VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Sami Valkosalo

Automaattinen kastelujärjestelmä pienkasveille

Tekniikka
2024

VAASAN AMMATTIKORKEAKOULU
Tietotekniikka

TIIVISTELMÄ

Tekijä	Valkosalo Sami
Opinnäytetyön nimi	Automaattinen kastelujärjestelmä pienkasveille
Vuosi	2024
Kieli	suomi
Sivumäärä	56 + 3 liitettä
Ohjaaja	Jani Ahvonen

Kasvien kasvattaminen kotiolosuhteissa voi osoittautua hankalaksi niiden vaativien kasvuympäristö vaatimuksien takia. Kasvi ei välttämättä pysty menestymään, mikäli se ei saa riittävästi valoa, vettä ja lämpöä. Näiden määreiden seuraaminen ja arviointi saattaa osoittautua haasteelliseksi ja pahimmassa tapauksessa johtaa kasvin kuolemaan. Kasvien kastelu tuo myös lisää työtä ja muistettavaa arkeen. Kastelu voi myös osoittautua mahdottomaksi, jos matkustelee usein.

Tämän opinnäytetyön tarkoituksena oli suunnitella ja toteuttaa järjestelmä, jolla voidaan automatisoida kasvien kastelu sekä kerätä kasvuympäristöstä hyödyllistä dataa kasvatuksen kehittämiseksi.

Järjestelmä rakennettiin käyttäen edullisia elektroniikkakomponentteja sekä internetistä ilmaiseksi saatavia ohjelmistoja. Järjestelmä on helposti laajennettavissa ja muokattavissa käyttövaatimuksien mukaan.

Avainsanat: esineiden internet, mikro-ohjaimet, ohjelmointi, anturit, kastelujärjestelmä

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietotekniikka

ABSTRACT

Author	Sami Valkosalo
Title	Automatic watering system for small plants
Year	2024
Language	Finnish
Pages	56 + 3 appendices
Name of Supervisor	Jani Ahvonen

Growing plants in home conditions can prove difficult due to their demanding growing environment requirements. A plant may not be able to thrive if it does not get enough light, water and heat. Determining and assessing these requirements can prove challenging and in the worst case lead to the death of the plant. Watering plants also adds extra work and memorizing to the everyday life. Watering may also prove impossible if the person travels a lot.

The purpose of this thesis was to design and implement a system to automate the watering of plants and to collect useful data on the growing environment to improve the growing process.

The system was build using low-cost electronic components and free software available on the internet. The system is easily expandable and customizable.

Keywords: Internet of things, microcontrollers, programming, sensors, irrigation system

SISÄLLYS

KUVA- JA TAULUKKOLUETTELO

LIITELUETTELO

1 JOHDANTO

Nykypäivänä kasveja voidaan kasvattaa monista erilaisista syistä. Kasveja voidaan kasvattaa esimerkiksi niistä saatavien syötävien osien takia, näyttävän ulkonäön takia tai harrastus- ja myynti tarkoitukseen. Kasvatukseen sisältyy kuitenkin paljon vastuuta ja työtä, sillä kasvit tarvitsevat säännöllisesti vettä, riittävästi auringon valoa sekä suotuisan kasvuympäristön. Suomessa on talvisin kylmää ja pimeää ja kesäisin saattaa olla pidempiäkin ajanjaksoja, kun ilma on erittäin kuivaa. Tällaisissa olosuhteissa saattaa eksoottisimmilla kasveilla olla hankaluuksia selvitä, ja olosuhteiden arvioiminen saattaa olla hankalaa.

Usein ongelmaksi saattaa muodostua kasvien kastelu matkustamisen aikana. Tällöin kasvit joutuisivat olemaan useita päiviä ilman vettä ja luultavasti kuihtuisivat kuoliaaksi. Itselleni tämä on ollut suurimpana esteenä kasvien hankinnalle, sillä toisinaan saatan olla viikkoja pois kotoa toisella paikkakunnalla.

Tämän opinnäytetyön tarkoituksena oli suunnitella ja toteuttaa järjestelmä, joka on kykenevä kaupallisten ratkaisujen tavoin automaattiseen kasteluun, mutta myös keräämään tietoa kasvuympäristön olosuhteista. Opinnäytetyössä perehdytään kastelun automatisointiin mullan kosteuden perusteella sekä kuinka kasvuolosuhteita voidaan arvioida sensoreilla ja tallentaa tietokantaan.

2 JÄRJESTELMÄN SUUNNITTELU

2.1 Yleiset ominaisuudet

Järjestelmän tulee viedä fyysisesti vähän tilaa ja olla helposti sijoitettavissa lähelle kasvia, jotta kasvuympäristöstä tehtävät mittaukset saadaan mahdollisimman tarkasti. Järjestelmän virrankulutus tulee olla mahdollisimman pieni ja toimia USB-adapterilla verkkovirrasta.

Järjestelmän tulee olla kykenevä itsenäiseen toimintaan mahdollisimman vähäisellä ylläpidolla ja pystyä palautumaan mahdollisissa virhetilanteissa kuten sähkö- ja verkkokatkoksissa. Käyttäjän tulee huolehtia vesiastian täyttämisestä vähintään kuukauden välein sekä vesipumpun paristojen vaihdosta 1-2 vuoden välein.

2.2 Kasvin kastelu

Kastelu tapahtuu pienellä vesiastiaan upotettavalla vesipumpulla, josta lähtee PVC-muoviletku kasville. Vesipumppu saa tarvittavan virran erillisestä virtalähteestä, joka muodostuu sarjaan kytketyistä litiumparistoista. Virtalähteen ja vesipumpun välillä on rele, jolla voidaan ohjata vesipumppua päälle ja pois. Relettä ohjataan mikro-ohjaimella, johon ohjelmoidaan automaatio hallinnoimaan kastelua.

Kasteluautomaatio ohjelmoidaan seuraamaan mullan kosteuden keskiarvoa, jonka laskiessa liian matalalle järjestelmä kastelee kasvin. Keskiarvon laskeminen tulee perustua vähintään viimeisen kymmenen minuutin ajalta kerätyistä mittauksista, jotta mittausnäyte on riittävän suuri.

2.3 Kasvuympäristön mittaukset

Kasvin lähiympäristöstä mitataan ilmankosteus ja lämpötila sekä kasviin kohdistuvan valon- ja mullankosteuden määrä. Mittaukset tapahtuvat niihin erikoistuneilla sensoreilla määritetyin aikavälein. Mittaustulokset luetaan mikro-ohjaimella ja tallennetaan langattomalla yhteydellä lähiverkossa olevaan tietokantaan.

Valoisuutta tulee pystyä mittaamaan vähintään 1–30 000 lx välillä ja ±0.2 lx tarkkuudella. Lämpötilaa tulee pystyä mittaamaan 0–50 °C välillä ja ±2 °C tarkkuudella. Ilmankosteutta tulee pystyä mittaamaan 0-100 % välillä ja vähintään ±5 % tarkkuudella. Mullan kosteus mitataan analogisena jännitearvona mikro-ohjaimen AD-muuntimelle, jolla se muunnetaan digitaaliseksi arvoksi.

2.4 Mikro-ohjainalusta

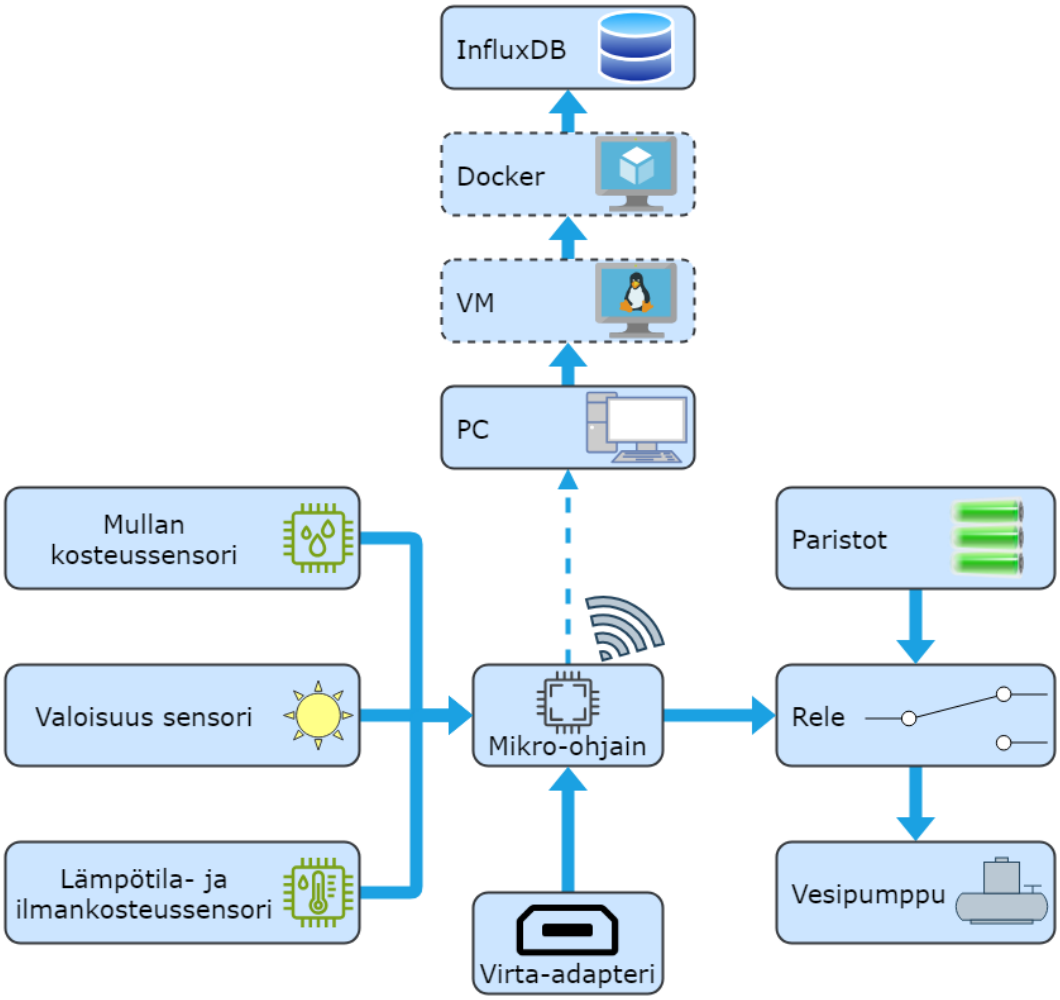
Mikro-ohjainalustan tulee sisältää riittävästi GPIO-pinnejä, jotta siihen voidaan kytkeä 3 kpl sensoreita sekä rele. Mikro-ohjaimessa tulee olla riittävän tehokas prosessori ja sisältää riittävästi Flash-muistia ohjelmistoa varten. Mikro-ohjaimen ominaisuuksien tulee sisältää AD-muunnin, I^2C-väylä ja Wi-Fi-mahdollisuus.

Laite tulee sijoittaa suljettuun koteloon, jotta se on suojassa mahdollisilta vesiroiskeilta. Alustan tulee olla fyysiseltä kooltaan pieni ja toimia USB-adapterilla verkkovirrasta.

2.5 Tietokanta kommunikaatio

Lähiverkkoon asennetaan Linux-käyttöjärjestelmään pohjautuva virtuaalikone, jossa isännöidään Docker-ajoympäristöä. Dockerilla luodaan container, johon asennetaan InfluxDB-tietokantajärjestelmä mittaustulosten tallennusta varten. Mikro-ohjain lähettää mittaustulokset tietokantaan langattomasti käyttäen HTTP-protokollaa.

Järjestelmän tulee pystyä muodostamaan Wi-Fi-yhteys automaattisesti käynnistyksen yhteydessä, mikäli määritetty verkkoyhteys on käytettävissä. Verkkokatkosten yhteydessä järjestelmän tulee pystyä uudelleen muodostamaan Wi-Fi-yhteys automaattisesti vähintään kymmenen minuutin sisällä verkon palautuksesta. Järjestelmän tulee pystyä toimimaan normaalisti ilman Wi-Fi-yhteyttä, mutta tällöin mittaustuloksia saatetaan menettää. Järjestelmä on esitetty lohkokaaviona kuvassa 1.



Kuva 1. Lohkokaavio järjestelmästä.

3 JÄRJESTELMÄN ELEKTRONIIKKA

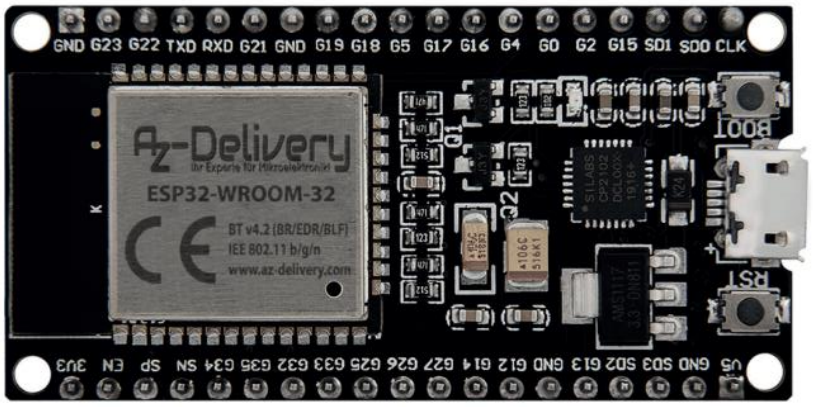
3.1 Mikro-ohjain

ESP32 on Espressif Systemsin valmistama mikro-ohjain, joka julkaistiin syyskuussa 2016 ESP8266-mikro-ohjaimen seuraajaksi. Vanhaan mikro-ohjaimeen verrattuna ESP32 tuo enemmän ohjelmoitavia GPIO-pinnejä, enemmän muistia, nopeamman Wi-Fin, sekä Bluetooth-tuen.

Järjestelmän ohjaukseen valittiin Az-Deliveryn valmistama ESP32-kehitysalusta sen edullisuuden, pienen koon, vähäisen virrankäytön ja ESP-WROOM-32-mikro-ohjaimen vuoksi. Kyseinen mikro-ohjain sisältää Tensilica Xtensa 32-bit LX6 -mikroprosessorin, jossa on kaksi ydintä ja joiden maksimi kellotaajuus on 240 MHz. Käytettävä mikro-ohjainkehitysalusta on nähtävissä kuvassa 2. /1/

ESP-WROOM-32-mikro-ohjaimen ominaisuuksia ovat:

- 5V käyttöjännite
- Virrankäyttö keskimäärin 80mA
- 34 kpl ohjelmoitavia GPIO-pinnejä
- 4MB Flash-muistia ja 512KB SRAM-muistia
- Sisäänrakennettu Wi-Fi



Kuva 2. Järjestelmässä käytettävä mikro-ohjainalusta.

3.2 Kastelujärjestelmä

3.2.1 Vesipumppu

Järjestelmässä käytettävä vesipumppu valittiin sen pienen käyttöjännitteen ja virrankulutuksen vuoksi. Vesipumpun valmistaja on ilmoittanut laitteen käyttöjännitteeksi 3-5V ja sähkönkulutukseksi 100-200mA. Koska mikro-ohjain pystyy antamaan korkeintaan 15mA GPIO-pinnistä, ei pumppua voida kytkeä suoraan mikro-ohjaimelle tai muuten se vaurioituisi. Pienen käyttöjännitteen ansiosta voidaan riittävä virtalähde muodostaa paristojen sarjaan kytkennällä.

Vesipumpun valinnassa päädyttiin upotettavaan malliin, koska pumpun ollessa vesiastiassa säästetään tilaa ja astiassa oleva vesi vaimentaa pumpusta aiheutuvaa ääntä. Lisäksi upotettavat vesipumput ovat pääsääntöisesti energiatehokkaampia, koska ne puskevat vettä suoraan ulos astiasta, jolloin painovoiman aiheuttama vastus on pienempi. /2/

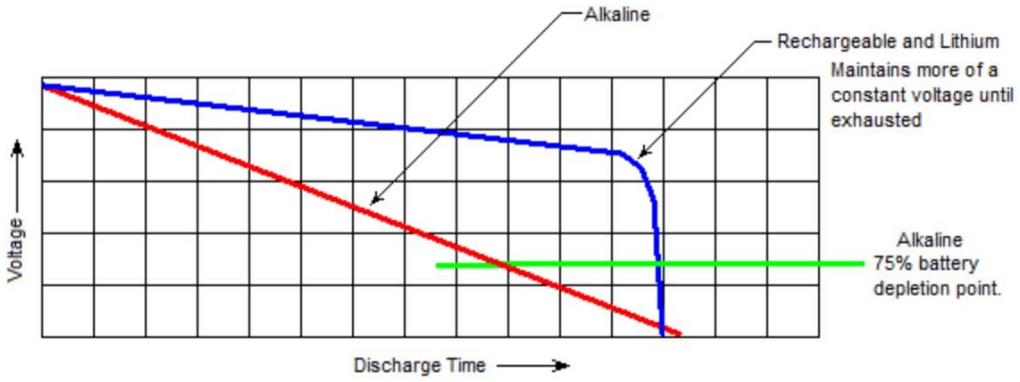
3.2.2 Virtalähde

Kastelujärjestelmän virtalähde päädyttiin muodostamaan kolmesta AA-Litiumparistosta. Yhden pariston napajännite on 1.5V, jolloin sarjaan kytkemällä kolme paristoa saadaan yhteisjännitteeksi 4.5V. Tämä on riittävä vesipumpun virtalähteeksi, jonka käyttöjännite on 3-5V.

Alkaliparistoille on tyypillistä, että niissä oleva napajännite pienenee käytön mukaan. Tämä aiheuttaisi ongelmia, koska vesipumpun tehokkuus pienenisi ajan myötä ja lopettaisi jossain kohtaa toimimasta, vaikka paristoissa olisi vielä sähkövarausta jäljellä.

Litiumparistot ylläpitävät jännitetason tasaisena lähes koko käyttöaikansa, jolloin vesipumppu toimii odotetusti koko ajan. Tästä syystä litiumparistot ovat myös paljon energiatehokkaampia kuin alkaliparistot. Litiumparistoissa on myös suurempi kapasiteetti, jolloin niitä tarvitsee vaihtaa harvemmin uusiin.

Alkali- ja litiumparistojen jännitteen putoamista käyttöajan myötä on havainnollistettu kuvassa 3. /3/



Kuva 3. Alkali- ja litiumparistojen jännite käyttöajan myötä. /3/

3.2.3 Rele

Kastelujärjestelmän virtapiirin katkaisuun päädyttiin valitsemaan Az-Deliveryn valmistama KF-301-relemoduuli. Relemoduulissa on 2 kpl ledejä, jotka näyttävät, milloin releen käämissä on virta ja missä tilassa kosketin on. Moduulissa on myös flyback-diodi, joka estää kelan sammuttamisesta aiheutuvaa virtapiikkiä vaurioittamasta mikro-ohjainta. Kelan ohjaus tapahtuu S8550 PNP-transistorilla.

Relemoduulin datalehdessä on käyttöjännitteeksi merkattu 5V, mutta kytkentäesimerkeistä käy ilmi, että relemoduuli toimii myös 3.3V käyttöjännitteellä. Releen koskettimet pystyvät käsittelemään maksimissaan 30V/5A tasajännitettä tai 50V/5A vaihtojännitettä. Koska kastelujärjestelmän virtalähde on paristoilla muodostettu 4.5V tasajännite ja vesipumppu kuluttaa maksimissaan 200mA, on rele riittävä virtapiirin käsittelyyn.

Relemoduulissa on liitännät ohjaussignaalille (IN), käyttöjännitteelle (VCC) ja maalle (GND). Relemoduuli toimii low level trigger -periaatteella, jolloin ohjaussignaalin ollessa low-tilassa rele aktivoituu ja yhdistää normally opened -koskettimen. Kun ohjaussignaali on high-tilassa, on releen kosketin vapautunut ja normally closed -kosketin yhdistetty. Releen pinnikaavio on kuvassa 4. /4/

Kuva 4. Relemoduulin pinnikaavio. /4/

3.3 Sensorit

3.3.1 Mullan kosteussensori

Mullan kosteutta on mahdollista mitata resistiivisellä tai kapasitiivisella sensorilla. Resistiivisessä sensorissa on kaksi piikkiä, jotka mittaavat niiden välillä olevaa resistanssia. Jos piikkien välillä oleva multa on kosteaa, on piikkien välillä vähän resistanssia, koska vesi on hyvä johde sähkölle. Jos taas piikkien välillä oleva multa on kuivaa, on resistanssi suurempi. Resistiivisessä sensorissa on kuitenkin ongelmana niiden lyhyt elinikä: koska piikeissä kulkee sähkövirta ja niissä oleva kupari on suoraan kosketuksessa mullan kanssa, aiheuttaa se piikeissä sähkökemiallista syöpymistä. Näin ollen sensorista tulee hyvin nopeasti käyttökelvoton. /5/

Kapasitiivisessa sensorissa kaksi johdinta muodostaa kondensaattorin, jonka kapasitanssi muuttuu ympärillä olevan mullan kosteuden perusteella. Kapasitanssi tarkoittaa johtimien kykyä varastoida sähkövarausta. Tavallisen levykondensaattorin kapasitanssi voidaan esittää yhtälöllä:

$$C = \frac{\epsilon A}{\delta} \quad (1)$$

Kapasitanssi muodostuu levyjen välillä olevan dielektrisen aineen suhteellisesta permittiivisyydestä ϵ, levyjen pinta-alasta A ja levyjen välisestä etäisyydestä δ. Yksinkertaistettuna sensori toimii levykondensaattorin tavoin, jolloin levyjen pinta-ala ja etäisyys ovat vakioita, mutta väliaineen eli mullan permittiivisyys muuttuu siinä olevan veden määrän mukaan. Sensorin johtimien ollessa vierekkäin eikä vastakkain, kuten levykondensaattorissa, muodostuisi lopullinen yhtälö paljon monimutkaisempana laskukaavana, kun sensorin muotoilu otetaan huomioon. Tämä ylittäisi opinnäytetyön laajuuden ja on tästä syystä jätetty esittämättä.

Sensorissa on TL555C ajastin, joka syöttää kanttiaaltoa vastuksen läpi sensorin mittausjohtimille. Kanttiaallon jännite pienenee johtimien kapasitanssi mukaan. Sen jälkeen se tasasuunnataan tasavirraksi, minkä perusteella voidaan arvioida mullan kosteutta. /6/

Käytettäväksi sensoriksi päädyttiin valitsemaan kapasitiivinen sensori, koska siinä ei ole paljasta kuparia kosketuksessa mullan kanssa. Tällöin sähkökemiallista syöpymistä ei tapahdu, jolloin käyttöikä ja mittaus tarkkuus on parempi. Kuva 5 on verkkokaupan sivuilta, josta käytettävä kapasitiivinen sensori on tilattu.

Kuva 5. Kapasitiivinen mullan kosteussensori.

Sensorissa itsessään on sisäänrakennettu jännitteensäädin, jolloin sitä voidaan operoida 3.3–5.5V DC -käyttöjännitteellä. Mittaustulos saadaan ulostulojännitteenä, joka vaihtelee välillä 0-3V DC. Kytkemällä sensori mikro-ohjaimen AD-muuntimeen saadaan mittaustulos muunnettua digitaaliseen muotoon.

3.3.2 Lämpötila- ja ilmankosteussensori

Ympäristön lämpötilan ja ilmankosteuden mittaukseen valittiin DHT20-sensorimoduuli. DHT20-sensori tuo laajemman mittausalueen ja tarkemmat mittaustulokset DHT11-sensoriin verrattuna. Lisäksi se on edullisempi kuin DHT22. Taulukossa 1 on lueteltu DHT11-, DHT20- ja DHT22-sensoreiden ominaisuuksia. /7;8;9/

Taulukko 1. Eri DHTxx-sensoreiden ominaisuuksia. /7;8;9/

	DHT11	DHT20	DHT22
Käyttöjännite	3.3–5.5V DC	2.2–5.5V DC	3.3–5.5V DC
Virrankulutus	Max: 1.5mA	Max: 0.98mA	Max: 1.5mA
Lämpötila mittausalue	0–50°C	-40–80°C	-40–80°C
Lämpötila tarkkuus	± 2°C	± 0.5°C	± 0.5°C
Lämpötila resoluutio	1°C	0.1°C	0.1°C
Ilmankosteus mittausalue	20~90 %	0~100 %	0~100 %
Ilmankosteus tarkkuus	± 5 %	± 3 %	± 2 %
Ilmankosteus resoluutio	1 %	0.024 %	0.1 %
Kommunikointi	1-wire	I^2C	1-wire

Suurimpana erona sensoreiden välillä on, että DHT20 käyttää 1-wire protokollan sijaan I²C-protokollaa kommunikoimiseen. Tällöin sensorin mittaustuloksen lukemiseen tarvitaan mikro-ohjaimelta VCC- ja GND-pinnien lisäksi 2 kpl GPIO-pinnejä, jotka on konfiguroitu SDA- ja SCL-pinneiksi. /10/

DHT20-sensori koostuu kapasitiivisesta kosteusanturista, lämpötila-anturista sekä mikropiiristä. Kapasitiivinen kosteussensori mittaa ilman suhteellista kosteutta ja toimii samalla tavalla kuin aiemmin esitetty kapasitiivinen mullan kosteussensori. Anturi absorboi ilmasta kosteutta, jolloin väliaineen kapasitanssi muuttuu. Lämpötila-anturi mittaa ilman lämpötilaa, joka luetaan ja prosessoidaan mikropiirillä. Kun ilmankosteus ja lämpötila on mitattu, prosessoi mikropiiri ne signaaliksi ja lähettää SDA-pinnin kautta mikro-ohjaimelle, jossa ne voidaan ohjelmallisesti muuntaa luettavaan muotoon käyttäen kaavoja 2 ja 3. /8/

$$RH[\%] = \left(\frac{S_{RH}}{2^{20}}\right) * 100\% \tag{2}$$

Kaavalla 2 voidaan signaalista S_{RH} laskea suhteellinen kosteus prosentteina.

$$T[°C] = \left(\frac{S_T}{2^{20}}\right) * 200 - 50 \tag{3}$$

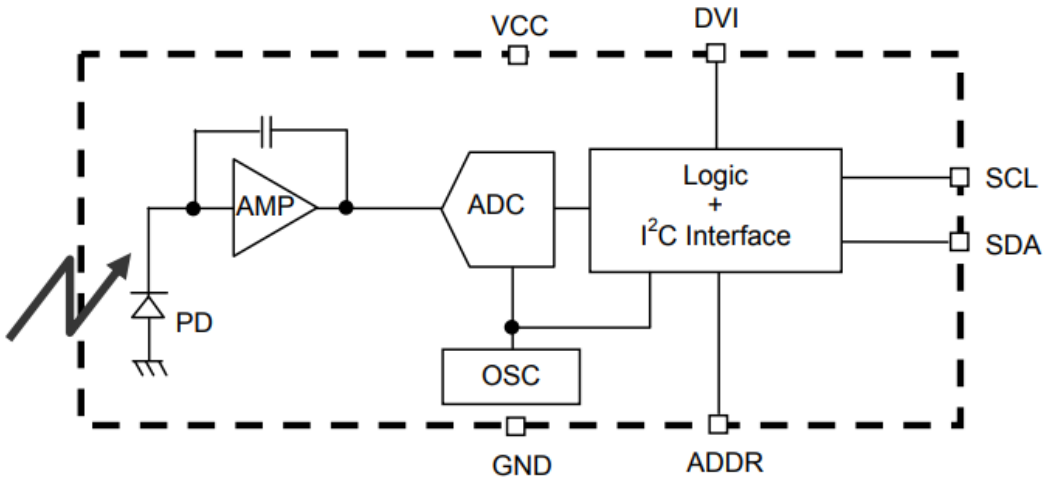
Kaavalla 3 voidaan signaalista S_T laskea lämpötila, jolloin tulos on celsiuksina.

3.3.3 Valoisuussensori

Ympäristön valoisuutta mitataan GY-302 moduulilla, joka perustuu BH1750-valoanturiin. Valoanturi tarvitsee toimiakseen 3–5 V käyttöjännitteen ja käyttää virtaa maksimissaan 190 µA. Anturi mittaa valoisuuden määrän 16bit:n resoluutiolla, jolloin valoisuuttaa voidaan mitata 1–65535 luxin välillä. Anturissa on sisäänrakennettu AD-muunnin, jolloin mittaus saadaan suoraan digitaalisena arvona. Mittaus lähetetään I²C-väylän kautta mikro-ohjaimelle. /11/

BH1750-sensorissa on valodiodi, joka muuntaa siihen kohdistuneen valon sähkösignaaliksi suhteessa valon määrään. Sähkösignaali muunnetaan jännitteeksi

operaatiovahvistimella, joka taas voidaan muuntaa AD-muuntimella digitaaliseen muotoon. Sensorissa itsessään on 16bit AD-muunnin, jolloin mittaus saadaan suoraan luxeina I²C-väylän kautta. Kuvassa 6 on esitetty BH1750-sensorin lohkokaavio, jossa PD on valodiodi ja AMP on operaatiovahvistin. /12/



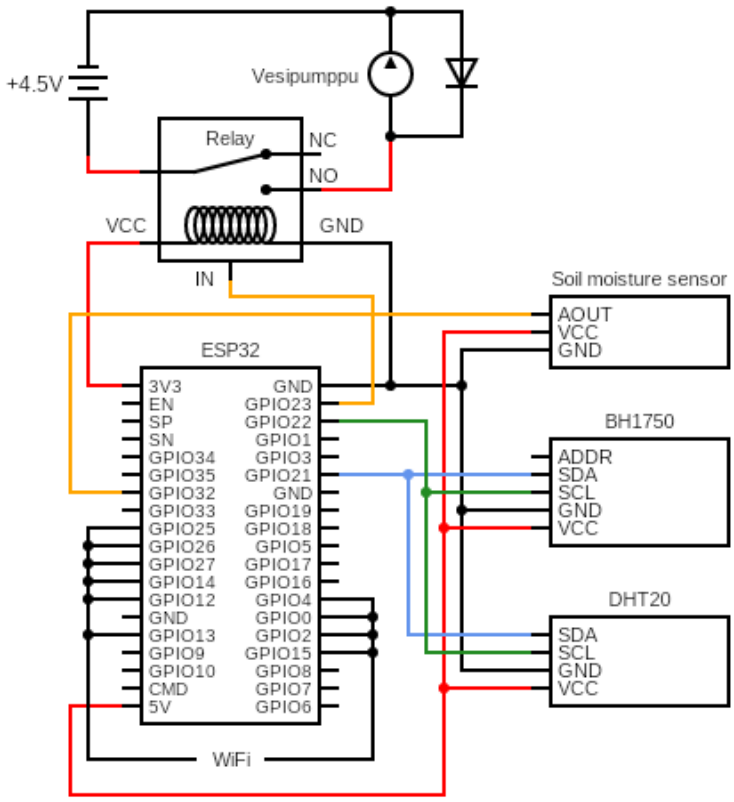
Kuva 6. BH1750-valoisuussensorin lohkokaavio. /12/

4 KYTKENTÖJEN TOTEUTUS

Järjestelmän toteutukseen tarvittavat kytkennät toteutettiin piirilevyllä ja hyppylangoilla. Mikro-ohjainalusta saa virran mikro-USB:n kautta ja tarvitsee virtalähteeksi 5VDC ja vähintään 500mA, jolloin adapterina voidaan käyttää puhelinlaturia.

Järjestelmässä käytetään ESP32 mikro-ohjaimen Wi-Fi-ominaisuutta. Wi-Fi käyttää ADC2-pinnejä, jolloin niitä ei ole mahdollista käyttää muuhun tarkoitukseen. /13/

ADC2-pinnejä ovat GPIO25, GPIO26, GPIO27, GPIO14, GPIO12, GPIO13, GPIO4, GPIO0, GPIO2 ja GPIO15. Tämä on esitetty kuvassa 7.



Kuva 7. Järjestelmän kytkentäkaavio.

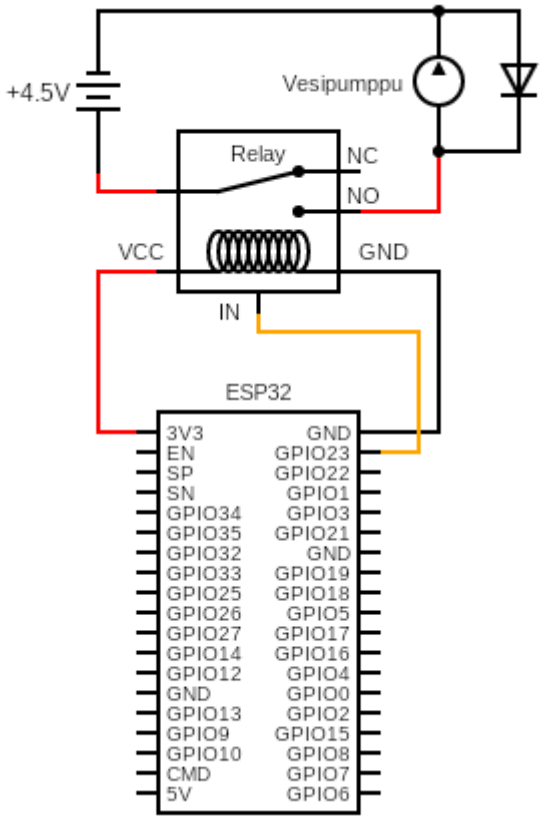
4.1 Kastelujärjestelmän kytkentä

Paristokotelon ja vesipumpun toinen johtimista on kytketty yhteen ja toinen johdin menee releen kautta. Vesipumpun johdin on kytketty releen sulkeutuvaan koskettimeen (NO, Normally Open). Vesipumpun rinnalle on kytketty diodi, jonka anodi on kytketty positiiviseen ja katodi negatiiviseen johtimeen. Diodi toimii flyback-diodina, jolloin sen tarkoitus on estää vesipumpun sammuessa aiheutuva jännitepiikki. /14/

Releen pienjännite pinnit on kytketty mikro-ohjaimelle seuraavasti:

- VCC → 3V3
- IN → GPIO23
- GND → GND

Rele saa +3.3V käyttöjännitteen 3V3-pinnistä ja releen tilaa ohjataan GPIO23-pinnistä. Kun rele saa low-signaalin mikro-ohjaimelta, sulkeutuu rele ja vesipumppu menee päälle. Rele vapautuu high-signaalilla, jolloin vesipumppu sammuu. Kastelujärjestelmän kytkentäkaavio on nähtävissä kuvassa 8.



Kuva 8. Kastelujärjestelmän kytkentäkaavio.

4.2 Sensorien kytkentä

Sensorit toimivat +5V käyttöjännitteellä, joten VCC-pinnit on kytketty mikro-ohjainalustan 5V-pinniin. GND-pinnit on yhdistetty alustan GND-pinniin.

Mullan kosteussensori antaa mittaustuloksen jännitteenä suhteessa mullan kosteuteen. Jotta mittaustulos voidaan lukea, täytyy AOUT-pinni kytkeä mikro-ohjaimen Analog to Digital Converter eli ADC-pinniin. Sopivia pinnejä ovat GPIO32–35, joista GPIO32 valikoitui käyttöön.

BH1750- ja DHT20-sensorit käyttävät I²C-protokollaa mittaustulosten kommunikointiin. Kun käytössä on useampi I²C-protokollaa käyttävä laite, on niiden kytkemiseen kaksi tapaa.

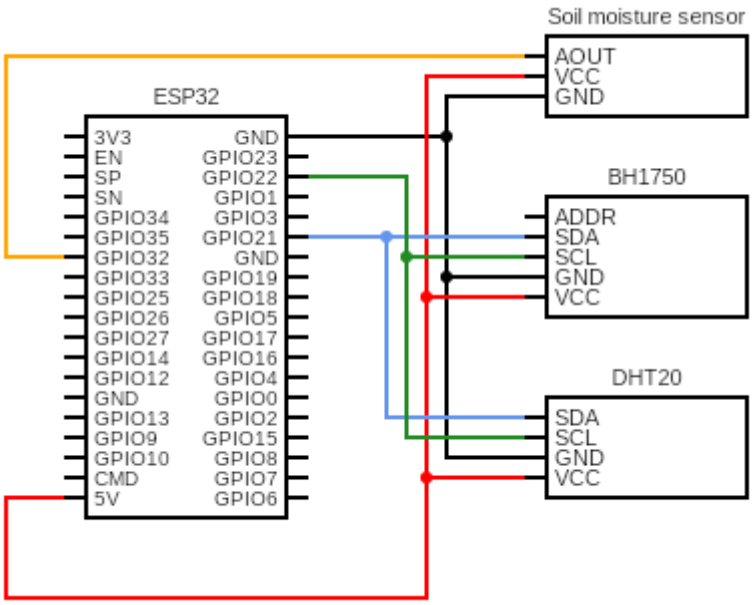
Ensimmäinen tapa on konfiguroida jokaiselle laitteelle oma I²C-väylä. Tämä tarkoittaisi sitä, että molemmilla laitteilla olisi omat GPIO-pinnit konfiguroituna, joten ne tarvitsisivat neljä pinniä mikro-ohjaimelta.

Toinen tapa vaatii, että laitteilla on eri laiteosoitteet kommunikointia varten. Tällöin sensoreiden SCL- ja SDA-pinnit voidaan kytkeä samoihin GPIO-pinneihin, jolloin pinnejä tarvitaan vain 2 kpl. /15/

Koska sensoreilla on eri laiteosoitteet, voidaan ne kytkeä samoihin GPIO-pinneihin, jotka on valittu seuraavasti:

- SDA → GPIO21
- SCL → GPIO22

Sensoreiden kytkentäkaavio on nähtävissä kuvassa 9.



Kuva 9. Sensorien kytkentäkaavio.

5 TIETOKANTA

5.1 Tietokanta ympäristö

Mittaustuloksien tallennukseen valittiin InfluxDatan kehittämä InfluxDB-tietokanta, joka on suunnattu aikasarjatietojen käsittelyyn. InfluxDB on hyvä vaihtoehto silloin, kun dataa on paljon ja aikaleimalla on suuri merkitys.

InfluxDB päätettiin asentaa Docker-ajoympäristöön, jota isännöidään virtuaalikoneella. Virtuaalikoneen käyttöjärjestelmäksi valittiin Ubuntu Server, joka on Canonicalin kehittämä Linux-jakelu. Virtuaalikoneen ja Dockerin käyttöön päädyttiin, koska ne mahdollistavat helpon siirrettävyyden ja eristetyn ympäristön.

5.2 InfluxDB:n asennus

Kirjoitus hetkellä uusin InfluxDB-versio dockerille on 2.7.4, joka saadaan haettua komennolla:

```
sudo docker pull influxdb:2.7.4
```

Tämän jälkeen luodaan kansio, johon tietokannan asetukset ja data tallennetaan, jotta ne säilyvät dockerin ulkopuolella. Tämä tapahtuu komennolla:

```
sudo mkdir influxdb-data
```

Tämän jälkeen luodaan docker container InfluxDB:lle komennolla:

```
sudo docker run -d \
        --name influxdb \
        -p 8086:8086 \
        --volume $PWD/influxdb_data:/var/lib/influxdb2 \
        influxdb:2.7.4
```

Parametrien selitykset ovat seuraavat:

- -d: Irrottaa containerin terminaalista ja vapauttaa sen muuhun käyttöön. Ilman tätä parametria terminaalin sammuttua sammuu myös containeri.
- --name: Annetaan containerille nimi.
- -p: Määritetään portti, jonka kautta containeriin voidaan kommunikoida. Ensimmäinen portti tarkoittaa isäntäkoneen porttia ja jälkimmäinen portti containerin porttia.
- --volume: Määritetään asema, johon InfluxDB:n asetukset ja data tallennetaan isäntäkoneella. Ensimmäinen osio on isäntäkoneen polku ja toinen osio on containerin sisällä oleva polku.
- Influxdb:2.7.4: Image, jota containerilla käytetään.

Kun container on luotu, voidaan sen tila tarkistaa komennolla:

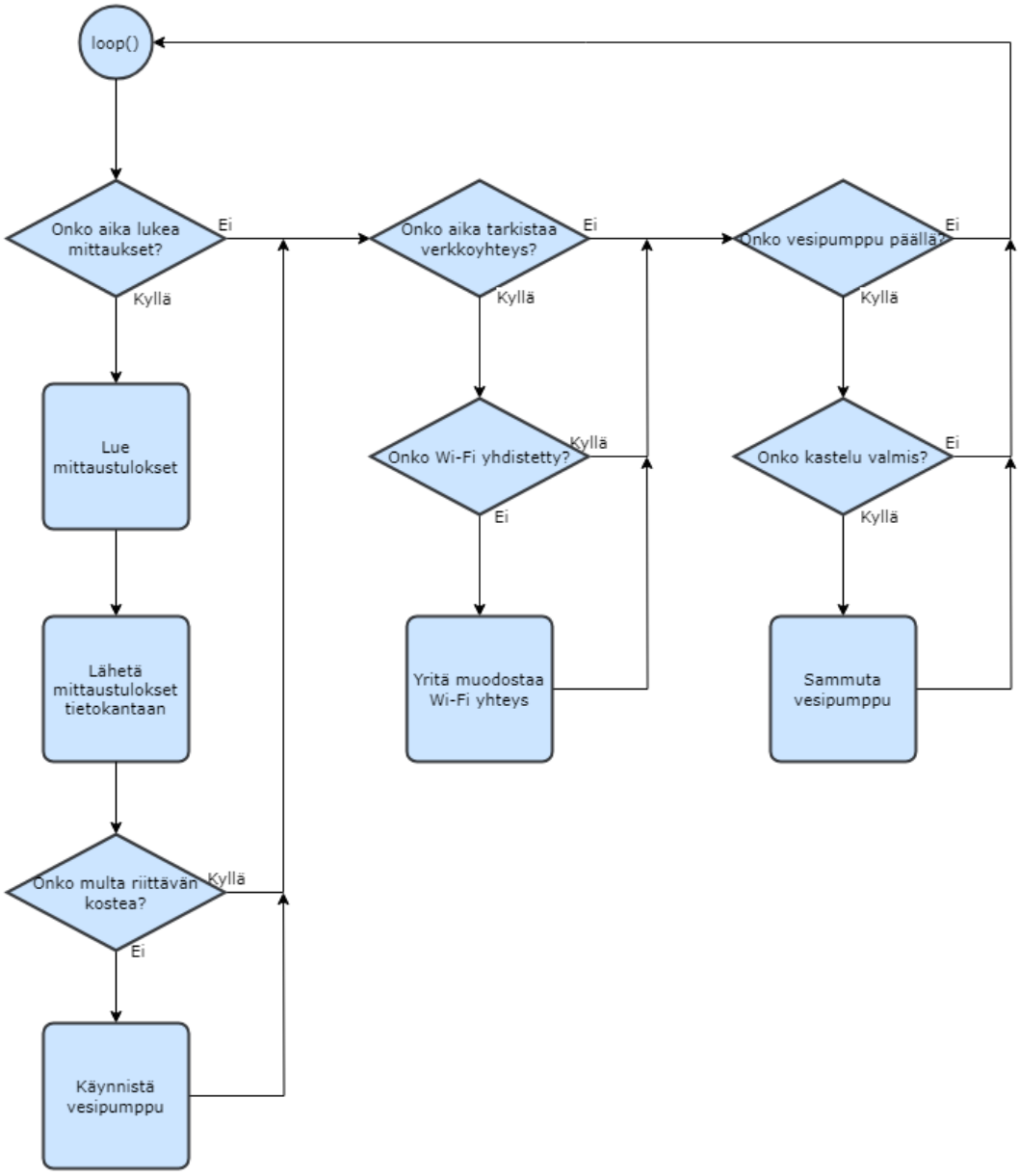
```
sudo docker ps -a
```

Kun InfluxDB-container on käynnissä, voidaan ottaa yhteys InfluxDB-istuntoon kirjoittamalla selaimeen isäntäkoneen IP-osoite ja määritetty portti. Avatessa InfluxDB:n ensimmäisen kerran alkaa esiasennus. Asennuksessa täytyy määrittää käyttäjänimi, salasana, organisaatio ja bucket. Bucket on se, johon data tallennetaan. Tämä voidaan nimetä käyttötarkoituksen mukaan, kuten kasvit tai sensorit.

Tämän jälkeen InfluxDB on valmis vastaanottamaan dataa.

6 OHJELMISTON SUUNNITTELU

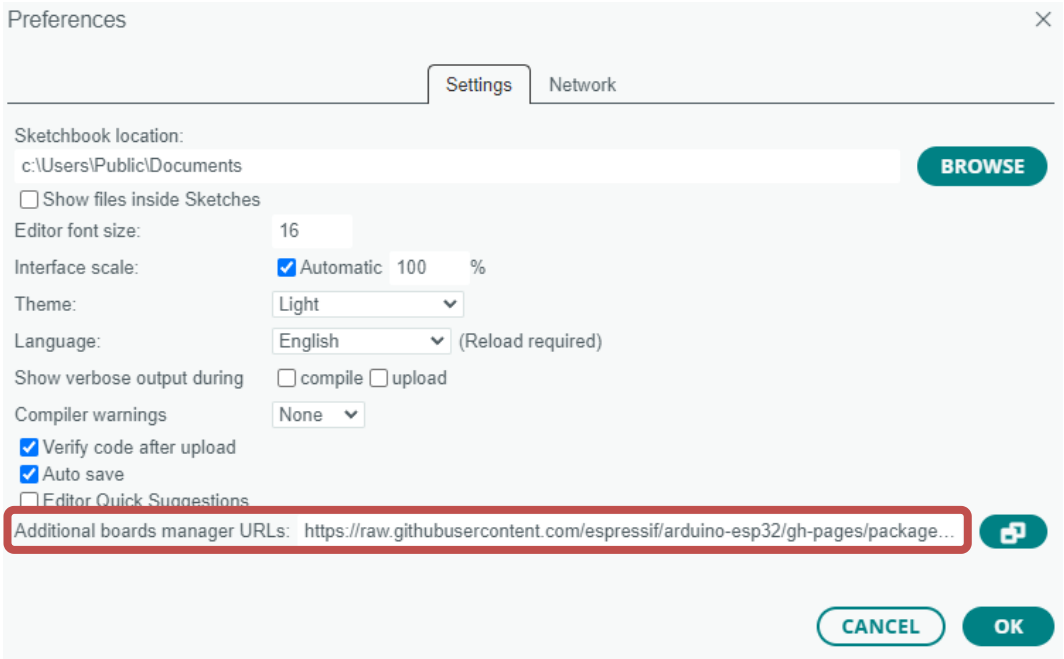
Mikro-ohjaimen ohjelmointi toteutettiin Arduino IDE -ohjelmalla ja sen omalla C++-pohjaisella koodikielellä. Arduino IDE:stä käytettiin uusinta versiota, joka oli kirjoitushetkellä 2.2.1. Kuvassa 10 on esitetty ohjelman toiminta vuokaaviolla.



Kuva 10. Vuokaavio ohjelman toiminnasta.

6.1 ESP32-tuen asennus

Ohjelmointia varten täytyy ensin asentaa ESP32-tuki, joka on saatavissa Espressif:n GitHubista (https://github.com/espressif/arduino-esp32). Asennus tapahtuu lisäämällä ulkopuolinen latausosoite Arduino IDE:n asetuksiin File → Preferences → Additional boards manager URLs-kenttään. Kuvassa 11 on punaisella merkattu Preferences-valikko, johon osoite pitää lisätä.

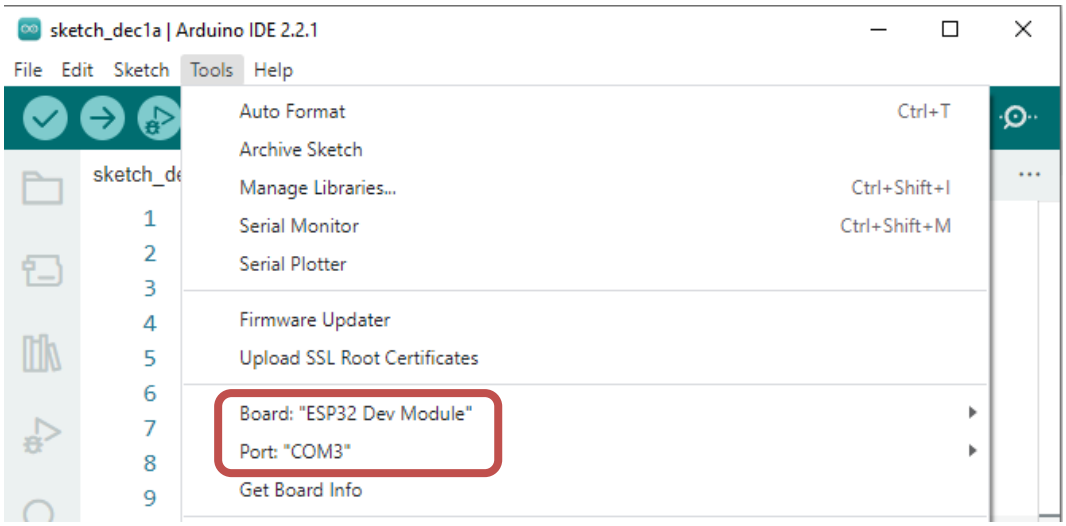


Kuva 11. ESP32-tuen latausosoitteen lisääminen Arduino IDE:hen.

Tämän jälkeen voidaan asentaa ESP32-tuki menemällä Arduino IDE:ssä valikkoon Tools → Board → Boards Manager ja hakemalla ESP32, jolloin pitäisi näkyville tulla esp32 by Espressif. Asennus tapahtuu Install-painikkeesta. Tämän opinnäytetyön toteutukseen käytettiin versiota 2.0.14, joka oli uusin versio kirjoitushetkellä.

Kun ESP32-tuki on asennettu, voidaan se valita valikosta Tools → Board → esp32, josta löytyy useita eri ESP32-kehitysalustavaihtoehtoja. Käytössä olevan kehitysalustan ohjelmointiin valitaan ESP32 Dev Module. Tämän jälkeen yhdistetään ESP32-kehitysalusta mikro-USB:llä tietokoneeseen ja valitaan oikea COM-portti

valikosta Tools → Port. Kuvassa 12 on nähtävissä käytössä olevan ESP32:n ohjelmointiin tarvittava alusta ja porttivalinta.



Kuva 12. Alustan ja portin valinta.

6.2 Ohjelmakirjastojen asennus

Opinnäytetyössä hyödynnettiin kolmannen osapuolen ohjelmakirjastoja sensoreiden lukemiseen ja konfigurointiin sekä kommunikointiin InfluxDB:n kanssa. Ohjelmakirjastoja voi hakea Arduino IDE:ssä valikosta Tools → Manage Libraries. Asennus tapahtuu Install-painikkeella.

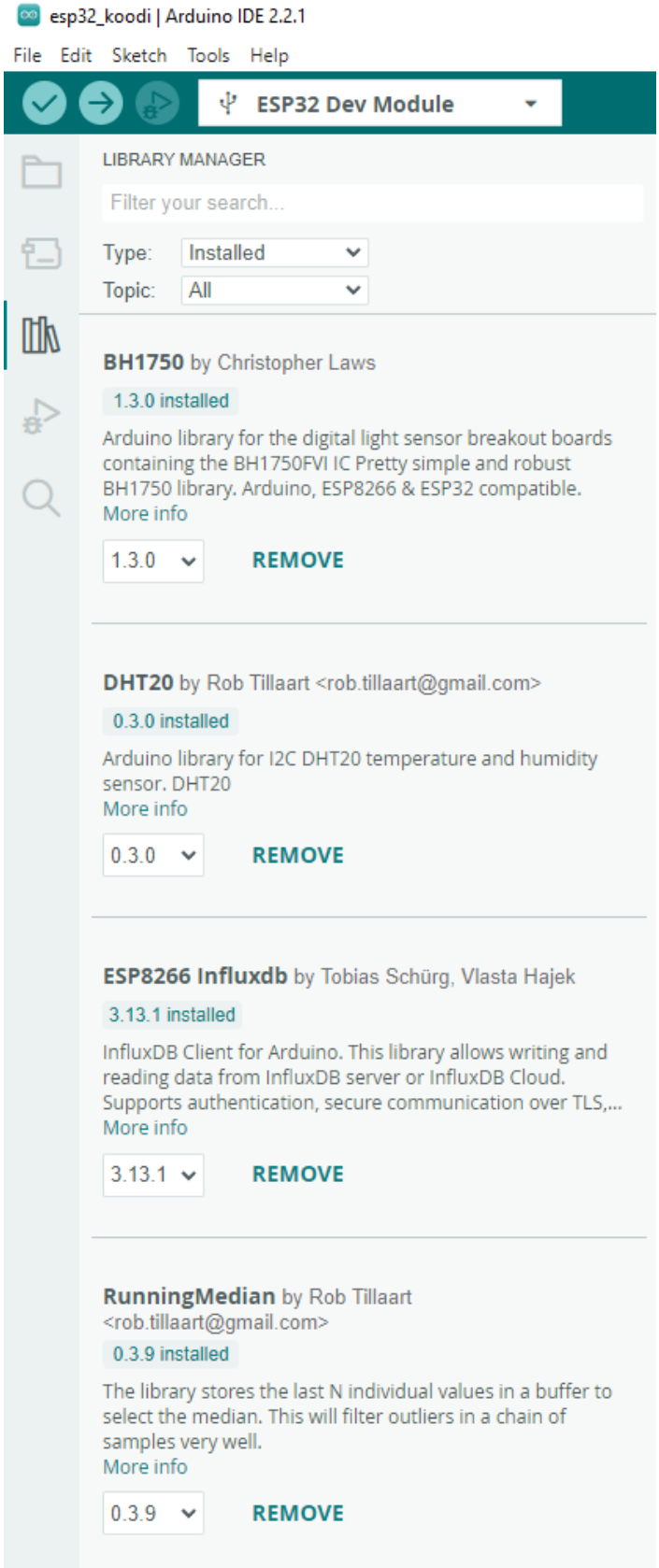
BH1750 valoisuussensorin lukemiseen ja konfigurointiin käytetään BH1750-nimellä löytyvää kirjastoa. Kirjasto löytyy GitHub-osoitteesta https://github.com/claws/BH1750

DHT20-lämpötila- ja ilmankosteussensorin mittaustulosten lukemiseen käytetään DHT20-nimellä löytyvää kirjastoa. Kirjasto löytyy GitHub-osoitteesta https://github.com/RobTillaart/DHT20

Mullan kosteuden keskiarvon laskemiseen käytetään RunningMedian-nimellä löytyvää kirjastoa. Kirjasto löytyy GitHub-osoitteesta https://github.com/RobTillaart/RunningMedian

InfluxDB:n kanssa kommunikointiin käytetään InfluxDB-Client-for-Arduino-nimellä löytyvää kirjastoa. Kirjasto löytyy GitHub-osoitteesta https://github.com/tobiasschuerg/InfluxDB-Client-for-Arduino

Kuvassa 13 on nähtävissä asennetut kolmannen osapuolen ohjelmakirjastot sekä niiden uusin versio kirjoitushetkellä.



Kuva 13. Asennetut kolmannen osapuolen ohjelmakirjastot.

6.3 Wi-Fi-verkkoyhteyden muodostaminen

Lähiverkon nimi eli SSID ja salasana on määritetty vakiomuuttujilla WIFI_SSID ja WIFI_PASSWORD. Wi-Fi-tilaksi on määritetty STA eli Station mode, jotta laite voidaan yhdistää lähiverkkoon. Ohjelman suoritusta jatketaan 15 sekunnin jälkeen tai kun verkkoyhteys on muodostettu.

```
#define WIFI_SSID "Verkon nimi"
#define WIFI_PASSWORD "Verkon salasana"

void connectWiFi() {
  int retry = 0;

  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  while (WiFi.status() != WL_CONNECTED && retry < 15) {
    delay(1000);
    retry++;
  }
}
```

Vakiomuuttuja WIFI_VERIFY_INTERVAL määrittelee aikavälin Wi-Fi-yhteyden tarkistukselle.

```
#define WIFI_VERIFY_INTERVAL 600000

if ((millis() - lastWiFiCheck) > WIFI_VERIFY_INTERVAL) {
    checkWiFi();
}
```

Mikäli Wi-Fi-yhteyttä ei ole, yritetään muodostaa uusi yhteys.

```
void checkWiFi() {
  if (WiFi.status() != WL_CONNECTED) {
    WiFi.disconnect();
    WiFi.reconnect();
  }
  lastWiFiCheck = millis();
}
```

6.4 InfluxDB-asetuksien määrittely

InfluxDB-kommunikointia varten on luotu InfluxDBClient-objekti, jolle on määritetty tarvittavat parametrit vakiomuuttujilla.

```
#define INFLUXDB_URL "http://1.2.3.4:8086"
#define INFLUXDB_TOKEN "************"
#define INFLUXDB_ORG "Koti"
#define INFLUXDB_BUCKET "Kasvit"

InfluxDBClient databaseClient(INFLUXDB_URL,
                              INFLUXDB_ORG,
                              INFLUXDB_BUCKET,
                              INFLUXDB_TOKEN
                              );
```

Jokaiselle mittaustulokselle on luotu oma Point-muuttuja, johon mittaustulos ja aikaleima tallennetaan.

```
Point dataPoint_LuminositySensor("luminosity_sensor");
Point dataPoint_SoilMoistureSensor("soil_moisture_sensor");
Point dataPoint_TemperatureSensor("temperature_sensor");
Point dataPoint_HumiditySensor("humidity_sensor");
Point dataPoint_Watering("watering");
```

Clientille määritellään datan kirjoitusasetukset seuraavilla vakiomuuttujilla:

- WRITE_PRECISION → Määritellään aikaleimojen merkkaaminen sekunnin tarkkuudella.

- MAX_BATCH_SIZE → Määritetään tallennettavan datan eräkooksi 24. Tällöin data tallennetaan suurissa erissä eikä jatkuvalla syötöllä, jolloin verkkoliikennettä saadaan vähennettyä. Kun eräkoko on saavutettu, tallennetaan data tietokantaan.
- MAX_BUFFER_SIZE → Määritetään puskurin kooksi 60. Tämän tulisi olla vähintään kaksi kertaa suurempi kuin eräkoko, koska vanha data säilyy puskurissa, vaikka verkkoyhteyttä tietokantaan ei olisi.
- FLUSH_INTERVAL → Määritetään tallennusaikaväliksi 60 s, jolloin puskurissa oleva data tallennetaan tietokantaan riippumatta siitä, onko eräkoko vielä saavutettu.

Mittauspisteille lisätään tagit device ja plant, jotka helpottavat mittauspisteiden tunnistamista tulevaisuudessa, jos useampia laitteita tai kasveja lisätään järjestelmään.

```
#define WRITE_PRECISION WritePrecision::S
#define MAX_BATCH_SIZE 24
#define MAX_BUFFER_SIZE 60
#define FLUSH_INTERVAL 60

void setupInfluxdb() {
  databaseClient.setWriteOptions(WriteOptions().
                                 writePrecision(WRITE_PRECISION).
                                 batchSize(MAX_BATCH_SIZE).
                                 bufferSize(MAX_BUFFER_SIZE).
                                 flushInterval(FLUSH_INTERVAL)
                                 );

  dataPoint_LuminositySensor.addTag("device", "ESP32");
  dataPoint_LuminositySensor.addTag("plant", "kahvipensas");

  dataPoint_SoilMoistureSensor.addTag("device", "ESP32");
  dataPoint_SoilMoistureSensor.addTag("plant", "kahvipensas");

  dataPoint_TemperatureSensor.addTag("device", "ESP32");
  dataPoint_TemperatureSensor.addTag("plant", "kahvipensas");

  dataPoint_HumiditySensor.addTag("device", "ESP32");
  dataPoint_HumiditySensor.addTag("plant", "kahvipensas");

  dataPoint_Watering.addTag("device", "ESP32");
  dataPoint_Watering.addTag("plant", "kahvipensas");
}
```

6.5 Mittaustuloksien lukeminen

Uusi sensorienluku tehdään aina, kun edellisestä mittauksesta on mennyt yli määritetyn aikavälin. Sensorien lukuaikaväliksi on määritetty 10 sekuntia.

```
#define SENSOR_READ_INTERVAL 10000

if ((millis() - lastSensorRead) > SENSOR_READ_INTERVAL)
```

Uusi mittaustulos tallennetaan globaaliin muuttujaan ja kirjoitetaan datapisteeseen tietokantaan tallennusta varten.

```
float luminosity;
int soilMoisture;

void getLuminosity() {
  luminosity = luminositySensor.readLightLevel();

  dataPoint_LuminositySensor.clearFields();
  dataPoint_LuminositySensor.setTime(time(nullptr));
  dataPoint_LuminositySensor.addField("value", luminosity);

  databaseClient.writePoint(dataPoint_LuminositySensor);
}

void getSoilMoisture() {
  soilMoisture = analogRead(SOIL_MOISTURE_PIN);

  dataPoint_SoilMoistureSensor.clearFields();
  dataPoint_SoilMoistureSensor.setTime(time(nullptr));
  dataPoint_SoilMoistureSensor.addField("value", soilMoisture);

  databaseClient.writePoint(dataPoint_SoilMoistureSensor);
}
```

Lämpötilan ja ilmankosteuden mittausta varten täytyy lähettää DHT20-sensorille ensin mittauspyyntö. Ensimmäinen mittauspyyntö lähetetään setup() funktiossa käyttäen DHT.requestData()-funktiota. Kun pyyntö on lähetetty, tulee odottaa vähintään 80 ms mittauksen valmistumiseksi, minkä jälkeen data voidaan lukea funktiolla DHT.readData(). Luettu data muunnetaan biteistä helpommin luettavaan muotoon funktiolla DHT.convert(), minkä jälkeen voidaan mittaustulokset tallentaa globaaleihin muuttujiin funktioilla DHT.getTemperature() ja DHT.getHumidity(). Kun mittaustulokset ovat tallessa, tehdään uusi mittauspyyntö sensorille ja tallennetaan nykyiset mittaustulokset omiin datapisteisiin.

```
float temperature;
float humidity;

void getTemperatureAndHumidity() {
  DHT.readData();
  DHT.convert();

  humidity = DHT.getHumidity();
  temperature = DHT.getTemperature();

  DHT.requestData();

  dataPoint_TemperatureSensor.clearFields();
  dataPoint_TemperatureSensor.setTime(time(nullptr));
  dataPoint_TemperatureSensor.addField("value", temperature);
  databaseClient.writePoint(dataPoint_TemperatureSensor);

  dataPoint_HumiditySensor.clearFields();
  dataPoint_HumiditySensor.setTime(time(nullptr));
  dataPoint_HumiditySensor.addField("value", humidity);
  databaseClient.writePoint(dataPoint_HumiditySensor);
}
```

6.6 Kasteluautomaatio

Kastelu tapahtuu mullan kosteussensorin mittaustulosten perusteella, johon raja-arvot on määritetty vakiomuuttujilla. Vakiomuuttuja AIR_THRESHOLD määrittää, milloin sensori on ilmassa ja mittaustuloksella ei tehdä mitään, ja WATERING_THRESHOLD määrittää, milloin multa on liian kuivaa ja kastelulle on tarvetta.

```
#define AIR_THRESHOLD 3200
#define WATERING_THRESHOLD 1600
```

Näiden raja-arvojen selvitys tapahtui alla olevalla testikoodilla ja pitämällä kosteussensoria ilmassa ja kuivassa mullassa.

```
#define SOIL_MOISTURE_PIN 32

void setup() {
      Serial.begin(9600);
}

void loop() {
      int soilMoisture = analogRead(SOIL_MOISTURE_PIN);
      Serial.println(soilMoisture);
      delay(1000);
}
```

Vakiomuuttujalla WATERING_INTERVAL on määritetty aikaväli kastelujen välille. Tällä voidaan määrittää esimerkiksi, ettei kastelu tapahdu useammin kuin kolmen päivän välein, vaikka multa olisi kuivaa.

```
#define WATERING_INTERVAL (3*24*60*60*1000)
```

Mittaustulosten tarkkuuden parantamiseksi on luotu puskuri, johon tallennetaan viimeisimmät 60 mittaustulosta, joka vastaa mittausta 10 minuutin ajalta.

```
#define RUNNING_SIZE 60

RunningMedian runningSoilMoisture = RunningMedian(RUNNING_SIZE);
```

Puskurissa olevista mittaustuloksista lasketaan keskiarvo lukuun ottamatta kahta pienintä ja suurinta mittaustulosta, jolloin saadaan suurimmat hypyt pois. Jotta kastelu tapahtuisi, täytyy runningSoilMoisture-puskurin olla täynnä, edellisestä kastelusta kulunut vähintään muuttujan WATERING_INTERVAL-verran aikaa ja puskurissa olevien mittaustulosten keskiarvon ylittää WATERING_THRESHOLD-raja. Kun edellä mainitut kriteerit täyttyvät, asetetaan releen GPIO-pinnin tilaksi LOW, jolloin rele aktivoituu ja vesipumppu menee päälle. Tieto vesipumpun aktivoinnista tallennetaan tietokantaan.

```
if (soilMoisture < AIR_THRESHOLD) {
      runningSoilMoisture.add(soilMoisture);

  if (runningSoilMoisture.isFull() &&
      (millis() - lastWatering) > WATERING_INTERVAL &&
      runningSoilMoisture.getAverage(RUNNING_SIZE - 4) > WATER-
      ING_THRESHOLD) {

            lastWatering = millis();
            relayState = LOW;
            digitalWrite(RELAY_PIN, relayState);

            dataPoint_Watering.clearFields();
            dataPoint_Watering.setTime(time(nullptr));
            dataPoint_Watering.addField("state", true);
            databaseClient.writePoint(dataPoint_Watering);
      }
}
```

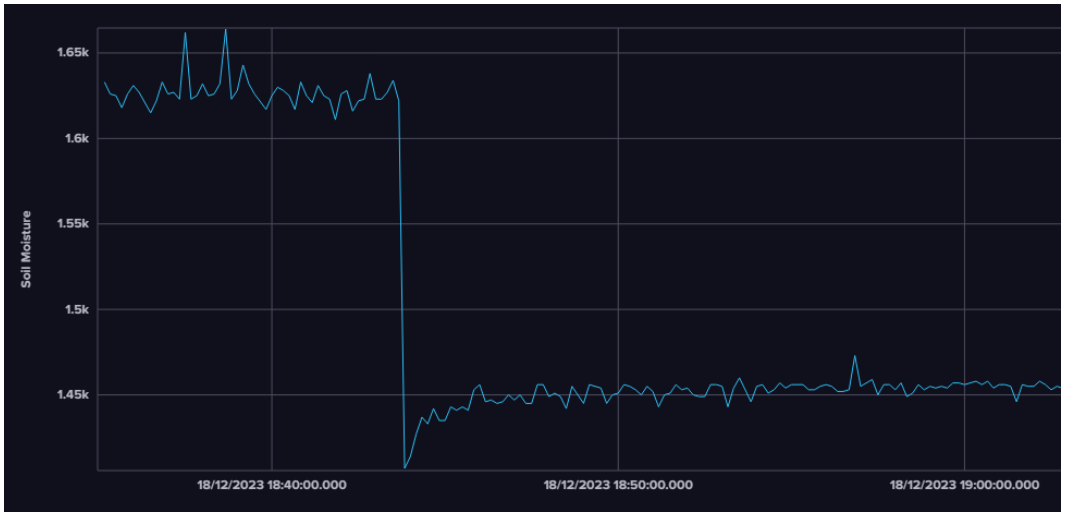
Vakiomuuttuja WATERING_DURATION määrittää, kuinka kauan vesipumppu on päällä. Vesipumppu sammutetaan, kun releen tila on LOW ja kastelussa on kestänyt määritetty aika. Sammutus tapahtuu asettamalla releen GPIO-pinni HIGH-tilaan, jolloin rele menee pois päältä. Tieto vesipumpun sammuttamisesta tallennetaan tietokantaan.

```
if (relayState == LOW && (millis() - lastWatering) >
      WATERING_DURATION)
{
    relayState = HIGH;
    digitalWrite(RELAY_PIN, relayState);

    dataPoint_Watering.clearFields();
    dataPoint_Watering.setTime(time(nullptr));
    dataPoint_Watering.addField("state", false);
    databaseClient.writePoint(dataPoint_Watering);
}
```

7 JÄRJESTELMÄN TESTAUS

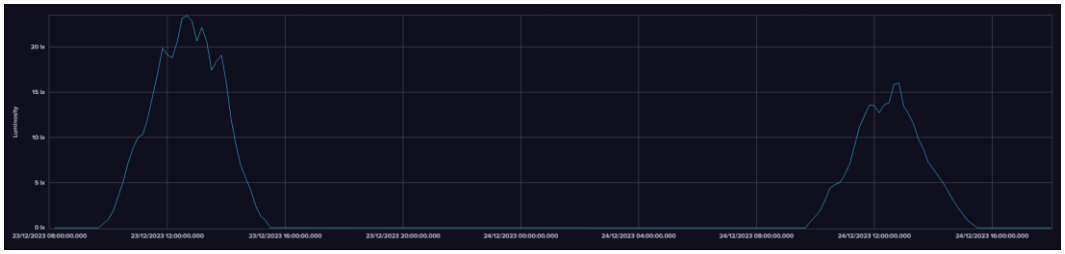
Järjestelmän kytkettäessä päälle onnistui se muodostamaan automaattisesti verkkoyhteyden kodin Wlaniin. Testatessa toimintaa verkkoyhteyden katketessa järjestelmä oli kykenevä jatkamaan toimintaa normaalisti ja muodostamaan yhteyden uudestaan 10 minuutin sisällä, kun verkko oli jälleen saatavilla. Sensoreiden mittaustulokset vastaavat ympäristössä tapahtuviin muutoksiin ja niiden tallentaminen tietokantaan onnistuu. Automaattinen kastelu tapahtuu mullan ollessa liian kuivaa ja kastelu kestää määritetyn ajan verran.

Kuvassa 14 on nähtävissä, kun mullan kosteus on ollut liian kuiva, ja automaatio on kastellut kasvin. Kuvan Y-akseli on AD-muunnettu mullan kosteusarvo ja X akseliaika.



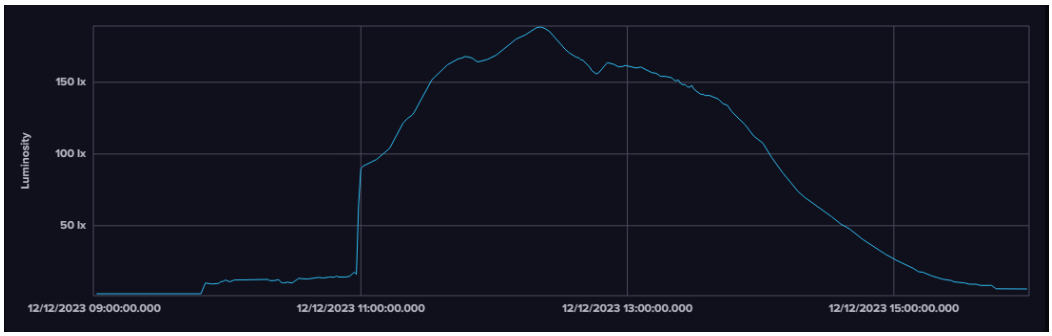
Kuva 14. Mullan kosteuden muutos johtuen kastelusta.

Kuvassa 15 on nähtävissä kahden päivän ajalta mitattu valoisuus, joka kuvastaa hyvin, kuinka lyhyt päivänvalo on Suomessa talvella. Vähäisen valoisuuden perusteella voidaan myös todeta, että molempina päivinä on ollut pilvistä, ja kasvi on saanut huonosti auringon valoa. Kuvan Y-akseli on valoisuus luxeina.



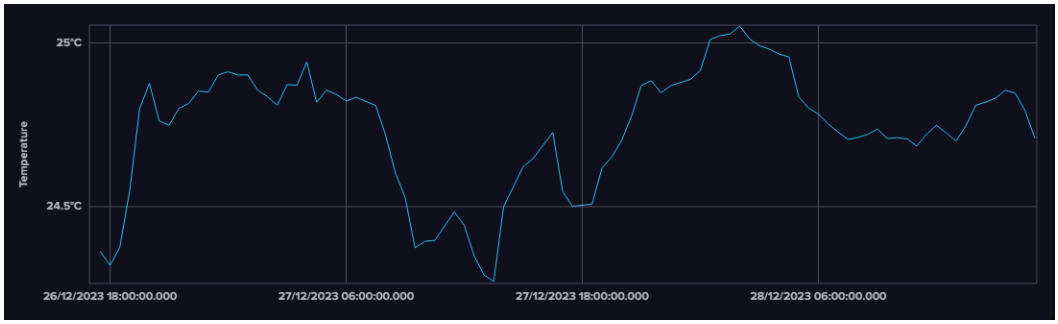
Kuva 15. Mitattu valoisuus pilvisinä päivinä.

Kuvassa 16 on nähtävissä mitattu valoisuus aurinkoisena ja pilvettömänä päivänä. Kuvan Y-akseli on valoisuus luxeina.



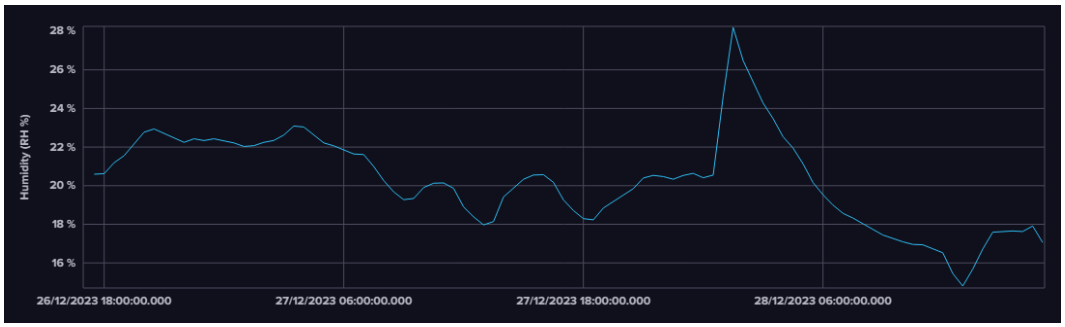
Kuva 16. Mitattu valoisuus valoisena päivän.

Kuvassa 17 on nähtävissä mitattu huoneen lämpötila kahden päivän ajalta. Lämpötila pysyy 24–25 °C välillä. Kuvan Y-akseli on lämpötila celsiuksina.



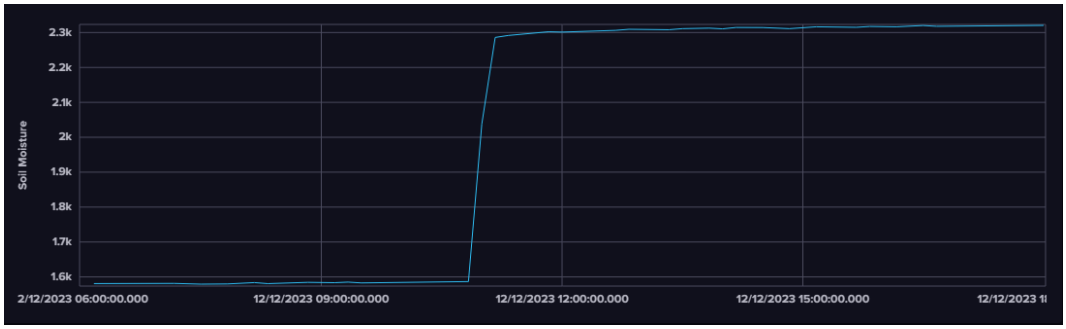
Kuva 17. Huoneen lämpötila.

Kuvassa 18 on nähtävissä huoneen ilmankosteus kahden päivän ajalta. Matala ilmankosteus johtuu talviajasta, jolloin lämmitys on normaalia korkeampi ja tällöin ilmankosteus on huono. Tämä saattaisi osoittautua ongelmaksi hieman eksoottisempien kasvien kanssa, sillä ne viihtyvät kosteissa olosuhteissa. Kuvan Y-akseli on suhteellinen ilmankosteus prosentteina.



Kuva 18. Huoneen ilmankosteus.

Mullan kosteussensorin asettelussa tuli huomatuksi, että mullan tiheydellä on suuri merkitys mittaustuloksiin. Jos sensoria liikuttaa mullassa, saattaa sensorin ja mullan välille jäädä ilmarako, joka vääristää mittaustuloksia. Kuvassa 19 on nähtävissä, kun sensoria on liikutettu ja mittaustuloksissa on huomattava muutos. Kuvan Y-akseli on AD-muunnettu mullankosteus arvo ja X-akseli aika.



Kuva 19. Virheellinen mittaustulos huonosti asetetusta mullan kosteussensorista.

8 YHTEENVETO

Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa järjestelmä, joka on kykenevä kastelemaan kodin pienkasveja sekä tekemään erilaisia mittauksia kasvuympäristöstä. Automaattinen kasvin kastelu toteutettiin pienellä upotettavalla vesipumpulla, joka kytketään päälle releellä, jota ohjataan ESP32-mikro-ohjaimella. Päällekytkentä tapahtuu mullan kosteuden perusteella. Kosteutta mitataan siihen erikoistuneella sensorilla. Kastelu toimii suunnitellusti, mutta jotta automaatio toimisi oikein, tulee mullan kosteussensorin olla oikein asetettuna multaan. Mikäli sensorin ja mullan välille jää ilmarakoja, aiheutuu virheellisiä mittaustuloksia ja tällöin automaatio voi kastella kasvia liian usein.

Tavoitteena oli myös kerätä ja tallentaa dataa kasvin kasvuympäristöstä, jota voitaisiin käyttää jälkeenpäin kasvuympäristön parantamiseen. Kerättävää dataa oli lämpötila, valoisuus sekä ilman- ja mullankosteus, joita mitattiin niihin erikoistuneilla sensoreilla. Sensorien antamat mittaustulokset ovat uskottavia ja reagoivat ympäristössä tapahtuviin muutoksiin. Mittaustuloksien tallennus onnistuu langattomasti lähiverkossa olevaan InfluxDB-tietokantaan, josta niitä voidaan helposti tarkastella. InfluxDB:ssä voidaan mittaustuloksilla tehdä erilaisia matemaattisia laskuja ja muodostaa dashboard, josta niiden seuraaminen ja analysointi on helppoa. Kuvassa 20 on nähtävissä järjestelmän seurantaan tehty dashboard.

Kuva 20. InfluxDB:ssä tehty dashboard.

Liitteenä 3 olevasta kuvasta voidaan nähdä, että järjestelmä vie kohtalaisen vähän tilaa ja on helposti siirreltävissä. Järjestelmä vaatii myös vähän ylläpitoa, mikä oli yksi päätavoitteista.

Kehitysideana voisi mullan kosteussensorin kokeilla sijoittaa kasviruukun kyljestä sivuttaissuunnassa, jolloin mittaustulokset saattaisivat vastata lähemmäksi kosteutta, joka on kasvin juuritasolla. Tällaista vaihtoehtoista menettelytapaa voisi olla mielenkiintoista testata käyttämällä toista sensoria, jolloin mittaustuloksia voisi vertailla toisiinsa samanaikaisesti. Toisena ideana järjestelmän voisi ohjelmoida hyödyntämään ESP32 mikro-ohjaimen deep sleep-ominaisuutta, jolla voitaisiin vähentää virrankulutusta merkittävästi.

Opinnäytetyön myötä opin ESP32-mikro-ohjaimen monipuolisista ominaisuuksista ja toiminnoista sekä kuinka sillä voidaan ohjata vesipumpun käyttöä. Opin syvällisemmin, kuinka eri sensorit toimivat käytännössä ja kuinka ohjelmallisesti niiden

mittaustuloksia voidaan lukea ja tallentaa. Virtualisoinnista ja InfluxDB-tietokannan käytöstä minulla oli jo hieman aikaisempaa kokemusta, mutta datan tallentaminen mikro-ohjaimen kautta oli uutta ja mielenkiintoista. Koska järjestelmää on nyt testattu ainoastaan talviaikana, on mielenkiintoista nähdä, millaisilta tulokset näyttävät eri vuodenaikoina. Opinnäytetyön myötä tunnen kehittyneeni tietoteknisten järjestelmien suunnittelussa ja toteutuksessa.

LÄHTEET

/1/ Az-Delivery. ESP32 NodeMCU Developmentboard datasheet. Viitattu 31.12.2023. https://cdn.shopify.com/s/files/1/1509/1638/files/ESP_-_32_NodeMCU_Developmentboard_Datenblatt_AZ-Delivery_Vertriebs_GmbH_10f68f6c-a9bb-49c6-a825-07979441739f.pdf?v=1598356497

/2/ Dean Foran. 2023. Comparing submersible and centrifugal water pump installations. Viitattu 31.12.2023. https://dtoswater.com/comparing-submersible-and-centrifugal-water-pump-installations/

/3/ Jeff Shepard. 2021. The difference between primary and secondary battery chemistries. Viitattu 31.12.2023. https://www.batterypowertips.com/difference-between-primary-secondary-battery-chemistries-faq/

/4/ Az-Delivery. KF-301 relay module datasheet. Viitattu 20.11.2023. https://cdn.shopify.com/s/files/1/1509/1638/files/KF-301_1-Relais_Modul_Datenblatt_AZ-Delivery_Vertriebs_GmbH_2fbfb0f0-d405-4fd8-b04e-1c3ae81a242f.pdf?v=1646995909

/5/ Mahdi Saleh, Imad H. Elhajj, Daniel Asmar, Isam Bashour, Samer Kidess. 2016. Experimental evaluation of low-cost resistive soil moisture sensors. Viitattu 10.01.2024. https://www.researchgate.net/publication/311530457_Experimental_evaluation_of_low-cost_resistive_soil_moisture_sensors

/6/ Joshua Hrisko. 2020. Capacitive Soil Moisture Sensor Theory, Calibration, and Testing. Viitattu 08.01.2024. https://agrilab.unilasalle.fr/projets/attachments/download/4088/Capacitive_Soil_Moisture_Sensors.pdf

/7/ DHT11 Digital relative humidity & temperature sensor datasheet. Viitattu 24.11.2023. https://www.electronicoscaldas.com/datasheet/DHT11_Aosong.pdf

/8/ Az-Delivery. DHT20 Humidity and Temperature module datasheet. Viitattu 24.11.2023. https://cdn.shopify.com/s/files/1/1509/1638/files/DHT20_-_April_2022.pdf?v=1651647663

/9/ AM2302/DHT22 Digital relative humidity & temperature sensor datasheet. Viitattu 24.11.2023. https://cdn-shop.adafruit.com/datasheets/Digital+humidity+and+temperature+sensor+AM2302.pdf

/10/ Bernd Albrecht. 2022. DHT20 - A new Temperature and Humidity Sensor. Viitattu 10.01.2024. https://www.az-delivery.de/en/blogs/azdelivery-blog-fur-arduino-und-raspberry-pi/dht20-ein-neuer-temperatur-und-luftfeuchtigkeitssensor

/11/ Az-Delivery. GY-302 BH1750 Light sensor module. Viitattu 31.12.2023. https://cdn.shopify.com/s/files/1/1509/1638/files/GY-302_Licht_Sensor_Modul_Datenblatt_AZ-Delivery_Vertriebs_GmbH.pdf?v=1608197155

/12/ ROHM SEMICONDUCTOR. BH1750FVI datasheet. Viitattu 09.01.2023. https://www.mouser.com/datasheet/2/348/bh1750fvi-e-186247.pdf

/13/ Espressif. Analog to Digital Converter. Viitattu 31.12.2023. https://docs.espressif.com/projects/esp-idf/en/v4.2/esp32/api-reference/peripherals/adc.html

/14/ Jeremy S Cook. 2021. What is a flyback diode and how does it work? Flyback protection diodes. Viitattu 10.01.2024. https://www.arrow.com/en/research-and-events/articles/flyback-protection-diodes

/15/ Xiaopei Liu. 2014. Multi-Machine Communication Based on I2C-Bus. Viitattu 10.01.2024. https://www.researchgate.net/publication/297308433_Multi-Machine_Communication_Based_on_I2C-Bus

LIITTEET

LIITE 1. ESP32-koodi.

```
#include <WiFi.h>
#include <Wire.h>
#include <BH1750.h>
#include <DHT20.h>
#include <InfluxDbClient.h>
#include <RunningMedian.h>

// Comment out line below to disable debug mode. Uncomment to enable debug.
// #define DEBUG

// Macro for debuging.
// Serial communication is not needed in use.
#ifdef DEBUG
  #define DEBUG_PRINT(x)   Serial.print(x)
  #define DEBUG_PRINTDEC(x, y) Serial.print(x, y)
  #define DEBUG_PRINTLN(x) Serial.println(x)
#else
  #define DEBUG_PRINT(x)
  #define DEBUG_PRINTDEC(x)
  #define DEBUG_PRINTLN(x)
#endif

#define RELAY_PIN 23                  // GPIO pin for relay
#define SDA_PIN 21                    // GPIO pin for I2C communication SDA pin
#define SCL_PIN 22                    // GPIO pin for I2C communication SCL pin
#define SOIL_MOISTURE_PIN 32          // GPIO pin for soil moisture sensor

#define SENSOR_READ_INTERVAL 10000    // Interval for sensor reads in ms. 10000ms = 10sec.
#define WIFI_VERIFY_INTERVAL 600000   // Interval to verify Wi-Fi connection in ms. 600000ms = 10min.

// Value when soil moisture sensor is considered being lifted out of soil and is measuring air.
#define AIR_THRESHOLD 3200

// Value when soil moisture is too low and need watering.
#define WATERING_THRESHOLD 1600

// Minimum interval between watering the plant. (days*hours*minutes*seconds*milliseconds)
#define WATERING_INTERVAL (3*24*60*60*1000)

// How long untill turn off watering. Value in ms.
#define WATERING_DURATION 2000
```

```
// Sample amount for running median/average measurement. Used for determining if watering is needed.
#define RUNNING_SIZE 60

// Wi-Fi credentials
#define WIFI_SSID "foo"
#define WIFI_PASSWORD "bar"

// InfluxDB info
#define INFLUXDB_URL "http://192.168.255.254:1234"
#define INFLUXDB_TOKEN "tokeni=="
#define INFLUXDB_ORG "Koti"
#define INFLUXDB_BUCKET "Kasvit"

// Set InfluxDB data write options.
// WRITE_PRECISION = Timestamp precision of written data
// MAX_BATCH_SIZE = Number of points that will be written to the database at once
// MAX_BUFFER_SIZE = Maximum number of points in buffer. Buffer contains new data that will be written to the database
and also data that failed to be written due to network failure or server overloading
// FLUSH_INTERVAL = Maximum time(in seconds) data will be held in buffer before points are written to the db
#define WRITE_PRECISION WritePrecision::S
#define MAX_BATCH_SIZE 24
#define MAX_BUFFER_SIZE 60
#define FLUSH_INTERVAL 60

// InfluxDB client for database operations.
// Requires:
//     INFLUXDB_URL    = URL to influxdb database.
//     INFLUXDB_ORG    = Organisation inside the database.
//     INFLUXDB_BUCKET = Bucket inside the database.
//     INFLUXDB_TOKEN  = Token with write permissions on specific bucket.
InfluxDBClient databaseClient(INFLUXDB_URL, INFLUXDB_ORG, INFLUXDB_BUCKET, INFLUXDB_TOKEN);

// Data points for influxdb.
Point dataPoint_LuminositySensor("luminosity_sensor");
Point dataPoint_SoilMoistureSensor("soil_moisture_sensor");
Point dataPoint_TemperatureSensor("temperature_sensor");
Point dataPoint_HumiditySensor("humidity_sensor");
Point dataPoint_Watering("watering");

BH1750 luminositySensor;          // Object for BH1750 luminosity sensor
DHT20 DHT;                        // Object for DHT20 temperature and humidity sensor

float luminosity;                 // Luminosity reading from sensor
float temperature;                // Temperature reading from sensor
```

```
float humidity;                 // Humidity reading from sensor
int soilMoisture;               // Soil moisture reading from sensor

int relayState = HIGH;          // Relay state. HIGH=not active, LOW=active.

// To get more accurate reading for soil moisture, we are using RunningMedian.
// RunninMedian can be used to get running value of median or average on set amount of measurement samples.
// Currently we are getting 10minutes worth of samples, which is 60 samples.
RunningMedian runningSoilMoisture = RunningMedian(RUNNING_SIZE);

// Last time watering was done.
// Initialize with WATERING_INTERVAL so we don't have to wait for the interval right after the reboot.
unsigned long lastWatering = WATERING_INTERVAL;

unsigned long lastSensorRead = 0;   // Last time sensors were read
unsigned long lastWiFiCheck = 0;    // Last time Wifi connection was verified

#ifdef DEBUG
 unsigned int sensorTimer = 0;
 int headerCounter = 0;
#endif

void setup() {
 // Serial is used for debuging and not needed in use.
 #ifdef DEBUG
  Serial.begin(9600);
 #endif

 connectWiFi();
 setupInfluxdb();

 // Configure correct time, for accurate timestamps.
 configTzTime("EET-2EEST,M3.5.0/3,M10.5.0/4", "pool.ntp.org", "time.nis.gov");

 pinMode(RELAY_PIN, OUTPUT);         // Configure output pin to control relay
 digitalWrite(RELAY_PIN, relayState);  // Write initial state for relay pin. HIGH=not active, LOW=active.

 Wire.begin(SDA_PIN, SCL_PIN);       // Initialize I2C pins
 DHT.begin();                        // Initialize DHT20 sensor
 luminositySensor.begin();           // Initialize luminosity sensor

 delay(1000);
 DHT.requestData();                  // Do first data request for DHT20 sensor.
}
```

```
void loop() {

 if ((millis() - lastSensorRead) > SENSOR_READ_INTERVAL) {

  #ifdef DEBUG
   sensorTimer = millis();
  #endif

  getLuminosity();
  getSoilMoisture();
  getTemperatureAndHumidity();
  lastSensorRead = millis();                // Save the time when sensors were read last time

  #ifdef DEBUG
   debugPrint();
  #endif

  // Check soilMoisture value to determine if sensor is out of the soil.
  if (soilMoisture < AIR_THRESHOLD) {
   runningSoilMoisture.add(soilMoisture);     // Add soilMoisture value to running buffer.

   // Here we verify if watering is needed.
   // 1. Check is runningSoilMoisture full. We dont wanna start watering based on couple samples.
   // 2. Check watering interval. This way we can limit watering eg. once a day, once in couple days or weeks.
   // 3. Get average of running soil moisture value and see if it's over the threshold. To calculate average we cut
   //    some of the values from the start and end, eg. middle 56 values are used to calculate average.
   if (runningSoilMoisture.isFull() &&
       (millis() - lastWatering) > WATERING_INTERVAL &&
       runningSoilMoisture.getAverage(RUNNING_SIZE - 4) > WATERING_THRESHOLD) {

     lastWatering = millis();                // Save the time when watering was done last time
     relayState = LOW;
     digitalWrite(RELAY_PIN, relayState);     // Set RELAY_PIN to LOW, which activates relay and starts watering.

     dataPoint_Watering.clearFields();
     dataPoint_Watering.setTime(time(nullptr));
     dataPoint_Watering.addField("state", true);

     databaseClient.writePoint(dataPoint_Watering);
   }
  }
 }

 // If relayState is LOW and WATERING has been finished, change relayState to HIGH and turn off watering.
 if (relayState == LOW && (millis() - lastWatering) > WATERING_DURATION) {
```

```
    relayState = HIGH;
    digitalWrite(RELAY_PIN, relayState);

    dataPoint_Watering.clearFields();
    dataPoint_Watering.setTime(time(nullptr));
    dataPoint_Watering.addField("state", false);

    databaseClient.writePoint(dataPoint_Watering);
  }

  // Occasionally check if Wi-Fi is connected. If not, attempt to reconnect.
  if ((millis() - lastWiFiCheck) > WIFI_VERIFY_INTERVAL) {
    checkWiFi();
  }
}

void setupInfluxdb() {
  // Set InfluxDB data write options.
  databaseClient.setWriteOptions(WriteOptions().
                          writePrecision(WRITE_PRECISION).
                          batchSize(MAX_BATCH_SIZE).
                          bufferSize(MAX_BUFFER_SIZE).
                          flushInterval(FLUSH_INTERVAL)
                          );

  // Add tag's for luminosity sensor datapoint.
  dataPoint_LuminositySensor.addTag("device", "ESP32");
  dataPoint_LuminositySensor.addTag("plant", "muorinkukka_Hope");

  // Add tag's for soil moisture sensor datapoint.
  dataPoint_SoilMoistureSensor.addTag("device", "ESP32");
  dataPoint_SoilMoistureSensor.addTag("plant", "muorinkukka_Hope");

  // Add tag's for temperature sensor datapoint.
  dataPoint_TemperatureSensor.addTag("device", "ESP32");
  dataPoint_TemperatureSensor.addTag("plant", "muorinkukka_Hope");

  // Add tag's for humidity sensor datapoint.
  dataPoint_HumiditySensor.addTag("device", "ESP32");
  dataPoint_HumiditySensor.addTag("plant", "muorinkukka_Hope");

  // Add tag's for watering occurances.
  dataPoint_Watering.addTag("device", "ESP32");
  dataPoint_Watering.addTag("plant", "muorinkukka_Hope");
```

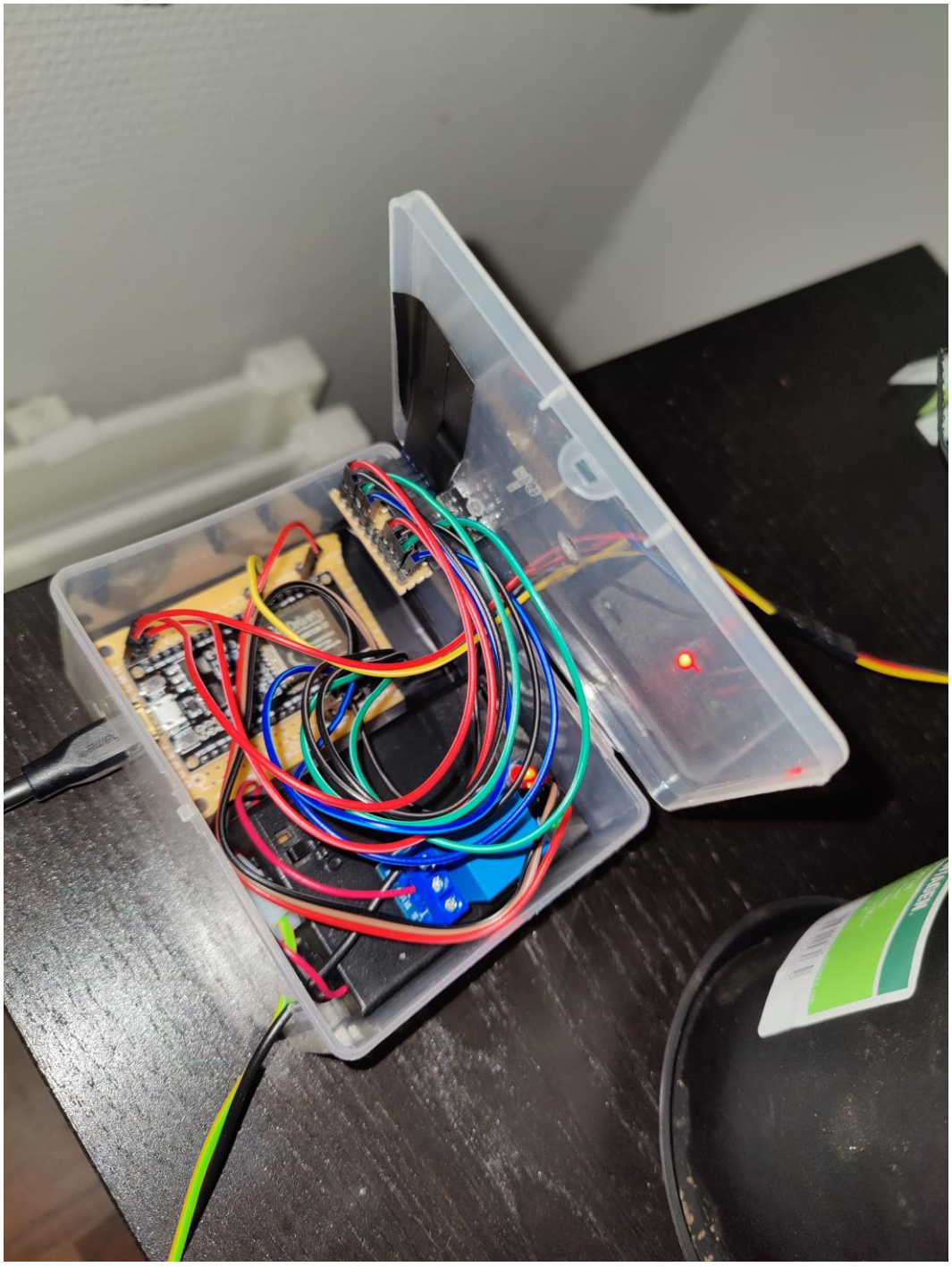
```
 #ifdef DEBUG
  if(databaseClient.validateConnection()) {
   DEBUG_PRINT("Connected to InfluxDB: ");
   DEBUG_PRINTLN(databaseClient.getServerUrl());
  }
  else {
   DEBUG_PRINT("InfluxDB connection failed: ");
   DEBUG_PRINTLN(databaseClient.getLastErrorMessage());
  }
 #endif
}

void connectWiFi() {
 int retry = 0;

 WiFi.mode(WIFI_STA);
 WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
 DEBUG_PRINT("Connecting to WiFi..");
 while (WiFi.status() != WL_CONNECTED && retry < 15) {
  DEBUG_PRINT('.');
  delay(1000);
  retry++;
 }

 #ifdef DEBUG
  if (WiFi.status() == WL_CONNECTED) {
   DEBUG_PRINTLN("\nConnected!");
   DEBUG_PRINT("IP address: ");
   DEBUG_PRINTLN(WiFi.localIP());
  }
  else {
   DEBUG_PRINTLN("Wi-Fi connection failed!");
  }
 #endif
}

void checkWiFi() {
 DEBUG_PRINTLN("Checking Wi-Fi connection.");
 if (WiFi.status() != WL_CONNECTED) {
  DEBUG_PRINTLN("Attempting to reconnect.");
  WiFi.disconnect();
  WiFi.reconnect();

  #ifdef DEBUG
   if (WiFi.status() == WL_CONNECTED) {
```

```
      DEBUG_PRINTLN("\nConnected!");
      DEBUG_PRINT("IP address: ");
      DEBUG_PRINTLN(WiFi.localIP());
     }
     else {
      DEBUG_PRINTLN("Wi-Fi connection failed!");
     }
   #endif
  }
  lastWiFiCheck = millis();
}

// Get new luminosity reading from the sensor and save new data to datapoint.
void getLuminosity() {
  luminosity = luminositySensor.readLightLevel();

  dataPoint_LuminositySensor.clearFields();
  dataPoint_LuminositySensor.setTime(time(nullptr));
  dataPoint_LuminositySensor.addField("value", luminosity);

  databaseClient.writePoint(dataPoint_LuminositySensor);
}

// Get new soil moisture reading from the sensor and save new data to datapoint.
void getSoilMoisture() {
  soilMoisture = analogRead(SOIL_MOISTURE_PIN);

  dataPoint_SoilMoistureSensor.clearFields();
  dataPoint_SoilMoistureSensor.setTime(time(nullptr));
  dataPoint_SoilMoistureSensor.addField("value", soilMoisture);

  databaseClient.writePoint(dataPoint_SoilMoistureSensor);
}

// Get new temperature and humidity readings from the sensor and save new data to datapoints.
void getTemperatureAndHumidity() {
  DHT.readData();                    // Read new data.
  DHT.convert();                     // Convert raw bit data to temperature and humidity values.

  humidity = DHT.getHumidity();          // Get humidity.
  temperature = DHT.getTemperature();    // Get temperature.

  DHT.requestData();                 // Do a new data request for the sensor.

  dataPoint_TemperatureSensor.clearFields();
```

```
  dataPoint_TemperatureSensor.setTime(time(nullptr));
  dataPoint_TemperatureSensor.addField("value", temperature);
  databaseClient.writePoint(dataPoint_TemperatureSensor);

  dataPoint_HumiditySensor.clearFields();
  dataPoint_HumiditySensor.setTime(time(nullptr));
  dataPoint_HumiditySensor.addField("value", humidity);
  databaseClient.writePoint(dataPoint_HumiditySensor);
}

#ifdef DEBUG
  void debugPrint() {

    if ((headerCounter % 20) == 0) {
      headerCounter = 0;
      DEBUG_PRINTLN("Luminosity \tSoil M / Median  / Avg \t\tTemperature \tHumidity \tRead time \tTotal length");
    }
    headerCounter++;

    DEBUG_PRINT(luminosity);
    DEBUG_PRINT(" lx");
    DEBUG_PRINT("\t");

    DEBUG_PRINT(soilMoisture);
    DEBUG_PRINT("  ");
    DEBUG_PRINT(" / ");
    DEBUG_PRINT(runningSoilMoisture.getMedian());
    DEBUG_PRINT(" / ");
    DEBUG_PRINT(runningSoilMoisture.getAverage(56));
    DEBUG_PRINT("\t");

    DEBUG_PRINT(temperature);
    DEBUG_PRINT(" °C");
    DEBUG_PRINT("\t");

    DEBUG_PRINTDEC(humidity, 1);
    DEBUG_PRINT(" %RH");
    DEBUG_PRINT("\t");

    DEBUG_PRINT(lastSensorRead-sensorTimer);
    DEBUG_PRINT(" ms");
    DEBUG_PRINT("\t");
    DEBUG_PRINT("\t");

    DEBUG_PRINTLN(databaseClient.pointToLineProtocol(dataPoint_LuminositySensor).length() +
```

```
            databaseClient.pointToLineProtocol(dataPoint_SoilMoistureSensor).length() +
            databaseClient.pointToLineProtocol(dataPoint_TemperatureSensor).length() +
            databaseClient.pointToLineProtocol(dataPoint_HumiditySensor).length()
          );
 }
#endif
```

LIITE 2. Kuva ESP32-laatikosta.



LIITE 3. Kuva koko järjestelmästä.