Vesa Virmanen

Tilausten etäsyöttö mobiilirobotille

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Automaatiotekniikka

Insinöörityö

17.5.2017

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Vesa Virmanen Tilausten etäsyöttö mobiilirobotille 36 sivua + 1 liitettä 3.5.2017
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Automaatiotekniikka
Suuntautumisvaihtoehto	
Ohjaaja(t)	Field Application Engineer, Tero Kauria, Lehtori, Jari Savolainen,

Tämän opinnäytetyön aiheena oli suunnitella ja toteuttaa demosovellus Omron Adept LD -mobiilirobotin toiminnollisuuksien esittelemiseksi. Työ toteutettiin yhteistyössä Omronin ja Metropolian kanssa, ja työn tarkoituksena oli viimekädessä hyödyttää molempia osapuolia.

Demosovellus päädyttiin toteuttamaan kolmessa eri osassa kaikkien suunniteltujen toimintojen saavuttamiseksi. Osioiksi muodostuivat käyttöliittymä tilauksien syöttöön, tietokanta niiden tallentamiseen sekä väliohjelma niiden välittämiseen mobiilirobotille. Käyttöliittymäksi valittiin selainpohjainen toteutus käyttäen HTML- ja PHP-ohjelmointikielijä, tietokannaksi MySQL-pohjainen tietokanta käyttäen apuna XAMPP-alustaa, ja väliohjelman tekoon C#-ohjelmointikieli Visual Studio alustalla.

Osioista jokainen käydään työn luettavuuden kannalta tarpeellisella tasolla läpi, menemättä turhan syvälle. Työn ensimmäisestä vaiheesta eli tietokannasta kerrotaan taulurakenne sekä tietokannan tarkoitus tilauksien säilyttämisessä. Toisessa vaiheessa perehdytään käyttöliittymän toimintoihin kuten tilausten lisäykseen ja monitorointiin. Kolmannessa vaiheessa lukijalle selostetaan väliohjelman toimintaperiaate ja sen sisältämät toiminnot. Lisäksi muutamiin työn aikana ilmenneisiin ongelmiin perehdytään tarkemmin.

Työn tuloksena saatiin toiminnallisilta osiltaan toimiva demosovellus tilauksien lähettämiseen mobiilirobotille. Työssä tehtyä käyttöliittymää ei kuitenkaan aivan saatu kuntoon, että sitä voisi esitellä esimerkiksi asiakastilaisuuksissa. Työ tulee todennäköisesti olemaan kehityksen kohteena jatkossakin.

Avainsanat	mobiilirobotti, ohjelmointi, käyttöliittymä, tietokanta

Author(s) Title Number of Pages Date	Vesa Virmanen Remote Sending of Orders to a Mobile Robot 36 pages + 1 appendix 3 May 2017
Degree	Bachelor of Engineering
Degree Programme	Automation Technology
Specialisation option	
Instructor(s)	Tero Kauria, Field Application Engineer Jari Savolainen, Senior Lecturer

The goal of this study was to plan and implement a demo application to show Omron Adept LD mobile robot in action. This study was implemented in co-operation with Metropolia University of Applied Sciences and Omron, and the goal was to benefit both parties in the end.

To achieve all planned functionalities, the demo application ended up being developed in three different sections. The following three different sections were formed, user interface to insert orders, database to store this information and a middleware to transfer orders from the database to the mobile robot. A web based application was chosen for the user interface using PHP and HTML programming languages. MySQL was chosen as the database with XAMPP as the platform, and C# with Visual Studio to make the middleware.

Each section will be covered in enough detail with readability in mind, without going too deep into the code. The first section covers the design and functionality of the MySQL database and the way it stores orders. The second section covers the functionalities of the web-based user interface and the way orders are made and inserted to the database. In the third section, the functionalities and design of the middleware are demonstrated to the reader. In addition, a couple of encountered problems are taken under the microscope.

Thus, a working demo application for sending orders to the mobile robot was developed. Unfortunately, the application's user interface was not qualified to be shown at customer events such as big fairs, yet. The developed application will probably be developed further in the future.

Keywords	mobile robot, programming, user interface, database

Sisällys

Lyhenteet

IP	Internet Protocol. Protokolla tietoliikennepakettien toimittamiseen perille internetin välityksellä.
TCP	Transmission Control Protocol. Protokolla kahden päätepisteen välille muodostettavan yhteyden luomiseksi.

1 Johdanto

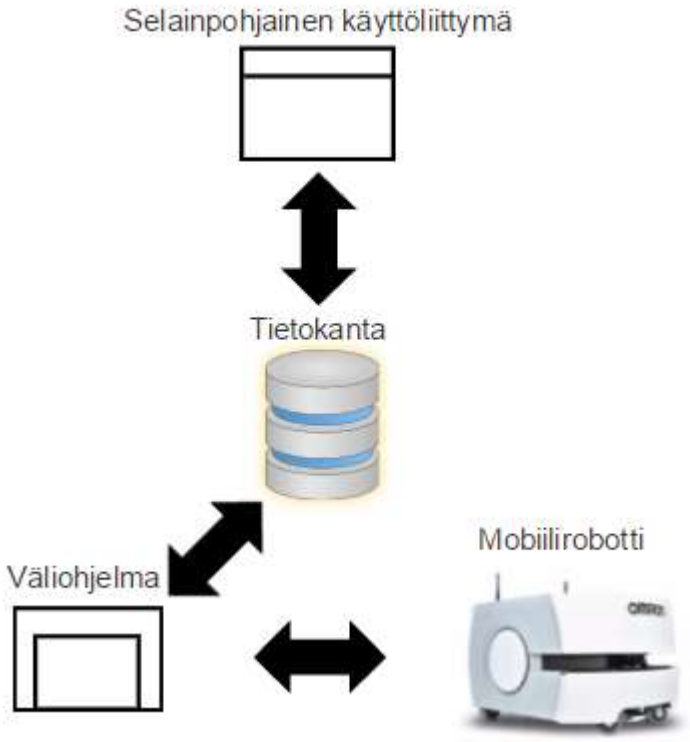
Tämä insinöörityö tehdään yhteistyössä Metropolia Ammattikorkeakoulun ja Omron Electronics Oy:n kanssa. Työn tarkoituksena on tehdä käyttäjäystävällinen demosovellus Omron Adeptin uuden, LD-nimisen mobiilirobotin ominaisuuksien esittelemiseksi. Lähtökohtana on luoda demosovellus, jota voi käyttää kuka tahansa riippumatta koulutustaustastaan. Tarkoituksena ei ole siis tehdä demoa, jota vain siihen perehtynyt osaisi käyttää, vaan niin helppokäyttöinen ja havainnollistava että sen avulla asiasta tietämätönkin osaa mobiilirobottia käskyttää ja syöttää sille tilauksia.

Työ sai alkunsa syksyllä 2016, jolloin Omron Adept -mobiilirobotti otettiin käyttöön Metropolia Ammattikorkeakoulun tiloissa. Metropolia oli ostanut mobiilirobotin ensimmäisenä Euroopassa. Käyttöönoton jälkeen demosovellus mobiilirobotin esittelemiseksi tuli puheeksi ja lopulta todettiin, että sovelluksen kehittäminen insinöörityönä olisi hyväksi sekä koululle että Omronille, sillä sitä voisi käyttää esimerkkinä erilaisilla messuilla tai myyntitilaisuuksissa.

2 Vaatimusmäärittely

Opinnäytetyön tekeminen alkoi keskustelulla työn toisen ohjaajan, Tero Kaurian kanssa, alkutalvesta 2016. Tapaamisessa keskusteltiin sovelluksen yleisestä rakenteesta, ja siitä mitä toiminnollisuuksia sen tulisi pitää sisällään. Tapaamisessa päädyttiin seuraavanlaiseen rakenteeseen, jossa on kolme pääkomponenttia (kuva 1).

- käyttöliittymä käskyjen antamiseksi mobiilirobotille sekä tilaustietojen monitoroimiseen
- tietokanta käyttöliittymältä tulevan tiedon ja käskyjen tallentamiseen
- mobiilirobotin ja tietokannan väliin väliohjelma, jonka tehtävä on välittää komennot tietokannasta mobiilirobotille sekä päivitystietoja mobiilirobotilta tietokantaan



Kuva 1. Ohjelmien väliset suhteet ja tiedon kulku.

Työn rakenteen selviämisen jälkeen, eri osien tekemiselle pohdittiin vielä sopivaa järjestystä. Pohdinnoissa päädyttiin siihen tulokseen, että tietokannan tekeminen tulee ensimmäisenä, sitten käyttöliittymä ja viimeiseksi väliohjelma. Järjestykseen päädyttiin pohtimalla testauksen toteutusta, ja muutenkin projektin yleistä rakennetta. Koska tietokanta on periaatteessa projektin sydän, on sieltä hyvä aloittaa. Muiden ohjelmien rakentaminen sen ympärille on helpompaa, kun tietää jo tietokannan sisällön ja rakenteen. Käyttöliittymän ja väliohjelman järjestys määräytyi pääasiassa, sillä että, koko ketjun testaus onnistuu parhaiten, kun väliohjelmaa ”ylempänä” olevat ohjelmat ovat jo valmiita eikä niitä tarvitse enää muuttaa väliohjelman tekemisen aikana. Eli käyttöliittymä päätettiin tehdä tietokannan jälkeen ja väliohjelma viimeisenä.

2.1 Käyttöliittymä

Käyttöliittymä päätettiin toteuttaa selainpohjaisena ohjelmana, jossa työkaluina käytettäisiin HTML- ja PHP-ohjelmointikieliä ja kehitysympäristönä Eclipse Php Neon -ohjelmistoa. Käyttöliittymästä pitäisi pystyä lisäämään uusia tehtäviä mobiilirobotille sekä seuraamaan avoimia tehtäviä ja niiden tiloja. Lisäominaisuuksiksi, mikäli aikataulu sallisi, määriteltiin käytönvalvonta ja kirjautumismahdollisuus.

2.2 Tietokanta

Tietokanta päätettiin toteuttaa MySQL-ohjelmointikielellä, käyttäen avuksi XAMPP-ohjelmiston tuomaa kehitysympäristöä. Tietokannalta vaadittiin tässä sovelluksessa kykyä varastoida kaikki tarvittu tieto, joka tulee mobiilirobotilta ja käyttöliittymästä kuten esimerkiksi tilaukset ja tilauksien osatilaukset. Varsinaista taulurakennetta ei tässä kohtaa vielä käyty läpi.

2.3 Väliohjelma

Väliohjelman haluttuja ominaisuuksia käytiin läpi hyvin vähän vielä tässä tapaamisessa, sillä se oli tarkoitus tehdä vasta viimeisenä vaiheena. Alustavasti kuitenkin mietittiin sopivaa ohjelmointikieltä sen tekemiseen ja alustavasti päädyttiin Java-ohjelmointikieleen. Kun väliohjelman tekemisen aloittaminen lähestyi, väliohjelman teossa päädyttiin kuitenkin käyttämään C#-ohjelmointikieltä, sillä paremmalta vaihtoehdolta näistä kahdesta.

Väliohjelman tulee pystyä muodostamaan TCP/IP-sokettiyhteys mobiilirobottiin, jota pitkin tieto saadaan kulkemaan tietokannasta mobiilirobotille ja mobiilirobotilta väliohjelman kautta tietokantaan. Lisäksi ohjelman tulee muodostaa yhteys tietokantaan ja osata hakea sieltä ja syöttää sinne tarvittavat tiedot. Näiden jälkeen ohjelman tulee välittää uudet tilaukset tietokannasta mobiilirobotille. Ohjelman tulisi myös pystyä päivittämään automaattisesti tilausten tietoja, kuten esimerkiksi tilaa. Tämän lisäksi ohjelmalla olisi hyvä olla jonkinlainen oma käyttöliittymä, josta sen toimintaa voitaisiin seurata. Käyttöliittymä ei kuitenkaan tulisi varsinaisen käyttäjän nähtäväksi.

3 Työkalut

Jo työn alussa oli selvää, että työ tulee vaatimaan paljon uuden opettelua sekä useiden eri ohjelmistojen ja ohjelmointikielten hallintaa. Työssä käytettävät ohjelmat valittiin suurimmaksi osaksi sen perusteella, että ne olivat jo osittain tuttuja ja hyväksi todettuja. Valintaperustana oli tietenkin myös se, että ohjelmat sopivat juuri projektin tarpeisiin. Projektin aikana käytössä oli yhteensä neljä eri ohjelmistoa ja niiden sisällä neljä eri ohjelmointikieltä.

Työkaluista Eclipse Php Neon ja XAMPP olivat tuttuja kesältä 2016, jolloin tein niiden avulla pientä nettisivuprojektia. Idean Visual Studio 2015:n käyttämiseen sain töissä olleesta koulutuksesta, jossa sitä käytettiin. Ihastuin todella yksinkertaiseen käyttöliittymän muodostamiseen sekä olio-ohjelmoinnin mahdollisuuksiin.

Ensimmäiseen osioon eli tietokannan tekoon käytettiin XAMPP-ohjelmistoa sekä sen mukana tulevaa phpMyAdmin-työkalua. Sen avulla tietokannan osuus suunniteltiin ja toteutettiin tässä insinöörityössä. Toisen osion eli käyttöliittymän teossa käytettiin Eclipse Php Neon -kehitystyökalua. Käyttöliittymä toteutettiin Eclipsen avulla käyttäen kielinä HTML-ja PHP-ohjelmointikieliä. Viimeinen osio eli väliohjelma toteutettiin käyttäen apuna Visual Studio 2015 -kehitysympäristöä ja sen sisällä C#-ohjelmointikieltä. Edellä mainittujen työkalujen lisäksi työssä käytettiin myös Socket Test 3 -nimistä ohjelmaa testaustarkoituksessa.

3.1 Visual Studio 2015

Microsoft Visual Studio on Microsoftin kehittämä ohjelmistojen integroitu kehitysympäristö. Se mahdollistaa monenlaisten ohjelmien kehityksen Windows-alustoille sekä mobiili-ja selainalustoille. Microsoft Visual Studio käyttää apunaan Microsoftin kehittämää ohjelmistokomponenttikirjastoa .NET Frameworkia. .NET mahdollistaa sovellusten nopeamman kehityksen, sen tarjoamien valmiiden kirjastojen ja komponenttien ansiosta. Useat ohjelmointikielet käyttävät hyväkseen .NET-kirjastoa, niihin lukeutuu muun muassa C#, C++ ja Visual Basic .NET. Nämä ohjelmointikielet ovat myös täten myös koottavissa Visual Studio -ohjelmistolla. [1, s. 12–14.]

Visual Studio 2015 tarjoaa monia ohjelmoijan työtä helpottavia ominaisuuksia, kuten koodin automaattisen täydennyksen, monipuolisen debuggerin, eli virheidenetsintätyökalun ja graafisten käyttöliittymien suunnitteluun tarkoitetun työkalun. Visual Studiosta on saatavilla useita eri versioita sekä eri versioiden sisällä erilaisille alustoille räätälöityjä paketteja. Työn tekohetkellä uusin ohjelmistoversio on Visual Studio 2017. Itse työssä käytettiin Visual Studio 2015 Community -nimistä versiota, joka on ladattavissa ilmaiseksi Microsoftin kotisivuilta. [2.]

C#, yksi .NET kehityskehykseen perustuvista ohjelmointikielistä, on vuonna 2000 julkaistu ohjelmointikieli Windows-alustoille. Sen kehityksen tavoitteena oli luoda yksinkertainen, moderni, monikäyttöinen ja olio-pohjainen ohjelmointikieli. [3.]

3.2 Eclipse Php Neon

Eclipse Php Neon on Eclipse IDE -kehitysympäristön lisäosa, joka on tarkoitettu selainaplikaatioiden rakentamiseen. Eclipse Php Neon sisältää työkalut ohjelmien kirjoittamiseen PHP-, HTML- ja JavaScript-kielillä. Ohjelmisto on ladattavissa ilmaiseksi Eclipse Foundationin kotisivuilta. [4.]

3.2.1 HTML

HTML, englanniksi *Hypertext Markup Language*, on standardisoitu kuvauskieli, jota käytetään internetsivujen luomiseen. HTML-dokumentit sijaitsevat internetpalvelimella ja verkkoselain, kuten esimerkiksi Google Chrome, saa dokumentit palvelimelta, jonka jälkeen selain esittää dokumentin sisällön käyttäjälle graafisena verkkosivuna. Selaimen saama HTML-dokumentti on kaikkien nähtävillä, sillä useista selaimista löytyy työkalu varsinaisen HTML-dokumentin tutkimiseen. HTML-dokumentti ei siis saa tai voi sisältää mitään arkaluonteista, koska HTML-tiedostoa voi tutkia kuka tahansa sivustolle pääsevä. Arkaluonteisen tiedon kuten esimerkiksi käyttäjätunnusten hallinnointi onkin muiden ohjelmointikielien, kuten PHP:n tehtävä.

3.2.2 PHP

PHP on oliopohjainen ohjelmointikieli. Sitä käytetään nettisivujen backendin tekemiseen. Backendillä tarkoitetaan tässä tapauksessa koodia ja toiminnallisuutta, jota internetiä

normaalisti selaava ei pysty näkemään, eli ei esimerkiksi HTML-koodia. Nettisivuilla, joilla on esimerkiksi kirjautumismahdollisuus käyttäjille, on käytössä jonkinlainen tietokanta, johon käyttäjien tiedot on tallennettu. Koska tietokannassa on yleensä tietoja, joita ei haluta muiden kuin tietyn käyttäjän tai käyttäjäryhmän, esimerkiksi järjestelmänvalvojat eli adminit, nähtäväksi, täytyy näiden tietojen hallinta ja käyttö sijaita itse fyysisellä palvelimella. Tätä tarkoitusta varten on kehitetty PHP-niminen ohjelmointikieli. Toisin kuin aiemmin mainittu HTML-koodi, PHP:lla tehtyä koodia ei näe selaimella, vaikka huomaisi, että jonkin verkkosivun tiedostopääte olisikin .php.

PHP-koodia tehdään HTML-dokumenttien sisään ja mikäli jonkun verkkosivun tiedostopäätteessä lukee pääte .php, kertoo se siitä, että sivun HTML-koodin sisällä on PHP-koodia. Tiedostopäätteen muuttuminen johtuu siitä, että PHP-koodia ei voi laittaa pelkän HTML-dokumentin sisään, vaan HTML-dokumentti pitää muuttaa PHP-muotoon, jotta PHP-kieltä voidaan siihen kirjoittaa. .php-päätteinen dokumentti voi siis sisältää vain PHP-kieltä tai sen sisältä voi löytyä myös HTML-ja JavaScrip-kieliä.

3.3 XAMPP

XAMPP on ilmainen vapaan lähdekoodin web-palvelinohjelmisto. Sen avulla pystytään ylläpitämään paikallista web-palvelinta, joka avaa mahdollisuudet helppoon web-sivustojen ja sovellusten paikalliseen testaamisen, ilman yhteyttä oikeaan palvelimeen tai internettiin. [5] XAMPP:n avulla Apache-palvelimen asentaminen tietokoneelle on hyvin vaivatonta, eikä se vaadi ohjelman asentamisen lisäksi ollenkaan konfigurointia. XAMPP:n asennus sisältää useiden muiden komponenttien lisäksi PHP-ohjelmointikielen sekä MySQL relaatiotietokantapalvelimen [6, s. 12]. XAMPP:iin on myös sisällytetty selaimessa toimiva kolmannen osapuolen hallintasovellus MySQL-tietokannoille, nimeltään phpMyAdmin.

3.3.1 MySQL

MySQL on vapaan lähdekoodin relaatiotietokantapalvelin, ja sen tietokantamoottorina Windows-käyttöjärjestelmissä toimii InnoDB. MySQL on maailman suosituin vapaan lähdekoodin tietokanta, ja se on varsin suosittu suurien yritysten keskuudessa. MySQL:n korkean profiilin käyttäjiin kuuluvat muun muassa Facebook, Youtube, Twitter ja Yahoo.

MySQL:n vahvuuksiin kuuluu joustavuus, tehokkuus, joustavat lisensointivaihtoehdot sekä hyvin aktiivinen yhteisö. [6, s. 622—628; 7.]

Taulukko 1. Yksinkertainen taulu nimeltään Henkilöt.

id	nimi	sukunimi	ikä
1	Matti	Meikäläinen	40
2	Maija	Meikäläinen	35
3	Silja	Meikäläinen	12

Relaatiotietokannat koostuvat yleisesti tauluista. Nämä taulut pitävät sisällään rivejä ja sarakkeita. Relaatiotietokannoissa sarakkeita kutsutaan attribuuteiksi. Jokaisessa tietokannan taulussa on aina yksi attribuutti, joka on pääavain. Pääavain toimii eräänlaisena tunnisteena taulun tietojen välillä ja sen avulla toisesta taulusta voidaan viitata sen taulun tiettyyn riviin. Viittauksissa eli relaatioissa pääavaimen parina toimii vierasavain. Vierasavain viittaa toisen taulun pääavaimeen ja se voi saada vain sellaisen arvon, joka löytyy jo pääavaimen kentästä.

Taulukosta 1 nähdään esimerkkirakenne yksinkertaiselle taululle nimeltä Henkilöt. Taulussa on 4 erilaista saraketta eli attribuuttia, id, nimi, sukunimi sekä ikä. Jokaisella attribuutilla on omat tietotyyppinsä ja tässä id-ja ikä-attribuuteilla on integer eli kokonaislukutyyppi ja nimi ja sukunimi ovat tekstityyppisiä. Lisäksi attribuutti id on tässä taulussa pääavain, ja se on käytännössä juokseva luku arvosta 1 äärettömään. Tauluun on tallennettuna 3 eri riviä tietoa, id-tunnuksilla 1—3. Taulun tietoihin päästään käsiksi SQL-komentokielellä.

```
SELECT * FROM Henkilöt WHERE sukunimi = 'Meikäläinen'

INSERT INTO Henkilöt(nimi, sukunimi, ikä) VALUES ('Hannu', 'Meikäläinen', 1)

UPDATE Henkilöt SET ikä = 41 WHERE nimi = 'Matti'
```

Esimerkkikoodi 1. SQL-kielen peruskomennot.

Tiedon hakemiseen, päivittämiseen ja lisäämiseen MySQL-tietokantaan käytetään yleistä ohjelmointikieltä SQL. Sen avulla relaatiotietokantaa, kuten MySQL, saadaan muokattua halutulla tavalla. Esimerkkikoodista 1 nähdään 3 yleisintä tietokannan muokkauskomentoa. SELECT-komennolla voidaan hakea tietoa tietokannasta hyvinkin tarkasti valikoiden. Esimerkkikoodissa taulusta Henkilöt haetaan, taulukko 1, kaikki tiedot, joissa sukunimi attribuutin arvo on ”Meikäläinen”. INSERT-komennolla tietokannan tiettyyn tauluun voidaan lisätä tietoa. Esimerkkikoodissa lisäämme Henkilöt-tauluun uuden henkilön, jonka nimeksi tulee Hannu, sukunimeksi Meikäläinen ja iäksi 1. Esimerkkikoodissa viimeisenä on päivityskomento UPDATE, jonka avulla jo tietokannasta löytyvää dataa voidaan päivittää lisäämättä kokonaan uutta riviä. Esimerkissä Henkilöt taulun ikä attribuutti päivitetään arvoon 41, jossa attribuutin nimi arvo on ”Matti”. Kolmen peruskomennon lisäksi SQL sisältää paljon lisää komentoja, jotka eivät ole järin relevantteja tämän työn kannalta.

3.3.2 phpMyAdmin

phpMyAdmin on ilmainen ohjelmistotyökalu, joka on tarkoitettu MySQL-tietokantojen hallinnointiin. Sen etuna on yksinkertainen hyvin toiminnollinen käyttöliittymä, joka toimii selaimen kautta. phpMyAdminin käyttöliittymän avulla pystytään tekemään kokonainen tietokanta hyvinkin helposti. Graafisien työkalujen avulla käyttäjän ei tarvitse itse kirjoittaa sanaakaan SQL-komentokielellä, vaan taulut ja attribuutti sekä niihin sijoitettava data voidaan luoda muutamaa nappia painamalla. phpMyAdmin mahdollistaa myös tietokannan hallinnoinnin etänä internetin välityksellä. [8.]

3.4 Socket Test 3

Socket Test 3 on ilmainen, vapaan lähdekoodin ohjelmisto TCP-sokettien testausta varten. Sen avulla pystytään muodostamaan TCP- tai UTP-clientti tai serveri. Näiden avulla sitä voidaan käyttää minkä tahansa TCP- tai UTP-serverin tai clientin testaamiseen, esimerkiksi simuloimaan jotain laitetta kuten tässä projektissa mobiilirobottia.

4 Mobiilirobotti

4.1 Yleistä mobiiliroboteista

Sana mobiilirobotti on varmasti monelle ennaltaan tuntematon, ja yleisesti sanasta robotti tulee mieleen joko teollisuusrobotti tai jokin kävelevä laite. Mobiilirobotteja on kuitenkin ollut olemassa jo useita vuosikymmeniä, mutta teollisuudessa ja loppukäyttäjillä niitä ei ole näkynyt kuin vasta viime vuosien aikana.

Teollisuudessa käytettävät mobiilirobotit ovat yksinkertaisesti ilmaistuna pääasiassa tarkoitettu kuljettamaan tavaraa paikasta a paikkaan b. Siinä missä monelle tuttu vihivaunu eli AGV, englanniksi *Autonomous Guided Vehicle*, kulkee sille tarkoin määritellyillä reiteillä vähän samaan tapaan kuin juna kiskoilla, mobiilirobotti pystyy itsenäisesti navigoimaan ympäristöään tarkastellen. Mobiiliroboteista käytetäänkin lyhennettä AIV eli *Autonomous Intelligent Vehicle*, eli vapaasti suomennettuna itsenäinen älykäs kulkuneuvo.

Mobiilirobotti eroaa vihivaunusta monella eri tapaa, suurimpana erona pidetään navigointia. Vihivaunu vaatii erilaisia magneettiteippejä lattioihin tai apuvälineitä sen navigoimiseen, eikä se esimerkiksi pysty väistämään sen tiellään olevia esteitä. Mobiilirobotti ei taas vaadi ollenkaan apuvälineitä navigoimista ajatellen. Mobiilirobotti käyttää apunaan erilaisia sensoreita, kuten laseria, joiden avulla se paikantaa itsensä muistissa sijaitsevaan karttaan. Kartan ja sensoreiden avulla mobiilirobotti pystyy lennosta muodostamaan reitin haluttuun määränpäähän ja kiertämään mahdollisia esteitä aivan itsenäisesti. Navigoimisen lisäksi erona on erilaiset käyttökohteet. Useiden eroavaisuuksien vuoksi mobiilirobotteja ei siis pidä sotkea vihivaunuihin.

Tällä hetkellä teollisuuteen tarkoitetut saatavilla olevat mobiilirobotit on tarkoitettu lähinnä suhteellisen pienien massojen kuljettamiseen. Esimerkiksi tässä insinöörityössä osana oleva Omron Adept LD -mobiilirobotti pystyy maksimissaan kantamaan 130 kilogrammaa. Mobiilirobottien suurimmat käyttökohteet onkin tavallisten kantotöiden korvaaminen, jotka ihminen on aiemmin suorittanut. Yksinkertaisen työn tekeminen mobiiliroboteilla, tai yleensä roboteilla, antaa ihmisille enemmän aikaa tehdä työtehtäviä, joihin robotit eivät sovellu.

4.2 Omron Adept LD

Omron Adept LD -mobiilirobotti on alun perin Adept Technologiesin kehittämä mobiilialusta tavaran kuljetukseen. Aiemmin Lynx-nimellä tunnettu mobiilirobotti uudelleenjulkistettiin Omronin ostaessa Adeptin alkuvuodesta 2016. Uudelleenjulkistuksen yhteydessä myös nimi muuttui, ja Lynxistä tuli LD-mobiilirobotti.

Kuva 2. Omron Adept LD-mobiilirobotti.

LD-mobiilirobotti on noin 60 kilogramman painoinen, 70 cm pitkä, 50 cm leveä ja vajaa 40 cm korkea pyörillä kulkeva ajoneuvo. Sen maksiminopeus on mallista riippuen 1,8—0,9 metriä sekunnissa ja sen kantokyky on maksimissaan 130 kilogrammaa. LD-mobiilirobotti on suunniteltu toimimaan itsenäisesti kommunikoiden erillisten koneiden ja logiikkaohjainten kanssa sekä ottaen tilauksia vastaan MES-järjestelmältä tai vastaavalta. Sitä voidaan käyttää pienissä sovelluksissa yksistään, tai parhaimmillaan yli 100 robottia voi olla saman laivueen alla. LD-mobiilirobotti navigoi käyttäen kahta erilaista laser-anturia, tarkemmin sanottuna laser-skanneria, jotka sijaitsevat laitteen etuosassa (kuva 2). Lisäksi laitteen takaosassa on kaksi ultraäänianturia turvatoimintoina peruutusta varten.

Ylempi laserskanneri on fyysisesti turvalaserskanneri, ja se on tarkoitettu pääasiassa mobiilirobotin navigoimiseen. Navigoinnin lisäksi skanneri toimii yhtenä turvatoiminnoista antaen mobiilirobotille turvaluokituksen ja täten sallien sen liikkumisen tilassa, jossa on ihmisiä. Pääskannerin skannausalue on 250 astetta, 199 millimetriä maasta ja skannausetäisyys maksimissaan noin 15 metriä. Pääskannerin lisäksi sen alapuolella lähellä

maanpintaa sijaitsee toinen, hieman pienempi skanneri. Tämän skannerin tarkoituksena on havaita pieniä kappaleita robotin tiellä, joita pääskanneri ei havaitse. Alaskanneri skannaa 270 asteen alueella, 63,5 mm maasta ja noin 4 metrin etäisyydellä. Alaskanneri toimii lisäksi apuna navigoimisessa tarkentaen sitä. Skannereiden lisäksi mobiilirobotissa on ultraäänianturit takana, jotka estävät törmäykset peruutustilanteissa sekä edessä puskuri, joka aktivoi hätäpysäytyksen sen aktivoituessa. [9.]

4.3 Kartta

LD-mobiilirobotin navigoinnin pohja perustuu sen muistiin tallennettavaan karttaan. Kartta sisältää lattiatason kuvan alueesta, jossa mobiilirobotin on tarkoitettu liikkuvan. Lisäksi kartta sisältää kaikki toiminnalliset elementit, mitä mobiilirobotin sovellus vaatii. Tämä kartta tehdään käytännössä siten, että mobiilirobotin avulla skannataan alue, jossa robotti tulee liikkumaan. Skannaamisen jälkeen kartta prosessoidaan ja siihen lisätään halutut toiminnalliset elementit kuten maalit ja alueet, joihin mobiilirobotti ei saa mennä. Kartta siis toimii navigoinnin pohjana mutta itse navigointi tapahtuu aiemmin mainittujen laserskannereiden ympäristöstä antaman datan perusteella sekä pyörissä sijaitsevien enkoodereiden avulla. Kartta antaa pohjan, johon sitten verrataan skannereilta ja enkoodereilta saatua tietoa. Vertauksen pohjalta mobiilirobotti tietää paikkansa hyvinkin tarkasti ja näin ollen pystyy suunnittelemaan karttansa sekä liikkumaan jouhevasti.

Kartta voi sisältää monia toiminnollisuuksia, mutta kaikista tärkeimpiä ovat maalit, engl *goals*. Ne toimivat eräänlaisina toiminnallisina pisteinä, joihin mobiilirobotin voi lähettää. Maali voi toimia esimerkiksi tavaran ottopisteenä ja sitten toinen maali jättöpisteenä. Maalit on tarkoitettu sisältämään monenlaisia toiminnollisuuksia, kuten esimerkiksi äänikomentoja, odotuskomentoja tai kättelyitä logiikkaohjaimen kanssa. Maalit ovat hyvin tärkeässä asemassa, kun robottia käskytetään ja sille annetaan töitä tai tilauksia.

4.4 Tilaukset

LD-mobiilirobotin pääasiallinen toimintatapa sovelluksissa on erilaisten tilausten suorittaminen. Tilauksella tarkoitetaan tässä yhteydessä tehtäväjonoa, jolla on tarkka suoritusjärjestys. Mobiilirobotille lähetettävä tilaus voisi olla esimerkiksi seuraavanlainen: nouto maalilta 1, jättö maalille 2 ja jättö maalille 3. Maalit 1—3 ovat käytännössä robotin

kartalle määriteltyjä pisteitä, joihin se voidaan lähettää. Oikeassa teollisuusympäristössä ne voisivat olla esimerkiksi paikkoja, jossa ihminen tai kone laittaa mobiilirobotin kyytiin tavaraa tai ottaa pois. Toiminnot, jättö ja nouto, ovat ohjelmallisia tunnuksia, joilla toiminnot voidaan erotella toisistaan. Käytännössä mobiilirobotti ei tee toimintotiedolla mitään, koska kaikki toiminnot, jotka maaliin tullessa halutaan tehdä, pitää rakentaa ohjelmallisesti kyseisen maalin alle.

Jokaisella tilauksen sisältämällä tehtävällä on siis kaksi eri parametria, kohdemaali sekä toiminto eli jättö tai nouto. Näiden kahden lisäksi on vielä yksi parametri, prioriteetti, joka tilausta tehtäessä vaaditaan. Prioriteetti määrittelee koko tilauksen tärkeyden. Jokaisen tilauksen ensimmäisen tehtävän prioriteetti ratkaisee koko tilauksen tärkeyden, näin ollen muut prioriteetit eivät merkitse ohjelmallisesti mitään, vaikka ne olisivat suurempia kuin ensimmäinen.

LD-mobiilirobottia voi myös käskeä usealla muulla eri keinolla, mutta juuri tilausten tekeminen on suositelluin tapa, etenkin siinä tapauksessa, että mobiilirobotteja toimii samalla alueella useita. Yhden LD-mobiilirobotin sovelluksessa tilaukset lähetetään suoraan mobiilirobotille, ja se hoitaa tilauksien järjestelyn sekä suorittamisen annettujen parametrien mukaan. Useamman kuin yhden LD-mobiilirobotin laivue taas vaatii erillisen älyn toimimaan eräänlaisena liikennevalvomona ja tilausten jakajana. Tässä tapauksessa tilausta ei lähetetäkään tietylle robotille vaan ainoastaan tälle älylle. Tämä Enterprise Manageriksi kutsuttu laite jakaa kaikki sille tulevat tilaukset siihen yhteydessä olevalle LD-mobiilirobotti-laivueelle. Enterprise Manager jakaa tilaukset älykkäästi aina lähimmälle tilausta olevalle robotille, joka ei suorita mitään muuta tilausta. Tilausten jaon lisäksi Enterprise Manager jakaa tietoa mobiilirobottien paikkatietoja toisille mobiiliroboteille, jotta liikennöinti olisi sulavaa. Enterprise Manager mahdollistaa keskitetyn tilausten syöttämisen esimerkiksi MES-järjestelmästä tai jostain muusta vastaavasta.

4.5 Kommunikointi

Sovellukset joihin LD-mobiilirobotti on tarkoitettu, vaativat yleensä jonkinlaista kättelyä laitteiden välillä, esimerkiksi vaihtamaan tietoa paikasta ja tilasta. Yksinkertainen ja helppolukuinen kommunikointirajapinta onkin siksi yksi LD:n tärkeimmistä ominaisuuksista.

LD-mobiilirobotin käyttämä rajapinta on nimeltään ARCL, *eng Advanced Robotics Communication Language*. Sen avulla mobiilirobottia voi käskeä lähes millä tahansa laitteella, joka pystyy TCP/IP-pohjaiseen sokettikommunikointiin.

ARCL:n avulla voidaan kommunikoida sekä yksittäisen mobiilirobotin että Enterprise Managerin kautta kokonainaisen mobiilirobottilaivueen kanssa. Yksittäisen LD:n kanssa kommunikointi tapahtuu suoraan mobiilirobotin ja esimerkiksi tietokoneen tai logiikan välillä TCP/IP-pohjaisella sokettikommunikoinnilla. Mikäli käytössä on mobiilirobottilaivue ja sitä myötä Enterprise Manager, muuttuu kommunikointiketju siten, että kaikki komennot välitetäänkin Enterprise Managerille, eikä suoraan yksittäiselle mobiilirobotille. Myös lähetettävissä olevat komennot muuttuvat, kun vaihdetaan yksittäisestä mobiilirobotista mobiilirobottilaivueeseen.

```
queueMulti <number of goals> <number of fields per goal> <goal1>
<goal1 args> <goal2> <goal2 args> … <goalN> <goalN args> [jobid]
```

Esimerkkikoodi 2. ARCL queueMulti-komennon syntaksi. [10.]

Kun mobiilirobotille tai -roboteille halutaan lähettää tilaus, joka sisältää useamman maalin, turvaudutaan queueMulti-nimiseen ARCL-komentoon. Esimerkkikoodista 2 nähdään queueMulti-komennon syntaksi eli rakenne. Ensimmäinen sana kertoo, että kyseessä on juurikin queueMulti-komento, tämä määrittelee komennon ja komentoa seuraa sen parametrit. Komennon määrittelyn jälkeen tulee maalien lukumäärä numerona, sen jälkeen toinen numero, joka kertoo kuinka monta argumenttia, tai kenttää jokaisella maalilla on. Tämä luku on tällä hetkellä aina 2, eli kaksi argumenttia, prioriteetti ja toiminto. Argumenttien lukumäärän jälkeen alkaa maalien sekä maalien argumenttien luettelo mallia: <maali> <toiminto prioriteetti>. Tilauksen sisältäessä useamman maalin, tulevat ne peräkkäin samaa kaavaa noudattaen. Kaikkien maalien jälkeen voidaan vielä halutessa liittää tilaukseen oma tilauskoodi *jobid*.

```
queuemulti 4 2 x pickup 10 y pickup 19 z dropoff 20 t dropoff 20
```

Esimerkkikoodi 3. Esimerkki queueMulti-komennosta. [10.]

Esimerkkikoodissa 3 nähdään esimerkki queueMulti-komennosta, joka lähettäisi robotin ensin prioriteetilla 10, maaliin ”x”, suorittamaan noudon, sen jälkeen maaliin ”y” suorittamaan jätön ja niin edespäin, ilman itsemääritettyä tilauskoodia. queueMulti-komentoa käytettäessä tulee huomioida, että jokaisen tilauksen ensimmäinen maali pitää olla nouto. Lisäksi on hyvä muistaa, että ainoastaan ensimmäisen maalin prioriteetti merkitsee, se määrittää koko tilauksen tärkeyden toisin sanoen tekojärjestyksen suhteessa muihin tilauksiin.

```
QueueMulti: goal "x" with priority 10 id PICKUP1 and jobid JOB1
successfully queued
QueueMulti: goal "y" with priority 19 id PICKUP2 and jobid JOB1
successfully queued and linked to PICKUP1
QueueMulti: goal "z" with priority 20 id DROPOFF3 and jobid JOB1
successfully queued and linked to PICKUP2
QueueMulti: goal "t" with priority 20 id DROPOFF4 and jobid JOB1
successfully queued and linked to DROPOFF3
EndQueueMulti
```

Esimerkkikoodi 4. queueMulti-komennon, esimerkkikoodista 3, palauttama viesti. [10.]

queueMulti-komennon jälkeen mobiilirobotti palauttaa QueueMulti-alkuisia rivejä niin monta kuin tilauksessa on maaleja (Esimerkkikoodi 4). Palautettu viesti ilmoittaa, että maalit on mobiilirobotin päässä linkitetty toisiinsa sekä laitettu tilausjonoon onnistuneesti. Viesteistä nähdään myös kaikki tilauksen osatilausten maalit ja niiden argumentit. Viesteistä nähdään myös argumenteista hieman eroava *id*-kohta, jossa lukee maalissa suoritettava toiminto *pickup* tai *dropoff* ja niiden perässä juokseva kokonaisluku. Tämä *id* on tarkoitettu mahdollistamaan tilausten osatilausten erottelun toistaan. Viimeisenä viestien argumenteissa nähdään tilauksen *jobid*-arvo. Mikäli käyttäjä ei määrittele tilaukselleen *jobid*-tilauskoodia, määrittää mobiilirobotti sen itse muotoon JOB+n, jossa n on juokseva kokonaisluku, esimerkkikoodin 4 tapauksessa *jobid*:ksi tuli JOB1. Palautettu viesti päättyy EndQueueMulti-riviin.

```
QueueUpdate: PICKUP1 JOB1 10 Pending None Goal "x" "None"
08/15/2013 06:02:59 None None 0
QueueUpdate: PICKUP2 JOB1 19 Pending ID_PICKUP1 Goal "y" "None"
08/15/2013 06:02:59 None None 0
QueueUpdate: DROPOFF3 JOB1 20 Pending ID_PICKUP2 Goal "z" "None"
08/15/2013 06:02:59 None None 0
QueueUpdate: DROPOFF4 JOB1 20 Pending ID_DROPOFF3 Goal "t" "None"
08/15/2013 06:02:59 None None 0
```

Esimerkkikoodi 5. Osa LD-mobiilirobotin palauttamasta viestistä QueueMulti-komennon jälkeen. [10.]

EndQueueMulti-rivin jälkeen mobiilirobotti alkaa lähettää päivitystietoja tilauksen kulusta QueueUpdate-alkuisilla viesteillä (esimerkkikoodi 5). Viesteistä nähdään taas kaikki tilauksen osatilausten parametrit. Perustietojen lisäksi viesteistä saadaan selville tilauksen osatilauksien tila, tilaukselle allokoitu mobiilirobotti, tilauksen syöttöpäivämäärä ja aika, osatilauksen valmistumispäivämäärä ja -aika sekä epäonnistumisien määrä. Jos näitä tietoja ajatellaan tämän insinöörityön kannalta, tärkeitä ovat vain tila sekä mobiilirobotin nimi. Osatilauksen tila on esimerkkikoodi 5:n kaikissa osatilauksissa ”Pending”, se tarkoittaa sitä, että tilaus odottaa allokoimista, jollekin mobiilirobotille. Tämän vuoksi paikassa, jossa kuuluisi lukea allokoidun mobiilirobotin nimi, lukee ”None”. Koko viestiketju esimerkkitilauksen lähettämisestä sen valmistumiseen on nähtävissä liitteessä 1.

queueMulti-komennon ohella ARCL-kieli sisältää useita kymmeniä erilaisia komentoja sekä yksittäiselle mobiilirobotille että Enterprise Managerille. Niihin lukeutuvat muun muassa mene lataukseen -komento ja poistu latauksesta-komento sekä sano-komento. Näiden komentojen läpikäyminen ei kuitenkaan ole relevanttia tämän insinöörityön kannalta, joten ne jätetään kiinnostuneen lukijan itse selvitettäväksi.

4.6 Mobile Planner

Mobile Planner on ohjelmisto LD-mobiilirobottien ohjelmoimista ja käyttöönottoa varten. Mobile Plannerin kautta luodaan aiemmin mainittu kartta sekä lisätään sinne halutut komennot ja maalit. Sen avulla voidaan myös monitoroida joko yksittäistä LD-mobiilirobottia tai kokonaista laivuetta. Monitoroinnin lisäksi Mobile Plannerin kautta pystytään lähet-

tämään mobiiliroboteille tai -robotille tilauksia ja ohjaamaan yksittäistä mobiilirobottia kerrallaan. Mobile Planneria ei kuitenkaan ole tarkoitettu toimimaan käyttöliittymänä tilausten syöttöön varsinaisissa sovelluksissa vaan sen ominaisuudet on tarkoitettu lähinnä testikäyttöön.

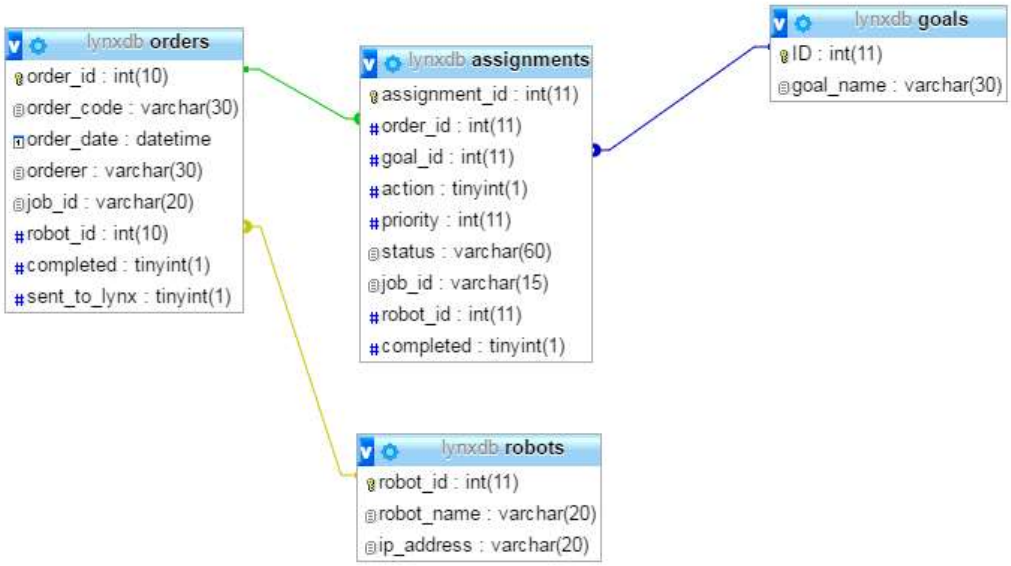
5 Toteutus

Työ toteutettiin välillä loppusyksy 2016 – kevät 2017. Työn toteutuksen tarkastelussa keskitytään toiminnallisuuden kannalta tärkeimpiin asioihin. Käytetyt ratkaisut pyritään selittämään ja avaamaan kuvien ja kattavien selityksien avulla. Läpikäytävät osiot ovat teko- eli aikajärjestyksessä aloittaen tietokannan teosta.

5.1 Tietokanta

5.1.1 Rakenne

Tietokannan rakennetta ja tauluja suunniteltaessa isossa osassa oli mobiilirobotin kommunikointiohjekirja ARCL-manual (Advanced Robot Command Language). Tässä ohjekirjassa selitettiin kattavasti komento, johon perustin oikeastaan koko tämän työn. Komennon avulla mobiilirobotille lähetetään uusi työ. Tietokanta rakentui pääpiirteittäin juuri kyseisen komennon rakenteen ja parametrien mukaan.



Kuva 3. Tietokannan rakenne ja relaatiot.

Tietokanta rakentuu neljästä eri taulusta, kuvion 3 mukaisesti. Näiden taulujen välillä on relaatioita, joiden avulla tietoja voidaan linkittää yhteen eri taulujen välillä. Nämä taulut pitävät sisällään kaikki lähetetyt tilaukset, osatilaukset, kaikki mobiilirobotille asetetut maalit sekä kaikki järjestelmään asetetut mobiilirobotit. Taulut sisältävät erilaisia attribuutteja, joilla on taas erilaisia ominaisuuksia kuten perusarvo, ”DEFAULT”, ja datatyyppi ,”Type”. Jokaisella attribuutilla on tarkkaan määritetty tehtävä. Kaikki attribuutit eivät suinkaan ole mobiilirobottia varten vaan suuri osa attribuuteista on tehty helpottamaan tiedon hakua tietokannasta. Taulut on nimetty hyvin selkeillä ja havainnollistavilla nimillä.

5.1.2 Taulut

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	order_id	int(10)			No	*None*	AUTO_INCREMENT
2	order_code	varchar(30)			No	*None*	
3	order_date	datetime			No	CURRENT_TIMESTAMP	
4	orderer	varchar(30)			No	*None*	
5	job_id	varchar(20)			Yes	NA	
6	robot_id	int(10)			Yes	1	
7	completed	tinyint(1)			Yes	0	
8	sent_to_lynx	tinyint(1)			No	0	

Kuva 4. *orders*-taulun attribuutit ja niiden ominaisuudet.

Kuviosta 4 nähdään, että taulu *orders* pitää sisällään 8 eri attribuuttia, jotka on tarkoitettu varastoimaan tilauksen yleiset tiedot, kuten tilaajan nimi tai koodi, tilauskoodi ja tilauksen tekopäivä. *orders*-taulun tehtävä on toimia osatilauksia sitovana elementtinä, eikä se sisällä mobiilirobotille tärkeitä, tai sille suoraan lähetettäviä tietoja. Attribuutit:

- *order_id*: taulun pääavain. Automaattinen juokseva numero, eli AUTO_INCREMENT, erottamaan tilaukset toisistaan tietokannassa. Tietotyyppi *int* (*integer*) eli kokonaisluku, suluissa pituus (10) eli 10 merkkiä.

- *order_code*: käyttöliittymästä tilaukselle annettava tilauskoodi. Tietotyyppi *varchar(30)* eli hyväksyy numeroita ja kirjaimia.

- *order_date*: automaattisesti tallentuva tilaushetken päivämäärä ja aika.

- *orderer*: käyttöliittymästä tilaukselle syötettävä tilaajan nimi. Tietotyyppi *varchar(30)*.

- *job_id*: tilaukselle ominainen koodi, jonka avulla tilaukset erotetaan toisistaan mobiilirobotin ja väliohjelman välisessä kommunikoinnissa. Saa arvon, kun tilaus lähetetään mobiilirobotille. Vakioarvona NA, joka tulee sanoista Not Announced. Tietotyyppi *varchar(20)*.

- *robot_id*: tilaukselle määrätty mobiilirobotti. Se asetetaan automaattisesti tilausta tehdessä arvoon 1, joka viittaa *robots*-taulun mobiilirobottiin, jonka *robot_id* on 1. Päivittyy kun tilaus lähetetään ja se allokoituu tietylle mobiilirobotille. Tietotyyppi *int(30)*.

- *completed*: tieto onko tilaus suoritettu vai ei. Päivittyy arvoon 1 kun kaikki tilauksen osatilaukset ovat valmiit. Tietotyyppi *tinyint(1)* eli 1 tai 0.

- *sent_to_lynx*: tieto onko tilaus lähetetty mobiilirobotille vai ei. Päivitetään arvoon 1 kun tilaus lähetetty mobiilirobotille. Tietotyyppi *tinyint(1)*.

Kuviosta 5 nähdään taulun *assignments* sisältämät 9 erilaista attribuuttia. Attribuutteihin tallennetaan tilauksen osatilaukset, kuten esimerkiksi jättö maalille ”maali1”. *assignments*-taulu on tärkein taulu mobiilirobotin kannalta ajateltuna.

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	assignment_id	int(11)			No	None	AUTO_INCREMENT
2	order_id	int(11)			No	None	
3	goal_id	int(11)			No	None	
4	action	tinyint(1)			No	None	
5	priority	int(11)			No	None	
6	status	varchar(60)			Yes	NA	
7	job_id	varchar(15)			No	NA	
8	robot_id	int(11)			No	1	
9	completed	tinyint(1)			No	0	

Kuva 5. *assignments*-taulun attribuutit ja niiden ominaisuudet.

- *assignment_id*: taulun pääavain. Automaattinen juokseva numero erottamaan osatilaukset toisistaan tietokannassa. Tietotyyppi *int(11).*

- *order_id*: vierasavain tauluun *orders, order_id*. Tilauksen tunnusluku tietokannassa johon kyseinen osatilaus liittyy, relaatio tauluun *orders*. Tietotyyppi *int(11).*

- *goal_id*: vierasavain tauluun *goals, goal_id*. Osatilauksen määränpää eli maali viittaa tauluun *goals*, josta löytyy tarkemmat tiedot maaleille. Tietotyyppi *int(11).*

- *action*: tehtävä, jonka mobiilirobotti suorittaa maalissa. Tietotyyppi *tinyint(1)*, 1 tarkoittaa noutoa eli *pickup* ja 0 jättöä eli *dropoff*.

- *priority*: osatilauksen ja samalla koko tilauksen prioriteetti. Ainoastaan tilauksen ensimmäisen osatilauksen prioriteetillä on merkitystä. Tietotyyppi *int(11)*.

- *status*: osatilauksen yksilöllinen tila, esimerkiksi ”Pending” tai ”Completed”. Vakioarvona ”NA”. Tietotyyppi *varchar(60)*.

- *job_id*: osatilauksen identifioimiseen päivitettäessä käytettävä kirjain-numerosarja. Kaikilla saman tilauksen alla olevilla osatilauksilla sama. Vakioarvona ”NA”. Tietotyyppi *varchar(15)*.

- *robot_id*: osatilaukselle allokoitu mobiilirobotti, kaikilla tilauksen alla olevilla osatilauksilla sama mobiilirobotti. Päivittyy kun tilaus lähetetään mobiilirobotille. Tietotyyppi *int(11)*.

- *completed*: tieto siitä onko osatilaus valmis. Tietotyyppi *tinyint(1)*.

	#	Name	Type	Collation	Attributes	Null	Default	Extra
	1	**robot_id**	int(11)			No	*None*	AUTO_INCREMENT
	2	**robot_name**	varchar(20)			No	*None*	
	3	**ip_address**	varchar(20)			No	*None*	

Kuva 6. *robots*-taulun attribuutit ja niiden ominaisuudet.

Taulut *goals* (kuvio 7) ja *robots* (kuvio 6) ovat huomattavasti pienempiä kuin edellä käsitellyt *orders* ja *assignments* (kuviot 4 ja 5). Tämä johtuu siitä, että ne ovat eräänlaisia aputauluja. Periaatteessa olisi mahdollista, että *goals*-ja *robots*-taulut jätettäisiin kokonaan pois ja määränpää kirjoitettaisiin assignments taulussa *goal_id*:n sijasta esimerkiksi kenttään *goal*. Mutta koska maaleja on useampia, niitä on helpompi käsitellä muissa tauluissa vain numeroilla eikä kirjainnumeroyhdistelmillä kuten ”goal1” tai ”maali2”. *robots*-taulun attribuutit:

- *robot_id*: taulun pääavain. Jokaiselle mobiilirobotille yksilöllinen juokseva numero mobiilirobottien identifioimiseen tietokannassa. Tähän viitataan muista tauluista. Tietotyyppi *int(11)*.

- *robot_name*: mobiilirobotin nimi. Tietotyyppi *varchar(20)*:

- *ip_address*: mobiilirobotin IP-osoite. Tietotyyppi *varchar(20)*:

	#	Name	Type	Collation	Attributes	Null	Default	Extra
	1	**ID**	int(11)			No	*None*	AUTO_INCREMENT
	2	**goal_name**	varchar(30)			No	*None*	

Kuva 7. *goals*-taulun attribuutit ja niiden ominaisuudet.

- *goal_id*: jokaiselle maalille yksilöllinen juokseva numero maalien identifioimiseen tietokannassa. Tietotyyppi *int(11)*.

- *goal_name*: maalin nimi mobiilirobotin kartassa. Tietotyyppi *varchar(30)*.

5.2 Käyttöliittymä

Tietokannan tekemisen jälkeen käyttöliittymään tulevien sivujen määrä haarukoitui noin 3—5 sivuun. Lopulta käyttöliittymä valmistui sisältäen neljä erilaista ja erilaisilla toiminnoilla varustettua näkymää eli sivua:

- aloitussivu, avoimien tilausten monitorointiin

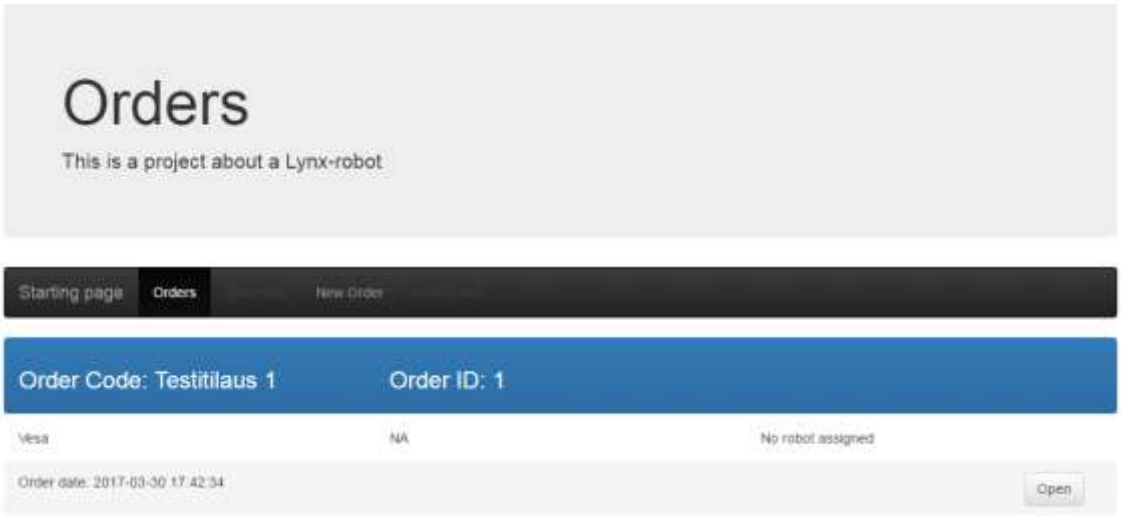
- töiden tarkastelu, tilausten tarkempaan tarkasteluun

- töiden lisäys, uusien töiden lisäys käyttöliittymän kautta

- työn viimeistely, uuden työn viimeistely ennen sen lähettämistä

Sivut sisältävät kaikki tilausten yksinkertaiseen hallintaan tarvittavat työkalut ja toiminnot, ilman turhia kosmeettisia ominaisuuksia. Käyttöliittymä on rakennettu selainpohjaiseksi sovellukseksi käyttäen HTML- ja PHP-kieliä.

5.2.1 Aloitussivu



Kuva 8. Käyttöliittymän aloitussivu, tiedostonimeltään *index.php*.

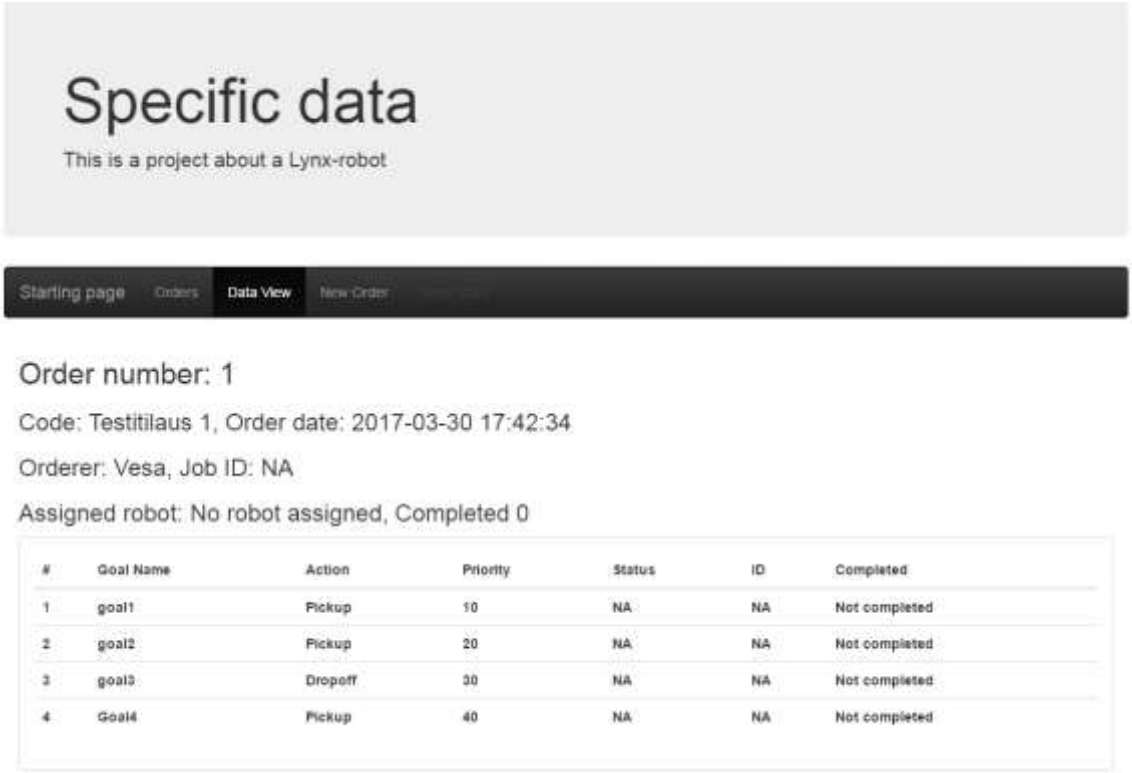
Kuviossa 8 näkyvä käyttöliittymän aloitussivu, tiedostonimeltään *index.php*, on hyvin pelkistetty, ja se sisältää vain muutamia toimintoja. Aloitussivu on tarkoitettu käytettäväksi eräänlaisena monitorointi ja vakiosivuna. Sivulta löytyvät otsikko, navigointipalkki ja navigointipalkin alta alue, johon avoimet tilauksen tulostetaan. Jokaisella näkyvällä tilauksella on lisäksi ”Open”-painike, jota painamalla pääsee näkemään tilauksen tarkemmat tiedot, kuten osatilausten tilat.

Navigointipalkissa on 5 eri klikattavissa olevaa tekstiä. Ensimmäinen, *Starting page,* osoittaa aloitussivulle inxed.php kuten myös sen vieressä oleva *Orders*. Aloitussivuksi suunniteltiin aluksi *Starting page*n taakse ihan oma sivu mutta ajatuksesta luovuttiin, koska sillä ei olisi ollut projektin kannalta juuri hyötyä. Seuraava painikkeena on harmaana oleva *Data View*. Harmaus kertoo siitä, että painike ei ole klikattavissa. Sivulle *Data View* pääsee vain painamalla tulostettujen tilausten *Open*-painiketta. Neljäntenä painikkeena navigointipalkissa on *New Order*, jonka takaa löytyy sivu töiden lisäämiseen. Viimeisenä painikkeena navigointipalkissa on *Order Data*, joka on myös harmaana *Data View*’n tapaan. Tässä pätee sama idea kuin aiemminkin että sivulle ei sallita pääsyä kuin tietystä painikkeesta. Tässä tapauksessa painike sijaitsee sivulla, josta uusi tilaus lisätään.

Kuvassa 8 nähdään navigointipalkin lisäksi yksi avoin tilaus nimeltään *Testitilaus 1*. Kaikki tietokannasta löytyvät avoimet tilaukset tulostetaan allekkain navigointipalkin alle

aina kun sivu avataan. Jokainen tilaus on itsessään ikään kuin laatikko, jonka sisälle asetetaan tilaukselle ominaiset tietokannasta haetut tiedot. Kyseiset tiedot haetaan tietokannasta kahdesta eri taulusta. *orders*-taulusta haetaan *order_id, order_code, order_date, orderer* sekä *job_id* ja *robots*-taulusta haetaan *robot_name*, jonka *robot_id* on sama kuin juuri haetun tilauksen *robot_id*. Tätä tilausta ei siis ole vielä lähetetty mobiilirobotille, koska sille ei ole vielä asetettu mobiilirobottia, eikä sillä ole vakioarvosta poikkeavaa *job_id*-arvoa.

5.2.2 Tilauksen tarkastelu

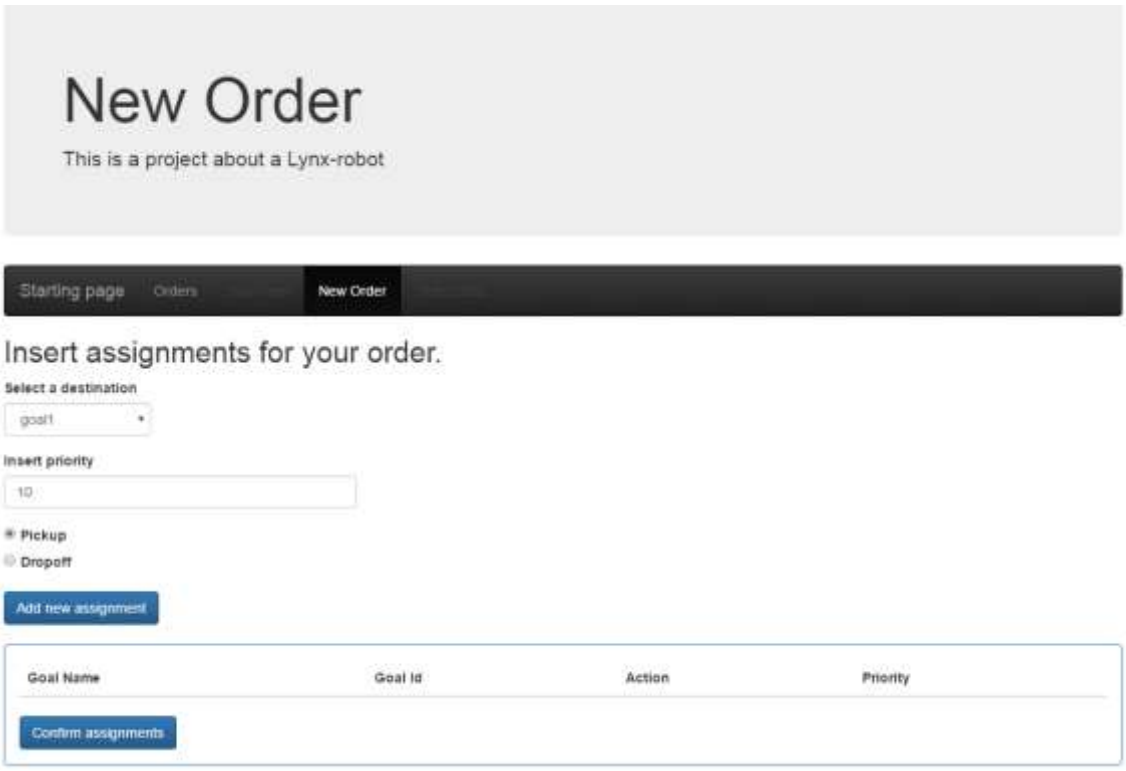


Kuva 9. Sivu valitun tilauksen tarkempaan tarkasteluun, tiedostonimeltään *data_view.php*.

Kun jonkun tilauksen *Open*-painiketta painetaan, avautuu kuvion 9 mukainen sivu. Sivu *Data View*, tiedostonimeltään data_view.php, sisältää valitun tilauksen perustiedot, jotka näkyivät jo aloitussivulla, sekä niiden lisäksi osatilausten tarkat tiedot. Sivun kautta pystytään seuraamaan tarkasti tilauksen etenemistä. Osatilausten valmistumista seuraamalla voidaan myös karkeasti päätellä mobiilirobotin sijainti suhteessa maaleihin.

Tulostettavat tiedot haetaan tietokannasta kahdella eri haulla. Ensin haetaan aloitussivullakin näkyvät perustiedot käyttäen samaa komentoa kuin aloitussivulla ja lopuksi haetaan osatilaukset, joiden *order_id* on sama kuin avatulla tilauksella, eli kuvan 9 tapauksessa *order_id*:llä 1. Osatilaukset sijoitetaan haun jälkeen taulukkoon, joka sitten tulostetaan sivulle.

5.2.3 Tilausten lisäys



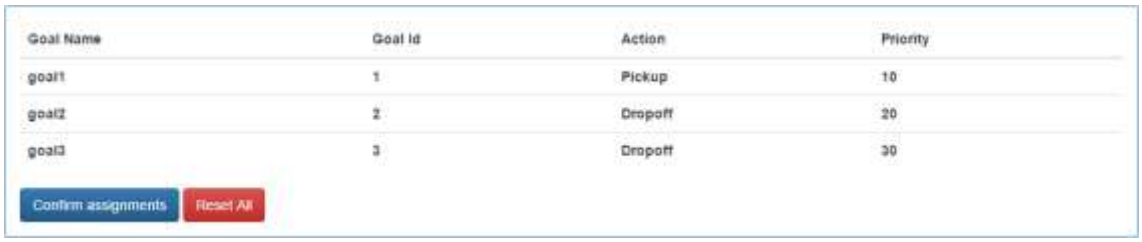
Kuva 10. Sivu tilausten lisäämistä varten, tiedostonimeltään newjob.php.

Tilauksien lisääminen aloitetaan kuvion 10 mukaiselta sivulta ja niiden lisäämisessä on kaksi vaihetta. Ensimmäinen vaihe on osatilausten lisääminen tilaukseen ja toinen vaihe on tilauksen viimeistely. Tämä luku käsittelee osatilausten lisäämistä. Osatilausten lisäämistä varten tarvitaan 3 erilaista syöttökenttää, maali, prioriteetti sekä tehtävä. Näiden alla on vielä taulumuotoinen luettelo jo syötetyille osatilauksille. Edellä mainittua kolmen syöttökentän tietoa voidaan pitää kaikkein tärkeimpänä mobiilirobotin toiminnan kannalta, sillä mobiilirobotille lähetetään ainoastaan kaikki tilauksen maalit, maalikohtaiset tehtävät sekä prioriteetit.

Ensimmäiseen kenttään, jossa otsikkona on *Select destination*, määritetään maali, johon mobiilirobotti halutaan tässä osatilauksessa lähettää. Osatilauksen määränpää eli maali valitaan alasvetovalikosta, jossa näkyvät kaikki maalit. Alasvetovalikossa näkyvät maalit haetaan suoraan tietokannan *goals*-taulusta ja tulostetaan alasvetovalikon sisään. Maalit eivät siis ole kiinteästi ohjelmoitu, vaan ovat sidoksissa tietokannasta löytyviin tauluihin, täten sovelluksesta saadaan dynaamisempi eikä maalien vaihtuessa tarvitse kuin lisätä uudet maalit tietokantaan.

Toiseen kenttään, *Insert priority*, annetaan osatilauksen prioriteetti. Prioriteetin syöttökenttä on automaattisesti sivun avautuessa arvossa 10, eikä sitä tarvitse muuttaa, ellei halua tilaukselle suurempaa tärkeysarvoa. Osatilauksia syötettäessä on otettava huomioon että, ainoastaan ensimmäisen osatilauksen prioriteetilla on merkitystä.

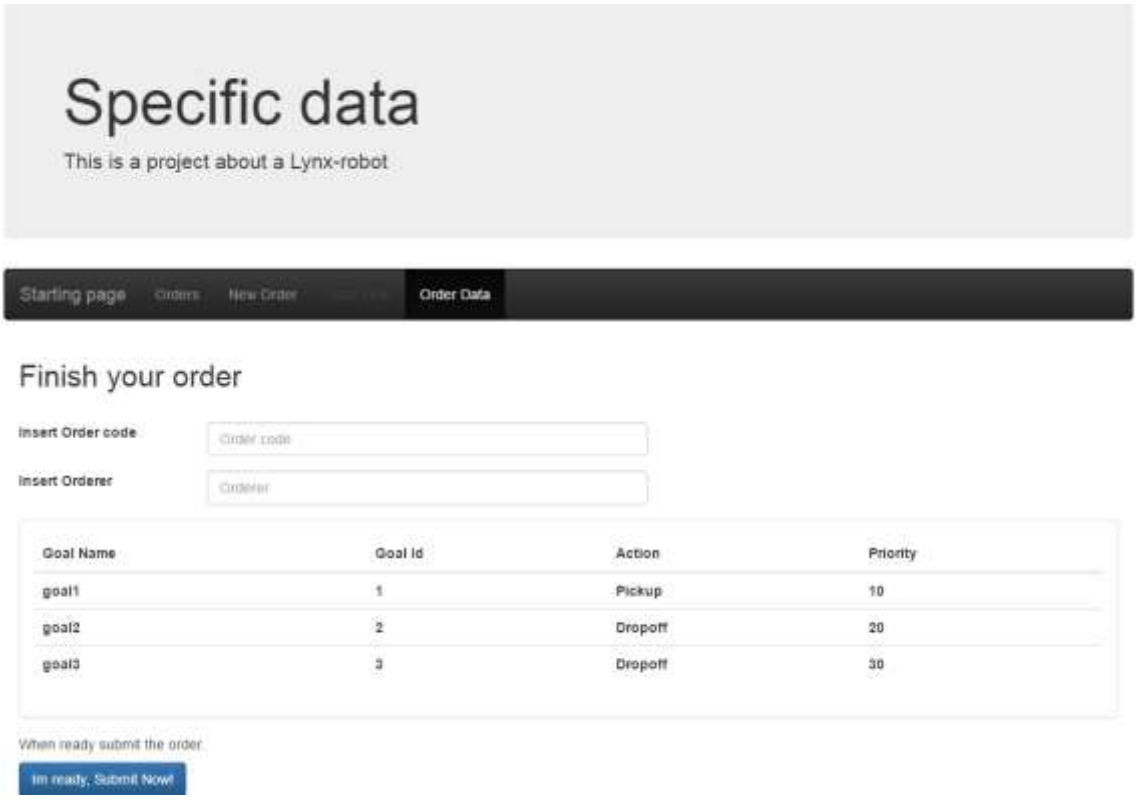
Kolmannessa kentässä valitaan, onko osatilauksessa määriteltävä tehtävä jättö vai nouto. Tämä tapahtuu valitsemalla jommankumman vaihtoehdon radionapista.



Kuva 11. Taulukko syötetyille osatilauksille.

Kun osatilauksen tekeminen on valmista, painetaan *Add new assignment*-painiketta. Kun uusi osatilaus lisätään, sen parametrit tallennetaan eräänlaiseen globaaliin muuttujaan ja tämän globaalin muuttujan arvo sitten tulostetaan taulukkopohjaan. Kun osatilauksia tulee enemmän, ne tulostetaan tilausjärjestyksessä allekkain taulukkoon kuvion 11 mukaisesti. Ensimmäisen osatilauksen lisättyään käyttäjä voi myös tyhjentää osatilauksen painamalla taulukon alareunassa punaisella pohjalla olevaa nappia *Reset all*. Tämä toiminto tyhjentää kaikki tehdyt osatilaukset, yksittäisen osatilauksen poistaminen ei ole mahdollista. Osatilausten nollaaminen tapahtuu myös silloin, mikäli käyttäjä poistuu kesken osatilauksen tekemisen esimerkiksi aloitussivulle. Kun kaikki osatilauksen on tehty, voidaan siirtyä tilauksen hyväksymiseen painamalla *Confirm assignments*-painiketta.

5.2.4 Tilauksen viimeistely



Kuva 12. Sivu tilauksen viimeistelyyn, tiedostonimeltää orderdata.php.

Kun osatilaukset ovat tehty ja painetaan osatilausten lisäykseen tarkoitetulla sivulla *Confirm assignments*-painiketta, päädytään tilauksen viimeistelysivulle, kuvio 12. Tilaus viimeistellään antamalla sille *Inser Order code*-kenttään haluttu tilaustunnus sekä *Insert Orderer* -kenttään tilaajan tunnus tai nimi, jonka jälkeen painetaan vasemmassa alareunassa olevaa painiketta *Im ready, Submit Now!*.

Tilaustunnus voi olla mitä vain, kenttään kelpaavat numerot sekä kirjaimet. Kenttää ei voi jättää tyhjäksi mutta yksikin kirjain tai numero riittää, tunnuksen maksimipituus on 30 merkkiä, kuten tietokannan rakennekuvasta (kuvio 4) nähdään. Samat lainalaisuudet pätevät myös tilaajan nimen tai tunnuksen vastaanottavaan kenttään. Kenttä hyväksyy numeroita ja kirjaimia sekä sen maksimipituus 30 merkkiä. Kumpaankin kenttää syötettävät tiedot ovat kiva tietää dataa ja ne toisaalta ne auttavat myös tilausten erottamisessa toisistaan. Mobiilirobotin toiminnan kannalta tilauskoodilla ja tilaajan nimellä ei ole merkitystä.

Syöttökenttien alla sijaitsee taulukko samalla periaatteella kuin edellisellä sivulla, josta osatilaukset syötettiin, kuva 11. Taulukossa näytetään kaikki osatilaukset, joita ollaan tilauksen syötön yhteydessä syöttämässä tietokantaan. Kun kaikki on valmista, painetaan taulukon alapuolella sijaitsevaa painiketta, *Im ready, Submit Now!*, jonka jälkeen uusi tilaus syötetään tietokantaan.

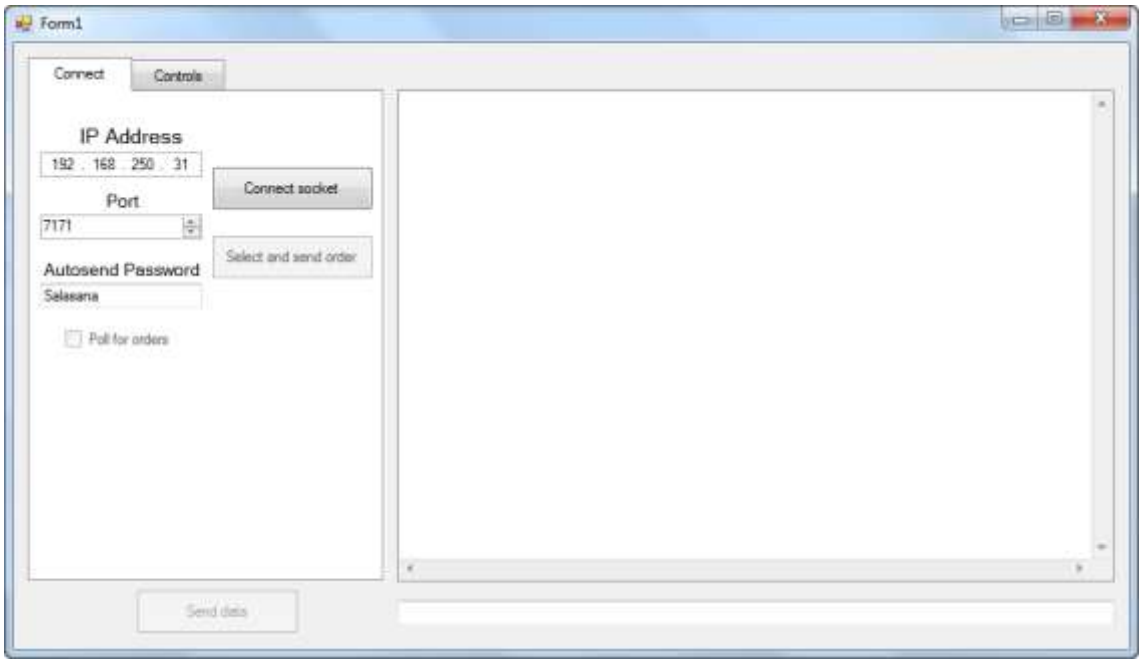
Tilauksen tallentaminen tietokantaan tapahtuu kahdessa erässä. Ensin *orders*-tauluun syötetään itse tilauksen tiedot, jonka jälkeen osatilaukset syötetään *assignments*-tauluun. Kun tilauksen tiedot syötetään *orders*-tauluun, luodaan samalla uusi *order_id*, jonka alle kaikki tiedot kuten tilaaja ja tilauskoodi menevät. Tämän *order_id* arvon avulla *assignments*-tauluun syötettävät osatilaukset saadaan liitettyä juuri syötettyyn tilaukseen.

Kun tilaus on tallennettu tietokantaan, siirrytään käyttöliittymässä automaattisesti aloitussivulle. Mikäli kaikki meni, kuten oli tarkoitus, pitäisi juuri syötetyn tilauksen näkyä ylimmäisenä tilauksien listassa. Seuraava osio käsitteleekin mitä seuraavaksi tapahtuu, kun tilaus on syötetty.

5.3 Väliohjelma

Insinöörityön viimeisenä osiona oli eräänlaisen väliohjelman kehittäminen Visual Studiolla tietokannan ja mobiilirobotin väliin. Väliohjelman tavoitteena oli toimivan kommunikointisovelluksen tekeminen, jonka avulla tietokannassa oleva tieto saataisiin mobiilirobotille ja siltä tuleva tieto saataisiin tallennettua tietokantaan pääkäyttöliittymän hyödynnettäväksi. Näitä toimintoja olisi vielä tarkoitus valvoa ja hallita väliohjelman oman yksinkertaisen käyttöliittymän avulla.

Väliohjelma sisältää sen hallintaan käytettävän käyttöliittymän sekä tietysti sen takana toimivan koodin eli ohjelman. Koodin jokaiseen riviin ei sen tarkemmin syvennytä, vaan toiminnot käydään läpi käyttöliittymän nappien ja kenttien kautta. Näiden nappien kautta hallitaan suurinta osaa koko ohjelman toiminnollisuuksista.

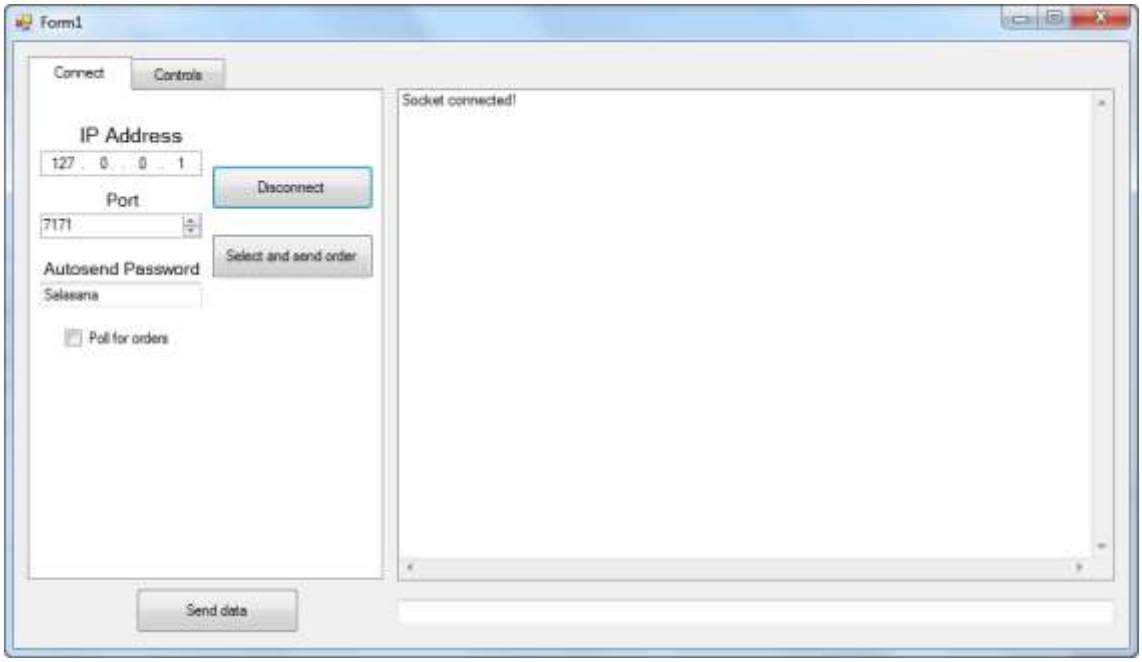


Kuvio 1. Väliohjelman käyttöliittymä.

Väliohjelman käyttöliittymä (kuvio 13) on rakennettu hyvin pitkälti testauksen helpottamista varten. Käytännössä väliohjelman olisi voinut toteuttaa ilman minkäänlaista käyttöliittymää, mutta käyttöliittymä helpottaa ohjelman testausta. Lisäksi käyttöliittymää on helppo laajentaa, mikäli sille tulee tarvetta. Käyttöliittymä sisältää kolme painiketta, yhden valintalaatikon sekä neljä erilaista teksti-/numerokenttää. Toiminnallisten kenttien lisäksi käyttöliittymästä löytyy iso tekstikenttä, jota käytetään mobiilirobotilta tulevien sekä sinne lähetettävien viestien esittämiseen käyttäjälle. Käyttöliittymän vasemmassa yläkulmassa on lisäksi kaksi välilehteä, joista välilehti *Connect* on auki, välilehti *Controls* ei liity tähän projektiin.

Käyttäjän avatessa väliohjelman, avautuu käyttöliittymä kuvion 13 näkymään, jonka jälkeen käyttäjän ensimmäinen tehtävä on luoda yhteys mobiilirobottiin. Mainittakoon että ohjelmalla voidaan myös luoda yhteys mihin tahansa laitteeseen, joka käyttää protokollana TCP/IP-protokollaa, tässä osiossa vastapuolena käytetään Socket Test 3 -ohjelmistoa. Ennen yhteyden muodostamista Connect-painikkeella, täytyy ohjelmalle antaa mobiilirobotin IP-osoite sekä portti, näiden tietojen avulla väliohjelma muodostaa yhteyden mobiilirobotin ja ohjelman välille. *Autosend Password* -kenttää tulee käyttää, mikäli yhdistetään mobiilirobottiin, se määrittää mobiilirobotille lähetettävän salasanan, jonka avulla siihen päästään kiinni. Väärä salasana aiheuttaa yhteyden muodostamisen epä-

onnistumisen. Kun tarvittavat tiedot on syötetty, muodostetaan yhteys painamalla *Connect*-painiketta. Painiketta painettaessa ohjelma ottaa kenttiin syötetyt tiedot ja yrittää yhdistää niiden osoittamaan päätepisteeseen. Mikäli annetut tiedot eivät ole oikein, tai kukaan ei vastaa annetusta päätepisteestä, antaa ohjelma siitä ilmoituksen tekstikenttään. Mikäli yhteyden muodostus onnistuu, antaa ohjelma siitä ilmoituksen *Socket connected!* ja kuviossa 13 harmaana olevat painikkeet tulevat aktiivisiksi, kuten kuviosta 14 nähdään.



Kuvio 2. Käyttöliittymän näkymä yhteyden muodostamisen jälkeen.

Yhteyden muodostamisen jälkeen käyttäjän käytettäväksi avautuvat painikkeet *Select and send order* ja *Send data* sekä *Poll for orders* -valintalaatikko. Myös *Connect* painikkeen teksti vaihtuu arvoon *Disconnect*, ja sitä painamalla voi juuri muodostetun yhteyden sulkea. Käyttäjällä on nyt kaksi eri tapaa, jolla lähettää viestejä yhdistettyyn päätepisteeseen, *Send data* + tekstikenttä yhdistelmä tai *Poll for orders* -valintalaatikko.

Yksinkertaisin lähetyskeino on näkymän alareunassa sijaitseva *Send data* -painike ja sen vieressä oleva tekstikenttä. *Send data*n avulla voidaan yhdistettyyn päätepisteeseen lähettää itsekirjoitettuja komentoja tekstimuodossa, eikä sitä painettaessa haeta tietokannasta mitään. Haluttu komento kirjoitetaan painikkeen vieressä olevaan tekstikent-

tään ja se lähetetään *Send data* -painiketta painamalla. Tämä toiminto on lähinnä testitarkoitusta varten eikä sinänsä liity tilausten lähettämiseen, ellei käyttäjä tiedä mobiilirobotin ymmärtämiä komentoja.

Toinen ja samalla varsinaisten tilausten lähettämiseen tarkoitettu vaihtoehto on laittaa *Autosend password* -kentän alla oleva *Poll for orders* -valintalaatikko päälle. Kun käyttäjä laittaa valintalaatikon päälle-tilaan, alkaa ohjelma sekunnin välein hakemaan lähettämättömiä tilauksia tietokannasta. Mikäli tietokannassa ei ole yhtään lähettävää tilausta, ei ohjelma palauta mitään arvoa, mutta mikäli lähetettäviä tilauksia löytyy, ohjelma hakee ne kaikki, ja yksitellen muodostaa niistä tekstipohjaisen komennon ja lähettää sen päätepisteeseen. Samalla kun haetut tilaukset lähetetään, kirjoittaa ohjelma juuri ne tilauksen lähetetyiksi tietokantaan, ettei tulisi tuplatilauksia. Näin vältytään myös siltä, ettei automaattisesti sekunnin välein tilauksia hakeva ohjelma hakisi aina samoja tilauksia ja lähettäisi niitä. *Poll for orders* -napin tarkoitus on siis tehdä tilausten lähetysprosessi automaattiseksi.

6 Testaus

Insinöörityön aikana käytin paljon aikaa ohjelmien ja mobiilirobotin välisen kommunikoinnin testaamiseen. Ohjelmista käyttöliittymä sekä tietokanta olivat yksinkertaisia testattavia mitä tulee niiden simulointiin, sillä käyttöliittymää ja tietokantaa oli helppo ylläpitää ja muokata lennosta omalla koneella paikallisesti. Suurin haaste testauksen kanssa olikin väliohjelman testaus tietokannan suuntaan sekä varsinkin kommunikointi mobiilirobotin kanssa.

Kuten tiedämme jo tässä vaiheessa, ei mobiilirobotin raahaaminen repussa onnistu. Jouduin keksimään tavan, jolla pystyisin simuloimaan mobiilirobottia, ettei väliohjelman tekeminen jäisi ainoastaan niihin hetkiin, kun olisin koulun automaatiolaboratoriossa mobiilirobotin kanssa. Jo aiemmin työkaluissa mainittu Socket Test 3 tuli todella tarpeeseen. Sen avulla pystyin simuloimaan TCP-palvelinta, jollainen mobiilirobottikin on, omalla tietokoneellani paikallisesti. Simuloinnin ansiosta sain rakennettua toimivan TCP-clientin väliohjelmaani, jonka avulla pystyin yhdistämään mobiilirobottiin, vastaanottamaan ja lähettämään TCP-paketteja sekä halutessani sulkemaan yhteyden. Socket Test-ohjelman luoma TCP-palvelin ei kuitenkaan riittänyt täysin mobiilirobotin simuloimiseen, vaan tehty ohjelma tuli tietysti testata vielä lopuksi mobiilirobotilla.

Varsinaisissa testauksissa oikean mobiilirobotin kanssa ilmaantui useita pieniä ongelmia mutta kaksi on erityisesti mainitsemisen arvoisia. Ensimmäinen ongelma liittyi robotilta tulevien viestien käsittelemiseen, joka ei toiminut suunnitellusti. Toinen ongelma oli paljon ensimmäistä ongelmaa vakavampi. Se liittyi mobiilirobotilta tulevien päivitystietojen käsittelemiseen sekä tietokantaan kirjoittamiseen.

6.1 Ongelma 1

Ensimmäisessä ongelmassa robotilta tulevat paketit tuntuivat tulevan liian nopeasti, eikä ohjelmani pystynyt niitä yksitellen käsittelemään eikä täten myöskään kirjoittamaan oikein muotoiltuna infokenttään. Käytännössä olin rakentanut ohjelmani toimimaan siten, että metodi, joka lukee TCP-paketteja, lukisi aina yhden paketin kerrallaan ja että paketti sisältäisi vain yhden rivin tekstiä. Kuvittelin siis, että mobiilirobotin lähettämät paketit tulisivat niin ”hitaasti”, että pystyisin nappaamaan yhden paketin ja kirjoittamaan sen infokenttään ennen kuin toinen tulisi. Aluksi ajattelin ongelman korjautuvan nopeuttamalla

tietyin väliajoin tapahtuvaa TCP-pakettien lukua mutta huomasin kuitenkin, ettei lukunopeudella ollut juuri ollenkaan merkitystä. Todellisuudessa paketit siis tulevat niin nopeasti, ettei niitä pysty erottelemaan pelkästään nopealla luvulla. Lisäksi huomasin, että mobiilirobotin lähettämät paketit sisältävät lähes aina useamman rivin tekstiä eli tavallaan toisistaan erillään olevaa dataa laitetaan menemään samassa paketissa. Ongelma tavallaan paisui sitä mukaa, mitä syvemmälle tutkin.

Ongelma ratkesi, kun huomasin useamman rivin datapakettia tutkiessani, että paketin jokaisen lauseen perässä oli merkki \n\r. Merkin tarkoitus on merkata rivinvaihdot, jotta dataa vastaanottava ohjelma voi pilkkoa tekstimassan ymmärrettäviksi lauseiksi ja arvoiksi. Merkkien avulla pystyin erottelemaan yhdessä tulleet lauseet toisistaan ja näin sain prosessoitua aina jokaisen lauseen datapaketista. Erottelu auttoi myös paljon viestien tulostamisessa inforuutuun.

6.2 Ongelma 2

Olin vakuuttunut, että ensimmäisen ongelman korjaamisen jälkeen ohjelmani toimisi juuri kuten olin sen suunnitellut ja toteuttanut, mutta toisin kävi. Toinen ongelma ilmeni, kun olin toista kertaa testaamassa ohjelmaa mobiilirobotin kanssa. Huomasin, ettei tilauksien tiedot päivittyneet tietokantaan suunnitellusti. Tietojen olisi pitänyt päivittyä tietokantaan silloin kun mobiilirobotti lähetti päivitystietoja väliohjelmalle, mutta mitään päivitystietoja ei mennyt läpi, ja päivityskäskyt näyttivät häviävän kyberavaruuteen.

Olin rakentanut päivitysprosessin käytännössä sillä tavalla, että tulevista viesteistä katsottaisiin ovatko ne QueueMulti- vai QueueUpdate-alkuisia. QueueMulti-alkuisista viesteistä katsottaisiin osatilauksen järjestysluvun numero-osio, mallia *pickup20* eli siis 20, ja sitä verrattaisiin tietokannassa oleviin *assignment_id* -arvoihin. Tällä tavalla pyrin aluksi päivittämään oikeat tiedot jokaiseen tilaukseen ja osatilaukseen.

QueueUpdate-alkuisista taas katsottaisiin ensin tulleen viestin työnumero, joka on tietokannassa tallennettuna attribuuttiin *job_id*. Sen hakemisen jälkeen haettaisiin vielä mobiilirobotilta tulevasta viestistä tilauksen järjestysnumero. Näitä kahta tietoa käytettäisiin yhdistämään juuri tietty viesti tiettyyn osatilaukseen, jotta juuri oikean osatilauksen tietoja saataisiin päivitettyä.

Ohjelmaa tehdessäni olin siinä luulossa, että tilauksen järjestysnumero olisi juokseva numero, jos näin olisi ollut, olisi sitä voinut verrata osatilauksen *assignment_id*-arvoon ja näin olisi ollut mahdollista määrittää tarkasti, mitä osatilausta päivittää. Tämä järjestysnumero onkin juokseva numero, mutta se nollaantuu aina kun mobiilirobotin sammuttaa ja käynnistää uudestaan. Tekemäni systeemi olisi toiminut, jos robotti olisi päällä heti ensimmäisestä tilauksesta lähtien aina ikuisuuteen saakka mutta yksikin sammutus nollaisi järjestysnumeron ja sotkisi päivitysmekanismin.

Ratkaisin ongelman lisäämällä tietokantani *assingments*-tauluun yhden attribuutin, nimeltään *id*, lisää. *id*-attribuutin tarkoituksena on pitää sisällään osatilauksen järjestysnumero. Seuraavaksi muutin ohjelman kulkua QueueMulti-alkuisen viestin jälkeen. Uusi päivitetty ohjelma tallentaakin jokaisen tilauksen kaikkien osatilausten järjestysnumerot tietokantaan oikeille paikoille, käyttäen apunaan *job_id*, *goal*, ja *priority* arvoja. Tällä tavalla pystytään päivittämään juuri oikeaa osatilausta.

7 Yhteenveto

Insinöörityön tavoitteena oli luoda toimiva demosovellus ja ohjelmisto esittelemään Omron Adept LD -mobiilirobotin toiminnallisuutta tilauspohjaisessa sovelluksessa. Projekti tuntui alussa varsin monimutkaiselta ja hankalalta mutta onneksi sain hyvää apua ohjaajiltani ja työt lähtivät käyntiin. Jouduin insinöörityön myötä opettelemaan paljon uutta ja koenkin että suurin hyöty itselleni muodostuikin eri ohjelmointikielten opettelusta. Ennen työtä en juuri pitänyt ohjelmoinnista, vaikka se kiinnostikin. Väliohjelmaa tehdessä huomasin kuitenkin, että tämähän on hauskaa, ja itsenäisesti toteutetut onnistumiset toivat todella mukavan tunteen.

Projektissa päästiin mielestäni hyvin sille asetettuihin tavoitteisiin. Toiminnallisuus saatiin halutulle tasolle mutta varsinaisen käyttöliittymän käytettävyys ja ulkoasu jäivät hieman vajavaiseksi ajatellen demon käyttötarkoitusta. Käyttöliittymän tulisi olla todella varmatoiminen ja helppokäyttöinen, mutta työssä tehty ei aivan täytä tällaisia vaatimuksia. Työn tehtävä olikin enemmän todeta sovelluksen toteutettavuus, ja käyttöliittymän viilaaminen oikeaan kuntoon voikin alkaa tämän työn päätyttyä.

Lähteet

1 Moghadampour, Godrat. 2013. C# -ohjelmointi. Hansaprint Oy, Vaasa.

2 Visual Studio IDE, Microsoft Visual Studio. Verkkodokumentti. Https://www.visualstudio.com/vs/. Luettu 11.4.2017.

3 ISO/IEC standardi 23270_2006. Http://standards.iso.org/ittf/PubliclyAvailableStandards/c042926_ISO_IEC_23270_2006(E).zip. Luettu 12.4.2017.

4 Eclipse for PHP developers. Verkkodokumentti. Http://www.eclipse.org/downloads/packages/eclipse-php-developers/neon3. Luettu 13.4.2017.

5 XAMPP Installers and Downloads for Apache Friends. Verkkodokumentti. https://www.apachefriends.org/index.html. Luettu 20.4.2017.

6 W. Jason Gilmore. 2008. Beginning PHP and MySQL, From Novice to Professional. Apress , Yhdysvallat.

7 MySQL :: MySQL Editions. Verkkodokumentti. https://www.mysql.com/products/. Luettu 16.4.2017.

8 phpMyAdmin. Verkkodokumentti. https://www.phpmyadmin.net/. Luettu 20.4.2017.

9 Mobiilirobotti | Omron, Suomi. Verkkodokumentti. https://downloads.omron.fi/IAB/Products/Robotics/Mobile%20Robots/Mobile%20Robot/I611/I611-E-01_LD%20Platform%20User's%20Guide.pdf. Luettu 27.4.2017.

10 Mobiilirobotti | Omron, Suomi. https://downloads.omron.fi/IAB/Products/Robotics/Mobile%20Robots/Mobile%20Robot/I617/I617-E-01_Advanced%20Robotics%20Command%20Language%20Reference%20Guide.pdf. Luettu 28.4.2017.

Mobiilirobotin lähettämät tilausten päivitystiedot

```
Example #1 - Using Default job id
queuemulti 4 2 x pickup 10 y pickup 19 z dropoff 20 t dropoff 20
QueueMulti: goal "x" with priority 10 id PICKUP1 and jobid JOB1 successfully queued
QueueMulti: goal "y" with priority 19 id PICKUP2 and jobid JOB1 successfully queued and
linked to PICKUP1
QueueMulti: goal "z" with priority 20 id DROPOFF3 and jobid JOB1 successfully queued and
linked to PICKUP2
QueueMulti: goal "t" with priority 20 id DROPOFF4 and jobid JOB1 successfully queued and
linked to DROPOFF3
EndQueueMulti
QueueUpdate: PICKUP1 JOB1 10 Pending None Goal "x" "None" 08/15/2013 06:02:59 None None 0
QueueUpdate: PICKUP2 JOB1 19 Pending ID_PICKUP1 Goal "y" "None" 08/15/2013 06:02:59 None
None 0
QueueUpdate: DROPOFF3 JOB1 20 Pending ID_PICKUP2 Goal "z" "None" 08/15/2013 06:02:59 None
None 0
QueueUpdate: DROPOFF4 JOB1 20 Pending ID_DROPOFF3 Goal "t" "None" 08/15/2013 06:02:59
None None 0
QueueUpdate: PICKUP1 JOB1 10 InProgress UnAllocated Goal "x" "Bullwinkle (.53)"
08/15/2013 06:02:59 None None 0
QueueUpdate: PICKUP1 JOB1 10 InProgress Allocated Goal "x" "Bullwinkle (.53)" 08/15/2013
06:02:59 None None 0
QueueUpdate: PICKUP1 JOB1 10 InProgress Driving Goal "x" "Bullwinkle (.53)" 08/15/2013
06:02:59 None None 0
QueueUpdate: PICKUP1 JOB1 10 Completed None Goal "x" "Bullwinkle (.53)" 08/15/2013
06:02:59 08/15/2013 06:03:20 0
QueueUpdate: PICKUP2 JOB1 19 InProgress UnAllocated Goal "y" "Bullwinkle (.53)"
08/15/2013 06:02:59 None None 0
QueueUpdate: PICKUP2 JOB1 19 InProgress Allocated Goal "y" "Bullwinkle (.53)" 08/15/2013
06:02:59 None None 0
QueueUpdate: PICKUP2 JOB1 19 InProgress Driving Goal "y" "Bullwinkle (.53)" 08/15/2013
06:02:59 None None 0
QueueUpdate: PICKUP2 JOB1 19 Completed None Goal "y" "Bullwinkle (.53)" 08/15/2013
06:02:59 08/15/2013 06:03:33 0
QueueUpdate: DROPOFF3 JOB1 20 InProgress UnAllocated Goal "z" "Bullwinkle (.53)"
08/15/2013 06:02:59 None None 0
QueueUpdate: DROPOFF3 JOB1 20 InProgress Allocated Goal "z" "Bullwinkle (.53)" 08/15/2013
06:02:59 None None 0
QueueUpdate: DROPOFF3 JOB1 20 InProgress Before Goal "z" "Bullwinkle (.53)" 08/15/2013
06:02:59 None None 0
QueueUpdate: DROPOFF3 JOB1 20 InProgress Driving Goal "z" "Bullwinkle (.53)" 08/15/2013
06:02:59 None None 0
QueueUpdate: DROPOFF3 JOB1 20 InProgress After Goal "z" "Bullwinkle (.53)" 08/15/2013
06:02:59 None None 0
QueueUpdate: DROPOFF3 JOB1 20 Completed None Goal "z" "Bullwinkle (.53)" 08/15/2013
06:02:59 08/15/2013 06:03:47 0
QueueUpdate: DROPOFF4 JOB1 20 InProgress UnAllocated Goal "t" "Bullwinkle (.53)"
08/15/2013 06:02:59 None None 0
QueueUpdate: DROPOFF4 JOB1 20 InProgress Allocated Goal "t" "Bullwinkle (.53)"
08/15/2013 06:02:59 None None 0
QueueUpdate: DROPOFF4 JOB1 20 InProgress Driving Goal "t" "Bullwinkle (.53)" 08/15/2013
06:02:59 None None 0
QueueUpdate: DROPOFF4 JOB1 20 Completed None Goal "t" "Bullwinkle (.53)" 08/15/2013
06:02:59 08/15/2013 06:04:03 0
```