SAVONIA

OPINNÄYTETYÖ - AMMATTIKORKEAKOULUTUTKINTO
TEKNIIKAN JA LIIKENTEEN ALA

SLAM-TOTEUTUS ETÄLÄSNÄOLOROBOTILLE

T E K I J Ä / T: Mika Leskinen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Tutkinto-ohjelma Tietotekniikan tutkinto-ohjelma	
Työn tekijä(t) Mika Leskinen	
Työn nimi SLAM-toteutus etäläsnäolorobotille	
Päiväys 3.12.2020	Sivumäärä/Liitteet 34/0
Toimeksiantaja/Yhteistyökumppani(t) Savonia Ammattikorkeakoulu	

Tiivistelmä

Opinnäytetyössä toteutettiin SLAM-järjestelmä Ohmni-etäläsnäolorobotille. Järjestelmän avulla ympäröivästä tilasta voidaan muodostaa pohjakartta, ja robotti voidaan laittaa kiertämään tilaa käyttäjän määrittelemien pisteiden kautta. Toteutuksessa käytetty etäläsnäolorobotti on yhdysvaltalaisen OhmniLabsin kehittämä Ohmni Developer Edition. Työn tilaaja on Savonia Ammattikorkeakoulun tutkimus- ja kehittämisosaston soveltavan hyvinvointiteknologian toimiala, joka tilasi robotin käyttöönsä osana Hyvinrobo-hanketta.

Toteutettu SLAM-järjestelmä on Linux-pohjaisen käyttöjärjestelmän päälle rakennettu Docker-ympäristö, johon luotiin ROS- ja Nodejs-pohjainen taustajärjestelmä. Käyttöliittymä toteutettiin web-sovelluksena siten, että järjestelmää voidaan käyttää lähiverkossa. Työssä perehdyttiin lisäksi yleisesti robotiikkaan ja esineiden Internetiin sekä näihin liittyvään historiaan ja tulevaisuudennäkymiin. Työssä myös tutkittiin robotiikka-alan ohjelmistokehitystä, avoimen lähdekoodin ratkaisuja ja sulautettujen järjestelmien kehitystyötä.

Opinnäytetyön tuloksena Ohmni-robottiin asennettiin toimiva SLAM-järjestelmä. Lopuksi arvioitiin järjestelmän toimintaa ja pohdittiin opinnäytetyön merkitystä sekä työn tilaajalle että tekijälle.

Avainsanat
SLAM, ROS, IoT, robotiikka, Ohmni, Linux, Docker, Nodejs

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport	
Degree Programme Degree Programme in Information Technology	
Author(s) Mika Leskinen	
Title of Thesis SLAM Implementation for a Remote Presence Robot	
Date 3.12.2020	Pages/Appendices 34/0
Client Organisation /Partners Savonia University of Applied Sciences	

Abstract

The purpose of this thesis was to implement a SLAM system for the Ohmni remote presence robot. The system can be used to create a map of a given area, and the robot can then be set to move around the area according to user-defined route points. The robot used in the thesis is called the Ohmni Developer Edition, made by OhmniLabs in the USA. The system was made for the Applied Wellbeing Technology branch of Savonia University of Applied Sciences, which purchased the robot as a part of their Hyvinrobo project.

The SLAM system consists of a ROS and Nodejs based backend built into a Docker environment, running on top of a Linux based operating system. The user interface was constructed as a web application in a way that allows for operation through a local network. The thesis work also involved the research of robotics and the Internet of Things in general, as well as the history and future insights of the two. Additional research was done on robotics software development, open source software and the development of embedded systems.

As a result of this thesis, a functioning SLAM system was installed into the Ohmni robot. The operation of the system was reviewed and the significance of the work to the commissioner and the author was evaluated.

Keywords
SLAM, ROS, IoT, robotics, Ohmni, Linux, Docker, Nodejs

ESIPUHE

Kiitokset Savonia Ammattikorkeakoulun TKI-osaston soveltavan hyvinvointiteknologian toimialan henkilöstölle tästä mainiosta opinnäytetyöaiheesta sekä kiitokset kaikille muille tahoille, jotka olivat kehityksessä tavalla tai toisella mukana.

Kiitokset Savonian opinto-ohjaaja Sami Lahdelle opinnäytetyön ohjauksesta, tuesta opintojen järjestämisessä sekä kärsivällisyydestä kaikissa opiskeluun liittyvissä asioissa.

Kiitokset Savonian opettajille erinomaisesta opetuksesta.

Kiitokset läheisille ja muille tavalla tai toisella mukana olleille tahoille tuesta opiskeluun liittymättömissä asioissa.

Kiitokset myös itselleni hitaasti mutta epävarmasti edenneistä opinnoista.

Kuopiossa 3.12.2020
Mika Leskinen

TERMIT JA LYHENTEET

API (Application Programming Interface) Ohjelmaan toteutettu rajapinta, joka mahdollistaa ohjelman kommunikoinnin toisten ohjelmien kanssa.

CSS (Cascading Style Sheets) Web-dokumenttien tyylien määrittelyyn käytetty muotoilukieli.

Docker Ohjelmistojen virtualisointiteknologia, jossa ohjelmistot suoritetaan käyttöjärjestelmän päällä niin sanotuissa containereissa, jotka käyttävät samaa käyttöjärjestelmäydintä, mutta ovat muuten toisistaan eristettyjä.

HTML (HyperText Markup Language) Standardi merkintäkieli web-dokumenttien esittämiseksi web-selaimessa.

HTTP (HyperText Transfer Protocol) Pyyntö-vastaus -tyyppinen tiedonsiirtoprotokolla web-resurssien, kuten HTML-sivujen jakamiseen.

Iot (Internet of Things) Esineiden internet, millä tarkoitetaan tavallisten laitteiden yhdistämistä osaksi globaalia tietoliikenneverkkoa.

IP (Internet Protocol) Tietoverkoissa yleisesti käytetty tietoliikenneprotokolla datapakettien siirtämiseen laitteiden välillä.

JSON (JavaScript Object Notation) Ohjelmoinnissa ja erityisesti web-kehityksessä yleisesti käytetty tietorakenneformaatti.

LiDAR (Light Detection And Ranging) Valon heijastumiseen perustuva etäisyyden mittaustekniikka.

REST (Representational State Transfer) Ohjelmistoarkkitehtuurityyli, joka helpottaa web-pohjaisten sovellusten kehittämistä määrittämällä yksiselitteisen tavan ohjelmien väliseen tiedonsiirtoon.

ROS (Robot Operating System) Kokoelma ohjelmistoja, joiden tarkoitus on yksinkertaistaa ja yhtenäistää robottien kehitystyötä.

SFTP (SSH File Transfer Protocol) SSH-yhteyksiä käyttävä, turvallinen tiedostojen siirtämiseen tarkoitettu tietoliikenneprotokolla.

SLAM (Simultaneous Localization And Mapping) Prosessi, jossa muodostetaan tai päivitetään karttaa aikaisemmin tuntemattomasta ympäristöstä ja samanaikaisesti arvioidaan jatkuvasti omaa sijaintia kartalla.

SoC (System on a Chip) Yksittäinen elektroniikkapiiri, johon on sisällytetty tavallisimmat tietokoneen tarvitsemat komponentit.

SSH (Secure Shell) Etäyhteyksissä käytetty turvallinen tietoliikenneprotokolla.

SISÄLTÖ

1 JOHDANTO

Viime vuosina on puhuttu paljon niin kutsutusta neljännestä teollisesta vallankumouksesta eli siitä, kuinka teknologia kehittyy juuri nyt nopeammin kuin koskaan ihmiskunnan historian aikana ja kuinka se vaikuttaa perustavanlaatuisella tavalla paitsi kaikkiin teollisuuden aloihin myös jokaiseen ihmiseen yksilötasolla. Käytännössä tällä tarkoitetaan sitä, että nimenomaan tietotekniikka-alan nopea kehitys mahdollistaa asioiden tekemisen eri tavalla kuin aikaisemmin. Teollisuusautomaatio ei varsinaisesti ole mikään uusi asia, mutta nykyteknologian avulla monet asiat on mahdollista toteuttaa huomattavasti tehokkaammin, nopeammin ja edullisemmin kuin vaikkapa 1960-luvulla.

Tämän niin kutsutun vallankumouksen suurin hyötyjä ei kuitenkaan ole perinteinen tuotantoteollisuus. Teknologian kehitys on tuonut tullessaan täysin uusia liiketoiminnan muotoja ja sitä kautta useita enemmän tai vähemmän hyödyllisiä tuotteita ja palveluita, joita tavallinen kuluttaja ei ole edes osannut aavistaa tarvitsevansa. Näistä monet ovat kuitenkin osoittautuneet ihmisten elämänlaadun kannalta jopa välttämättömiksi. Tavallisen kuluttajan lisäksi teknologian kehityksestä hyötyvät juuri näiden uusien tuotteiden ja palveluiden tarjoajat, sillä nykyteknologia on edullista ja yrittäjyyteen sisältyy siten aikaisempaa vähemmän taloudellisia riskejä.

Yksi esimerkki edellä mainitun kaltaisista uusista tuotteista on tämän opinnäytetyön keskeinen aihe, niin kutsuttu etäläsnäolorobotti (remote presence robot). Opinnäytetyön on tilannut Savonia Ammattikorkeakoulun TKI-osasto, joka hankki käyttöönsä yhdysvaltalaisen OhmniLabsin valmistaman Ohmni-etäläsnäolorobotin osana Savonian Hyvinrobo-hanketta. Tilaajan edustajina toimivat Savonian hyvinvointiteknologian TKI-asiantuntijat Tiina Arpola ja Jesse Honkanen.

Opinnäytetyön tavoitteena oli kehittää Ohmni-etäläsnäolorobottiin SLAM (Simultaneous Localization And Mapping) -ratkaisu, jonka avulla robotti osaa laatia ympäristöstään pohjakartan, kulkea itsenäisesti kartalle määritettävää reittiä sekä väistää reitille mahdollisesti osuvat esteet. Lopputuloksena aikaansaatu järjestelmä laajentaa robotin toiminnallisuutta mahdollistaen robotin käyttämisen myös muissa kuin sille alun perin suunnitelluissa tehtävissä. Koska toteutus pohjautuu robotiikka-alalla yleisesti käytettyihin ratkaisuihin, se mahdollistaa myös monenlaisen jatkokehittämisen robotin parissa esimerkiksi uusien opinnäytetyöaiheiden muodossa.

Opinnäytetyö oli luonteeltaan pääasiassa kehitystyö, jonka tuloksena syntyi loppukäyttäjän kannalta suhteellisen yksinkertainen SLAM-järjestelmä. Siitä huolimatta järjestelmä ei sellaisenaan olisi valmis esimerkiksi kaupalliseen käyttöön, vaan lopputulos on enemmänkin ns. ”proof of concept” -tyyppinen, kokeellinen ratkaisu, joka osoittaa, että tämäntyyppinen järjestelmä on mahdollista toteuttaa valitulle alustalle. Suuri osa itse työskentelystä koostui erilaisten teknologioiden tutkimisesta ja sopivien ratkaisujen etsimisestä, ja työ olikin tekijälleen kokonaisuudessaan erinomainen oppimistapahtuma.

2 ROBOTIIKKA JA ESINEIDEN INTERNET

2.1 Keskeiset käsitteet

Yksinkertaisimmillaan sanalla robotti tarkoitetaan automaattisesti toimivaa konetta, joka kykenee suorittamaan sille annettuja tehtäviä itsenäisesti. Tehtävät voivat olla joko yksittäisiä suoritteita tai koostua useammasta suoritteesta. Itse termi on johdettu tšekinkielisestä sanasta ”robota”, joka tarkoittaa pakkotyötä. Robot-sanaa käytti ensimmäisen kerran tšekkiläinen näytelmäkirjailija Karel Čapek vuoden 1920 scifi-näytelmässään R.U.R (Rossumovi Univerzální Roboti eli Rossum’s Universal Robots).

Robotiikalla tarkoitetaan kaikkea robotteihin liittyvää toiminnanharjoittamista, kuten robottien suunnittelua ja valmistusta. Termi on tiettävästi esiintynyt ensimmäisen kerran yhdysvaltalaiskirjailija Isaac Asimovin vuonna 1942 julkaistussa scifi-novellissa Runaround.

Esineiden Internet eli IoT (Internet of Things) tarkoittaa asioiden ja esineiden (esim. laitteet, anturit, ihmiset, eläimet) kytkemistä osaksi globaalia tietoverkkoa. Mitä tahansa asiaa tai esinettä, joka kykenee itsenäisesti kommunikoimaan IP-protokollaa käyttäen toisten asioiden tai esineiden kanssa, voidaan kutsua IoT-laitteeksi. IoT-termin teki tunnetuksi brittiläinen teknologiavaikuttaja Kevin Ashton vuonna 1999.

2.2 Historia ja kehitys

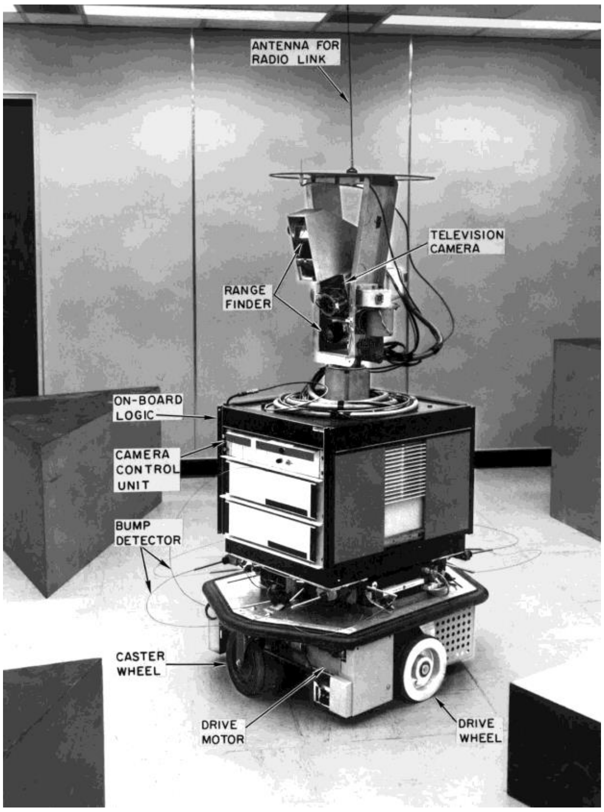
Eräänlaisia robotteja on tiettävästi ollut olemassa jo ennen ajanlaskun alkua (”automaton, automata”, itsenäisesti toimivia koneita, joiden merkitys oli lähinnä taiteellinen), mutta tässä kappaleessa käsitellään nimenomaan modernin ajan robotiikkaa ja sen roolia teollisuuden kehityksessä.

Ensimmäinen sarjatuotettavaksi tarkoitettu teollisuusrobotti Unimate (tai tarkemmin prototyyppi Unimate #001), otettiin koekäyttöön General Motorsin autotehtaalla New Jerseyssä, Yhdysvalloissa vuonna 1959. Kyseessä oli robotti, jonka tehtävänä oli nostaa kuumia kappaleita painevalukoneesta jäähdytettäväksi. Pian sarjatuotannon aloittamisen jälkeen Unimate-robotteja oli teollisuuden käytössä noin 450 kappaletta, ja Unimaten vaikutusta tuotantoteollisuuden kehitykseen pidetään yhtenä merkittävimmistä viimeisen sadan vuoden aikana. (Robotic Industries Association, 2020)

Unimaten kehittäjän, Joseph F. Engelbergerin mukaan robotiikan mahdollistava tekijä on ollut toisen maailmansodan aikana tapahtunut teknologinen edistyminen, ja 1960-luvulle tultaessa sekä amerikkalaiset että japanilaiset olivat valmiita hyödyntämään automatiikkaa suuremmassa mittakaavassa. Ensimmäiset teollisuusrobotit Euroopassa ja Japanissa otettiin käyttöön vuonna 1967. (Mickle, 1998-1999)

Voidaan siis sanoa, että moderni robotiikka ja teollisuusautomaatio nykyisessä muodossaan on saanut alkunsa 1950- ja 1960-lukujen taitteessa. Sittemmin teollisuuden prosessien automatisoinnista on tullut itsestäänselvyys, ja robotiikkaa on alettu hyödyntämään enemmän myös muissa käyttötarkoituksissa.

Ensimmäinen "älykäs" robotti oli yhdysvaltalaisen Stanfordin tutkimusinstituution (SRI International) vuosina 1966-1972 kehittämä Shakey. Älykkyydellä tarkoitetaan tässä tapauksessa sitä, että robotti kykeni itsenäiseen ympäristön havainnointiin, päättelyyn, suunnitteluun, virheidenkäsittelyyn ja kommunikointiin luonnollisella kielellä. Robotti käytti hyväkseen SRI:n ongelmanratkaisuteknologiaa (Stanford Research Institute Problem Solver eli STRIPS), joka oli yksi varhaisimmista tekoälyn muodoista (Cassel, 2017). Shakey-robotti on esitetty alla olevassa kuvassa (KUVA 1).



KUVA 1: Shakey-robotti (SRI International, 1968)

Tekoälyn hyödyntäminen niin robotiikassa kuin muussakin tietotekniikassa on yleistynyt suuresti tällä vuosituhannella. Muun muassa puolijohteiden valmistusmenetelmien, tekoälyalgoritmien ja tietoverkkojen kehittyminen on johtanut siihen, että tarvittava teknologia on edullista ja kaikkein arkipäiväisimmätkin tietotekniset laitteet kykenevät toimimaan "älykkäästi". Yksi asiaan vaikuttava merkittävä kehityssuunta on ollut myös se, että nykyään suuri osa tietojenkäsittelystä tehdään muualla kuin itse laitteessa, jolloin laitteen ei edes tarvitse olla kallis ja monimutkainen. Vaativimmat toiminnot voidaan suorittaa verkon välityksellä nimenomaan tähän tarkoitukseen kehitetyissä ns. reunalaitteissa ("edge computing", "fog computing") tai pilvipalveluissa ("cloud computing").

Toinen tällä vuosituhannella yleistynyt käsite on IoT. Edellä mainitut seikat ovat johtaneet myös siihen, että yhä suurempi osa tavallisista, ei-tietoteknisistä laitteista on nykyään varustettu jonkinlaisella tietokoneella ja kytketty osaksi globaalia tietoverkkoa. Näiden laitteiden tavoitteena on parantaa ihmisten elämänlaatua siten, että niiden avulla voidaan automatisoida arkipäiväisiä toimintoja ja niitä voidaan hallita etänä mistä tahansa.

Tämän lisäksi ne tarjoavat toimittajilleen valtavan määrän tietoa, jota palveluntarjoajat voivat hyödyntää omissa toiminnoissaan, kuten liiketoiminnan suunnittelussa, laadunvarmistuksessa sekä uusien tekoälyratkaisujen kehittämisessä ja parantamisessa.

2.3 Kaupallisuus ja tulevaisuudennäkymät

Robotiikka ja IoT ovat voimakkaasti kasvavia teollisuudenaloja. Robotiikkatuotteiden markkina-arvon on ennustettu olevan vuoteen 2025 mennessä kokonaisuudessaan hieman alle 210 miljardia dollaria (Scerra, 2020), ja IoT:n vuonna 2026 jopa yli 1100 miljardia dollaria (Fortune Business Insights, 2019). Nämä teollisuudenalat kasvavat nykyisin noin 25% vuosivauhdilla (Compound Annual Growth Rate, CAGR).

Toimintojen automatisoinnin vaikutuksen työllisyyteen ja palkkatasa-arvoon on arvioitu olevan pääasiassa positiivinen. Toisin sanoen automatisoinnin tuloksena syntyvien uusien työpaikkojen lukumäärä on korkeampi kuin niiden työpaikkojen, jotka automatisaatio korvaa. Tämä vaikuttaa myös siten, että palkkakeskiarvo nousee, kun matalapalkkaisemmat työt siirretään robottien vastuulle ja aikaisemmin matalapalkkaisemmissa töissä olleet ihmiset siirtyvät uudelleenkouluttautumisen kautta korkeapalkkaisempiin töihin. Uudelleenkouluttautuminen ei kuitenkaan aina ole mahdollista, mikä joidenkin arvioiden mukaan voi aiheuttaa palkkojen jakautumisen epätasa-arvoisemmin kuin aikaisemmin (International Federation of Robotics, 2017).

3 ROBOTIIKAN OHJELMISTOKEHITYS

3.1 Avoimen lähdekoodin ohjelmistot

Avoimen lähdekoodin ohjelmistolla eli ”Open Source” -ohjelmistolla tarkoitetaan tietokoneohjelmistoa, jonka lähdekoodi on vapaasti tarkasteltavissa, muokattavissa ja jaettavissa. Samaa avoimuuden periaatetta voidaan soveltaa tietokoneohjelmistojen lisäksi mihin tahansa materiaaliin, kuten laitteistoihin, kirjallisiin julkaisuihin tai projekteihin.

Avoimen lähdekoodin ohjelmistot ja ohjelmakirjastot on usein lisensoitu siten, että niiden käyttäminen johdannaisteoksissa on täysin vapaata. Johdannaisteoksia voidaan edelleen jakaa joko vapaasti tai kaupallisesti millä tahansa lisenssillä, yleensä ainoastaan sillä edellytyksellä, että alkuperäiset tekijänoikeusmerkinnät säilytetään. Tällaisia lisenssejä ovat esimerkiksi MIT-lisenssi ja Apache 2.0 -lisenssi (Synopsys, Inc., 2019). Tunnettuja MIT-lisenssillä jaettuja ohjelmistoja ja ohjelmakirjastoja ovat esimerkiksi Atom-tekstieditori, cURL-http-asiakasohjelma ja .NET Core. Esimerkkejä Apache-lisenssillä jaetuista ohjelmistoista ovat mm. CUPS-tulostinohjelmisto ja Apache HTTP Server.

Toinen yleisesti käytetty lisensointitapa on se, että ohjelmistoa voidaan käyttää ja muokata vapaasti, mutta mikäli johdannaisteoksia aiotaan jakaa, ne tulee jakaa samalla lisenssillä kuin alkuperäinen ohjelmisto. Käytännössä tämä tarkoittaa sitä, että tällaisella lisenssillä jaettua ohjelmistoa ei voi käyttää suljetun lähdekoodin ohjelmistossa, mikäli suljetun lähdekoodin ohjelmisto on tarkoitettu edelleen jaettavaksi. Tällaisia lisenssejä ovat esimerkiksi GNU General Public License eli GPL ja GNU Lesser General Public License eli LGPL (Synopsys, Inc., 2019). Tunnettuja GPL-lisenssillä jaettuja ohjelmistoja ovat esimerkiksi MySQL-tietokantaohjelmisto ja Abiword-tekstinkäsittelyohjelma.

3.2 Linux

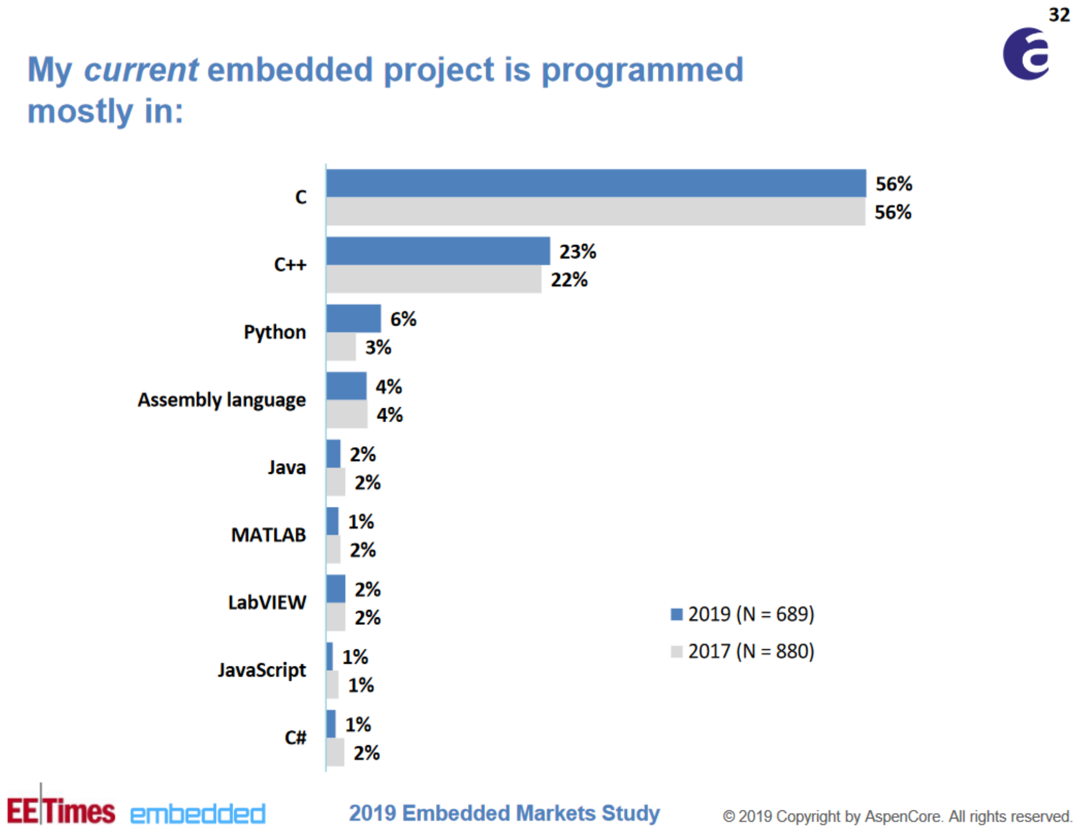
Linux on suomalaissyntyisen Linus Torvaldsin alun perin vuonna 1991 julkaisema avoimen lähdekoodin (GPL) käyttöjärjestelmäydin, joka perustuu monilta osin 1970- ja 1980-luvuilla suositun UNIX-käyttöjärjestelmän suunnitteluperiaatteisiin. Koska Linux on avoimen lähdekoodin ohjelmisto, se soveltuu muokattavuutensa ansiosta erityisen hyvin IT- ja elektroniikka-alan kehittäjille ja harrastajille.

Nykyään Linux-ytimeen pohjautuvia käyttöjärjestelmiä käytetään paljon esimerkiksi palvelinkoneissa ja mobiililaitteissa (Android), mutta merkittävin käyttökohde Linuxille ovat sulautetut järjestelmät. Näitä järjestelmiä ovat esimerkiksi reitittimet, kytkimet ja muut verkkolaitteet, teollisuusautomaatiolaitteet, valvonta- ja hälytysjärjestelmät, useat ”älykkäät” kuluttajalaitteet sekä robotiikka- ja IoT-laitteet. Sulautettujen järjestelmien kehittäjien keskuudessa Linux-pohjaisten ratkaisujen osuus kaikista käytössä olevista käyttöjärjestelmistä on suurin. (AspenCore, Inc., 2019)

3.3 Ohjelmointikielet ja arkkitehtuurit

Sulautettujen järjestelmien kehityksessä käytetään kaikista eniten C-kieltä (AspenCore, Inc., 2019). C-kieli on ns. alustariippumaton siinä mielessä, että C-kieliset ohjelmat voidaan helposti kääntää käytännössä mille tahansa prosessoriarkkitehtuurille. Järjestelmäkehittäjän näkökulmasta C-kielen tärkein ominaisuus on matalan tason pääsy laitteistoresursseihin, minkä tuloksena C-kielisistä ohjelmista saadaan nopeampia ja mm. muistinkäytön kannalta tehokkaampia kuin korkeamman tason ohjelmointikielillä toteutetuista ohjelmista. Perinteisten työpöytäsovellusten kehityksessä C-kieltä ei enää käytetä lähinnä siksi, että tällaiset ohjelmat on helpompi toteuttaa korkeamman tason kielillä. Lisäksi muisti- sekä muut laitteistoresurssit ovat nykyään työpöytäsovelluskehittäjän näkökulmasta katsottuna käytännössä rajattomat.

Toinen järjestelmäkehittäjien usein suosima kieli on C++, jota usein kuvaillaan ”luokalliseksi C-kieleksi”. C-kielen tavoin C++-kieli tarjoaa matalan pääsyn laitteistoresursseihin ja myös C++-kieliset ohjelmat voidaan kääntää käytännössä kaikille prosessoriarkkitehtuureille. C++-kieltä voidaan pitää tavallaan korkeamman tason versiona perinteisestä C-kielestä siinä mielessä, että se tarjoaa C-kielen ominaisuuksien lisäksi mahdollisuuden oliopohjaiseen ohjelmointiin. C++ on myös tietyiltä osin suunniteltu suoraan yhteensopivaksi C-kielen kanssa. Sulautettujen järjestelmien kehityksessä käytettyjen ohjelmointikielten suosituimmuusjärjestys AspenCoren vuoden 2019 markkinointitutkimuksen mukaan on esitetty alla olevassa kuvassa (KUVA 2).



KUVA 2: Ohjelmointikielten suosio sulautettujen järjestelmien kehityksessä

Koska C- ja C++ -kieliset ohjelmat voidaan kääntää suoraan prosessorin ymmärtämään muotoon, näiden ohjelmien suorittaminen sulautetuissa järjestelmissä ei vaadi varsinaisen käyttöjärjestelmän asentamista laitteeseen. Itse asiassa useimmat käyttöjärjestelmät itsessään on kirjoitettu juuri C-kielellä, mukaan luettuna Linux. Kaikkein yksinkertaisimpiin sulautettuihin järjestelmiin ei ole kannattavaa asentaa kokonaista käyttöjärjestelmää, varsinkaan jos järjestelmän toiminnallisuus on rajoitettu ainoastaan muutamaan yksinkertaiseen toimintoon. Käyttöjärjestelmän asentaminen on kuitenkin perusteltua erityisesti silloin, kun laitteen toiminnallisuutta halutaan muuttaa jälkikäteen esimerkiksi loppukäyttäjän toimesta. Tällöin haluttu toiminnallisuus voidaan toteuttaa erillisinä, käyttöjärjestelmän päällä suoritettavina ohjelmina.

Yksi esimerkki edellä mainitusta on varsinkin IT- ja elektroniikkaharrastajien suosima Raspberry Pi -alusta, joka on ns. yhden piirilevyn tietokone eli SBC (Single Board Computer). Alustan kehittäjä, Raspberry Pi Foundation, tarjoaa laitteelle ladattavaksi Debian/Linux-pohjaisen käyttöjärjestelmän (Raspberry Pi OS, ent. Raspbian), ja sen mukana työkalut mm. Python-ohjelmien kehittämiseen laitteelle. Python-kielisiä ohjelmia ei käännetä suoraan konekielelle, vaan ne suoritetaan käyttöjärjestelmän päällä ajettavalla ns. Python-tulkilla (”Python interpreter”). Koska tällainen Raspberry Pi-Python -yhdistelmä on varsin helppokäyttöinen ja aloittelijaystävällinen, se soveltuu hyvin harrastus- ja opetuskäyttöön sekä erilaisten prototyyppien kehittämiseen. Toisaalta tämä yhdistelmä on matalammalta tasolta katsottuna niin monimutkainen, resurssi-intensiivinen ja tavallaan kallis, ettei sitä ole mielekästä käyttää tuotantoympäristöihin tarkoitetuissa järjestelmissä sellaisenaan.

3.4 Sensorit ja toimilaitteet

Sulautetut järjestelmät ja erityisesti robotit ovat usein laitteita, jotka havainnoivat ympäristöään ja toimivat tavalla tai toisella näiden havaintojen perusteella. Ympäristön havainnointiin käytetään sensoreita, jotka yksinkertaisimmillaan muuttavat jonkin fysikaalisen ominaisuuden, kuten lämpötilan, etäisyyden tai nopeuden, sähköiseksi signaaliksi. Tämä signaali muutetaan lopulta laitteen ymmärtämään muotoon ja lähetetään laitteelle. Sensorit muuttavat mitattavan suureen usein analogiseksi signaaliksi, jolloin sensorin ja laitteen välille tarvitaan ns. ADC-piiri eli AD-muunnin (Analog to Digital Converter). Joissakin sensoreissa ja toisaalta myös useissa prosessoreissa tällainen on valmiiksi sisäänrakennettuna.

Toimilaitteet toimivat päinvastaisesti, eli ne muuttavat sähköisen signaalin fysikaaliseksi ominaisuudeksi, kuten liikkeeksi, valoksi tai lämmöksi. Jos haluttu fysikaalinen ominaisuus on luonteeltaan analoginen, laitteen ja toimilaitteen välille tarvitaan ns. DAC-piiri eli DA-muunnin (Digital to Analog Converter), jollainen voi edellä mainitun AD-muuntimen tapaan olla sisäänrakennettuna joko toimilaitteeseen tai itse prosessoriin.

Käytännössä kaikissa prosessoreissa on jonkinlainen rajapinta ulkoisten signaalien vastaanottamiseksi ja lähettämiseksi (I/O eli Input/Output), mutta rajapinnan toteutustapa vaihtelee prosessoriarkkitehtuurin mukaan. Yksinkertaisimmillaan prosessori käsittelee nämä signaalit itse, jolloin prosessorin kaikki muu toiminta pysäytetään I/O-toiminnon ajaksi.

Useimmissa I/O-laitteissa on kuitenkin erillinen ohjauspiiri, joka mahdollistaa näiden toimintojen suorittamisen asynkronisesti keskeytysten (interrupt) ja DMA:n (Direct Memory Access) avulla ilman, että prosessorin tarvitsee erikseen odottaa toimintojen valmistumista.

3.5 ROS

ROS (Robot Operating System) on kokoelma avoimen lähdekoodin työkaluja, joiden tarkoitus on yksinkertaistaa ja yhtenäistää robottien kehitystyötä. ROS ei nimestään huolimatta ole käyttöjärjestelmä sanan varsinaisessa merkityksessä, mutta se sisältää eräänlaisen käyttöjärjestelmän tapaisen ydinprosessin, jonka tehtävänä on ohjata niin kutsuttuja ROS-nodeja. Nämä nodet muodostavat ROS-järjestelmän varsinaisen toiminnallisuuden. Sekä ROS-ydin että ROS-nodet vaativat toimiakseen myös varsinaisen käyttöjärjestelmän. Tällä hetkellä ROS toimii ainoastaan UNIX-tyyppisissä järjestelmissä (mukaan luettuna Linux), ja se on kehittäjien toimesta testattu ja toimivaksi todettu ainakin Ubuntussa ja Mac OS X:ssä.

ROS-ytimen pääasiallinen tehtävä on vastata prosessien (eli ROS-nodejen) välisestä viestinnästä. Viestintä on toteutettu siten, että ROS-nodet voivat julkaista (publish) ja tilata eli vastaanottaa (subscribe) viestejä haluamillaan aihekanavilla (topic). Yleisellä tasolla voidaan sanoa, että usein viestien julkaisijoita ovat järjestelmään liitetyt sensorit, jotka jakavat mittaustietoa tietyillä aihekanavilla. Viestien vastaanottajia puolestaan ovat erilaiset toimilaitteet, jotka toimivat vastaanottamansa mittaustiedon perusteella ohjelmoijan haluamalla tavalla.

ROS:iin sisältyy lisäksi Linux-jakeluillekin tyypillinen pakettienhallintaratkaisu (rospack) ja työkalut omien pakettien/nodejen luomiseen (catkin). ROS-ohjelmat kirjoitetaan lähes poikkeuksetta joko C++:lla tai Pythonilla. (ROS Wiki, 2020)

3.6 SLAM ja navigointi

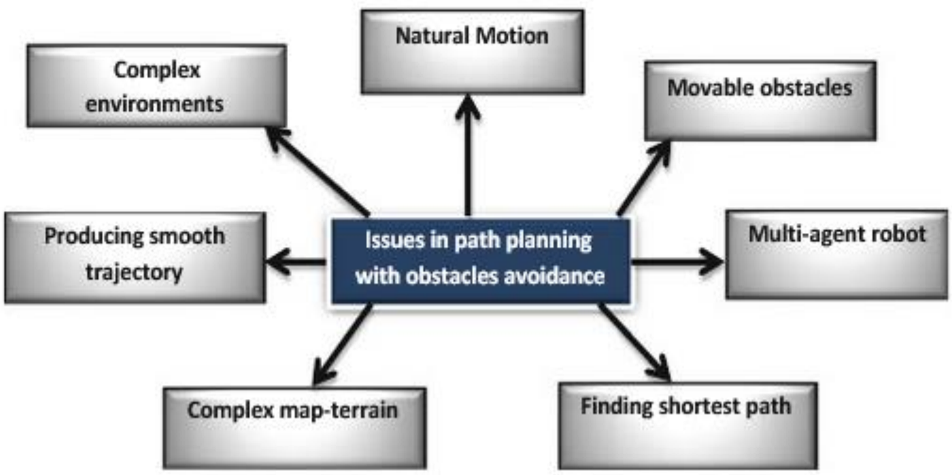
Robotit ovat usein itsenäisesti liikkuvia laitteita, jolloin niissä täytyy olla jonkinlainen navigointijärjestelmä. Yksinkertaiseen ”älykkääseen” liikkumiseen riittää periaatteessa kosketusanturi, jonka avulla laite tunnistaa törmäykset, ja kun laite huomaa törmänneensä esteeseen, se vaihtaa liikkumissuuntaa poispäin esteestä. Usein robotin törmääminen esteisiin ei kuitenkaan ole suotavaa, ja tällöin kosketusanturin tilalla voidaan käyttää esimerkiksi ultraäänianturia.

Pelkkä esteiden tunnistaminen ei kuitenkaan ole navigointia, vaan robotin tulee lisäksi tietää laajemmin, millainen ympäristö on, osata arvioida oma sijaintinsa ympäristön suhteen sekä osata suunnitella reitti nykyisestä sijainnista haluttuun sijaintiin. Ympäristön kartoittaminen tehdään yleensä optisilla sensoreilla, eli käytännössä 2D-laseretäisyysmittareilla tai tähän tarkoitukseen valmistetuilla syvyysnäkökameroilla. Sensoreista saadusta tiedosta muodostetaan kartta, joka tallennetaan sopivassa muodossa järjestelmään.

Robotin sijainnin arvioinnissa voidaan käyttää hyödyksi ympäristön kartoittamisen yhteydessä kerättyä tietoa siten, että tiedosta valitaan tiettyjä ominaisuuksia vertailukohdiksi ja näitä verrataan uusien mittaustietojen ominaisuuksiin. Jos ympäristö kuitenkin on sellainen, ettei siinä ole erityisiä vertailukelpoisia ominaisuuksia, tämä tapa ei yksistään ole riittävä sijainnin yksiselitteiseen päättelemiseen. Tällöin sijainnin määrittämisessä voidaan laitteen ominaisuuksien mukaan käyttää apuna muun muassa laitteen liikkumiselimistä saatua tietoa (esim. pyörien odometria), GPS-paikannusta tai mahdollisen inertia-anturin (IMU) tietoja.

Jotta robotti voisi itsenäisesti liikkua paikasta toiseen, sille pitäisi ensinnäkin jollakin tavalla pystyä kertomaan, mihin sen tulisi liikkua. Sen jälkeen robotin pitäisi osata suunnitella lyhyin reitti kohteeseen ottaen huomioon reitillä olevat staattiset eli kiinteät esteet, ja lisäksi sen pitäisi liikkuessaan osata väistää reitille mahdollisesti osuvat dynaamiset eli liikkuvat esteet. Tämän vuoksi reittisuunnittelussa käytetään usein hyväksi kahta erillistä karttaa, joista ensimmäinen on ns. globaali kartta, joka kuvaa koko sitä aluetta, millä robotti on liikkunut. Toinen on ns. lokaali eli paikallinen kartta, joka käytännössä kuvaa sitä aluetta, millä robotti sijaitsee tietyllä ajanhetkellä, kuten yhtä huonetta. Robotin havaitsemat dynaamiset esteet lisätään paikalliseen karttaan, jota päivitetään yleensä useita kertoja sekunnissa.

Reittisuunnittelualgoritmien kehityksessä tärkeimmät huomioonotettavat seikat ovat tehokkuus, tarkkuus ja turvallisuus. Kuten useimpiin matemaattisloogisiin ongelmiin, myös reittisuunnitteluun on olemassa useita algoritmeja, joista kaikki eivät välttämättä sovellu hyvin yhteen tiettyyn ongelmaan. Robottien reittisuunnittelussa haastavaa onkin juuri oikeiden algoritmien valinta (Koubaa, et al., 2018). Alla olevassa kuvassa on esitetty robottien reittisuunnittelun kannalta olennaiset seikat, jotka tulee ottaa huomioon algoritmien toteutuksessa (KUVA 3).



KUVA 3: Robottien reittisuunnittelun kannalta olennaiset seikat (Koubaa, et al., 2018)

4 OHMNI-ETÄLÄSNÄOLOROBOTTI

4.1 Käyttötarkoitus

Ohmni-robotti on yhdysvaltalaisen OhmniLabsin kehittämä ns. etäläsnäolorobotti, joka mahdollistaa ihmisten välisen kommunikoinnin videoyhteydellä verkon yli. Tavallisen videopuheluominaisuuden lisäksi Ohmnia voidaan myös liikutella etänä, jolloin se soveltuu etätapaamisiin monenlaisissa ympäristöissä. Tämä on hyödyksi esimerkiksi hoitolaitoksissa ja sairaaloissa, joissa videopuhelun vastaanottajalla ei välttämättä ole mahdollisuutta itse liikkua paikasta toiseen.

Valmistajan mukaan Ohmni-robotin avulla voidaan esimerkiksi tehostaa ajankäyttöä terveydenhuollossa, kun osa potilaskäynneistä voidaan tehdä robotin avulla eikä henkilöstön siten tarvitse pukeutua osastoilla tarvittaviin suojavarusteisiin yhtä usein. Robotti myös mahdollistaa tehokkaan etätyöskentelyn, kun työpaikalla voidaan liikkua paikasta toiseen ja järjestää esimerkiksi virtuaalisia haastatteluja ilman, että työntekijän tarvitsee varsinaisesti olla paikalla. Lisäksi robottia voidaan käyttää oppilaitoksissa, jolloin esimerkiksi liikuntakyvyttömät tai vaikeasti sairaat oppilaat voivat osallistua opetukseen kuin he olisivat oikeasti läsnä. (OhmniLabs, Inc., 2020)

4.2 Laitteisto

Fyysisesti Ohmni on noin 140 cm korkea, metallirunkoinen, pyörillä kulkeva laite, jonka yläpäähän on sijoitettu pystysuunnassa käänneltävä IPS-kosketusnäyttö ja 4K-kamera sekä alaspäin osoittava pienemmän resoluution navigointikamera. Rungon puolivälissä sijaitsevat virta- ja ohjauspainikkeet sekä kaiutin-mikrofoniyhdistelmä. Alustaan on asennettu robotin ohjauselektroniikka, latausliittimet sekä ajolaitteet, joita ovat kaksi vetävää pyörää edessä ja yksi tukipyörä takana. Robotin rakenne on esitetty alla olevassa kuvassa (KUVA 4).

KUVA 4: Ohmni-robotin rakenne

Ohmnin ohjauselektroniikka perustuu Intel Atom X5 Z8350 -piiriin, joka on vuonna 2016 julkaistu 64-bittisen, neliytimisen prosessorin ja 2 GB DDR3L -muistin sisältävä mobiilikäyttöön tarkoitettu SoC (System on a Chip). Tämän lisäksi Ohmnissa on 16 GB eMMC-tallennustila, IEEE 802.11ac -WLAN-sovitin sekä 95 Wh LiFePO4 -akku (OhmniLabs, Inc., 2020).

Ohmniin on myös valmistajan verkkosivulta tilattavissa erilaisia lisäosia, joista olennaisimpia ovat USB- ja sarjaporttilaajennussarja sekä Slamtecin RPLidar-A2M8, joka on esteiden tunnistukseen ja ympäristön kartoittamiseen yleisesti käytetty pyörivä 2D-lasersensori (LiDAR).

4.3 Ohjelmisto

Ohmni-robotin käyttöjärjestelmänä toimii Ohmni OS, joka on OhmniLabsin räätälöimä Android/Ubuntu -yhdistelmä. Ohmni OS:n tavoitteena on tarjota käyttäjälle Androidista tutun käyttöliittymän lisäksi täyden Ubuntu-järjestelmän teho ja monipuolisuus (Ohmnilabs, 2020).

Ohmnin kosketusnäyttö mahdollistaa tavanomaisten Android-sovellusten käyttämisen laitteessa samalla lailla kuin missä tahansa tablettitietokoneessa. Loppukäyttäjän kannalta ainoa näkyvä Android-sovellus on kuitenkin OhmniLabsin oma videopuheluohjelmisto, ja muiden Android-ohjelmien suorittaminen ei varsinaisesti kuulu robotin tavanomaiseen toimenkuvaan.

Laitteen videopuheluominaisuus on tarkoitettu käytettäväksi valmistajan verkkosivulta löytyvän web-sovelluksen kautta. Sovellukseen kirjauduttuaan käyttäjä voi tarkastella omia robottejaan, muuttaa robottien asetuksia sekä tietenkin ottaa videopuheluyhteyden haluamaansa robottiin. Videopuhelutilassa käyttäjä voi liikutella robottia ja keskustella robotin läheisyydessä olevien ihmisten kanssa. Jos laitteeseen on asennettu 2D-LiDAR, robotti myös tunnistaa lähellä olevat esteet ja pysähtyy, jos käyttäjä on suuntaamassa esimerkiksi ihmisiä päin. Lataukseen kytkemisen helpottamiseksi laitteessa on lisäksi ns. Autodock -toiminto, joka ohjaa robotin automaattisesti lataustelakkaan yhdellä napin painalluksella.

4.4 Sovelluskehitys

Sovelluskehittäjälle robotti tarjoaa neljä API-kerrosta, joita ovat:

- Web-API-kerros: valmistajan oma sovelluskehys, joka mahdollistaa sovellusten kehittämisen robotille yhtä helposti kuin minkä tahansa web-sovelluksen
- Natiivi-JS -kerros: laajennettava nodejs-pohjainen rajapinta robotin ohjaukseen ja laitteiston hallintaan
- Docker-kerros: mahdollistaa esimerkiksi täyden Ubuntun ajamisen laitteessa, ja Ubuntun päällä voidaan puolestaan ajaa mitä tahansa Linuxille tarkoitettuja sovelluksia, kuten laiteohjelmia, datankäsittelyohjelmistoja tai ROS:ia.
- Kernel-kerros: Ohmnin Linux-ydin, johon voidaan tarvittaessa lisätä esimerkiksi laitteistoajureita.

(Ohmnilabs, 2020)

Ohjelmistopinon alimmainen kerros (kernel-kerros) perustuu tavanomaiseen Linux-ytimeen, jonka lähdekoodi on avoin ja siten ohjelmistokehittäjän muokattavissa. Jos robotissa halutaan käyttää esimerkiksi jotakin tiettyä laitetta, jota robotti ei oletuksena tue, Linux-ytimeen voidaan lisätä laitteen vaatimat ominaisuudet ja koko ydin kääntää uudelleen.

Kernel-kerroksen päällä ajetaan OhmniLabsin omaa Androidiin ja Ubuntuun perustuvaa käyttöjärjestelmää (Ohmni OS), johon on toteutettu robotin ohjaukseen käytetty natiivi-JS -kerros. Tämä kerros sisältää joukon nodejs-ohjelmia, joiden koodi on niin ikään suoraan kehittäjän muokattavissa. Valmistaja kuitenkin suosittelee, että kaikki robotin toimintaan vaikuttavat muutokset natiivi-JS -kerrokseen toteutetaan niin kutsuttuina plugin-moduuleina. Tällöin esimerkiksi Ohmnin omat käyttöjärjestelmäpäivitykset eivät ylikirjoita kehittäjän muutoksia näihin ohjelmiin.

Käyttöjärjestelmän päälle voidaan toteuttaa myös ns. Docker-kerros, joka mahdollistaa esimerkiksi useimpien Linux-jakeluiden suorittamisen laitteessa Ohmni OS:n lisäksi. Näihin Linux-jakeluihin voidaan sisällyttää kehittäjän haluama toiminnallisuus perinteisinä Linux-ohjelmina. Koska Docker-kerros ei sinänsä ole pakollinen eikä Ohmnin vakio-ohjelmisto sitä käytä, kerroksen toteuttamisessa on otettava huomioon, että Ohmnin oma natiivi-JS -kerros voi käyttää esimerkiksi sarjaportteja ja muita laitteita suoraan. Tämän vuoksi osa Ohmnin omista ohjelmistoista tai toiminnallisuuksista joudutaan mahdollisesti kytkemään pois päältä, mikäli Docker-kerrosta on tarkoitus käyttää.

Ohjelmistopinossa ylimpänä on Ohmnin Web-API -kerros, joka mahdollistaa minkä tahansa tavallisen web-pohjaisen sovelluksen integroinnin robottiin. Ulkoista web-sovellusta voidaan käyttää joko Ohmnin näytöltä silloin, kun videopuheluominaisuus ei ole käynnissä (”standalone”) tai videopuhelun yhteydessä niin kutsuttuna overlay-komponenttina (”in-call”). Web-API -kerroksen avulla robottiin voidaan toteuttaa omia käyttöliittymäelementtejä ja ohjata robotin toimintaa API-kutsuilla, jotka on esitetty alla (KUVA 5).

Supported Ohmni WebAPI calls

```
Ohmni.move()
Ohmni.setLightColor()
Ohmni.setNeckPosition()
Ohmni.setNeckTorqueEnabled()
Ohmni.showPageOnBot("url") (version >= 4.0.2)
Ohmni.hidePageOnBot() (version >= 4.0.2)
Ohmni.requestBotInfo() (version >= 4.0.8.2)
Ohmni.captureVideo(resolutionWidth, resolutionHeight)
Ohmni.captureAudio()
Ohmni.stopCaptureAudio()
Ohmni.setBotVolume(Number);
Ohmni.getBotVolume();
Ohmni.on('cdata', callback);
```

KUVA 5: Ruudunkaappaus Ohmnin web-API -kutsuista

5 SLAM-OHJELMISTON TOTEUTUS

5.1 Yleistä

Seuraavissa kappaleissa on esitetty SLAM-toiminnallisuuden toteuttamisen kannalta olennaisimmat ohjelmistot, toimenpiteet sekä muut huomioitavat seikat. Opinnäytetyössä käytetty robotti on Ohmnin kehittäjille suunnattu malli (”Ohmni Developer Edition”), jota ei ainakaan tällä nimikkeellä ole enää saatavilla. Valmistaja on hiljattain siirtynyt eräänlaiseen Developer License -malliin, jossa kehittäjälle tarjotaan tavanomaisen Ohmni-robotin lisäksi määräaikaista kehittäjälisenssiä, joka mahdollistaa pääsyn valmistajan web-API -rajapintaan.

Tässä työssä edellä mainittua rajapintaa ei kuitenkaan käytetty, vaan kaikki web-pohjaiset sovellukset toteutettiin itse laitteeseen. Muun muassa tästä syystä SLAM-toiminnallisuus on tarkoitettu käytettäväksi ainoastaan lähiverkossa. Toteutettu ratkaisu sisältää myös muita elementtejä, joiden vuoksi järjestelmän käyttäminen lähiverkon ulkopuolelta olisi sekä epäkäytännöllistä että tietoturvan ja verkon kuormituksen kannalta haasteellista.

5.2 Vaatimukset ja laitteisto

Opinnäytetyön tavoitteena oli tuottaa Ohmni-robottiin toiminnallisuus, jonka avulla voidaan luoda pohjakartta tilasta ja laittaa robotti kiertämään tilaa kartan perusteella määritettyjen pisteiden kautta. Tätä tarkoitusta varten robottiin toteutettiin SLAM-toiminto, johon liittyi useita vaatimuksia muun muassa laitteiston, ohjelmiston ja käytettävyyden kannalta.

Laitteiston osalta olennaisin vaatimus on, että robotin pitää pystyä tuottamaan tietoa ympäröivästä maailmasta sekä omasta sijainnistaan ja asennostaan ympäristön suhteen. Ohmni-robottiin on ympäristön havainnointia varten asennettu 2D-LiDAR, ja sijainnin sekä asennon määrittämiseen voidaan käyttää LiDAR:n lisäksi Ohmnin pyörissä sijaitsevia asentotunnistimia (”wheel encoders”).

Jotta antureilta saatavaa mittaustietoa voitaisiin käyttää SLAM-toteutuksessa, robotissa pitää olla tarpeeksi laitteistoresursseja ROS:n ja muiden tarvittavien ohjelmien suorittamiseen. Ohmnin 2 GB:n RAM-muisti ja mobiilikäyttöön suunniteltu prosessori vastaavat tähän vaatimukseen heikosti, mutta jos robotissa ei suoriteta SLAM:n lisäksi muita resurssi-intensiivisiä toimintoja, ne riittävät tarkoitukseen.

Loppukäyttäjän kannalta SLAM-toiminnon käyttämisen tulisi olla mahdollisimman helppoa ja yksinkertaista, mikä vaatii joitakin muutoksia Ohmnin omaan natiivi-JS -ohjelmistoon. Käytännössä tämä tarkoittaa sitä, että Ohmnin omaan ohjelmistoon on lisättävä määritykset siitä, miten SLAM-toiminto käynnistetään ja pysäytetään. Lisäksi Ohmnin omat käyttöjärjestelmäpäivitykset on otettava pois käytöstä, jotta ne eivät ylikirjoittaisi em. määritykset sisältäviä tiedostoja.

5.3 Arkkitehtuuri ja teknologiat

Opinnäytetyössä toteutettu SLAM-järjestelmä sijoittuu Ohmnin kehittäjämanuaalissa esitetyistä API-kerroksista Docker- ja natiivi-JS -kerroksiin. Natiivi-JS -kerrokseen lisättiin nodejs/express -pohjainen palvelin, jonka kautta voidaan käynnistää itse toiminnallisuuden sisältävä Docker-container. Tämä Docker-container sisältää Ubuntu-käyttöjärjestelmän, johon on valmiiksi asennettu ROS-ympäristö sekä OhmniLabsin oma ROS-node (tb_control), joka vastaa pääasiassa Ohmnin ajolaitteiden hallinnasta. Ohmnin omaa natiivi-JS -ohjelmistoa muokattiin siten, että em. express-palvelin käynnistyy automaattisesti robotin käynnistyksen yhteydessä.

Ubuntu-ympäristöön toteutettiin niin ikään nodejs/express-pohjainen palvelin, jonka avulla voidaan ohjata SLAM-toiminnon tarvitsemia prosesseja, kuten ROS-nodeja. Lisäksi palvelimelta myös lopulta ladataan SLAM-järjestelmän ohjaamiseen tarkoitettu web-pohjainen käyttöliittymä. OhmniLabsin tarjoama Docker-container käynnistää oletuksena ROS:n ydinprosessin ja tb_control-noden automaattisesti. Se ei tässä ratkaisussa ollut tarkoituksenmukaista, joten Docker-ympäristöä muokattiin siten, että näiden prosessien käynnistys suoritetaan em. palvelinohjelmasta.

5.4 Docker-ympäristö

Ensimmäinen vaihe SLAM-toteutuksessa oli Docker-ympäristön asentaminen robottiin. Ohmnin omassa käyttöjärjestelmässä on valmiiksi itse Docker-ohjelma, ja toteutuksessa tarvittava Ubuntun sisältävä Docker-container (ohmnilabsvn/ohmni_ros) haettiin OhmniLabsin Dockerhub-kokoelmasta. Tämän jälkeen haettu container käynnistettiin ja Ubuntuun asennettiin mm. seuraavat ohjelmat:

- net-tools (verkkolaitteiden hallintaohjelma, sisältää mm. ”ifconfig” -ja ”netstat” -komennot)
- ros-melodic-rosbash (kokoelma hyödyllisiä ROS-komentoja, mm. ”rosls”)
- openssh-server (SSH-palvelin)
- motion (mahdollistaa web-kamerasyötteen jakamisen HTTP:n yli)

Lisäksi Ubuntuun luotiin uusi käyttäjä, jolle annettiin oikeudet suorittaa komentoja pääkäyttäjänä. SSH-palvelimen portiksi määritettiin 2224, koska oletusportti 22 oli jo varattu Ohmnin oman käyttöjärjestelmän SSH:lle. Docker-containereihin ei yleensä määritellä erillisiä SSH-palveluita, mutta Ohmnin käyttöjärjestelmän SSH-palvelin ei syystä tai toisesta toiminut suoraan tiedostojen siirtämiseen tarkoitetun SFTP-protokollan kanssa.

Muutosten jälkeen käynnissä olevasta Docker-containerista tehtiin uusi Docker-image, jotta em. muutoksia ei tarvitsisi tehdä uudelleen jokaisen Dockerin käynnistyksen yhteydessä. Lopulta Dockerin käynnistysparametrit määritettiin siten, että Ubuntu voi käyttää Ohmnin käyttöjärjestelmän luomia laitetiedostoja (jaettu ”/dev” -hakemisto). Ubuntu-käyttäjän kotihakemisto on niin ikään jaettu Ohmni OS:n kanssa siten, että Ubuntu-järjestelmän luomat tiedostot säilyvät levyllä myös sen jälkeen, kun Docker suljetaan.

5.5 ROS-ympäristö

OhmniLabsin oman ROS-noden (tb_control) ja ROS-ytimen lisäksi ROS-ympäristöön toteutettiin seuraavat ROS-nodet:

- rplidar_ros (LiDAR:n ohjaus)
- tf_broadcaster (staattisten transformaatioviestien julkaisu)
- rosbridge_server (ROS:n ja web-sovellusten välinen websocket-rajapinta)
- slam_toolbox (kartoitustoiminnot)
- robot_pose_publisher (robotin asennon ja sijainnin julkaisu)
- move_base (navigaatiotoiminnot)

ROS-ydin koostuu kolmesta osasta, joita ovat pääprosessi (ROS Master), jaettu parametrivarasto (Parameter Server) sekä tapahtumaloki (rosout). Pääprosessin tehtävänä on valvoa muita ROS-prosesseja sekä vastata nodejen välisestä viestinnästä toimimalla eräänlaisena osoitekirjana. Tämän osoitekirjan avulla nodet löytävät toisensa ja voivat sen jälkeen jatkaa viestintää keskenään. Parametrivarastoon voidaan tallentaa avain-arvopareja, kuten esimerkiksi ROS-nodejen käynnistysparametreja. Tapahtumalokin tehtävänä on kirjata ROS-nodejen tuottamia lokiviestejä.

Kaikki ROS-nodet tarvitsevat toimiakseen käynnissä olevan ROS Master -prosessin. Kun pääprosessi on käynnissä, muita ROS-nodeja voidaan käynnistää ”rosrun”- ja ”roslaunch” -komennoilla, joista jälkimmäinen vaihtoehto lukee nodejen käynnistysparametrit erikseen määritellyistä ”.launch”-tiedostoista. Roslaunch-komento myös käynnistää ROS:n pääprosessin, mikäli se ei ole valmiiksi käynnissä.

OhmniLabsin tb_control-noden pääasiallinen tehtävä on vastaanottaa toimilaitteille lähetettyjä viestejä ja käskeä toimilaitteita toimimaan viesteissä kuvatulla tavalla. Lisäksi se tarjoaa tietoa toimilaitteiden tilasta, kuten pyörien ja kaulan asennosta, sekä yleistä tilatietoa esimerkiksi siitä, onko robotti kytkettynä lataustelakkaan.

RPLidar-node on Slamtecin kehittämä ROS-ohjelma, joka julkaisee LiDAR:lta tulevaa mittaustietoa muiden nodejen käytettäväksi ”/scan” -aihekanavalla. Lisäksi se tarjoaa ROS-palvelun (service), jonka avulla LiDAR:n moottori voidaan tarvittaessa käynnistää ja pysäyttää. LiDAR:n käynnistysparametrit, kuten sarjaportin tiedostopolku ja LiDAR:n malli, voidaan määritellä noden ”.launch” -tiedostossa.

LiDAR julkaisee mittaustiedot absoluuttisina arvoina, ja se on asennettu noin 25 cm robotin kääntymiskeskipisteen takapuolelle. Tästä syystä LiDAR:lta saatava sijaintitieto ei suoraan vastaa itse robotin sijaintia ympäristössä. ROS:iin kirjoitettiin ja käännettiin oma tf_broadcaster -node, jonka tehtävänä on julkaista niin kutsuttu transformaatio eli eräänlainen koordinaattimuunnos robotin alustan ja LiDAR:n välillä. Käytännössä ohjelman ainoa tehtävä on kertoa muille nodeille, että robotin keskipiste pituussuunnassa on oikeasti noin 25 cm LiDAR:sta eteenpäin.

Rosbridge_server-noden tehtävänä on toimia kommunikointirajapintana ROS:n ja web-sovellusten välillä. Se mahdollistaa käyttöliittymän ja ROS-nodejen välisen viestienvaihdon muuttamalla käyttöliittymän lähettämät JSON-muotoiset viestit ROS:n ymmärtämään muotoon ja päinvastoin. Kommunikoinnissa käytetään kaksisuuntaista websocket-protokollaa, jolloin käyttöliittymän ei tarvitse tehdä erillisiä, ajastettuja kutsuja palvelimelle ja viestienvaihto on lähes reaaliaikaista.

Slam Toolbox on kokoelma avoimen lähdekoodin työkaluja, jotka kattavat lähes kaikki 2D-kartoittamisessa tarvittavat toiminnot. Tässä työssä slam_toolbox-noden pääasiallinen tehtävä on kuunnella LiDAR:lta tulevia viestejä, luoda niiden perusteella ympäristöstä globaali kartta ja tarjota luotu kartta muiden nodejen käytettäväksi ”/map”-aihekanavalla ns. ”OccupancyGrid”-muotoisina viesteinä. Se myös tarjoaa ROS-palvelun, jonka avulla kartta voidaan tarvittaessa tallentaa levylle ja aikaisemmin tallennettu kartta ladata. Karttaa päivitetään automaattisesti sitä mukaa, kun robotti liikkuu ympäristössä. Kartan koko, tarkkuus, päivitystiheys ja muut parametrit voidaan määritellä slam_toolbox-noden ”.launch”-tiedostossa.

Robot_pose_publisher-node julkaisee robotin sijainti- ja asentotiedon ”/robot_pose” -aihekanavalla x- ja y-koordinaattien sekä kiertokulman yhdistelmänä globaalin kartan suhteen. Sen pääasiallinen tehtävä on tarjota em. tieto web-käyttöliittymälle, jotta robotti voidaan sijoittaa oikein käyttöliittymässä näytettävään karttaan.

Move_base-node on osa ROS-pohjaisten robottien reittisuunnitteluun tarkoitettua ”ros_planning/navigation” -ohjelmistopakettia. Sen tehtävänä on ylläpitää karttaa ympäristössä havaituista esteistä sekä paikallisella tasolla (local costmap) että laajemmin (global costmap). Lisäksi se tarjoaa ROS-palvelun, joka vastaanottaa konkreettisia liikkumiskomentoja. Tällainen komento on käytännössä ROS-viesti, joka sisältää robotille määrätyn kohteen x- ja y-koordinaattien sekä kiertokulman yhdistelmänä. Kun move_base-node vastaanottaa tällaisen viestin, se suunnittelee lyhyimmän reitin nykyisestä sijainnista haluttuun sijaintiin edellä mainittujen karttojen perusteella ja lähettää kohteeseen pääsemiseksi tarvittavat komennot liikkumisesta vastaaville toimilaitteille, eli tässä tapauksessa OhmniLabsin tb_control-nodelle.

Useimmat navigointiin liittyvät parametrit, kuten robotin minimi- ja maksimiliikkumisnopeudet, alustan muoto ja koko, esteiden havainnointietäisyys, koordinaattikehykset sekä karttojen päivitystiheydet voidaan määritellä move_base-noden asetustiedostoissa. Nämä tiedostot ovatkin suurin yksittäinen tekijä, jolla robotin käytännön toimintaa voidaan hienosäätää ympäristöön ja käyttötarkoitukseen sopivaksi.

5.6 Taustajärjestelmät

Jotta SLAM-järjestelmää voitaisiin käyttää järkevästi, ROS-ympäristön ja käyttöliittymätason väliin toteutettiin kaksi erillistä taustajärjestelmää, joista ensimmäinen suoritetaan Ohmnin natiivi-JS -kerroksessa ja toinen Dockerissa. Käytännössä nämä molemmat ovat nodejs/express -pohjaisia HTTP-palvelimia, joille voidaan antaa komentoja REST-tyyppisillä HTTP-kutsuilla. Kumpaankin palvelimeen on niin ikään toteutettu erillinen selainpohjainen käyttöliittymä.

Koska Ohmnissa ei ole vakiona web-palvelinta, natiivi-JS -tasolla toimiva taustajärjestelmä oli mahdollista määrittää toimimaan siten, että se kuuntelee vapaana olevaa TCP-porttia 80. Useimmat web-selaimet tunnistavat tämän portin HTTP-palveluksi, joten loppukäyttäjän tarvitsee aluksi syöttää selaimeensa ainoastaan Ohmnin lähiverkon IP-osoite. Tällöin käyttäjälle ladataan web-pohjainen käyttöliittymä, jonka avulla SLAM-järjestelmä voidaan käynnistää ja sammuttaa. Lisäksi taustajärjestelmä tarjoaa API-päätepisteet (endpoint), joiden avulla Ohmnin automaattiset käyttöjärjestelmäpäivitykset (OTA-updates) voidaan asettaa joko päälle tai pois. SLAM-järjestelmän käynnistäminen tarkoittaa käytännössä sitä, että koko SLAM/ROS -toiminnallisuuden sisältävä Docker-container käynnistetään ”docker run” -komennolla. Sammuttaminen puolestaan aiheuttaa sen, että Docker-container pysäytetään ja poistetaan Dockerin prosessiluettelosta ”docker stop”- ja ”docker rm” -komennoilla.

SLAM-järjestelmän käynnistyksen yhteydessä Ohmnin oma natiivi-JS -tason ohjelmisto otetaan kokonaan pois käytöstä, jotta se ei yrittäisi käyttää LiDAR:n ja Ohmnin ajolaitteiden tarvitsemia USB-sarjaportteja samaan aikaan Docker-tason ohjelmistojen kanssa. Kun SLAM-järjestelmä sammutetaan, Ohmnin oma ohjelmisto otetaan takaisin käyttöön. Koska Ohmnin ohjelmiston pysäyttäminen on tarpeen, taustajärjestelmää ei voinut toteuttaa OhmniLabsin suosittelemalla tavalla plugin-moduulina. Tällöin Ohmnin ohjelmiston pysäyttäminen lopettaisi myös taustajärjestelmän toiminnan, ja SLAM-toiminto olisi siten mahdotonta myöhemmin sammuttaa ilman SSH-yhteyttä laitteeseen. Taustajärjestelmä toteutettiin tästä syystä omana nodejs-prosessina, joka käynnistyy samaan aikaan Ohmnin ohjelmiston kanssa irrotettuna Ohmnin pääprosessista. Jotta tämä oli mahdollista, Ohmnin omaan ohjelmistoon lisättiin tarvittavat rivit uuden nodejs-prosessin käynnistämiseksi. Muutosten jälkeen Ohmnin omat käyttöjärjestelmäpäivitykset otettiin pois käytöstä, etteivät ne rikkoisi SLAM-järjestelmän toimintaa.

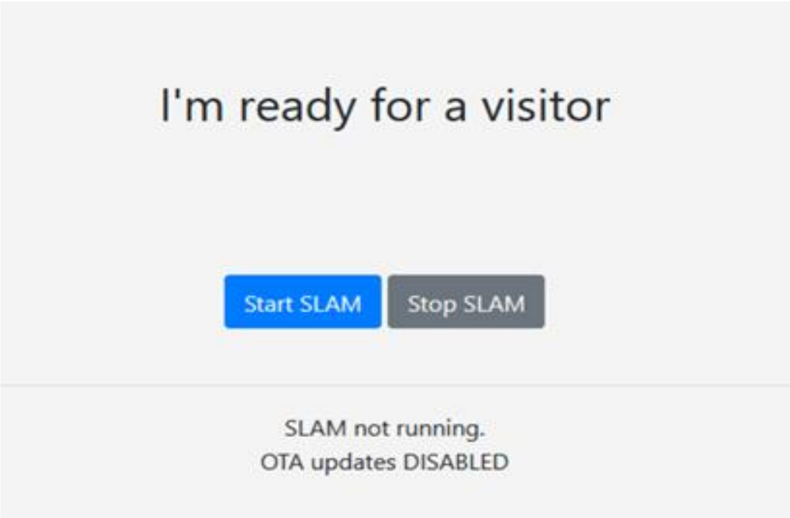
Toinen toteutetuista taustajärjestelmistä käynnistyy SLAM-ohjelmiston käynnistyksen yhteydessä ja se suoritetaan Docker-ympäristössä. Tämä järjestelmä vastaa pääasiassa SLAM-toiminnallisuuden tarvitsemien prosessien ohjauksesta (edellisessä luvussa esitetyt ROS-nodet). Prosesseja hallitaan nodejs-ohjelmassa määritettyjen API-päätepisteiden avulla siten, että niitä voidaan käynnistää ja pysäyttää tarpeen mukaan.

ROS-prosessien lisäksi tämän taustajärjestelmän kautta hallittavia ohjelmia ovat Ohmnin USB-kameroita ohjaavat prosessit (motion). Ohmnin kameroille on Docker-ympäristöön luotu ”motion.conf” -tiedostot, joissa voidaan määritellä kameroiden asetukset. Näistä olennaisimpia ovat kameran kuvataajuus, resoluutio sekä HTTP-portti, jonka kautta kameran syöte voidaan hakea web-käyttöliittymään. Prosessienhallinnan lisäksi Docker-ympäristössä suoritettava taustajärjestelmä vastaa muun muassa tallennettujen karttojen käsittelystä sekä käyttäjän määrittelemien reittien tallentamisesta ja lataamisesta.

Edellä mainittu taustajärjestelmä myös tarjoaa käyttäjälle esitettävän SLAM-järjestelmän yleisnäkymän, joka on perinteisillä web-ohjelmointitekniikoilla (HTML, Javascript, CSS) toteutettu selainpohjainen sovellus. Käyttöliittymän ja ROS:n välisessä websocket-kommunikaatiossa ja kartan esittämisessä käytetään hyväksi RobotWebTools:n ”roslibjs”- ja ”ros2djs” -Javascript-kirjastoja.

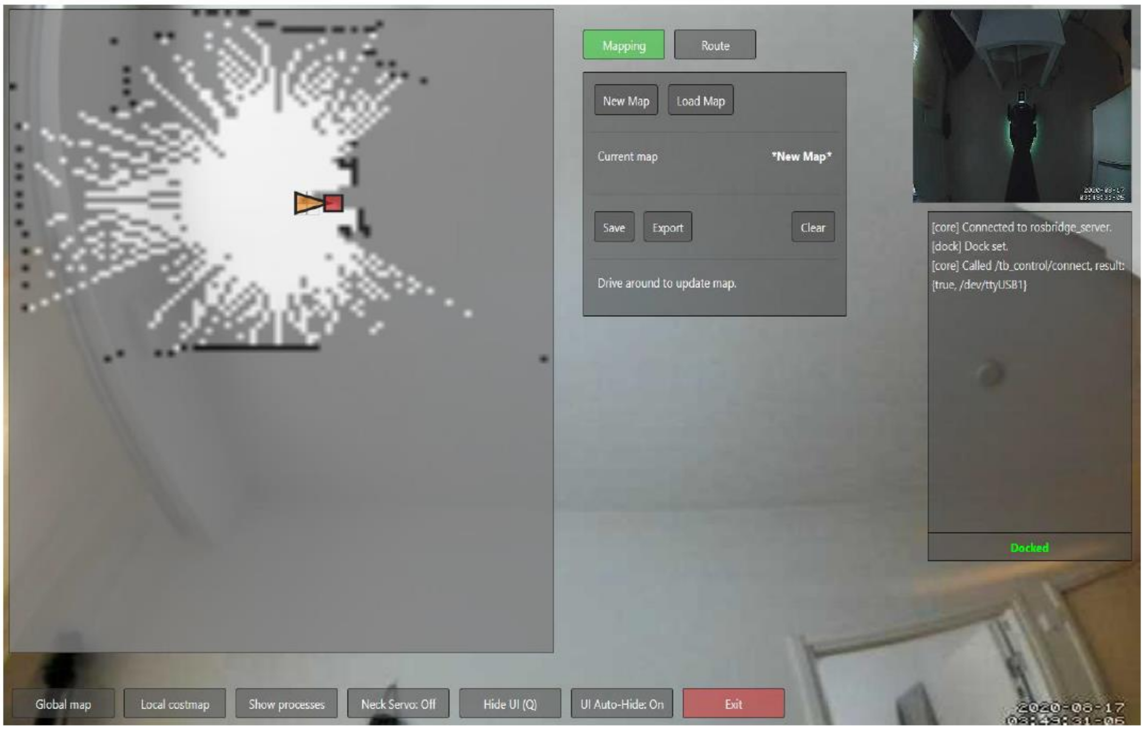
5.7 Käyttöliittymä

Kun selaimella navigoidaan robotin lähiverkon IP-osoitteeseen, käyttäjälle esitetään yksinkertainen web-sivu, jossa on painikkeet SLAM-järjestelmän käynnistämiseksi ja pysäyttämiseksi. Lisäksi käyttäjälle annetaan tieto siitä, onko SLAM-järjestelmä jo käynnissä ja ovatko robotin automaattiset käyttöjärjestelmäpäivitykset käytössä. Alkunäkymä on esitetty alla olevassa kuvassa (KUVA 6).



KUVA 6: SLAM-järjestelmän alkunäkymä

SLAM:n käynnistyspainikkeen painamisen jälkeen käyttäjälle esitetään linkki varsinaiseen Docker-ympäristön taustajärjestelmästä ladattavaan käyttöliittymään, jonka yleisnäkymä on esitetty alla (KUVA 7).



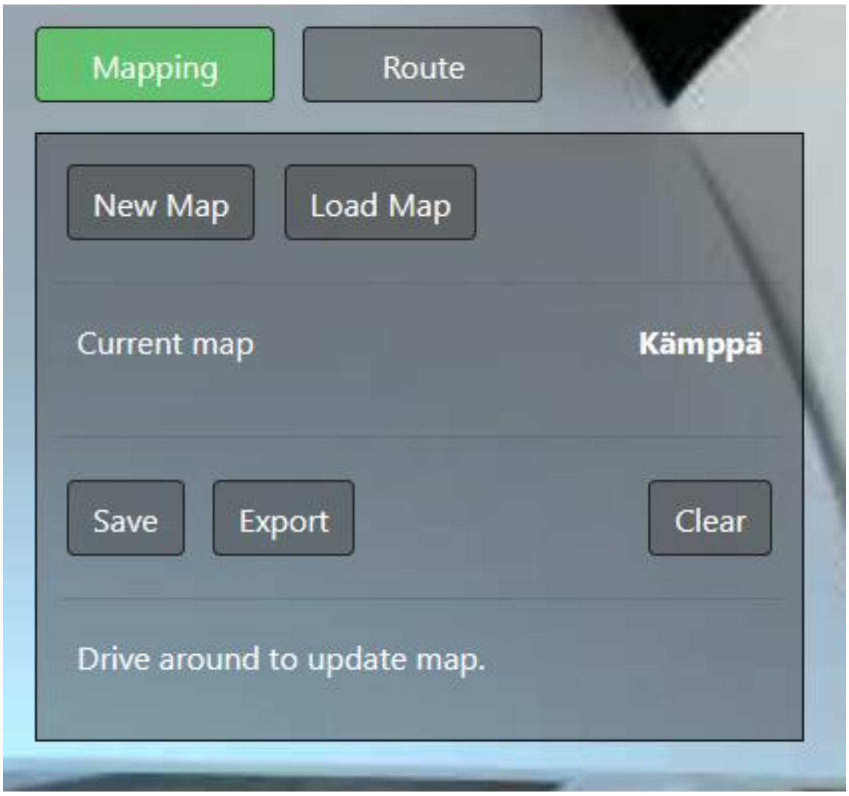
KUVA 7: SLAM-järjestelmän yleisnäkymä

Yleisnäkymä koostuu seuraavista osista:

- Pääkameran kuvasyöte (taustalla)
- Karttanäkymä (vasemmassa reunassa)
- Tilan valintapainikkeet ja tilakohtainen ohjausnäkymä (kartan vieressä oikealla)
- Navigointikameran kuvasyöte (oikeassa yläkulmassa)
- Lokiviestit ja telakoinnin tila (navigointikameran kuvasyötteen alla)
- Yleiset ohjauspainikkeet (alareunassa)

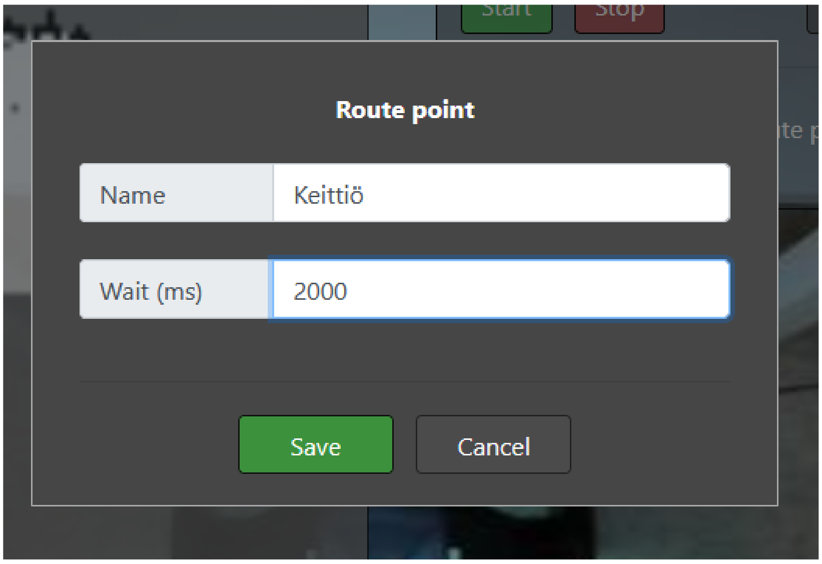
Yleisnäkymässä robottia voidaan ohjata W-, A-, S- ja D-näppäimillä, ja robotin kaulaa voidaan käännellä pystysuunnassa R- ja F-näppäimillä, jos kaulan servomoottori on kytketty päälle. Tilan valintapainikkeilla voidaan valita robotin tila, joka voi olla joko kartoitus (Mapping) tai reitti (Route).

Kartoitustilassa järjestelmä päivittää karttanäkymää sitä mukaa, kun robotti liikkuu ympäristössä. Lisäksi käyttäjällä on mahdollisuus tallentaa valmis kartta nimellä, ladata aikaisemmin tallennettu kartta levyltä, pyyhkiä karttanäkymässä oleva kartta tyhjäksi tai aloittaa uusi kartta. Kartoitustilan ohjausnäkymä on esitetty alla (KUVA 8).



KUVA 8: Kartoitustilan ohjausnäkymä

Reittitilassa käyttäjä voi määritellä reittipisteitä, joiden kautta robotti voidaan laittaa kiertämään kartoitettua tilaa. Reittipiste valitaan kartalta kaksoisklikkaamalla, jolloin käyttäjälle annetaan mahdollisuus nimetä valitsemansa piste. Käyttäjä voi myös halutessaan määrittää ajan, jonka robotti pysyy pisteessä ennen kuin se siirtyy seuraavaan. Reittipisteen määrittäminen on esitetty seuraavassa kuvassa (KUVA 9).



KUVA 9: Reittipisteen määrittäminen

Kun käyttäjä on määritellyt haluamansa määrän reittipisteitä, robotti voidaan laittaa kiertämään tilaa näiden pisteiden kautta reittitilan ohjausnäkymän ”Start”-painikkeella. Robotti voidaan tarvittaessa pysäyttää näkymän ”Stop”-painikkeella. Lisäksi käyttäjällä on mahdollisuus tallentaa reitti nimellä, ladata aikaisemmin tallennettu reitti levyltä, tyhjentää nykyinen reitti tai aloittaa uusi reitti. Määritettyä reittipistettä voidaan tarvittaessa muokata tai reittipiste poistaa klikkaamalla pistettä ohjausnäkymän listasta. Reittitilan ohjausnäkymä on esitetty alla olevassa kuvassa (KUVA 10).

KUVA 10: Reittitilan ohjausnäkymä

SLAM-järjestelmän muodostama kartta ja käyttäjän määrittelemät reittipisteet esitetään yleisnäkymän vasemmassa reunassa sijaitsevassa karttanäkymässä. Robotti näytetään kartalla keltaisena kolmiona, jonka terävin kulma osoittaa robotin etenemissuuntaan, ja lataustelakka näytetään kartalla punaisena neliönä. Reittipisteet näkyvät kartassa vihreinä ympyröinä. Kun robotti liikkuu määritettyä reittiä, reitillä seuraavana vuorossa oleva piste korostetaan kartalla pisteen yläpuolelle sijoitettavalla vihreällä kolmiolla, ja piste korostetaan myös listassa niin ikään vihreällä värillä. Karttanäkymä reittipisteineen on esitetty alla (KUVA 11).

KUVA 11: Karttanäkymä

Yleisnäkymän alareunassa sijaitsevilla painikkeilla voidaan valita karttanäkymässä näytettäväksi joko globaali kartta (Global Map) tai paikallisen tason estekartta (Local costmap). ”Show/Hide processes” -painike näyttää tai piilottaa prosessiluettelon, jonka avulla SLAM-järjestelmän prosesseja voidaan ohjata. ”Neck Servo: On/Off” -painikkeella voidaan kytkeä robotin kaulan servomoottori tarvittaessa päälle ja pois. ”Hide-UI” -painikkeella (tai Q-näppäimellä) voidaan piilottaa käyttöliittymän elementit, jotta ne eivät olisi kamerasyötteen edessä silloin, kun robottia liikutetaan. ”UI Auto-Hide: On/Off” -painikkeella voidaan ohjata sitä, piilotetaanko em. elementit automaattisesti robottia liikutettaessa. Punaista ”Exit”-painiketta painettaessa robotin LiDAR sammutetaan ja käyttäjä ohjataan SLAM-järjestelmän alkunäkymään.

5.8 Dokumentointi

SLAM-järjestelmästä luotiin tilaajalle käytön kannalta olennaisimmat asiat sisältävä loppukäyttäjän ohje. Lisäksi tilaajalle toimitettiin jatkokehityksessä tarpeelliset tiedot sisältävä tekninen dokumentti, jossa on esitetty järjestelmän käyttämät tietoliikenneportit, tärkeimmät tiedostosijainnit ja tiedostojen käyttötarkoitus sekä etäyhteyteen tarvittavat SSH-tunnukset. Taustajärjestelmien ja käyttöliittymän koodissa on yleisellä tasolla kerrottu tärkeimpien muuttujien ja funktioiden tarkoitus.

6 TESTAUS JA JATKOKEHITYS

6.1 Robotin toiminta

Järjestelmän kehitysvaiheessa SLAM-toiminnallisuutta testattiin varsin pienessä ja suljetussa tilassa, jolloin järjestelmä toimi pääsääntöisesti siten kuin sen oli tarkoituskin. Kun järjestelmää testattiin tilaajan omissa, suuremmissa tiloissa, robotilla oli aluksi vaikeuksia kulkea suoraan pitkällä käytävällä, ja matkanteko oli varsin mutkikasta. Tähän oli todennäköisesti syynä se, että LiDAR on asennettu robotin takaosaan siten, että Ohmnin pystytanko estää osittain LiDAR:ia näkemästä eteenpäin. LiDAR:n valmistajan koodissa esteiden minimihavainnointietäisyys oli asetettu arvoon, joka on pienempi kuin LiDAR:n etäisyys pystytangosta, jolloin Ohmni päätteli pystytangon olevan este. Arvo muutettiin suuremmaksi ja LiDAR:n ohjelmisto käännettiin uudelleen, ja vaikka tämä ei luonnollisesti itse näkyvyysongelmaa korjannutkaan, robotti vaikutti toimenpiteen jälkeen liikkuvan sujuvammin.

6.2 Käytettävyys ja muokattavuus

SLAM-järjestelmän käyttöliittymästä pyrittiin tekemään mahdollisimman yksinkertainen ja helppokäyttöinen, ja muun muassa tästä syystä käyttöliittymään ei toteutettu monimutkaisia asetusvalikoita. Monet työssä käytetyistä ROS-ohjelmista ovat sellaisia, että niiden toiminnan hienosäätäminen vaatii tiettyjen tiedostojen muokkaamista ja joidenkin ohjelmien kohdalla jopa itse lähdekoodin muuttamista ja ohjelman uudelleenkääntämistä. Vaikka nämä ohjelmat ovat avoimia ja lähdekoodi helposti muokattavissa, ja siten suurin osa asetuksista olisi toki mahdollista määritellä käyttöliittymästä käsin, tällaista toimintoa ei ajallisista syistä toteutettu.

Loppukäyttäjän kannalta itse SLAM-toiminnallisuus, eli tilan kartoittaminen ja reitin määrittäminen, on kuitenkin melko suoraviivaista ja järjestelmän käyttäminen helppoa. Toiminnan hienosäätäminen kulloiseenkin ympäristöön sopivaksi on toisaalta varsin hankalaa, ja vaatii jonkin verran osaamista taustalla vaikuttavista teknologioista. Käyttöliittymän ulkoasun ja käyttöliittymäelementtien sijoittelun olisi myös voinut toteuttaa järkevämmin.

Käyttöliittymän koodi koostuu pääasiassa yksinkertaisista funktioista, joiden käyttötarkoitus on lyhyesti kuvattu lähdekoodin kommenteissa. Jatkokehityksen kannalta ongelmaksi voi muodostua se, että lähes kaikki koodi sijaitsee yhdessä tiedostossa, jonka koko on varsin suuri. Funktiot oli lopuksi tarkoitus jakaa toiminnallisuutensa mukaan useisiin tiedostoihin ja sisällyttää pääohjelmaan moduuleina, mutta ajallisista syistä näin ei tehty. Käyttöliittymän toteutuksessa ei myöskään käytetty valmiita sovelluskehyksiä (React tms.), ja asynkroniset toiminnot toteutettiin callback-periaatteella sen sijaan, että olisi käytetty selkeämpää promise-lähestymistapaa.

6.3 Jatkokehityssuunnitelmat

SLAM-järjestelmän pohjana käytetty ROS-alusta mahdollistaa monenlaisten robotiikkasovellusten kehittämisen laitteeseen. Yksi opinnäytetyön aikana esille tulleista ajatuksista oli se, että robotti voisi kulkiessaan analysoida kameran kuvasyötettä ja tunnistaa reitillä mahdollisesti vastaan tulevat ihmiset. Tällöin robottiin voisi toteuttaa ominaisuuden, jonka avulla se pystyisi hakeutumaan vuorovaikutukseen esimerkiksi palvelutalon asiakkaiden kanssa ja tarjoamaan apua mahdollisissa ongelmissa eräänlaisena liikkuvana hälytyspainikkeena.

Itse SLAM-toiminnallisuuden kannalta kehitettäviä asioita ovat ainakin seuraavat:

- Edellisissä kappaleissa mainitut käytettävyyteen ja muokattavuuteen liittyvät seikat
- USB-sarjaporttien tunnistaminen: Ohmnin ohjauslaitteiden ja LiDAR:n ”/dev” -hakemistossa olevat laitetiedostot vaihtavat toisinaan paikkoja keskenään
- LiDAR:n sijainti: Ohmnin pystytanko peittää näkyvyyden eteenpäin
- LiDAR näkee vain esteet, jotka ovat noin puolen metrin korkeudessa
- SLAM-järjestelmä toimii vain lähiverkossa, eikä yhteys ole salattu
- Kartat ja kulkureitit tallentuvat vain laitteeseen, eivätkä esimerkiksi pilvipalveluun
- Pääkameran kuvasyötteen (motion) laatu SLAM-käyttöliittymässä on heikko

Käytettävyyteen ja muokattavuuteen liittyvät ongelmat ovat ratkaistavissa paremmalla käyttöliittymäsuunnittelulla ja valmiilla sovelluskehyksillä. USB-sarjaporttien laitetiedostojen määrittäminen ei Ohmnissa toimi samalla lailla kuin tavallisimmissa Linux-jakeluissa, ja ongelman ratkaiseminen vaatisi mahdollisesti jopa kernel-kerroksen muutoksia ohjelmistoon.

LiDAR olisi mahdollista sijoittaa myös robotin pystytangon etupuolelle, mutta pystytanko ei ole suora eikä oikeanlaista kiinnikettä ollut toteutuksen aikana saatavilla. Jotta robotti havaitsisi myös lattiatason esteet, alustaan voitaisiin asentaa ultraäänitunnistin, jonka voisi määrittää toimimaan ROS-ympäristössä yhdessä navigointiohjelmiston kanssa.

SLAM-järjestelmän käyttäminen lähiverkon ulkopuolelta voitaisiin mahdollistaa useammallakin tavalla. Robottiin on asennettu USB-mobiililaajakaistasovitin eli mokkula, joka luo robotille oman WLAN-lähiverkon, ja nykyisellään loppukäyttäjän on yhdistettävä päätelaitteensa tähän verkkoon käyttääkseen SLAM-toiminnallisuutta. Suoraviivaisin ratkaisu olisi määrittää mokkula siten, että SLAM-järjestelmän käyttämät tietoliikenneportit olisivat avoinna Internetiin ja järjestelmää voitaisiin käyttää mokkulan julkisen IP:n perusteella. Tällainen ratkaisu vaatisi kuitenkin useita epäkäytännöllisiä toimenpiteitä, kuten kiinteän julkisen IP-osoitteen tai dynaamisen DNS-määrityksen asettamisen, itseallekirjoitetun SSL-sertifikaatin käyttämisen ja kaikkien avointen palveluiden suojaamisen jonkinlaisella tunnistautumismekanismilla. Toinen vaihtoehto olisi yhdistää robotti itse kehitettävään julkiseen web-palveluun, joka toimisi melko samalla tavalla kuin OhmniLabsin oma web-sovellus. Myös tällainen ratkaisu edellyttäisi useiden tietoturvakysymysten huomiointia, ja lisäksi se vaatisi huomattavia muutoksia kaikkiin jo toteutettuihin ohjelmistoihin. Toisaalta SLAM-järjestelmän etäkäyttö saattaisi osoittautua liian hitaaksi, sillä ROS:n websocket-rajapinnan läpi liikkuva tietomäärä on suuri ja tiedonsiirron tulisi käytettävyyden kannalta olla mahdollisimman reaaliaikaista.

7 TULOKSET JA POHDINTA

7.1 Tavoitteiden saavuttaminen

Toteutettu SLAM-järjestelmä mahdollistaa tilan kartoittamisen, ja robotti voidaan laittaa kiertämään kartoitettua tilaa käyttäjän määrittelemällä reitillä. Näihin alkuperäisiin vaatimuksiin toteutettu järjestelmä vastaa hyvin. Toteutus myös valmistui ennakoidussa aikataulussa muutamista teknisistä ongelmista huolimatta. Toisaalta toteutuksen viimeistely jäi pääasiassa juuri näiden teknisten ongelmien vuoksi tekemättä.

7.2 Toteutuksen haasteet

Kaikista ilmeisin haaste toteutuksessa oli se, ettei opinnäytetyön tekijällä ollut aikaisempaa tietoperustaa, saati kokemusta ROS-ympäristöstä. Varsinkin työn alkuvaiheessa suurin osa työskentelystä oli ROS:n tutkimista ja erilaisia kokeiluja. Aiheeseen liittyvää tietoa on saatavilla verkossa suhteellisen vähän verrattuna esimerkiksi tavanomaiseen web-kehitykseen. Ohmni-robotti ei myöskään ainakaan vielä ole kehittäjien yleisesti käyttämä alusta, ja suurin osa verkosta saatavasta tiedosta ei siten ollut suoraan sovellettavissa työhön.

Itse työn aikana esille tulleista haasteista merkittävin oli Ohmnin lataustelakan vioittuminen toteutuksen loppuvaiheessa. Asiointi OhmniLabsin kehittäjätuen kanssa oli kuitenkin erittäin sujuvaa ja valmistaja vaikuttaa olevan oikeasti kiinnostunut asiakkaistaan. Uuden lataustelakan saapuminen Yhdysvalloista kesti noin kaksi viikkoa, eikä robottia voinut tuona aikana käyttää. Muun muassa tästä syystä kaikkia suunniteltuja toimenpiteitä toteutuksen viimeistelyssä ei ehditty suorittamaan.

Järjestelmän testausvaiheessa Ohmni alkoi esittämään näytöllään kolmansien osapuolien mainoksia, joiden sisältö oli luonteeltaan osittain sopimatonta. Näiden taustalla oli ilmeisesti jokin laitteeseen aikaisemmin asennettu haitallinen Android-sovellus. Ohmnin käyttöjärjestelmä ja siten myös koko SLAM-toiminnallisuus katsottiin parhaaksi asentaa uudelleen, mikä aiheutti jonkin verran ylimääräistä työtä.

7.3 Opinnäytetyön merkitys

Opinnäytetyön lopputuloksena syntynyt järjestelmä laajentaa Ohmni-robotin toiminnallisuutta siten, että robottia voidaan käyttää myös muissa kuin sille alun perin suunnitelluissa tehtävissä. ROS-ympäristön asentaminen laitteeseen mahdollistaa myös sen, että uudet robotiikkasovellukset voivat käyttää työssä toteutettujen ROS-ohjelmien tuottamaa tietoa omissa toiminnoissaan. Tämä puolestaan tarjoaa tilaajalle monenlaisia jatkokehitysmahdollisuuksia robotin parissa esimerkiksi uusien opinnäytetyöaiheiden muodossa.

Opinnäytetyö oli myös tekijälleen monella tavalla hyödyllinen oppimistapahtuma. Se mahdollisti sekä useiden uusien asioiden oppimisen että myös aikaisemmin opittujen taitojen kehittämisen niillä osaamisalueilla, joihin tekijä aikoo tulevaisuudessa erikoistua.

7.4 Oppiminen opinnäytetyöprosessin aikana

Opinnäytetyö kehitti tekijän osaamista useilla käytännön osa-alueilla. Teknisistä asioista tekijälle täysin uusia olivat kaikki ROS:iin liittyvät seikat sekä robottien navigoinnissa käytetyt reittisuunnittelualgoritmit ja ympäristön havainnointiin liittyvät asiat. Myös Dockerin soveltaminen käytäntöön oli tekijälle uutta. Aikaisemmin opituista taidoista opinnäytetyö kehitti eniten sekä palvelin- että selainpuolen Javascriptin osaamista. Javascript-ohjelmoinnin kannalta työ opetti erityisen hyvin sen, miten tiettyjä asioita ei kannata tehdä jatkossa.

Sen lisäksi, että opinnäytetyö syvensi teknistä osaamista, se kehitti myös muita ohjelmistokehityksessä ja muutenkin työelämässä olennaisia taitoja. Näistä taidoista ylivoimaisesti tärkein on tiedonhankintataito. Opinnäytetyössä hyödynnetyistä ohjelmistoista ja teknologioista ei ollut juurikaan saatavilla valmiita oppaita, ja siten perinteinen googleta-kopioi-liitä -lähestymistapa ei ollut riittävä. Useimmat ongelmatilanteet selvisivät vain perehtymällä syvällisemmin ohjelmien lähdekoodiin sekä ohjelmistoista käytyihin kehittäjäkeskusteluihin.

Opinnäytetyö kehitti myös kokonaisuuksien hallintaa ja ajankäytön suunnittelua, jotka usein ovat ohjelmistokehityksessä kaikista haastavimpia osa-alueita. Näistä tärkein toteutuksen aikana ja varsinkin sen jälkeen opittu asia on, että kaikki ohjelmistokokonaisuudet tulisi suunnitella huolellisesti. Suunnittelussa tulisi myös erityisesti ottaa huomioon kokonaisuuden tilaajan näkemys siitä, mitä kokonaisuudella on tarkoitus saavuttaa. Aikataulut on lisäksi syytä suunnitella siten, että kaikki alustavat tuntimääräarviot kerrotaan vähintään kahdella, jotta toteutus ehditään myös viimeistellä tarkoitetulla tavalla.

8 LÄHTEET

AspenCore, Inc. 2019. *2019 Embedded Markets Study.* s.l. : AspenCore, Inc., 2019.

Cassel, David. 2017. Remembering Shakey, the First Intelligent Robot. *The New Stack.* [Online] 5. 3 2017. [Viitattu: 19. 11 2020.] https://thenewstack.io/remembering-shakey-first-intelligent-robot/.

Fortune Business Insights. 2019. Internet of Things (IoT) Market. *Fortune Business Insights.* [Online] 1. 7. 2019. https://www.fortunebusinessinsights.com/industry-reports/internet-of-things-iot-market-100307.

International Federation of Robotics. 2017. *The Impact of Robots on Productivity, Employment and Jobs.* Frankfurt, Germany : International Federation of Robotics, 2017.

Koubaa, Anis;ym. 2018. *Robot Path Planning and Cooperation.* s.l. : Springer International Publishing AG, 2018.

Mickle, Paul. 1998-1999. 1961: A peep into the automated future. *The Capital Century.* [Online] 1998-1999. [Viitattu: 19. 11 2020.] http://www.capitalcentury.com/1961.html.

Ohmnilabs. 2020. Ohmni Developer Manual. *Ohmni Developer Manual.* [Online] 2020. https://docs.ohmnilabs.com/.

OhmniLabs, Inc. 2020. OhmniLabs. *OhmniLabs.* [Online] OhmniLabs, Inc., 2020. [Viitattu: 22. 11 2020.] https://ohmnilabs.com/.

Robotic Industries Association. 2020. Unimate - The First Industrial Robot. *Robotics Online.* [Online] Robotic Industries Association, 2020. [Viitattu: 19. 11 2020.] https://www.robotics.org/joseph-engelberger/unimate.cfm.

ROS Wiki. 2020. Documentation - ROS Wiki. *ROS Wiki.* [Online] 2020. http://wiki.ros.org/.

Scerra, Melania. 2020. Size of the global market for industrial and non-industrial robots between 2018 and 2025. *Statista.* [Online] 16. 4. 2020. https://www.statista.com/statistics/760190/worldwide-robotics-market-revenue/.

SRI International. 1968. Shakey. *SRI International's Artificial Intelligence Center.* [Online] SRI International, 1968. [Viitattu: 3. 12 2020.] http://www.ai.sri.com/shakey/.

Synopsys, Inc. 2019. Top open source licenses and legal risk for developers. *Synopsys.* [Online] Synopsys, Inc., 21. 10 2019. [Viitattu: 19. 11 2020.] https://www.synopsys.com/blogs/software-security/top-open-source-licenses/.