Roger Kupari

ÄLYKÄS VEDENPUHDISTUSJÄRJESTELMÄ

Prototyypin ohjausjärjestelmän kehitys

Opinnäytetyö
CENTRIA-AMMATTIKORKEAKOULU
Tieto- ja viestintätekniikan koulutusohjelma
Kesäkuu 2018

TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Centria-ammattikorkeakoulu	**Aika** Kesäkuu 2018	**Tekijä/tekijät** Roger Kupari
Koulutusohjelma Tieto- ja viestintätekniikan koulutusohjelma		
Työn nimi ÄLYKÄS VEDENPUHDISTUSJÄRJESTELMÄ. Prototyypin ohjausjärjestelmän kehitys.		
Työn ohjaaja Sähkö- ja automaatiotekniikan lehtori Hannu Ala-Pöntiö		**Sivumäärä** 47 + 10
Työelämäohjaaja Tuotekehitysinsinööri Lucas Manderbacka		

Opinnäytetyön toimeksiantajana toimi kokkolalainen yritys Double M Properties Ab. Opinnäytetyön aihe jakautui kahteen osa-alueeseen: suolanpoistolaitteen prototyypin sähköistämis- ja ohjausratkaisuun. Työn pääasiallinen tarkoitus oli toimia suunnannäyttäjänä jatkokehitystä silmällä pitäen.

Sähköistämisratkaisu pohjautui pienoisjännitejärjestelmään, jossa käytettiin yhtenä energian lähteenä aurinkoenergiaa. Sähköistämisosio sisälsi lisäksi erilaisia suojaratkaisuja sekä tarvittavia komponenttiratkaisuja.

Ohjausratkaisu toteutettiin Raspberry Pi -tietokoneella. Raspberryn tehtävänä oli tukea sähköisiä ratkaisuja ja mahdollistaa erilaiset käyttötoimenpiteet ilman edellytystä käyttäjän syvemmistä tietoteknisistä taidoista.

Lopuksi työssä pohdittiin muun muassa sähkönkäytön volyymia ja siitä syntyviä kehitystarpeita.

Asiasanat
Anturitekniikka, aurinkokeräin, aurinkopaneeli, Arduino, Mysql, Node-RED, ohjausjärjestelmä, pienoisjännite, prototyyppi, Python, Raspberry, suolanpoistaja

ABSTRACT

Centria University of Applied Sciences	Date	Author
	June 2018	Roger Kupari
Degree programme Information Technology		
Name of thesis SMART WATERCLEANING SYSTEM. Developing a control system for a prototype		
Instructor Lecturer of Electrical and Automation Engineering Hannu Ala-Pöntiö		**Pages** 47 + 10
Supervisor Research and Development Engineer Lucas Manderbacka		

This thesis was commissioned by Double M Properties Ab, which is located in Kokkola, Finland. The subject of this work was divided under two sections: the controlling system under development and the electrical system solutions of the desalinator prototype. The main purpose was to give direction for the further development processes.

The part of electrical solution was based on low electric system. One of the energy sources used was solar power. In addition the electrical unit also included some components and safety solutions.

The controlling unit was implemented by using the Raspberry Pi computer. The Raspberry's task was to support the electrical unit and allow the system user to use the whole system with just basic computer skills.

The last part of this thesis considers the usage of electricity and the emerging needs for development, among other things.

Key words
Arduino, control system, desalinator, low voltage, Mysql, Node-RED, prototype, Python, Raspberry, sensor technology, solar collector, solar panel

KÄSITTEIDEN MÄÄRITTELY

1-WIRE	Tietoliikenneprotokolla
AFS	Adaptive Front-lightning System, mukautuva ajovalojärjestelmä
AMPEERI	(A) Virran yksikkö
ANALOGINEN	Viesti voi sisältää useampia arvoja väliltä 0–9 (tässä työssä)
ANIONI	Negatiivisesti varautunut ioni
ANODI	Positiivinen elektrodi
ARDUINO	Kehitysalusta
AURINKOKERÄIN	Paneeli, jolla otetaan vastaan lämpö auringon säteilystä
AURINKOPANEELI	Paneeli, jolla tuotetaan sähköenergiaa auringon säteilyn avulla
BAR	Paineen yksikkö
BAUD	Tiedonsiirron nopeuden ilmaiseva yksikkö, tapahtumaa/sekunnissa
CPU	Mikropiiri, jolla pystytään tekemään matemaattisia että loogisia päätöksiä
CRON	Aikaan perustuva toistorutiini
DEKOODERI	Menetelmä, jolla vastaanotettava tieto puretaan toiseen muotoon
DIGITAALINEN	Viestissä on arvo joko 1 tai 0 (tässä työssä)
DIP-KYTKIN	Kytkin, joka käsittää monivalintaisen asennon
ECU	Engine Control Unit. Moottorin ohjausyksikkö (autotekniikassa)
ED	Electrodialysis. Elektrodialyysi
EDR	Electrodialysis revesal. Käänteinen elektrodialyysi
ENKOODERI	Menetelmä, jolla lähetettävä tieto pakataan toiseen muotoon
FRONTEND	Web-sivu, asiakasrajapinta
GND	Nollapotentiaali / maapotentiaali (Viitattaessa sähköisiin järjestelmiin)
GPIO	General-purpose input/output. Monikäyttöiset sisään-/ulostulot
I/O	Asiayhteydestä riippuen (sisääntulo/ulostulo) tai digitaalinen (1 tai 0)
INVERTOINTI	Käänteinen arvo
IP-LUOKITUS	Kotelointiluokitus vierasesineiden ja/tai aineiden sisäänpääsyn arvioimiseksi
MEGAJOULE	Energian yksikkö
JAVASCRIPT	Ohjelmointikieli
KATIONI	Positiivisesti varautunut ioni
KATODI	Negatiivinen elektrodi
KERNEL	Keskusyksikön käyttöjärjestelmän ydin.

M-BUS	Tiedonsiirtoprotokolla
MAA	Maapotentiaali tai nollapotentiaali (Viitattaessa sähköisiin järjestelmiin)
MED	Multi-Effect Distillation. Monivaihelauhdutustislaus
MCU	Kuten CPU, mutta sisältää muistit ja mallista riippuen oheisväyliä.
MSF	Multi-Stage Distillation. Monivaihehöyrystystislaus
MQTT	Tiedonsiirtoprotokolla
MYSQL	Tietokantaohjelmisto
NODE-RED	Kehitysympäristö graafiseen web-palveluun
NOLLA	Nollapotentiaali (Viitattaessa sähköisiin järjestelmiin)
LATENTTILÄMPÖ	Olomuodon muutokseen tarvittava energia
PIENJÄNNITE	Tasajännitettä 120–15 000 volttia, vaihtojännitettä 50–1 000 volttia
PIENOISJÄNNITE	Tasajännitettä 0–120 Volttia, vaihtojännitettä 0–50 volttia
PYTHON	Ohjelmointikieli
RO	Reverse Osmosis, käänteinen osmoosi
RS-232	Tietoliikenneprotokolla
RS-485	Tietoliikenneprotokolla
TISLAUS	Aineiden erottamismenetelmä
TTL	transistor-transistor logic, käyttöjännitteen potentiaaliero 5V
ULTRAÄÄNI	Aaltoliike, jonka värähtelytaajuus on korkea
USB	Universal Serial Bus. Yleismallinen sarjaliikenneväylä
VAC	Vaihtojännite
VDC	Tasajännite
VCD	Vapor Compression Distillation. Höyrynpuristustislaus
VOLTTI	(V) Jännitteen yksikkö
V+	Positiivinen käyttöjännite
VATTI	(W) Tehon yksikkö
VATTITUNTI	(Wh) Energian yksikkö
WIFI	Langaton verkko

KUVIOT

KUVAT

TAULUKOT

1 JOHDANTO

Tämä suolanpoistajan ohjausjärjestelmää koskeva työ on tehty kokkolalaisen yhtiön projekti- ja kehitysyksikössä Sokojalla. Opinnäytteessä on esitetty varsin laaja viitekehys koskien suolanpoistotekniikoiden menetelmiä. Viitekehys on pyritty avaamaan siten, että sen pystyy tulkitsemaan perusyleissivistyksen avulla. Itse suolanpoistaja on Suomessa hieman tuntemattomampi laite.

Suolanpoistoon liittyät laitteet on yleisesti varmaankin tuntemattomampia maissa, missä on riittävät makeanveden varannot. Tämä projekti on tarjonnut allekirjoittaneelle mielenkiintoisen mahdollisuuden tutustua suolanpoistotekniikoihin. Toivottavasti työ antaa mahdollisuuden lukijallekin perehtyä suolanpoistoon riittävin tiedoin.

Työn toiminnallisen osuuden sisältö käsittää laitteiston sähköistys- ja ohjelmistoratkaisun ohjausympäristökäyttöön. Sähköisissä valinnoissa tuli ottaa huomioon työryhmän aikaisemmin valitsemat komponentit ja ohjata sitä kokonaisuutena suuntaan, jossa sillä on puitteet toimia yhdenmukaisesti. Ohjelmallisen osuuden tarkoitus oli mahdollistaa järjestelmän sähköisten osien tulkinta ja niiden tietynlainen keskinäinen kommunikointi.

Ohjelmallinen toteutus liittyy ohjelmasuorituksiin pohjautuen kahteen toisistaan poikkeaviin ympäristöihin. Erityisesti kahden ohjelmointikielen välisen kommunikointitavan ja kehitysympäristövaatimukset ovat jääneet kirjallisessa työssä vähemmälle johdatukselle.

Ohjelmallisen osuuden tulkinnassa auttaa perustietämys ohjelmoinnista sekä hieman syvempi tietämys esitetystä kehitysympäristöstä. Kehitysympäristön toimintatapaa on hieman avattu, mutta käytöstä pystyisi kirjoittamaan vaikka kokonaisen tutkielman, joten sen käyttöön perehdytään työssä vain hieman.

Työn lopussa otetaan kantaa järjestelmän jatkokehityksellisiin asioihin sekä annetaan perusta tiedoille, joita voidaan käyttää sähköistysjärjestelmän päivityksessä. Sähköistysjärjestelmään liittyvät sähkönvarastointi sekä sähköntuottoon liittyvät komponentit vaikuttavat olevan joiltain osin päivityksen tarpeessa. Suurin osa kuvioista, kuvista ja liitteistä on kommentoitu englanniksi, joka on projektissa käytetty työkieli.

2 OPINNÄYTETYÖN TAVOITE JA TARKOITUS

Opinnäytetyössä käsiteltävät asiat liittyvät suolanpoistajan prototyypin testialustan kehitykseen. Kehityksen tuloksellinen tarkoitus on toimia rajapintana työryhmän sekä laitteiston ”kommunikoinnin” välillä. Alustan on tarkoitus tukea kehitetyn suolanpoistolaitteiston yleistä toimivuutta, ohjattavuutta sekä tiedon analysointia. (Manderbacka 2018; Sander 2018a.)

2.1 Opinnäytetyön tavoite

Opinnäytetyön tavoite on auttaa työryhmää sekä sähköisin että ohjelmallisin keinoin onnistumaan tutkimuksessa. Työryhmän tutkimuksen tavoite on löytää vastaus siihen, toimiiko suolanpoistaja menetelmällä, jolla sen on suunniteltu toimivan. Tämä opinnäytetyö ei ota kantaa laitteen prosessimenetelmään liittyviin tutkimustuloksiin tai kehitystarpeisiin. (Manderbacka 2018; Sander 2018a; Sivula 2018.)

Tavoite koostuu kahdesta osa-alueesta: prototyypin sähköistämisratkaisuista sekä ohjelmallisesta toteutuksesta. Ohjelmallinen osuus liittyy vahvasti muun muassa laitteiston sähköiseen hallintaan. Sähköistämisen tavoitteena on yhdistää työryhmän ennalta valitsemat komponentit sekä valitsemani komponentit siten, että ne voivat toimivat yhteisenä kokonaisuutena.

Ohjelmallisen osuuden tavoite koostuu muun muassa sähköisten ratkaisujen hallinnasta siten, että työn valmistututtua prototyyppiä voidaan kutsua laitteistoksi käyttäjän näkökulmasta. Ohjelmallinen osuus käsittää myös tiettyjen sähköisten komponenttien sekä anturien arvojen tallennusrutiinit ja luo perustan laitteiston tilan yleiseen valvontaan. Opinnäytetyöstä syntyviä tuloksia ja ratkaisuja on tarkoitus käyttää käsiteltyjen aiheiden jatkokehityksellisissä toimenpiteissä.

2.2 Opinnäytetyön tarkoitus

Opinnäytetyön tarkoitus on luoda työryhmälle tarvittavat ympäristöt prototyypin sähköiseen hallintaan, tietojen rekisteröintiin sekä laitteiston yleiseen hallintaan ja valvontaan. Opinnäytetyön tarkoituksena yleisesti on myös toimia asiakirjana, jota voidaan tarvittaessa käyttää sekä sähköisten että ohjelmallisten ratkaisujen jatkokehityksessä.

3 TOIMEKSIANTAJAN ESITTELY

Toimeksiantaja on kokkolalainen Double M Properties Ab -niminen yhtiö (DPM). Yrityksen ydinliiketoiminta käsittää kolme eri sektoria: kiinteistöjen vuokraus, holding-toiminta sekä projekti- ja kehitysliiketoiminta. Holding -liiketoiminnan kannalta merkittävimmät yritykset ovat Ab Hiltop Oy sekä Ab HUR Oy. Hiltop yrityksenä sekä HURin ensimmäiset toteutukset ovat syntyneet tiloissa, joissa tämä opinnäytetyöprojekti on toteutettu. (Sivula 2018.)

Opinnäytetyö on toteutettu toimeksiantajan projekti- ja kehitystoimintayksikössä. Yksikkö käsittää muun muassa merkittävän EU-rahoitteisen lämmön varastointiin liittyvän Heliostorage-kehitysprojektin toteutuksen. Projekti kuuluu Evakot-hankkeeseen ja siinä on mukana Centria tutkimus ja kehitys, Geologian tutkimuskeskus (GTK), Heliostorage (DPM), Ahola Transport sekä Finn Spring. (Sivula 2018.)

Yksikön muita projekteja ovat muun muassa tuulivoima-akku, joka on kehitettävänä tällä hetkellä Tampereen teknillisessä yliopistossa. Markkinoilla olevat Ballwall-ratkaisut jalkapallon tekniikkaharjoitteluun lukeutuvat myös yrityksen meriitteihin. (Sivula 2018.)

4 SUOLANPOISTAJA

Suolanpoistaja (desalinator) on laite/laitos, joka on tarkoitettu pääasiallisesti suolan poistamiseen vedestä. Maailman vesivarannoista yli 97 % koostuu merivedestä. Suolanpoistajan suurimmat markkinat kohdistuvat meriveden prosessointiin juomavedeksi. Suolanpoistoprosessi on patentoitu vuonna 1852 Englannissa. Patentin aikaan prosessille oli kysyntää etenkin merimiesten keskuudessa, sillä heillä saattoi olla pulaa matkoillaan juomavedestä. (O'Neill 2015, 457–458; Voutchkov 2016.)

Vuonna 2015 on ollut arviolta noin 18 000 kappaletta erilaisia suolanpoistolaitoksia, jolloin niiden tuotantokapasiteetti on ollut yli 86 miljoonaa kuutiometriä. Kysynnän odotetaan markkinoilla edelleen kasvavan lisääntyvän veden puutteen vuoksi. Suolanpoistajat tuottavat tällä hetkellä noin yhden prosentin koko maailman juomavedestä. (O'Neill 2015, 457–458; Voutchkov 2016.)

Suolanpoistajia on markkinoilla erilaisiin tarpeisiin, pienistä vapaa-ajan asunnon tarpeet täyttävistä laiteista suuriin tuotantolaitoksiin. Suolanpoistajia on myös Suomen markkinoilla, esimerkiksi joki-, järvi- ja merivesien puhdistukseen. Markkinat Suomessa kattavat sekä mahdolliset kuluttajatarpeet että myös hieman suuremmatkin tarpeet. Suomessa myytävissä laitteissa vaikuttaisi yleisin suolanpoistotekniikka pohjautuvan kalvosuodatustekniikkaan. (KMV-tuotteet Oy; Kupari Solutions Oy 2018.)

Tässä työssä käsiteltävä suolanpoistolaitteen prototyyppi on tarkoitettu kansainvälisille markkinoille, eikä sen pääasiallinen kohderyhmä ole Suomessa. Laitteen tarkoitus on puhdistaa merivedestä juomakelpoista vettä, prosessimenetelmänä on suolanpoiston lämpötekniikka. Työryhmän tavoite on selvittää, onko rakennettu laite toimiva, onko se riittävän energiatehokas, toimiiko se halutun mukaisesti ja minkälaista tuotantokapasiteettia voidaan tavoitella. Opinnäyte haluttiin pitää julkisena asiakirjana, joten tässä työssä ei käsitellä testituloksia tai tuotesalaisuuden piiriin kuuluvia asioita. (Manderbacka 2018; Sander 2018a.)

4.1 Suolanpoistomenetelmien yleisimmät tekniikat

Desalination tarkoittaa suomeksi käännettynä suolanpoistoprosessia. Hieman syvällisemmin tarkasteltuna se on prosessi, jossa merivettä prosessoidaan pääasiassa juomavedeksi tai puhtaaksi vedeksi teolli-

suuden käyttöön. Tämän lisäksi prosessituotteet voidaan jatkokäsitellä soveltuvaksi kunkin makean veden tarpeen täyttämiseksi. Suolanpoisto on yleisempää sellaisilla alueilla, missä ei ole luonnon makean veden varantoa käytettävissä kansalaisten ja teollisuuden tarpeiden täyttämiseksi. Yleensä tällaisilla alueilla on kuitenkin merivesivarantoja, joita voidaan prosessoinnin tuloksena käyttää alueen makean veden tarpeisiin. Yleisimmin käytetyt suolanpoistoprosessit pohjautuvat kalvosuodatus- tai lämpötekniikkaan. (Seawater desalination technologies 2015, 1; Krishna 2004, 1–2.)

4.1.1 Kalvosuodatustekniikka

Kalvosuodatustekniikka perustuu pääosin kolmeen menetelmään: elektrodialyysiin (ED), Käänteiseen elektrodialyysiin (EDR) tai käänteiseen osmoosiin (RO). Kalvosuodatustekniikka ei suoranaisesti liity aiheena olevaan prototyyppiin, mutta tässä sekä seuraavassa alaluvussa esitellään yleisimmät suolanpoistossa käytetyt menetelmät teoriassa. (Krishna 2004, 2.)

Elektrodialyysi menetelmänä perustuu tasasähkövaraukseen, joka välitetään elektrodeilla. Elektrodien välissä on usein useampia kationi- ja anionikalvoja eli kalvoja, jotka läpäisevät vain kationeja tai anioneja. Sähköisesti varautuneet, vastakkaismerkkiset varaukset pyrkivät hakeutumaan toistensa vaikutuspiiriin. Esimerkiksi kun natrium on sähköisesti positiivisesti varautunut ja katodi on negatiivisesti varautunut: natrium on ensin läpäissyt kationikalvon, minkä jälkeen se pyrkii hakeutumaan katodin vaikutuspiiriin. Kalvojen määrää säätelemällä voidaan vaikuttaa puhdistetun veden laatuun. Tätä menetelmää ei yleensä käytetä korkeiden suolapitoisuuksien poistoon. (Kaasalainen 2007, 18–19; Krishna 2004, 3–4; Paul 2018.)

Käänteinen elektrodialyysi perustuu fyysisesti samanlaiseen ratkaisuun kuin elektrodialyysi, mutta tietyin väliajoin elektrodien varaukset vaihdetaan vastakkaismerkkisiksi. Varauksien vaihto helpottaa kalvojen puhtaanapitoa. Käänteisosmoosissa vettä ajetaan puoliläpäisevän kalvon läpi paineella, jolloin vesi läpäisee kalvon mutta suolat jäävät kalvon pinnalle. Näin ollen siis prosessituote on suolaton vesi (Kaasalainen 2007, 18–19; Krishna 2004, 3–4; Paul 2018.)

4.1.2 Lämpötekniikka

Lämpötekniikka suolanpoistossa perustuu pääosiin kolmeen menetelmään: monivaihehöyrystystilaukseen (MSF), monivaihelauhdutustislaukseen (MED) tai suoraan suomennettuna; höyrynpuristustislaukseen (VCD). Lämpötekniikka suolanpoistossa linkittyy vahvasti tislausprosessiin. (Krishna 2004, 2).

Veden höyrystyminen normaalia alemmassa lämpötilassa vaatii paineen, joka on alle normaalipaineen eli toisin sanoen alipaineen. Veden ominaislatenttilämpö on 2,26 megajoulea kilolle normaalipaineessa. Alipaineessa ominaislatenttilämpö on jotain muuta. Myös prototyypissä käytettävä menetelmä pohjautuu lämpötekniikkaan, joten palataan tarkemmin alipaineeseen hieman myöhemmin tässä työssä. (Inkinen & Tuohi 2006, 386–387; Sander 2018a; ThermExcel 2003.)

Monivaihehöyrystystislausmenetelmä perustuu suolaveden höyrystymiseen paine-erolla. Tislausprosessi tapahtuu useammassa monitasoisessa kammiossa. Ensimmäinen kammio käsittää ylipaineen, jossa suolavesi lämmitetään. Lämmittämisen jälkeen vesi johdetaan seuraavaan kammioon, jossa tapahtuu veden kiehuminen, samoin kuin paineen vapautus. Veden kulkeutuminen tapahtuu siis spontaanisti seuraaville tasoille. Veden kulkeutuminen johtuu paine-erosta, sillä seuraavissa kammioissa on aina pienempi paine verrattuna edelliseen. Lopuksi MSF-menetelmällä prosessituote (suolaton vesi), on vain murto-osa verrattuna syötettyyn suolaveden määrään. (Krishna 2004, 2–3.)

Monivaihelauhdutustislausmenetelmä perustuu suolaveden höyrystymiseen säiliöiden välisien progressiivisien alipaine-erojen vaikutuksesta. Veden kiehumispiste saavutetaan alemmalla lämpötilalla alipaineen ansiosta. MED-menetelmällä edellinen säiliö tarjoaa väliaineen lämmölle, ja näin ollen väliaineen höyrystymiselle aina seuraavalle säiliölle. (Krishna 2004, 3.)

Höyrynpuristustislausmenetelmää käytetään yleensä toisen prosessin yhteydessä, mutta sitä voidaan käyttää myös itsenäisesti. VCD-menetelmä vaikuttaa ainakin näin teoriassa hieman turbon kaltaiselta ratkaisulta. Menetelmässä kompressori puristaa esimerkiksi lämmitysjärjestelmässä muodostuneen vesihöyryn, jonka prosessitulos on kuitenkin sama kuin edellisissä tavoitteina eli puhdas vesi. (Krishna 2004, 3.)

4.2 Suolanpoistaja sulautettuna järjestelmänä

Sulautetut järjestelmät voidaan määritellä sellaiseksi kokonaisuudeksi, jossa esimerkiksi mekaanisia elementtejä voidaan kuunnella, ohjata hallita, ja/tai hallinnoida yleensä sähköisten ilmiöiden avulla. Mekaaniset elementit tässä tapauksessa tarkoittavat esimerkiksi antureita ja ohjattavia komponentteja, kuten lämpöantureita tai moottoreita. Mekaaniset elementit, joissa tapahtuu mekaanisia muutoksia ja muutokset ovat tulkittavissa perustuen sähköisiin ilmiöihin, kutsutaan yleensä antureiksi. Jos mekaaniset muutokset tässä yhteydessä tapahtuvat myötävaikuttamalla sähköisten ilmiöiden perusteella fyysisiin muutoksiin, elementtiä voidaan kutsua komponentiksi. Sulautettu järjestelmä yleensä sisältää myös sellaisia elementtejä, joissa ei ole aistinvaraisesti havaittavissa mekaanisia tapahtumia. (Rouse 2016.)

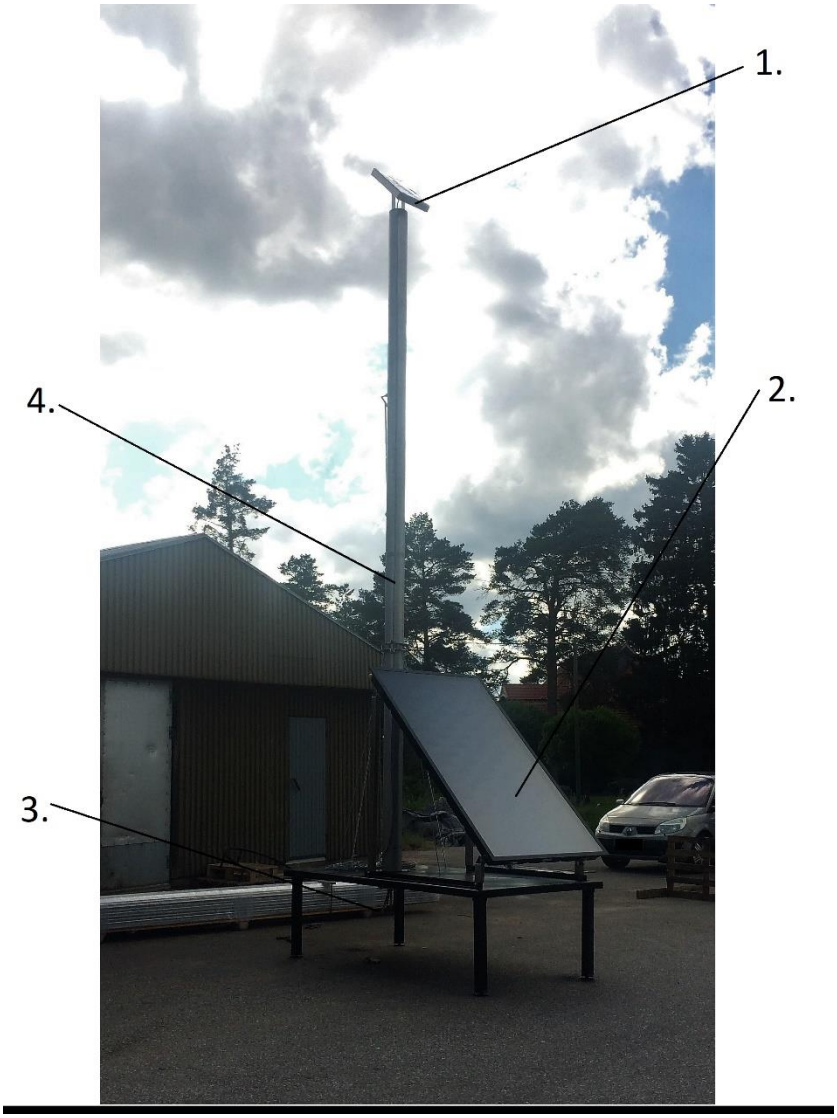
Sulautettu järjestelmä yleensä pohjautuu joko mikrokontrolleriin (MCU), mikroprosessoriin (CPU) tai näiden erilaisiin yhdistelmiin. Sulautettu järjestelmä yleensä käsittää järjestelmälle tehdyn yksilöllisen ohjelmiston. Yhteistä näiden elementtien sekä ohjelmiston välillä on se, että ne eivät välttämättä toimi ilman toisiaan halutulla tavalla. Sulautettu järjestelmä saattaa olla yhdistelmä erilaisia komponentteja, ja ne saattavat pohjautua useampaan erilliseen mikropiiriin, joilla jokaisella on oma tehtävänsä. (Rouse 2006; Rouse 2016.)

Esimerkkinä voi toimia auton järjestelmistä moottorinohjausjärjestelmä (ECU) sekä uudenaikaiset valojärjestelmät, esimerkiksi (AFS). Moottorinohjaus- ja valojärjestelmät toimivat omien komponenttiensa sekä mikropiiriensä varassa itsenäisesti. Järjestelmien kommunikointi tapahtuu kuitenkin saman väylän kautta. Käyttäjälle järjestelmät näyttävät tältä osin siis yhdeltä. (Aermech 2014; Rouse 2016; Texas Instruments 2013.)

Työssä käsiteltävän suolanpoistajan prototyypin keskusyksikkö on varustettu mikroprosessorilla. Muut sijoitetut komponentit saattavat sisältää erilaisia mikropiirejä erilaisiin käyttötarkoituksiin. Keskusyksikön tehtävä on yhdistää kaikki lähtevä sekä sisään tuleva informaatio. Informaation täytyy olla sellaisessa muodossa, että prototyypin keskusyksikön ohjelmisto pystyy hallitsemaan sekä tulkitsemaan erisuuntaista liikennettä. (KMtronic a; Raspberry Pi Foundation.)

5 PROTOTYYPIN RAKENNE

Tämän opinnäytetyön aiheena oleva suolanpoistojärjestelmässä (KUVA 1) itse prosessin kehitys ei ole opinnäytetyön tarkoitus. Työn tarkoitus löytyy sähköisistä ja ohjelmallisista ratkaisuista. Ratkaisujen tulisi tukea itse prosessia, prosessikehitystä, prosessitestausta, laitteiston hallintaa ja ylläpitoa. Kokonaiskuvan hahmottamiseksi tässä kappaleessa käydään läpi lyhyesti myös prosessiin liittyviä asioita.



KUVA 1. Prototyypin runko vuonna 2017

Prosessissa kaikki käytettävä lämpöenergia pyritään ottamaan auringon säteilystä. Aurinkokeräimenä käytetään Rothin valmistamaa Heliostar 218 S4 -keräintä, joka näkyy kuvassa merkittynä numerolla 2

(KUVA 1). Aurinkokeräimen vesikierto toimii erillisenä kiertona järjestelmässä. Jäähdyttävä vesi kiertää kierukassa, joka on merkitty numerolla 4 ja joka sijaitsee kuvassa 1 (KUVA 1). Kierukka sijaitsee kuvassa 1 (KUVA 1) näkyvän putken sisällä, lämmitettävän putken ympärillä (KUVA 2). Suuremman putken sisällä on kierukan lisäksi tislausputki, johon syötetään alakautta suolavesi. (Manderbacka 2018; Sander 2018a.)

KUVA 2. Kierukka ja tislausputki

Tislausputki, kierukka sekä lämmitys-, syöttövesien ja prosessituotteen liitännät irrotettuna järjestelmästä (KUVA 2). Putkistoon (KUVA 1) kohta 4, tuotetaan normaalipainetta alempi paine myöhemmin esiteltävin menetelmin. Prototyypin suolanpoistomenetelmä perustuu alipainetislaukseen. Alipaineella veden ominaislatenttilämpö on niin sanotusta normaalitilasta poikkeava. Alipaineen myötä veden höyrystymislämpötila on alhaisempi, verrattuna normaalipaineessa tarvittavaan lämpötilaan. (Inkinen & Tuohi 2006, 386–387; Manderbacka 2018; Sander 2018a; ThermExcel 2003.)

6 PROTOTYYPIN TEOREETTINEN TOIMINTAPERIAATE

Prototyypin prosessimenetelmä on vasta kehitysvaiheessa. Laite on rakennettu prosessimenetelmän testaus-, tutkimus-, ja mahdollisiin kehitystarpeisiin. Ohjausjärjestelmä rakennetaan muun muassa tukemaan sekä edesauttamaan näiden tutkimuskohteiden tuloksien analysointia. Järjestelmän tehtäviin lukeutuu myös komponenttien ohjaus, jolla vaikutetaan itse prosessin kulkuun. Itse prosessimenetelmään tai siitä syntyviin analyyttisiin tuloksiin ei palata ainakaan syventävästi enää myöhempänä. (Manderbacka 2018; Sander 2018a.)

Järjestelmän alipaine tuottaa laitteiston toiminnan kannalta energian näkökulmasta muutamia hyötyjä. Suolavesi kulkeutuu tislausputkeen alipaineen avulla. Veden raja-arvoa korkeuden suhteen tislausputkessa säädetään alipaineen voimakkuuden mukaan sekä myöhemmin esiteltävin mekaanisin keinoin. Paineen ollessa absoluuttisen nolla-arvon ja normaalipaineen, eli noin yhden baarin välisellä painealueella: järjestelmässä saavutetaan edellä mainittu alhaisempi veden höyrystymis- sekä kiehumispiste. Yksi prosessiin liittyvistä tavoitteista on saada vesi höyrystymään noin 60–70 celsiusasteen lämpötilassa. (Manderbacka 2018; Sander 2018a.)

Energian näkökulmasta suolaveden liikuttamiseen järjestelmän sisällä tarvittava energia saavutetaan tyhjiöpumpun tuottamalla paineella. Lisäksi lämpöenergian tarve höyrystymispisteen saavuttamiseksi pienenee. Höyrystyessään vesihöyry nousee tislausputkessa ylöspäin saavuttaen mekaanisen elementin. Kyseinen elementti edelleen mahdollistaa vesihöyryn poistumiseen ulomman putken (KUVA 1 kohta 4), sisäpinnalle. Ulomman putken sisäpinnalle päästyään vesihöyry jäähtyy sekä muuttuu vesipisaroiksi. Edelleen luonnollisten olosuhteiden myötä pisarat valuvat alas ja lopulta päätyvät erilliseen puhtaan veden säiliöön. Kun tislausputkessa oleva suolavesi kyllästyy suolamäärän kasvaessa, se kulkeutuu sille erikseen tarkoitettuun säiliöön. (Manderbacka 2018; Sander 2018a.)

7 TYÖRYHMÄN VALITSEMAT KOMPONENTIT

Suurin osa työryhmän valitsemista komponenteista on hankittu yrityksen edellisiin projekteihin, ja syystä tai toisesta ne ovat jääneet ylimääräisiksi. Esimerkiksi moottoriventtiilit ovat toisesta projektista ja niiden hankintaan on vahvasti liittynyt hinta. Moottoriventtiilit on kilpailutettu, ja halvimman tarjouksen antanut toimija on valittu tuotteen toimittajaksi. Esimerkkinä toimii moottoriventtiilien hinta-erot, sisältäen rungon ja moottorin; Toimittaja 1: yli 500 euroa, toimittaja 2: 38 euroa toimitettuna. Hinnat ovat kappalehintoja. (Sivula 2018.)

Allekirjoittaneella on ollut täysi mahdollisuus vaikuttaa suurimpaan osaan tässä alaluvussa esiteltävien komponenttien valintaan tai muutoksiin. Tarvetta muutoksiin ei ole kuitenkaan ollut. Taustalla on myös vaikuttanut omien vahvuuksien tunteminen, joka ei ole parhaimmillaan putkistotekniikassa. Lisäksi etenkin sähköisillä komponenteilla on ollut rajoitteena fyysiset liitännät, toiminta tietyissä olosuhteissa sekä ohjausjärjestelmän käyttöjännite. Tehtävänä oli sulauttaa järjestelmä yhdenmukaiseksi ohjelmallisesti että sähköisesti parhaaksi katsotulla tavalla. Alla on esitetty kaupalliset sähköiset komponentit ja anturit, jotka liittyvät jollain tavalla ohjausjärjestelmään. Joidenkin komponenttien valmistajia ei tunneta, ja osa komponenteista on hankittu prototyypin haltijan varastosta. Liitteen 1 (LIITE 1) dokumenteista ilmenee tässä luvussa esiteltyjen komponenttien kytkentäperiaatteet.

Nesteen tasoanturit on sijoitettu suolaveden syöttösäiliön yhteyteen, fyysisesti noin 5 cm päähän toisistaan. Anturit ovat tyyppiä I/O, ja ne eivät tarvitse sähkövirtaa toimiakseen. Anturin lepoasento on normaalisti auki, eli ”O”. Kun nestettä on anturin kohdalla, se muuttaa asentoaan tilaan ”I”, eli kiinni. Laitteistossa tunnetaan putkistojen ja säiliöiden tilavuudet. Anturien sijoittelun ansiosta ne ilmaisevat myös suolaveden tason muualla järjestelmässä. (Manderbacka 2018; Sander 2018a.)

Magneettiventtiili on sijoitettu putkistoon ja sen avulla hallitaan alipaineistusta. Paineella vaikutetaan myös nesteen liikkumiseen järjestelmässä. Magneettiventtiili muuttaa tilaansa ohjattaessa ja ohjauksen lakatessa. Lisäksi järjestelmään on liitetty moottoriventtiileitä, joiden malli on HH-FLO, T24-B2-B. Moottoriventtiileillä hallitaan suolaisen veden poistoa järjestelmästä. Poistoputkessa on kaksi moottoriventtiiliä, jotta poistotoimenpide voidaan suorittaa häiritsemättä järjestelmässä vallitsevaa alipainetta. Moottoriventtiilit pitävät viimeksi ohjatun asennon. Moottoriventtiilin ohjaus on kolmijohtiminen, jossa maa on yhteinen. Venttiileiden ohjausjännite on 12 VDC. (Manderbacka 2018; Sander 2018a.)

Kiertovesipumppu on sijoitettu lämpökiertoon ja sillä kierrätetään aurinkokeräimen lämmittämää nestettä järjestelmässä. Pumpun maksimikapasiteetti on 800 litraa tunnissa, ja sen suurin toimintalämpötila on +120 celsiusastetta. Kiertovesipumpulla tuotettavalla virtauksen määrällä pidetään tai pyritään pitämään osaltaan huoli myös siitä, että lämmitysneste ei kiehu. Lisäksi järjestelmässä on alipainepumppu, jonka kapasiteettia ei tunneta. Pumppujen käyttöjännite on 12 VDC. (Manderbacka 2018; Sander 2018a.)

Painesensorit mallia (MPX4250AP, 867B-04) on sijoitettu mittaamaan painetta tislausputkessa. Painetta mitataan ulommassa putkessa (KUVA 1 kohta 4), sekä meriveden sisäänotossa. Sensorien käyttöjännite on noin 5 VDC. Lineaarinen jänniteviesti on tyypillisesti välillä 0,25–4,75 VDC, suhteessa absoluuttiseen paineeseen. Anturin painemittauskyvyn ollessa tyypillisesti välillä 0,2–2,5 bar. (Freescale Semiconductor Inc 2009; Manderbacka 2018; Sander 2018a.)

Suoraan ohjausjärjestelmään kytketyt lämpöanturit pohjautuvat digitaaliseen 1-Wire -tekniikkaan. Käytössä on yksittäisiä kaapeliantureita sekä Dimense Oy:n valmistama lämmön ja paineen kestävä anturikaapeli. Molemmat ratkaisut perustuvat DS18B20 -piiriin. (Dimense Oy 2018; Maxim Integrated 2015; Manderbacka 2018; Sander 2018a.)

Energiamittarina toimii Sharky 775. Mittarin ja keskusyksikön välinen tietoliikenne tapahtuu M-bus -protokollan mukaisesti. Lisäksi meno-paluu-puolen lämpötilojen mittaukset tapahtuvat PT 500 -antureilla ja virtaussensorina toimii mittarin sisäinen ultraäänisensori. Mittari on paristotoiminen ja pariston käyttöiäksi on valmistaja määrittänyt vähintään yksitoista vuotta. (DIEHL Metering 2012; Manderbacka 2018; Sander 2018a.)

Aurinkopaneelina toimii kiinalaisvalmisteinen Elfelandin valmistama 30 W -tehoinen 12 voltin tasajännitettä tuottava ratkaisu. Aurinkopaneeli tullaan suurella todennäköisyydellä jossain vaiheessa päivittämään. Päivitysajankohta sijoittuu todennäköisesti opinnäytteen jälkeiseen aikaan. (Manderbacka 2018; Sander 2018a.)

Aurinkopaneelin ja akun välisenä lataussäätimenä toimii MPPTSUN-brändin SWC30A-mallinen lataussäädin. Lataussäädin on konfiguroitavissa eri akkutyypeille ja sen maksimi latausvirta on 30 ampeeria. Säätimestä voidaan ottaa laitteistolle 12 VDC käyttövirtaa maksimissaan 20 ampeerin verran. (Dongguan Sunworld Co., LTD 2017; Manderbacka 2018; Sander 2018a.)

8 SÄHKÖISTÄMIS- JA KESKUSRATKAISU

Tässä luvussa paneudutaan hieman syvemmin opinnäyttetyön yhteen osa-alueeseen eli sähköistysratkaisuun. Mainitut kaapelien kuormitettavuudet on määritetty tulkitsemalla ajantasaista pienjännitestandardia. Vaikka laitteisto on rakennettu toimimaan pienoisjännitteellä, on se tehty pienjännitestandardin mukaisesti suurimmilta osin.

8.1 Sähköistäminen ja siihen liittyvien komponenttien valintoihin liittyneet seikat

Järjestelmän käyttöjännitteeksi on valittu akkukäytöllä 12 VDC. Järjestelmän pääsulake (LIITE 1), joka on sijoitettu akun ja lataussäätimen väliin, on mitoitettu 40 ampeerin kuormitettavuudelle. Lataussäätimen maksimilatausvirran ollessa 30 ampeeria ja maksimikäyttövirran rajoittuessa 20 ampeeriin pääsulake on valittu suojaamaan suoraa oikosulkua, joka voi tapahtua lataussäätimen ja akun välillä. Syöttö-, ja päävirtakaapelit ovat poikkipinnoiltaan 6 millineliömetriä. Pienjännitestandardin referenssiasennustavan C mukaan 6 mm^2 kaapelin kuormitettavuus on 46 ampeeria. Referessiasennustapa toimii piirin toteutuneiden hankalimpana asennustapaolosuhteena. (Dongguan Sunworld Co; SFS 6000-5-52 2017, 16; SFS 6000-5-52 2017, 36; SFS 6000-5-52 2017, 38.)

Kuten edellä todettu, lataussäätimen kautta saatava maksimikäyttövirta järjestelmän käyttöön on enintään 20 ampeeria. Prototyypin keskuksen käyttämä ja sen kautta kulkeva sähköinen kuorma otetaan kokonaisuudessaan lataussäätimen kautta, siihen tarkoitetuilta liittimiltä. Keskuksen ja lataussäätimen välisen syötön sulake on mitoitettu 15 ampeerin kuormalle (LIITE 1). Syötön sulake suojaa mahdollisilta vikatilanteilta sekä on oikosulkusuojana keskuksen ja lataussäätimen välillä. (Dongguan Sunworld Co; SFS 6000-5-52 2017, 16; SFS 6000-5-52 2017, 36; SFS 6000-5-52 2017, 38.)

Ryhmäsulakkeet on mitoitettu arvioimalla komponenttien kuormaa. Tiedossa on ollut, että keskusyksikkönä toimivan Raspberryn käyttämä virta on maksimissaan 2,5 ampeeria ja valitun relekortin käyttövirta on noin ampeerin luokkaa. Keskuksen päävirtapiiri (LIITE 1) on jaettu viiteen eri ryhmään, joista kolme ensimmäistä ovat käyttölaitteiden sulakkeet mitoitukseltaan kolme ampeeria. Kaksi viimeistä ryhmää ovat releohjauksiin, ja ne ovat mitoitukseltaan viisi ampeeria. Relekortin kärkiryhmittelyä on pyritty jakamaan tasaisesti kahteen sulakeryhmään. Jatkuvasti pyörivät komponentit ovat sijoitettuna samaan ryhmään. (KMtronic a; Raspberry Pi Foundation.)

Prototyypin korkeus lähentelee noin kuutta metriä, joten runko päätettiin maadoittaa (LIITE 1) mahdolliselta salamaniskulta. Prototyyppi on sijoitettu tontilla teollisuushallin läheisyyteen, ja sen korkeus ylittää rakennuksen harjakorkeuden. Kymmenen metrin säteellä on myös muutamia puita, jotka ovat korkeampia kuin prototyyppi. Laite painaa arviolta muutama sata kiloa, se on suurimmaksi osaksi terästä ja se on asetettu asfaltin päälle. Jos salama iskee suoraan laitteeseen, suojaus ei todennäköisesti suojaa laitteistoa rikkoontumiselta. Suojaus on päätetty asentaa ympäröivän omaisuuden suojaksi. Suojauksen myötä salaman osuessa sen energia todennäköisesti liikkuu lyhyintä reittiä maahan, eikä esimerkiksi valokaaren muodossa vaikuta ympäröivään omaisuuteen. Suojaustavaksi valittiin 16 mm^2 poikkipintainen kuparikaapeli kiinnitettynä terässauvaan. Kuparikaapeli on kiinnitetty prototyypin runkoon ja terässauva on edelleen upotettu maahan, noin 0,5 metrin syvyydelle. Sijainnista ja sen todennäköisestä vaihtumisesta johtuen maadoitusta ei voitu toteuttaa rengaselektrodityyppisesti. (SFS 6000-5-54 2017.)

8.2 Keskuksen ja sen toiminnallisten komponenttien valintaan liittyvät seikat

Keskuksen kotelon valintaan liittyi kaksi päävaatimusta: koko sekä veden kestävyys. Koko mitoitettiin siten, että kaikki komponentit olivat keskuksen sisään sijoitettavissa. Lisäksi muut keskuksen kiinteät elementit kuten kaapelikouru oli helposti asennettavissa. Keskuksen kotelointiluokaksi on määritetty IP 66. Kotelointiluokka kertoo keskuksen olevan pölytiivis ja veden sisäänpääsyn suojauksen tason kestävän voimakkaan vesisuihkun. Prototyyppi on sijoitettuna ulkotilaan, joten kotelointisuojaus tuli ottaa tämän vuoksi huomioon. (Sähköturvallisuuden Edistämiskeskus ry.)

Tiedossa oli, että Raspberryn käyttöjännite on 5 VDC ja maksimi käyttövirta on 2,5 ampeeria. Raspberry oli valittu keskusyksiköksi aiemman kokemuksen perusteella. Aluksi työryhmän sisällä pohdittiin tupakansytyttimen pohjaa ja siihen sopivaa regulaattorimallia. Keskusyksikön käyttöjännitteen regulaattoriksi valittiin toisenlainen ratkaisu. Regulaattoriksi valittiin kolmijohtoinen valmispakattu regulaattori, josta kaksi johdinta käsittää syötön ja yksi johto reguloidun jännitteen. Reguloitu jännite saadaan käyttöön regulaattorista USB-liittimestä. Valintojen hinnoiksi muodostui yhteensä hieman yli 40 euroa. (Amazon.com, Inc; Raspberry Pi Foundation, USB; Raspberry Pi Foundation, Power Supply.)

Analogisten ja digitaalisten sensorien lukuun valittiin erillinen Arduino -kehitysalusta aiemman kokemuksen perusteella. Analogisia sensoreita ei pysty suoraan lukemaan Raspberryllä, sillä sen ominaisuuksiin ei kuulu analogista muunninta. Anaogisten sensoreiden luku Raspberryllä olisi mahdollista li-

säämällä Raspberryyn erillisen konvertterin. Erillinen konvertteri olisi kuitenkin johtanut ainakin kahteen vaihtoehtoiseen ongelmaan: USB-väyläinen konvertteri olisi vienyt yhden USB-portin. GPIO-väyläinen konvertteri olisi vaatinut Raspberryn kotelon pois ottamista ja mikropiirin vahingoittumisriski olisi ollut suurempi. Dokumentaatiossa varoitetaan muun muassa kytkemästä liian suuria jännitteitä (yli 3,3 VDC), joka saattaa johtaa ongelmiin mikroprosessorin toiminnassa. Arduino sen sijaan kykenee tulkitsemaan analogisia jänniteviestejä 5 VDC asti. Arduinon valinnan myötä ongelma ratkaistiin muutamalla eurolla. (Raspberry Pi Foundation.)

Relekortti valittiin yhden työryhmän jäsenen ehdotuksesta. Taustalla vaikutti myös myyjäorganisaation tarjous kahdesta relekortista ja yhdestä RS-485 muuntimesta. Suunnitteluasteella pohdimme työryhmässä, että suolanpoistajassa saattaa olla tarve kahden relekortin kärjille, eli kuudelletoista relelähdölle. Vastaavaa relekorttia käytetään toimeksiantajan toisessa projektissa, joten jos suolanpoistajassa ei ole sille käyttöä niin sijoituspaikka kortille varmasti löytyy. Valinnan hinnaksi muodostui reilu sata euroa. (KMtronic a.)

Lämpötilasensorit käyttävät 1-Wire tekniikkaa ja niiden luku onnistuisi monella eri tapaa valitulla ympäristöllä. Sensorien luku onnistuisi Arduinolla, että myös Raspberryllä. Molemmissa tapauksissa kytkentä vaatisi ylösvetovastuksen. Kuten edellä esitetty, Raspberryn GPIO liitännät ovat erityisen herkkiä tietyille sähköisille ilmiöille. GPIO herkkyys sekä laitteiston sijoitus ja testausympäristö saattaisivat jossain vaiheessa tietää ongelmia. Arduinolla luettaessa sensoreita, olisi tietänyt lisää juotoksia ja mahdollisesti tulevaisuuden ongelmia. Lämpötilasensorien lukijaksi valittiin USB-porttiliitännällä varustettu DS9490R 1-wire-adapteri. Valinnan hinnaksi muodostui noin kolmekymmentä euroa. (Arduino; Maxim Integrated 2011; Raspberry Pi Foundation.)

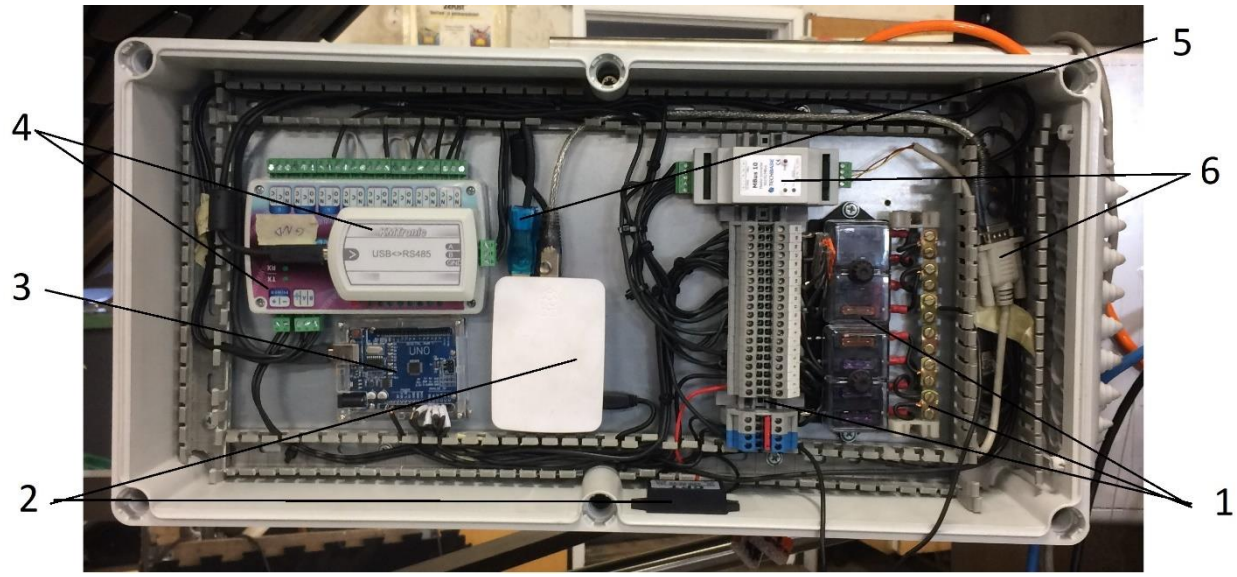
Työryhmän valitseman energiamittarin, Sharky 775:n M-Bus väylän lukeminen asetti hieman haasteita. Edellä esitetty järjestelmän käyttöjännite (12 VDC), tuotti suurimman haasteen löytää ympäristöön sopiva konvertteri. Muita ehdottomia vaatimuksia konvertterin suhteen oli: sen täytyy toimia Linux -ympäristössä, tietoa pitää pystyä tulkitsemaan komentorivipohjaisesti sekä hinnan piti pysyä kohtuurajoissa. Selvitin myös Sharkyn RS-232 -kortin liittämistä M-Bus kortin tilalle. Valmistajan henkilökohtaisen palvelun myötä totesin, että protokollan vaihtaminen ei ole sopiva vaihtoehto suolanpoistajan ympäristöön. RS-232 -kortti olisi vaatinut energiamittarin liittämistä sähköverkkoon erillisellä virtalähteellä, invertterin liittäminen järjestelmään ei kuitenkaan ollut mielestäni tässä tapauksessa edes vaihtoehto. Lopulta vaatimusten mukainen konvertteri löytyi ja valinnaksi muodostui Techbasen valmistama Mbus10, joka on M-bus RS-232 -standardiversiokonvertteri. Valmistaja lupaa konvertterin toimivan

vielä noin 10 VDC jännittellä. Valintaan liittyi lisäksi myös USB – RS-232 -muunnin, jotta viestin saa tulkittua keskusyksikön päässä. Tiedon tulkintaa varten valittiin Atenin valmistama RS-232 standardiversion UC-232A -mallin muunnin. M-Bus -protokollan mukaiseen sanoman tulkitsemiseen liittyvien komponenttien hinnaksi muodostui noin sata euroa. (ATEN International Co., Ltd 2017; DIEHL Metering 2012; Techbase Group Sp.z o.o.)

8.3 Keskuksen komponentit ja kaapeloinnit

Keskuskoteloksi valittiin Enston Cubo C -sarjasta sopivan kokoinen IP 66 -luokan kotelo sekä tarvikkeet. Tässä luvussa esitellään keskuksen kaapelivalinnat sekä keskuskotelon sisään sijoitettavat komponentit ja laitteistot. Keskuksen sisäisessä johdotuksessa on käytetty H07 V2-R -tyyppistä, poikkipinnaltaan 1,5 mm^2 monisäikeistä, yksijohtimista kuparikaapelia. Johtimen kuormitettavuus referenssiasennustavalla B2 on enintään 16,5 ampeeria. Tästä kaapelityypistä on poikettu joidenkin komponenttien osalta sekä 1-Wire -väyläkaapeloinnissa, jotka esitellään myöhemmin. (STK-Tietopalvelut Oy; SFS 6000-5-52 2017, 28; SFS 6000-5-52 2017, 38.)

Numeromerkintöjen (KUVA 3) komponentit/laitteistot löytyvät samanarvoisista alaluvuista. Laitteiston sähköistysratkaisu on dokumentoitu ja esitetty liitteen 1 (LIITE 1) dokumentaatiossa, eikä siihen paneuduta yksilöllisesti enempää kuin edellä ja seuraavassa on kerrottu. Joitain komponenttien kaapelityyppejä ei ole tässä luvussa esitetty, mutta ne löytyvät edellä mainitusta dokumentaatiosta. Sähköistäminen ja siinä esitetyt suojalaitteet perustuvat samaan periaatteeseen kuin keskuksen sähköinen toiminta. Sähköisillä ja erityisesti suojalaiteratkaisuilla yritetään valmistautua mahdollisiin vikatilanteisiin mahdollisimman hyvin sekä mahdollistamaan laitteiston turvallinen käyttö. Vikatilanteen sattuessa toiminnan tulee edelleen mahdollistamaa poiskytkentä. Vian ilmetessä järjestelmä on suunniteltu siten, että se käsittelee vikatilanteen aiheuttamatta laitteistolle, sen komponenteille, laitteille tai ympäristölle fyysistä tai muuta haittaa.



KUVA 3. Laitteistot ja komponentit asennettuna keskukseen

8.3.1 Komponenttien suojaus ja potentiaaliryhmittely

Keskukseen on sijoitettu pienjännitekäytössä ollut nolla-/maadoituskiskon osa (KUVA 3, kohta 1). Kiskossa on liitäntäraudat, joihin on helposti kytkettävissä erikokoisia johtimia. Tässä tapauksessa kytkettävinä johtimina ovat 1,5 mm^2 sekä 6 mm^2 poikkipintaiset johtimet. Tässä keskusratkaisussa kisko toimii virroittavana VDC+ -kiskona. Kiskon tehtävä on toimia rinnankytkentäalustana juuri ennen sulakeryhmittelyä. Lisäksi näkyvillä on sulakepidin normaalikokoisille lattasulakkeille. Sulakepitimellä toteutetaan edellä mainittu sulakeryhmittely, joka ilmenee tarkemmin liitteestä 1 (LIITE 1). Riviliittimillä on toteutettu edelleen ryhmittelyn liitokset sekä nollapotentiaalin liitokset.

8.3.2 Raspberry Pi sekä regulaattori käyttöjännitteelle

Valmispakattu regulaattorista otetaan Raspberryn käyttöjännite, jonka sisääntulo on 12 VDC ja ulostulo viisi volttia tasajännitettä kolmen ampeerin virralla (KUVA 3, kohta 2). Raspberry Pi toimii järjestelmän keskeisimpänä ja kriittisimpänä laitteena. Jos Raspberry menee rikki tai jää sellaiseen tilaan, että sitä ei pysty käyttämään, koko suolanpoisto on vailla ohjausta. Raspberryn kautta ohjataan kaikkia järjestelmään liitettyjä sähköisiä komponentteja ja suolanpoistoprosessi tulee käsitellyksi ohjelmallisesti siinä. Ohjelmallinen puoli on toteutettu kirjoitushetkellä tuoreimmalla Raspbian käyttöjärjestelmällä, joka pohjautuu Debian Linux -jakeluun. (Amazon.com, Inc; Raspberry Pi Foundation.)

Raspberry Pi:tä voi kutsua taskukokoiseksi tietokoneeksi. Raspberry on asetettu toimimaan siten, että se etsii ja yhdistää automaattisesti WiFi -yhteyden sijaintipaikan reitittimeen. Raspberry model 3B:n ydin rakentuu ARM Cortex A53 -neliydinsuorittimeen. Varsinaiseksi tehotietokoneeksi siitä ei ole, mutta se kestää kyllä hyvin yksinkertaisten prosessien ajoa. Raspberry on täynnä mahdollisuuksia, mutta yksi heikkous lukeutuu sen muistijärjestelmään. Rasperry käynnistää ja ajaa käyttöjärjestelmän muistikortilta. Ajan x kuluttua vaarana on mahdollisuus muistikortin korruptoineisuuteen, joka tarkoittaa muistisäikeiden vioittumista. Järjestelmäkäynnistys muistikortilta tuo myös mukanaan ongelman mahdolliseen tuotteistukseen. Kuka tahansa, joka pääsee muistikortin käsiksi, saa sieltä kaiken haluamansa tiedon myös ulos. Tietoa ei voi salata juurikin käynnistyslohkon sijainnin vuoksi. On tapoja muuttaa fyysisesti tiedon tallennusmenetelmää, mutta hyvätkään harrastevälineet eivät välttämättä luo parhaimpia tuloksia. Prototyypin kehityksen tukena Raspbery on kuitenkin loistava alusta. (Raspberry Pi Foundation, BCM2837.)

8.3.3 Arduino sensorilukualustana

Tässä projektissa Arduino (KUVA 3, kohta 3) toimii sensorilukualustana edellä esitellyille kahdentyyppisille antureille. Arduinon tehtävänä on käskettäessä lukea arvoja ja palauttaa anturin portista saatu tieto vastasanomana. Arduinon yksilölliseen ohjelmarakenteeseen palataan hieman myöhemmin tässä työssä. Arduino on yhdistetty Raspberryyn USB-väylän kautta. Väylä muodostaa laitteiden välille rajapinnan sarjamuotoiseen kommunikointiin, ja se toimii myös Arduinon käyttöjännitteen lähteenä. Luettavat sensorit on juotettu tarkoitukseen sopivin juotospaloin yksilöllisille paikoilleen. Painesensorit on kytkettynä analogisiin portteihin ja tasosensorit digitaalisiin portteihin. Kytkennät ilmenevät toteutusta varten tehdyistä piirustuksista (LIITE 1). Jokainen anturiportti käsittää fyysisen alasvetovastuksen, jolloin voidaan todeta esimerkiksi johdon katkeamistilanteet. Alasvetovastuksen tehtävänä on siis ”vetää” portin tila ”LOW” -tilaan. jos jännitettä ei ole. Vastuskytkennällä estetään jännitteen kelluminen. (Arduino.)

8.3.4 KMtronic-relekortti ja RS485-muunnin

Relekorttina toimii KMtronicin digitaalinen ratkaisu (KUVA 3, kohta 4). Relekortteja voidaan tarvittaessa lisätä siten, että saavutetaan 255 kappaletta vaihtokosketinlähtöjä. Jokainen vaihtokosketin on yksilöity kortilla DIP-kytkin valinnalla. DIP-kytkimellä valitaan relekortin tunnistenumero. Relekorteissa

voidaan ohjata yksilöllistä kärkeä joko desimaalilukuohjauksella tai heksadesimaaliohjauksella. KMtronic valmistaa myös muita vastaavia relekortteja kuin RS-485 -ohjattavia. Relekortin käyttöjännite on 12 VDC ja sen kärjillä pystyy ohjaamaan maksimissaan 250 VAC 10 ampeerin kuormaa. Tässä tapauksessa Raspberryn ja relekortin välillä tarvitsee erillisen USB – RS-485 -muuntimen. Muunnin kytketään Raspberryyn USB-kaapelilla ja kolmella johtimella relekortille. Muunnin ei tarvitse kolmansien osapuolien ajureita, vaan se muuntaa suoraan USB-viestin RS-485 -muotoon. (KMtronic a; KMtronic b.)

8.3.5 DS9490R 1-wire -adapteri

Projektissa käytetään 1-Wire-adapterina mallin DS9490R-adaperia (KUVA 3, kohta 5). Adapteri kytketään Raspberryn USB -porttiin ja lämpötila-anturilinjaan RJ11 -liitännällä käyttäen kolmea johdinta. Johtimet ovat V+, GND ja Data. Yhteen 1-Wire -linjaan pystyy liittämään kymmeniä antureita, sillä kaikilla antureilla on oma yksilöllinen 64-bittinen osoitteensa lukua varten. Väylän heikkous liittyy suuriin massaluentoihin. Vikatilanteen ilmentymä saattaa olla yhden anturin oikosulku, jolloin väylä ei palauta yhdenkään kytketyn anturin tietoa. Oikeastaan suurien järjestelmien ainoa tapa selvittää viallinen anturi on kytkeä antureita irti yksi kerrallaan siihen asti, kunnes luku taas onnistuu. DS9490R:n 1-Wire -käyttö Raspberryllä edellyttää erillisen ohjelmistorakenteen asentamista. (Maxim Integrated 2011.)

8.3.6 M-Bus-konvertteri ja RS232-muunnin

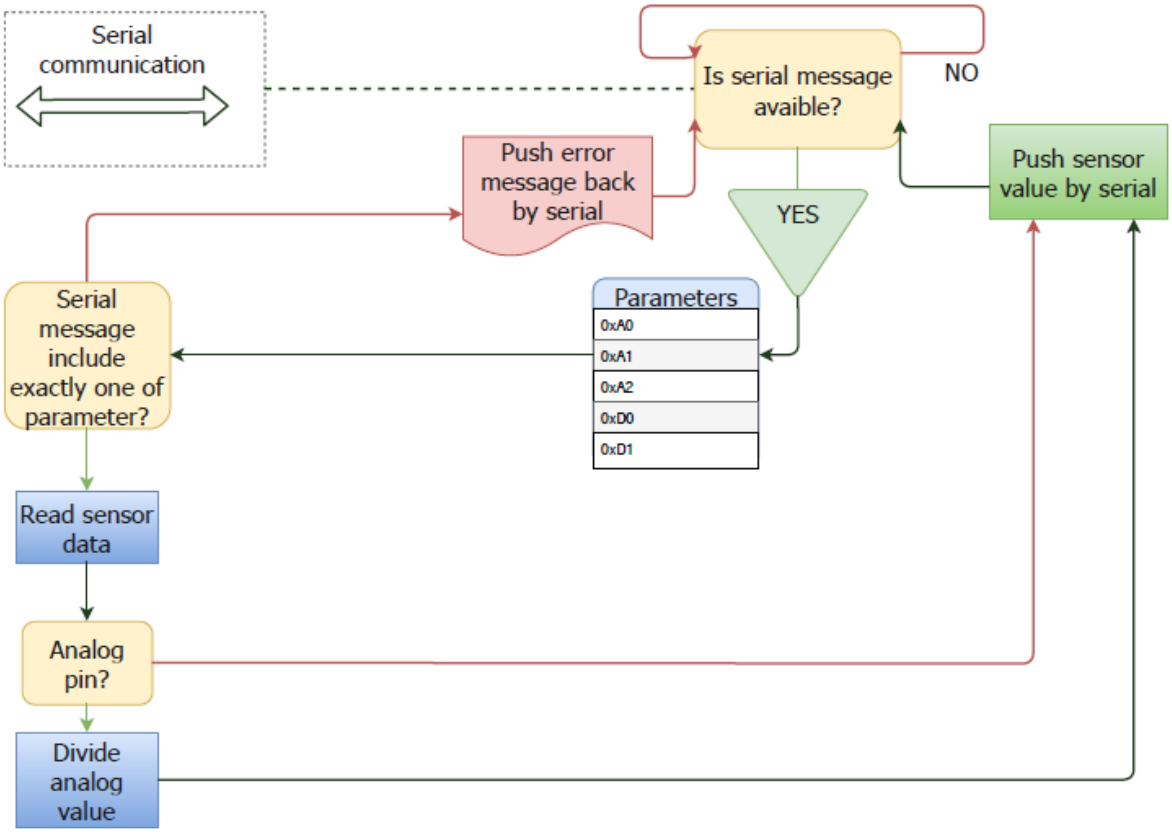
M-Bus-konvertteri muuntaa edellä esitetyn Sharky 775:n tietoliikenneviestit RS-232 -muotoon. Sanomat edelleen muunnetaan M-Bus -konvertterin ja Raspberry Pi:n välillä RS-232 – USB -muuntimella, tämä muunnin on valittu mBus10 -konvertterin vaatimusten mukaisesti. Fyysisesti RS-232 -muuntimesta on käytössä tässä tarkoituksessa seuraavat johtimet: Tx, Rx ja GND. Eli lähetys, vastaanotto ja maa, jotka on toteutettu erillisen johtoadapterin avulla. Markkinoilla on myös TTL-tason USB – RS-232 -muuntimia. Tässä työssä potentiaalieron ollessa yli viisi volttia konvertterin välittämänä saattaa johtaa TTL-tason muuntimen vahingoittumiseen, pahemmassa tapauksessa koko keskusyksikön rikkoutumiseen. (ATEN International Co., Ltd 2017; Future Technology Devices International Limited (FTDI) 2016; Techbase Group Sp.z o.o.)

Sharky 775:n M-Bus -kortissa on kahden johtimen liitäntä, ja vastaavasti samat liitännät löytyvät konvertterista. Näillä laitteilla toimiessa sillä ei ole merkitystä, miten päin liitännät kytketään laitteiden välille, viesti kulkee joka tapauksessa. Raspberryllä ajettava ohjelmaratkaisu eikä viestisisällön analysointi, eivät ikävä kyllä ole yhtä mutkattomia, mihin palataan myöhemmin tässä työssä. (DIEHL Metering 2012; Techbase Group Sp.z o.o.)

9 ARDUINON SOVITTAMINEN OSAKSI JÄRJESTELMÄÄ

Arduinon tehtävä on siis käytännössä hallita anturilukuprosessi tiettyjen järjestelmään liitettyjen sensorien osalta. Kysyttäessä Arduino välittää tietoa eteenpäin Rasperrylle. Anturit koostuvat kolmesta analogisesta painesensorista sekä kahdesta I/O -tasosensorista. Kommunikointi tapahtuu USB-väylän kautta ja liikenne on sarjamuotoista. Sarjaliikenteen nopeudeksi on määritetty 19200 baudia.

Arduino suorittaa ohjelmaa (LIITE 2) ikuisessa silmukassa, ja kun sisään tulevaa liikennettä havaitaan, välitetään sanoma valueRet-funktiolle, joka lukee yksilöllisen anturin tilan tai tason. Funktio edelleen palauttaa luetun tiedon ikuiseen silmukkaan. Jos kyselyparametri on oikea, vastataan anturin tila- tai vastaavasti jännitetasosanoma alkuperäisen sanoman lähettäjälle. Ohjelma on esitetty graafisesti liitteessä 3 (LIITE 3) sekä kuviossa 1 (KUVIO 1), josta saa hieman havainnollisemman kuvan perusperiaatteesta. (Arduino, Serial; Freescale Semiconductor Inc 2009; Manderbacka 2018; Sander 2018a.)



KUVIO 1. Sensorilukualustan toimintaperiaate vuokaaviomallisena

Kuviossa 1 (KUVIO 1) esiintyvät parametrit välitetään tekstimuotoisena ja niitä verrataan tekstimuotoisina. Parametrissä esiintyvät kirjaimet on valittu lähinnä loogisista syistä. Parametrien A- ja D -kirjaimet viittaavat ainakin näin opinnäytetyön tekijän mielestä sanoihin ”analog” ja ”digital”. Edelleen kirjainta seuraava numeerinen arvo viittaa ensimmäiseen käytettävään porttiin tai porttijärjestykseen. Parametrin numerossa alempiarvoinen luku on fyysisesti sijoitettu alempiarvoiseen porttiin Arduinoon.

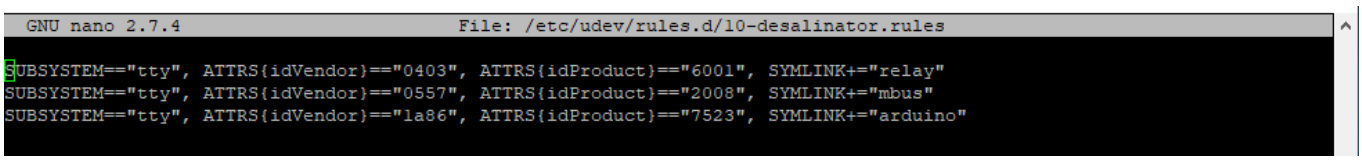
10 RASPBERRYN JÄRJESTELMÄTASON ALUSTUKSET

Järjestelmätason alustukset käsittävät ylläpitäjän toimessa luodut käskykantarakenteet, jotka heijastuvat jäljempänä esitettävien kehitysvaiheiden tulokselliseen suoritustapaan. Lisäksi tässä luvussa esitellään lyhyesti käytetyt ja käytännössä välttämättömät kolmansien osapuolien kirjastot. Kirjastot mahdollistavat myöhempänä esitettävän kaltaisen toteutetun ohjaus- ja sensorilukurajapinnan.

10.1 USB-osoitteellistaminen

Käytettävässä Raspberry -mallissa on fyysisesti neljä USB-porttia, ja ne on tässä tapauksessa kaikki varattu jonkin komennettavan ja/tai kuunneltavan laitteen käyttöön. Kehityksen alkumetreillä ongelmia syntyi porttien loogisen osoittimen parametrin muuttumisen johdosta. Osoitin oli saattanut vaihtaa parametriään yllättäen ja täten osoittaa siis väärään fyysiseen USB-porttiin. Osoittimen muuttumista ei edeltänyt mikään tietty käyttäjän toimi, pois lukien järjestelmän käynnistykset. Normaalisti USB-laitetta tai -laitteistoa hallitaan Raspbian ympäristössä esimerkiksi /dev/ttyUSB1 -osoitteen avulla. (Drake 2008.)

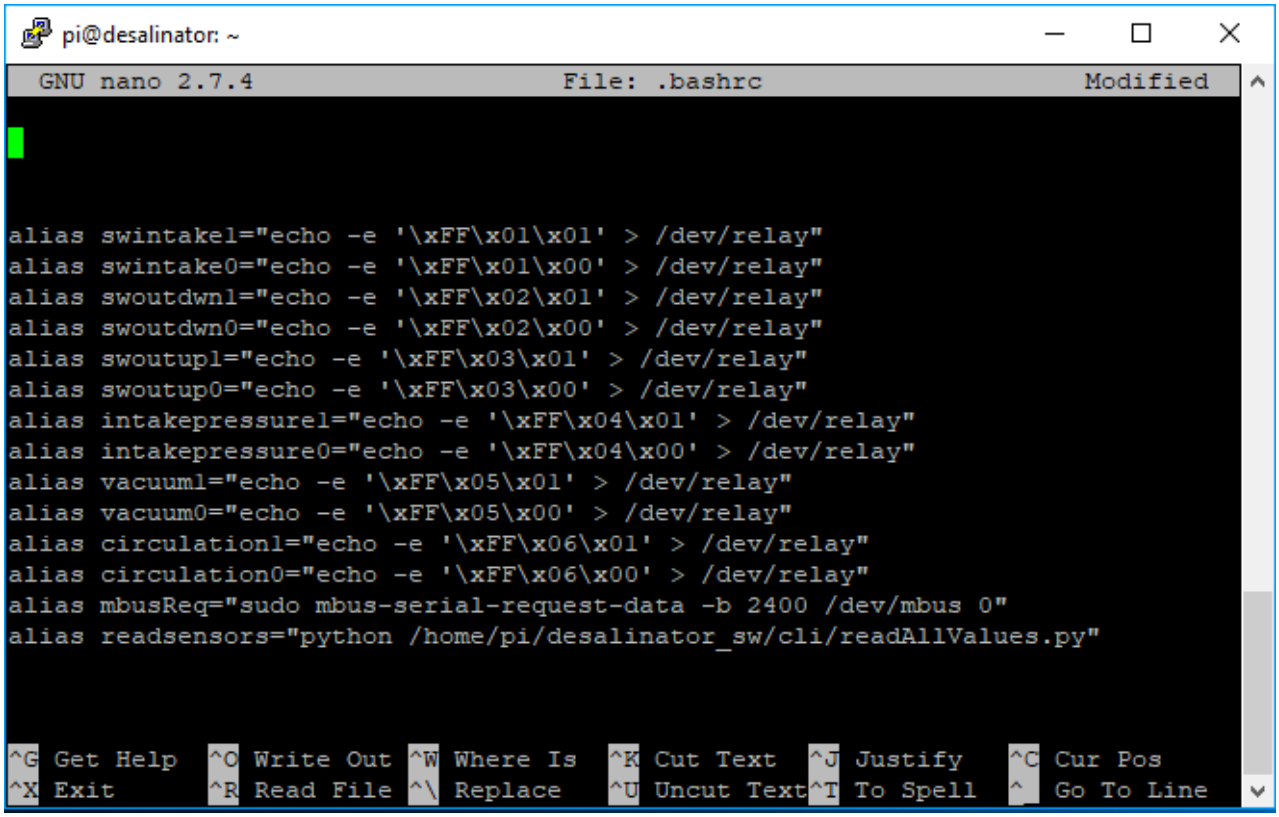
Osoitteellistamisella mahdollistetaan laitteen kutsu omalla, keksityllä parametrillä. Tämä käytetty osoitteellistamistapa muodostaa symbolisen linkin laitteiston ja Linux -ytimen välille, joka edelleen ilmenee käyttäjänäkymässä. Yksilöintitiedot löytyvät laitteen yhteystietoja luettaessa. Tämän työn ratkaisussa (KUVA 4) juuri on /dev/ ja sitä seuraavat nimet asetettiin kyseisten komponenttien tai laitteiden mukaisesti. (Drake 2008.)



KUVA 4. USB-osoitteellistaminen

10.2 Aliasmääritykset

Oikeastaan järjestelmän merkittävämmät käyttäjä-aliakset (KUVA 5) liittyvät KMtronic-relekortin kärkien ohjauskäskyihin, jotka ovat kirjoitushetkellä toteutettuna. Aliaksen avulla voidaan kutsua pitkää käskyjonoa käyttäjän määrittelemällä merkkijonolla. (Frankel 2015; KMtronic b.)



KUVA 5. Aliasmääritykset

Relekortin ohjauskomennot ovat epämukavia kirjoittaa, releen käskykannan ollessa muotoa ”\0xFF\x06\x00”. Komennon yksilöivä osa on heittomerkkien sisällä, joka alkaa ”x06” tarkoittaa kärkeä numero kuusi. Seuraavana, ”x00” tarkoittaa kärjen ohjausta normaalitilaan. Vastaavasti x00 -kohdalla ”x01” tarkoittaa kärjen ohjausta. Lisäksi viimeisenä määritetään, missä osoitteessa ohjauksen looginen osoite, jonka perusteella myös lukijan on helppo käsittää aliasten päämäärä. (KMtronic b.)

10.3 Libmbus-kirjasto M-Bus-lukuun

Libmbus kirjasto toimii en- ja dekooderina M-Bus-laitteiston ja esimerkiksi tietokoneen välillä. Kirjaston avulla pystytään helposti kysymään laitteistolta tietoa M-Bus protokollan edellyttämällä tavalla. Tässä työssä kirjasto vaikuttaa taustalla ja on välttämätön kolmannen osapuolen kirjasto. Libmusin asennus vaatii hieman syvempää tietämystä Linux-käyttöjärjestelmästä. Kirjastoa käytetään suhteellisen yksinkertaisilla komennoilla, ja se palauttaa saamansa tiedot kohtuullisen selkeässä muodossa. Liitteessä 5 (LIITE 5) on esitetty kirjaston avulla saatava data. (Aukia 2013; Domotiga 2014; Stenebo 2017.)

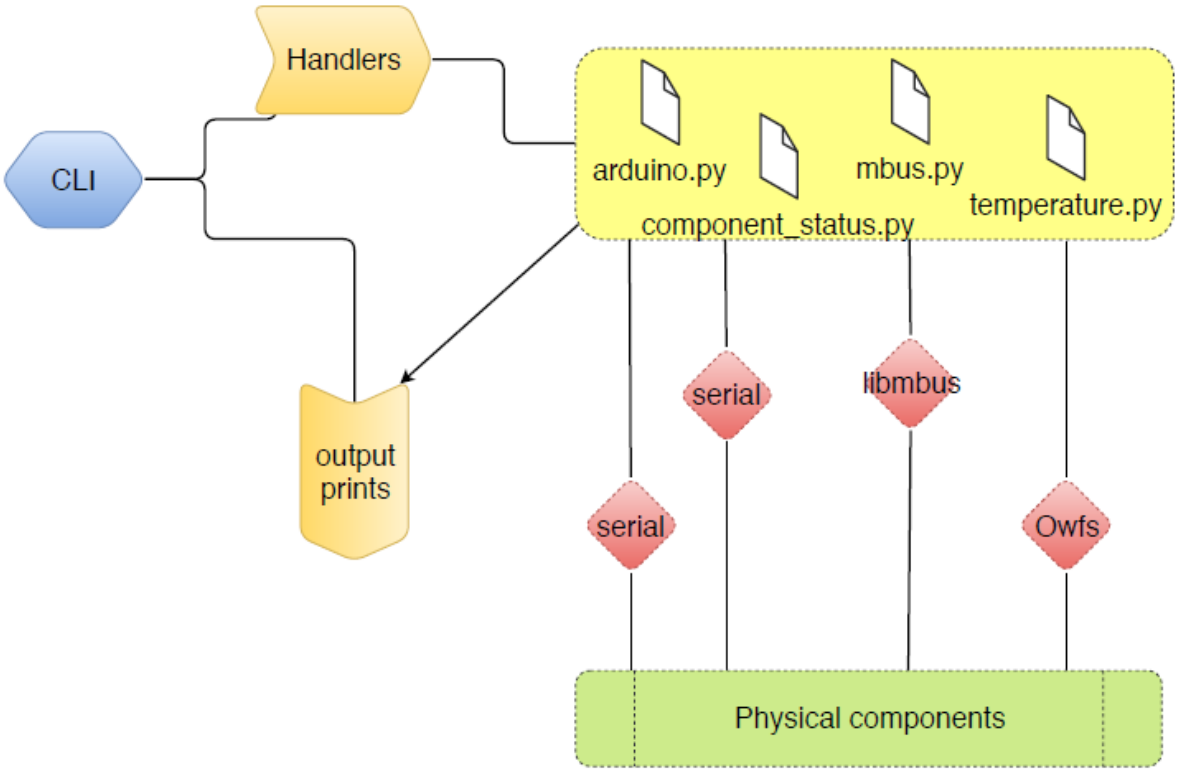
10.4 Owfs-kirjasto 1-wire-lukuun

Owfs-kirjasto tekee laitteeseen virtuaalisen tiedostojärjestelmän. Kirjasto toimii 1-wire -protokollan dekooderina. Tiedostojärjestelmä muodostuu määritettyyn polkuun, ja kun 1-wire -sensoreita havaitaan, tiedostojärjestelmään tulee näkyväksi havaittujen sensorien tiedot. Järjestelmän juuresta katsottuna yksittäinen anturi muodostaa kansion, jonka nimi on anturin yksilöllinen tunniste. Edelleen tunnistekansio sisältää tiedoston ”temperature”, joka sisältää tässä tapauksessa lämpötilatiedon.

Myöhemmin esiteltävä anturin lukutekniikka on hidas, koska se perustuu tiedostojärjestelmän lämpötilatiedostojen arvojen lukemiseen. Lukutekniikka käy siis jokaisen yksilöllisen polun erikseen läpi. Myös tämän kirjaston asentaminen vaatii hieman syvempää Linux-tietämystä. Tiedostojärjestelmään asetettiin erillinen asetuskomento, sillä käyttöjärjestelmän kernel-päivityksen myötä viitatun lähteen tietojen alustuskomento ei toiminut. (Babcock; JST Lawrence 2016; Owfs Development site 2018.)

11 KOMENTOKEHOTEPOHJAINEN KÄYTTÖYMPÄRISTÖ

Järjestelmätason alustuksien lisäksi komentokehotepohjainen sensorinluku sekä komponenttien ohjausympäristö sijaitsevat Raspberryssä. Komentokehotepohjainen käyttöympäristö suorittaa ennalta määrättyjä prosesseja, ja joissain tapauksissa palauttaa arvoja käyttäjän nähtäväksi. Tässä luvussa esitetään Python-käsittelijöitä ja niiden funktioita, jotka toimivat pääohjelma tukena (KUVIO 2).



KUVIO 2. Komentokehote-ohjauksen ohjelmallinen rakenne

Kuviossa 2 (KUVIO 2) esitettynä komentokehoteosuuden ohjelmallinen rakenne. Kerroslajitteluluonnos löytyy koko ohjelmiston tasolta liitteestä (LIITE 9), joka sisältää myös hahmotelman fyysisistä elementeistä. Seuraavissa alaluvuissa esitellään hieman tarkemmin eri käsittelijöiden sekä pääohjelman ratkaisuja sekä toimivuutta kokonaisuudessaan.

11.1 Arduino-käsittelijä

Arduino-kehitysalustan tehtävä oli siis käskettäessä lukea sensoritietoa ja palauttaa luettu arvo kysyjälle. Kyseessä on ensimmäinen tätä työtä varten kehitetty käsittelijä. Nyt käsiteltävä käsittelijä (LIITE 4) sisältää kaksi funktiota, joista kriittisempi liittyy liikenteeseen Arduinon ja Rasperryn välillä. Tarkemmin katsoen käsittelijä sisältää kolme funktiota, joista yksi ei ole käytössä. Funktio, joka ei ole käytössä palauttaa yhden sensoriarvon perustuen käyttäjän antamaan funktioparametriin.

Edellä esitetyn ”kriittisen” funktion, joka toimii käsittelijän Arduino-käsittelijän pääfunktiona: tehtävänä on muodostaa sarjaliikenneyhteys, tulkita toisaalta järjestelmässä toimitettu parametri ja parametrin ollessa haluttu, välittää se Arduinolle. Edelleen funktion tehtävä on ottaa vastaan Arduinon vastasanoma. Vastasanoma voi olla kolmea eri tyyppiä: Analoginen jännitelukema välillä 0,00–5,00 Volttia, I/O -tieto muodossa 1,00 tai 0,00.

Parametrin perusteella tiedetään, minkä muotoista viestiä Arduinolta tullaan vastaanottamaan. Jos vastaanotettava viesti liittyy paineeseen, se käsitellään funktiossa ”calcPressure()” kaavan (1) mukaisesti. Jos vastaanotettava viesti on tyyppia I/O, se käsitellään totuusmuuttujana eli arvona ”True” tai ”False”. Totuusmuuttujan arvo ilmaisee suoraan, onko sensorin kohdalla vettä vai ei. Lisäksi mahdollinen sanoma voi sisältää virheviestin.

Vastaanotettu painesensorin jännitelukema tarkoittaa sitä, että siitä täytyy laskea painelukema ja työssä käytetty yksikkö on baari (bar). Painesensorin lähdeviitteessä on annettu kaava, jonka perusteella paineen viitearvon saa laskettua. Lopuksi käsitelty tieto välitetään takaisin, josta ajurin funktiota on kutsuttu. Tiedon palautuessa se tulkitaan myöhemmin esiteltävin menetelmin. (Freescale Semiconductor Inc 2009, 4.)

Käsittelijässä käytettävä laskentakaava muodostuu seuraavasti:

$$(1) \qquad p = \left(\frac{\frac{V_{out}}{V_s} + 0.04}{\frac{0.004}{100}} \right) \left(\frac{1013\ hPa}{978\ hPa} \right)$$

Kaavassa p on mittaushetken painearvo, V_{out} on mittaushetkellä saatu jännitelukema, V_s on valmistajan määrittämä jännitelukema 5.1 V, luvut 0.04 ja 0.004 ovat valmistajan määrittämät korjausluvut. Luku 100 jakajana, kun halutaan yksiköksi baari. Viimeiseksi korjauskertoimeksi on saatu jakamalla lähdeviitteestä virallinen ilmanpaineen mittaustulos suhteessa sensoreista saatuun mittaustulokseen käyttämällä tätä kaavaa (1). Jännitteen yksikkönä on voltti (V) ja baari (bar) paineen yksikkönä. (Foreca; Freescale Semiconductor Inc 2009, 4.)

11.2 Libmbus python-käsittelijä

M-Bus-tiedon yksilölliseen hakemiseen ja edelleen sen toisenlaiseen esittämisen tueksi on tehty tiedonkäsittelijä, joka etsii Libmbusilta saatavasta sanomasta halutut tiedot. Fyysisesti M-Bus-kortti on Sharky:ssa sijoitettu portti 2: een. Tällä kortin sijoituksella mahdollistetaan meno- ja paluupuolen lämpötilatieto M-bus -viestissä. Liitteessä 5 (LIITE 5) on esitetty myös käsittelijän muuttujatietoja. Liite koskee esimerkiksi aliaksen kutsua, haettavien arvojen nimikkeitä sekä palautustietoja. Käsittelijän pääfunktiota kutsutaan toisaalla järjestelmässä. Pääfunktio palauttaa tiedon siitä, mistä arvo on peräisin, ja itse arvon.

11.3 Käsittelijä lämpötilatiedon esittämiseen komentokehotteessa

Tehdyn käsittelijän toiminta perustuu kolmen tiedoston varaan. Tässä työssä käytetty ohjelma sekä tiedostot komentokehotepuolella on esitetty liitteessä (LIITE 6). Kun ohjelman ”getValues()” -pääfunktiota kutsutaan toisaalla järjestelmässä, lähtee lukuprosessi käyntiin. Järjestelmän käynnistyksen yhteydessä kaikkia sensoreita ei yleensä löydy, mikä luo tarpeen Owfs -kirjaston käynnistyskäskylle. Käynnistyskäsky on sijoitettu ohjelmassa heti pääfunktion alkuun, että myös virheenkäsittelyn yhteyteen. (Babcock; Owfs Development site 2018.)

Käynnistyskäskyn jälkeen kutsutaan funktioita, jotka palauttavat erillisiin tekstitiedostoihin sijoitetut sensorien yksilölliset osoitteet sekä niiden fyysiset sijoitukset järjestelmässä. Kun nämä tiedot on haettu muuttujiin, alkaa toistorakenne lukemaan yksilöllisen tunnisteen perusteella lämpötilatietoja Owfs-tiedostojärjestelmästä. Jos virheitä ei ajon aikana ilmene, palautetaan luetut tiedot yksilöllisillä tunnisteilla sekä paikkailmaisulla kysyjälle. Jos virheitä ilmenee, palautetaan kysyjälle virheviesti ja suoritetaan Owfs-kirjaston käynnistyskäsky uudelleen. (Babcock; Owfs Development site 2018.)

11.4 Käsittelijä releen kärkitiedon lukemiseen

Aliaksista poiketen käsittelijässä (LIITE 7) muodostetaan funktion sisällä sarjayhteys releeseen ja lähetetään tilakyselysanoma. Lähetyssanoman kohta ”xA1” kohdistuu fyysisesti relekorttiin, joka on dip-kytkimellä asetettu arvoon yksi. Relekorttikohdistusta seuraa ”x00”, joka edelleen tarkoittaa tilakyselyä.

Rele vastaa muodossa "0xFF/0xA1/0x00/0x00/0x00 ...", jossa ensimmäiset nollatilat tarkoittavat edellisen kappaleen perusteella kärjen numero yksi tilaa relekortissa numero yksi. Tila ”00” luonnollisesti viittaa kärjen lepotilaan ja ”01” ohjattuun tilaan. Tiloja esitetään niin monelle kärjelle kuin kortissa niitä fyysisesti sijaitsee, tässä työssä käytetyssä kortissa on kahdeksan kärkeä.

Käsittelijässä lopuksi invertoidaan suolaveden poistosäiliön ylempi moottoriventtiili komponenttilistaan, joka palautetaan listamuotoisena. Funktio palauttaa myös toisen listatiedon, joka perustuu yksinomaan releen kärkitietoon.

11.5 Järjestelmän komponenttiohjaukset komentokehotteessa

Erityisesti ohjattavien komponenttien aliaksia kutsutaan myös graafisen ohjelman kautta. Python-ohjelmissa on tiettyjä heikkouksia aliaksien suorituksissa, jotka ovat kuitenkin komentokehotteessa ajettavissa normaaliin tapaan. Aliakset esitettynä taulukossa 1 (TAULUKKO 1).

TAULUKKO 1. Ohjauskomennot

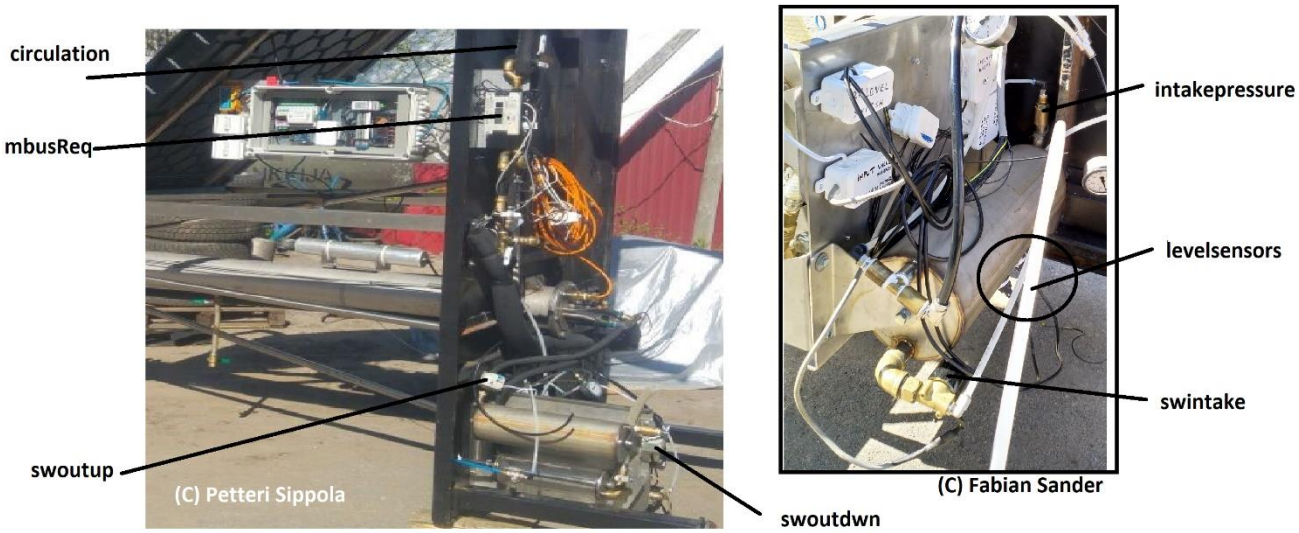
Komento	Ohjaus
circulation1	Kiertovesipumppu päälle
circulation0	Kiertovesipumppu pois
vacuum1	Tyhjiöpumppu päälle
vacuum0	Tyhjiöpumppu pois
intakepressure1	Suolaveden syötön magneettiventtiili auki
intakepressure0	Suolaveden syötön magneettiventtiili kiinni
swintake1	Suolaveden syötön moottoriventtiili auki
swintake0	Suolaveden syötön moottoriventtiili kiinni

(jatkuu)

TAULUKKO 2. (jatkuu)

swoutdwn0	Alempi suolaveden poiston moottoriventtiili kiinni
swoutdwn1	Alempi suolaveden poiston moottoriventtiili auki
swoutup0	Ylempi suolaveden poiston moottoriventtiili auki
swoutup1	Ylempi suolaveden poiston moottoriventtiili kiinni
mbusReq	Tulostaa M-Bus-datan Sharky -energiamittarilta
readsensors	Tulostaa kaikki sensori- ja energiamittariarvot

Kuvassa 6 (KUVA 6) ilmenee taulukon 1 (TAULUKKO 1) komponenttien sijainnit sekä eri säiliöiden paikat. Tyhjiöpumppu, sekä kaksi painesensoria on sijoitettu aurinkopaneelin kotelon sisään, mikä ilmenee kuvasta 1 (KUVA 1).



KUVA 6. Komennot ja komponentit (mukaillen Sander 2018b; Sippola 2018)

Kuvassa (KUVA 6) oikealla on näkyvissä suolaveden syöttösäiliö, jossa myös huonosti erottuvat tasoanturit. Vasemmalla kuvasta katsottuna ylempi säiliö on suolaveden poistosäiliö, sekä sen alapuolella kirkas puhtaan veden säiliö. Vasemmalla näkyvä oranssi kaapeli on edellä esitetty Dimense Oy:n valmistama lämpöanturikaapeli (Dimense Oy 2018).

11.6 Kaikkien sensoritietojen luku komentokehotteessa

Alias ”readsensors” (TAULUKKO 1) viittaa lukuohjelman (LIITE 8) suoritukseen. Lukuohjelman ohjelmatiedosto suoritetaan sarjamuotoisesti ylhäältä alas, eikä se sisällä paikallisia funktioita. Ohjelman

alussa määritetään muuttujiin edellä esitetyt ajurit liittyen Arduinoon eli paine- ja tasosensoritietoon, lämpötilatietoon, sekä M-Bus-tietoon.

Ohjelma kutsuu käsittelijän sisäistä funktiota, joka kommunikoi alempien ohjelmatasojen kanssa. Koska käsittelijät on sijoitettu Raspbianin tiedostojärjestelmässä eri paikkoihin, niiden paikallistamistekniikassa käytetään Pythonin ”sys”-kirjaston ”path” -funktiota. Sys-kirjasto siirtää ohjelman sisäisen osoittimen tiedostopolkuun, johon sillä viitataan. Kun osoitin on oikeassa paikassa, voidaan polussa sijaitsevan tiedosto tuoda lukuohjelman käyttöön import-komennolla. Heti import-komennon jälkeen ”as”-komennolla voidaan tiedoston nimeä muuttaa aliaksen tyyppisesti paikallisesti. Pääohjelman tuottama tieto esitetään kuvassa 7 (KUVA 7). (Python Software Foundation a.)

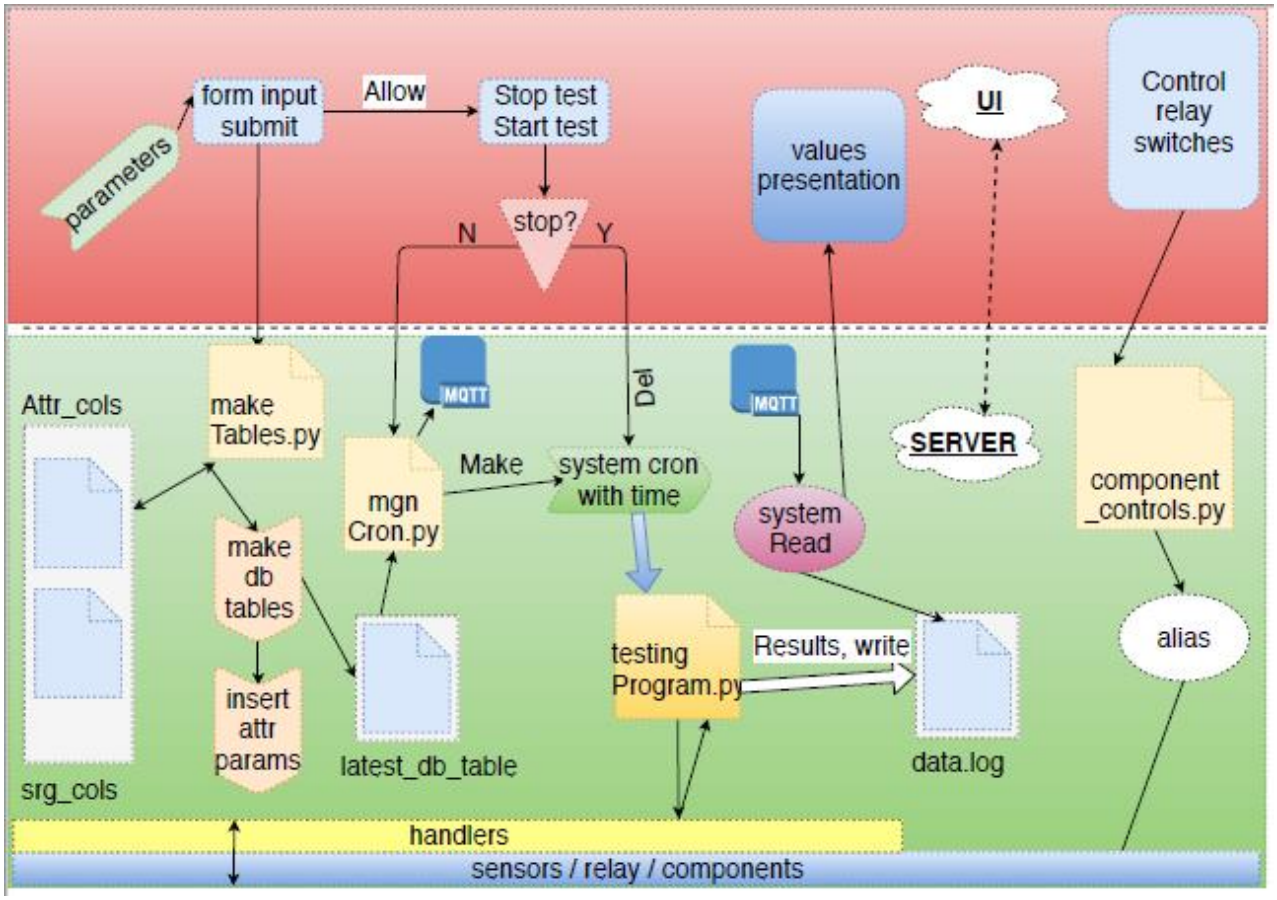
```
pressures and levels ...(1/4)
temperatures ...(2/4)
mbus ...(3/4)
relay ...(4/4)
pressure and levels
inner top pressure:  0.78
outer top pressure:  0.89
bottom pressure:     1.24
top level sensor:    True
bottom level sensor: True

temperatures
1. (dimense_27_bottom                ) : 28.76AE33090000EB: 31.75 'C
2. (dimense_77                       ) : 28.5575200900009A: 36.9375 'C
3. (dimense_127                      ) : 28.226ABD0700001C: 40.375 'C
4. (dimense_177                      ) : 28.0F9C2009000052: 43.75 'C
5. (dimense_227                      ) : 28.7A4DBD0700003A: 47.5 'C
6. (dimense_ 277                     ) : 28.A18D330900001C: 53 'C
7. (dimense_327                      ) : 28.9AAE33090000BF: 57.6875 'C
8. (dimense_377                      ) : 28.443034090000FB: 73.875 'C
9. (dimense_427_top                  ) : 28.039C200900002F: 77.4375 'C
10. (individual sensor, salt drain   ) : 28.FF488CA11605: 17.3125 'C
11. (individual sensor, drinking water) : 28.FFDAA5A11605: 18.0625 'C
12.(individual sensor, sw-input      ) : 28.FF2E15B01605: 17.75 'C

mbus data
Total Energy (kWh): 1904.0
Total Volume (m^3): 1218.439
VolumeFlow (liters): 17.0
Power (kW): 0.836
Flow temperature (Celcius): 86.1
Return temperature (Celcius): 43.4
Current Energy (kWh): 0.0

 relay statuses, False = normal stage
False
False
False
False
False
False
False
False
done
```

KUVA 7. Kaikkien tietojen esittäminen komentokehotteessa

Sensoritiedot tulostetaan, kun ohjelmasuoritus on valmis. Toisin sanoen ne tulostetaan, kun jokainen käsittelijä on palauttanut omat arvonsa ohjelman käyttöön. Kyselyjärjestys on: paine-, taso, lämpötilasensorit, M-Bus sekä releen kärkitiedot. Tiedot palautuvat Python lista -muodossa, jonka vuoksi lukuohjelmassa on erilliset tulostussilmukat jokaisen listan riveille. Tiedot tulostuvat komentokehotteessa kukin omille riveilleen kuvan 7 (KUVA 7) mukaisesti (Python Software Foundation b).

12 JÄRJESTELMÄN GRAAFINEN KÄYTTÖLIITTYMÄ

Graafinen käyttöliittymä toteutettiin käyttäen Node-RED-ohjelmointiympäristöä. Opinnäytteen tekijän osaaminen rajoittuu asiakasrajapinnan toiselle puolelle, eli graafista ohjelmointitaitoa ei juurikaan ole. Node-RED oli siis oiva ratkaisu tähän projektiin, sillä myöhemmin esiteltävä graafinen käyttöliittymä on toteutettu vain raahaamalla objekteja työpöydälle ja edelleen muokattu niiden ilmentyminen välilehdillä. Käyttöliittymän käsittelyfunktiot on kirjoitettu JavaScriptillä. Tässä luvussa tutustutaan pintapuolisesti graafisen osuuden toteutukseen. Liitteestä 9 (LIITE 9) löytyy graafisen ohjausympäristön yleisriippuvuudet tuotettuihin ohjelmatiedostoihin. (JS Foundation.)



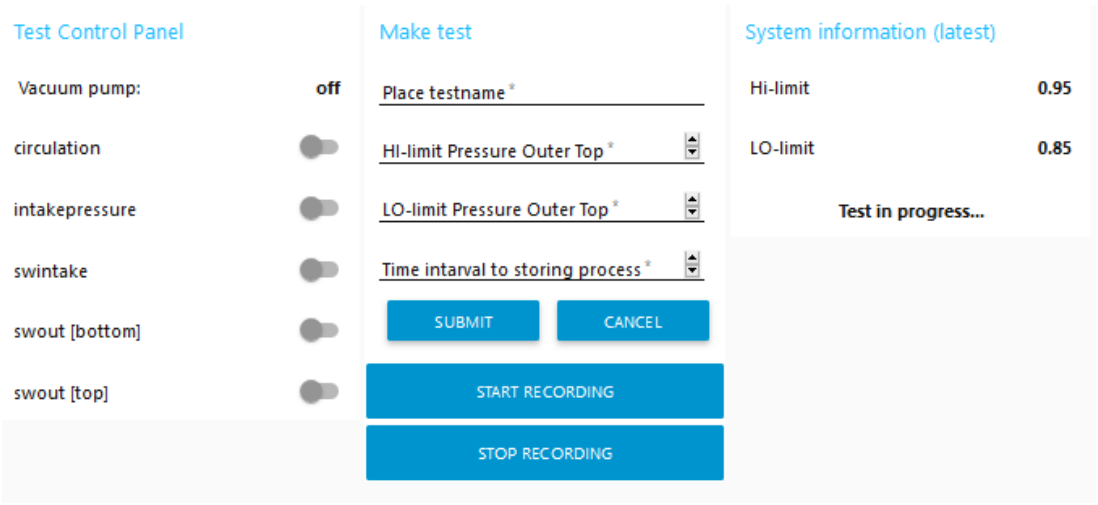
KUVIO 3. Liitännäisriippuvuudet testialustan osalta

Kuviossa 3 (KUVIO 3) on esitetty yleisperiaatteelliset riippuvuudet testialustan osalta. Liitännäisiä, funktioita sekä prosessia on vaikea asettaa yhteen piirrokseen. Parhaiten graafisen ohjelman luonne ja riippuvuudet kuitenkin aukeavat lukijalle tarkastelemalla tätä lukua ja erityisesti liitettä 10 (LIITE 10).

12.1 Frontend-näkymä

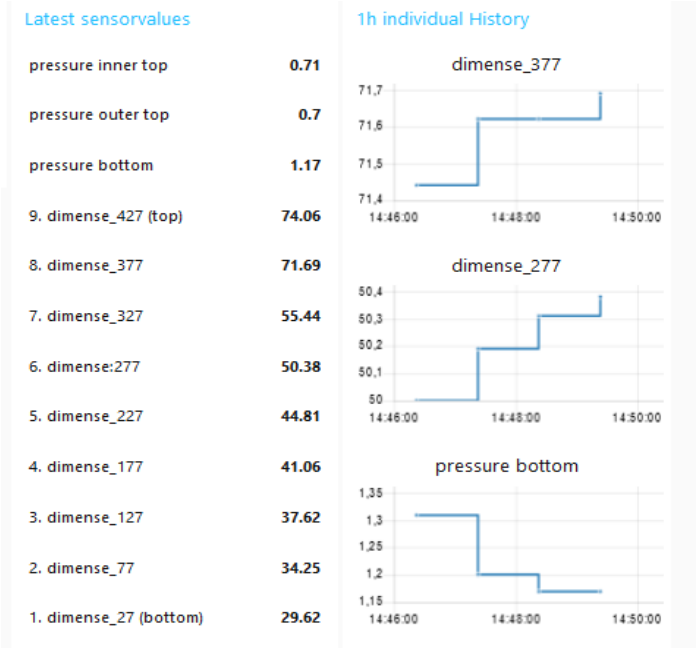
Asiakasrajapinta eli graafisen käyttöliittymän puoli, joka on tehty siten, että se olisi käytettävissä tietokoneen perustaidoilla. Tässä osiossa esitellään käyttöliittymän merkittävin osuus eli sivuston välilehti, jolla luodaan suolanpoistolaitteiston testausparametrit, käynnistetään testi, pysäytetään testi, sekä seurataan testin tuottamia arvoja reaaliajassa. Sivustolla on tämän ”TestBench” -välilehden lisäksi kolme muuta välilehteä, jotka kukin rajoittuvat seuraaviin osa-alueisiin: komponenttien ohjauksiin, tietyn hetken arvojen kysymiseen ja esittämiseen sekä keskusyksikön sammutus- ja uudelleenkäynnistysoperaatioihin. Prosessit, jotka toimivat käyttöliittymän taustalla, esitellään hieman myöhemmin tässä työssä.

Tässä esiteltävä välilehti (TestBench) sisältää kaiken muun lukuun ottamatta järjestelmän käynnistysoperaatiopainikkeita. Muista välilehdistä poiketen testauksen aikana tuotetut arvot tulevat käyttäjän näkyville automaattisesti. Muiden välilehtien käyttö testiajon aikana voi johtaa ohjelmallisiin konflikteihin, eikä haluttuja arvoja välttämättä saada tallennettua. Käyttäjää muistutetaan ponnahdusikkunalla testiajon alkamisajankohtana, että testiajon aikana ei voida käyttää muita välilehtiä. Käyttäjänäkymä esitetään kuviona (KUVIO 4), josta ilmenee testiajon aikainen komponenttien hallinta, parametrien syöttö sekä viimeisin järjestelmän palauttama tieto. Todellisuudessa välilehti sisältää myös kuvion 5 (KUVIO 5) mukaiset tiedon seurantaan liittyvät objektit.



KUVIO 4. Graafisen käyttöliittymän käyttäjänäkymä osa 1

Testipenkin näkymässä (KUVIO 4) pystytään ohjaamaan jokaista järjestelmän komponenttia, pois lukien tyhjiöpumppua. Tyhjiöpumpun tila ohjautuu automaattisesti mitattavien arvojen ja annettujen parametrien suhteen, automaattiohjaukseen palataan myöhemmin. Käyttäjän tulee täyttää testilomake, jonka tulee sisältää yksilöllisen kutsunimen testiajolle, mitattavien tyhjiöarvojen ylä- sekä alarajat. Lisäksi käyttäjän tulee määrittää aika-arvon minuutteina, jonka funktiona on ohjelmasuoritusintervalli. Testilomake on yhteydessä tietokantaan sekä itse ohjelma-ajoon että ohjelma-ajon taustarutiineihin. Järjestelmä informaatio-osio palauttaa sanallista palautetta taustaprosessien onnistumisesta tai vastaavasti epäonnistumisesta.



KUVIO 5. Graafisen käyttöliittymän käyttäjänäkymä osa 2

Testipenkin arvojen esitysnäkymässä (KUVIO 5) on valittu viimeisimmän mittausajankohdan esitysmuodoksi arvopohjainen tulkintamalli. Node-RED-diagrammi-objektilla tuotetaan myös graafista dataa, joka käsittää yhden tunnin historian itsessään. Graafinen osuus ja muuttuvien diagrammielementtien määrä hipoo Raspberryn tehokasta käyttörajaa. Työssä käytetyn Raspberry -mallin muisti on jaettu grafiikkaohjaimen kanssa, ja sen yhteismäärä on yksi gigatavu. Sivun lataaminen testiajon ollessa suorituksessa ja samanaikainen arvojen tulostus käyttäjän näkyville ei toimi yhtä suorituskykyisesti kuin muissa edellä mainituissa välilehdissä. Tapahtumat testipenkissä kuitenkin johtavat onnistumiseen, eikä konflikteja synny. Testausta suorittavan tahon on joissain tapauksissa myös hyvä nähdä järjestelmän reagointeja tehtyihin toimenpiteisiin. (JS Foundation; Raspberry Pi Foundation.)

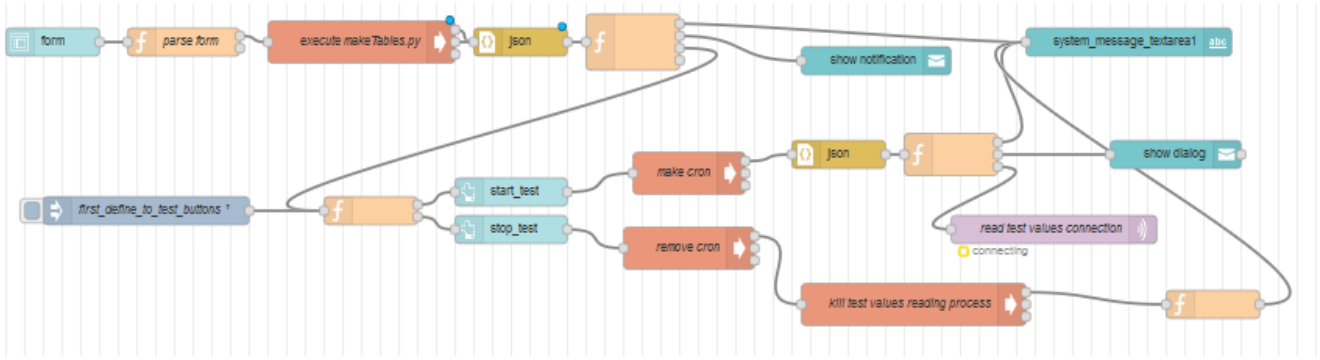
12.2 Käyttäjänäkymän hallinta ja taustaprosessit

Tässä alaluvussa perehdytään hieman tarkemmin graafisen käyttöliittymän taustaprosesseihin sekä niiden suoritustapoihin. Edellisen tapaan jatkan edelleen käsittelyä vain testipenkin osalta. Taustaprosessien osuus jo testipenkin osalta on erittäin laaja ja monisäikeinen. Tässä keskitytään tärkeimpiin ratkaisuihin sekä taustatoimintojen pääpiirteisiin. Myöhemmin tässä viitattavat Python -kieliset ohjelmat tai -käsittelijät löytyvät liitteestä (LIITE 10) kokonaisuudessaan. Liitteen sisällöstä on peitetty tietokantaan liittyvät tiedot, sillä esimerkiksi salasanat ja tietokantakäyttäjät eivät ole julkista tietoa.

12.2.1 Make test -osio

Kuten aiemmin aihetta sivuttu, lomake on yhteydessä muun muassa tietokantaan ja sen tietoja käytetään testiohjelma ajossa. Lomakkeen ryhmään on liitetty myös testin aloitus- ja lopetuspainikkeet. Lomakkeen lähettäminen johtaa tietokannan taulujen luontiin sekä parametrien asettamiseen sille tarkoitettuun tauluun. Kahteen eri tietokantaan luodaan siis samanniminen taulu, joka viittaa luodun testin nimeen. Ensimmäinen tietokanta käsittää parametrit, joita käytetään ohjelma-ajossa. Toinen tietokanta on tarkoitettu eri rekisteröitävien testiarvojen tallentamiseen. Testiarvoja voidaan täten tulkita käyttöliittymässä tai viedä taulun tietoja suoraan tietokannasta, esimerkiksi Excel-taulukkoon jatkoanalysointia varten. Hallintaosuuden totetutus esitetään kuvioissa 6 ja 7 (KUVIO 6, KUVIO 7). (Smith 2018; The phpMyAdmin devel team 2018.)

Testin aloitus, eli ”Start recording” luo Cron rutiinin järjestelmään, jossa ajetaan edellä esitetyn aikaintervallin puitteissa testiohjelma. Testiohjelmassa käytetään edellä esitettyjen ylä- ja alaraja-arvojen mukaisia parametrejä säätelemässä tyhjiöpumpun pois- ja vastaavasti päälläoloaikaa. Testin aloitus johtaa myös Mqtt-tiedonsiirtotoimenpiteeseen, jonka päämäärä esitetään seuraavassa alaluvussa (Both 2017).

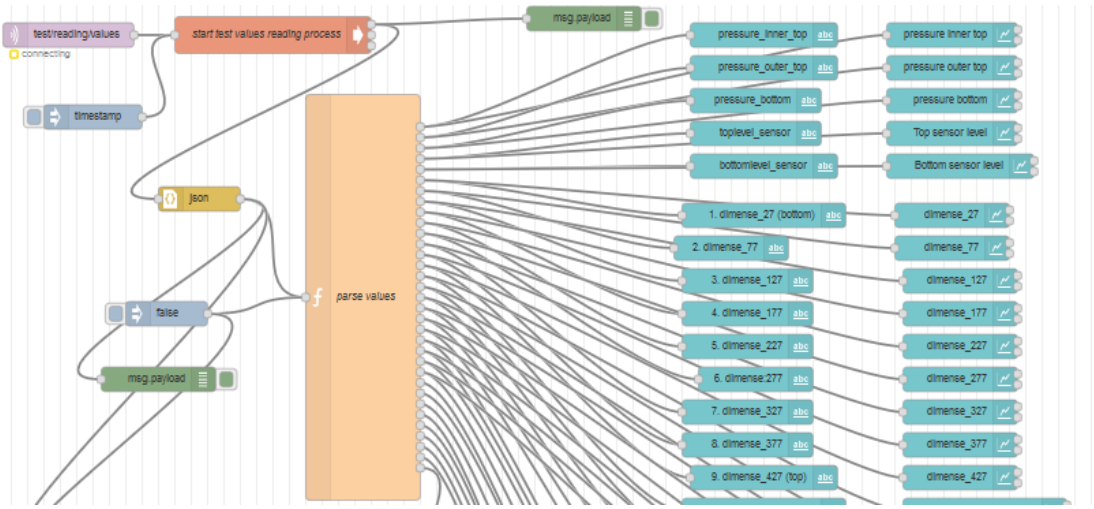


KUVIO 6. Käyttöliittymän make test -osion toteutus

Kuviossa 6 (KUVIO 6) ilmenee Node-RED asiakaspuolen hallintatapa sekä testilomakkeen ja testipainikkeiden näennäinen sijoittelu. Funktiot ovat sijoitettuna elementtien väliin analysoimaan järjestelmälle toimitettavia argumentteja. Jos funktiot on sijoitettu punaisten järjestelmäsuoritusten jälkeen, on silloin niiden tehtävä tulkita suorituksesta palautuvaa dataa. (JS Foundation.)

12.2.2 Tiedon esittäminen

Testiohjelma tuottaa käyttäjän määrittämän aikavälin puitteissa väliaikaista tietoa ”data.log” -tiedostoon kun testiajoon liittyvät edellä esitetyt toimet on onnistuneesti suoritettu. Lokitiedostoa luetaan järjestelmätasolla reaaliaikaisesti, ja siihen kirjoitettava tieto välittyy näin asiakasrajapintaan käsittelyn jälkeen.



KUVIO 7. Tiedon esityksen mahdollistaminen

Kuten kuviosta 6 (KUVIO 6) vodaan todeta, että kun testi aloitetaan painikkeella, sen jälkeisillä toimilla mahdollistetaan tiedon lähetys Mqtt-elementillä. Edelleen tiedon esittämisen perustana toimii kuviossa 7 (KUVIO 7) järjestelmäkomennon suoritus (tail -f), joka mahdollistetaan Mqtt -tilauksella. Kyseinen järjestelmäsuoritus on voimassa niin kauan, kunnes se keskeytetään. Keskeyttäminen tapahtuu edellä mainituin käyttäjän toimin keskeyttämällä testi. Testin keskeyttäminen johtaa edelleen järjestelmätason prosessin tunnistus- ja keskeytystoimenpidekomentoon. Fyysisesti nähtävä (KUVIO 7) suurin funktio käsittää suorituksesta palautuvan tiedon analysoinnin, sekä ohjauksen oikeille esityspaikoilleen. (Both 2017.)

13 JÄRJESTELMÄN VIRRANKULUTUS

Suolanpoistolaitteiston sähköiset mittaukset päätettiin sijoittaa aikaan, jolloin se on kokonaisuutena katsottuna käyttövalmis tai valmis jatkokehityksellisiin toimenpiteisiin. Tässä luvussa esiteltävät mittaustuloksien on tarkoitus toimia jatkokehityksen aikaisena tuloslistana sähkön varastoinnin sekä aurinkopaneelia koskevissa kysymyksissä.

13.1 Mittaustulokset

Mittaustulokset ovat esitettynä taulukossa (TAULUKKO 2), järjestelmän pääsulakkeen ollessa 40 ampeeria. Lataussäätimen ja järjestelmäsyötön välisen sulakkeen ollessa 15 ampeeria (LIITE 1). Mittaukset on suoritettu akun ja pääsulakkeen välistä siten, että lataava teho on kytkettynä pois järjestelmästä

TAULUKKO 2. Virtamittaustulokset

Mittalaite: Onniline 920 yleismittari, tarkkuus DC/10A alueella = 1.8% ± 3 (Onnline 2009, 14).			
Kuormitustekijä **Mittaustila**	**Sulakeryhmä n:o** **Maksimikuorma I [A]**	**Kokonaisvirran muutos ΔI [A]**	**Näennäinen Kokonaisvirta I [A]**
Järjestelmä Yleisvalmius	1.-5. 3, 3, 3, 5, 5	**0.33**	**0.33**
Vacuum Päällä	5. 5	**0.75**	**1.08**
Circulation Päällä	5. 5	**1.36**	**2.44**
Feed valve Ohjaus	4. 5	**1.37**	**3.81**
Lwr dw-out Ohjaus	4. 5	**0.11**	**3.92**
Upr dw-out Ohjaus	4. 5	**0.06**	**3.98**
Sw feed pressure	4.	**0.35**	**4.33**

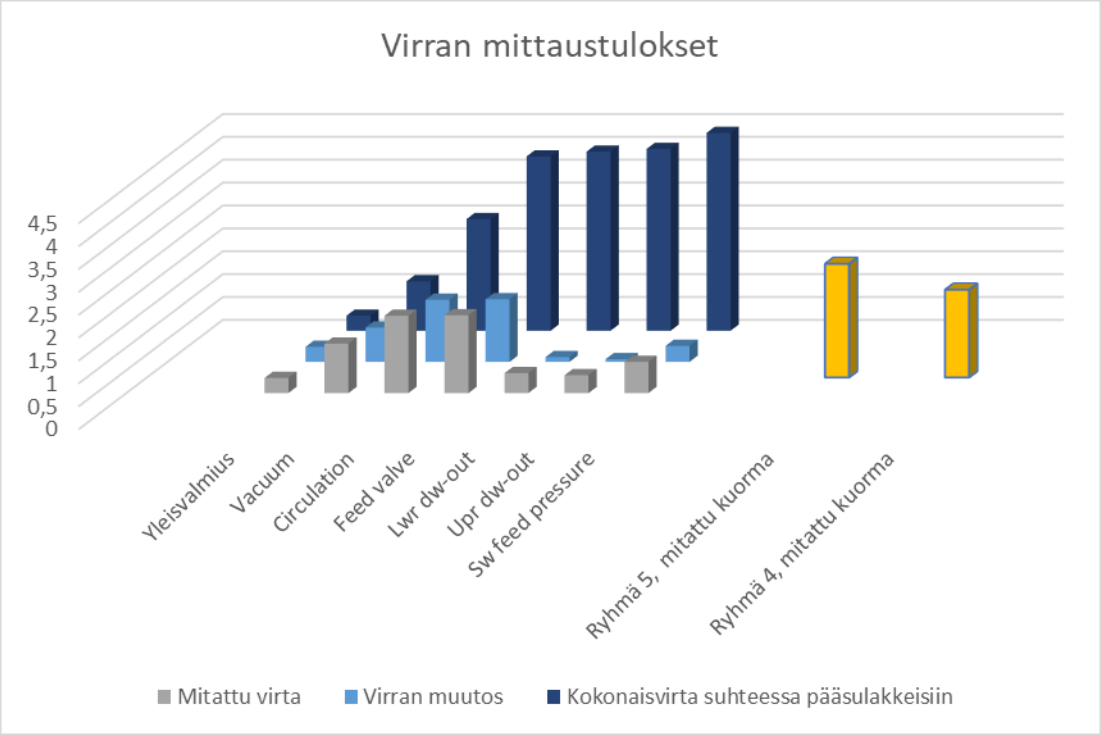
(jatkuu)

TAULUKKO 2. (jatkuu)

Ohjaus	5		

13.2 Tuloksien havainnollistaminen

Mittaustulokset esitellään graafisessa muodossa (KUVIO 8), jossa on pyritty kohdentamaan eri komponenttien virrankulutusta ryhmäkohtaisesti. Järjestelmän virrankulutus on pieni suhteessa esimerkiksi omakotitalon sähköistysjärjestelmään. Virrankulutuksen suhteellisen vähäisyyden vuoksi, tässä esitettävät kokonaisvirrat ovat laskettu epätarkimman korjauskertoimen kahden desimaalin mukaan (Onnline 2009, 14).



KUVIO 4. Graafinen havainnollistaminen virtamittauksista

14 POHDINTA JA YHTEENVETO

Työn tavoite oli toteuttaa laitteen sähköistäminen ja ohjelmallinen ohjausympäristöratkaisu siten, että se toimii ja on avoin tarvittaville tulevaisuuden jatkokehitystoimenpiteille. Opinnäytetyön toiminnallinen osuus on toteutettu noin 1,5 kuukauden työaikaa vastaavalla panoksella. Toiminnallinen osuus sisältää seuraavat osa-alueet: Sähköpiirisuunnittelu, sähköisten komponenttien kytkennät ja kiinnitykset, kokonaisvaltainen ohjelmistokehitys.

Mekaanisen työn, eli käytännössä kaapelireittien, komponenttien ja keskuksen asennuksen alku oli hieman kangertelevaa. Hidastaviksi tekijöiksi muodostui keskuksen komponentti- ja laitehankinnat, sillä suurin osa niistä on hankittu ulkomailta ja näin niiden toimitusaika on viikkoja. Työn asennuksia odotutti myös erilaiset muut mekaaniset kehitystarpeet, jotta esimerkiksi sähköiset komponentit saatiin runkoon kiinnitettyä sopivalla tavalla. Muut mekaaniset ratkaisut toteutettiin työryhmässä dynaamisesti laitteiston tarpeita mukaillen, nämä muut ratkaisut eivät kuitenkaan sisälly opinnäytetyön aihepiiriin.

Opinnäytetyösuunnitelmaan sisältyi anturiratkaisu prosessituotteen suolapitoisuuden mittaukseen. Anturia ei kuitenkaan ollut mekaanisesti mahdollista liittää suoraan järjestelmään, anturin mekaanisten vaatimusten vuoksi. Anturi on käsikäyttöinen, mutta siitä pystyy myös tulkitsemaan muun muassa mikrokontrolleriteknologialla. Anturin liittämisestä suoraan järjestelmään luovuttiin, sillä sen käyttöön liittyvä toiminta ei näiden mekaanisten rajoitteiden vuoksi ollut mahdollista ilman käyttäjän fyysistä läsnäoloa. Anturi on sähköisten ominaisuuksien puolesta liitettävissä suoraan tulkittavaksi järjestelmään heti, kun mekaaninen analysointiympäristö on anturilaitteen vaatimusten mukainen.

Suolanpoistolaitteiston sähköinen ratkaisu pohjautuu käytettävyyteen, vikatilanteiden ennakointiin ja turvallisuuteen. Sähköistäminen on toteutettu syöttäviltä ja kuormittavilta osin mukaillen pienjännitestandardia. Työ käsittää kuitenkin pienoisjännitekäytön, joten toteutettu ratkaisu on riittävä. Ukkossuojauksen osalta sen riittävyys on varmistettu erään kansallisen viraston virkamieheltä, mutta allekirjoittanut unohti kysyä lupaa tiedonannon merkitsemisestä lähteisiin, joten siinä on viitattu standardiin.

Opinnäytetyöprojektin ohjelmallinen osuus oli tarkoitus ensimmäisen suunnitelman mukaan sopeuttaa puhtaasti komentokehotekäyttöön. Työryhmän suosituksesta ohjelmallinen toteutus kohdennettiin pohjautuen graafiseen käyttöliittymään. Graafinen ohjelmointi ei ole allekirjoittaneen parhaimpia puolia,

joten tämän vuoksi toteutusalustaksi valittiin työssä esitelty kehitysympäristö. Ohjelmistoratkaisussa on menty tietyin osin allekirjoittaneen epämukavuusalueelle, mutta kaikesta huolimatta kokonaisuus, jonka kehtaa ja uskaltaa esittää on saavutettu. Graafinen kehitysympäristö tuo osaltaan mukanaan myös lisää vaikeusastetta jatkokehityksellisiin tarpeisiin.

Jatkokehittäjän ajattelutapa olisi hyvä olla lohkomainen. Graafista toteutusta ja sen toiminnallisuutta kannattaa lähteä tutkimaan suoraan asiakasrajapinnan puolelta. Kun asiakasrajapinnan tutkittava elementti on valittu, kannattaa seurata elementtiä asiakasrajapinnan ylläpito-osuuteen ja edelleen tutkia järjestelmätason komentoja mahdollistavia elementtejä. Asiakasrajapinnassa alustettavat komennot yleensä viittaavat laitteistoa varten kehitettyihin Python kielisiin ohjelmatiedostoihin jotka edelleen vaikuttavat alemman tason ohjelmarajapintoihin, eli kommunikointeihin keskusyksikön ulkopuolelle.

Python ohjelmarakenne on pyritty toteuttamaan työssä suoraviivaisesti, jotta jokaisella Python ohjelmalla olisi helposti hahmotettava tehtävä ja näin ollen toivottavasti auttaa tulevaisuuden kehittäjää. Graafisen osuuden (ui) Python käsittelijät mukailevat suurimmaksi osaksi komentokehotepuolen (cli) vastaavia käsittelijöitä. Erona komentokehote- sekä graafisen osuuden käsittelijöillä on tiedon palautustyyppi. Palautustyyppinä toimii graafisissa käsittelijöissä dictionary, kun taas se on komentokehotekäsittelijässä lista. Tiedon esitystapana graafisisen puolen Python ohjelmissa on json, jotta Node-RED funktiot saavat arvot kohdennettua yksilöllisesti oikeille paikoilleen.

Sähköisen osuuden tavoitteet saavutettiin täysin suhteessa kehitysvaiheeseen. Suolaveden suolaisuuden mittauksen anturiratkaisu puuttuu järjestelmään liittämisestä. Anturiratkaisun puuttuminen johtuu mekaanisista syistä jotka eivät kuulu tämän työn aihepiiriin.

Tehtyjen virtamittausten pohjalta esitän, että kehityksen aikainen 7 ampeeritunnin akku korvataan tarvittaessa tulevaisuuden käyttöpaikan mukaan. Tulevaisuuden käyttöpaikan olosuhteiden ja laitteiston käyttöajan vaikutus tulee ottaa huomioon akkuvalintaa tehdessä. Myös aurinkopaneeli kannattaa mitoittaa käyttöpaikan olosuhteiden mukaan. Virtamittausten myötä voidaan todeta, että laitteiston maksimi hetkellinen ottoteho voi olla noin 72 vattia. Laskettu tehoarvo sisältää 20 % heilahtelukertoimen. Aurinkoenergian saatavuuden mukaan käyttöpaikalla, tulisi paneelin kyetä lataamaan aurinkoisena aikana akku täyteen. Lataussäädin jakaa tarvittaessa osan latausenergiasta syötölle. Lataussäätimestä ei löydy dokumenttia, joka ilmaisisi hyötysuhteen latauksen suhteen. Voimme kuitenkin päätellä, että hyötysuhde ei ole merkittävä.

Jos laitteiston käyttöaste on 100 % aurinkoenergian saatavuusalueella: Voimme edellä todetun puitteissa todeta 120W/12VDC aurinkopaneelin olevan käyttötarpeeseen sopiva, eikä akustoa tässä tapauksessa tarvitse päivittää. Jos käyttöaste ylittää aurinkoenergian käytettävyyden ajallisesti, ylittävä aika tulee suhteuttaa laitteiston kulutuksesta akun mitoitukseen. Ylittävä aika johtaa myös positiiviseen suuntaan aurinkopaneelin antotehon suhteen. Ylärajana paneelin antoteholle toimii lataussäätimen dokumentoinnista ilmenevä 360W/12VDC paneeli. Tähän järjestelmään ei voida liittää 24 VDC akustoa tai aurinkopaneelia.

Ohjelmallisesti järjestelmä toimii yhdenmukaisesti, mutta graafisesta näkökulmasta katsottuna järjestelmässä on edelleen joitain puutteita, jotka koskevat myös jatkokehityksellisiä ominaisuuksia. Perusteet ohjelmallisen osuuden vakauttamiseen tämän työn puitteissa oli olemassa, mutta allekirjoittaneelle ei tarjoutunut mahdollisuutta suorittaa sitä. Ohjelmiston näkökulmasta suurimmaksi ongelmaksi muodostui muun muassa edellä mainittu monisäikeisyys. Monisäikeisyys johtaa juurensa graafisen käyttöliittymän kehitysympäristöstä ja sen avulla suoritettavista taustaprosessirakenteista kokonaisuutena.

Työtä tarkastellessa kokonaisvaltaisesti sähköinen osuus vastaa laitteiston vaatimuksia, ja se on suolaveden anturiratkaisua lukuuottamatta aikaisemmin sovitun mukainen. Ohjelmallinen osuus vastaa suurimmilta osin laitteiston vaatimuksia, ja se on pyritty toteuttamaan työryhmän toteutuksen aikaisten toiveiden mukaisesti. Ohjelmallinen osuus ei vastaa alkuperäistä suuunnitelmaa, laitteiston komentokehoteympäristöön prosessitietoihin perustuvan toisto-ohjelmiston puuttuessa täysin. Tuotettu graafinen käyttöliittymä sisältää rajapinnan automaattiseen ohjausohjelmistoon, mutta sen heikkoudet perustuvat seuraaviin osa-alueisiin: virheen paikantamiseen, virheen ilmaisukykyyn sekä jatkokehitysvaatimuksiin. Näiltä osin ohjelmallinen osuus ei siis ole tavoitteiden, laitteiston vaatimusten eikä suunnitelman mukainen.

Jälkikäteen ajateltuna tuotettu työ on melko laaja ja sitä varten on täytynyt tutkia laajasti osin toisistaan poikkeavaa tietoa. Kirjallisesta osuudesta oli pakko karsia joitain tietoja, mutta merkittävimmät tiedot merkittiin. Kirjallinen osuus käsittää tämän kirjallisen annin sekä tehdyt kuviot. Vaikka työ onnistui suurimmilta osin, henkilökohtaisesti jäi hieman vaivaamaan kehitykseen jäänyt ongelma. Tätä työtä koskevat ratkaisut, ohjelmiston riippuvuudet ja sen puutteet on yksilöllisesti esitetty jatkokehittäjälle. Järjestelmäkehitys jatkuu siis edelleen opinnäytetyön jälkeen.

LÄHTEET

Aermech. 2014. ECU (Engine Control Unit) Cars,ECM,Parts,Functioning. Saatavissa: http://aermech.com/ecu-engine-control-unit-carsecmpartsfunctioning/. Viitattu 29.5.2018.

Amazon.com, Inc. 2014. DC Converter,GEREE DC Buck Module 12V to 5V 3A USB Output Step Down Voltage Regulator Charge for Samsung Galaxy , iPad iPhone 4S 5 6/6 Plus etc. Saatavissa: https://www.amazon.com/Converter-GEREE-Voltage-Regulator-Samsung/dp/B00O6R9PCC. Viitattu 7.5.2018.

Arduino. Serial. Saatavissa: https://www.arduino.cc/reference/en/language/functions/communication/serial/. Viitattu 10.5.2018.

Arduino. What is Arduino? Saatavissa: https://www.arduino.cc/en/Guide/Introduction. Viitattu 7.5.2018.

ATEN International Co., Ltd. 2017. USB-to-Serial Adapter. User manual. Saatavissa: http://assets.aten.com/product/manual/uc232a-w_2017-05-04.pdf. Viitattu 8.5.2018.

Aukia, A. 2013. Supported devices. GitHub, Inc. Saatavissa: https://github.com/rscada/libmbus/wiki/Supported-devices. Viitattu 25.5.2018.

Babcock, A. Setup for OWFS 2.7p2 by David Babcock. Saatavissa: http://owfs.org/index.php?page=gentoo-scripts. Viitattu 26.5.2018.

Both, D. 2017. How to use cron in Linux. Saatavissa: https://opensource.com/article/17/11/how-use-cron-linux. Viitattu 9.6.2018.

DIEHL Metering. 2012. SHARKY 775. Installation and User Guide. Saatavissa: https://www.wms.nl/application/files/8115/0607/4640/Handleiding_Sharky775_v1.6.pdf. Viitattu 4.5.2018.

Dimense Oy. 2018. TERMO – Lämpötilamittauskaapeli. Saatavissa: https://dimense.fi/mittalaitteet-ja-anturit/termo/. Viitattu 4.5.2018.

Domotiga. M-bus. Saatavissa: https://www.domotiga.nl/projects/domotiga/wiki/M-Bus. Viitattu 25.5.2018.

Dongguan Sunworld Co., LTD. 2017. LCD Solar Charge controller. Details. Saatavissa: http://www.ecosolarpanel.com/content/?388.html. Viitattu 5.5.2018.

Drake, D. 2008. Writing udev rules. Saatavissa: http://www.reactivated.net/writing_udev_rules.html. Viitattu 15.5.2018.

Foreca. Ilmanpaine. Saatavissa: https://www.foreca.fi/Finland/Kokkola/Maalaaka. Viitattu 17.5.2018.

Frankel, R. 2015. How to Set Up Command Aliases in Linux/Ubuntu/Debian. Saatavissa: http://www.hostingadvice.com/how-to/set-command-aliases-linuxubuntudebian/. Viitattu 19.5.2018.

Freescale Semiconductor Inc. 2009. MPX4250A. Datalehti. Saatavissa: http://www.mouser.com/ds/2/302/MPX4250A-783416.pdf. Viitattu 4.5.2018.

Future Technology Devices International Limited (FTDI). 2016. TTL-232RG. Saatavissa: http://www.ftdichip.com/Support/Documents/DataSheets/Cables/DS_TTL-232RG_CABLES.pdf. Viitattu 8.5.2018.

Inkinen, P. & Tuohi, J. 2006. Momentti 1. 4.–9. painos. Helsinki: Kustannusosakeyhtiö Otava.

JS Foundation. Documentation. Saatavissa: https://nodered.org/docs/. Viitattu 7.6.2018.

JST Lawrence. 2016. Installing Dallas 1-Wire. Saatavissa: http://www.noveldevices.co.uk/rp-1-wire. Viitattu 25.5.2018.

Kaasalainen, J. 2007. Voimalaitoksen vedenkäsittelyn uudet menetelmät. Lappeenrannan teknillinen yliopisto, Energia- ja ympäristötekniikan osasto. En2010200. Kandidaatintyö. Saatavissa: https://www.doria.fi/bitstream/handle/10024/30957/TMP.objres.718.pdf?sequence=. Viitattu 28.4.2018.

KMtronic. a. RS485 Relay Controller - Eight Channel. Saatavissa: https://www.kmtronic.com/RS485%20Relay%20Controllers?product_id=73. Viitattu 7.5.2018.

KMtronic. b. USB RS485 8 Relay boards by Raspberry Pi. Saatavissa: https://info.kmtronic.com/usb-rs485-8-relay-boards-by-raspberry-pi.html. Viitattu 19.5.2018.

KMV-tuotteet Oy. Suolanpoisto. Saatavissa: http://www.kmv.fi/tuotteet/puhdas-vesi/suolan-poisto/. Viitattu 28.5.2018.

Krishna, H. 2004. Introduction to Desalination Technologies. Report 363. Austin: Texas Water Development Board. Saatavissa: http://www.twdb.texas.gov/publications/reports/numbered_reports/doc/r363/c1.pdf. Viitattu 8.4.2018.

Kupari Solutions Oy. 2018. Tuotteet. Saatavissa: http://www.juomavesi.fi/tuotteet/. Viitattu 28.5.2018.

Manderbacka, L. 2018. Tuotekehitysinsinöörin haastattelu 4.5.2018. Double M Properties Ab. Kokkola.

Maxim Integrated. 2015. DS18B20. Datalehti. Saatavissa: https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf. Viitattu 4.5.2018.

Maxim Integrated. 2011. DS9490R/DS9490B. Datalehti. Saatavissa: https://datasheets.maximintegrated.com/en/ds/DS9490-DS9490R.pdf. Viitattu 7.5.2018.

O'Neill, K. 2015. Feeling Salty: Regulating Desalination Plants in the United States and Spain. Cornell International Law Journal. Volume 48. Issue 2 Issue 2 – Spring 2015, 452–459. Saatavissa: https://scholarship.law.cornell.edu/cgi/viewcontent.cgi?article=1867&context=cilj. Viitattu 28.5.2018.

Onnline. 2009. TRMS IP67 Digital multimeter | 920. Manual. Saatavissa: http://kesko-onninen-pim-resources-production.s3-website-eu-west-1.amazonaws.com/pimdocuments/6754001%20manual%20%28Onnline%20920%20v090630%29.pdf. Viitattu 10.6.2018.

Owfs Development site. 2018. Owfs man page. Saatavissa: http://owfs.org/index.php?page=owfs. Viitattu 25.5.2018.

Paul, D. 2018. David H. Paul Inc. Membrane Technologies for Industrial Water Treatment. Saatavissa: http://www.waterworld.com/articles/iww/print/volume-6/issue-5/features/membrane-technologies-for-industrial-water-treatment.html. Viitattu 28.4.2018.

Python Software Foundation. a. sys – System-specific parameters and functions. Saatavissa: https://docs.python.org/2/library/sys.html. Viitattu 27.5.2018.

Python Software Foundation. b. list. Saatavissa: https://docs.python.org/2/glossary.html#term-list. Viitattu 27.5.2018.

Raspberry Pi Foundation. Raspberry Pi Documentation. Saatavissa: https://www.raspberrypi.org/documentation/hardware/raspberrypi/usb/README.md. Viitattu 7.5.2018.

Raspberry Pi Foundation. 2018. Raspbian Release notes. Saatavissa: http://downloads.raspberrypi.org/raspbian/release_notes.txt. Viitattu 15.5.2018.

Rouse, M. 2006. microprocessor (logic chip). TechTarget. Saatavissa: https://whatis.techtarget.com/definition/microprocessor-logic-chip. Viitattu 29.5.2018.

Rouse, M. 2016. embedded system. TechTarget. Saatavissa: https://internetofthingsagenda.techtarget.com/definition/embedded-system. Viitattu 29.5.2018.

Sander, F. 2018a. Tuotekehitysinsinöörin henkilökohtainen tiedonanto, haastattelu 4.5.2018. Double M Properties Ab. Kokkola.

Sander, F. 2018b. Kuva 6, oikea puoli

Seawater desalination technologies. 2015. Legislative Council of the Hong Kong Special Administrative Region. Research Office, Information Services Division, Legislative Council Secretariat. Fact Sheet FS07/14-15. Saatavissa: https://www.legco.gov.hk/research-publications/english/1415fs07-seawater-desalination-technologies-20150930-e.pdf. Viitattu 8.4.2018.

SFS 6000-5-52. Pienjännitesähköasennukset. Osa 5-52: Sähkölaitteiden valinta ja asentaminen. Johtojärjestelmät. 2017 Helsinki: Suomen Standardisoimisliitto SFS.

SFS 6000-5-54. Pienjännitesähköasennukset. Osa 5-54: Sähkölaitteiden valinta ja asentaminen. Maadoittaminen ja suojajohtimet. 2017 Helsinki: Suomen Standardisoimisliitto SFS.

Sippola, P. 2018. Kuva 6, vasen puoli.

Sivula, T. 2018. Kehitysjohtajan haastattelu ja henkilökohtainen tiedonanto, tapaaminen 30.5.2018. Double M Properties Ab. Kokkola.

Smith, M. 2018. What is MQTT and When You Should Use It. Saatavissa: https://www.pubnub.com/blog/what-is-mqtt-use-cases/. Viitattu 8.6.2018.

Stenebo, P. 2017. Libmbus. Saatavissa: http://bends.se/?page=anteckningar/automation/m-bus/libmbus. Viitattu 25.5.2018.

STK-Tietopalvelut Oy. CPCF406013T. Saatavissa: https://www.ensto.com/fi/tuotteet/teollisuuskotelot/kestomuovikotelot/ensto-cubo-o-c-w-ja-p/ensto-cubo-c--kotelot-laippa-aihiot-polykarbonaatti/CPCF406013T/. Viitattu 5.5.2018.

Sähköturvallisuuden Edistämiskeskus ry. IP-luokitus. Saatavissa: https://www.stek.fi/Perustietoa_sahkosta/Sahkojarjestelmat/fi_FI/IP_luokitus/. Viitattu 3.6.2018.

Techbase Group Sp.z o.o. M-Bus / RS-232 transmission converter. Datalehti.Saatavissa: http://www.a2s.pl/en/images/_articles/MBus/20090814_MBus10_v2_eng.pdf. Viitattu 8.5.2018.

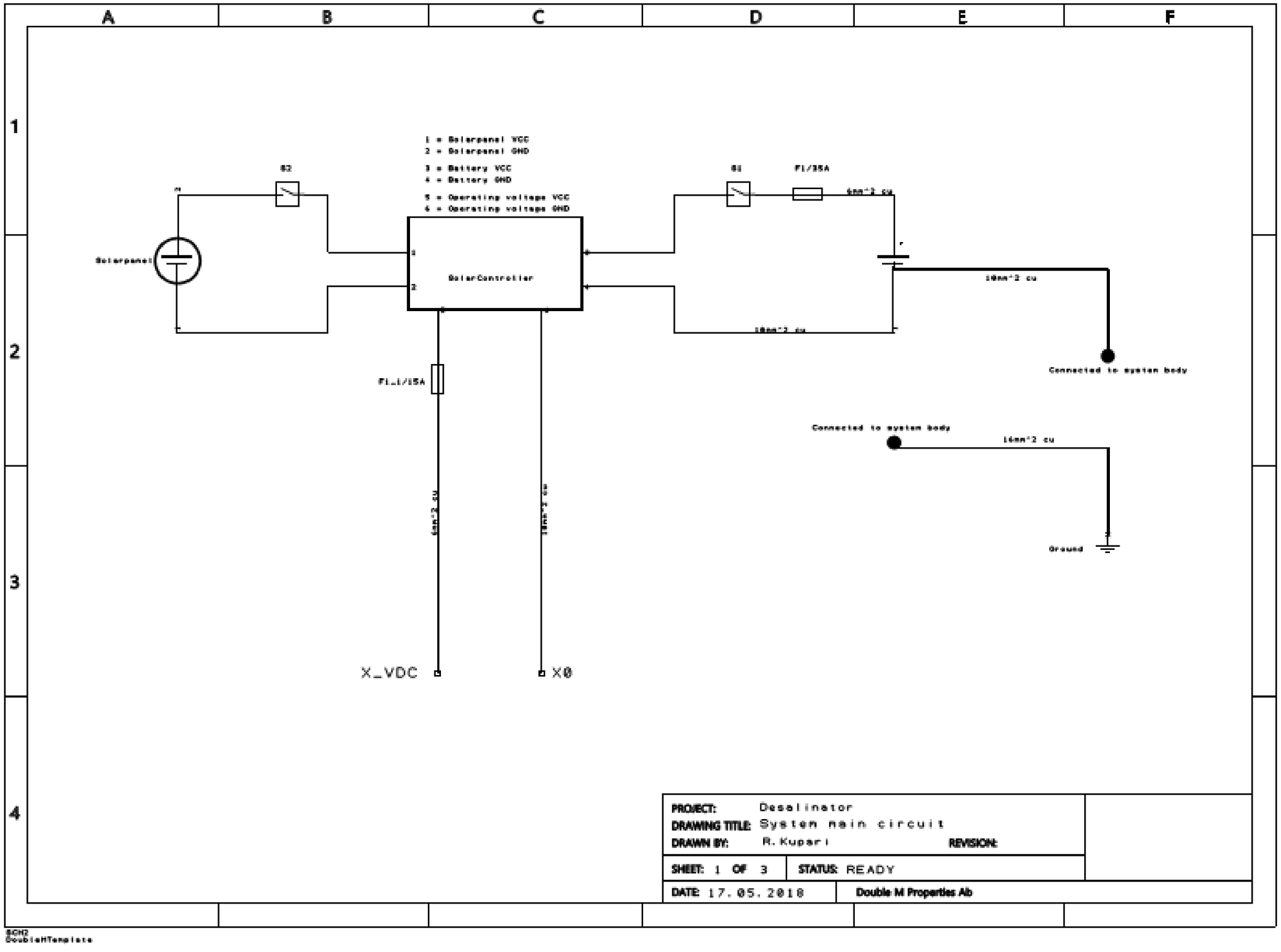
ThermExcel. 2003. Steam characteristics (from 0 to 30 bar). Saatavissa: https://www.thermexcel.com/english/tables/vap_eau.htm. Viitattu 13.6.2018.

Texas Instruments. 2013. Automotive Adaptive Front-lighting System Reference Design. Saatavissa: http://www.ti.com/lit/ug/spruhp3/spruhp3.pdf. Viitattu 29.5.2018.

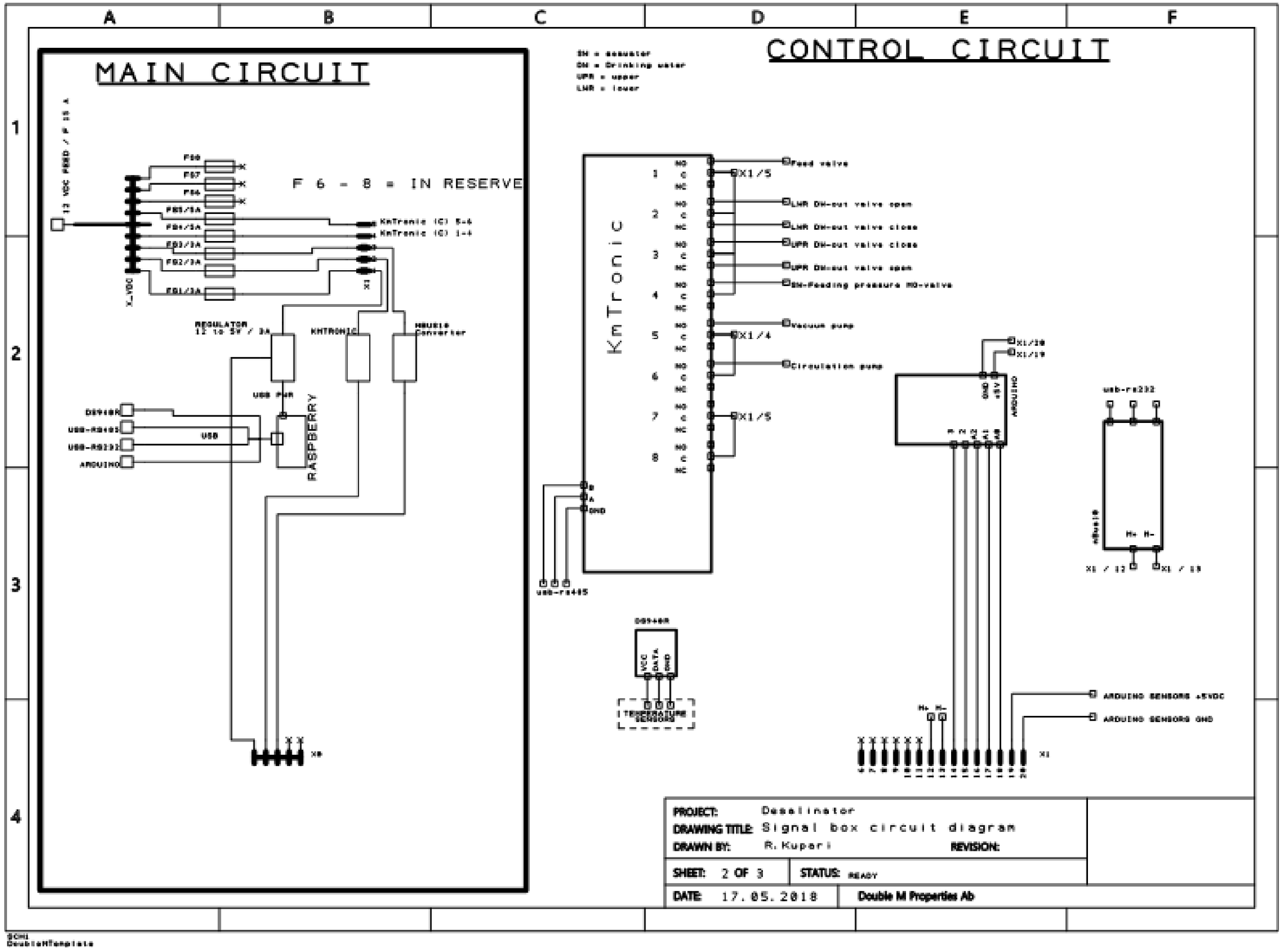
The phpMyAdmin devel team. 2018. Import and export. Saatavissa: https://docs.phpmyadmin.net/en/latest/import_export.html. Viitattu 9.6.2018.

Voutchkov, N. 2016. Desalination – Past, Present and Future. International Water Association. Saatavissa: http://www.iwa-network.org/desalination-past-present-future/. Viitattu 28.5.2018.

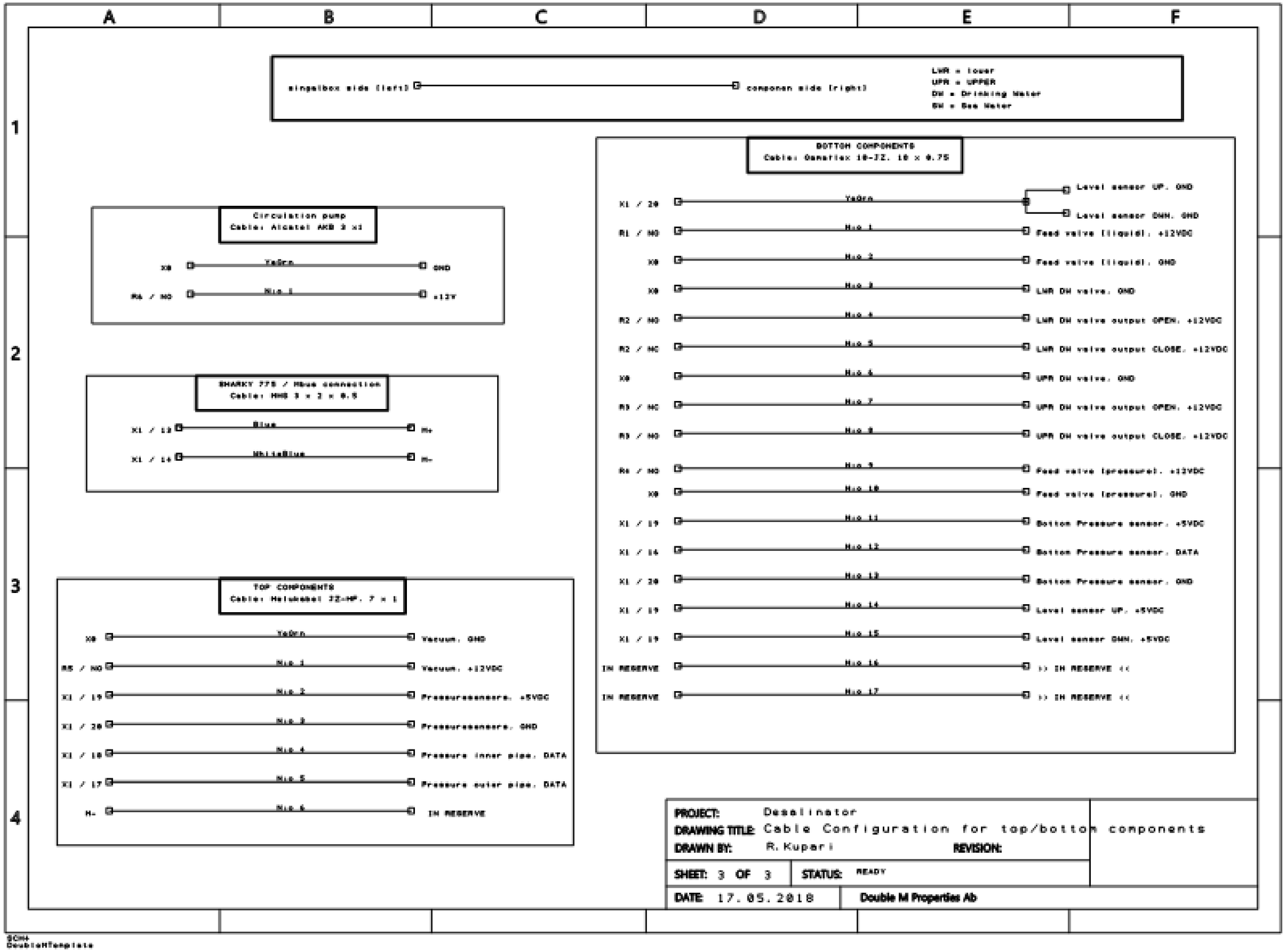
Järjestelmän päävirtapiiri



Keskuksen ohjaus- ja päävirtapiiri



Johdotuskaaviot

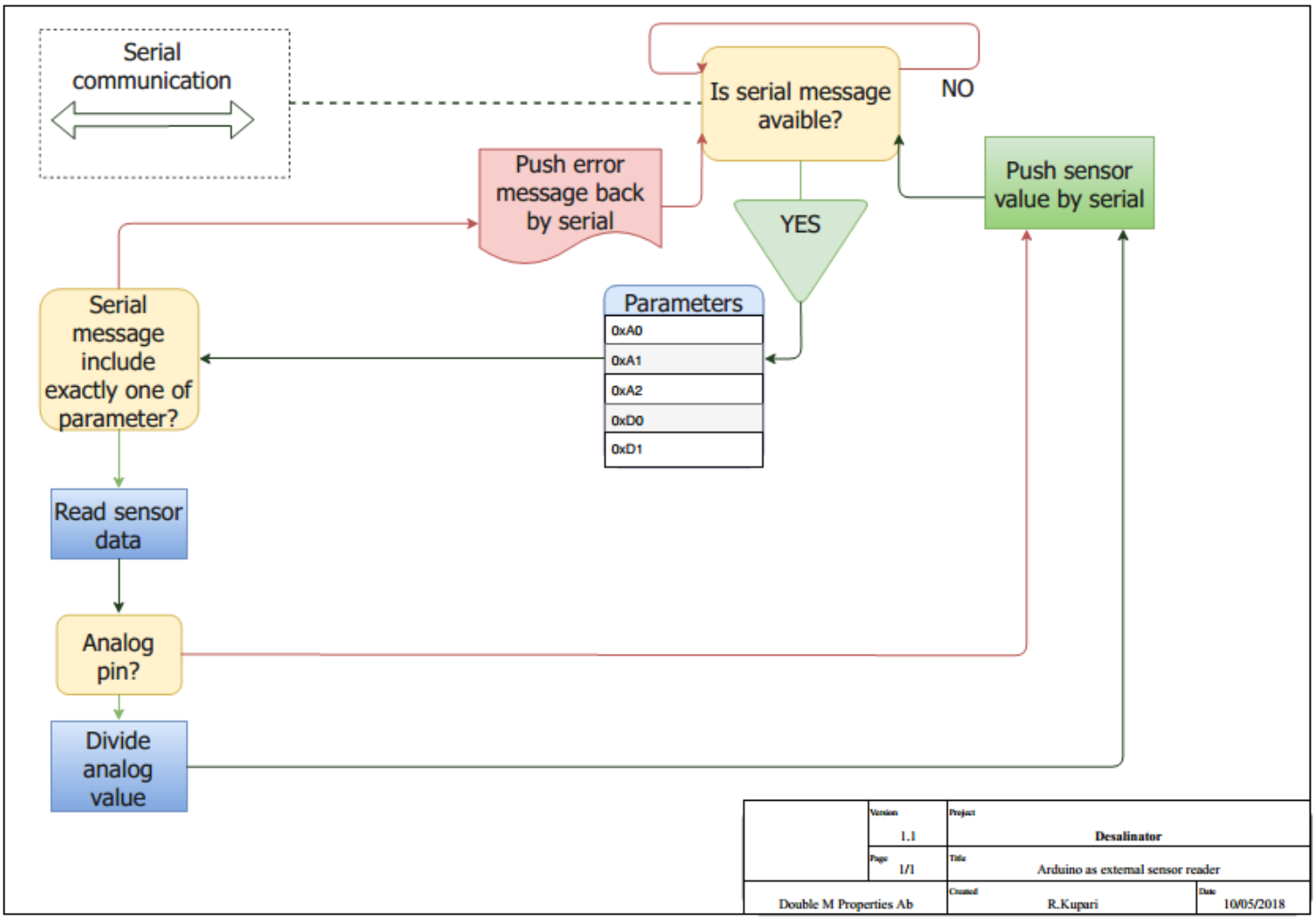


Arduinon sisäinen ohjelma

```
int pressure1 = A0; // innertop
int pressure2 = A1; // outertop
int pressure3 = A2; // bottom
int level1 = 2; // top
int level2 = 3; // low


String income; // presenting the global variable where the incoming parameter will
be stored

void setup()
{

  Serial.begin(19200); // Serial speed
  Serial.setTimeout(50); // Timeout to serial reading process, milliseconds
  pinMode(level1, INPUT);
  pinMode(level2, INPUT);
  digitalWrite(level1, LOW);
}

float valueRet(String hex)
{
  int sensorVal = 999; // just initialize the return value -> this will be changed
if pin is readed

  //-------------------------------------------------------------------------------
-------------------
  // Firs cheking is the incoming parameter correct
  // if so process will continue to reading pin and store new individual value
  // to variable which will been returned

  // if the incoming parameter doesnt match any initialized parameter (else part)
  // program will write directly on serial line to raspberry error messange


  if (hex == "0xA0") {sensorVal = analogRead(pressure1);}
  else if (hex == "0xA1") {sensorVal = analogRead(pressure2);}
  else if (hex == "0xA2") {sensorVal = analogRead(pressure3);}
  else if (hex == "0xD0") {sensorVal = digitalRead(level1);}
  else if (hex == "0xD1") {sensorVal = digitalRead(level2);}
  else {Serial.print("Wrong given argument [0x  + A0, A1, A2, D0, D1]");}


  //-------------------------------------------------------------------------------
-------------------


  // Here will be tested one more time is the parameter correct
  // if so, the oin has been readed and sensorVal have some other value in store
  // this stored value is has passed analog-to-digital converting process
  // so the division included to "analog" visibility value
  // and it will been returned there where this function has been called (to loop)

  // IF the parameter is not correct, this function will no return anything
```

```
  if (hex == "0xA0" || hex == "0xA1" || hex == "0xA2")
    {
      //Serial.println(sensorVal);
      float voltage = sensorVal * (5.0 / 1023.0);
      //Serial.println("ok");
      return voltage;
    }

  else if (hex == "0xD0" || hex == "0xD1")
    {
      return sensorVal;
    }


}


// The endless loop

void loop()
{

  if (Serial.available()) // IF incoming message occur
    {

      income = Serial.readString(); // Read value of incoming parameter from buf-
fer
      float sendVal = valueRet(income); // forwarding readed value and making call
to valueRet function
      Serial.println(sendVal); // forwarding valueRet return value to Raspberry

    }

}
```

Vuokaavuo Arduinon sisäisestä päätöksenteosta



Arduino python käsittelijä (cli/handlers/arduino/arduino.py)

```
import serial, time



    #serial communication parameters
address = '/dev/arduino'
baudrate = 19200
timeout = 1

# serial messages to Arduino
    # Top pressure sensor, placed inner pipe is connected to arduino port "A0"
    # Top pressure sensor, placed outer pipe is connected to arduino port "A1"
    # Bottom pressure sensor, placed bottom of pipe is connected to arduino port
"A2"
    # Top level sensor, placed seawater intake tank physically higher stage is
connected to arduino pin "2"
    # Low level sensor, placed seawater intake tank physically lower stage is con-
nected to arduino pin "3"
        # sending message type is string, see content of message from getValue or
getValues functions
        # Arduino will answer message with voltage value (Level sensors will be
high as 1.00 or low as 0.00)
            # 1.00 means true for water and vice-versa


    # will return calculated pressure value from voltage value
        # formula is from pressuresensors datasheet
        # CorrectionFactor is evolved with calculated value (without correction)
divided with current pressure value (foreca/kokkola)
        # this function is called from each pressuresensor values
def calcPressure(Vout):
    Vs = 5.1
    CorrectionFactor = 1.013/0.978
    pressure = float(((Vout/Vs+0.04)/0.004)/100*CorrectionFactor)
    pressure = round(pressure, 2)
    return pressure


    # this function is not in use in system
        # will return individual value of parameter sensor identity
        # level sensorvalues will be returned by boolean variable
        # Pressuresensor variables will be returned by float which rounded on 2
dec
def getValue(sensor):
    if sensor == "innertop":
        serialCom = serial.Serial(address, baudrate=baudrate, timeout=timeout)
        time.sleep(2)
        serialCom.write("0xA0")
        value1 = serialCom.readline()
        pressure1 = calcPressure(float(value1))
        return pressure1


    if sensor == "outertop":
        serialCom = serial.Serial(address, baudrate=baudrate, timeout=timeout)
        time.sleep(2)
        serialCom.write("0xA1")
        value2 = serialCom.readline()
```

```
        pressure2 = calcPressure(float(value2))
        return pressure2

    if sensor == "bottom":
        serialCom = serial.Serial(address, baudrate=baudrate, timeout=timeout)
        time.sleep(2)
        serialCom.write("0xA2")
        value3 = serialCom.readline()
        pressure3 = calcPressure(float(value3))
        return pressure3



    if sensor == "toplevel":
        serialCom = serial.Serial(address, baudrate=baudrate, timeout=timeout)
        time.sleep(2)
        serialCom.write("0xD0")
        value4 = serialCom.readline()
        if float(value4) < 1:
            return False
        else:
            return True

    if sensor == "bottomlevel":
        serialCom = serial.Serial(address, baudrate=baudrate, timeout=timeout)
        time.sleep(2)
        serialCom.write("0xD1")
        value5 = serialCom.readline()
        if float(value5) < 1:
            return False
        else:
            return True

    serialCom.close()



    # This function is in use on readAllvalues.py program
        # this function will call local calcPressure -function
        # all produced values will be returned on list object which contain loca-
tion information on same index
def getValues():

    # connection
    serialCom = serial.Serial(address, baudrate=baudrate, timeout=timeout)
    time.sleep(2)


    # INNER TOP PRESSURE SENSOR
    serialCom.write("0xA0")
    value1 = serialCom.readline()
    pressure_inner_top = calcPressure(float(value1))

    # OUTER TOP PRESSURE SENSOR
    serialCom.write("0xA1")
    value2 = serialCom.readline()
    pressure_outer_top = calcPressure(float(value2))
```

```
# BOTTOM PRESSURE SENSOR
serialCom.write("0xA2")
value3 = serialCom.readline()
pressure_bottom = calcPressure(float(value3))



# TOP LEVEL SENSOR OF INTAKE TANK
serialCom.write("0xD0")
value4 = serialCom.readline()
if float(value4) < 1:
    topLevel = False
else:
    topLevel = True

# BOTTOM LEVEL SENSOR OF INTAKE TANK
serialCom.write("0xD1")
value5 = serialCom.readline()
if float(value5) < 1:
    bottomLevel = False
else:
    bottomLevel = True

serialCom.close()

returnValue = [
'inner top pressure:  ' +str(pressure_inner_top),
'outer top pressure:  ' + str(pressure_outer_top),
'bottom pressure:     ' + str(pressure_bottom),
'top level sensor:    ' + str(topLevel),
'bottom level sensor: ' + str(bottomLevel)
]

return returnValue
```

M-bus sanoma

```
<MBusData>

   <SlaveInformation>
      <Id>53214365</Id>
      <Manufacturer>HYD</Manufacturer>
      <Version>32</Version>
      <ProductName></ProductName>
      <Medium>Heat: Outlet</Medium>
      <AccessNumber>59</AccessNumber>
      <Status>00</Status>
      <Signature>0000</Signature>
   </SlaveInformation>

   <DataRecord id="0">
      <Function>Instantaneous value</Function>
      <Unit>Energy (kWh)</Unit>
      <Value>1945</Value>
      <Timestamp>2018-06-11T10:46:38</Timestamp>
   </DataRecord>

   <DataRecord id="1">
      <Function>Instantaneous value</Function>
      <Unit>Volume (m m^3)</Unit>
      <Value>1223899</Value>
      <Timestamp>2018-06-11T10:46:38</Timestamp>
   </DataRecord>

   <DataRecord id="2">
      <Function>Instantaneous value</Function>
      <Unit>Volume flow (m m^3/h)</Unit>
      <Value>28</Value>
      <Timestamp>2018-06-11T10:46:38</Timestamp>
   </DataRecord>

   <DataRecord id="3">
      <Function>Instantaneous value</Function>
      <Unit>Power (W)</Unit>
      <Value>977</Value>
      <Timestamp>2018-06-11T10:46:38</Timestamp>
   </DataRecord>

   <DataRecord id="4">
      <Function>Instantaneous value</Function>
      <Unit>Flow temperature (1e-1 deg C)</Unit>
      <Value>731</Value>
      <Timestamp>2018-06-11T10:46:38</Timestamp>
   </DataRecord>
```

```
  <DataRecord id="5">
      <Function>Instantaneous value</Function>
      <Unit>Return temperature (1e-1 deg C)</Unit>
      <Value>428</Value>
      <Timestamp>2018-06-11T10:46:38</Timestamp>
    </DataRecord>

    <DataRecord id="6">
      <Function>Instantaneous value</Function>
      <Unit>Energy (kWh)</Unit>
      <Value>0</Value>
      <Timestamp>2018-06-11T10:46:38</Timestamp>
    </DataRecord>

    <DataRecord id="7">
      <Function>Instantaneous value</Function>
      <Unit>Operating time (hours)</Unit>
      <Value>335</Value>
      <Timestamp>2018-06-11T10:46:38</Timestamp>
    </DataRecord>

</MBusData>
```

M-Bus käsittelijä (cli/handlers/mbus/mbus.py)

```
import subprocess

    # This function will call alias witch is pointing to mbus data request
    # result or error will be returned
def getTheData():

    request = subprocess.Popen(["/bin/bash", "-i", "-c", "mbusReq"],
stdout=subprocess.PIPE, shell=False)
    data, error = request.communicate()
    data = data.split()

    return data, error

#variables for next function uses
values = []
retData = []

valueNamesAndUnits = [
'Total Energy (kWh): ',
'Total Volume (m^3): ',
'VolumeFlow (liters): ',
'Power (kW): ',
'Flow temperature (Celcius): ',
'Return temperature (Celcius): ',
'Current Energy (kWh): '
]


    # this function will be called from readAllValues.py
        # Mbusdata will be handled by each value specific id 0 - 6
        # Handled values and their names will be returned
        # id(1) will be divided with value 1000 = m m^3 to m^3
        # id(3) will be divided with value 1000 = W to kW
        # id(4) and id(5) will be divided with value 10 so get Celcius values
def getValues():
    data, error = getTheData()
    if data:
        for id in range(0,7):
            searchSegment1 = "id="+'"'+str(id)+'"'+">"
            searchBetween1 = "<Value>"

            segment1 = data.index(searchSegment1)

            for i in data[segment1:]:
                if searchBetween1 in i:
                    RawValue = i
                    break

            numberValue = ''.join(number for number in RawValue if number.isdi-
git())
            values.append(float(numberValue))

            if id == 1 or id == 3:
                values[id] = values[id]/1000 #id = m^3 and 3 = kW

            elif id == 4 or id == 5:
                values[id] = values[id]/10 # id 4 and 5 = celcius
```

```
    for i in range(0,7):
        retData.append(valueNamesAndUnits[i] + str(values[i]))

    return retData

else:
    return error
```

Lämpötilasensoreiden osoite-konfiguraatio (lib/sensorInfo/sensor_conf.txt)

28.76AE33090000EB
28.5575200900009A
28.226ABD0700001C
28.0F9C2009000052
28.7A4DBD0700003A
28.A18D330900001C
28.9AAE33090000BF
28.443034090000FB
28.039C200900002F
28.FF488CA11605
28.FFDAA5A11605
28.FF2E15B0160

Lämpötilasensoreiden osoite-konfiguraatio (lib/sensorInfo/sensor_positions.txt)

```
1. (dimense_27_bottom                )
2. (dimense_77                       )
3. (dimense_127                      )
4. (dimense_177                      )
5. (dimense_227                      )
6. (dimense_ 277                     )
7. (dimense_327                      )
8. (dimense_377                      )
9. (dimense_427_top                  )
10. (individual sensor, salt drain   )
11. (individual sensor, drinking water)
12.(individual sensor, sw-input      )
```

Lämpötilasensoreiden käsittelijä (cli/handlers/temperature/temperature.py)

```
import os


    #this function will get sensor addresses (usually begin with 28.)
        # and their physical called positions
        # connection between these two textfiles will occured by line index
            # eg. The first address line will linked to first position
def getAdressesAndPositions():

    # get addresses of sensors
    file = open("/home/pi/desalinator_sw/lib/sensorInfo/sensor_conf.txt", "r")
    addresses = file.readlines()
    addresses = map(str.strip, addresses)

    # get physical positions of sensors
    file2 = open("/home/pi/desalinator_sw/lib/sensorInfo/sensor_positions.txt",
"r")
    positions = file2.readlines()
    positions = map(str.strip, positions)


    file.close()
    file2.close()
    return addresses, positions


    # This function will call function above
    # This function will be called from readAllValues.py
    # Function will first start the owfs virtual filesystem
        # Then its looking data by readed sensor address
            # if data will not be visible in filesystem, it will start filesystem
again
            # if some data is missing, list index will be replaced with error
        # Function will return all readed data (and errors if occured) on list ob-
ject.
def getValues():

    retVal = []
    retErr = []

    # calling adresses and positions
    address, position = getAdressesAndPositions()

    # First start owfs system
    os.system('sudo bash /etc/init.d/owfs start > /dev/null')

    ii = 0

    for i in address:
        try:
            tempFile=os.path.join("/","mnt","1wire",i,"temperature")
            tempByAddress=open(tempFile,'r')
            degree=tempByAddress.read()
            temp = (position[ii] + " : " + i + ": " + degree + " 'C")
            retVal.append(temp)
            tempByAddress.close()
            ii += 1
```

```
        except Exception as e:
            os.system('sudo bash /etc/init.d/owfs start > /dev/null')
            retErr.append(i)

    if len(address) != len(retVal):
        for i in retErr:
            if len(i) > 5:
                retVal.append("System cant read some temperature information from
address:  " + i)

    return retVal
```

Releen status käsittelijä (cli/handlers/component_status/component_status.py)

```
import serial, time


# handler will call KMtronic with specific request message. In this case "0xA1"
mean relay dip position "1".
    # future dip positions will be as "0xA2" wich mean dip position "2", etc..
# KMtronic will answer all 8ch statuses in one message as
"0xFF/0xA1/0x00/0x00/0x00 ..."
    #first value after "0xA1" which is "0x00" means in this case the first relay
is normal position
        #If value will be "0x01" it means realay is controlled. Relay will return
eight values.


    # serial connection function
        # this function will catch the return message and return it
def getFromRelay():
    serialCom = serial.Serial('/dev/relay',9600, timeout=3)
    request = chr(0xFF) + chr (0xA1) + chr (0x00)
    serialCom.write(request)
    time.sleep(1)
    raw_result = str(serialCom.readlines())
    serialCom.close()
    return raw_result


    #this function is called from readAllValues.py program
        # this function will call the above connection function
        # Received message will be handled in this function. Names and values will
be included own list objects with same index values.
            # see the relay component wiring from electrical documentation
            # Seawater top output value will be inverted in this function

def getValues():
    relay = []
    component = []
    fullStatus = getFromRelay()


        # Status of relay
    for i in range(10, len(fullStatus)):
        if fullStatus[i] == 'x':
            if fullStatus[i+2] == '1':
                relay.append(True)
            else:
                relay.append(False)



        # Status of components
    for i in range(10, len(fullStatus)):
        if fullStatus[i] == 'x':
            if fullStatus[i+2] == '1':
                component.append(True)
            else:
                component.append(False)
```

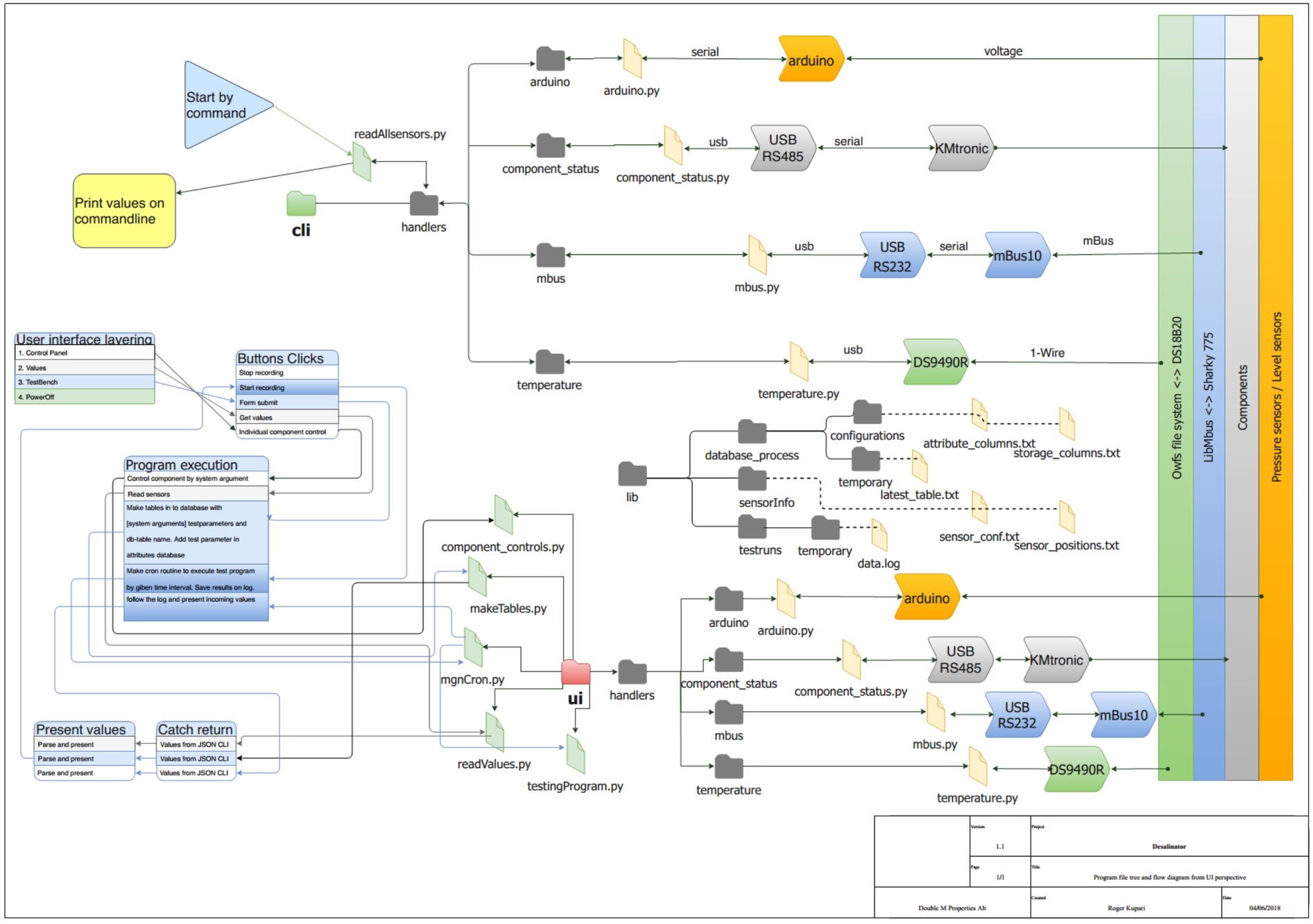
```
    # upper sw-out valve is normally opened physically
if component[2]:
    component[2] = False
else:
    component[2] = True

return component, relay
```

Komentokehotteen pääohjelma (cli/readAllValues.py)

```
import sys


# this program will call handlers and present returned data on command line


# path to handlers

#pressure and levelsensors -> by arduino
arduino = '/home/pi/desalinator_sw/cli/handlers/arduino'

#temperature sensors
temperature = '/home/pi/desalinator_sw/cli/handlers/temperature'

# mbus data from sharky
mbus = '/home/pi/desalinator_sw/cli/handlers/mbus'

# Statuses from relay
relay = '/home/pi/desalinator_sw/cli/handlers/component_status'


print "getting values... "

# getting data from arduino
sys.path.append(arduino)
import arduino

print "pressures and levels ...(1/4)"

#store result in variable
arduinoValues = arduino.getValues()


# getting data from 1-wire system
sys.path.append(temperature)
import temperature

print "temperatures ...(2/4)"
#store result in variable
temperatureValues = temperature.getValues()

#getting mbusdata from sharky
sys.path.append(mbus)
import mbus

print "mbus ...(3/4)"
mbusValues = mbus.getValues()

print "relay ...(4/4)"
sys.path.append(relay)
import component_status

components_status, relay_status = component_status.getValues()
```

```
print "pressure and levels"
for i in arduinoValues:
    print i

print "\ntemperatures"
for i in temperatureValues:
    print i

print "\nmbus data"
for i in mbusValues:
    print i

print "\n relay statuses, False = normal stage"
for i in relay_status:
    print i

print "done"
```

Yleismallinen vuokaavio koko suolanpoisto-ohjelmiston yleisriippuvuuksista



Graafisen käyttöliittymän taustalla toimivat python ohjelmat (ui/handlers/sensors/arduino/arduino.py)

```
import serial, time


# This handler is almost same as commandline side arduino handler
# except the return value type is dictionary object


address = '/dev/arduino'
baudrate = 19200
timeout = 1



def calcPressure(Vout):
    Vs = 5.1
    CorrectionFactor = 1.013/0.978
    pressure = float(((Vout/Vs+0.04)/0.004)/100*CorrectionFactor)
    pressure = round(pressure, 2)
    return pressure

def getValue(sensor):
    if sensor == "innertop":
        serialCom = serial.Serial(address, baudrate=baudrate, timeout=timeout)
        time.sleep(2)
        serialCom.write("0xA0")
        value1 = serialCom.readline()
        pressure1 = calcPressure(float(value1))
        return pressure1


    if sensor == "outertop":
        serialCom = serial.Serial(address, baudrate=baudrate, timeout=timeout)
        time.sleep(2)
        serialCom.write("0xA1")
        value2 = serialCom.readline()
        pressure2 = calcPressure(float(value2))
        return pressure2

    if sensor == "bottom":
        serialCom = serial.Serial(address, baudrate=baudrate, timeout=timeout)
        time.sleep(2)
        serialCom.write("0xA2")
        value3 = serialCom.readline()
        pressure3 = calcPressure(float(value3))
        return pressure3



    if sensor == "toplevel":
        serialCom = serial.Serial(address, baudrate=baudrate, timeout=timeout)
        time.sleep(2)
        serialCom.write("0xD0")
        value4 = serialCom.readline()
        if float(value4) < 1:
            return False
        else:
```

```
            return True

    if sensor == "bottomlevel":
        serialCom = serial.Serial(address, baudrate=baudrate, timeout=timeout)
        time.sleep(2)
        serialCom.write("0xD1")
        value5 = serialCom.readline()
        if float(value5) < 1:
            return False
        else:
            return True

    serialCom.close()

def getValues():

    # connection
    serialCom = serial.Serial(address, baudrate=baudrate, timeout=timeout)
    time.sleep(2)


    # INNER TOP PRESSURE SENSOR
    serialCom.write("0xA0")
    value1 = serialCom.readline()
    pressure_inner_top = calcPressure(float(value1))

    # OUTER TOP PRESSURE SENSOR
    serialCom.write("0xA1")
    value2 = serialCom.readline()
    pressure_outer_top = calcPressure(float(value2))


    # BOTTOM PRESSURE SENSOR
    serialCom.write("0xA2")
    value3 = serialCom.readline()
    pressure_bottom = calcPressure(float(value3))



    # TOP LEVEL SENSOR OF INTAKE TANK
    serialCom.write("0xD0")
    value4 = serialCom.readline()
    if float(value4) < 1:
        topLevel = False
    else:
        topLevel = True

    # BOTTON LEVEL SENSOR OF INTAKE TANK
    serialCom.write("0xD1")
    value5 = serialCom.readline()
    if float(value5) < 1:
        bottomLevel = False
    else:
        bottomLevel = True

    serialCom.close()

    returnValue = {
    'arduino_inner_top_pressure':pressure_inner_top,
```

```
'arduino_outer_top_pressure':pressure_outer_top,
'arduino_bottom_pressure':pressure_bottom,
'arduino_top_level_sensor':topLevel,
'arduino_bottom_level_sensor':bottomLevel
}

return returnValue
```

(ui/handlers/sensors/component_status/component_status.py)

```
import serial, time


# this handler will be most same than commandline component status handler
# Except here will be component statusses presented on strings
    # Return value type is dictionary object


def getFromRelay():
    serialCom = serial.Serial('/dev/relay',9600, timeout=3)
    request = chr(0xFF) + chr (0xA1) + chr (0x00)
    serialCom.write(request)
    time.sleep(1)
    raw_result = str(serialCom.readlines())
    serialCom.close()
    return raw_result


def getValues():
    relay = []
    component = []
    fullStatus = getFromRelay()


        # Status of relay
    for i in range(10, len(fullStatus)):
        if fullStatus[i] == 'x':
            if fullStatus[i+2] == '1':
                relay.append(True)
            else:
                relay.append(False)



        # Status of components
    for i in range(10, len(fullStatus)):
        if fullStatus[i] == 'x':
            if fullStatus[i+2] == '1':
                component.append(True)
            else:
                component.append(False)

        # individual component stages
    if component[0]:
        component[0] = "open"
    else:
        component[0] = "closed"

    if component[1]:
        component[1] = "open"
    else:
        component[1] = "closed"

    if component[2]:
        component[2] = "closed"
    else:
        component[2] = "open"
```

```
if component[3]:
    component[3] = "open"
else:
    component[3] = "closed"

if component[4]:
    component[4] = "on"
else:
    component[4] = "off"

if component[5]:
    component[5] = "on"
else:
    component[5] = "off"

if component[6]:
    component[6] = "controlled stage"
else:
    component[6] = "normal stage"

if component[7]:
    component[7] = "controlled stage"
else:
    component[7] = "normal stage"

'''
retRelay = {
'R1': relay[0],
'R2': relay[1],
'R3': relay[2],
'R4': relay[3],
'R5': relay[4],
'R6': relay[5],
'R7': relay[6],
'R8': relay[7]
}
'''


retComponent = {
'relay_sw_intake_valve': component[0],
'relay_sw_out_lwr_valve': component[1],
'relay_sw_out_upr_valve': component[2],
'relay_intake_pressure_valve': component[3],
'relay_vacuum_pump': component[4],
'relay_circulation_pump': component[5],
'relay_R7': component[6],
'relay_R8': component[7]
}


return retComponent
```

(ui/sensors/mbus/mbus.py)

```
import subprocess


# This handler will be almost same than commandline mbus handler
    # except return value type is dictionary object


# catch the mbus output from sharky
def getTheData():

    request = subprocess.Popen(["/bin/bash", "-i", "-c", "mbusReq"],
stdout=subprocess.PIPE, shell=False)
    data, error = request.communicate()
    data = data.split()

    return data, error

values = []


valueNamesAndUnits = [
'Total Energy (kWh)',
'Total Volume (m^3)',
'VolumeFlow (liters)',
'Power (kW): ',
'Flow temperature (Celcius)',
'Return temperature (Celcius)',
'Current Energy (kWh)'
]



def getValues():
    data, error = getTheData()
    if data:
        for id in range(0,7):
            searchSegment1 = "id="+'"'+str(id)+'"'+">"
            searchBetween1 = "<Value>"

            segment1 = data.index(searchSegment1)

            for i in data[segment1:]:
                if searchBetween1 in i:
                    RawValue = i
                    break

            numberValue = ''.join(number for number in RawValue if number.isdi-
git())
            values.append(float(numberValue))

            if id == 1 or id == 3:
                values[id] = values[id]/1000 #id 1 = m^3 and 3 = kW

            elif id == 4 or id == 5:
                values[id] = values[id]/10 # id 4 and 5 = celcius
```

```
        retData = {
        'mbus_total_energy_kwh': values[0],
        'mbus_total_Volume_m3': values[1],
        'mbus_volume_flow_liters': values[2],
        'mbus_power_kw': values[3],
        'mbus_flow_temp_c': values[4],
        'mbus_return_temp_c': values[5],
        'mbus_current_energy_kwh': values[6]
        }

        return retData

    else:
        return error
```

(ui/handlers/sensors/temperature/temperature.py)

```
import os

# This handler is almost same as commandline temperature handler
    # Return value type is dictionary object

def getAdressesAndPositions():

    # get addresses of sensors
    file = open("/home/pi/desalinator_sw/lib/sensorInfo/sensor_conf.txt", "r")
    addresses = file.readlines()
    addresses = map(str.strip, addresses)

    # get physical positions of sensors
    file2 = open("/home/pi/desalinator_sw/lib/sensorInfo/sensor_positions.txt",
"r")
    positions = file2.readlines()
    positions = map(str.strip, positions)


    file.close()
    file2.close()
    return addresses, positions


def getValues():

    retVal = []
    retErr = []

    # calling adresses and positions
    address, position = getAdressesAndPositions()

    # First start owfs system
    os.system('sudo bash /etc/init.d/owfs start > /dev/null')

    ii = 0

    for i in address:
        try:
            tempFile=os.path.join("/","mnt","1wire",i,"temperature")
            tempByAddress=open(tempFile,'r')
            degree=tempByAddress.read()
            temp = (degree)
            retVal.append(temp)
            tempByAddress.close()
            ii += 1

        except Exception as e:
            os.system('sudo bash /etc/init.d/owfs start > /dev/null')
            retErr.append(i)

    if len(address) != len(retVal):
        for i in retErr:
            if len(i) > 5:
                retVal.append("System cant read some temperature information from
address:  " + i)
```

```
retData = {
'temperature_dimense_27' : float(retVal[0]),
'temperature_dimense_77' : float(retVal[1]),
'temperature_dimense_127' : float(retVal[2]),
'temperature_dimense_177' : float(retVal[3]),
'temperature_dimense_227' : float(retVal[4]),
'temperature_dimense_277' : float(retVal[5]),
'temperature_dimense_327' : float(retVal[6]),
'temperature_dimense_377' : float(retVal[7]),
'temperature_dimense_427' : float(retVal[8]),
'temperature_individual_salt_drain' : float(retVal[9]),
'temperature_individual_dw' : float(retVal[10]),
'temperature_individual_sw_input' : (retVal[11]),

}

return retData
```

(ui/component_controls.py)

```
import sys, subprocess


# This program will be executed from graphical interface controller part (valves
and motors)
    # program will call alias (wich is commanding the individual channe of relay,
more information = see aliases)
    # system argument will be captured and its onward to be the alias word


var = sys.argv[1]


subprocess.Popen(["/bin/bash", "-i", "-c", str(var)])
```

(ui/makeTables.py)

```
import sys, json
import mysql.connector as my


#THIS program will be called from grapphical user interface / testBench / "make
test" -form submit button
    #System will define the databases and store the given test attributes in fu-
ture use

    # JSON printing is extra-important, without it user interface cannot read (and
present) results


    #paths to names of attribute database columns (desalination_test_attr)
    #paths to names of storage columns (desalination)
attributes_columns_path = '/home/pi/desalinator_sw/lib/database_process/configura-
tions/attribute_columns.txt'
storage_columns_path = '/home/pi/desalinator_sw/lib/database_process/configurati-
ons/storage_columns.txt'

    #Databaseinformation
dbHost = '*****'
dbDes = '*****'
dbDes2 = '******'
dbUser = '*****'
dbPasswd = '*******'

    #Database columns configuration variables
float42 = ' float(4,2), '
float52 = ' float(5,2), '
float51 = ' float(5,1), '
float73 = ' float(7,3), '
float62 = ' float(6,2), '
float53 = ' float(5,3), '
float32 = ' float(3,2), '
tf = ' BOOLEAN, '
txt = ' TEXT, '
commaonly = ', '


    # This function will be read and return attribute and storage columns
def databaseColumns():
    attributes = open(attributes_columns_path)
    attr_raw = attributes.readlines()
    attr = map(str.strip, attr_raw)

    storage = open(storage_columns_path)
    storage_raw = storage.readlines()
    stor = map(str.strip, storage_raw)

    attributes.close()
    storage.close()

    return attr, stor

    # this function will create the table of attributes in ..test_attr database
def createAttrTable(tableName):
        # see the order of variables by lib/database/configurations included files
```

```
    attribute, storage = databaseColumns()

    try:
        db = my.connect(host=dbHost, user=dbUser, password=dbPasswd, data-
base=dbDes2)
        c = db.cursor(buffered = True)
        cmd = ("""
        CREATE TABLE """ + tableName + """ (
        id INT NOT NULL AUTO_INCREMENT, """ +

        attribute[0] + float42 +
        attribute[1] + float42 +
        attribute[2] + float42 +
        """PRIMARY KEY (id)
        )""");
        c.execute(cmd)

        db.commit()
        c.close()
        db.close()


        return True

    except Exception as e:
        sys.stderr.write(str(e))
        return False


        # this function will create the storage "test values storage" in de-
salination database
def makeStorageTable(tableName):
    # see the order of variables by lib/database/configurations included files
    attribute, storage = databaseColumns()

    try:
        db = my.connect(host=dbHost, user=dbUser, password=dbPasswd, data-
base=dbDes)
        c = db.cursor(buffered = True)
        cmd = ("""
        CREATE TABLE """ + tableName + """ (
        id INT NOT NULL AUTO_INCREMENT, """ +

        storage[0] + float52 +
        storage[1] + float52 +
        storage[2] + float52 +
        storage[3] + float52 +
        storage[4] + float52 +
        storage[5] + float52 +
        storage[6] + float52 +
        storage[7] + float52 +
        storage[8] + float52 +
        storage[9] + float52 +
        storage[10] + float52 +
        storage[11] + float52 +
        storage[12] + float62 +
        storage[13] + float73 +
        storage[14] + float51 +
        storage[15] + float53 +
```

```
        storage[16] + float52 +
        storage[17] + float52 +
        storage[18] + float52 +
        storage[19] + tf +
        storage[20] + tf +
        storage[21] + float32 +
        storage[22] + float32 +
        storage[23] + float32 +

        storage[24] + txt +
        storage[25] + txt +
        storage[26] + txt +
        storage[27] + txt +
        storage[28] + txt +
        storage[29] + txt +
        storage[30] + txt +
        storage[31] + txt +
        """PRIMARY KEY (id)
        )""");
        c.execute(cmd)

        db.commit()
        c.close()
        db.close()


        return True

    except Exception as e:
        sys.stderr.write(str(e))
        return False

    # this function will put attribute values on ...test_attr database
def insertAttributes(tableName, hi, lo, time):
    attribute, storage = databaseColumns()

    try:
        db = my.connect(host=dbHost, user=dbUser, password=dbPasswd, data-
base=dbDes2)
        c = db.cursor(buffered = True)
        cmd = ("""INSERT INTO """ +
        tableName
        +
        """
        ( """+
        attribute[0] + commaonly +
        attribute[1] + commaonly +
        attribute[2] +
        """)
        VALUES (%f, %f, %f)"""%(float(hi), float(lo), float(time)));
        c.execute(cmd)
        #print '"'+ tableName +'"' + "  table has done in database "

        db.commit()
        c.close()
        db.close()

        return True
```

```
    except Exception as e:
        sys.stderr.write(str(e))



    # system argument catching
tableName = str(sys.argv[1])
vacuum_hi = str(sys.argv[2])
vacuum_lo = str(sys.argv[3])
interval = str(sys.argv[4])


    # if tables is created with no errors
        # attributes will be saved in database
            # latest name of table (wich is same on both databases [desalination]
and [desalination_test_attr]) will be saved on text file
if createAttrTable(tableName):
    if makeStorageTable(tableName):
        if insertAttributes(tableName, vacuum_hi, vacuum_lo, interval):
            file = open("/home/pi/desalinator_sw/lib/database_process/tempo-
rary/latestTable.txt", "w")
            file.writelines(str(tableName))
            file.close()
            toClient = {"topic": "Database tables Done", "interval": int(inter-
val)}
            print (json.dumps(toClient))
```

(ui/mgnCron.py)

```
import sys, json
from crontab import CronTab
import mysql.connector as my

# JSON printing is extra-important, without it user interface cannot read (and
present) results

# this program will be called from graphical user interface / testBench / start
test -button or stop test -button
    # System will capture system argument which controlling decisions of program
    # if argument is "1", system will delete the old tablename if exist (more in-
formation; see makeTables.py)
        #system will get the latest tableName from text file which will be used to
get the time interval attribute value
            # time interval will be placed on cron routine specific every minute
{time interval}
                # system will excecute testingProgram.py and its result will be
written on /home/pi/desalinator_sw/lib/testruns/temporary/data.log
                    # user interface will monitored this log file and will present
the occured results
    # If the argument is "0" as "stop test button" the log file will written with
nothing [empty]
        #and same writing process on tablename

    # So when user end this testdrive the database which WAS in use, the channels
to use it again is gone.
        # results is still in database, but system cant use the same table without
human based performance
            # with this way trying to make sure that, the user does not use same
table with dofferent tests.


dbHost = '*****'
dbDes = '******'
dbUser = '*****'
dbPasswd = '*****'


onoff = str(sys.argv[1])


def getInterval():
    ltable = open("/home/pi/desalinator_sw/lib/database_process/temporary/la-
testTable.txt", "r")
    tableName_raw = ltable.readlines()
    retVal = ""
    for c in tableName_raw:
        retVal += c
    tableName = retVal

    try:

        db = my.connect(host=dbHost, user=dbUser, password=dbPasswd, data-
base=dbDes)
        c = db.cursor(buffered = True)

        cmd = ("SELECT time_interval from " + tableName)
        c.execute(cmd)
```

```
        result_raw = c.fetchall()[0]

        c.close()
        db.close()

        return int(result_raw[0])

    except Exception as e:
        sys.stderr.write(str(e))

def makeCron(addTime):

    user = CronTab(user='pi')
    task = user.new(command='python /home/pi/desalinator_sw/ui/testingProgram.py >
/home/pi/desalinator_sw/lib/testruns/temporary/data.log')
    task.minute.every(addTime)
    user.write()

def deleteCron(parameter):
    if parameter == 0:
        user = CronTab(user='pi')
        user.remove_all()
        user.write()

        emptyLog = open('/home/pi/desalinator_sw/lib/testruns/temporary/data.log',
"w")
        emptyLog.writelines(" ")
        emptyLog.close()

        removeLatestTable = open("/home/pi/desalinator_sw/lib/database_pro-
cess/temporary/latestTable.txt", "w")
        removeLatestTable.writelines(" ")
        removeLatestTable.close()

    else:
        user = CronTab(user='pi')
        user.remove_all()
        user.write()

        emptyLog = open('/home/pi/desalinator_sw/lib/testruns/temporary/data.log',
"w")
        emptyLog.writelines(" ")
        emptyLog.close()



on = {"cron": "on"}
off = {"cron": "off"}


if int(onoff) == 1:
    deleteCron(1)
    time = getInterval()
    makeCron(time)
    print (json.dumps(on))

else:
    deleteCron(0)
    print (json.dumps(off))
```

(ui/readValues.py)

```
import sys, time, json

# this program will be excecuted from user interface / values [get values button]
and print will contain all sensor values

# JSON printing is extra-important, without it user interface cannot read (and
present) results

# path to handlers

arduino = '/home/pi/desalinator_sw/ui/handlers/sensors/arduino'
temperature = '/home/pi/desalinator_sw/ui/handlers/sensors/temperature'
mbus = '/home/pi/desalinator_sw/ui/handlers/sensors/mbus/'
relay = '/home/pi/desalinator_sw/ui/handlers/sensors/component_status'


# getting data from arduino
sys.path.append(arduino)
import arduino
arduinoValues = arduino.getValues()

# getting data from 1-wire system
sys.path.append(temperature)
import temperature
temperatureValues = temperature.getValues()

#getting mbusdata from sharky
sys.path.append(mbus)
import mbus
mbusValues = mbus.getValues()

sys.path.append(relay)
import component_status
components = component_status.getValues()



    # inserting data in one object
sensorRes = arduinoValues.copy()
sensorRes.update(temperatureValues)
sensorRes.update(mbusValues)
sensorRes.update(components)



json_raw = json.dumps(sensorRes)


print json_raw
```

(ui/testingProgram.py)

```
import sys, json, time, serial
import mysql.connector as my


# this program will be more complex than others in time of writing.
    # this program will be executed by cron routine between the set interval (more
information, see mgnCron.py)
        # The execution ends when user will end the test session (more informa-
tion, see mgnCron.py)

    # functions of this program will be commented individually


#Database variables
dbHost = '*****'
dbDes = '*****'
dbDes2 = '******'
dbUser = '******'
dbPasswd = '******'


    # this function will call all sensor values and return it with dictionary ob-
ject
def getSensorValues():
        #paths to sensor reading handlers
    arduino = '/home/pi/desalinator_sw/ui/handlers/sensors/arduino'
    temperature = '/home/pi/desalinator_sw/ui/handlers/sensors/temperature'
    mbus = '/home/pi/desalinator_sw/ui/handlers/sensors/mbus'
    relay = '/home/pi/desalinator_sw/ui/handlers/sensors/component_status'

    # getting data from arduino
    sys.path.append(arduino)
    import arduino
    arduinoValues = arduino.getValues()

    # getting data from 1-wire system
    sys.path.append(temperature)
    import temperature
    temperatureValues = temperature.getValues()

    #getting mbusdata from sharky
    sys.path.append(mbus)
    import mbus
    mbusValues = mbus.getValues()

    sys.path.append(relay)
    import component_status
    components = component_status.getValues()



        # inserting data in one dictionary
    sensorRes = arduinoValues.copy()
    sensorRes.update(temperatureValues)
    sensorRes.update(mbusValues)
    sensorRes.update(components)

    return sensorRes
```

```
    # this function read the latest database tablename and will return it type of
string
def getLatestTableName():
    ltable = open("/home/pi/desalinator_sw/lib/database_process/temporary/la-
testTable.txt", "r")
    tableName_raw = ltable.readlines()
    retVal = ""
    for c in tableName_raw:
        retVal += c
    ltable.close()

    return retVal


    # This function will read defined attribute [desalination_test_attr] -database
and test result [desalination] -database
        # it will return the values in list object
def databaseColumns():
    attributes_columns_path = '/home/pi/desalinator_sw/lib/database_process/confi-
gurations/attribute_columns.txt'
    storage_columns_path = '/home/pi/desalinator_sw/lib/database_process/configu-
rations/storage_columns.txt'
    attributes = open(attributes_columns_path)
    attr_raw = attributes.readlines()
    attr = map(str.strip, attr_raw)

    storage = open(storage_columns_path)
    storage_raw = storage.readlines()
    stor = map(str.strip, storage_raw)

    attributes.close()
    storage.close()

    return attr, stor


        # This function calls to get sensor values, latest table name and defined
column names
            # Sensor values is gonna stored on test results [desalination] -data-
base
                # if storing process is executed successful function will return
boolean true
                    # else its returning false
def storeValuesOnDatabase():
    sensors = getSensorValues()
    tableName = getLatestTableName() + " ( "
    xxx, storage_raw = databaseColumns()
    storage = []
    for i in range(0, len(storage_raw)):
        if i == len(storage_raw)-1:
            storage.append(storage_raw[i])
        else:
            storage.append(storage_raw[i] + ", ")
```

```
    try:

        db = my.connect(host=dbHost, user=dbUser, password=dbPasswd, data-
base=dbDes)
        c = db.cursor(buffered = True)

        cmd = ("INSERT INTO " + tableName +

        storage[0]+
        storage[1]+
        storage[2]+
        storage[3]+
        storage[4]+
        storage[5]+
        storage[6]+
        storage[7]+
        storage[8]+
        storage[9]+

        storage[10]+
        storage[11]+
        storage[12]+
        storage[13]+
        storage[14]+
        storage[15]+
        storage[16]+
        storage[17]+
        storage[18]+
        storage[19]+

        storage[20]+
        storage[21]+
        storage[22]+
        storage[23]+
        storage[24]+
        storage[25]+
        storage[26]+
        storage[27]+
        storage[28]+
        storage[29]+

        storage[30]+
        storage[31]
        +
        """
        )
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
        %f, %f, %f, %f, %f, %f, %f,
        %s, %s,
        %f, %f, %f,
        '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s')"""
        %(
        sensors['temperature_dimense_27'],
        sensors['temperature_dimense_77'],
        sensors['temperature_dimense_127'],
        sensors['temperature_dimense_177'],
        sensors['temperature_dimense_227'],
        sensors['temperature_dimense_277'],
        sensors['temperature_dimense_327'],
```

```
            sensors['temperature_dimense_377'],
            sensors['temperature_dimense_427'],
            sensors['temperature_individual_salt_drain'],
            sensors['temperature_individual_dw'],
            sensors['temperature_individual_sw_input'],
            sensors['mbus_total_energy_kwh'],
            sensors['mbus_total_Volume_m3'],
            sensors['mbus_volume_flow_liters'],
            sensors['mbus_power_kw'],
            sensors['mbus_flow_temp_c'],
            sensors['mbus_return_temp_c'],
            sensors['mbus_current_energy_kwh'],
            sensors['arduino_top_level_sensor'],
            sensors['arduino_bottom_level_sensor'],
            sensors['arduino_outer_top_pressure'],
            sensors['arduino_inner_top_pressure'],
            sensors['arduino_bottom_pressure'],
            sensors['relay_sw_intake_valve'],
            sensors['relay_sw_out_lwr_valve'],
            sensors['relay_sw_out_upr_valve'],
            sensors['relay_intake_pressure_valve'],
            sensors['relay_vacuum_pump'],
            sensors['relay_circulation_pump'],
            sensors['relay_R7'],
            sensors['relay_R8'],
            ));

            c.execute(cmd)


            db.commit()
            c.close()
            db.close()

            return True

        except Exception as e:
            sys.stderr.write(str(e))
            return False


            # This function will read the database values and return the values in
dictionary object
def readDatabaseValues():
    xxx, storage_raw= databaseColumns()
    storage = []
    outputStorage = []
    storage.append('id, ')
    outputStorage.append('id')

    for i in range(0, len(storage_raw)):
        outputStorage.append(storage_raw[i])
        if i == len(storage_raw)-1:
            storage.append(storage_raw[i])

        else:
            storage.append(storage_raw[i] + ", ")
```

```
    tableName = getLatestTableName()

    try:

        db = my.connect(host=dbHost, user=dbUser, password=dbPasswd, data-
base=dbDes)
        c = db.cursor(buffered = True)

        cmd = ("SELECT " +
        storage[0]+
        storage[1]+
        storage[2]+
        storage[3]+
        storage[4]+
        storage[5]+
        storage[6]+
        storage[7]+
        storage[8]+
        storage[9]+
        storage[10]+
        storage[11]+
        storage[12]+
        storage[13]+
        storage[14]+
        storage[15]+
        storage[16]+
        storage[17]+
        storage[18]+
        storage[19]+
        storage[20]+
        storage[21]+
        storage[22]+
        storage[23]+
        storage[24]+
        storage[25]+
        storage[26]+
        storage[27]+
        storage[28]+
        storage[29]+
        storage[30]+
        storage[31]+
        storage[32]
        + " FROM " + tableName + " WHERE id =(SELECT max(id) from " +tableName +
");")
        c.execute(cmd)
        result_raw = c.fetchall()[0]

        c.close()
        db.close()




        bothValues = dict(zip(outputStorage, result_raw))
```

```
        return bothValues


    except Exception as e:
        sys.stderr.write(str(e))


        # this function will return the attributes values from database from indi-
vidual table
def returnAttributes():
    attributes_raw, xxx = databaseColumns()

    attributes = []
    outputAttributes = []
    attributes.append('id, ')
    outputAttributes.append('id')

    for i in range(0, len(attributes_raw)):
        outputAttributes.append(attributes_raw[i])
        if i == len(attributes_raw)-1:
            attributes.append(attributes_raw[i])

        else:
            attributes.append(attributes_raw[i] + ", ")



    tableName = getLatestTableName()

    try:

        db = my.connect(host=dbHost, user=dbUser, password=dbPasswd, data-
base=dbDes2)
        c = db.cursor(buffered = True)

        cmd = ("SELECT " +
        attributes[0]+
        attributes[1]+
        attributes[2]+
        attributes[3]

        + " FROM " + tableName + " WHERE id =(SELECT max(id) from " +tableName +
");")
        c.execute(cmd)
        result_raw = c.fetchall()[0]

        c.close()
        db.close()

        bothValues = dict(zip(outputAttributes, result_raw))
        del bothValues['id']

        return bothValues
```

```
    except Exception as e:
        sys.stderr.write(str(e))


        # this function will look specific pressure sensor value and managet cont-
rol for vacuum pump by limit values which been placed in UI
def setVacuum(values):


    stat1 = values.copy()
    hi = stat1['vacuum_hi']
    lo = stat1['vacuum_lo']
    value = stat1['pressure_outer_top']


        # Serial communication to relay
    serialCom = serial.Serial('/dev/relay',9600, timeout=3)
    vacuum1 = chr(0xFF) + chr (0x05) + chr (0x01)
    vacuum0 = chr(0xFF) + chr (0x05) + chr (0x00)

    if (value >= hi):
        serialCom.write(vacuum1)
        time.sleep(1)
        serialCom.close()


    if value <= lo:
        serialCom.write(vacuum0)
        time.sleep(1)
        serialCom.close()


    if value > hi and value > lo:
        returnValue = {"vacuum_command": "on"}

    if value < hi and value < lo:
        returnValue = {"vacuum_command": "off"}


    return returnValue




    # Function uses
if storeValuesOnDatabase():
    valuesInDatabase = readDatabaseValues()
    attributesInDatabase = returnAttributes()

    valuesToUi = valuesInDatabase.copy()
    valuesToUi.update(attributesInDatabase)
    stateOfVacuum = setVacuum(valuesToUi)
    valuesToUi.update(stateOfVacuum)
    print json.dumps(valuesToUi)
```

```
else:
    err = {"error": "error occured on testingProgram.py"}
    print err



    #print(readDatabaseValues())
```