteilla, mutta niiden uudelleen hajottaminen pienempiin kokonaisuuksiin on joko haastavaa tai kannattamatonta.

Löydettyjä kokonaisuuksia olivat tehonhallinta, tilakone, ajovoimansiirron hallinta, apuhydrauliikan hallinta, työhydrauliikan hallinta, jäähdytysjärjestelmän hallinta, sähköturvallisuuden valvonta sekä sähkönjakeluverkosta lataamisen hallinta. Näistä tehon-, puomin- ja ajovoimansiirronhallinta on tarkoitus toteuttaa keskusyksiköllä, kuten aikaisemmin on mainittu. Näiden lisäksi myös tilakone päätettiin toteuttaa keskusyksiköllä, koska tilakone tarvitsee sisääntuloksi informaatiota useammalta eri väylältä, joihin kaikkiin ainoastaan keskusyksikkö on suoraan yhteydessä. Keskusyksikön ohjelmisto moduulit sisään- ja ulostuloineen on esitetty kuvassa 23.



Kuva 23. *Keskusyksikön ohjelmistomoduulit*

Keskusyksikön ohjelmistomoduuleista tehonhallinnan idea on päättää, miten käyttöteho tuotetaan ja miten se jaetaan tehoa kuluttaville toimilaitteille, kuten luvussa 5.1.1 on esitetty. Jotta ohjausalgoritmi kykenee tuottamaan jokaisen voimalinjan laitteen ohjauksen, tarvitsee ohjausalgoritmi ideologisesti sisääntuloksi vähintään:

- halutut toimilaitteiden ohjaukset, kuten esimerkiksi ajomoottorien pyörimisnopeudet,
- jokaisen tehoa käyttävän komponentin ottaman hetkellisen tehon,
- akuston varaustason tai jännitteen ja sen hetkelliset maksimipurku- ja latausvirrat,
- polttomoottorilta otetun hetkellisen momentin ja sen pyörimisnopeuden sekä
- generaattorin tuottaman hetkellisen tehon.

Demonstraatiokoneen tapauksessa tuoton ja kulutuksen vertailun voi tehdä suoraan voimalinjan komponenttien tuottamien ja ottamien virtojen avulla, sillä järjestelmän kaikki invertterit raportoivat niiden läpi kulkevan virran. Lisäksi inverttereille voi asettaa dynaamisesti CAN-viestien avulla virran maksimi- ja minimiarvon, joiden avulla invertteri rajoittaa itse ottamaansa tehoa. Tämä mahdollistaa tehonhallinnan toteuttamisen ohjauksen referenssistä riippumattomana. Kuten luvussa 5.1.1. on esitetty, tehonhallinta-algoritmissa tulee huomioida myös dieselmoottorin käyttö sen eliniän ja aiheuttamien päästöjen optimoimiseksi.

Ohjauksen referenssi on tarkoitus muodostaa ajovoimansiirron osalta ajovoimansiirron hallinnan avulla. Ideana on ottaa sisääntulona kaasupolkimen asento ja antaa ulostulona kummallekin ajovoimansiirron sähkömoottorille referenssi pyörimisnopeudesta tai vääntömomentista, mutta myös määrittää ajovoimansiirron moottoreille vääntömomentin minimi- ja maksimiarvot. Ohjauksen määrittämisessä tulee huomioida nelivedon tuomat haasteet. Koska ajovoimansiirron akseleita ei ole mekaanisesti kytketty toisiinsa tulee akseleiden synkronointi toteuttaa ohjelmallisesti. On myös huolehdittava, ettei pitoa menetetä hallitsemattomasti rajoittamalla maksimimomenttia tarpeen mukaan. Jotta lisäksi regeneroinnin hyöty voidaan maksimoida, tulee jarrutuksen aikana moottorien tuottamaa negatiivista momenttia kyetä säätämään dynaamisesti. Tarkoituksena on muodostaa tilanteeseen sopiva jarruttava momentti säilyttäen kuitenkin pito renkailla. Optimaalisimmassa tapauksessa jarruttaminen tapahtuisi suoraan sähkömoottoreilla eikä työkoneen mekaanisia jarruja tarvitsisi käyttää muussa kuin hätätilanteessa.

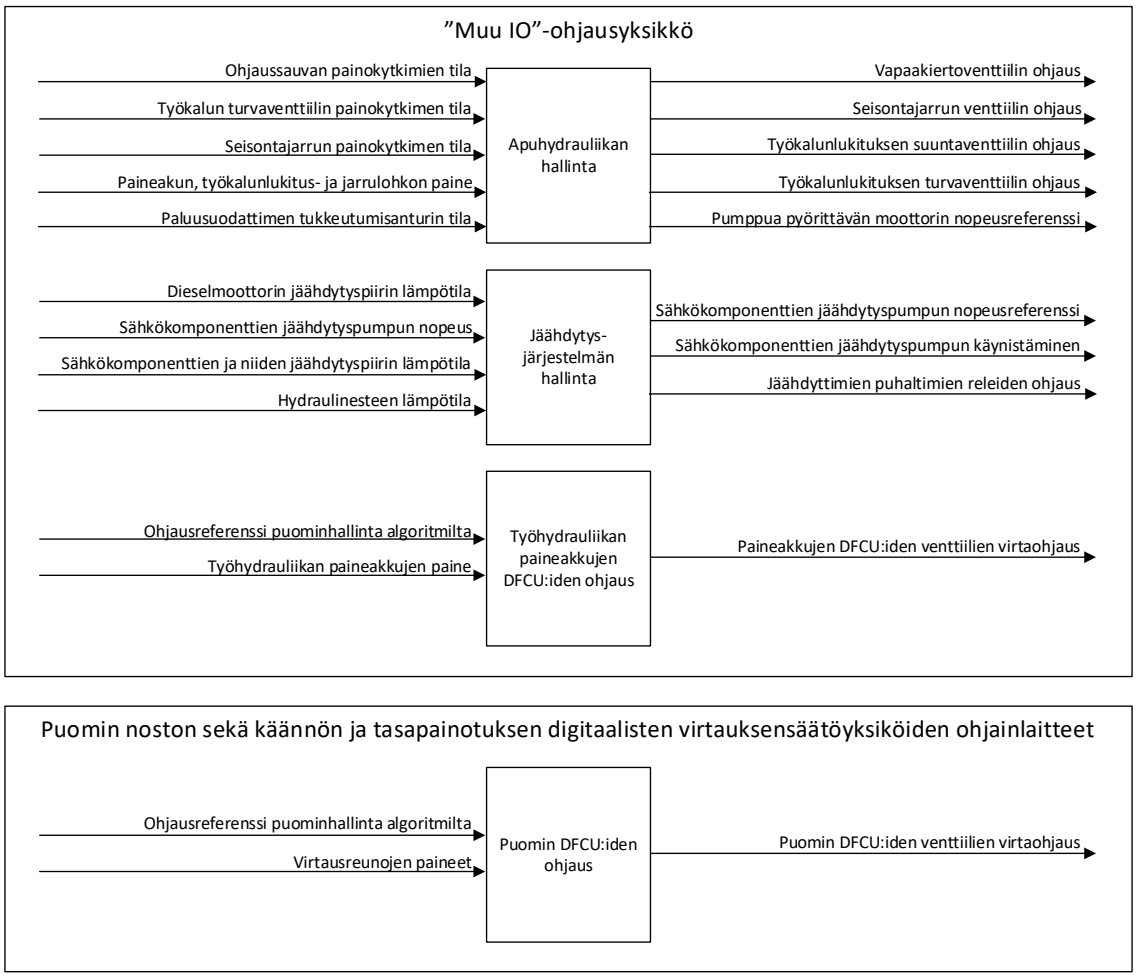
Puomin hallinnassa ideana on liikuttaa puomistoa pohjautuen ohjaussauvalta saatuun referenssiin. Tätä varten ulostulona tarvitaan ohjaukset työhydrauliikan piirin pumpulle

sekä hydraulisylintereitä liikuttaville ja paineakkujen varausta sääteleville digitaalisten virtauksensäätöyksiköiden ohjainlaitteille. Yksinkertaisten implementointien, kuten vakiopainejärjestelmien, kohdalla hydraulipiirin paineenhallinta ja puomiston liikkeen ohjaukset voitaisiin helposti erottaa toisistaan. Tämä ei kuitenkaan ole lähtökohtaisesti mielekästä uudempien ohjausalgoritmien kohdalla, jossa hydraulijärjestelmää pyritään säätämään kokonaisuutena. Ohjausalgoritmin sisäänmenona on saatavilla puomin asema ja eri linjojen painetasot sekä pumpun pyörimisnopeus ohjaussauvan akselien aseman lisäksi.

Kontaktorien ja muiden järjestelmän komponenttien tilan määrittämiseksi tarvitaan tilakone. Tilakoneen ideana on raportoida järjestelmän toiminnan tilasta sekä koordinoida kaikkien korkeajännitekomponenttien käynnistyssekvenssit ja huolehtia niiden olevan valmiina korkeajännitteen kytkemiseksi ennen kontaktorien sulkemista. Tilakoneen oleellisiksi tiloiksi löydettiin kaksi eri ajomoodia – dieselillä ja ilman – sekä vaihtovirtalatauksen mahdollistava tila. Ajomoodi dieselmoottorin kanssa mahdollistaa luonnollisesti hybridivoimalinjaan liittyvän tutkimuksen tekemisen ja ajomoodi ilman dieseliä puolestaan täyssähköisiin työkoneisiin liittyvän tutkimuksen tekemisen.

Tilakoneen toiminnassa tulee kuitenkin huomioida tilasta toiseen siirtymisen ehdot. Esimerkiksi demonstraatiotyökoneen tapauksessa latauskontaktori, jonka takana järjestelmän vaihtovirtalaturi on, ei saa olla kiinni, jos dieselgeneraattorilla tuotetaan järjestelmään tehoa. Lähtökohtaisesti kontaktorien eliniän maksimiomiseksi niiden avaamista tulee myös välttää, jos niiden läpi kulkee merkittävästi virtaa. Yksinkertaisuuden vuoksi tilakoneen tilan vaihtaminen onkin järkevää sallia vain tilanteessa, jossa demonstraatiokone on pysähdyksissä ja dieselmoottori ei ole käynnissä. Tilan vaihtamiseksi tilakone tarvitsee sisääntuloiksi näin ollen järjestelmältä halutun tilan ja komponenttien toimintatilojen lisäksi diesel- ja sähkömoottorien pyörimisnopeuden sekä kontaktorien läpi kulkevan virran.

Kuten fyysisen arkkitehtuurin määrittämisen yhteydessä esitettiin, päätettiin digitaalisten virtauksensäätöyksiköiden venttiilikohtainen ohjaus eriyttää puomin hallinnasta ja ohjata digitaalista virtauksensäätöyksikköä proportionaaliventtiilin tavoin. Puomin noston ja käännön digitaalisten virtauksensäätöyksiköiden ohjaamiseen on omat ohjainlaitteet, mutta paineakkujen digitaalisia virtauksensäätöyksiköitä ohjataan Muu IO -ohjausyksiköllä. Muu IO -ohjausyksiköllä on tarkoitus toteuttaa myös apuhydrauliikan sekä jäähdytysjärjestelmän hallinnat. Näiden ohjainlaitteiden ohjelmistomoduulit ja niiden sisään- ja ulostulot on esitetty kuvassa 24.



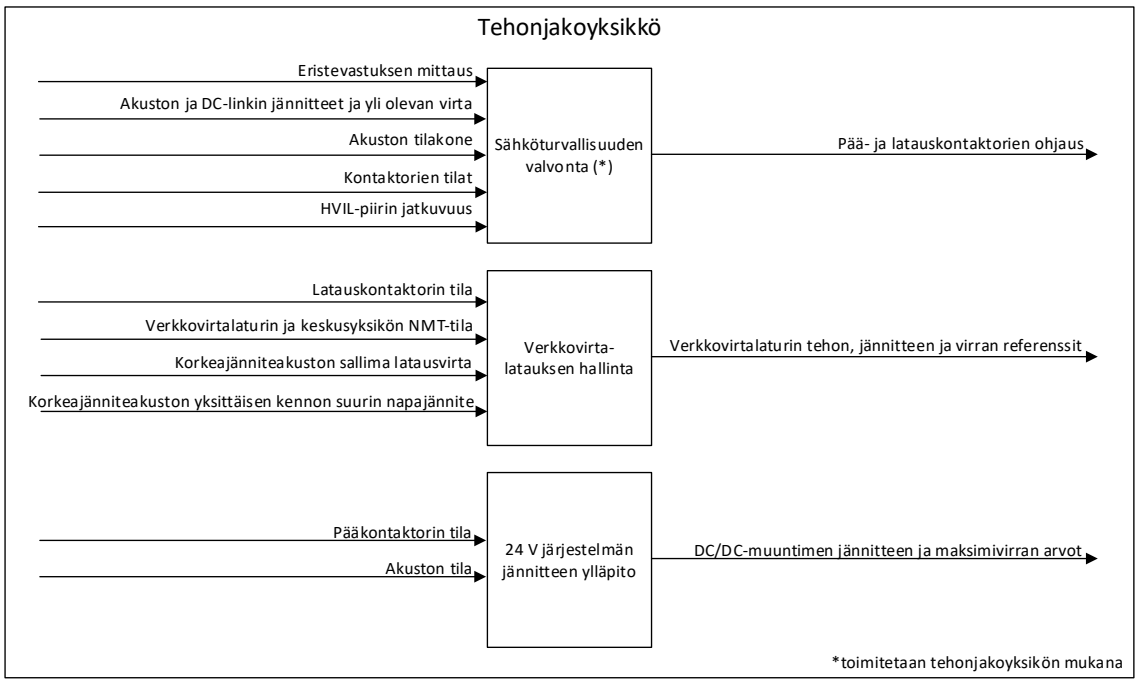
Kuva 24. *Puomiston digitaalisten virtauksensäätöyksiköiden ohjausyksiköiden sekä Muu IO -ohjausyksikön ohjelmistomoduulit*

Muu IO -ohjausyksikön ohjelmistomoduuleista apuhydrauliikan hallinnassa on tarkoitus muodostaa pyörimisnopeuden referenssi piirin hydraulipumpulle, jolla ylläpidetään ohjauksentehostuksen ja jarrujen käyttöpainetta perustuen piirin syöttöpaineen mittaukseen. Tarkoituksena on lisäksi muodostaa piiristä löytyvien seisontajarrun vapautukseen ja työkalun vaihtoon liittyvien venttiilien ohjaukset. Näistä venttiilien ohjaukset voitaisiin eriyttää syöttöpaineen hallinnasta. Apuhydrauliikkaa päätettiin kuitenkin käsitellä yhtenä loogisena kokonaisuutena, sillä niiden mahdollisesta hajauttamisesta ei koettu olevan hyötyä modulaarisuuden lisäämiseksi.

Jäähdytysjärjestelmän hallinnassa ideana on puolestaan ylläpitää korkeajännitekomponenttien lämpötilaa niille määritetyn 65 °C maksimikäyttölämpötilan alapuolella. Jäähdytysjärjestelmä on toteutettu nestekierrolla ja siihen on kytketty ajovoimansiirron vaihdelaatikot ja kaikki korkeajännitekomponentit pois lukien vaihtovirtalaturi ja akusto. Ohjauksen kannalta jäähdytyspiiristä löytyy muuttuva kiertonopeuksinen pumppu, nesteen lämpötila-anturi ja jäähdyttimen puhallin.

Jäähdytysjärjestelmän hallintaan sisällytettiin lisäksi dieselmoottorin jäähdyttimen ja hydraulijärjestelmän jäähdyttimien ohjaus, sillä niiden hajauttamisen ei koettu tuovan hyötyä modulaarisuuden lisäämisessä. Ohjaamisen kannalta dieselmoottorin jäähdytyspiirin puhallinta voidaan ohjata dieselin CAN-väylän avulla raportoiman jäähdytysnesteen lämpötilan avulla ja hydraulijärjestelmän jäähdyttimen puhallinta nestesäiliön pinnankorkeusanturin, joka raportoi myös lämpötilan, avulla. Dieselin jäähdytysjärjestelmän puhallinta tulee ohjata siten, ettei nesteen lämpötila ylitä 110 °C ja hydraulijärjestelmän jäähdytysjärjestelmän puhallinta siten, ettei hydraulinesteen lämpötila ylitä 60 °C.

Koska tehonjakoyksikkö ostettiin demonstraatiokoneeseen räätälöitynä, onnistui sen sisäiseen ohjelmoitavaan logiikkaan sisällyttämään omaa ohjauskoodia, kuten aikaisemmin mainittiin. Ohjelmistomoduuleista sähköturvallisuuden valvonta toimitettiin tuotteen mukana, mutta verkkovirtalatauksenhallinta toteutettiin itse. Koska 24 V-järjestelmän jännitteen ylläpito eli DC/DC-muuntimen ohjaus tulee olla aktiivinen aina, kun demonstraatiotyökonetta ladataan tai käytetään, huomattiin, että yksinkertaisesti DC/DC-muuntimen ohjaus voidaan toteuttaa perustuen pääkontaktorin tilaan, jolloin ohjausta ei tarvitse sisällyttää tehonhallintaa ja verkkovirtalatauksen hallintaan erikseen. Ohjaus voidaan toteuttaa yksinkertaisesti asettamalla DC/DC-muuntimen ulostulojännite 28,8 V aina, kun pääkontaktori on kiinni. Tehonjakoyksikön ohjelmistomoduulit on esitetty kuvassa 25. Lisäksi tehonjakoyksiköllä on tarkoitus reitittää akkuväylältä isäntänä toimivan korkeajänniteakuston moduulin kokoama raportti koko akuston toimintatilasta yleisväylään.



Kuva 25. *Tehonjakoyksikön ohjelmistomoduulit*

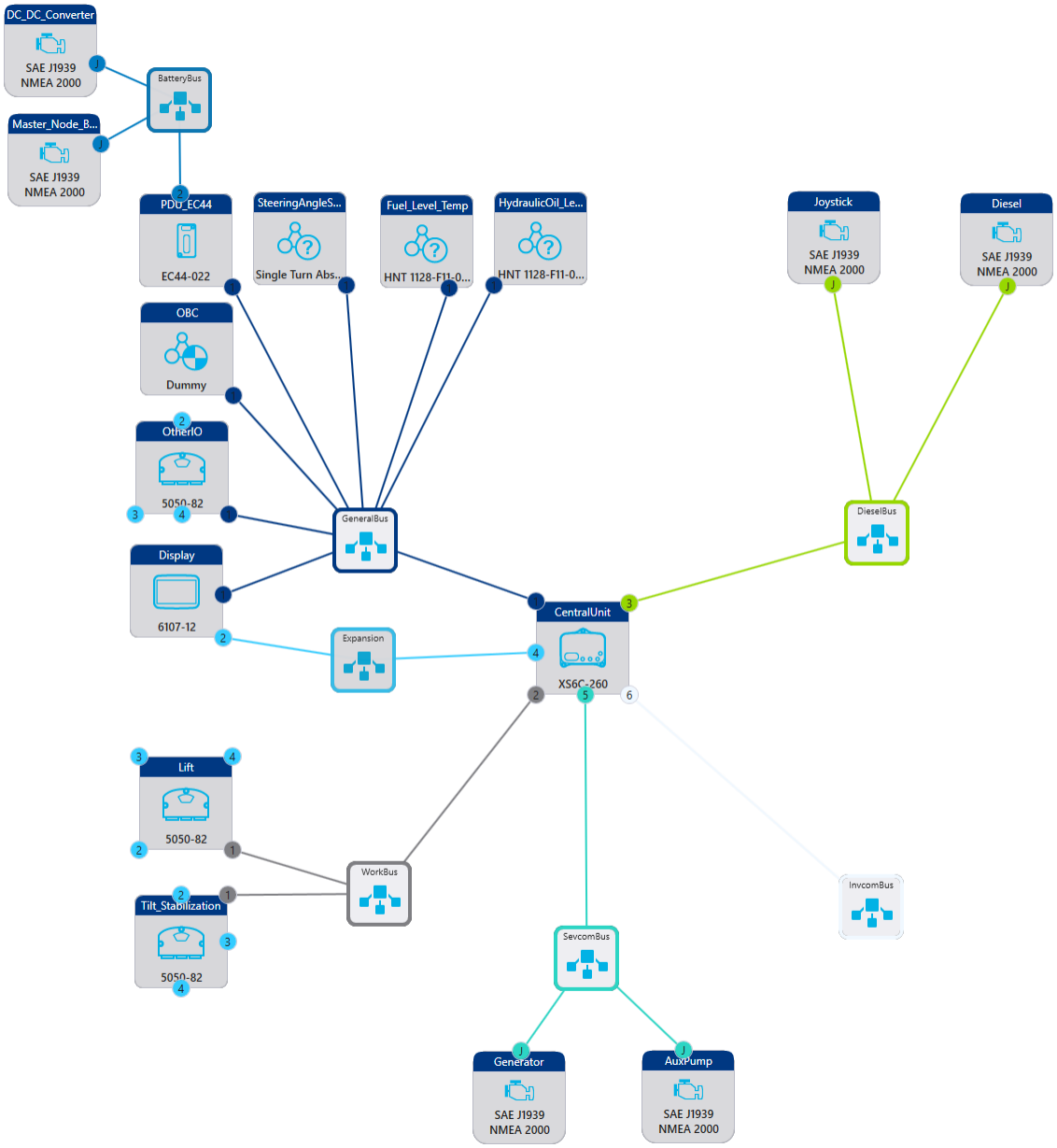
Sähköturvallisuuden valvonta on oleellinen osa työkonetta aina, kun järjestelmään lisätään korkeajännitekomponentteja ja vikatilanteessa helposti syttyvää akkuteknologiaa. Sähköturvallisuuden valvonnan tarkoitus on seurata aktiivisesti järjestelmän eristevastusta, korkeajännitekomponenttien liittimien paikalla olosta kertovan HVIL-piirin (engl. high voltage interlock loop, HVIL) jatkuvuutta sekä korkeajänniteakuston jännitetasoa ja kuormitusta. Mikäli toiminnassa huomataan puutteita, korkeajänniteakusto irrotetaan muusta järjestelmästä automaattisesti avaamalla kontaktorit. Kontaktorit, korkeajännitekomponenttikohtaiset sulakkeet, eristevastus- ja HVIL-valvonta on sisäänrakennettu tehonjakoyksikköön. Tehonjakoyksikkö hoitaa myös korkeajännitekomponenttien esilatauksen automaattisesti kontaktorien sulkemisen yhteydessä.

Verkkovirtalatauksen hallinnassa on ideana puolestaan ohjata vaihtovirtalaturia perustuen korkeajänniteakuston varaustasoon. Koska demonstraatiokoneen tapauksessa akusto ilmoittaa suoraan latausvirran maksimiarvon, voidaan se syöttää vaihtovirtalaturin virran referenssiksi. Vaihtovirtalaturin jännitereferenssiksi voidaan puolestaan asettaa yksinkertaisesti hieman täyden korkeajänniteakuston napajännitettä korkeampi jännite eli noin 410 V. Latauksen aikana tulee kuitenkin seurata yksittäisten kennojen jännitettä ja lataus tulee lopettaa, mikäli yksikään kenno saavuttaa 4,15 V napajännitteen. Kehittyneemmän ohjausalgoritmin käyttämisestä ei ole demonstraatiotyökoneen tapauksessa hyötyä, koska vaihtovirtalaturin kykenee maksimissaan 10 A latausvirtaan korkeajänniteakuston salliman 160 A latausvirran sijaan. Algoritmi tarvitsee sisääntulona korkeajänniteakuston kennojen suurimman napajännitteen ja maksimilatausvirran, mutta myös latauskontaktorin tilan ja verkkovirtalaturin ja keskusyksikön NMT-tilat.

NMT-tilat tarvitaan, koska niillä tunnistetaan toiminnan tila. Kun demonstraatiokone viedään verkkovirtalataukseen, aktivoi vaihtovirtalaturin käynnistäminen relelogiikan, joka käynnistää tehonjakoyksikön, DC/DC-muuntimen ja akuston hallintayksikön, mutta ei keskusyksikköä. Tällöin tehonjakoyksikkö voi itse sulkea kontaktorit ja käynnistää latauksen. Mikäli demonstraatiokoneessa päävirtakytkin on käännetty päälle, on koko järjestelmä päällä mukaan lukien keskusyksikkö. Tällöin kontaktorien sulkemisesta vastaa keskusyksikön tilakone. Tilanne, jossa koko järjestelmän halutaan olevan päällä verkkovirtalatauksen aikana, on tyypillinen tutkimusalustoille, kun niiden ohjausjärjestelmän ohjelmistoa halutaan muokata.

5.3 Kommunikointirajapinnat

Kuten aikaisemmin mainittiin, demonstraatiokoneeseen valittujen ohjainlaitteiden kommunikaatioon liittyvä määrittely onnistuu kootusti Multitool-ohjelmalla tuettujen tiedonsiirtoprotokollien avulla. Tässä työssä EPECin valmistamat ohjainlaitteet tuotiin projektiin sovelluksen alasvetovalikosta, CANopen-laitteet niiden mukana toimitettujen EDS-tiedostojen avulla ja J1939-laitteille luotiin J1939-objektit. Tämän jälkeen laitteet yhdistettiin kuvassa 26 esitetyn arkkitehtuurin mukaisesti ja väylien baudinopeudet valittiin.



Kuva 26. *Demonstraatiokoneen Multitool-projekti*

Ennen ohjainlaitekohtaisten Codesys-projektien luontia ohjainlaitteiden IO konfiguroitiin ja niiden vastaanottamat ja lähettämät viestit määritettiin. CANopen-laitteiden kohdalla riitti EDS-tiedoston mukana tulleiden RPDO-viestien lähettäjien ja TPDO-viestien vastaanottajien määrittely. J1939-laitteiden kohdalla niiden lähettämät ja vastaanottamat viestit jouduttiin määrittämään manuaalisesti perustuen komponenttien mukana tulleisiin manuaaleihin. Komponenttien käyttämistä standardoiduista parametriryhmistä suurin osa löytyi valmiiksi Multitool-ohjelmasta hakemalla parametriryhmän numerolla. Ohjelmoitavien logiikoiden välinen tiedonsiirto toteutettiin puolestaan CANopen-standardin avulla.

Näiden muutoksien jälkeen ohjainlaitteille luotiin Codesys-projektit Multitool-ohjelman avulla. Multitool muodostaa ohjainlaitteille Codesys-kehitysympäristöön koodipohjan käyttöliittymän asetuksien pohjalta. Se esimerkiksi parametrisoi automaattisesti globaaleihin muuttujiin viesteistä löytyvät parametrit ja ohjainlaitteesta mahdollisesti löytyvän IO:n. Käyttäjän ei ole tarkoitus muokata koodipohjaan generoitua koodia, vaan tehdä tarvittaessa muutokset käyttäen Multitool-ohjelmaa ja generoida sen avulla script-tiedosto, joka voidaan ajaa Codesys-ympäristössä. Koodipohja alustaa myös käyttäjälle Main- ja MainInit-ohjelmat, joiden alle voidaan kirjoittaa oma ohjelma, mikäli se halutaan ajaa koodipohjan kanssa samassa tehtävässä. Tätä ohjelmaa päivittämiseen käytetty script-tiedosto ei ylikirjoita sen ajamisen yhteydessä.

Demonstraatiotyökoneen kommunikaatiorajapintojen osalta Codesys-ympäristössä tehtäväksi jäi CANopen- ja J1939-laitteiden viesteistä löytyvien parametrien tarpeen mukaan skaalaaminen, offset-arvojen lisääminen ja datatyyppien muuttaminen. Tätä varten jokaiselle laitteelle luotiin oma kommunikointirajapintaan liittyvä ohjelma. Esimerkiksi J1939-standardissa polttomoottorin pyörimisnopeuspyyntö, jonka SPN on 898, lähetetään väylälle positiivisella 16 bittisellä luvulla, jossa yksi bitti merkitsee 0,125 RPM (J1939-DA 2023). Haluttaessa lähettää moottorille 2000 rpm pyyntö tulee väylälle lähettää arvo 16 000. Toisaalta sallittu vääntö prosentteina maksimista, jonka SPN on 518, lähetetään väylälle positiivisella 8 bittisellä luvulla, jossa offset-arvo on 125 % (J1939-DA 2023). Täten haluttaessa sallia moottorin tuottavan maksivääntö eli 100 % tulee väylälle lähettää arvo 225. Huomioitavaa on, että skaalauskertoimen jakaminen tai kertominen ja offset-arvon lisääminen tai vähentäminen riippuu siitä, ollaanko lähettämässä vai vastaanottamassa parametria. Tyypillisesti väylälle lähtevä parametri pyrkii hyödyntämään sen datatyypin arvoalueen kokonaan, minkä avulla laskutoimituksen suunta voidaan loogisesti päätellä.

Siitä huolimatta, että CAN-viestikehyksessä on itsessään tarkastuslaskenta sisäänrakennettuna, on joidenkin laitteiden lähettämien ja vastaanottamien viestien datakenttään lisätty erillinen tarkastuslaskentaparametri. Viestikehykseen sisäänrakennetun tarkastuslaskennan ja siihen liittyvät standardin mukaiset toiminnot tapahtuvat automaattisesti, mutta datakentästä löytyvä ylimääräinen tarkastuslaskenta jää käyttäjän toteutettavaksi Codesys-kehitysympäristössä. Tarkastusalgoritmi riippuu täysin laitevalmistajasta ja demonstraatiotyökoneen CAN-komponenttien kommunikaatiorajapintojen luomiseen jouduttiin käyttämään muutamia erilaisia algoritmeja.

Mikäli viestissä oli ylimääräinen tarkastuslaskentaparametri, oli siinä myös laskuri, jonka arvo muuttui yhdellä jokaiseen uuteen viestiin. Mikäli tarkastuslaskenta tai laskurin arvo on väärin, ei laite vastaanota kyseistä viestiä ja menee vikatilaan, mikäli tietyssä aikamääreessä ei ole tullut yhtään hyväksyttyä viestiä. Tämän vuoksi ylimääräisen tarkastuslaskennan toteuttaminen oli välttämätöntä lähetettäviin viesteihin. Tarkastuslaskenta ja laskurin arvon tarkastaminen toteutettiin kuitenkin myös vastaanotettaviin viesteihin, sillä prosessissa pystyttiin hyödyntämään lähetettävien viestien tarkastuslaskentaan tehtyjä funktioita. Mahdollinen tarkastuslaskenta toteutettiin laitekohtaisiin ohjelmiin, kuten parametrien formatointikin. Mikäli laite käytti ylimääräistä tarkastuslaskentaa luotiin laitekohtaisiin ohjelmiin ChecksumError -parametri, joka on totta, mikäli yhdessäkään laitteelta vastaanotetussa viestissä esiintyy virhe.

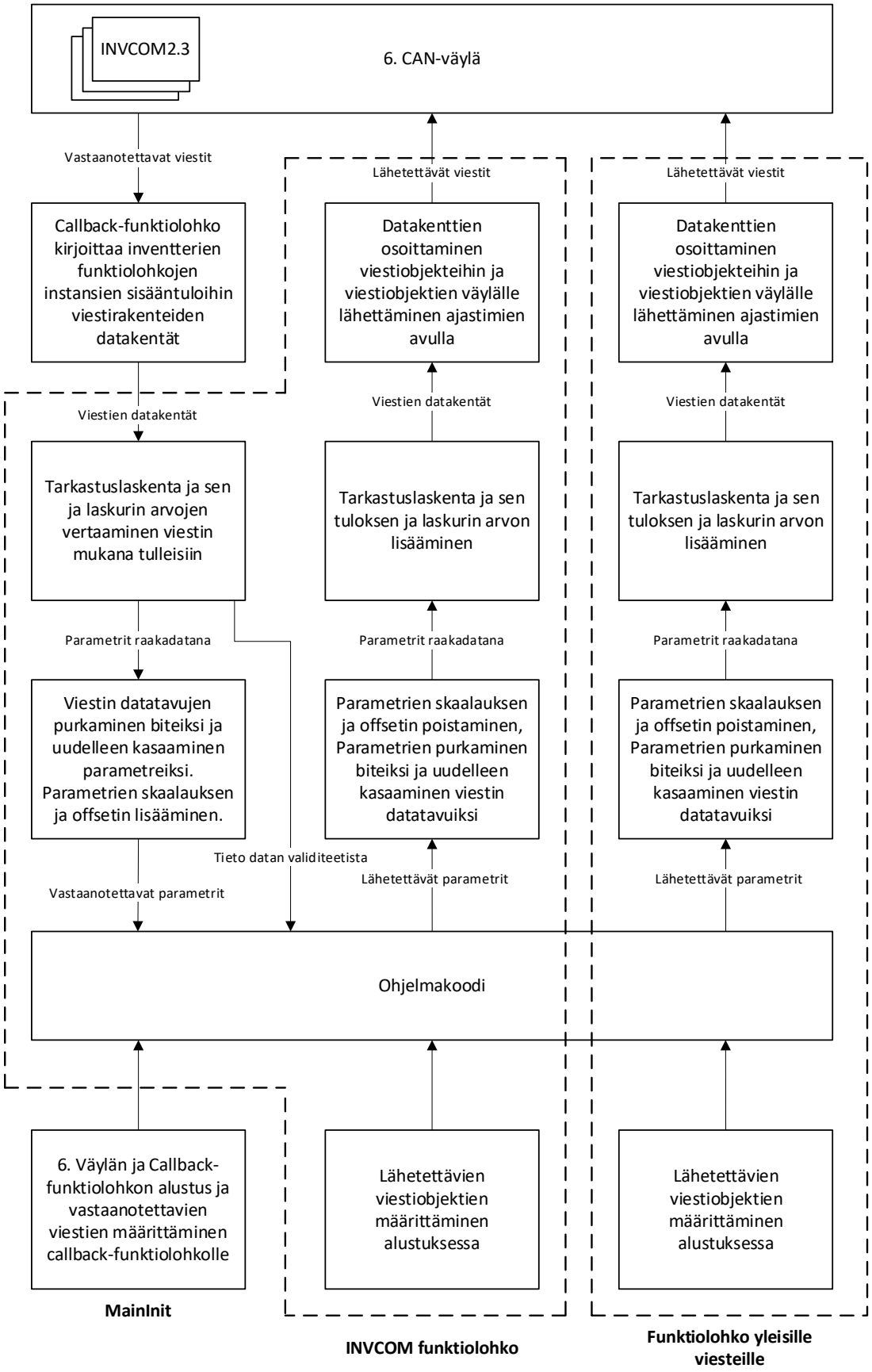
Viimeisenä laitteiden kommunikaatiorajapintoihin liittyvänä tehtävänä oli INVCOM2.3-invertterien rajapinnan muodostaminen keskusyksikön ohjelmaan. Koska invertterit eivät käytä kommunikaatioon mitään standardisoitua ylemmän tason standardia, jouduttiin rajapinta rakentamaan alusta asti. Prosessi alkoi CAN-väylän manuaalisella alustamisella MainInit-ohjelmaan, jonne koodipohjan luoma kuudes väyläobjekti alustettiin käyttäen sen alustusmetodia. MainInit-ohjelmassa väyläobjektille lisättiin myös vastaanotettavat viestit viestien lisäysmetodilla. Jotta viestien datakentät saadaan osoitettua ja päivitettyä jokaisella laskentakierroksella inverttereille implementoitiin viestienvastaanottoa varten Callback-funktiolohko, jonka alta löytyvällä vastaanottometodilla voidaan osoittaa inverttereille niiden vastaanottamat viestit.

Yhden invertterin rajapinnan rakentaminen päätettiin toteuttaa funktiolohkona, josta alustettiin omat instanssit jokaiselle kolmelle INVCOM2.3-invertterille koodipohjan Main-ohjelmaan. Invertterien funktiolohkolle luotiin raakadatarakenteet, joihin aikaisemmin mainitun Callback-funktiolohkon avulla haetut viestirakenteet voidaan kirjoittaa. Koska jokaisella invertterille on identtinen viestiliikenne, tehtiin funktiolohkon alle jokaiselle viestityypille oma metodi. Jokainen metodi ottaa sisään viestirakenteen datakentät, jonka jälkeen se hajottaa ne biteiksi sen sisäiseen rakenteeseen, yhdistää tietyt bitit raakaparametriksi

sekä lisää mahdollisen skaalauksen ja offset-arvon ja antaa ulostulona parametrit sopivalla datatyypillä.

Koska jokaisessa viestissä löytyy lisäksi tarkastuslaskenta ja laskuri, toteutettiin funktiolohko, joka tarkistaa viestin tarkastussumman ja laskurin arvon oikeellisuuden sekä aiheuttaa vian, mikäli oikeellisen viestin vastaanottamisesta on liian kauan aikaa. Tämän jälkeen invertterin funktiolohkoon luotiin tarkastuslaskentafunktiolohkosta instanssit jokaiselle viestityypille ja luotiin yksinkertainen ja helppokäyttöinen totuusarvoparametri kertomaan vastaanotettavien viestien oikeellisuudesta. Mikäli parametrin arvo on totta, voidaan vastaanotettavaa data pitää luotettavana ja mikäli parametrin arvo on epätosi, on vähintään yhdessä viestissä jokin siirtovirhe.

Samalla ideologialla, mutta vastakkaiseen suuntaan, toteutettiin lähetettävien viestien rakentaminen. Toisin sanoen viestikohtaiset metodit purkavat parametrit ja juoksevan laskurin arvon viestin raakadatatavuiksi, jonka jälkeen siitä lasketaan tarkastussumma. Tämän jälkeen datatavut kirjoitetaan viestirakenteisiin. Viestirakenteet puolestaan luotiin invertterin funktiolohkon alle luodulla lähtevien viestien alustustoiminolla, joka ajetaan vain käynnistämisen yhteydessä. Lopuksi viestirakenteet lähetetään väylälle kohdistamalla viestit väyläobjektille sen viestinlähetysmetodin avulla käyttämällä ajastimia. Invertterikohtaisen prosessidatan lähettämisen lisäksi kaikki väylällä olevat invertterit vastaanottavat muutaman yhteisen viestin, jotka luotiin samalla ideologialla kuin invertterikohtaiset, mutta erillisiin ohjelmiin. Tiivistettynä INVCOM2.3-inverttereille luotu rajapinta on esitetty kuvassa 27.



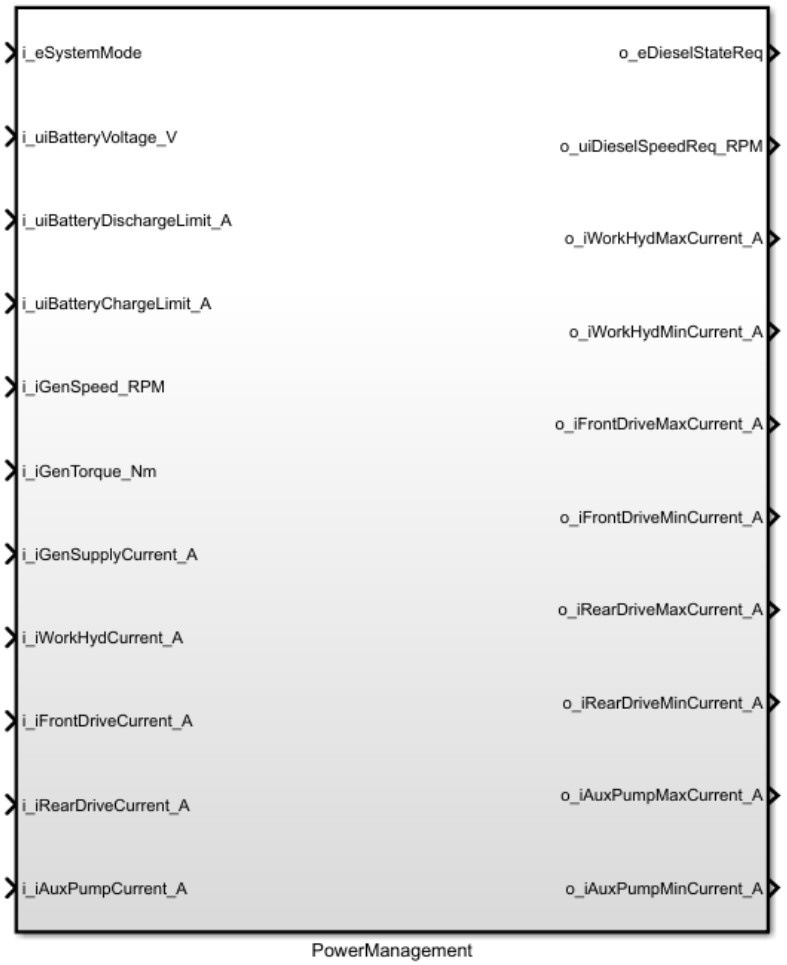
Kuva 27. *INVCOM2.3-invertterien kommunikaatiorajapinta*

Rajapintojen luomisen yhteydessä käytettiin SIL-menetelmää virheiden etsimiseksi. Tähän sisältyi ohjelman kääntämistä ja siihen liittyvien virheiden korjaamista, mutta myös simulointia. Codesys-ympäristö mahdollistaa koodin ajamisen virtuaalisesti, jolloin toiminnallisia virheitä löydettiin jo ohjelmointivaiheessa ennen ohjelmiston viemistä ohjainlaitteille. Tämän jälkeen ohjelmistoja testattiin HIL-menetelmällä siirtämällä ohjelmat ohjainlaitteille ja kytkemällä ne vuorollaan CAN-väylään, johon myös tietokone oli kytkettynä CAN-adapterin avulla. Testausmenetelmässä ohjainlaitteiden lähettävien viestien oikeellisuutta ja vastaanotettavien viestin oikeanlaista käsittelyä tutkittiin tietokoneella sekä korjattiin löytyneitä virheitä.

Tähän mennessä tehty testaaminen pystyttiin tekemään toimistossa, mutta seuraavassa vaiheessa siirryttiin laboratorioon tekemään IIL-testausmenetelmään. Toisin sanoen vuoron perään komponenttien viestintärajapintaa testattiin lähettämällä ohjainlaitteilla komentoja komponenteille ja vastaanottamalla komponenteilta informaatiota. Aluksi kuitenkin suurin osa komponenteista vaati kommunikaatioon liittyvien asetuksien, kuten väylänopeuden ja solmunumeron muuttamisen, jotta se toimii myöhemmin demonstraatiokoneessa kuvassa 22 esitetyn arkkitehtuurin mukaisesti. IIL-menetelmässä varsinkin tarkastuslaskentaan ja laskureihin liittyviä virheitä tuli esille. Huomioitavaa kuitenkin, on ettei korkeajännitekomponenttien oikeellista toimintaa kyetty testaamaan, sillä työkoneen korkeajännitejärjestelmä ei kerennyt valmistumaan diplomityön tekemisen määräaikana. Prosessin lopulla komponentit eivät kuitenkaan näyttäneet menevän vikatilaan virheellisen kommunikaatiorajapinnan takia.

5.4 Ohjausalgoritmien mallipohjainen kehitys järjestelmällä

Mahdollisuus mallipohjaiseen ohjausalgoritmien kehitykseen Matlab–Simulink-ympäristössä oli yksi ohjausjärjestelmään kohdistuva vaatimus. Koska järjestelmään valitut ohjainlaitteet ohjelmoidaan Codesys-ympäristössä käyttäen IEC 61131–3 mukaista ohjelmointikieltä, jolle kääntämistä Simulink tukee suoraan, on mallin tuominen Codesys-ympäristöön suhteellisen yksinkertaista. Tiivistettynä prosessissa luodaan haluttu toiminnallisuus Simulink-ympäristön alijärjestelmälohkoon, joka käännetään automaattisella koodingeneroinnilla IEC 61131–3 standardin ST-kielellä (engl. structured text language) kirjoitetuksi funktiolohkoksi. Tämän jälkeen funktiolohko voidaan tuoda Codesys-ympäristöön ja siitä voidaan luoda instanssi muuhun ohjelmaan normaalien käytäntöjen mukaan. Ennen kuin alijärjestelmän lohko voidaan tuoda Codesys-ympäristöön, tulee kuitenkin huomioida muutama asia, joita käsitellään seuraavaksi esimerkin avulla. Esimerkissä Codesys-ympäristöön tuodaan Simulink-ympäristöstä kuvassa 28 esitetty demonstraatiokoneen alustava tehonhallinnan algoritmi.



Kuva 28. *Tehonhallinnan lohko Simulink-ympäristössä*

Jotta lohkon sisääntulojen ja ulostulojen käsittely on Codesys-ympäristössä myöhemmin helpompaa, on parametrien nimeämiseen järkevää kiinnittää huomiota. Kuvan 28 esimerkissä parametrien etuliitteeksi on asetettu ”i_” ja ”o_” merkitsemään sisäänmenoja ja ulostuloja ja tämän perään tunniste parametrin datatyypistä Codesys-ympäristön (2023a) dokumentaatiossa esitettyjen suositusten mukaisesti. Lisäksi parametrien perään on asetettu tunniste parametrin SI-yksiköstä. Parametrien yksiköiden lisääminen on mielekästä, sillä muistinkäytön ja laskennan optimoimiseksi PLC-koodin datatyyppeinä pyritään käyttämään kokonaislukuja desimaalilukujen sijasta. Esimerkiksi demonstraatiokoneen tapauksessa samalla datatyypillä käsitellään akuston kennojen jänniteitä millivoltteina, kun taas koko akuston yliolevaa jännitettä voltteina. Tämä voisi johtaa sekaannukseen, jos yksikköä ei olisi ilmoitettu parametrin nimessä.

Sisään- ja ulostulojen datatyypit voidaan asettaa Simulink-ympäristön mallinnusvälilehden alta löytyvästä ”Model explorer”-ikkunasta, kun avautuneen ikkunan rakennepuusta valitaan käännettävä lohko. Vakiona Simulink käyttää datatyyppinä 64 bittistä reaalilukua, joka on usein tarpeettoman suuri ja lisää turhaan logiikan laskennallista kuormaa.

Datatyypiksi kannattaa valita suoraan pienin mahdollinen datatyyppi mikä mahdollistaa informaation siirtämisen ja muuttaa datatyyppiä myöhemmin, mikäli laskenta sitä vaatii. Taulukossa 11 on esitetty yleisimpien Codesys-ympäristössä käytettyjen datatyyppien vastaavuudet Simulink-ympäristössä.

Taulukko 11. *Datatyypit muunnokset Simulink ja Codesys -ympäristöjen välillä (Codesys 2023b & Mathworks 2023)*

Codesys (parametrin suositeltu etuliite)	**Simulink**
BOOL (x)	boolean
SINT (si)	int8
USINT (usi)	uint8
INT (i)	int16
UINT (ui)	uint16
DINT (di)	int32
UDINT (udi)	uint32
REAL (r)	single
LREAL (lr)	double

Kuvassa 29 on asetettu esimerkin käännettävän lohkon sisäänmenojen ja ulostulojen datatyypit. Tarvittaessa parametreille voi asettaa yksiköt, alkuarvot, rajat sekä dimension. Vakiona alkuarvo numeropohjaisille muuttujille on nolla ja totuusarvomuuttujille epätosi. Mikäli sisääntulo tai ulostulo on vektori tai matriisi, tulee sen koko ennalta määrittää, sillä koodingenerointi ei tue kooltaan muuttuvia vektoreita tai matriiseja.

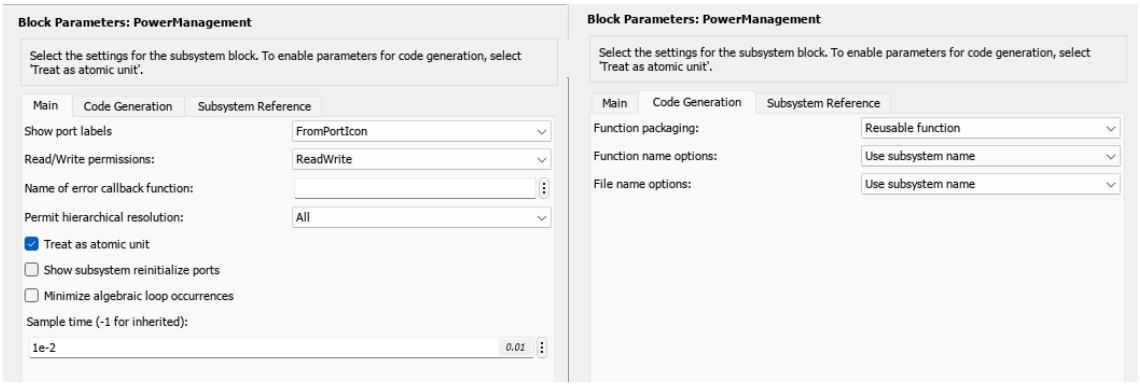
Contents of: PowertrainControl/PowerManagement (only)

Column View: System I/O — Show Details

Name	BlockType	Port	OutDataTypeStr	LockScale	OutMin	OutMax	PortDimensions	Unit	SampleTime	SignalType	IconDisplay	InitialOutput
BlockType: Inport (11)												
i_uiBatteryVoltage_V	Inport	2	uint16	☐	[]	[]	-1	V	-1	real	Port number	
i_uiBatteryDischargeLimit_A	Inport	3	uint16	☐	[]	[]	-1	A	-1	real	Port number	
i_uiBatteryChargeLimit_A	Inport	4	uint16	☐	[]	[]	-1	A	-1	real	Port number	
i_iWorkHydCurrent_A	Inport	8	int16	☐	[]	[]	-1	A	-1	real	Port number	
i_iRearDriveCurrent_A	Inport	10	int16	☐	[]	[]	-1	A	-1	real	Port number	
i_iGenTorque_Nm	Inport	6	int16	☐	[]	[]	-1	N*m	-1	real	Port number	
i_iGenSupplyCurrent_A	Inport	7	int16	☐	[]	[]	-1	A	-1	real	Port number	
i_iGenSpeed_RPM	Inport	5	int16	☐	[]	[]	-1	rev/min	-1	real	Port number	
i_iFrontDriveCurrent_A	Inport	9	int16	☐	[]	[]	-1	A	-1	real	Port number	
i_iAuxPumpCurrent_A	Inport	11	int16	☐	[]	[]	-1	A	-1	real	Port number	
i_eSystemMode	Inport	1	uint8	☐	[]	[]	-1	inherit	-1	real	Port number	
BlockType: Outport (11)												
o_uiDieselSpeedReq_RPM	Outport	2	uint16	☐	[]	[]	-1	rev/min	-1	real	Port number	
o_iWorkHydMinCurrent_A	Outport	5	int16	☐	[]	[]	-1	A	-1	real	Port number	
o_iWorkHydMaxCurrent_A	Outport	4	int16	☐	[]	[]	-1	A	-1	real	Port number	
o_iRearDriveMinCurrent_A	Outport	9	int16	☐	[]	[]	-1	A	-1	real	Port number	
o_iRearDriveMaxCurrent_A	Outport	8	int16	☐	[]	[]	-1	A	-1	real	Port number	
o_iGenTorqueReq_Nm	Outport	3	int16	☐	[]	[]	-1	N*m	-1	real	Port number	
o_iFrontDriveMinCurrent_A	Outport	7	int16	☐	[]	[]	-1	A	-1	real	Port number	
o_iFrontDriveMaxCurrent_A	Outport	6	int16	☐	[]	[]	-1	A	-1	real	Port number	
o_iAuxPumpMinCurrent_A	Outport	11	int16	☐	[]	[]	-1	A	-1	real	Port number	
o_iAuxPumpMaxCurrent_A	Outport	10	int16	☐	[]	[]	-1	A	-1	real	Port number	
o_eDieselStateReq	Outport	1	uint8	☐	[]	[]	-1	inherit	-1	real	Port number	

Kuva 29. *Tehonhallintalohkon sisäänmenot ja ulostulot*

Datatyyppien asettamisen jälkeen ikkunan oikeassa reunassa näkyvät lohkon asetukset, jotka on myös esitetty kuvassa 30. Asetuksista tulee valita ”Treat as atomic unit” -vaihtoehto, joka määrittää lohkoa käsiteltävän pienimpänä laskettavana kokonaisuutena. Ilman tätä valintaa Simulink suorittaa mallin laskemisen optimoidussa järjestyksessä, ja lohkoja käytetään vain visualisointiin. Koodin generointi ei myöskään onnistu ilman asetuksen valintaa. Mikäli tekeillä olevan ohjainta halutaan testata simulaatioympäristössä, voidaan lohkolle asettaa näytteistysaika ”Treat as atomic unit” -vaihtoehdon valitsemisen jälkeen. Tämä mahdollistaa lohkon toimivan aina kiinteällä näytteistysvälillä riippumatta Simulinkin ratkaisijasta, ja näin ollen se simuloi todellisen PLC-logiikan toimintaa. Vaihtoehtoisesti Simulinkin käyttämä ratkaisija voidaan asettaa käyttämään kiinteää laskentaväliä. Jos kehitteillä olevaa ohjainta halutaan testata Simulink-ympäristössä osana simuloitua järjestelmää, tämä ei kuitenkaan usein ole toivottua, sillä laskentaväliltään muuttuvat ratkaisijat lyhentävät merkittävästi tarvittua laskenta-aikaa.

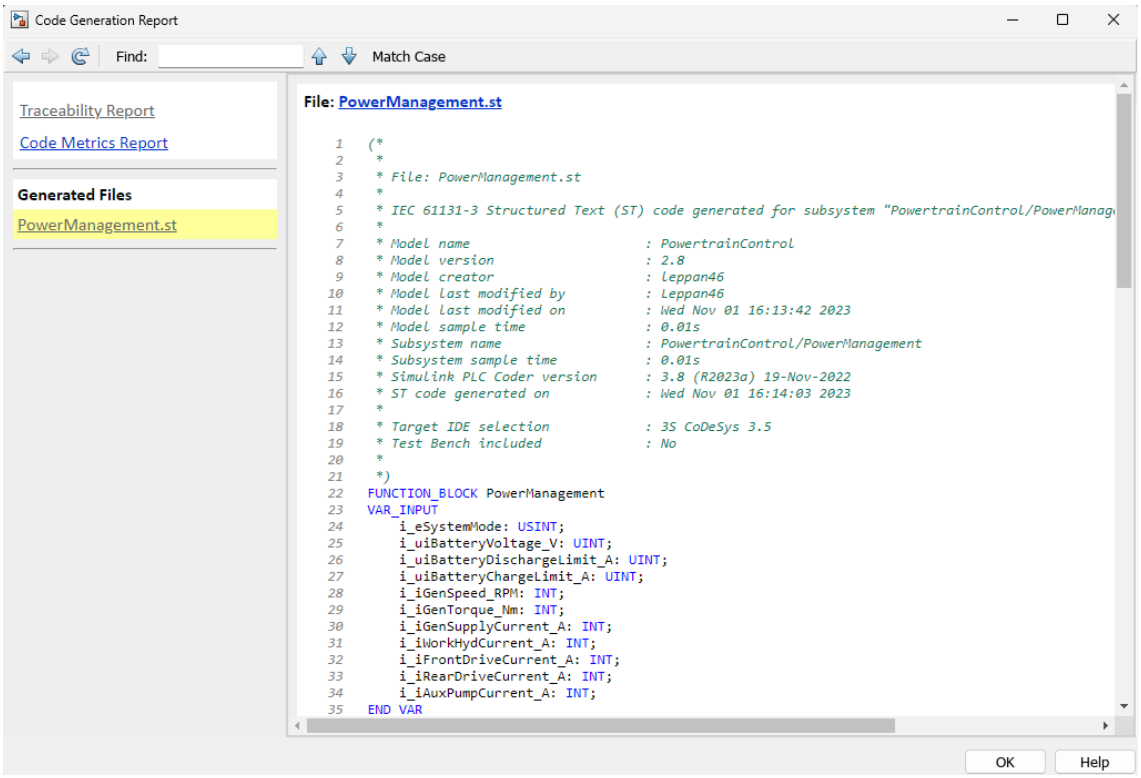


Kuva 30. *Simulink-lohkon asetukset koodingenerointia varten*

”Treat as atomic unit” -vaihtoehdon valitsemisen jälkeen myös koodin generointiin liittyvät asetukset avautuvat. Mikäli käännettävässä koodissa on uudelleen käytettävää koodia, kuten useampi instanssi samasta alijärjestelmästä, voidaan funktion pakkauksen ”reusable function” -valinnalla mahdollistaa näiden alijärjestelmien kääntäminen omiksi funktiolohkoikseen Codesys-ympäristöön. Tällöin useassa paikassa käytetty funktiolohko alustetaan omiksi instansseiksi pääfunktiolohkossa eikä pääfunktiolohkoon tule molemmille alijärjestelmille omaa ohjelmakoodin osaa. Lohkon parametreihin liittyen käännettävän lohkon ja sen tiedoston nimeämiskäytännöiksi suositellaan valittavan käännettävän lohkon nimi alijärjestelmän nimivalinnalla, jolloin tiedoston ja sen sisältävien funktiolohkojen nimet noudattavat generoitavaa lohkoa eikä Simulink-mallin nimeä, kuten oletuksena. Asetuksesta on hyötyä varsinkin, jos Simulink-malli pitää sisällään useampia erikseen käännettäviä lohkoja.

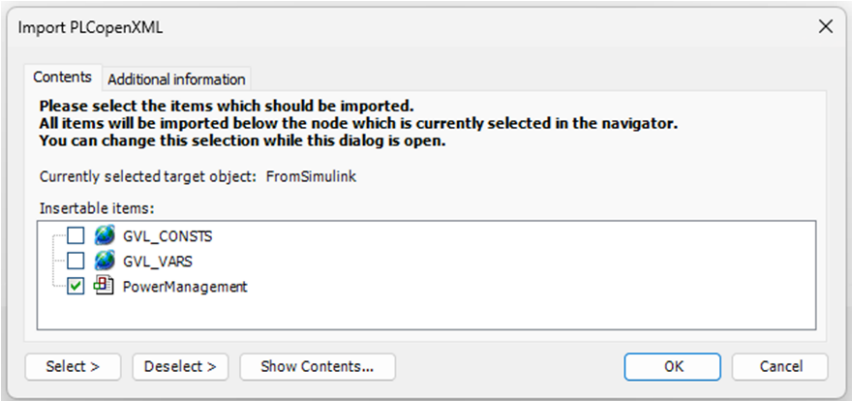
Lohkokohtaisten asetuksen lisäksi PLC-koodin generointiasetuksiin tulee valita oikea ohjelmointiympäristö (engl. integrated development environment, IDE). Tämä onnistuu Simulink-ympäristön mallinnusvälilehden alta löytyvistä mallin asetuksista. Avautuneen mallin asetuksista löytyy välilehti ”PLC Code Generation”, josta kehitysympäristön voi valita. Valittujen ohjainlaitteiden tapauksessa valitaan ”3S CoDeSys 3.5”. Mikäli kehitysympäristö on asennettu oletuspolkuun polkua ei tarvitse vaihtaa. Muussa tapauksessa polku pitää valita manuaalisesti. Samasta välilehdestä voidaan valita myös kansion nimi, johon automaattinen koodin generointi muodostaan funktiolohkojen siirtämiseen tarkoitetun XML-tiedoston. Kyseinen kansio tulee löytymään Matlabin konsoliin asetetusta polusta koodin generoinnin jälkeen. Haluttaessaan muihin koodin generoinnin asetuksiin voi tutustua. Demonstraatiokoneen tehonhallinnan tapauksessa koodin rajapinta-asetuksista poistettiin asetus ”remove top level ssmethod type”, jonka valinnan jälkeen generoitavan lohkon sisäisten tilakoneiden lähtötila ei tule funktiolohkon sisäänmenoksi, vaan tilakone lähtee liikkeelle aina määritetystä alustuspisteestä.

Tämän jälkeen Simulink-lohkon kääntäminen PLC-koodiksi voidaan tehdä painamalla hiiren oikealla näppäimellä lohkon päältä ja valitsemalla avautuvasta valikosta PLC-koodi kohdan alta löytyvä generoi koodi alijärjestelmälle vaihtoehto. Tämän jälkeen koodin generointi alkaa. Mikäli prosessissa ilmenee virheitä, ne näkyvät Simulinkin diagnostiikkaikkunassa. Tyypillisimmät virheet johtuvat koodin generointia tukemattomien Simulink-kirjastojen käytöstä tai virheellisistä datatyypeistä. Prosessi päättyy automaattisesti avautuvaan kuvan 31 kaltaiseen raporttiin, josta voi muun muassa tarkastella generoitunutta koodia. Koska demonstraatiokoneen tehonhallinta-algoritmin alla ei ollut uudelleen käytettävää koodia muodostui prosessissa ainoastaan yksi ST-kielellä kirjoitettu funktiolohko.



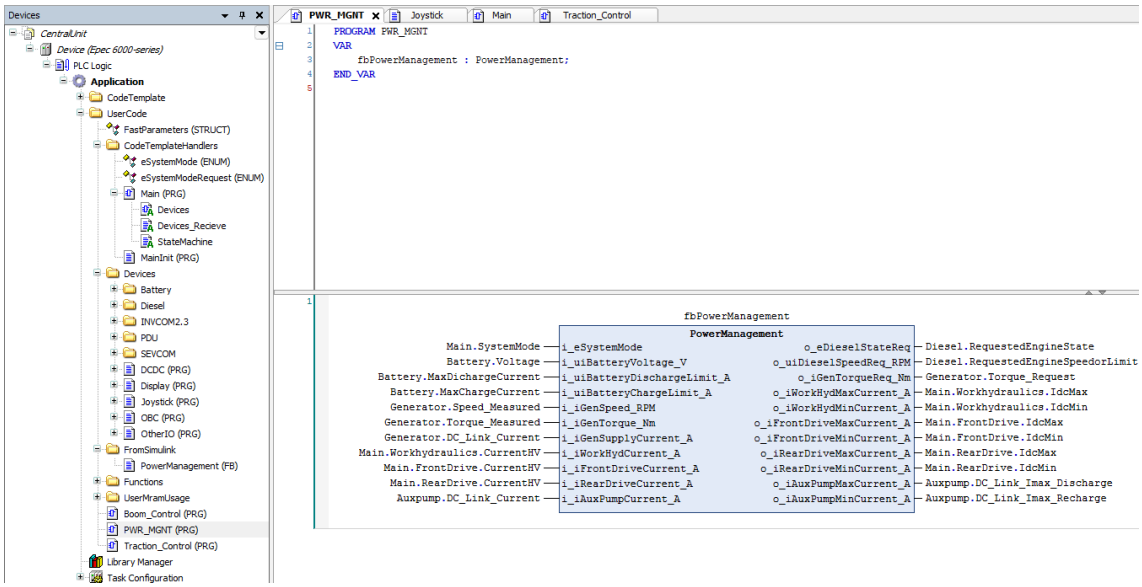
Kuva 31. *Koodin generointiraportti*

Mikäli virheitä prosessissa ei ilmennyt voidaan seuraavaksi tuoda generoitu koodi Codesys-ympäristöön. Tämä tapahtuu avaamalla projekti ja valitsemalla aktiiviseksi projektin rakennepuusta se kohta, johon generoidut funktiolohkot halutaan tuoda. Rakennepuuhun suositellaan luotavaksi yhteinen ylätason kansio kaikille automaattisesti generoidulle ohjauskoodin osille. Esimerkin tapauksessa kansio annettiin nimi ”FromSimulink”. Tämän jälkeen yläpalkista löytyy Project-valikon alta ”import PLCopenXML” -vaihtoehto, jolla generoitunut XML-tiedosto voidaan tuoda projektiin. Oikean XML-tiedon, joka löytyy aikaisemmin määritetystä kansiosta, valitsemiseen jälkeen avautuu kuvan 32 mukainen ikkuna.



Kuva 32. *PLCopenXML-tiedoston tuontiasetukset*

Mikäli XML-tiedossa on useampi funktiolohko näkyvät kaikki funktiolohkot avautuneessa ikkunassa. Jos generoitavaan Simulink-lohkoon on määritelty muuttuvia parametrejä, jotka eivät ole lohkon sisäänmenoja tai ulostuloja, ne generoituvat nähtävästi globaaleihin listauksiin GVL_CONSTS ja GVL_VARS. Tehonhallinta-algoritmissa tällaisia ei ole, minkä vuoksi listauksia ei ole valittuna. Painamalla OK tulevat valitut tiedostot aktiivisena olevaan kohtaan rakennepuussa. Tämän jälkeen funktiolohkoa voidaan käyttää normaalien käytäntöjen mukaan. Demonstraatiokoneen keskusyksikön Codesys-projektin rakennepuu ja PWR_MGNT-ohjelma, jossa generoidusta lohkosta tehdään instanssi, on esitetty kuvassa 33.



Kuva 33. *Tehonhallinnan funktiolohko tuotuna keskusyksikön Codesys-projektiin*

Generoitu koodi on rakenteeltaan yleisesti ottaen vaikeasti hahmotettavaa, jonka vuoksi tarpeelliset muutokset ovat syytä tehdä Simulink-ympäristössä. Muutoksien tekemisen jälkeen funktiolohkon uudelleen tuominen tapahtuu kuten ensimmäiselläkin kertaa. Mikäli generoitua funktiolohkoa käytetään FBD-kielellä kirjoitetussa ohjelmassa, kuten kuvassa 33, täytyy instanssi muistaa päivittää painamalla hiiren oikeaa näppäintä lohkon päällä ja valitsemalla päivitä parametrit komento.

5.5 Jatkokehitys

Diplomityössä demonstraatiotyökoneen ohjausjärjestelmän arkkitehtuuri saatiin muodostettua ja kommunikaatiorajapinnat tehtyä ja testattua. Työn ohessa toteutettiin myös huomattavasti ohjauslogiikkaa liittyen tutkimuksen kannalta merkityksettömien kokonaisuuksien ohjaukseen, kuten esimerkiksi luvussa 5.2.2 esitellyt tilakone, sähköverkosta lataamisen hallinta, apuhydrauliikan hallinta sekä jäähdytyspiirien toimintalogiikka. Myös työkoneen liikuttamiseksi ensimmäinen testiversio latauksenhallinnasta tehtiin käyttäen

yksinkertaisia sääntöjä ja lisäksi toteutettiin ajovoimansiirron akselien yksinkertainen ohjaus perustuen kaasupolkimen referenssiin. Suurinta osaa tehdystä ohjauslogiikasta ei kuitenkaan ehditty testaamaan demonstraatiokoneessa, sillä demonstraatiokoneen kokoonpano oli kesken. Kokoonpanon edistyessä diplomityön ohella tehtyä ohjauslogiikkaa tuleekin testata, korjata ja jatkokehittää. Tehtäväksi jää tutkimuksen kannalta merkityksellisten kokonaisuuksien ohjaamisen toteuttamisen lisäksi esimerkiksi graafinen käyttöliittymä näytölle, johon diplomityön ohessa ei ehditty panostaa.

Ohjausarkkitehtuurin kannalta tehtäväksi jää arkkitehtuurin validointi eli sen toteaminen, täyttääkö arkkitehtuuri luvussa 5.1 esitetyt vaatimukset. Kun kaikki demonstraatiotyökoneen väyläliitännäiset komponentit saadaan kytkettyä, tulee lisäksi myös varmistaa, ettei väylien kuormitusaste ole merkittävästi yli 50 %. Tämä voidaan tehdä kytkemällä tietokone CAN-analysaattorilla väylien diagnostiikkaliityntöihin ja seuraamalla väyläkuormitusta esimerkiksi avoimeen lähdekoodiin perustuvalla ilmaisella Busmaster (2023) ohjelmalla. Myöhemmässä vaiheessa, kun raskaampia ohjausalgoritmeja, kuten tehonhallinnassa globaalin ekvivalentin polttoaineen kulutuksen minimoinnin strategiaa tai tekoälyyn perustuvia ohjausstrategioita, testataan, tulee seurata keskusyksikön ohjelmien laskenta-aikoja. Jotta ohjelma toimii siltä odottelulta tavalta, tullee laskennan valmistua kiertoajan puitteissa eli toisin sanoen ohjelman laskenta-ajan tulee olla ohjelman kiertoaikaa lyhyempi.

Tällä hetkellä keskusyksikön koko ohjelmisto on toteutettu samaan ohjelmaan koodipohjan kanssa, jota ajetaan 10 ms kiertoajalla perustuen kommunikaatiorajapinnan vaatimuksiin. Mikäli laskenta ei onnistu ohjelmalle määritettyyn kiertoaikaan, voidaan raskas osa ohjelmistosta siirtää omalle tehtävälleen ja pohtia voidaanko uuden tehtävän kiertoaikaa kasvattaa vaikuttamatta negatiivisesti ohjausalgoritmin toimintaan. Mikäli uuden tehtävän kiertoaikaa ei voida kasvattaa, voidaan tutkia, voitaisiinko tehtävä siirtää toiselle laskentaytimelle, mikäli sillä hetkelle keskusyksikölle on julkaistu moniydinlaskentaa tukeva ohjelmistopäivitys. Hajauttamista voidaan toteuttaa myös järjestelmästä löytyvälle näytölle tai laitehankintojen avulla helposti esimerkiksi useammalle XS6C-yksikölle tai muille yli viisi CAN-kanavaa sisältäville ohjelmoitaville logiikoille. Jotta hajauttamisen aiheuttamat laskennan synkronointiin liittyvät haasteet saadaan minimoitua, hajauttamista kannattaa lähteä lähtökohtaisesti toteuttamaan vasta tarpeen tullen ja edellä esitetyssä järjestyksessä eli ensin siirtämällä ohjelmistomoduuleja eri tehtäville, sitten ytimille ja vasta lopuksi toisille ohjainlaitteille.

6. YHTEENVETO JA JOHTOPÄÄTÖKSET

Tämän työn tavoitteena on demonstraatiotyökoneen ohjausjärjestelmän arkkitehtuurin ja kommunikaatiorajapintojen muodostaminen työssä valittaville ohjelmoitaville logiikoille. Ennen suunnitteluin aloittamista selvitettiin, kuinka kehitysprosessia tulisi lähestyä, niin järjestelmäsuunnittelun kannalta kuin ohjausjärjestelmäsuunnittelun kannalta. Työssä päätettiin hyödyntää VDI:n (2206:2021) viitemallin mukaista järjestelmäsuunnittelun näkökulmaa. VDI:n (2206:2021) viitemalli painottaa vaatimusten määrittelyprosessin perusteellisuutta erityisesti projektin alussa sekä vaatimusten validointia mahdollisimman aikaisessa vaiheessa, minkä vuoksi tässä työssä kiinnitettiin erityistä huomiota vaatimusten määrittelyprosessiin.

Haasteita viitemallin mukaisen ideologian toteuttamisessa aiheutui demonstraatiokoneen käyttötarkoituksesta eli ohjausalgoritmien ja mitoitustyökalujen kehittämisestä sekä verifioimisesta. Viitemallin ideologian mukaan suunnittelussa tehtävät päätökset perustuvat täysin hyvän vaatimuksen kriteerit täyttäviin vaatimuksiin, jollaisten määrittäminen tutkimusalustalle on lähtökohtaisesti haasteellista. Hyvänä esimerkkinä toimii laskentatehonvaatimuksen määrittäminen, joka on yksi merkittävä tekijä ohjausarkkitehtuurin valinnassa. Merkittävää onkin, ettei tässä vaiheessa projektia voida täysin validioida koko ohjausarkkitehtuurisuunnittelun onnistumista.

Yleisenä huomiona voidaan lisäksi mainita, että VDI:n viitemalli painottaa tiimien välistä yhteistyötä päätöksenteossa. Kaikki demonstraatiotyökoneen voimalinjojen komponentit oli valittu ennen ohjausjärjestelmäsuunnittelun aloittamista. Siitä huolimatta, että valmiiksi valitut komponentit mahdollistivat suoraviivaisen lähestymisen ohjausjärjestelmän arkkitehtuurisuunnitteluun, komponenttivalinnoissa ei ollut huomioitu komponenttien yhteensopivuutta ohjausjärjestelmän kanssa. Suurimmat haasteet aiheutuivat Bosch INVCOM2.3-invertterien käyttämästä ei-standardoidusta CAN-tiedonsiirrosta, joka aiheutti työmäärällisesti suhteellisen suuren integraatiotyön tarpeen.

Ohjausjärjestelmäsuunnittelun näkökulman kuvaamiseksi työssä on esitelty mallipohjainen työkierto. Mallipohjaista työkiertoa pystyttiin soveltamaan SIL-, HIL- ja IIL-menetelmien osalta kommunikaatiorajapintojen toteuttamisessa. Projektissa saadun kokemuksen perusteella näiden testimenetelmien hyödyntäminen toi esille huomattavan määrän ohjelmointivirheitä jo ohjelmointivaiheessa. Huomatut virheet eivät olisi välttämättä tul-

leet esille suoraan ohjelmakoodin rivejä katsomalla. Lopputuloksena kommunikaatiorajapinnat saatiinkin testattua ja verifioitua toimiviksi ennen demonstraatiotyökoneen valmistumista, mikä osaltaan helpottaa myöhemmin tehtävää koneen käyttöönottoa.

Ennen implementaatiovaiheen aloittamista selvitettiin ohjausjärjestelmäarkkitehtuurin valintaan vaikuttavia tekijöitä. Eniten ohjausjärjestelmän arkkitehtuuria määrittäväksi tekijäksi löydettiin laskentatehon vaatimuksen ja kytkettävien laitteiden määrään kasvaminen esimerkiksi autonomian myötä. Mikäli useamman laitteen toimintaa tarvitsee koordinoida, on tyypillisin ratkaisu ylemmän tason ohjainlaitteen lisääminen järjestelmään. Myös valmistettavan tuotteen modulaarisuus ja esimerkiksi pilviliitännäisyys tunnistettiin arkkitehtuuria määrittäviksi asioiksi.

Toisin kuin ajoneuvoteollisuudessa työkoneteollisuudessa ei voida lähtökohtaisesti siirtyä työssä esitetyn kaltaiseen alueelliseen arkkitehtuuriin. Tämä johtuu siitä, että työkoneteollisuudessa pyritään hyödyntämään komponenttivalmistajien valmiita komponentteja, eikä esimerkiksi ohjainlaitteita tehdä lähtökohtaisesti työkonevalmistajan spesifikaation mukaisiksi pienten tuotantomäärien takia. Valmiit työkoneisiin soveltuvat ohjainlaitteet eivät tällä hetkellä tue Ethernet-pohjaisia tai edes uuden sukupolven CAN-pohjaisia tiedonsiirtoratkaisuja, jotka mahdollistaisivat alueellisten ohjainlaitteiden ja keskusyksikön välisen nopean, robustin ja suurikaistaisen tiedonsiirron.

Tällä hetkellä klassisen CAN-väylän tarjoama 1 Mbit/s maksimitiedonsiirtonopeus rajaakin arkkitehtuurisia ratkaisuja työkoneiden kohdalla. Lähtökohtaisesti hajauttamista kannattaakin lähestyä sellaisten toimintakokonaisuuksien hajauttamisella, joiden välillä ei tarvitse siirtää merkittävästi dataa. Tarpeen mukaan yhden toimintakokonaisuuden hajauttamisessa kannattaa puolestaan hyödyntää laskennan hajauttamista useammalle ytimelle, mitä jotkut työkoneisiin soveltuvat ohjainlaitteet ovat lähivuosina alkaneet tukea.

Tulevaisuuden työkoneiden tiedonsiirtoratkaisuihin tarvitaan lisää tiedonsiirtokaistaa, vaikka mitään laskennallisesti raskasta, kuten autonomista ajamista tai tekoälypohjaisia ohjausalgoritmeja, ei oltaisikaan toteuttamassa. Tämä johtuu siitä, että esimerkiksi ajovoimansiirron sähköistämisen myötä yhä useampi komponentti kytketään järjestelmään tiedonsiirtoratkaisun avulla. Väyläkuormituksen 50 % -mitoitusperiaatetta noudattamalla esimerkiksi demonstraatiotyökoneen CAN-liitännäisiä komponentteja ei olisi voinut kytkeä tyypillisistä työkoneista löytyvällä parilla väylällä. Suurempi kaista olisi vaadittu demonstraatiotyökoneen tapauksessa myös digitaalisten virtauksensäätöyksiköiden venttiilien virtasäädettyyn ohjaamiseen.

CAN-väylän osalta tässä työssä selvitettiin sen toimintaperiaatetta kommunikaatiorajapintojen toteuttamiseksi ja testaamiseksi. Yhtenä selvityksen kohteena oli myös, voidaanko demonstraatiotyökoneen komponentit, jotka käyttävät kolmea eri CAN-pohjaista tiedonsiirtoratkaisua, kytkeä samaan väylään. Selvityksen pohjalta todettiin, että kaikki demonstraatiotyökoneen CAN-väyläratkaisut perustuvat ISO 11898 -standardin CAN2.0 A ja B väyläversioihin, jotka eivät eroa toisistaan datansiirtoon tarkoitettua viestikehystukea lukuun ottamatta. Versioista CAN2.0A tukee normaalia viestikehystä, mutta siihen pohjautuvien CAN-lähetinvastaanottimien täytyy kyetä toimimaan väylällä, jossa liikkuu myös jatkettua kehystä käyttäviä viestejä. CAN2.0B-versioon pohjautuvat lähetinvastaanottimet tukevat puolestaan niin jatkettua kuin normaalia kehystä.

Demonstraatiokoneen CAN-pohjaisista laitteista CANopen-laiteet ja INVCOM2.3-invertterit käyttävät CAN2.0A-lähetinvastaanottimia ja J1939-laitteet sekä ohjelmoitavat logiikat CAN2.0B version mukaisia lähetinvastaanottimia. Koska komponenttien ei tarvitse siirtää dataa keskenään, vaan ainoastaan ohjelmoitavien logiikkojen kanssa, voitaisiin teoriassa kaikki komponentit kytkeä samalle väylälle huomioimalla muutama työssä esitetty haaste, jos kaikkien laitteiden baudinopeudet voitaisiin asettaa samoiksi. Loppujen lopuksi eri ylemmän tason standardien käyttäminen samalla väylällä pystyttiin kiertämään demonstraatiotyökoneen tapauksessa arkkitehtuurisilla valinnoilla.

Työn päätutkimuskysymykseen eli kysymykseen ”millainen ohjausjärjestelmän arkkitehtuuri demonstraatiotyökoneeseen tulisi valita?” vastattiin prosessinomaisesti luvussa 5 ottamalla huomioon sovelluskohteeseen liittyvät lähtökohdat ja rajoitukset laajan vaatimuksien määrittelyprosessin kautta. Demonstraatiokoneeseen valittiin lähtökohtaisesti yksi suhteellisen tehokas ohjelmoitava logiikka, jolla on tarkoitus toteuttaa kaikki tutkimuksen kannalta oleelliset ohjausalgoritmit ja esitettiin, kuinka myöhemmässä vaiheessa laskentaresursseja voidaan kasvattaa hajauttamalla. Päätöksen taustalla oli pyrkimys pitää arkkitehtuuri lähtökohtaisesti yksinkertaisena ja ohjelmakoodi helposti muokattavana yhdestä paikkaa. Tutkimuksen kannalta merkityksettömät, mutta työkoneen toiminnan kannalta oleelliset, toiminnallisuudet puolestaan hajautettiin tarkoituksena säilyttää näiden toimivuus riippumatta tutkimuksen aikaisista ohjelmakoodimuutoksista tai myöhemmässä vaiheessa tehdyistä laitteistopäivityksistä. Merkittävin haaste muodostui siitä, ettei tulevaisuudessa demonstraatiotyökoneella testattavien ohjausalgoritmien laskentaresurssien tarvetta voi määrittää etukäteen. Näillä näkymin työssä valittu arkkitehtuuri näyttää soveltuvan demonstraatiotyökoneeseen, mutta lopullinen validointi voidaan tehdä vasta demonstraatiotyökoneen valmistuessa.

LÄHTEET

Ahopelto, M. (2019). Towards Automation and Improved Fuel Economy with System Architecture Design of a Non-Road Working Machine. Väitöskirja. Tampereen yliopisto. Saatavissa: https://urn.fi/URN:ISBN:978-952-03-1290-9

Askaripoor, H., Hashemi Farzaneh, M., & Knoll, A. (2022). E/E Architecture Synthesis: Challenges and Technologies. Electronics, 11(4), 518.

Backas, J. (2018). Energy Efficient Control of Hydrostatic Drive Transmissions: A Non-linear Model-Based Approach. Väitöskirja. Tampereen yliopisto. Saatavissa: https://urn.fi/URN:ISBN:978-952-15-4245-9

Blanchard, B. S., & Blyler, J. (2016). System engineering management (5. edition). Hoboken, New Jersey: Wiley.

Bosch Rexroth (2022). C-API, verkkosivu. Saatavissa (viitattu 2.11.2022): https://www.boschrexroth.com/en/xc/products/product-groups/mobile-hydraulics/mobile-electronics/bodas-software/bodas-tools/c-api

Busmaster (2023). Open Source Software tool to Simulate, Analyze and Test data bus systems such as CAN and LIN, ohjelmisto. Saatavissa (viitattu 20.11.2023): https://rbei-etas.github.io/busmaster/

CAN in Automation, CiA. (2023a). About us, verkkosivu. Saatavissa (Viitattu 10.11.2023): https://www.can-cia.org/about-us/

CAN in Automation, CiA. (2023b). Technical documents, verkkosivu. Saatavissa (viitattu 27.11.2023): https://www.can-cia.org/groups/specifications/

CAN in Automation, CiA. (2023c). Designing a CAN network, verkkosivu. Saatavissa (Viitattu 10.11.2023): https://www.can-cia.org/can-knowledge/can/design-can-network/

CiA-102 (1994). CAN Physical Layer for Industrial Applications. CAN in Automation.

CiA-106. (2023). Connector pin-assignment recommendations. CAN in Automation.

CiA-301. (2011). CANopen application layer and communication profile. CAN in Automation.

CiA-303–1 (2023). Device and network design – Part 1: CANopen physical layer. CAN in Automation.

CiA-306 (2005). CANopen electronic data sheet (EDS). CAN in Automation.

Clean Propulsion Technologies (2023). Paving the Way to a Greener Tomorrow: Revolutionizing Off-Road Vehicles for a Sustainable Future, verkkosivu. Saatavissa (viitattu 22.11.2023): https://cleanpropulsion.org/arkistot/2048

Codesys. (2022). Codesys is the leading manufacturer-independent IEC 61131-3 automation software for engineering control systems, verkkosivu. Saatavissa (viitattu 1.11.2022): https://www.codesys.com/

Codesys. (2023a). Rules and recommendations for identifier and variable names, verkkosivu. Saatavissa (viitattu 1.11.2023): https://help.codesys.com/api-content/2/codesys/3.5.12.0/en/_cds_identifiers/

Codesys. (2023b). Integer datatypes, verkkosivu. Saatavissa (viitattu 1.11.2023): https://help.codesys.com/api-content/2/codesys/3.5.14.0/en/_cds_datatype_integer/

Dasa. (2022). Controllers mobile machines, verkkosivu. Saatavissa (viitattu 1.11.2022): https://www.dasa.se/en/products/controllers/controllers-mobile-machines/

Direktiivi 2006/42/EC. (2019). Euroopan parlamentti ja neuvosto. L157/24, 2006. Saatavissa (viitattu 22.06.2022): https://eur-lex.europa.eu/eli/dir/2006/42/2019-07-26

Di Natale, M., Zeng, H., Giusto, P., & Ghosal, A. (2012). Understanding and Using the Controller Area Network Communication Protocol Theory and Practice. New York: Springer.

dSpace. (2022). Product Finder, verkkosivu. Saatavissa (viitattu 4.11.2022): https://www.dspace.com/en/ltd/home/products/products.cfm

Ehsani, M., Singh, K., Bansal, H. & Mehrjardi, R. (2021). State of the Art and Trends in Electric and Hybrid Electric Vehicles. Proceedings of the IEEE, 109(6), 967–984.

EPEC (2022). Software development environment, verkkosivu. Saatavissa (viitattu 1.11.2022): https://epec.fi/epec-oy-products/application-development-environment/

EPEC (2023a) Connectivity, verkkosivu. Saatavissa (Viitattu 25.11.2023): https://epec.fi/epec-oy-products/connectivity/

EPEC (2023b). Products, verkkosivu. Saatavissa (Viitattu 20.11.2023): https://epec.fi/epec-oy-products/

Forrai, A. (2013). Embedded Control System Design: A Model Based Approach. Heidelberg: Springer Berlin.

GHS Infotronic. (2023). Online CRC Calculation, verkkosivu. Saatavissa (viitattu: 20.11.2023): https://www.ghsi.de/pages/subpages/Online%20CRC%20Calculation/indexDetails.php?

Graessler, I., Hentze, J., & Bruckmann, T. (2018). V-models for interdisciplinary systems engineering. Proceedings of the DESIGN 2018 15th International Design Conference, 747–756.

Graessler, & Hentze, J. (2020). The new V-Model of VDI 2206 and its validation. Automatisierungstechnik, 68(5), 312–324.

Hawe hydraulik (2022). Mobile controllers, Packages & Sensors, verkkosivu. Saatavissa (viitattu 1.11.2022): https://www.hawe.com/topics/electronic-controls-with-stw/mobilecontroller/

Ishida, K. & Higurashi, M. (2015). Hybrid Wheel Loaders Incorporating Power Electronics. Hitachi Review, 64(7). 398–402.

Ixia. (2014). Automotive Ethernet: An Overview, tiedosto verkkosivuilta. Saatavissa (viitattu 30.10.2023): https://support.ixiacom.com/sites/default/files/resources/whitepaper/ixia-automotive-ethernet-primer-whitepaper_1.pdf

Huova, M., Karvonen, M., Ahola, V., Linjama, M., & Vilenius, M. (2010). Energy efficient control of multiactuator digital hydraulic mobile machine. 7th International Fluid Power Conference, 22-24 March 2010, Aachen, Germany

Huova, M., Heikkilä, M., Linjama, M., Tikkanen, S., & Huhtala, K. (2018). Comparison of energy saving methods for loader. Proceedings of the 10th Colloquium Mobile Hydraulics, 16-17 October 2018, Brunswick, Germany

Immonen, P. (2013). Energy efficiency of a diesel-electric mobile working machine. Väitöskirja. Lappeenrannan teknillinen yliopisto. Saatavissa: https://urn.fi/URN:ISBN:978-952-265-415-1

ISO/IEC 7498-1. (1994). Information technology – Open systems Interconnection – basic reference model. The International Organization for Standardization.

J1939. (2023). Serial Control and Communications Heavy-Duty Vehicle Network - Top-Level Document. Society of Automotive Engineers.

J1939-1. (2021). On-Highway Equipment Control and Communication Network. Society of Automotive Engineers.

J1939-11. (2016). Physical Layer, 250 Kbps, Twisted Shielded Pair. Society of Automotive Engineers.

J1939-14. (2022). Physical Layer, 500 kbit/s. Society of Automotive Engineers.

J1939-21. (2022). Data Link Layer. Society of Automotive Engineers.

J1939-31. (2023). Network Layer. Society of Automotive Engineers.

J1939-71. (2022). Vehicle Application Layer. Society of Automotive Engineers.

J1939-81. (2017). Network Management. Society of Automotive Engineers

J1939-DA. (2023). SAE J1939 Digital Annex. Society of Automotive Engineers.

Koohi-Fayegh, S. & Rosen, M. A. (2020). A review of energy storage types, applications, and recent developments. Journal of Energy Storage, 27.

Linjama, M. (2011). Digital fluid power – State of the art. Proceedings Of The Twelfth Scandinavian International Conference on Fluid Power Volume 3, SICFP'11. 331– 353.

Lukka, K. (2001). Konstruktiivinen tutkimusote, verkkosivu. Saatavissa (viitattu 24.10.2023): https://metodix.fi/2014/05/19/lukka-konstruktiivinen-tutkimusote/

Mathworks. (2022a). Simulink: Design. Simulate. Deploy, verkkosivu. Saatavissa (viitattu 4.11.2022): https://se.mathworks.com/products/simulink.html

Mathworks. (2022b). Communicate with Hardware Using Connected IO, verkkosivu. Saatavissa (viitattu 4.1.2022): https://se.mathworks.com/help/supportpkg/nucleo/ug/connected-io.html

Mathworks. (2022c). Simulink Real-Time: Perform rapid control prototyping and hardware-in-the-loop testing, verkkosivu. Saatavissa (viitattu 4.11.2022): https://se.mathworks.com/products/simulink-real-time.html

Mathworks. (2022d). INCOVA Designs Intelligent Valve-Control System for a 20-Ton Excavator, verkkosivu. Saatavissa (viitattu 4.11.2022): https://se.mathworks.com/company/user_stories/incova-designs-intelligent-valve-control-system-for-a-20-ton-excavator.html?s_tid=srchtitle

Mathworks. (2023). Integers, verkkosivu. Saatavissa (viitattu 1.11.2023): https://se.mathworks.com/help/matlab/matlab_prog/integers.html

Microchip. (2000). CRC Generating and Checking, tiedosto verkkosivuilla. Saatavissa (viitattu 20.11.2023): https://ww1.microchip.com/downloads/en/AppNotes/00730a.pdf

Microsoft. (2022). Visual Studio: IDE and Code Editor for software Developers and Teams, verkkosivu. Saatavissa (viitattu 6.11.2022): https://visualstudio.microsoft.com/

Munassar, N. M. A., & Govardhan, A. (2010). A Comparison Between Five Models Of Software Engineering. International Journal of Computer Science Issues, 7(5), 94–94.

Nokka J. (2018). Energy Efficiency Analyses of Hybrid Non-Road Mobile Machinery by Real-Time Virtual Prototyping. Väitöskirja. Lappeenrannan teknillinen yliopisto. Saatavissa: https://urn.fi/URN:ISBN:978-952-335-193-6

Ponsse. (2022). Ponsselta teknologialanseeraus: sähkökäyttöinen metsäkone, verkkosivu. Saatavissa (viitattu 19.12.2022): https://www.ponsse.com/fi/yhtio/uutiset/a_p/P4s3zYhpxHUQ/c/ponsse-launches-new-technology-an-electric-forest-machine#/

Ponsse. (2023). Ponsse manager vie metsäkoneiden digitalisaation uudelle tasolle, verkkosivu. Saatavissa (viitattu 10.5.2023): https://www.ponsse.com/fi/services/online-services/ponsse-manager#/

Sandvik. (2022). Maanalaiset lastauskoneet ja dumpperit, verkkosivu. Saatavissa (viitattu 27.7.2022): https://www.rocktechnology.sandvik/fi/laitteet/maanalaiset-lastauskoneet-ja-dumpperit/

Serrao, L., Onori, S. & Rizzoni, G. (2011). A Comparative Analysis of Energy Management Strategies for Hybrid Electric Vehicles. Journal of Dynamic Systems, Measurement, and Control, 133(3).

Shabbir, W. & Evangelou, S. (2016). Exclusive Operation Strategy for the Supervisory Control of Series Hybrid Electric Vehicles. IEEE Transactions on Control Systems Technology, 24(6), 2190–2198.

Shaibani, M., Mirshekarloo, M. S., Singh, R., Easton, C. D., Cooray, M. C. D., Eshraghi, N., … Majumder, M. (2020). Expansion-tolerant architectures for stable cycling of ultrahigh-loading sulfur cathodes in lithium-sulfur batteries. Science Advances, 6(1)

Siemens. (2022). Simcenter Amesim: Optimize system performance from early design stages, verkkosivu. Saatavissa (viitattu 4.1.2022): https://www.plm.automation.siemens.com/global/en/products/simcenter/simcenter-amesim.html

Skjetne, R., & Egeland, O. (2006). Hardware-in-the-loop testing of marine control systems. Modeling, Identification and Control, 27(4), 239–258.

Socci, V. (2015). Implementing a model-based design and test workflow. 2015 IEEE International Symposium on Systems Engineering (ISSE), 130–134.

Speedgoat. (2022). Solutions: Accelerate testing of Control design and embedded controllers, verkkosivu. Saatavissa (viitattu 4.11.2022): https://www.speedgoat.com/solutions

Technion. (2022). Controllers, verkkosivu. Saatavissa (viitattu 1.11.2022): https://technion.fi/controllers/

The Economists. (2010) Tech.View: Cars and software bugs, verkkosivu. Saatavissa (viitattu 12.10.2022): http://www.economist.com/blogs/babbage/2010/05/techview_cars_and_software_bugs

Tiainen, L. (2014). Digitaalihydraulisen venttiilistön ohjauselektroniikan suunnittelu. Diplomityö. Tampereen teknillinen yliopisto. Saatavissa: https://urn.fi/URN:NBN:fi:tty-201401041009

Transport & Environment. (2022) Ford, Volvo Cars and broad industry coalition appeals to EU to ensure all new cars and vans are zero emissions from 2035 and to establish mandatory charging infrastructure targets: Carmakers Ford and Volvo Cars join 26 companies in public call to EU legislators ahead of crucial votes, verkkosivu. Saatavissa: (viitattu 22.7.2022) https://www.transportenvironment.org/discover/ford-volvo-cars-and-broad-industry-coalition-appeals-to-eu-to-ensure-all-new-cars-and-vans-are-zero-emissions-from-2035-and-to-establish-mandatory-charging-infrastructure-targets/

Tupitsina, A., Montonen, J.-H., Alho, J., Immonen, P., Lauren, M., Lindh, P. & Lindh, T. (2021). Simulation tool for dimensioning power train of hybrid working machine. Modeling, Identification and Control, 42(4), 143–158.

Tupitsina, A., Linjama, M., Laurila L., Multanen, P. & Lindh T. (2023) Selection method of hybridization topology for mobile work machine, julkaisematon julkai-su.

Vector. (2023) CAN Bus Load Calculation, verkkosivu. Saatavissa (viitattu 8.11.2023): https://support.vector.com/kb?id=kb_article_view&sysparm_article=KB0012332&sys_kb_id=99354e281b2614148e9a535c2e4bcb6d&spa=1

Volvo. (2017). News: Lx1 prototype hybrid wheel loader delivers around 50% fuel efficiency improvement during customer testing, verkkosivu. Saatavissa (viitattu 9.1.2023): https://www.volvoce.com/united-states/en-us/about-us/news/2017/lx1-prototype-hybrid-wheel-loader/

Volvo. (2022). Electric machines, verkkosivu. Saatavissa (viitattu 27.7.2022): https://www.volvoce.com/europe/en/products/electric-machines/

Wille Machines. (2018). Wille 655, tiedosto verkkosivuilta. Saatavissa (viitattu 18.8.2022): https://www.willemachines.com/sites/default/files/attachments/2018_665iv_EN.pdf

LIITE A: DEMONSTRAATIOKONEEN KÄYTTÖTAPAUSKAAVIO