Jani Nylund

ROS MOBIILISSA ROBOTISSA

Sähkötekniikan koulutusohjelma

2017

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

ROS MOBIILISSA ROBOTISSA

Nylund, Jani
Satakunnan ammattikorkeakoulu
Sähkötekniikan koulutusohjelma
Toukokuu 2017
Ohjaaja: Asmala, Hannu
Sivumäärä: 35
Liitteitä: 1

Asiasanat: ROS, Robotiikka, kauko-ohjaus

Opinnäytetyön tarkoituksena oli tutustua ROS, Robot Operating System, avoimen lähdekoodin järjestelmään ja kehittää tämän avulla ominaisuuksia mobiiliin robottiin. Työn lähtökohtana toiminut kehitysrobotti Samkbot yhdistettiin kuluttajatason RGBD-sensoriin ja tehokkaaseen kehitysalustaan, joka toimii robotin prosessointiyksikkönä. Näiden avulla tehtävänä oli implementoida ROS:ia vahvasti hyödyntävä kauko-ohjaus sekä mahdollistaa robotin ympäristön kartoitus. Opinnäytetyö kertoo ROS-järjestelmän käytännön implementoinnista mobiilin robotin näkökulmasta ja esittelee järjestelmän kannalta oleellisia ROS:in ominaisuuksia.

ROS IN A MOBILE ROBOT

Nylund, Jani
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Degree Programme in Electrical Engineering
May 2017
Supervisor: Asmala, Hannu
Number of pages: 35
Appendices: 1

Keywords: ROS, robotics, remote control

The purpose of this thesis was to explore ROS, Robot Operating System, open source robotics system and use it to develop features for a mobile robot. The development robot Samkbot used as a baseline for this thesis was combined with a consumer grade RGBD-sensor and a powerful development platform which functioned as the processing unit. With this hardware the task was to implement remote control and mapping functionalities utilizing heavily on ROS. This thesis tells about a practical implementation of ROS from the view of mobile robot development and showcases related key features of ROS.
.

SISÄLLYS

KÄSITELUETTELO

OSFR	Open Source Robotics Foundation, Inc. on maailmanlaajuisen robottiyhteisön perustama, itsenäinen ja voittoa tavoittelematon organisaatio.
ROS	Robot Operating System, avoimen lähdekoodin metakäyttöjärjestelmä robottien ohjelmointiin.
XML	Extensible Markup Language, merkintäkielistandardi, jota käytetään formaattina tiedonvälitykseen järjestelmien välillä ja dokumenttien tallentamiseen.
JSON	Javascript Object Notation, kevyt ja tekstipohjainen tiedonvaihtoformaatti.
RGBD	Värillisen (RGB) videokuvan ja syvyystiedon (D) yhdistelmä.
URDF	Unified Robot Description Format on robottien kuvailemiseen tarkoitettu XML-spesifikaatio.

1 JOHDANTO

Robottien ohjelmointiin tarkoitettu avoimen lähdekoodin metakäyttöjärjestelmä ROS, Robot Operating System, on viime vuosina saavuttanut huomattavaakin suosiota. Ohjelmisto on ollut käytössä jopa kansainvälisellä avaruusasema ISS:llä ja sen pohjalta on rakennettu erilaisia robotiikan parissa työskenteleviä yrityksiä. Näin ROS olikin luonnollinen valinta kehitysrobotti Samkbotin pohjaksi.

Samkbotin kannalta työn tavoitteena oli saada robotti yhdistettyä ROS-järjestelmään ja tämän avulla luoda ympäristöstä kartta, sekä toteuttaa manuaalinen kauko-ohjaus. Robotin tuli pystyä toimimaan sille ennestään tuntemattomassa ympäristössä pystyen samalla paikallistamaan itsensä suhteessa tuohon ympäristöön. Ominaisuuksien toteuttamiseen pyrittiin käyttämään mahdollisimman paljon valmiita ROS-paketteja, sekä ROS-yhteisön ympärille luotuja työkaluja. Näiden pohjalta tarkoituksena oli luoda yhtenäinen toiminnallinen kokonaisuus. Liikkuvana alustana toiminut mobiili robotti Samkbot yhdistettiin kuluttajatason RGBD-sensoriin, tehokkaan prosessorin omaavaan vähävirtaiseen kehitysalustaan sekä moottoriohjaimen ja mittalaitteiden kanssa kommunikoivaan Arduino-laitteeseen.

Opinnäytetyö esittelee ROS-järjestelmän ominaisuuksia mobiilin robotin näkökulmasta ja kertoo ROS:in käytännön implementoinnista kehitysrobotti Samkbottiin. Työn tuloksina syntyi robotin ohjaamisen mahdollistava ROS-paketti, Javascript-kieltä hyödyntävä ulkoisen kauko-ohjauksen käyttöliittymä sekä suunnitellut ominaisuudet toteuttava ja automaattisesti käynnistyvä ROS-pakettien ketju.

2 ROS

Robot Operating System (*ROS*) on avoimen lähdekoodin metakäyttöjärjestelmä robottien ohjelmointiin. Alunperin Willow Garagen ja Stanfordin yliopiston yhteistyönä kehittämä ROS pyrkii helpottamaan robottiohjelmistojen suunnittelua hyödyntämällä yhteisöllisyyttä ja koodin uudelleenkäyttöä. Se pitää sisällään muun muassa erilaisia työkaluja, laiteajureita, ohjelmistokirjastoja sekä käytäntöjä. ROS muodostaa prosessien vertaisverkon, jossa ohjelmistot kommunikoivat joustavasti keskenään käyttäen erilaisia viestintäjärjestelmiä. Nykyisellään UNIX-pohjaisille järjestelmille suunniteltua ROS:ia ylläpitää OSFR eli Open Source Robotics Foundation. (ROS www-sivut 2016.)

ROS-Industrial on avoimen koodin projekti, joka laajentaa ROS:in käyttöä teollisen automaation ja valmistuksen pariin. Se pyrkii hyödyntämään ROS:in parhaita puolia yhdessä teollisuuden turvallisten robottikontrollereiden kanssa luoden vankkaa ohjelmistoa, jonka laatu täyttää teollisen valmistuksen standardit. Projekti ei pyri korvaamaan olemassaolevia tekniikoita vaan hyödyntämään ROS:in korkean tason ominaisuuksia kuten 2-D- ja 3-D-havainnointia, käänteistä kinematiikkaa ja liikesuunnittelua vähentääkseen kehitystyökuluja ja yksinkertaistaakseen robottien ohjelmointia tehtävätasolle. Projekti sisältää rajapintoja yleisille teollisuuden sensoreille, manipulaattoreille, tarttujille ja laiteverkoille. ROS-Industrial toimii kansainvälisen ROS-Industrial Konsortion tukemana. Konsortio tarjoaa jäsenilleen kulujaettua sovellettua tutkimus- ja kehitystyötä kehittyneeseen tehdasautomaatioon. Organisaatioon kuuluu useita tunnettuja yrityksiä kuten esimerkiksi Ford, Siemens, ABB ja 3M. (ROS-Industrial www-sivut 2016.)

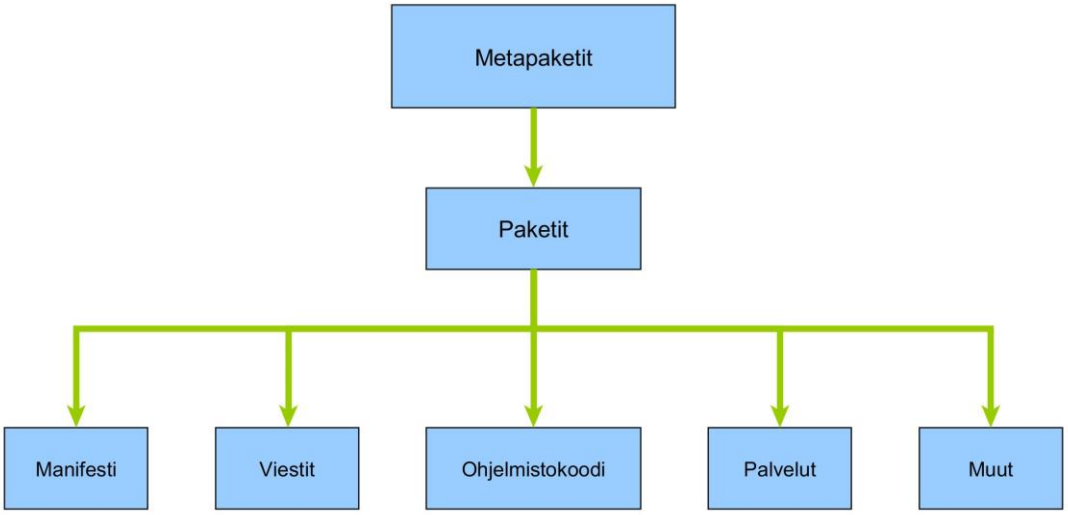
2.1 Rakenne ja käsitteet

ROS:in rakenteen muodostavat kolme konseptitasoa, joista jokaisella on oma tehtävänsä. Tiedostojärjestelmätaso kertoo lähinnä miten tiedostot on järjestelty kovalevyllä. Laskentakuvaajataso (*Computation Graph Level*) on ROS-prosessien vertaisverkko, jonka kaikki osaset tuottavat dataa kuvaajalle. Tämä on keskeinen osa ROS-

järjestelmää. Yhteisötaso kerää yhteen ROS-resurssit, jotka mahdollistavat erilaisten ROS-yhteisöjen ohjelmisto- ja tiedonjakamisen. (ROS www-sivut 2016.)

2.1.1 Tiedostojärjestelmätaso

Paketit (*packages*) ovat ROS:in päämuotoisia rakennuspalikoita. Niillä ohjelmistot järjestetään tiettyyn muotoon. Paketit voivat sisältää muun muassa ohjelmatiedostoja, ohjelmistokirjastoja, asetustiedostoja tai mitä tahansa loogisesti samaan paikkaan järjesteltävää. Metapaketit ovat erikoispaketteja, joita käytetään edustamaan usean toisiinsa liittyvän paketin ryhmää. Näitä on aiemmin kutsuttu myös pinoiksi (*stacks*). Manifestit ovat tiedostoja, jotka kuvailevat paketteja ja metapaketteja sekä antavat tiedon mahdollisista lisensseistä, pakettien riippuvuussuhteista ja versioista. Viestitiedostot (*msg*) sisältävät viestien kuvaukset ja määrittelevät tietorakenteet ROS:in käyttämille viestityyppeille. Palvelutiedostot (*srv*) kuvaavat ROS-palveluita ja määrittelevät niiden käyttämien pyyntö-vastaus parien tietorakenteet. Kuvassa 1 on havainnollistettu tiedostojärjestelmätason rakennetta. (ROS www-sivut 2016.)



Kuva 1. ROS tiedostojärjestelmätason periaatteellinen rakenne.

2.1.2 Laskentakuvaajataso

ROS:issa laskentaa suorittavia prosesseja kutsutaan solmuiksi (*node*). Nämä mahdollistavat modulaarisen tavan suunnitella robottiohjelmistoja. Monimutkaistakin järjestelmää voidaan yksinkertaistaa jakamalla laskentatehtävät osiin. Esimerkiksi yksittäinen solmu voi hoitaa robotin moottoreiden hallitsemisen, toinen paikannuksen ja kolmas tarjota visuaalisen näkymän järjestelmästä. Prosessien ei myöskään tarvitse sijaita samalla prosessointiyksiköllä. Solmujen luoma vertaisverkko tuo ohjelmistoon vikasietoisuutta kun yksittäiset ongelmat eivät kaada koko järjestelmää. ROS-solmuja ohjelmoidaan käyttäen asiakasohjelmakirjastoja kuten roscpp tai rospy. Asiakasohjelmakirjastot tarjoavat ohjelmointirajapintoja erilaisille ohjelmointikielille kuten C++, Python tai Lisp.

ROS hyödyntää isäntä-renki -järjestelmää, jossa isäntä (*master*) toimii laskentakuvaajatason nimipalvelimena. Isäntä kommunikoi solmujen kanssa, tallentaa niiden rekisteröintitiedot sekä antaa solmuille tavan löytää muita järjestelmän solmuja ja kommunikoida keskenään. Solmujen varsinainen tiedonvaihto tapahtuu suoraan solmujen välillä. Isäntä sisältää myös ROS-parametripalvelimen (*parameter server*), joka on jaettu monimuuttujasanakirja. Parametripalvelimen avulla solmut voivat noutaa, tallentaa ja muokata ajonaikaisia parametrejä. Palvelin on pääosin tarkoitettu staattisen datan kuten järjestelmän konfiguraatioparametrien tallentamiseen ja mahdollistaa hakuavaimella saavutettavan tiedon säilyttämisen keskitetyssä paikassa. (ROS www-sivut 2016.)

Solmut kommunikoivat keskenään viesteillä. ROS-viestit ovat yksinkertaisia tietorakenteita. Ne koostuvat listasta tyypitettyjä tietokenttiä, jotka voivat sisältää myös sisäkkäisiä tietorakenteita. Käytännössä yksittäinen kenttä koostuu tietokentän nimestä ja tietotyypistä. Tietotyyppinä voidaan käyttää esimerkiksi aikaisemmin määriteltyä viestityyppiä tai ROS:in primitiivisiä tietotyyppejä kuten kokonaislukuja, liukulukuja ja merkkijonoja. Viestit kulkevat solmulta toiselle nimettyjä väyliä pitkin. Näitä väyliä kutsutaan ROS:issa aiheiksi (*topic*). (ROS www-sivut 2016.)

ROS-aiheet hyödyntävät anynyymia tilaaja-julkaisija -mallia, jossa julkaisija ja tilaaja eivät ole tietoisia toisistaan. Kun solmu lähettää viestin aiheeseen, siitä tulee ai-

heen julkaisija (*publisher*). Kun solmu haluaa vastaanottaa viestejä, sen tulee toimia aiheen tilaajana (*subscriber*). ROS-aiheissa tiedon tuottaminen ja kulutus eivät ole kytketty toisiinsa. Näin samalla aiheella voi olla useita julkaisijoita ja tilaajia. ROS vaatii ainoastaan käytettyjen viestityyppien olevan samoja. Tämä varmistetaan vertailemalla viestitiedostojen (*msg*) MD5-tarkistussummia toisiinsa. Viestien kuljetukseen ROS:issa voidaan käyttää joko TCP/IP- tai UDP-pohjaista siirtoyhteyttä. Tämän lisäksi voidaan kuitenkin käyttää tai luoda erilaisia paketteja, jotka toimivat tiedonvälittäjinä ROS-järjestelmän ulkopuolelle ja voivat näin hyödyntää myös muita viestintäprotokollia. (ROS www-sivut 2016.)

Usein robotiikan sovelluksissa tarvitaan myös niinsanotun pyyntö-vastaus -mallin mukaista vuorovaikutusta. Tähän ROS:in ratkaisu ovat palvelut (*service*). Ne koostuvat kahdesta viestirakenteesta, jotka määritellään palvelutiedostossa. ROS-palveluissa yksi solmu toimii palvelimena ja tarjoaa palvelua. Kun toinen solmu haluaa käyttää palvelua, se lähettää ensimmäiselle palvelupyynnön ja jää odottamaan vastausta. Jos palvelurutiini onnistuu, palvelinsolmu vastaa määritellyn viestirakenteen mukaisesti. Myös ROS-palveluiden validoinnissa hyödynnetään MD5-tarkistussummia. (ROS www-sivut 2016.)

Eritoten robottien kehitys- ja testausvaiheessa on hyödyllistä pystyä tallentamaan tai uudelleenkäyttämään järjestelmässä liikkuvaa dataa. Tätä varten on olemassa ROS-laukku (*bag*). Se on ROS:in pääasiallinen tiedostoformaatti viestien tallentamiseen ja kirjaamiseen. Laukuilla tietoa voidaan toistaa uudelleen, muokata, analysoida ja visualisoida. Laukku voidaan kytkeä osaksi ’elävää’ järjestelmää, jolloin se käyttäytyy kuin mikä tahansa muu tietoa lähettävä solmu. Bag-tiedostoja voidaan käyttää myös tiedon pidempiaikaiseen säilytykseen sillä ROS tarjoaa hyvät työkalut niiden ylläpitämiseen ja päivittämiseen. (ROS www-sivut 2016.)

Jotta laskentakuvaajatason toiminta olisi mahdollista, käyttää ROS jokaiselle resurssille, viestille, palvelimelle ja osaselle omaa nimeä. Nämä ovat ROS:in nimiresursseja (*graph resource name & package resource name*). Nimet antavat järjestelmälle ominaisuuden tunnistaa ja jakaa resursseja. Nämä voivat myös toimia toisten nimien alaisuudessa eli niinsanotuissa nimiavaruuksissa. Tämä mahdollistaa ROS:in muodostaman hierarkisen järjestelmärakenteen. (ROS www-sivut 2016.)

2.1.3 Yhteisötaso

ROS-jakelut (*distribution*) pitävät sisällään versioituja kokoelmia metapaketeista, joiden tarkoitus on helpottaa käyttäjää asentamalla samalla kertaa yhteensopivat päivitetyt ROS:in pääohjelmakirjastot sekä mahdolliset suositellut ohjelmakirjastot. Jakelupaketit pyritään julkaisemaan jokaisen vuoden toukokuussa. Parillisina vuosina julkaistaan niinsanottu LTS- eli 'Long Term Support'-jakelu, jota pyritään ylläpitämään viisi vuotta julkaisusta. Parittomina vuosina taas julkaistaan normaali jakelu, jota tuetaan noin kaksi vuotta. Julkaisutahdin on tarkoitus kulkea käsi kädessä ROS:in pääsääntöisen käyttöjärjestelmän (*Ubuntu*) julkaisujen kanssa. Erilaisten instituuttien ylläpitämät tietovarastot (*repository*), joissa ylläpidetään ohjelmakirjastoja, paketteja ja jakeluita, ovat yksi ROS:in tärkeistä yhteisötason elementeistä. (ROS www-sivut 2016.)

Pääasiallisena dokumentaatiolähteenä ROS käyttää yhteisöllistä avointa tietosanakirjamallia (*ROS Wiki*). Käyttäjät voivat luoda tilin ja lisätä oman dokumentaationsa tai päivitttää ja muokata jo olemassa olevaa tietoa. Ohjelmistovirheiden ilmoittamiseen ROS:issa käytetään lipukejärjestelmää (*ticket*), jossa jokainen ilmoitus kirjataan ja avataan lipukkeeksi kyseisen paketin kehittäjille mahdollisia korjauksia ja kommentteja varten. Uusien päivitysten pääkeskustelukanavana käytetään ROS-postituslistaa (*mailing list*). Kysymyksiä ohjelmiston käytöstä voidaan kysyä postituslistalla tai erityisesti ROS kysymysfoorumilla (*ROS answers*), jossa toiset käyttäjät ja ylläpitäjät voivat vastata kysymyksiin ja kertoa omia neuvojaan. Foorumin kysymykset näkyvät kaikille ja se onkin yksi tapa etsiä ratkaisuja käyttäjän kohtaamiin ongelmiin. Yhteisötasoon kuuluu myös ROS:in virallinen blogi, johon lisätään uutisia, kuvia ja videoita yhteisön toiminnasta. (ROS www-sivut 2016.)

2.2 Korkeamman tason käsitteet

ROS:in ydin pyrkii toimimaan sitomatta käyttäjää yhdenlaiseen ohjelmistoarkkitehtuuriin. Se tarjoaa erilaisia tiedonvälitysmuotoja säätämättä niille tiukkoja käyttö- tai nimirajoitteita. Tämä mahdollistaa järjestelmän helpon implementoinnin erilaisiin

arkkitehtuureihin. Rakennettaessa suurempia järjestelmiä, korkeamman tason konseptit ovat kuitenkin tarpeellisia. (ROS www-sivut 2017.)

2.2.1 Actionlib – Keskeytettävät tehtävät

ROS-paketti Actionlib määrittelee standartoidun tavan työskennellä keskeytettävien tehtävien parissa. ROS-toiminnot (*action*) tarjoavat työkalut tavoitekeskeisten ja pitkäaikaisten toimintojen kuten esimerkiksi ovenkahvan tunnistukseen tai navigointiin. Toiminnot määritellään noudattaen erityistä toimintospesifikaattia, joka rajaa halutun tavoitteen, tehtävänaikaisen palautteen sekä vain kertaalleen lähetettävän toiminnon tuloksen. Tavoite voi pitää sisällään esimerkiksi navigoinnin kohteen. Tässä tapauksessa palautteeksi voidaan määritellä robotin ajonaikainen sijainti ja tulokseksi robotin lopullinen asento. Valmis spesifikaatti talletetaan erityiseen action-tiedostoon. Actionlib-paketti hyödyntää ROS-aiheiden varaan rakennettua protokollaa, jossa asiakas (*ActionClient*) ja serveri (*ActionServer*) kommunikoivat yksinkertaisen rajapinnan välityksellä. Asiakas voi pyytää serveriä toteuttamaan toimintoja lähettämällä tavoitteita sekä kuunnella serverin lähettämää palautetta toiminnon etenemisestä. Tarvittaessa asiakas voi pyytää serveriä perumaan pyydetyn tavoitteen. (ROS www-sivut 2017.)

2.2.2 Common_msgs - Keskeiset viestityypit

Common_msgs -metapaketti pitää sisältään laajalti muissa ROS-paketeissa käytössä olevia viestityyppejä ja määrittää näin niiden sisällön rajat luomatta turhia pakettien välisiä riippuvuussuhteita. Paketin viestejä käytetään esimerkiksi ROS-tehtävissä, diagnostiikassa, navigoinnissa, sensoreiden datan välityksessä, videokuvassa. pistepilvissä sekä geometristen primitiivien esittämisessä. Tullakseen hyväksytyksi osaksi common_msgs –pakettia, uuden viestipaketin tulee olla ollut aktiivikäytössä ROS:in sisällä, testattu ja arvosteltu. (ROS www-sivut 2017.)

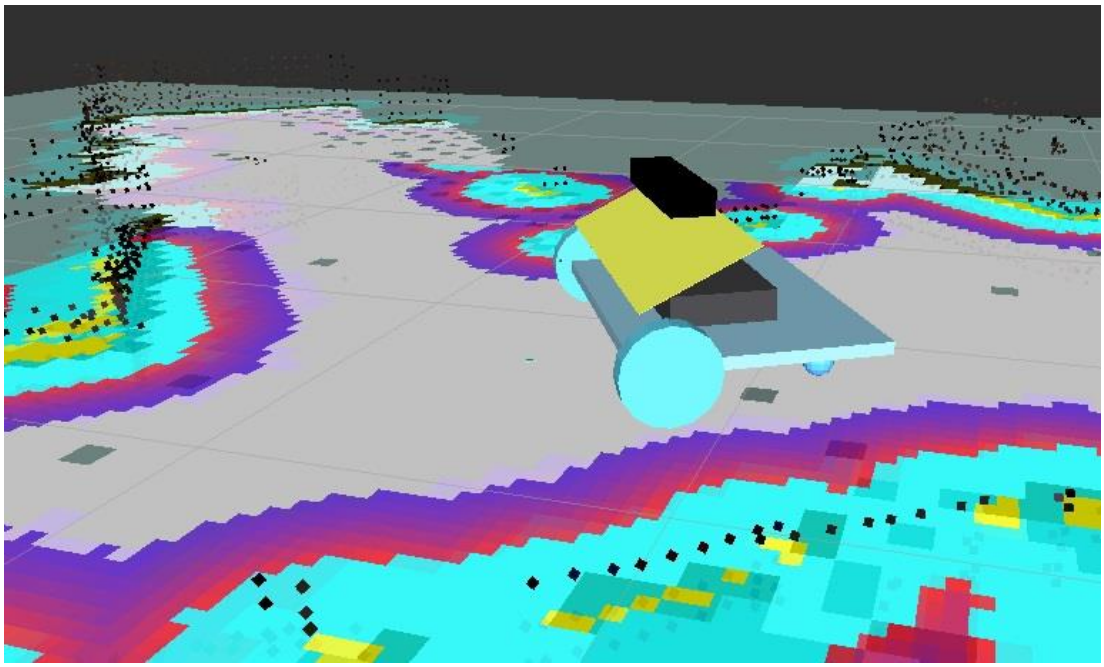
2.2.3 URDF – Robotin kuvaus

URDF eli Unified Robot Description Format on robottien kuvailemiseen tarkoitettu XML-spesifikaatio, joka kuvailee robottia liikkuvien nivelten (joint) ja jäykkien yhteiden (link) yhdistelmänä. Elementit muodostavat niin sanotun hierarkisen puurakenteen. Jokaisella elementillä, ensimmäistä ja viimeisiä lukuun ottamatta, on aina ainoastaan yksi edeltäjä ja yksi tai useampi seuraaja. Ensimmäisellä elementillä ei ole edeltäjää ja tätä kutsutaan juurielementiksi. Viimeisillä elementeillä on yksi edeltäjä, mutta ei seuraajia. Yhde-elementtejä kuvataan tarkemmin lisäelementeillä, jotka antavat kiinteille kappaleille ominaisuuksia kuten hitaus (inertial) , visuaalinen ulkoasu (visual) ja törmäysgeometria (collision). Jokaisella lisäelementillä on omia alaelementtejään, jotka määrittelevät ominaisuudet mm. kiinteisiin muotoihin, lukuihin, väreihin ja suhteellisiin koordinaatteihin. Myös nivelelementtejä määritellään lisäelementeillä ja näiden alaelementeillä, jotka kuvailevat mm. nivelen tyyppiä, dynaamisia ominaisuuksia ja käytön turvarajoja. Yhdessä URDF-elementit muodostavat kinemaattisen ja dynaamisen kuvauksen robotista, tämän visuaalisen esityksen sekä törmäysmallin. Spesifikaatio sisältää myös muita elementtejä ja laajennoksia robotin toiminnan kuvaamiseen ja yhdistämiseen erilaisten ROS-pakettien kanssa. Tämän lisäksi kustomoituja elementtejä voidaan luoda ja hyödyntää käyttämällä XML-formaattia jäsentelevää ohjelmaa normaalin standardin ulkopuolisten elementtien tulkitsemiseksi. (ROS www-sivut 2017.)

Laajojen robottikuvausten kanssa työskentelyyn voidaan hyödyntää XML-makrokieltä nimeltä Xacro. Sen avulla voidaan kirjoittaa lyhyempiä XML-rakenteita, jotka laajenevat suuremmiksi kuvauksiksi ja mahdollistavat näin rakenteiden uudelleenkäytön kuvausten sisällä. Xacron avulla voidaan määritellä toistuvia arvoja sekä yksinkertaisia matemaattisia lausekkeita helpottamaan luodun URDF mallin muokkausta ja ymmärtämistä. Lisäksi on mahdollista käyttää ehdollisia if-lausekkeita lisäämään logiikkaa robottikuvauksen laatimiseen. Lopullisesta Xacro-kuvauksesta muodostetaan URDF-tiedosto erillisellä komennolla. (ROS www-sivut 2017.)

Visualisiointia varten Samkbotista laadittiin yksinkertainen simuloitava URDF-malli Xacroa hyödyntäen. URDF kuvaa Samkbotin ulkomuotoja yksinkertaisten kuvioiden ja muotojen perusteella. Samkbotin yksinkertaistettua URDF-mallia on havainnollis-

tettu kuvassa 2. Mallia voidaan käyttää esimerkiksi visualisoitaessa robotin kulkua kartalla. Tällöin simulointiohjelmassa nähdään liikkuvan URDF:stä muodostettu 3D-approksimaatio. Varsinainen muunnos Xacro-kuvauksesta URDF-malliksi tapahtui ROS-paketilla nimeltä xacro.



Kuva 2. Kuvakaappaus Rviz-ohjelmasta. Samkbotin URDF-malli visualisoituna kartalle, jossa näkyvät myös kartoitukseen käytetty pistepilvi sekä kulukartta.

2.2.4 TF - Koordinaatistomuunnokset

Fyysisen ympäristönsä kanssa vuorovaikutuksessa olevan robotin on usein tärkeää tietää miten sen kukin osa liikkuu suhteessa ulkopuoliseen maailmaan ja itseensä. Tiedon saavuttamiseksi tulee robotilla olla jonkinlainen koordinaattijärjestelmä. Järjestelmän kasvaessa useiden koordinaattikehysten ja niiden välisten tietomuunnosten huomioiminen hankaloituu. TF-kirjasto on suunniteltu pitämään kirjaa koordinaattikehyksistä ja mahdollistamaan näiden välisten tietomuunnosten kyselyn ilman, että kyselyn suorittajan täytyy olla tietoinen kaikista muista järjestelmän käyttämistä kehyksistä. Kirjaston avulla esimerkiksi robotin sensoreiden havaitseman esineen mittaustieto voidaan muuntaa esinettä käsittelevän robotin osan koordinaatistoon. Koska esine ja käsittelijä ovat nyt samassa kehyksessä, on mahdollista laskea kuinka paljon

käsittelevän osan täytyy liikkua koskettaakseen esinettä. TF tukee kyselyiden suorittamista paikan lisäksi myös tietyn aikaikkunan sisällä. (Foote 2013.)

Käytännössä ROS-koordinaatistomuutosten ymmärtämisen huomattiin olevan erittäin tärkeää työskenteltäessä lähes minkä tahansa ROS-paketin kanssa. Tämä ei kuitenkaan tarkoita TF-kirjaston matemaattisten konseptien tai sisäisten komponenttien ymmärrystä vaan yleisesti käytössä olevien konventioiden hahmottamista. Käyttäjälle vaikeimmat konseptit on abstraktoitu pois ja koordinaatistomuunnosten kulun ymmärtämistä helpottamaan on luotu ohjeita REP:ien muodossa. Näiden ohjeiden ja konventioiden ymmärtämisen lisäksi tulee ottaa huomioon käytettyjen ROS-pakettien tarjoamat ”rajapinnat”. Jokainen ROS-paketti määrittelee itse miten toimii TF-kirjaston kanssa. Tämä vuorovaikutus ja sen luomat seuraukset sekä vaatimukset on yleensä kerrottu paketin ROS-wiki-sivuilla. Esimerkiksi Samkbotissa TF-kirjaston kanssa kommunikoi muutoksen karttakehyksestä odometriakehykseen julkaiseva RTABMAP, odometriakehyksestä robotin peruskehykseen julkaiseva Samkbot-ajuri tai visuaalinen odometriaohjelmisto sekä kiinteitä, samana pysyviä, muutoksia julkaisevat paketit. Lopulliseksi TF-puuksi muodostunutta järjestelmään on havainnollistettu liitteessä 1.

REP:t eli ROS parannusehdotukset ovat ROS:in sisäisiä suunnitteludokumenttejä, jotka tarjoavat tietoa ROS-yhteisölle. Niitä on yhteensä kolmea erilaista tyyppiä. Yksi kuvaamaan uusia piirteitä ja tekniikoita, yksi tarjoamaan yleistä ohjeistusta tai tietoa ja yksi kuvaamaan jotakin ROS:iin liittyvää prosessia tai muutosta. REP:n laatijan yhtenä tehtävänä on pyrkiä rakentamaan dokumentit siten, että ROS-yhteisö voi olla sisällöstä yksimielinen. Mobiilin robotin suunnittelijan näkökulmasta erityisesti ehdotukset numero 103 ja 105 ovat arvokasta tietoa. REP 103 kuvaa ROS:in sisällä käytettyjä standardiyksiköitä sekä koordinaattikehyksiin liittyviä konventioita. REP 105 pitää sisällään tietoa erityisesti mobiilien robottien koordinaattikehyksiin liittyen. Se määrittelee myös yleisesti käytettyjä kehyksiä sekä näiden välisiä suhteita ja kartoitukseen liittyviä konventioita. (ROS www-sivut 2017.)

2.3 ROS-työkalut

2.3.1 Roscore

Roscore on ohjelmakokoelma, joka toimii ROS-pohjaisen järjestelmän ytimenä. Se käynnistää ROS-isännän, parametriserverin ja *rosout* tapahtumahistorian eli lokin keräämisen. Roscore on minimivaatimus solmujen väliseen kommunikointiin ja ROS-järjestelmän toimintaan. (ROS www-sivut 2016.)

2.3.2 Roslaunch & launch-tiedostot

Roslaunch on työkalu ROS-käynnistystiedostojen lukemiseen ja suorittamiseen. Näiden *launch*-tiedostojen avulla voidaan käynnistää ohjelmia, solmuja, sekä asettaa järjestelmän parametrejä tai muita tarvittavia ominaisuuksia. Käynnistystiedostoja laaditaan ROS:in määrittelemässä HTML-kieltä muistuttavassa XML-formaatissa. Roslaunch:in arkkitehtuuri nojaa vahvasti ROS:in periaatteeseen, jossa monimutkaisia järjestelmiä rakennetaan yhdistämällä useita yksinkertaisia järjestelmiä. Useat ROS-paketit hyödyntävät tätä ominaisuutta ja sisältävät omat käynnistystiedostonsa, joita on mahdollista linkittää käyttäjän omaan launch-tiedostoon. Näin voidaan helposti rakentaa kokonaisia järjestelmiä, jossa yhdellä roslaunch-käskyllä käynnistetään kaikki, jopa usealla prosessointiyksiköllä sijaitsevat, järjestelmän vaatimat ohjelmistot kerralla. Roslaunch käynnistää myös roscoren, mikäli sellaista ei ole käynnistetty. (ROS www-sivut 2016.)

2.3.3 Catkin

Catkin toimii ROS:in virallisena ohjelmistorakennusjärjestelmänä. Se on yhdistelmä avoimen lähdekoodin CMake-ohjelmistorakennustyökalun makroja sekä Python-kielisiä skriptejä. Catkiniä käytetään muodostamaan ROS-pakettien rakennuksessa tarvittavat käännöskohteet, tiedostot ja ohjelmistot ohjelmakoodista sekä ohjaustiedostoista. Sen avulla muodostetaan ROS-pakettien välisiä riippuvuussuhteita sekä paketin ja sen rakennusympäristön riippuvuussuhteita. Catkin korvaa aikaisemmin

ROS:ssa käytetyn rosbuild-työkalun ja sitä voidaan käyttää myös täysin ROS:in ulkopuolella. Toimintaan vaikuttavat erityisesti erikseen rakennettavan paketin sisällä määritellyt CMakeLists.txt ja package.xml –tiedostot. Ensimmäinen pitää sisällään rakennusprosessin valmistavat ja toteuttavat tiedot. Toinen, eli package.xml, antaa tietoa paketista ja sen riippuvuussuhteista. (ROS www-sivut 2017.)

2.3.4 Rviz

ROS-työkalu Rviz helpottaa datan visualisointia. Se toimii yleisenä 3D-visualisointiympäristönä roboteille, sensoreille ja algoritmeille. Koska ROS:in sisällä data on yleensä liitetty johonkin referenssikehykseen tai koordinaatistoon, voidaan datan visualisoinnista tehdä intuitiivisempää tarkastelemalla sitä esimerkiksi samassa näkymässä kuin robotin 3D-mallia. Rviz mahdollistaa graafisten näkymien nopean suunnittelun tarjoamalla käyttöliittymäänsä erilaisia konfiguroitavia paneeleja sekä valikoiman liitännäisiä. Näkymät voidaan myös tallentaa ja käyttää uudelleen. Rviz pystyy visualisoimaan useita ROS-tietotyyppejä sisältäviä datavirtoja.
(Quigley, Gerkey & Smart 2015, 126-127.)

Samkbotissa Rviz:iä hyödynnettiin eritoten testausvaiheessa ja näin voitiin visuaalisesti varmistaa esimerkiksi kartoituksen toimiminen. Yleisimmin käytetty Rviz-konfiguraatio tallennettiin Samkbot-paketin sisälle mahdollista myöhempää käyttöä varten. Konfiguraatio visualisoi Kinectin syvyyskuvasta muodostettua Laserscan-viestiä ja PointCloud2-pistepilviviestiä sekä näyttää robotin URDF-mallin sijoitettuna oikeaan asemaan Rtabmapin muodostamalla kartalla. Kokoonpano näyttää myös move_base:n suunnitteleman polun haluttuun kohteeseen. Rviziä voidaan käyttää visualisointiin myös robotin ulkopuolelta ROS:in muodostaman verkon avulla, kunhan otetaan huomioon tietoverkon kaistarajoitukset. Esimerkiksi 3D-kartan esittäminen pistepilven avulla saattaa käyttää huomattavia määriä dataa ja hidastaa verkon sekä mahdollisesti rajallisen prosessointikyvyn omaavan robotin toiminnan epäkäytännölliselle tasolle.

2.3.5 Rqt

Rqt on ohjelmistokehys, joka mahdollistaa erilaisten graafisten käyttöliittymätyökalujen käytön lisäosien muodossa. Työkalujen avulla voidaan olla vuorovaikutuksessa robotin kanssa, kuunnella ja julkaista erilaisia ROS-viestejä, diagnosoida ongelmia sekä kirjata ja tarkastella ajonaikaisia järjestelmätietoja ja kokoonpanoa. Ohjelmistokehyksen sisältämiä työkaluja voidaan käyttää myös erillään, mutta ohjelmiston mukana tulevat lisäosat keräävät yhteen yleisesti ROS-ympäristössä käytettyjä työkaluja helposti käytettävään muotoon. (ROS www-sivut 2017.)

Rqt:n työkaluja hyödynnettiin Samkbotin luomisessa viestien tarkastelussa. Julkaistut videokuvaa toimittavat aiheet voitiin nähdä videona käyttöliittymässä ja tekstimuotoisten viestien sisältöä sekä julkaisunopeuksia voitiin tarkkailla samalla rajoittaen käyttäjälle kerrallaan näytettävän tiedon määrää. Erityisen hyödyn Rqt tarjosi nopeaan ongelmanratkaisuun viestien tarkkailun muodossa sekä TF-muunnosten visualisoinnissa.

3 SAMKBOT

Toiminnallisuuden tuottamiseksi Samkbottiin tuli tehdä muutamia ulkoisia muutoksia. Alunperin tehoiltaan hyvin rajallinen prosessointiyksikkö korvattiin tehokkaammalla Wandboard-yksiköllä. Antureiden ja toimilaitteiden ohjaukseen liitettiin Arduino Leonardo –mikrokontrolleri, joka toimii rajapintana fyysisen laitteiston ja Wandboardin pyörittämän ohjelmiston välillä. Lisäksi järjestelmään liitettiin kuluttajatason RGBD-kamera Microsoft Kinect, jota hyödynnetään kartoitukseen ja kaukoohjauksen visualisointiin. Näiden lisäksi järjestelmän virransyöttöön täytyi lisätä hyvän hyötysuhteen omaava viiden voltin tasajännitemuunnin ja tarvittavat johdotukset laitteille.

3.1 Moottorit ja moottoriohjain

Samkbotin voimanlähteenä toimii kaksi Devantech Ltd:n valmistamaa 24 voltin EMG49 –vaihteistomoottoria, jotka pitävät sisällään integroidut pulssianturit pyörimisnopeuden mittaamiseksi. Yksi moottori pystyy tuottamaan noin 16 kg/cm momentin nopeudella 122 kierrosta/min. Moottoreiden ohjaamiseen käytetään erityisesti EMG49:än kanssa käytettäväksi suunniteltua Devantech MD49 –moottoriohjainta. Laite on sarjaliikenneohjattu kahden moottorin H-siltaohjain, joka kykenee lukemaan moottoreiden pulssiantureita ja lähettämään tiedon eteenpäin. MD-49 kykenee säätämään moottorien tehoja dynaamisesti halutun nopeuden saavuttamiseksi kuuntelemalla pulssiantureiden tuottamaa dataa. Ohjain pystyy mittaamaan moottoreiden virtaa ja suojaamaan niitä oikosuluilta, ylivirralta ja haitallisilta jännitteiltä. Moottoriohjaimen säätö tapahtuu sarjaliikenneyhteydellä. Sen käskytys sekä tietojen kysely tapahtuu lähettämällä ensin nollatavu ja tämän jälkeen vastaava käskytavu ja mahdollinen asetusarvotavu. Tämän jälkeen laite vastaa lähettämällä mahdolliset vastaustavut takaisin käskyttäjälle. (Robot Electronics www-sivut 2016.)

3.2 Arduino Leonardo

Arduino Leonardo on Atmelin valmistamaan Atmega32u4 –mikrokontrolleripiiriin pohjautuva mikrokontrollerialusta. Se sisältää 20 input/output –pinniä, joista seitsemää voidaan käyttää pulssitaajuusmodulaatioon ja kahtatoista analogisina sisäänmenoina. Erityisen kontrollerista tekee sen sisäänrakennettu USB-tuki, jonka avulla voidaan toteuttaa sarjaliikennekommunikaatio ja toimittaa laitteen käyttämä virta. (Arduino www-sivut 2016.)

Samkbotissa Arduino Leonardoa käytetään luomaan oma, pääprosessointilaitteistosta erillinen, ohjauskerros fyysisen laitteiston ohjaukselle. Ohjauskerroksen käyttö mahdollistaa robotin prosessointiyksikön vaihtamisen toiseen huolehtimatta olemassaolevan laitteiston liittämisestä järjestelmään. Riittää kun uusi yksikkö voi kommunikoida USB:n välityksellä ja pystyy hyödyntämään aikaisemmin ohjelmoitua ajuriohjelmistoa, johon on implementoitu sopiva viestirajapinta moottoriohjaimen ja Arduinoon liitetyn laitteiston ohjaamiseen.

Arduino ohjelma lukee USB-väylän kautta tulevaa sarjaliikennedataa mahdollisten käskyjen varalta. Se vertaa saatuja viestejä implementoituun viestiprotokollaan ja toteuttaa protokollan mukaisten käskyjen toiminnot. Osaa käskyistä käytetään lukemaan robottin jännite- ja virtamittareita, asetuskytkimiä sekä asettamaan robotin tilaa kuvaavia LED-lamppuja päälle. Suurinta osa käskyistä käytetään kuitenkin hyödyntämään MD49 moottoriohjaimen sarjaliikenneyhteyttä. Ohjaimen vaatiman käskyprotokolan mukaiset tavut lähetetään laitteelle ja vastaavasti vastaanotetut tavut tulkitaan sopivaan muotoon lähetettäväksi USB:tä pitkin prosessointiyksikölle. Arduino on asetettu lukemaan moottorien pulssiantureita mahdollisimman nopeasti, kun prosessointiyksiköltä ei saada moottoriohjaimen tietoja lukevia viestikäskyjä. Näin toteutettuna Arduino tai MD-49 voidaan korvata myös toisella alustalla tai kontrollerilla vaikuttamatta järjestelmän toimintaan kokonaisuutena. Vaihtaessa tulee vain huomioida sovittujen laitteiden välisten rajapintojen sääntöjen noudattaminen sekä mahdolliset liitännät muihin fyysisiin laitteisiin tai komponentteihin. Järjestelmärakenne tukee kuitenkin eritoten prosessointiyksikön vaihdettavuutta, jonka vaihdossa voidaan jopa välttää laitekohtainen ohjelmointi.

3.3 Microsoft Kinect

Microsoft Kinect on alunperin Xbox 360 –pelikonsolille julkaistu RGBD-sensori. Laite sisältää RGB-kameran, infrapunakameran ja infrapunaprojektorin. Näiden avulla saadusta datasta voidaan tuottaa ympäröivää maailmaa mallintavia 3D-pistepilviä. Samkbotissa Kinectiä käytetään ympäristön kartoittamiseen, sekä manuaalisen kauko-ohjauksen helpottamiseen videokuvan avulla. Laite on yhteydessä Wandboard-prosessointiyksikköön USB-väylällä. Se välittää infrapunakameran sekä –projektorin avulla tuotettua syvyyskuvadataa ja tavallisen RGB-kameran avulla tuotettua kuvadataa Openni-ajuriohjelmalle, joka puolestaan muuntaa tiedon ROS:lle sopivaan muotoon.

3.4 Wandboard

Samkbotin prossointiyksikkönä toimii Wandboard Quad -niminen kehitysalusta. Wandboard perustuu NPS Semiconductorsin valmistamaan i.MX6 System-On-Chip:iin. Käytännössä kehitysalusta on 32-bittistä ARM Cortex A9 -prosessoria hyödyntävä mikrokontrolleri, joka sisältää muun muassa HDMI-portin, USB-liittimiä, langattoman verkon, sekä 2 gigatavua muistia. Wandboardin käyttöjärjestelmäksi asennettiin Ubuntu Linux.

3.5 Mekaaninen rakenne ja liikemalli

Samkbotin mekaaninen rakenne on hyvin yksinkertainen. Sen kaksi moottoria on kytketty kahteen pyörään, jotka sijaitsevat samalla akselilla. Lisäksi rungon vastakkaisella puolella on kaksi vapaasti liikkuvaa pyörää, jotka pitävät rungon tasapainossa. Kun robotin pyörät liikkuvat samaa nopeutta, se kulkee suoraan. Kun robotin pyörät liikkuvat eri nopeutta, robotti kääntyy. Tämä tapahtuu aina jonkin pyörien yhteiselle, mutta jatketulle, akselille sijoitetun kuvitteellisen pisteen ympäri. Robotin yksinkertaistettua liikemallia hyödynnettiin muunnettaessa pulssiantureilta tulevien signaalien tulkitsemisessa liikenopeuksiksi. Malli ei ota kuitenkaan huomioon esimerkiksi pyörien mahdollista liukumaa tai nopeutta muuttavia pienesteitä.

4 ROS SAMKBOTISSA

Tehtävänä oli mahdollistaa kehitysrobotin toiminta ROS:ia hyödyntäen. Samkbotin oli tarkoitus pystyä kartoittamaan itselleen tuntematon ympäristö samalla paikantaen itsensä suhteessa tuohon ympäristöön. Lisäksi robotissa tuli olla kauko-ohjausmahdollisuus. ROS:in yhdistäminen Samkbottiin päätettiin toteuttaa luomalla samkbot-paketti. Tämä pitää sisällään robotin ohjaamiseen vaaditun ajuriohjelman, järjestelmäkokonaisuuden käynnistämiseen tarkoitetut launch-tiedostot, tarvittavat asetus- ja resurssitiedostot, käyttöjärjestelmäkohtaiset komentojonotiedostot sekä käyttöliittymän luomiseen käytetyt tiedostot. Kappale esittelee Samkbotissa käytetyt keskeiset ROS-paketit ja ohjelmistot sekä kertoo niiden hyödyntämisestä tavoiteltuja robotin ominaisuuksia rakennettaessa.

4.1 Samkbot ajuriohjelma

Toimiakseen ROS-ympäristössä tulee robotin tarjota jonkinlainen yhteys ohjaus- ja mittaustiedon välittämiseen. Samkbotin kohdalla yhteys toimi- ja mittalaitteita hallitsevaan Arduino Leonardo-mikrokontrolleriin päätettiin luoda python-kielellä. Arduino on yhdistettynä USB-väylän avulla Wandboardiin. Ajuriohjelma tulkitsee ja muokkaa saapuvat ROS-viestit Arduinolle paremmin sopivaan muotoon sekä lähettää ne sarjaliikennedatana USB-väylää pitkin. Arduinolta saapuvat viestit prosessoidaan ROS:lle sopivaan muotoon ja julkaistaan ROS-viesteinä järjestelmään. Samkbotin ROS-pakettiin luotiin erillinen, ROS:ista riippumaton, käyttöjärjestelmän ja USB-väylän kanssa kommunikoiva ohjelmakirjasto sekä kirjastoa käyttävä ROS-pohjainen ajuriohjelma. Kirjastoon on implementoitu viestiprotokolla, jota noudattamattomat viestit eivät toteuta käskyjä. Vastaava protokolla on ohjelmoitu myös Arduino-laitteeseen.

Ajuriohjelma kuuntelee /cmd_vel nimisestä ROS-aiheesta Twist-viestejä, jotka muokataan vastaamaan moottoreiden ohjauskäskyjen skaalaa ennen lähettämistä eteenpäin. Arduino vastaanottaa moottorinopeuksia kuvaavat viestit ja asettaa moottorit toimimaan kyseisellä nopeudella. Vastaavasti moottoriohjaimelta tulleet pulssiantureiden viestit välitetään eteenpäin Arduinolta Samkbot-ajurille. Tämä muuntaa moot-

toreiden pyörimisdatan metreiksi alustan geometristen mittatietojen perusteella. Ajuri pitää kirjaa Samkbotin liikkumista etäisyyksistä ja julkaisee etäisyyksien sekä nopeuksien perusteella Odometry-viestejä /odom-aiheeseen. Lisäksi ohjelma on asetettu kuuntelemaan Arduinoon kytketyn kytkimen painallusta mahdollisen sammutuskäskyn varalta. Käsky laukaisee komentorivijonon, joka pyytää ROS:ia sulkemaan itsensä ja laukaisee sitten pienen viiveen jälkeen Ubuntu-käyttöjärjestelmän sammutusprosessin.

Varsinaiseen ajuriohjelman ja sarjaliikennelaitteen väliseen protokollaan implementoitiin myös muita käskyjä, joita ei otettu käyttöön ROS-pohjaisessa ajurissa. Näitä on mahdollista hyödyntää muokkaamalla ajuriohjelman python-koodia. Käskyjen avulla voidaan esimerkiksi lukea Arduinon liitäntöjen mittatietoja tai asettaa laitteeseen kytkettyjä merkkivaloja.

4.2 Hyödynnetyt paketit ja ohjelmistot

4.2.1 Openni

Openni_camera –paketti pitää sisällään Kinectin syvyys-, RGB- ja infrapunakuvastreamien julkaisuun tarkoitetun ajuriohjelmiston (ROS www-sivut 2017). Samkbotissa pakettia käytettiin erillisen openni_launch-paketin välityksellä. Openni_launch hoitaa ajuriohjelmiston käynnistyksen launch-tiedostojen välityksellä ja mahdollistaa kuvadatan muuntamisen esimerkiksi pistepilviksi. Pohjimmillaan kuvadatan muuntamiseen tarkoitettu toiminnallisuus on paketissa toteutettu hyödyntämällä launch-tiedostojen avulla image_pipeline-metapakettia ja TF-kirjastoa. Samkbotin kohdalla kuvadatan muuntamiseen tarkoitettu toiminnallisuus erotettiin omaan launch-tiedostoon, jotta kuvadatan muunto voitiin toteuttaa alennetulla kuvaresoluutiolla.

4.2.2 RTAB-Map

SLAM eli samanaikainen paikannus ja kartoitus on robotiikan ongelma, jossa robotin täytyy samanaikaisesti kartoittaa ympäristöään ja paikallistaa itsensä suhteessa luo-

tuun karttaan. Tämän ongelman ratkaisuksi Samkbotin kohdalla valittiin RTAB-Map -paketti. Työn alkuvaiheessa kokeiltiin muutamia mahdollisia ROS-paketteja toimivan ja RGBD-sensoria hyödyntävän 2D-kartoitusjärjestelmän implementointiin. Jo aikaisessa vaiheessa huomattiin RTAB-Map:in soveltuvan käyttötarkoitukseen oivasti. Useat muut paketit on suunniteltu toimimaan lasertekniikkaan pohjautuvien sensoreiden kanssa ja vaativatkin melko tarkkoja etäisyysmittauksia tai suurta nopeutta mittauksilta. Lisäksi päätös haluttiin tehdä aikaisessa vaiheessa, jotta voitiin minimoida säätöihin vaadittavat työtunnit. RTAB-Map:in ulkomuotoihin perustuvaa kartoitusta voidaan käyttää myös esimerkiksi stereokuvauksen kanssa, jolloin ohjelman vaatima syvyystieto muodostetaan kahden RGB-sensorin kuvakulman erovaisuuksien avulla.

RTAB-Map, Real-Time Appearance-Based Mapping, on avoimen lähdekoodin reaaliaikainen ulkoasuun perustuva kartoitusohjelmisto. Se hyödyntää muistinhallintajärjestelmää, joka mahdollistaa reaaliaikaisen toiminnan tavoitteiden täyttymisen. Järjestelmä rajoittaa sensoreilta tulevan havainnon vertailukohteiden määrää ja näin aikaisemmin vierailtujen kohteiden erottaminen uusista eli silmukan sulkeumien havaitseminen nopeutuu. RTAB-Map pyrkii tarjoamaan kartoitus- ja paikannusratkaisun, jonka toiminta ei ole riippuvainen käytetystä ajasta tai kartoitetun ympäristön koosta. (Labbé & Michaud 2013, 1-2.)

Ohjelmisto määrittelee käytetyn kartan verkkorakenteena, joka koostuu solmuista ja näitä yhdistävistä linkeistä. Solmut tallentavat odometrian antaman asennon, sensoreiden tuottaman datan ja kuvadatasta prosessoidut kuvaukset eli visuaaliset sanat jokaista kartan lokaatiota kohti. Linkit vastaavasti sisältävät solmujen välisten geometristen muunnosten tiedot. RTAB-Map käyttää hyväkseen verkon optimointia. Havaitessaan silmukan sulkeuman ja tämän mukaisen poikkeaman odometriassa voidaan virhe välittää kaikille kartan solmuille samalla minimoiden kartassa esiintyviä poikkeamia. (Labbé & Michaud 2014, 1-2.)

Rtabmap- ja rtabmap_ros -paketti pitävät sisällään RTAB-Mapin itsenäisen ohjelmistokirjaston ja kirjaston toiminnallisuuden välittävän ROS-ohjelmiston. Samkbotissa RTAB-Map on asetettu kuuntelemaan robotin ajurilta saapuvaa odometriatietoa ja Kinect-kuvadatasta muodostettua rekisteröityä syvyys- ja RGB-kuvaa, kamerakuvan

metatietoja sekä LaserScan-tietotyyppiä. Näistä Rtabmap muodostaa 3D-kartan, josta muodostettua 2D-karttaa käytetään Samkbotin liikesuunnitteluun move_base-paketissa. RTAB-Mapin kartan muodostamiseen käytettyjen visuaalisten sanojen prosessoimiseen valittiin lisenssiltään avoin algoritmi. Tämä ja muut paketin parametrit määriteltiin erillisessä launch-tiedostossa. Ohjelmisto mahdollistaisi robotin täydessä 3D-virtuaaliympäristössä toimimisen, mutta vaatisi tämän ominaisuuden myös ympäröiviltä ROS-paketeilta.

4.2.3 Move_base

ROS:in navigation-metapaketti on 2D-navigointiin erikoistunut yhdistelmä useita navigointiin tarvittavia paketteja, joiden avulla voidaan esimerkiksi robotin anturei-den tietoa, karttatietoa ja navigointikohteita hyödyntämällä suunnitella robotin reitti ja lähettää tarvittavat nopeuskäskyt robotille. Osana metapakettia move_base-paketti pyrkii tarjoamaan keinot navigoida haluttuun kohteeseen kartalla. Se yhdistää glo-baalin ja paikallisen suunnittelijan navigoidakseen globaaliin kohteeseen samalla noudattaen paikallisia rajoitteita ja väistäen esteitä. Move_base tarjoaa ominaisuuksi-aan ROS-toimintona, jonka kulkua ja toteutumista voidaan tarkkailla tai jopa perua kokonaan. Se ylläpitää erillisiä costmap_2d –pakettiin perustuvia kulukarttoja, jotka kuvaavat vapaasti liikuttavia, varattuja ja tuntemattomia alueita kartalla. Kulukartat täytetään pohjakartan esteiden, robotin jalanjäljen sekä asetusten perusteella ja näin saadusta tiedosta voidaan muodostaa reitit, joita pitkin robotin on mahdollista kulkea. Move_base myös voidaan asettaa suorittamaan korjaavia toimintoja mikäli robotti havaitsee olevansa jumissa. (ROS www-sivut 2017.)

4.2.4 Robot Web Tools

Robot Web Tools on Michiganin yliopiston, Georgia Techin ja Fetch Roboticsin sponsoroima kokoelma avoimeen lähdekoodin työkaluja web-pohjaisten robottiso-vellusten rakentamiseen. Työkaluihin kuuluu esimerkiksi ROS:in javascript-ohjelmakirjasto, 2d- ja 3d-visualisaatiokirjastot, kauko-ohjausohjelmisto sekä video-kuvan streamaamiseen tarkoitettu ohjelmakirjasto. Robot Web Tools mahdollistaa

webpohjaisen käyttöliittymän luomisen ROS-pohjaiselle robotille yksinkertaisen javascriptin avulla. (Robot web tools www-sivut 2017.)

Web_video_server –ROS-paketti kuuntelee videokuvaa kyseleviä HTTP-pyyntöjä ja asettuu kuuntelemaan saapuvien pyyntöjen mukaisia ROS-aiheita. Ohjelmisto käynnistää pakatun videokuvastreamin ja antaa näin tilaisuuden ROS:in kuvamuotoisten viestien katselemiseen selaimen välityksellä. Paketti mahdollistaa kuvadatan pakkauksen säätelemisen ja pyrkii pitämään kuvaviiveen minimissään. (ROS www-sivut 2017.)

Rosbridge_suite -metapaketti tarjoaa JSON-ohjelmointirajapinnan ROS:ia hyödyntämättömille ohjelmille. Rosbridge on kokoelma paketteja, jotka antavat ROS:in ulkopuolisille ohjelmille tavan kommunikoida ROS:in kanssa JSON-pohjaisten viestien avulla. Se hyödyntää Websocketteja ja TCP:tä muodostaakseen yhteyden selaimeen ja luo näin tietoväylän esimerkiksi Javascriptiä hyödyntävien käyttäjäsovellusten ja ROS:in välille. Rosbridge on osa Robot Web Tools -projektia. (Robot Web Tools Github-sivut 2016.)

Roslibjs pyrkii olemaan ROS:in pääasiallinen Javascript kirjasto. Se on tarkoitettu ROS toiminnallisuuden toteuttamiseen Javascriptin ja rosbridgen avulla. Kirjasto toteuttaa useita ROS:in perusominaisuuksia kuten aiheiden julkaisua ja tilausta, palvelupyyntöjä sekä TF-kirjaston vuorovaikutukset. Roslibjs:n toimintoja on laajennettu rakentamalla sen pohjalta työkaluja kuten ros2djs ja ros3djs, jotka puolestaan ovat 2D- ja 3D-visualisointia avustavia Javascript-kirjastoja. Nav2djs laajentaa edelleen roslibjs:n ja ros2djs:n toimintaa tarjoamalla työkalun, jonka avulla voidaan mm. näyttää robotin luoma kartta ja robotin positio kartalla sekä lähettää navigointitavoitteita robotille. Kirjastot kuuluvat yhdessä Robot Web Tools-projektiin. (ROS www-sivut 2017.)

4.2.5 Depthimage_to_laserscan -paketti

Depthimage_to_laserscan –paketti vastaanottaa Kinectin muodostaman leikatun syvyyskuvan. Kuvan keskivaiheen syvyystietojen ja määriteltyjen ROS-parametrien

mukaisesti paketti muodostaa tiedoista LaserScan-viestejä, jotka sitten julkaisee eteenpäin. (ROS www-sivut 2017.)

4.2.6 Static_transform_publisher

TF-pakettiin kuuluva static_transform_publisher –työkalu julkaisee staattisen muunnoksen yhdestä koordinaatistokehyksestä toiseen. Tätä hyödynnettiin Samkbotissa luomaan syvyyskuvasta muodostetuille LaserScan –viesteille erillinen ’laser’-kehys sekä ilmaisemaan Kinect-laitteen sijainnin poikkeama robotin sijaintipisteen lokaatiosta.

4.3 Käyttöliittymä ja etäohjaus

Manuaalinen kauko-ohjaus toteutettiin luomalla yksinkertainen html-verkkosivu, joka toimii käyttöliittymänä. Ohjaus rakennettiin toimimaan hyödyntäen Javascript-ohjelmointikieltä. Käyttöliittymä tarkkailee selaimelta tulevia järjestelmätapahtumia kuten hiiren klikkauksia, näppäimistöpainalluksia ja älypuhelimen näytön kosketuksia. Tapahtumat kohdistetaan haluttuun toimintoon ja muunnetaan sopivaksi json-viestiksi, joka lähetetään websockettien avulla serverille. Rosbridge vastaanottaa viestit ja muuntaa ne vastaavasti ROS:lle sopivaan muotoon. Viestit ohjataan edelleen oikeaan aiheeseen ROS:in sisällä. Tässä tapauksessa Twist-viestimuodossa ’/cmd_vel’-aiheeseen ja edelleen aihetta kuuntelevalle Samkbotin python-kieliselle ajurille. Käyttöliittymään toteutettiin myös nopeuden säätö ohjaukselle. Säädettävissä on robotin pyörien suuntainen nopeus, joka on myös samalla robotin paikallaan kääntymisen nopeus, sekä vauhdista kääntymisen suhteellinen nopeus.

Kauko-ohjauksen helpottamiseksi voidaan käyttöliittymä asettaa kuuntelemaan ROS:in sisällä julkaistua Kinectin RGB kuvadataa tai tarvittaessa järjestelmään liitettävän webkameran kuvaa. Videokuva välitetään luomalla web_video_server –paketilla suoratoistoyhteys ROS:in kuvadataa sisältävästä aiheesta. Paketin avulla voidaan säädellä kuvan laatua sekä pakkausmuotoa ja näin vähentää verkon kuormitusta. Käytännössä kuva toistuu pienellä viiveellä, joka riippuu käytetyistä asetuksis-

ta ja verkon ominaisuuksista. Asetuksia voidaan muuttaa erilaisten URL-parametrien avulla myös käytön aikana.

Robot Web Toolsin nav2djs Javascript-widgetin avulla luotiin käyttöliittymään mahdollisuus seurata robotin sijaintia kartalla ja osoittaa sille navigoitavia lokaatioita. Nav2dj lähettää karttakuvan klikkausten perusteella robotille uusia navigaatiotavoitteita. Tavoitteet välitetään JSON-viestinä robotille, jossa Rosbridge muuntaa viestit move_base:n hyväksymiksi navigaatiotavoitteiksi. Reitin mukainen liikkuminen kohteeseen saavutetaan välittämällä move_base:n tuottamat Twist-nopeuskäskyt Samkbotin ajurille ja edelleen Arduinolle, joka käskee moottoriohjainta liikuttamaan robotin moottoreita. Robotin liikkuessa kartta päivittyy havaintojen perusteella ja tieto välittyy automaattisesti selaimelle.

4.3.1 Automaattinen käynnistys

Käynnistyksessä hyödynnetään Ubuntu-käyttöjärjestelmän käynnistysjärjestelmää, jonka avulla ajetaan bash-komentojonotiedosto. Tämä puolestaan odottaa kunnes langaton verkkoyhteys on saanut IP-osoitteen muodostettua ja avaa sen jälkeen roslaunch-komennolla ROS-pakettien toiminnallisuuden käynnistävän launch-tiedostojen ketjun toimimaan erillisenä taustaprosessina.

5 YHTEENVETO & KOKEMUKSET

5.1 Kohdatut ongelmat

Projektin edetessä huomattiin nopeasti Wandboard-kehitysalustan prosessointitehon rajallisuus. Vaikkakin kyseessä on kokoisekseen tehokas ja virrankäytöltään kevyt alusta, oli työn kuluessa vaikea sivuuttaa eroja esimerkiksi testausvaiheessa käytettyyn ja Intelin i5-prosessorilla varustettuun kuluttajatason kannettavaan tietokoneeseen. Käytännössä käsiteltävän datan määrää jouduttiin pienentämään erinäisin keinoin, jotta voitiin kompensoida menetetyn prosessointitehon määrää. Vaikkakin tyypillisesti ROS-järjestelmässä voitaisiin melko helposti jakaa työtaakkaa useamman prosessointiyksikön kesken, päätettiin korjaukset toteuttaa Samkbotin sisällä. Yhtenä keinona käytettiin kuvataajuuden laskemista Kinect-ajurista. Tämän lisäksi kuvadataa karsittiin pudottamalla datan joukosta vähemmän kriittisiä pikselirivejä kuva-alueen ylä- ja alaosista sekä sovittamalla kuvadatan kanssa kulkevat CameraInfo-viestit vastaamaan uusittua kuva-alaa. Rivejä säästettiin kuitenkin tarpeeksi muun järjestelmän toimintaan. Karsiminen toteutettiin Image_pipeline –metapaketin sisältämillä image_proc- ja depth_image_proc-paketilla. Prosessointitehoa säästettiin rivien karsimisella myös tiedonkulun myöhemmässä vaiheessa, kun Rtabmapin ei tarvitse yhdistellä löytämiään piirteitä ja karttoja valtaisan suuresta pikselimassasta. Kuva-alan pienennyksessä opittuja taitoja hyödynnettiin myös esteiden väistämiseen tarkoitetun Laserscan-viestien muodostamisessa. Nyt näitä voitiin muodostaa useita samasta syvyyskuvasta.

Arduinon kanssa työskenteltäessä havaittiin järjestelmässä ajoittaisia ongelmia. Toisinaan Arduinolta saapuvien viestien sisältö ei vastannut odotettua dataa. Koetut yhteysongelmat johtuivatkin lopulta katkenneesta kaapelista, useasta kaapelitestistä huolimatta. Lisäksi alkuperäistä ongelmaa korjattaessa kohdattiin hyvin samankaltaisia seurauksia aiheuttanut ja koodivirheestä johtunut moottoriohjaimen sarjaliikennepuskurin täyttyminen. Ongelmien kohtaaminen kuitenkin johti alkuperäistä koodia laadukkaampaan ohjelmakoodiin sekä Arduinon ohjelmistossa, että python-kielisessä ROS-ajurissa. Varsinainen vian paikannus tapahtui poissulkemalla vaihtoehtoja ja samalla suorittamalla järjestelmän toiminnasta kokeilevia olettamuksia.

Kinect-ajurin havaittiin olevan ajoittain oikukas yhdistämään laitteeseen, etenkin ROS:in uudelleenkäynnistysten yhteydessä. Korjaukseksi tähän ongelmaan toteutettiin pienimuotoinen bash-skripti eli komentojonotiedosto bash-komentojonotulkille. Skripti hyödyntää ROS-työkaluja kuten roslaunchia ja rostopicia implementoiden yksinkertaisen ratkaisun. Yhdistysyrityksen jälkeen skripti kuuntelee ajurin julkaisemaa aihetta kuvadatan varalta. Mikäli aiheesta ei löydy viestejä ja tarpeeksi aikaa on kulunut, järjestelmä pysäyttää yrityksen ja kokeilee uudestaan. Käytännössä skripti hidastaa Samkbotin järjestelmien käynnistystä, mutta samalla varmistaa toimivan käytön.

5.2 Ammatillinen kasvu

Kokemuksena työn toteutus opetti minulle valtavasti. Aihetta valitessa olin hieman sivuuttanut toimivan järjestelmän toteuttamiseen vaadittavan tiedon määrää. Minulla ei käytännössä ollut kokemusta ohjelmoinnista. Olin kyllä aikaisemmin tutustunut hieman ohjelmointiin esimerkiksi C++ ja Javascript –kielillä, mutta en toteuttaakseni toimintoja varsinaiseen käytännön järjestelmään. Työn kuluessa ohjelmoinnin opettelu olikin yksi jatkuvista tehtävistä. Aina, kun kohtasin uuden ongelman, käännyin netin puoleen ja aloin tutkia toteutukseen vaadittavaa ohjelmointitietoa.

Toteutuksessa käytin C++, Python, Javascript ja Bash –ohjelmointikieliä. Arduino-ohjelmoinnin toteutin C++:lla, mutta tätä olisin voinut käyttää yhtä hyvin myös robotin ominaisuuksien toteuttamiseen ja ROS:in kanssa kommunikointiin. Python-kieli oli kuitenkin valintani, sen helpon ymmärrettävyyden vuoksi, Samkbotin laiteajurin toteutukseen. Javascriptiä, joka itselläni oli parhaiten hallussa, käytin yksinkertaisen käyttöliittymän toteuttamiseen ja ROS:in kanssa toimimiseen selaimen välityksellä. Yleisesti ohjelmoinnin opettelusta, etenkin tässä työssä käytettyjen kielten saralla, tulee varmasti olemaan hyötyä tulevaisuuden haasteissa. Työ onkin herättänyt minussa uutta mielenkiintoa ohjelmointia kohtaan. Bash-kieltä käytin automaattisen käynnistyksen sekä järjestelmän luotettavuutta parantavien skriptien luomiseen. Bash on myös useimpien Linux-jakeluiden sisältämä komentotulkki, joten sen opettelusta saattaa olla hyötyä myös myöhemmin.

Ennen opinnäytetyön aloittamista olin vain hieman kokeillut Linux:in käyttöä. Työhön kuitenkin vaadittiin mm. ohjelmistojen asentamista, konfiguraatiota ja käyttöä Linux-ympäristössä. Aluksi lähdin kokeilemaan Linux:in asentamista Wandboardille jopa yhdistämällä Linux-ytimen eli kernelin ajureihin, tiedostojärjestelmään ja käynnistyksen microsd-kortilta toteuttavaan ohjelmistoon. Koska halusin toimivan visuaalisen käyttöjärjestelmän kehitystyön helpottamiseksi, päädyin kuitenkin lopulta valmiimpaan ratkaisuun. Tälläinen löytyi Wandboard-kehitysalustan verkkosivuilta. Kyseisen levykuvan asennuksen kanssa riitti muistikortille kirjoittaminen sopivan ohjelman avulla. Myöskin kehitystyössä hyödynnetyn kannettavan tietokoneen kanssa käytettäväksi asennettiin vastaavanlainen Ubuntu-levykuva ulkoiselle kovalevylle. Tämä toimi kannettavan vaihtoehtoisena käyttöjärjestelmänä ja mahdollisti kokonaisen järjestelmän asennuksen ja ROS:in käytön muokkaamatta varsinaista tietojärjestelmää. Ohjelmistojen asennus ja käyttö Linux-ympäristössä totutti nopeasti vaadittuun komentorivipohjaiseen toimintaan. Uskon projektissa opituista Linux-taidoista olevan valtavasti hyötyä myöhemmin.

Samkbotin vaatimassa kaapeloinnissa ja laitteiden liittämisessä ei ollut ongelmia. Tässä pääsin soveltamaan koulutukseni eli sähkötekniikan mukaista tietotaitoa. Lisäksi kokemus tietotekniikan ja erilaisten elektronisten laitteiden kanssa toimisesta oli tarpeen. Vaikka tästä työn osuudesta ei juuri kirjotettukaan opinnäytetyössä, oli tehty työ kuitenkin järjestelmän kannalta kriittistä ja vaati tarkkaavaisuutta. Käytettyjen komponenttien sähköiset ominaisuudet tuli selvittää manuaaleista ja soveltaa käytäntöön esimerkiksi kytkentöjä suunniteltaessa.

ROS:in opetteleminen oli myöskin koko työn ajan hyvin keskeinen teema. Tähän käytin paljon aikaa ja uskon hyötyväni käytetystä ajasta ROS:in käytön yleistyessä. ROS on kuitenkin alati kehittyvä ja melko monimuotoinen järjestelmä, joten opetteluni ei ole vielä ohi, vaan tulen ehdottamasti jatkamaan järjestelmään tutustumista ja käyttöä. Ongelmien ratkaisussa jouduin usein turvautumaan ROS-verkkosivujen wiki-osion lisäksi ’kysymykset ja vastauset’-osioon, jossa tavalliset käyttäjät voivat esittää ROS:ia koskevat kysymyksensä ja saada vastauksia ROS-yhteisöltä. Kun valmiita ratkaisuja ei löytynyt, aloin soveltaa luovaa ja systemaattista ongelmanratkaisua. Mielestäni ongelmanratkaisutaitoni kypsyivät työn edetessä ja minun on nyt

helpompi jakaa ongelmia pienempiin palasiin sekä systemaattisesti jäljittää ongelmien lähteitä ja ratkaisuja.

5.3 Saavutetut hyödyt

Samkbotin toteutuksessa saavutettiin haluttu kauko-ohjaus ja kartoitus soveltamalla ROS-järjestelmää. Tämän lisäksi Samkbot voidaan asettaa navigoimaan haluttuun kohteeseen kartalla. Yhdistelmää ei kuitenkaan saatu käytetyn ajan puitteissa toteutettua vakaasti verkko-ongelmista johtuen.

Suurimman hyödyn uskon kuitenkin tulevan ROS-järjestelmän esittelemisestä. Henkilökohtaisesti minulla ei ole epäilystäkään siitä, että ROS on tulossa jäädäkseen. Tämä tarkoittaa, että myöskään koulutus ei voi olla huomioimatta ROS:in saavutuksia. Toteutuksesta saatu järjestelmäkokoonpano voi toimia hyvänä alkuna ROS-pohjaiselle jatkokehitykselle Satakunnan Ammattikorkeakoulussa.

5.4 Jatkokehitysmahdollisuudet

ROS:in kanssa jatkokehitysmahdollisuudet ovat käytännössä rajattomat, sillä ROS on hyvin monimuotoinen robotiikan järjestelmä. ROS:iin voidaan myös integroida monia järjestelmän ulkopuolisia ohjelmistoja, teknologioita ja protokollia. Tässä kappaleessa listataan kuitenkin nykyisen järjestelmän kannalta kriittisiä sekä toimintaa ja kehitystä helpottavia ratkaisuideoita, joiden kehittämistyöstä olisi huomattavaa apua.

Kehityöllä on tärkeää olla selkeä rakenne. Tämän avuksi tulisi koostaa ROS:in käyttämistä säännöistä ja konventioista helposti opeteltava kokonaisuus, jota voitaisiin käyttää ROS-järjestelmien kehityksessä Satakunnan Ammattikorkeakoulussa ja mielellään myös sen ulkopuolella – tukien avoimen lähdekoodin järjestelmien kehitystä. Lisäksi voitaisiin laatia kattava paketti- ja metapakettipohja erilaisten ROS-pakettien luomiseen. Tulisi määritellä yleisesti työnkulku pakettien kehitykseen sekä kehityksestä saatujen tulosten hyödyntämiseen myös muiden käyttöön. Tälläisen kehitystyön tueksi olisi hyvä määritellä myös pitkäaikaisen kehityksen tavoitteita isomman ryh-

män toimesta. Näitä tavoitteita voitaisiin hyödyntää esimerkiksi rahoituksen hankkimisessa kehitysprojekteille.

Lähempänä Samkbottia kehityskohteet ovat samoja kuin monella mobiililla robotilla. Yksi mahdollisista kohteista on manuaalista kauko-ohjausta tukevan törmäyksenesto toteuttaminen karttatietoa käyttäen. Tämä voitaisiin toteuttaa jopa etäprosessoimalla karttatietoa Javascriptin avulla etäohjaavassa laitteessa. Toteutus vapauttaisi samalla Samkbotin resursseja muuhun käyttöön. Lisäksi voitaisiin implementoida törmäyksenesto hyväksikäyttäen sensoreita ja sähkömekaanisia komponentteja. Samalla tulisi asentaa laitteiseen tarvittavat hätä-seis-katkaisijat ja sähköisesti ohjattava virrankatkaisu moottoriohjaimelle.

Jotta robotti voisi toimia älykkäästi koeympäristössä, olisi hyvä pystyä rajaamaan robotin liikkumisympäristö ohjelmallisesti. Tämä ominaisuus voitaisiin sitoa esimerkiksi karttatiedon ”ulkomuotoon”, jolloin saatua karttatietoa verrattaisiin rajoituksen luomishetkellä tallennettuun karttatietoon. Liittyen turvalliseen käyttäytymiseen, robottiin voitaisiin implementoida puheentunnistusjärjestelmä, joka tunnistaa tärkeitä avainsanoja ja tarvittaessa pysäyttää robotin ohittaen sen normaalin ohjausjärjestelmän. Yksi robottijärjestelmien kohtaamista haasteista on ehdottomasti helppokäyttöisyys ja tähän yhdistetty nopea käyttöönotto. Robotin käyttäytymistä tulisi pystyä nopeasti muuttamaan ympäristön tarpeiden mukaan. Tällaiseen ominaisuuteen tarvitaan ”älyä”. SMACH:ia voidaan hyödyntää monimutkaisten suunnitelmien toteuttamiseen ns. tilakoneiden avulla (ROS www-sivut 2017). Tilakoneiden yhdistäminen visuaaliseen ohjelmointiin olisi mielestäni luonnollinen tapa lähestyä ongelmaa.

5.5 ROS käyttöjärjestelmänä

ROS poistaa ja vähentää robottijärjestelmän kehitystyöstä monia ongelmia. Miinuksena kuitenkin huomattiin ROS:in vaativan huomattavan määrän tutustumista yleisiin konventioihin ja järjestelmän toimintaan, jos halutaan luoda helposti uudelleenkäytettäviä ohjelmistoja. Tiedon pirstaleisuus on yhä yksi avoimen lähdekoodin järjestelmien tyypillisistä ongelmista ja hankaloittaa myös ROS:in opettelua. Siksi opetteluun onkin hyvä seurata jonkin kirjan tai vaikka verkkokurssin tuomaa rakennetta samalla toteuttaen esimerkiksi ROS-sivuston tarjoamia tutoriaaleja. Kokonaisuutena ROS tarjoaa kuitenkin työkalut hyvän robottijärjestelmän suunnitteluun ja toteutukseen. Vaikka ROS ei olekaan niin sanotusti ”valmis paketti”, sen tuomat hyödyt kuten ohjelmakoodin helppo uudelleenkäyttö, erilaiset rajapinnat, yhdisteltävyys muihin järjestelmiin ja erityisesti pakettien toteuttamisessa käytetyt rakenteet ja ohjenuorat helpottavat käyttäjän työtä valtavasti.

LÄHTEET

ROS www-sivut. Viitattu 04.05.2016 ja 21.4.2017. http://www.ros.org

Arduino www-sivut. Viitattu 08.10.2016. https://www.arduino.cc/

Robot Electronics www-sivut. Viitattu 08.10.2016 http://www.robot-electronics.co.uk/

Tully Foote. tf: The transform library. In Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on. Apr. 2013

Quigley, M., Gerkey B. & Smart, W. D. 2015. Programming Robots with ROS. Sebastopol, CA: O'Reilly Media Inc.

Labbé M. & Michaud F. 2013. Appearance-Based Loop Closure Detection for Online Large-Scale and Long-Term Operation. Teoksessa IEEE Transactions on Robotics, vol. 29, no. 3, s. 734-745.

Labbé M. & Michaud F. 2014. Online Global Loop Closure Detection for Large-Scale Multi-Session Graph-Based SLAM. Teoksessa Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, s. 2661-2666.

LIITE 1 SAMKBOTIN TF-PUU