TALLINNA TEHNIKAÜLIKOOL

INSENERITEADUSKOND

Tartu kolledž

ENOCEAN RAADIOLIIKLUSE JÄLGIMISSEADME LOOMINE TARTU KOLLEDŽI ELUSLABORATOORIUMILE

CREATION OF ENOCEAN RADIO TRAFFIC MONITORING DEVICE FOR TARTU COLLEGE LIVING LABORATORY

BAKALAUREUSETÖÖ

Üliõpilane: Konstantin Abarenkov

Üliõpilaskood: 183550EDTR

Juhendaja: Ago Rootsi, lektor

Tartu 2021

AUTORIDEKLARATSIOON

Olen koostanud lõputöö iseseisvalt.
Lõputöö alusel ei ole varem kutse- või teaduskraadi või inseneridiplomit taotletud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

"....." ..... 2021..

Autor: .....
/ allkiri /

Töö vastab bakalaureusetöö/magistritööle esitatud nõuetele

"....." ..... 2021..

Juhendaja: .....
/ allkiri /

Kaitsmisele lubatud

"....."..... 2021 .

Kaitsmiskomisjoni esimees .....
/ nimi ja allkiri /

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Konstantin Abarenkov (sünnikuupäev: 02.10.1985)

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „ENOCEANI RAADIOLIIKLUSE JÄLGIMISSEADME LOOMINE TARTU KOLLEDŽI ELUSLABORATOORIUMILE“,
mille juhendaja on Ago Rootsi,

1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.

3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

---

[1] *Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil.*

_____ (*allkiri*)

_____ (*kuupäev*)

TALTECH TARTU KOLLEDŽ LÕPUTÖÖ ÜLESANNE

Üliõpilane: Konstantin Abarenkov 183550EDTR

Õppekava, peaeriala: EDTR17/18 - Telemaatika ja arukad süsteemid

Juhendaja(d): Lektor, Ago Rootsi, +372 56629821

Konsultant: ……..(nimi, amet)

……..…… (ettevõte, telefon, e-post)

Lõputöö teema:

(eesti keeles) „ENOCEANI RAADIOLIIKLUSE JÄLGIMISSEADME LOOMINE TARTU KOLLEDŽI ELUSLABORATOORIUMILE"

(inglise keeles) CREATION OF ENOCEAN RADIO TRAFFIC MONITORING DEVICE FOR TARTU COLLEGE LIVING LABORATORY

Lõputöö põhieesmärgid:

1. Luua teoreetiline alus seadme loomiseks ning tarkvara kirjutamiseks
2. Panna seade kokku ning testida koostisosade ühilduvust
3. Kirjutada ning testida tarkvara, mis on vajalik seadme ülesannete täitmiseks

Lõputöö etapid ja ajakava:

Nr	Ülesande kirjeldus	Tähtaeg
1.		
2.		
3.		

Töö keel: Eesti keel **Lõputöö esitamise tähtaeg:** "....."..........2021.a

Üliõpilane: Konstantin Abarenkov ……..... "....."..............2021.a

/allkiri/

Juhendaja: Ago Rootsi ..... "....."..............2021.a

/allkiri/

Programmijuht: Helle Hallik ..... "....."..............2021.a

/allkiri/

SISUKORD

EESSÕNA

Käesoleva bakalaureusetöö ülesande on sõnastanud TalTech Tartu kolledži õppejõud Ago Rootsi. Tema poolt on korraldatud ka seadme koostiskomponentide hange. Juhendajaga olid kooskõlastatud nii lõputöö põhi- kui ka alamteemad. Empiirilise osa koostamise jaoks otsis autor infot avalikest allikatest ning teadusartiklitest. Allikate loetelu on esitatud töö lõpus. Lõputöö teema on suunatud Eluslaboratooriumi EnOceani seadmete parameetrite jälgimisseadme loomisele.

Autor tänab oma juhendajaid Ago Rootsi toetuse ja abi eest. Samuti soovib autor tänada oma peret toetuse ja mõistva suhtumise eest. Autor annab õiguse talletada antud tööd avaldamispiiranguteta TalTech Raamatukogu digikogus. EnOcean, Modbus, jälgimisseade, Eluslaboratoorium, bakalaureusetöö.

LÜHENDITE JA TÄHISTE LOETELU

API – *Application Programming Interface*, rakendustarkvara liides. See on ühte tarkvara teisega ühendav liides, mis võimaldab seadmetel või rakendustel andmeid vahetada või käske edastada

GND – *ground*, ühisklemm

HTTP – *HyperText Transfer Protocol*, võrguprotokoll andmete edastamiseks arvutivõrkudes

JSON – *JavaScript Object Notation*, andmevahetuse formaat

Rx – vastuvõtja

Tx – saatja

Transiiver – seade digitaal- kui analoogsignaalide saatmiseks ja vastuvõtmiseks

WiFi – IEEE 802.11 võrgutehnoloogia, mis kasutab raadiolaineid traadita kiire internetiühenduse tagamiseks

KFS – küberfüüsikalised süsteemid

EnOcean – tööväljavõrkude raadioprotokoll, mida kasutatakse eelkõige automaatikasüsteemides. Protokolli arendust ja tuge organiseerib EnOcean Allianss, millel on üle 400 liikme. Riistvaraliselt on EnOcean raadiseadmed väga väikese energiatarbega, mis võimaldab neid alternatiivseid energiaallikaid nagu näiteks miniatuursed päikesepaneelid, töös hoida.

CRC8 – *Cyclic Redundancy Checks,* vigade tuvastamise meetod, mida kasutatakse digitaalses andmesides

ESP3 – *EnOcean Serial Protocol 3,* EnOcean protokolli edastuskiht versioon 3, mis on praegu kasutuses.

ASCII – *American Standard Code for Information Interchange,* teksti digitaalseks edastamiseks ettenähtud kooditabel, kus iga sümbol (tähemärk, kirjavahemärk, number jne) on kirjeldatud 7 bitiga. (Kokku 128 sümbolit.)

RTU - *Remote terminal unit,* kaugterminaaliüksus

LSB- *Last Significant Bit,* vähima kaaluga (väärtusega) bitt

SISSEJUHATUS

Praegu Euroopa Liidus on võetud kindel suund keskkonnahoidliku tehnoloogia arendamiseks, mille põhieesmärgiks on CO_2 heitme oluline vähendamine. Hoonete energiakasutuse ehk kulutõhususe tõstmine on rohepööre lahutamatu osa. Näiteks, üks viis hoonete küttelt (ja jahutuselt) energiat kokku hoida, on muuhulgas sisekliimaautomaatika protsesside nutikas juhtimine. Vajadustest ühendada küberruumi füüsikalise maailmaga sündis uus eriala. TalTech ütleb selle kohta nii: „Info- ja kommunikatsioonitehnoloogiate (IKT) rakenduste arendamine on suure hoo sisse saanud just viimastel aastatel. Asjade internet, küberfüüsikalised süsteemid ning pilve- ja tarkvaratehnoloogiad on loonud täiesti uusi võimalusi, alates iseliikuvatest sõidukitest kuni tööstusrobotiteni" [1].

Eluslaboratoorium on loodud KFS peaeriala õppe- ja uurimis-keskkonnaks kus saab katsetada erinevat riistvara ja algoritme seades eesmärgiks hea sisekliima minimaalse energiakuluga. Eluslaboratooriumi funktsioonide hulk ja käsiteldav problemaatika on märksa laiem kui sisekliimaautomaatika, mille tugisüsteemi üht seadet käesolev lõputöö käsitleb, kuid see jääb väljapoole käesoleva töö mahtu. Oluline osa erinevate lahenduste otstarbekuse ja tõhususe analüüsist annab teha aja jooksul kogutud mõõte- ja seisundiandmete põhjal. Meie Eluslaboratooriumi kogutakse see info MeiePilve, mis on oma olemuselt andmebaas.

Mida täielikum on sellisel viisil kogutud andmebaas, seda tulemuslikum on selle analüüs. Käesoleva lõputöö ülesandeks on seadme väljatöötamine, mis aitab MeiePilve lisada EnOcean raadioprotokolli kasutatavate andurite poolt edastatavat infot, mida automaatikas alati otseselt ei kasutata ja mille edastamine Eluslaboratooriumi lüüs-kontrollereid kasutades koormaks ebavajaliku infoga ja oleks nende piiratud aadressvälja ja suhteliselt kõrge hinna tõttu ka ebarentaabel. TalTech Tartu Kolledži kämpuse hooned on muinsuskaitselised ja raadioandurid on õppeotstarbega automaatikasüsteemi jaoks paindlikum lahendus – võimaldades leida anduritele katseliselt parimaid paiknemiskohti. Juhtmelahendustega olnuks see tülikas.

Lisaks, aitab selline automaatikasüsteemist otseselt sõltumatu süsteem lahendada tulemuslikumalt rikkeolukordi kui automaatikakontrollerid mõne järjekordse ümberseadistamise käigus n-ö „kokku jooksevad" ja ise infot ei edasta. Õppetöös kasutatava automaatika rikkeolukorrad pole seejuures haruldased. Selline õppetööga seotud automaatika praktiliselt ei väljugi faasist, mida tuntakse järelseadistuse faasina: Sageli muudetakse seadistusi ja süsteemi konfiguratsiooni.

Kuigi seda pole otseselt planeeritud kuid valmiv seade võimaldab muuhulgas ka valideerida andmeid, mida salvestatakse automaatikasüsteemist otse. Mingite kahtluste korral saab käima panna paralleelse salvestuse loodava seadme abiga ja pärast kahel viisil kogutud andmeid võrrelda.

Kokkuvõtteks, käesoleva bakalaureusetöö eesmärgiks on lahendada üht Eluslaboratooriumi jaoks vajalikku KFS-ga seotud ülesannet. Täpsemalt, luua TalTech Tartu kolledži ruumidesse EnOceani vajalike seadmete raadioliikluse jälgimise süsteem. Loodav süsteem peab koguma, teisendama ja edastama lokaalsele MeiePilve andmebaasile informatsiooni huvipakkuvatest seadmetest. Peamiselt peaks süsteem fikseerima ning edastama uut informatsiooni MeiePilve andmebaasi (nt ruumi temperatuuri muutust, lüliti vajutamist, akna avamist jne). MeiePilve andmebaasile edastatud informatsioon peab olema ka MeiePilve serveri jaoks õiges formaadis (JSON).

1. LÄHTEÜLESANNE

Üheks Eluslaboratooriumi toimimise oluliseks protsessiks on selle automaatikaosa poolt kogutud mõõdiste ja automaatika enda muutujate seisundite säilitamine aegridadena MeiePilve andmebaasis. Aegrida on kogum mõõdistest, millega on seotud mõõtmise ligikaudne aeg.

Eluslaboratooriumi paljud seadmed edastavad andmeid ja seisundiinfot kasutades EnOcean raadioprotokolli. Eluslaboratooriumi funktsioneerimise seisukohalt on oluline, et kogu see informatsioon saaks logitud MeiePilv andmebaasi. Osa sellest informatsioonist jõuabki sinna Eluslaboratooriumi automaatikasüsteemi Schneider Electric MPM tüüpi lüüskontrollerite kaudu.

On aga palju sellist EnOcean raadioliiklust (üksikud aknakontaktid, osade ruumikontrollerite töötasandi seadmed, valgustuse lülitid), mille „jälgimine" pole lüüskontrollerite tasandil töötava automaatika seisukohast oluline ja selle informatsiooni edastamine MPM lüüs-kontrollerite kaudu oleks nende suhteliselt kallite seadmete piiratud aadressvälja raiskamine ning seetõttu pole kuluefektiivne. Õppe- ja teadustöö ning ka rikete kõrvaldamise tarbeks on aga vajalik, et võimalikult suur hulk raadioteel edastatavast infost saaks logitud MeiePilv andmebaasi.

1.1 Ülesande sisu

Käesoleva Lõputöö lähteülesanne on Eluslaboratooriumi praeguse projektijuhi, Ago Rootsi poolt sõnastatud alljärgnevalt:

LÄHTEÜLESANNE.

Tuleb välja töötada seade, mis kogub EnOcean raadioliikluse kaudu edastatavat informatsiooni ja salvestab selle kasutuskõlblikul kujul ning formaadis aegridadena MeiePilv andmebaasi. Erinevalt ühest varasemast tööst [2], kus kasutati EnOcean 320 seeria transiiverit tuleb seekord kasutada tööstuslikku toodet: Thermokon GMBH STC65+RS485 Modbus lüüsi, millele on tootja poolt loonud toetavad tarkvarad airScan ja STC65+RS485 Modbus, mis oluliselt kiirendavad ja lihtsustavad EnOcean perifeeriaseadmete sidumist lüüsiga, lüüsi esmast häälestamist ja konfiguratsioonide varukoopiate tegemist. Kõik see teeb Eluslaboratooriumi seadistamise oluliselt lihtsamaks ning võimaldab rikkis lüüsi väikese aja- ja energiakuluga välja-vahetada. Oluliselt suurem on ka tulemuse töökindlus.

Lõputöö tulemusena tuleb:

- leida sobiv kontroller või mikroarvuti, mis tagaks mitme lüüsi andmete kohandamise ja logimise MeiePilv andmebaasi;

- valida sobiv võrgutopoloogia: „lüüsid <–> kontroller / arvuti<–> MeiePilv", mis oleks pikaajaliseks andmekogumiseks piisavalt tõrkekindel (mitte halvem kui on Tartu kolledži LAN tõrkekindlus, sest selle kaudu käib suhtlus MeiePilv andmeserveriga);
- töötada välja vahevara, mis võimaldab erinevate EnOcean EEP konfiguratsioonidega EnOcean seadmete poolt edastatud pakettide andmeid EnOcean lüüsi registritest maha lugeda ja edastada selle informatsiooni MeiePilv API-le sobivalt vormistatult andmebaasi.
- vahevara juurde peab kuuluma eraldi ülekandetabel, kuhu saab, erivahendeid või eritarkvara kasutamata fikseerida seosed lüüside kanalinumbrite ja nende kanalite kaudu jälgitavate EnOcean seadmete MeiePilv tähiste vahel nii, et MeiePilve kirjutatav info oleks seotud jälgitava seadme MeiePilv tähisega, mitte lüüsi kanalinumbri või EnOcean ID-ga.
- vahevara peab kas ise või siis ülalkirjeldatud tabeli abiga ära tundma iga EnOcean seadme EEP profiili ja sellest lähtudes lüüsi registrite sisusid vastavalt tõlgendama.
- Vahevara peab olema suuteline tõlgendama kõigi käesoleva töö ajal olemasolevate MeiePilv EnOcean seadmete EEP profiilide andmeid. Nendeks profiilideks on:
  - LCD ruumikontroller (Thermokon SR06)
  - LSS10020032 magnetkontaktandur
  - PTM210 DB lüliti (ühe ja kahe klahviga)
  - 10020052/1020053 valgustuse andurid
  - LSS10020051 kohalolekuandur

2. LÕPUTÖÖ METOODIKA

Lõputöö tulemusena peavad loodud seadme eri osad olema võimelised omavahel suhtlema ehk nende töö peab olema „kooskõlastatud”. Informatsiooni kogumise eest vastutab lüüs. Informatsiooni töötlemise ning edastamise eest peab vastutama eraldi seadme osa, mille peal hakkab töötama autori loodud programm. Selle seadme osa valik ning programmi sisu selgub järgneva lõputöö käigus. Mõtteliselt saab lõputöö jagada kaheks osaks: empiiriliseks ning praktiliseks osaks. Empiirilises osas loob autor aluse praktilisele osale.

2.1 EnOceani protokoll ja andmeside omadused

Seadmed, mis lõputöö käigus ühendatakse lüüsiga ning mille andmeid hakatakse andmesideks töötlema ja edastama, kasutavad EnOceani raadioprotokolli. Seega on vaja põhjalikult tutvuda eelmainitud protokolli ja selle iseärasustega. See on tähtis nii lüüsist andmete õiges vormingus lugemiseks kui ka hiljem selleks, et saadud lüüsist oleks võimalik andmeid õigesti tõlgendada. Seega on vaja selgeks teha:

1. EnOceani seadmete profiilid ning andmete edastamise iseärasused;
2. andmete maht, nende liigid, tõlgendamise võimalused ning EnOceani paketi struktuur.

2.2 MeiePilve andmebaasi infovahetuse nõuded

- MeiePilve andmebaasiga tutvumine
- Toimingud ning MeiePilve ettevalmistamine andmete vastuvõtmiseks
- Nõuded MeiePilve andmebaasi edastatavale andmepaketile

2.3 Seadme koostisosade ja ülesehituse valik

- EnOceani andmepakette vastu võtva lüüsi andmevahetuse, ülesehituse ning ühilduvuse analüüs
- Andmeid töötleva ning edastava seadme valik, selle ülesehituse ja ühilduvuse analüüs
- Seadme kokkupanemine (seade peab olema paigaldatav Eluslaboratooriumi automaatikakilpi ning saama sealt elektritoidet)

2.4 Tarkvara

- Tarkvara programmi nõudmiste formuleerimine ning algoritmi disain
- Programmeerimiskeele valik
- Programmi testimine ja silumine vastavalt esitatud nõudmistele

2.5 Seadme testimine ja optimeerimine

- Seadmelõplik komplekteerimine, kilpi paigutamine ning testimine Eluslaboratooriumis
- Seadme arhitektuuri optimeerimine
- Tarkvara kohandamine

3. MEIEPILV

3.1. MeiePilve server

MeiePilv on server, mille peal töötab MongoDB andmebaasil töötav rakendus. MongoDB on NoSQL ehk mitterelatsiooniline dokumendipõhine andmebaas, kus andmeid hoitakse JSON formaadis dokumentides. Mitterelatsiooniline andmebaas ei pea omama kindlat eelmääratud andmemudelit, seega saab käsitleda mis tahes struktureerimata andmeid. Üldiselt tähendab see, et andmete kirjed salvestatakse hierarhilise puuna, mis annabki võimaluse eri kujuga dokumentide salvestamiseks (ka logid, pildid, seadmete andmed). [3] Viimase nelja aasta jooksul on MeiePilve lisatud võimekust lugeda andmeid NetAtmo tootepõhisest pilvest ning võtta vastu LoRaNeti ja oBIXi protokollide vahendusel saadetud andmeid. Neist oBIX protokolli kasutavad ka Eluslaboratooriumi MPM lüüskontrollerid andmevahetuseks MeiePilvega. Lisaks sellele on MeiePilvel ka liides otse TCP IP võrgu kaudu andmete vastuvõtuks konkreetse ülesehituseha JSON pakettidena.

Käesoleva töö raames on oluline, et andmebaasi saaks kasutada mõõtmistulemuste salvestamiseks ja haldamiseks. Serveril olevaid andmeid salvestatakse korraga RAID1 (*Redundant Array of Independent Disks Mode 1*) kahele kõvakettale. RAID kettasüsteem on iseseisev andmeturbesüsteem, sest hoiab ära andmekadu riistvaralisel tasandil [4]. MeiePilvel on eri andmete jaoks olemas omakorda oma vahevarad, mis töötavad füüsilises mõttes samas serverarvutis..

MeiePilv lõputöö alguses (suvi 2021) ka laivõrgus kättesaadav, sest serveril oli olemas WAN (*wide-area network*) IP. Ka MeiePilv püsib pidevas muutumises. Kui lõputöö alguses oli MeiePilv server ühendatud laivõrku, siis sügisel 2021 viidi see turvakaalutlustel üle TalTech Tartu kolledži sisevõrku. MeiePilv serveri haldusliides on http:\\ ja see pole laivõrgus turvaline. Kuna laiem (väljaspool Taltech'i) huvi MeiePilv serveri vastu esialgu puudub, on ta täna kättesaadav vaid TalTech sisevõrgust, sealhulgas ka väljastpoolt TalTech võrku VPN kaudu. Lõputöö eesmärgina valmiva seadme jaoks, mis asub samuti TalTech LAN võrgus see probleemi ei põhjusta.

3.2. EnOcean Monitor

MeiePilv andmebaasi struktuuripuu esimese tasandi harudeks on nn. võrgud (*network*). See haru on haldusliideses andministraatori õigustes vabalt loodav eeldusega, et harude nimetused omavahel ei kattu. Reeglina on võrk seotud ka ühega MeiePilv erinevate andmeallikate vastuvõtuvõimekusega (vt. 3.1.). „Enocean Monitor" on spetsiaalselt EnOcean liikluse jälgimiseks loodud võrgu nimetus. Füüsiliselt on see võrk seotud MeiePilv LAN kaudu andmete vastuvõtu- võimekusega. Lõputöö käigus

loodav seade hakkab lüüsi poolt vastuvõetud ning kontrolleri või mikroarvuti poolt töödeldud andmeid saatma just „Enocean Monitor" võrku.

MeiePilv andmebaasi struktuuripuu teise tasandi harudeks on seadmed (*device*) ja sealt järgmine hargnemine on mõõdised (*mesurment*). Andmete edastuse näide JSON andmeedastusvormingu vahendusel on toodud koodilõigul 1. JSON andmeedastusvorming on hea andmete edastamise formaat, sest see on programmeerimiskeelest sõltumatu. JSON andmete pakett kujutab endast assotsiatiiv-massiivi, kus salvestatud info on esitletud nimi - väärtus paaride kogumina. Näiteks MeiePilve võrk EnOcean Monitor päis peab sisaldama seadme nime ning võrgu nime (koodilõik 1).

```
{
"device_name":"25MK1"
"network_name":"EnOcean Monitor"
}
```

Koodilõik 1: JSON formaadis MeiePilve paketi päis

MeiePilv JSON paketi kehand (*body*) peab sisaldama ajamäära (*time*) ja sellele ajamäärale vastavad andmeid komadega eraldatuna. See on vaja olukorraks, kui seade edastab rohkem kui ühte mõõdetavat parameetrit (nt ruumiregulaator). Selleks, et sünkroniseerida MeiePilv serveri aega ja sinna andmeid edastava seadme ajaarvestust, kasutatakse kehandis ajamäära. Kui seade edastab MeiePilve andmed kohe pärast mõõtmist, pannakse ajamäär paketis nulliks („time": 0). See tähendab, et MeiePilv server seob paketi vastuvõtu aja oma ajaarvestuse järgi talle saadetud andmetega. Kui seade edastab andmeid viivitusega või kogub mitu erineval ajal tehtud mõõdistekogumit, märgitakse iga kogumi ajamääraks ajanihe saatmise hetke ja mõõtmise vahel. See on miinusmärgiga arv millisekundites, mille MeiePilv siis aegrea kirje vormistamisel saatmise hetkest maha arvestab. Andmete saatmiseks kulub küll ka mingi aeg, kuid automaatikas toimuvate protsesside suhteline aeglus lubab selle maksimaalselt millisekundites viivituse arvestamata jätta. Kui on aga mõõtmise ja andmete kohalejõudmise vahel on teadaolevalt mingi pikem ajavahemik, saab selle alati ajamäära kirjesse kirjutada. Näiteks juhul kui saatmisel ei saadud ühendust ja andmed saadetakse mingi aja järel uuesti. Niisugune ajaarvestuse süsteem võimaldab ära hoida kõikvõimalikke probleeme ajaga, mis tulenevad suve- ja talveaja ümberlülitusest ja ebatäpsetest kelladest alamseadmetes mis pilve andmeid edastavad. Nii on kõik andmed serveris aegridadena sama ajaarvestuse järgi. (koodilõik 2).

```
"body": [
 {
 "time": 0,
 "values": [25.5, 47.9]
 }
]
```

Koodilõik 2: JSON MeiePilve paketi kehand

Igal EnOceani seadmel on oma unikaalne identifitseerimiskood, mis üldjuhul on seadme peal ka kirjas. Meie lahenduse juures see praktilist väärtust aga ei oma. Palju olulisem ja informatiivsem on teine unikaalne kood, mis antakse Eluslaboratooriumis igale seadmele enne paigaldamist. Näiteks koodilõigul 1 toodud koodi 25MK1 esimesed kaks numbrit „24” tähistavad selle õpperuumi numbrit, kus seade paikneb ehk 204 (0 on keskelt ära võetud, kuna see on kõikide õpperuumide numbrites sees). „MK” tähistab seadme tüüpi – antud juhul on tegu magnetkontaktanduriga. Ja viimane number, „1”, tähistab antud tüüpi seadme järjekorranumbrit selles ruumis. Kui mõni seade läheb töökorrast välja ning see asendatakse teise samalaadse seadmega, siis kantakse eelmise seadme number üle uuele seadmele.

Töö autori arvates on selline tähistusviis sobiv, sest MeiePilve kasutaja saab numbrit lugedes kiiresti aru, kus antud seade paikneb ning mida see mõõdab või juhib. MeiePilve iga seadme nimeks (*device name)* peabki panema Eluslaboratooriumis antud nime. Järgmiseks tuleks määratleda antud seadme parameetrid ning mõõtühikud, mida seade saadab ning server vastu võtab (lisa 1). Nagu eespool mainitud, on tähtis roll väärtuste järjestusel saadetud loendis. Kui MeiePilve server leiab vastava nimega seadme, hakkab ta ülevalt alla täitma väärtuste lahtreid.

Seda, kas andmed on MeiePilve serverisse jõudnud või mitte, saab aru serverilt saadud vastuse põhjal. Kui andmed ei kajastu MeiePilve süsteemis, ei tähenda see veel seda, et andmed pole serverisse jõudnud. Probleem võib olla ka seadme nimetuses. Kui nimed ei sobi omavahel kokku, siis MeiePilve server neid andmeid ka ei salvesta. See on asjaolu, millega peab edaspidi programmis arvestama. Oluline on, et kasutaja saaks aru, kas andmepakett jõudis kohale või mitte (ja kui mitte, siis mis põhjusel).

4. ENOCEAN ANDMESIDE

Hooneautomaatika ning nutikate juhtimissüsteemide tarbeks on praeguseks loodud mitu raadiolainete vahendusel infot edastavat standardit. Antud lõputöös keskendub autor EnOcean Alliance standardile (ISO / IEC 14543-3-11). EnOceani tehnoloogia töötas umbes 20 aastat tagasi välja Siemens. Selle standardi suhteline ühilduvuse lihtsus ja toodete valik olid olulisteks eduteguriteks konkureerival turul. EnOcean Alliance peakontor asub Californias, koosneb 400 ettevõttest ning tänapäeval pakutakse üle 5000 toote. Viimase 20 aasta jooksul on EnOceani seadmeid paigaldatud enam kui miljonisse hoonesse. [6] EnOcean protokolli käsitlus käesolevas töös tuleneb töö lähteülesadest, mis keskendub Eluslaboratooriumi EnOcean liikluse jälgimisele.

4.1. EnOcean seadmete omadused

EnOceani seadmete põhiomadused:

- saavad seadme toimimiseks vajalikku energiat ümbritsevast keskkonnast (nt päikeseenergia, füüsiline energia), seetõttu on EnOceani seadmed ei vaja ka keemilisi vooluallikaid;
- tarkvara on suhteliselt lihtne;
- tüüpiline EnOceani traadita seade võib saata andmeid välitingimustes kuni 300 m ja sisetingimustes 30 m kaugusele. [7]

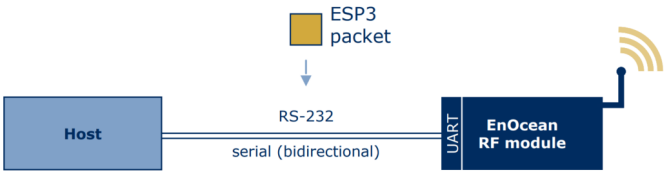
Võimekus kasutada tööks vajalikku energiat ümbritsevast keskkonnast on teinud EnOceani seadmed viimasel ajal väga populaarseks. Ühelt poolt tagab selline lahendus seadmete kõrge autonoomsuse; teisalt on kõik EnOceani standardi järgi töötavad seadmed „rohelised” ehk nt pole vaja toota (ega hiljem utiliseerida) "keemilisi toiteallikaid".

4.2. EnOceani protokoll

- EnOceani andmepakett on vaid 14 baiti pikk ning seda edastatakse kiirusega 125 kbit/s.
- Kasutatavad ülekandesagedused on järgnevad: 902 MHz (Põhja-Ameerika), 928,35 MHz (Jaapan), 868,3 MHz (Euroopa, Aafrika, Hiina, Lähis-Ida), 315 MHz (Aasia).[8]

Selle lõputöö lähtub sellest, et seadmed hakkavad andmepakettide edastamiseks kasutama EnOcean Serial Protocol 3-e (ESP3). Kommunikatsioon ESP3 protokolli järgi põhineb täisdupleks 3-juhtmelisel UART-ühendusel (Rx, Tx, GND). Tarkvara abil

kontrollitakse *handshake*'i ehk käepigistust nagu RS-232 standardi puhul (joonis 1). [9] UART sideprotokolli kasutavad standardsed EnOcean kommunikatsioonimoodulid andmevahetuseks kontrolleriga, mille juurde ta kuulub. USB pulk sisaldab USB – UART lüüsi, mille väljundisse on ühendatud EnOcean sidemoodul (*transiiver*). Töös kasutatav lüüs koosneb MODBUS – UART lüüsist ja EnOsean raadiomoodulist.

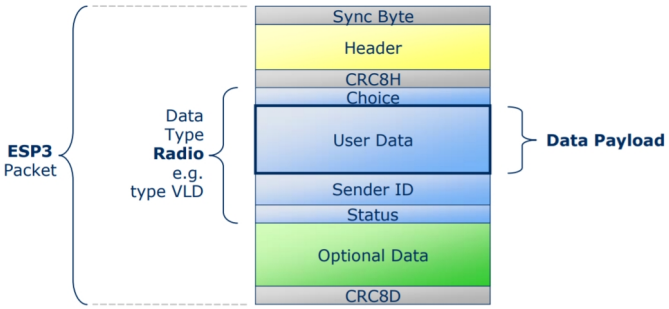


Joonis 1: EnOceani autonoomsed seadmed [9]

ESP3 täiustab eelmist, ESP2 protokolli, lisades uut funktsionaalsust:

- vastuvõetud raadiosignaali tugevuse ja „alampakettide" arvu teatamine;
- suurem läbilaskevõime tänu palju kõrgemale ülekandekiirusele;
- veakindlus, kuna pakettides kasutatakse CRC8 veakontrolli;
- valmisolek tulevaste laienduste ühilduvuseks. [9]

ERP1 (*EnOcean Radio Protocol Version 1*) algandmetega telegramm on ESP3 paketi osa, millest mõõdetud ning saadetud väärtused moodustavad andmemahust vaid teatud osa (joonis 2). [9]



Joonis 2: ESP3 paketi struktuur [9]

Kuna töös kasutatakse EnOcean liikluse jälgimiseks selleks ettenähtud lüüsi, kus saadetud paketid salvestatakse ettenähtud kohtadele lüüsi MODBUS registrites, ei peatuta käesolevas töös EnOcean protokolli detailidel.

4.3. Eluslaboratooriumi EnOceani seaded

Lõputöö käigus valmiv seade hakkab vastu võtma andmeid kuuest eri tüüpi EnOceani seadmest. Järgnevalt tutvustab autor nende seadmete iseärasusi.

4.3.1. EnOcean ruumiregulaator EEP A5-04-01

Ruumiregulaator kasutatakse Eluslaboratooriumis Thermokon SR06 LCD regulaatoreid või nende variatsioone (joonis 3). Tulevikus plaanitakse paigaldada LC-SR04 rH.

Joonis 3: Ruumikontroller Thermokon SR06 LCD

Ruumikontrolleri peamised parameetrid on toodud tabelis 1.

Mõõdetavad väärtused	Ruumi temperatuur ja õhuniiskus
Temperatuuri mõõtmisvahemik	0–40 °C. Täpsus ±0,4 K (21 °C juures)
Niiskuse mõõtmisvahemik	0–100 %. Täpsus ±5 % (vahemikus 30–70 %)
Temperatuuri LSB vahemik	0–250
Niiskuse LSB vahemik	0–250
Toiteallikas	päikesepatarei, mis laeb 3,7 V ja 60 mAh. Alternatiivid: CR1632 patarei, mikro-USB

Tabel 1. Thermokon SR06 LCD ruumikontroller [10]

Käesoleva lõputöö jaoks on olulised LSB-väärtuste ja mõõdetavate temperatuuride vahemikud, kuna lüüs hoiab oma registris antud seadme puhul huvipakkuvaid väärtusi (temperatuur, niiskus) numbrina just vahemikus 0–255. Selleks, et luua programmi, mis interpreteeriks ning pärast saadaks õige väärtuse MeiePilve, on vaja teada neid kahte numbrit ning valemit millega arvutatakse mõõdise tegelikku väärtust. Üldvalem näeb välja järgmiselt [10]:

$$Väärtus = \frac{Mõõtmisvahemik_{maks} - Mõõtmisvahemik_{min}}{LSBVahemik_{maks} - LSBVahemik_{min}} (LüüsiVäärtus - LSBVahemik_{min}) - Mõõtmisvahemik_{min}$$

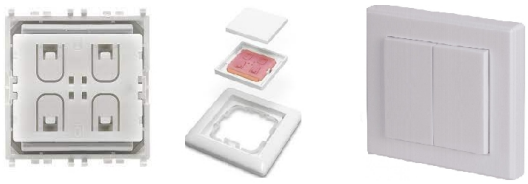
Selle ruumikontrolleri õhutemperatuuri, õhuniiskuse ning nende väärtuste minimaalsed LSB väärtused on 0. Seega lihtsustub valem veelgi. Nt kui kuueteistkümnendsüsteemist kümnendsüsteemi teisendatud lüüsi õhutemperatuuri

väärtus on 186, siis see tähendab, et ruumi õhutemperatuur on 40/250*186=29,76 °C

LSB vahemikku erinevates allikates nimetatakse erinevalt: *range, valid range, scale.* Autor hakkab kasutama edaspidi LSB vahemiku terminit. See vahemik nii nagu mõõtevahemik on määratud EnOcen seadme profiiliga (EEP). Enamasti kasutatakse LSB vahemikke 0-255 ja 0-250. Kuid see vahemik sõltub bittide arvust, mida selleks otstarbeks kasutatakse ning tarkvaraga määratud piirangutest. Nt kümnebitilise temperatuurisensori A5-02-20 LSB vahemik on 0-1023 [11].

4.3.2. EnOcean EEP F2-02-01 lüliti

Eluslaboratooriumis kasutatakse EnOcean lülititest PTM210 DB EEP F2-02-01. Lüliti saab vajaliku energia paketi edastamiseks klahvi vajutamisest. Lüliti saadab andmepaketi nii vajutamisel kui ka tagasi positsioonile tulles. Lülitiga saab ka valguse tugevust reguleerida. Kuid antud lõputöö jaoks ei ole sel omadusel tähtsust. Seade hakkab fikseerima seda, kas lüliti on sisse- või väljalülitatud. Paketi sisu järgi saab aru, mis asendis lüliti parasjagu on. Pakettide arv sõltub ka nuppude arvust (joonis 4).



Joonis 4: EnOcean PTM210 DB lüliti (ühe ja kahe klahviga) [12]

4.3.3. EnOcean EEP A5-06-01 ja EEP A5-06-02 valgustuse andurid

Valgusandur O_2LINE 10020052 EEP A5-06-01 mõõdab välisvalgustuse tugevust ja edastab andmed juhtmevabalt sobitatud vastuvõtjale niipea, kui heleduse muutus ületab 500 luksi. Kui muutusi pole toimunud, edastatakse signaali iga 100 sekundi järel. Mõõtepiirkond on 300 kuni 30000 luksi.

Sisevalgustuse anduri O_2LINE 10020053 EEP A5-06-02 mõõtepiirkond on 50 kuni 1020 luksi. Andur edastab andmeid niipea, kui heleduse muutus ületab 20 luksi, või iga 20 minuti tagant. Need kaks andurit saavad vajalikku energiat päikesepaneelist. Sisevalgustuse andur saab töötada täispimeduses neli päeva (joonis 5). [13,14]

Joonis 5: EnOcean välisvalgustuse EEP A5-06-01 (vasakul) ja sisevalgustuse EEP A5-06-02 (paremal) andurid [13,14]

4.3.4. EnOcean EEP D5-00-01 magnetkontaktandur

Akna või ukse asendi magnetkontaktandur on üks kompaktsematest anduritest, mis saab vajaliku energia päikesepaneelist. Magnetkontaktandur saadab 1-baidilise paketi, kui magnet viiakse anduri alumise osa juurde või kui magnet liigub sealt ära (joonis 6).

Joonis 6: EnOceani O_2Line LSS10020032 magnetkontaktandur [15]

4.3.5 EnOcean EEP A5-07-01 kohalolekuandur

Kohalolekuandurina kasutatakse Eluslaboratooriumis EnOcean O_2Line LSS10020051 EEP A5-07-01. Vajalikku energiat saab andur päikesepaneelist. Esmaseks täislaadimiseks kulub 200 luksi juures 25 tundi. Lisatoiteallikaks võib panna 3-5 V DC või CR2032 patarei, millega saab andur täispimeduses töötada 5 aastat ning tavatingimustes (200 luksi kaks tundi päevas) 15 aastat.

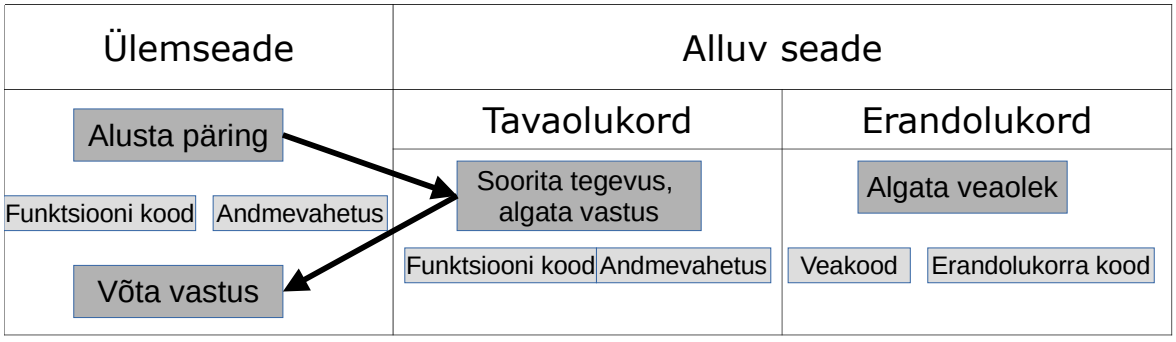
Andur tuvastab liikumist 10 meetri raadiuses. Kui liikumine on tuvastatud, saadab andur paketi ning seejärel aktiveerub anduris 2-minutiline taimer. Kui andur tuvastab pärast 2 minutit taas liikumise, saadakse pakett jälle ning taimer taaskäivitub. Kui liikumist ei tuvastata, siis saadetakse kaks paketti: üks 10 min ja teine 30 min pärast viimatist avastatud liikumist. Kui selle aja jooksul tuvastatakse liikumine, siis saadetakse kohe ka pakett, et liikumine on tuvastatud (joonis 7). [16]

Joonis 7: EnOcean O_2Line LSS10020051 kohalolekuandur [16]

4.4. EnOceani pakettide vastuvõtmine

EnOceani pakettide vastuvõtmise lüüsiks on Thermokon STC65+RS485 Modbus. Käesoleva lõputöö autor lüüsi valida ei saanud – et lüüs oli ette antud lähteülesandega. Thermokoni lüüs võtab EnOceani protokolliga andmeid vastu, kuid väljastab andmeid Modbus protokolliga.

Modbus protokoll põhineb klient-server tsentraliseeritud eelisõigusega sidesüsteemi arhitektuuril. Selle töötas välja 1979. aastal Modicon (praegu Schneider Electric). Modbus on avatud protokoll, mis on tänu oma lihtsusele tänapäeval üks enam levinumatest protokollidest maailmas. Protokolli arhitektuuri järgi on üks seade ülemseade (*master*) ning teised alluvseadmed (*slave*). Alamseadmed on igaüks oma aadressiga ja „suhtlust" alustab peaseade pöördudes alamseadme poole aadressi järgi. Ülemseade alustab sideseanssi ehk saadab käsu instruktsioonidega (funktsiooni koodis) alluvale seadmele, mis tegevused peavad tehtud saama. Alluv seade töötleb käsu ning saadab vajalikud andmed vastu. Andmevahetuse skeem tava- ja erandolukorras on toodud joonisel 8. [17,18,19]



Joonis 8: MODBUS-protokolli andmevahetus tava- ja erandolukorras [17,18,19]

Modbus protokolliga saab edastada infot kahes jadaedastusrežiimis: ASCII ja RTU (*Remote terminal unit*). Modbus ASCII jadaedastusrežiimis saadetud 8-bitine (1 bait) pakett kodeeritakse kuueteistkümnendsüsteemis, sisaldades kahte 4-bitist ASCII-tähemärki. Modbus RTU-s on andmed kodeeritud binaarsüsteemi kaudu ning ühes 8-bitilises baidis sisaldub kaks 4-bitist kuueteistkümnendsüsteemi tähemärki. Seetõttu on sama saatmiskiirusega RTU jadaedastusrežiimi informatsiooni tihedus kõrgem. Seega on Modbus RTU jadaedastusrežiim tänu eespool kirjeldatud omadustele kiirem. Lüüs Thermokon STC65+RS485 Modbus võib edastada infot mõlemas, st nii ASCII kui ka RTU jadaedastusrežiimis. [18, 19]

Tuginedes eespoolkirjeldatud eelistele, otsustas ka lõputöö autor kasutada lüüsi andmete lugemiseks just Modbus RTU jadaedastusrežiimi. Signaali leviala suurendamiseks on lüüs varustatud välise antenniga ning andmevahetuseks on 32 (Rx), 32 (Tx), 32 (VA) kanalit. Lüüs vajab 15..24 V = (±10 %) toiteallikat (joonis 9). Lüüsi MODBUS RTU andmeedastuskiirus on valitav vahemikus 9600 – 57600. Paketis saab kasutada või mitte kasutada paarsusbitti. Stopp-bitte võib olla kas 1 või 2. Režiimid on valitavad lüüsi trükiplaadil paiknevate seadelülitite abil. Tehtud seadme jaoks edastuskiiruseks valis autor 9600 kuna protsessid pole kiired ja aeglasem kiirus on töökindlam. Arvutiga lüüsi seadistamise võimalused on toodud Lisas 1. [20]

Joonis 9: Thermokon STC65+RS485 Modbus lüüs [20]

Thermokon lüüsi tootjapoolse tarkvaras on juba eelseadistatud andmetüübid ning määratud registrid, kus hakkab EnOceani andurite andmeid hoidma. Käesoleva lõputöö jaoks tähendab see seda, et valmisoleva Thermokon lüüsi tarkvaraga on võimalik ühendada EnOceani seadmed lüüsi kanaliga ning võtta vastu ja lühiajaliselt salvestada lüüsi registrites vajalikke EnOceani seadmete andmeid. Seega Autori pool loodud programm peab olema võimeline lugema ning tõlgendama lüüsi registritest vajalikke andmeid.

5. ANDMETÖÖTLUSSEADME VALIK

Lüüsi andmete vastuvõtmiseks, interpreteerimiseks ning MeiePilv andmebaasi saatmiseks on vaja vastavat seadet. Lähtuvalt ülesannetest, peaks antud seade olema kas mikrokontroller või mikroarvuti. Autor lähtub oma valikus järgnevatest eeldustest, mis peavad olema andmetöötlusseadmel:

- RS485 jadaliides andmevahetuseks lüüsidega;
- LAN liides andmete edastamiseks MeiePilv serverisse;
- Konfiguratsioonifaili salvestamine andmekandjale, mis oleks ligipääsetav ka muude vahenditega kas konfiguratsioonist reservkoopia tegemiseks või siis faili sisu uurimiseks ning vajadusel muutmiseks;
- Kuna lüüs / lüüsid puhverdavad andmevoogu, ei pea juhtseade töötama reaalajas;
- Käesoleva lõputöö eesmärgi saavutamiseks pole graafiline kasutajaliides ilmtingimata vajalik, kuid graafiline kasutajaliides suurendaks oluliselt seadme kasutusmugavust.

Toodud parameetritest lähtudes autor otsustas valida mikroarvuti kuna see, võrdluses mikrokontrolleriga, rahuldab kõige paremini nõudmisi juhtseadmele. Liidestest puudub küll RS485 liides, kuid mikroarvutitel on harilikult olemas ka USB port ja sinna saab ühendada USB–RS485 konverteri. Mikrokontrollerite ja mikroarvutite võrdlus on toodud lisas 2.

5.1. Mikroarvuti valik ja Raspberry Pi

Mikroarvutite puhul on nii mudelite kui ka brändide valik mitmekesine. Viimaste osas on pakkumine iseäranis lai, nt: Odroid, Qualcomm DragonBoard, ASUS Tinker Board, Libre Renegade Computer Board, Raspberry Pi, LattePanda, Rock PI Rockchip RK, Banana Pi jt. Brändi valikul lähtus autor järgmistest aspektidest:

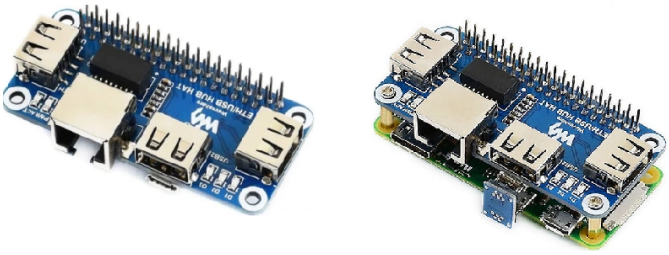
- levik;
- töökindlus;
- liideste kättesaadavus;
- hind;
- baastarkvara kasutusmugavus.

Kõigi nende parameetrite alusel oli parim valik Raspberry Pi. Lisaks on Raspberry Pi kohta hulgaliselt foorumeid ning raamatuid, kust võib probleemide esinemise korral abi saada. See asjaolu on autori jaoks tähtis, kuna õpingute käigus puutus autor kokku ainult mikrokontrolleritega (Arduino). Praeguseks on Raspberry Pi võimekas ja

levinud kompaktne mikroarvuti, mida nt 2020. aastal müüdi üle 7,4 miljoni eksemplari. [22]

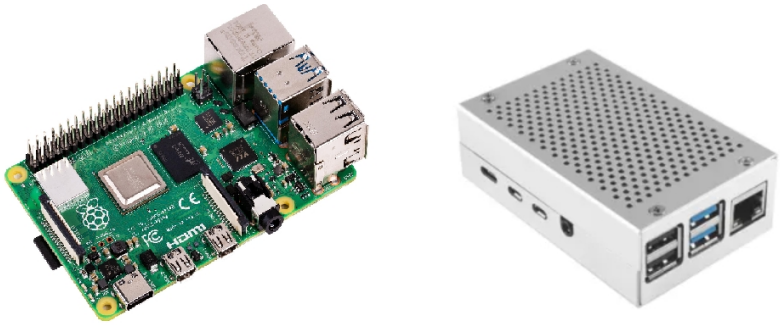
Mikroarvuti mudeli valikul lähtus autor eelmises peatükis väljatoodud eelistustest. See tähendab, et seadme mikroarvuti mudelil peab olema hiire, klaviatuuri, monitori, SD-kaardi, LAN, USB või UART–RS485 mooduli väljundid. Praeguste „tüüpiliste" mikroarvutite protsessorite puhul ei ole programmi suhtelise lihtsuse tõttu nõudmised protsessorile ning operatiivmälule väga kriitilised. Muidu seade ei tohi eraldada liigselt soojust, mistõttu ta võib üle kuumeneda. Lisas 3 on toodud kõige populaarsemad Raspberry Pi mudelid.

Lisas 3 tabelist on näha, et Raspberry Pi Zero on soodsam kui Raspberry Pi 3B+, kuid puuduvad vajalikud pordid/väljundid, nt USB port. See tähendab, et Raspberry Pi Zero plaat vajab vastavat laiendusplaati (joonis 10).



Joonis 10: Raspberry Pi Zero (vasakul) ja Raspberry Pi Zero W+ USB, LAN HUB (paremal)

Vastava laiendusplaadi hind jääb vahemikku 11–13 €, lisaks on vaja GPIO 40 pin liidest (mikro USB sild). Kokkuvõttes on hinna erinevus Raspberry Pi 3B+ ja lisatarvikutega Raspberry Pi Zero vahel suhteliselt väike. Kuna aga Raspberry Pi 3B+ on võimsam, on alust oletada, et pideva töö käigus ei hakka programm võimekamat protsessorit üle koormama ega seade liigset soojust eraldama. Lisaks on Raspberry Pi 3+ mudelil koheselt saadaval metallkorpus, mis sobis mõõtude poolest hästi ka kasutusesolevasse automaatikakilpi 25AK1 (joonis 11).



Joonis 11: Raspberry Pi 3B+ metallkorpuses (paremal) ning ilma selleta (vasakul)

5.2. Andmetöötlusseadme lõplik valik

Tuginedes eespool toodule, otsustas töö autor andmetöötlusseadmeks valida Raspberry Pi 3B+ metallkorpuses mikrokoarvuti. Raspberry Pi 3B+ hakkab lugema ning töötlema andmeid Thermokon STC65+RS485 Modbus lüüsis vastuvõetud ning salvestatud andmeid. Autor põhjendab seda valikut alljärgnevalt:

1. Mikroarvutil on olemas operatsioonisüsteem. Seetõttu on mugavam seadet ühendada võrku ja salvestada informatsiooni mikro SD kaardile. See võimaldab ka luua graafilist kasutajaliidest. Lisaks sellele on võimalik mugavalt luua kaugühendust.
2. Raspberry Pi 3B+ plaadil on vajalikud liidesed, pordid realiseeritud ühes plaadis (nt USB, LAN võimekus ilma lisaplaadita) ning olemas on ka vajalik korpus, mis sobib oma mõõtude poolest olemasolevasse automaatikakilpi.
3. Koostisosade võimsus (nt protsessor, muutmälu) on selle seadme jaoks piisavad ning isegi mõnetise varuga. See peaks hoidma mikroarvutit ülekuumenemise eest, see omakorda aga pikendama Raspberry Pi 3B+ kasutuskõlblikku tööaega.
4. Raspberry Pi 3B+ mikroarvutil on hea hinna ja kvaliteedi suhe.
5. Mugavam häälestatavus ja programmeerimine otse masinas kui talle monitor, hiir ja klahvistik külge ühendada.
6. Raspberry Pi mikroarvutite ja tarkvara kohta on olemas piisavalt infoallikaid.

6. SEADME ARENDUS

EnOcean liikluse jälgimisseade autor koostas ühest juhtseadmest ja esialgu kahest lüüsist. Lüüside arvu saab hiljem vajadusel suurendada kuna MODBUS võimaldab samale RS485 siinile ühendada kuni 247 seadet. Kahe seadmega saab kokku 32*2=64 jälgimiskanalit, mis on Eluslaboratooriumi praeguse mahu juures piisav. EnOcen raadioliikluse Eluslaboratooriumi jälgimissüsteemis infoliiklus on toodud lisas 5. Kus nooltega on tähistatud infoliiklus.

Lüüsidele Thermokon STC65+RS485 Modbus on tootja koostanud abistava tarkvara STC-RS485-Modbus, mis võimaldab siduda EnOcean seadmeid lüüsi kanalitega, mis lihtsustab oluliselt keerukat sidumise faasi. Tarkvara võimaldab teha lüüsi konfiguratsioonist varukoopia ja lüüsi väljavahetamise korral konfiguratsiooni üle kanda. Ka võimaldab see tarkvara mõne EnOcean seadme väljavahetamisel siduda lüüsi kanaliga selle uue EnOcean seadme ja süsteemi muu osa töötab tõrgeteta edasi.

Väljatöötatava EnOcean raadioliikluse jälgimisseadme juhtseadme tarkvara juurde kuulub nn konfiguratsioonifail, mis seob lüüside kanalid vastavate kanalite kaudu jälgitavate EnOcean seadmete tähistusega Eluslaboratooriumis. MeiePilv logides on seega jälgitavad EnOcean seadmed oma Eluslaboratooriumi aadressidega, mis teeb oluliselt mugavamaks salvestatud andmete kasutamise praktikas.

Konfiguratsioonifaili salvestatakse Raspberry Pi micro SD kaardile. Varukoopia sellest failist saab teha USB pulgale, mille saab ühendada mõnda selle mikroarvuti USB porti.

Jälgimisseade sai lõpuks paigutatud Eluslaboratooriumi automaatikakilpi 25AK1 ja lüüsid väikesesse eraldi automaatikakilpi 25AK2, mis viikase eemale automaatikakilbist 25AK1. Kilbis 25AK1 paikneb kaks suhteliselt võimast EnOcean saatjat. Lüüside antennide paiknemisel nende saatjate vahetus läheduses saaks tõenäoliselt häiritud kaugemate jälgitavate EnOcean seadmete raadiosignaalide vastuvõtt. Ülalkirjeldatud lahkuviimine oli preventatiivne meetod, mille tulemusena liikusid lüüside vastuvõtuantennid lähemale hoone Puiestee 80A keskkohale. (Eluslaboratooriumi automaatikaosa paikneb praegu hoones Puiestee 80A (lisa 6).

Automaatikakilbis 25AK1 on olemas nii 24 VDC toitepinge lüüside jaoks kui ka 5 VDC toide Raspberry Pi tarbeks ning LAN ühendus Tartu kolledži sisevõrku, kuhu on ühendatud ka MeiePilv serverarvuti.

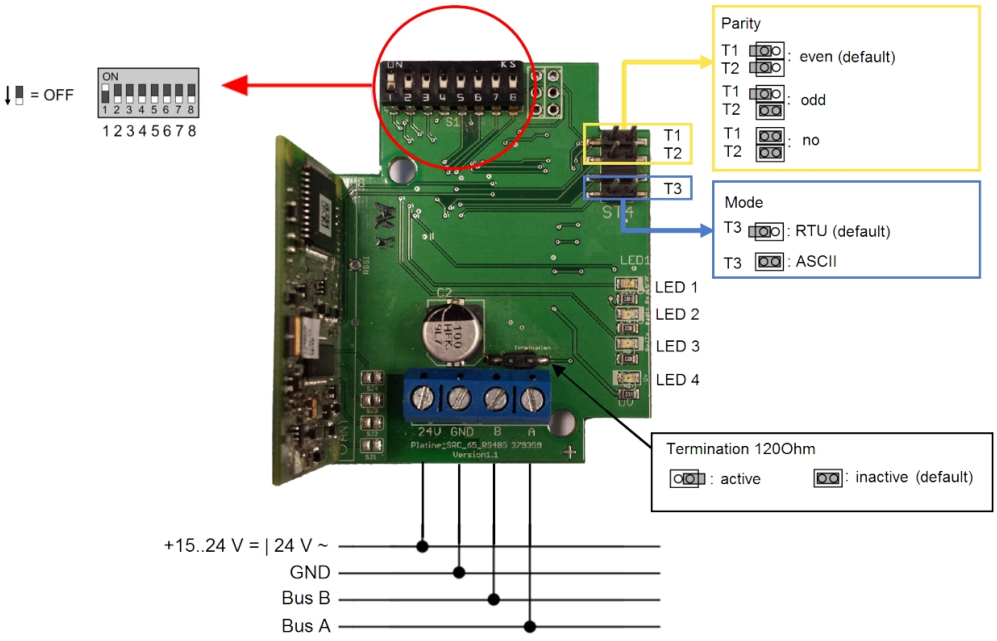
Automaatikakilpides seadme koostisosad sai kinnitatud DIN latile, milleks tuli ühele lüüsile ja Raspberry Pi korpusele lisatakse DIN lati kinnitused. (uuemal lüüsil on DIN lati kinnitus olemas.

Järgnevalt detailsemalt seadmete paigutamisest, ühendamisest ja vajaliku programmvarustuse koostamisest.

6.1. Seadme toite- ja andmesideühendused

Esialgu sai autor enda käsutusse ühe Thermokon STC65+RS485 Modbus vanema versiooni ning tegi vajalikud ühendused selle lüüsiga. (Joonis 12). Hiljem autor ühendas paralleelselt veel teist Thermokon STC65+RS485 Modbus uue lüüsi versiooni.

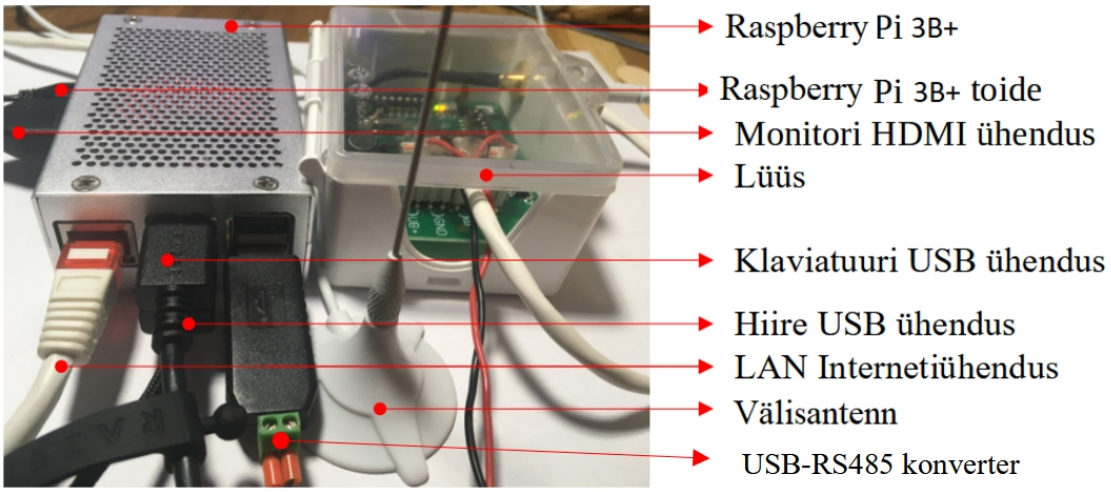
Joonis 12: Thermokon STC65+RS485 Modbus ühendused [19]



Lüüsi ühendused (joonis 12):

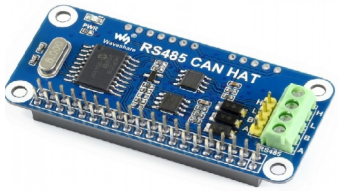
- 2 voolujuhet (UB+ ja GND), mis on ühendatud vooluadapteriga;
- 2 RS485 kommunikatsiooni ühenduse juhet (B, A), mis on ühendatud USB – RS485 konverteriga;
- Isa-ema ühendusega välisantenn. [19]

Lüüsist väljuv USB – RS485 konverter ühendatakse Raspberry Pi 3+. Autor ühendas programmi arendamise käigus lüüsi lauarvutiga, kuhu oli paigaldatud ka vajalik Thermokon tarkvara. Sealjuures tekkis töö käigus olukord, kus oli vaja Eluslaboratooriumi sülearvutil USB – RS485 konverteri draivereid uuendada. See tähendab, et antud seade koosneb sellistest moodulitest, mis teatud ulatuses on ühendatavad või asendatavad teiste seadmete moodulitega. Raspberry Pi 3B+ mikroarvutile oli tarkvara arendamise käigus ühendatud ka monitor, hiir ja klaviatuur. Raadioliikluse jälgimise süsteemi seade on toodud joonisel 13. Automaatkilpi paigaldatud seade on toodud lisas 5.



Joonis 13: Raadioliikluse jälgimise süsteemi seade

Esialgu plaanis autor lüüsi ühendada Raspberry Pi 3B+ mikroarvutiga UART–RS485 mooduli vahendusel (joonis 14), mis on mikroarvutiga ühendatav ning mahub ka korpuse sisse. Kuna aga seadme üheks omadusteks peab olema ka komponentide kerge asendatavus ning kasutusmugavus, siis jäi autor USB–RS485 konverteri juurde. USB pistikut saab vajadusel mikroarvutist kiiresti välja võtta ning ühendada teise arvuti külge näiteks selleks, et ühendada lüüs tarkvaraga, mis on mõeldud Windows operatsioonisüsteemi jaoks. Lisaks teeb see seadme mõnevõrra ka odavamaks.

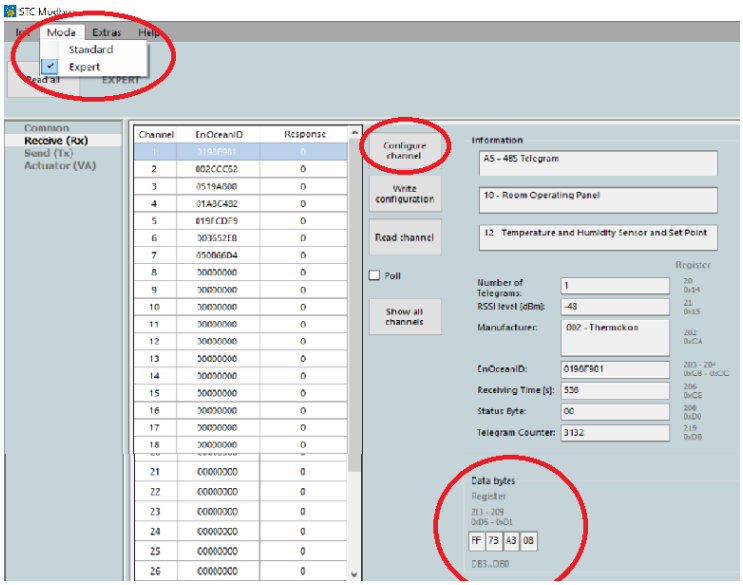


Joonis 14: UART–RS485 moodul

6.2. EnOceani seadme lüüsiga ühendamine

Selleks, et EnOceani seadmetelt vajalikku infot kätte saada, pidi seadmed ühendama esialgu Thermokon STC65+RS485 Modbus lüüsiga. Eespool mainitud lüüsil on EnOcean sidesuunal 32 kanalit, millest igaühega saab siduda ühe EnOcean saatja ehk siis meie kontekstis anduri. MODBUS suunas vastab igale kanalile mingi arv registreid, millesse salvestatakse vastav kanali kaudu laekuv EnOcean informatsioon. Selleks pidi installima vastava Thermokon tarkvara – STC-RS485-Modbus. Vajutades nuppu *Configure channel*, saab tarkvara abil erinevate viisidega ühendada vastava kanali kindla seadmega. Lülitades sisse funktsiooni *Expert View*, sai autor teada, millistesse registritesse hakkab lüüs EnOceani konkreetse anduri andmeid

kuueteistkümnendsüsteemis salvestama (joonis 15). Neid andmeid hakkab omakorda mikroarvuti lugema vastavatest registritest meile huvipakkuvatel tingimustel ehk ettemääratud olukorras ja kindla intervalliga. Pärast andmete lugemist peab mikroarvuti programm teisendama andmed õigele kujule, pakkima need õigesse formaati ja edastama MeiePilve serverisse.



Joonis 15: STC-RS485-Modbus tarkvara

Töö autori hinnangul oli Thermokoni tarkvara suhteliselt kasutajasõbralik.

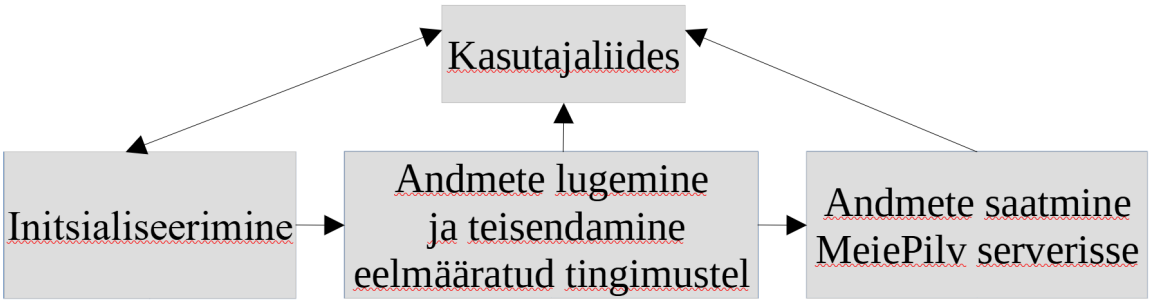
6.3. Tarkvara struktuur ja ülesanded

Nagu eespool toodud, valiti andmete vastuvõtmiseks, teisendamiseks ja edastamiseks mikroarvuti Raspberry Pi 3B+. Raspberry Pi mikroarvutil on võimalik tarkvara kirjutada erinevates keeltes, kuid eelistatuim on Pythoni programmeerimiskeel, mille kohta on olemas palju infot. Autori jaoks see oli omaette väljakutse, kuna õppetöö käigus oli omandatud teadmised C/C++ ja Java keeltes. Tänu programmeerimisvõtete suhtelisele universaalsusele kohanes aga autor kiiresti ka Pythoni süntaksiga. Esialgu kasutati koodi kirjutamiseks ja testimiseks lauarvutit ning PyCharm IDE-t. Selline otsus langetati, kuna lauarvuti on mikroarvutist kiirem ning võimsam, lisaks ei soovinud autor võimaliku vigase koodi tõttu Raspberry protsessorit üle koormata.

Enne tarkvara kirjutamist koostas autor ülesannete nimekirja, mida loodav programm peab lahendama.

1. Lugema ja teisendama lüüsi EnOceani andurite andmeid.
2. Lugema ja rakendama kasutaja poolt sisestatud CSV-failist EnOceani andurite loetud tulemuste tõlgendamiseks ja teisendamiseks vajalikke andmeid.
3. Sisaldama funktsioone, mis teisendavad ning edastavad õiges formaadis tõlgendatud andmed MeiePilve serverisse.
4. Olema võimeline seadistama Modbus protokolli füüsilise kihi (RS485) paketi parameetreid nagu andmevahetuse kiirus, paarsusbiti olemasolu ja stopp-bittide arv
5. Tarkvaral peab olema kasutajasõbralik graafiline kasutajaliides.
6. Programm peab töötlema võimalikke vigu ja tõrkeid.

Järgmisena koostas autor esmase programmi funktsionaalse struktuuri kontseptsiooni (joonis 16).



Joonis 16: Tarkvara funktsionaalse struktuuri kontseptsioon

Esialgu peab tarkvara läbi viima initsialiseerimise protseduuri. Sellel otstarbel tegi autor kaks konfiguratsioonifaili. Üks on vajalik EnOceani andurite parameetrite seadistamiseks ning teine töö seadistamiseks. Seega peab tarkvara initsialiseerimise protseduuri käigus esiteks lugema kasutaja poolt *devicelist.csv* failist EnOceani andurite kohta vajalikke andmeid:

- Esiteks, kanali numbrit ning lüüsi aadressi, millega on ühendatud antud EnOceani andur.
- Teiseks, EnOceani anduri tüüpi, mida käsitletakse neljandas peatükis. Iga EnOceani anduri tüübi parameetreid hakkab omakorda hoidma enda sees eraldi klass, mille alusel programm loob objekti, mis hakkabki hoidma kindla seadme parameetreid.
- Kolmandaks fikseeritakse, *devicelist.csv* failis registrite numbrid, kust hakatakse andmeid lugema ning mida saab lüüsi tarkvara kaudu teada.

- Neljandaks, selleks et lüüsist saadud andmeid oleks võimalik tõlgendada, hoitakse antud konfiguratsiooni failis tarkvara jaoks vajalikke andmeid mõõtmisvahemiku ning LSB skaala kohta.

Autor valis EnOceani andurite algandmete hoidmiseks just CSV-formaadi, kuna selle formaadi faile saab Exceli abil avada, täiendada ning vajalikke seadmeid lisada või eemaldada. Samas on tänu CSV-formaadi laiale levikule üldiselt saadaval ka teegid, mis saavad antud formaadi failidest ka vajalikke andmeid kätte.

Autor plaanib teha ka kasutajaliidese. Seega hakkab teine konfiguratsiooni fail hoidma programmi tööks vajalikke andmeid. Kasutajaliidese kaudu peab olema võimalik muuta ka teisi initsialiseerimiseks vajalikke andmeid, näiteks MeiePilve serveri aadressi, võrgu nime, mikroarvuti lüüsiga ühenduses oleva pordi nime ning muudetavat andmelugemise sagedust. Neid andmeid plaanib autor hoida ka eraldi konfiguratsiooni failis *defaultValues.json*. See on vajalik selleks, et uuel käivitamisel kasutaks programm vaikimisi just viimaseid andmeid. Tänu sellele failile saab kasutaja initsialiseerimiseks vajalikke parameetreid muuta ka ilma programmi käivitamiseta. Nende andmete hoidmiseks valis autor JSON-formaadi, eeskätt selle universaalsuse ja formaadi struktuuri pärast. JSON-formaadi struktuur on Pythoni sõnastiku (*dictionary*) struktuurile sarnane, samuti on saadaval teegid, mille vahendusel saab andmeid mõlemas suunas kergesti konverteerida.

Registrite väärtuste lugemiseks kindlatel tingimustel on vajalik ka eraldi funktsioon, mis saaks Modbusi lüüsist andmeid lugeda. Selleks valis autor vabavarana saadaval oleva minimalModbus teegi [25]. Pärast registriväärtuse lugemist peab abifunktsioon teisendama registriväärtuse õigele kujule vastavalt CSV-failist saadud algandmetele. Seejärel, vastavalt joonisel 16 toodud skeemile, peab teine funktsioon pärast andmete lugemist ning teisendamist jälgima vajalike parameetrite muutust. Kui näiteks kontrolleri abil mõõdetud temperatuur muutus, siis see funktsioon peab tuvastama ning saatma selle kohta õigel kujul andmed MeiePilve serverisse.

Autor kasutas Pythoni jaoks vabavarana kättesaadavat CSV-teeki. Kirjutatud ja autori poolt testitud *DataReader(filename)* funktsioon loeb CSV-failist andmeid ning väljastab andmemassiivi, kus iga loetud CSV-faili rea väärtused on nüüd esitatud võti-väärtus paaridena.

6.4. Tarkvara kirjutamine

Tarkvara kirjutamisel kirjutas autor iga ülesande lahendamise jaoks oma funktsiooni, mida sai vajadusel eraldi testida ning pärast põhiprogrammi sisse viia. Näiteks kasutas autor selleks mõeldud funktsioonis Pythoni jaoks vabavarana kättesaadavat CSV-teeki. Kirjutatud ja autori poolt testitud *DataReader(filename)* funktsioon saab sisendina

failinime, loeb CSV-failist andmeid ja väljastab Pythoni *dictionary* ehk sõnastiku, kus iga loetud CSV-faili rida on nüüd sõnastiku üksus, kus väärtused on nüüd esitatud paaridena. Selle programmi jaoks tähistab iga CSV-faili rida ühte seadet ning tema parameetreid (koodilõik 3).

Pythoni sõnastik kujutab endast korrastamata, muudetavat ning indekseeritavat kogumit. Loksulgudes andmed on esitatud võtme ja väärtuse paaridena. See on mugav seetõttu, et kui järjendi puhul saab väärtustele ligi indeksite kaudu, siis sõnastikus mingi väärtuse kättesaamise võtmeks saab olla sõne (*string*). See tõstab tarkvara kasutajasõbralikkust ning väldib võimalike vigade teket. Küll aga ei saa seal muuta veergude nimesid; samas kui kogemata veergude järjekorda muuta, siis Pythoni sõnastiku poolt võtme järgi väljastatavad tulemused ei muutu [26].

JSON-failist andmete lugemiseks ning kirjutamiseks mõeldud funktsioonis kasutas autor vabavarana kättesaadavat JSON-teeki, kus lugemisfunktsioon *dataJSONReader(filename)* võtab samamoodi vastu failinime ning väljastab Pythoni sõnastiku. JSON-faili andmete kirjutamiseks võtab funktsioon *dataJSONWriter(filename, jsonString)* omakorda failinime ja Python sõnastiku vastu ning kirjutab need andmed vastava nimega faili sisse, kustutades eelnevad andmed ära.

```
def dataReader(filename):
    try:
        devicelist = []
        csvFileReader =
csv.DictReader(open(filename))
        for row in csvFileReader:
            devicelist.append(row)
        return devicelist
    except IOError:
        print("Ei saanud csv faili
lugemiseks faili avada")
```

```
def dataJSONReader(filename):
    try:
        jsonReader = open(filename, "r")
        jsonFileContent = jsonReader.read()
        pythonDict =
json.loads(jsonFileContent)
        return pythonDict
    except ValueError as e:
        print("JSON faili sisu formaat ei vasta
nõetele: ", e)
    except IOError:
        print("Ei saanud JSON faili lugemiseks
avada")
```

```
def dataJSONWriter(filename, jsonString):
    try:
        outJsonString = jsonString
        with open(filename, 'w') as outfile:
            json.dump(outJsonString, outfile)
    except IOError:
        print("Ei saanud JSON faili kirjutamiseks avada")
```

Koodilõik 3: CSV- ja JSON-failist andmete lugemise ning faili andmete kirjutamise funktsioonid

Järgnevalt on vaja loetud andmete alusel kirjutada funktsioon, mis loob EnOceani andurite objekte. Objektid hakkavad enda sees hoidma nii eelnevalt mainitud andmeid kui ka nende töötlemiseks vajalikke meetodeid. Objektid peavad omakorda olema loodud varem defineeritud klasside alusel. Neljandas peatükis kirjeldatud EnOceani andurite andmete lugemiseks, salvestamiseks, töötlemiseks ning edastamiseks lõi töö autor kokku kuus klassi:

1. ControllerHT klass loodi ruumikontrolleri parameetrite kirjeldamiseks.
2. Switch klass ühe nupuga lüliti jaoks.
3. SwitchD klass kahe nupuga lüliti omaduste kirjeldamiseks.
4. WindowI klass magnetkontaktanduri parameetrite kirjeldamiseks
5. LightSI klass valgustatuse anduri andmete käsitlemiseks
6. oSPIR klass kohalolekuanduri andmete käsitlemiseks (Lisa 2).

Selleks et teha programm võimalikult universaalseks, tõstis autor antud ülesande raames pidevalt programmi „abstraktsuse" taset nii tervikuna kuid ka klasside tasemel. See tähendab, et autor asendas kindlad parameetrite väärtused võimalikult palju seadistatavate muutujatega, mis võimaldab seadistatavate muutujate väärtusi muuta *devicelist.csv, defaultValues.json* failide ja kasutajaliidese abil. Näiteks, asendas selle tulemusena autor programmi loomise käigus kaks klassi ühega. Varem kirjeldas üks klass sise- ning teine välisvalgustuse EnOceani anduri parameetreid. Klassi objekide seadistatavate parameetrite suurenemine laiendas võimalike EnOceani andurite valikut, mida võiks loodud seadmega ühendada. See tähendab, et mõne teise EnOceani anduri, mis polnud antud lõputöös kirjeldatud, saab ühendada loodud seadmega siis, kui selleks on olemas sobilikud klasside omadused (mis on võetud kokku tabelis 3).

	Tagastatavate väärtuste arv	Tagastatava väärtuse tüüp	Tingimus	Mõõtmisvahemik	LSB vahemik
controllerHT	2	*float*	puudub	2	2
switch	1	1/0	a>x	0	0
switchD	1	x/1/2/3	*self*, a==x	1	0
windowI	1	1/0	a==x	1	0
lightSI	2	*int, float*	puudub	2	2
oSPIR	2	1/0, *float*	a>x, puudub	2	1

Tabel 3. Seadme programmi klasside omadused (*float* – ujukomaga arv; *int* – täisarv; a – lüüsist saadud väärtus; x – seadistatav väärtus)

Esmane klasside jaotus oli tinglik. Tulevikus saab lisada kõiki EnOceani andureid, mille mõõdetavate parameetrite väärtusi saab töödelda mõni tabelis 3 mainitud klass. Näiteks controllerHT klassi objekt võib töödelda ning tagastada kaht ujukomaga väärtust. Anduri mõlema mõõdise mõõtevahemik ning LSB vahemik on *devicelist.csv* konfiguratsiooni faili kaudu seadistatav. Seadistatavate mõõtevahemikute ning LSB vahemikute arv on tabelis 3 tähistatud täisarvuga. LightSI klassi objekt omab samu

omadusi, kuid võtab vastu ning tagastab ühe täisarvu ning teise ujukomaga arvu. Switch klassi objekt saadab boolean väärtust 0 või 1. Kui lüüsist saadud väärtus ületab 0, tagastatakse 1. WindowI klassi objekt tagastab samuti 0 või 1, kuid tingimusel, et lüüsis saadud väärtus võrdub või ei võrdu konfiguratsioonifailis eelmääratud arvuga.

SwitchD klassi kasutatakse antud projektis kahe nupuga lüliti seisundite mõõteväärtuste töötlemiseks. Klass sobib ka andurite jaoks, kus on vaja fikseerida nelja seisundit. Kui lüüsist saadud väärtus võrdub *devicelist.csv* konfiguratsiooni failis mõne antud väärtusega, siis tagastatakse 1, 2 või 3. Kui lüüsis saadud arv ei võrdu mitte ühegi eespool mainitud väärtusega, siis tagastatakse konfiguratsiooni failis vastava lahtri esimeseks numbriks nt 0. oSPIR klassi objekt töötleb kahte väärtust. Üks väärtus on *boolean* ning teine ujukomaga arv. *Boolean* väärtuse puhul saadakse 1 või 0 tingimusel, kui lüüsist saadud väärtus on konfiguratsiooni faili väärtusest suurem või mitte. Teine väärtus, mida oSPIR klassi objekt tagastab, on ujukomaga number (tabel 3).

Tabelis 4 on toodud *devicelist.csv* konfiguratsioonifailis eri klasside objektide seadistatavate parameetrite näidisväärtused. seadme esmase testimise käigus lisati seadistatavaid parameetreid veergudena pidevalt juurde koos vajalike muudatustega tarkvara programmis. Kusjuures tänu Pythoni sõnastiku kasutamisele saab veergude järjekorda vajadusel ka muuta. Lisaks sellele ja eespool mainitud teegi kasutamisele pole ka ridade ning seega seadmete arv selle konfiguratsiooni failiga piiratud.

channel	SLadress	devicetype	MPname	registers	range	scale
1	2	controllerHT	Ruumikontroller 01	210 211	0 40 0 100	0 250 0 250
2	1	switch	Lyliti 01	229	0	0
3	1	windowI	Magnetkontakt 01	249	8	0
4	1	lightSI	Valgus 01	270 272	0 1020 0 5	0 255 0 250
5	1	lightSI	23MK4	291 293	0 30000 0 5	0 255 0 250
6	1	switchD	23MK3	309	0 16 80 21	0
7	1	oSPIR	PIR 01	330 332	127 0 5	0 250

Tabel 4. *devicelist.csv* konfiguratsiooni fail

Tabeli 4 esimene rida kirjeldab tingimusi, kuidas peab käsitlema ühest EnOceani seadmest saadud andmeid. *Channel* 1 tähendab, et seade on ühendatud lüüsi esimese kanaliga. See on vajalik selleks, et kui tuleb veateade, siis selle parameetri järgi saab kiiresti aru, mis seadmega on tegu. SLadress tähistab lüüsi aadressi, millega on konkreetne EnOceani seade ühendatud. Tänu selle parameetri lisamisele tõusis võimalike ühendatavate seadmete hulk märgatavalt. Ühes STC65+RS485 Modbus lüüsis on 32 kanalit ehk nii palju seadmeid saab ühe lüüsiga ühendada. Vabu lüüsi

aadresse on 1-247, kuna 0 ei saa kindla lüüsi aadressiks olla. Seega teoreetiliselt saab loodud seadmega ühendada maksimaalselt 32*247=7904 EnOceani seadet (kui kasutada STC65+RS485 Modbus lüüse). Praktiliselt autor katsetas seda võimalust kahe lüüsiga.

Devicetype controllerHT tähendab, et selle EnOceani seadme omadusi hakkab kirjeldama objekt, mis on loodud programmis kirjeldatud controllerHT klassi alusel. Osa nendest omadustest on kirjeldatud järgnevate seadistatavate parameetritega: Mpname, registers, range, scale. MPname lahtrisse tuleb panna soovitud seadme nimi, mis peab kokku langema nimetusega, mis sellele seadmele MeiePilves antud on. Tabelis 4 on numbrilised *registers, range, scale* väärtused eraldatud tühikutega. Järjekord on oluline ning iga klassi puhul on nende parameetrite numbriliste väärtuste hulk erinev. Järjekorda ning numbriliste parameetrite arvu muuta ei saa. Seadistuse käigus saab muuta ainult nende parameetrite väärtusi. Registers 210 211 tähendab, et antud klassi objekt loeb andmeid lüüsi Modbus registritest nr. 210 ja 211. Antud seadme puhul loetakse nendest registritest temperatuuri ning suhtelise õhuniiskuse väärtusi. Selleks, et neid andmeid oleks võimalik teisendada õigele kujule, on vaja teada EnOcean seadme vastava parameetri mõõtevahemikku (mõõtepiirkonda) ja LSB muutumispiirkonda. *Range* 0 40 0 100 tähendab, et esimesest registrist 210 saadud väärtuse mõõtevahemik on 0 kuni 40 ning teisest registrist 211 saadud mõõtevahemik on 0 kuni 100. *Scale* 0 250 0 250 omakorda tähendab, et esimesest registrist 210 saadud väärtuse LSB vahemik on 0 kuni 250 ning teisest registrist 211 saadud LSB vahemik on ka 0 kuni 250.

Tabelis 4 on välja toodud ka see, kuidas on võimalik lisada EnOceani seadet, mis mõõdab sama parameetrit, kuid on erineva otstarbega. Näiteks kanaliga 4 on ühendatud sisevalgustust mõõtev EnOceani andur oma mõõtevahemikega, aga kanaliga 5 on ühendatud välisvalgustust mõõtev EnOceani andur oma mõõtevahemikega.

Programmi tööd konfigureerivas failis defaultValues.json salvestatakse andmed, mis määravad programmi tööd:

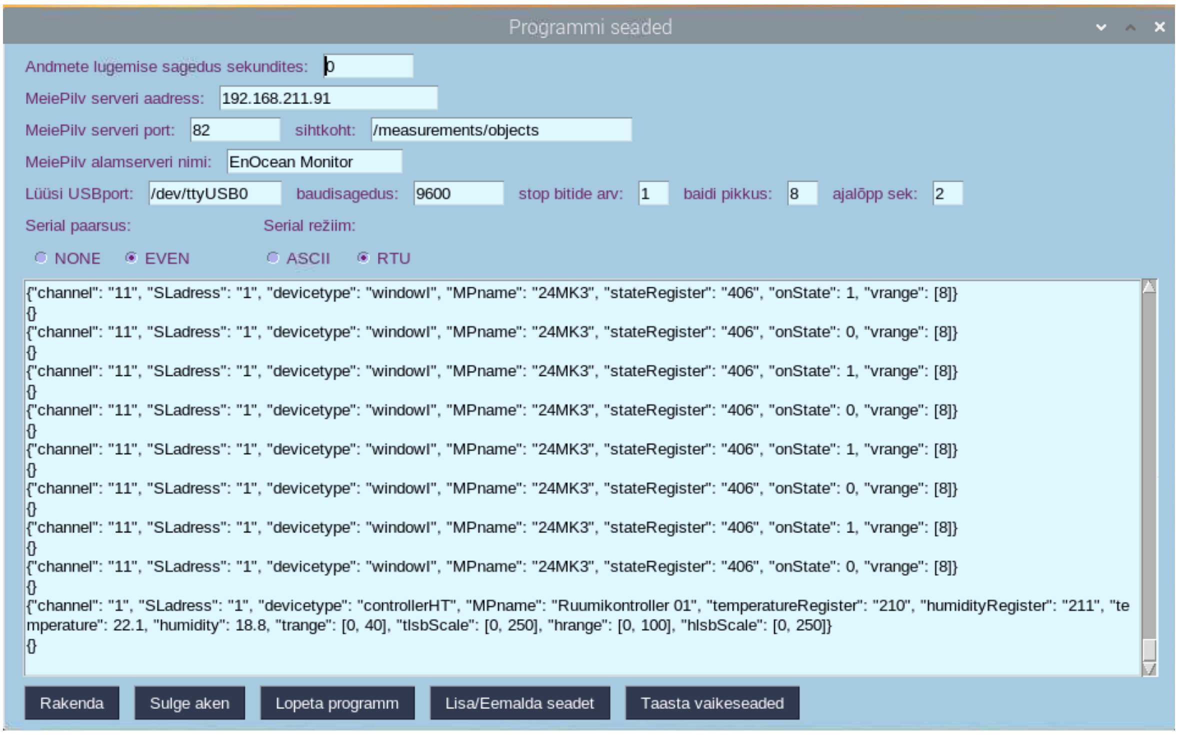
- *pause*: kui tihti mõõdetakse EnOceani andurite parameetreid
- *serverPort*: MeiePilv serveri port
- *subserverName*: MeiePilve serveri võrgu nimi, kuhu salvestatakse EnOceani andurite mõõdised
- *serverAdress*: MeiePilve serveri IP aadress
- *MSdestination*: paketi sihtkoht (MeiePilv võrk nt EnOcean Monitor)

- *baudRate*: Lüüsi/de RS-485 pordi edastuskiirus
- *stopBits*: stopp bittide arv lüüsi andmepaketis
- *byteSize*: RS485 kaadri andmevälja pikkus bittides
- *timeOut*: lüüsi vastuse maksimaalne ooteaeg sekundites
- *serialParity*: paarsuskontroll
- *serialMode*: RTU / ASCII

Konfiguratsioonifail defaultValues.json salvestab viimase kasutuskorra käigus tehtud muudatusi tehtud muudatusi. Muudatusi konfiguratsioonifailis saab teha kas otse failis või kasutajaliidese kaudu.

Pärast EnOcean seadmete iseloomustavate objektide loomist programmis defineeritud klasside alusel loetakse ning teisendakse eelmääratud registritest vajalikud andmed *ConnectionWithModbus* funktsiooni abil ning käivitatakse lõpmatu tsükkel, mis *trackChanges* funktsiooni abil jälgib, kas loetavates andmetes toimusid muutused või mitte. Kui muutus toimus ehk kui mõõdetud väärtus erineb eelmisest väärtusest, siis funktsioonide *objectMeiePilvmessage* ja *meiePilvmessage* abil teisendatakse EnOcean seadme andmeid õigele kujule ning funktsiooni *httpMeiPilvMessage* abil saadetakse MeiePilve serverisse. EnOcean seadmete iseloomustavate objekti antud parameetri väärtust muudetakse samuti - vana väärtust asendatakse uuega (Lisa 2).

Muudatused konfiguratsioonifailis defaultValues.json saab teha ka kasutajaliidese kaudu. Kasutajaliidese eest vastutav funktsioon käivitub pärast initsialiseerimisprotseduure paralleelprotsessina. Pärast vajutamist nupule „Rakenda" salvestatakse tehtud muudatused defaultValues.json konfiguratsioonifaili ning programm taaskäivitub. Vajutades nupule „Sulge aken", kasutajaliidese aken sulgub, kuid programm jätkab tööd. Programmi saab lõpetada käsurea kaudu või taaskäivitades Raspberry mikroarvuti. Kui vajutada nupule „Lõpeta programm", siis programm lõpeta töö. Vajutades nuppule „Lisa/Eemalda seadet" avaneb devicelist.csv konfiguratsiooni fail mille abil saab EnOcean raadioliikluse jälgimisseadmega ühendada/lahtiühendada EnOcean anduri. Kasutajaliideses ilmub vaikimisi info, mida viimati salvestati. Kui muudatusi pole vaja sisse viia, siis pole midagi vaja teha. Programm jätkab tööd nende andmete alusel, mis olid viimati salvestatud (joonis 17).



Joonis 17: Kasutajaliides

Kasutajaliidese loomisel kasutati vabavarana kättesaadavat PySimpleGUI teeki. Kasutajaliidisel on aken, kuhu ilmuvad erinevad teated. Näiteks kui toimus muutus ning vastav pakett saadetakse MeiePilve serverisse, väljastatakse ekraanile json-formaadis kõik selle objekti andmed. Siit saab kontrollida, kas kõik andmed said konfiguratsioonifaili õigesti pandud. Kui andmed jõudsid MeiePilv serverisse, ilmuvad kasutajaliidese ekraanile tühjad looksulud „{}”. Kui näiteks tegime vea ja unustasime sisestada MeiePilv võrku jälgitavat EnOcen seadet, siis näeme veateade: „{„Error”: „Partly correct”, „ErrorText: Some devices not found or not active”}”. See tähendab, et programm saatis omalt poolt õiges formaadis andmeid välja kuid MeiePilv server eespool mainitud põhjusel ei saa andmed vastu võtta.

6.5. Tarkvara testimine ning täiendamine

Esialgu testis autor seadet kodutingimustes ning pärast ka Tartu kolledži sidelaboratooriumis. Valmis seadet autor paigutas sidelabori A205 automaatikakilpi ja testis EnOceani seadmetega. Andmepaketid saadeti MeiePilve serveri EnOcean Monitor võrku.

Enne testimist tuli Raspberry operatsioon ja paketti kuuluv tarkvara uuendada ning installida programmi tööks vajalikud teegid: minimalmodbus, csv, sys, os, json, time, threading, PySimpleGUI, re, http.client. Lisaks kirjutas autor programmi käivitamise

lihtsustamiseks selle jaoks skripti ning tegi töölauale ikooni, mille kahekordsel vajutamisel käivitub loodud programm (koodilõik 4).

```
#!/usr/bin/env
[Desktop Entry]
Name=MyApp
Comment=Starts an AS400 session
Exec=python3 fprog1.py
Terminal=true
Type=Application
Icon=/home/pi/Desktop/Supportive files/icon.jpeg
```

koodilõik 4: Programmi käivitamise skript

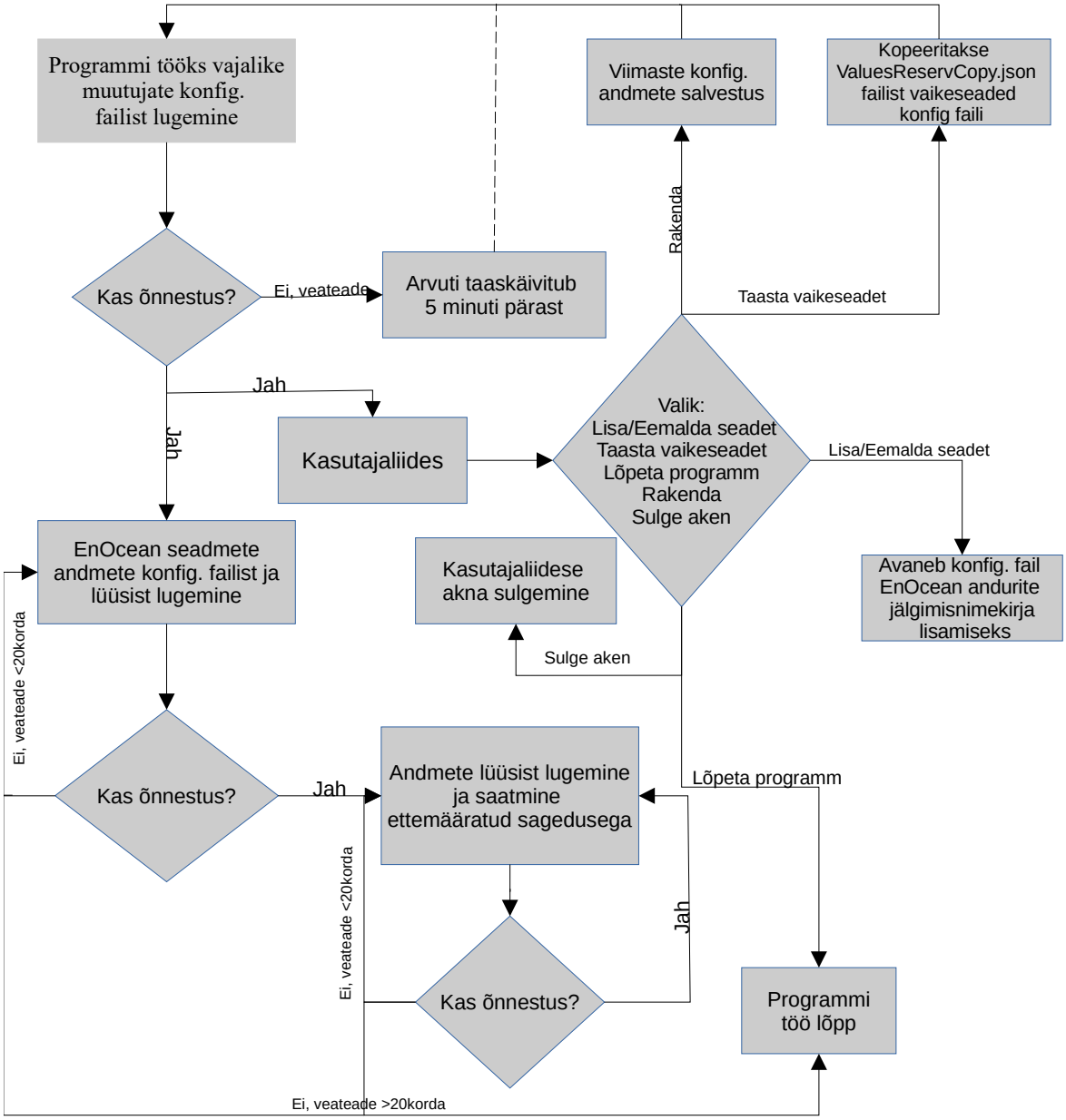
Testimise käigus autor tuvastas, et vigade käsitlus programmi poolt oli puudulik. Seda oli parandatud, lisades Pythoni vigade „püüdmiseks" mõeldud käsud: *try, except, else.* See on vajalik selleks, et kui kuskil tekib viga, siis programm ei „läheks katki" ehk ei lõpetaks tööd. Näiteks võib juhtuda, et lüüsist ei õnnestu esimese korraga saada vajalikke andmeid ning selleks, et programm jätkaks tööd, peab eespool mainitud käske kasutades kirjeldama, mida peab vea ilmnemisel tegema.

Lisaks autor optimeeris Raspberry seadistamiseks vajalike seadmete (hiir, monitor, klaviatuur) hulka. Seda võimaldas VNC Viewer vabavara kasutuselevõtt. Tänu sellele sai Raspberry LAN-ühenduse kaudu ligi ka teisest arvutist.

Kodutingimustes autor testis seadme kõikide tabelis 4 toodud EnOcean seadmetega kuna sellel ajal MeiePilv server oli veel väljaspool ülikooli kättesaadav. Sellel ajal said kõik peamised vead kõrvaldatud. Järgnevalt seade oli testitud Eluslaboratooriumis millega oli ühendatud ühe korruse magnetkontaktandurid. Selle testi tulemusena sai kindlaks tehtud, et ruumi-, võrguvahetus ning EnOcean seadmete lisamine ei mõjutanud seadme tööd.

Eluslaboratooriumi katsetamse tulemused on toodud lisas 6. Graafikutel on näha Sidelabori akende magnetkontaktandurite ja ruumikontrolleri mõõtmistulemusi. Testimise jaoks aknaid oli avatud mõni päev sagedamini. Testitulemused näitavad, mis analüüsimise võimalusi annavad andmed, mis oli saadud lõputöö käigus loodud seadme poolt.

6.6. Koodi lõplik struktuur



Joonis 18: Programmi lõplik koodi struktuur

Joonisel 18 on toodud programmi koodi struktuur, millest on näha, et programmi eri etappidel on arvestatud võimalike vigadega. Kui programmi käivitamisel ei õnnestu lugeda konfiguratsioonifailist programmi toimimiseks vajalikke andmeid, väljastatakse kasutajale vastav veateade koos teavitusega, et arvuti taaskäivitub viie minuti pärast. Kui kasutaja ei vajuta selle aja jooksul nupule „Lõpeta programm", arvuti taaskäivitub. Kui eespoolnimetatud andmete lugemine õnnestus, siis loetakse konfiguratsioonifailist EnOceani seadmete andmeid ning tekitatakse vastavad objektid. Paralleelselt ilmub ekraanile ka kasutajaliides. Kui EnOceani andurid fikseerivad mõõdetavates parameetrites muudatuse, siis objekti andmeid uuendatakse ning info saadetakse

MeiePilv serverisse. Muudatused saab programmi sisse viia kasutajaliidese kaudu. Vajutades nupule „Rakenda”, käivitub programm uute seadistustega. Vajutades nupule „Sulge aken”, jätkab programm tööd, kuid kasutajaliides suletakse. Kui vajutada „Lõpeta programm” nupule, siis programm lõpetab töö, salvestades enne viimased muudatused.

7. VÕIMALIKUD ARENGUSUUNAD

EnOceani seadmete profiilide automatiseeritud tuvastuse võimekuse eest vastutab lüüs. Kuna EnOceani seadmete profiilide hulk on suur ning täieneb/uueneb pidevalt, siis autori arvates on otstarbekas jätta see ülesanne lüüsi tootja tarkvara arendajatele. Samas lihtsustaks seadme kasutust see, kui peaks tekkima Thermokon tarkvara Linux versioon selleks, et EnOceani seadmed saaks lüüsiga ühendada/lahti ühendada midagi ringi tõstmata. Esiteks, selleks on võimalik ka luua veebilahenduse.

Järgnevalt on vaja nii paigutada EnOcean seadmeid teistesse Eluslaboratooriumi ruumidesse, kui ühendada need loodud EnOcean radioliikluse jälgimisseadmega. Lisaks, uurida EnOcean radioliikluse jälgimisseadme teisi rakendusvõimalusi näiteks: teiste süsteemide valideerimisvõimalusi, integreerimise võimalus andmetöötlustarkvaraga.

Tulevikus võib mõelda ka sellele, et luua algoritme mis analüüsiks kogutud EnOcean raadioliikluse jälgimisseadmega andmeid. Programm võiks pakkuda nii ettemääratud agregeerituid andmeid ning otsida võimalikke seoseid kogutud mõõdiste vahel.

KOKKUVÕTE

Eluslaboratooriumis kasutatakse hulgaliselt EnOceani andureid ja seadmeid. Nende seadmete parameetrid on vajalikud automaatika toimimise, juhtimise efektiivsuse jm analüüside koostamiseks. Selle lõputöö eesmärgiks oli lahendada antud ülesannet ehk luua seade, mis edastaks EnOceani seadmete poolt kogutud andmeid MeiePilve andmebaasi.

Selline Eluslaboratooriumi muudest osadest sõltumatu infokanal võimaldab kontrollida süsteemi tööd ja kanda MeiePilv andmebaasi ka andmeid, mida Elusalboratooriumi automaatika oma töös ei kasuta. Vastavalt lähteülesandele tuli kasutada EnOcean liikluse vastuvõtjana lüüsis Thermokon STC65+RS485 Modbus.

Lõputöö käigus koostati seade, mis koosneb ülalnimetatud lüüsidest ja RasBerry Pi 3B-l lahendatud juhtseadmest. Saadud jälgimisseadme juhtseade on paigaldatud Eluslaboratooriumi automaatikakilpi 25AK1 ja kaks EnOcean - Modbus RTU lüüsi automaatikakilpi 25AK2. Kirjutatud on andmeedastust ja formaaditeisendusi teostav tarkvara ja kasutajaliides. Seade on laiendatav kuni 247 lüüsini, millest igaühel 32 kanalit. Füüsiline ühenduspiir on seega 7904 EnOcean seadet. See aga ületab peaaegu kahe suurusjärgu võrra EnOcean seadmete võimaliku koguarvu, mis saab töötada ühes levialas.

Jälgimisseade on katsetatud esmalt eraldi ja siis Eluslaboratooriumi taristu koosseisus Katsetamsie käigus seadmeppolseid tõrkeid ei esinenud ja MeiePilve laekunud informatsioon on korrektne. Seega võib lugeda katsetamise tegelikes tingimustes kordaläinuks.

Töö lõpus on toodud mõned soovitused kuidas valminud süsteemi võiks edasi arendada. Loen oma lõputöö ülesande edukalt lahendatuks.

Autor on antud töö tulemustega rahul, sest kõik seatud eesmärgid said täidetud ning kohati isegi ületatud. Õpingute käigus saadud teadmised ja oskused, näiteks nagu programmeerimisloogika, aitasid palju kiiremini jõuda soovitud eesmärgini isegi tundmatute seadmete ning tarkvaraga. Autor näeb ka enda tehtud EnOceani raadioliikluse jälgimisseadme seadmes mitut võimalust, kuidas saaks seda seadet edasi arendada, muutes seda veelgi kasutajasõbralikumaks ning universaalsemaks.

SUMMARY

The living laboratory uses a wide range of EnOcean sensors and equipment. The parameters of these devices are necessary for the preparation of analyzes of automation operation, control efficiency. The aim of this thesis was to a device that would transfer the data collected by EnOcean devices to the MeiePilv database.

The task of this thesis was to make a device that monitors the EnOcean data traffic of the Tartu College Living Laboratory and stores the content of the captured data packets in the Life Laboratory database MeiePilv. This information channel, will be independent of other parts of the Living Laboratory. It will allow to check the operation of the system and enter data into the MeiePilv database that the automation systems of the Living Laboratory does not use in their work.

According to the initial task, Thermokon STC65+RS485 Modbus had to be used as an EnOcean traffic receiver component. Device consisting of the two above-mentioned gateways and a control device- RasBerry Pi 3B+ what was mounted to the automation shield 25AK1 of the Living Laboratory. Software and user interface for data transfer and format conversion was written in Python language. The device is expandable by 247 gateways, each with 32 channels. The physical connection limit is thus 7904 EnOcean devices.

There were no device unsolvable errors during the testing and the information received by MeiePilv database was correct. Thus, the testing can be considered as successful. The author is satisfied with the results of this work, because all the set goals were met and in some places even exceeded.

The knowledge and skills acquired during the studies, such as programming logic, helped to achieve the desired goal much faster, even with unknown equipment and software. The author also sees several ways in which the EnOcean radio traffic monitoring device he has made could be further developed, making it even more user-friendly and be integrateid in another systems.

KASUTATUD KIRJANDUSE LOETELU

[1] Taltech kodulehekülg [Võrgumaterjal]

https://taltech.ee/telemaatika-ja-arukad-susteemid

[2] G.Randla. ”EnOcean raadioliikluse jälgimise võimekuse loomine Tartu kolledži Eluslaboratooriumile”, 2020, Tartu. [Võrgumaterjal]

https://digikogu.taltech.ee/en/Download/3f7000cb-97a3-474b-8205-e0c2c4de78c2

[3] C. Győrödi, R. Győrödi, G. Pecherle and A. Olah, "A comparative study: MongoDB vs. MySQL," 2015 13th International Conference on Engineering of Modern Electric Systems (EMES), 2015, pp. 1-6, doi: 10.1109/EMES.2015.7158433.

[4] N. Joukov *et al*., "RAIF: Redundant Array of Independent Filesystems," 24th IEEE Conference on Mass Storage Systems and Technologies (MSST 2007), 2007, pp. 199-214, doi: 10.1109/MSST.2007.4367974.

[5] Y. Zhang, N. Ansari, M. Wu and H. Yu, "On Wide Area Network Optimization," in IEEE Communications Surveys & Tutorials, vol. 14, no. 4, pp. 1090-1113, Fourth Quarter 2012, doi: 10.1109/SURV.2011.092311.00071.

[6] M St. Louis, G. Martin.”Building Automation Opportunities to Meet NYC Emission Laws in Existing Buildings”, 2020 EnOcean Alliance Inc [Võrgumaterjal]

https://www.caba.org/wp-content/uploads/2021/06/IS-2021-127.pdf

[7] A Anders. „EnOcean Technology – Energy Harvesting Wireless”, 2011, White Paper [Võrgumaterjal]

https://www.enocean.com/fileadmin/redaktion/pdf/white_paper/
WP_EnOcean_Technology_en_Jul11.pdf

[8] The Ecosystem of the EnOcean Alliance, 2015 EnOcean Alliance Inc [Võrgumaterjal]

https://www.enocean-alliance.org/wp-content/uploads/2016/11/
Whitepaper__Getting_started_with_EnOcean_Alliance.pdf

[9] EnOcean Serial Protocol 3 (ESP3) V1.51. EnOcean, 2020 [Võrgumaterjal]

https://www.enocean.com/esp

[10] Datasheet SR06 LCD from version 2.5. Thermokon Sensortechnik GmbH., 2020 [Võrgumaterjal]

https://www.smartbuildingproducts.co.uk/wp-content/uploads/2021/01/SR06_LCD_EasySens_Datasheet_en.pdf

[11] EnOcean Equipment Profiles (EEP) [Võrgumaterjal]

https://www.enocean-alliance.org/wp-content/uploads/2017/05 EnOcean_Equipment_Profiles_EEP_v2.6.7_public.pdf

[12] EnOcean PTM210 DB lüliti manuaal [Võrgumaterjal]

https://www.enocean-alliance.org/product/ptm-215/

[13] EnOcean O2Line 10020052 välisvalgustuse anduri manuaal [Võrgumaterjal]

https://manualzz.com/doc/11011290/enocean-trio2sys-outdoor-light-sensor-manual

[14] EnOcean O2Line 10020052/1020053 sisevalgustuse anduri manuaal [Võrgumaterjal]

https://download.schneider-electric.com/files p_enDocType=User+guide&p_File_Name=II-EnOcean-Indoor-Light.pdf&p_Doc_Ref=II-EnOcean-Indoor-Light

[15] EnOceani O2Line LSS10020032 magnetkontaktanduri manuaal [Võrgumaterjal]

https://www.se.com/uk/en/product/LSS10020032/ecostruxure-building-expert-enocean-room-occupancy-sensor/

[16] EnOcean O2Line LSS10020051 kohalolekuanduri manuaal [Võrgumaterjal]

https://www.se.com/nz/en/product/LSS10020051/ecostruxure-building-expert-enocean-pir-occupancy-sensor/

[17] MODBUS-RTU Applied to the XR10CX Control manuaal [Võrgumaterjal]

https://www.ccontrols.com/support/dp/PV500-67.pdf

[18] Modicon Modbus Protocol Reference Guide.PI–MBUS–300 Rev. J 1996 MODICON, Inc., Industrial Automation Systems One High Street North Andover, Massachusetts 01845 [Võrgumaterjal]

https://modbus.org/docs/PI_MBUS_300.pdf

[19] SIDELIIDESED JA -PROTOKOLLID. Taltech. Madis Lehtla. Sissejuhatus digitaaltehnikasse Praktikumijuhend

http://www.tud.ttu.ee/im/Madis.Lehtla/WEB/

[20] Thermokon STC65+RS485 Modbus manuaal [Võrgumaterjal]

https://www.thermokon.de/direct/alterra-base/mimes/get/e728605078af15fa2e728605078af15fa2

[21] Raspberry Pi kodulehekülg [Võrgumaterjal]

https://www.raspberrypi.org/

[22] Arduino kodulehekülg [Võrgumaterjal]

https://www.arduino.cc/

[23] M. Batz. Raspberry Pi single board computer market share of total PC market in 2020. pi3g.com, 2021 [Võrgumaterjal]

https://pi3g.com/2021/05/19/raspberry-pi-single-board-computer-market-share-of-total-pc-market-in-2020/

[24] Ehrmann, G., & Ehrmann, A. (2020). Suitability of common single circuit boards for sensing and actuating in smart textiles. Communications in Development and Assembling of Textile Products, 1(2), 170-179. https://doi.org/10.25367/cdatp.2020.1.p170-179

[25] MinimalModbus teegi kirjeldus [Võrgumaterjal]

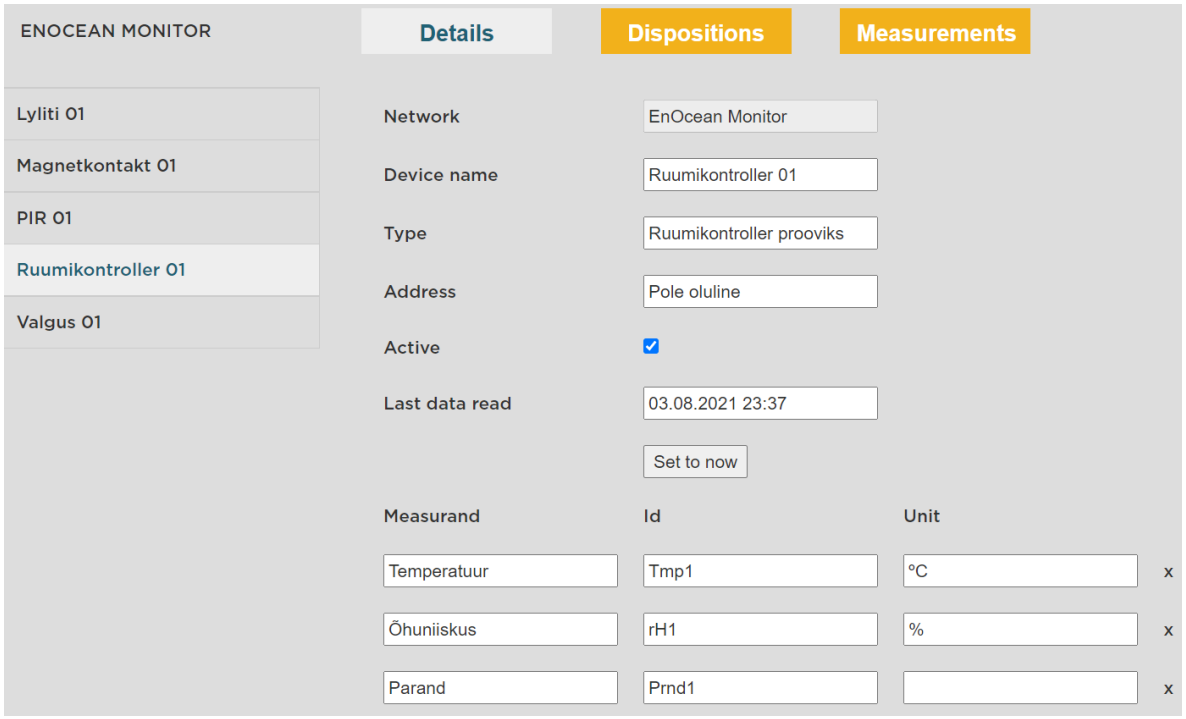
https://minimalmodbus.readthedocs.io/en/stable/

[26] Programmeerimise õpik. Andmestruktuurid [Võrgumaterjal]

https://progeopik.cs.ut.ee/10_andmestruktuurid.html

LISAD

Lisa 1. MeiePilve serverisse seadme lisamine



Lisa 2. Mikrokontrolleri ja mikroarvuti võrdlus [20][21]

Näitaja	**Mikrokontroller**	**Mikroarvuti**
Tootlus	1 tuum, kümned-sajad MHz, kümned kilobaidid operatiivmälu, kümned-sajad kilobaidid püsimälu	1 või rohkem tuuma, sajad-tuhanded MHz, kümned megabaidid operatiivmälu, gigabaidid püsimälu
Multitegumtöötlus	Üldiselt ei, kuid saab emuleerida	Jah, juhitakse operatsioonisüsteemiga
Internetiga ühilduvus	Tavaliselt on vaja lisamooduleid ja teeke; võimsamatel mikrokontrolleritel on see võimekus ka olemas	Üldjuhul on see võimekus kohe olemas
Autonoomsus	Energiatarbe on kuni kümneid mA; saab töötada patareidest päevi	Energiatarbe on sajad-tuhanded mA, seega vajab püsitoidet
Reageerimiskiirus	100 % ajakontroll ja signaali ülekande üle	Multitegumtöötluse pärast võib mõni protsess hiljaks jääda
Keeltevalik	Tavaliselt C/C++	Python, JavaScript, Bash jt
Võimekus töötada videoga	Tavaliselt ei saa	Jah, OpenCV, HDMI väljund
Võimekus töötada heliga	Võimekatel mikrokontrolleritel saab, kuid on vaja lisamoodulit	Tavaliselt operatsioonisüsteem toetab MP3, OGG, WAV ning on olemas audiväljund ja/või HDMI

Lisa 3. Raspberry Pi mikroarvuti mudelite võrdlus [23,24]

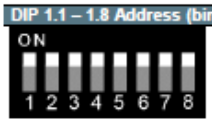
Plaat	**Raspberry Pi 4B**	**Raspberry Pi 3B+**	**Raspberry Pi Zero W**	**Raspberry Pi Zero**
Ligikaudne hind	38-78 €	37 €	11 €	6 €
Kiibisüsteem	BCM2711	BCM2837B0	BCM2835	BCM2835
Tuumad	4	4	1	1
Tuuma tüüp	Cortex-A72	Cortex-A53	ARM1176JZF-S	ARM1176JZF-S
Juhtimis-seadme taktsagedus	1.5 GHz	1.4 GHz	1 GHz	1 GHz
Muutmälu	1-8 GB LPDDR4	1 GB DDR2	512 MB	512 MB
USB 3.0	2	ei ole	ei ole	ei ole
USB 2.0	2	4	ei ole	ei ole
micro OTG	ei ole	ei ole	1	1
Video	2x micro HDMI	HDMI	mini HDMI	mini HDMI
Analoog audio väljund	3.5 mm	3.5 mm	ei ole	ei ole
SPI liides	jah	jah	jah	jah
I²C liides	jah	jah	jah	jah
GPIO liides	40 pini	40 pini	40 pini	40 pini
Mälu	microSD	microSD	microSD	microSD
WiFi	802.11 b/g/n/ac	802.11 b/g/n/ac	802.11n	ei ole
Bluetooth	5	4.2, BLE	4.1	ei ole
LAN	jah	jah	ei ole	ei ole
Suurus	85.6 x 56.5 x 11 mm3	85.6 x 56.5 x 17 mm3	65 x 30 x 5 mm3	65 x 30 x 5 mm3
Kaal	46 g	45 g	9 g	9 g
Toide	1.25 A @ 5 V	1.13 A @ 5 V	180 mA @ 5 V	160 mA @ 5 V
Toitepistik	USB-C	Micro USB or GPIO	microUSB or GPIO	microUSB or GPIO

Lisa 4. Thermokon STC65+RS485 Modbus seadistused

»CONNECTION CONFIGURATION - STC65+ RS485-MODBUS

Factory default: RTU, Address 1, 9600Bd, Parity even, Version 32Rx/8Tx

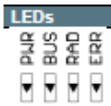
DIP 1.1 – 1.8 Address (binary coded)



The address of the device is set binary coded in the range 1...247 via an 8-fold dip switch. The address 0 is reserved for a broadcast and is initiated by the master.

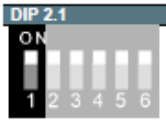
Dip switch	1 = on	2 = on	3 = on	4 = on	5 = on	6 = on	7 = on	8 = on
Value	2^0 (1)	2^1 (2)	2^2 (4)	2^3 (8)	2^4 (16)	2^5 (32)	2^6 (64)	2^7 (128)

LEDs



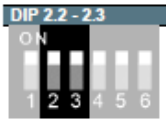
PWR	Power supply OK LED permanently ON → Version 32Rx / 8Tx active LED flashes → 32Rx / 32Tx / 32VA active
BUS	Indicator RS485 data traffic
RAD	Indicator EnOcean radio traffic
ERR	Indicator Error message

DIP 2.1



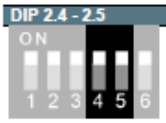
1	Transmission mode
off	Modbus RTU
on	Modbus ASCII

DIP 2.2 - 2.3



2	3	Baud rate
off	off	9600
on	off	19200
off	on	38400
on	on	57600

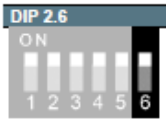
DIP 2.4 - 2.5



When ASCII mode is enabled, EVEN or ODD parity must be selected. "No parity" is not available in ASCII mode.

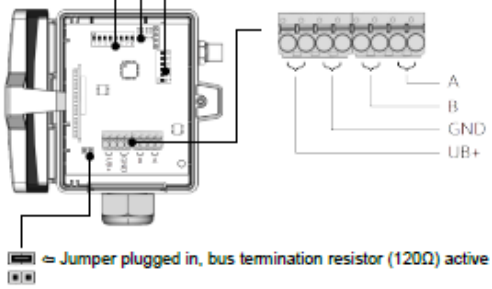
4	5	Parity
on	off	even – 1-Stopbits
off	on	odd – 1 Stopbit
off	off	none – 2 Stopbit

DIP 2.6



Rx = Recipient channels
Tx = Transmission channels
VA = Valve actuators (i.e. SAB)

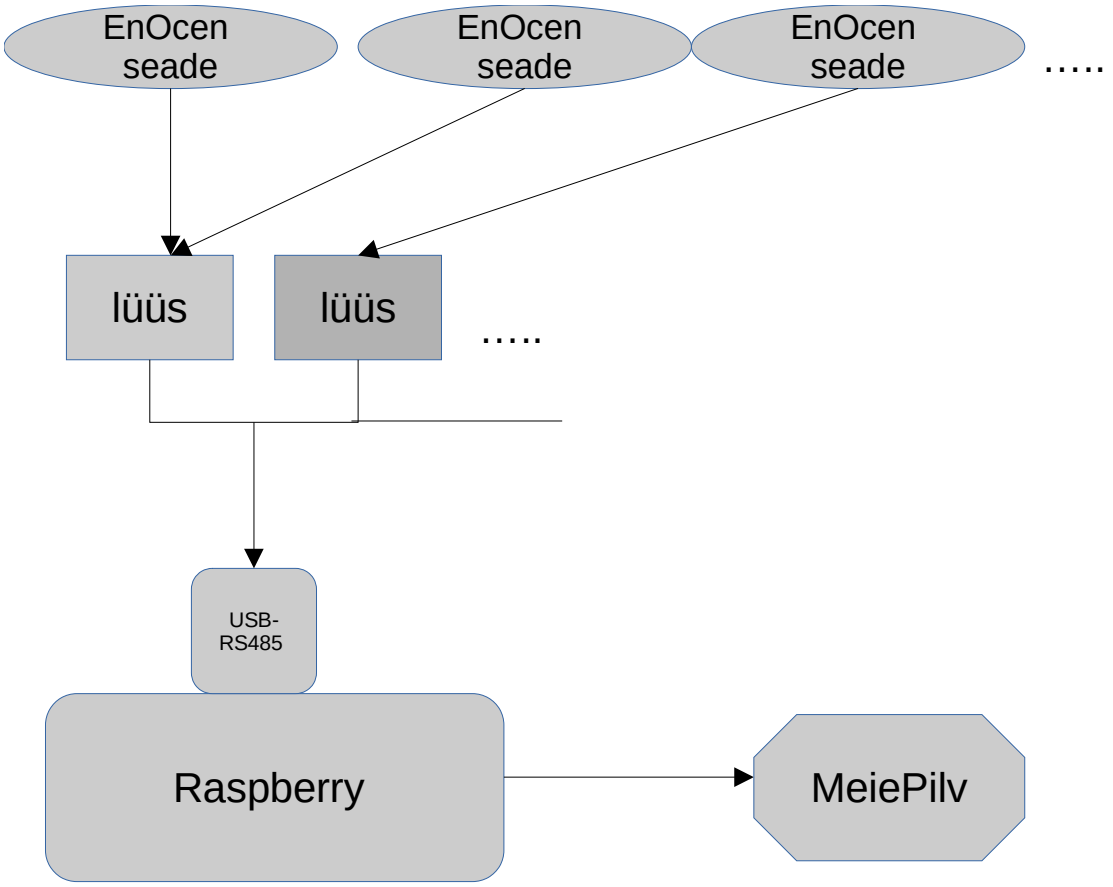
6	Version
off	32Rx / 8Tx
on	32Rx / 32Tx / 32VA



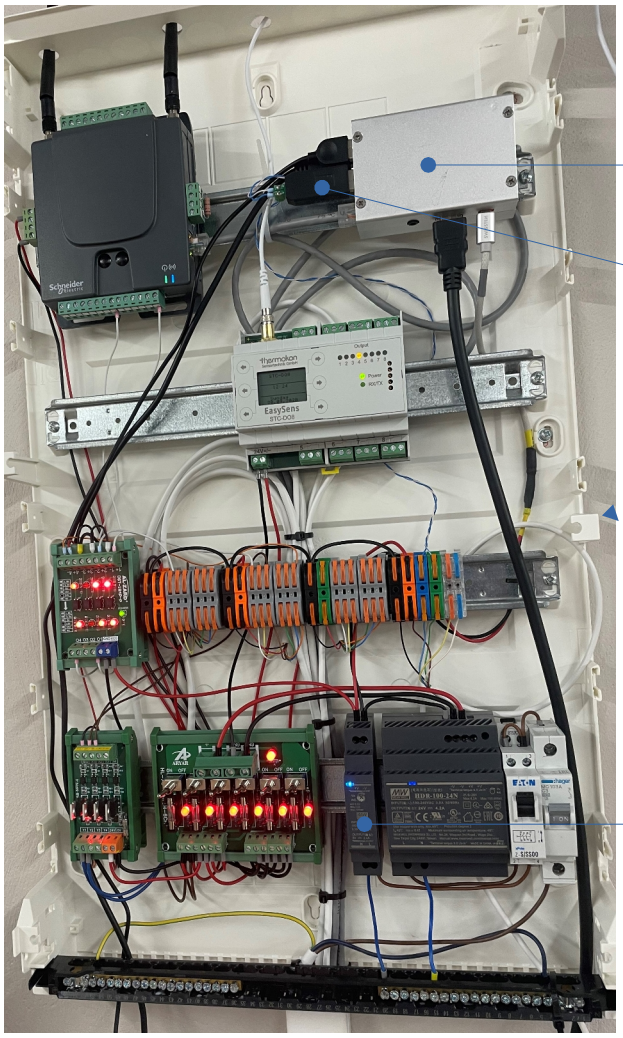
⇦ Jumper plugged in, bus termination resistor (120Ω) active

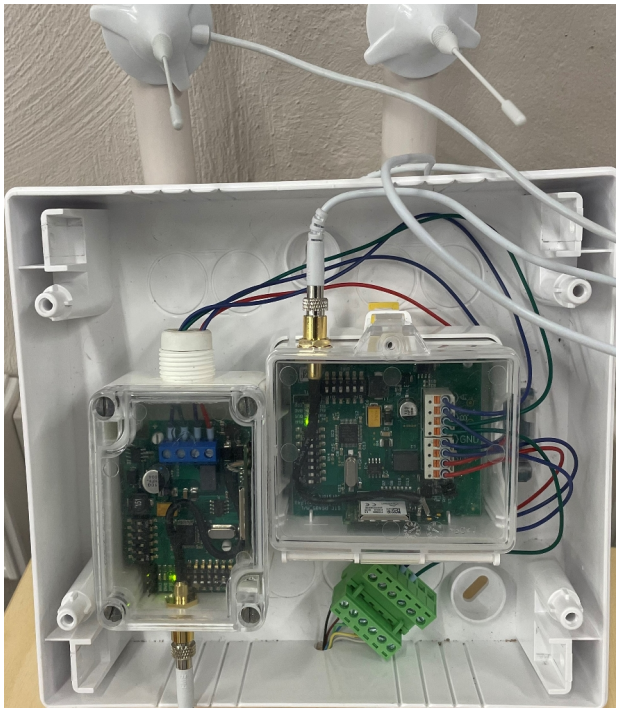
A = TxD+ / RxD+ = A+/ non-inverted signal
B = TxD- / RxD- = B-/ inverted signal

Lisa 5. EnOcen raadioliikluse jälgimissüsteemi skeem

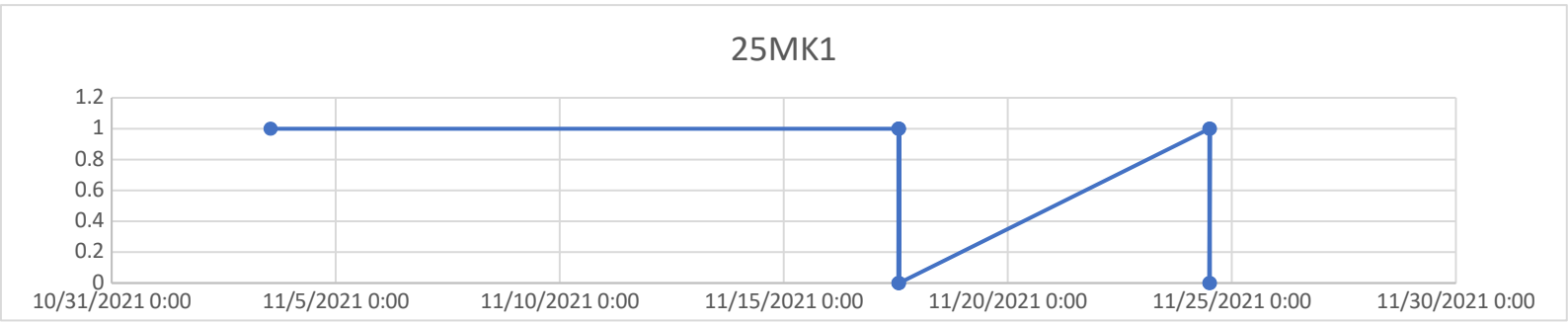


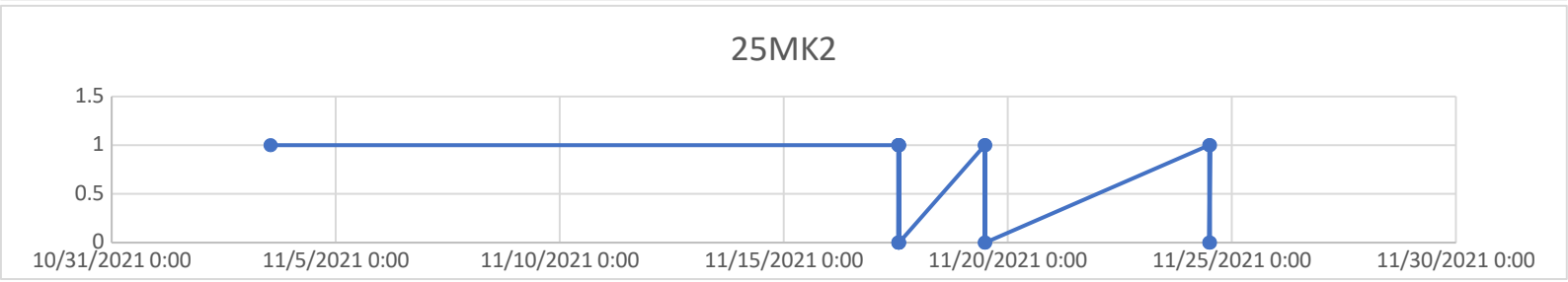
Lisa 6. Valmis seade automaatikakilbis

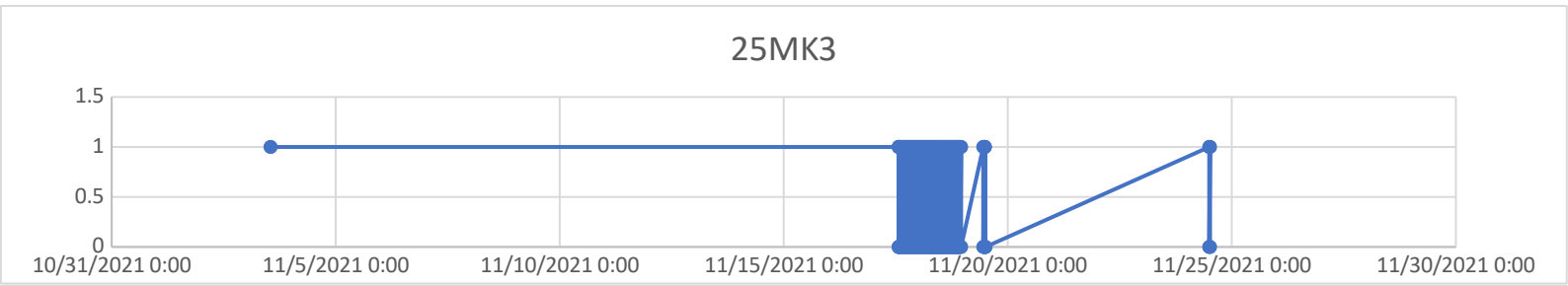


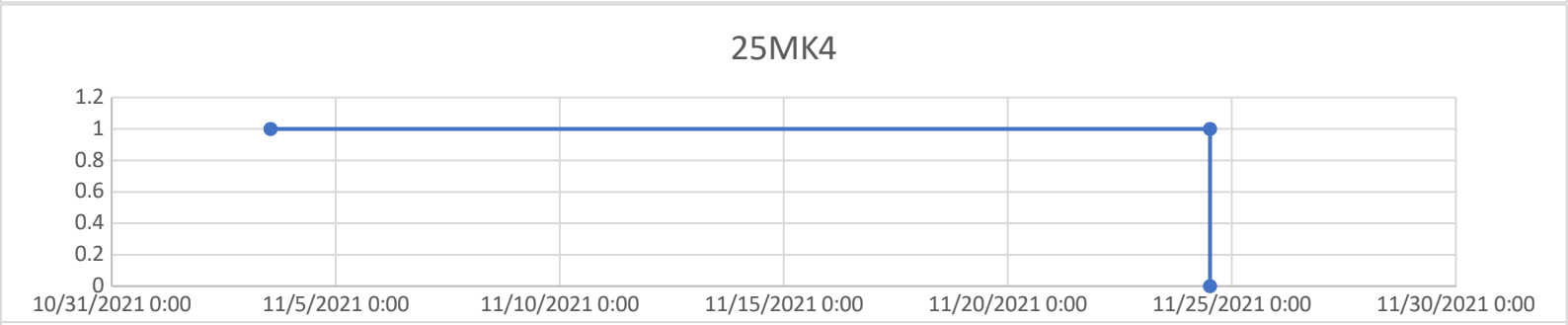


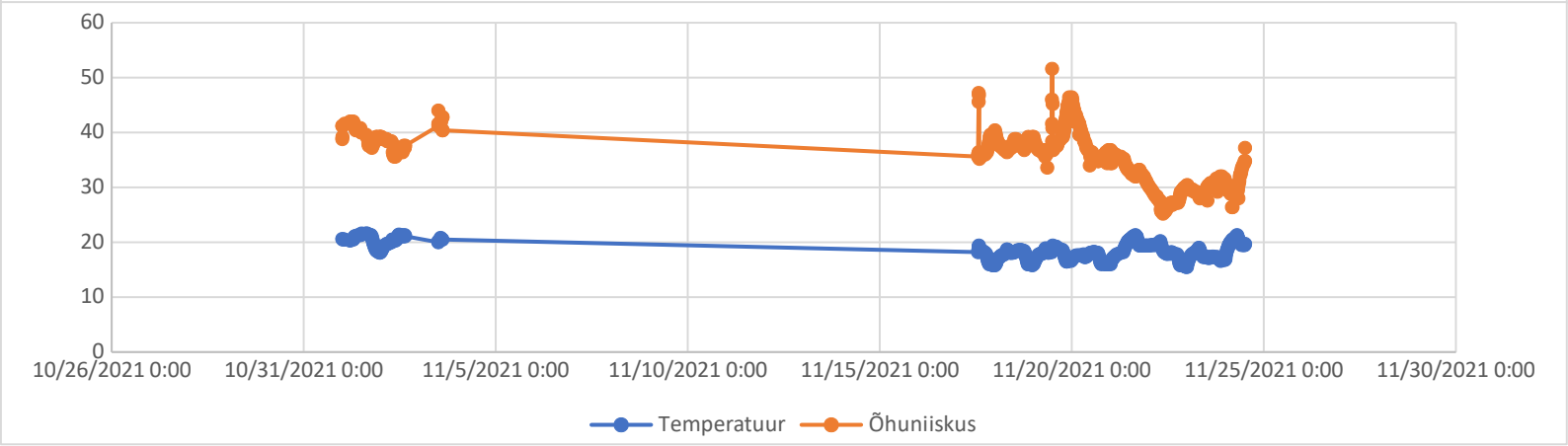
Lisa 7. Eluslaboratooriumi seadme testi tulemused











Lisa 8. Programmi kood

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import minimalmodbus
import csv
import sys
from sys import platform
import os
from subprocess import Popen
from json import JSONEncoder
import json
import shutil
import time
from threading import Thread, Event
import PySimpleGUI as sg
import re
import http.client


def jsonFileName():
    jsonFilename = "defaultValues.json"
    return jsonFilename

def reservCopyJsonFileName():
    jsonFilename = "defaultValuesReservCopy.json"
    return jsonFilename

def csvFileName():
    csvFilename = "devicelist.csv"
    return csvFilename

def loadingDataFromReservCopy():
    shutil.copyfile(reservCopyJsonFileName(), jsonFileName())

def rebootProgramm():
    mainProgramm = False
    time.sleep(5)
    python = sys.executable
    os.execl(python, python, *sys.argv)

def endProgramm ():
    mainProgramm = False
    time.sleep(2)
    sys.exit()

def modbusValueToASCIIValue(rangeMax, rangeMin, lsbMax, lsbMin, modbusValue):
    ASCIIValue = (rangeMax - rangeMin) / (lsbMax - lsbMin) * (modbusValue - lsbMin) - rangeMin
    return ASCIIValue


def numericFilter(string):
    filteredString = re.sub("[^0-9]", "", string)
    if not filteredString:
```

```
        return 0
    return filteredString


def httpMeiPilvMessage(message, hostname, port, MSdestination):
    global errorCounter
    global mainProgramm
    global allowedErrors

    if mainProgramm and errorCounter < allowedErrors:
        try:
            connection = http.client.HTTPConnection(hostname, port)
            headers = {'Content-type': 'text/plain',
                       'Permission': 'a902eef5a4c28aebb401a260e0e45c0cbb424bbc62c1f72bd92215d3451ed1d0'}
            connection.request('POST', MSdestination, message, headers)
            response = connection.getresponse()
            print(response.read().decode())
            errorCounter = 0
        except http.client.HTTPException as e:
            print("Probleem internetiühendusega.")
            print("Kontrolli programmi seadistusi. Vajadusel vajuta nuppu 'Taasta vaikeseaded'")
            time.sleep(10)
            errorCounter += 1
            httpMeiPilvMessage(message, hostname, port, MSdestination)
    else:
        endProgramm()

def ConnectionWithModbus(register, PORT, SLadress, baudRate, byteSize, stopBits, timeOut, serialParity,
serialMode):
    global errorCounter
    global mainProgramm
    global allowedErrors

    if mainProgramm and errorCounter < allowedErrors:
        try:
            thermokon = minimalmodbus.Instrument(PORT, int(SLadress))
            if serialMode == True:
                thermokon.mode = minimalmodbus.MODE_RTU
            elif serialMode == False:
                thermokon.mode = minimalmodbus.MODE_ASCII
            else:
                thermokon.mode = minimalmodbus.MODE_RTU

            if serialParity == True:
                thermokon.serial.parity = minimalmodbus.serial.PARITY_EVEN
            elif serialParity == False:
                thermokon.serial.parity = minimalmodbus.serial.PARITY_NONE
            else:
                thermokon.serial.parity = minimalmodbus.serial.PARITY_EVEN

            thermokon.serial.baudrate = int(baudRate)
            thermokon.serial.bytesize = int(byteSize)
            thermokon.serial.stopbits = int(stopBits)
            thermokon.serial.timeout = int(timeOut)
            thermokon.close_port_after_each_call = True
            thermokon.clear_buffers_before_each_transaction = True
            registerValue = thermokon.read_register(register)
            errorCounter = 0
            return registerValue
```

```
        except IOError:
            print("Ei saanud andmeid lugeda registrist: ", register)
            print("Kontrolli ühendust lüüsiga. Korraga saab olema ühendatud ainult üks Master seade ")
            print("Kontrolli programmi seadistusi. Vajadusel vajuta nuppu 'Taasta vaikeseaded'")
            time.sleep(10)
            errorCounter += 1
            ConnectionWithModbus(register, PORT, SLadress, baudRate, byteSize, stopBits, timeOut,
serialParity, serialMode)
    else:
        endProgramm ()

def dataReader(filename):
    try:
        devicelist = []
        csvFileReader = csv.DictReader(open(filename))
        for row in csvFileReader:
            devicelist.append(row)
        return devicelist
    except IOError:
        print("Ei saanud csv faili lugemiseks faili avada")
        time.sleep(20)

def dataJSONReader(filename):
    try:
        jsonReader = open(filename, "r")
        jsonFileContent = jsonReader.read()
        pythonDict = json.loads(jsonFileContent)
        return pythonDict
    except ValueError as e:
        print("JSON faili sisu formaat ei vasta nõetele: ", e)
    except IOError:
        print("Ei saanud JSON faili lugemiseks avada")
        time.sleep(20)

def dataJSONWriter(filename, jsonString):
    try:
        outJsonString = jsonString
        with open(filename, 'w') as outfile:
            json.dump(outJsonString, outfile)
    except IOError:
        print("Ei saanud JSON faili kirjutamiseks avada")
        time.sleep(20)

class controllerHT:
    def __init__(self, channel, devicetype, MPname, temperatureRegister, humidityRegister, SLadress):
        self.channel = channel
        self.SLadress = SLadress
        self.devicetype = devicetype
        self.MPname = MPname
        self.temperatureRegister = temperatureRegister
        self.humidityRegister = humidityRegister
        self.temperature = 0
        self.humidity = 0
        self.trange = [0, 40]
        self.tlsbScale = [0, 250]
        self.hrange = [0, 100]
        self.hlsbScale = [0, 250]

    def temperatureRegisterValue(self):
```

```
        value = ConnectionWithModbus(int(self.temperatureRegister), defaultValues.get('modbusPort'),
int(self.SLadress),
                                     defaultValues.get('baudRate'), defaultValues.get('byteSize'),
                                     defaultValues.get('stopBits'), defaultValues.get('timeOut'),
                                     defaultValues.get('serialParity'), defaultValues.get('serialMode'))
        return modbusValueToASCIIValue(self.trange[1], self.trange[0], self.tlsbScale[1], self.tlsbScale[0],
value)

    def humidityRegisterValue(self):
        value = ConnectionWithModbus(int(self.humidityRegister), defaultValues.get('modbusPort'),
int(self.SLadress),
                                     defaultValues.get('baudRate'), defaultValues.get('byteSize'),
                                     defaultValues.get('stopBits'), defaultValues.get('timeOut'),
                                     defaultValues.get('serialParity'), defaultValues.get('serialMode'))
        return modbusValueToASCIIValue(self.hrange[1], self.hrange[0], self.hlsbScale[1], self.hlsbScale[0],
value)


class switch:
    def __init__(self, channel, devicetype, MPname, stateRegister, SLadress):
        self.channel = channel
        self.SLadress = SLadress
        self.devicetype = devicetype
        self.MPname = MPname
        self.stateRegister = stateRegister
        self.onState = 0
        self.stateCriticalValue = 0

    def switchState(self):
        value = ConnectionWithModbus(int(self.stateRegister), defaultValues.get('modbusPort'),
int(self.SLadress),
                                     defaultValues.get('baudRate'), defaultValues.get('byteSize'),
                                     defaultValues.get('stopBits'), defaultValues.get('timeOut'),
                                     defaultValues.get('serialParity'), defaultValues.get('serialMode'))
        if (value > self.stateCriticalValue):
            return 1
        else:
            return 0


class switchD:
    def __init__(self, channel, devicetype, MPname, stateRegister, SLadress):
        self.channel = channel
        self.SLadress = SLadress
        self.devicetype = devicetype
        self.MPname = MPname
        self.stateRegister = stateRegister
        self.onState = 0
        self.vrange = [0, 16, 80, 21]

    def switchState(self):
        value = ConnectionWithModbus(int(self.stateRegister), defaultValues.get('modbusPort'),
int(self.SLadress),
                                     defaultValues.get('baudRate'), defaultValues.get('byteSize'),
                                     defaultValues.get('stopBits'), defaultValues.get('timeOut'),
                                     defaultValues.get('serialParity'), defaultValues.get('serialMode'))
        if (value == self.vrange[3]):
            return 3
        elif (value == self.vrange[1]):
```

```
            return 1
        elif (value == self.vrange[2]):
            return 2
        else:
            return self.vrange[0]


class windowI:
    def __init__(self, channel, devicetype, MPname, stateRegister, SLadress):
        self.channel = channel
        self.SLadress = SLadress
        self.devicetype = devicetype
        self.MPname = MPname
        self.stateRegister = stateRegister
        self.onState = 0
        self.vrange = [8]

    def windowIState(self):
        value = ConnectionWithModbus(int(self.stateRegister), defaultValues.get('modbusPort'),
int(self.SLadress),
                                     defaultValues.get('baudRate'), defaultValues.get('byteSize'),
                                     defaultValues.get('stopBits'), defaultValues.get('timeOut'),
                                     defaultValues.get('serialParity'), defaultValues.get('serialMode'))
        if (value == self.vrange[0]):
            return 1
        else:
            return 0


class lightSI:
    def __init__(self, channel, devicetype, MPname, lightRegister, powerSupplyRegister, SLadress):
        self.channel = channel
        self.SLadress = SLadress
        self.devicetype = devicetype
        self.MPname = MPname
        self.lightRegister = lightRegister
        self.powerSupplyRegister = powerSupplyRegister
        self.light = 0
        self.volts = 0.0
        self.lrange = [0, 1020]
        self.llsbScale = [0, 255]
        self.vrange = [0.0, 5.1]
        self.vlsbScale = [0.0, 255.0]

    def lightRegisterValue(self):
        value = ConnectionWithModbus(int(self.lightRegister), defaultValues.get('modbusPort'),
int(self.SLadress),
                                     defaultValues.get('baudRate'), defaultValues.get('byteSize'),
                                     defaultValues.get('stopBits'), defaultValues.get('timeOut'),
                                     defaultValues.get('serialParity'), defaultValues.get('serialMode'))
        return modbusValueToASCIIValue(self.lrange[1], self.lrange[0], self.llsbScale[1], self.llsbScale[0],
value)

    def powerSupplyRegisterValue(self):
        value = ConnectionWithModbus(int(self.powerSupplyRegister), defaultValues.get('modbusPort'),
int(self.SLadress),
                                     defaultValues.get('baudRate'), defaultValues.get('byteSize'),
                                     defaultValues.get('stopBits'), defaultValues.get('timeOut'),
                                     defaultValues.get('serialParity'), defaultValues.get('serialMode'))
```

```
        return modbusValueToASCIIValue(self.vrange[1], self.vrange[0], self.vlsbScale[1], self.vlsbScale[0],
value)


class oSPIR:
    def __init__(self, channel, devicetype, MPname, occupancyRegister, powerSupplyRegister, SLadress):
        self.channel = channel
        self.SLadress = SLadress
        self.devicetype = devicetype
        self.MPname = MPname
        self.occupancyRegister = occupancyRegister
        self.powerSupplyRegister = powerSupplyRegister
        self.occupancy = 0
        self.orange = [127]
        self.volts = 0.0
        self.vrange = [0.0, 5.0]
        self.vlsbScale = [0.0, 250.0]

    def occupancyState(self):
        value = ConnectionWithModbus(int(self.occupancyRegister), defaultValues.get('modbusPort'),
int(self.SLadress),
                                     defaultValues.get('baudRate'), defaultValues.get('byteSize'),
                                     defaultValues.get('stopBits'), defaultValues.get('timeOut'),
                                     defaultValues.get('serialParity'), defaultValues.get('serialMode'))
        if (value > self.orange[0]):
            return 1
        else:
            return 0

    def powerSupplyRegisterValue(self):
        value = ConnectionWithModbus(int(self.powerSupplyRegister), defaultValues.get('modbusPort'),
int(self.SLadress),
                                     defaultValues.get('baudRate'), defaultValues.get('byteSize'),
                                     defaultValues.get('stopBits'), defaultValues.get('timeOut'),
                                     defaultValues.get('serialParity'), defaultValues.get('serialMode'))
        return modbusValueToASCIIValue(self.vrange[1], self.vrange[0], self.vlsbScale[1], self.vlsbScale[0],
value)


def tranformToListOfObjects(arrayofDevices):
    listOfObjects = []
    for DevArr in arrayofDevices:
        deviceType = DevArr.get('devicetype')
        if (deviceType == 'controllerHT'):
            registers = DevArr.get('registers')
            arrRegisters = registers.split()
            controller = controllerHT(DevArr.get('channel'), DevArr.get('devicetype'), DevArr.get('MPname'),
                                      arrRegisters[0], arrRegisters[1], DevArr.get('SLadress'))
            range = DevArr.get('range')
            rangeList = range.split()
            scale = DevArr.get('scale')
            scaleList = scale.split()
            controller.trange[0] = int(rangeList[0])
            controller.trange[1] = int(rangeList[1])
            controller.hrange[0] = int(rangeList[2])
            controller.hrange[1] = int(rangeList[3])
            controller.tlsbScale[0] = int(scaleList[0])
            controller.tlsbScale[1] = int(scaleList[1])
            controller.hlsbScale[0] = int(scaleList[2])
```

```
    controller.hlsbScale[1] = int(scaleList[3])
    controller.temperature = controller.temperatureRegisterValue()
    controller.humidity = controller.humidityRegisterValue()
    listOfObjects.append(controller)

elif (deviceType == 'switch'):
    controller = switch(DevArr.get('channel'), DevArr.get('devicetype'), DevArr.get('MPname'),
                        DevArr.get('registers'), DevArr.get('SLadress'))
    controller.stateCriticalValue = int(DevArr.get('range'))
    controller.onState = controller.switchState()
    listOfObjects.append(controller)

elif (deviceType == 'switchD'):
    controller = switchD(DevArr.get('channel'), DevArr.get('devicetype'), DevArr.get('MPname'),
                         DevArr.get('registers'), DevArr.get('SLadress'))
    stateCriticalValues = DevArr.get('range')
    stateCriticalValueList = stateCriticalValues.split()
    controller.vrange[0] = int(stateCriticalValueList[0])
    controller.vrange[1] = int(stateCriticalValueList[1])
    controller.vrange[2] = int(stateCriticalValueList[2])
    controller.vrange[3] = int(stateCriticalValueList[3])
    controller.onState = controller.switchState()
    listOfObjects.append(controller)

elif (deviceType == 'windowI'):
    controller = windowI(DevArr.get('channel'), DevArr.get('devicetype'), DevArr.get('MPname'),
                         DevArr.get('registers'), DevArr.get('SLadress'))
    stateCriticalValue = DevArr.get('range')
    stateCriticalValueList = stateCriticalValue.split()
    controller.vrange[0] = int(stateCriticalValueList[0])
    controller.onState = controller.windowIState()
    listOfObjects.append(controller)

elif (deviceType == 'lightSI'):
    registers = DevArr.get('registers')
    arrRegisters = registers.split()
    controller = lightSI(DevArr.get('channel'), DevArr.get('devicetype'), DevArr.get('MPname'),
                         arrRegisters[0], arrRegisters[1], DevArr.get('SLadress'))
    range = DevArr.get('range')
    rangeList = range.split()
    scale = DevArr.get('scale')
    scaleList = scale.split()
    controller.lrange[0] = int(rangeList[0])
    controller.lrange[1] = int(rangeList[1])
    controller.vrange[0] = float(rangeList[2])
    controller.vrange[1] = float(rangeList[3])
    controller.llsbScale[0] = int(scaleList[0])
    controller.llsbScale[1] = int(scaleList[1])
    controller.vlsbScale[0] = int(scaleList[2])
    controller.vlsbScale[1] = int(scaleList[3])
    controller.volts = controller.powerSupplyRegisterValue()
    controller.light = controller.lightRegisterValue()
    listOfObjects.append(controller)

elif (deviceType == 'oSPIR'):
    registers = DevArr.get('registers')
    arrRegisters = registers.split()
    controller = oSPIR(DevArr.get('channel'), DevArr.get('devicetype'), DevArr.get('MPname'),
                       arrRegisters[0], arrRegisters[1], DevArr.get('SLadress'))
```

```
            range = DevArr.get('range')
            rangeList = range.split()
            scale = DevArr.get('scale')
            scaleList = scale.split()
            controller.orange[0] = int(rangeList[0])
            controller.vrange[0] = float(rangeList[1])
            controller.vrange[1] = float(rangeList[2])
            controller.vlsbScale[0] = int(scaleList[0])
            controller.vlsbScale[1] = int(scaleList[1])
            controller.volts = controller.powerSupplyRegisterValue()
            controller.occupancy = controller.occupancyState()
            listOfObjects.append(controller)

    return listOfObjects


def meiePilvmessage(devName, subserverName, externalValues):
    message = ""
    upCom = '"'
    newL = '\n'
    message += "{" + newL + upCom + "device_name" + upCom + ": " + upCom + devName + upCom + "," 
+ newL + upCom + "network_name" + upCom + ": " + upCom + subserverName + upCom + "," + newL + 
upCom + "body" + upCom + ":  [" + newL + " {" + newL + "  " + upCom + "time" + upCom + ": " + "0," + 
newL + "  " + upCom + "values" + upCom + ": " + str(
        externalValues) + newL + "    }" + newL + " ]" + newL + "}" + newL
    return message


def objectMeiePilvmessage(obj, subserverName):
    someObject = obj
    devName = someObject.MPname
    if (someObject.devicetype == 'controllerHT'):
        listOfValues = [someObject.temperature, someObject.humidity]
        message = meiePilvmessage(devName, subserverName, listOfValues)
        return message
    elif (someObject.devicetype == 'switch'):
        listOfValues = [someObject.onState]
        message = meiePilvmessage(devName, subserverName, listOfValues)
        return message
    elif (someObject.devicetype == 'switchD'):
        listOfValues = [someObject.onState]
        message = meiePilvmessage(devName, subserverName, listOfValues)
        return message
    elif (someObject.devicetype == 'windowI'):
        listOfValues = [someObject.onState]
        message = meiePilvmessage(devName, subserverName, listOfValues)
        return message
    elif (someObject.devicetype == 'lightSI'):
        listOfValues = [someObject.light, someObject.volts]
        message = meiePilvmessage(devName, subserverName, listOfValues)
        return message
    elif (someObject.devicetype == 'oSPIR'):
        listOfValues = [someObject.occupancy, someObject.volts]
        message = meiePilvmessage(devName, subserverName, listOfValues)
        return message
    else:
        return ObjectsEncoder().encode(someObject)
```

```
class ObjectsEncoder(JSONEncoder):
    def default(self, object):
        if isinstance(object, controllerHT):
            return object.__dict__
        elif isinstance(object, switch):
            return object.__dict__
        elif isinstance(object, switchD):
            return object.__dict__
        elif isinstance(object, windowI):
            return object.__dict__
        elif isinstance(object, lightSI):
            return object.__dict__
        elif isinstance(object, oSPIR):
            return object.__dict__
        else:
            return json.JSONEncoder.default(self, object)


def trackChanges(arrayofDevices, subserverName, hostName, port):
    for DevArr in arrayofDevices:
        controller = DevArr
        if (controller.devicetype == 'controllerHT'):
            oldTemp = round(controller.temperature, 1)
            newTemp = round(controller.temperatureRegisterValue(), 1)
            oldHum = round(controller.humidity, 1)
            newHum = round(controller.humidityRegisterValue(), 1)

            if (oldTemp != newTemp or oldHum != newHum):
                controller.temperature = newTemp
                controller.humidity = newHum
                jsonString = ObjectsEncoder().encode(controller)
                print(jsonString)
                meiePilvJson = objectMeiePilvmessage(controller, subserverName)
                httpMeiPilvMessage(meiePilvJson, hostName, port, MSdestination)

        elif (controller.devicetype == 'switch'):
            if (controller.onState != controller.switchState()):
                controller.onState = controller.switchState()
                jsonString = ObjectsEncoder().encode(controller)
                print(jsonString)
                meiePilvJson = objectMeiePilvmessage(controller, subserverName)
                httpMeiPilvMessage(meiePilvJson, hostName, port, MSdestination)

        elif (controller.devicetype == 'switchD'):
            if (controller.onState != controller.switchState()):
                controller.onState = controller.switchState()
                jsonString = ObjectsEncoder().encode(controller)
                print(jsonString)
                meiePilvJson = objectMeiePilvmessage(controller, subserverName)
                httpMeiPilvMessage(meiePilvJson, hostName, port, MSdestination)

        elif (controller.devicetype == 'windowI'):
            if (controller.onState != controller.windowIState()):
                controller.onState = controller.windowIState()
                jsonString = ObjectsEncoder().encode(controller)
                print(jsonString)
                meiePilvJson = objectMeiePilvmessage(controller, subserverName)
                httpMeiPilvMessage(meiePilvJson, hostName, port, MSdestination)
```

```
        elif (controller.devicetype == 'lightSI'):
            oldLight = round(controller.light)
            newLight = round(controller.lightRegisterValue())
            if (oldLight != newLight):
                controller.light = newLight
                controller.volts = round(controller.powerSupplyRegisterValue(), 2)
                jsonString = ObjectsEncoder().encode(controller)
                print(jsonString)
                meiePilvJson = objectMeiePilvmessage(controller, subserverName)
                httpMeiPilvMessage(meiePilvJson, hostName, port, MSdestination)

        elif (controller.devicetype == 'oSPIR'):
            if (controller.occupancy != controller.occupancyState()):
                controller.occupancy = controller.occupancyState()
                controller.volts = round(controller.powerSupplyRegisterValue(), 2)
                jsonString = ObjectsEncoder().encode(controller)
                print(jsonString)
                meiePilvJson = objectMeiePilvmessage(controller, subserverName)
                httpMeiPilvMessage(meiePilvJson, hostName, port, MSdestination)


def gui():
    global defaultValues
    global pause
    global serverPort
    global subserverName
    global modbusPort
    global baudRate
    global stopBits
    global byteSize
    global timeOut
    global serverAdress
    global MSdestination
    global mainProgramm
    global serialParity
    global serialMode

    if not mainProgramm:
        window.close()
        endProgramm()

    sg.theme('BluePurple')
    layout = [[sg.Text('Andmete lugemise sagedus sekundites:'),
               sg.Input(default_text=defaultValues.get('pause'), size=(10, 1), key='-IN1-')],
              [sg.Text('MeiePilv serveri aadress:'),
               sg.Input(default_text=defaultValues.get('serverAdress'), size=(25, 1), key='-IN5-')],
              [sg.Text('MeiePilv serveri port:'),
               sg.Input(default_text=defaultValues.get('serverPort'), size=(10, 1), key='-IN2-'), sg.Text('sihtkoht:'),
               sg.Input(default_text=defaultValues.get('MSdestination'), size=(30, 1), key='-IN6-')],
              [sg.Text('MeiePilv alamserveri nimi:'),
               sg.Input(default_text=defaultValues.get('subserverName'), size=(20, 1), key='-IN3-')],
              [sg.Text('Lüüsi USBport:'),
               sg.Input(default_text=defaultValues.get('modbusPort'), size=(15, 1), key='-IN4-'),
               sg.Text('baudisagedus:'),
               sg.Input(default_text=defaultValues.get('baudRate'), size=(10, 1), key='-IN7-'),
               sg.Text('stop bitide arv:'),
               sg.Input(default_text=defaultValues.get('stopBits'), size=(3, 1), key='-IN8-'), sg.Text('baidi
pikkus:'),
               sg.Input(default_text=defaultValues.get('byteSize'), size=(3, 1), key='-IN9-'), sg.Text('ajalõpp
```

```
sek:'),
            sg.Input(default_text=defaultValues.get('timeOut'), size=(3, 1), key='-IN10-')],
           [sg.Text('Serial paarsus:                    Serial režiim:')],
           [sg.Radio('NONE', 'parity', default=not defaultValues.get('serialMode'), size=(5, 1)),
            sg.Radio('EVEN', 'parity', default=defaultValues.get('serialParity'), size=(5, 1), key='-IN11-'),
            sg.Text('      '), sg.Radio('ASCII', 'mode', default=not defaultValues.get('serialMode'),
                         size=(5, 1)),
            sg.Radio('RTU', 'mode', default=defaultValues.get('serialMode'), size=(5, 1), key='-IN12-')],
           [sg.Output(size=(130, 20))],
           [sg.Button('Rakenda'), sg.Button('Sulge aken'), sg.Button('Lopeta programm'),
            sg.Button('Lisa/Eemalda seadet'), sg.Button('Taasta vaikeseaded')]]
    window = sg.Window('Programmi seaded', layout)

    while True:
        event, values = window.read()
        pause = int(numericFilter((values['-IN1-'])))
        serverPort = int(numericFilter((values['-IN2-'])))
        subserverName = (values['-IN3-'])
        modbusPort = (values['-IN4-'])
        serverAdress = (values['-IN5-'])
        MSdestination = (values['-IN6-'])
        baudRate = int(numericFilter((values['-IN7-'])))
        stopBits = int(numericFilter((values['-IN8-'])))
        byteSize = int(numericFilter((values['-IN9-'])))
        timeOut = int(numericFilter((values['-IN10-'])))
        serialParity = (values['-IN11-'])
        serialMode = (values['-IN12-'])

        if event == 'Rakenda':
            defaultValues['pause'] = pause
            defaultValues['serverPort'] = serverPort
            defaultValues['subserverName'] = subserverName
            defaultValues['modbusPort'] = modbusPort
            defaultValues['serverAdress'] = serverAdress
            defaultValues['MSdestination'] = MSdestination
            defaultValues['baudRate'] = baudRate
            defaultValues['stopBits'] = stopBits
            defaultValues['byteSize'] = byteSize
            defaultValues['timeOut'] = timeOut
            defaultValues['serialParity'] = serialParity
            defaultValues['serialMode'] = serialMode
            dataJSONWriter(jsonFileName(), defaultValues)
            rebootProgramm()

        if event == 'Lopeta programm' or sg.WIN_CLOSED or event == 'Sulge aken' or stopEvent.set():
            window.close()
            endProgramm ()

        if event == 'Lisa/Eemalda seadet':
            try:
                Popen(['xdg-open', csvFileName()])
            except:
                print("Ei saanud Seadmete lisamiseks konfiguratsiooni faili avada")
                continue

        if event == 'Taasta vaikeseaded':
            loadingDataFromReservCopy()
            rebootProgramm()
```

```
    window.close()


try:
    try:
        mainProgramm = True
        errorCounter = 0
        allowedErrors = 20
        defaultValues = dataJSONReader(jsonFileName())
        pause = defaultValues.get('pause')
        serverPort = defaultValues.get('serverPort')
        subserverName = defaultValues.get('subserverName')
        serverAdress = defaultValues.get('serverAdress')
        MSdestination = defaultValues.get('MSdestination')
        modbusPort = defaultValues.get('modbusPort')
        baudRate = defaultValues.get('baudRate')
        stopBits = defaultValues.get('stopBits')
        byteSize = defaultValues.get('byteSize')
        timeOut = defaultValues.get('timeOut')
        serialParity = defaultValues.get('serialParity')
        serialMode = defaultValues.get('serialMode')

    except:
        print("Tekkis viga. Programm käivutub 30 pärast sekundi")
        rebootProgramm()
    else:
        if mainProgramm:
            stopEvent = Event()
            timeoutGui = Thread(target=gui)
            timeoutGui.daemon = True
            timeoutGui.start()
            timeoutGui.join(timeout=3)
            stopEvent.set()
            devicelist = []
            objectDevicesList = []
            devicelist = dataReader(csvFileName())
            objectDevicesList = tranformToListOfObjects(devicelist)

        while mainProgramm:
            trackChanges(objectDevicesList, subserverName, serverAdress, serverPort)
            time.sleep(int(pause))

except (IOError, EOFError) as e:
    print("Tekkis viga.", e)
    print("Arvuti taaskaivitub 5 minuti parast")
    mainProgramm = False
    time.sleep(300)
    if platform == "linux" or platform == "linux1" or platform == "linux2":
        os.system('sudo shutdown -r now')
    elif platform == "darwin":
        os.system('shutdown -r now')
    elif platform == "win32":
        os.system("shutdown /r /t 1")
    else:
        os.system('sudo shutdown -r now')
```