SAVONIA
ammattikorkeakoulu

OPINNÄYTETYÖ - AMMATTIKORKEAKOULUTUTKINTO
TEKNIIKAN JA LIIKENTEEN ALA

SAVONIAN MASI-KAIVURI

Opinnäytetyö

TEKIJÄ Eetu Miettinen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Tutkinto-ohjelma Konetekniikan tutkinto-ohjelma	
Työn tekijä Eetu Miettinen	
Työn nimi Savonian MaSi-kaivuri	
Päiväys 24.5.2023	Sivumäärä/Liitteet 45/1
Toimeksiantaja/Yhteistyökumppani(t) Savonia-ammattikorkeakoulu	

Tiivistelmä

Opinnäytetyön tavoitteena oli integroida radio-ohjattu pienoismallikaivuri osaksi Savonian MaSi-hanketta. Päätavoitteena projektissa oli tehdä pienoismallikaivuriin tarvittavat mekaaniset sekä ohjelmistolliset muutokset, jotta sitä voitaisiin hyödyntää tulevaisuudessa perustana digitaalisen kopion luomiselle sekä tekoälytoimintojen kehittämiselle. Muutokset sisälsivät muun muassa kaivurin ohjaus- ja tiedonkeruujärjestelmän suunnittelemisen sekä valmistamisen, tiedonsiirtoon käytettävän yhteysprotokollan valinnan sekä käyttöönoton ja alkuperäisen ohjainlaitteen vaihtamisen toiseen.

Tuloksena pienoismallikaivuri muokattiin toimimaan verkkoyhteyden avulla, ja sen hydraulijärjestelmän paineita voitiin seurata reaaliajassa. Myös kaivurin ohjauslaite saatiin vastaamaan oikean kaivinkoneen ohjauslaitetta. Tämän lisäksi projektiin lisättiin lukuisia pienempiä parannuksia, jotka toiminnoillaan helpottivat ja edesauttoivat kaivurin käyttöä. Kaikki projektin alussa asetetut vaatimukset toteutettiin, ja kaivurista saatiin hyvä pohja jatkokehitystä varten.

MaSi-hankkeen jatkuessa kaivurin kehitystyö kohdistuu jatkossa enemmän simulointiympäristöihin sekä tekoälytoimintoihin. Ideana olisi valmistaa testiympäristö, jossa etäohjattavaa kaivuria voitaisiin käyttää tekoälyn avustuksella erilaisissa tehtävissä, omavalmisteisilla työkaluilla. Myös 5G-verkon hyödyntäminen kaivurin tiedonsiirtojärjestelmässä oli suunnitteilla.

Avainsanat
Kaivuri, etäohjaus, digitaalinen kopio, kehitystyö

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport	
Degree Programme Degree Programme in Mechanical Engineering	
Author Eetu Miettinen	
Title of Thesis Savonia MaSi-excavator	
Date 24 May 2023	Pages/Appendices 45/1
Client Organisation /Partners Savonia University of Applied Sciences	

Abstract

The aim of the thesis was to integrate a radio-controlled miniature excavator as part of Savonia's MaSi project. The thesis was comissioned by Savonia University of Applied Sciences. The main objective of the project was to make the necessary mechanical and software changes to the model excavator so that it could be utilized in the future as a basis for creating a digital replica and developing artificial intelligence functions. These changes included designing and manufacturing the excavator's control and data system, selecting and implementing the communication protocol used for data transfer, and replacing the original control device with another one.

As a result, the miniature excavator was modified to operate via a network connection, and the pressure levels in its hydraulic system could be monitored in real-time. The control device of the excavator was also made to correspond to the control device of a real excavator. In addition, numerous smaller enhancements were added to the project, which facilitated and aided the use of the excavator. All the requirements set at the beginning of the project were met, and the excavator proved to be a good foundation for further development.

As the MaSi project continues, the development work for the excavator will focus more on simulation environments and artificial intelligence functions. The idea is to create a test environment where the remote-controlled excavator could be used with the help of artificial intelligence to perform various tasks with self-made tools. The utilization of the 5G network in the excavator's data transfer system was also planned.

Keywords

Excavator, remote-control, digital twin, development

SISÄLTÖ

KUVALUETTELO

1 JOHDANTO

1.1 Tausta ja tavoitteet

Teknologian nopea kehitys on viime vuosikymmeninä muuttanut yhteiskuntaa monin eri tavoin. Erityisesti teknologiateollisuudessa digitaalisten kaksosten ja tekoälyn rooli on kasvanut merkittävästi, ja niitä hyödynnetäänkin yhä laajemmin erilaisissa koneissa ja laitteissa. Yksi teknologian kehityksen suuntaviivoista on etäohjaus, joka mahdollistaa laitteiden ohjaamisen kauempaa tietoliikenneyhteyden avulla. Etäohjaus toimii usein yhteistyössä digitaalisten kopioiden ja tekoälyn kanssa, mikä parantaa ohjaustarkkuutta, laitteiden reagointikykyä ja käyttökokemusta.

Tämän opinnäytetyön suoranaisena kohteena ei ole digitaaliset kopiot tai tekoäly, vaan toimivan pohjan luonti niitä varten. Projektissa keskitytään radio-ohjatun pienoismallikaivurin muokkaamiseen siten, että sitä on mahdollista ohjata tietoliikenneyhteyden ylitse. Verkkoyhteyden käyttö ohjaustiedon lähettämiseen mahdollistaa myös helpon tiedon vastaanottamisen, eli kaivurin ajonaikaisia tapahtumia tulisi myös voida seurata.

Lyhykäisyydessään projektin minimivaatimukset olivat seuraavanlaiset:

1. Kaivurin ohjausjärjestelmän muutos toimimaan verkon ylitse
2. Yhteysprotokollan valinta ja käyttöönotto
3. Kaivurin ohjaus esimerkiksi näppäimistöllä tai peliohjaimella
4. Tiedonkeruu kaivurista ajon aikana.

Tulevaisuudessa projektia voitaisiin hyödyntää kaivurin digitaalisen kopion sekä tekoälytoimintojen kehityksen parissa. Kaivurin ajonaikaista dataa, kuten esimerkiksi asentoa, hydrauliikan painetta, lämpötilaa sekä liikkeitä voitaisiin verrata simulaatiomallin kanssa reaaliajassa. Datan kerääminen sekä analysointi mahdollistaisi digitaalisen kopion kehittämisen entistä tarkemmaksi, kuin myös sen vertailun oikeaan laitteeseen. Projekti tehtiin osittaisessa yhteistyössä toisen opiskelijan kanssa.

1.2 MaSi-hanke

MaSi (”Liikkuvan työkoneen simulointiympäristö”) -projekti pyrki hankkimaan Savonia-ammattikorkeakoululle simulointiympäristön ja lisäämään simulaatio-osaamista Pohjois-Savossa. Hanke pyrki auttamaan alueen yrityksiä kehittämään kilpailukykyään ja tulevaisuuden teknologioita. Hankkeen tavoitteena oli luoda simulaatioympäristö, integroida se Savonian toimintoihin ja vahvistaa yhteistyötä yritysten kanssa. Hanke tuki kilpailukykyä, edisti teknologian käyttöönottoa, paransi koulutusta ja lisäsi yhteistyötä korkeakoulutuksen ja TKI-toiminnan välillä.

1.3 Projektin valinta

Projekti valikoitui opinnäytetyökseni, koska ennen opinnäytetyön aloitusta olin käynyt Machine Simulation Basics -kurssin, jossa samaisesta pienoismallikaivurista luotiin 3D-malli digitaalista kopiota varten. Kurssi meni pitkälti ohjelmistojen oppimisessa, mutta halu projektin jatkokehitykseen jäi. Myöskään sen hetkisellä työnantajallani ei ollut tarjota mahdollisuutta tarpeeksi laajalle opinnäytetyön aiheelle. Opinnäytetyön valintaa edesauttoi myös kiinnostus ohjelmoinnin opetteluun. Minulla oli pienimuotoista kokemusta perustason ohjelmoinnista, mutta olio-ohjelmointia tai Pythonia en ollut ennen käyttänyt.

2 ETÄOHJATTAVAT LAITTEET SEKÄ SIMULAATIOYMPÄRISTÖT

2.1 Etäohjattavat työkoneet

Etäohjatut työkoneet ovat laitteita, joita käytetään erilaisten tehtävien suorittamiseen ilman, että operaattorin tarvitsee olla fyysisesti koneen läheisyydessä. Tämä mahdollistaa työskentelyn vaarallisissa, saavuttamattomissa tai epämiellyttävissä ympäristöissä, kuten korkeilla paikoilla, radioaktiivisilla tai räjähdysvaarallisilla alueilla, syvissä vesissä tai pölyisissä ja meluisissa ympäristöissä. Etäohjaus voi tapahtua esimerkiksi langattoman yhteyden, kuten Wi-Fi:n, satelliittiyhteyden tai mobiiliverkon kautta. Etäohjattavat työkoneet kattavat laajan valikoiman laitteita eri teollisuudenaloilta, kuten sotateollisuus, maanrakennus, kaivostoiminta, metsäteollisuus, pelastus- ja etsintätehtävät.

Hyvänä esimerkkinä toimii Sandvik AutoMine -järjestelmä, joka mahdollistaa maanalaisten kuormaajien sekä porien käytön etäohjauksella sekä niiden automatisoinnin. AutoMine voi ohjata kaivoskoneita etäohjauskeskuksesta käsin, mikä suojaa työntekijöitä vaarallisilta työolosuhteilta, kuten maanalaisilta räjähtäviltä kaasuille tai sortuvilta tunneleilta. Se mahdollistaa myös reaaliaikaisen tiedon keräämisen ja analysoinnin, joka auttaa parantamaan prosessien tehokkuutta, tunnistamaan pullonkaulat ja ennakoida mahdolliset ongelmat ennen niiden ilmenemistä. AutoMine voi myös suorittaa joitain tehtäviä täysin autonomisesti, mikä säästää työntekijän aikaa (Sandvik RockTechnology 2022).

Kuva 1. Sandvik AutoMine -porauslaite (Sandvik 2022)

2.2 Simulointiympäristöt ja digitaaliset kaksoset

Simulaatioympäristö on tietokonepohjainen alusta, joka mahdollistaa digitaalisten kaksosten testaamisen, analysoinnin ja kehittämisen. Digitaaliset kaksoset pyrkivät jäljittelemään mahdollisimman tarkasti vastaavia todellisia kohteita (IBM 2022).

Simulaatioympäristöt, joissa hyödynnetään digitaalisia kaksosia, ovat hyvin arvokkaita useilla eri aloilla. Niiden avulla voidaan testata uusia ideoita ja ratkaisuja ennen niiden käyttöönottoa todellisissa tilanteissa, mikä voi johtaa ajan, rahan ja resurssien säästämiseen sekä riskien minimoimiseen. Esimerkiksi, jos halutaan muuttaa kaivinkoneen nostopuomin rakennetta niin, että sen painopiste siirtyy, voidaan tällainen muutos testata nopeasti simulaatioympäristössä. Tämä auttaa havaitsemaan mahdolliset heikkoudet ja vahvuudet ennen kuin muutokset toteutetaan todellisuudessa.

Toinen esimerkki voisi olla tilanne, jossa kaivinkoneen käyttöympäristön ilman lämpötilaa nostetaan simulaatioympäristössä. Tällöin voidaan tutkia, kuinka lämpötilan muutos vaikuttaa koneen toimintaan ja aiheuttaako se mahdollisesti ongelmia.

Digitaalisia kaksosia voidaan käyttää myös rinnakkain oikean koneen kanssa, jolloin työn aikana kerättävillä mittausarvoilla voidaan ennakoida potentiaalisia vikatilanteita ja aikatauluttaa tarvittavia huoltotoimenpiteitä. Simulaatioympäristöt, joissa käytetään digitaalisia kaksosia, voivat sisältää monimutkaisia malleja, jotka perustuvat oikean työkoneen todellisiin mittausarvoihin. Näin ollen, ne tarjoavat arvokasta tietoa, jota voidaan hyödyntää koneiden ja järjestelmien kehittämisessä ja optimoinnissa (IBM 2022)

Kuva 2. Sandvikin hyödyntämä porauslaitteen testausympäristö Mevea-simulointiympäristössä (Mevea, 2018)

3 RADIO-OHJATTU PIENOISMALLIKAIVURI

3.1 Kaivurin toimintaperiaate sekä tekniset ominaisuudet

Kuva 3. Radio-ohjattu kaivuri ennen muutoksia (Miettinen 2022)

Savonia oli hankkinut radio-ohjattuja pienoismallikaivureita koekaniineiksi projektia varten. Projektissa käytetty pienoismallikaivuri oli toimintatavaltaan lähellä oikeaa kaivinkonetta, oikean kaivinkoneen tavoin sen puomia, vartta sekä kauhaa voitiin ohjata hydraulijärjestelmän avulla. Paine hydraulijärjestelmälle tuotettiin pumpun avulla, mitä pyöritettiin erillisellä voimanlähteellä, oikean kaivurin tapauksessa yleensä polttomoottorilla, ja kyseisen pienoismallikaivurin tapauksessa sähkömoottorilla. Pumpulta saatu paine ohjattiin halutuille sylintereille portaattomasti säätyvän venttiilipöydän avulla, minkä venttiileitä voitiin ohjata pienten servomoottoreiden avulla. Servomoottoreita taas ohjattiin radio-ohjauslaitteiden avulla. Virran toimintoihin kaivuri sai sen rungon takaosaan sijoitetusta akusta.

3.1.1 Radio-ohjaus

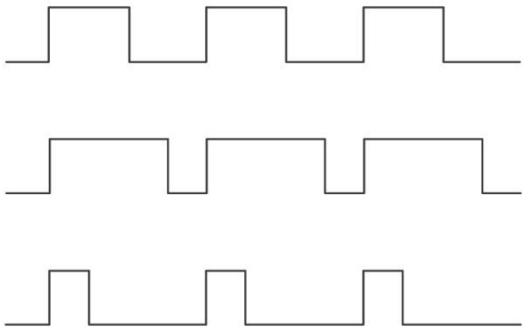
Radio-ohjattavien (RC) laitteiden ohjausjärjestelmä koostuu pääsiassa kahdesta osasta: kauko-ohjaimesta sekä vastaanottimesta. Näiden avulla käyttäjä voi ohjata RC-laitteen tapahtumia langattomasti. Yleensä ohjaamiseen käytetään kauko-ohjainta, johon käyttäjä syöttää ohjauskomennot sauvojen, kytkimien tai ohjauspyörän avulla.

Kuva 4. Kaivurin alkuperäinen kauko-ohjain (Miettinen 2023)

Vastaanotin on RC-laitteen sisällä sijaitseva elektroninen laite, joka vastaanottaa kauko-ohjaimelta tulevat radioaallot. Vastaanotin dekoodaa radioaaltojen sisältämät ohjaussignaalit ja välittää ne muille komponenteille, kuten servomoottoreille. Välitys tapahtuu yleensä PWM-signaalin avulla (Buchi 2014, 10)

3.1.2 PWM-ohjattu elektroniikka

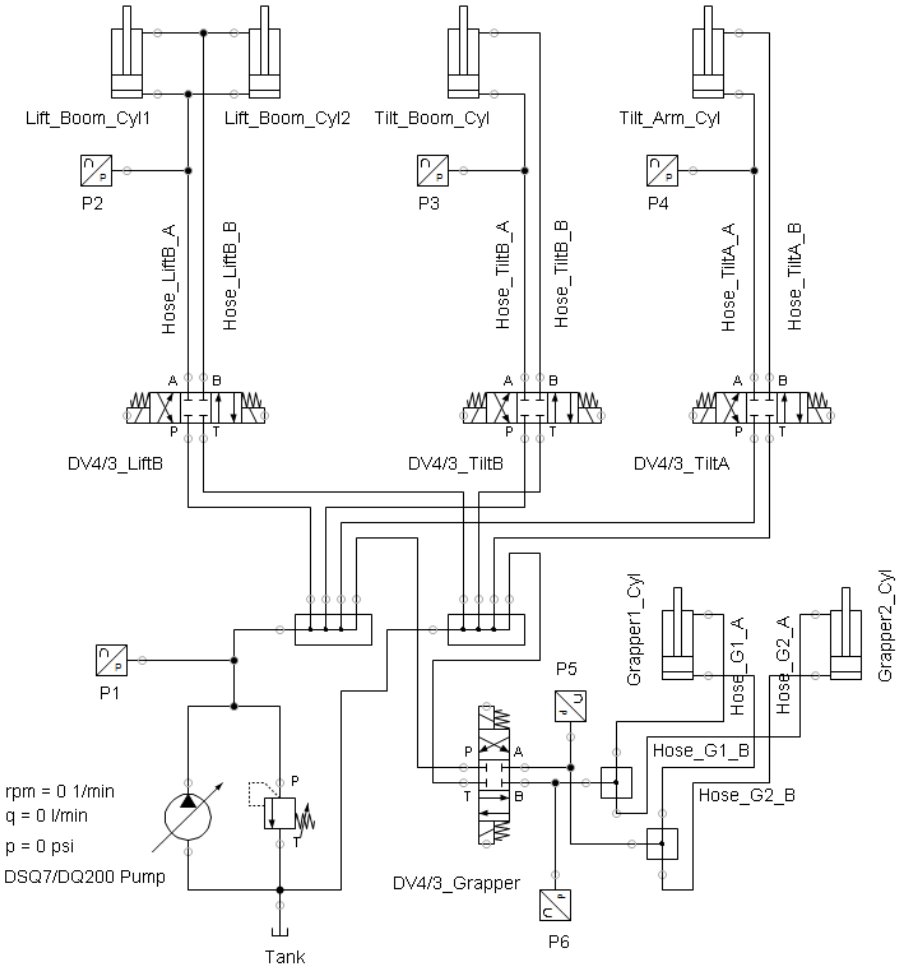
PWM (Pulse Width Modulation, Pulssinleveysmodulaatio) on modulaatiomenetelmä, jolla digitaalista tietoa voidaan siirtää analogisessa muodossa säätämällä pulsseja. PWM-signaali koostuu sarjasta neliöpulsseja, joiden leveys muuttuu tietyn aikavälin sisällä. Pulssin leveys eli kesto kuvastaa informaatiota, jota halutaan siirtää.



Kuva 5. Havainnekuva eripituisista PWM-pulsseista (CircuitBread 2022)

Käytännössä radio-ohjaimen lähettämä tieto saapuu vastaanottimelle, mikä tulkkaa saapuvat PWM-signaalit ja ohjaa niiden perusteella laitteen toimintoja. Esimerkiksi servomoottorin asentoa voidaan säätää pulssin leveydellä, jolloin lyhyempi pulssi kääntää servoa esimerkiksi enemmän vasemmalle ja pidempi pulssi enemmän oikealle (CircuitBread 2022).

3.1.3 Hydrauliikka



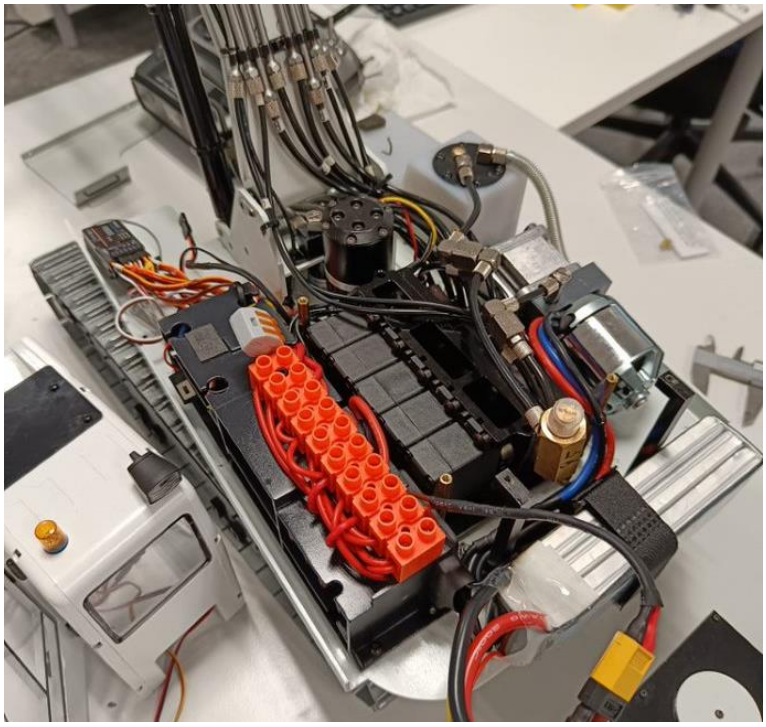
Kuva 6. MaSi-kaivurin hydraulikaavio (Hänninen 2022)

Pienoismallikaivurin pääliikkeet perustuivat hydrauliikkaan. Hydrauliikka on tekniikka, jossa nestettä, kuten hydraulista öljyä, käytetään voiman ja liikkeen siirtämiseen ja ohjaamiseen. Hydrauliikassa nesteen painetta hyödynnetään mekaanisen voiman tuottamiseen. Paine tuotetaan hydraulipumpulla (kaaviossa ”DQ200 Pump”), joka muuntaa mekaanisen energian hydraulisen energian muotoon nostamalla nesteen painetta. Pumppu siirtää nestettä järjestelmän osasta, yleensä tankilta, toiseen ja synnyttää painetta, joka saa aikaan esimerkiksi liikkeen hydraulisylinterissä (Kauranne 2013, 137 & Keinänen & Kärkkäinen 2005, 170).

Hydraulisylinterit (kaaviossa ”Lift_Boom_Cyl1”) ovat mekaanisia laitteita, jotka muuttavat nesteen paineen lineaariseksi liikkeeksi. Sylintereissä on männän ja männänvarren muodostama liikkuva osa, joka siirtyy, kun nesteen paine kasvaa tai pienenee sylinterin sisällä.

Hydraulisylintereiden liikkeitä ohjataan yleensä venttiileiden (kaaviossa ”DV4/3_LiftB”) avulla. Koska järjestelmässä tarvitaan lähes aina enemmän kuin yksi venttiili, käytetään niissä venttiilipöytiä, jotka sisältävät useamman venttiilin, joiden avulla nesteen virtausta ja painetta voidaan säädellä. Näin voidaan ohjata hydraulisylintereiden liikettä ja suorittaa erilaisia toimintoja.

Hydraulijärjestelmän toiminta voi tuottaa paineita, jotka voivat vahingoittaa järjestelmän muita komponentteja. Tämän takia hydraulijärjestelmässä on hyvä olla ylipaineventtiili, joka kierrättää hydraulinesteen pumpulta takaisin säiliöön, kun tietty nestepaineen arvo ylitetään (Kauranne 2013, 259)



Kuva 7. Viisi servomoottoria kiinni kaivurin venttiilipöydässä (Miettinen 2022)

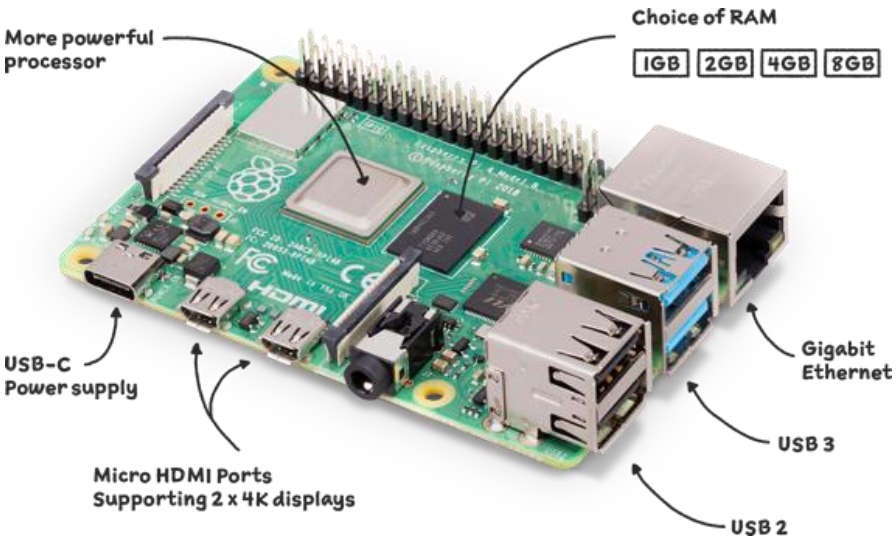
Kuvassa 8 näkyy pienoismallikaivurin viisi servomoottoria, jotka ohjaavat kaivurin venttiilipöytää. Venttiilipöydän oikeassa alakulmassa näkyy messinkinen säädettävä ylipaineventtiili, joka kierrättää nesteen takaisin oikeassa yläkulmassa näkyvälle tankille, kun paine ylittää venttiilin säätöarvon. Oikeassa reunassa näkyy myös pumppu sekä suurikokoinen kiiltävä hopeanvärinen moottori, joka pyörittää pumppua. Varsinainen pumppuosa on Volkswagenin DSG-vaihdelaatikon öljypumppu DQ200, johon pienoismallivalmistaja on valmistanut oman sovitteen kyseiselle sähkömoottorille.

4 OHJAUSJÄRJESTELMÄN MUUTTAMINEN

Pienoismallikaivurin valmistelu projektia varten täytyi aloittaa vaihtamalla alkuperäinen radio-ohjaus monipuolisempaan järjestelmään. Järjestelmän tuli olla kykeneväinen ohjaamaan erilaisia laitteita sekä keräämään tietoa antureilta, ja sitä pitäisi pystyä laajentamaan helposti tulevaisuudessa. Järjestelmän tulisi olla myös kohtuullisen vähävirtainen, hyvin tuettu sekä dokumentoitu ja aloittelijaystävällinen.

4.1 Kaivurin ohjaus Raspberry Pi:n avulla

Raspberry Pi on kompakti, yhden piirilevyn tietokone, joka on suunniteltu erityisesti opetuskäyttöön sekä harrastajien ja kehittäjien projekteihin. Vaikka laite on edullinen ja energiatehokas, se tarjoaa riittävän suorituskyvyn monenlaisiin sovelluksiin. Raspberry Pi on yhteensopiva lukuisten erilaisten anturien, moottorien sekä lisälaitekorttien kanssa, ja sitä voidaan ohjelmoida useilla eri ohjelmointikielillä. Laaja tukiyhteisö ja monipuoliset käyttömahdollisuudet auttavat minitietokoneen käytössä ja sen opettelussa (Raspberry 2022).



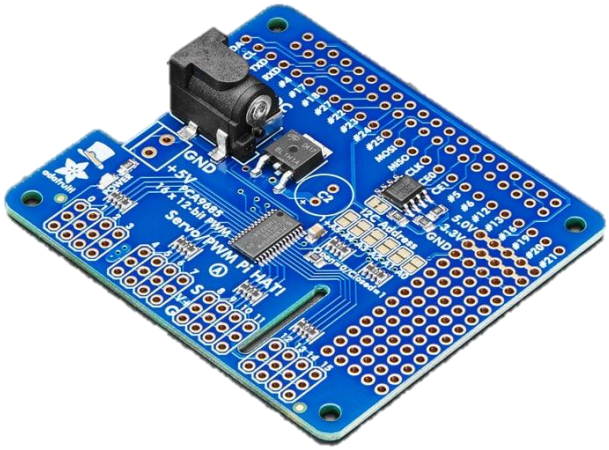
Kuva 8. Raspberry Pi 4-tietokone (Raspberry 2022)

Projektiin valikoitui Raspberry Pi 4 Model B, koska se täytti useita tärkeitä vaatimuksia: edullisuuden, laskenta- ja energiatehokkuuden sekä kaivurin koon aiheuttamat kokovaatimukset. Pi 4 on hyvin tuettu järjestelmä, josta löytyy runsaasti esimerkkejä. Raspberry Pi:n GPIO-pinnit ja lisälaitekortit mahdollistavat erinomaiset päivitysmahdollisuudet ja käytännössä rajattomat lisäominaisuudet.

Käyttöjärjestelmäksi laitteeseen valittiin Raspberry Pi OS, joka on suoraan kyseiselle tietokoneelle suunniteltu Linux-pohjainen käyttöjärjestelmä.

4.1.1 Raspberry Pi:n lisälaitekortit

Raspberry Pi:n "hatut" ovat lisälaitekortteja, jotka on suunniteltu liitettäväksi suoraan Raspberry Pi –piirilevylle sen GPIO-liitäntään (General Purpose Input/Output). Ne laajentavat laitteen toiminnallisuutta ja mahdollistavat erilaisten antureiden, moottorien ja muiden komponenttien käytön. Hatut on suunniteltu noudattamaan tiettyä fyysistä ja sähköistä standardia, mikä tekee niiden käytöstä helppoa ja vaivatonta Raspberryn ohjelmointiympäristöissä. Lisälaitekortteja on saatavilla monia erilaisia eri valmistajilta, ja useampaa niistä voidaan käyttää samanaikaisesti GPIO-liitännän avulla (Raspberry 2014).



Kuva 9. Adafruit PWM-lisälaitekortti (Adafruit 2022)

Kaivurin servomoottorien ohjaamista varten projektiin valittiin Adafruit Industries -yrityksen valmistama ”16-Channel PWM / Servo HAT”- lisälaitekortti, joka mahdollistaa kuudentoista eri servomoottorin ohjaamisen PWM-signaalilla. Se valikoitui projektiin sen helppokäyttöisyyden sekä suuren kanavamäärän tuennan vuoksi.

4.2 Ohjainlaitteen valinta

Alkuperäinen idea oli käyttää tietokoneeseen liitettyä peliohjainta kaivurin ohjauslaitteena. Kuitenkin suunnitelmaa muutettiin, ja projektissa päätettiin hyödyntää ensisijaisena ohjauslaitteena koulun käytössä olevaa Motion Platform -laitetta (2DOF). 2DOF tarjosi kaivinkoneen kaltaisen käyttökokemuksen kahdella ohjaussauvalla ja telojen ohjaamiseen tarkoitetuilla polkimilla. Lisäksi laitteeseen kuului myös Windows-pohjainen tietokone, jota voitiin hyödyntää ohjaussauvojen arvojen lukemisessa. Ohjauslaitteet olivat liitetty tietokoneeseen National Instrumentsin USB I/O-lukulaitteella, jonka lukemiseen oli saatavilla valmiita ohjelmakirjastoja.

Kuva 10. Motion Platform (2DOF) ohjauslaite (Miettinen 2022)

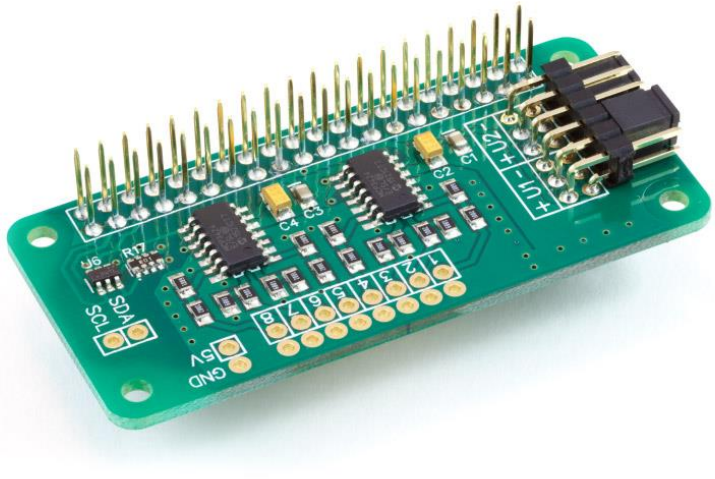
4.3 Tiedonkeruujärjestelmä

Projektin yhtenä vaatimuksena oli kerätä käytönaikaista dataa kaivurista. Keskityimme vielä tässä vaiheessa projektia vain yhteen mittausjärjestelmään, joka keskittyi hydrauliikan käyttöpaineen mittaamiseen linjakohtaisesti. Projektissa mukana ollut kanssaopiskelija suunnitteli sekä valmisti anturipöydän, mihin kytkimme kuusi paineanturia.

Kuva 11. Kaivuriin kytketyt paineanturit (Miettinen 2022)

Kyseinen paineanturi välittää painetiedon lineaarisesti jännitteen avulla siten, että pienimmän paineen ollessa kyseessä jännite on noin 0,5 V ja suurimman paineen ollessa kyseessä jännite on noin 4,5 V. Jännite voidaan muuttaa digitaaliseksi tiedoksi A/D-muuntimen avulla.

A/D-muunnin (analogia-digitaalimuunnin) on elektroninen laite, joka muuntaa jatkuvia analogisia signaaleja, kuten jännitetasoja tai antureiden tuottamia vastusarvoja, diskreeteiksi digitaalisiksi signaaleiksi, kuten binäärisiksi biteiksi (0 ja 1). Muunnosprosessin avulla analogista signaalia voidaan käsitellä digitaalisissa järjestelmissä (ElectronicsTutorials 2022).

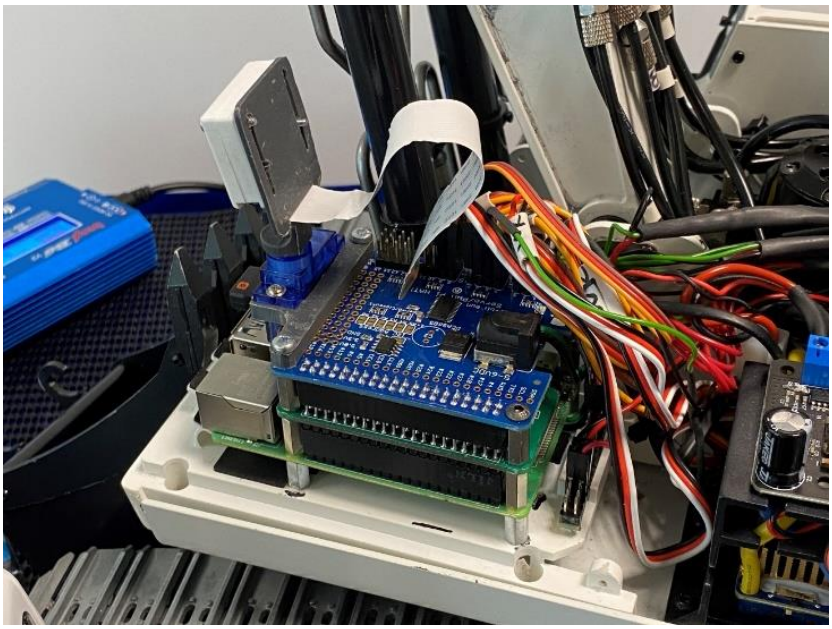


Kuva 12. ABelectronics ADCPi -muunnin (ABelectronics 2022)

Muunnokseen projektissa käytettiin ABelectronics ADCPi -lisälaitekorttia, joka tarjosi kahdeksan kanavaa analogisten signaalien lukemiseen. Kortti valittiin projektiin sen helppokäyttöisyyden ja laajan Python-ohjelmakirjaston takia. Joitakin kortin kanavia jäi hyödyntämättä, minkä ansiosta lisämittaukset, kuten öljyn lämpötilan seuranta, voidaan toteuttaa vaivattomasti tulevaisuudessa. Oma osuuteni projektissa ei keskittynyt tiedonkeruujärjestelmän kehittämiseen tai valmistamiseen, mutta se on kuitenkin hyvä mainita sen tärkeyden vuoksi projektikokonaisuudessa.

4.4 Kokoonpano

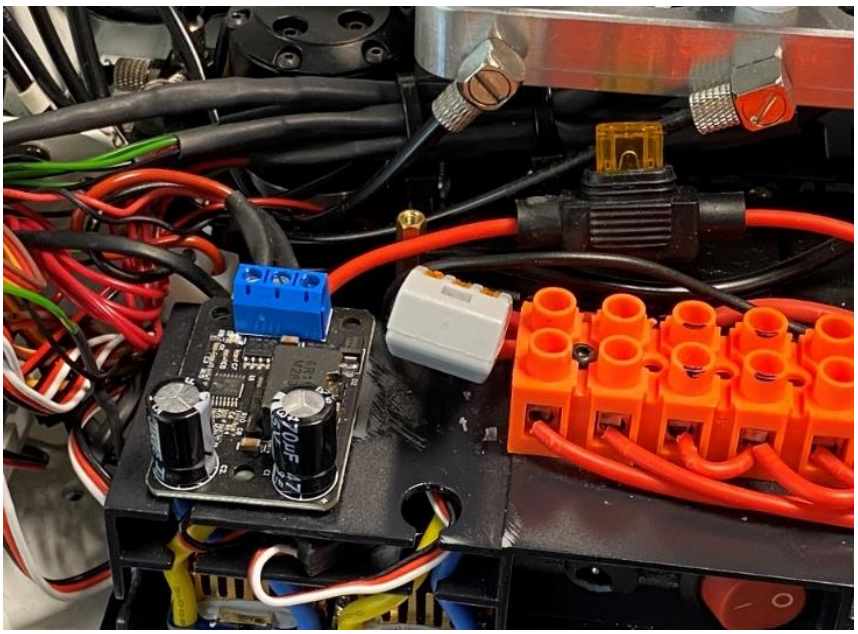
Laitekokoonpano suunniteltiin mahtumaan pienoismallikaivurin ohjaamoon. Kiinnitys tapahtui poraamalla reiät ohjaamon pohjalevyyn ja kiinnittämällä kokoonpano hyödyntäen lisälaitekorttien jalkoja. Kaikki tarvittavat johdotukset hoidettiin liittimillä, jotta laitteet voitaisiin tarvittaessa irrottaa nopeasti.



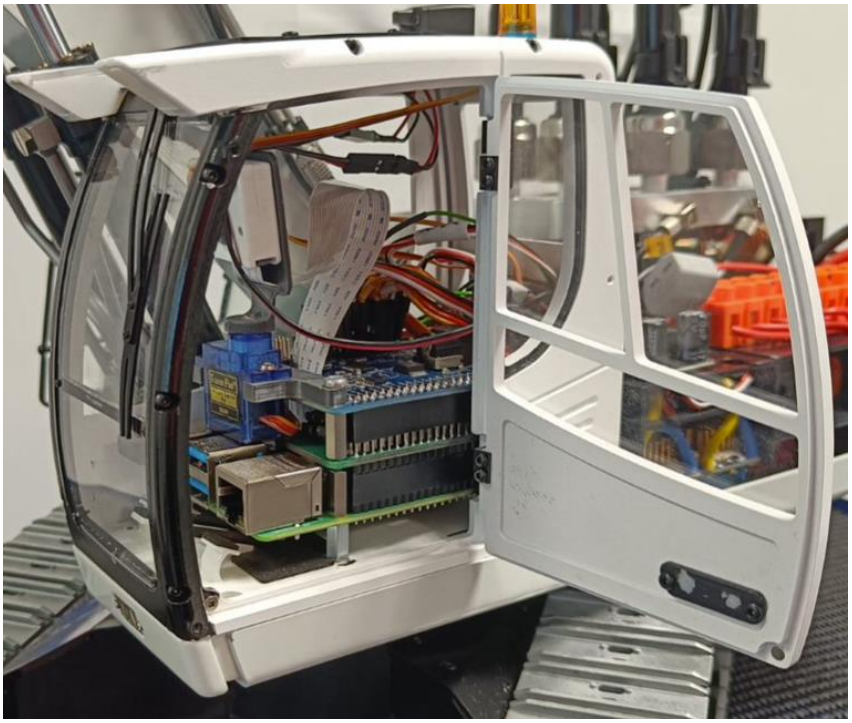
Kuva 13. Kiinnitetty laitekokoonpano (Miettinen 2022)

Laitekokoonpanon etureunaan suunniteltiin ja 3D-tulostettiin myös kiinnike servomoottorille, ja servomoottoriin suunniteltiin kiinnike kameramoduulille. Kameraa pystyttiin kääntämään PWM-ohjatun servon avulla.

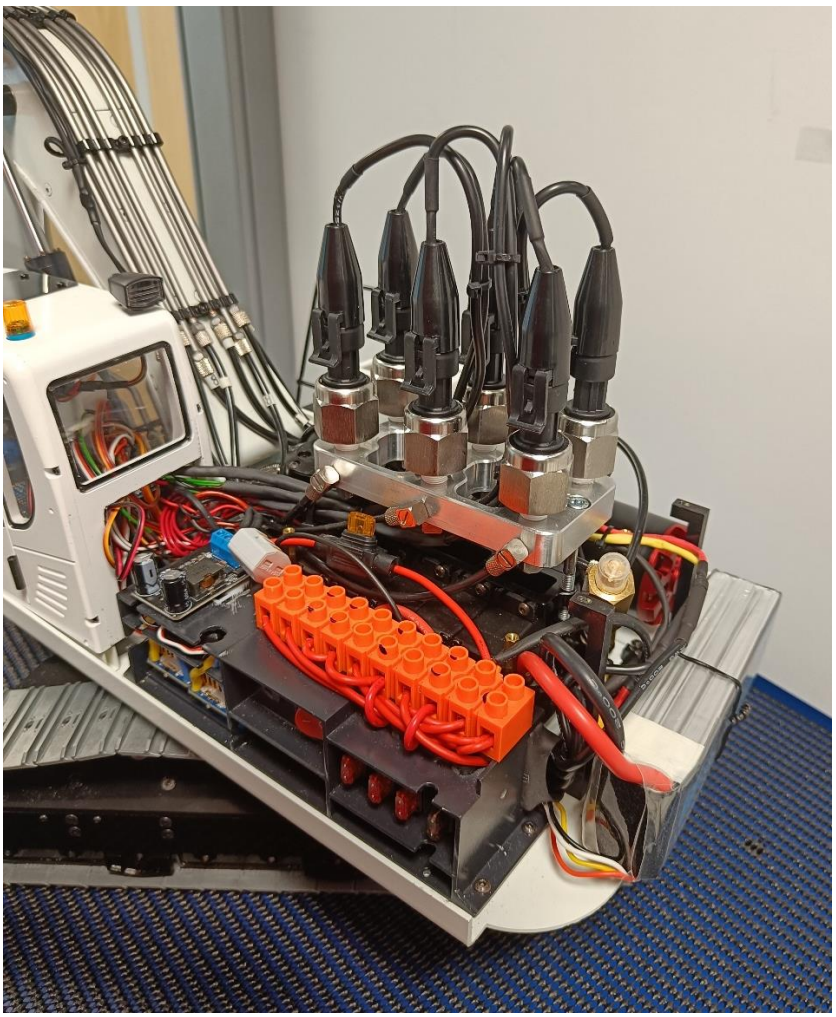
Kaivurissa käytettävä jännite poikkesi Raspberry Pi:n vaatimasta käyttöjännitteestä. Tämän ongelman ratkaisemiseksi käytettiin DC/DC-konvertteria, jonka avulla saimme laskettua kaivurin käyttämän noin 12V jännitteen Raspberrylle sopivaksi 5V jännitteeksi. Lisäksi virransyöttöön lisättiin sulake mahdollisten häiriötilanteiden varalta.



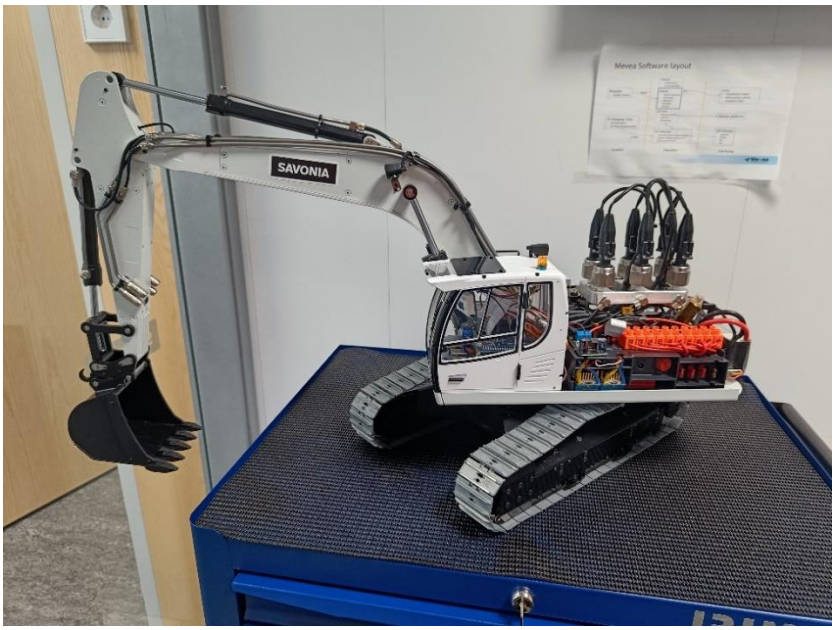
Kuva 14. DC/DC-konvertteri (Miettinen 2022)



Kuva 15. Laitekokoonpano sovitettuna kaivurin ohjaamoon. Kamerakuva antaa autenttisen kuvan oikean kaivinkoneen käyttäjän näkökulmasta (Miettinen 2022)



Kuva 16. Anturipöytä asennettuna (Miettinen 2022)



Kuva 17. Mekaanisesti valmis kaivuri (Miettinen 2022)

5 OHJELMAN KEHITTÄMINEN KAIVURILLE

5.1 Python-ohjelmointi

Projektin varsinainen haastava osuus oli tarvittavien toimintojen ohjelmointi. Projekti suunniteltiin jo alusta alkaen toteutettavaksi Pythonilla, sillä se vaikutti helpoiten opittavalta ja siitä oli olemassa paljon kirjastoja sekä selkeitä esimerkkejä.

Python on yleiskäyttöinen, korkean tason ohjelmointikieli, joka on tunnettu helppokäyttöisyydestään ja selkeästä luettavuudesta. Pythonin laaja standardikirjasto sekä runsas kolmansien osapuolten tarjoama kirjastovalikoima tekevät siitä erittäin suositun ohjelmointikielen monenlaisissa projekteissa. (Python Software Foundation 2022)

Projektin ohjelmoinnin tapahtumien selkeyttämiseksi on hyvä esitellä yleisimmät projektissa käytetyt menetelmät yksinkertaisten esimerkkien avulla. Näin lukija voi ymmärtää itse projektissa käytettyä ohjelmaa käytännön tasolla, eikä suoran ”raakakoodin” laajempaa avaamasta välttämättä tarvita. Tällä lähestymistavalla pyritään auttamaan erityisesti niitä lukijoita, jotka eivät ole ohjelmoinnin asiantuntijoita (opinnäytetyön kirjoittaja mukaan lukien).

5.1.1 Silmukat ja ehtolauseet

Ehtolauseet ovat ohjelmoinnin rakenteita, jotka mahdollistavat erilaisten toimintojen suorittamisen riippuen tietyistä ehdoista. Yleinen ehtolause on if-else-rakenne, jossa if-lauseen ehto testataan, ja jos se on tosi, suoritetaan siihen liittyvät toiminnot. Muussa tapauksessa suoritetaan else-lauseen toiminnot (Vanhala & Nikula 2022, 25).

```
1  luku = 7
2
3  if luku > 5:
4      print("Luku on suurempi kuin 5")
5  else:
6      print("Luku on 5 tai pienempi")
```

Kuva 18. Esimerkkikuva if-ehtolauseesta (Miettinen 2023)

Silmukat ovat ohjelmoinnissa käytettyjä rakenteita, joiden avulla voidaan toistaa tiettyjä toimintoja useita kertoja. Yleisiä silmukoita ovat for-silmukka ja while-silmukka. For-silmukka toistaa toimintoja määritellyn määrän kertoja, kun taas while-silmukka toistaa toimintoja niin kauan kuin tietty ehto on voimassa. (Vanhala & Nikula 2022, 34).

```
for i in range(5):
    print(i)

```

Kuva 19. For-silmukka. Tämä silmukka tulostaa luvut 0-4 (Miettinen 2023)

```
i = 0
while i < 5:
    print(i)
    i += 1

```

Kuva 20. While-silmukka. Tulostaa samat arvot kuin Kuva 19 (Miettinen 2023)

```
i = 0
while True:
    print(i)
    i += 1
    if i >= 5:
        break

```

Kuva 21. While True -silmukkaa voidaan jatkaa äärettömiin, tai lopettaa käskyllä break. Tulostaa samat arvot (Miettinen 2023)

5.1.2 Funktiot

Funktio on ohjelmoinnissa käytetty koodirakenne, joka suorittaa tietyn tehtävän tai laskutoimituksen. Funktion avulla voi kutsua koodia ohjelman eri osista tarpeen mukaan. Niiden käyttö auttaa vähentämään koodin toistoa, tekee ohjelmasta helpommin luettavan ja ylläpidettävän sekä mahdollistaa koodin uudelleenkäytön. Funktiot luovat selkeälukuisen pohjan ohjelman toiminnalle (Vanhala & Nikula 2022, 40).

```
# Hauku-funktio
def hauku(nimi):
    print(f"{nimi} sanoo: Hau!")

# Kutsutaan Hauku-funktiota
hauku("Rekku") # Tulostaa: "Rekku sanoo: Hau!"
```

Kuva 22. Esimerkkikuva (Miettinen 2023)

Yllä olevassa esimerkissä havainnollistetaan funktion toimintaa yksinkertaisen koiran haukuntaa kuvaavan esimerkin avulla. Esimerkkiohjelma määrittelee yksinkertaisen funktion nimeltä ”hauku”, joka

ottaa yhden argumentin, ”nimi”. Tämä funktio tulostaa annetun nimen ja lisää siihen tekstin "sanoo: Hau!". Funktion jälkeen koodi kutsuu hauku-funktiota antaen sille argumentiksi merkkijonon "Rekku". Tämä aiheuttaa funktion tulostavan "Rekku sanoo: Hau!". Argumenttia muutettaessa koiran nimi muuttuu, mutta itse tekeminen pysyy samana.

5.1.3 Luokat ja oliot

Python-luokat ovat ohjelmarakenteita, jotka auttavat organisoimaan ohjelmistoja tehokkaammin. Luokat toimivat mallipohjina, joiden avulla voidaan luoda samankaltaisia kappaleita eli olioita ohjelmassa. Luokat auttavat organisoimaan ohjelmia jakamalla ne loogisiin osiin. Tämä vähentää ohjelman toistoa, helpottaa ylläpitoa ja virheiden käsittelyä sekä mahdollistaa ohjelman uudelleenkäytön ja laajennettavuuden (Kasurinen 2009, 177).

```
# Määritellään muuttuja nykyinen vuosi
nykyinenvuosi = 2023

# Määritellään luokka Koira
class Koira:
    # Alustusmetodi, joka asettaa koiran nimen ja iän
    def __init__(self, nimi, ikä):
        self.nimi = nimi
        self.ikä = ikä

    # Metodi, joka tulostaa koiran haukun
    def hauku(self):
        print(f"{self.nimi} sanoo: Hau!")

    # Metodi, joka laskee ja tulostaa koiran syntymävuoden
    def laske_ikä(self):
        syntymävuosi = nykyinenvuosi-self.ikä
        print(f"{self.nimi} on syntynyt vuonna {syntymävuosi}!")

# Luodaan oliot (instanssit) luokasta Koira
koira1 = Koira("Rekku", 3)
koira2 = Koira("Hauva", 5)

# Kutsutaan olioiden metodia hauku
koira1.hauku()  # Tulostaa: "Rekku sanoo: Hau!"
koira1.laske_ikä()    # Tulostaa: "Rekku on syntynyt vuonna 2020!"
koira2.hauku()  # Tulostaa: "Hauva sanoo: Hau!""
koira2.laske_ikä()    # Tulostaa: "Hauva on syntynyt vuonna 2018!"
```

Kuva 23. Esimerkkikuva Koira-luokasta ja sen metodeista (Miettinen 2023)

Funktion tavoin koiran haukku voidaan toteuttaa myös luokan ja metodin avulla. Esimerkkiohjelma määrittelee ensin nykyisen vuoden ja luokan Koira. Luokka sisältää kolme metodia (Metodilla tarkoitetaan luokan sisällä määritettyä funktiota): alustusmetodi init, joka asettaa koiran nimen ja iän,

hauku-metodi, joka tulostaa koiran haukun, ja laske_ikä -metodi, joka laskee ja tulostaa koiran syntymävuoden.

Seuraavaksi ohjelma luo kaksi oliota (instanssia) luokasta Koira: koira1 ja koira2. Näille olioille annetaan eri nimet ja iät. Lopuksi ohjelma kutsuu kummankin olion hauku- ja laske_ikä -metodeja, jotka tulostavat koiran haukun ja syntymävuoden.

5.1.4 Listat ja sanakirjat

Listat ja sanakirjat ovat Pythonin sisäisiä tietorakenteita, jotka mahdollistavat monipuolisen ja joustavan tietojen käsittelyn. Listat ovat järjestettyjä kokoelmia, jotka voivat sisältää mitä tahansa tietotyyppiä, esimerkiksi lukuja, merkkijonoja, muita listoja tai sanakirjoja. Listoissa alkiot (yksittäinen arvo tai tieto tietorakenteessa) on järjestetty tietyssä järjestyksessä ja niitä voidaan indeksoida (numeroida) niiden sijainnin perusteella (Vanhala & Nikula 2022, 62).

Sanakirjat ovat joukkoja, jotka säilyttävät avain-arvo -pareja. Toisin kuin listat, sanakirjat eivät ole järjestettyjä, joten niitä ei voi indeksoida numeroindeksillä. Sen sijaan sanakirjan arvoja päästään käsiksi niiden avainten avulla, eli sanakirjan sisältämät arvot voidaan ikään kuin nimetä. Sanakirjat voivat sisältää listoja, ja listat vastaavasti sanakirjoja (Parker 2021, 296).

```
1  # Määritellään ensimmäinen lista
2  lista_a = ["a_arvo1", "a_arvo2", "a_arvo3"]
3
4  # Määritellään toinen lista
5  lista_b = ["b_arvo1", "b_arvo2", "b_arvo3"]
6
7  # Määritellään sanakirja, joka käyttää listoja a- ja b-arvojen arvoina
8  sanakirja = {"a_arvot": lista_a, "b_arvot": lista_b}
```

Kuva 24. Esimerkkikuva listoista ja sanakirjasta (Miettinen 2023)

Esimerkkiohjelmassa määritetään kaksi kappaletta kolmealkioista listaa, ”lista_a” ja ”lista_b”, ja listat lisätään sanakirjaan ”sanakirja”. Esimerkkinä, jos halutaan hakea yllä olevasta sanakirjasta arvo b_arvo2, voidaan se hakea komennolla ”sanakirja["b_arvot"][1]”. Pythonissa indeksointi alkaa nollasta, joten [1] on todellisuudessa listan toinen alkio.

5.1.5 Paketit, moduulit ja API

Python-paketti on tapa järjestää moduuleja hierarkkisesti kansioihin ja yhdistää samankaltaisia tai toisiinsa liittyviä moduuleja yhteen loogiseen kokonaisuuteen. Paketti koostuu yhdestä tai useammasta moduulista.

Python-moduuli taas on tiedosto, joka sisältää Python-koodia, kuten funktioita, luokkia ja muuttujia. Moduulit tekevät ohjelmakoodista järjestelmällisempää ja helpompaa ylläpitää, koska ne mahdollistavat koodin jaottelun loogisiin osiin. Yksinkertaisesti sanottuna Pythonin paketit ja moduulit ovat valmiiksi tehtyjä ohjelmia ja ohjelmistokokonaisuuksia, joita on helppo hyödyntää osana toista ohjelmaa (Kasurinen 2009, 159).

API (Application Programming Interface), eli sovellusohjelmointirajapinta, on määriteltyjen sääntöjen kokoelma, joka mahdollistaa eri sovellusten välisen viestinnän. Se toimii kuin välittäjä, joka hoitaa datan siirtämisen eri järjestelmien välillä ja antaa käyttäjälle mahdollisuuden jakaa sovellustensa tietoja ja toimintoja muiden ohjelmien kanssa. API:n avulla tietoja voidaan siirtää helposti myös eri ohjelmointikielien välillä (IBM 2022)

5.1.6 Tapahtumakäsittelijät ja samanaikaiset tehtävät

Projektissa käytettiin tapahtumakäsittelijöitä ilmaisemaan ja siirtämään uutta tietoa. Monissa ohjelmointikielissä, kuten Pythonissa, samanaikaisten tehtävien suoritus vaatii huolellista suunnittelua, sillä ohjelmansuorittaja pystyy käsittämään vain yhden pyynnön kerrallaan. Yksinkertaisissa tehtävissä tämä ei aiheuta ongelmia, mutta kun ohjelman tulee esimerkiksi lukea ja vastaanottaa tietoa eri aikoina, tilanne voi muodostua haasteelliseksi.

```
1  # Esimerkki tiedonlähde
2  from esimerkki import tieto
3  import time
4
5  # Määritetään tieto_old tyhjäksi
6  tieto_old = ''
7
8  # Ikuinen silmukka joka tarkastaa muutoksen
9  while True:
10     if tieto != tieto_old:
11         print(f"Uusi tieto on: {tieto}")
12         tieto_old = tieto
13     # Viive, jotta ohjelma ei kuluta liikaa
           prosessorin aikaa
14     time.sleep(1)
```

Kuva 25. Esimerkkikuva (Miettinen 2022)

Yllä oleva esimerkki havainnollistaa ”huonosti” toteutettua tiedon tarkastusmenetelmää, jossa hyödynnetään ikuisen while True-silmukan sisällä tapahtuvaa tarkastusta. Ohjelma hakee tiedon "esimerkki"-moduulista ja tulostaa tekstin aina, kun se havaitsee muutoksen (vertaamalla uutta tietoa vanhaan if-ehtolausessa). Tämän toteutustavan ongelmana on, että ohjelma on jatkuvasti ”while True” silmukan sisällä, eikä se kykene suorittamaan muita tehtäviä silmukan ulkopuolella. Lisäksi time.sleep()-funktion käyttö tarkastusviiveen luomiseen ei ole paras mahdollinen ratkaisu, koska jos tiedot muuttuvat viivettä nopeammin, muutos havaitaan väärässä ajassa. Tämä voi johtaa virheellisiin tuloksiin ja heikentää ohjelman suorituskykyä.

Pythonissa yleisesti rinnakkaissuorituksen mahdollistamiseen käytetty tekniikka on Threading, joka mahdollistaa tehtävien lähettämisen omille säikeilleen. Threading on tehokas tapa suorittaa useita tehtäviä samanaikaisesti, mutta sen käyttö vaatii kokenutta asiantuntemusta virheiden välttämiseksi.

Nopeuttaaksemme projektin etenemistä ratkaisimme nämä ongelmat hyödyntämällä Asyncio-asynkronointia rinnakkaissuorituksen mahdollistamiseksi sekä Events-tapahtumakäsittelijää muutosten seuraamiseen. Asynkronisen ohjelmoinnin avulla ohjelman eri osia voidaan suorittaa ”samanaikaisesti”. Kun yksi tehtävä odottaa jotain (esimerkiksi viestin lähetysviiveen välissä), asynkronointi voi antaa ohjaimen toiselle tehtävälle, jolloin aikaa ei kulu hukkaan. Tämä poikkeaa Pythonin standardikirjaston Time odotusfunktion time.sleep -toiminnasta, joka pysäyttää koko ohjelman suorituksen määritellyksi ajaksi. Asynkronointi sen sijaan mahdollistaa muiden määrättyjen tehtävien suorittamisen "nukkumisen" aikana, mikä tehostaa ohjelman suorituskykyä. Sekä Asyncio että Events löytyivät valmiina Pythonkirjastoina.

```python
import asyncio
from esimerkki import tieto

# Määritetään tieto_old tyhjäksi
tieto_old = ''

# Tämä asynkroninen funktio tarkastaa muutoksen
async def tarkasta_muutos():
    global tieto_old
    while True:
        if tieto != tieto_old:
            print(f"Uusi tieto on: {tieto}")
            tieto_old = tieto
        # Odotetaan sekunti ennen seuraavaa tarkastusta
        await asyncio.sleep(1)

# Tämä asynkroninen funktio tekee jotain muuta
async def tee_muuta():
    while True:
        print("Teen muuta samalla!")
        await asyncio.sleep(0.1)

# Luo tehtävä tarkasta_muutos()-funktiosta ja tee_muuta-funktiosta, ja suorita ne
asyncio.run(asyncio.gather(tarkasta_muutos(), tee_muuta()))
```

Kuva 26. Esimerkkikuva (Miettinen 2022)

Yllä olevaan esimerkki toteuttaa saman kuin aikaisempi esimerkki, mutta hyödyntäen Asyncio-kirjastoa. Ohjelma toteuttaa saman ”huonon” tiedonhaun kuin edellinen esimerkki, mutta lisäksi se suorittaa rinnalla tee_muuta -funktion, jossa nimensä mukaisesti voitaisiin suorittaa muita haluttuja tehtäviä samanaikaisesti.

Tiedon hakemista voidaan yksinkertaistaa hyödyntämällä esimerkiksi Events-tapahtumakäsittelijää. Määrittelemällä tietoEvent -objekti esimerkki -moduulissa, ja lisäämällä se tapahtumiin, voimme kuunnella sitä helposti muissa moduuleissa. Alla olevassa esimerkissä vastaanotettava tieto tulostetaan aina, kun tietoEvent -tapahtuma päivittyy.

```
1  import asyncio
2  import esimerkki
3
4  # Tämä funktio ilmoittaa muutoksesta Eventsin kautta
5  def on_tietoEvent(tieto):
6      print(f"Uusi tieto on: {tieto}")
7
8  # Tämä asynkroninen funktio tekee samalla muuta
9  async def tee_muuta():
10     while True:
11         print("Teen muuta samalla!")
12         await asyncio.sleep(0.1)
13
14 # Liitetään tietoEventsiin funktio, joka suoritetaan, kun tieto muuttuu
15 esimerkki.tietoEvent.on_change += on_tietoEvent
16
17 # Luodaan tehtävä tee_muuta-funktiosta ja suoritetaan se
18 asyncio.run(tee_muuta())
```

Kuva 27. Esimerkkikuva (Miettinen 2022)

Yllä oleva esimerkki on selvästi yksinkertaisempi kuin edellä mainittu, sillä sen tiedontarkastusfunktio on korvattu tapahtumakäsittelijäfunktiolla, joka suoritetaan aina kun tieto määritetään muuttuneeksi esimerkki -moduulissa. Tapahtumakäsittelijä vaatii hieman enemmän kirjoitusta esimerkki -moduuliin, mutta sen hyödyntäminen muissa moduuleissa vaatii vain muutaman rivin ohjelmakoodia.

5.2 Tiedonsiirto

Tiedonsiirto laitteiden välillä tuli projektissa tapahtua verkkoyhteyden välityksellä kaksisuuntaisesti, mahdollisimman pienellä viiveellä. Laitteiden yhdistäminen toisiinsa tuli olla mahdollisimman suoraviivaista, ja käytön helpottamiseksi mieluiten täysin automaattista.

Projektin alussa ideana oli käyttää Socket-ohjelmointirajapintaa (API), sillä sen parissa oli runsaasti esimerkkejä ja tutoriaaleja. Kuitenkin lähes heti alussa havaittiin, että Socketin käyttöön liittyi merkittäviä heikkouksia. Esimerkiksi yhteyksien portittamisvaatimukset sekä staattisten IP-osoitteiden käyttö aiheuttivat ongelmia.

Tämän takia kokeiltiin tiedonsiirtojärjestelmänä myös MQTT-protokollaa, joka tarjosi monia etuja verrattuna Socketiin. MQTT mahdollisti helpon yhdistämisen eri laitteiden välillä, useampien aihekanakanavien käytön sekä monisuuntaisen tiedonsiirron.

Socket-pohjaiset ratkaisut toimivat paikallisverkossa, mutta MQTT:n käyttö vaati lisäksi yhteyden MQTT-palvelimeen, mikä onneksi löytyi jo ennestään koululta. Lopulta valinnaksi päätyi siirtyä käyttämään MQTT-tiedonsiirtojärjestelmä, sillä se tarjosi paremman ratkaisun projektin vaatimuksiin ja mahdollisti tehokkaamman kommunikaation eri laitteiden välillä.

5.2.1 MQTT-Palvelin

MQTT (Message Queuing Telemetry Transport) on kevyt viestintäprotokolla, joka on suunniteltu erityisesti Internet of Things (IoT) -laitteiden väliseen tiedonsiirtoon (MQTT.org 2022).

MQTT-protokollan toimintaa voidaan havainnollistaa vertaamalla sitä ikään kuin IoT-laitteiden WhatsApp-ryhmäkeskusteluun:

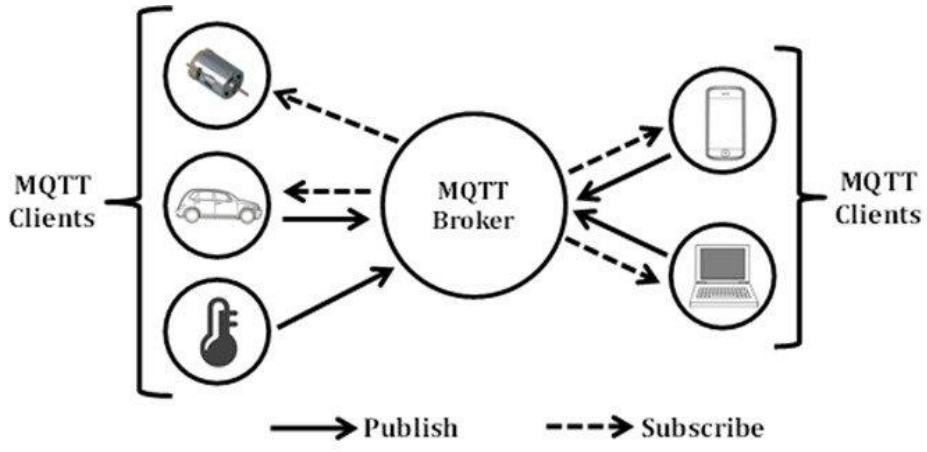
-MQTT-palvelin (Broker) toimii välittäjänä IoT-laitteiden välillä. Laitteet voivat lähettää viestejä palvelimelle ja vastaanottaa viestejä muilta laitteilta internetyhteyden välityksellä.

-Kun laite haluaa lähettää viestin, se julkaisee sen palvelimelle. Julkaisu tapahtuu aiheen avulla. Aihe on yksinkertaisesti merkkijono, joka kuvaa viestin sisältöä, esimerkiksi "Lämpötila".

-Palvelin vastaanottaa viestin ja tallentaa sen jonoon. Palvelin lähettää sitten viestin kaikille laitteille, jotka ovat tilanneet saman aihekanavan.

-Laitteet voivat tilata aihekanavia, joiden viestit kiinnostavat niitä. Kun uusi viesti tulee, palvelin lähettää sen kaikille tilaajille.

-Kun laite vastaanottaa viestin, se voi käsitellä sen ja suorittaa tarvittavat toimet.

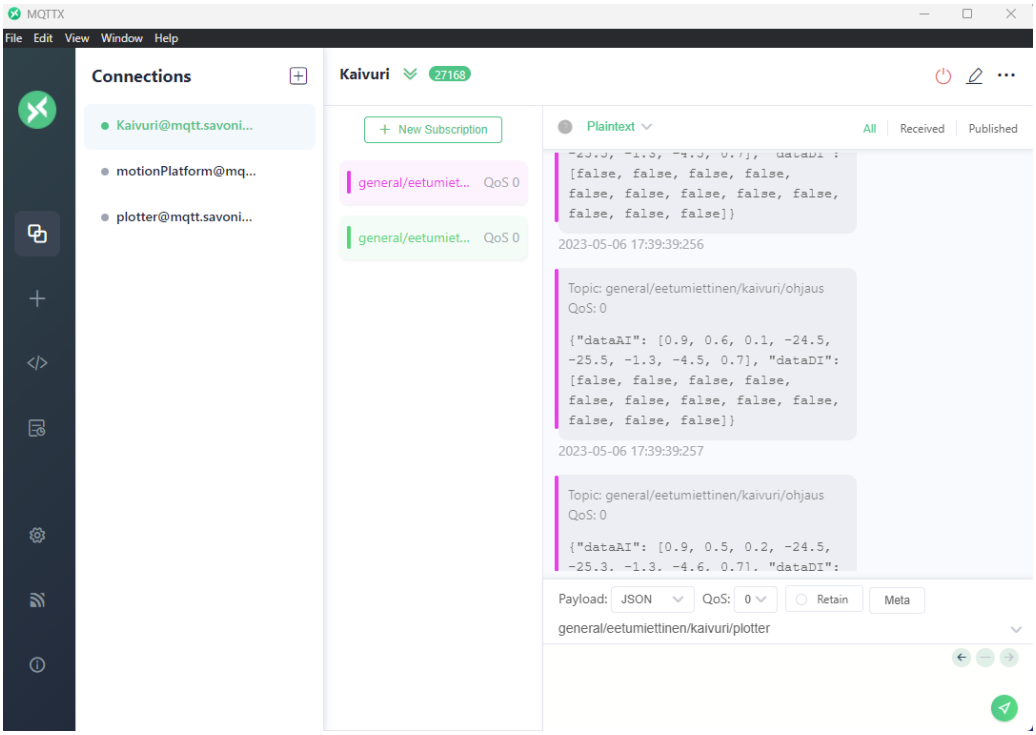


Kuva 28. Havainnekuva (Akintade & Yesufy & Kehinde 2019)

Savonialta jo ennestään löytyvää MQTT-palvelinta hyödynnettiin projektissa. Laitteille "Motion platform (ohjainlaite)" ja "Kaivuri" luotiin omat käyttäjätunnukset, joiden avulla niiden oli mahdollista kommunikoida MQTT-palvelimen kautta. Lisäksi palvelimelle määriteltiin aihekanavat "plotter" ja "ohjaus", joiden avulla laitteiden välinen viestintä järjestettiin.

Aihekanava "plotter" oli tarkoitettu graafisen käyttöliittymän ja mittauslaitteiden väliseen viestintään. Tämän kanavan avulla voitiin välittää kaivurin hydraulikanavien painetietoa reaaliajassa. Painetieto pystyttiin lukemaan aihekanavalta ja näyttämään käyttäjälle havainnollisessa muodossa.

Toinen aihekanava, "ohjaus", keskittyi kaivurin ohjauskomentojen välittämiseen. Tämän kanavan avulla käyttäjän lähettämät ohjaustiedot, kuten puomin kääntö, nostaminen tai pumpun nopeuden muutos, voitiin siirtää laitteille. Kaivuri pystyi vastaanottamaan nämä komentosignaalit ja suorittamaan niiden perusteella tarvittavat toimenpiteet.



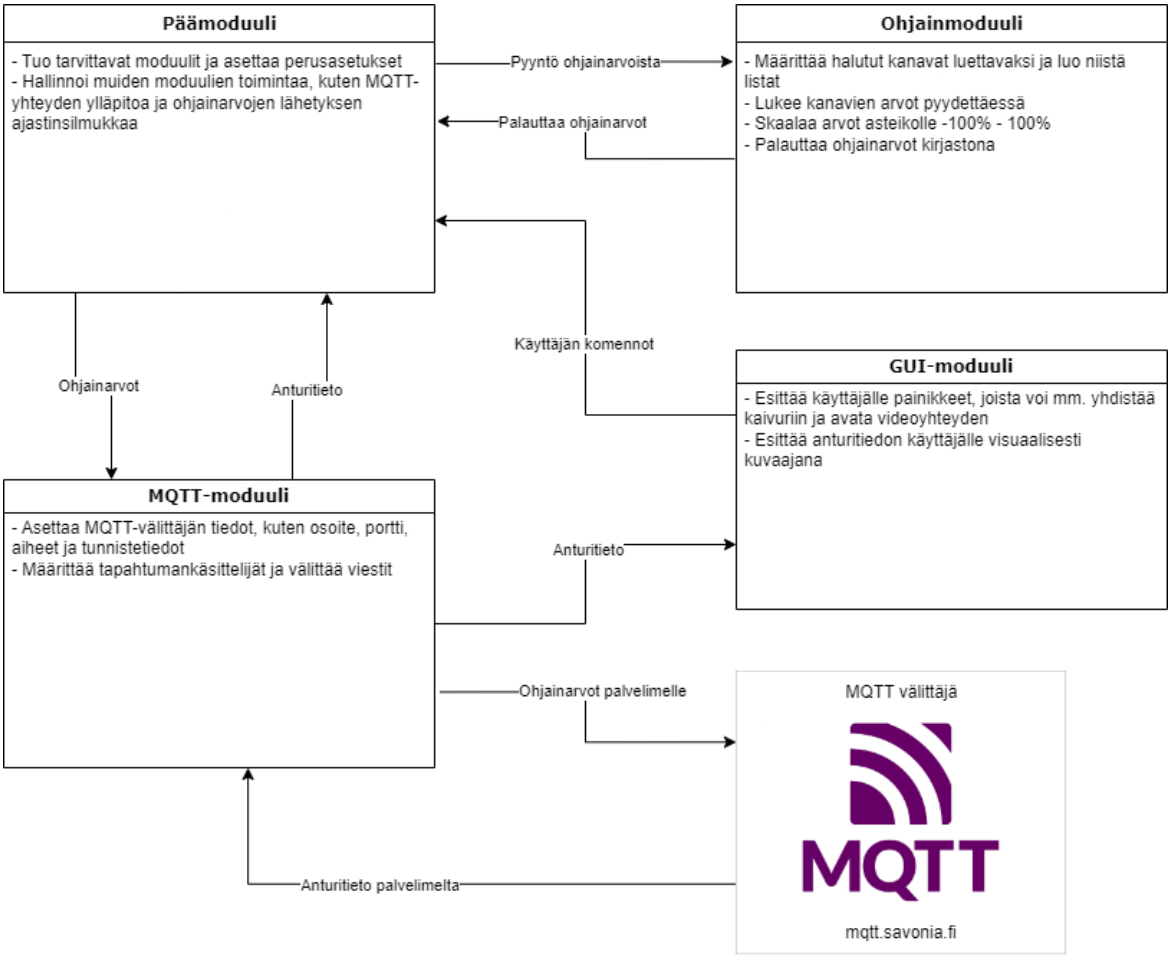
Kuva 29. MQTT-palvelimen kokeilemista MQTTX-sovelluksella (Miettinen 2022)

6 KAIVURIN OHJELMOINTI

Projektissa ohjelmat päätettiin tehdä eri moduuleihin, jotta niiden lukeminen olisi helppoa ja jotta niitä voitaisiin hyödyntää helposti myös muissa ohjelmissa tulevaisuudessa. Myös virheen sattuessa se vaikuttaisi vain yhteen moduuliin, joten virheen löytäminen oli helpompaa. Tämä mahdollisti myös toisien moduulien nopean vaihtamisen/lisäämisen muiden moduulien lisäksi, joten esimerkiksi ohjainlaitteelle kirjoitettu MQTT-moduuli oli hyvin nopeasti käännettävissä toimimaan myös kaivurin puolella.

Alla oleva kaaviokuva havainnollistaa ohjainlaitteen ohjelmamoduulien toimintaa ja niiden välisiä vuorovaikutuksia.

6.1 Ohjainlaitteen ohjelmointi



Kuva 30. Ohjainlaitteen ohjelman havainnollistava kaaviokuva (Miettinen 2022)

6.1.1 Päämoduuli

Päämoduuli oli vastuussa muiden moduulien käytöstä sekä lähetysviiveen määrittämisessä. Viive toteutettiin Asyncio-asynkronoinnin avulla, jotta moduuli pystyi hallitsemaan muita tehtäviä samanaikaisesti. Testailujen jälkeen noin 10 millisekunnin viive ohjaintiedon välillä osoittautui toimivaksi, ei liian nopeaksi ja järjestelmää rasittavaksi eikä myöskään liian hitaaksi, käytössä pätkivän tuntuiseksi.

Lisäksi moduuliin tehtiin yksinkertainen asynkroninen funktio, joka laski lähetyt ohjausviestit tietyltä aikaväliltä, ja tulosti summan. Tällä voitiin karkeasti tarkkailla lähetysnopeutta. Myös vastaanotettu painetieto tulostettiin päämoduulissa tapahtumakäsittelijän avulla, lähinnä kokeiluna.

Molemmat funktiot toimivat äärettömässä while True-silmukassa. Tulevaisuudessa silmukan True-ehto voitaisiin vaihtaa esimerkiksi ehtolauseeseen, jos funktioiden silmukan toimintaa haluttaisiin ohjata.

6.1.2 Ohjainmoduuli

Ojainmoduuli oli vastuussa ohjauslaitteiden arvojen lukemisesta. Arvot luettiin 2DOF-laitteesta NI-DAQmx API:n avulla, joka keskusteli 2DOF-laitteeseen kytketyn National Instruments USB I/O-lukulaitteen kanssa. Ohjelman kirjoituksen helpottamiseksi luettaviksi halutuista kanavista luotiin lista, jota hyödynnettiin for-silmukan kanssa. Silmukka kävi läpi kaikki listan kanavat, ja lisäsi ne NI-DAQmx API:n määrityksiin. Samaa menetelmää käytettiin myös ohjainlaitteen painokytkimien määrittämisessä. Tämän ansiosta ohjelma pidettiin siistinä, ja toistavalta ohjelman kirjoittamiselta vältyttiin.

Ohjainsauvoilta saatu signaali oli alueella 0,5-4,5, ja se päätettiin muuttaa selkeämpään muotoon, jotta sen suhteellinen arvo olisi helpommin havaittavissa (vertaa 3,94 = 72%). Muutos tehtiin myös sen varmistamiseksi, että arvot vastaisivat paremmin esimerkiksi peliohjaimilta saatavia arvoja, mikäli sellaista haluttaisiin käyttää Motion Platformin sijasta. Muunnos suoritettiin kaavalla

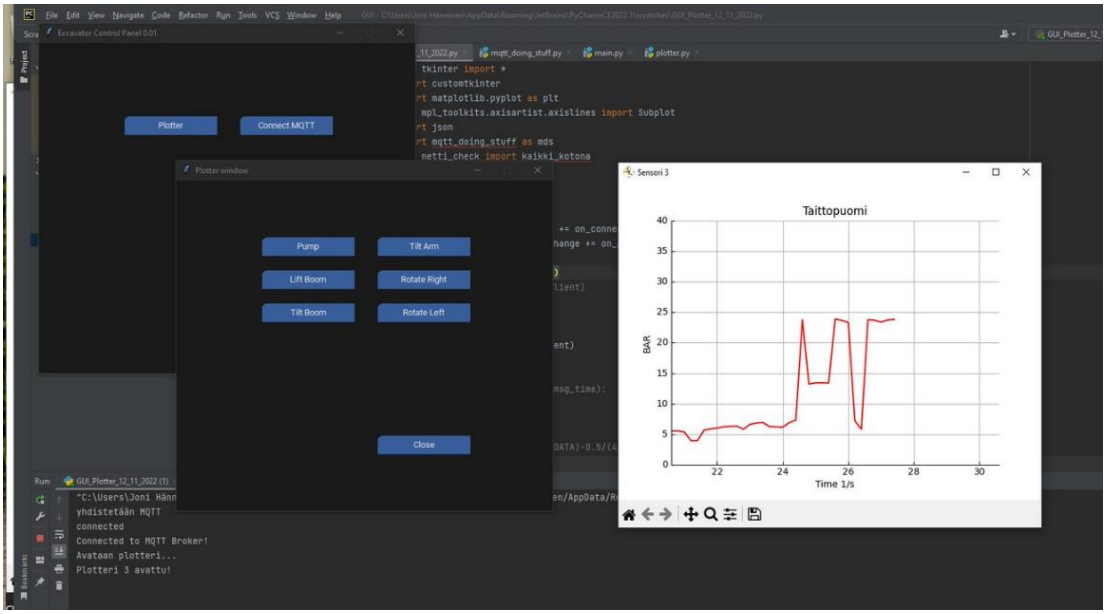
$$Arvo = (SauvanArvo - 2{,}5) * 50 \qquad (1)$$

jossa "SauvanArvo" on sisään tuleva sauvan arvo, "2,5" on sauvan arvon keskikohta ja "50" on kerroin. Näin arvot saatiin toimimaan välillä -100 - +100(%). Muunnos suoritettiin for-silmukan avulla, joten se käsitteli kaikki muut kuin totuusarvoiksi (TRUE/FALSE) määritellyt kanavat. Muutetut arvot pyöristettiin yhden desimaalin tarkkuuteen, mikä selkeytti testausta sekä tarkastelua. Ohjainlaitteen painokytkimien arvot olivat suoraan totuusarvoina, joten niiden osalta muutoksia ei tarvinnut tehdä.

Lopuksi moduuli keräsi ohjainsauvojen sekä painokytkimien arvot omiksi listoiksi alussa määritellyn perusteella, jotka sitten lisättiin yhteen sanakirjaan. Sanakirja muutettiin ennen palautusta JSON-muotoon. Näin kaikki ohjaustiedot saatiin helposti siirrettyä kerralla, selkeästi luettavassa muodossa.

6.1.3 GUI-moduuli

Ohjainlaitteen GUI-moduulia käytettiin esittämään kaivurilta tuleva tieto graafisessa muodossa. Moduuliin oli luotu valikot, joista pääsi avaamaan halutun kanavan kuvaajan, joka päivittyi reaaliajassa. Siihen määritettiin painikkeet videostriimin avaamiseen sekä MQTT-palvelimeen yhdistämistä varten. GUI-moduulin ohjelmointi oli kanssaopiskelijan käsialaa, joten siihen ei syvennytä tässä opinnäytetyössä tämän tarkemmin.



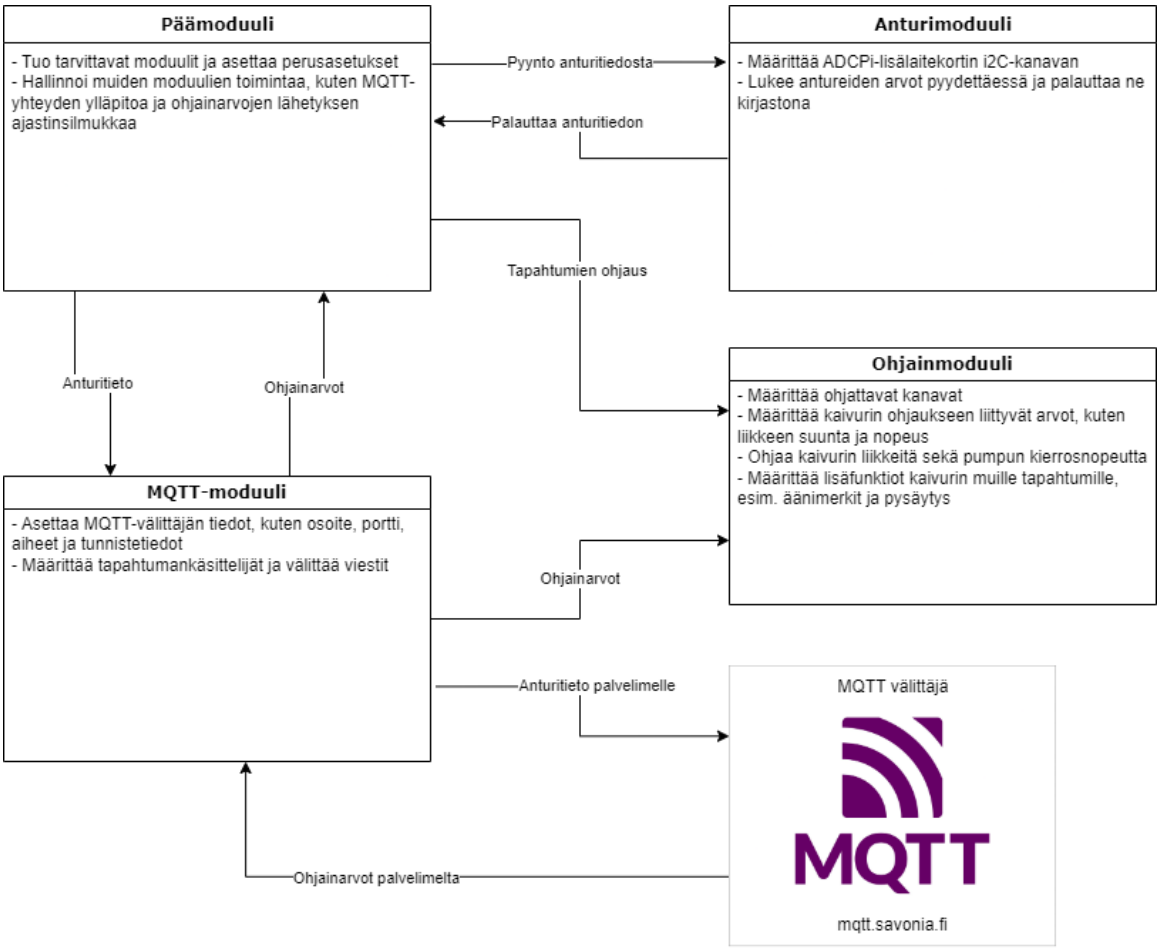
Kuva 31. GUI-moduulin valikot sekä kanavan ”Taittopuomi” kuvaaja (Hänninen 2022)

6.1.4 MQTT-moduuli

MQTT-moduuli oli vastuussa yhteyden määrittämisestä sekä tietojen välittämisestä. Moduuliin määritettiin halutut aihekanavat, sekä käyttäjätunnukset ja salasanat. Moduuliin tehtiin funktiot yhdistämiselle, lähettämiselle ja vastaanottamiselle. Käsitelty tieto sekä tapahtumat välitettiin muille moduuleille MQTT-moduulissa määritetyn tapahtumakäsittelijän avulla. Tällä mahdollistettiin tiedon helppo välittäminen.

MQTT-moduuli sisälsi myös tarkastusehdot funktioiden tapahtumille. Jos esimerkiksi viestiä ei kyetty lähettämään, tulosti moduuli virheviestin käyttäjälle.

6.2 Kaivurin ohjelmointi



Kuva 32. Kaivurin ohjelman havainnollistava kaaviokuva (Miettinen 2022)

Kaivurin moduulirakenne toteutettiin samaan tapaan kuin ohjainlaitteen rakenne, ja sen luomiseen käytettiin pohjina ohjainlaitteen moduuleja.

Kaivurissa valmis ohjelmisto määritettiin käynnistymään automaattisesti, kun kaivuriin kytkettiin virrat. Tähän käytettiin Raspberry Pi:n oman käyttöjärjestelmän palvelua Systemd. Ohjelman käynnistykseen lisättiin ehto, joka antaa suorittaa ohjelman vasta, kun kaivuri on yhdistetty verkkoon. Täten kaivurin käyttöönotto ei vaatisi muuta kuin akun kiinnittämisen ja virtakytkimen painamisen. Myös videokuvan ohjelmisto määritettiin käynnistymään samoin ehdoin.

6.2.1 Päämoduuli

Kaivurin päämoduuli toimi hyvin samankaltaisesti 2DOF-päämoduulin tavoin. Muutoksena moduuliin lisättiin funktio, joka laski vastaanotetut signaalit määritellyltä aikaväliltä, ja lähetti pysäytyskäskyn ohjainmoduulille ohjaussignaalien määrän pudotessa alle raja-arvon määrittämän ehdon.

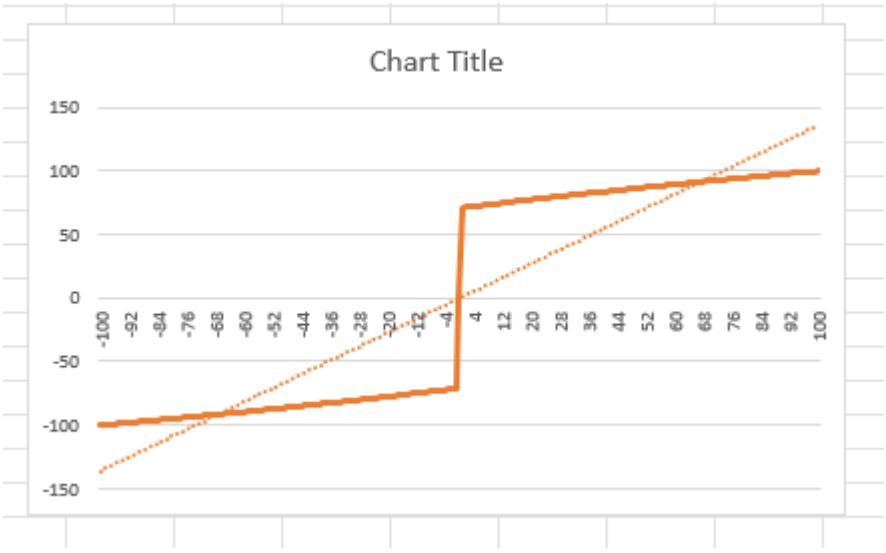
6.2.2 Ohjainmoduuli

Ohjainmoduuli vastasi kaivurin liikkeiden hallitsemisesta. Se määritti asetusarvot PWM-lisälaitekortille, jota hyödynnettiin signaalin tuottamisessa kaivurin servomoottoreita varten. Ohjelmaan luotiin useita funktioita eri käyttötarkoituksia varten; Ajofunktio varsinaisten ajonaikaisten liikkeiden toteuttamiseen, nollausfunktio kaivurin liikkeiden pysäyttämiseen esimerkiksi virhetilanteessa, ja äänimerkkifunktio ilmoittamaan käyttäjälle kaivurin tilasta, esimerkiksi katkenneesta yhteydestä.

Äänimerkkifunktio oli toteutettu kaivurin telojen sähkömoottoreiden avulla, sillä ne pitivät hyvin selkeää ”piippaavaa” ääntä niiden ollessa käytössä. Ääni kuului myös silloin, kun moottoreille syötettiin niin matalan liikkeen ohjauskäsky, että moottorit eivät todellisuudessa liikkuneet ollenkaan. Tämä huomattiin helpoksi tavaksi viestiä käyttäjälle mahdollisista kaivurin ongelmista.

Nollausfunktio oli alun perin kirjoitettu hyvin manuaalisesti, jossa jokainen nollattava kanava oli erikseen määritetty, ja niiden arvot määritettiin yksitellen asetusarvoksi funktion suorituksen aikana. Tämä aiheutti helposti ongelmia, sillä käytettyjä kanavia muokatessa tuli myös muistaa muokata nollausfunktioita. Koska kanavat olivat jo alussa nimetty toimintonsa mukaan, päätettiin nollausfunktio muokata viisaammaksi. Tämä toteutettiin for-silmukalla, joka käy läpi kaikki määritellyt kanavat, ja asettaa niiden arvot asetusarvoiksi. Silmukan sisään oli täten helppo lisätä totuusehtoja, jotka tarkastivat käsiteltävän kanavan nimen, ja muuttivat arvon kyseisen kanavan asetusarvoksi. Tämä mahdollisti sen, että uusia kanavia lisätessä käyttäjän ei tarvinnut huolehtia kanavan nollaamiskäskyn kirjoittamisesta, ellei käyttäjä erikseen halunnut muuttaa asetusarvoa johonkin vakiosta poikkeavaan.

Ajofunktiossa ohjainlaitteelta vastaanotetut arvot olivat lineaarisessa muodossa ohjainsauvan liikkeen suhteen. Lineaarinen muoto ei kuitenkaan vaatinut ylimääräistä koodia ohjelman toteuttamiseen, joten sitä käytettiin vielä tässä vaiheessa projektia.



Kuva 33. Ohjaussauvan arvon kokeiltu muutoskäyrä (Miettinen 2022)

Yllä näkyvä ohjainsauvan arvokäyrä esittää sauvojen ääriarvojen välillä olevan muutoksen. Käyrä nostaa ohjainsauvan arvot välittömästi siihen kohtaan, jossa venttiilin kääntö alkaa vaikuttamaan hydraulinesteen virtaukseen. Tällä olisi haettu parannettua tarkkuutta hydraulisylinterien ohjaamiseen, sillä suuri osa ohjaussauvan liikkeestä asettuisi venttiilin aktiiviselle alueelle. Ajankäytöllisistä syistä muutoskäyrä jätettiin kuitenkin pois projektista vielä tässä vaiheessa.

Ohjainsauvojen arvot syötettiin servoille seuraavasti:

$$ServoKulma = keskikohta + SauvanArvo * kerroin \quad (2)$$

Missä keskikohta tarkoittaa servomoottorin lepokulmaa, SauvanArvo on ohjainlaitteelta saatava arvo ja herkkyyskerroin määrittelee servon kääntymisen herkkyyden. Servomoottorin toimintasuuntaa voitiin muuttaa vaihtamalla plusmerkki miinusmerkiksi.

Venttiilien mahdollinen nopea edestakainen liike lähellä nollakohtaa ei ollut ongelma, sillä ajofunktioon oli kirjoitettu ehdoksi kuollut kulma. Ohjausarvot eivät siis muuttuneet, jos ohjainlaitteelta tulleet arvot eivät ylittäneet tiettyä rajapistettä. Tämän lisäksi ohjelma piti lukua myös raja-arvot ylittäneiden kanavien lukumäärästä, jonka perusteella hydraulipumpun kierrosnopeutta voitiin muuttaa dynaamisesti tarpeen mukaan. Enemmän aktiivisia kanavia, suurempi pumpun kierrosnopeus. Pumpun kierrosluvun kerroin oli myös muutettavissa ajon aikana, joten kaivuria voitiin käyttää myös siten, että aktiivisten kanavien lukumäärän ollessa nolla pumppu ei pyörinyt. Laskuri huomioi vain hydrauliikkaan vaikuttavat kanavat, joten esimerkiksi teloilla ajo ei vaikuttanut pumpun kierrosnopeuteen.

Ohjainlaitteen painokytkimiä varten luotiin liuta totuusehtoja, joita voitiin hyödyntää erilaisissa tapahtumissa. Esimerkiksi vasemman (2DOF) ojainsauvan kytkimiä käytettiin pumpun kierroslukukertoimen säätämiseen, ja oikean sauvan kytkimiä käytettiin kameran kääntämiseen. Muut vielä käyttämättömiksi jääneet kytkimet määritettiin tulostamaan painetun kytkimen kanava käyttäjälle.

6.2.3 Anturimoduuli

Kaivurin anturimoduuli oli vastuussa paineantureiden arvojen käsittelystä. Arvot luettiin ADCPi-lisälaitekortilla. Projektin aikana huomattiin lisälaitekortin I2C-kanavan vaihtuvan aika-ajoin kahden vaihtoehdon välillä tuntemattomasta syystä, joten ohjelmaan lisättiin tarkastusehto, joka kokeilee molemmat kanavat ja valitsee käyttöön toimivan vaihtoehdon. Molempien kanavien ollessa virheellinen ohjelma palauttaa virheilmoituksen käyttäjälle.

Tämän jälkeen ohjelman tiedonkeräysfunktiota kutsuttaessa se kerää tiedon halutuista kanavaista, tekee tiedoista listan, ja palauttaa sen muiden moduulien käyttöön JSON-muodossa.

6.2.4 MQTT-moduuli

Kaivurin MQTT-moduuli toimi käytännössä identtisesti ojainlaitteen MQTT-moduulin kanssa. Ainoina muutoksina oli käyttäjätunnukset sekä lähettävä/vastaanottava kanava.

7 YHTEENVETO

7.1 Saavutetut tulokset

Projektin aikana saavutettiin ja osittain jopa ylitettiin alussa asetetut vaatimukset. Kaivurin ohjaus toteutettiin Motion Platformin ohjauslaitteistolla, joten ohjaus toimii oikean kaivinkoneen tavoin. Tiedonsiirto saatiin sujuvaksi ja tapahtumaan hyvin pienellä viiveellä, mikä tekee kaivurin käytöstä entistä realistisempaa ja miellyttävämpää. Lukuisten ”Quality of Life”-lisäysten avulla kaivurin käyttö ei tunnu niin prototyyppimäiseltä, vaan sen käyttöönotto oli hyvinkin suoraviivaista. Myös videoyhteys saatiin toimimaan välttävästi, joten kaivua oli mahdollista ajaa ilman suoraa näköyhteyttä.

Ohjelmoinnin osalta eri moduuleista saatiin selkeitä ja helposti jatkokehityksessä hyödynnettäviä. Moduuleista saatiin suhteellisen helppolukuisia, ja ne pyrittiin kommentoimaan mahdollisimman selkeästi, jotta niiden tarkoitus ja toimintatapa olisi selkeä. Myös opinnäytetyön kirjoittajan kokemus ohjelmoinnista kehittyi vahvasti, se lähinnä henkilökohtaisena saavutuksena.

Kuva 34. Kaivurin videoyhteyden testaamista (Hänninen 2022)

Kuva 35. Kaivuri esillä alihankintamessuilla (Miettinen 2022)

7.2 Turvallisuuden arviointi

Vaikka kyseessä onkin ainoastaan pienoismallikaivuri, on sillä silti potentiaalia aiheuttaa merkittävää vahinkoa ympäröiville esineille tai pahimmassa tapauksessa jopa henkilövahinkoja. Myös kaivurin käyttämien litiumpolymeeriakkujen kanssa tulisi noudattaa erityistä varovaisuutta, sillä väärin käsiteltynä ne aiheuttavat vakavan tulipaloriskin.

Tästä syystä laadittiin perusteellinen riskienarviointitaulukko auttamaan tunnistamaan ja hallitsemaan mahdollisia vaaroja sekä uhkia. Taulukon avulla voidaan huomioida ja käyttää asianmukaisia turvatoimia sekä -käytäntöjä riskien hallitsemiseksi. Taulukko löytyy liitteenä opinnäytetyön lopusta.

7.3 Haasteet ja oppimiskokemukset

Kaivurin mekaanisen puolen muutostyöt sujuivat nopeasti ilman ongelmia, mutta uuden ohjelmointikielen, Pythonin, omaksuminen sekä tehokkaan ja laadukkaan ohjelmakoodin luominen tuotti ongelmia. Ennen projektin aloittamista minulla ei ollut aikaisempaa kokemusta Python-ohjelmoinnista, joten otin projektin vastaan suurena haasteena ja oppimiskokemuksena.

Projektin alkuvaiheessa lähestymistapani ohjelmoinnin suhteen oli liian aggressiivinen. Tämä tarkoitti, että kirjoitin ohjelmaa liian nopeasti ja ilman riittävää suunnittelua. Tämän seurauksena jouduin kirjoittamaan useita ohjelman osia uudelleen myöhemmin projektin aikana, kun huomasin, että ne eivät vastanneet odotuksiani tai olivat muuten puutteellisia. Tämä aggressiivinen lähestymistapa johtui osittain siitä, että en vielä täysin ymmärtänyt Pythonin periaatteita tai parhaita käytäntöjä.

7.4 Projektin jatkokehitys

MaSi-hankkeen jatkuessa tavoitteena on kehittää kaivurista älykäs robotti, jota voitaisiin hyödyntää myös Mevea-simulointiympäristön tarkan mallinnuksen kanssa. Tätä varten kaivuriin lisättäisiin useita antureita, kuten esimerkiksi kiihtyvyysanturi, öljyn lämpötila-anturi ja pumpun kierrosnopeusanturi. Projektissa hyödynnettäisiin myös 5G-teknologiaa viestintä- ja tiedonsiirtoratkaisuissa, mikä mahdollistaisi suuremman etäisyyden käyttäjän ja kaivurin välillä.

Kaivurin ajokokemusta 2DOF -ohjauslaitteella voitaisiin parantaa esimerkiksi 360-asteisten kameroiden ja virtuaalitodellisuuden avulla. Käyttäjä voisi ohjata kaivuria etänä VR-laseilla, joihin voitaisiin integroida Heads-Up Display näyttämään kaiken tarpeellisen tiedon koneen toiminnasta, kuten kabiinin asennon suhteessa telastoon sekä akun varaustason. Myös Motion Platformin kaksiakselista liikettä voitaisiin käyttää kaivurin asentojen simuloimisessa.

Lisäksi opiskelijat voisivat suunnitella itse tehtyjä osia kaivuriin, kuten topologiaoptimoidut nostovarret tai vaihtoehtoisesti pyörät telaketjujen tilalle, parantaen koneen toiminnallisuutta ja tehokkuutta.

Yhdistämällä kaikki edellä mainitut ominaisuudet kaivurin käyttöön voitaisiin luoda testiympäristö, jossa etäohjattavaa kaivuria voitaisiin käyttää tekoälyn avustuksella suorittamaan erilaisia tehtäviä opiskelijoiden suunnittelemilla vaihdettavilla työkaluilla.

Ohjelmoinnin puolesta jatkokehitystä olisi vielä paljon, sillä kokemukseni ohjelmoinnista kehittyy koko ajan. Esimerkiksi tapahtumakäsittelijän tapahtumista kannattaisi luoda oma moduuli, jonka kautta tieto välitettäisiin muihin päämoduuleihin. Tällä saataisiin huomattavasti yksinkertaistettua uusien moduulien luomista ja tapahtumakäsittelijä olisi helpommin hyödynnettävissä. Asynkronoitu käyttö olisi parempi muokata säikeistetyksi (threading). Myös käyttäjätietojen syöttö olisi järkevämpi tapahtua päämoduulin kautta, jotta varsinaista MQTT-moduulia ei tarvitsisi muokata tietojen muuttuessa.

8 LOPPUSANAT

Tämän opinnäytetyön myötä olemme saavuttaneet ja ylittäneet alkuperäiset kaivuriprojektille asetetut tavoitteet. Vaikka projekti osoittautui ajankäytöllisesti haastavaksi, erityisesti Python-ohjelmoinnin opettelun vuoksi, onnistuimme kuitenkin saavuttamaan tavoitteemme ja pääsimme oppimaan paljon uutta. Kokemattomuudestamme huolimatta pystyimme kehittämään taitojamme ohjelmoinnissa ja ymmärtämään paremmin muun muassa olio-ohjelmoinnin sielunmaisemaa käytännön projekteissa. Tämä opinnäytetyö on osoittautunut arvokkaaksi oppimiskokemukseksi, joka on tuonut esille uusia näkökulmia ja mahdollisuuksia.

Projektin onnistuminen osoittaa, että olen kyennyt toimimaan tehokkaasti ja joustavasti projektin tekijänä. Työn tulokset tarjoavat hyvän pohjan jatkojalostukselle, ja olen varma, että tämän projektin myötä saadut kokemukset sekä itseluottamus omaan osaamiseen tulevat olemaan hyödyksi tulevissa projekteissa ja työtehtävissä.

Lopuksi haluan kiittää kaikkia, jotka ovat tukeneet ja auttaneet minua tämän opinnäytetyön toteuttamisessa. On ollut innostavaa ja opettavaista työskennellä tämän projektin parissa, ja toivon, että se herättää kiinnostusta ja inspiroi muita vastaavien projektien kehittämiseen.

LÄHTEET

Büchi, Roland 2014. Radio Control with 2.4ghz. E-kirja. Norderstedt: Books on Demand. Viitattu 22.5.2023.

CircuitBread 2022. What is a PWM signal? Verkkojulkaisu. https://www.circuitbread.com/ee-faq/what-is-a-pwm-signal. Viitattu 10.7.2022.

ElectronicsTutorials 2022. Analogue to Digital Converter. Verkkojulkaisu. https://www.electronics-tutorials.ws/combination/analogue-to-digital-converter.html. Viitattu 20.6.2022.

IBM 2022. What is an API? Verkkojulkaisu. https://www.ibm.com/topics/api. Viitattu 20.5.2022.

IBM 2022. What is an Digital Twin? Verkkojulkaisu. https://www.ibm.com/topics/what-is-a-digital-twin. Viitattu 24.6.2022.

Kasurinen, Jussi Pekka 2019. Python 3 -ohjelmointi. 1. painos. Jyväskylä: WSOYpro/Docendo

Kauranne, Heikki & Kajaste, Jyrki & Vilenius, Matti 2013. Hydraulitekniikka. 2., uudistettu painos. Helsinki: Sanoma Pro Oy

Keinänen, Toimi & Kärkkäinen, Pentti 2005. Automaatiojärjestelmien hydrauliikka ja peumatiikka. 1. painos. Helsinki: WSOY.

Mevea 2022. About. Verkkojulkaisu. https://mevea.com/about/. Viitattu 15.5.2022.

MQTT.org 2022. About MQTT. Verkkojulkaisu. https://mqtt.org/faq/. Viitattu 15.5.2022.

National Instruments 2022. Using a NI DAQ Device with Python and NI DAQmx. Verkkojulkaisu. https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z0000019Pf1SAE. Viitattu 1.5.2022.

Parker, James 2021. Python: An Introduction to Programming. Second Edition. E-kirja. Dulles: Mercury Learning and Information. Viitattu 24.6.2023.

Peltomäki, Juha 2023. Python 3 -ohjelmoinnin perusteet. Tuntematon painos. Helsinki: Books on Demand. Viitattu 3.5.2023.

Pypi.org 2021. Paho-mqtt 1.6.1. Verkkojulkaisu. https://pypi.org/project/paho-mqtt/. Viitattu 4.5.2022.

Python Software Foundation 2022. General Python FAQ. Verkkojulkaisu. https://docs.python.org/3/faq/general.html#what-is-python. Viitattu 3.6.2022.

Raspberry 2022. Introducing Raspberry Pi HATs. Verkkojulkaisu. https://www.raspberrypi.com/news/introducing-raspberry-pi-hats/. Viitattu 10.6.2022.

Raspberry 2022. Raspberry Pi 4. Esite. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/. Viitattu 3.4.2022.

Sandvik RockTechnology 2022. AutoMine laiteautomaatio ja etäkäyttö. Esite. https://www.rocktechnology.sandvik/fi/laitteet/automaatio/automine-laitteistoautomaatio-ja-et%C3%A4k%C3%A4ytt%C3%B6/. Viitattu 25.6.2022.

Vanhala, Erno & Nikula, Uolevi 2022. Python 3 -ohjelmointiopas. E-kirja. Lappeenranta: LUT-yliopisto. Viitattu 10.5.2023.

LIITE 1: KAIVURIN RISKIENARVIOINTILOMAKE

Riski	Todennäköisyys (1-3)	Syy	Riskiluokitus (1-3)	Toimenpide
Puristumisvaara	2	Kaivurin käsittely sen ollessa käynnissä	2	Kaivuria käsiteltävä vain virtojen ollessa poiskytkettynä
Puomin aiheuttama vahinko	3	Käyttäjän kokemattomuus	1	Huomioidaan puomin ulottuvuus ajon aikana, varmistetaan riittävästi tilaa ympärillä. Kamerakuvalla ajettaessa toinen kenkilö tarkkailemaan kaivuria.
Materiaalivahinko	2	Kaivuri lakkaa vastaamasta ohjainlaitteeseen ja esimerkiksi putoaa pöydältä	1	Varmistetaan, että kaivurin käyttöympäristö minimoi vahingot karkauksen sattuessa
Roiskevaara	1	Kaivurin hydrauliletkut haurastuneet	1	Vaihdetaan vioittuneet letkut ja tarkastetaan letkuliitännät
Materiaalivahinko	1	Kaivuria rasitetaan suurella kuormalla, kun ulkoinen virtalähde on käytössä. Virtalähde menee vikatilaan	1	Ulkoisen virtalähteen ollessa käytössä pidetään pumpun kierrosnopeus mahdollisimman matalana
Materiaalivahinko	1	Väärän ulkoisen virtalähteen käyttö	2	Ulkoista virtalähdettä käytettäessä varmistetaan, että jännite vastaa haluttua arvoa (12V)
Tulipalo	1	Akkupaketin virheellinen säilytys	3	Akku säilytettävä suojassa iskuilta sekä kosteudelta. Pidempiaikainen säilytys tehtävä akun varauksen ollessa alle 30%. Pidettävä erillään palonarasta materiaalista.
Tulipalo	1	Akkupaketin virheellinen käyttö	3	Akun kuntoa tarkkailtava mahdollisen kaasuuntumisen varalta (akku turpoaa). Akun lataus suoritettava oikealla ohjelmalla ja sopivalla latausteholla. Akku tulee irroittaa laturista heti latauksen valmistuttua
Tulipalo	1	Virtalähteen kytkentä kaivuriin , kun myös akku on kytkettynä.	3	Kahden erillisen virtalähteen samanaikainen kytkeminen kaivuriin ehdottomasti kielletty
Tulipalo	1	Akku tulee kosketuksiin hydraulinesteen kanssa	3	Kaivuri tarkastettava vuotojen varalta aina ennen käyttöönottoa
Myrkytys	1	Hydraulinesteen virheellinen käsittely	3	Hydraulinesteen käsittelyssä noudatettava valmistajan myöntämiä suojausohjeita