TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Andre Toikka 143025IABB

SERVOSÜSTEEMI TELGEDE VAHELDUVJUHTIMISE ALGORITMI KOOSTAMINE DELTA ELECTRONICS SEADMETEL

bakalaureusetöö

Juhendaja: Andres Rähni

Tehnikateaduste magister

Tallinn 2018

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Andre Toikka

21.05.2018

Annotatsioon

Käesolev bakalaureusetöö käsitleb servokontrolleriga mootori juhtimist ja mitme servokontrolleri vahelduvat juhtimist minimaalse ajalise viitega võimalikult lihtsasti rakendatava tarkvaralahendusega. Töö tutvustab servosüsteemi ülesehitust, seadmeid ühendava CAN-võrgu tööpõhimõtteid ning tootjaspetsiifliste programmide kasutamist võrgu konfigureerimiseks ja kontrolleri programmeerimiseks. Servokontrolleri juhtimise teostamiseks esitatakse kolm varianti ning valitakse neist sobivaim, pidades silmas töökindlust ja jõudlust, ning realiseeritakse see programmikoodina, selgitades lähenemist ja võimalikke vigu. Seejärel kirjeldatakse ja lisatakse programmile osa, mis võimaldab mitme telje juhtimist lihtsustatult ja kasutataval kujul ka enam kui 2 servokontrolleriga süsteemi juhtimiseks.

Bakalaureusetöös kasutatakse Delta Electronics, Inc. seadmeid: kontrollerit DVP-SV2, CANopen laiendmoodulit DVPCOPM-SL ja servokontrollereid ASDA-A2. Töö kirjutamisel on realiseeritud näidisrakendus, mis kasutab töös tutvustatavat koodilahendust ja rakendab esitatud mustrit mitme servokontrolleri juhtimisel.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 50 leheküljel, 4 peatükki, 18 joonist, 1 tabelit.

Abstract

Creation of a servo system control algorithm for sequential operations using Delta Electronics devices

This thesis will present how a technical solution, including communications and software, was built to control multiple AC servomotors in sequence, operating taking turns. The control units (drives) of the two motors and the programmable logic controller (PLC) were connected via a CAN bus.

After conducting a selection of devices and configuring the bus device, the different approaches to communicating with the drive during normal movement controlling are analyzed, and the best approach, with reliability and performance in mind, is chosen. An example control routine is devised for one control unit with explanations for operating, then duplicated for a second control unit and a model along with working code, for sequential movement controlling, is introduced.

The devices used in this thesis are: controller DVP-SV2, CANopen module DVPCOPM-SL, drives ASDA-A2. All of the devices are produced by Delta Electronics, Inc. During the writing process of this thesis, an example servo system was set up to test and improve on the concepts introduced.

In the thesis, some manufacturer-specific tools are used for configuring the servo system network (Delta CANopen Builder) and writing the controller (PLC) software (Delta ISPSoft). For writing the program, the instruction list (IL) language is used and all examples presented are in instruction list. The thesis also covers some common pitfalls associated with writing cyclic code and misunderstanding cyclic continuity.

The thesis is in Estonian and contains 50 pages of text, 4 chapters, 18 figures, 1 table.

Lühendite ja mõistete sõnastik

CAN	Controller Area Network, CAN-siin (CAN-bus)
PLC	Programmable Logic Controller
IL	Instruction List
Draiv	Servokontroller (ka servovõimendi või servomuundur)
PDO	Process Data Object (siinil saadetav eeldefineeritud struktuuriga sõnum)
SDO	Service Data Object (seadistamiseks mõeldud sõnum)
HMI	Human Machine Interface (operaatoripaneel)
OD	Object Dictionary (CAN-võrgu seadme andmeobjektide kogum)
TxPDO	Transmit PDO (alamseadmest ülemseadmesse saadetav PDO)
RxPDO	Receive PDO (ülemseadmest alamseadmesse saadetav PDO)

Sisukord

Jooniste loetelu

Tabelite loetelu

Sissejuhatus

Neljanda tööstusrevolutsiooni (Industry 4.0 trendi) keskmes on programmeeritavad loogikakontrollerid ehk PLC-d, mis on parim viis tööstuse automatiseerimiseks [1]. Paljud suured PLC tootjad kasutavad oma seadmetes modernseid operatsioonisüsteeme, mis on kohaldatud ajast kriitiliselt sõltuvate rakenduste toetamiseks. Tekkinud on avatud lähtekoodiga tarkvara loogikakontrolleri programmide käivitamiseks [2] [3]. Nõudlus automatiseerimise järele on kasvamas, mis on tinginud ka odavamate ja avatud lähtekoodi ning avatud disainiga PLC-de tekke [4]. Moodsad tehaseseadmed on ühendatud Asjade Internetti (IoT) ning on pidevalt monitooritavad. Nendesse saab tarkvara kirjutada ka kõrgema taseme programmeerimiskeeltes [5]. Mikrokontrollerite kasutamine hobiprojektides on nende odava hinna tõttu levinud ning robootikal on suurenev roll igas kooliastmes.

Eeltoodud faktoreid silmas pidades leiab autor, et tänapäeval ei ole kokkupuude loogikakontrolleritega piiratud automaatikainseneridele, vaid kontrolleritega tegelema võib asuda iga tehnikahuviga inimene. See ajendas autorit valima lõputöö teemaks tehnilist valdkonda, millega on oma töös tihedalt seotud – servosüsteemi seadistuse ja tarkvara loomine. Käesoleva töö eesmärk on tutvustada ühe mootorite juhtimiseks mõeldud süsteemi ülesehitust, toimimist ja juhtimist. Servosüsteem koosneb antud töö kontekstis vähemalt kahest servoajamist ehk mootorist ja servokontrollerist (draiv).

Töös käsitletakse servokontrollerite vahelduvalt juhtimist, mis on tööstuses oluline töö element. Minimeerides ajalise viite kahe telje töö vahel on võimalik saavutada rahalist võitu näiteks tootmisliinidel, kus kiiruse suurenedes suureneb toodetavate detailide arv, või lõppkasutajale suunatud rakendustes, kus väiksem ooteaeg suurendab kliendi rahulolu (näiteks automatiseeritud pakiautomaatides). Töö teine eesmärk on formuleerida muster erineva arvu servoajamite lihtsustatult juhtimiseks.

Peatükis 1 käsitletakse seadmete valikul järgitavaid nõudeid ja süsteemi andmesidevõrgu ülesehitust ning toimimist. Peatükiks 2 kirjeldatakse võrgu konfigureerimist funktsionaalsuse saavutamiseks. Peatükis 3 antakse ülevaade kasutatava tehnoloogiaga

kontrolleri programmeerimisest ja andmeobjektide mäluaadresside määramisest. Peatükis 4 valitakse positsioneerimisalgoritmi lahendus ühe draivi juhtimiseks ning realiseeritakse see. Samuti kirjeldatakse vahelduvjuhtimise tööpõhimõtet signaliseerimismuutujate abil ning luuakse näidiskood telgede vahelduvjuhtimiseks.

1 Süsteemi arhitektuur

Järgnevalt kirjeldatakse juhitavate seadmete valiku kriteeriumeid ja põhjendatakse seadmevalikut, kirjeldatakse süsteemi ülesehitust ning andmesidevõrgu toimimist.

1.1 Seadmete valik

Servokontrolleri ja servomootorite juhtimiseks kasutatakse käesolevas töös Delta Electronics seadmeid, sest ettevõttes, kus töö läbi viiakse, on toodetel kasutatud selle tootja kontrollereid ja draive. Tootja seadmete maaletooja on varasemalt assisteerinud nimetatud ettevõtte mehhatroonikuid erinevate probleemide lahendamisel ning kuna seadmed on kasutusel sadades klientidele tarnitud masinates, on ka teadmuse loome seisukohast oluline nende seadmetega tööd ja uurimistegevust jätkata. PLC ehk mikroprotsessoril põhineva kontrolleri kasutamine on paindlik ja kuluefektiivne, sest muudatused süsteemi loogikas ei nõua muudatusi elektriskeemis [6]. Tööstusrakendustes kasutati 1960. ja 1970. aastatel palju releesid, taimereid ja rakendusespetsiifilisi piiratud funktsionaalsusega kontrollereid, mis on tänapäeval asendatud PLC-ga [7]. PLC-d on laialdaselt kasutusel kõigis tööstusvaldkondades, meditsiinis, robootikas, kliimaautomaatikas ja mujal valdkondades [8].

Käesolevas töös ühendatakse loogikakontroller (PLC) mitme servokontrolleriga, mis mootoreid juhivad. Eelistatakse kihilist disaini eesmärgiga lihtsustada juhtimismudeli rakendamist eri suurustega servosüsteemides. Alternatiivsetes kõik-ühes lahenduses oleks juhitavate mootorite arv fikseeritud ning nende arvu suurendamine vajaks seadme vahetust. Kihilises disainis piisab uue servokontrolleri lisamisest ahelasse. Servomootoreid kasutatakse, sest need võimaldavad täpsemat juhtimist rakendustes, mis nõuavad mikromeetrist või suuremat täpsust telgede liigutamisel [9].

Tootja pakutav seadmete sortiment on lai, seetõttu seatakse järgnevalt kriteeriumid, millele valitavad seadmed peaks vastama. Siinkohal ei käsitle töö mootori võimsust ega sellest tulenevat mudeli valikut, vaid valitakse sobiv servokontrolleri seeria, mille mudelite valik on piisavalt lai katmaks ära levinud tööstuslikud vajadused.

Tööstuslikule kontrollerile esitatavad nõuded:

- Piisavalt mälu keerukamate algoritmide hoiustamiseks
- Püsimälu seadistuste hoidmiseks
- Kiire liides draividega suhtlemiseks
- Liides operaatoripaneeliga (HMI) seadme juhtimiseks ja info saamiseks
- Hea sisseehitatud tugi tarkvara kirjutamisele ja selle töö online-seiramisele
- Võimalus laiendada analoog I/O moodulitega

Draivile (servokontrollerile) esitatavad nõuded:

- Ühe mootori juhtimiseks
- Juhitav üle CAN-i siini
- Võimaldab täpset juhtimist impulssanduriga (enkoodriga) mootorit kasutades

Käesolevates nõuetes eeldame, et vahelduvjuhtimist hakatakse teostama mõnes lahenduses, milles on vajadus täpse positsioneerimise ja sisendite-väljundite järele (nt. lõpplülitid, nullandurid). Mälu järele vajadus on püstitatud suuremas mahus, kui juhtkoodi jaoks vaja oleks, et rahuldada muid soove (positsioonide konfigureerimine, loendurid, logid) ning tagada lihtsamini laiendatavat süsteemi. Selline lähenemine leiab kasutust näiteks tööstuses paberi tootmise liinil [10].

Füüsiline suhtluskanal on seadme disainis tõenäoliselt lähedal draividele ja/või mootoritele ning tõenäoliselt veetakse kõrgepingejuhtmed kõrvuti või sama kaabliketti pidi sidekaablitega. Seega võivad indutseeritavad elektromagnetlained või seadme väliskeskkonnast kiirgavad elektromagnetlained häirida sidekanali signaali puhtust. CAN-i siini kasutamise soov tuleneb selle paremast mürakindlusest võrreldes alternatiividega (nt. RS232). Mürakindlus saavutatakse edastades kahe keerupaar-juhtmega diferentsiaalpinget [11]. Delta draivide hulgas on selliseid, mis toetavad kahe juhtmega *high-speed* CAN-i ja kasutavad CANopen protokolli [12].

Delta seadmete valikus on lisaks CAN-siinile ka EtherCAT-iga variante, mille ülekandekiirus on võrreldes CAN-i maksimaalse kiirusega ligi 200 kordne ning mis võimaldab seadmete võrgu keerulisemat ülesehitust, kuid tüüpilises rakenduses ei ole selle kasutamine vajalik ja õigustatud. Kui loodav rakendus on mingil põhjusel aeglane, siis on tõenäolisem, et kiirendamine saavutatakse muude optimeerimisvõtetega, kui

võrgukihi kiirema vastu välja vahetamine, sest niiviisi saadav võit on suurusjärgus mõnikümmend mikrosekundit. Muud optimeerimisviisid, mida käesolev töö käsitleb, võimaldavad paarikümne millisekundilist parendamist operatsioonitsükli kohta.

Eeltoodud parameetritele vastavuses on valitud seadmed, milleks on:

- ASDA-A2-M draiv koos mootoriga
- kontroller DVP-SV2
- kontrolleri CANopen laiendusmoodul DVPCOPM-SL

Teatud rakendustes, kus on tarvis juhtida kolme või enamat servomootorit sama täpsusega, võiks rahalise kulu ja süsteemi mõõtmete vähendamiseks kasutada sama tootja ASDA-M või ASDA-MS seeria draive. Nende seeriate draivid sisaldavad ka kontrolleri funktsionaalsust ning andmeside laiendusmooduleid. M või MS seeria seadmetega ehitatavaid süsteeme on võimalik laiendada käesolevas töös kasutatavate ASDA-A2 draividega. Samuti on ASDA-M draivi võimalik kasutada alluvseadme režiimis ning liita see ASDA-MS poolt juhitavasse süsteemi, mis oleks otstarbekas juhul, kui on vajadus juhtida seitset või enamat mootorit. Alternatiividena toodud seeriad M ja MS sisaldavad ka g-koodi interpretaatorit, mis lihtsustab nende kasutamist tootmispinkide jaoks [13] [14].

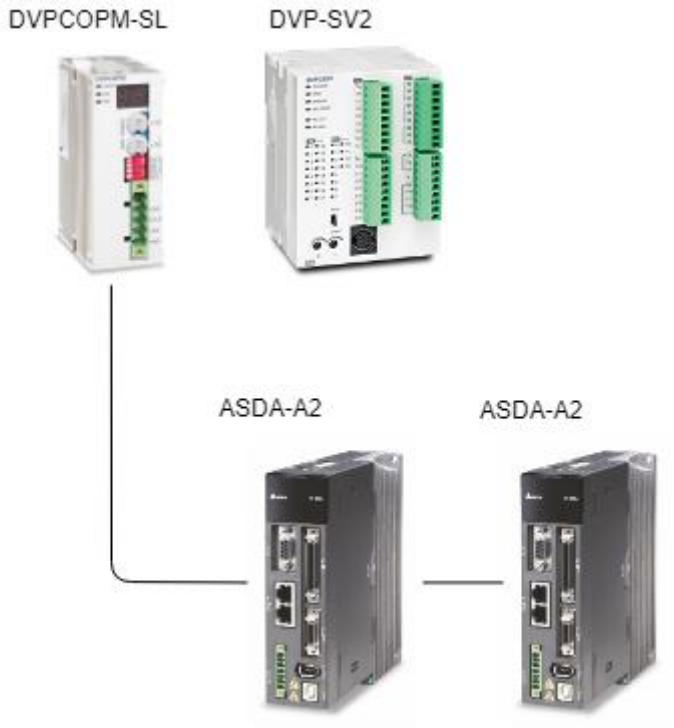
Delta ASDA-A2 seeria mootorite kasutamise eelis võrreldes paljude teiste tootjatega on suuremad tolerantsid mootori ja reduktori paigaldusel, mis tähendab madalamat seadme hinda nii draividele kui lõpptootele, milles neid kasutatakse [9]. ASDA-A2 draivid leiavad kasutust näiteks tootmisseadmetes [9] [15], meditsiinis test-seadmetes (implantaadi survekindluse testimise masin) [16] ja DVP PLC-d on kasutatud muuhulgas ka koduautomaatikas [17].

1.2 Andmesidevõrgu ülesehitus

DVP-SV seeria kontrolleritele mõeldud CANopen laiendmoodul DVPCOPM-SL ühendatakse füüsiliselt kontrolleri küljele. Ülemseadme (*master*) režiimis tööle pandud laiendusmoodul suhtleb võrku paigaldatavate alluvseadmetega (*slaves*) ning vahendab nendelt saadud informatsiooni loogika täiturkontrollerisse. DVPCOPM-SL kontroller töötab vastavuses CANopen-i standardse protokolliga DS301v4.02 [12]. Moodul võimaldab alluvseadmete, käesoleval juhul servokontrollerite, konfigureerimist, juhtimist

ja võrgus olemasolu kontrollimist. Laiendmoodulil on piisavalt andmeobjekte iga alluvseadme kohta, mälu alluvseadmete veasõnumite salvestamiseks. Delta pakutav laiendmoodul loetakse käesolevas töös käsitletava lahenduse saavutamiseks piisavaks.

Alluvseadmed ühendatakse võrgus jadamisi nii, et laiendusmoodulist alguse saav elektriahel on veetud läbi kõikidest draividest, kasutades selleks draivide vahel hästi isoleeritud kahe soonega kaableid. Viimasesse siinil olevasse draivi on võrgu korrektseks tööks ühendatud terminaatorpistik. Taolist ühendusviisi nimetatakse lineaarseks võrgustruktuuriks. Joonisel 1 on toodud süsteemi arhitektuuri vaade.



Joonis 1 Süsteemi arhitektuur

1.3 CAN-võrgu toimimine

Kontrolleri CANopen laiendmoodul ja draivid on ühendatud CAN-võrku. CAN on siinistandard ja protokoll, mis arendati välja ettevõttes Bosch 1980. aastatel eesmärgiga kasutada seda autodes kaabelduseks kuluva materjali vähendamiseks. CAN-i füüsiline kiht koosneb laialdaselt kasutatud ISO 11898-2 ja ISO 11898-3 standardite järgi kahest juhtmest ning lubab vastavalt kiirusi kuni 1mbit/s ja 125kbit/s. Ülekandekiirus sõltub siini pikkusest ning transiiveri kiirusest. Kaht soont nimetatakse CAN_H ja CAN_L ning nendes edastatakse diferentsiaalpinget, mille eelist kirjeldatakse peatükis 2.1. Aeglasem,

ühe liini katkemist tolereeriv CAN lubab kuni 32 seadet võrgus, *high-speed* CAN 110 seadet [18].

CANi standard deklareerib 4 tüüpi kaadreid: *data* kaader (andmekaader, allika juurest andmete edastamiseks ühe või mitme vastuvõtjani), *remote* kaader (päringusõnumi kaader, kasutusel täiendava info pärimiseks vastava sõnumi kohta), *error* kaader (veakaader, võrgus vea tuvastanud sõlm raporteerib vea), *overload* kaader (ülekoormuskaader, millega võrgu sõlm taotleb lisaaega vastamiseks) [18].

CANi võrgu reaalajas toimuva edastuse ja robustsuse tagavad füüsilise kihi ülesehitus, sest kõik seadmed on samale liinile ühendatud, ning võimekus tuvastada vigu ja vigane seade automaatselt välja lülitada. CANi võrgu sõlmed sünkroniseerivad sõnumikaadri teatud biti peale kohalikud kellad, et tagada korrektne bitiajastus. Sõlmede paralleelse edastamise võimalikku konflikti välditakse CAN-võrgus nö vahekohtu ehk arbitreerimise meetodil. Arbitreerimist teostatakse kaadri esimese 11 biti alusel, mis määravad ka sõnumi prioriteedi. Sõlm jälgib kõnelemise alustamisel siini ning kui tuvastab siinilt enda edastatud signaali, jätkab. Kui sõlm tuvastab siinilt dominantse biti (loogiline 0), ise edastades retsessiivset (loogiline 1) bitti, katkestab sõlm edastamise, sest teine sõlm edastab siinil prioriteetsemat sõnumit [18].

Delta seadmes kasutatavad CAN-võrgus suheldes CANopen protokolli, mis kuulub OSI mudeli järgi 7. kihile ehk rakenduskihile. Osaliselt on rakenduskiht defineeritud *CAN in Automation* grupi publitseeritud standarditega, ning CANopen protokoll laiendab rakenduskihi funktsionaalsust. Rakenduskihi funktsionaalsused on muuhulgas ka kõnealuses näidissüsteemis kasutatav Network Management (NMT), mille abil ülemseade juhib alamseadmete käivitamist, ning CANopen Object Dictionary (OD), mis võimaldab erisuguste andmeobjektide defineerimist. Andmeobjektide edastuseks on defineeritud PDO-d (*Process Data Object*) ja SDO-d (*Service Data Object*). Andmeobjektide edastamise tingimiseks teatud edastustüüpide puhul tingib SYNC (sünkroniseerimise) sõnum, mida saab ka käesoleva töö raames loodavas süsteemis kasutada [19].

2 Süsteemi konfigureerimine ja juhtimine

Järgnevalt käsitletakse rakenduse andmesidevõrgu konfigureerimisele draivide ja mootorite juhtimise funktsionaalsuse saavutamise seisukohalt.

2.1 Võrgu seadistus

Lähtuvalt eeltoodud sõnumivahetuse arbitreerimise kirjeldusest, ei tohi CAN-võrgu vigadeta toimimiseks mitu seadet kasutada sama seadme identifikaatorit sõnumis. Igale ASDA-A2 draivile määratakse eraldi identifikaator (*node ID*), mis salvestub seadme püsimällu ning määrab 7 bitti CANi kaadri 11-bitisest tuvastamisväljast (COB-ID). Täiendavalt tuleb defineerida võrgus liikuvad sõnumid, mida alamseadmed saadavad ülemseadmele ja ülemseadmed alamseadmetele tavalise töö käigus, ehk protsessi andmeobjektid (*process data objects* ehk PDO). Samuti on Delta DVPCOPM-SL moodulit kasutades võimalik määrata alamseadme seadistamiseks mõeldud sõnumid (*service data objects* ehk SDO), mis saadetakse alamseadmetele automaatselt nende võrgus tuvastamisele järgnevalt.

Töö käigus saadetavate protsessi andmeobjekte sisaldavate sõnumite ehk PDO-de seadistus on kasulik võrgu töökoormuse vähendamiseks, sest võrreldes SDO sõnumite 4 baidiga sisaldavad need 8 baiti kasulikke andmeid. Eeldefineeritud struktuuri tõttu ei pea sõnumid sisaldama infot saadetava andme tüübi kohta. PDO tüüp on seadistatav. Näiteks saab PDO saatmise ajendiks määrata siinil liikuva SYNC sõnumi, taimeri täis loendamise või andmete muutumise [19]. Seega ei ole ülemseadmel vajalik andmeid eraldi küsida. Samuti on võimalik PDO-dele määrata prioriteet, ning vastavalt arbitreerimise tööpõhimõttele on kõrgema prioriteediga sõnumit edastaval seadmel kokkupõrke korral edastamise eesõigus.

CAN-võrgu topoloogia määratakse koos protsessi andmeobjektide sõnumite definitsioonidega Delta CANopen Builder programmiga [12].

2.2 Automaatsed seadistusobjektid

Draivile õigete toimimisparameetrite määramiseks on ASDA seeria draivide puhul mitu varianti, sealhulgas olemasoleva seadistusfaili kodeerimine seadmesse üle USB

haldusliidese, seadme paneelilt või USB haldusliideselt parameetrite käsitsi määramine ning üle CAN-i võrgu konfigureerimine. CAN-i võrgu kaudu seadistamiseks kasutatakse SDO-sid ehk seadme halduseks ja seadistamiseks mõeldud sõnumeid. DVPCOPM-SL nimetab ühenduse tekkel alamseadmele saadetavaid seadistussõnumeid automaatseteks SDO-deks (*auto SDO*).

Prototüübi realiseerimisel on lihtsam kasutada toodud variantidest haldusliidest ASDA-Soft, mis USB ühenduse olemasolul võimaldab lihtsalt välja lugeda draivi konfiguratsiooni ning väärtusi modifitseerida. Lisaks võimaldab mainitud haldustarkvara teostada ka *jog*i ehk aeglast kontrollitud liikumist ning automaatset liikumishäälestust *auto-gain tuning* liikumisparameetrite määramiseks.

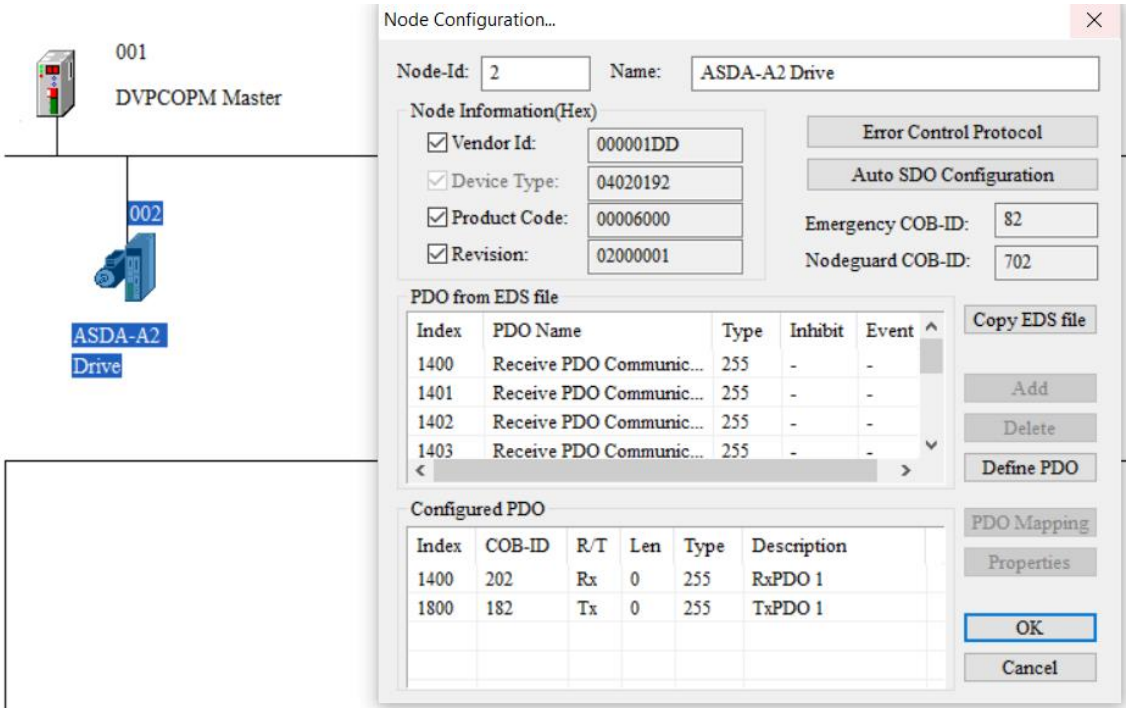
Esialgse seadistuse teostamise ja katsetamise järgselt võib liikumise häälestuse parameetrid seadistada automaatselt SDO sõnumitega saadetavaks. Taolise lahenduse peamine eelis on kiirem draivide vahetamine või suurem töökiirus lahenduse tootmisel, kui haldusliidese ühendamise asemel tuleb draivile vaid tema identifikaator määrata ning kõik opereerimiseks vajalikud parameetrid saadetakse selle ühendamisel CAN-võrku.

Lisaks füüsilist liikumist määravatele parameetritele (*gear ratio, feed constant*) on taolises rakenduses sageli tarvilik määrata ka lähtestamise (*homing*) ehk nullanduri otsingu teostamise meetod ning kiirus. ASDA draivil saab nullasendi otsingut teostada draivi sisendi või lõpplüliti signaali järgi ning lähtuvalt seadme füüsilisest ehitusest võib tekkida vajadus modifitseerida signaali või selle serva otsingu liikumise kiirust. Täiendavalt võib kiiret positsioneerimist nõudvates rakendustes, kui rakenduse mehaanilised tolerantsid seda lubavad, määrata suurem sihtpositsiooni aken (*position window*). Sihtpositsiooni aken määrab ala sihtpositsiooni (*target position*) ümber, mida draiv peab sihtasendina aktsepteeritavaks. Seeläbi võib liikumise lugeda lõpetatuks, sest positsioon on piisavalt saavutatud, kuigi draiv jätkab veel täpset koordinaadile positsioneerimist. Lisaks saab SDO sõnumitega ära määrata ka draivi väljundid, mille näidisrakendus on mootori piduri juhtimine või sihtpunkti saavutamise (*target reached*) signaali edastamine, ning sisendid, eesmärgiga näiteks määrata hädaseiskamise signaal, lähtestusanduri (*home sensor*) signaal või valida jõulimiit.

2.3 CANopen Builderi kasutamine võrgu seadistamiseks

Delta CANopen Builderi võimaldab luua võrgu topograafia ja programmeerida see DVPCOPM-SL laiendmoodulile kasutades DVP-SV2 kontrolleril asuvat RS232 või RS485 porti [12]. Võrgu topoloogia kirjeldamine CANopen Builderi tarkvaras on vajalik, sest ülemseade (kontrolleri) peab olema teadlik, mis alamseadmete (draivide) olemasolu võrgus oodata. Topograafiat kirjeldades võimaldab programm konfigureerida draividega suhtlemisel kasutatava PDO sõnumite järjekorra ja formaadi. Lisaks võimaldab tarkvara seada ka *auto-SDO*-d, võimaldades sealjuures realiseerida peatükis 3.2 mainitud lähenemist, kus kogu liikumisparameetrite hulk saadetakse draivi sellega esmase ühenduse loomisel. Antud seadmete puhul peab piirduma maksimaalselt 20 automaatse SDO-ga.

Olles avanud CANopen Builder tarkvara, luuakse uus projekt menüüst *File -> New*. Lehe keskele ilmuvale tühjale liinile saab lisada seadmeid valides need lehe vasakul servas asuvast menüüst *Device*. Käesoleva töö kontekstis valitakse seade "DVPCOPM Master" menüüst Master ning seadmed "ASDA-A2 Drive" menüüst Slave -> Vendor -> DELTA ELECTRONICS, INC. Joonisel 2 on toodud sõlme konfigureerimise vaade, mis avaneb topeltklikkides sõlmel.



Joonis 2 Näidis topoloogiast ühe lisatud draiviga

Loodud seadistuse üle kandmiseks kontrollerisse seadistatakse ühendus menüüst Setup -> Communication Setting -> System channel. Seejärel valitakse menüüst "Online" või vajutatakse F10.

2.4 Andmeobjektide struktuuri loomine

2.4.1 Juhtimisalgoritmis kasutatavad andmeobjektid

PLC ja draivide vahelises suhtluses tuleb kasutada andmeobjekte, mis võimaldavad täita eesmärki juhtimaks servomootorit impulssanduri tagasiside alusel. Peamine andmeobjekt draivist tagasiside saamiseks on staatusesõna (*statusword*) ning andmeobjekt draivi töö juhtimiseks on juhtsõna (*controlword*), mille bittide tähendused on toodud käesoleva töö osades LISA 1 ja LISA 2.

Lisaks eelmainitud staatuse- ja juhtsõnale tuleb draivile selle liikuma panemiseks saata ka juhtimisrežiim (*modes of operation*), positsioneerimise omadused (kiirus ehk *target velocity*, kiirenduse kestus ehk *acceleration* ja aeglustus ehk *deceleration*) ning sihtkoordinaat (*target position*). Draivi liikumisest tagasiside saamiseks tuleb andmeobjektide struktuuri koostamisel lisaks staatusesõnale draivist kontrollerisse saata ka alarmi numbriline väärtus. Sõltuvalt rakendusest võidakse lisada ka nt reaalne positsioon (*position actual*), kui on vaja jälgida masina asendit, jpm väärtused.

Mootorile voolu rakendamiseks ja draivi töö alustamiseks tuleb juhtsõna esimesed neli bitti (*switch on, enable voltage, quick stop (b-contact), enable operation*) kõrgeks kirjutada. Liikumise alustamiseks, olles liigutamiseks vajalikud parameetrid (kiiruse profiil ja sihtkoht) servokontrollerisse kirjutanud, tuleb 4. bitt (*new set-point (positive trigger)*) kõrgeks kirjutada. Kui rakendus soovib käimasoleva liikumise kestel anda uue liikumise käsu, tuleb 5. bitt (*change set immediately*) väärtustada kõrgeks. Selline liigutamine toimib positsiooni järgi juhtimise režiimis (*profile position*), mille saab aktiveerida kirjutades juhtimisrežiimi muutuja 1. biti kõrgeks. DVPCOPM-SL mälujaotuse omapära tõttu tuleb 1. kõrge bitt dubleerida ka teise baiti, ehk kümnendsüsteemi väärtus juhtimisrežiimi muutujal modes of *operation* on 257.

Töötlemise järjekord ja juhtimise põhimõtted on kirjeldatud peatükis 4.

2.4.2 PDO struktuuri määramine CANopen Builderis

Peatükis 2.4.1 toodud juhtimiskirjeldusest lähtuvalt on tarvis koostada PDO sõnumite struktuur, määrates sellesse liikumise saavutamiseks ja liikumise tagasiside kontrollimiseks vajalikud andmeobjektid.

Draivile käsu andmiseks modifitseerib kontrolleri kood draivi juhtimiseks mõeldud meediumit ehk juhtsõna (*controlword*). DVPCOPM tuvastab muutuse kontrolleri PDO-sse kuuluva muutuja väärtuses ning saadab PDO. TxPDO (*transmit PDO*) tüüpi sõnumitesse defineeritavad andmeobjektid saadetakse draivist kontrollerisse, RxPDO (*receive PDO*) sõnumid saadetakse kontrollerist draivi. Liikumise saavutamiseks konfigureeritakse juhtseadme saadetavate PDO-de ehk alamseadmete vastuvõetavate PDO-de (RxPDO) hulka järgmised parameetrid:

- target position (sihtasendi impulssanduri väärtus)
- target velocity (sihtasendile liikumise kiirus PUU ühikutes)
- target acceleration (sihtasendile kiirendamise ajaline kestus)
- target deceleration (sihtasendile jõudes rakendatava aeglustuse ajaline kestus)
- modes of operation (juhtimisrežiim)
- controlword (juhtsõna)

PDO-de struktuur defineeritakse joonisel 2 toodud sõlme konfigureerimise aknas. Aktiivsed PDO-d, mis seadmesse konfigureeritakse, on toodud "Configured PDO" segmendis. Sinna segmenti saab lisada uue PDO, valides segmendist "PDO from EDS file" PDO ning vajutades "Add". PDO-sse kuuluvaid andmeobjekte saab lisada topeltklikkides PDO kirjel "Configured PDO" segmendis.

Eelnimetatud liigutamiseks vajalikud parameetrid lisatakse PDO-desse nii, et juhtsõna on madalama prioriteediga sõnumis. See võimaldab kontrolleril samas tsüklis saata kõik positsioneerimiseks vajalikud andmeväljad ning võimaldab olla kindel, et kui madalama prioriteediga juhtsõna on draivi jõudnud, on sinna jõudnud ka kõrgema prioriteediga positsioon, kiirus, kiirendus, aeglustus.

Opereerimise tagasiside saamiseks konfigureeritakse draivide TxPDO-de hulka järgmised andmeobjektid:

- statusword (staatusesõna)

- alarmi number
- tegelik positsioon (position actual)
- controlword-i (juhtsõna) tagasiside
- modes of operation tagasiside

Tabel 1 PDO tabeli näidis

Indeks	COB-ID	R/T	Baite	Tüüp	Kirjeldus	Objektid
1400	202	Rx	8	255	RxPDO 1	Profile acceleration Profile deceleration
1401	302	Rx	8	255	RxPDO 2	Target position Profile velocity
1402	402	Rx	4	255	RxPDO 3	Controlword Modes of operation Modes of operation
1800	182	Tx	4	255	TxPDO 1	P0-01 (alarm) Statusword
1801	282	Tx	4	255	TxPDO 2	Controlword Modes of operation Modes of operation
1802	382	Tx	4	255	TxPDO 3	Position actual value

Tabelis 1 on toodud näidisrakenduseks loodud PDO-d. RxPDO 3 ja TxPDO 2 sisaldavad "Modes of operation" (juhtimisrežiim) andmeobjekti topelt seetõttu, et kuigi väärtuse esitamiseks kasutatakse ühte baiti (8 bitti), saab kontrolleris lihtsamini adresseerida 16-bitiseid WORD tüüpi muutujaid ning ka järgnevad muutujad ei lähe nihkesse. TxPDO 3-le, mis sisaldab draivi tegelikku impulssanduri tõlgendamisest saadavat positsiooniväärtust, seatakse ka *inhibit timer* menüüst "Properties", väärtusega näiteks 100ms. Selliselt välditakse liigselt positsiooniväärtustega sõnumi saatmist, sest liikumise käigus uueneb positsioon tihti, kuid samas saadakse positsiooniväärtus tagasisideks piisavalt tihti.

3 Kontrolleri tarkvara

Järgnevalt antakse ülevaade PLC programmide tööpõhimõtetest ja andmesidevõrgus edastatavate objektide mäluaadresside määramisest.

3.1 Sissejuhatus

PLC programmi kirjutamiseks kasutatav tarkvara on sageli tootja-spetsiifiline [6] [7]. Delta DVP-seeria kontrollerisse tarkvara kirjutamiseks kasutatakse tarkvara Delta ISPSoft, mis on sisuliselt IDE (*integrated development environment*) ehk keskkond tarkvara kirjutamiseks, kompileerimiseks, simuleerimiseks ja PLC-sse laadimiseks. Lisaks võimaldab ISPSoft tarkvara töö jooksvat seiramist ehk muutujate väärtuste vaatamist ja väärtustamist [20]. Organisatsiooni PLCopen standard IEC 61131-3 sisaldab viit levinud PLC-de programmeerimiskeelt: SFC (Sequential Function Chart), LD (Ladder Diagram), FBD (Function Bock Diagram), IL (Instruction List), ST (Structured Text) [5]. Kuigi ISPSoft ja osad Delta teiste seeriate kontrollerid toetavad neid kõiki, on DVP seerial tugi SFC, LD ja IL jaoks [21].

Programmikeel, mida kasutada, tuleb valida sõltuvalt rakenduse planeeritavast keerukusest ja programmeerija isiklikust eelistusest. Autori hinnangul võib LD kood, kuigi see annab edasi selgelt täitmisjärjekorra, muutuda suurema koodi korral raskemini loetavaks ning IL on sellisel juhul klassikalise treppimisega lihtsam kasutada. Pascali programmeerimiskeelele sarnanev ST on loodud pidades silmas tootjateülest süntaksit ja võimalust kasutada rohkem *high-level programming* elemente [5] [11]. Käesolevas töös kasutatakse programmeerimiseks käsulisti (*instruction list*i).

PLC koodi täidetakse ehk käivitatakse tsükliliselt või katkestustega (*interruptions*). Ühe tsükli täitmise kestus on programmitsükli täitmise aeg (*scan time*) [5] [11]. See võimaldab erinevatele koodi osadele anda eri prioriteete. Näidisrakendustes leidub olukordi, kus teatud olukordadele, näiteks diskreetse sisendi muutumisele, on vaja reageerida koheselt, sõltumata tarkvara tsükli täitmise aja pikkusest. Sellistel juhtudel on tarvis kasutada katkestusi. Katkestusi saab DVP-SV2 kontrolleri puhul seadistada rakenduma ka teatud diskreetsete sisendite peale [21]. PLC programmitsükkel koosneb toodud järjekorras sisendite info laadimisest programmis kasutatavatesse muutujatesse, rakenduse

programmi käivitamisest, väljundite lülitamisest ja muudest tegevustest nagu sisemiste loendurite ja taimerite uuendamisest [7].

Programmikoodis on võimalik lähtuvalt seadme eesmärgist teostada erinevaid operatsioone. Erinevate instruktsioonidega luuakse tingimused (näiteks diskreetse sisendi kontroll loogikavõrdlusega), mille täitumisel täidetakse koodi (näiteks lülitatakse diskreetset väljundit). Dokumendis DVP-PLC Application Manual peatükis 5.5 on toodud kasutatavate instruktsioonide nimekiri [21]. Järgnevalt on toodud programminäidis, milles sisendi X1 väärtuse tõusmise peale aktiveeritakse loendamine ja X1 langemise peale see deaktiveeritakse, salvestades loenduri väärtuse.

```
ldp X1 (* rising edge detection *)
    set M1
ldf X1 (* falling edge detection *)
    rst M1
    dmov D1000 D1002 (* save last count value to D1002 *)
ld M1
   dinc D1000 (* dword increment, counter takes up memory D1000 and D10001 *)
```

Joonis 3 Käsulisti programminäidis

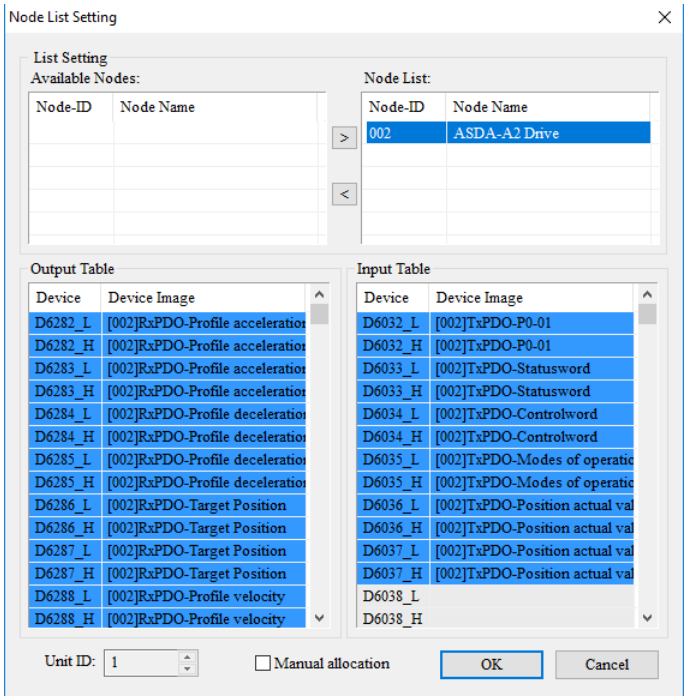
Toodud näites ei kasutata muutujaid, vaid viidatakse näite lihtsuse ja mälutüüpide kuvamise huvides otse mäluaadressidele. Levinuimad ja käesolevas töös kasutatud instruktsioonid on toodud töö osas LISA 3.

ISPSoftis uue programmi loomiseks on vaja luua POU (Program Organisation Unit) tehes lehe vasakul servas olevas menüüs "Programs" peal parema kliki ja valides "New...". Avanevas aknas tuleb programmile anda nimi, valida keel (käesoleva töö kontekstis käsulist) ja anda POU-le tüüp. Tüübiga "Cyclic Task" programme käivitatakse programme selles järjekorras, milles nad vasakul menüüs "Programs" all loetletud on. ISPSoftis koodi kompileerimine ja selle kontrollerisse laadimine toimub ülemise menüü valikute "Compile" ja "PLC" -> "Transfer" -> "Upload" kaudu.

3.2 Andmeobjektide aadresside määramine

Peatükis 2.4.2 määratud PDO-de mälus paiknemine ja sõnumistruktuur kandub üle sellesse, mis mäluaadressidele CAN laiendmoodul neid iga tsükli alguses kopeerib.

Selleks, et muutujad oleks mäluaadressidel ligipääsetavad, tuleb Delta CANopen Builderis menüüst valida Network -> Node List. Avanevas aknas on "Available Nodes" segmendis loodud sõlmed, mis tuleb aktiveerida peale vajutades. Aktiveerimisjärgselt saab noole nupule vajutades draivi kopeerida aktiivsete sõlmede nimekirja. Järgneval joonisel on näidatud tulemus pärast ühe sõlme lisamist aktiivsete sõlmede nimekirja.



Joonis 4 CANopen aadressid

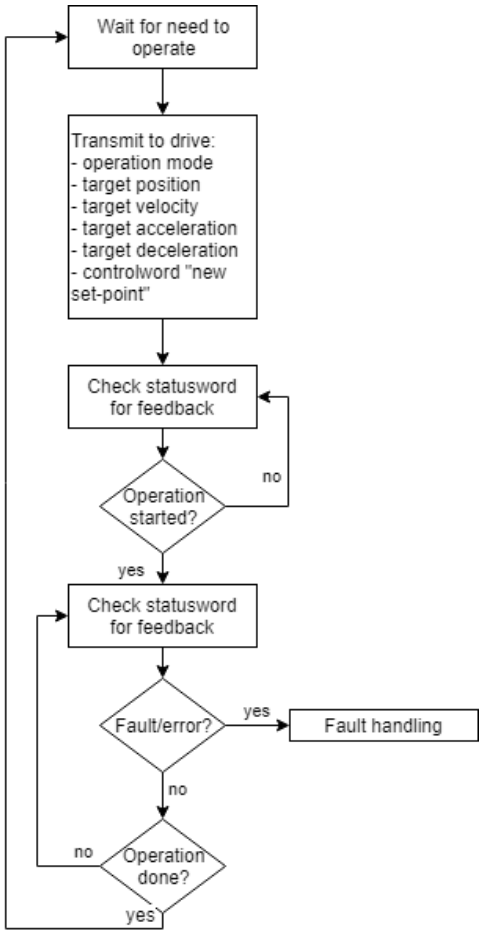
Tabelites "output table" ja "input table" on toodud vastavalt kontrollerist draivile saadetavad ja draivist kontrollerile saadetavad andmeobjektid. Väljal "device" toodud väärtus on mäluaadress ning ühe muutuja kirjete arv viitab sellele, kui mitu baiti see andmeobjekt kasutab. Kahebaidised andmeobjektid kasutavad ühe D-registri ja tuleb muutujaid lisades deklareerida WORD tüübiga, neljabaidised kaks D-registrit ning tuleb deklareerida DWORD tüübiga. ISPSoftis muutujaid (symbols) nimetades on hea tava nimetada saadetavad muutujad eesliidesega *tx_* ning sissetulevad muutujad eesliidesega *rx_*. Toodud tabelist näitena defineeritakse muutuja *tx_acceleration* aadressile D6282 tüübiga DWORD ja *rx_alarm* aadressile D6032 tüübiga WORD.

4 Servokontrolleri juhtalgoritm

Järgnevalt võrreldakse erinevaid variante draivi juhtimiseks valitud seadmetega ning valitakse neist tõhusaim ja töökindlaim. Seejärel realiseeritakse algoritm programmikoodis. Samuti kirjeldatakse mitme draivi juhtimisalgoritmi, teostatakse selle eelduseks olevad signaliseerimismuutujad ja algoritm ise programmikoodis. Rakendamise näidisena käsitletakse kahe eri tüüpi käsu andmist kahele teljele nii paralleelselt kui vahelduvalt.

4.1 Ühe draivi juhtalgoritmi mudel

Juhtimisalgoritm peab realiseerima draivi käsu saatmise ning käsu tagasiside ehk vea tekkimise või positsioneerimise eduka lõppemise tuvastamist. Joonisel 5 esitatakse kontseptuaalne mudel ühe draivi tavapärase töö juhtimisalgoritmile.



Joonis 5 Juhtalgoritmi mudel

Juhtimisalgoritmi realisatsioonis tekib valikukoht selles, et mil moel tuvastatakse, et draiv on pärast sellesse käsu alustamist määrava juhtsõna kirjutamist käsu tegelikult vastu võtnud ning asunud seda täitma. Erinevaid variante kaardistavad järgnevad alapeatükid.

4.1.1 Positsioneerimise algoritmi variant A

Lähtuvalt staatusesõna bittide definitsioonist ja tootja dokumendi näitest kasutab algoritmi variant A staatusesõna bitti *set-point acknowledgement* selleks, et tuvastada, kas draiv on käsu kätte saanud või mitte [22]. Käesoleva töö kirjutajale teadmata põhjusel ei modifitseeri draiv selle biti olekut positsiooni režiimis. Autor uuris CANi sidet logimisseadmega ning ei tuvastanud selle biti muutust, ning tõdeb, et on võimalik, et selle biti kasutuselevõtt nõuab täiendavad konfigureerimist. Seetõttu jääb esmalt kõige loogilisemana tundunud variant kasutamata.

4.1.2 Positsioneerimise algoritmi variant B

Teine lahendusvariant on käsu alustamise tagasisideks kasutada staatusesõna 10. bitti *target reached*. Selle biti väärtus on positsioneerimise režiimis pidevalt "1", kui draiv ei liigu uuele asendile. Kui draiv liigub uuele asendile, on selle biti väärtus "0". Taoline lahendus on töötav, kuid lahenduse keerukust tõstab see, et kui draiv ei ole lähtestatud (nullasend on määratlemata), on biti väärtus "0" ka siis, kui draiv ei liigu uuele asendile, seega on tarvis programmikoodis neid olukordi (positsioneerimine pooleli vs lähtestus tegemata) eristada. Täiendav ohukoht on suure programmikoodi või väga väikeste liikumisdistantside korral see, et kui ühe kooditsükli täitmise aeg on pikem kui ühe liigutamise kestus, võib see bitimuudatus jääda märkamata. Sõnum jõuab CAN-võrgus küll DVPCOPM-SL moodulisse, kuid järgmine sõnum jõuab enne järgmist programmikoodi tsüklit, ning vahepealne draivi staatusesõna muudatus jääb märkamata.

4.1.3 Positsioneerimise algoritmi variant C

Seni erinevates olukordades (eri näidisrakenduste korral) enim töökindlust omav lahendus on selline, kus draivile RxPDO-s saadetav juhtsõna on määratud ka draivi poolt välja saadetavasse TxPDO-sse. Sellise lahenduse korral saab PLC teada, kui draiv on saanud uue *controlword*i ning selle töödelnud, ning PLC programm saab jätkata toodud tegevustega:

1) Algoritmi järgmise käivitamiskorra optimeerimiseks muuta juhtsõnas biti "new set-point" väärtus tagasi 0-ks (draiv alustab positsioneerimist selle biti *rising edge trigger*i peale, ning muutes eelmise liigutuse kestel biti väärtuse madalamaks, saadakse järgmise liikumise tööd kiirendada).

2) Jälgida staatusesõnas bitti "target reached" tuvastamaks käsu lõpetamist.

3) Jälgida staatusesõnas bitti "fault" või alarmi väärtust tuvastamaks ebaõnnestumist käsu täitmisel.

Käesolev lahendus võimaldab tuvastada ka CAN-võrgu side probleeme, kui näiteks draiv ei saa kätte uut juhtsõna, on võimalik PLC-l see olukord tuvastada ja töö peatada. Samuti ei teki võimalust, et PLC ei registreeri draivipoolset käsu alustamist.

4.2 Ühe draivi juhtimisalgoritmi realisatsioon

Valitud algoritmi (C, peatükk 5.1.3) teostades tuleb programmikood segmenteerida lähtuvalt draivi poolt teostatavatest operatsioonidest (liikumistest). Iga samm liikumise teostamisel teostatakse eraldi käsu faasis. Joonisel 6 on toodud näidiskood liikumise alustamiseks:

```
ld= state 200
    dmov 2500000 tx_targetposition (* target encoder position *)
    dmov 5400000 tx_velocity (* 3000 RPM *)
    dmov 1000 tx_acceleration (* 1 second to achieve full speed *)
    dmov 1000 tx_deceleration (* 1 second to decelerate to a stop *)
    mov 257 tx_opmode (* also 2#100000001 *)
    mov 63 tx_controlword
    and= rx_feedback_opmode 257
    and= rx_feedback_controlword 63
        (* positioning started *)
```

Joonis 6 Näidiskood liikumise alustamiseks

Näidiskoodis kasutatavad muutujad on deklareeritud ISPSoft-is programmi "Local symbols" segmendis, ehk on kasutatavad ühe programmi piires. Muutujate tüüp on valitud WORD või DWORD (*doubleword*) vastavalt sellele, kas aadresside tabelis (peatükk 4.2) võtab muutuja ära vastavalt 2 baiti või 4 baiti. Muutujad

rx_feedback_opmode ja *rx_feedback_controlword* viitavad draivilt saadavale tagasisidele. Tagasiside on aluseks algoritmi töö jätkamisele, sest viitab sellele, et servokontrolleri protsessor on oma kohalikud muutujad väärtustanud sellele saadetuga, ning kui alarmi/viga ei esine, siis töötleb neid. Näidiskoodis toodud väärtused kümnendsüsteemi kujul võib loetavuse huvides asendada ka kahendkujuga, näiteks 15 => 2#1111 või 257 => 2#100000001.

Kuna programmikoodi käivitamine on tsükliliselt ning tsükli pikkus varieerub sõltuvalt programmikoodi suurusest, tuleb arvestada, et ülaltoodud koodis väärtustatakse muutujad *tx_opmode* ja *tx_controlword* mitme tsükli jooksul. Toodud nüanss oleks oluline, kui PLC kood ise peaks otsustama, millal saadetakse draivi PDO, seega seda tuleb silmas pidada teatud riistvarale arendades, aga DVPCOPM-SL laiendmooduli kasutamise korral otsustab moodul ise, millal on vaja saata uuenenud PDO [12]. Kui tarkvara käivitamise tsükli lõpus on PDO-s saadetavate andmete väärtus sama, mis viimases PDO-s saadetud, siis asünkroonseks konfigureeritud PDO puhul sõnumit välja ei saadeta..

Tsükliline täitmine ja tingimuste kontroll lisab tarkvara kirjutamisel ohukoha, mis väljendub ka eeltoodud koodis, kus sama algtingimuse jätkuks tehakse operatsioonid, mis nullivad esialgse tingimuse.

```
ld= state 200
    (* omitted copying data *)
    mov 257 tx_opmode
    mov 63 tx_controlword
    and= rx_feedback_opmode 257
    and= rx_feedback_controlword 63 (* RIDA 5 *)
        mov 15 tx_controlword (* drive back to waiting mode – optimization *)
        and= rx_feedback_controlword 15 (* RIDA 7 *)
            (* start checking for "target reached" bit *)
             mov 201 state
```

Joonis 7 Täidetamatu koodi veaolukorra näidis

Joonisel 7 on tegemist koodiga, mida kunagi ei täideta (*unreachable code*), registri *state* väärtus ei muutu kunagi 201-ks. Muutuja *tx_controlword* väärtustamisel 15-ks saadetakse draivi PDO ning järgmistel tsüklitel kontrollitakse rx_feedback_controlword muutuja võrdumist 15-ga. Kui draiv saadab PDO, millega muutub *rx_feedback_controlword* 15-

ks, ei ole järgmisel tsüklil enam real 5 olev võrdlus 63-ga korrektne ja koodi täitmisel ei jõuta enam rx_feedback_controlword võrdlemiseni 15-ga real 7. Staatilise analüüsi vahendid on piisavalt pädevad paljusid levinud vigu kontrollima [23]. Antud juhul on tegemist rakenduse spetsiifikaga, mida ei tuvasta ka ISPSoft kompileerimisel. Joonisel 8 on toodud lahendus veaolukorrale (tx_controlword väärtustamise asemel minnakse järgmisesse faasi).

```
ld= state 200
    mov 257 tx_opmode
    mov 63 tx_controlword
    and= rx_feedback_opmode 257
    and= rx_feedback_controlword 63
        mov 201 state
```

Joonis 8 Täidetamatu koodi vältimine sammude segmenteerimisega

Käsu järgmises faasis (201) kirjutatakse süsteemi töö optimeerimise eesmärgil juhtsõna 4. ja 5. bitt madalaks. Sellega tagatakse, et järgmist käsku andes on juhtsõna 4. bitt (*new set-point*) madal, siis ei pea seda kirjutama esialgu madalaks ja uuesti kõrgeks. Kirjeldatud operatsioon on vajalik, kuna draivi protsessor alustab liikumiskäsu 4. biti *rising-edge trigger*i peale. Et draiv ei seiskuks ja tööd ei lõpetaks, peavad esimesed neli bitti (*switch on, enable voltage, quick stop (b-contact), enable operation*) jääma kõrgeks.

```
ld= state 201
    mov 15 tx_controlword
    and= rx_controlword_feedback 15
    band rx_statusword 10
        mov 0 state
```

Joonis 9 Sammu 201 kood

Joonisel 9 toodud koodis, kui staatusesõna (*rx_statusword*, mis on samuti väärtustatud viitamaks DVPCOPM-SL lisamooduli poolt automaatselt muudetavale mälualale) 10. bitt (*target reached*) on kõrge ehk draiv on töö lõpetanud ja juhtsõna tagasiside on võrdne viimati draivi saadetud väärtusega ehk draiv on käsule reageerinud, loetakse käsk lõpetatuks. Käsu lõpetamise tingimuseks loetakse muutuja *state* väärtuse muutumine tagasi nulli.

Igas rakenduses on vajalikud rutiinid draivi töökorda seadmiseks (*alarm reset*) ja impulssanduri väärtuse järgi juhtimise korral ka nullanduri leidmiseks (*homing*). Alarm reseti käsk nimetatakse käesolevas näidises 50-ks ja lähtestamise käsk 100-ks. Reset toimib seades juhtsõna 8. bitt (*fault reset)* kõrgeks ning reseti toimimise edukuse tingimuseks on staatusesõna 2. bitt (*operation enabled*) kõrge olek ning vigadele viitavate bittide (3. bitt ehk *fault* ja 13. bitt ehk *following error*) madal olek. Lähtestamise korral kasutatakse juhtimisrežiimi *homing mode*, ehk tx_opmode väärtus on 1542.

4.3 Mitme draivi vahelduvjuhtimise algoritm

Olles realiseerinud ühe draivi juhtimisalgoritmi, tuleb sellele sarnanevaid realisatsioone rakendada ka teisi draive juhtivates koodi programmides. Eelnevalt toodud algoritmi realisatsioon rakendub telje *Telg1* juhtimiseks, ning sarnane järgnevus tuleb teostada ka teisele teljele *Telg2*, mis samuti teostab impulssanduri väärtuse järgi asendile liigutamist. Teisele teljele algoritmi lisamiseks luuakse uus programm (ISPSoftist Programs peal parem klikk, misjärel valitake "New") ning kopeeritakse üle kood, ning lisatakse muutujad (*state, rx_..., tx_...*) uutele mäluaadressidele. CANopen laiendmoodulist tulenevad aadressid tuleb seada vastavalt teostatud CANopen Builderi konfiguratsioonile.

Vahelduvjuhtimise algoritm, mida näidisrakenduste jaoks realiseeritakse, paneb *Telg2* programmi täitma kohe, kui *Telg1* programm on oma käsu täitmise lõpetanud. See algoritm peab olema realiseeritud eraldi programmis.

4.3.1 Telgede signaliseerimisvõimekuse teostus

Telgede *Telg1* ja *Telg2* programme vahelduvalt käivitada võimaldav algoritm tuleb realiseerida eraldi programmis, mille loomisel toimitakse sarnaset *Telg2* programmi loomisele eelmises peatükis. Kompileerimisel ISPSoft küll tõstab koodi taas kokku, kuid loetavuse ja täiendavate telgede lisamisel keerukuse madalal hoidmiseks luuakse uus algoritm eraldi programmina. Loodav algoritm peab telje programmide töö alustamiseks ja tagasiside kontrollimiseks kasutama globaalseid muutujaid, mis erinevalt kohalikest sümbolitest (*local symbols*) on ligipääsetavad kõigist programmidest. Globaalsete muutujate loomine toimub projektimenüüs leitavas "Global Symbols" aknas.

Eesmärgiga telje programmi juhtida tuleks defineerida muutujad teljele numbrilise käsu andmiseks, selle käivitamiseks ja tagasiside saamiseks nii õnnestunud kui ka

ebaõnnestunud käsu täitmisel. Selleks deklareeritakse niiöelda signaliseerimismuutujad Telg1Command (WORD), Telg1Do (BOOL), Telg1Done (BOOL) ja Telg1Fail (BOOL) ning lisatakse käsu vastu võtmist võimaldav programmi osa Telg1 programmi algusesse (joonis 10).

```
ld Telg1Do
    rst Telg1Do
    and= state 0
        rst Telg1Fail
        rst Telg1Done
        mov Telg1Command state
```

Joonis 10 Signaliseerimismuutujate teostus

Selles osas reageerib telje juhtprogramm käsule ning kopeerib käsu numbrilise väärtuse oma *state* muutujasse, alustades sellega käsu täitmist. Toodud koodile tuleks lisada ka antud käsu valiidsuse kontroll ja veaga lõpetamine juhul, kui varasemalt juba käib käsu täitmine. Seejärel lisatakse programmi lõppu käsu eduka täitmise korral täidetav tingimus, mis käsu eduka lõpetamise korral väärtustab *Telg1Done* 1-ks ja veaga lõpetamise korral väärtustab *Telg1Fail* 1-ks. See on toodud joonisel 11.

```
ld= state 0
    set Telg1Done
ld= state -1
    set Telg1Fail
```

Joonis 11 Signaliseerimismuutujate väärtustamine käsu (eba)õnnestumisel

Kuna programmi töö käigus soovitakse üldiselt tuvastada ka võimalikke vigu ja seetõttu ka ülemprogrammi täitmine lõpetada, väärtustatakse vea tekkimisel iga käsu täitmisel state -1-ks (joonis 12).

```
ld> state 0
band rx_statusword 3
    mov -1 state
```

Joonis 12 Käsu lõpetamine draivi vea korral

Koodi lisandub käivitusplokk, mille tingimus on täidetud vaid ühel tsüklil masina käivitamisel. Erieesmärgiline register M1002 on väärtustatud tõeseks vaid esimesel tsüklil [21] ning käivitusplokk on toodud joonisel 13.

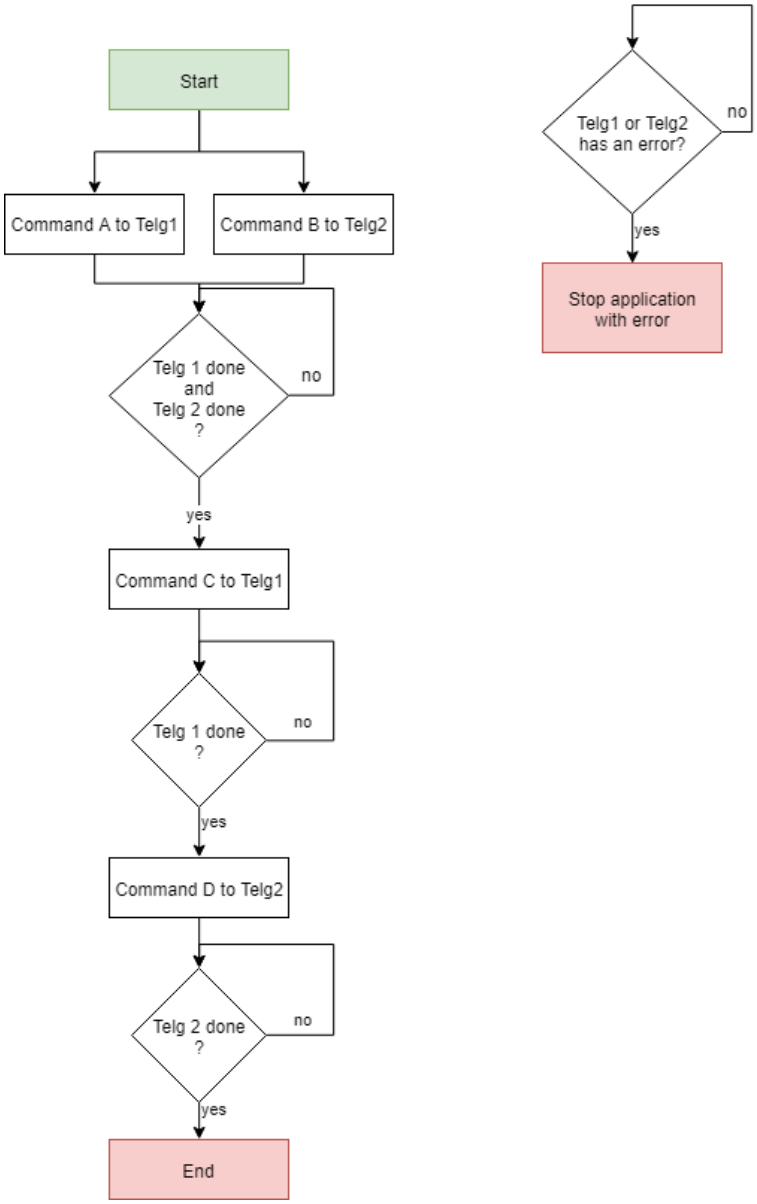
```
ld M1002
    rst Telg1Do
    set Telg1Done
    rst Telg1Fail
```

Joonis 13 Signaliseerimismuutujate lähtestamine käivitamisel

Tekkinud programmikood on toodud töö osas LISA 4.

4.3.2 Vahelduvjuhtimise algoritmi mudel

Kui juhitavatel telgedel on lisatud signaliseerimismuutujad ja kood, saab neid muutujaid kasutada muudes programmides soovitud telje liikumiste alustamiseks. Joonisel 14 on toodud mudel draivide vahelduvjuhtimise algoritmist. Joonisel on näidisena käsud A ja B paralleelselt täidetavad ehk puudub mehaaniline tõkend. Käsud C ja D on näites sellised, mida eri teljed samaaegselt teostada ei saa (puudub loogilisus või esineb mehaaniline takistus), sestap tuleb need töödelda järjekorras. Paralleelselt käsu järgnevuse täitmisele tuleks igal tsüklil kontrollida ka võimalikku viga, millisel juhul katkestatakse peajärgnevus.



Joonis 14 Telgede vahelduvjuhtimise näidisalgoritmi mudel

4.4 Mitme draivi vahelduvjuhtimise algoritmi realisatsioon

Sarnaselt telgede programmikoodile tekitatakse ka vahelduvjuhtimise programmis tegevusele viitav olekumuutuja, käesoleval juhul nimega *state*. Muutuja vajadus tuleneb koodi tsüklilisest käivitamisest ning ühe draivi juhtimisalgoritmi realisatsiooni peatükis käsitleti juhtumit, mil koodijärgnevus sellist muutujat kasutamata katkes. Täiendavalt on *state* muutuja vahend, mille kaudu võib väline rakendus (operaatoripaneel või mõni muu DVP-SV2 COM1/COM2 porti ühendatud seade) käivitada käske. Rakendustes on levinud sellisel juhul operaatoripaneeli (HMI) kasutamine [10] [17].

Näitena käsitletakse järgnevust, mil Telg1 ja Telg2 programmikoodides on realiseeritud käsud 200 ja 300, kusjuures käsku 200 võivad Telg1 ja Telg2 teostada samaaegselt, ent käskude 300 käivitamine on võimalik vaid järjekorras, üks telg korraga. Kirjeldatud järgnevuse realisatsioonis tuleb signaliseerimismuutujaid kasutades anda käsud mõlemale teljele ning kontrollida käsu teostavust. Enne toimiva lahenduse kirjutamist analüüsitakse viga joonisel 15 toodud programmikoodis.

```
ld= state 1
    mov 200 Telg1Command
    set Telg1Do
    mov 200 Telg2Command
    set Telg2Do
    mov 2 state

ld= state 2
and Telg1Done
and Telg2Done
    (* next step *)
```

Joonis 15 Vahelduvjuhtimise kood veaga

Toodud kood ei töötaks korrektselt, sest Telg1 ja Telg2 programme käivitatakse sõltuvalt programmide järjekorrast ISPSoftis. Seega *state* 2 mineku hetkel ei ole Telg1 ja Telg2 programmid veel muutujate *Telg1Do* ja *Telg2Do* olekut registreerinud ja käsku käivitanud, seetõttu on muutujad *Telg1Done* ja *Telg2Done* veel väärtustega 1. Kirjeldatud olukorra lahenduse mustrina võib läbivalt kasutada siiret protsessitava programmi lõppu. Selleks deklareeritakse pärast käskude täitmise vahemikku siirde märgis `end:` ja lisatakse *state* 1 lõppu siirde instruktsioon `cj` (koodinäide joonisel 16).

```
ld= state 1
    mov 200 Telg1Command
    set Telg1Do
    mov 200 Telg2Command
    set Telg2Do
    mov 2 state
    cj end
```

Joonis 16 Faas 1 lõppu lisatud siire

Kui teljed 1 ja 2 on töö lõpetanud, väärtustuvad Telg1Done ja Telg2Done taas tõeseks ning järgmisena antakse näidiskäsk 300, mille puhul ei tohi paralleeltäitmist rakendada, telg 1-le. Selle õnnestumisel antakse käsk 300 telg 2-le. Käske luuakse vastavalt realiseeritava rakenduse nõuetele ja käsu number määratakse sageli kirjutaja eelistusest lähtudes. Programmikoodi lõppu on lisatud eelnevalt kirjeldatud eesmärgiga viide `end:`. Vahelduvjuhtimise kood on toodud joonisel 17.

```
ld= state 2
and Telg1Done
and Telg2Done
    mov 300 Telg1Command
    set Telg1Do
    mov 3 state
    cj end

ld= state 3
and Telg1Done
    mov 300 Telg2Command
    set Telg2Do
    mov 4 state
    cj end

ld= state 4
and Telg1Done
    mov 0 state
    cj end

end:
```

Joonis 17 Vahelduvjuhtimise kood

Koodile lisatakse veel kontroll eesmärgiga monitoorida telgede käskude täitmise kestel muutujaid Telg1Fail ja Telg2Fail, mille väärtus muutub tõeseks, kui mõnel teljel on tekkinud viga ja masina töö tuleks seisata. Näidiskood on toodud joonisel 18.

```
ld Telg1Fail
or Telg2Fail
    mov -1 state
```

Joonis 18 Telje veale reageerimine

Teostatud programmikood on täielikul kujul toodud töö osas LISA 5.

Süsteemi juhtiv rakendus või seade, näiteks operaatoripaneel HMI või arvuti [24], võib realiseerida nupud, mille vajutuse korral väärtustatakse *state* muutuja. Näiteks käsu alustamiseks väärtustatakse muutuja 1-ks. Täiendavalt tuleks väärtuse -1 korral ekraanile kuvada veateadet ning muu mittenullise oleku korral keelata nuppude vajutamisega masina töö juhtimine, sest töö on juba pooleli. Kirjutaja soovi korral võib lisaks muutujale *state* võtta kasutusele ka täiendavad muutujad, et juhtrakenduse kirjutamine oleks lihtsam. Käesoleva töö käigus on vajamineva koodi hulka sihilikult minimeeritud, et kirjeldada toimiv lahendus võimalikult lihtsalt.

Realiseeritud järgnevus, kus antakse paralleeltöötluseks käsud 200 ja seejärel jadamisi käsud 300 mõlemale teljele, on üksnes näidis signaliseerimisvõimekuse rakendamisest vahelduvjuhtimise saavutamiseks. Autori kogemusel tagab toodud lähenemine loetava koodi ka enam kui kahe servokontrolleri korral. Servokontrollerite lisandumisel võib koodi loetavuse seisukohast olla kasulik ka täiendav abstraheerimine kahekihilisest (juhtprogramm – teljeprogramm) kolmekihiliseks (juhtprogramm – telgesid koondav programm – teljeprogramm) süsteemiks.

5 Kokkuvõte

Käesoleva töö raames kirjeldati vajalikud protseduurid Delta Electronics seadmeid kasutava servosüsteemi tarkvaraliseks sätestamiseks: tutvustati Delta seadmete spetsiifilisi programme ISPSoft ja CANopen Builder, näitlikustati CAN-võrgu seadistamine ja suhtluse andmeobjektide konfigureerimine. Seejärel käsitleti vajalikke tegevusi staatuse- ja juhtsõna andmeobjektide kasutamisel servokontrolleri juhtimiseks. Esitati väljatöötatud juhtprogrammi kood, pöörates sealjuures tähelepanu olulistele tarkvaralistele elementidele ja levinud vigadele, mis tulenevad PLC koodi tsüklilisest mitte-blokeerivast käivitamisest. Servokontrolleri juhtprogrammile lisati signaliseerimismuutujad, mille alusel loodi näidiskäskudest koosnev mitme telje vahelduvjuhtimise programm.

Loodud vahelduvjuhtimise algoritm ja selles kasutatav lähenemine on kasutatavad erinevates tööstuslikes rakendustes. Loodud abstraktsioonikiht võimaldab seadmete töö planeerijal ilma spetsiifilise telje programmi modifitseerimata selle telje käske erinevates ahelates käivitada. Toodud kirjeldused ja loodud vahelduvjuhtimise muster täidavad töö eesmärgid.

Edasiste uurimissammudena võiks töös kasutatavad juhtimisalgoritmid asendada objekt-orienteeritud programmeerimisele sarnaneva lahendusega, kus luuakse funktsiooniplokid (*function block*), mida saab kasutada kõigi seda tüüpi telgede juhtimiseks. Funktsioone saab käivitada telje programmidest. Telje programmide kood oleks selle tulemusena väiksem ning telje programmi käsu lisamine oleks lihtsam. Teine võimalik suund oleks valida tootja, kelle seadmed ja programmeerimiskeskkonnad toetavad paremini IEC 61131-3 sätestatud objekt-orienteeritud lähenemisi. Mahukama uurimisvõimalusena võiks käsitleda taoliste PLC seadmete võrgu haldamist ja monitoorimist, kus PLC või sellega kontaktis olev seade on ühendatud Internetti.

Kasutatud kirjandus

[1] J.-Y. T. K.-C. Chena ja G.-C. Chen, „Application of Programmable Logic Controller to Build-up an Intelligent Industry 4.0 Platform,“ *Procedia CIRP,* kd. 63, pp. 150-155, 2017.

[2] M. Sousa, J. Baum ja A. Romanenko, „MatPLC: Towards Real-Time Performance,“ Proceedings of the 5th Real-Time Linux Workshop, 2003.

[3] M. Sousa, A. Carvalho ja R. Ferreira, „Embedding the MatPLC,“ 2005.

[4] T. R. Alves, M. Buratto, F. Mauricio de Souza ja T. V. Rodrigues, „OpenPLC: An Open Source Alternative to Automation,“ *IEEE Global Humanitarian Technology Conference (GHTC),* 2014.

[5] K.-H. John ja M. Tiegelkamp, IEC 61131-3: Programming Industrial Automation Systems, Berlin: Springer, 2010.

[6] W. Bolton, Programmable logic controllers, Oxford: Newnes, 2009.

[7] M. Hanak, „Advanced PLC programming methods,“ Czech Technical University in Prague, Prague, 2017.

[8] E. R. Alphonsus ja M. O. Abdullah, „A review on the applications of programmable logic controllers (PLCs),“ *Renewable and Sustainable Energy Reviews,* kd. 60, pp. 1185-1205, 2016.

[9] P.-Y. Cheng, P.-J. Chen ja Y.-T. Lin, „Small mechanical press with double-axis servo system for forming of small metal products,“ *The International Journal of Advanced Manufacturing Technology,* kd. 68, pp. 2371-2381, 2013.

[10] R. Radha ja V. M. Basavaraj, „Automation of Sectional Drive Paper Machine Using PLC and HMI,“ *International Journal of Engineering Research and General Science,* kd. 3, nr 4, pp. 842-847, 2015.

[11] W. C. Dunn, Fundamentals of Industrial Instrumentation and Process Control, US: McGraw-Hill Professional, 2005.

[12] Delta Electronics, „DVPCOPM-SL CANopen Communication Module Operation manual,“ Delta Electronics, Inc., Taoyuan.

[13] Delta Electronics, „Delta AC Servo System ASDA-M Series,“ Delta Electronics, Inc., Taoyuan, 2013.

[14] Delta Electronics, „Delta Robot Controller with Servo Drive Integrated ASDA-MS,“ Delta Electronics, Inc., Taoyuan, 2016.

[15] M. Paprocki, A. Wawrzak, K. Erwinski, K. Karowski ja M. Klosowiak, „PC-based CNC machine control system with LinuxCNC software,“ *Measurement Automation Monitoring,* kd. 63, nr 1, pp. 15-19, 2017.

[16] A. Carvalho, „Vertical positioning surveillance by magnetostrictive transducer,“ *Journal of Physics: Conference Series,* kd. 648, 2015.

[17] V. S. Deshpande, A. S. C. A. K. Vibhute ja P. Smitha, „Home automation using PLC and SCADA,“ *Multidisciplinary Journal of Research in Engineering and Technology*, kd. 1, nr 1, pp. 111-118, 2014.

[18] M. Di Natale, „The CAN 2.0b Standard,“ *Understanding and using the Controller Area Network,* New York, Springer-Verlag, 2012, pp. 9-26.

[19] H. Boterenbrood, „CANopen high-level protocol for CAN-bus,“ NIKHEF, Amsterdam, 2005.

[20] Delta Electronics, „ISPSoft User Manual,“ Delta Electronics, Inc., Taoyuan, 2013.

[21] Delta Electronics, „DVP-PLC Application Manual,“ Delta Electronics, Inc., Taoyuan, 2013.

[22] Delta Electronics, *CANopen Technical Guide | ASDA-A2,* Taoyuan: Delta Electronics, Inc., 2012.

[23] R. Huuck, „Semantics and Analysis of Insruction List Programs,“ *Electronic Notes in Theoretical Computer Science,* kd. 115, pp. 3-18, 2005.

[24] S. S. Bidwai, V. B. Kumbhar ja A. R. Nichal, „Real Time Automated Control using PLC-VB Communication,“ *International Journal of Engineering Research and Applications,* kd. 3, nr 3, pp. 658-661, 2013.

Lisa 1 – Draivi staatusesõna bittide kirjeldus

Järgnevalt on toodud ASDA-A2 draivi staatusesõna (CANopen OD-6041) bittide tähenduste kirjeldus. Bitid 12 ja 13 on kasutatava juhtrežiimi spetsiifilised ning käesoleva töö skoobist lähtuvalt kirjeldatakse *position profile* (positsiooni) ja *homing mode* (homing) režiimide bitid.

Bitt	**Kirjeldus**	
0	*Ready to switch on*	
1	*Switch on*	
2	*Operation enabled (servo on)*	
3	*Fault (drive will servo off)*	
4	*Voltage enabled*	
5	*Quick stop*	
6	*Switch on disabled*	
7	*Warning (drive still servo on)*	
8	*N/A*	
9	*Remote* (CANopen juhtimine aktiveeritud)	
10	*Target reached* (positsioonirežiimis sihtasend saavutatud)	
11	*N/A*	
	Positisiooni režiim	**Homing režiim**
12	*Set-point acknowledge* (positsioneerimiskäsu kinnitus)	*Homing attained* (homing tehtud, nullasend leitud)
13	*Following error*	*Homing error*

Bitid 14-15 ei ole kasutuses.

Lisa 2 – Draivi juhtsõna bittide kirjeldus

Järgnevalt on toodud ASDA-A2 draivi juhtsõna (CANopen OD-6040) bittide tähenduste kirjeldus. Bitid 4, 5 ja 6 omavad tähendust vastavalt kasutatavale juhtrežiimile. Töö skoobist lähtuvalt kirjeldatakse positsiooni ja homing režiimide bitid.

Bitt	**Kirjeldus**	
0	*Switch on*	
1	*Enable voltage*	
2	*Quick stop (B-contact)*	
3	*Enable operation*	
	Positisiooni režiim	**Homing režiim**
4	New set-point (positive trigger)	Homing operation start
5	*Change set immediately*	*N/A*
6	*Absolute (0) / relative (1)*	*N/A*
7	*Fault reset*	
8	*Halt*	

Bitid 9-15 ei ole kasutuses.

Lisa 3 – Kasutatud ja levinuimad instruktsioonid

Järgnevalt on toodud käesolevas töös ja levinuimalt kasutatud instruktsioonid koos nende eestikeelsete tähendustega. Terve nimekiri instruktsioonides on dokumendis DVP-SV Application Manual peatükkides 3.1 ja 5.5 [21].

Instruktsioon	**Andmetüübid**	**Kirjeldus**
ld	X, Y, M, T, C	A-kontakti kontroll (uus väärtus instruktsioonipinus)
ldi	X, Y, M, T, C	B-kontakti (inversioon) kontroll (uus väärtus instruktsioonipinus)
and	X, Y, M, T, C	Jadaühendus A-kontakt (modifitseerib instruktsioonipinu viimast väärtust)
ani	X, Y, M, T, C	Jadaühendus B-kontakt (modifitseerib instruktsioonipinu viimast väärtust)
or	X, Y, M, T, C	Paralleelühendus A-kontakt (modifitseerib instruktsioonipinu viimast väärtust)
ori	X, Y, M, T, C	Paralleelühendus B-kontakt (modifitseerib instruktsioonipinu viimast väärtust)
anb		Instruktsioonipinus kahe viimase tulemuse AND
orb		Instruktsioonipinus kahe viimase tulemuse OR
mps		Viimase tulemuse salvestamine
mrd		Viimati salvestatud tulemuse lugemine
mpp		Viimase tulemuse eemaldamine
ldp	X, Y, M, T, C	Rising edge (tõusva serva) kontroll
ldf	X, Y, M, T, C	Falling edge (langeva serva) kontroll
andp	X, Y, M, T, C	Tõusva serva kontroll jadaühenduses
andf	X, Y, M, T, C	Langeva serva kontroll jadaühenduses
orp	X, Y, M, T, C	Tõusva serva kontroll paralleelühenduses
orf	X, Y, M, T, C	Langeva serva kontroll paralleelühenduses
set	Y, M, S	Väärtuse loogiliseks 1-ks muutmine
rst	Y, M, S	Väärtuse loogiliseks 0-ks muutmine
(d)mov S D	T, C, D	(doublewordi) kopeerimine (asukohast S asukohta D)

Instruktsioone mps, mrd, mpp annab kasutada tingimuste paremaks järjestikku kirjutamiseks. Näiteks:

```
ld X0
    (* selle taseme kood käivitatakse juhul, kui X0 on kõrge *)
    mps
    and X1
        (* käitumine juhul, kui X0 ja X1 on kõrged *)
    mrd
    and X2
        (* käitumine juhul, kui X0 ja X2 on kõrged *)
    mrd
    ld Tingimus1
    and Tingimus2
    orb
        (* käitumine juhul, kui (X0 on kõrge) VÕI (Tingimus1 ja Tingimus2 on
kõrged) *)
```

Instruktsioonidele ld, and, or saab lisada loogikaoperaatori, nt `ld>`.

```
ld M1000
    (* always execute on every cycle *)
    dinc D1000
ld> D1000 35000 (* if D1000 reaches 35000 *)
    dmov 0 D1000 (* reset D1000 to 0 *)
```

Järgnevalt on toodud andmetüüpide tähistuste tähendused.

Andmetüübi tähistus	**Andmetüübi kirjeldus**
X	Sisend
Y	Väljund
M	Relee (*internal relay*)
D	Register (*data register*)
T	Taimer (*timer*)
C	Loendur (*counter*)

Lisa 4 – Telg1 programmikood

(* initialization block – prepare program for accepting commands *)

```
ld M1002
    rst Telg1Do
    set Telg1Done
    rst Telg1Fail

(* check for incoming commands *)
ld Telg1Do
    rst Telg1Do
    and= state 0
        rst Telg1Fail
        rst Telg1Done
        mov Telg1Command state

(* command 50: reset *)
ld= state 50
    mov 0 tx_opmode
    mov 143 tx_controlword
    and= rx_feedback_opmode 0
    and= rx_feedback_controlword 143
        mov 51 state

ld= state 51
bani rx_statusword 3 (* fault *)
bani rx_statusword 13 (* following error *)
band rx_statusword 2 (* operation enabled *)
    mov 0 state

(* command 100: homing *)
ld= state 100
    mov 1542 tx_opmode
    mov 15 tx_controlword (* servo on *)
    and= rx_feedback_opmode 1542
```

```
    and= rx_feedback_controlword 15
        mov 101 state

ld= state 101
    mov 31 controlword (* start homing + servo on *)
    and= rx_feedback_controlword 31
        mov 102 state

ld= state 102
band rx_statusword 10 (* target reached *)
band rx_statusword 12 (* homing attained *)
    mov 103 state

ld= state 103
    (* put drive into positioning mode for quicker controlling in next issued
command *)
    mov 257 tx_opmode
    mov 15 tx_controlword
    and= rx_feedback_opmode 257
    and= rx_feedback_controlword 15
        mov 0 state

(* command 200: moving *)
ld= state 200
    dmov 2500000 tx_targetposition (* target encoder position *)
    dmov 5400000 tx_velocity (* 3000 RPM *)
    dmov 1000 tx_acceleration (* 1 second to achieve full speed *)
    dmov 1000 tx_deceleration (* 1 second to decelerate to a stop *)
    mov 257 tx_opmode (* write operating mode = positioning *)
    mov 63 tx_controlword (* write control word = new set point, servo on *)
    and= rx_feedback_opmode 257
    and= rx_feedback_controlword 63
        (* controls received by drive *)
        mov 201 state

ld= state 201
```

```
    (* reset controlword to 15 so movement can be started quickly next time *)
    mov 15 tx_controlword
    and= rx_controlword_feedback 15
    band rx_statusword 10 (* check for target reached *)
        mov 0 state

(* check for fault encountered during homing or normal operation *)
ld>= state 100
band rx_statusword 3
    mov -1 state

(* set Telg1 status bits that are read by the controlling module *)
ld= state 0
    set Telg1Done (* command done successfully *)
ld= state -1
    set Telg1Fail (* command done with error *)
    mov 0 state (* reset state so reset command can be given *)
```

Lisa 5 – Vahelduvjuhtimise programmi kood

```
ld Telg1Fail
or Telg2Fail
    mov -1 state

ld= state 1
    mov 200 Telg1Command
    set Telg1Do
    mov 200 Telg2Command
    set Telg2Do
    mov 2 state
    cj end

ld= state 2
and Telg1Done
and Telg2Done
    mov 300 Telg1Command
    set Telg1Do
    mov 3 state
    cj end

ld= state 3
and Telg1Done
    mov 300 Telg2Command
    set Telg2Do
    mov 4 state
    cj end

ld= state 4
and Telg1Done
    mov 0 state
    cj end

(* other commands to be implemented here *)
```

```
end:
```