TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Kaspar Haavajõe

KAUGJUHITAVATE PROGRAMMEERITAVATE KONTROLLERITE VÕRGUSEADISTUSTE TAASTAMINE

Bakalaureusetöö

Juhendaja: Rein Paluoja
PhD

Kaasjuhendaja: Rene Pihlak
MA

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: Kaspar Haavajõe

15.05.2018

Annotatsioon

Lõputöö eesmärk on välja töötada tarkvaraline lahendus programmeeritava kontrolleri võrguseadistuste taastamiseks juhul, kui neid pole või need on valed. Olulisemateks probleemideks on arvuti ja kontrolleri vahelise andmevahetuse tõlgendamine ning andmevahetuse taasesitamine selleks, et läbi viia vastavaid seadistusi, operatsioone. Töö tulemusena valmib tarkvaraline lahendus, mille abil on võimalik eeldefineeritud seadistustega seadistada labori kontrollereid ning lisaks läbi viia hulk kontrolleritega seotud elementaaroperatsioone. Lisaks tarkvarale töötatakse välja üldine meetod arvuti ja konkreetse kontrolleri vahelise andmevahetuse analüüsimiseks ja taasesitamiseks.

Lõputöö on kirjutatud Eesti keeles ning sisaldab teksti 38 leheküljel, 7 peatükki, 18 joonist, 4 tabelit.

Abstract

Recovery of remotely controlled PLC's network configuration

The purpose of this thesis is to develop software that would handle the recovery of programmable controllers' network configuration as the configuration can be false or there is none of it. Most important problems include interpretation of computer-to-controller commmunication as well as the reproduction of the communication in order to carry out corresponding configurations, operations. As a result of this thesis, a software will be developed allowing the configuration of controllers in laboratory using predefined configuration files. In addition, several base operations associated with controllers will be implemented. Besides software, a universal procedure will be developed considering the computer-to-controller data communication analysis and reproduction.

The thesis is in Estonian and contains 38 pages of text, 7 chapters, 18 figures, 4 tables.

Lühendite ja mõistete sõnastik

PLC	*Programmable Logic Controller*, programmeeritava loogikaga kontroller
USB	*Universal Serial Bus*, universaalne järjestiksiin
EP	*Endpoint*, USB seadme funktsionaalne allsüsteem
CRC	*Cycle Redundancy Check*, tsükkelkoodkontroll
HID	*Human Interface Device*, inimesega vahetus suhtluses olev USB seade
UART	*Universal Asynchronous Receiver/Transmitter*, universaalne asünkroontransiiver
LCS	*Longest Common Subsequence*, pikim ühine alamjada
COM	*Communications port*, järjestikport e. järjestikliides
DOM	*Document Object Model*, dokumendiobjektide mudel

Sisukord

Jooniste loetelu

Tabelite loetelu

1. Sissejuhatus

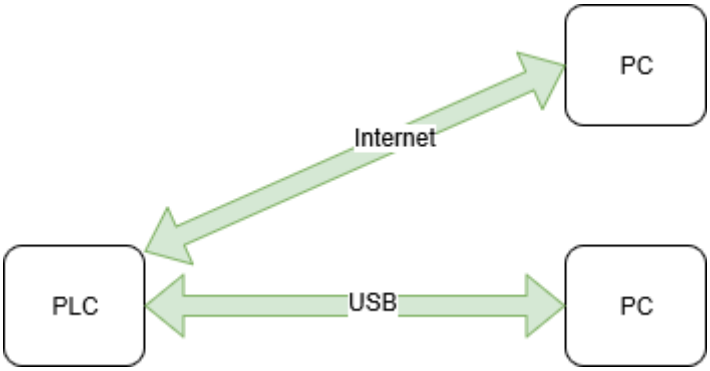
Aines „Programmeeritavad kontrollerid I/II“ kasutatakse tööstuslikku kontrollerit (PLC) Unitronics. Ainet võimaldab läbi viia labor, mis sisaldab kümneid kontrollereid, mida programmeerida saab. Praegusel hetkel on võimalik programmeerida seadmeid kahel viisil. Esimeseks võimaluseks on kasutada vastavat USB juhet. Alternatiiviks on kasutada võrguühendust ning programmeerida seade üle võrgu (vt joonis 1). Üks viisidest eeldab, et üliõpilane on füüsiliselt seadme juures, teise puhul on tegu aga nn kauglaboriga.

Igal kontrolleril on seadistatav staatiline IP-aadress (see tekib koos programmiga), mis teda võrgus identifitseerib ja mille abil on võimalik temaga ühendust võtta. Lisaks sellele identifitseerib kontrollereid ka unikaalne nimi, mis tuleb samuti sama programmiga kaasa. Üheks suurimaks probleemiks on aga see, et kasutajal on võimalik programmeerida oma kontroller nii, et ta juhtumisi jätab IP-aadressi või kontrolleri nime seadistamata ning kontroller „kaob“ võrgust. Peale sellist vigast programmeerimist ei ole kasutajal võimalik enam seadmega suhelda. Lisaks sellele pole ka teistel huvilistel võimalik seda seadet üle võrgu programmeerida.

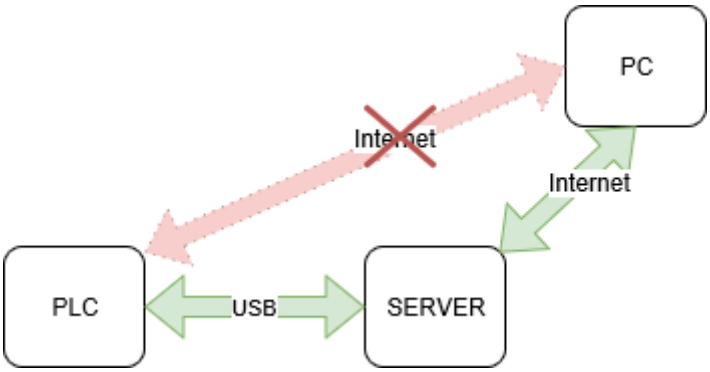
Oodatavaks lahenduseks oleks süsteem, mis võimaldaks algseadistada kontrolleri võrguseadistusi juhul, kui selline olukord on tekkinud. Algseadistused erinevad kontrollerite puhul, kuna igal kontrolleril on unikaalne nimi ja IP-aadress. Võrguseadistuste taastamine toimuks läbi keskse serveri. See tähendaks, et kui juhtubki midagi laboris oleva seadme võrguseadistustega, siis eksisteerib keskne seade, mille andmevahetus labori seadmetega on garanteeritud (vt joonis 2). Server iseenesest ei ole aga käesoleva töö sisuks. Antud töö üritab arendada välja protseduuri, kuidas teha võimalikuks võrguseadete taastamine keskse serveri ja suvalise laboris oleva PLC seadme vahel. See eeldab, et luuakse tarkvara, mis suudab PLC-dele käsklusi jagada. Antud töö uurib erinevaid viise, kuidas teostada serveri ja PLC seadme vahelist suhtlust ning eesmärgiks on luua üldine lahendus, mis võimaldaks suvalisele seadmele laboris anda soovitud seadistused võimalikult mugavalt. Praegune olukord tähendab sisuliselt seda, et

laboreid haldav isik peab füüsiliselt minema kohale ja kontrolleritele vastavaid seadistusi tegema.

Esmalt uuritakse PLC seadmete programmeerimiseks kasutatavaid tarkvaralisi vahendeid ning nende funktsionaalsust. Seejärel annab autor ülevaate erinevatest võimalustest võrguseadistuste taastamisega seotud probleemi lahendamiseks. Järgnevalt analüüsitakse detailselt erinevaid andmevahetuse taasesitamise võimalusi, rakendatakse neid ning lõpuks testitakse.



Joonis 1. Kaks viisi PLC-ga suhtlemiseks

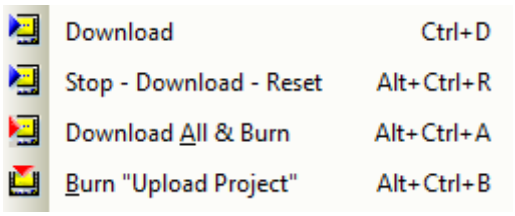


Joonis 2. Plaanitav suhtlus, kui PLC "kaob" võrgust

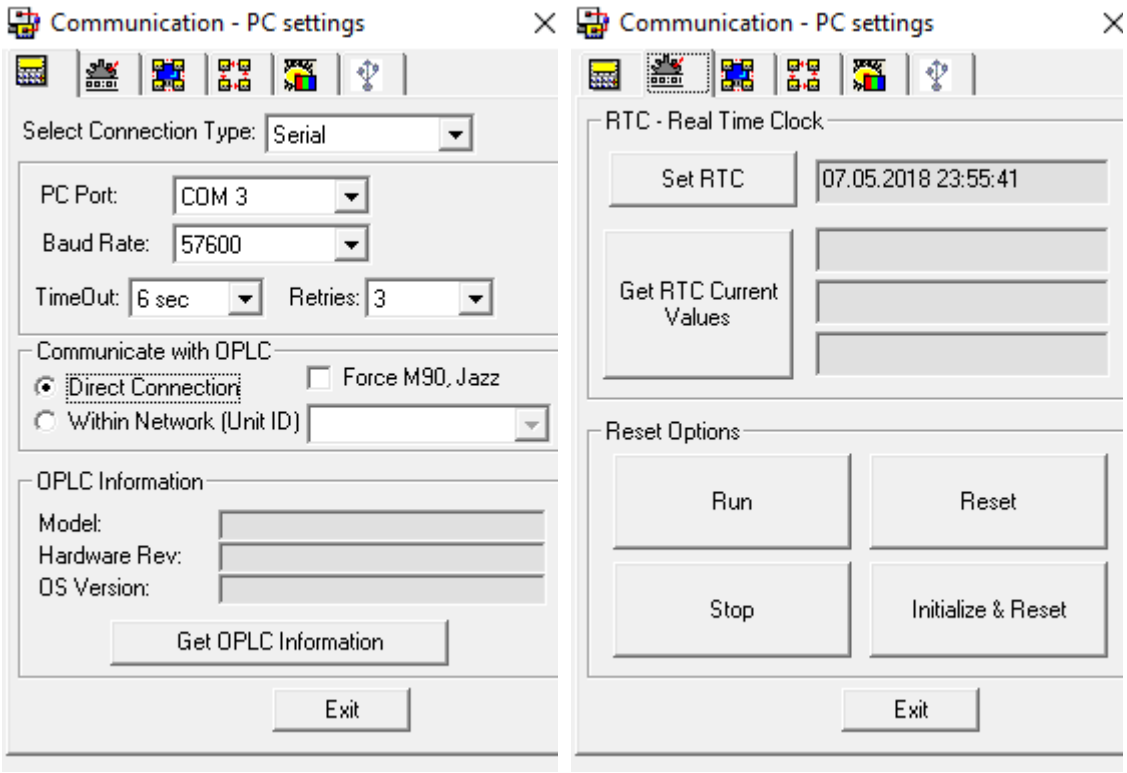
2. Unitronics Visilogic ja UniDownLoader

Unitronics Visilogic on tarkvarakeskkond, kus on võimalik koostada programme tööstusliku kontrolleri (PLC) programmeerimiseks. Programmeerimine toimub suuresti redeldiagrammide abil *drag'n drop* meetodit kasutades. Lisaks tarkvara laialdasele funktsionaalsusele, mida autor välja tooma ei hakka, saab Unitronics Visilogic tarkvara abil programmi ka PLC mällu laadida. Selleks tuleb valida tarkvaras käsklus „Stop - Download - Reset" (vt joonis 3), mis kõigepealt peatab PLC senise programmi, siis laeb PLC programmi mällu uue koodi ning lõpuks taaskäivitab PLC koos uue programmiga. Lisaks sellele, et pidevalt peaks käivitama tarkvara Visilogic ning selle abil programmeerima kontrollerit, saab koodist/projektis teha programmeeritava koopia ning seda kasutada eraldiseisva väiksema tarkvaraga „UniDownLoader" (vt joonis 4), kus lisaks koodi laadimisele mällu saab teostada veel mitut olulist operatsiooni. Enimkasutatavamad funktsionaalsused mõlemas tarkvaras kokku on järgmised:

- Get OPLC information - operatsioon, mille abil saab PLC-d identifitseerida
- RUN - operatsioon, mis käivitab programmi, kui viimane on peatunud
- RESET – operatsioon, mis taaskäivitab kontrolleri
- STOP – operatsioon, mis peatab kontrolleri töö
- PLC Status - tagastab PLC erinevate osade olekud
- STOP-DOWNLOAD-RESET - operatsioon, mida kasutatakse programmi laadimiseks PLC mällu



Joonis 3. Unitronics Visilogic operatsioonide valik



Joonis 4. UniDownloader operatsioonide valik

3. Võimalikud alternatiivid

On selge, et kui pikemas perspektiivis on eesmärgiks kontrollerite programmeerimine üle võrgu, siis ei saa lootma jääda inimfaktorile (kohapeal seadistamine). Selle asemel tuleks otsida tehnoloogilisi alternatiive protsessi kaugjuhtimiseks.

3.1 Remote desktop

Lihtsustatud lahendus kontrollerite algseadistamiseks oleks kaugühenduse loomine suvalise arvuti ning kontrollerit juhtiva arvuti vahel. Kui peaks tekkima olukord, kus kontroller ei ole enam võrgust kättesaadav, saab keegi vajadusel kasutada eemalolevast arvutist kohaliku arvuti töölauda ning seal algseadistada kontrolleri kasutades eespool nimetatud kasutajaliideseid. On selge, et see lahendus ei automatiseeri protsessi ning lõpuks peab ikka inimene jälgima seda protsessi pidevalt. Tuleks arvestada ka seda, et tulevikus on plaan kõik kontrollerid ühendada keskse serveriga, mis neid haldab ning

seega on keeruline ette kujutada, kuidas suur hulk kasutajaid kasutab samaaegselt ühe arvuti töölauda.

3.2 Winium

Üheks võimaluseks automatiseerida programmi laadimist PLC-sse, on teha seda kasutades olemasolevat Unitronics UniDownLoader kasutajaliidest ning sisuliselt automatiseerida selle füüsiline kasutamine. Veebiarenduses on tihti kasutusel vahendid tarkvara testimiseks kõige kõrgemal tasemel ehk kasutaja tasemel. Sobivas programmeerimiskeeles kirjutatakse valmis automaattestid, mis käivitamisel hakkavad kasutama kasutajaliidesega tarkvara nii, nagu seda teeks kasutaja. Testitakse erinevate valikute olemasolu ja funktsionaalsust. Näiteks saab testida seda, et nuppudele vajutamisel toimub ikka see, mis on ette nähtud. Veebiarenduse puhul on kasutusel tarkvara nimega Selenium. Kuna testitakse nähtava kasutajaliidese erinevaid osi, siis on selge, et kõik need osad peavad olema kuidagi identifitseeritavad. Seleniumi puhul kasutatakse selleks DOM elemendi identifikaatorit, klassikuuluvust, nime, suhtelist teekonda juurelemendist (*xpath*) vms.

Analoogiline võimalus on ka Windows operatsioonisüsteemide töölaua rakenduste puhul. Vabavaralise tarkvarana võib kasutada näiteks Winium (Windows + Selenium) nimelist tarkvara. Windows kasutajaliideste elemente inspekteerida on aga tunduvalt keerulisem kui brauseris, kuid see on põhimõtteliselt võimalik. Nimelt on selle jaoks olemas Windows SDK-s spetsiaalne vahend „inspect.exe“, mis kuvab välja töölauarakenduste erinevad objektid ning nende objektidega seotud identifikaatorid [1] [2].

Programmi laadimise automatiseerimisel kasutajaliidese tasemel on mitmeid puudujääke. Esiteks, lahendus töötaks vaid Windows operatsioonisüsteemis. On tõenäoline, et server, mis laboris kontrollerite seadistamist haldama hakkab, kasutab Linux operatsioonisüsteemi. Lisaks on saadud lahendus raskesti hallatav kuna programmi täitmise ajal ei saa toimuda serveris mitte midagi muud. Vastasel korral ei pruugi kasutajaliidese kasutamine soovitud tulemust anda. Veelgi enam, tulevikus planeeritud kasutajaliidest puudutavad tarkvaralised uuendused muudaksid lahenduse kasutamise võimatuks.

Kokkuvõtteks, saadud lahendus ei ole piisavalt universaalne, et seda saaks käsitleda, kasutada eraldiseisva tükina, tarkvaralise moodulina.

3.3 Andmevahetuse uurimine

Selleks, et realiseerida universaalne lahendus, tuleb süveneda olemasoleva süsteemi tehnilisse tausta. On selge, et vajutades kasutajaliideses näiteks „Reset“ nuppu taaskäivitatakse PLC. Kuna PLC on arvuti külge ühendatud USB kaabli abil, võib oletada, et USB protokolli vahendusel edastatakse kindlaksmääratud sõnum PLC-le. Seega, kui oleks võimalik jälgida detailselt arvuti ja PLC vahelist USB kommunikatsiooni, peaks teoreetiliselt seal kajastuma iseärasus seoses PLC taaskäivitamisega. Samuti peaksid seosed ilmnema ka teiste operatsioonide puhul. Nii oleks võimalik pöördprojekteerida kõikvõimalikud operatsioonid, mis seni on teostatavad vaid läbi kasutajaliidese.

4. Lahendused, probleemid lahendamisel

Käesolevas peatükis tegeletakse sisulise lahenduse otsimisega võrguseadistuste taastamisega seotud probleemile. Uuritakse, millised probleemid tekivad, kui on soov lahendada võrguseadistuste taastamise probleemi andmevahetuse taasesitamise teel. Analüüsides andmevahetust töötatakse välja meetod selle taasesitamiseks ning seega ka võrguseadistuste taastamiseks.

4.1 USB andmevahetus

Esmaseks eesmärgiks USB andmevahetuse jälgimisel on leida sobiv tarkvara USB kommunikatsiooni pealtkuulamiseks. Esialgse loogika järgi peaks tarkvara võimaldama andmete filtreerimist parameetrite järgi ning tõenäoliselt peaks ka olema võimalus andmeid salvestada/eksportida, et neid töödelda juba järgnevate vahenditega.

4.1.1 Device Monitoring Studio

Device Monitoring Studio (DMS) on tarkvara, mis võimaldab monitoorida, logida, analüüsida USB seadmete vahelist kommunikatsiooni. Tarkvara töötab kõikides Windows-i versioonides, kuid Linux-is mitte. Pealtkuulatud pakette saab ka salvestada ning hiljem uuesti analüüsida. Reaalajas andmevahetuse kuulamisel toimub mustrite avastamine kuni 1GB andmete ulatuses. Selgusetuks jääb, millises formaadis on võimalik andmeid eksportida/salvestada (csv, xml, json, vms). Esialgu võib oletada, et andmevahetus tuleb mahukas ning seega peaks eksisteerima mõni mugav viis saadud info töötlemiseks mõnes tuntud formaadis. DMS on tasuline tarkvara, kus tasuta saab kasutada vaid piiratud koguses funktsionaalsust. Üldiselt jääb vahend silma selle poolest, et struktureerib selgelt toimuvat andmevahetust. Näiteks sissetulevad paketid värvitakse punaseks ja väljaminevad siniseks [3].

4.1.2 Free USB Analyzer

Tegu on jällegi tasulise tarkvaraga, kuid teatud funktsionaalsus on limiteeritud koguses tasuta. Tarkvara kasutamine on piiratud natuke erinevalt võrreldes eelmise vahendiga. Nimelt saab päevas teha selle tarkvaraga viis sessiooni, kusjuures sessioon ei tohi ületada 10 minutit. Sessiooni all peetakse silmas ajavahemikku, mille jooksul toimub pealtkuulamine. Selgub, et Free USB Analyzer on vaid üks osa suuremast komplektist ning sinna üldisesse komplekti kuuluvad veel lisaks tarkvara, millega spetsiaalseid protokolle pealt kuulatakse. Olgu siinkohal nimetatud näiteks Free Serial Analyzer, Free Network Analyzer, Free Virtual COM Analyzer. Tarkvaral on mida pakkuda: võimalus USB liiklust filtreerida. Arusaamatuks jääb siiski asjaolu, kuidas saadud andmeid salvestada, mis nende formaat oleks jne. Linux operatsioonisüsteemi vahend ei toeta [4].

4.1.3 Wireshark

Kõikidest teistest andmevahetuse inspekteerimisega tegelevatest tarkvaradest on Wireshark vast kõige kuulsam. Wireshark on tunduvalt laiema haardega tarkvara kuna võimaldab jälgida mistahes protokolli väga erinevatelt kihtidel: USB, TCP, UDP, HTTP jne. Suureks plussiks on filtreerimise olemasolu. Tegemist on täiesti vabavaralise, avatud lähtekoodiga tarkvaraga, mille funktsionaalsus ei jää alla eespool nimetatud vahenditele. Lisaks toetab tarkvara ka Linux operatsioonisüsteemi. Selgub, et andmeid saab töötlemiseks salvestada mitmes erinevas formaadis. Olgu nimetatud xml, C päise laiendiga fail (.h), json, csv kui ka tavaline .txt laiendusega fail.

4.1.4 Riistvaraline vahend

Lisaks nimetatud valikutele eksisteerib ka võimalus sättida riistvaraline kontroller või spetsiaalne vahend arvuti ja pealtkuulatava USB seadme vahele. See vahend oleks USB signaali füüsiline vahendaja (USB analüsaator) ning kuvaks vajadusel välja kõik paketid. Üheks selliseks seadmeks on näiteks Beagle USB analüsaator. Esialgu tundub riistvaraline lahendus liiast, sest eeldab kas spetsiaalse riistvara loomist (FPGA) liikluse kuulamiseks või spetsiaalse riistvara ostmist [5]. Alternatiivne lahendus oleks ka otsilloskoop, mille abil signaali kuju kuvada, kuid kuna informatsiooni on palju, siis see lahendus ei ole piisav, et analüüsida andmevahetust.

4.2 USB protokoll

Universal Serial Bus (USB) protokoll on olemuselt suhteliselt keeruline. Siiski on tegu nii mõneski mõttes kihilise protokolliga ning lõpptulemusena piisab vaid sobiva kihi analüüsimisest, jälgimisest. See lihtsustab arusaamist protokollist. Algatuseks oleks oluline USB protokolli põhimõistete tutvustamine, kuna neid on palju ja need on kohati mittetriviaalsed.

Füüsilisel tasemel on USB võrgu (so hulk USB seadmeid) puhul tegemist tähttopoloogilise võrguga. See tähendab, et on üks *host* (juhtseade), mis juhib andmevahetust ning lõplik hulk teisi seadmeid, mis on kõik ühendatud keskse seadme (juhtseadme) külge. Juhtseadmeks on väga tihti arvuti. Kuna juhtseade kontrollib andmevahetust, siis on viimane alati andmevahetuse alustaja. Kuigi andmevahetust juhtseadme ja teiste seadmete vahel nimetatakse kahesuunaliseks, siis sisuliselt ei ole need seadmed „iseteadlikud“ ning vastavad lihtsalt neile saadetud päringutele. Kahesuunaline andmevahetus taandub kas seadmest lugemisele või seadmesse kirjutamisele. Mõlemat teostab juhtseade. Võib tekkida õigustatud küsimus, et kuidas saab näiteks arvutihiir töötada, kui andmevahetust alustab ainult arvuti. Arvuti ei saa põhimõtteliselt teada, millal on õige aeg andmevahetuse alustamiseks. Nii arvutihiir kui ka klaviatuur kuuluvad sellisesse USB seadmete klassi nagu HID. See tähendab, et juhtseade käib teatud intervalliga pidevalt pärimas (lugemas) HID seadme olekut ja kui see muutub, siis on juhtseade sellest teadlik.

Loogilisel tasemel on USB protokolli puhul tegu siinitopoloogilise võrguga. See tuleneb sellest, et loogilisel tasemel saadab juhtarvuti päringu kõikidele võrgus olevatele seadmetele, kuid vastab vaid see, kellele see adresseeritud oli. Seega kõigil seadmetel on põhimõtteliselt kasutada ühine loogiline siin.

Igal USB seadmel on mingi arv *endpointe* (EP). EP on USB seadme funktsionaalne allsüsteem, mille vastutusalasse kuuluvad kindlad ülesanded. EP-de arv võib varieeruda vastavalt seadme funktsionaalsusele, kuid enamasti ei ole USB seadmel üle kahe EP. Üks vastutab sisuliste andmete vahetuse eest, teine vastutab seadmega seotud olekute eest. Lisaks kehtib reegel, et igal USB seadmel peab olema vähemalt üks EP [6].

Iga EP juurde kuulub kaks puhvrit. Viimaseid nimetatakse IN ja OUT puhvriteks. Kui seade tahab juhtseadmele midagi saata, kirjutab ta oma andmed sobiva EP IN puhvrisse.

Juhtseade loeb peale seda need andmed IN puhvrist. Nagu näha, siis juhtseade peab teadma, millal andmeid lugeda. Kui juhtseade tahab saata andmeid USB seadmele, siis kirjutab ta oma andmed USB seadme OUT puhvrisse. Sealt edasi loeb USB seade puhvrist need andmed. Siit järeldub jällegi, et initsiatiivi andmevahetuseks ei alusta mitte kunagi USB seade, vaid juhtseade. Lisaks selgub, et kogu tarkvaraline tarkus, mis seisneb USB andmevahetuse taga, asub juhtseadmes. See on ka peamine põhjus, miks igal seadmel peab olema oma kindlaksmääratud draiver. Just viimane on see, mis omab kogu informatsiooni, mida on vaja andmete mõlemasuunaliseks vahetuseks [6].

Eespool nimetatud lugemis- ja kirjutamisoperatsioone nimetatakse üldistatult transaktsioonideks. Iga transaktsioon koosneb kolmest paketist:

- *token packet* e. loapakett
- *data packet* e. andmepakett
- *handshake packet* e. käepigistuspakett

Iga transaktsioon ei pea sisaldama sisulisi andmeid. Seega võib andmepakett olla ka tühi. Loapakett määrab ära selle, millisele USB seadmele ning millisele EP-le transaktsioon on mõeldud. Lisaks sellele määrab loapakett ära ka selle, mis suunas transaktsioon kulgeb (IN või OUT). Andmepakett kannab endas sisulisi andmeid ning käepigistuspakett kinnitab transaktsiooni. USB standardi kohaselt on defineeritud kolm eri tüüpi transaktsiooni. Need on ühtlasi ka elemendid, mida Wireshark tarkvara välja kuvab ning neid edaspidi ka analüüsima hakatakse:

- *control transfer* e. juhttransaktsioon
- *bulk transfer* e. suuremahuline transaktsioon
- *interrupt transfer* e. katkestustransaktsioon

4.2.1 Juhttransaktsioon

Juhtseade teostab selliseid transaktsioone, kui on vaja informatsiooni USB seadme kohta või tema seisundite kohta. Lisaks, kui eesmärgiks on muuta või teada saada USB seadme parameetreid, siis seda tehakse nende transaktsioonide abil. USB standardi järgi peab igal seadmel vähemalt üks EP olema. Seda tähistatakse kui EP0 ning see on funktsionaalne

allsüsteem, mis vastutab just eespool nimetatud probleemide eest. Kõik juhttransaktsioonid tehakse EP0-le ning kui seda ei oleks, pole võimalik isegi teada saada seda, mis seadmega on tegu. Selleks, et seda teada saada ning lisaks veel hankida informatsiooni, kuidas selle „tundmatu“ seadmega suhelda tuleb (nt piirangud, andmevahetuskiirus jne), on ette nähtud spetsiaalsed juhttransaktsioonid ning iga USB seade peab olema võimeline neile vastama. Need on transaktsioonid, mille abil arvuti näiteks tuvastab, millist draiverit antud seadme puhul kasutama peaks või kui draiverit pole arvutis, siis vähemalt teab juhtseade seda, millist tarkvara internetist allalaadimiseks otsida. Juhttransaktsioonid on ainuke viis, kuidas arvuti teeb põhimõtteliselt vahet HP printeril või Logitech arvutihiirel.

Kui USB seade ühendada juhtseadme külge (arvuti), siis esimese asjana omistatakse seadmele aadress. Peale seda teostab arvuti suurel hulgal juhttransaktsioone, et teha kindlaks seadme parameetrid ning vajadusel neid ka konfigureerida. Lisaks paljudele parameetritele, mida autor siinkohal lihtsuse huvides välja tooma ei hakka, kuuluvad parameetrite hulka ka tootja identifikaator, seadme identifikaator, maksimaalne paketi suurus, seadme klassikuuluvus jne [6].

4.2.2 Suuremahuline transaktsioon

Suuremahulised transaktsioonid on sisuliste andmete vahetamiseks seadmete vahel. Nende transaktsioonide puhul rakendatakse veakontrolli (CRC) ning selliste transaktsioonide peale on tihti ehitatud mõni kõrgetasemeline protokoll. Põhiline erinevus juhttransaktsioonidega võrreldes on see, et suuremahulisi transaktsioone ei tehta mitte EP0-i, vaid mistahes teise võimalikku EPsse. *Mass Storage* seadmed on klassikaline näide USB seadmete klassist, kus suuremahulisi transaktsioone ohtralt kasutatakse [6].

4.2.3 Katkestustransaktsioon

Katkestustransaktsioonid on mitteperioodiliste teadete edastamiseks. Nagu eespool sai mainitud, on katkestustransaktsioonid iseloomulikud HID klassi kuuluvatele USB seadmetele ning selle transaktsiooni põhiliseks tunnuseks on see, et toimub juhtseadme poolne pidev järelpärimine [6].

4.3 USB andmevahetuse jälgimine Wiresharkis

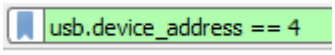
Et teada saada, milline informatsioon liigub arvuti ja PLC vahel erinevate operatsioonide korral, kasutab autor tarkvara nimega Wireshark. Põhilised küsimused, millele vastuseid otsitakse, on järgmised:

- Kas iga kord, kui konstantsete parameetritega teostatakse kindel operatsioon, on operatsioon sama?
- Kui kindlal operatsioonil parameetreid muuta, siis milliseks kujuneb andmevahetus?
- Kui palju transaktsioone toimub erinevate operatsioonide korral?
- Kas operatsioonidel on andmevahetusest tulenev ühisosa?
- Kuidas võrrelda kahte erinevat andmevahetust?

Et nendele küsimustele vastata, tuleb läbi viia hulga katseid ning teha sealjuures tarku otsuseid, et analüüsida vaid olulist informatsiooni, sest transaktsioonide keskmine arv ulatub üldiselt sadadesse.

4.3.1 Andmevahetuse filtreerimine

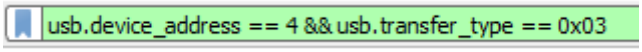
Lihtsuse huvides tasub alustada sellise operatsiooniga nagu „GET OPLC information“, mille abil UniDownLoader kasutajaliideses kuvati välja mõningased andmed PLC kohta. Kuna Wireshark kuvab kõikide seadmete USB andmevahetuse samaaegselt, siis esimese asjana on vaja rakendada filtreid selleks, et filtreerida vaid need transaktsioonid, mis on seotud PLC-ga. PLC aadressi saab teada, kui jälgida andmevahetust hetkel, kui seade ühendatakse arvutiga. Kui seadme aadress oleks 4, siis esimeseks filtriks olekski:



Joonis 5. Aadressi järgi filtreerimine

Järgmine asi, mida vaadata, on transaktsioonide arv. Seda numbrit saab korduskatsetel võrrelda ning see annab aimu, kas andmevahetus on iga kord sama. „GET OPLC informatsion“ puhul on kolme katse tulemusena transaktsioonide arv vastavalt 331, 334, 331. Tõenäoliselt einevad numbrid omavahel, sest nende hulgas on duplikeeritud transaktsioone. Lisaks sellele selgub, et eksisteerib väike arv suuremahulisi transaktsioone ning neid on keskmiselt ~2% kõikidest transaktsioonidest. Suuremahulised transaktsioonid asuvad andmevahetuse lõpupool. Saab kiiresti selgeks,

et need transaktsioonid on olulised, kuna samade operatsioonide puhul on need alati konstantsed, kuid erinevate operatsioonide puhul on teatud erinevused. Suuremahulisi transaktsioone saab samuti Wiresharkis teistest eristada (vt joonis 6). Arvule 0x03 vastab seal suuremahuline transaktsioon. Samas 0x02 oleks juhttransaktsioon.



Joonis 6. Transaktsioonitüüp filtrina

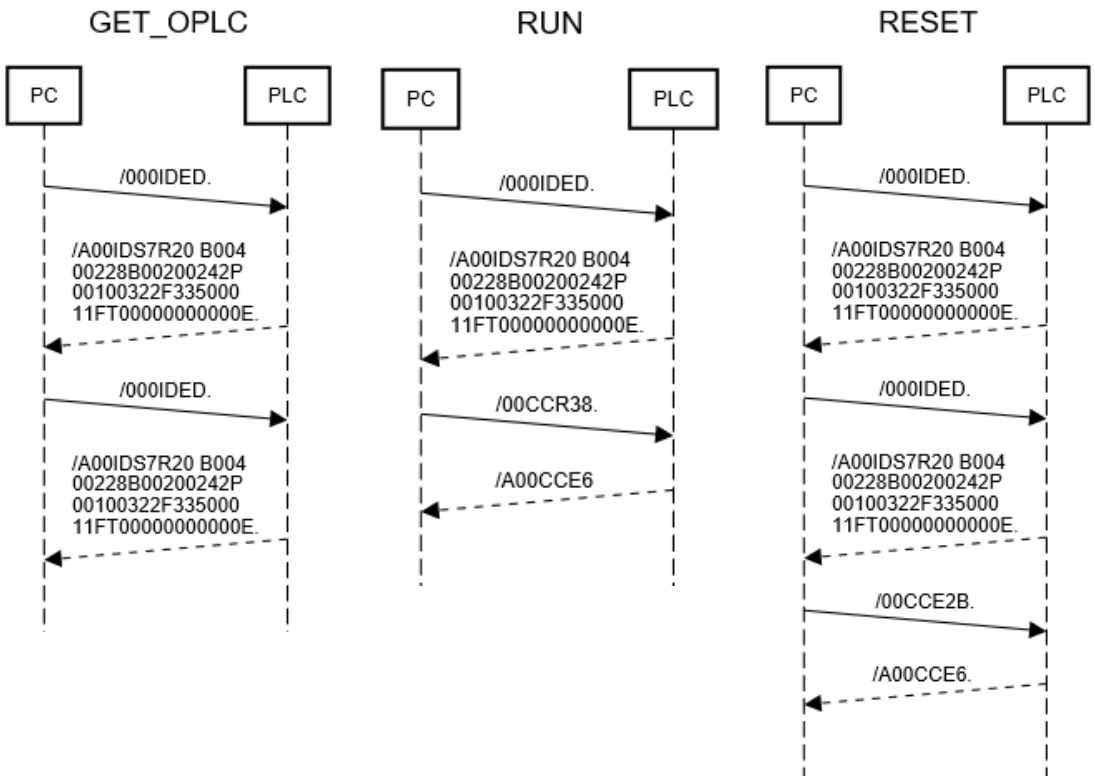
Saab teha lihtsustatud eelduse, et kuna suuremahulised transaktsioonid on sisuliste andmete esitamiseks, siis tasub just neid täpsemalt uurida ja võrrelda. Samas on autor veendunud, et ilma juhttransaktsioonide toimumiseta operatsioone läbi viia pole võimalik, kuna toimub teatav seadistamine, mille tähendus on veel ebaselge. Edaspidi võetakse aga lahenduse implementeerimisel juhttransaktsioonide olemasolu arvesse. Tabelist 1 selgub, et kõikide lihtsamate operatsioonide puhul on transaktsioonide arv suhteliselt väike ning sisulisi andmeid vahetatakse suhteliselt vähe. Samas viimane operatsioon, mille sisuks on PLC-sse laadida programm, mis seadistab kontrolleri unikaalse nime ja staatilise IP-aadressi, on tunduvalt mahukam, sisaldades kokku 2179 transaktsiooni, millest 112 on sisulised andmed. Autor arvab, et 112 transaktsiooni kulubki reaalse masinkoodi ülekandmiseks kontrollerisse. Lihtsuse huvides tasub kõigepealt analüüsida ja implementeerida lahendus lihtsamate operatsioonidele ning alles hiljem pöörduda tagasi programmi laadimise juurde.

Tabel 1. Transaktsioonide arv erinevate operatsioonide puhul

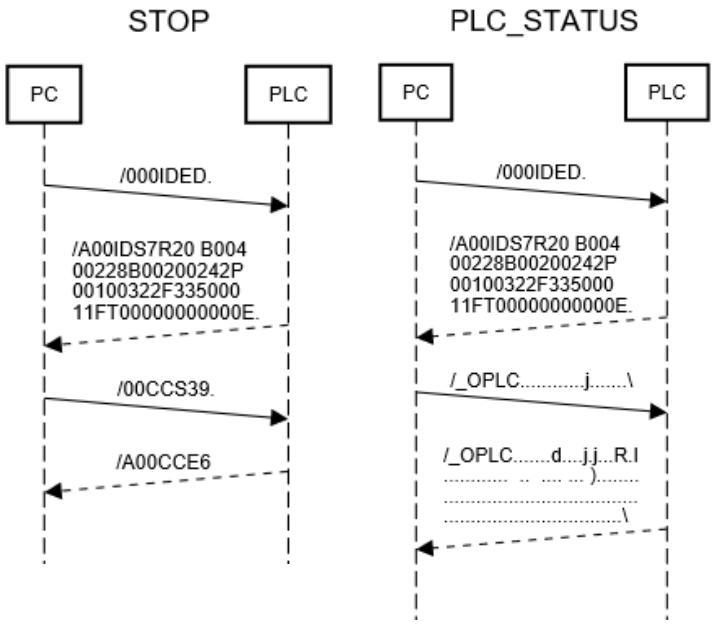
Operatsioon	Transaktsioonide arv kokku	Suuremahuliste transaktsioonide arv
GET OPLC information	331, 334, 331	4
RESET	341, 341, 341	6
RUN	334, 334, 334	4
STOP	334, 331, 334	4
PLC STATUS	338	4
STOP-DOWNLOAD-RESET	2179	112

Allpool (vt joonis 7, 8) on kujutatud kõikide lihtsamate operatsioonide interaktsioonidiagrammid, kus toimub sisuliste andmete vahetamine arvuti ja PLC vahel. Andmed on kodeeritud ASCII väärtusteks, kuna ka Wireshark presenteerib neid nii ning

ühtlasi on selline kirjapilt rohkem loetavam. Kuna igale võimalikule baidi väärtusele ei vasta tingimata ASCII sümbol, siis on need baidid asendatud punktiga.



Joonis 7. GET_OPLC, RUN ja RESET operatsioonide interaktsioonidiagrammid



Joonis 8. STOP ja PLC STATUS operatsioonide interaktsioonidiagrammid

4.3.2 Järeldused

„GET OPLC information“ operatsioon teeb sisuliselt kaks identset päringut PLC-le ja saab samuti kaks identset vastust. Kuna on teada, et sellest vastusest peaks saama välja lugeda erinevaid PLC-d identifitseerivaid andmeid, siis see selgitab ka seda, miks kõikvõimalike operatsioonide alguses viiakse läbi tegelikult sama protseduur. Üldistatult võib väita, et iga operatsioon koosneb kahest faasist: seadme identifitseerimine ja siis juba spetsiifilise käsu täimine ning vastuse saamine. Diagrammidelt on näha, et peale identifitseerimist on kõik käsud vastavalt operatsioonile erinevad ning seega eeldab autor, et just need käsud realiseerivad reaalse operatsiooni. Lisaks on RUN, STOP, RESET operatsioonide puhul lõpus ka sarnane sõnum USB seadmelt. Kuigi midagi kindlalt ei saa väita, arvab autor, et tegu on PLC poolse kinnitusega, et operatsioon on edukalt lõpule viidud.

4.4 USB andmevahetuse taasesitamine

Andmevahetus, mis toimub arvuti ja PLC vahel, on korraldatud vastava draiveri poolt. See draiver on installeeritud Windows operatsioonisüsteemi ning omab teadmisi sellest, milline peab olema andmevahetus vastavate operatsioonide korral. Võib oletada, et kui tegu on mõne spetsiifilise draiveriga, ei ole see kasutatav mitte millegi muu, kui Unitronics tarkvara poolt. Seega on tarvis andmevahetus pöördprojekteerida ning selleks läheb vaja vahendit, millega saaks mugavalt teha USB transaktsioone.

Pikemal uurimisel selgub, et vahendeid, millega USB transaktsioone edukalt teha, peaks leiduma igas programmeerimiskeeles. Javas oleks võimalik kasutada teegina Java4Usb-d, Pythonis PyUSB-d jne. Kõikide selliste teekide ühisosa on see, et nad põhinevad omakorda C programmeerimisekeeles loodud teegil nimega libusb. Windows operatsioonisüsteemis tekkis probleem kõikide võimalike ülalnimetatud teekidega. Linux-is ei tekkinud esialgsel katsetamisel probleeme ja seega sai kasutusele võetud Linux operatsioonisüsteem, et USB andmevahetust taasesitada.

Kuna libusb on olemuselt väga lihtne teek, siis ei pidanud autor vajalikuks kasutada Linux-is testimiseks libusb pealisehitusi nagu pyUSB või Java4USB. Olulised libusb funktsionaalsused, mida edaspidi ka kasutama hakatakse, on esitatud tabelis 2 [7].

Tabel 2. Libusb teegi kasutatavad funktsioonid

Funktsioon	Kirjeldus
libusb_open_device_with_vid_pid()	Tagastab viite objektile, mida on võimalik hiljem kasutada teostamaks transaktsioone. Identifitseerib objekti tootja id (VID) ja toote id (PID) järgi.
libusb_control_transfer()	Võimaldab teha juhttransaktsioone.
libusb_bulk_transfer()	Võimaldab teha suuremahulisi transaktsioone.
libusb_interrupt_transfer()	Võimaldab teha katkestustransaktsioone
libusb_detach_kernel_driver()	Eemaldab kerneli poolt automaatselt antud draiveri. See on vajalik, sest samal ajal ei saa ühel seadmel olla mitu draiverit.
libusb_attach_kernel_driver()	Seadmele lisatakse automaatne kerneli draiver.

4.4.1 Andmete formaat

Wireshark-ist nähtud transaktsioonid tuleks teisendada sobivale kujule. See on vajalik, et neid saaks töödelda ning siis juba taasesitada need kasutades libusb teeki. Wireshark pakub iseenesest filtreeritud transaktsioonide eksportimiseks csv, txt, json formaati. Lisaks klassikalistele formaatidele on võimalik transaktsioone teisendada ka C programmeerimiskeele massiivideks. Teisenduse tulemusena oleks vaja baidijada, mis iseloomustab igat transaktsiooni. Kuna viimane teisendusviis järjestab kõik transaktsiooni baidid baidijadana, siis on otstarbekas seda kasutada. Selline baidijada sisaldab transaktsiooni kohta kogu informatsiooni: USB seadme aadress, EP number, sisuliste andmete hulk (kui neid on), transaktsiooni suund ning samuti sisulised andmed ise (kui neid on). Baidijada 23. bait määrab ära transaktsiooni suuna. Nii on joonisel (vt joonis 9) esimese transaktsioonina kujutatud juhttransaktsiooni, samas teise puhul on tegu suuremahulise transaktsiooniga ning tema välju tuleb tõlgendada teisiti. Juhttransaktsioone saab tõlgendada vastavalt tabelile nr. 3 ning suuremahulisi transaktsioone tuleb tõlgendada vastavalt tabelile nr. 4.

```
static const unsigned char pkt68[36] = {
      0x1c, 0x00, 0xf0, 0xda, 0xce, 0xb5, 0x0d, 0x85,
      0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x08, 0x00,
      0x00, 0x01, 0x00, 0x03, 0x00, 0x00, 0x02, 0x08,
      0x00, 0x00, 0x00, 0x00, 0x41, 0x00, 0xff, 0xff,
      0x00, 0x00, 0x00, 0x00
};
static const unsigned char pkt367[35] = {
      0x1b, 0x00, 0x10, 0x00, 0x3d, 0xb5, 0x0d, 0x85,
      0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x09, 0x00,
      0x00, 0x01, 0x00, 0x03, 0x00, 0x01, 0x03, 0x08,
      0x00, 0x00, 0x00, 0x2f, 0x30, 0x30, 0x49, 0x44,
      0x45, 0x44, 0x0d
};
```

Joonis 9. Juht ning suuremahuline transaktsioon baidijadana

Tabel 3. Juhttransaktsiooni baidijada tõlgendamine

Tähendus	**Näiteväärtus**	**Asukoht baidijadas**
bmRequestType	0x41	29. bait
bRequest	0x00	30. bait
wValue	0xFFFF	32, 31 baidid
wIndex	0x0000	34, 33 baidid
wLength	0x00	35. bait
transfer type	0x02	23. bait

Tabel 4. Suuremahulise transaktsiooni baidijada tõlgendamine

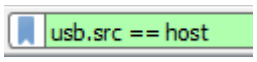
Tähendus	**Näiteväärtus**	**Asukoht baidijadas**
transfer type	0x03	23. bait
EP number	0x01	22. bait
data length	0x08 = 8 baiti	24. bait
data	`0x2f,0x30,0x30,0x49,0x44, 0x45,0x44,0x0d`	Viimased 8 baiti.

4.4.2 Programmi genereerimine

Esimese kitsendusena tuleb vaatluse alla vaid transaktsioonid, mille suund on arvuti poolt USB seadme poole. Interaktsioonidiagrammist järeldus, et RESET operatsiooni korral ei ole sisuliselt oluline, mida USB seade vastab, sest käskluse saadab arvuti ning mingit valideerimist otseselt vaja ei ole. Jah, seade saadab küll arvutile oletatava kinnituse, et operatsioon oli edukas, kuid see pole operatsiooni reaalseks läbiviimiseks hetkel oluline.

Filtreerimaks vaid transaktsioone, suunaga seadme poole, rakendatakse Wireshark-is filtrit (vt joonis 10). Kuna baidijada tähendus on nüüd mingil määral selge, siis järgmiseks

sammuks on baidijada konverteerimine libusb teegi poolt käivitatavateks transaktsioonideks/funktsioonideks. Sisuliselt on plaanis automaatselt genereerida kompileeruv C kood, mis taasesitaks transaktsioone, mis vastavate parameetritega sai eelnevalt filtreeritud.



Joonis 10. Kindlasuunaliste transaktsioonide filtreerimine

Baidijada konverteerimiseks on kõige mugavam kirjutada skript, mis konverteerimise ellu viib. Selleks kirjutatakse Python programmikood, mis igast transaktsiooni kujutavast baidijadast loeb välja olulise informatsiooni (vt tabel 3, 4) ning genereerib vastavat tüüpi transaktsiooni koos argumentidega. Sellise programmi käivitamisel konverteeritakse kõik eelpool nimetatud baidijadad (vt joonis 9) vastavateks C programmeerimiskeele ridadeks (vt joonis 11)

```
libusb_control_transfer(dev_handle, 0x41, 0x00, 0xffff, 0x0000,
in_buffer, 0x00, 1000);

unsigned char out_buffer_0[8] = {0x2f, 0x30, 0x30, 0x49, 0x44, 0x45,
0x44, 0x0d};
libusb_bulk_transfer(dev_handle, 1, out_buffer_0, 8, jaactual, 1000);
```

Joonis 11. Genereeritud read

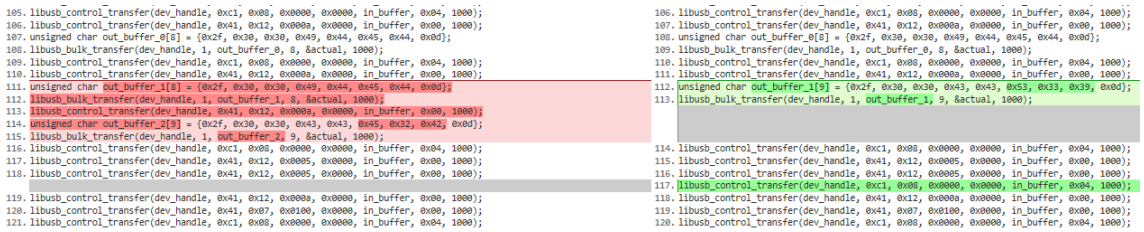
Nagu näha, on tulemuseks üks juhttransaktsioon ning üks suuremahuline transaktsioon, kus sisuliste andmete maht on 8 baiti. Seega eksisteerib lahendus andmevahetuse taasesitamiseks. Kuna igale transaktsioonile vastab nüüd sisuliselt üks rida, tekivad teatud võimalused vastata vastamata küsimustele, mis varem (vt 4.3 sissejuhatus) tõstatati. Näiteks see, kuidas erinevad operatsioonid omavahel erinevad ning kuidas leida need erinevused.

4.4.3 Kahe teksti sarnasus

Arvutiteadusest on teada, et kahe sõne sarnasust saab mõõta sellise algoritmiga nagu LCS (*longest common subsequence*). LCS pikim ühine alamsõne on algoritm, mis leiab pikima sümbolite jada, mis leidub nii esimeses kui ka teises sõnes. Nii nagu üks sõne koosneb sümbolitest, koosneb ka üks PLC operatsioon transaktsioonidest. Kuna iga transaktsioon asub eraldi real, siis LCS taandub pikima ühise transaktsioonide jada leidmisele.

Klassikaline vahend, mis LCS algoritmi implementeerib, on Diff. Diff võimaldab välja tuua muutused kahe tehtud operatsiooni transaktsioonide vahel. Kõik, mis kahe operatsiooni puhul jäi konstantseks, kuulub kindlasti ka alamjadasse ning seega tuvastatakse need kui samad tegevused. Teiselt poolt on Diff ka väga paindlik, kuna transaktsioonid ei pea esinema jäigalt üksteise järgi. Transaktsioonide vahel võib esineda teatud segadust, kuid niikaua, kui transaktsioon aitab kaasa pikima transaktsioonide alamjada moodustamisse, ei puuduta teda sellised häired. Sisuliselt kuuluvad alamjadasse vaid olulised transaktsioonid, kuna ainult siis saavutatakse maksimaalne sarnasus operatsioonide vahel. Võrreldes samu operatsioone omavahel, saab lõpliku kinnituse eeldus, et samade operatsioonide transaktsioonide hulk on identne. Kui võrrelda omavahel RESET ja STOP operatsioone, siis mitmesajast transaktsioonist on vaid mõned erinevad. Näiteks STOP operatsiooni puhul tehtud 45. transaktsioon puudub RESET transaktsioonide järjestusest (vt joonis 12). Peale selle selgub, et ainsateks sisulisteks erinevusteks ongi suuremahuliste transaktsioonide andmed ja nende maht (vt joonis 13). See erinevus oli suuresti juba kajastatud erinevate operatsioonide interaktsioonidiagrammides.

Joonis 12. Erinevus 45. transaktsioonis



Joonis 13. Erinevused suuremahulistes transaktsiooonides

4.4.4 Testimine

Võib tekkida õigustatud küsimus selles osas, kas ka lahendus reaalselt töötab. Kuigi on loodud protseduur, mis teoreetiliselt peaks töötama, tuleb läbi viia ka mõned katsed. Kuna protseduur on suhteliselt üldine, ei pea selle kasutamine olema seotud just Unitronics PLC andmevahetusega. Just seepärast testitakse lahendust esmalt kahel erineval seadmel. Nendeks seadmeteks on Arduino UNO arendusplaat ning HP printer.

Arduino UNO arendusplaadi jaoks kirjutatakse valmis programm, mis programmeerib arendusplaadi. Samaaegselt salvestatakse andmevahetus arvuti ja Arduino vahel. Saadud transaktsioonid filtreeritakse ning konverteeritakse C koodiks kasutades eelpool kirjeldatud protseduuri. Üllatuseks toimub mitmesaja rea kompileerimisel ja käivitamisel tõesti arendusplaadi programmeerimine ning selle tõestusena hakkab LED tuli vilkuma. LED tule perioodiline vilkumine oli täpselt see, mis originaalsete vahenditega arendusplaadile programmeeriti.

HP printeri jaoks on protseduur analoogiline. Transaktsioonide hulk on muidugi tunduvalt suurem Arduino arendusplaadi omast, kuid ka siin tasub katsetamine end edukalt ära. Printer alustab printimist ja prindib täpselt selle, mida vaja.

Unitronics PLC erinevate operatsioonide katsetamisel tekib aga tagasilöök. Esialgu seletamatul põhjusel ei reageeri PLC näiteks RESET operatsioonile kuidagi. Selles järeldub, et lahenduses on midagi, millega pole arvestatud. Samas tundub toimunu äärmiselt veider, kuna sisuline erinevus Arduino UNO ja PLC programmeerimisel vahel puudub.

4.4.5 Järeldused senisest lahendusest

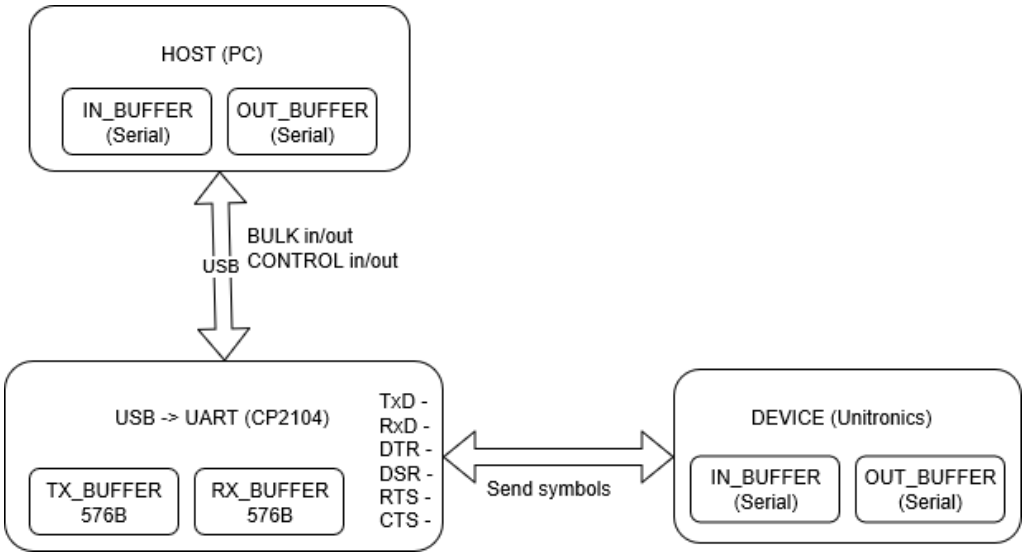
Senine meetod USB liikluse taasesitamiseks spetsiaalse teegiga ebaõnnestus. Kõige tõenäolisem põhjus seisneb selles, et kuna USB transaktsioonidest enamus on juhttransaktsioonid ning nende sisuline tähendus pole teada, siis pole võimalik ka öelda, kui õige on neid üldse saata. Tehes aga sama operatsiooni mitmeid kordi, jääb USB liiklus identseks ning seega mõnes mõttes ei tohiks nende tähendus oluline olla. Maksimaalselt võib erineda nende transaktsioonide järjestus. Näiteks Diff-i abil võrreldes tuli transaktsioonide järjestuse erinevus tihti esile. Sellest võib järeldada, et kohati ei ole isegi saatmise järjekord oluline. Ka ajastus on asi, millega eksida ilmselt ei saanud, kuna võrreldes esmakordselt esitatud ning taasesitatud transaktsioonide saatmise ajalist intervalli, siis need kattuvad.

Kuna selline lahendus ei tööta just PLC puhul, siis ilmselt peab eksisteerima mingi iseärasus, mis suhtlust takistab. Selleks, et siiski operatsioonid tööle saada PLC-l, oleks arukas veelgi rohkem süveneda andmevahetuse tehnilisse sisusse.

4.5 Järjestikliidese andmevahetuse taasesitamine

4.5.1 Draiveri analüüs

Süvenedes rohkem andmevahetuse tehnilistesse detailidesse selgub, et senine lahendus on olnud kohati abstraktne. Unitronics tarkvara poolt kasutatava draiveri kohta tehtud detailsemas uuringus saab selgeks nii mõndagi. Varasemalt tehtud eeldus, et draiveri puhul on tegu Unitronics spetsiifilise tarkvaraga, osutus valeks. Nimelt on draiveri puhul tegemist tarkvaraga, mis ei suhtlegi PLC-ga otse. Draiver suhtleb hoopis riistvaralise kiibiga CP201x, mille ülesandeks on USB andmevahetuse teisendamine UART andmevahetuseks. Sarnast kiipi kasutatakse väga paljudes erinevates kontrollerites just samal eesmärgil ning kõikidel operatsioonisüsteemidel on vaikimisi selle kiibi jaoks draiver olemas. Need faktid osutuvad väga oluliseks, kuna nendest järeldub, et Unitronics tarkvara kasutab üldlevinud seadet oma sõnumite edastamiseks ning kuna vähemalt Linux-i puhul on tegu avatud lähtekoodiga draiveriga, siis saab aimdust sellest, millised võimalused Unitronics tarkvaral põhimõtteliselt olla saavad mõne seadmega suhtlemiseks. Kogu funktsionaalses on ära piiratud selle draiveri tarkvaraga. Kõige olulisem avastus on see, et CP210x puhul on tegu nn. virtuaalse järjestikliidesega (*virtual COM*). See tähendab, et kuigi draiver suhtleb kiibiga USB protokolli kasutades, siis arvutis olev rakendus suhtleb draiveriga järjestikprotokolli (RS232) järgides (vt joonis 14). Ehk saavutatakse olukord, kus PLC-ga suhtlemisel abstraheeritakse vahepealne USB ühendus ning kogu suhtlemine taandub kahe UART seadme omavaheliseks suhtluseks. Erinevates programmeerimiskeeltes leidub sellise suhtluse jaoks erinevaid teeke. Pythonis PySerial, Javas JSSC jne. Linuxis taanduks draiveriga suhtlemine seadmefaili (nt. /dev/ttyUSB0) kirjutamiseks ja lugemiseks [8].



Joonis 14. Uus teadaolev andmevahetuse skeem

4.5.2 UART

UART (*Universal Asynchronous Receiver/Transmitter*) on mikrokiip, mis implementeerib järjestikliidese, mille abil on võimalik suhelda teiste järjestikliidest kasutavate seadmetega. Järjestikliides on RS-232 standardiga ühilduv port. RS-232 protokollist tulenevalt on andmevahetuse elementaarne ühik üks sümbol ning selle sümboli pikkuseks on olenevalt seadistusest 7, 8, 9 bitti. Lisaks võib rakendada sümbolitele paarsuskontrolli. Mitme sümboli eristamiseks üksteisest kasutatakse algus- ja lõppbitte. Kuna tegemist on asünkroonse andmevahetusega, siis mõlemad osapooled peavad teadma nii sümbolite saatmise kui ka vastuvõtmise kiirust. Vastasel korral on tulemused valed. See määratakse ära sellise atribuudiga nagu *baudrate* .

4.5.3 CP210x

CP210x on kiip, mis teisendab USB andmevahetuse UART andmevahetuseks. CP210x kiibi andmelehe (*datasheet*) uurimisel tuleb välja palju olulist informatsiooni. Nimelt on seal esitatud tabel, kus on ära kirjeldatud kõikvõimalike USB juhttransaktsioonide tähendused. Juhttransaktsioone saab USB seadme tootja ka ise implementeerida ning enamus juhttransaktsioone, mis eelnevalt arusaamatuks jäid, on nüüd kergesti tõlgendatavad selle tabeli abil. Näiteks tuleneb sellest tabelist, et ühe juhttransaktsiooni eesmärgiks on järjestikliidese andmevahetuskiiruse seadmine antud väärtusele. Lisaks sellele saab seal teada, kas paaruskontrolli tehakse ning milline on lõppbittide arv. See on

oluline informatsioon, sest ainult õiged parameetrid võimaldavad andmevahetust kahe seadme vahel läbi viia [9].

4.5.4 Katsetamine

Nüüd, kui on teada, et PLC-ga peaks saama suhelda järjestikliidest kasutades ning et originaalne Unitronics tarkvara seda just nii ainult saigi teha, siis tuleks katsetada, kas erinevate käskluste saatmine läbi järjestikliidese ka töötab. Siinkohal on võetud eelduseks, et käsklused on interaktsioonidiagrammidel välja toodud sümbolite jada ehk suuremahulistes USB transaktsioonides olevad andmed. Selleks kasutakse tarkvara nimega RealTerm, mis sisuliselt on järjestikliidese klient. Et suhtlemine oleks võimalik, on vaja teada viit asja:

- Paarsuskonrolli tüüp – USB juhttransaktsioonidest tuli välja, et paarsuskontrolli ei kasutata.
- Andmevahetuse kiirus – PLC-le on manuaalselt seadistatud andmevahetuskiirus, millega viimane sümboleid loeb. Seega peab ka arvutipoolne rakendus olema sama kiirusega, sest tegu on asünkroonse andmevahetusega.
- Andmebittide arv – USB juhttransaktsioonidest tuli välja, et andmebitte oli täpselt ühe baidi jagu ehk 8 bitti.
- Lõppbittide arv – USB juhttransaktsioonidest tulenevalt kasutatakse selleks 1 bitti.
- Vookontrolli parameetrid – USB juhttransaktsioonidest tulenevalt vookontrolli ei kasutata.

Tekidades saadud parameetritega ühendus, saab alustada RealTerm-iga andmete saatmist. Proovides samm-sammult erinevaid käsklusi saata, vastab PLC lõpuks sümbolite jadaga. Võrreldes saadud vastuseid interaktsioonidiagrammil olevate PLC vastustega, siis need langevad kokku. Järelikult andmevahetus arvuti ja PLC vahel õnnestus. Selle kinnituseks õnnestus teha ka PLC-le korrektne RESET operatsioon. Kõik see tähendab seda, et tarkvara, mis PLC-sid juhtima hakkab, peab kasutama järjestikliidest. Antud töö tarkvaraline lahendus kasutab selleks sellist Java teeki nagu JSSC.

4.5.5 Sobiva formaadiga faili genereerimine

Kuna USB protokoll on arvuti ja PLC vahelise suhtluse puhul vaid alumiseks kihiks ning sisuline suhtlus on ehitatud (RS-232) selle peale, siis tasubki edaspidi vaid jälgida seda

kõrgetasemelist andmevahetust. Katsetamine andis kinnitust, et sümboleid saadetakse seadmete vahel kasutades selleks USB suuremahulisi transaktsioone. Seega oleks põhimõtteliselt võimalik välja filtreerida need transaktsioonid, mis olulist informatsiooni kannavad ning siis need vajadusel taasesitada.

Selleks tasub jällegi kirjutada näiteks Python skript, mis teisendab andmed sobivale kujule. Kuna igale arvuti poolt saadetud sümbolite jadale vastab PLC alati, siis tasub teha kindlas formaadis fail, mis sisaldab päringuid, mis arvuti antud operatsiooni täitmise jaoks tegema peab ning vastuseid, mida PLC-lt iga päringu kohta vastu oodata (vt joonis 15). Selliseid faile saab edaspidi tarkvara kasutada, et kindlaks teha:

- Mida saata? – kõik, mis jääb noolest vasakule
- Mis järjekorras saata? – ridade järjekord
- Mida vastu oodata? – kõik, mis jääb noolest paremale

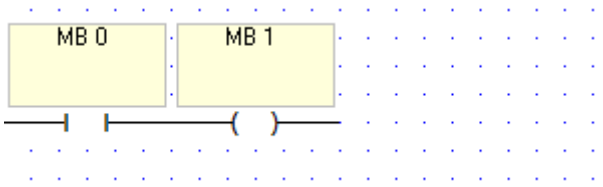
```
0x2f, 0x30, 0x30, 0x49, 0x44, 0x45, 0x44, 0x0d --> 0x2f, 0x41, 0x30, 0x30, ..
0x2f, 0x30, 0x30, 0x49, 0x44, 0x45, 0x44, 0x0d --> 0x2f, 0x41, 0x30, 0x30, ..
0x2f, 0x30, 0x30, 0x43, 0x43, 0x45, 0x32, 0x42, 0x0d --> 0x2f, 0x41, 0x30, ..
```

Joonis 15. RESET operatsiooni genereeritud fail

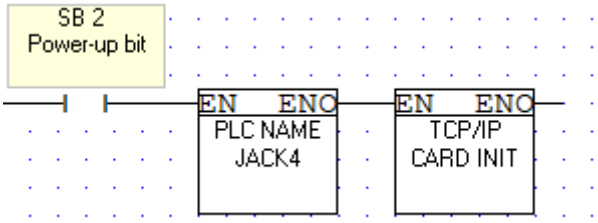
4.5.6 Kahe teksti sarnasus

Kui eelnevalt toimus USB andmevahetuse taasesitamine ning seetõttu ka mitme erineva andmevahetuse omavaheline võrdlus, siis nüüd, kui on olemas sobiv formaat kõrgemal tasemel oleva andmevahetuse esitamiseks, on erinevate operatsioonide võrdlemine sellel tasemel analoogiline. Kuna lihtsamad operatsioonid koosnevad maksimaalselt kolmest reast (vt joonis 15), siis nende võrdlemine ei ole keeruline. Võrguseadistuste programmeerimisega kaasneb aga ~64 rida ning seal on taas vaja kasutusele võtta Diff kahe teksti vahelise sarnasuste, erinevuste avastamiseks.

Selleks, et võrrelda võrguseadistuste programmeerimise tulemusena saadavat andmevahetust erinevate PLC-de puhul, koostatakse kaks PLC programmi: minimalistlik programm, kus võrguseadistusi pole määratud (vt joonis 16) ning programm, mis seadistab PLC-le staatilise IP ning selle seadme nime (vt joonis 17).



Joonis 16. Redeldiagramm, kus võrguseadistused puuduvad



Joonis 17. Redeldiagramm, kus on olemas võrguseadistus

Sarnasuste leidmiseks programmeeritakse mitmel erineval PLC seadmel kõigepealt esimene ja siis teine programm. See tähendab, et kõigepealt simuleeritakse olukorda, kus seade „kaob“ võrgust ning seejärel taastatakse võrguseadistused. Teisel programmil on sisuliselt kaks huvitavat sisendit: IP aadress ja seadme nimi. Neid varieerides ning seejärel andmevahetust pealt kuulates saab leida erinevused kahe andmevahetuse vahel. Selleks konverteeritakse saadud andmevahetused sobivasse formaati (vt joonis 15) ja võrreldakse Diff-i abil.

Mitme erineva andmevahetuse võrdlemisel selgub, et kuigi täpselt samade parameetrite korral jääb andmevahetus identseks, siis vaid ühe parameetri minimaalse muutmisega kaasneb tohutu erinevus andmevahetustes. Kui IP-aadressi viimases järgus olev number muudetakse ühe võrra suuremaks, siis kaasneb sellega suur hulk erinevaid ridu andmevahetustest genereeritud failides. Samas näitab Diff, et need erinevused esinevad täpselt samades kohtades.

Eelpool saadud erinevuste loogiline seletus saab vaid olla selles, et antud andmevahetuse tasemel on kasutusel Unitronics tarkvara poolt loodud andmevahetusprotokoll, mille sisust on raske aru saada. Väga tõenäoliselt kasutatakse iga päringu ja vastuse juures kindlat veatuvastust ning see on ka põhjus, miks minimaalne muutus toob kaasa andmevahetuses suured erinevused. Veatuvastuse olemasolu kinnatab ka fakt, et kui muuta IP-aadress ära mitte programmi sees, vaid genereeritud ridade sees, siis vastav rida,

mis andmetena IP-aadressi PLC-le saadab, vastust sealt tagasi ei saa. See tähendab, et PLC seadme poolt toimus IP-aadressi valideerimine. Selle valideerimise ainuke võimalik viis sai aga olla mõni veatuvastusalgoritm.

4.5.7 Testimine ja järeldus

Seda, kas võrguseadistuste programmeerimine õnnestub, saab testida parsseerides sobivas formaadis genereeritud failist järjest read ning esitates need saadud järjekorras kasutades järjestikliidest. Seda lõplik tarkvaraline lahendus Javas ka tegema hakkab. 64 realise faili taasesitamisel selgub, et PLC programmeerimine õnnestub ning PLC seadistusi uurides selgub, et nii seadme nimi kui ka staatiline IP-aadress muutusid sellisteks nagu antud faili puhul tahetud. Lisaks õnnestub ka kohalikus võrgus tehtud päring uuele IP-aadressile. Seega võib öelda, et vähemalt teatud seadistuste programmeerimine PLC-le on õnnestunud.

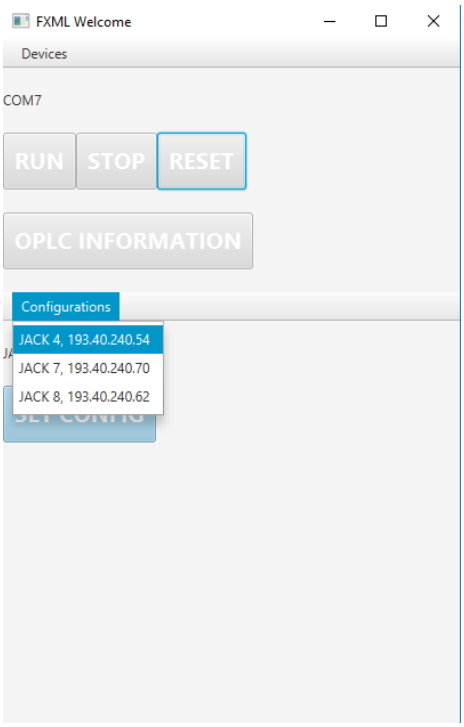
Kuna laboris on N erinevat PLC seadet, siis kõikide nende seadistamiseks on kaks võimalust:

- Modifitseerides ühte faili, muutes seal parameetreid
- Genereerides iga PLC jaoks oma võrguseadistuste fail

Nagu selgus andmevahetuste võrdlustest Diff-i abil, siis ei piisa failis parameetrite muutmisest, et PLC-d korrektselt programmeerida. Ühe lihtsa parameetri muutmine tooks kaasa ka paljude teiste ridade muutumise antud hetkel arusaamatul moel. Ilmselt on tegu kindla protokolliga, tehnilises mõttes olekuautomaadiga, mille uurimine on aga väga suure keerukusega ning seega tundub antud labori probleemi lahendamiseks piisav, kui genereerida N võrguseadistuste faili (N suurusjärk ~20). Tegelikult võib genereerida nii palju faile, kui palju on erinevaid võrguseadistusi ning rakendada neid vastavalt tahtele sobivale PLC seadmele. Seega võib genereeritud võrguseadistuste failide arv olla ka suurem, kui N.

5. Kasutajaliides tarkvarale

Võrguseadistuste taastamiseks ja elementaaroperatsioonide teostamiseks mõeldud tarkvara implementeeritakse keeles Java ning järjestikliidesega suhtlemiseks kasutatakse teeki JSSC. Tarkvaras kasutatakse andmevahetuse analüüsi tulemusena saadud andmeid, et teostada nii võrguseadete taastamine kui ka erinevad operatisoonid (RUN, STOP, RESET jne). Tarkvara tuleb toime veatöötlusega erinevate probleemide puhul. Lisaks on ära lahendatud keerulised kasutusjuhud, kus seadmed, millega tarkvara suhtlema peab, nõuavad erinevaid andmevahetuskiiruseid (*baudrate*). Tarkvara funktsionaalsuse demostreerimiseks on arendatud ka primitiivne platvormist sõltumatu kasutajaliides kasutades JavaFX-i. Reaalsesse kasutusse kasutajaliides siiski ei lähe, sest tegu on eelkõige serveri rakendusega.



Joonis 18. Kasutajaliides funktsionaalsuse demostreerimiseks

Tarkvaras on eeldefineeritud 3 võrguseadistuste faili mis annavad valitud seadmele (COM7) vastava nime ja IP-aadressi. Sisemiselt taasesitatakse analüüsi käigus saadud ning sobivale kujule teisendatud andmevahetusread (vt joonis 15). Elementaaroperatsioonid viivad sisemiselt läbi operatsioone nagu seda on kujutatud interaktsioonidiagrammidel (vt joonis 7, 8).

6. Kokkuvõte

Lõputöö eesmärgiks oli leida töötav tarkvaraline lahendus, mille abil taastada suvalise PLC seadme võrguseadistused laboris juhul, kui viimane on võrgust „kadunud“. Lisaks sellele uuriti ka elementaarsete käskluste (nagu RUN, STOP, RESET, PLC STATUS jne) andmist PLC seadmetele. Universaalse lahenduse väljatöötamine eeldas arvuti ja PLC vahelise andmevahetuse mitmekülgset analüüsi alustades selle interpreteerimisest ning lõpetades andmevahetuste omavahelise võrdlemisega.

Lõputöö käigus õppis autor tundma USB protokolli ülesehitust ning üldisemalt seda, kuidas arvuti suhtleb USB seadmetega. See on kasulik teave, sest nagu selgus erinevate draiverite uuringust, siis mitte ühtegi USB draiverit ei ole võimalik implementeerida, kui pole olemas arusaama USB protokollist. Selline süvendatud teadmine annab loodetavasti aimdust sellest, kuidas toimub riistvaraga suhtlemine operatsioonisüsteemi tasemel.

Pikemas perspektiivis pakub huvi kontrollerite üldine haldamine. Kasutaja broneerib ajavahemiku, laeb (Visilogic tarkvara abil) koodi ja testib seda TTÜ-s asuval kontrolleril. See tähendab, et kõik asjad, mida seni tehakse läbi spetsiaalse kasutajaliidese, peaksid toimuma automatiseeritult keskse serveri ja PLC seadmete vahel. On hulk probleeme, mis sellise automatiseerimisega kaasnevad ning käesolev töö on ära lahendanud teatava alamhulga nendest probleemidest (PLC võrguseadistuste taastamine, haldamine). Antud töö on andnud aimdust sellest, millise keerukusega on tarkvara, mille abil toimub hulga erinevate kontrolleritega suhtlemine.

Käesoleva bakalaureusetöö edasiarendusena oleks võimalik uurida seda, kuidas arvuti ja PLC vaheline kommunikatsiooniprotokoll täpselt töötab. Praegusel juhul saavutatakse lahendused suuresti andmete taasesitamise tulemusena, kuid on olemas reaalsed põhjused, miks neid andmeid vastavalt esitatakse ning milline on nende interpretatsioon. Tõenäoliselt tasuks uurida teaduslikke viise, kuidas katseandmeid kasutades mõnda protokolli esindavat olekuautomaati konstrueerida.

7. Kasutatud kirjandus

[1] Microsoft, „Inspect,“ [Võrgumaterjal]. Kättesaadav: https://msdn.microsoft.com/en-us/library/windows/desktop/dd318521(v=vs.85).aspx. [Kasutatud 15 mai 2018].

[2] GitHub, Inc., „GitHub,“ [Võrgumaterjal]. Kättesaadav: https://github.com/2gis/Winium. [Kasutatud 15 mai 2018].

[3] HHD Ltd., „Hhdsoftware,“ [Võrgumaterjal]. Kättesaadav: https://www.hhdsoftware.com/device-monitoring-studio. [Kasutatud 15 mai 2018].

[4] HHD Software Ltd., „Freeusbanalyzer,“ [Võrgumaterjal]. Kättesaadav: https://freeusbanalyzer.com/. [Kasutatud 15 mai 2018].

[5] Total Phase, Inc., „totalphase,“ [Võrgumaterjal]. Kättesaadav: https://www.totalphase.com/products/beagle-usb12/. [Kasutatud 15 mai 2018].

[6] C. Peacock, „Beyondlogic,“ [Võrgumaterjal]. Kättesaadav: https://beyondlogic.org/usbnutshell/usb3.shtml. [Kasutatud 15 mai 2018].

[7] doxygen, „libusb.sourceforge.net,“ 24 Märts 2018. [Võrgumaterjal]. Kättesaadav: http://libusb.sourceforge.net/api-1.0/. [Kasutatud 15 Mai 2018].

[8] Checker Software Inc., „diffchecker,“ [Võrgumaterjal]. Kättesaadav: https://www.diffchecker.com/. [Kasutatud 15 mai 2018].

[9] Silicon Laboratories, „Silabs.com,“ [Võrgumaterjal]. Kättesaadav: https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers. [Kasutatud 15 mai 2018].

[10] Silicon Laboratories Inc., „silabs.com,“ 2017. [Võrgumaterjal]. Kättesaadav: https://www.silabs.com/documents/public/data-sheets/CP2102-9.pdf. [Kasutatud 15 mai 2018].