TALLINNA TEHNIKAÜLIKOOL
INSENERITEADUSKOND
Tartu kolledž

RUUMI KESKKÜTTE JUHTMOODULI VÄLJA TÖÖTAMINE ELUSLABORATOORIUMI TARBEKS

DEVELOPMENT OF A CENTRAL HEATING CONTROL MODULE FOR THE LIVING LABORATORY

RAKENDUSKÕRGHARIDUSTÖÖ

Üliõpilane: Janno Järvpõld
/nimi/

Üliõpilaskood 178270EDTR

Juhendaja: Ago Rootsi, lektor
/nimi, amet/

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.

juuni 2021.a

Autor: Janno Järvpõld / allkirjastatud digitaalselt /

Töö vastab bakalaureusetöö/magistritööle esitatud nõuetele

juuni 2021.a

Juhendaja: Ago Rootsi / allkirjastatud digitaalselt /

Kaitsmisele lubatud

juuni 2021.a

Kaitsmiskomisjoni esimees: Helle Hallik / allkirjastatud digitaalselt /

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Janno Järvpõld (sünnikuupäev: 23.09.1997)

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose

„RUUMI KESKKÜTTE JUHTMOODULI VÄLJATÖÖTAMINE ELUSLABORATOORIUMI TARBEKS“,

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.*

/ allkirjastatud digitaalselt /

juuni 2021.a

TalTech Tartu kolledž

LÕPUTÖÖ ÜLESANNE

Üliõpilane: Janno Järvpõld, 178270EDTR

Õppekava, peaeriala: EDTR17/17 - Telemaatika ja arukad süsteemid

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

Lõputöö teema:

(eesti keeles) RUUMI KESKKÜTTE JUHTMOODULI VÄLJA TÖÖTAMINE ELUS-LABORATOORIUMI TARBEKS

(inglise keeles) DEVELOPMENT OF A CENTRAL HEATING CONTROL MODULE FOR THE LIVING LABORATORY

Lõputöö põhieesmärgid:

1.
2.
3.

Lõputöö etapid ja ajakava:

Nr	Ülesande kirjeldus	Tähtaeg
1.		
2.		
3.		

Töö keel: Eesti keel **Lõputöö esitamise tähtaeg:** 31. mai 2021.a

Üliõpilane: Janno Järvpõld / allkirjastatud digitaalselt / juuni 2021.a

Juhendaja: Ago Rootsi / allkirjastatud digitaalselt / juuni 2021.a

Programmijuht: Helle Hallik / allkirjastatud digitaalselt / juuni 2021.a

SISUKORD

EESSÕNA

Lõputöö teema sõnastas Tallinna Tehnikaülikooli Tartu kolledži lektor Ago Rootsi. Sammuti pärinevad temalt viited erinevatele materjalidele ja käesoleva tööga seotud olemasolevatele süsteemidele.

Eluslabor, keskküte, juhtmoodul, bakalaureusetöö.

Lühendite, tähiste ja mõistete loetelu

API - Rakendustarkvara liides ehk API (ing. k. *Application Program Interface*) on arvutiprogrammides alamprogrammi määratluste, protokollide ja tööriistade komplekt rakendustarkvara ehitamiseks.

GND – Maandusklemm, vastavalt kontekstile ka ühine klemm elektriskeemis. (ing. k. *ground*)

JSON - JSON (ing. k. *JavaScript Object Notation*) on lihtsustatud andmevahetusvorming, mis põhineb JavaScripti programmeerimiskeele alamhulgal. JSON kasutab inimloetavat teksti atribuutide-väärtuste paaridest ja massiividest koosnevate andmeobjektide salvestamiseks ja edastamiseks.

Wi-Fi – Wi-Fi on IEEE 802.11 standarditel põhinev traadita võrguprotokollide perekond, mida kasutatakse tavaliselt seadmete kohtvõrgu ja Interneti-ühenduse loomiseks, ning mis võimaldab paljudel kasutajatel Wi-Fi-levialades samaaegselt luua juhtmevaba ühendus Internetiga.

MPM kontroller – MPM (ing. k. *Multi-Purpose Manager*) kontroller on kontroller, mis on võimeline juhtima kõiki hoone sisekliima, valgustuse ja ka muid parameetreid. Käesolevas kontekstis Schneider Electric kontroller – värat MPM-UN-CI4-5045. Võib toimida iseseisva kontrollerina või siis väratina mõne kõrgema taseme kontrolleri jaoks.

PID kontroller – PID (proportsionaal-integraal-diferentsiaal) kontroller on tehisjuhtimissüsteemides üks levinumaid tagasisidestatud juhtimisskeeme, kus rööbiti on lülitatud proportsionaal-, integraal- ja diferentsiaalregulaatorid.

PWM – Pulsilaiusmodulatsioon (PWM, ing. k. *Pulse-Width Modulation*) ehk impulsilaiusmodulatsioon ehk laiusimpulssmodulatsioon on modulatsiooni liik, milles väljundpinge keskväärtuse reguleerimiseks muudetakse impulsside laiust.

I²C – I²C on mitme võimaliku ülemseadmega jadasiin, mida kasutatakse väikese kiirusega perifeeriaseadmete ühendamisel emaplaatide, manussüsteemide, mobiiltelefonide või muude seadmetega.

DC – alalisvool (ing. k. *Direct Current*)

GPIO - GPIO (ing. k. G*eneral Purpose Input/Output*) on mitmeotstarbeline sisend/väljund. Kasutaja saab ise kontrollida signaali käitumist ning määrata, kas see

toimib sisendi või väljundina. GPIO-sid vaikimisi ei kasutata ning neil pole vaikimisi määratud kindlat eesmärki.

HTTP - Hüperteksti edastusprotokoll (inglise keeles *Hypertext Transfer Protocol*) on protokoll teabe edastamiseks arvutivõrkudes. HTTP on andmete edastamise aluseks veebis.

UNIX ajatempel – UNIX ajatempel on viis jälgida aega jooksva sekundite arvuna. UNIX ajatempel võrdub sekundite arvuga, mis on möödunud kella 00:00:00 1. jaanuaril 1970 UTC aja järgi.

WAN - Laivõrk (ing. k. *Wide Area Network*) on telekommunikatsiooni võrk või arvutivõrk, mille kokkuleppeline ulatus on üle 1 km.

spindel – Spindliks nimetatakse üldjuhul mingi masina pöörlevat osa. Selle töö kontekstis tähendab spindel klapiajami liikuvat osa.

DC – DC muundur – Muundab kõrgema pingega alalisvoolu madalama pingega alalisvooluks või vastupidi.

Reguleerimisvahemik - Käesoleva töö kontekstis tähendab juhtimist mõjutavate tingimuste vahemikku, kus temperatuuri hoidmine on üldse võimalik. Vahemik on altpoolt piiratud küttevee minimaalse temperatuuriga, mille juures mingite konkreetsete ilmastikutingimuste korral üldse on võimalik ruumi vajaliku temperatuurini kütta. Kõrgemate temperatuuride poolt on see piiratud tingimustega kus muudest soojusallikatest (päikesekiirgus, arvutid, inimesed) emiteeritav energia ületab soojakadu ruumis ja keskkütteradiaatorite klappide sugemine ei peata temperatuuritõusu.

SISSEJUHATUS

Küberfüüsikaliste süsteemide õpetamiseks on tarvilik, et küberfüüsikalist süsteemi oleks õppekeskkonnas võimalik uurida ja vaadelda. Tartu kolledži kampus – selle hooned, tehnosüsteem ja inimesed – on kõige parem meile olemasolev küberfüüsiline (-sotsiaalne) süsteem. Sellel on olemas kõik meie jaoks tähtsad omadused ja funktsioonid, ning me saame kõiki süsteemi osi kavandada oma katsete jaoks, täpselt enda vajadustest lähtuvalt. Oleme oma õppe- ja uurimistöös kampuse igapäevaelu keskel ja saame tänu sellele süsteemis toimivaid protsesse hõlpsasti jälgida. Olles selle süsteemi osa, saame süsteemi teistelt osadelt vahetut tagasisidet ning paljudele asjadele saame hinnangu anda oma isiklike kogemuste põhjal. Selliselt üles ehitatud keskkonda nimetame eluslaboriks. Tüüpilisest laborist erineb eluslabor eelkõige selle poolest, et ei looda tehiskeskkonda üksikute protsesside uurimiseks, vaid terviklik süsteem töötab, ja seda arendatakse, jooksvalt. [1]

Eluslabor tähendab hoone vaatlemist kui üksteisega suhtlevate süsteemide süsteemi. Sellisteks süsteemideks on näiteks füüsilise hoone termaalne süsteem, energia-jaotussüsteem, andurite ja täiturmehhanismide süsteem, sotsiaalne süsteem. Seega arendame meetodeid kogu kirjeldatud süsteemide süsteemi simuleerimiseks, modelleerimiseks ja kontrolliks koos rakendusega hoone energiatarbe minimeerimiseks tagades samal ajal inimestele sobiva sisekliima. [1]

Tartu kolledži poolt arendatav eluslaboratoorium on keskendunud praeguses arengufaasis eelkõige Puiestee 80A õppehoone sisekliimaautomaatikale. Automaatikas tagasisidena kasutatavad sisekliimaandmed ja automaatika enda seisundid kogutakse aegridadena MeiePilv nimelisse andmebaasi. Pikemas perspektiivis on planeeritud arendada Puiestee 80A digikaksikuid, millest käesolevat tööd puudutab eelkõige hoone soojustehnilisi näitajaid ja tehnosüsteemi käsitlev osa. Nimetatud osa peaks võimaldama optimeerida sisekliimaautomaatika tööd taotledes mugavat sisekliimat minimaalse energiakuluga. Digikaksik võimaldab uusi lahendusi virtuaalselt läbi mängida ja alles siis rakendada algoritme füüsikalises maailmas. Digikaksik toetub MeiePilv andmebaasi kogutud andmetele ja digikaksiku poolne automaatika juhtimine käib MeiePilv kaudu. Seadesuuruste logimine andmebaasi on abiks analüüsil ja eriolukorras ka rikete leidmisel keerukas süsteemis. Nii talletatud info on suurepärane materjal analüüsiks, algoritmide silumiseks ja kindlasti ka õppetöös.

Käesoleva töö eesmärk tuleneb vajadusest täitursõlmede järele, mis toimiksid MeiePilv andmebaasi logitavate jooksvate tagasisideandmete ja digikaksiku poolt sinna kirjutatavate seadesuuruste põhjal.

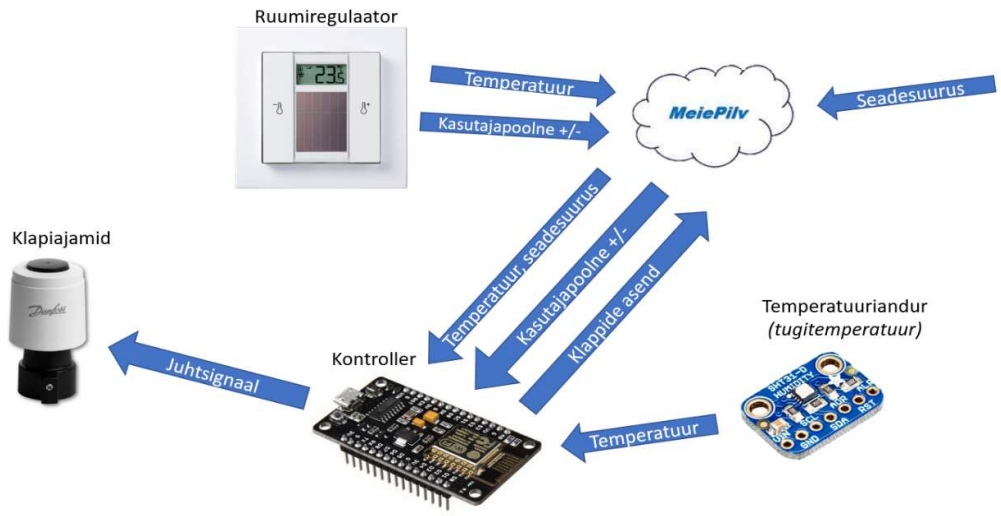
Tarvis oli välja töötada seade, mis juhib konkreetse ruumi temperatuuri kasutades MeiePilve salvestatud seadetemperatuuri ja tagasisidena ruumiregulaatori poolt samuti MeiePilv andmebaasi jooksvalt edastatavat temperatuurinäitu ja ruumi kasutaja poolt tehtavat kitsas piiris +/- häälestust.

1. LÄHTEÜLESANNE

1.1. Ülesande püstitus

Töö käigus väljatöötatav seade peab vastama vähemalt alljärgnevatele nõuetele:

- Juhib kuni nelja kahepositsioonilist termoelektrilist ajamit.
- Loeb võrdlustemperatuuri MeiePilv andmebaasist ja jälgib seejuures andmete pädevust.
- Loeb seadetemperatuuri MeiePilv andmebaasist.
- Teostab ruumi temperatuuri juhtimist nende andmete põhjal.
- Omab lahendust avariiolukorraks, kui peaks katkema side MeiePilv andmebaasiga, või sinna ei laeku värskeid võrdlusandmeid. Kummalgi toodud juhul ei või ruumi temperatuur seadeväärtusega võrreldes kontrollimatult muutuda. Lubatav muutumine reguleerimisvahemikus kummaski suunas maksimaalselt 1 kraad.
- Juhtmoodul peab töötama Puiestee 80A automaatikasüsteemi toitepingel, mis on 24 V. See on tüüpiline automaatikasüsteemide toitepinge.
- Juhtmoodul peab salvestama MeiePilv andmebaasi klapiajamite sisse- või väljalülitumiste momendid aegridadena kujul: „0" kui ajamite toide lülitati välja ja „8" kui lülitati sisse nii et moodustuks aegrida sisse- ja välja lülitumise hetkedest hilinemisega mitte üle 3 sekundi.
- Sobitumiseks eluslaboratooriumi ülejäänud osadega peab loodav süsteem olema integreeritud nii, nagu on toodud joonisel 1-1.



Joonis 1-1: Moodulite vaheline suhtluse skeem

1.2. Lähteülesande lahendamisega kaasnevad täiendavad ülesanded

Lähteülesandest tulenevalt mõeldi läbi kõikvõimalikud tõrkekohad, mis võivad väljatöötatava seadme tööd häirida ja tuvastati alljärgnevad probleemid, millele tuleb lahendused leida:

1.2.1 Tõrked andmevahetusel MeiePilv serveriga

Moodulite vahel edastatakse infot Wi-Fi võrgu kaudu, seega on alati, erinevatel põhjustel, olemas võimalus sidekatkestusteks. Samuti võivad ette tulla katkestused või tõrked serveri töös, näiteks võidakse serveri töö katkestada hooldustöödeks. Sellest tulenevalt oli vaja teha kindlaks, et töö käigus loodav prototüüp oleks võimeline juhtima ruumi temperatuuri ka siis, kui MeiePilve serverist ei ole võimalik värskeid andmeid kätte saada.

1.2.2 Tõrked/iseärasused ruumiregulaatori andmete MeiePilv andmebaasi kandmise süsteemis

Kolledži ruumides on sisekliima kohta info saamiseks Thermokon SR06 LCD tüüpi ruumiregulaatorid (Joonis 1-2).

Joonis 1-2: Thermokon SR06 LCD tüüpi ruumiregulaatorid [2]

Ruumiregulaator väljastab andmed EnOcean raadiosideprotokollis. Prototüübi loomise hetkel jälgis EnOcean andmeliiklust ja logis seda MeiePilv andmebaasi automaatikasüsteemi MPM kontroller. Järgneva aasta lõputööde hulgas on aga töö, mille eesmärgiks on luua eraldi seade, mis logib kogu EnOcean liiklust MeiePilv andmebaasi, andmete MeiePilve logimise viis ise aga ei mõjuta kavandatud seadme tööd. Prototüüp peab valmis olema olukorraks, kus andmete logimine MeiePilve on peatunud, värskeid andmeid ruumiregulaatorilt pole. Selle põhjuseks võib olla näiteks sidekatkestus või EnOcean raadioliiklust jälgiva mooduli toite puudumine. Samuti peab arvestama ruumiregulaatori iseärasustega:

- Ruumiregulaator saadab infot kord 100 sekundi jooksul
- Mõni pakett võib minna levis kaduma
- Talvel, eriti koolivaheajal, kui ruumides tuled ei põle, ei pruugi olla valgust ruumiregulaatori päikesepaneeli tarbeks piisavalt ja mingi perioodi jooksul andmeid ei tulegi. Seade võimaldab patareitoidet, aga praegusel hetkel seda ei kasutata ning ka patarei võib saada tühjaks.

1.2.3 Toitekatkestused

Toitekatkestused võivad olla nii üldised kui lokaalsed. Üldise toitekatkestuse puhul on tähtis see, et seade läheks peale toitekatkestust toite taastumisel sujuvalt ja kiiresti tööle. Toitekatkestuse järel või sisselülitamisel võib järgmise paketini minna 100 sekundit, võimalik, et enamgi, kuna ka edastussüsteem peab uuesti tööle hakkama, ning selle aja sees peab prototüüp olema võimeline temperatuuri reguleerima ilma väliste andmeteta. Lokaalse toitekatkestuse puhul, näiteks kui prototüübil on toide, kuid vähemalt osa ülejäänud süsteemist (Wi-Fi, MeiePilv, ruumiregulaator) on toiteta, peab prototüüp olema võimeline töötama täiesti iseseisvalt, ilma mujalt pärinevate

andmeteta. Seda ka siis, kui prototüüp pole peale sisse lülitamist mitte ühtegi paketti MeiePilvest kätte saanud.

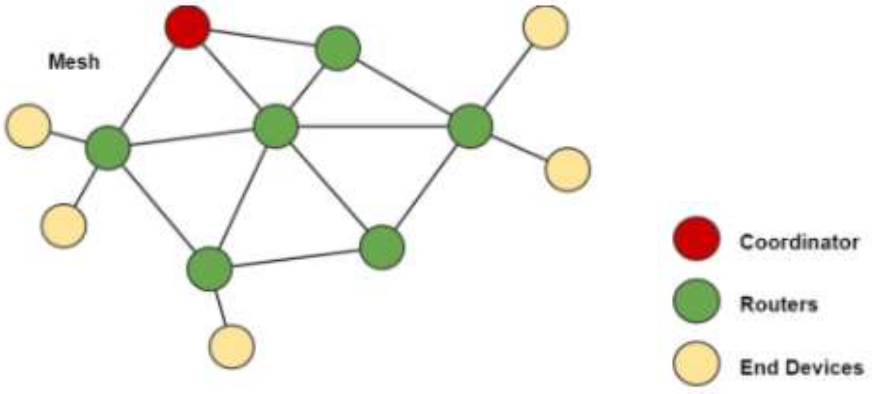
2. VÕIMALIKUD SOBIVAD TOOTELAHENDUSED

Olles täpsustanud lähteülesandest tulenevad eesmärgid ja piirangud, asusin otsima võimalikke olemasolevaid tooteid, mis sobiksid käesoleva ülesande lahendamiseks või mida oleks võimalik sobivaks kohandada. Selgus, et käesoleva töö lähteülesanne on ebatraditsiooniline ja sellest tulenevalt on sobivaid tooteid väga vähe. Reeglina ei kasutata lokaalse andmebaasi vahendust – andur, kontroller ja täitur on omavahel ühendatud kas otse või siis pilveteenuse vahendusel. Neist teisel juhul toimub juhtimine pilves. MeiePilv on aga üksnes andmebaas ja on eluslaboratooriumi automaatika kontekstis üksnes informatsiooni vahendaja.

Detailsema analüüsi jaoks jäid valikusse kaks seadet / lahendust:

2.1. Danfoss Z-Wave radiaatoriklapi termostaat

See seade on põhimõtteliselt radiaatoriklapp ja ruumikontroller ühes. Seade kasutab Z-Wave raadiosideprotokolli, mis on mõeldud peamiselt koduautomaatikas kasutamiseks. Z-Wave põhineb võrevõrgu tehnoloogial (*mesh-network)*, mis tähendab, et igast võrku installitud seadmest saab signaali kordaja. Selle tulemusena levib võrk seda paremini, mida rohkem seadmeid võrgus on. Seadmed saadavad info sihtpunkti „lainena", hüpetena seadmelt seadmele. [3]



Joonis 2-1: Võrevõrgu tehnoloogial põhineva võrgu illustratsioon [4]

Radiaatoriklapp saab info seadesuuruse kohta Z-Wave kontrollerilt, mis kuvatakse seadme LCD ekraanile ja mis on võimalik seadme noolenuppe kasutades üle kirjutada. [5]

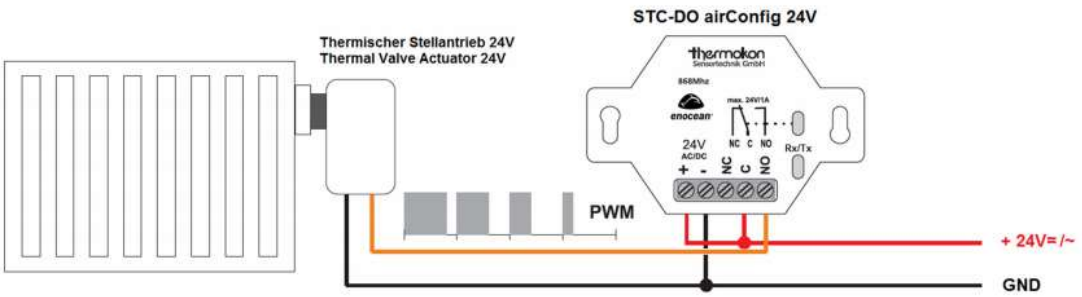
Joonis 2-2: Danfoss Z-Wave radiaatoriklapi LCD ekraan [5]

Seadme probleem püstitatud ülesandest lähtudes on see, et seade ei võimalda klapi asendit MeiePilve logida, samuti ei ole mõistlik kasutada täiesti uut suhtlusprotokolli, kui ruumides on juba olemas EnOcean protokolli kasutavad seadmed – seadmed ei ühildu juba olemasolevasse süsteemi. Lisaks sellele kasutab seade patareitoidet, mis tähendaks, et süsteem vajaks patareide vahetamise näol täiendavat hooldust.

2.2. Thermokon STC-DO 24 V releeväljundiga multi-funktsionaalne transiiver

Joonis 2-3: Thermokon STC-DO 24 V releeväljundiga multifunktsionaalne transiiver [6]

Seade loeb informatsiooni otse ruumikontrollerilt kasutades EnOcean protokolli, ning juhib saadud info põhjal sõltuvalt konfiguratsioonist kas küttesüsteemi, jahutussüsteemi või valgusteid. Seade võimaldab juhtimist nii kahepositsioonilise reguleerimise kui ka PI-kontrolleri ja PWM-väljundina. Seadmele kirjutatakse seadetemperatuur sisse kasutades seadmele spetsiifilist tarkvara. [6]



Joonis 2-4: Seadme ühendusskeem juhtimaks radiaatorit [6]

Kuigi see seade kasutab suhtlemiseks EnOcean protokolli ja kasutab toiteks 24 V automaatikapinget, ei sobi ka see seade ülesande lahendamiseks, kuna seade ei saada välja informatsiooni klapi seisundi kohta. Samuti ei suudaks see seade temperatuuri reguleerida siis, kui mingil põhjusel peaks ruumiregulaator oma töö lõpetama, näiteks siis, kui talvel ruumiregulaatori päikesepatarei piisavalt valgust ei saa.

2.3. Järeldused

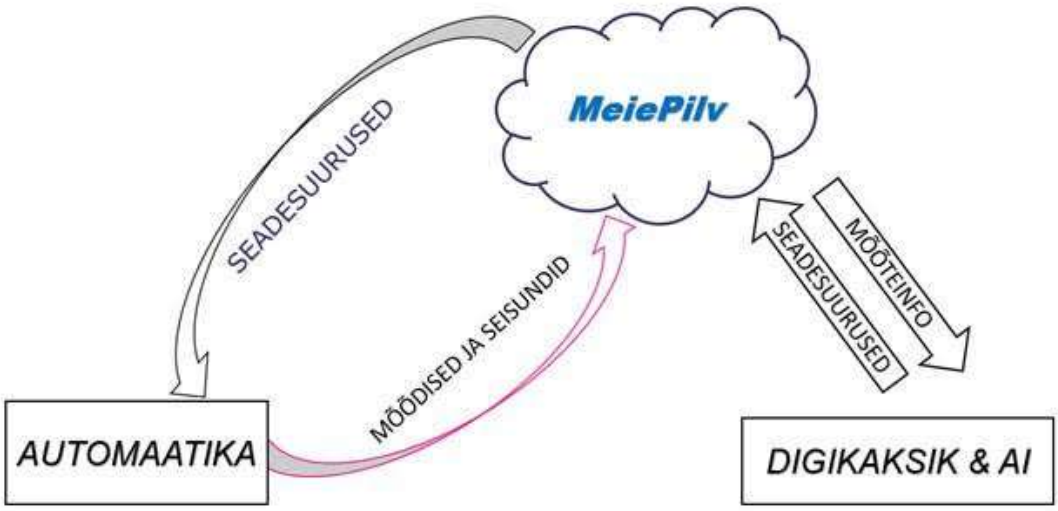
Kumbki toode lähteülesande nõudmistele ei vastanud. Küll aga pälvis tähelepanu Danfoss radiaatoriklapi termostaadi lahendus, kus ruumi temperatuuri juhtimisel kasutatakse klapi enda termomeetrit ja automaatikakontroller annab ette ainult temperatuuri, mida hoida. Seejuures ei eeldata, et klapiajamis paiknev termomeeter oleks täpne. Keskkontroller juhindub ruumiregulaatori poolt registreeritavast temperatuurist ja tõstab või langetab klapiajamile etteantavat hoidetemperatuuri, mis arvuliselt ei pruugi langeda kokku ruumi tegeliku temperatuuriga. Selles kontekstis on klapiajamile etteantav temperatuur vaid suhteline parameeter reguleerimisprotsessis.

Lahendus tervikuna siiski ei sobi. Ülesande lahendamiseks otsustasin teha oma lahenduse ruumi temperatuuri juhtimiseks. Kasutan selles lahenduses Danfossi radiaatoriklapi termostaadi lahendusest pärinevat ideed varustada väljatöötatav juhtseade oma termoanduriga. Termoandur, eriti kui see ei pea olema täpne, on tänapäeval odav ja mõjutab vähe seadme kogumaksumust. Küll aga võimaldab see lahendus paljude võimalike rikkeolukordade puhul vältida ruumi temperatuuri suuri kõikumisi.

3. MEIEPILV

MeiePilv on MONGO DB tarkvaral baseeruv andmebaas, mis on mõeldud peamiselt hooneautomaatikaga seotud mõõtmistulemuste salvestamiseks ja haldamiseks. Server asub spetsiaalselt selleks kokku pandud arvutis C-maja serveriruumis. MeiePilvel on olemas WAN (*wide area network*) IP, mis tähendab, et see on laivõrgus kättesaadav, samuti on serveril oma API ja veebiliides. Veebiliides võimaldab kasutajal sõltuvalt kasutajaõiguste tasemest salvestusprotsessi administreerida, andmeid mingi tunnuse alusel otsida, graafikuid ja tabeleid vaadata ning andmeid CSV formaadis alla laadida. API võimaldab luua rakendusi, mis suudavad serveriga suhelda, sinna andmeid salvestada ja neid sealt lugeda. [7]

MeiePilv on keskne sõlm Eluslabori kontseptsioonis, serverisse püütakse võimalikult palju erinevaid sisekliima- ja seisundinäitajaid pikaks ajaks talletada, selleks, et neid otseselt info kui ka hiljem võrdlusandmetena õppe- ja teadustöös kasutada. Pikemas perspektiivis liidetakse süsteemi veel hoone sisekliima digitaalne kaksik. See on virtuaalne kuvand hoone eelkõige soojusfüüsikalistest omadustest, mis peaks ükshetk võimaldama automaatika juhtimisalgoritme virtuaalselt simuleerida, selleks et neid siluda ja siis hiljem reaalses hoones rakendada.

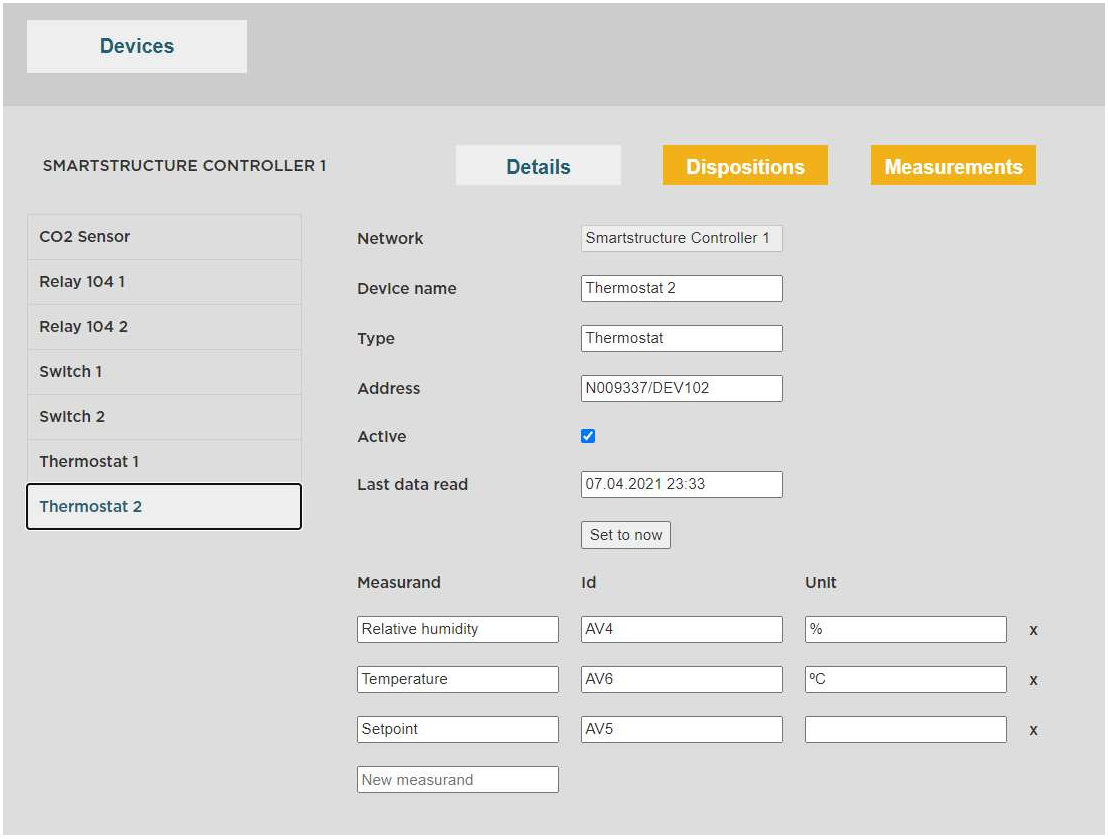


Joonis 3-1: Automaatikasüsteemi, MeiePilve ja digikaksiku omavahelised seosed vastavalt hetkel kehtivale kontseptsioonile

MeiePilv on andmebaas, kuhu kogutakse mõõteandmeid (ilmajaama andmed, sisekliima andmed, automaatika seisundite andmed) aegridadena. Aegridade ajaskaala lähtub serveri kellast, mida sünkroniseeritakse välise ajaserveriga. Kuna salvestatavad

andmed kajastavas suhteliselt aeglaseid protsesse, seotakse andmed ajaskaalaga laekumise hetke või koos andmetega saatetava ajalise nihke kirje kaudu mõõdetuna andmete serverisse laekumise hetkest.

MeiePilvel on võimekus võtta vastu informatsiooni erineva edastusviisiga infoallikatest: TCP/IP, oBIX, LoRa Net ja välised pilveteenused, millest hetkel on kasutuses NetAtmo ilmajaama pilv. Need erinevad edastusviisid eeldavad mõnevõrra erinevaid nii riist- kui tarkvaralisi liideseid. oBIX ja NetAtmo tarbeks on süsteemis eraldi vahevarad, mis tagavad infoedastuse. MeiePilv andmebaasi vaates on need erinevad võrgud, mida eristatakse atribuudi „*Network*" alusel.



Joonis 3-2: Töö käigus kasutatud ruumikontrolleri seadistused MeiePilve veebiliideses

Nii tulevane EnOcean andmesidet jälgiv seade kui ka kavandatav juhtseade ühilduvad MeiePilvega ilma vahevarata ja töötavad vaheserveris (*network*) nimetusega „OmaWiFi", töö käigus EnOcean andmesidet vahendav MPM kontroller töötas vaheserveris „Smartstructure Controller 1" (joonis 3-2). Vaheserverites salvestatakse andmed seadme kaupa, igal seadmel serveris on unikaalne id, mille kaudu saab selle seadme mõõdiseid pärida, andmeid saadetakse serverisse aga kasutades vaheserveri ja seadme nime. Enne, kui serverisse mingit infot saata saab, peab selleks ühte

vaheserveritest vastava seadme konfigureerima. Nii info küsimiseks kui ka saatmiseks peab teadma, milline on antud seadme konfiguratsioon, kuna mõõdised nii tulevad serverist kui ka saadetakse serverisse lihtsalt ajatempli ja numbritena, lisakontekst selle kohta, mida need numbrid tähendavad, eksisteerib ainult MeiePilve serveris endas.

4. TEHNILISE LAHENDUSE KIRJELDUS

Lähteülesande ja sellest tulenevate ülesannete lahendamiseks sobiv seade peaks sisaldama alljärgnevat:

Kuna klapiajamite energiatarve on nii suur (kuni 1,2 A ruumi kohta), peaks seade töötama automaatikasüsteemi 24 V toitesüsteemil. Kuna seadmel on vajalik infovahetus MeiePilv andmebaasiga LAN või Wi-Fi võrgu kaudu ning vajalik on teha arvutusi ja sellest lähtuvalt klapiajameid juhtida, on seadmes vajalik kontroller. Side tagamiseks MeiePilv serveriga eelistan Wi-Fi ühendust kuna see teeb seadme paigutamise mugavamaks. Hoones on kõikjal Wi-Fi leviala. Wi-Fi kasutamine loob eelistuse just Wi-Fi võimekusega kontrolleri valikuks.

Et vältida või vähemalt leevendada peatükis 1 kirjeldatud võimalikke probleeme, peaks olema seadmes ka oma termomeeter. Võtan siin eeskujuks Danfoss klapiajamite lahenduse, kus ajam mõõdab ka ise ruumi õhutemperatuuri. Seade ei pruugi tingimata olla temperatuuri tagasiside info saamiseks parimas kohas, kuid kui tagada seadme temperatuuriandurile piisav ruumi õhu juurdepääs, kajastab seadme poolt registreeritav temperatuur mõningase nihkega ruumi tegelikku temperatuuri. Selle nihke saab välja arvutada ruumiregulaatorilt tuleva temperatuuriinfo kaudu ja seda nihet arvesse võttes saab lühema perioodi jooksul ruumi temperatuuri piisava täpsusega juhtida. (Danfossi klapiajam on keskkütteradiaatori klapi küljes ja töötab sellest hoolimata adekvaatselt). Kui nihe salvestada kontrolleri energiasõltumatusse mällu, on võimalik nii juhtida temperatuuri ka pikema perioodi jooksul, kuigi temperatuuri kõikumine on siis mõnevõrra suurem. Selline lahendus võimaldab ka kohe toitekatkestuse järel juhtida ruumi temperatuuri adekvaatselt.

Kirjeldatud lahenduse tarbeks peab seadmel olema oma termomeeter, mis tuleb paigutada eemale seadme kuumenevatest osadest ja tagada talle õhu juurdepääs. Selleks peaksid korpusel olema õhupilud nii ülemises kui ka alumises osas ja termoandur peaks asuma alaosas õhupilu lähedal. Seadme soojenevad osad tekitavad tõusva õhuvoolu läbi korpuse ja nii asub termoandur sisenevas õhuvoolus.

Kuna mikrokontrollerid ei tööta enamasti rohkem kui 12 V toitepingel nende pingeregulaatori sisendil, siis on seadmes vajalik ka DC – DC muundur, mis muundab 24 V toitepinge madalamaks pingeks.

Klapiajamite töövool on maksimaalselt 1,25 mA, kuid ilmselt võib tootelt tootele mõnevõrra ka varieeruda. Arvestan seadme väljundi koormusvooluks varuga vähemalt 2 A. Võimalikest lülituselementidest (relee või lülitustransistor) valin indutseeritud

kanaliga väljatransistori. Seda on võimalik juhtida otse kontrolleri väljundiga, see ei tee lülitumisel müra ja sellel puuduvad kuluvad liikuvad detailid nagu releel.

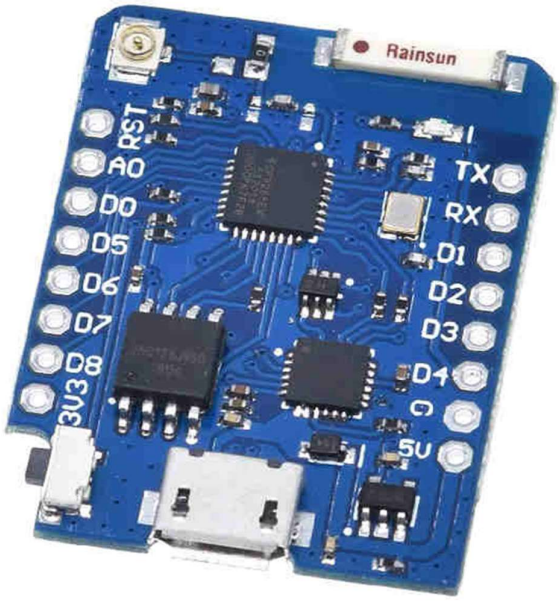
5. PROTOTÜÜP

Tehnilise lahenduse üldisest kirjeldusest lähtuvalt on valitud prototüübi komponendid. Prototüüp koosneb järgmistest sõlmedest: mikrokontrolleri arendusplaat, temperatuuriandur, toitemoodul ja transistorlüliti

5.1. Prototüübi komponentide valik

5.1.1 Mikrokontroller - ESP8266 D1 mini

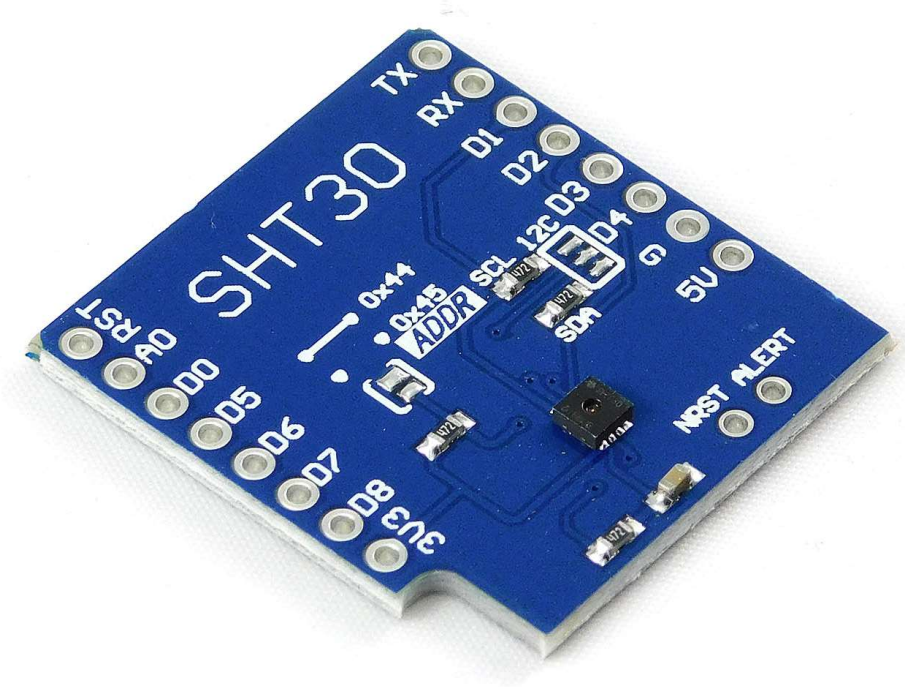
ESP8266 on väike, kuid suhteliselt võimekas mikrokontroller. Kontrolleri valikul olid kõige olulisemad parameetrid Wi-Fi võimekus ja madal energiatarbe (ja seeläbi vähene soojuse eraldumine). Üheks faktoriks valikul tegemisel oli ka minu enda varasem kogemus kontrolleri ja selle tarkvara ja programeerimiskeele ja -keskkonnaga (C++ ja Arduino IDE-ga). Nendest omadustest lähtudes jäi valik ESP8266 ja tema edasiarenduse ESP32 vahele. ESP8266 osutus valituks seetõttu, et vanema seadmena eksisteerib sellele rohkem, pikemalt arendatud ja tänu sellele töökindlamaid teeke. ESP8266 D1 mini modifikatsioonile on olemas sellele omane toitemoodul, mida prototüübis kasutan. ESP32 suurem võimekus ei ole antud ülesande juures vajalik ega oluline. WeMos D1 mini modifikatsioon valisin, kuna see võimaldab kõik vajalikud abiplaadid omavahel pistikliitmikega kokku panna ning seeläbi luua kompaktse prototüübi ja lihtsustada oluliselt prototüüpimise protsessi. [8]



Joonis 5-1: ESP8266 D1 mini mikrokontroller [9]

5.1.2 Temperatuuriandur - SHT30-DIS-B

SHT30 on digitaalne niiskus- ja temperatuuriandur. Vahemikus 0 – 65 °C mõõdab see temperatuuri täpsusega ±0,3 °C, toitepinge on vahemikus 2,15 – 5,5 V. Suhtlemiseks mikrokontrolleriga kasutab andur I²C protokolli. Temperatuurianduri valikul lähtusin nii anduri täpsusest kui ka ühilduvusest mikrokontrolleri ja toitemooduliga – toodetakse seda temperatuuriandurit kasutavat plaati, mis on osa WeMos D1 tooteperest, mida ka kasutasin. Faktoriks oli ka see, et see temperatuuriandur oli koos mikrokontrolleriga koolis olemas. ESP8266-le on olemas lihtsasti kasutatav SHT30 teek, mis teeb selle anduri kasutamise mugavaks. [10]

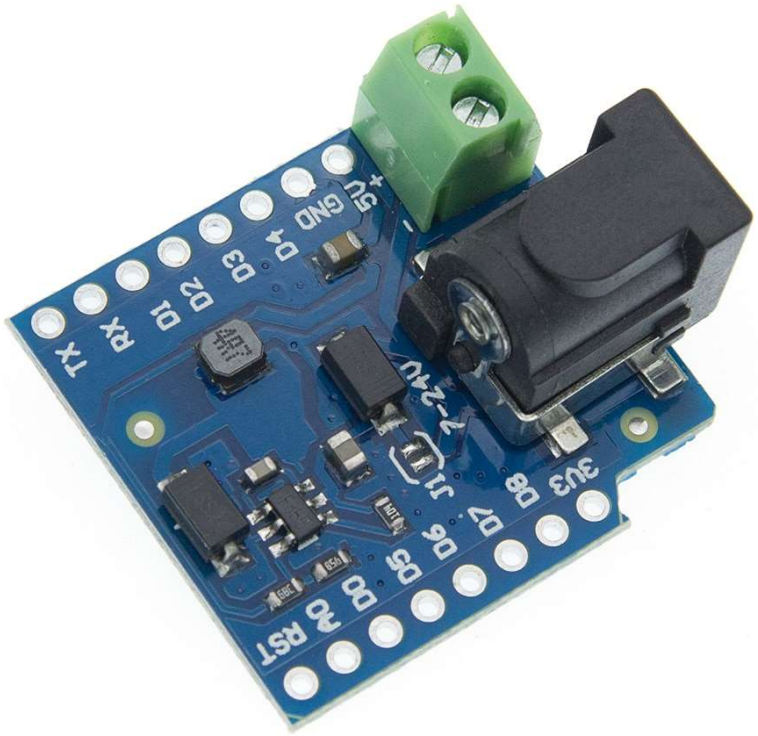


Joonis 5-2: SHT30 temperatuuriandur [11]

5.1.3 Toitemoodul - DC Power Shield V1.1.0

DC Power Shield V1.1.0 on pingeregulaator, mis on lahendatud DC – DC muundurina. Valitud moodul võimaldab sisendpingeid 7 – 24 V DC muundada ESP8266 arendusplaadi toitepingeks (5 V). DC – DC muundurite kasutegur on kõrge (u' 95 %), mis võimaldab peaaegu viiekordset pingelangetamist teha väga väikese energiakaoga ja tänu sellele ka madala soojaeraldusega. Edasine pinge langetamine ESP8266 toitepingeni (3,3 V) tehakse WeMos D1 arendusplaadil pidevstabilisaatoriga. Kokkuvõttes kujuneb prototüübile väga hea kindlus toitepinge kõikumisele ja häirekindlus toite kaudu üle kanduvatele häirepingetele. [12]

Seadme väljundvool on kuni 1 A, mis on enam kui piisav, arvestades seda, et ESP8266 kõrgeim voolutarve käivitumisel on ~350 mA, ja voolutarve töö käigus ~100 mA [12] [13]. Seade sai valitud, kuna osana WeMos D1 tooteperest on see seade otseselt loodud ESP8266 D1 mini jaoks.



Joonis 5-3: DC Power Shield V1.1.0 pingeregulaator [14]

5.1.4 Transistorlüliti - IRF520 MOSFET

IRF520 on isoleeritud paisuga võimsusväljatransistor. Seade suudab lülitada kuni 9,2 A püsivat voolu ja opereerib tööpingetel kuni 100 V. Sellel väljatransistoril on madal lävipinge: 1,5 V, mis tähendab, et mikrokontrolleri 3,3 V GPIO pin suudab seda töökindlalt lülitada, seega kasutatakse seda seadet tihti mikrokontrollerite väljundiga suurevõimsuseliste voolutarbijate juhtimiseks. Ajami lülitamiseks valiti transistorlüliti seetõttu, et erinevalt releelülitist ei kulu see lülitamisel, seega ei pea juhtalgoritmi disainil lülituste arvu pärast muretsema ning seeläbi saab temperatuuri ruumis täpsemini juhtida. [15]

Joonis 5-4: 0-24 V isoleeritud paisuga võimsusväljatransistormoodul IRF520 MOSFET [16]

5.1.5 Termoelektriline ajam - TWA-A NC 24 V

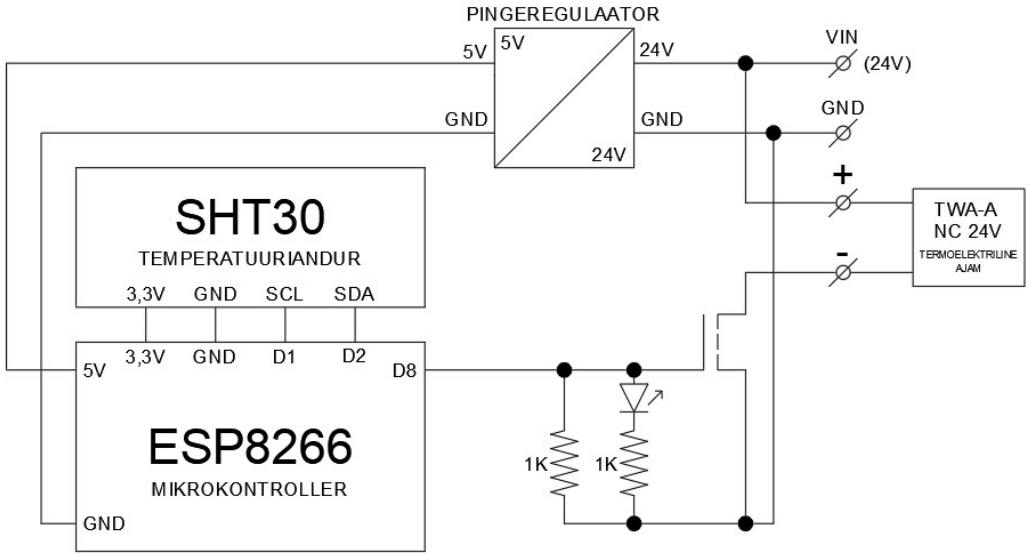
TWA-A NC 24 V on väike termoelektriline kahepositsiooniline ajam, mis sobib mitut tüüpi ventiilide ja põrandakütte kollektorite juhtimiseks. Seadme nimipinge on 24 V, võimsustarve 2 W, spindli liikumise aeg ~3 minutit. Ajam on ette nähtud normaalselt suletud klappide juhtimiseks, see tähendab et klapp avaneb siis, kui ajamile rakendada juhtpinge. Ajamil on visuaalne indikaator näitamaks, mis positsioonis klapp on. [17]

Ajam otseselt prototüübi koosseisu ei kuulu, kuid on siin välja toodud eelkõige sobitumise aspektist käsitlemiseks.

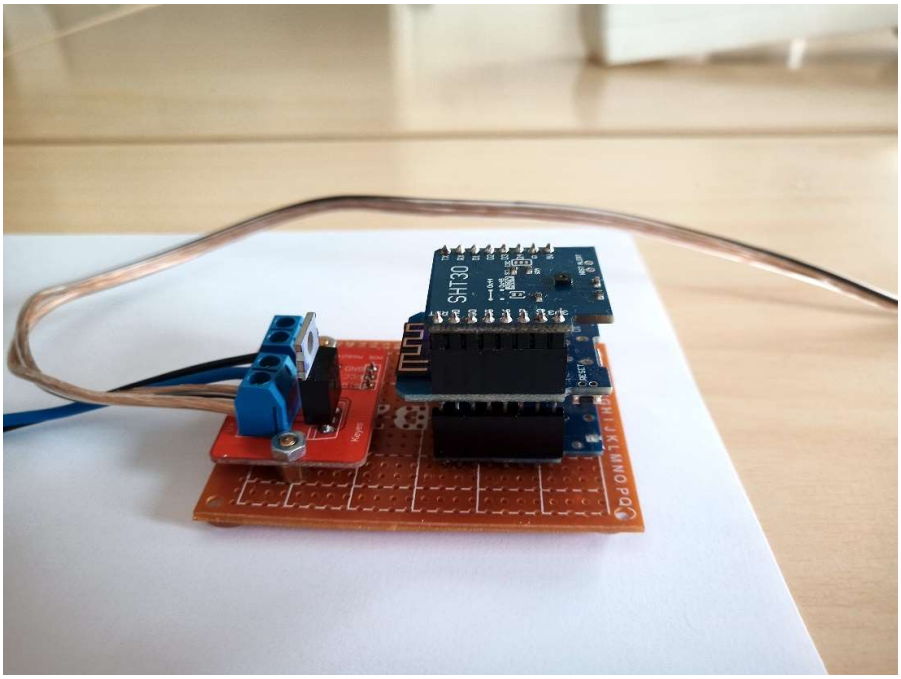
Joonis 5-5: TWA-A NC 24 V termoelektriline ajam [18]

5.2. Prototüübi koostu

Kuna ESP8266 arendusplaat ja toite- ning SHT laiendusplaadid on valitud samast tooteperest, mis on ette nähtud omavahel ühendada pistmikega, siis osutus prototüübi koostu suhteliselt lihtsaks.



Joonis 5-6: Prototüübi elektroonikaskeem



Joonis 5-7: Prototüüp kokku pandunda

Kahjuks prototüübile korpust luua ei jõudnud, peamiselt koroonaviirusega seotud piirangute tõttu. Korpuse loomisel ja paigaldamisel peaks lähtuma järgmisest:

- Korpus peab võimaldama hea õhu läbivoolu, eriti üle temperatuurianduri, vastasel juhul võib mikrokontrolleri poolt eraldatud soojus anduri näitu ruumi tegelikust temperatuurist mööda kallutada.

- Prototüüp peaks paiknema korpuses nii, et temperatuuriandur oleks võimalikult lähedal õhupiludele, kust õhk korpusesse sisse voolab (Mikrokontrolleri poolt eraldatav soojus tekitab korpuses kerge õhuvoolu).

- Korpust ei ole soovitatav paigaldada automaatikakilpi, kuna kilp isoleerib seadme ümbritsevast keskkonnast ja kuna kilbis on ka soojenevaid komponente, siis korreleerub temperatuur kilbis ruumi üldise temperatuuriga liiga vähesel määral.

- Korpus peaks paiknema vähemalt 1 meetri kaugusel lähimast radiaatorist, siseseinal, soovitatavalt kohal, kus päike sellele peale ei paista.

6. PROGRAMM

Kontrollerile ESP8266 koostasin juhtprogrammi, mis tagab seadme töötamise. Sobivat valmis programmilahendust leida ei õnnestunud. Küll aga kasutasin alljärgnevaid vabavaralisi teeke:

- ESP8266WiFi.h – teek, millest pärinevad ESP8266 Wi-Fi'ga seotud funktsioonid.
- ESP8266HTTPClient.h – teek, mis lubab käivitada ESP8266 peal http kliendi – vajalik MeiePilv serveriga suhtlemiseks.
- ArduinoJson.h – teek, mille abil saab koostada ja lahti mõtestata JSON stringe – vajalik MeiePilve saadetava info formeerimiseks ja MeiePilvest loetava info tõlgendamiseks.
- WEMOS_SHT3X.h – temperatuurianduri SHT30 teek, mille abil saab lugeda temperatuurianduri näitu.
- WiFiUdp.h – teek, mis võimaldab saata ja vastu võtta pakette üle UDP – vajalik ajaserveriga suhtlemiseks.
- TimeLib.h – võimaldab lihtsalt muundada UTC ajatempli kergesti loetavasse ajaformaati – lihtsustab oluliselt aastaaja kontrollimist.
- EEPROM.h – teek, millest pärinevad ESP8266 EEPROM'ga seotud funktsioonid – vajalik andmete kontrolleri energiasõltumatusse mällu salvestamiseks.

6.1. Juhtalgoritmi valik

Kuna proportsionaalse juhtimisega klapiajameid kasutatakse reaalselt väga harva - need on kallid ja Puiestee 80A kahes ruumis juba on kahepositsioonilised termoelektrilised klapiajamid olemas - keskenduti ka prototüübi loomisel kahepositsioonilisele variandile. Otsus kahepositsioonilise kontrolleri kasuks langetati järgmistel põhjustel:

- praeguses keskküttelahenduses kasutatakse vabalt varieeruva temperatuuriga keskküttevett;
- olemasolevad, tänaseks tootmisest maha võetud Danfoss RTD klapid on väga ebalineaarsed (läbivooluava pindala sõltuvus spindli nihkest);

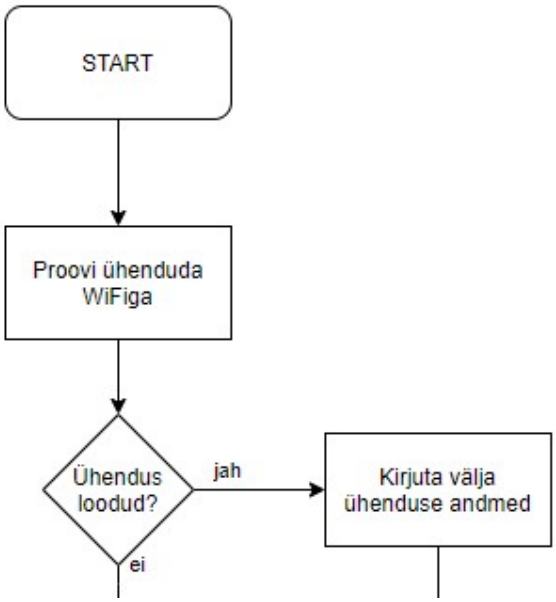
- PID konstantide seadistamiseks oleks olnud tarvis kasutajakonsooli, mis teeb lahenduse märksa kallimaks;
- klapiajamid on väga aeglased;
- katsed näitasid, et temperatuuri stabiilsus on saavutatav ka ilma PID juhtimiseta.

6.2. Programmi töö selgitus

Esimese sammuna programmi käivitamisel initsialiseeritakse vajalikud globaalmuutujad ja konstandid:

- võrguühendusega seotud konstandid – Wi-Fi SSID ja parool, MeiePilve serveri IP ja port, MeiePilve seadmete ID-d;
- klappide seisu andmete asukoht MeiePilves;
- ajaserveriga seotud muutujad ja konstandid – ajaserveri IP ja port, puhver tulevate ja väljuvate pakettide jaoks;
- temperatuuri juhtimise, ajaarvestamise ja andmete salvestamisega seotud muutujad;
- ESP8266 arendusplaadi sisendite ja väljundite seadistused.

Seejärel tulevad algseadistused, kus pannakse paika sisendite ja väljundite töörežiimid, alustatakse järjestikühendus kiirusega 115200 bitt/s ja tehakse esmane katse ühenduda Wi-Fi'ga. Kui Wi-Fi'ga ühendumine õnnestub, kirjutatakse Wi-Fi andmed välja Arduino IDE järjestikmonitori ja lülitatakse sisse ESP8266 sisseehitatud LED – see annab märku, et Wi-Fi ühendus on olemas.

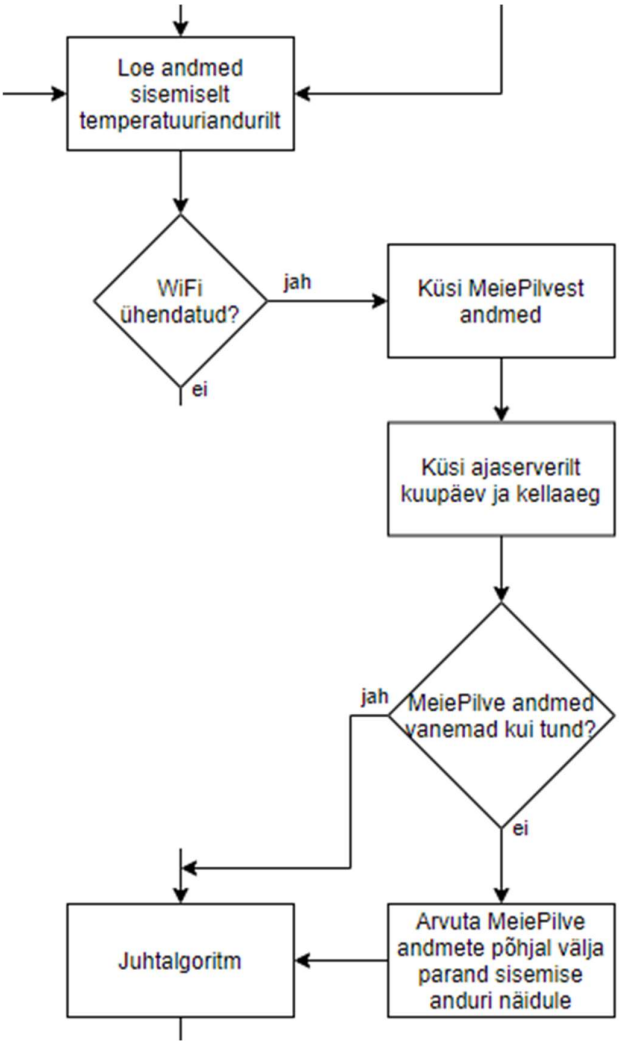


Joonis 6-1: Koodi esmane käivitumine

Järgmisena algab programmi põhiosa, kus iga 10 sekundi järel loetakse kontrolleriga ühendatud füüsilise temperatuurianduri näitu. Seejärel kontrollitakse Wi-Fi ühendust. Kui ühendus on olemas, siis jätkatakse tööd Wi-Fi režiimis - MeiePilve serverile saadetakse kaks päringut:

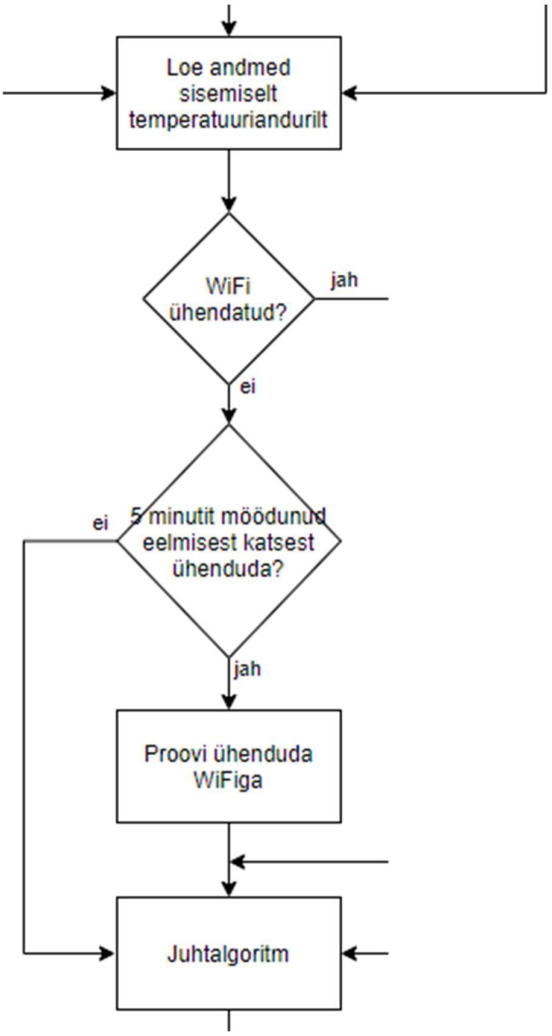
- temperatuuri ja ruumiregulaatoril oleva temperatuurireguleerimise nupu kohta;
- seadetemperatuuri kohta.

Siis saadetakse ajaserverile päring kuupäeva ja kellaaja kohta, ning võrreldakse MeiePilve esimese päringu andmetega kaasa tulnud ajatemplit ajaserverist saaduga. Kui MeiePilve andmed on vähem kui tund aega vanad, siis arvutatakse nende põhjal välja parand kontrolleriga ühendatud füüsilise temperatuurianduri näidule ning kirjutatakse see kontrolleri energiasõltumatusse mällu. Kui aga MeiePilve andmed on üle tunni aja vanad, siis neid ei kasutata ja töö jätkub viimase arvutatud parandiga.



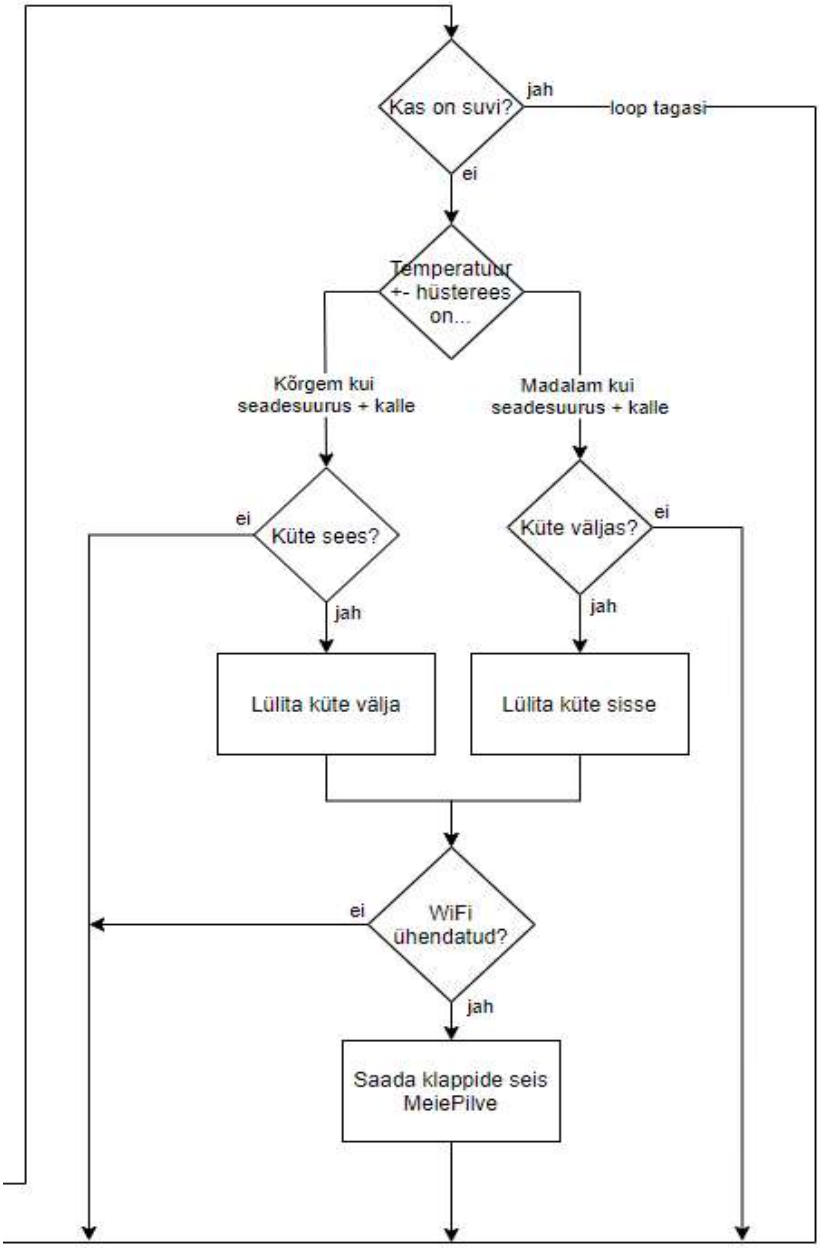
Joonis 6-2: Koodi töö Wi-Fi-režiimis

Juhul kui Wi-Fi-ühendust ei ole suudetud luua, siis jätkatakse tööd võrguühenduseta režiimis: parand loetakse energiasõltumatust mälust ja kontrollitakse, palju aega on möödunud eelmisest katsest Wi-Fi-ühendust luua – kui on möödunud rohkem kui viis minutit, siis proovitakse ühendust uuesti luua.



Joonis 6-3: Koodi töö võrguühenduseta režiimis

Peale temperatuuriandmete kätte saamist liigutakse edasi juhtalgoritmi juurde. Kuna suvel on vesi radiaatorites külm, siis esimese sammuna kontrollitakse ajaserverilt saadud ajatempli põhjal, kas on suvi. Kui on, siis kütet sisse ei panna. Kui aga on muu aastaaeg, siis võrreldakse (hüstereesiga) temperatuuri seadetemperatuuriga. Kui temperatuur on seadetemperatuurist kõrgem ja küte on sees, siis lülitatakse küte välja, kui aga temperatuur on seadetemperatuurist madalam ja küte on väljas, siis lülitatakse küte sisse. Juhul, kui klappide seisus on toimunud muutus, saadetakse uued andmed Wi-Fi-ühenduse olemasolul MeiePilve serverisse.



Joonis 6-4: Juhtalgoritm

Programmi töö täielik plokkskeem on toodud lisas 1.

6.3. Suhtlus MeiePilv serveriga

Käesoleva töö käigus loodav regulaator saab enamiku sisendsuurustest, mille alusel ta klapiajameid juhib, MeiePilvest. Vastavalt lähteülesandele on vajalik ka klapiajami juhtväljundi seisundimuutuste salvestamine MeiePilv andmebaasi. Seetõttu on

regulaatori juhtprogrammist oluline osa seotud päringute tegemisega MeiePilvest ja andmete saatmisega MeiePilve. Infovahetus toimub MeiePilv API vahendusel.

6.3.1 MeiePilves andmete pärimine

MeiePilvest andmete pärimine käib vastava seadme ID kaudu. Andmete küsimiseks saadetakse http päring aadressile serveris, kus asub küsitav informatsioon. Näiteks ühe serveris oleva seadme viimase mõõdistuse küsimiseks saadetakse http: GET päring MeiePilve WAN aadressi portile 82, asukohta /measurements/last/ + selle seadme ID serveris. Serveri vastus tuleb JSON formaadis, see sisaldab sõltuvalt küsitud seadmest ühte või enamat andmevälja ja UNIX formaadis ajatemplit, mis vastab sellele ajahetkele, mil saadetud andmed serverisse kirjutati.

```
310 // SERVERIST ANDMETE LUGEJA
311 //=========================
312 // Küsib serverist ühe seadme viimased andmed
313
314 String LoeServerist(String id) {
315   String payload;
316
317   HTTPClient http;                          // Käivitan HTTP kliendi
318
319   http.begin("http://193.40.13.86:82/measurements/last/" + id);  // Määran andmete sihtkoha
320   int httpCode = http.GET();                // Saadan päringu
321
322   if (httpCode > 0) {                       // Kontrollin, et vastuskood poleks 0 või -1
323     payload = http.getString();             // Loen vastuse stringina maha
324     Serial.print("Serverilt saadud info: ");
325     Serial.println(payload);                // Kirjutan vastuse välja
326   }
327   http.end();                               // Sulgen ühenduse
328   return payload;
329 }
```

Joonis 6-5: funktsioon, mis küsib MeiePilvest ühe seadme viimase mõõdistuse

6.3.2 MeiePilve info saatmine

MeiePilve info saatmine käib läbi vaheserveri- ja seadme nime, seadme ID-d andmete saatmiseks teadma ei pea. Andmete saatmiseks peab saatma http: POST päringu MeiePilve WAN aadressi portile 82, asukohta /measurements/objects, peale päringu saatmist saadetakse JSON formaadis string, mis sisaldab nii saadetavat informatsiooni kui ka seda, kuhu MeiePilves see informatsioon pannakse – vaheserveri- ja seadme nime.

```
// SERVERILE ANDMETE EDASTAJA
//============================
// Argumendis "info" sisaldub edastatava POST pöördumise BODY
// Funktsiooni enda väärtus on TRUE kui server annab vastuse 201 ja
// FALSE kui server edastab veateate. Mitmesugused pöördumise andmed võtab
// see funktsioon programmi pääisest ehk siis üldmuutujatest

boolean Kirjatuvi(String info)
{
  int httpVastuskood = 0;                  // muutuja http vastuskoodile
  boolean abi = false;                     // abimuutuja vaikimisi valeks

  HTTPClient http;                         // ... käivitan http kliendi
  http.begin(host, port, sUrl);            // määran sihtkoha (oluline kirje)

  // Sean andmesisuks text/plain - edastatav info on tekst
  http.addHeader("Content-Type", "text/plain");
  httpVastuskood = http.POST(info);        // Muutujas "info" on tekst mida edastada
  Serial.println (httpVastuskood);
  if (httpVastuskood == 201)               // kui serveri vastuskood on 201, siis ...
  {
    abi = true;                            // saatmine õnnestus ja funktsioon saab olema true ...,
  }
  // ... muidu mitte
  http.end();                              // Vabastan serveri
  return abi;                              // funktsiooni väärtuseks abimuutuja väärtus
};                                         // funktsiooni lõpp
```

Joonis 6-6: funktsioon, mis saadab MeiePilve http päringu

See funktsioon pärineb Taltech Tartu kolledži õppeaine „Mikroprotsessorsüsteemid" materjalidest.

6.4. Anomaalsete situatsioonide välistamine

Käesoleva töö alajaotuses 1.2 käsitletud võimalike tõrkepõhjuste leevendamine või vältimine on samuti tarkvaraline. Järgnev käsitlus on toodud võimalike tõrkepõhjuste kaupa.

6.4.1 Tõrked andmevahetusel MeiePilv serveriga

Juhul, kui MeiePilv serverist pole võimalik ruumiregulaatori andmeid kätte saada Wi-Fi ühenduse katkemise tõttu, peab kontroller olema valmis töötama ka ilma ruumiregulaatori andmeteta, kasutades temperatuuri reguleerimiseks prototüübi enda temperatuuriandurit. Selleks on programmis loodud kaks eraldi režiimi:

- võrguühendusega režiim;
- võrguühenduseta režiim.

Võrguühendusega režiimis kasutatakse parandit, mis on välja arvutatud MeiePilvest saadud temperatuuri ja prototüübi temperatuurianduri näidu vahest. Võrguühenduseta

režiimis kasutatakse parandit, mis arvutati siis, kui viimati saadi MeiePilves ajakohased andmed – see loetakse mikrokontrolleri energiasõltumatust mälust.

6.4.2 Tõrked/iseärasused ruumiregulaatori andmete MeiePilv andmebaasi kandmise süsteemis

Kuna pole välistatud olukord, kus ruumiregulaator lõpetab andmete saatmise MeiePilve, oli vaja programmis valideerida MeiePilve tulevate andmete ajakohasust. Nende andmete ajakohasust valideeritakse kasutades NTP (võrguaja protokoll, ing. k. *Network Time Protocol*) serverit. NTP on võrgusuhtlusprotokoll, mida kasutatakse võrgu klientide vahel kella sünkroniseerimiseks koordineeritud universaalajaga (UTC). Kontroller saadab serverile paketti, millega küsib serverilt praegust ajahetke. Sellele vastab server pakettiga, mille baidid 40 – 43 näitavad seda, mitu sekundit on möödunud ajahetkest 00:00 01.01.1900. See muundatakse UTC ajatempliks, mida on võimalik võrrelda MeiePilvest tulnud ajatempliga. Kui nende vahe on liiga suur (rohkem kui üks tund), siis neid andmeid ei kasutata.

6.4.3 Toitekatkestused

Kuna toitekatkestuse puhul võib kuluda mõni aeg peale toite taastumist selleks, et suhtlus ruumiregulaatori ja MeiePilve vahel taastuks, siis oleks hea, kui kontroller suudaks peale sisselülitamist alustada tööd kohe ilma ruumiregulaatori andmeteta. Selleks kirjutatakse programmis parand ESP8266 energiasõltumatusse EEPROM mälusse, mis säilib ka peale kontrolleri väljalülitamist, ning loetakse sealt pärast toite taastumist uuesti sisse.

7. KATSETAMINE

Prototüüpi katsetasin jooksvalt töö käigus, katsetamise eesmärgiks oli teha kindlaks, et kontroller suudab temperatuuri hoida normi piires ka rikete tekkimisel. Prototüüpi kokku pannes katsetasin funktsionaalsust ühe kaupa, viimased katsed viidi läbi reaalselt ruumis A203.

7.1. Sidekatkestuse kontroll

Esimeseks katseks oli kontrolleri sidekatkestuse kontroll. Katse eesmärgiks oli teha kindlaks, et kontroller:

- alustab tööd võrguühenduseta režiimis juhul, kui programmis sätestatud Wi-Fi võrku ühendumine ei õnnestu;
- läheb üle võrguühenduseta režiimi juhul, kui programmi töö ajal Wi-Fi-ühendus katkeb;
- suudab peale sidekatkestust Wi-Fi-ühenduse uuesti luua.

Alguseks proovisin kontrollerit ühendada võrku, mida polnud olemas, tulemuseks oli ootuspäraselt see, et kontrolleril ei õnnestunud Wi-Fi-ühendust luua ning see alustas tööd võrguta režiimis. Seejärel ühendasin kontrolleri nutitelefoni hot-spotiga, ootasin, kuni kontroller Wi-Fi-režiimis tööle hakkas, ja siis lülitasin nutitelefonist hot-spoti välja, mille tulemusel kontroller läks õigesti üle võrguta režiimi. Viimaseks lülitasin nutitelefoni hot-spoti sisse tagasi, ning kontroller ühendus Wi-Fi'ga uuesti.

Katse oli õnnestunud, kontrolleri muutis töörežiimi õigesti ja suutis peale Wi-Fi ühenduse katkemist uuesti ühenduse luua.

7.2. Andmete valideerimise kontroll

Teiseks katseks oli andmete valideerimise kontroll. Kontrolleri juhtprogrammis toimub MeiePilv serverist loetud andmete ajakohasuse kontroll. Sellega hoitakse ära juhtimine aegunud andmete põhjal. Katse eesmärgiks oli teha kindlaks, et kontroller:

- kasutab andmeid, mis on ajakohased;
- ei kasuta andmeid, mis pole ajakohased.

Selle testimiseks lasin kontrolleril alguses pärida temperatuuriinfo selliselt seadmelt MeiePilves, mille viimane mõõdis oli ajakohane (vähem kui tund aega vana), selle võttis kontroller vastu. Seejärel lasin kontrolleril pärida info seadmelt, mille mõõdised ei olnud viimase tunni jooksul MeiePilve jõudnud, neid andmeid kontroller vastu ei võtnud ja läks ootuspäraselt üle võrguühenduseta juhtimise režiimi.

Katse oli õnnestunud, kontroller võttis vastu ainult ajakohaseid andmeid.

7.3. Termoelektrilise ajami testimine

Kolmandaks katseks oli termoelektrilise ajami test. Katse eesmärgiks oli teha kindlaks, et klappide lülitus töötab ja saada ettekujutus sellest, kui kiiresti ajam klapi avab/sulgeb.

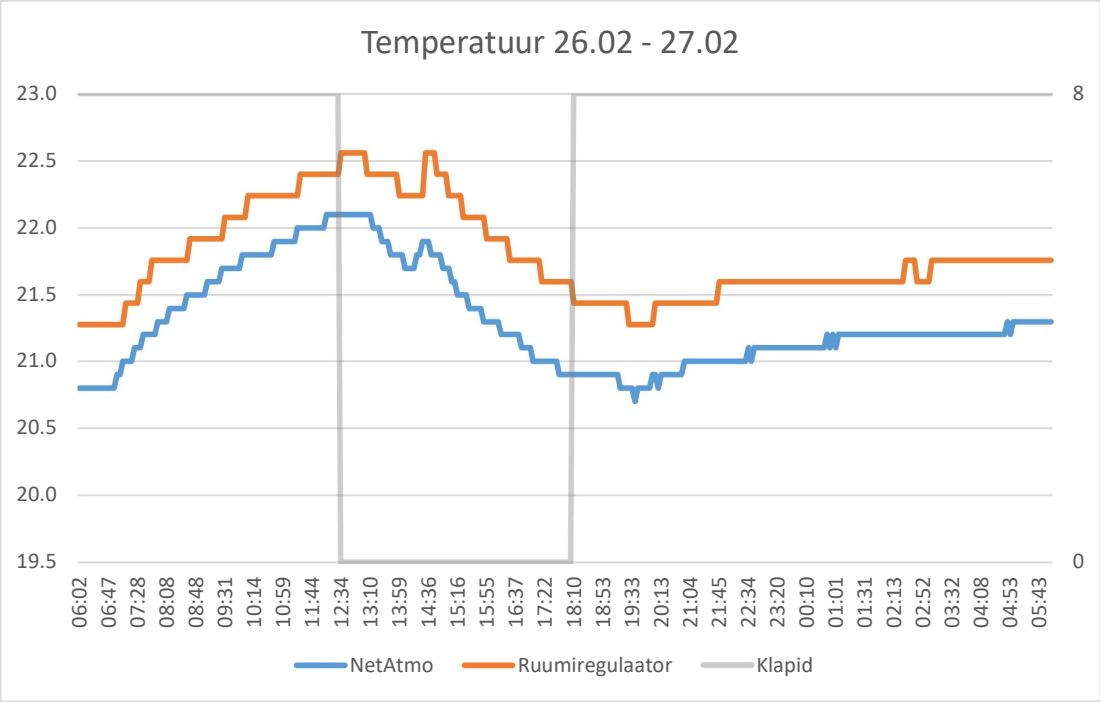
Selleks ühendasin portotüübi ajami ja automaatikasüsteemi toiteplokiga ning modifitseerisin kontrolleri koodi nii, et juhtpinge oleks alati sees. Ajam reageeris pingele aeglaselt, kinnisest algolekust täielikult lahtisesse olekusse liikumiseks kulus ajamil umbes 15 minutit, seejuures tarbis ajam ~0,085 A voolu, mis ühtib seadme tehnilise spetsifikatsiooniga ($\frac{2\ \mathrm{W}}{24\ \mathrm{V}} = 0{,}0833\ \mathrm{A}$) [17]. Seejärel modifitseerisin kontrolleri koodi uuesti, seekord nii, et juhtpinge oleks alati väljas. Ajam liikus umbes sama aeglaselt kinnisesse olekusse tagasi.

Katsest selgus, et ajami käik on suhteliselt aeglane, see aga tähendab, et tuleb arvestada sellega, et ajamil kulub suhteliselt pikk aeg juhttoimele reageerimiseks. Samuti võimaldab selline aeglane käik pseudosujuvjuhtimist, mis tähendab, et kuigi ajam on kahepositsiooniline, on seda võimalik hoida ka vahepealsetes positsioonides.

7.4. Prototüübi töö reaalses keskkonnas

Neljandaks katseks oli prototüübi töö kontrollimine reaalses keskkonnas. Katse eesmärgiks oli teha kindlaks, et prototüüp vastab sellele seatud nõuetele ja katse tulemuste põhjal uurida, mida võiks teha teisiti/paremini. Katse viisin läbi ruumis A203, sest ruum on väike, seal ei toimunud õppetööd ja ruumis asus töötav ruumiregulaator, mille mõõdised aktiivselt MeiePilve saadeti. Ruumis on kaks radiaatorit. Esiteks ühendasin prototüübi ruumi radiaatoriklappe juhtivate ajamitega ning testisin, et need töötaks. Kui olin kindel, et ajamid ja klapid töötavad nii, nagu peaks, ühendasin prototüübi kooli Wi-Fi'ga ja jätsin selle ruumi 24ks tunniks tööle. Prototüübi seadesuuruseks oli koodi püsiprogrammeeritud 22 °C, juhtalgoritm lubas hüstereesi 0,5 °C. Temperatuur väljas oli katse jooksul aastaaja kohta suhteliselt kõrge, 26. veebruari päeval 7 °C ja öösel 0 °C. Katse jooksul logiti MeiePilve andmed klappide

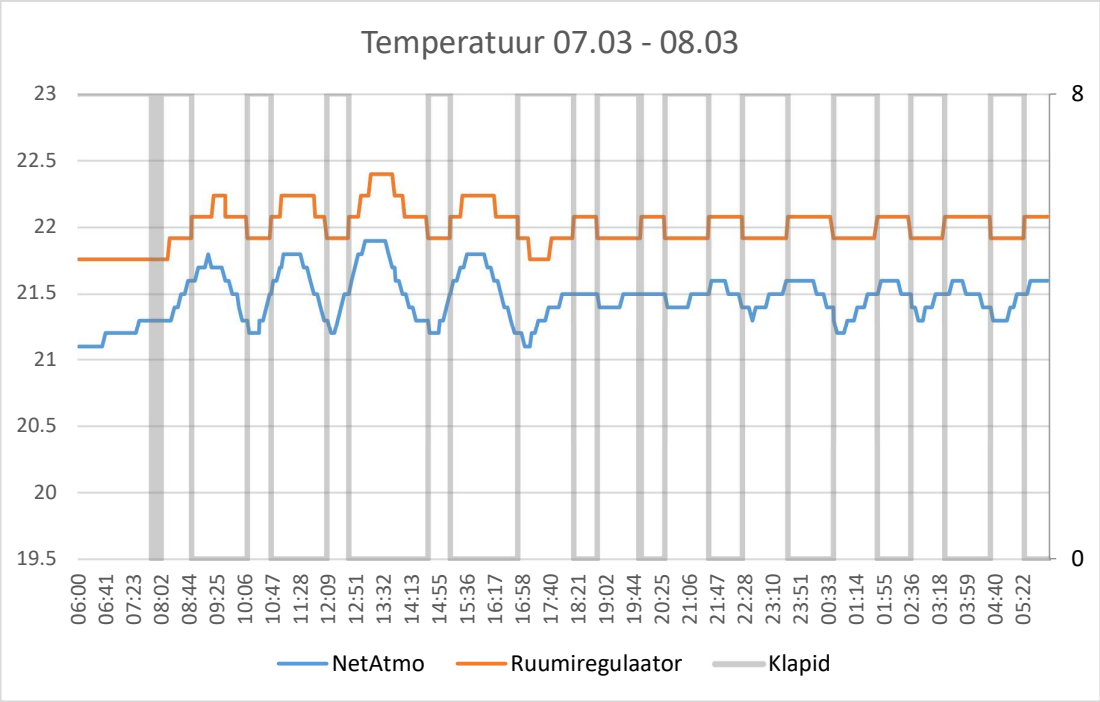
oleku kohta, samuti mõõtis ruumi temperatuuri eraldi NetAtmo temperatuuriandur. Kahjuks seoses Covid 19–ga ei toimunud hoones õppetööd ja energia kokkuhoiu eesmärgil kõigutati küttevee temperatuuri ööpäeva lõikes: öösiti oli see mõnevõrra madalam.



Joonis 7-1: Temperatuuriandurite näit [ºC] ruumis A203 ajavahemikus 26.02.2021 06:00 kuni 27.02.2021 06:00

Jooniselt saab välja lugeda, et ruumiregulaator näitab umbes 0,5 °C kõrgemat temperatuuri, kui NetAtmo temperatuuriandur. Klapid lülituvad ümber harva, 24 tunni jooksul tehakse vaid kaks lülitust. Katsest järeldasin, et prototüüp hoiab suhteliselt hästi temperatuuri (±0,6 °C seadesuurusest), aga tuleks kontrollida ruumiregulaatori info õigsust ja katsetada, kuidas käitub prototüüp siis, kui sellele lubada väiksem hüsterees.

Viies katse sarnanes suuresti neljanda katsega, ainuke muutus oli see, et juhtalgoritmis muudeti hüstereesi. Katse eesmärgiks oli testida prototüübi tööd väiksema hüstereesiga ja uurida, kas seeläbi saab temperatuuri kõikumist vähendada. Seadesuuruse sai sellel katsel kontroller juba MeiePilvest, see püsis muutumatuna 22 °C, juhtalgoritmi hüsterees oli seatud 0 °C. Temperatuur väljas oli 07. märtsi päeval -1 °C ja öösel kuni -8 °C.



Joonis 7-2: Temperatuuriandurite näit [ºC] ruumis A203 ajavahemikus 07.03.2021 06:00 kuni 08.03.2021 06:00

Jooniselt on näha, et temperatuur ruumis püsib seadesuurusele lähemal, kui eelmisel katsel, ruumiregulaatori näidu järgi kõigub temperatuur vähem kui ±0,5 °C seadesuurusest, suurem osa ajast püsib temperatuur vahemikus 22 ± 0,25 °C. Klapid teevad väga palju ümberlülitusi, valdav osa ajast lülituvad klapid ümber kord iga poole tunni tagant, kohati aga ka mitu korda minutis. Kuna klappe juhitakse transistorlülitiga, mille eluiga ei sõltu lülituste arvust, siis pole suur lülituste arv probleem, seega võib sellest katsest järeldada, et antud olukorras sobib ilma hüstereestia juhtalgoritm paremini.

KOKKUVÕTE

Tartu kolledži poolt arendatav eluslaboratoorium on keskendunud praeguses arengufaasis eelkõige Puiestee 80A õppehoone sisekliimaautomaatikale. Automaatikas tagasisidena kasutatavad sisekliimaandmed ja automaatika enda seisundid kogutakse aegridadena MeiePilv nimelisse andmebaasi. Pikemas perspektiivis on planeeritud arendada Puiestee 80A digikaksikuid, millest käesolevat tööd puudutab eelkõige hoone soojustehnilisi näitajaid ja tehnosüsteemi käsitlev osa. Digikaksik võimaldab uusi lahendusi virtuaalselt läbi mängida ja alles siis rakendada algoritme füüsikalises maailmas.

Käesoleva töö eesmärk tuleneb vajadusest täitursõlmede järele, mis toimiksid MeiePilv andmebaasi logitavate jooksvate tagasisideandmete ja digikaksiku poolt sinna kirjutatavate seadesuuruste põhjal.

Tarvis oli välja töötada seade, mis juhib konkreetse ruumi temperatuuri kasutades MeiePilve salvestatud seadetemperatuuri ja tagasisidena ruumiregulaatori poolt samuti MeiePilv andmebaasi jooksvalt edastatavat temperatuurinäitu ja ruumi kasutaja poolt tehtavat kitsas piiris +/- häälestust.

Seadme jaoks kujundasin tehnilise lahenduse, mille komponentideks on mikrokontroller, temperatuuriandur, toitemoodul ja transistorlüliti. Seadmes eraldi temperatuurianduri kasutamine võimaldab ka võrguühenduse puudumisel juhtida ruumi temperatuuri adekvaatselt. Transistorlülitit kasutan, kuna see ei tee lülitumisel müra ja sellel puuduvad liikuvad, ja seetõttu kuluvad, detailid nagu releel.

Seadme esimeseks ja peamiseks komponendiks valisin mikrokontrolleri ESP8266. See mikrokontroller osutus valituks, kuna sellel on Wi-Fi võimekus, madal energiatarve ja olin sellega varem kokku puutunud. Ülejäänud komponendid valisin peamiselt ESP8266 mikrokontrolleriga ühilduvusest lähtuvalt.

Järgmiseks koostasin mikrokontrolleri juhtprogrammi. Juhtalgoritmiks valisin kahepositsioonilise kontrolleri. Esimese sammuna võtab programm seadme oma temperatuurianduri näidu, siis küsib MeiePilv serverilt andmed temperatuuri ja seadetemperatuuri kohta, ning arvutab temperatuurianduri näidu ja serveri temperatuuri põhjal välja parandi temperatuurianduri näidule. Siis võrdleb programm saadud temperatuuri näitu seadetemperatuuriga ja lülitab sellele vastavalt ümber radiaatoriklappe. Kui toimub lülitus, siis saadetakse selle kohta info ka MeiePilve.

Viimaseks osaks tööst jäi seadme katsetamine. Katsetasin seadme erinevaid funktsioone, ning lõpuks lasin sellel ühe ruumi keskkütte radiaatoreid juhtida nelja kuu jooksul. Katsete kokkuvõtteks võin öelda, et seade vastab sellele töö alguses seatud nõuetele.

Töö tulemusega olen ise rahul – töö tulemus vastab töö alguses seatud eesmärkidele. Seadme edasi arendamise poolest võiks sellele luua korpuse, arendada veel juhtimisloogikat (näiteks kasutada kahepositsioonilise juhtimise asemel PID-kontrollerit) ja ühildada see seade, mis tegeleb ruumi kütmisega, seadme/süsteemiga, mis tegeleks ruumi jahutamise/ventileerimisega.

SUMMARY

The living laboratory developed by Tartu College is currently focused on the indoor climate automation of the Puiestee 80A study building. The indoor climate data used in automation as feedback and the automation's own states are collected as a time series in a database called MeiePilv. In the long run, it is planned to develop a digital twin of the Puiestee 80A study building, of which the present work is primarily concerned with the thermo-technical parameters and technical system of the building. The digital twin allows new solutions to be played through virtually and only then to implement the algorithms in the physical world.

The aim of the present thesis stems from the need for actuators that would operate using feedback data logged into the MeiePilv database and the setpoints written to it by the digital twin.

It was necessary to develop a device that controls the temperature of a specific room using the set temperature and continuously transmitted room temperature measurements stored in the MeiePilv database, as well as a narrow +/- setting made by the room user.

I designed a technical solution for the device, the components of which are a microcontroller, a temperature sensor, a power supply module, and a transistor switch. The use of a separate temperature sensor in the device allows the device to control the room temperature adequately even in the absence of a network connection. I chose to use a transistor switch because, unlike a relay switch, it lacks wearing parts and switching makes no noise.

I chose the ESP8266 microcontroller as the first and main component of the device. This microcontroller was chosen because it has Wi-Fi capability, low power consumption, and because I have used similar devices before. I chose the other components mainly based on compatibility with the ESP8266 microcontroller.

Next, I created a control program for the microcontroller. I chose a two-position controller as the control algorithm. As the first step, the program takes a reading from the device's own temperature sensor, then asks the MeiePilv server for temperature and setpoint data, and finally calculates a correction to the temperature sensor reading based on the server temperature. The program then compares the corrected temperature reading with the set temperature and switches the radiator valves accordingly. If the radiator valves are switched, then information about it will also be sent to MeiePilv.

The last part of the work was testing the device. I tested the various functions of the device separately, and finally used it to control the central heating radiators in one room for four months. To sum up the tests, I can say that the device meets the requirements set at the start of work.

I am satisfied with the result of this thesis – the resulting device corresponds to the goals set at the beginning of work. In terms of further development of the device, a housing could be created for the device, further control logic could be developed (for example, using a PID controller instead of two-position control), and a room cooling/ventilation system could be developed to be used in conjunction with the device developed as a result of this thesis.

KASUTATUD KIRJANDUSE LOETELU

[1] „Taltech Tartu Kolledži Eluslabor," Taltech, [Võrgumaterjal]. Available: https://www.taltech.ee/tartu-kolledz/laborid.

[2] „Thermokon SR06 LCD," Thermokon, [Võrgumaterjal]. Available: https://www.thermokon.de/en/products/easysensr-transmitter/room-operating-units/sr06-lcd/.

[3] „Z-Wave," Silicon Laboratories, [Võrgumaterjal]. Available: https://www.z-wave.com/learn.

[4] „Industrial or Commercial Wireless Mesh Technologies," Data Respons, [Võrgumaterjal]. Available: https://datarespons.com/industrial-or-commercial-wireless-mesh-technologies/.

[5] „Z-Wave Radiator Valve Thermostat," Danfoss, [Võrgumaterjal]. Available: http://manuals-backend.z-wave.info/make.php?lang=en&sku=DAN_LC-13&cert=ZC08-13120001.

[6] „STC-DO 24V 868 MHz Datasheet," Thermokon, [Võrgumaterjal]. Available: https://www.thermokon.de/download-archive/EasySens%20-%20Empf%C3%A4nger/Aktoren/STC-DO/Produktbl%C3%A4tter/STC-DO_24V_EasySens_datasheet_en.pdf.

[7] „Pilvelabori platform viib digiõppe uuele tasemele," Taltech, [Võrgumaterjal]. Available: https://taltech.ee/uudised/pilvelabori-platvorm-viib-digioppe-uuele-tasemele.

[8] „ESP8266 Technical Reference," Espressif, [Võrgumaterjal]. Available: https://www.espressif.com/sites/default/files/documentation/esp8266-technical_reference_en.pdf.

[9] AliExpress, [Võrgumaterjal]. Available: https://www.aliexpress.com/item/32809039923.html?spm=a2g0s.9042311.0.0.27424c4djXQCc8.

[10] „Digital Humidity Sensor SHT3x (RH/T)," Sensirion AG, [Võrgumaterjal]. Available: https://www.sensirion.com/en/environmental-sensors/humidity-sensors/digital-humidity-sensors-for-various-applications/.

[11] Nettigo, [Võrgumaterjal]. Available: https://nettigo.eu/products/sht30-shield-for-wemos-d1-mini.

[12] „D1 mini shields - Battery shield," Wemos, [Võrgumaterjal]. Available: https://www.wemos.cc/en/latest/d1_mini_shield/dc_power.html.

[13] „ESP FAQ - Hardware Design," Espressif, [Võrgumaterjal]. Available: https://docs.espressif.com/projects/espressif-esp-faq/en/latest/hardware-related/hardware-design.html.

[14] AliExpress, [Võrgumaterjal]. Available: https://www.aliexpress.com/item/32880924006.html

[15] „IRF520 N-Channel Power MOSFET," Components101, [Võrgumaterjal]. Available: https://components101.com/mosfets/irf520-pinout-datasheet-features.

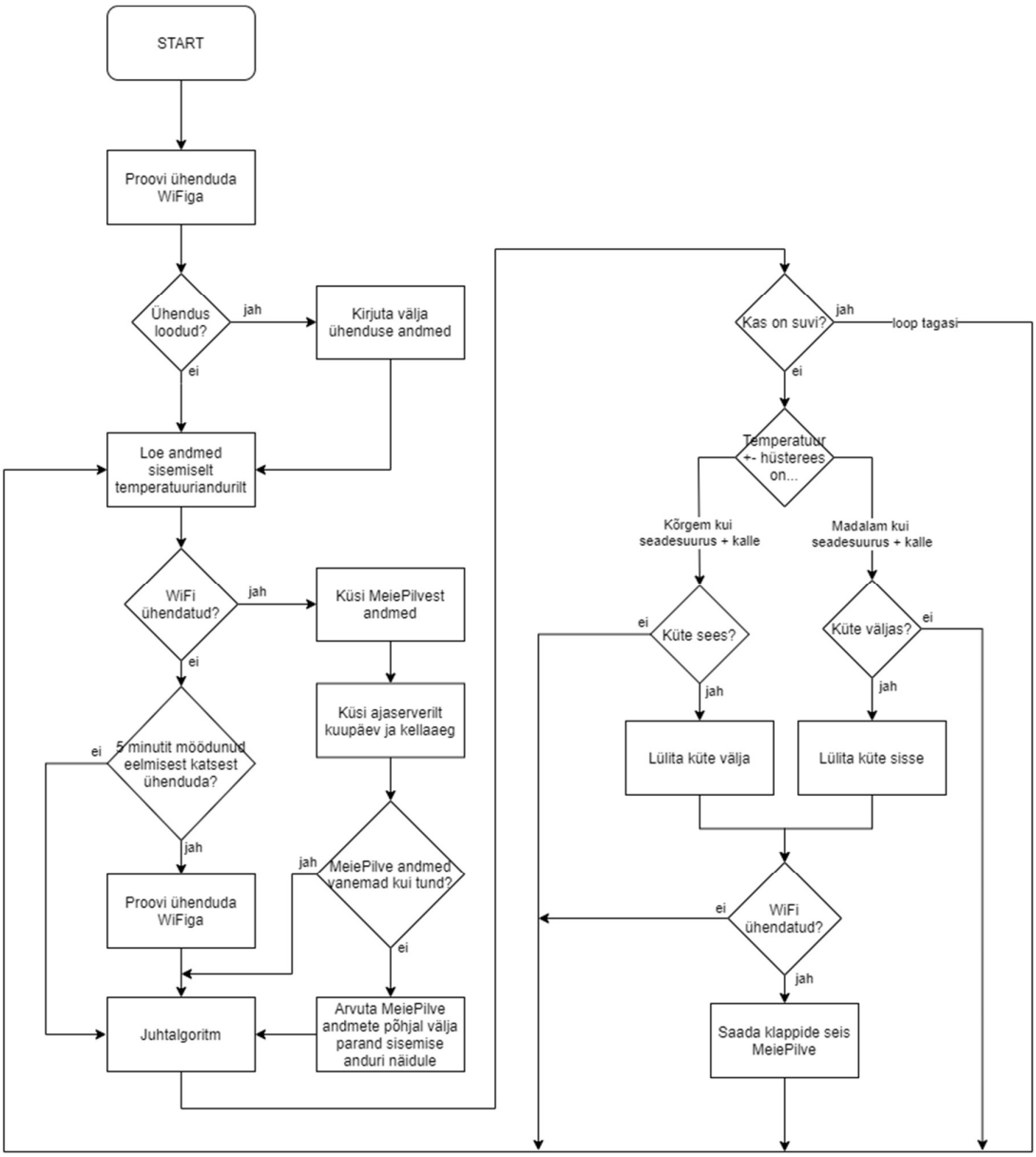
[16] AliExpress, [Võrgumaterjal]. Available: https://www.aliexpress.com/item/33010401389.html.

[17] „TWA Thermal Wax Actuator Data Sheet," Danfoss, [Võrgumaterjal]. Available: https://assets.danfoss.com/documents/83031/AI187386475671en-011001.pdf.

[18] HemmaTema Sweden AB, [Võrgumaterjal]. Available: https://hemmatema.se/varumarken/danfoss/twa-a-nc-termomotor-24v-ac-dc/.

LISAD

Lisa 1: Programmi blokkskeem



```
#include <ESP8266WiFi.h>                 // Lisan WiFi
#include <ESP8266HTTPClient.h>             // Lisan ESP 8266 HTTP kliendi
#include <ArduinoJson.h>                 // Lisan Json raamatukogu
#include <WEMOS_SHT3X.h>                   // kutsun välja WeMos SHT30 teegi
#include <WiFiUdp.h>
#include <TimeLib.h>
#include <EEPROM.h>

// VÕRGUÜHENDUSEGA SEOTUD KONSTANDID
// ================================
const char* ssid   = "****";
const char* parool = "****";
const char* sUrl = "****"; // url kirje päisesse
const char* id_temperatuur = "****";
const char* id_seadesuurus = "****";
const char* host = "****";         // Serveri IP
const int   port = **;                 // Serveri port

// KLAPPIDE SEISU ANDMETE ASUKOHT MEIEPILVES
// =========================================
const char* nimi = "Klapid";               // Koht serveris, kuhu klappide seis saadetakse
const char* vork = "OmaWiFi";               // Võrguliigi nimi serveri jaoks.

// MÕÕTMISE AJAARVESTUSEGA SEOTUD KONSTANDID JA MUUTUJAD
//
=====================================================
const int   intervall = 10000;            // Pöördumiste intervall (10 sekundit)
unsigned long eAeg = millis();             // Eelmine ajamärk
unsigned long pAeg = millis();             // Hetke ajamärk

// ÜHENDUSE AJAARVESTUSEGA SEOTUD KONSTANDID JA MUUTUJAD
//
=====================================================
const int   intervallWiFi = 300000;        // Pöördumiste intervall (5 minutit)
unsigned long eAegWiFi = millis();         // Eelmine ajamärk
unsigned long pAegWiFi = millis();         // Hetke ajamärk

// DHT PORDID
// ==========
SHT3X sht30(0x45);                     // Deklareerin SHT30 plaadi I2C pordi

// KÄIVITUSE JA VIGADEGA SEOTUD MUUTUJAD
//======================================
int katseid = 10;                      // WiFi ühenduse loomise katsete arv

// ANDMESALVESTUSEGA SEOTUD KONSTANDID JA MUUTUJAD
//================================================
const int maht = 1;                     // Mõõtmistsüklite arv ühes saatmistsüklis
float temperatuur_andur;                  // Globaalmuutuja SHT30 mõõdetud
temperatuurile
float niiskus;                         // Globaalmuutuja õhuniiskusele

// TEMPERATUURI JUHTIMISEGA SEOTUD KONSTANDID JA MUUTUJAD
//================================================
===
```

```
float seadesuurus = 22;
float temperatuur = 0;
float temperatuur_server = 0;
float parand = -3;
float hysterees = 0;
float kalle = 0;
long aeg_server = 0;
int juhtpin = 15;
int kuu = 1;
int state = 0;

// AJASERVERIGA SEOTUD KONSTANDID JA MUUTUJAD
//=========================================
unsigned int localPort = 2390;            // local port to listen for UDP packets
IPAddress timeServerIP;                   // time.nist.gov NTP server address
const char* ntpServerName = "time.nist.gov";
const int NTP_PACKET_SIZE = 48;           // NTP time stamp is in the first 48 bytes
of the message
byte packetBuffer[ NTP_PACKET_SIZE];      // buffer to hold incoming and outgoing
packets
WiFiUDP udp;                              // A UDP instance to let us send and receive
packets over UDP

// ALGSEADISTUSED
// ==============
void setup()
{
  pinMode(juhtpin, OUTPUT);
  pinMode(LED_BUILTIN, OUTPUT);
  Serial.begin(115200);
  Serial.println("");
  Serial.print("Ühendun ");
  Serial.print(ssid);
  Serial.println("");
  parand = LoeParandEEPROM(0);

  WiFi.mode(WIFI_STA);                    // Töömooduse seadistus, mis hoiab ära
tugijaama tekke.
  WiFi.begin(ssid, parool);               // WiFi ühenduse loomine. Pöördumine
tugijaama poole
  YhendaWiFi();                           // Proovi ühendada WiFi
  if (YhendusKontroll())                  // Kui ühendus loodi ..
  {
    Staatus();                            // Andmete print WiFi kohta
    udp.begin(localPort);                 // UDP ühenduse loomine (ajaserver)
  }
  else {                                  // Kui ühendust luua ei õnnestunud ..
    eAegWiFi = millis();                  // Alustan intervalli (5min WiFi)
  }
  eAeg = millis();                        // Alustan intervalli (10sec mõõtmine)
}

// ALGORITMI PÕHIOSA
// =================
void loop() {
  pAeg = millis();
```

```
  if (pAeg - eAeg >= intervall) {              // Kui eelmisest mõõtmisest möödunud
10sec ..
    eAeg = pAeg;
    Metroloog();
    if (YhendusKontroll()) {                 // .. ja võrguühendus loodud ..
      SoeluJson(LoeServerist(id_temperatuur)); // .. võtan serverist info temperatuuri ..
      SoeluJson(LoeServerist(id_seadesuurus)); // .. ja seadesuuruse kohta
      Serial.print("SEADESUURUS: ");
      Serial.println(seadesuurus);
      if (GetTime() < aeg_server + 3600) {     // Kui MeiePilve info vähem kui tund
aega vana ..
        parand = Parand();                   // .. arvuta uus parand (serverilt saadud temp.
..
        Serial.println(parand);
        KirjutaParandEEPROM(0, parand);
        Serial.print("EEPROM aadressile 0 kirjutati: ");
        Serial.println(LoeParandEEPROM(0));
        Serial.println("Andmed ajakohased!");  // .. kasutame ainult parandi
arvutamiseks)
      }
      else {
        kalle = 0;                           // Kui info vananenud, siis nupu positsiooniks 0
(keskel)
        Serial.println("Andmed aegunud!");
      }
    }
    else {                                   // Kui ühendust pole ..
      pAegWiFi = millis();                   // .. loen taimeri seisu
      if (pAegWiFi - eAegWiFi >= intervallWiFi) { // Kui aega kulunud vähemalt
intervalli võrra (5 min) ..
        YhendaWiFi();                        // .. proovin luua ühendust
        eAegWiFi = pAegWiFi;                 // WiFi intervall nulli
      }
      Serial.print("EEPROM-ist loetud parand: ");
      Serial.println(parand);
    }
    Serial.print("Temperatuur: ");
    Serial.println(temperatuur);
  }
  JuhtAlgoritm();                            // Juhtalgoritm jooksutatakse igal loopil
}
// WiFi STAATUSE ALAMPROGRAMM
//============================
// Kirjutab välja info loodud Wi-Fi ühenduse kohta
void Staatus() {
  Serial.println("");
  Serial.println("WIFI ANDMED");
  Serial.println("-----------");
  Serial.print("SSID:  ");
  Serial.println(WiFi.SSID());
  IPAddress ip = WiFi.localIP();             // IP aadress muutujasse
  Serial.print("IP on: ");
  Serial.println(ip);
  long signaal = WiFi.RSSI();                // signaalitugevus muutujasse
  Serial.print("Levi:  ");
  Serial.print(signaal);
  Serial.println(" dBm");
```

```
  Serial.println("");
};

// SERVERILE ANDMETE EDASTAJA
//============================
// Argumendis "info" sisaldub edastatava POST pöördumise BODY
// Funktsiooni enda väärtus on TRUE kui server annab vastuse 201 ja
// FALSE kui server edastab veateate. Mitmesugused pöördumise andmed võtab
// see funktsioon programmi pääisest ehk siis üldmuutujatest

boolean Kirjatuvi(String info)
{
  int httpVastuskood = 0;                // muutuja http vastuskoodile
  boolean abi = false;                  // abimuutuja vaikimisi valeks

  HTTPClient http;                      // ... käivitan http kliendi
  http.begin(host, port, sUrl);            // määran sihtkoha (oluline kirje)

  // Sean andmesisuks text/plain - edastatav info on tekst
  http.addHeader("Content-Type", "text/plain");
  httpVastuskood = http.POST(info);        // Muutujas "info" on tekst mida edastada
  Serial.println (httpVastuskood);
  if (httpVastuskood == 201)              // kui serveri vastuskood on 201, siis ...
  {
    abi = true;                         // saatmine õnnestus ja funktsioon saab olema true
...,
  }
  // ... muidu mitte
  http.end();                           // Vabastan serveri
  return abi;                           // funktsiooni väärtuseks abimuutuja väärtus
};                                     // funktsiooni lõpp

// SHT30-lt ANDMETE LUGEJA
//=============================================
// Loeb temperatuurianduri näidud

void Metroloog()
{
  do                                  // Väike tsükkel vigaste mõõdiste elimineerimsieks
  {
    delay(10);                          // Pisike paus enne uut mahalugemist
    sht30.get();                         // Loen info
    temperatuur_andur = sht30.cTemp - 3;    // Loen maha temperatuuri; teen parandi
-3 kraadi
    niiskus = sht30.humidity;             // Loen maha õhuniiskuse

  }                                    // Kui ei saa mõistlikku mõõdist, proovi uuesti
  while (isnan(niiskus) || isnan(temperatuur_andur));       // isnan()kontrollib kas
tulemus on OK

  Serial.print("Õhuniiskus: ");           // Prindin silumiseks välja
  Serial.print(niiskus, 1);
  Serial.print(" %, Temp: ");
  Serial.print(temperatuur_andur, 1);
  Serial.print(" °C,  ");
  Serial.print("Ajamärk ");
  Serial.println(millis());
```

```
  Serial.println("");
}

// JSON PAKETI VORMISTAJA
//=======================
// Vormistab saatmiseks BODY stringi ArduinoJSON teeki kasutades.
// Taavi Kase poolt tehtud variandi põhjal

String JsByrokraat()
{
  StaticJsonDocument<280> doc;             // Deklareerin JSON massiivi
  StaticJsonDocument<140> body;            // "body" jaoks eraldi JSON alammassiiv
  char buf[240];                          // Moodustan char tüüpi massiivi

  doc["device_name"] = nimi;              // Sisestan seadme nime
  doc["network_name"] = vork;              // Sisestan võrgu nime
  // teen massiivi massiivi sees
  JsonArray measures = doc.createNestedArray("body");
  body["time"] = 0;                       // Kui mõõtetulemused kohe ära saadetakse, on
ajanihe null.
  // väärtuste massiiv "values" objektile
  JsonArray values = body.createNestedArray("values");
  values.add(String(state));               // Lisan klapi seisu
  body.add(values);                       // Loon eelnevast väärtuse objektist koos ajaga
body objekti
  measures.add(body);                      // Lisan body objekti body massiivile
  serializeJson(doc, buf);                // Teisendame JSON objekti String objektiks
  return buf;                            // Tagastan funktsiooni väärtuse
}

// ÜMARDAJA
//==========
// Ümardab arvu "arv" 1 koht pärast koma

float Ymard(float arv)
{
  float tulem;                           // abimuutuja tulemi jaoks
  int vahe;                              // vaheväärtus teisendamisel

  vahe = int(arv * 10);                   // teisendan 10 korda suuremaks täisarvuks
  tulem = vahe / 10.0;                    // teen kümnendmurruks tagasi
  if (((10 * arv) - vahe) >= 0.5)          // kui sajandike mahalõikamisega tekkis
ümardusviga
  {
    tulem = tulem + 0.1;                  // siis parandan selle
  }
  return tulem;
}

// WIFI-ÜHENDUSE LOOJA
//====================
// Proovib luua WiFi-ühenduse

void YhendaWiFi() {
  int n = katseid;
  while (WiFi.status() != WL_CONNECTED)     // Kontroll, kas on ühendatud
  {
```

```
    if (n < 1)                        // Kui katsed said ammendatud ..
    {
      Serial.println("");
      Serial.println("-----------");

      Serial.println("Kahjuks ei õnnestunud luua WiFi ühendust.");
      Serial.println("Kontrollige logimisandmeid ja proovige uuesti.");

      break;                          // .. katkestatakse küsimine
    }
    n = n - 1;                         // Loen maha ühe katse iga proovimise korral.
    delay(1000);                         // Paus kahe kontrollimise vahel
    Serial.print(".");
  }
}

// SERVERIST ANDMETE LUGEJA
//=========================
// Küsib serverist ühe seadme viimased andmed

String LoeServerist(String id) {
  String payload;

  HTTPClient http;                       // Käivitan HTTP kliendi

  http.begin("http://193.40.13.86:82/measurements/last/" + id);  // Määran andmete
asukoha serveris
  int httpCode = http.GET();               // Saadan päringu

  if (httpCode > 0) {                    // Kontrollin, et vastuskood poleks 0 või -1
    payload = http.getString();            // Loen vastuse stringina maha
    Serial.print("Serverilt saadud info: ");
    Serial.println(payload);               // Kirjutan vastuse välja
  }
  else {
    Serial.print("Seadmelt ID: ");
    Serial.print(id);
    Serial.print(" ei õnnestunud infot lugeda. httpCode: ");
    Serial.println(httpCode);
  }
  http.end();                            // Sulgen ühenduse
  return payload;
}

// JSON PAKETI LUGEJA
//===================
// Loeb JSON paketist välja vajaminevad andmed

void SoeluJson(String s) {
  if (s.length() == 0) return;
  int ajutine = 127;
  StaticJsonDocument<200> doc;             // Annan 200 baiti mälu JSONile

  unsigned int len = 200;
  char json[len];

  s.toCharArray(json, len);
```

```
  deserializeJson(doc, json);

  if (doc["values"].size() == 1) {
    seadesuurus = doc["values"][0];
  }
  else {
    aeg_server = doc["time"];
    Serial.print("Serveri ajamärk: ");
    Serial.println(aeg_server);
    TeisaldaUnixUTC(aeg_server);
    temperatuur_server = doc["values"][1];    // Võtan "values" jadast teiste liikme
(temperatuur)
    ajutine = doc["values"][2];               // Võtan "values" jadast kolmanda liikme
(nupu seis)
    kalle = (map(ajutine, 0, 255, -200, 200) / 100.0); // Teisendan nupu seisu
vahemikku +- 2 kraadi
    Serial.print("Kalle: ");
    Serial.println(kalle);
  }
}

// PARANDI ARVUTAJA
//=======================
// Arvutab välja anduri temperatuurinäidu parandi serveri temperatuurinäidu suhtes

float Parand() {
  if (temperatuur_server > temperatuur_andur + 10 || temperatuur_server <
temperatuur_andur - 10) { // Kui serverist loetud temperatuur erineb anduri
temperatuurist +-10 kraadi, siis serveri temperatuuri ei kasutata
    return LoeParandEEPROM(0);
  }

  float ajutine = temperatuur_server - temperatuur_andur;

  Serial.print("Parand: ");
  Serial.println(ajutine);

  return ajutine;
}

// JUHTALGORITM
//=============
// Lihtne hüstereesiga juhtalgoritm, mis lülitab ümber radiaatoriklappe vastavalt
temperatuuriandurilt ja serverilt saadud andmetele

void JuhtAlgoritm() {
  if (kuu == 6 || kuu == 7 || kuu == 8) {   // kui on suvi siis hoitakse klappe kinni
    state = 0;
    digitalWrite(juhtpin, LOW);
  }
  else {
    temperatuur = temperatuur_andur + parand;
    if (temperatuur >= (seadesuurus + kalle) + hysterees) {
      if (state != 0 && YhendusKontroll()) {  // Kui state muutub ja WiFi-ühendus on
olemas...
        Kirjatuvi(JsByrokraat());                    // ... saada klapi seis kohe MeiePilve
```

```
      Serial.println("Ei küta");
     }
     state = 0;
     digitalWrite(juhtpin, LOW);
    }
    else if (temperatuur <= (seadesuurus + kalle) - hysterees) {
     if (state != 8 && YhendusKontroll()) {  // Kui state muutub ja WiFi-ühendus on
olemas...
      Kirjatuvi(JsByrokraat());                    // ... saada klapi seis kohe MeiePilve
      Serial.println("Kütab");
     }
     state = 8;
     digitalWrite(juhtpin, HIGH);
    }
   }
}

// NTP PACKET SENDER ; NTP PAKETTI SAATJA
//===================
// Saadab NTP päringu antud ajaserverile
unsigned long sendNTPpacket(IPAddress& address)
{
  Serial.println("saadan NTP paketi...");
  memset(packetBuffer, 0, NTP_PACKET_SIZE); // Kõik baidid puhvris nulli
  // Initsialiseeri NTP päringu jaoks vajalikud väärtused
  packetBuffer[0] = 0b11100011;           // LI, Version, Mode
  packetBuffer[1] = 0;                    // Stratum, or type of clock
  packetBuffer[2] = 6;                    // Polling Interval
  packetBuffer[3] = 0xEC;                 // Peer Clock Precision
  // 8 nullbaiti Root Delay & Root Dispersion jaoks
  packetBuffer[12]  = 49;
  packetBuffer[13]  = 0x4E;
  packetBuffer[14]  = 49;
  packetBuffer[15]  = 52;

  // Nüüd, kui kõigil NTP väljadel on väärtus, saadab päringu
  udp.beginPacket(address, 123);          // NTP päringud lähevad pordile 123
  udp.write(packetBuffer, NTP_PACKET_SIZE);
  udp.endPacket();
}

// AJAPÄRING SERVERILT
//=====================
// Pärib serverilt aja
long GetTime() {
  WiFi.hostByName(ntpServerName, timeServerIP);  // Hangi juhuslik server serverite
hulgast

  sendNTPpacket(timeServerIP);            // Saada serverile NTP pakett
  delay(3000);                            // Anna serverile hetk aega vastamiseks

  int cb = udp.parsePacket();             // Salvesta serveri vastus muutujasse cb
  if (!cb) {                              // Kui muutuja cb false (0/tühi), siis järelikult vastust
ei tulnud
    return 0;
  }
  else {
```

```
    udp.read(packetBuffer, NTP_PACKET_SIZE);// Loe pakett puhvrisse

    // Ajamärk algab 40 baidist ja on 4 baidi ehk 2 wordi pikkune
    // Esimesena loe puhvrist need 2 wordi välja

    unsigned long highWord = word(packetBuffer[40], packetBuffer[41]);
    unsigned long lowWord = word(packetBuffer[42], packetBuffer[43]);
    // Kombineeri need 2 wordi 1 longiks
    // See on NTP aeg (sekundite arv peale 1. jaanuar 1900)
    unsigned long secsSince1900 = highWord << 16 | lowWord;

    // Muundab NTP aja tavaliseks ajaks
    Serial.print("Unix aeg = ");
    // Unix aeg algab 1. jaanuaril 1970. See on 2208988800 sekundit
    const unsigned long seventyYears = 2208988800UL;
    // Lahuta 70 aastat
    unsigned long epoch = secsSince1900 - seventyYears;
    // prindi Unix aeg:
    Serial.println(epoch);
    TeisaldaUnixUTC(epoch);
  }
}

long TeisaldaUnixUTC(long epoch) {
  Serial.print("UTC aeg on: ");
  Serial.print((epoch  % 86400L) / 3600);
  Serial.print(':');
  if ( ((epoch % 3600) / 60) < 10 ) {
    Serial.print('0');
  }
  Serial.print((epoch  % 3600) / 60);
  Serial.print(':');
  if ( (epoch % 60) < 10 ) {
    Serial.print('0');
  }
  Serial.println(epoch % 60);

  kuu = month(epoch);
  Serial.print("Kuu: ");
  Serial.println(kuu);

  return epoch;
}


bool YhendusKontroll() {
  if (WiFi.status() == WL_CONNECTED) {
    digitalWrite(LED_BUILTIN, LOW);
    return true;
  }
  else {
    digitalWrite(LED_BUILTIN, HIGH);
  }
  return false;
}

void KirjutaParandEEPROM(int addr, float parand) {
```

```
  float temp = parand * 10;

  EEPROM.begin(512);  //Initialize EEPROM
  if (parand < 0) {
    EEPROM.write(addr + 1, 1);
    temp *= -1;
  }
  else {
    EEPROM.write(addr + 1, 0);
  }
  EEPROM.write(addr, temp);
  EEPROM.commit();
}

float LoeParandEEPROM(int addr) {
  EEPROM.begin(512);  //Initialize EEPROM
  float temp = EEPROM.read(addr);
  temp *= 0.1;
  if (EEPROM.read(addr + 1) == 1) {
    temp *= -1;
  }
  return temp;
}
```