Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Niilo Koivunen

Komentosiltalaitteiden simulointi LabVIEW-ohjelmointiympäristössä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Sähkö- ja automaatiotekniikan tutkinto-ohjelma

Insinöörityö

25.11.2019

Tekijä Otsikko Sivumäärä Aika	Niilo Koivunen Komentosiltalaitteiden simulointi LabVIEW-ohjelmointiympäristössä 50 sivua 25.11.2019
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	sähkö- ja automaatiotekniikka
Ammatillinen pääaine	automaatiotekniikka
Ohjaajat	R&D Product Owner, RCS Olli Niemi lehtori Reijo Leinonen

Opinnäytetyön tarkoituksena oli kehittää komentosiltalaitteiden simulointiohjelma automaatiojärjestelmien testausta varten LabVIEW-ohjelmointiympäristöllä. Työ tehtiin ABB Oy:n Marine and Ports -yksikölle, joka kehittää sähkö- ja automaatioratkaisuja meriteollisuuden tarpeisiin. Työn tavoitteena oli kehittää ohjelmisto, joka kommunikoi automaatiojärjestelmän kanssa simuloiden fyysisiä komentosiltalaitteita. Insinöörityön puitteissa ei kuitenkaan ollut tarkoitus luoda ohjelmistoon kaikkia laitteita, vaan työssä toteutettiin simulointi azimuth-kahvasta sekä ohjelmiston ohjelmarakenne. Ohjelmisto liitetään automaatiojärjestelmään samoin kuin oikea fyysinen laite ja sitä voidaan käyttää tavallisella kannettavalla tietokoneella.

Digitalisaation myötä laivojen komentosiltaympäristö on muuttunut paljon muutamassa vuosikymmenessä. Nykyään komentosillat ja niiden laitteet ovat vahvasti digitaalisia luoden tarpeen tuottaa komentosiltaympäristöön entistä enemmän ohjelmistoja ja älykkäitä laitteita. Ohjelmistoja ei voida kuitenkaan kaikissa tapauksissa testata perinteisesti kokonaisilla järjestelmillä. Tämä olisi aikaa vievää ja vaatisi suuren määrän komentosiltalaitteita. Tästä heräsi tarve suorittaa automaatio-ohjelmistojen testaus käyttäen simuloituja laitteita.

Työssä käytettiin graafista LabVIEW-ohjelmointiympäristöä ja Kvaserin USB-CAN-adapteria. Lopputuloksena syntyi toimiva simulointiohjelmisto, jossa on toteutettuna ja testattuna azimuth-kahva ja pääohjelma. Työssä on kuitenkin jatkokehitettävää, muun muassa käyttöliittymän osalta. Uusia simuloitavia laitteita on jatkossa lisättävä, jotta ohjelmistolla on mahdollista kattaa kaikki testaustarpeet.

Avainsanat	labview, simulaatio, komentosilta, alus, CAN, automaatio

Author Title Number of Pages Date	Niilo Koivunen Simulating bridge devices with LabVIEW software 50 pages 25 November 2019
Degree	Bachelor of Engineering
Degree Programme	Electrical and Automation Engineering
Professional Major	Automation Technology
Instructors	Olli Niemi, R&D Product Owner, RCS Reijo Leinonen, Senior Lecturer

The purpose of this thesis study was to develop a software that simulates bridge devices for testing with automation systems. The thesis study was carried out for ABB's Marine and Ports unit in Vuosaari, Helsinki. The unit develops electrical and automation solutions for the marine industry. In this thesis study, a software was implemented for communicating with the automation system by simulating physical bridge devices. However, it was not the aim of the thesis study to create all the hardware in the software: only the simulation of the Azimuth lever and the structure of the software were carried out. The software is connected to an automation system, similarly as a real physical device, and can be used on a standard laptop. LabVIEW graphical programming environment and Kvaser's USB-CAN adapter were used.

With digitalization, the bridge environment of ships has changed significantly over past decades. Nowadays bridges and their devices are heavily digitalized, creating the need to produce more software and add more digital devices for the bridge environment. Everything cannot be traditionally tested with complete automation systems. That would be time consuming and requires many bridge devices. This led to the need to test automation software using completely simulated devices.

LabVIEW graphical programming environment and Kvaser's USB-CAN adapter were used. The result is a working software that had an Azimuth lever implemented and tested. There is still room for improvement, including the user interface. New simulated hardware needs to be added to the software in the future so that the software can cover all testing needs.

Keywords	labview, simulation, bridge, ship, CAN, automation

Sisällys

Lyhenteet

ABS	*Antilock Brake System*. Lukkiutumattomat jarrut
CAN	*Control Area Network.* Alkujaan autoteollisuuden käyttöön kehitetty kustannustehokas sarjaliikenneväylä.
COB-ID	*Communication Object Identifier*. CANopen-protokollan viestin tunnistenumero.
EDS	*Electronic Data Sheet*. Elektroninen datalehti.
EMCY	*Emergency Object*. CANopen-protokolla, jolla siirretään virheilmoituksia
NMT	*Network Management*. CANopen-verkonhallintaprotokolla.
OD	*Object Dictionary*. Objektikirjasto.
OSI	*Open System Interconnection*. Avoin viitemalli tietokoneiden väliseen kommunikointiin.
PDO	*Process Data Objects*. CANopen-protokolla sykliseen tiedonsiirtoon.
SDK	*Software Development Kit*. Kehityspakkaus ohjelmien kehittämiseksi.
SDO	*Service Data Objects*. CANopen-protokolla laitteiden konfigurointiin.

1 Johdanto

Työn tilaajana toimi ABB Oy:n Marine and Ports -yksikkö, ja se toteutettiin vuoden 2019 aikana. Marine and Ports -yksikkö kehittää sähkö- ja automaatioratkaisuja meriteollisuuteen ja Suomessa sillä on toimipisteet Vuosaaressa ja Haminassa. Marine and Ports työllistää yli 1700 henkilöä maailmanlaajuisesti 20 eri maassa. [1.]

Insinöörityössä toteutettiin komentosillan fyysisten laitteiden simulointiympäristö automaatio-ohjelmistojen testauksen mahdollistamiseksi pienemmällä työmäärällä, lyhyemmässä ajassa ja vähemmillä kustannuksilla. Ohjelmisto jäljittelee komentosillan laitteistoa ja kytkeytyy CAN-väylään samoilla liittimillä ja protokollilla kuin fyysinen laite. Eri ohjelmistoversioita testattaessa fyysisten laitteiden konfiguraatio eroaa toisistaan ja testattavaa ohjelmistoa vaihdettaessa myös laitteiden konfiguraatio on vaihdettava. Tämä teettää huomattavasti työtä ja varaa komentosillan laitteiston pitkäksi aikaa yhtä testiä varten, johon insinöörityössä toteutettu ohjelmisto tuo helpotusta. Ohjelmiston avulla fyysisiä komentosillan laitteita ei tarvita ja laitekonfiguraatio voidaan tehdä ohjelmallisesti. Samalla aikaa laitteet näyttäytyvät automaatiojärjestelmään päin kuin oikeat laitteet. Testausohjelmiston vaatimuksiksi asetettiin helppokäyttöisyys, laajennettavuus ja mahdollisuus käyttää ohjelmistoa tavallisella kannettavalla tietokoneella.

Ohjelmisto toteutettiin graafisella ohjelmointiympäristöllä LabVIEW 2019 käyttäen ohjelman toteutukseen LabVIEW'n graafista G-kieltä. Ohjelmointiympäristön valinnan jälkeen päädyttiin käyttämään rajapintana automaatiojärjestelmään Kvaserin USBcan Light 4x HS -adapteria. Adapteri tukee jopa neljää CAN-väylää, ja sille on julkaistu CANlib SDK ja LabVIEW-kirjasto, joita työssä käytettiin ohjelmarajapinnan toteuttamiseen adapterin ja ohjelmiston väliin.

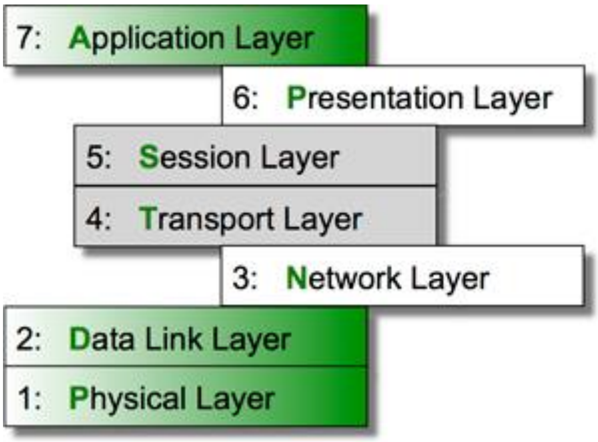
Suomenkielisten, yleisesti käytössä olevien termien puuttuessa työssä on käytetty englanninkielisiä termejä niiden tunnettuuden ja selkeyden vuoksi.

2 Control Area Network -väylä

Työssä käytetään Control Area Network -väylää (CAN) rajapintana automaatiojärjestelmän ja simulointiohjelman välillä. Komentosillalla olevat laitteet ovat kaikki liitettynä CAN-verkkoon, jonka kautta ne kommunikoivat ohjelmoitavien logiikoiden kanssa. Luonnollisesti tämän takia insinöörityössäkin on käytetty CAN-verkkoa rajapintana, jotta fyysiset liittimet ja kommunikointiprotokolla ovat yhteensopivia fyysisten laitteiden kanssa.

CAN-väylän on alkujaan kehittänyt Bosch autoteollisuuden tarpeisiin 1980-luvulla. Elektronisten laitteiden käyttö autoissa lisääntyi aiheuttaen kaapelimäärän suuren kasvun, joka johti kompleksiseen rakenteeseen ja suuriin kustannuksiin. Tuohon aikaan laitteet kytkettiin point-to-point-yhteyksillä laitteiden välillä ja jokaiselle oli varattava omat johtimet. Tähän CAN-väylä toi ratkaisun. CAN-väylän avulla on mahdollista rakentaa väylä, jonka kautta useat eri laitteet voivat kommunikoida, eikä jokaiselle tarvita omia johtimia. Väyläratkaisu madalsi kaapelointikustannuksia ja CAN-väylän tarvitsema elektroniikka on suunniteltu mahdollisimman kustannustehokkaaksi. CAN-väylän myötä jokainen kytkettävä laite oli mahdollista tehdä älykkääksi ja osa tiedonkäsittelystä tehdään jo itse laitteella.

CAN-väylä sisältää myös poikkeuskäsittelyn ja viestien priorisoinnin, ja sitä onkin mahdollista käyttää monenlaisissa applikaatioissa. Autoteollisuuden lisäksi muu teollisuus löysikin CAN-väylän sen joustavuuden, häiriösietoisuuden ja kustannustehokkuuden vuoksi. Nykyään sitä käytetään muun muassa junissa, metroissa, raitiovaunuissa, lentokoneissa ja tietenkin laivoissa. Yllättäviäkin käyttötarkoituksia sille löytyy: monissa sairaalalaitteissa käytetään CAN-väylää muun muassa valojen, kameroiden, röntgenlaitteiden sekä potilaspetien ohjauksessa. CAN-väylä pohjautuu Open System Interconnection-malliin (OSI), ja se sisältää fyysisen kerroksen, siirtoyhteyskerroksen ja sovelluskerroksen (kuva 1). [2; 3.]



Kuva 1. OSI-mallin kerrokset. [3.]

2.1 Fyysinen kerros

CAN-väylässä on toteutettuna useita erilaisia fyysisiä kerroksia. Fyysinen kerros määrää käytettävät signaalitasot, kaapelin impedanssin, bittikehyksen ja väylänopeuden. Fyysisen kerroksen kaapelointiratkaisuja on useita erilaisia. Käyttötarkoitukseen sopivan kaapeloinnin valinta riippuukin tarvittavasta väylänopeudesta, häiriönsietokyvystä sekä kustannuksista. Tässä työssä käytettiin High-speed-standardia ja kaapelointina kierrettyä parikaapelia, sekä DB9-liittimiä (kuva 2).

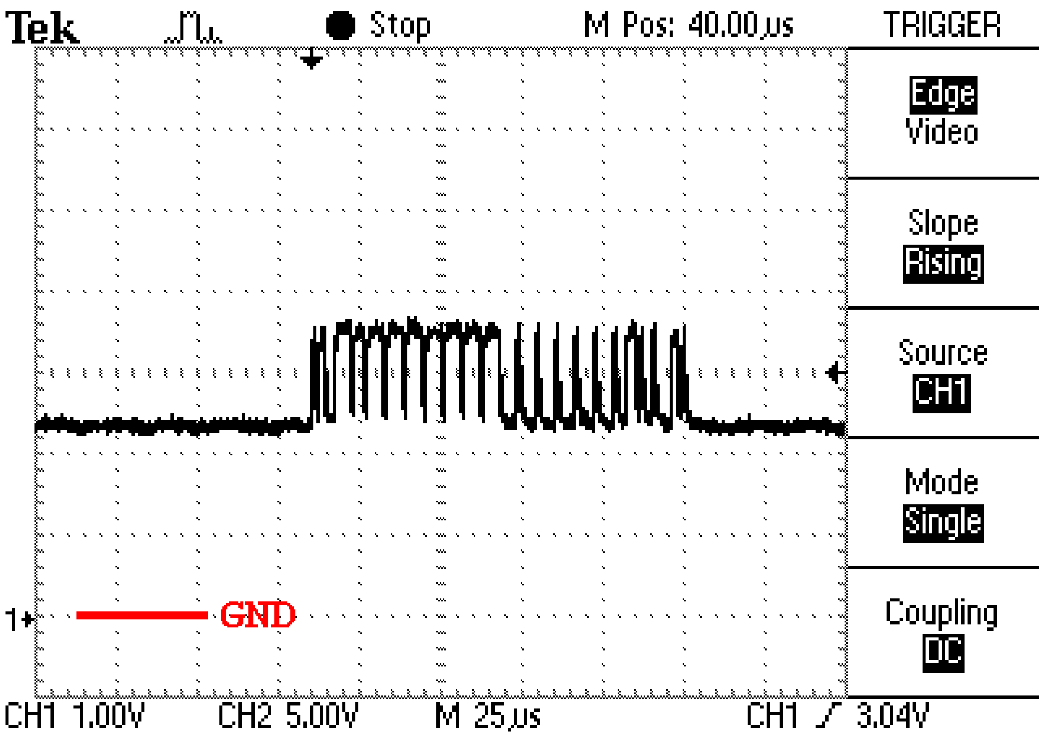
Pin	Signal	Description
1	-	Reserved for CiA
2	CAN_L	CAN bus-line dominant low
3	CAN_GRD	CAN ground
4	-	Reserved for CiA
5	(CAN_SHLD)	Optional CAN shield
6	(GND)	Optional power ground
7	CAN_H	CAN bus-line dominant high
8	-	Reserved for CiA
9	(CAN_V+)	Optional (external) power supply

Kuva 2. CAN-väylän pinnijärjestys DB9-liittimessä. [27.]

High-speed CAN

High speed CAN, standardi ISO 11898-2, on yksi yleisimpiä väyläratkaisuja, ja se yleensä toteutetaan kierretyllä parikaapelilla. Väyläratkaisun etuna on jopa 1 Mbit/s:n nopeus. Väylännopeus vaikuttaa sen pituuteen. Esimerkiksi 125 kbit/s väylän pituus voi olla jopa 500 metriä, kun 500 kbit/s nopeudella maksimipituus tippuu 100 metriin. [4.]

Tyypillisesti ratkaisua käytetään esimerkiksi moottorinohjausyksikössä, Antilock Brake System -jarruissa (ABS) ja päästöjen vähennysjärjestelmissä. Järjestelmään kytketyt lähetinvastaanottimet voidaan kytkeä joko 5 V:n tai 3,3 V:n jännitetasoihin ja jopa eri jännitetasoja voidaan käyttää samassa väylässä. Eri jännitetasojen käyttäminen kuitenkin heikentää väylän häiriönsietokykyä. Väylän kaapelointi koostuu kahdesta johtimesta, joista toinen on CAN_H (HIGH) ja toinen CAN_L (LOW). Väylällä on loogisia tiloja kaksi: resessiivinen, jolloin tila on looginen 1 ja kummankin linjan jännitepotentiaali on tällöin 2,5 V ja jännite-ero 0V, sekä dominantti, jolloin CAN_H-johtimen jännitepotentiaali on 3,5 V ja CAN_L-johtimen 1,5 V, jolloin jännite-ero on 2 V ja tila on looginen 0 (kuva 3). Tämän lisäksi CAN-verkko on terminoitu päästään 120 ohmin vastuksella. [2; 3; 5.]



Kuva 3. Oskilloskooppikuva CAN-signaalista. [4.]

Low-Speed CAN

Low-Speed CAN -väylän nopeus on maksimissaan 125 Kbit/s ja hitaimmillaan 40 Kbit/s. Low Speed CAN -protokollassa jokainen laite tarvitsee oman päätevastuksensa, mutta johtimien määrä on vastaava kuin High Speed CAN -väylässä, eli kaksi kappaletta. Väyläratkaisu kuitenkin toimii, vaikka toinen datalinjoista olisi poikki. Tässä tapauksessa kommunikointi hoituu vain toista datalinjaa pitkin ja vian korjaantuessa kommunikointi palaa käyttämään kahta datalinjaa. Tämän johdosta jokainen laite vaatii oman päätevastuksensa. Low Speed CAN -väylän käyttö on kuitenkin vähentynyt High Speed CAN -väylän myötä. [6.]

Muut fyysiset tasot

Aiemmin mainittujen lisäksi on olemassa useita muita fyysisiä tasoja, joista osa on kehitetty hyvin spesifiin käyttöön. Näistä esimerkkinä on Single-Wire CAN, joka nimensä mukaisesti on toteutettu yhdellä datalinjalla. Tätä käytetään lähinnä autoteollisuudessa kohteissa, joissa tarvitaan kustannustehokkuutta. Se on terminoitu yhdestä päästä, ja sen miniminopeus on 33,3 kbit/s ja maksiminopeus 83,3 kbit/s. [6.]

2.2 Siirtoyhteyskerros

CAN on multi-master-sarjaliikenneväylä, jonka viansietokyky on hyvä. CAN-väylässä viestit lähetetään väylään, mikä tarkoittaa sitä, että jokainen väylässä oleva node pystyy vastaanottamaan minkä tahansa muun noden lähettämän viestin. Alun perin CAN-protokollaan oli toteutettuna vain yksi siirtoyhteyskerros, jota nykyään kutsutaan Classical CAN:ksi ja aiemmin sitä on kutsuttu CAN 2.0 A/B:ksi. Classical CAN esiteltiin ensi kerran vuonna 1986 ja implementointiin 1988. Se olikin pitkään ainoa siirtoyhteyskerros CAN-protokollassa, mutta sen enimmäistiedonsiirtonopeus on 1 Mbit/s. Tiedonsiirtonopeuden ja datakehyksen kasvattamiseksi julkaistiin vuonna 2012 CAN FD, jonka kehityksen Bosch laittoi alulleen, ja se standardisoitiin 2015 standardissa ISO 11898-1. FD tulee sanoista Flexible Datarate ja sen eri kommunikoinnin vaiheissa käytetään vaihtelevia datanopeuksia. Erona siinä Classic CAN:iin on tuki jopa 64 bitin datakehyksille ja 5 Mbit/s nopeuksille. [7; 8.] 1

2.3 Sovelluskerros

CAN-pohjaiset verkkoratkaisut pohjautuvat OSI-malliin, joskin kerrokset 3–6 ovat harvemmin implementoituja CAN-ratkaisuissa. Kerros 1 on aiemmin esitelty fyysinen kerros ja kerros 2 määrittelee kommunikoinnin perustasolla. Kerroksella 7 on sovelluskerros, joka on ylemmän tason protokolla. Ylemmän tason protokolla vastaa ohjelman toiminnallisuudesta ja voi sisältää verkonhallinnallisia toimintoja. Tämä tarkoittaa esimerkiksi CAN-nodejen käynnistystä ja pysäytystä sekä niiden valvontaa tapahtumapohjaisissa verkoissa. Esimerkiksi väylältä kadonneiden nodejen tunnistus voi olla tämän kerroksen tehtävä. Tämänkaltaisia nodeja voidaan valvoa esimerkiksi heart-beat-viesteillä, joissa node lähettää tietyn väliajoin statustietonsa väylään. Esimerkkinä tämänkaltaisesta toiminnallisuudesta on esimerkiksi tässä työssä käytetyn CANopen application layer -toteutuksen NMT-viestit, jotka lähetetään tietyin väliajoin ja jotka paljastavat laitteen operatiivisen tilatiedon väylään. [9.]

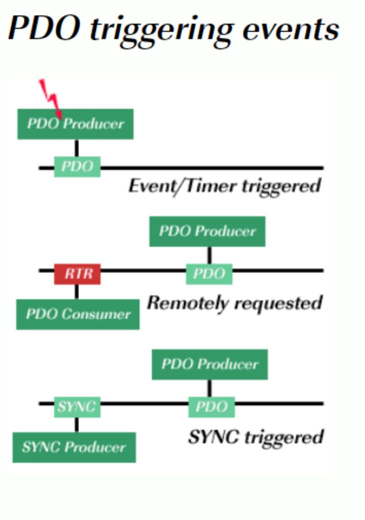
2.4 CANopen

CANopen on korkeamman tason protokolla, eli se on application layer -kerroksen protokolla OSI-mallissa. CANopen esiteltiin marraskuussa 1994 CiA:n esiteltyä ensimmäisen version CANopen-prokollan spesifikaatiosta. Spesifikaatio alkuperäinen nimi oli “CAL-based communication profile for industrial systems”, ja se kehitettiin Espritin (European Strategic Program on Research in Information Technology) tutkimusohjelman alla tutkimusprojektissa nimeltä ASPIC (Automation and Control Systems for Production Units using an Installation Bus Concept). Tutkimuksen päämääränä oli kehittää arkkitehtuuri ja laitteet mahdollistamaan olemassa olevien valmistussolujen modulaarinen ja joustava yhdistäminen. Tohtori Mohammed Farsi (University of New Castle) johti tutkimusta, ja hän yhdessä Stefan Reitmeierin (Bosch) päättivät käyttää CAN application layer (CAL) -protokollaa pohjana. CAL oli CiA:n kehittämä ja puhtaasti OSI-mallin 7. kerroksen protokolla eli application layer. Tavoitteena olikin kehittää joustavampi application layer -protokolla, joka olisi helppo implementoida. Jo ensimmäisessä versiossa määriteltiin Process Data Objectit (PDO) ja Service Data Objectit, samoin kuin PDO-viestin synkroninen lähetys ja Network management- (NMT) sekä Emergency messages -viestit (EMCY). CANopenista tulikin yksi onnistuneimmista Espritin tutkimusprojekteista, ja se julkaistiinkin CIA 301-spesifikaationa. Heti alussa useat yritykset implementoivat protokollan korkeamman tason protokollaksi sovelluksiinsa. Useampia iteraatioita protokollan spesifikaatioon kuitenkin tarvittiin ennen kuin ensimmäinen vakaa versio voitiin julkaista. Ensimmäinen versio, jota voitiin käyttää tuotteissa ja järjestelmissä oli 3.0, ja se on edelleen joissakin sovellutuksissa käytössä. CANopen-protokolla tarjoaa joustavat konfiguraatiomahdollisuudet ja erilaiset viestityypit. CANopen-protokollaan kuuluu useita eri viestityyppejä, joita ovat muun muassa Process Data Objects, Service Data Object, Network Management ja Emergency Objects. Nämä viestityypit käydään seuraavassa lävitse. [10; 11; 12.]

Process Data Objects

Process Data Objects (lyh. PDO) on prosessidatan reaaliaikaiseen tiedonsiirtoon tarkoitettu kommunikointimuoto CANopen-standardissa. PDO-kommunikoinnin datakehys muodostuu yhteensä kahdeksasta tavusta, joten yksi PDO-viesti voi sisältää huomattavasti enemmän dataa kuin myöhempänä esitellyn Service Data Objects -viesti.

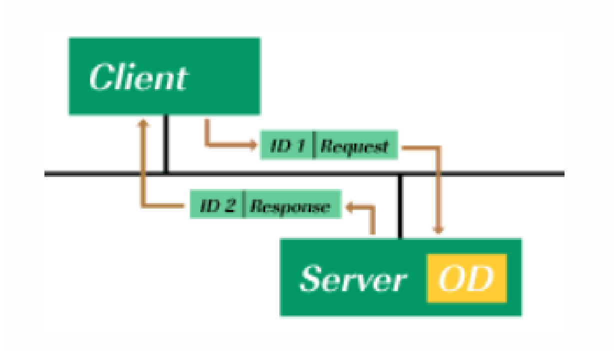
PDO-kommunikointi voi olla tapahtumapohjaista, kyselypohjaista, syklistä tai asyklistä (kuva 4). Tapahtumapohjaisessa kommunikoinnissa laitteen Object Dictionaryyn (lyh. OD) on määritelty tietty tapahtuma, joka laukaisee viestin lähetyksen. Tapahtuma voi olla esimerkiksi lämpötila-arvon nouseminen tietyn rajan yli, jonka takia viestin lähetys tapahtuu. Kyselypohjaisessa kommunikoinnissa väylän master-laite lähettää Remote Transmission Request -viestin (RTR), johon slave-laite vastaa PDO-viestillä. Syklisessä kommunikaatiossa PDO-viestin lähetys on kytketty yhteen Synchronization Object -viestin (SYNC) saapumisen kanssa, eli laite lähettää PDO-viestin aina SYNC-viestin saapuessa. Asyklisessä tiedonsiirrossa laite lähettää viestin vasta tietyn tapahtuman sattuessa. Tapahtumapohjainen viestintä vastaa asyklistä, mutta viestiä ei lähetetä heti tapahtuman sattuessa, vaan PDO-viesti lähetetään vasta SYNC-viestin saapuessa. [13; 14.]



Kuva 4. Process Data Objects-liipaisun malli. [14.]

Service Data Objects

Service Data Objects mahdollistaa pääsyn laitteen OD-parametreihin (Object Dictionary). SDO-viestin tarkoitus onkin joko lukea tai kirjoittaa yhteen Object Dictionaryn osoitteeseen kerrallaan. SDO-viestintä on aina kaksisuuntaista: toinen osapuoli lähettää joko kirjoitus- tai lukupyynnön, jonka toinen osapuoli vahvistaa joko lähettämällä pyydetyn datan tai kirjoittamalla Object Dictionaryyn halutun datan (kuva 5). SDO-viestintä on siis kyselypohjaista, eikä se sisällä syklistä viestintää. [15;16.]



Kuva 5. Service Data Objects-kommunikoinnin malli. [15.]

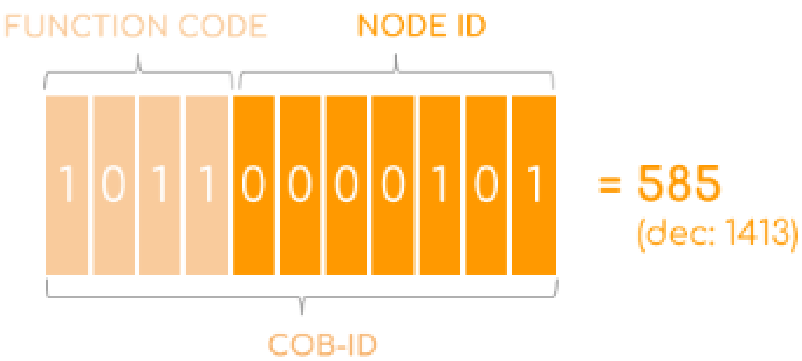
Laitteiden parametrisointi Pre-Operational -tilassa tehdään SDO-kommunikoinnilla laitteen Object Dictionaryyn. Jos SDO-viestinnässä tapahtuu virhe joko asiakkaan (engl.

client) tai palvelimen (engl. server) kommunikaatiossa, lähettää virheen havainnut osapuoli SDO Abort -viestin, joka keskeyttää SDO-kommunikoinnin. [15;16.]

Communication Object Identifier

Communication Object Identifier (COB-ID) koostuu kahdesta osasta, jotka ovat laitteen ID-numero sekä viestin tyypin kertova funktiokoodi (kuva 6). Laitteen ID-numero koostuu seitsemästä bitistä. ID-numero on laitekohtainen ja spesifioi viestin tiettyyn laitteeseen. Funktiokoodista selviää viestin tyyppi, joita on tyypiltään useita erilaisia ja jokaiselle on annettu oma funktiokoodi. [10.]

	COMMUNICATION OBJECT	FUNCTION CODE (4 bit, bin)	NODE IDs (7 bit, bin)	COB-IDs (hex)	COB-IDs (dec)	#
1	NMT	0000	0000000	0	0	1
2	SYNC	0001	0000000	80	128	1
3	EMCY	0001	0000001-1111111	81 - FF	129 - 255	127
4	TIME	0010	0000000	100	256	1
5	Transmit PDO 1	0011	0000001-1111111	181 - 1FF	385 - 511	127
	Receive PDO 1	0100	0000001-1111111	201 - 27F	513 - 639	127
	Transmit PDO 2	0101	0000001-1111111	281 - 2FF	641 - 767	127
	Receive PDO 2	0110	0000001-1111111	301 - 37F	769 - 895	127
	Transmit PDO 3	0110	0000001-1111111	381 - 3FF	897 - 1023	127
	Receive PDO 3	1000	0000001-1111111	401 - 47F	1025 - 1151	127
	Transmit PDO 4	1001	0000001-1111111	481 - 4FF	1153 - 1279	127
	Receive PDO 4	1110	0000001-1111111	501 - 57F	1281 - 1407	127
6	Transmit SDO	1011	0000001-1111111	581 - 5FF	1409 - 1535	127
	Receive SDO	1100	0000001-1111111	601 - 67F	1537 - 1693	127
7	HEARTBEAT	1110	0000001-1111111	701 - 77F	1793 - 1919	127



Kuva 6. COB-ID:n muodostuminen. [10.]

Electronic Data Sheet -tiedosto

Electronic Data Sheet (EDS) -tiedosto sisältää laitekohtaiset Device Object -parametrit ihmisystävällisessä muodossa. Se toimii pohjana laitteen Object Dictionarylle ja mahdollistaa erilaisten konfiguraatio- tai kehitystyökalujen käyttämisen laitteen hallintaan. [10.]

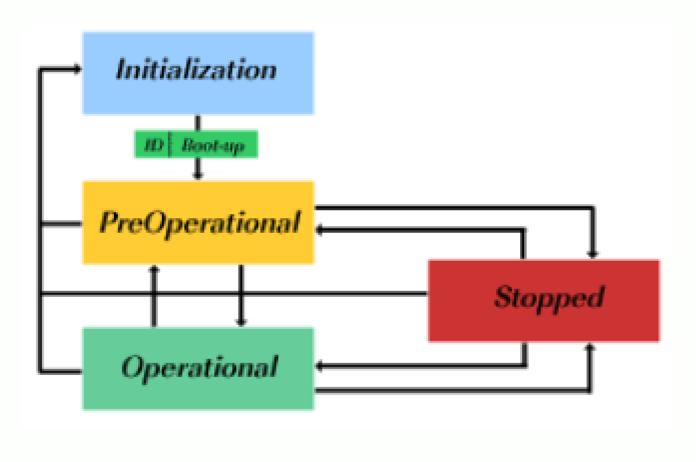
Network Management

Network Management eli NMT on protokolla, jonka avulla laite voi viestiä ajantasaisen tilatietonsa väylän laitteille. Tiloja on neljä: stopped, initialization, pre-operating ja operating (kuva 7). Nämä tilat kertovat väylän laitteille varsinaisen tilatiedon lisäksi sen, mitä kommunikointia voidaan sillä hetkellä käyttää. stopped-tilassa master pystyy kommunikoimaan laitteen kanssa vain NMT-viesteillä, eli pyytää sitä siirtymään uuteen tilaan.

Initialization-tilassa laite alustaa itsensä alkuperäisille parametreille tai asetetuille power-on-parametreille. Initialization-tilassa on kolme vaihetta: initializing, reset application ja reset communication. Objekti-kirjaston parametrien alustus tapahtuu reset application -vaiheessa. reset communication -vaiheessa alustetaan kommunikointi ja tämän jälkeen siirrytään automaattisesti pre-operation-tilaan.

Pre-operation-tilassa NMT-viestien lisäksi voidaan kommunikoida SDO-viesteillä, eli parametrisoida laite masterin avulla käyttäen asynkronista tiedonsiirtoa. Tässä vaiheessa laite myös lähettää boot-up-viestin, jolla se indikoi laitteen siirtymistä pre-operation-tilaan. Tässä tilassa voidaan lähettää myös SYNC-, heart-beat- ja time stamp -viestejä, jos viestit ovat konfiguroitu oikein ja tuettuja. Tämän jälkeen laite siirtyy operational-tilaan.

Operational-tilassa laitteen kanssa voidaan kommunikoida myös synkronisilla PDO-viesteillä, ja laite on normaalissa käyttötilassaan. Tästä tilasta voidaan siirtyä mihin tahansa muuhun tilaan joko slave- tai master-laitteen toimesta. [17.]



Kuva 7. Network management-tilakaavio. [17.]

Emergency Object

Emergency Object (EMCY) -viesti on laitteen lähettämä diagnostiikkatieto, joka ilmoittaa, jos laite on vikatilassa. EMCY-viestin lähetys perustuu tuottaja-/kuluttajamalliin, missä tietyt laitteet voivat tuottaa EMCY-viestin, toisten ollessa sen kuluttajia. EMCY-viestiin kuuluu 16-bittinen vikakoodi, jota seuraa 8-bittinen vikarekisteri ja lopuksi viisitavuinen viesti, jonka sisältö riippuu kulloisenkin laitevalmistajan laiteprofiilista (kuva 8). EMCY-viestin Communication Object Identifier (COB-ID) muodostuu osoitteesta 0x080, johon lisätään laitteen osoitenumero heksadesimaalimuotoisena. Tätä ei pidä sekoittaa SYNC-viestin COB-ID-numeroon, joka on kiinteä 0x080 kaikille laitteille. [18; 19.]

Vikarekisteri on muuttuja, joka on listattu laitteen Object Dictionaryssä ja se kontrolloi laitteen vikatilakonetta. Laitteen havaitessa vikatilanteen vikatilakone menee Error-tilaan ja tuottaa EMCY-viestin. Jos tässä tilassa tulee uusi vika, vikatilakone pysyy vikatilassa, kunnes kaikki viat ovat poistuneet. [18; 19.]

Can header	rtr	len	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x080 + node	0	8	Error Code		Error Register	Vendor specific data				

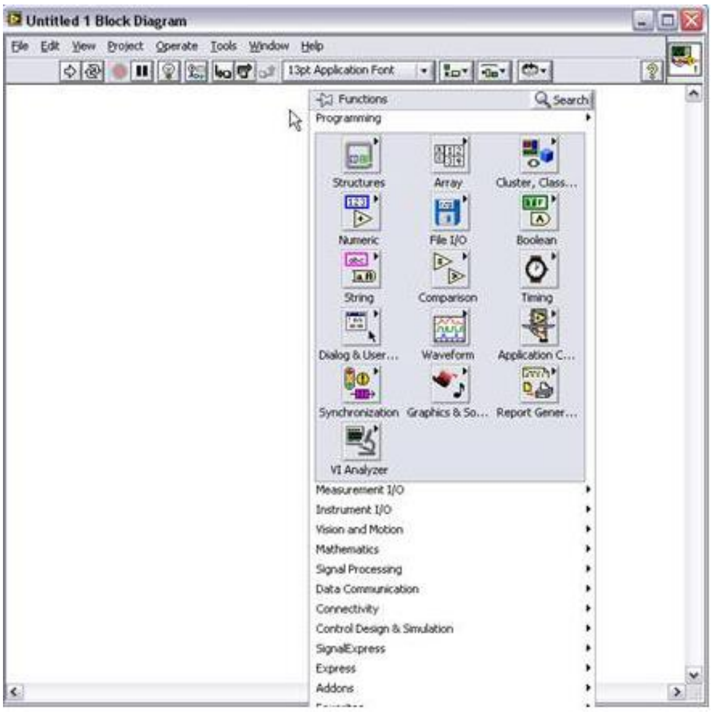
Kuva 8. EMCY-viestin rakenne. [18.]

Standardivikakoodit on määritelty standardeissa CiA DS-301 [20.] ja DS-401 [21.]. Näitä ovat esimerkiksi lämpötila-, jännite-, virta-, kommunikointi- ja yleisvikakoodit. Tarkemmin vikakoodit ovat löydettävissä edellä mainituista standardeista.

3 LabVIEW

LabVIEW on lyhennetty muoto kokonimestä Laboratory Virtual Instrument Engineering Workbench. LabVIEW on graafinen ohjelmointiympäristö, joka pohjautuu G-kieleen. Sitä ei kuitenkaan pidä sekoittaa koneohjelmoinnissa käytettyyn G-koodiin, sillä kyseessä ovat erilaisiin käyttötarkoituksiin luodut ja käytetyt kielet. LabVIEW'n on kehittänyt National Instruments. Ohjelmointiympäristö juontaa juurensa vuoteen 1986, jolloin ryhmä insinöörejä ja tieteilijöitä alkoi kehittää tehokasta tapaa tehdä automaattisia mittauksia, analysointeja ja testejä käyttämällä hyväkseen tietokoneohjelmistoja. Kehityksen tuloksena syntyi LabVIEW'n G-kieli, joka graafisena kielenä on hyvin korkean tason kieli, mutta käännettynä konekielelle sen suoritus on tehokasta. Kielen graafisuus tekee ohjelmoinnin aloittamisesta helppoa. Ohjelmoinnissa pitää kuitenkin kiinnittää huomiota ohjelmien selkeyteen ja perussääntönä ohjelman rakenne luetaan vasemmalta oikealle, sekä ylhäältä alas. Tämän lisäksi ohjelmien olisi hyvä mahtua yhdelle ruudulle, eikä ohjelmointilohko saisi peittää johtimia alleen. Jokaisessa ohjelmassa on oltava käyttöliittymä eli etupaneeli, sekä itse ohjelmaosuus, jota kutsutaan diagrammiksi. LabVIEW'ta käytettiin tässä työssä itse ohjelman tekemiseen, sisältäen käyttöliittymän ja rajapinnat. [22.]

Ohjelmointi LabVIEW'ssa tehdään diagrammissa, johon ohjelma rakennetaan. Uutta ohjelmaa aloittaessa diagrammi on tyhjä sivu, johon on mahdollista tuoda funktiovalikosta (kuva 9) haluttuja toimintoja tai luoda työkaluvalikon (kuva 10) avulla esimerkiksi tekstiä tai uuden johtimen, jolla ohjelman osia voidaan yhdistää toisiinsa. [28.]

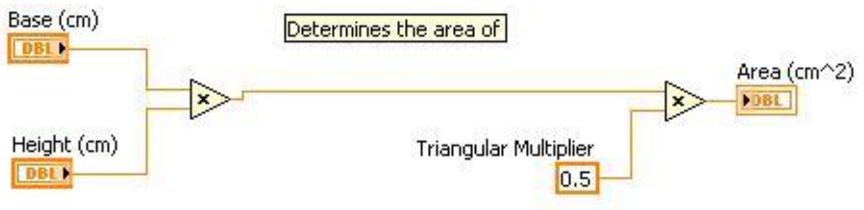


Kuva 9. Tyhjä diagrammi ja funktiovalikko. [28.]

Funktiovalikosta on mahdollisuus luoda erilaisia struktuureja, kuten case-lauseke, while-silmukka tai event-struktuuri. Myös numeerisia lohkoja on mahdollista lisätä, ne sisältävät muun muassa eri datatyyppien lohkoja, matemaattisia funktioita tai erilaisia datan muuntolohkoja. Funktiovalikosta löytyy myös lohkot vertailujen tekemiseen, Boolen operaattorit, ajastimien luomiseen, tekstisyötteiden käsittelyyn, taulukoiden luomiseen sekä tiedostojen käsittelyyn. Funktiovalikosta diagrammiin lohkon lisääminen tapahtuu vetämällä valikosta haluttu lohko oikeaan kohtaan diagrammia. Tämän jälkeen johdotustyökalulla se voidaan liittää muihin lohkoihin, tai käsittelytyökalulla siirtää lohkoa uuteen paikkaan. [28.]

Kuva 10. Työkaluvalikko diagrammissa. [28.]

Esimerkiksi kuvassa 11 lasketaan kolmion pinta-ala. Vasempaan laitaan diagrammia on lisätty kaksi DBL-lohkoa nimeltään Base (cm) ja Height (cm). DBL on lyhenne sanasta double, joka tarkoittaa muuttujan muotoa. LabVIEW'ssa double-muuttuja on floating-point muuttuja, jonka koko on 64 bittiä. Muuttujat on johdotettu yhteen kertolaskufunktioon, joka kertoo muuttujien kenttiin syötetyt luvut keskenään. Tästä saatu tulo olisi neliön pinta-ala, joten se on johdotettu seuraavaan kertolaskufunktioon, joka kertoo tulon kertoimella 0,5. Tästä kertolaskusta saadaan kolmion pinta-ala, joka viedään DBL-muuttuja lohkoon, jonka nimi on Area (cm^2). Koska ohjelma ei ole esimerkiksi while-silmukan sisällä, loppuu ohjelman suoritus tämän jälkeen.



Kuva 11. Kolmion pinta-alan laskeminen LabVIEW'ssa. [28.]

Kuvassa 12 on ohjelman etupaneeli, eli käyttöliittymä. Käyttöliittymässä näkyy DBL-muuttujien tekstikentät, mutta ei muuta. Height (cm)- ja Base (cm) -kentät ovat vapaasti muokattavissa, mutta Area (cm^2) -kenttä on vain luku -tilassa. Syötettäessä halutut luvut Height (cm)- ja Base (cm) -kenttiin ja painettaessa ohjelman yläpalkista run-painiketta, laskee ohjelma kolmion pinta-alan Area (cm^2) -kenttään.

Kuva 12. Etupaneeli kolmion pinta-alan laskemiseksi. [28.]

4 Komentosiltalaitteet

Komentosilta on laivan ohjaamo, ja se on laivan ohjaus- ja komentopaikka. Komentosillalla on laitteita navigointia ja laivan ohjailua varten. Nykyaikaisella komentosillalla ohjauslaitteet on liitetty kenttäväyliin ja väylät on yhdistetty automaatiojärjestelmiin. Kenttäväylästandardeja on useita, mutta tässä työssä käytetyt laitteet toimivat CAN-väylässä käyttäen CANopen-protokollaa ja työ perustuu tämän väylän ympärille luotuun kommunikointimalliin. Komentosillalla on käytössä useita eri tarpeisiin kehitettyjä laitteita, mutta tämän työn kannalta tärkeimmät ovat azimuth-kahva, miniwheel ja nappipaneeli. Laitteiden tarkoituksena on mahdollistaa aluksen operointi missä tahansa tilanteessa ja helpottaa tilanteiden havainnointia.

Aluksissa ruori on usein korvattu erilaisilla ohjainlaiteratkaisuilla, joiden toimintaperiaate voi hieman erota toisistaan. Eri ohjainlaitetyyppejä käytetään usein eri ohjaustilanteissa, niin kutsutuissa ohjausmoodeissa. Tällaisia ohjausmoodeja voi olla esimerkiksi manouvering, open sea ja dynamic position (dp). Laitteiden vikasietoisuuden täytyy olla hyvä ja kriittisten toiminnollisuuksien pitää olla varmistettuja. Tämän takia väyläratkaisussa väylät ovat usein redundanttisia, näitä kutsutaan master- ja backup-väyliksi. Toisen väylän vikaantuessa vaihdetaan automaattisesti toimivalle väylälle, ja tämän pitää olla huomioitu myös tehdyssä komentosiltalaitesimulaattorissa. [29.]

Marine and Ports -yksiköllä on Vuosaaressa kaksi laivasimulaattoria, jotka ovat tarkoitettu asiakastapaamisiin, koulutuksiin ja tuotekehityksen käyttöön (kuva 13). Simulaattoreiden toiminnallisuus vastaa oikeata laivaa, ja ne sisältävät vastaavat ohjainlaitteet kuin laivoissa käytetään.

Kuva 13. Marine and Ports -yksikön laivasimulaattori Vuosaaressa.

Azimuth-kahva

Azimuth-kahvaa käytetään ABB:n Azipod-ruoripotkurin ohjauslaitteena (kuva 14.). Kahva on ympäripyörivä, ja siinä on erillinen vipu kierroslukuohjearvon antoa varten. Täten se sisältää kaksi akselia, joista kummatkin ovat sähköakseloituja. Akseleita voidaan ohjata slave-tilassa ulkopuolisella ohjearvolla, tällöin slave-tilassa olevat kahvat seuraavat master-kahvan ohjearvoa, jolloin kaikki kahvat ovat aina lähes samassa asennossa. Tämä mahdollistaa komentopaikan vaihdon, ilman suuria hyppäyksiä ohjearvoissa. Master-tilassa kirjoitetaan väylään akselin antureiden ohjearvosignaalia. Tämän lisäksi akseleille voidaan antaa sähköisiä tuntopisteitä (engl. detent).

Kahva liitetään CAN-väylään, ja sen ylemmän tason protokolla on CANopen. Kahvassa on painetut asteikot kummallekin akselille, kulmaohjearvoon asteikko on alueella +/-180 astetta ja kierroslukuohjearvon asteikko +/-10. Kierroslukuohjearvon asteikosta saadaan prosenttiohjearvo kertomalla se kymmenellä. Kulmaohjearvon asteikossa käytetään punaista väriä port-puolella (vasemmalla) ja vihreätä starboard-puolella (oikealla).

Kuva 14. Azimuth lever. [23.]

Miniwheel

Miniwheel on kiekkomainen ohjainlaite, jota käytetään ohjauslaitteena pääasiassa avomerellä väylällä ajettaessa (kuva 15). Sillä ei ole mahdollista antaa kierroslukuohjearvoa, vaan se on tarkoitettu ainoastaan kulmaohjearvon antoon. Normaalisti sillä pystyy operoimaan kulmalukemien -35 astetta ja 35 astetta välillä. Ohjainlaitteen mekaaninen liikerata on kuitenkin rajattu +/-60 asteeseen, asteikon ollessa +/-35 astetta. Tällä ratkaisulla saadaan suurempi liikerata ohjainlaitteelle ja täten tarkempi ohjaus.

Kuva 15. Miniwheel-ohjauslaite.

Nappipaneeli

Nappipaneelilla vaihdetaan ohjausmoodeja ja ohjauspaikkoja (kuva 16). Sitä käytetään myös laitteistojen käynnistykseen ja vikojen kuittaukseen. Nappipaneelissa on painikkeita, joita voidaan ohjelmoida eri tarkoituksiin, ja yleensä siihen on myös integroitu merkkivaloja. Nappipaneeli on tyypillisesti liitetty CAN-väylään ja se konfiguroidaan CAN-väylän kautta.

Kuva 16. Nappipaneeli asennettuna konsoliin.

5 USB-CAN-adapteri

Kannettavan tietokoneen liittämiseksi automaatiojärjestelmän CAN-väylään tarvitaan adapteri. Työssä päädyttiin käyttämään USB-liityntään liitettävää adapteria, jonka avulla kommunikointi CAN-väylän kanssa on mahdollista. Tämän ansiosta simulaattoriohjelmiston käyttäminen tavallisella kannettavalla tietokoneella on mahdollista ja sen liittäminen automaatiojärjestelmään helppoa.

Työssä päädyttiin käyttämään Kvaser USBcan Light 4x HS USB-CAN-adapteria, sen tarjoaman hyvän liitettävyyden ja monipuolisen kirjaston takia. Kvaser tarjoaa valmiin LabVIEW-kirjaston CAN-kommunikointiin sekä tämän lisäksi CANlib-SDK:n. Valittu USBcan Light 4x HS (kuva 17) on kustannustehokas ratkaisu jopa neljän CAN-väylän kommunikointiin käyttäen vain yhtä USB-porttia. Fyysinen liityntä CAN-väyliin on mahdollista joko DB26-liitännän tai neljän DB9-liitännän kautta. Työssä käytettiin DB9-liityntöjä liityttäessä adapterista automaatiojärjestelmään niiden helpon kytkettävyyden ja saatavuuden vuoksi (kuva 18). Laite saa tarvitsemansa käyttötehon tietokoneen USB-portin kautta, joten tietokoneeseen liittämiseen tarvitaan vain USB-A-tyyppinen portti. USBcan Light 4xHS kykenee CAN-kommunikoinnissa nopeuksiin 50 kbit/s–1 Mbit/s, joten se soveltuu hyvin tarvittavalle125 kbit/s-nopeudelle. [24.]

Kuva 17. Kvaser USBcan Light 4x HS on monipuolinen USB-CAN-adapteri. [26.]

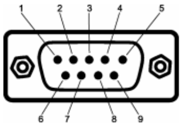


Figure 6: The male D-SUB 9 connector pin numbers

HD D-SUB	CAN 1	CAN 2	CAN 3	CAN 4	Function
1	2				CAN_L channel 1
2		2			CAN_L channel 2
3			2		CAN_L channel 3
4				2	CAN_L channel 4
5					Not connected
6					Not connected
7					Not connected
8	4				Not used
9					Not connected
10	9				Not used
11	7				CAN_H channel 1
12		7			CAN_H channel 2
13			7		CAN_H channel 3
14				7	CAN_H channel 4
15					Not connected
16					Not connected
17					Not connected
18		4			Not used
19	3				GND channel 1
20		3			GND channel 2
21			3		GND channel 3
22				3	GND channel 4
23					Not connected
24					Not connected
25					Not connected
26					Not connected

Table 6: Pin configuration of the 26-pin HD D-SUB (and HD26-4xDS9 splitter)

Kuva 18. DB9-liitynnän kaavio. [24.]

6 Työn eteneminen

Ohjelmiston kehittäminen aloitettiin määrittelemällä, mitä ominaisuuksia ja toiminnallisuuksia ohjelmistolta vaaditaan ja kuinka sitä on tarkoitus käyttää. Tämän jälkeen suunniteltiin ohjelmiston rajapintoja ja millaisilla laitteilla ohjelmiston toteutus onnistuu. Kaikkea ei tässä vaiheessa pystytty määrittelemään ja suunnittelemaan ennalta, vaan osittain ohjelmiston rakenne ja ominaisuudet muuttuivat toteutusta tehdessä. Ohjelmiston päätoiminnallisuudet ja laitteistovaatimukset pysyivät kuitenkin alussa määriteltyjen mukaisina ja muutokset koskivatkin pienempiä kokonaisuuksia. Seuraavaksi käydään läpi, kuinka määrittely-, suunnittelu- ja toteutusvaiheet etenivät.

6.1 Määrittely

Määrittelyvaiheessa selkeytettiin, mitä tarpeita projektille asetetaan ja kuinka projektia lähdetään viemään eteenpäin. Tässä vaiheessa projektin osapuolille piti saada yhteneväiset käsitykset siitä, mitä projektin lopputulos pitää sisällään ja kuinka laaja projekti tulee olemaan. Määrittelyvaiheessa todettiin tarpeen olevan kehitettyjen automaatio-ohjelmistojen testaus simulointiohjelmistolla, joka liittyy automaatiojärjestelmän CAN-väylään. Tämänkaltainen ohjelmisto säästäisi ohjelmistokehittäjiltä aikaa ja vaivaa, koska nykyisellä ratkaisulla ohjelmistojen testaus oli aikaa vievää ja usein tarvitsee fyysisten laitteiden uudelleenjärjestelyä ja niihin liittyvien automaatio-ohjelmistojen muokkaamista. Tämä johtaa usein uudelleenkaapeloinnin tarpeeseen ja laitteille asennuspaikkojen tekemiseen.

Useissa tapauksissa tarve on myös monen ohjelmistoversion samanaikaiseen testaamiseen, joten fyysisiä laitteistoja tarvittaisiin useita ja niiden ollessa suhteellisen monimutkaisia ja kalliita, tämä tuottaa tarpeettoman suuria kustannuksia ja tilantarpeet moninkertaistuisivat.

Tämän sijaan voitaisiin käyttää fyysisten laitteiden toimintaa jäljittelevää ohjelmistoa, joka kytkeytyisi samaan rautarajapintaan oikeiden laitteiden kanssa. Tällainen ohjelmisto voisi nopeuttaa ohjelmistoversioiden testausta ja sitä kautta parantaa ohjelmistojen laatua. Simulaattoriohjelmistolta vaadittaisiin helppoa liitettävyyttä ja sen pitäisi toimia pc-pohjaisilla kannettavilla tietokoneilla.

6.2 Suunnittelu

Suunnitteluvaihe aloitettiin miettimällä tulevan ohjelmiston rakennetta ja käyttäjien tarpeita. Tässä vaiheessa päädyttiin siihen, että tarvitaan graafinen käyttöliittymä, josta pitää olla helposti muokattavissa tarvittavien laitteiden määrä ja niiden laitekohtaiset asetukset. Ohjelmiston pitää olla myös myöhemmin helposti muokattavissa ja laajennettavissa eri laitteiden tarpeisiin.

Ohjelmiston toiminnallisuuksien täytyi olla vastaavat kuin aidolla komentosillalla ja sen piti olla automaatiojärjestelmän näkökulmasta vastaava kuin fyysinen laite. Sen lisäksi siitä piti olla helposti nähtävissä laitteiden asennot ja asentojen säädön täytyi olla vaivatonta. Eri laiteyhdistelmiä piti pystyä lisäämään käyttöliittymään kulloisenkin tilanteen mukaan ja näiden laiteasetusten täytyi olla säädettävissä. Ohjelmisto ei saanut olla raskas ja eri laitetilat piti olla nähtävissä helposti.

Tässä vaiheessa suunnittelua projektin ymmärrettiin olevan iso ja se rajattiin ohjelmiston toteutukseen sekä yhden laitteen mallinnukseen. Ensimmäinen lisättävä laite olisi Azimuth-kahva. Tämä on eniten käytetty laite komentosillalla ja toiminnoiltaan vaativin toteuttaa ohjelmallisesti.

Suunnittelussa oli otettava huomioon laitteiston siirrettävyys ja sen toivottiin toimivan tavallisella kannettavalla tietokoneella. Tämä rajasi ohjelmoitavat logiikat pois mahdollisuuksista. Mahdollisuudeksi jäi tietokoneapplikaatio ja sille rautarajapinnaksi USB-laite. Rautarajapinnan piti olla myös helposti siirrettävissä, joten siihen valittiin USB-CAN-adapteri. Adapterin piti tukea mahdollisimman montaa CAN-väylää samanaikaisesti.

Komentosillalla on useita eri laitteita, ja jokaiseen on kytketty kaksi eri CAN-väylää. Nämä ovat main- ja backup-väylät. Tämän ansiosta järjestelmän vikasietoisuus on hyvä ja laitteisto pysyy käyttökuntoisena yhden vian sattuessa kohdalle. Tämä myös nostaa väylämäärän nopeasti suureksi, vaikka osa laitteista onkin ketjutettuna väylässä. Tästä syystä käytetyn USB-CAN-adapterin valinnassa täytyy huomioida CAN-väylien määrä. Adapterin tulee mahdollistaa mahdollisimman monen CAN-väylän luominen yhden USB-portin kautta.

Rautarajapinnaksi oli mahdollisuutena valita monia erilaisia laitteita. Valinnan tekemiseen vaikuttavia tekijöitä olivat: ohjelmoinnin helppous, grafiikan luonnin mahdollisuus, ohjelmiston keveys, ohjelmiston muokattavuus jälkikäteen, ohjelmoinnin nopeus ja tekijän osaamisen taso. Ennen rautarajapinnan valintaa oli kuitenkin tehtävä päätös käytettävästä kehitysympäristöstä.

Vaihtoehtoina kehitysympäristön valintaan olivat Visual Studio, Qt sekä LabVIEW. Näistä Visual Studio ja Qt ovat perinteisiä kehitysympäristöjä, joissa käytetään ei-graafista ohjelmointikieltä. Ohjelmointi Qt:ssä tapahtuu C++- ja QML-kielillä ja Visual Studiossa C#:lla, jotka ovat kummatkin laajasti käytettyjä kieliä. C-pohjaiset kielet ovat kuitenkin vaativia kehittää ja tarkoituksena oli luoda ohjelmisto, joka olisi kenen tahansa muokattavissa. Tästä syystä valinta lopulta päätyi LabVIEW’n ja täten graafiseen ohjelmointiympäristöön. LabVIEW’ssä tehty ohjelma on helposti kaikkien ymmärrettävissä ja muokattavissa. LabVIEW’lle oli saatavilla myös Kvaserilta valmis kirjasto USB-CAN-adapterille. Nämä kaikki syyt puolsivat LabVIEW’n käyttöä projektissa, vaikka minulla ei siitä henkilökohtaisesti ollut aiempaa kokemusta toisin kuin muista ympäristöistä. Uuden ympäristön opettelu oli kuitenkin mielenkiintoista.

Kun käytettävä kehitysympäristö oli valittu, voitiin valita monista vaihtoehdoista myös rautarajapinta. Työssä päädyttiin Kvaserin USBscan light 4x HS-malliin. [26.] USBscanissa on neljä CAN-väylää, joista jokainen mahdollistaa jopa 1 Mbit/s nopeuden. Laite on myös edullinen, ja sillä on hyvä tuotetuki ja laajat kirjastot. Liitäntä tapahtuu USB type-A -liitännällä esimerkiksi kannettavaan tietokoneeseen. CAN-väylään liityntä tapahtuu 26-pinnisen D-Sub-liitynnän kautta, jonka mukana toimitetaan myös adapteri, joka muuttaa liitynnän neljään yhdeksänpinniseen D-SUB-liittimeen. Ohjelmistoa laitteen mukana ei toimiteta, mutta Kvaserin sivuilta on ladattavissa ilmainen CanKing-ohjelmisto, jolla laitetta voidaan käyttää CAN-sanomien tiedonkeruuseen.

6.3 Toteutus

Työ aloitettiin testaamalla Kvaserin USBscan Light 4 x HS USB-CAN -adapteria. Testiin käytettiin Kvaserin CanKing-ohjelmistoa, jolla voi monitoroida CAN-väylässä liikkuvaa dataa. CAN-liikennettä väylään simuloitiin ABB:n AC500 ohjelmoitavalla logiikalla. PLC-ohjelmaksi valittiin komentosiltajärjestelmän laitekommunikointiin tarkoitettu ohjelma, joka toimii rajapintana muun järjestelmän ja laitteiden välillä. Tässä vaiheessa järjestelmään ei vielä kytketty laitteita kiinni, ainoastaan monitoroitiin dataa ja testattiin adapterin toiminnallisuuksia.

Tämän jälkeen tutustuttiin CANopen-standardin PDO-viestin rakenteeseen. PDO-viesti on syklinen viesti ja sisältää yleensä prosessidataa. Tämän viestin ja PLC-ohjelman rakenteen tutkimisella oli tarkoitus saada selville tarkka datamalli, jonkalaista voitaisiin toteuttaa LabVIEW-ohjelmoinnilla. PDO-kommunikointiin löytyy paljon dokumentteja, joten tämä tehtiin lähinnä oman osaamisen varmistamiseksi.

Aluksi päätettiin luoda syklinen datansiirto PDO-kommunikoinnin avulla LabVIEW'llä, jotta voitaisiin varmistua rajapintojen ja konseptikokonaisuuden toimivuudesta.

LabVIEW'ssä käytettiin ohjelmistorajapintana adapterille Kvaserin CANlib SDK:ta ja LabVIEW-kirjastoa. LabVIEW-kirjastoon oli toteutettuna CAN-kommunikoinnin perusohjelmat, muun muassa väylästä luku, kirjoitus sekä väylän avaus ja sulku. Aluksi testattiin väylän alustusta oikeille kommunikointiparametreille. Tätä testattiin alustamalla väylä ja sen jälkeen yrittämällä lukea CAN Read-aliohjelmalla väylässä liikkuvaa kommunikaatiota.

Väylän lukeminen onnistui hyvin valitusta COB-ID:stä. Nyt voitiin olla varmoja adapterin, SDK:n, LabVIEW-kirjaston ja ohjelman yhteensopivuudesta. Tämän jälkeen oli selvitettävä Azimuth-kahvan manuaalista ja käytetystä logiikkaohjelmasta oikeat COB-ID:t halutuille parametreille.

Tässä vaiheessa kuitenkin keskityttiin käyttöliittymäsuunnitteluun ja grafiikan luontiin. Valmiissa ohjelmistossa oli oltava selkeä käyttöliittymä, josta on helppo asettaa kommunikointiparametrit sekä ohjata laitteita vastaavasti kuin oikeita. Grafiikan pyörittämistä ja luontia testattiin LabVIEW'ssä. Tämä osoittautui alkuun melko haastavaksi, sillä LabVIEW'ssä ei ole suoraan toimintoa, jolla voisi pyörittää kuvaa akselinsa ympäri. Tähän löytyi kuitenkin aliohjelma, joka käänsi jpg-formaatissa olevan kuvan bittikartaksi ja muokkasi bittikarttaa halutun käännön aikaansaamiseksi. Tämän jälkeen kuva koostettiin taas jpg-kuvaksi ja näytettiin käyttöliittymässä. Tämä toimi kohtuullisen hyvin, mutta vei liikaa tehoja useata kuvaa pyöritettäessä. Tämä oli kuitenkin tarpeeksi hyvä ratkaisu, ja seuraavaksi palattiin tutkimaan CAN-väylän kommunikointia.

Työssä käytettiin CAN-väylää 125 kbit/s nopeudella, joten ohjelman puskurit täyttyivät nopeasti, ja tähän oli keksittävä ratkaisu. Ohjelmassa ei ole tarvetta koville päivitysnopeuksille, joten datavirrasta suodatettiin vain halutut COB-ID:t. Niitä luettiin vain tietyin väliajoin ja puskurit tyhjennettiin tämän jälkeen. Tämä auttoi jatkamaan työtä eteenpäin.

Tässä vaiheessa koossa oli PDO-kommunikoinnin luenta, mutta vielä ei pystytty lähettämään viestejä CAN-väylään. Onnistuneen PDO-viestinnän lukemisen jälkeen pystyttiin kuitenkin jatkamaan luomalla XControl-moduuli LabVIEW-ohjelmaan. Tämän moduulin tehtävänä on näyttää käyttöliittymässä laitteeseen liittyvät kuvat, liukusäätimet ja numeroarvot. Se myös sisältää laitteen toiminnallisuuden ja parametrit. Tämä moduuli on toiminnoiltaan vastaava kuin LabVIEW'n omat moduulit ja erilaiset mittarit, mutta erona on mahdollisuus luoda haluamansa käyttöliittymä ja laitteen toiminta kehitysympäristön valmiiksi tarjoamien mahdollisuuksien sijaan. Sisään tuleva data täytyy kuitenkin käsitellä ennen kuin se voidaan tuoda XControlille ja strukturoida datastruktuuriin. Tätä tarkoitusta varten tehtiin PDO-datastruktuuri, joka sisältää vastaavat parametrit kuin fyysisen laitteen PDO-kommunikoinnin datastruktuuri.

Luotua datastruktuuria on myös mahdollista käyttää useissa eri LabVIEW-lohkoissa, ja siihen voidaan lukea joko koko struktuuri tai vain sen tietty osa. PDO-struktuuri sisältää muun muassa halutun kulma- ja kierroslukuohjearvot, sekä tiedon kuuluisiko laitteen olla slave- vai master-tilassa. XControl oli vaativa ohjelmoida ohjelmakierrollisesti. Lopulta päädyttiin luomaan event-pohjainen aliohjelma, joka lukee tiettyjen muuttujien tilaa ja toimii tilan muutosten perusteella. Tätä varten osasta muuttujista piti luoda Val(Signl) -lohkot, ilman näitä lohkoja muuttujien arvojen muutos ei liipaissut eventiä. XControl-moduulista luetaan dataa ulos master-tilassa PDO-struktuurin muodossa. Tässä vaiheessa kasassa oli PDO:n luenta ja XControllin ohjaus slave-moodissa. Tämän perusteella kehitettiin PDO:n lähetyslohko, jonka toteutus on vastaavanlainen kuin PDO:n lähetys, mutta käänteisessä järjestyksessä.

PDO:n lähetyslohko liitettiin alkuperäiseen ohjelmaan ja toimintoja testattiin lähettämällä master-tilassa viestejä, joka onnistuikin muutaman ohjelman iteraatiokierroksen jälkeen. Ohjelmaan oli vielä luotava SDO- ja NMT-kommunikointi sekä backup-toiminnot NMT:lle ja PDO:lle. Pelkästään toimiva kommunikaatio ei riittäisi, vaan pitäisi keksiä tapa luoda useita instansseja samasta aliohjelmasta käyttöliittymän kautta ja refaktoroida ohjelman koodia.

SDO-kommunikaatiolle luotiin aluksi SDO-datastruktuuri, johon SDO-data luettaisiin. SDO-kommunikaatioon ei myöskään riitä pelkästään toimiva datan luenta, vaan saapunut viesti pitää kuitata takaisin master-laitteelle. Jos viestiä ei kuitata, lopettaa master-laite uusien viestien lähettämisen ja SDO-kommunikaatio menee virhetilaan. Tämän toteutusta varten perehdyttiin SDO-kommunikointiin, josta kerrotaan lisää kappaleessa 2.4.2.

Riittävien tietojen keräämisen jälkeen testattiin, kuinka kommunikointi toimii oikealla Azimuth-kahvalla lukemalla SDO-viestejä Tke:n CanTracer-ohjelmalla, käyttäen Kvaserin USB-CAN -adapteria. Suunnittelun jälkeen toteutettiin SDO-lohkoa kappaleessa 2.4.2 kuvatulla tavalla. Aluksi lähetetyt kuittausviestit väylään eivät olleet oikeanmallisia ja tämä aiheutti kommunikointiin vikatilan.

Ongelman löydyttyä tarkastettiin viestin malli oikean kahvan kommunikaatiosta ja se korjattiin ohjelmaan oikean tyyppiseksi. SDO-kommunikaation datat luettiin SDO-struktuuriin, mutta ne piti myös näyttää käyttäjälle jollakin tavalla. Alkuvaiheessa luotiin erillinen SDO popup -sivu, joka sisältää taulukon SDO-struktuurin arvoista. Tämä ei kuitenkaan ole kovinkaan intuitiivinen tapa esittää esimerkiksi tuntopisteiden paikkoja, joten käyttöliittymään luotiin tuntopisteiden kohdille mustat pallot, jotka esittävät tuntopisteiden kohtia oikeassa laitteessa. Tuntopisteiden paikat päivittyvät käyttöliittymässä SDO-struktuurin muutoksien mukaisesti. NMT-kommunikaatio oli ainoa puuttuva lohko backup-toiminnallisuuksien lisäksi.

NMT-lohkossa lähetetään laitteen tilasta kertova viesti väylään. Viestin lähetys tapahtuu syklisesti ajastimen perusteella. Laitteen tilatieto saadaan laitteen pääohjelmassa olevasta enum-tyyppisestä muuttujasta, joka on simuloidun laitteen tapauksessa asetettu staattisesti Operational-tilaan niin kauan kuin päälooppia suoritetaan. Lohko on muuten varsin yksinkertainen eikä tämän toteutuksessa ollut ongelmia, tosin lohkoon voisi myöhemmässä kehityksessä lisätä eri tiloja. PDO:n ja NMT:n backup-toiminnallisuudet luotiin tämän jälkeen. Toiminnallisesti nämä ovat vastaavat kuin main-väylässä, erot löytyvät käsiteltävän datan määrästä, koska kaikkia bittejä ei käytetä backup-toiminnoissa ja laitteiden laitenumeroissa. Laitenumerot eroavat main- ja backup-väylillä, joten ne täytyy asettaa erikseen backup-toimintoihin.

Backup-toiminnallisuuksien luonti olikin yksinkertaista jo toimivien main-toiminnollisuuksien takia ja ne saatiin kopioitua lähes suoraan. Ohjelman kaikki lohkot olivat tässä vaiheessa valmiita, ja niitä päästiin testaamaan yhdessä. Pääohjelmasta tehtiin sekvenssityyppinen, jossa alussa alustetaan ohjelma, jonka jälkeen avataan väylät ja siirrytään pääohjelman suorittamiseen. Pääohjelman suorittamisen lopuksi avatut väylät suljetaan. Tämänkaltainen ohjelmarakenne oli luontevaa rakentaa sekvenssityyppiseksi, mutta ohjelman olisi voinut rakentaa myös tapahtumapohjaiseksi. Sekvenssiohjelmaan päädyttiin sen todennäköisesti helpomman ylläpidettävyyden ja luettavuuden vuoksi.

Aliohjelmassa oli yhteinen XControl azimuth- ja thrust-akseleille, tässä vaiheessa ne päädyttiin eriyttämään, koska tämä ei mahdollistanut toisen akselin oloa master-tilassa, toisen ollessa slave-tilassa. Tämänkaltainen toiminnallisuus on mahdollista fyysisellä kahvalla, joskin harvoin käytetty. Asia oli kuitenkin korjattava, koska silloin tällöin se on tarpeellinen. Thrust-akselille luotiin oma XControl, johon eriytettiin thrust-akselin toiminnallisuudet. Perustoiminnoiltaan tämä XControl on vastaava kuin azimuth-akseli.

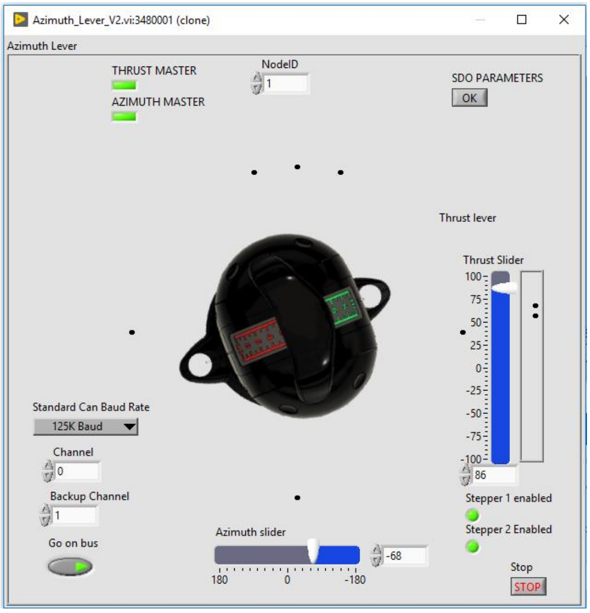
Tämä kuitenkin vaati aliohjelmaan uuden kutsun XControllille ja uudet muuttujien linkitykset PDO-struktuuriin. Kun thrust-akseli oli erotettu azimuth-akselista, oli vielä tehtävä pääohjelma, joka kutsuisi aliohjelmaa niin monta kertaa kuin järjestelmässä oli oltava Azimuth-kahvoja. Tätä varten ohjelmalle piti luoda pääohjelma ja sille käyttöliittymäsivu. Käyttöliittymäsivulle laitettiin kahvaa esittävä kuvake ja sen alle painike, josta voi luoda uuden instanssin azimuth-kahvan aliohjelmasta. Painiketta painamalla luodaan aina uusi instanssi. Teoriassa aliohjelmien määrälle ei ole rajaa, mutta käytännössä käytettävän tietokoneen suorituskyky määrää aliohjelmien enimmäismäärän. Toimivan pääohjelman jälkeen oli ohjelmisto tältä osin valmis ja seuraavaksi siirryttiin testausvaiheeseen.

6.4 Testaus

Ohjelmistoa testattiin tuotekehityslaboratorion automaatiojärjestelmillä ja sen toimintaa verrattiin oikeiden laitteiden käyttäytymiseen. Testaus tapahtui kytkeytymällä tuotekehityslaboratorion laivasimulaattorin CAN-väylään oikeiden laitteiden tilalle. Simulaattoriohjelmiston käyttäytymistä testattiin vertaamalla sitä oikeiden laitteiden käyttäytymiseen ja CAN-liikenteeseen. Tällä saatiin varmistettua ohjelmiston toiminta ja sen yhteensopivuus oikeisiin laitteisiin ja niiden kommunikaatioväyliin. Ohjelmiston kuormittavuutta testattiin myös luomalla aliohjelmasta samanaikaisesti kymmenen instanssia. Instanssien suuri määrä ei vaikuttanut modernilla kannettavalla tietokoneella negatiivisesti suorituskykyyn.

7 Ohjelman rakenne

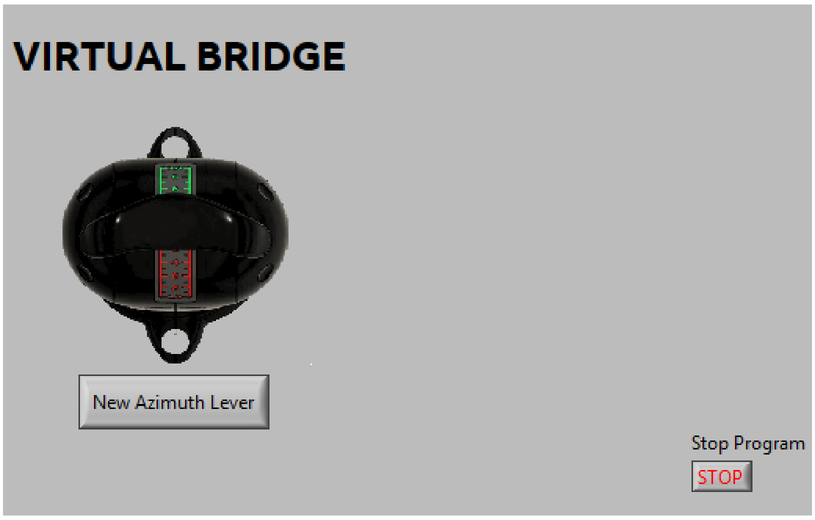
Ohjelma koostuu pääohjelmasta ja eri laitteita simuloivista aliohjelmista. Insinöörityön laajuuteen kuului luoda yksi aliohjelma simuloimaan Azimuth-kahvan toiminnollisuuksia ja mahdollistaa uusien laitteiden lisäys ohjelmaan helposti. Tarkoitusta varten toteutettiin pääohjelma, josta on mahdollista kutsua instansseja samasta tai eri aliohjelmasta useita kappaleita. Jokainen laitetta simuloiva aliohjelma toimii itsenäisesti sille asetetuin parametrein.



Kuva 19. Azimuth Lever -aliohjelma master-tilassa.

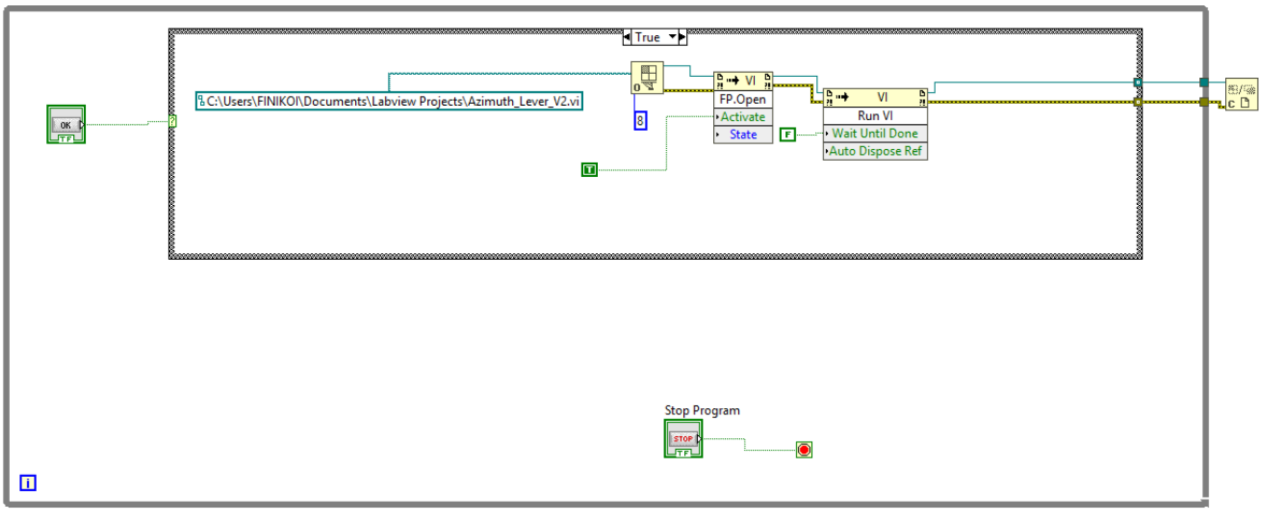
Ohjelma koostuu siis pääohjelmasta ja yhdestä aliohjelmasta nimeltään Azimuth lever. Aliohjelman ollessa aktiivisena ja liitettynä väylään nähdään kuvan 19. mukainen käyttöliittymäsivu. Käyttöliittymässä havainnollistetaan myös simuloidun kahvan olevan master- tai slave-tilassa, joka indikoidaan vasemman ylälaidan thrust master- ja azimuth master -merkkivaloilla. Master-tilassa akseleiden liukusäätimet ovat käytettävissä ja niistä voidaan antaa uusi ohjearvo kummallekin akselille erikseen. Käyttöliittymien toiminnallisuus on toteutettuna useilla pienemmillä aliohjelmilla, joita tässä kutsutaan myös lohkoiksi. Tässä luvussa käydään läpi ohjelman eri aliohjelmien rakenne ja toteutus.

7.1 Pääohjelma



Kuva 20. Pääohjelman käyttöliittymä.

Pääohjelman aloitussivulta (kuva 20) voi luoda uusia instansseja eri aliohjelmista. Painikkeesta voi luoda käytännössä rajattoman määrän instansseja aliohjelmasta, mutta käytännössä enimmäismäärä rajautuu käytettävissä olevan tietokoneen suorituskyvyn mukaan. Jokainen instanssi toimii kuten oikea fyysinen laite. Rajapinnat automaatiojärjestelmään ovat vastaavat kuin fyysisellä laitteella. Ohjelman oikeasta alalaidasta löytyy ohjelman pysäytyspainike, joka lopettaa ohjelman suorittamisen.

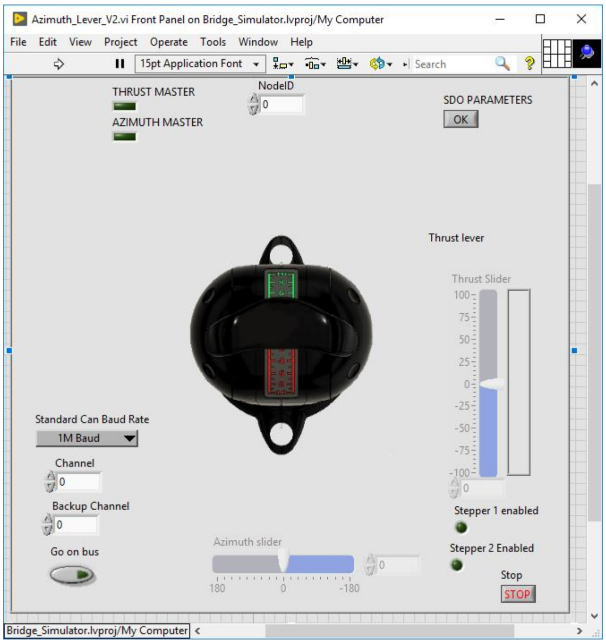


Kuva 21. Pääohjelman rakenne.

Pääohjelman varsinainen toteutus näkyy kuvassa 21. Painikkeen New Azimuth Lever painalluksen arvolla “true” käynnistetään uusi instanssi Azimuth_lever.V2.vi aliohjelmasta. Aliohjelma pyörii, kunnes painetaan Stop program -painiketta. Aliohjelman pysähtyessä suoritetaan mahdollisten virheilmoitusten käsittely ja suljetaan ohjelman kahvat (engl. handle).

7.2 Azimuth lever -aliohjelma

Aliohjelman käyttöliittymästä löytyy keskeltä Azimuth-kahvan kuvake (kuva 22). Kuvake kääntyy 360 astetta simuloiden oikean Azimuth-kahvan kääntymistä fyysisessä komentosillassa. Kulmaohjearvo indikoidaan myös alareunan Azimuth sliderissa sekä tekstikentässä sen vieressä. Kierroslukuohjearvon muutoksessa Azimuth-kahvan kuvake ei muutu, mutta ohjearvon muutos osoitetaan oikean laidan Thrust slider -liukusäätimessä sekä kentässä sen alapuolella.

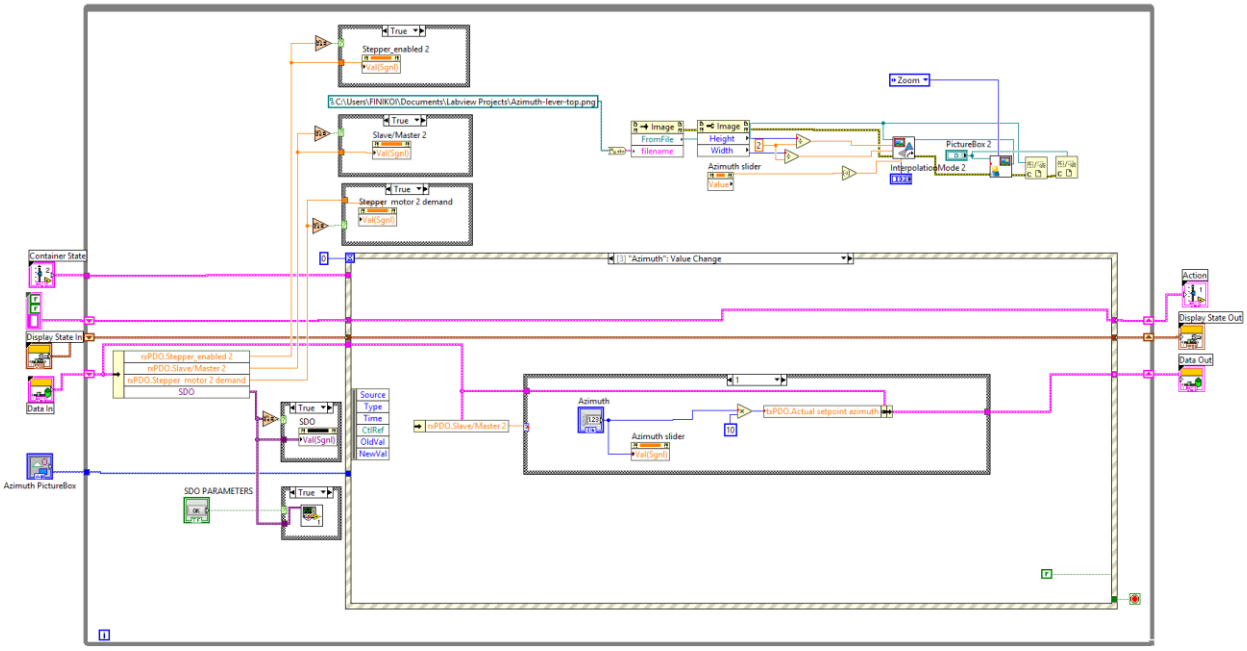


Kuva 22. Aliohjelman käyttöliittymä.

Ohjelman vasemmasta yläkulmasta löytyy Thrust master- ja Azimuth lever -merkkivalot, joilla osoitetaan, ollaanko kyseisen toiminnon master- vai slave-moodissa. Master-moodin ollessa valittuna toimintoa voi hallita käyttöliittymästä joko liikuttamalla slideria tai syöttämällä arvon käyttöliittymän tekstikenttään. Slave-moodissa aliohjelma seuraa ulkoisia ohjearvoja ja ilmaisee niiden muutosta. Signaalit Stepper 1 enabled ja Stepper 2 enabled ilmaisevat, ovatko fyysisen laitteen moottorit ohjattavissa. Jos moottorit poistetaan käytöstä, ohjelman Azimuth-kahvan kuvake ei käänny signaalien mukaan. Tässä tapauksessa signaalien muutos ilmaistaan ainoastaan numeroarvojen muutoksena. Ohjelman vasemmasta laidasta voi asettaa CAN-väylän parametrit. Valittavissa on alasvetovalikosta haluttu baudrate väliltä 125 kbit/s aina 1 Mbit/s:n nopeuteen asti. Tämän valinnan alapuolelta voidaan valita CAN-väyläadapterin haluttu kanava. Adapterissa on käytössä neljä eri kanavaa, ja näistä voidaan kytkeytyä kahteen yhdestä aliohjelmasta. Ohjelmaan on implementoitu myös backup-toiminnallisuus, joka tyypillisesti kytkeytyy eri väylään varsinaisen toiminnallisuuden kanssa, jotta vikatilanteissa laitteisto pysyisi toimintakuntoisena. Keskeltä ylälaidasta löytyy kenttä, johon laitteen laitenumero (NodeID) on syötettävissä. Tämän lisäksi vasemmasta alalaidasta löytyy painike, jolla laite liitetään CAN-väylään.

7.3 XControl

XControlilla on mahdollista luoda uusi mittari tai säädin ohjelmaan. LabVIEW tarjoaa valmiita säädin ja mittarimoduuleita, mutta pelkästään näillä ei kyennyt luomaan haluamaani toiminnallisuutta. Tämän takia loin kaksi uutta XControlia, omansa sekä kulma- että kierroslukuohjearvolle. XControlin luonti onnistuu LabVIEW'n käyttöliittymän sisältä ja moduulia pystyy käyttämään useissa eri LabVIEW-ohjelmissa tuomalla sen ohjelman rakenteeseen.



Kuva 23. Kulmaohjearvon XControl.

Kuvassa 23 on näkyvissä kulmaohjearvon XControl-moduulin rakenne ja käytetyt lohkot. Ohjelman vasemmassa laidassa on sisään tulevat signaalit, joita ovat

- Container State
- Display State In
- Data In
- Azimuth PictureBox.

Container State -klusteri sisältää tiedon XControlilta pyydetystä tilasta, eli sillä voidaan vaihtaa XControl esimerkiksi indikaattoritilasta kontrolleritilaan. Indikaattoritilassa XControl ei ole ohjattavissa esimerkiksi hiirellä. Tämän lisäksi se sisältää tietoa siitä, onko ohjelma RUN-moodissa ja mikä ohjelman referenssinumero on. [25.]

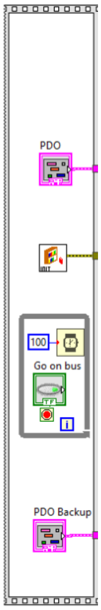
Display State -klusteri sisältää tietoja XControlin sisäisestä tilasta ja siitä, missä tilassa se on tällä hetkellä sekä onko näytettävä data muuttunut. Display Staten muutoksesta voidaan myös laukaista tapahtuma. Tämän lisäksi ohjelmaan tuodaan sisälle Azimuth PictureBox, joka on valmiiksi rajattu alue, mihin kuva voidaan sijoittaa.

Data In -struktuurilla tuodaan PDO-struktuuri XControlin sisään, ja tämä sisältää varsinaisen CAN-väylästä luetun datan. Struktuurista luetaan halutut muuttujat ulos, joiden muuttuessa laukaistaan Value Changed -tapahtumat. Kuvassa 23 näkyy Azimuth-arvon muutoksesta liipaistavan tapahtuman ohjelma, jossa tarkastetaan, onko akseli slave- vai master-tilassa, ja sen perusteella pätellään, luetaanko Azimuth-muuttujalta uusi arvo PDO-struktuurin txPDO-muuttujiin.

Oikeassa ylälaidassa sijaitsee Azimuth-kahvan kuvan kääntöohjelma, jossa luetaan kuva tiedostopolusta. Kuva puretaan bittikartaksi, jota muokataan annetun Azimuth slider -kulmaohjearvon mukaisesti. Tämän jälkeen kuva taas kootaan ja näytetään käyttöliittymässä. Lopuksi XControlista viedään ulos kulloisetkin tilatiedot sekä PDO-struktuuri, johon on kirjoitettu uudet muuttujan arvot.

7.4 Väylän alustus ja datastruktuurien luonti

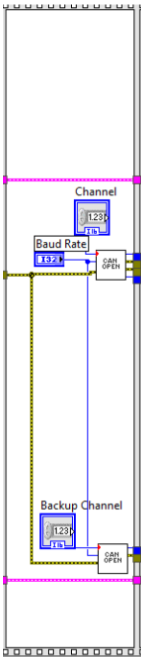
Ohjelman alustuksessa alustetaan PDO-datastruktuurit normaalille sekä backup-kommunikaatiolle (kuva 24). Datastruktuureissa on määritelty Azimuth-kahvan kommunikaatioparametrit oikeaa laitetta vastaaviksi. Tämän lisäksi aliohjelmassa alustetaan Kvaserin CANlib-kirjasto Init-lohkolla. Aliohjelma sisältää myös while-loopin ja 100 millisekunnin ajastimen. 100 millisekunnin välein tarkistetaan Go on bus -painikkeen tila, ja jos painikkeen tila on ”true”, lopetetaan while-loopin suorittaminen, alustetaan aiemmin mainitut parametrit ja siirrytään seuraavaan sekvenssiin.



Kuva 24. Väylän alustus ja datastruktuurien luonti.

7.5 Väyläparametrien anto ja väylän avaus

Väylänparametrisoinnissa ohjelmalle annetaan väyläparametrit, jotka tässä tapauksessa ovat väylän baudrate, väylän kanavanumero sekä backup-väylän kanavanumero. Nämä parametrit viedään Can open -aliohjelmaan, joka toimii rajapintana Kvaserin CANlib-kirjaston aliohjelmalle (kuva 25). Kvaserin aliohjelma keskustelee CAN-adapterin kanssa ja avaa oikean väylän ja tuo väylän kahvan käyttöön. Kahva on tärkeä, koska se viittaa oikeaan rautarajapintaan liitettyyn väylään ja tämän avulla tiedämme keskustelevamme oikean väylän kanssa. Kahvat on suljettava ohjelman loputtua, koska niitä voi olla auki enimmillään 64 kappaletta ennen muuttujan muistivuotoa, joka aiheuttaa virhetilanteen kommunikaatioon.



Kuva 25. Väylän avaus LabVIEW-aliohjelmassa.

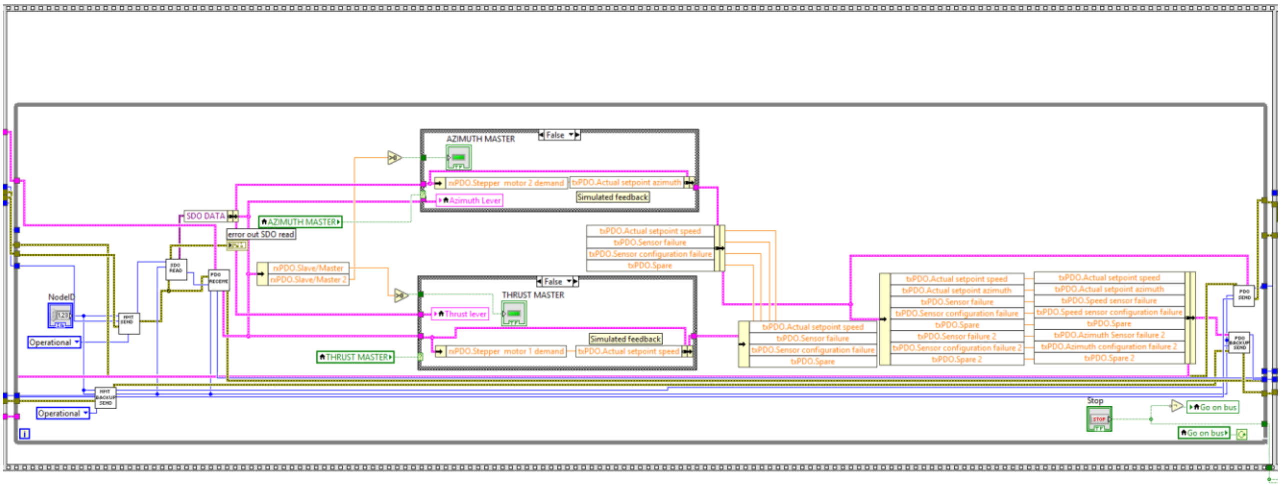
7.6 Käynninaikaisen ohjelman suoritus

Aliohjelman suorituslohkossa suoritetaan varsinaiset toiminnallisuudet ohjelman eri moodeissa. Tämä ohjelman osa on while-loopin sisällä, jota suoritetaan, kunnes Stop-painiketta painetaan. Aluksi luetaan ohjelmalle annettu laitenumero (NodeID) ja asetetaan laitteen tilaksi Operational. Tämän jälkeen aiemmalta ohjelmalohkolta saadut kanavakohtaiset kahvat tuodaan aluksi NMT-send ja backup NMT -send lohkoille. NMT-send-lohkot lähettävät kumpaankin CAN-väylään NMT-viestin laitteen siirtymisestä Operational-tilaan. Tämän jälkeen kanavien kahvat tuodaan SDO-read-lohkoon. Tämä lohko suorittaa SDO-viestien lukemisen väylästä, jonka jälkeen luetut parametrit tallennetaan SDO-datastruktuuriin. SDO-kommunikoinnin virheelle on myös oma diagnostiikkalohkonsa, josta on satavilla virhetilanteessa tietoa virheen syystä.

SDO-lohkon jälkeen kahva ja diagnostiikkatiedot tuodaan PDO-read-lohkoon. Tämä lohko lukee varsinaista prosessikommunikointia. Lohkoon tuodaan sisälle alustettu PDO-datastruktuuri, johon lohko tallentaa väylästä luetut parametrit ja päivitetty datastruktuuri tuodaan lohkosta ulos.

Luetusta PDO-datasta luetaan bittejä rxPDO.Slave/Master ja rxPDO.Slave/Master 2. Nämä bitit määrittävät azimuth- ja thrust-akselien tilan, eli ovatko ne master- vai slave-tilassa. Master-tilassa ollaan, kun bitin tila on true, muussa tapauksessa bitin tila on false, eli ollaan slave-tilassa. Kummallakin akselilla on omat case-lausekkeensa, joissa on määritetty toiminnallisuus master- ja slave-tiloissa. Kuvan 26 tapauksessa ollaan slave-tilassa, jossa annetaan master/slave-tieto merkkivaloille azimuth master sekä thrust master.

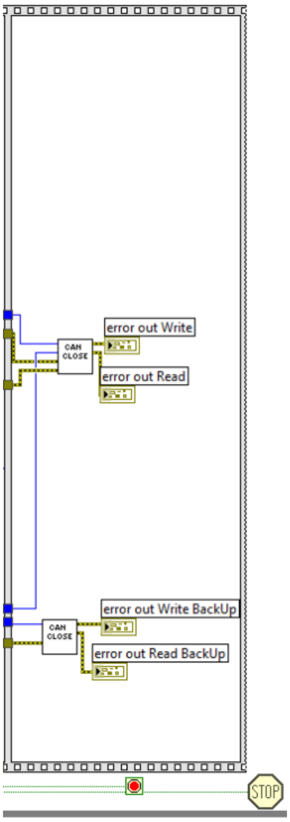
PDO-struktuuri luetaan myös azimuth- ja thrust-akseleiden LabVIEW XControleille. Slave-tilanteissa simuloidaan signaali myös PDO:n lähetysstruktuuriin. Tämä tehdään, jotta toiminnallisuus pysyisi vastaavana kuin oikeassa kahvassa, eikä ohjelmassa ole varsinaista anturitietoa saatavissa. Tämän jälkeen akseleiden datastruktuurit yhdistetään tarpeellisilta osin, eli tässä tapauksessa luetaan thrust-akselin kierroslukuohjearvo, sensoritiedon oikeellisuus, sensorin konfiguraatiotieto sekä varabitin Azimuth-akselin struktuuri. Tämän jälkeen luetaan vielä backup-kommunikaatiota varten PDO-struktuurista sen tarvitsemat tiedot backup-kommunikaation datastruktuuriin. Tämän jälkeen kahvat luetaan pdo send- ja pdo backup send -lohkoihin, jotka lähettävät PDO-viestit CAN-väyliin.



Kuva 26. Aliohjelman main-loop.

7.7 Ohjelman suorittamisen lopetus

Aiemmalta ohjelmalta tuodaan kanavien kahvat ja diagnostiikkatiedot tähän ohjelman osaan. Can close -aliohjelma huolehtii kahvojen sulkemisesta, sekä muodostaa diagnostiikkaviestit error out -lohkoille (kuva 27). Jos ohjelmassa on tapahtunut virhe, näistä lohkoista saadaan diagnostiikkatiedot ulos selväkielisenä. Tämän jälkeen ohjelman suoritus lopetetaan ja ohjelman ulkopuolinen while-loop pysäytetään stop-aliohjelmalla, joka on osa LabVIEW'n vakiokirjastoa.



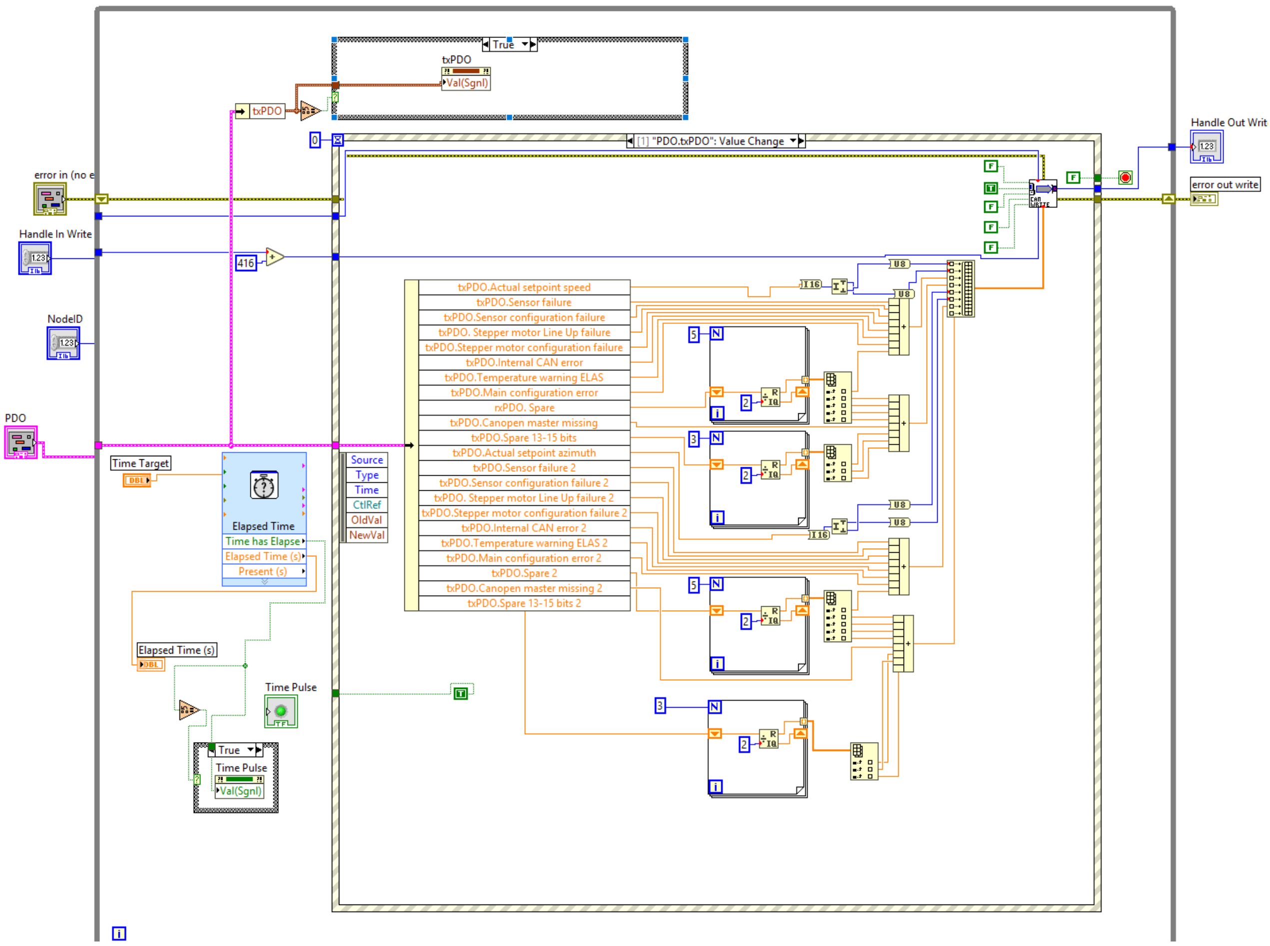
Kuva 27. Tässä lohkossa suoritetaan kanavien kahvojen sulkeminen.

7.8 Process Data Object-struktuurin lähetys

PDO Send-lohkon tehtävänä on kirjoittaa PDO-datastruktuurin sisältämät tiedot fyysiseen CAN-väylään (kuva 28). Tämä lohko käyttää hyväkseen Kvaserin CANlib-kirjaston Can Write -lohkoa. Sisään tulevia parametreja ovat

- Error in
- Handle in Write
- NodeID
- PDO.

Lohko sisältää ajastetun toiminnallisuuden, PDO-lähetys liipaistaan ajastimella Time Target -muuttujaan annetun arvon mukaisin väliajoin. Kun aika on kulunut, suoritetaan PDO-struktuurin lähetys. Tämä lähetys suoritetaan myös PDO.txPDO-struktuurin muuttuessa, kuten kuvasta 28 voidaan todeta. PDO.txPDO-struktuuri muuttuu annettaessa uusi ohjearvo joko kulma- tai kierroslukuohjearvolle ohjelman ollessa master-tilassa. Tässä lohkossa myös muodostetaan taulukot CANopen-protokollaa varten. PDO-struktuurista poimitaan tarvitsemamme bitit, jotka myöhemmässä vaiheessa yhdistetään tavuksi. Tämän jälkeen nämä tavut taulukoidaan kahdeksaksi riviksi taulukkoon, jonka jälkeen Kvaserin CANlib-kirjaston Can Write -lohko kirjoittaa ne väylään.



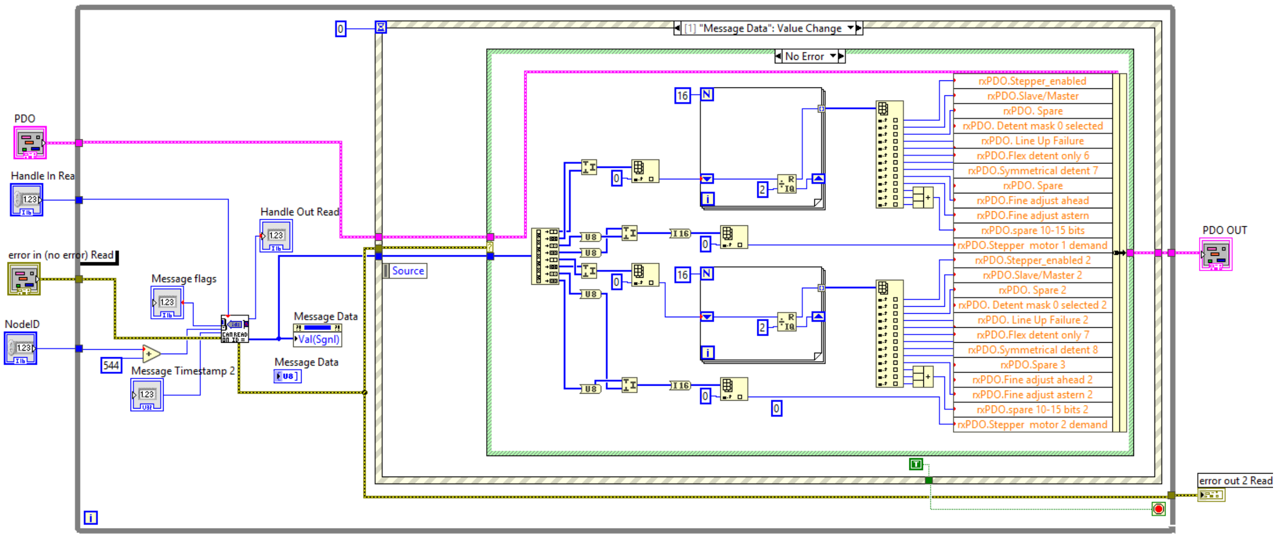
Kuva 28. Process Data Objectsin lähetyksen aliohjelma.

7.9 Process Data Object -struktuurin lukeminen

PDO Read-lohko käsittelee tulevat PDO-viestit. Viestit luetaan CAN-väylästä käyttäen Kvaserin CANlib-SDK:n Can read on ID -aliohjelmalla (kuva 29). Azimuth-kahvan COB-ID on PDO-kommunikoinnille desimaaleina 544 + laitenumero (NodeID). PDO Read-lohkossa sisään tulevia parametreja ovat

- PDO
- Handle In Read
- error in(no error)
- NodeID.

Lohkon alussa tapahtuu datan luku CAN-väylästä Can read on ID -aliohjelmalla. Aliohjelma lukee väylään tullen viestin, jos viestin COB-ID-numero vastaa aliohjelmalle sisään tuotua COB-ID-numeroa. Oikean viestin tullessa se luetaan Message Data -nimiseen muuttujaan, jonka muutosta tarkkaillaan Value Change -eventillä. Muuttujan tilan muutoksessa saapunut viesti käsitellään, pilkotaan kahdeksaksi tavuksi, jotka pilkotaan jokainen kahdeksaksi bitiksi. Bitit luetaan PDO-struktuuriin niitä vastaaviin bitteihin, tämän struktuurin rakenne on tehty vastaavaksi oikean Azimuth-kahvan mukaan.Tämän jälkeen PDO-struktuuri luetaan ulos ohjelmasta ja aliohjelman suoritus lopetetaan.



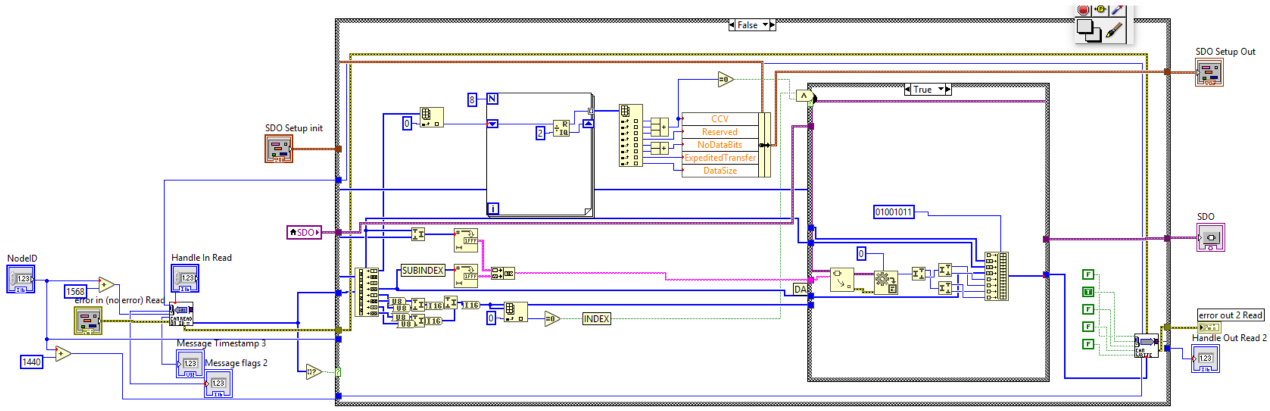
Kuva 29. Process Data Objectsin luvun aliohjelma.

7.10 Service Data Object-struktuurin lukeminen

SDO Read-lohkossa luetaan väylästä saapuvat SDO-viestit CANlib SDK:n CAN READ ON ID -lohkoa käyttäen. SDO-kommunikoinnin COB-ID on tässä tapauksessa desimaaleina 1544+NodeID. Sisään tulevia parametreja ovat

- NodeID
- Handle In READ
- Error in(no error) Read.

Ohjelmassa käytetty Can read on ID -lohko lukee CAN-väylästä viestejä laitteen SDO-parametria vastaavalla COB-ID:llä. Tämän jälkeen tarkistetaan, sisältääkö luettu viesti arvoja vai onko se täysin tyhjä. Kuvassa 30 nähdään viestin käsittely tilanteessa, jossa viesti sisältää arvoja. Aluksi viesti pilkotaan tavuiksi, joista erotellaan SDO-parametrin index- ja subindex-numerot sekä niiden sisältämä arvo. Tämän lisäksi SDO-parametri sisältää tietoa käytetystä protokollasta ja bittimäärästä, jotka käsitellään erillään varsinaisista SDO-parametreista. Saapuneet parametrit tallennetaan SDO-struktuuriin, jonka lisäksi CAN-väylään lähetetään viesti. Tämän viestin tarkoitus on vahvistaa SDO-parametrien saapuminen perille sekä vahvistaa niiden käsittely. Tapauksessa, jossa vahvistusviestiä ei lähetettäisi, pysähtyisi SDO-kommunikointi virheeseen eikä master-laite lähettäisi seuraavaa parametria. Viestin lähetyksen tai epäonnistuneen kommunikoinnin jälkeen lohkon suoritus lopetetaan ja lohko suoritetaan seuraavan kerran seuraavassa ohjelmasyklissä.



Kuva 30. Service Data Objectsin luvun aliohjelma.

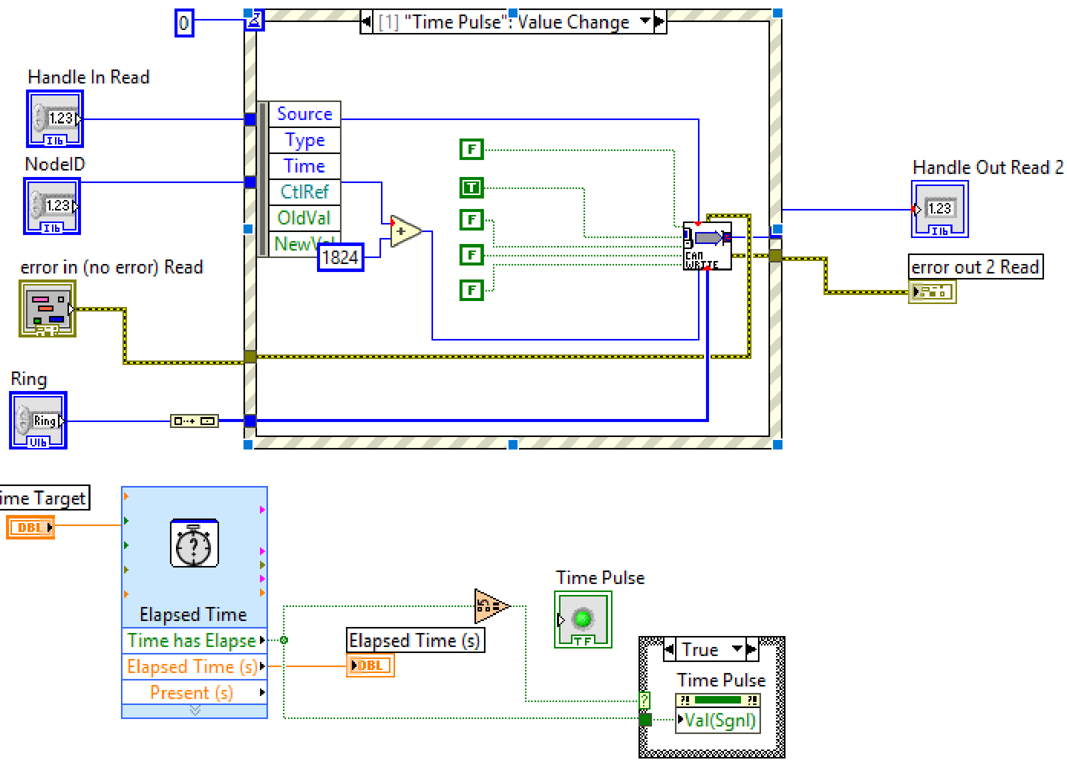
7.11 Network Management -viestin lähetys

NMT send -lohko lähettää laitteen kulloistakin tilaa CAN-väylään (kuva 31). Viestin lähetys on ajastettu ja se lähetetään Time Target -muuttujaan tallennetun arvon välein. Tähän lohkoon sisään tulevia parametreja ovat

- Handle In Read
- NodeID

- erro in(no error) Read
- Ring.

Ring-muuttuja tuo lohkon sisään tiedon kulloinkin lähetettävästä arvosta. Ring-muuttuja on oikeammin enum-tyyppinen muuttuja, joka sisältää laitteelle mahdolliset tilat. NMT-parametrien lähetyksen täytyy olla syklistä, jonka takia lohkoon on toteutettu ajastin. Viestin lähetyksen jälkeen lohkosta annetaan ulos CAN-väylän kahva ja diagnostiikka-tiedot.



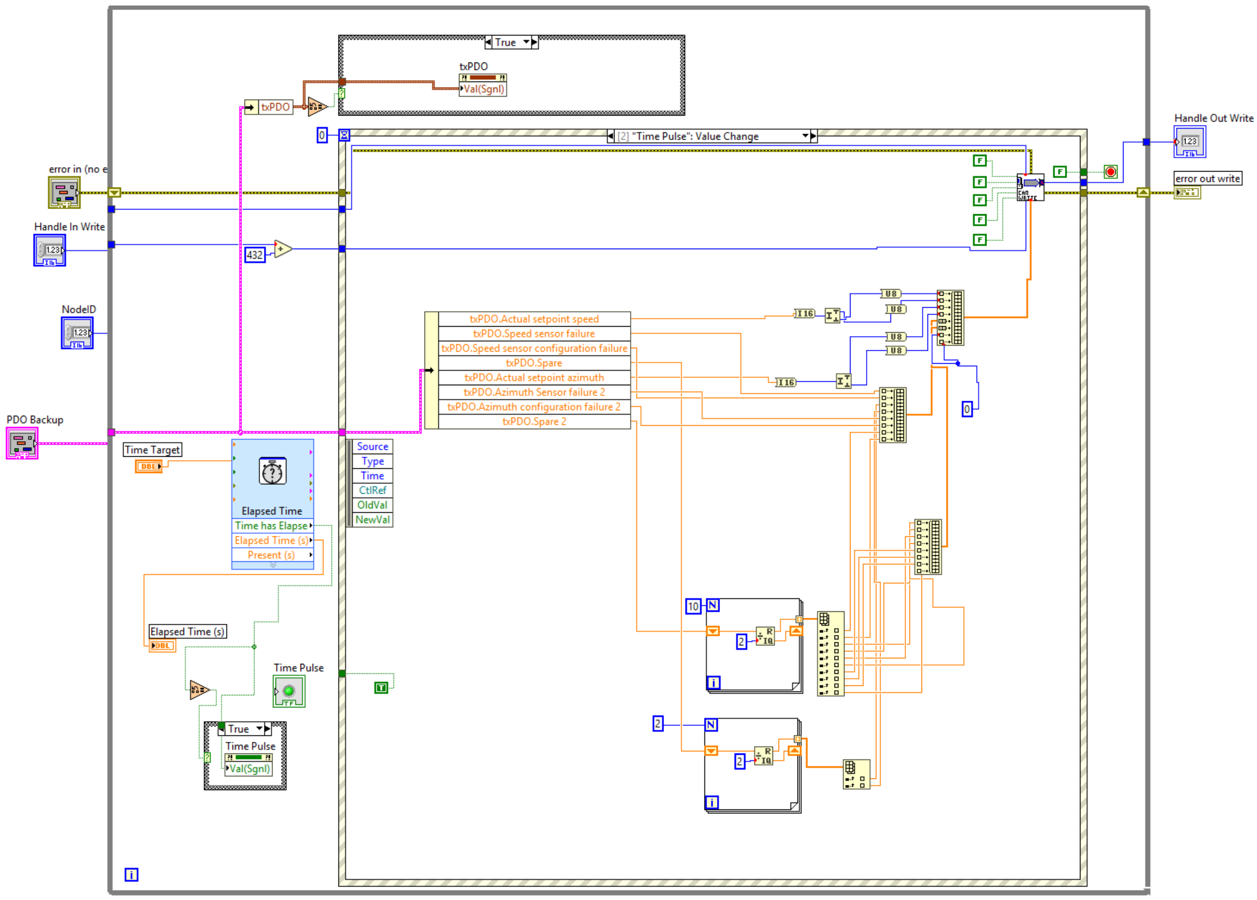
Kuva 31. Network Management viestin lähetys.

7.12 Process Data Object backup -viestin lähetys

PDO backup send -lohkon tehtävä on lähettää PDO-viestit myös backup-väylään (kuva 32). Backup-väylässä PDO:n COB-ID on 432(DEC) + laitenumero (NodeID). Tähän lohkoon sisään tulevia parametreja ovat

- Error in(no error)
- Handle In Write
- NodeID
- PDO Backup.

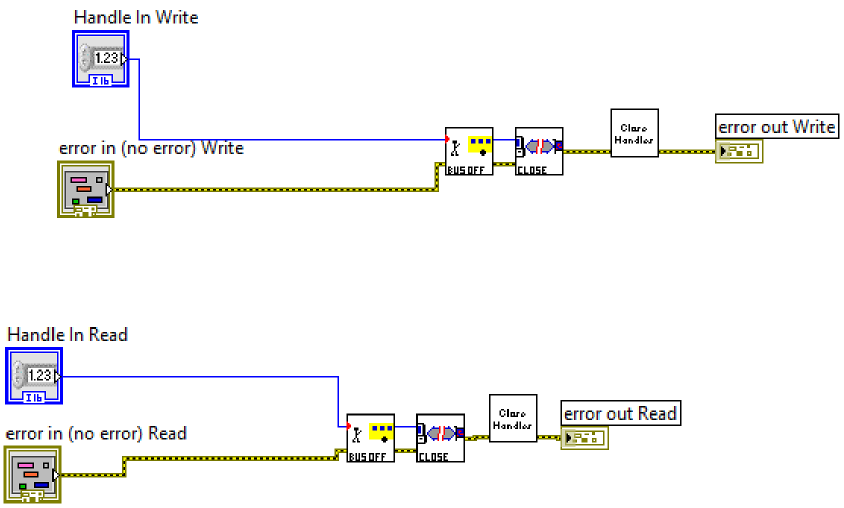
Lohko on rakenteeltaan vastaava kuin main-väylän PDO send-lohko ja myös tämä lohko lähetetään asetetuin väliajoin. Toiminnaltaan lohko on myös hieman yksinkertaisempi, koska lähetettävä datamäärä sisältää vain tärkeimmät muuttujat. Lohkon lähetys tapahtuu Can Write -lohkolla, ja lähetyksen jälkeen luetaan kahva sekä diagnostiikkatiedot ulos.



Kuva 32. PDO-viestin lähetys backup-väylään.

7.13 CAN-väylän sulkeminen

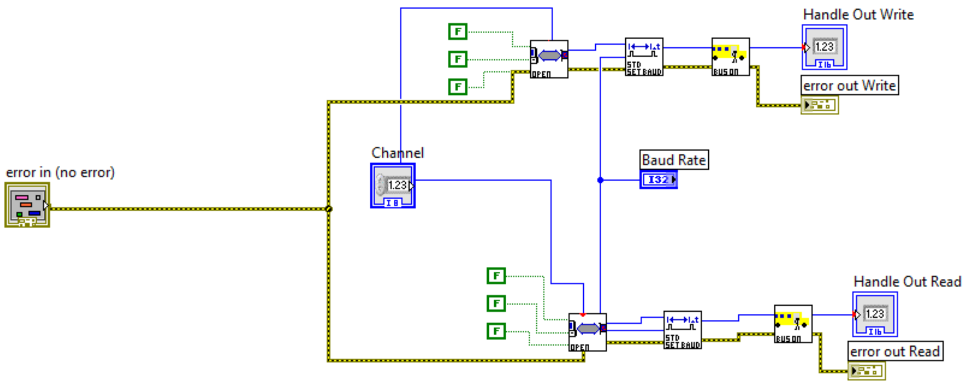
Can close -lohkon tehtävänä on sulkea väylille avatut kahvat (kuva 33). Avatut kahvat tuodaan lohkoon ja ne viedään Kvaserin CANlib-SDK:n Bus off -lohkoon, jonka tehtävä sulkea väyläkommunikointi. Tämän jälkeen kahva viedään close-lohkoon, joka sulkee ohjelman alussa avatun väylän. Tämän jälkeen vielä Close Handles -lohko varmistaa kaikkien kahvojen olevan kiinni, jonka jälkeen diagnostiikkatiedot viedään error out -lohkoon, josta ne voidaan lukea selväkielisenä.



Kuva 33. CAN-väylien sulkeminen.

7.14 CAN-väylän avaus

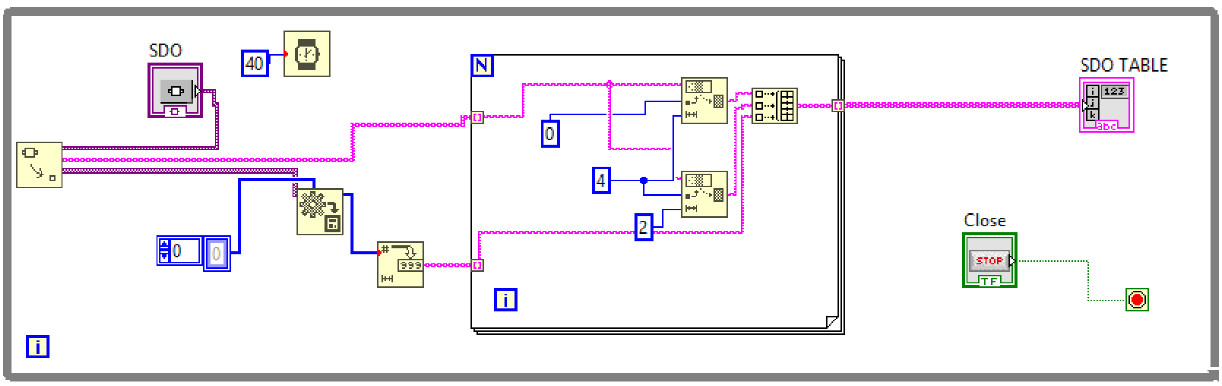
CAN open -lohkon tehtävä on avata uudet CAN-väylät normaali- ja backup-kommunikaatiolle (kuva 34). Lohkoon tuodaan sisään aiemmalta lohkolta error in(no error) -diagnostiikkatieto. Tämän lisäksi käyttöliittymältä tuodaan channel-muuttuja, jonka muoto on 8-bittinen kokonaislukumuuttuja (engl. integer) ja Baud Rate -muuttuja (32-bittinen kokonaislukumuuttuja). Kommunikaatiorajapinta väyläadapterin kanssa on toteutettu Kvaserin CANlib-SDK:n lohkoilla open, std set baud ja bus on. Open-lohko avaa uuden CAN-väylän, johon std set baud -lohko asettaa käyttöliittymästä saadun baudrate-nopeuden. Lopuksi bus on -lohko asettaa väylän aktiiviseen tilaan ja lohkosta ulos viedään väyläkohtaiset kahvat ja diagnostiikkatiedot.



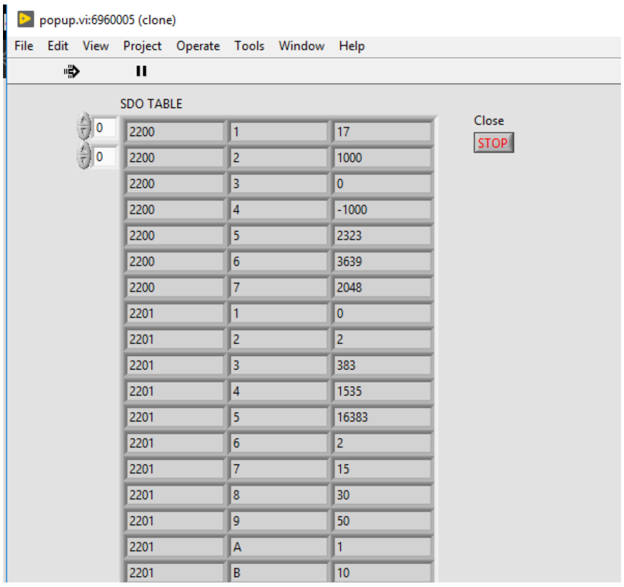
Kuva 34. CAN-väylien avaaminen.

7.15 Service Data Objects -ponnahdusikkuna

SDO Popup -lohko näyttää tallennetut SDO-parametrit taulukkomuotoisena datana (kuva 35). Lohkosta aukeaa käyttöliittymäsivu erillisenä ikkunana pääkäyttöliittymän päälle. Käyttöliittymäsivulla aukeaa taulukko (kuva 36), jossa on kolme saraketta. Taulukon vasemmanpuoleisin sarake kertoo Service Data Objectsin ID-numeron, keskimmäinen numero taas bitin numeron kyseisessä datastruktuurissa, ja oikeanpuoleisin arvo on SDO-parametriin tallennettu arvo.



Kuva 35. SDO-tiedot nähdään taulukkomuodossa.



Kuva 36. Service Data Objects -taulukko.

8 Yhteenveto

Työssä oli tarkoitus luoda simulointiohjelmisto laivojen komentosillanlaitteiden simulointiin ja toteuttaa ohjelmaan Azimuth-kahvan malli. Tavoitteena oli luoda mahdollisimman helposti muokattava ohjelmisto, jonka koodi olisi helppolukuista ja lähes kenen tahansa ymmärrettävää. Työssä saavutettiin asetetut tavoitteet ohjelmiston ja mallin toimivuuden osalta, mutta työssä kuitenkin havaittiin valitun LabVIEW-kehitysympäristön haasteet koodin helppolukuisuuden ja muokattavuuden suhteen.

Ohjelman teossa on syytä kiinnittää huomiota hyviin käytäntöihin ja pilkkoa se tarpeeksi pieniin ja selkeisiin osiin. Tämän osalta työssä on vielä jatkossa kehitettävää ja ohjelmiston koodin siistimistä on vielä tehtävä. Jatkossa ohjelmistoon täytyy myös lisätä uusien laitteiden malleja ja näiden toimivuutta täytyy testata olemassa olevan mallin kanssa.

Työn ansiosta tietämys CAN-väylästä, sen eri standardeista ja laitteistoista nousi mikä helpottaa tulevaisuuden kehitystyötä. LabVIEW:sta saatu kokemus on myös tärkeätä ja kokemuksen ansiosta tulevaisuuden laajennoksia on helpompi suunnitella ennalta ja uusien ohjelmistokehitysprojektien kehitysympäristön valintaan saatiin lisäargumentteja. Työssä käytetty USB-CAN-adapteri osoittautui myös hyödylliseksi työkaluksi CAN-väylän monitoroinnissa ja diagnosoinnissa erillisen ohjelmiston avulla.

Työ oli toteutukseltaan haastava, mutta samalla hyvin opettava. Minulla ei ollut aiemmin syvempää kokemusta CAN-väylästä, johon tämä työ antoi hyvän kokonaisnäkemyksen. Tämän lisäksi työn parissa pääsin opettelemaan graafista ohjelmointia LabVIEW'lla, joka vaatii hieman erilaisen lähestymisen perinteiseen tekstipohjaiseen ohjelmointiin verrattuna.

Haasteena työssä oli LabVIEW'n ohjelmarakenteen tekeminen ja suunnitteleminen mahdollisimman modulaariseksi ja helposti muokattavaksi, mutta silti suorituskykyiseksi. Näen etten onnistunut tässä täydellisesti, mutta uskon ohjelmiston toimivan käyttötarkoituksessaan hyvin ja siihen olevan suhteellisen helppo lisätä uusia laitteita. Pyrin rakentamaan ohjelmiston niin, että kukin aliohjelma mahtuu hyvin näyttöpäätteelle, sillä LabVIEW ei sisällä suurennus- tai pienennystoimintoa. Tämän takia isojen ohjelmien rakenteen seuraaminen ja ohjelman toiminnallisuuden selvittäminen on hankalaa.

Valittu USB-CAN-adapteri toimi hyvin ja Kvaserin SDK- ja LabVIEW-kirjasto toimivat ongelmitta, ja ne olivat helppokäyttöisiä. Adapteria varten hankittiin myös Tke:n CanTrace-ohjelmiston, jonka avulla CAN-väylän liikenteen monitorointi on helpompaa kuin Kvaserin ilmaisella Canking-ohjelmistolla. Tämä mahdollistaa samalla adapterilla laitteen simuloinnin ja CAN-väylän monitoroinnin, joka monipuolistaa ja helpottaa testausta ja vianhakua.

Haastavuutensa myötä työ oli minulle hyvin inspiroiva ja motivoiva. Oli mielenkiintoista tutustua CAN-väylän toimintaan ja sen eri standardeihin ja luulen tästä olevan tulevaisuudessa minulle paljon hyötyä. LabVIEW oli myös kiinnostava kehitysympäristö ja siihen perehtyminen oli antoisaa. Sen luomat mahdollisuudet ja toisaalta myös sen puutteet toivat hyvän käsityksen työn kuluessa siitä, mihin sitä voisi tulevaisuudessa käyttää ja mitä se voi tarjota.

Lähteet

1 Energiatehokasta merimatkaa. 2019. Verkkoaineisto. ABB Oy, Marine and Ports. <https://new.abb.com/fi/abb-lyhyesti/suomessa/yksikot/marine-and-ports>. Luettu 25.10.2019.

2 Controller Area Network (CAN) Overview. 2019. Verkkoaineisto. National Instruments. <https://www.ni.com/fi-fi/innovations/white-papers/06/controller-area-network--can--overview.html>. Luettu 20.8.2019.

3 CAN knowledge. 2019. Verkkoaineisto. CAN in Automation (CiA). <https://www.can-cia.org/can-knowledge/>. Luettu 20.8.2019.

4 The CAN Protocal Tutorial. 2019. Verkkoaineisto. Kvaser AB. <https://www.kvaser.com/can-protocol-tutorial/>. Luettu 28.8.2019.

5 CAN high-speed transmission. 2019. Verkkoaineisto. CAN in Automation (CiA). <https://www.can-cia.org/can-knowledge/can/high-speed-transmission/>. Luettu 29.8.2019.

6 CAN Physical Layer and Termination Guide. 2011. Verkkoaineisto. National Instruments. <http://www.rpi.edu/dept/ecse/mps/CAN-LabVIEW_info/NI-Tutorial-9759-en.pdf>. Luettu 02.09.2019.

7 CAN data link layers in some detail. 2019. Verkkoaineisto. CAN in Automation (CiA). <https://www.can-cia.org/can-knowledge/can/can-data-link-layers/>. Luettu 05.09.2019.

8 CAN FD EXPLAINED - A SIMPLE INTRO (2019). 2019. Verkkoaineisto. CSS Electronics. <https://www.csselectronics.com/screen/page/can-fd-flexible-data-rate-intro>. Luettu 10.09.2019.

9 CAN Application Layer for industrial applications. 2019. Verkkoaineisto. CAN in Automation (CiA). <http://read.pudn.com/downloads157/doc/comm/701058/CiA%20CanOpen/CiA%20201207%20DS%20V1.1%20CAN%20Application%20layer%20for%20industrial%20applications(TC_201to207v01010002).pdf>. Luettu 12.09.2019.

10 CANOPEN EXPLAINED - A SIMPLE INTRO (2019). 2019. Verkkoaineisto. CSS Electronics. <https://www.csselectronics.com/screen/page/canopen-tutorial-simple-intro/language/en 3.9.2019>. Luettu 12.09.2019.

11 The Basics of CANopen. 2019. Verkkoaineisto. National Instruments. <https://www.ni.com/fi-fi/innovations/white-papers/13/the-basics-of-canopen.html>. Päivitetty 5.3.2019. Luettu 13.09.2019.

12 CANopen. 2019. Verkkoaineisto. ABB Oy<https://new.abb.com/drives/fi/liitettavyys/kenttavaylayhteydet/canopen>. Luettu 20.09.2019.

13 PDO - OPERATING THE CANOPEN NETWORK. 2019. Verkkoaineisto. CSS Electronics. <https://www.csselectronics.com/screen/page/canopen-tutorial-simple-intro/language/en#pdo-process-data-object>. Luettu 22.09.2019.

14 Process data object (PDO). 2019. Verkkoaineisto. CAN in Automation (CiA). <https://www.can-cia.org/can-knowledge/canopen/pdo-protocol/>. Luettu 23.09.2019.

15 Service data object (SDO). 2019. Verkkoaineisto. CAN in Automation (CiA). <https://www.can-cia.org/can-knowledge/canopen/sdo-protocol/>. Luettu 24.09.2019.

16 SDO services. 2019. Verkkoaineisto. CANopen Solutions. <https://www.canopensolutions.com/english/about_canopen/SDO-services.shtml>. Luettu 24.09.2019.

17 Network management (NMT). 2019. Verkkoaineisto. CAN in Automation (CiA). (https://www.can-cia.org/can-knowledge/canopen/network-management/). Luettu 28.09.2019.

18 Emergency Messages – CanOpen. 2015. Verkkoaineisto. ByteMe. <http://www.byteme.org.uk/canopenparent/canopen/emergency-messages-canopen/>. Luettu 29.09.2019.

19 EMCY write protocol. 2019. Verkkoaineisto. CAN in Automation (CiA). <https://www.can-cia.org/can-knowledge/canopen-fd/emcy-write-protocol/>. Luettu 29.09.2019.

20 Technical documents. 2019. Verkkoaineisto. CAN in Automation (CiA). <https://www.can-cia.org/groups/specifications/>. Luettu 1.10.2019.

21 CiA® 401 series: Device profile for generic I/O modules. 2019. Verkkoaineisto. CAN in Automation (CiA). <https://www.can-cia.org/can-knowledge/canopen/cia401/>. Luettu 1.10.2019.

22 What is LabVIEW?. Verkkoaineisto. 2019. Electronicsnotes. <https://www.electronics-notes.com/articles/test-methods/LabVIEW/what-is-LabVIEW.php>. Luettu 28.09.2019.

23 Maneuvering the world’s largest cruise ships with decimeter accuracy. 2013. Verkkoaineisto. ABB Oy. <https://new.abb.com/news/detail/13961/maneuvering-the-worlds-largest-cruise-ships-with-decimeter-accuracy>. Päivitetty 30.1.2019. Luettu 19.09.2019.

24 Kvaser USBcan Light 4xHS User’s Guide. 2015. Verkkoaineisto. Kvaser AB. <https://www.kvaser.com/software/733013098176/V1_7_699/kvaser_usbcan_light_4xhs_userguide.pdf>. Luettu 2.10.2019.

25 Using the XControl Facade Ability. 2018. Verkkoaineisto. National Instruments. <http://zone.ni.com/reference/en-XX/help/371361R-01/lvconcepts/creating_an_XControl_facade/>. Luettu 5.10.2019.

26 Kvaser USBcan Light 4xHS. 2019. Verkkoaineisto. Kvaser AB. <https://www.kvaser.com/product/kvaser-usbcan-light-4xhs/>. Luettu 4.10.2019.

27 CAN physical layer. 2019. Verkkoaineisto. CAN in Automation (CiA). <https://www.can-cia.org/can-knowledge/can/systemdesign-can-physicallayer/>. Luettu 3.11.2019.

28 LabVIEW Block Diagram Explained. 2019. Verkkoaineisto. National Instruments. <https://www.ni.com/getting-started/labview-basics/>. Luettu 9.11.2019.

29 30 Types of Navigation Equipment and Resources Used Onboard Modern Ships. 2019. Verkkoaineisto. <https://www.marineinsight.com/marine-navigation/30-types-of-navigational-equipment-and-resources-used-onboard-modern-ships/>. Luettu 10.11.2019.