WLAN-verkon jatkuvatoiminen tiedonkeruujärjestelmä

Henri Sassali
Opinnäytetyö
Kevät 2012
Tietotekniikan koulutusohjelma
Oulun seudun ammattikorkeakoulu

ALKULAUSE

Tämä opinnäytetyö on tehty Rautaruukki Oyj:n Raahen tehtaan toimeksiannosta talvella 2011-12. Opinnäytetyö on tehty pääsääntöisesti kotonani Pattijoella sekä työpaikallani Ruukin Raahen tehtaan terässulatolla.

Haluan kiittää opinnäytetyön valvojana toiminutta Oulun seudun ammattikorkeakoulun Raahen tekniikan ja talouden kampuksen tuntiopettajaa Juha Rätyä, joka auttoi työn hahmottelussa ja kirjallisen osion määrittelemisessä. Kiitokset kuuluvat myös Rautaruukki Oyj:n edustajalle, kollegalleni ja työnvalvojalle Juha Kangasluomalle, jolta sain hyviä vinkkejä ja tietoja työhön liittyen. Vapaudesta tämän työn tekemiseen ja avusta tietokantaongelmien ratkaisuun tulee kiittää myös esimiestäni Esa Tuomikoskea.

Erityiskiitokset haluan lausua ystävälleni Mauno Mattilalle niistä yli viisitoista vuotta sitten opettamistasi BASIC-ohjelmoinnin perusteista, jotka saivat minut kiinnostumaan tästä alasta.

Raahessa 8.2.2012

Henri Sassali

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietotekniikan koulutusohjelma

Tekijä: Henri Sassali
Opinnäytetyön nimi: WLAN-verkon jatkuvatoiminen tiedonkeruujärjestelmä
Työn ohjaajat: Juha Räty, Juha Kangasluoma
Työn valmistumislukukausi ja -vuosi: Kevät 2012 Sivumäärä:45 + liitteet

---

Insinöörityön tavoitteena oli toteuttaa järjestelmä, joka mittaa laaja-alaisen ulkoilma-WLAN–verkon toimivuutta. Nykyisellään lähes koko Ruukin Raahen tehtaan ulkoalueen kattava WLAN–verkko toimii vaihtelevasti eikä verkon diagnosointiin ole hyviä työkaluja. Jatkuvasti kehittyvät WLAN–verkkoa käyttävät laitteet ja sovellukset asettavat vaatimukset verkon toiminnalle koko ajan korkeammalle.

Työ aloitettiin etsimällä sopivat lähteet tarvittaville tiedoille ja suunnittelemalla toteutustapa niiden saamiseksi. Jokaisen toteutustavan toimivuus ja vaatimukset testattiin, minkä pohjalta aloitettiin järjestelmän rakentaminen. Työnsuunnittelussa käytettiin Windows-pohjaisia työkaluja ja varsinainen ohjelmointityö on tehty Microsoft Windows Visual Studio 2010 –kehitysympäristössä C#-ohjelmointikielellä. Mittaustulokset tallennettiin Microsoft SQL Server 2008 –tietokantaan.

Järjestelmästä saadut mittaustiedot auttavat vian paikantamisessa, verkon kehittämisessä ja ohjaavat verkon ylläpitäjiä parempiin laitehankintoihin. Ajoneuvojen sijaintitietojen seurantaa voi hyödyntää myös logistiikkaosaston henkilöstö. Järjestelmässä ei ole varsinaisia mittalaitteita, vaan mittalaitteina toimivat verkkoon yhdistetyt tietokoneet.

Työn tuloksena syntyi jatkuvatoiminen WLAN–verkon tiedonkeruujärjestelmä, jota voi hyödyntää kunnossapito-, kehitys- ja logistiikkaosastoilla työskentelevä henkilöstö.

Asiasanat: WLAN-verkko, tiedonkeruu, Visual Studio 2010, C#-ohjelmointikieli, SQL

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology

Author: Henri Sassali
Title of thesis: WLAN-network's continuous data logging system
Supervisors: Juha Räty, Juha Kangasluoma
Term and year of completion: Spring 2012 Number of pages:45 + appendices

---

The purpose of this Bachelor's thesis was to create a system, which measures the functionality of WLAN-network located in a wide outdoor area. At the moment the WLAN-network covering almost the whole outdoor area of Ruukki's Raahe factory sometimes functions non-steadily and there are no good tools for network diagnostics. The continuously developing devices using WLAN-network set the functionality requirements of the network all the time higher.

The creation of the system was started by searching sources for the needed measurement data and then planning the right implementation ways for gathering them. The functionality of the implementations were tested and then used as a base for the creation. Windows-based tools were used in the planning work and the main coding work was done in a Microsoft Windows Visual Studio 2010 development environment and in a C#-programming language. The measurements were stored into a Microsoft SQL Server 2008 -database.

The measurements got from the system help service personnel to locate problems and to develop the network. They also direct network administrators to acquire better devices. Also, the staff working in logistics department can benefit from the system's vehicle tracking feature. There are no exact measurement devices in the system, but the computers connected to the network work as measuring devices.

The result of this thesis was a continuously working WLAN-network's data logging system, which personnel working in service, development and logistics departments can benefit from.

Keywords: WLAN-Network, data logging, Visual Studio 2010, C#-programming language, SQL

Sisällys

1 JOHDANTO

Ruukin Raahen tehtaalla on lähes koko tehdasalueen kattava langaton lähiverkko. Tehdasalue on kooltaan noin 529 hehtaaria. Verkkoa käyttävät pääsääntöisesti tuotantolaitteisiin, kuten esimerkiksi nostureihin, trukkeihin ja kurottajiin asennetut tietokoneet. Verkkoa kutsutaan tuotannon WLAN-verkoksi. Lähes kaikissa ajoneuvoissa on tietokoneet, joilta ajoneuvojen kuljettajat käyttävät erilaisia toiminnanohjausjärjestelmiä. Ajoneuvoja eli verkon käyttäjiä on noin 40 kappaletta. Ajoneuvojen tietokoneita kutsutaan tuotantokoneiksi. /1/

Tuotantoverkkoon kuuluu useita kymmeniä tukiasemia. Niitä ei hallita yksittäisinä laitteina, vaan jokaista tukiasemaa hallitsee kontrolleri. Kontrollereita on käytössä yhteensä neljä kappaletta. Tukiasemat ja kontrollerit ovat *Ciscon* valmistamia. /2/

Aina ajoittain tuotantoverkon toiminnassa esiintyy ongelmia. Ongelmat ilmenevät tuotannonohjausjärjestelmien toimimattomuutena tai järjestelmien toiminnan hitautena. Useasti ajoneuvojen kuljettajat ilmoittavat niistä puhelimitse verkkoa ylläpitäville henkilöille. Ongelmien syitä on vaikea selvittää, ellei kyseessä ole jokin selkeä mekaaninen vika kuten vaikka rikkinäinen antenni. /3/

Tämän työn tarkoituksena on luoda järjestelmä aiemmin mainittujen ongelmien selvittämiseen ja langattoman verkon toiminnan monitorointiin. Järjestelmän keräämän tiedon on myös tarkoitus auttaa verkon käytettävyyden kehittämisessä. Työssä ei niinkään oteta kantaa, mistä mahdolliset yhteyksiä koskevat ongelmat johtuvat, vaan pyritään näyttämään, missä ja milloin ongelmia ilmenee.

Laiterekisteriä eli perustietoja laitteista ylläpidetään tällä hetkellä *Excel*–taulukossa. Taulukon ongelma on, ettei sitä voi muokata kuin yksi käyttäjä kerrallaan ja tietojen kirjaamisessa voi tapahtua inhimillisiä virheitä. Tämän työn puitteissa on tarkoitus muuttaa taulukko tietokantapohjaiseksi ratkaisuksi, johon järjestelmä kirjaa automaattisesti mahdollisimman kattavasti laitteistoa koskevat muutokset.

2 MÄÄRITELMÄ

Tämän opinnäytetyön tavoitteena on luoda järjestelmä auttamaan laaja-alaisessa ulkoilma-WLAN-verkossa esiintyvien ongelmien selvittämisessä. Tällä hetkellä ongelmien selvittämiseen käytetään erilaisia menetelmiä, kuten ping-komentoa, kontrollereiden vikalokien seurantaa tai verkkoyhteyden toiminnan seuraamista ajoneuvosta ajonaikana.

Ping-komento on yleisin verkkoyhteyksien vianselvityksissä käytetty TCP/IP-komento. Tietokone, jolla ping-komento suoritetaan, lähettää kohde koneelle ICMP-protokollan mukaisen viestin, johon kohdekone vastaa lähettämällä takaisin saman protokollan mukaisen vastausviestin. Jos vastausviestiä ei saavu sille koneelle, millä ping-komento suoritettiin tietyn ajan sisällä (yleensä 5 sekuntia), verkkoyhteys näiden kahden koneen välillä voidaan todeta toimimattomaksi. /27/

Ping-komento on helppokäyttöinen ja nopea, mutta sen tuloksia ei aina voida pitää täysin luotettavina Quality of Servicen (QoS) takia. QoS on kokoelma eri standardeja ja mekanismeja korkean suorituskyvyn takaamiseksi kriittisille sovelluksille. Tilanteissa, joissa kohdekoneen sovellukset tarvitsevat laajalti verkkoyhteyttä, QoS saattaa luokitella ping-viestit alikriittisiksi ja viivästyttää niihin vastaamista. Koneella, jolla ping-komento suoritetaan, ei nähdä aiheuttiko QoS lisäviivettä vastausaikaan. Tässä opinnäytetyössä on tarkoitus testata verkkoyhteyden toimivuus sovelluksen suorittamilla tietokantaoperaatioilla, joiden kriittisyys on rinnastettavissa muihin sovelluksiin. /26/

Ongelmatilanteiden selvittäminen kontrollereiden vikalokeista on hyvin työlästä, sillä lokit sisältävät paljon ongelmatilanteiden selvityksen kannalta turhaa tietoa. Lokien tekstiä ei myöskään voi suoraan suodattaa kontrollereiden käyttöliittymältä, vaan suodatus täytyy tehdä kopioimalla teksti erilliseen sovellukseen suodatusta varten. Tässä työssä luotavalla järjestelmällä pyritään esittämään ongelmatilanteisiin liittyvien laitteiden MAC-osoitteet, joita voidaan käyttää hakusanoina lokien seurannassa.

Lisäksi yksi tämän työn tavoitteista on, että verkkoyhteyksien toimintaa voi seurata omalta henkilökohtaiselta tietokoneelta, jolloin toimintaa ei tarvitse lähteä seuraamaan itse ajoneuvoon.

2.1 Järjestelmän perusvaatimukset

Järjestelmän tulee osoittaa, missä ja miten langaton lähiverkko toimii, sekä auttaa verkon ylläpitäjiä vianhakutilanteissa ja verkon toiminnan kehittämisessä. Laitteiston kunnossapitäjiä varten järjestelmään rakennetaan heidän töitään helpottavia ominaisuuksia, kuten ajoneuvojen sijaintien seuranta sekä aktiivisesti päivittyvä laiterekisteri. Ajoneuvojen sijainnin seurantaa voivat hyödyntää myös ajoneuvourakoitsijan työnjohto ja muut kuljetussuunnittelijat.

Työssä ei käytetä varsinaisia mittalaitteita, vaan mittalaitteina toimivat samaan mitattavaan verkkoon yhdistetyt ja erilaisiin ajoneuvoihin asennetut tietokoneet. Ajoneuvojen kuljettajat käyttävät tietokoneilla erilaisia tuotannonohjausjärjestelmiä, josta he saavat esimerkiksi logistiset työohjeet. Häiriöttömän tuotannon kannalta järjestelmien, tietokoneiden ja verkkoyhteyksien toiminta on kriittisessä asemassa. Työssä luotava tiedonkeruusovellus ei siis saa altistaa muita toimintoja häiriöille.

2.2 Laitteisto järjestelmää varten

Järjestelmän rakentamista varten on käytettävissä HP PROLIANT DL380 G7 –palvelin. Palvelimen käyttöjärjestelmänä on Windows Server 2008. Palvelimelle on asennettu MS SQL Server 2008 R2 –tietokanta, jota työssä on käytettävä. MS SQL Server -tietokanta on palvelu tiedon säilyttämiseen ja käsittelyyn relaatiomuodossa. /4/

KUVA 1. Tehdasolosuhteet kestävä testikone

Ohjelmointityötä varten käytettävissä on kannettava tietokone. Testikoneena toimii tuotantokoneiden kanssa identtinen, tehdasolosuhteet kestävä tietokone (KUVA 1). Muutoin järjestelmää tulee pystyä käyttämään henkilökohtaisilta tietokoneilta.

2.3 Kehitysympäristö ja ohjelmointikieli

Ohjelmointikieli ja kehitysympäristö työtä varten oli vapaavalintainen. Työssä päädyttiin käyttämään C#-ohjelmointikieltä ja Microsoft Visual Studio 2010 -kehitysympäristöä, koska insinöörityön tekijällä oli kyseisestä kielestä ja ympäristöstä runsaasti kokemusta aiemmista ohjelmistoprojekteista.

C#-ohjelmointikieli on yksi niistä keskitason ohjelmointikielistä, joilla ohjelmoijat voivat luoda suoritettavia sovelluksia. C#-ohjelmointikieli yhdistää tehokkaan mutta monimutkaisen C++-ohjelmointikielen ja enemmän helppokäyttöisen mutta monisanaisen Visual Basic -ohjelmointikielen. /6/

C#-ohjelmointia varten yleisin käytetty kehitysympäristö on Microsoft Visual Studio 2010. Kehitysympäristö on helppokäyttöinen, selkeä ja sisältää valmiit toiminnallisuudet sovelluksen ja sovelluspäivitysten julkaisuun. Edellä mainitut seikat vaikuttivat myös vahvasti kehitysympäristön valinnassa.

2.4 Mitattavat tiedot

Järjestelmän tulee mitata langattoman verkkoyhteyden toimintaan vaikuttavat olennaisimmat tiedot. Lähes jokaisessa langattoman verkkokortin mukana tulevassa diagnostiikkasovelluksessa esitetään vastaanotetun signaalin voimakkuus ja laatu. Näitä voitiin siis pitää vähimmäisvaatimuksina mitattaviksi tiedoiksi. Lisäksi järjestelmälle olennaista on selvittää, mihin tukiasemaan laite on yhdistettynä.

Langattoman verkon toiminnan selvityksessä käytettävän sovelluksen tulisi saada selville seuraavat verkkoyhteyttä koskevat tiedot:

- vastaanotetun signaalin voimakkuus
- signaalin laatu
- signaalin kohina
- häiriötaso
- pakettien lähetyksen suorituskyky.

Lisäksi järjestelmälle olennaista on selvittää tukiasemien peittoalueet niillä alueilla, missä tuotantolaitteet liikkuvat. Näkemällä kahden eri tukiaseman peittoalueet voidaan havaita, missä tuotantolaitteet vaihtavat tukiasemaa eli missä niin sanottu ”roaming” tapahtuu. Roaming tarkoittaa tilannetta, jossa verkkoon yhdistetty laite vaihtaa käyttämäänsä tukiasemaa. /7/

3 TOIMINTAYMPÄRISTÖ

3.1 Tuotannon WLAN-verkko

Tuotantoverkkoon kuuluu yhteensä 111 eri tukiasemaa (KUVA 2), jotka kaikki ovat Ciscon valmistamia. Niitä ei hallita yksittäisinä laitteina, vaan hallintaa hoidetaan keskitetysti kontrollereilla. Kontrollereita on neljä kappaletta, joista yksi on varalaite. Yhden kontrollerin hallinnassa voi olla maksimissaan 50 tukiasemaa. Kontrollerit ovat Ciscon valmistamia, tarkemmalta malliltaan AIR-WLC4402-50-K9 V01 (LIITE 2). /2/

KUVA 2. WLAN-tukiasema

Tuotantoverkko kattaa lähes koko tehdasalueen. Langattoman verkon toiminnan kannalta tehdasalue on olosuhteiltaan haastava. Alueella on useita kymmeniä kooltaan isoja ja korkeita halleja sekä muita rakennuksia, joiden ulkoseinät ovat yleensä metallirakenteisia. Metalliset rakenteet kuten peltiseinät aiheuttavat heijastuksia langattoman verkon signaaleihin. Tehdasalueella on myös lukuisia muita langattomia radioyhteyksiä käyttäviä järjestelmiä, jotka saattavat aiheuttaa häiriöitä tuotannon WLAN-verkossa.

3.2 Verkon käyttäjät

Tuotantoverkkoa käyttävät pääasiassa nostureihin, trukkeihin, kurottajiin asennetut tietokoneet. Satamassa verkkoa käyttävät myös satamahenkilöstön käyttämät käsipäätteet. Verkkoa käytetään ympäri vuorokauden.

Kuva 3. Kurottaja

3.3 Havaitut ongelmat

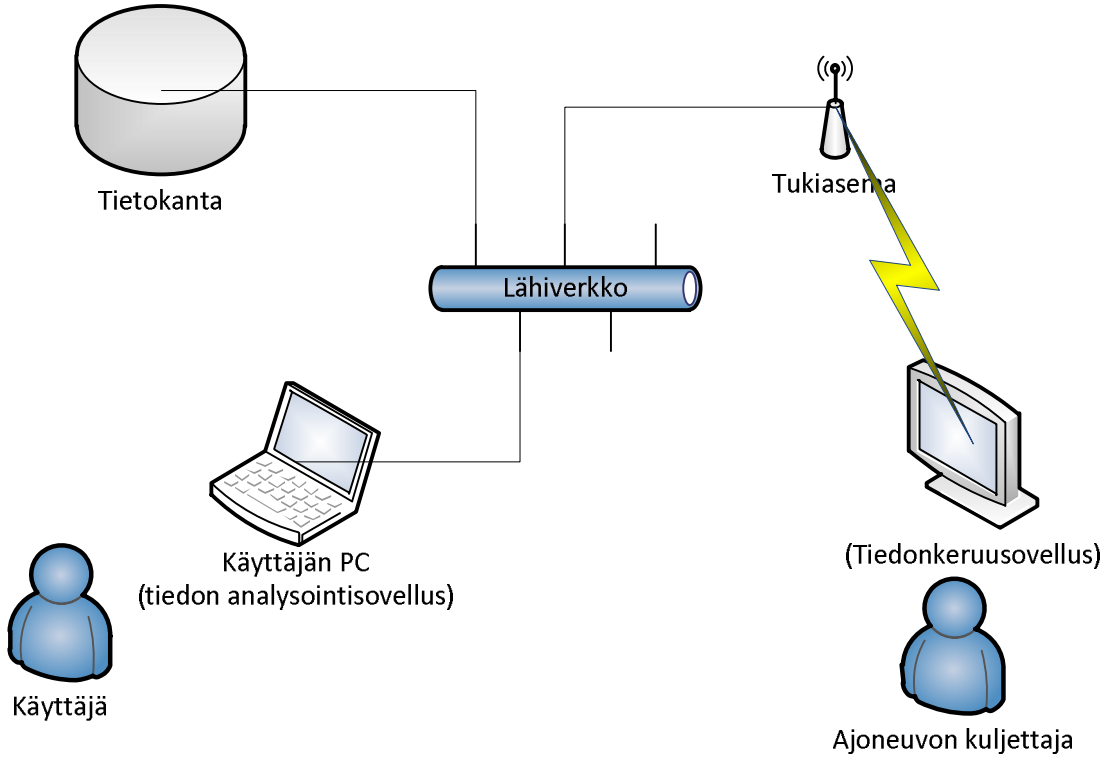
Aina ajoittain ja useasti samoilla alueilla verkon toiminnassa on havaittu ongelmia. Ongelmatilanteet ovat vaikeita selvittää, koska useasti kunnossapitohenkilöstön saapuessa paikalle ongelmia ei ole enää havaittavissa.

Osa tukiasemista on MESH-tukiasemia, jotka eivät ainoastaan jaa lähiverkkoa langattomasti, vaan ovat itsekin yhdistettynä lähiverkkoon langattomasti. MESH-tukiasemat keskustelevat siis langattomasti keskenään, eikä varsinaiseen laitteeseen tule muuta kuin sähkönsyöttö. Normaalitukiasemaan tulee sähkönsyötön lisäksi verkkoyhteys verkkokaapelin välityksellä. /2/

MESH-tukiasemien välisissä yhteyksissä on epäilty olevan ongelmia, jotka ilmenevät esimerkiksi tilanteita, jolloin roaming ei toimi oikein. Ajoneuvo on saattanut olla 10 metrin päässä tukiasemasta, mutta on kuitenkin yhdistettynä 150 metrin päässä olevaan tukiasemaan. Yleisin syy verkon toimimattomuuteen on kuitenkin rikki mennyt antenni ajoneuvossa. /2,3/

4 TOTEUTUS

Työ aloitettiin suunnittelemalla järjestelmä pääpiirteittäin. Määrittely asetti selkeät lähtökohdat suunnittelulle, koska mittalaitteina tulisi toimia jo olemassa olevat tietokoneet ja tietokannan tulisi olla erillisellä palvelimella.



KUVA 4. Hahmotelma toteutettavasta järjestelmästä

Järjestelmä päädyttiin rakentamaan kahdesta eri sovelluksesta, tiedonkeruusovelluksesta sekä tiedon analysointisovelluksesta. Erillistä palvelinsovellusta ei katsottu tarpeelliseksi tehdä, sillä tietokanta itsessään sisältää jo monia ominaisuuksia, esimerkiksi käyttäjäoikeuksien määrittelyn. Toteutuksen voi jakaa neljään eri vaiheeseen: mittatietojen lähteiden selvittämiseen, tiedonkeruusovelluksen luomiseen, tietokannan rakentamiseen sekä tiedon analysointisovelluksen luomiseen. Vaiheista on kerrottu omissa luvuissaan.

5 MITTATIETOJEN LÄHTEET

5.1 GPS-paikannin

Järjestelmää varten oli kartoitettava sopivat GPS–paikantimet ajoneuvojen sijaintitietojen saamiseksi. GPS-järjestelmän käyttöä pidettiin itsestään selvyytenä, koska tarkoitus on mitata vain ulkoalueella olevaa langatonta verkkoa ja tarkemmin sitä käyttävien tuotantokoneiden verkkoyhteyttä.

GPS on Yhdysvaltojen puolustusministeriön rakentama kahdestakymmenestäneljästä eri kiertoradalla kiertävästä satelliitista koostuva satelliittipaikannusjärjestelmä. GPS on alun perin tarkoitettu sotilaskäyttöön mutta 80-luvulla Yhdysvaltojen hallitus salli järjestelmän siviilikäyttöön. GPS toimii missä päin maailmaa tahansa, sääolosuhteista ja kellonajasta huolimatta. Järjestelmän käyttö on maksutonta. /8/

GPS-satelliitit kiertävät maapallon kaksi kertaa päivässä ja lähettävät informaatiota signaaleina maapallolle. GPS-paikantimet vastaanottavat signaalit ja käyttävät kolmiomittausta käyttäjän tarkan sijainnin laskemiseen. Käytännössä GPS-paikannin vertaa aikaa, jolloin satelliitti lähetti signaalin, aikaan, jolloin paikannin vastaanotti signaalin. Aikaero kertoo paikantimelle etäisyyden signaalin lähettäneeseen satelliittiin. Laskutoimitus suoritetaan useammalle satelliitille, jolloin saadaan selville paikantimen tarkka sijainti. /8/

Erillisiä lisälaitteita varten ajoneuvojen tietokoneissa oli käytettävissä vain USB–portti, joten laitteen tulisi olla kyseiseen porttiin liitettävä. Lisäksi laitteiston tulisi kestää tehdasolosuhteet ja olla yhteensopiva Windows XP/7 –käyttöjärjestelmien kanssa.

Edellä mainitut vaatimukset täyttäväksi laitteeksi osoittautui GlobalSat BU-353(SiRF III) GPS -paikannin, joka hankittiin työtä varten. Laitteen mukana tuleva ajuri luo tietokoneelle virtuaalisen sarjaportin, johon laite yhdistettynä lähettää sijaintitiedot NMEA 0183 –standardin mukaisena viestinä. Käytännössä ajuri siis huijaa tietokonetta luulemaan, että siihen on kytketty jokin sarjaliikennetietoa lähettävä laite, vaikka todellisuudessa tietokoneeseen on yhdistetty USB-laite. /9/

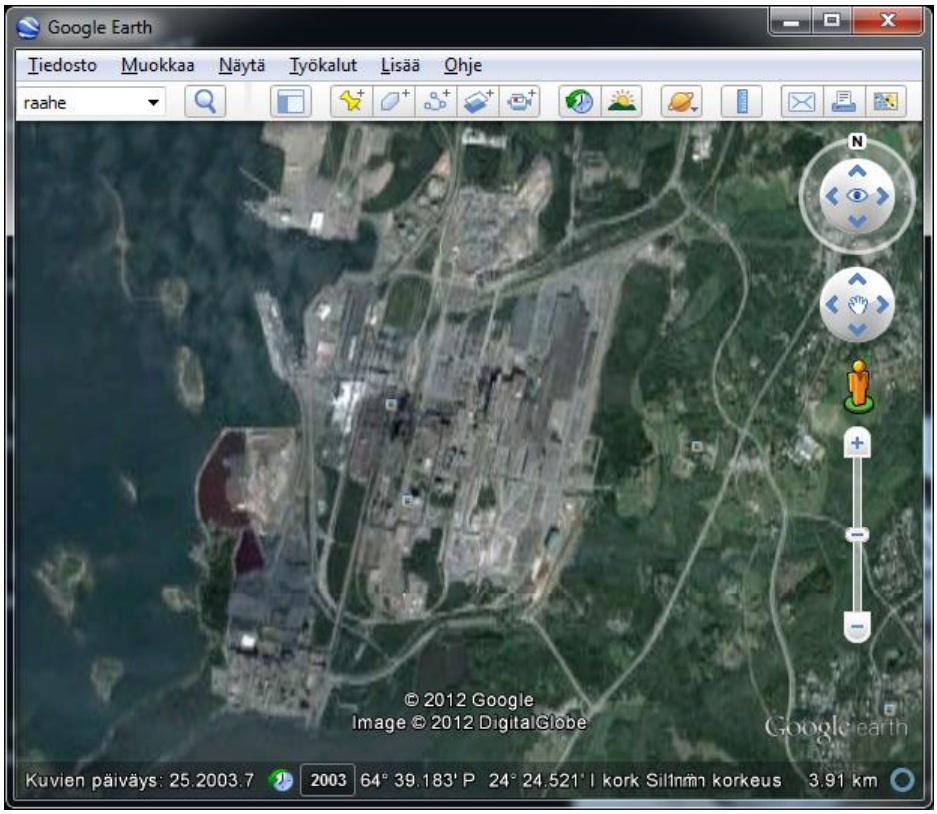
KUVA 5. GPS-paikannin

NMEA 0183 –standardi on rajapintastandardi, joka määrittää sähköisten signaalien vaatimukset, tiedonsiirtoprotokollan ja tiedonsiirtoajan sekä lauseformaatit 4800 baudisille sarjaliikenneväylille. Tieto on luettavassa ASCII-formaatissa ja voi sisältää tietoa esimerkiksi sijainnista, nopeudesta, korkeudesta tai taajuuksien käytöstä. /10/

Laitteen toimivuus testattiin ajurin asennuksen jälkeen lähes jokaiselta Windows XP -työasemalta löytyvällä Hyperterminal-sovelluksella. Testauksen tuloksena voitiin laite todeta toimivaksi ja NMEA-viestin rakenne nähtiin konkreettisesti. Yhteensopivuuden takaamiseksi sama testi toteutettiin myös Windows 7 –käyttöjärjestelmällä.

5.2 GPS-paikantimen testisovellus

Laitteen mittatarkkuuden testausta varten ohjelmoitiin sovellus, joka näyttäisi laitteen sijainnin kartalla. Karttapohjana päädyttiin käyttämään Googlea Earth –sovelluksesta otettua kuvakaappausta, koska sovelluksesta pystyi lukemaan suoraan kartan äärilaitoja vastaavat GPS–koordinaatit. Koordinaatteja tarvitaan sijaintitiedon piirtämiseen sovelluksessa.



KUVA 6. Tehdasalue Google Earth -sovelluksessa

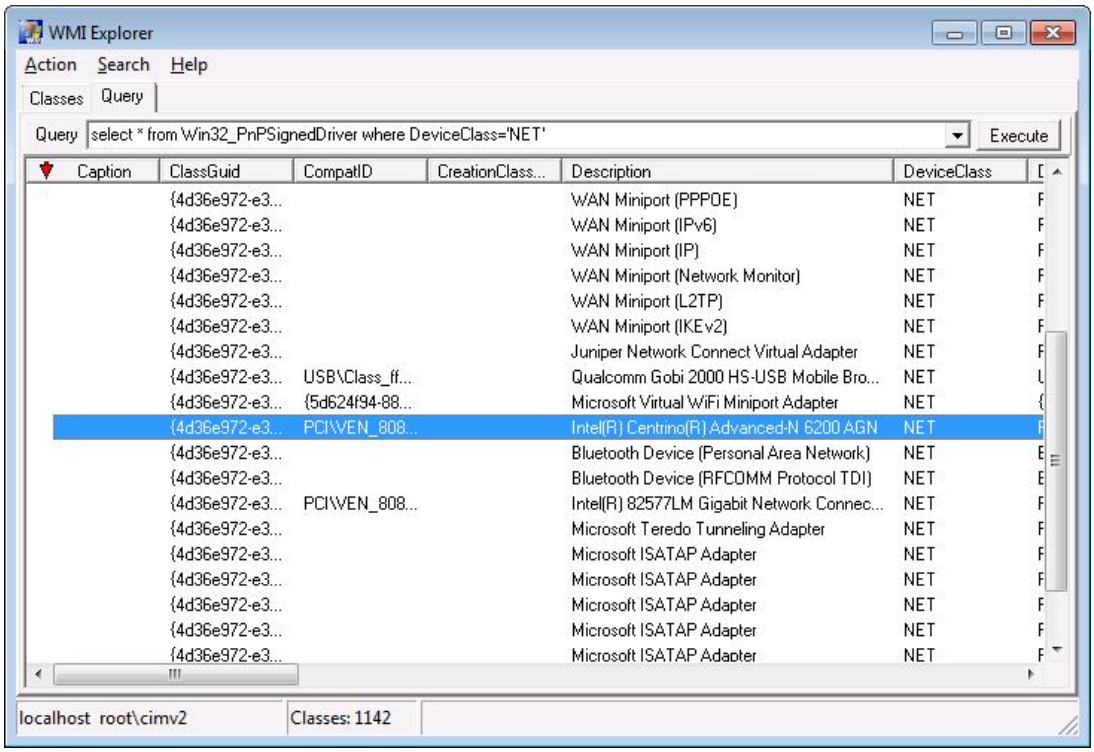
Sovelluksessa sarjaporttiyhteyden hallintaan käytettiin SerialPort-luokan objektia ja sarjaportin lukuun SerialDataReceived-metodia, joka suoritetaan aina, kun portti vastaanottaa tietoa GPS-laitteelta. NMEA-viestistä suodatettiin pois kaikki muu informaatio lukuun ottamatta $GPGGA-merkeillä alkavia rivejä. $GPGGA-viestistä poimittiin N- ja E-koordinaatit sijaintitiedon piirtoa varten.

Kartan näyttämiseen sovelluksessa käytettiin PictureBox-komponenttia. PictureBox eli kuvalaatikko on kuvien esittämiseen käytettävä objekti. Laitteen sijainti piirrettiin ympyränä kartan päälle. Piirtäminen toteutettiin Graphics-luokan DrawCircle-metodilla. /11/

Valmis testisovellus asennettiin testikoneelle ja sijaintitiedon tarkkuus testattiin ajamalla autolla ympäri tehdasaluetta. Sijaintitieto näytti auton sijainnin kaikissa tilanteissa parin metrin tarkkuudella, joka oli luotavalle järjestelmälle riittävä tarkkuus. Testisovellusta varten luotuja metodeja hyödynnettiin myöhemmin luotavassa tiedon analysointisovelluksessa sekä tiedonkeruusovelluksessa.

5.3 WMI-rajapinta

Luotettavien sijaintitietojen varmistuttua oli aika kartoittaa lähteet langattoman verkon tiedoille. Hyvä vaihtoehto olisi Microsoftin WMI-rajapinta. WMI eli Windows Management Instrumentation on standartoitu teknologia tietokoneenhallintaan liittyvän informaation käyttämiseen. /12/



KUVA 7. WMI Explorer -sovellus

WMI-rajapinnassa on kattavasti saatavilla tiedot kaikista tietokoneeseen asennetuista laitteista sekä niiden ajureista. Rajapinnan tietojen tarkastelua varten asennettiin kehityskoneelle ilmainen WMI Explorer -sovellus. Tarkastelu antoi hyviä tuloksia ja kehityskoneelta löytyivät kattavat tiedot niin langattomasta verkkokortista, sen merkistä, mallista, ajureista kuin verkkoyhteyteen liittyvistä tiedoistakin. Hyvien tuloksien pohjalta WMI Explorer –sovellus asennettiin myös varsinaiseen testikoneeseen, jossa kuitenkin ilmeni, että testikoneen langaton verkkokortti ei tukenut rajapintaa verkkoyhteyteen liittyvien tietojen osalta. Vaihtoehtona oli langattoman verkkokortin vaihtaminen rajapintaa tukevaksi kortiksi, mutta tämä olisi lisännyt kustannuksia mittavasti eikä järjestelmän toiminnan jatkuvuus tulevaisuudessa olisi ollut kovinkaan varmaa.

Tarkat laitekohtaiset tiedot kuten verkkokortin merkki, malli, ajuriversio ja muut tiedot löytyivät kuitenkin jokaisesta koneesta WMI–rajapinnasta. Niinpä aiempaan testisovellukseen lisättiin metodi *GetDeviceInfo()*, jolla testattiin tietojen haku ohjelmallisesti. Lopputuloksena saatiin valmis toiminnallisuus laitekohtaisten tietojen saamiseksi. Tiedot ovat hyödynnettävissä myöhemmin luotavaa laiterekisteriä varten.

```
WqlObjectQuery wqlQuery = new WqlObjectQuery(@"SELECT DriverDate,DriverVersion FROM Win32_PnpSignedDriver");
ManagementObjectSearcher searcher = new ManagementObjectSearcher(wqlQuery);
foreach (ManagementObject obj in searcher.Get())
{
    Console.WriteLine(obj["DriverDate"]);
    Console.WriteLine(obj["DriverVersion"]);
}
```

KUVA 8. Ohjelmallinen tiedonhaku WMI-rajapinnasta

Käyttötarkoitus langattomaan verkkoyhteyteen liittyvien tietojen hakuun hankaloituivat entisestään, sillä haut rajapinnasta kestivät noin sekunnin luokkaa. Tietojen päivittymiseltä edellytettiin noin 100 millisekunnin aikaa, jotta tiedot yhteydestä ja ajoneuvon sijainnista täsmäävät sillä hetkellä, kun tiedonkeruusovellus tallentaa tiedot tietokantaan.

5.4 NativeWifi

WMI–rajapinnan osoittautuessa huonoksi vaihtoehdoksi, oli tiedot langattomasta verkosta saatava muualta. Tietolähdettä etsittiin Microsoft Windows SDK -paketista, joka on erilaisia työkaluja ja rajapintoja sisältävä paketti sovellusten luomiseen Windows-käyttöjärjestelmille. /13/

Paketista löytyi NativeWifi–rajapinta (wlanapi.dll), joka antaa ohjelmoijalle mahdollisuuden hallita langattomia verkkoyhteyksiä. Rajapinnan tietueista löytyi tiedot signaalin laadusta ja signaalin voimakkuudesta eli juuri niitä tietoja, mitä järjestelmää varten tarvittaisiin. Rajapinta oli kuitenkin yhteensopiva vain C/C++-ohjelmointikielillä ja käyttöjärjestelmävaatimus oli Windows Vista- sekä sitä uudemmat Windows-käyttöjärjestelmät. Yhteensopimattomuutta Windows XP -käyttöjärjestelmän kanssa ei pidetty ongelmana, sillä Ruukin Raahen tehtaalla on käynnissä muutos, jossa kaikki tuotantokoneet tullaan muuttamaan Windows 7 -käyttöjärjestelmäpohjaisiksi. /14/

Koska järjestelmän ohjelmointikielenä käytettiin C#-kieltä, olisi NativeWifi-komponentin ja luotavan tiedonkeruusovelluksen väliin täytynyt ohjelmoida ns. ”wrapper”-luokka, joka käytännössä toimisi rajapintana rajapinnalle. /15/

Wrapperin ohjelmointiin olisi kulunut liikaa aikaa, joten päätettiin selvittää olisiko jossain saatavilla valmiina vastaavaa luokkaa/rajapintaa. Selvityksessä löytyi vain yksi avoimen lähdekoodin C#-kielelle tehty rajapinta Managed Wifi API. Tarkemmin ottaen Managed Wifi API on luokkakirjasto langattomien verkkokorttien hallintaan. /16/

Luokkakirjaston testattiin testisovelluksessa, johon lisättiin viittaus löydettyyn luokkakirjastoon sekä NativeWifi (wlanapi.dll) -komponenttiin. Luokan käyttö osoittautui yllättävän vaikeaksi, sillä dokumentointi oli todella puutteellinen. Luokkakirjaston internetsivuilta löytyi esimerkki, jota muokaten päädyttiin saamaan halutut tiedot langattomasta verkkoyhteydestä.

6 TIETOKANTA

Kerätyn tiedon taltiointiin ja myöhempää käsittelyä varten tarvittiin tietovarasto eli tietokanta. Tietokanta on jäsennetty kokoelma erilaista tietoa, jotka tallennetaan tietokoneelle. Järjestelmää varten käytettävissä oli *Microsoft SQL Server 2008* –tietokanta. Tietokannan hallintaan käytettiin *Microsoft SQL Server* -asennuksen mukana tulevaa SQL Server Management Studiota, joka on integroitu ympäristö kaikkien SQL Server -komponenttien käyttöön, muokkaukseen ja hallintaan. /17,18/

Järjestelmää varten palvelimelle luotiin uusi tietokanta WLANCDLS. Kerättävää tietoa varten luotiin tietokantaan taulu Wlan_data, johon lisättiin tarvittavat kentät eri tietoja varten. Tietoa relaatiotietokannassa säilytetään tauluissa. Koko tietokanta rakentuu taulujen ympärille, jotka sisältävät sarakkeita (kenttiä) sekä rivejä. Sarakkeet esittävät lukuisia eri tietotyyppejä taulussa. Jokainen rivi taulussa vastaa yhtä tallennusta. /19/

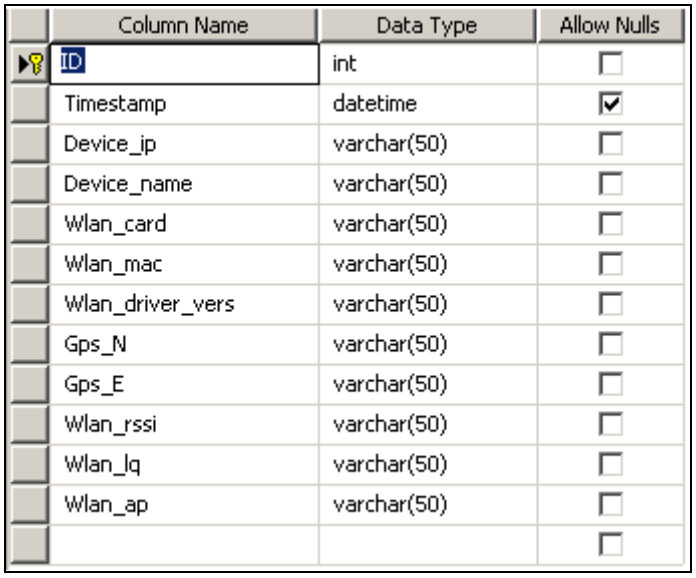
Tässä vaiheessa työtä kenttien tietotyyppien optimaalisuuteen ei kiinnitetty suurta huomiota, vaan pyrittiin luomaan kentistä sovellusteknisesti helppokäyttöisimmät, jonka vuoksi lähes kaikki kentät asetettiin VARCHAR(50)-tyyppisiksi. Tietokannan optimoinnista on kerrottu enemmän luvussa 7.4.

Tiedon tallettajalle eli tiedonkeruusovellukselle luotiin käyttäjätunnus *wclient*, jolle asetettiin kirjoitusoikeudet WLANCDLS-kantaan. Myöhemmin luotavaa tiedon analysointisovellusta varten luotiin lisäksi myös käyttäjätunnus *wuser*, jolle asetettiin lukuoikeudet.

Maksimitiedontalletusajaksi Wlan_data -tauluun asetettiin yksi vuosi, kentän Timestamp mukaan. Tallennustilan tarvetta on arvioitu luvussa 7.4.

Kuvasta 9 voidaan havaita kaikki tiedot, mitä tiedonkeruusovellus tallentaa. Seuraavassa luettelossa on kerrottu mitä eri sarakkeiden lyhenteet tarkoittavat.

- ID = juokseva tunniste, jolla varmistetaan jokaisen rivin yksilöllisyys
- Timestamp = aikaleima
- Device_ip = tuotantokoneen IP-osoite
- Device_name = tuotantokoneen nimi
- Wlan_card = langattoman verkkokortin nimi ja malli
- Wlan_mac = langattoman verkkokortin MAC-osoite
- Wlan_driver_vers = langattoman verkkokortin ajurin versio
- Gps_N = N-koordinaatti, tallennushetken sijainti
- Gps_E = E-koordinaatti, tallennushetken sijainti
- Wlan_rssi = Received Signal Strength Indication, signaalin voimakkuus (dB)
- Wlan_lq = link (signal) quality, signaalin laatu (%)
- Wlan_ap = tukiaseman MAC-osoite.



Column Name	Data Type	Allow Nulls
ID	int	☐
Timestamp	datetime	☑
Device_ip	varchar(50)	☐
Device_name	varchar(50)	☐
Wlan_card	varchar(50)	☐
Wlan_mac	varchar(50)	☐
Wlan_driver_vers	varchar(50)	☐
Gps_N	varchar(50)	☐
Gps_E	varchar(50)	☐
Wlan_rssi	varchar(50)	☐
Wlan_lq	varchar(50)	☐
Wlan_ap	varchar(50)	☐
		☐

KUVA 9. Wlan_data –taulu SQL Server Management Studiossa

Tietokantatauluun kirjoittaminen ja sen sisällön lukeminen testattiin ohjelmallisesti. Tietokantayhteyden muodostamisessa käytettiin kehitysympäristön valmiiden luokkien SqlDataAdapter ja SqlConnection objekteja, joilla voidaan suorittaa haluttuja tietokantaoperaatioita.

7 TIEDONKERUUSOVELLUS

Tässä kappaleessa on kerrottu tiedonkeruusovelluksen luomisesta vaiheittain. Tiedonkeruusovellus on langatonta verkkoa käyttävälle ajoneuvossa olevalle tietokoneelle asennettava sovellus. Se kerää ajoneuvon sijaintitiedot, langattomaan verkkoyhteyteen liittyvät tiedot ja tietokoneen laitetiedot. Kerätyt tiedot sovellus lähettää tietyin aikasyklein tietokantapalvelimelle.

Sovellus käynnistyy automaattisesti samalla, kun tietokone käynnistetään ja sovelluksen ikoni näkyy järjestelmäpalkissa, yleensä oikealla alanurkassa. Ikonia klikkaamalla käyttäjälle avautuu sovelluksen graafinen käyttöliittymä, josta hän näkee sovelluksen keräämät tiedot. Vikatilanteissa kuljettaja näkee vaivatta esimerkiksi koneen IP-osoitteen, jonka hän voi sisällyttää vikailmoitukseensa.

7.1 Käyttöliittymä

Tiedonkeruusovelluksen ohjelmointi aloitettiin luomalla kehitysympäristössä uusi Windows Forms -projekti. Windows Formsia käytetään graaafisten käyttöliittymien luomiseen eri sovelluksille. Form eli ikkuna on graafinen elementti, joka näkyy tietokoneen työpöydällä. Projektille eli sovellukselle annettiin nimi *WlanDataLogger*. Graafinen käyttöliittymä sovellukseen luotiin Windows Form Designerillä. /20/

Langattoman verkkoyhteyden tietojen ja ajoneuvon sijaintitietojen näyttämiseen käyttöliittymällä käytettiin TextBox-luokan objekteja. Textbox eli tekstikenttä mahdollistaa tekstin syöttämisen graafisen käyttöliittymän kautta. Koska tietoa pyrittiin vain näyttämään käyttöliittymällä, tekstikenttiä käytettiin päinvastoin niiden varsinaista käyttötarkoitusta eli kenttiin asetettiin haluttu teksti Textbox.Text -ominaisuudella. /21/

Myöhemmin luotavien metodien toiminnan seuraamiseen ja mahdollista vianhakua helpottamaan luotiin monitorointi-ikkuna Multiline TextBox -objektilla. Käyttöliittymästä on mahdollista vain seurata sovelluksen toimintaa, joten kaikkien käytettyjen objektien Read Only -ominaisuus asetettiin todeksi, TRUE (LIITE 4).

7.2 Metodit

Sovellukseen luotiin omat metodit eri toiminnallisuuksille:

- *GetDeviceInfo()*
- *GetWlanInfo()*
- *SqlWrite()*
- *RestoreInfo()*

GetDeviceInfo() hakee laitteen tiedot WMI–rajapinnasta. Haettava tieto ei muutu sovelluksen ajon aikana, joten metodi suoritetaan vain sovelluksen käynnistyessä. Saadut tiedot talletetaan muuttujiin.

```
try
{
    using (SqlConnection connection = new SqlConnection(connectionString))
    using (SqlCommand command = connection.CreateCommand())
    {
        command.CommandText = @"INSERT INTO wlan_data (ID, Timestamp, Gps_N, Gps_E, Wlan_ap, Wlan_lq, Wlan_rssi)
                                VALUES (@ID, GetDate(), @Gps_N, @Gps_E, @Wlan_ap, @Wlan_lq, @Wlan_rssi)";
        command.Parameters.AddWithValue("@ID", id);
        command.Parameters.AddWithValue("@Gps_N", textNorth.Text);
        command.Parameters.AddWithValue("@Gps_E", textEast.Text);
        command.Parameters.AddWithValue("@Wlan_ap", textWlanAP.Text);
        command.Parameters.AddWithValue("@Wlan_lq", textLinkQ.Text);
        command.Parameters.AddWithValue("@Wlan_rssi", textRSSI.Text);
        connection.Open();
        command.ExecuteNonQuery();
        textMonitor.AppendText("System: Measurements stored to database" + System.Environment.NewLine);
        this.RestoreData();
    }
}
catch (SqlException ex)
{
    Console.WriteLine(ex.Message);
    this.StoreData();
}
```

KUVA 10. SqlWrite()-metodin tietokantaoperaatio

GetWlanInfo() hakee langattomaan verkkoyhteyteen liittyvät tiedot Managed Wifi API –luokkakirjaston tietueista. Tiedot päivitetään käyttöliittymän TextBox–objekteihin.

SqlWrite() tallettaa kerätyt tiedot tietokantaan. Mikäli tietokantaoperaatio epäonnistuu, esimerkiksi verkkoyhteyden käydessä poikki, tiedot tallennetaan rengaspuskureihin.

RestoreInfo()-metodi tallettaa rengaspuskuriin varastoidut tiedot eli ne tiedot, joita *SaveInfo()*-metodi ei onnistunut tallettamaan. *RestoreInfo()*-metodi suoritetaan onnistuneen *SaveInfo()*-metodin suorituksen jälkeen, mikäli rengaspuskureissa on sisältöä. Jälkikäteen kirjoitetun tiedon erottaa siitä, ettei tietokantarivillä ole ollenkaan aikaleimaa Timestamp. Niiden rivien GPS-koordinaateista, joiden Timestamp-kenttä on tyhjä, voidaan todeta sijainti sille paikalla, missä langaton verkkoyhteys ei toiminut. Tuotantokoneiden kellonaikaa ei voida käyttää epäonnistuneiden tallennuksien ajanhetken tallennuksessa, koska ne poikkeavat suurella todennäköisyydellä tietokannan kellonajasta.

Sovelluksen tarkempi toiminta käy ilmi luvun 7.5 vuorovaikutuskaaviosta. Luotu sovellus on täysin automaattinen, eikä se vaadi tietokoneen eli ajoneuvon kuljettajalta käyttötoimia.

7.3 Ajastimet

Ajastin (Timer) mahdollistaa toistuvien tapahtumien luomisen Windows-sovelluksissa. Metodien *GetWlanInfo()* ja *SqlWrite()* suoritusta ohjataan ajastinluokan objekteilla. Jokaiselle metodille luotiin oma ajastin. Ajastimet ovat perusta sovelluksen automaattiselle toiminnalle. /22/

7.4 Tietokannan optimointi

Tiedonkeruusovellus tallettaa kaikki tiedot tietokantaan viiden sekunnin välein, myös ne tiedot, jotka eivät sovelluksen ajon aikana muutu. Jos tiedon talletus tehdään viiden sekunnin välein yhteensä neljästäkymmenestä ajoneuvosta, tietoa kertyy teoriassa vuorokauden aikana seuraavasti:

((24 * 60 * 60) / 5) * 40 = 691200 * 161 = 111283200 tavua eli ~106 megatavua / vrk.

TAULUKKO 1. Wlan_data ennen optimointia

Wlan_data -taulu				
Kenttä	Tietotyyppi	Esimerkki sisältö	Pituus merkkeinä	Tilantarve tavuina
ID	int	10	2	4
Timestamp	datetime	2012-11-15 14:50:00.000	16	8
Device_ip	varchar(50)	123.123.123.123	15	17
Device_name	varchar(50)	DTFIRAAP0111	12	14
Wlan_card	varchar(50)	Atheros AR928X Wireless Network Adapter	39	41
Wlan_mac	varchar(50)	BC:AE:C5:21:AD:BE	17	19
Wlan_driver_vers	varchar(50)	8.0.0.372	9	11
Gps_N	varchar(50)	12341234	8	10
Gps_E	varchar(50)	12341234	8	10
Wlan_rssi	varchar(50)	99	2	4
Wlan_lq	varchar(50)	99	2	4
Wlan_ap	varchar(50)	BC:AE:C5:21:AD:BF	17	19
			Yhteensä:	161

Täytyy huomioida, että tämä laskenta on teoreettinen ja se ei ota huomioon tietokannan sisältämiä sisäisiä algoritmeja tilantarpeen minimoimiseksi. Hyvin todennäköistä on, että näin isoon yksittäiseen tauluun kohdistuvat tietokantaoperaatiot olisivat tulevaisuudessa mittaustiedon kertyessä hyvin hitaita. Hitauden välttämiseksi päätettiin kerättävät tiedot jakaa kahteen eri tauluun. Yhteen tauluun tulisi muuttumaton tieto ja toiseen muuttuva tieto.

Muuttumattomalle tiedolle luotiin taulu *Device_data* (taulukko 3) ja muuttuvalle tiedolle taulu *Wlan_data* (taulukko 2). Tauluja yhdistävä kenttänä käytettiin kenttää ID. Yhdistäminen tapahtui käyttämällä tietokanta-avaimia.

TAULUKKO 2. Muuttuva tieto

Wlan_data -taulu				
Kenttä	Tietotyyppi	Esimerkki sisältö	Pituus merkkeinä	Tilantarve tavuina
ID	int	10	2	4
Timestamp	datetime	2012-11-15 14:50:00.000	16	8
Gps_N	int	12341234	8	4
Gps_E	int	12341234	8	4
Wlan_rssi	int	99	2	4
Wlan_lq	int	99	2	4
Wlan_ap	varchar(50)	BC:AE:C5:21:AD:BF	17	19
			Yhteensä:	47

TAULUKKO 3. Muuttumaton tieto

Device_data -taulu				
Kenttä	Tietotyyppi	Esimerkki sisältö	Pituus merkkeinä	Tilantarve tavuina
ID	int	10	2	4
Timestamp	datetime	2012-11-15 14:50:00.000	16	8
Device_ip	varchar(50)	123.123.123.123	15	17
Device_name	varchar(50)	DTFIRAAP0111	12	14
Wlan_card	varchar(50)	Atheros AR928X Wireless Network Adapter	39	41
Wlan_mac	varchar(50)	BC:AE:C5:21:AD:BE	17	19
Wlan_driver_vers	varchar(50)	8.0.0.372	9	11
			Yhteensä:	114

Tietotyypit ja niiden tallennustilan tarve on esitetty taulukoissa 2 ja 3 optimoinnin jälkeen. Mikäli laitteiston tiedot pysyvät samana, voidaan todeta, että teoriassa optimoinnilla säästettiin levytilaa ensimmäisen rivin tallennuksessa 0 %, mutta jo sadan tallennetun rivin jälkeen ~30 %.

7.5 Vuorovaikutuskaavio

Vuorovaikutuskaaviossa on esitetty sovelluksen toiminta vaiheittain. Huomioitavaa on, että sovellusta ei pysäytetä lainkaan, vaan se on käynnissä aina niin kauan kuin itse tuotantokonekin.

KUVA 11. Tiedonkeruusovelluksen vuorovaikutuskaavio

8 TIEDON ANALYSOINTISOVELLUS

Kerätyn tiedon tulkintaan tarvittiin erillinen sovellus. Sovelluksen on tarkoitus vastata järjestelmän kannalta olennaisiin kysymyksiin eli, missä ja miten langaton verkkoyhteys toimii. Sovellusta tulevat käyttämään verkon ylläpitäjät, ajoneuvourakoitsijan työnjohto sekä mahdollisesti muut logistiikan parissa työskentelevät henkilöt. Verkkoa ylläpitäviä henkilöitä kiinnostaa verkon toiminta, kun taas muita käyttäjiä pelkästään ajoneuvojen sijainti. Niinpä sovelluksen toiminta päätettiin jakaa eri toimintoihin, toimintamoodeihin, joita käyttäjä voi halutessaan vaihtaa.

8.1 Toimintamoodit

Verkkoa ylläpitäville henkilöille päätettiin luoda kaksi eri toimintamoodia. Ensimmäinen toimintamoodi olisi ajoneuvojen seuranta, jossa näkyisi sijainnin lisäksi tiedot sen hetkisestä verkkoyhteydestä (LIITE 3). Toisessa toimintamoodissa käyttäjät voisivat katsoa mittaustulokset tukiasemakohtaisesti, joista voi havaita tukiasemien peittoalueet.

Muille käyttäjille eli ajoneuvourakoitsijan työnjohdolle ja logistiikan parissa työskenteleville henkilöille luotaisiin pelkkä ajoneuvojen sijaintitietojen seuranta.

Tiukan aikataulun vuoksi insinöörityön suhteen muita toimintamoodeja ei päätetty luoda. Tätä vajavaisuutta täydentämään päätettiin kuitenkin lisätä kenttä, johon voi syöttää vapaavalintaisen SQL-lausekkeen. Lausekkeella löytyneet tiedot (tietokantarivit) sovellus esittää graafisesti kartan päällä.

8.2 Käyttöliittymä

Tiedon analysointisovelluksen ohjelmointi aloitettiin luomalla kehitysympäristössä uusi Windows Forms -projekti. Projektille eli sovellukselle annettiin nimi *WlanAnalyzer*. Graafinen käyttöliittymä sovellukseen luotiin Windows Form Designerilla.

Kerätty tieto pyritään esittämään mahdollisimman pitkälle graafisesti. Sovelluksen päänäkymäksi eli kartaksi asetettiin näkymä Google Earth –sovelluksesta otetusta kuvakaappauksesta. Karttakuvan näyttämiseen käytettiin PictureBox-luokan objektia.

Alun perin päänäkymänä oli tarkoitus käyttää layout-piirrosta tehdasalueesta mutta sen käytöstä luovuttiin sen epäselvyyden vuoksi. Sijaintitietojen piirtäminen layout-piirroksen päälle ei myöskään olisi ollut itsestään selvää, sillä tietokoneen näytöllä näkyvä piirros ei osoita pohjoiseen niin kuin kuvakaappaus. Käytännössä tämä tarkoittaa sitä, etteivät GPS-koordinaatit ja näytön X- ja Y-koordinaatit ole samassa suhteessa toisiinsa. Piirtämisessä olisi otettava huomioon piirroksen suunta asteina pohjoiseen päin sekä kartan mittakaava suhteessa GPS-koordinaatteihin.

Toimintamoodien valintaa varten käytettiin ComboBox-luokan objektia. ComboBox eli pudotusvalikko on tekstikentän ja listan yhdistelmä, joka mahdollistaa vain yhden elementin valinnan kerralla. /23/

Toimintamoodikohtaisten lisävalintojen luettelointiin lisättiin myös ListBox eli luettelo, josta on mahdollista valita esimerkiksi yksittäinen ajoneuvo seurantaa varten. Luettelo sijoitettiin pudotusvalikon alapuolelle. /24/

Päävalikko sovellukseen saatiin menuStrip-luokan objektilla. Päävalikkoon lisättiin alavalikot File, josta sovellus voidaan sulkea. Valikkoon voidaan myöhemmin lisätä alavalikko esimerkiksi laiterekisterin käynnistämiseen.

8.3 Toiminnallisuuden suunnittelu

Ennen varsinaista ohjelmointityön aloittamista sovelluksen toiminnallisuus pyrittiin suunnittelemaan mahdollisimman joustavaksi eri toimintamoodien kannalta. Eri moodeilla, joita hyvin todennäköisesti ajansaatossa lisätään, voidaan tehdä täysin toisistaan poikkeavia tiedon esityksiä.

Suunnittelu aloitettiin miettimällä, mitkä ovat sellaisia toimintoja, jotka joudutaan suorittamaan aina toimintamoodista riippumatta. Sellaisiksi toiminnoksi paljastui tiedonhaku tietokannasta ja tiedon esittäminen sovelluksessa. Toimintamoodikohtaisia toimintoja taas olisivat eri seikat, kuten mitä tietoa tietokannasta haetaan ja miten tieto tullaan esittämään. Suunnittelun pohjalta päätettiin luoda omat metodit määrittämään, mitä tietoa tietokannasta haetaan, suorittamaan tietokantaoperaatiot ja esittämään haetut tiedot graafisesti.

8.4 Metodit

Sovelluksen toiminnallisuuden toteuttamiseksi lisättiin sovelluksen pääluokkaan MainForm useita eri metodeja:

- *SetupApp()*
- *UpdateListBox()*
- *UpdateQuery()*
- *DoQuery()*
- *PaintResults(DataRow[] foundRows, int row_count)*
- *Gps_n_to_y(int gps_n)*
- *Gps_e_to_x(int gps_e).*

SetupApp() alustaa sovelluksen lisäämällä eri toimintamoodit valintalistalle. *PaintResults()*-metodia varten lasketaan indeksit *Y_index* ja *X_index* kaavoilla 1 ja 2.

$$Y\_index = \frac{Kartan\ korkeus}{GPS_N - GPS_S}$$, missä *(KAAVA 1)*

GPS_N = N-koordinaatti kartan pohjoisimmasta kohdasta

GPS_S = N-koordinaatti kartan eteläisimmästä kohdasta

$$X\_index = \frac{Kartan\ leveys}{GPS_E - GPS_W}$$, missä *(KAAVA 2)*

GPS_E = E-koordinaatti kartan itäisimmästä kohdasta

GPS_W = E-koordinaatti kartan läntisimmästä kohdasta

UpdateListBox() päivittää listanäkymän hakemalla tiedot listalle tietokannasta valitun toimintamoodin mukaan. Mikäli tietokantaoperaatio epäonnistuu, esitetään virheilmoitus ja sovellus suljetaan.

UpdateQuery() päivittää SQL–lauseen muuttujaan, jota käytetään halutun tiedon hakemiseen tietokannasta. SQL-lauseen sisältö riippuu valitusta toimintamoodista.

DoQuery() suorittaa *UpdateQuery()*-metodissa määritetyn tietokantaoperaation. Tietokannasta haetut rivit asetetaan taulukkomuuttujaan *DataRow[] foundRows*, jotka lähetetään edelleen *PaintResults()*-metodille.

PaintResults() piirtää graafisesti mittatiedot valitun toimintamoodin mukaisesti. Taulukkomuuttujasta poimitaan N- ja E-koordinaatit, jotka muutetaan *Gps_n_to_y(int gps_n)* ja *Gps_e_to_x(int gps_e)* metodeilla X- ja Y-koordinaateiksi. Piirtäminen toteutettiin Graphics-luokan objektin DrawLine- ja DrawEllipse-metodeilla, joille välitettiin X- ja Y-koordinaatit.

Gps_n_to_y(int gps_n) vastaanottaa kokonaislukuna N–koordinaatin ja muuttaa sen karttapohjan Y-koordinaatiksi (Kaava 3).

$$Y = (GPS_N - N) * Y\_index, \text{ missä} \qquad \textit{(KAAVA 3)}$$

GPS_N = N-koordinaatti kartan pohjoisimmasta kohdasta
Y = lopullinen sijainti Y-akselilla

Gps_e_to_x(int gps_e) vastaanottaa kokonaislukuna E–koordinaatin ja muuttaa sen karttapohjan X-koordinaatiksi (Kaava 4).

$$X = (GPS_E - E) * X\_index, \text{ missä} \qquad \textit{(KAAVA 4)}$$

GPS_E = E-koordinaatti kartan itäisimmästä kohdasta
X = lopullinen sijainti X-akselilla

8.5 Tietokantaoperaatiot

Jokaisella eri toimintamoodilla suoritetaan erilainen tietokantaoperaatio, tietokantahaku. Tietoa tallennettaessa useampaan tauluun, kuten luvussa tietokannan optimointi pyrittiin tekemään, ovat haittavaikutuksina monimutkaisemmat tietokantahaut.

Esimerkiksi viimeisimmän ajoneuvon sijaintitiedon hakeminen yksittäisellä hakulauseella kahta taulua yhdistämällä osoittautui erittäin työlääksi ja jäi lopulta tekemättä. Hakuja helpottamaan päätettiin luoda näkymiä, jotka ovat virtuaalisia tauluja, mitkä eivät itsessään sisällä lainkaan tietoa. Käytännössä näkymät ovat tietokantahakuja, jotka ovat tallennettuna tietokantaan. Ne voivat sisältää tietoa yhdestä tai useammasta eri taulusta sekä niihin voidaan kohdistaa tietokantahakuja samalla tavalla kuin normaaleihin tauluihinkin. /25/

Tietokantaan luotiin näkymät *Latest_ID_by_Device_name* ja *Latest_positions*. Näkymästä *Latest_ID_by_Device_name* voidaan hakea laitetunnuksen mukaan sitä vastaava viimeisin ID, jota voidaan käyttää hakemaan tietyn laitteen keräämät tiedot tai ajoneuvojen viimeisimmät sijainnit. Näkymän SQL-lauseet on esitetty kuvissa 12 ja 13.

```
SELECT     MAX(ID) AS ID, Device_name, MAX(Timestamp) AS Timestamp
FROM       dbo.device_data AS d
GROUP BY Device_name
```

KUVA 12. Näkymän Latest_ID_by_Device_name SQL-lause

```
SELECT     w.Gps_N, w.Gps_E, w.Timestamp, l.ID
FROM       dbo.wlan_data AS w INNER JOIN
             dbo.Latest_ID_by_Device_name AS l ON w.ID = l.ID
WHERE      (w.Timestamp =
                (SELECT     MAX(Timestamp) AS Expr1
                 FROM       dbo.wlan_data
                 WHERE      (w.ID = ID)))
```

KUVA 13. Näkymän Latest_positions SQL-lause

Ajoneuvojen sijaintien seurantaa varten luotu *Latest_positions* -näkymä kertoo jokaisen ajoneuvon viimeisimmän sijainnin. On huomioitava, että näkymä hakee viimeisimmän sijainnin vaikka viikon takaa, joten näkymää ei suoraan voida käyttää reaaliaikaiseen sijaintien seurantaan, vaan Timestamp-kenttään on kohdistettava lisäehto.

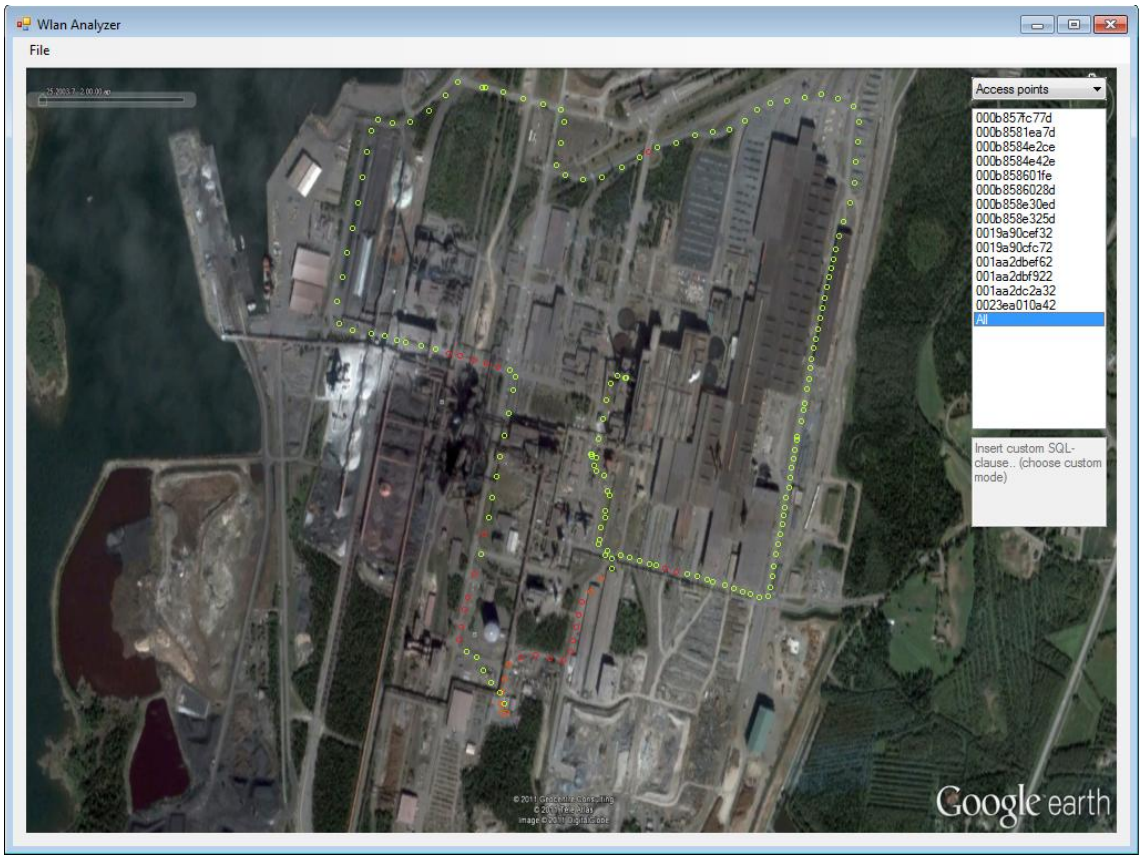
8.6 Laiterekisteri

Laiterekisteri tai paremminkin laiterekisterin käyttöliittymä päätettiin jättää toistaiseksi tekemättä, koska Ruukin Raahen Windows 7 -päivitysprojekti toteutetaan vasta myöhemmin kuluvaa vuotta tai vasta seuraavana vuonna. Kuten jo aiemmin todettiin, on tämä järjestelmä langatonta verkkoyhteyttä koskevien tietojen osalta Windows XP -yhteensopimaton. Toisena syynä voidaan pitää insinöörityön kirjallisen osion aikataulua.

Jokainen tuotantokone, johon tiedonkeruusovellus asennetaan, alkaa kuitenkin kerätä laitetiedot tietokantaan. Ainoa tieto mikä jää manuaalisesti syötettäväksi on se, mihin ajoneuvoon kyseinen laitetunnusta vastaava tuotantokone on asennettu.

9 TESTAUS

Koska kehityskoneessa oli mahdollista käyttää samaa GPS-paikanninta kuin tuotantokoneessa, pystyttiin järjestelmän eri sovellusten toiminta testaamaan jo siinä vaiheessa, kun niitä oltiin ohjelmoimassa. Näin onnistuttiin välttymään 99-prosenttisesti sovellusvirheiltä.



KUVA 14. Mittaustuloksia tiedon analysointisovelluksessa ajoneuvosta, jolla ajettiin tehdasalueen ympäri

Järjestelmän lopullista testausta varten yhden tuotantokoneen käyttöjärjestelmä vaihdettiin Windows 7 –pohjaiseksi ja koneelle asennettiin tiedonkeruusovellus. Testauksessa huomattiin, että langattoman verkkoyhteyden käydessä poikki kirjoitus tietokantaan keskeyttää hetkellisesti koko muun sovelluksen toiminnan. Syyksi tähän paljastui, ettei tietokantaoperaatiolle oltu annettu maksimisuoritusaikaa. Kun muutimme maksimisuoritusajan yhdeksi sekunniksi, sovellus toimi jälleen normaalisti.

Mitattuja tietoja verrattiin langattomien verkkokorttien mukana tuleviin kortin valmistajien omiin diagnostiikkasovellusten antamiin mitta-arvoihin ja voitiin todeta, että mitatut tiedot vastasivat valmistajien sovellusten esittämiä lukuja.

10 JATKOKEHITYSMAHDOLLISUUDET

10.1 Helpotus laiterekisterin ylläpitoon

Tiedonkeruusovellus käynnistyessään kerää laitekohtaiset tiedot siltä koneelta, millä se käynnistetään. Sovellus vertaa keräämiään tietoja tietokannassa oleviin tietoihin. Mikäli tietokannasta löytyy vastaavat tiedot, sovellus tallettaa mittatiedot tietokannasta saamallaan identifioimalla tunnuksella ID. Mikäli taas tietokannasta ei löydy vastaavia tietoja, sovellus luo tietokantaan uuden rivin, ja käyttää sen rivin tunnusta identifiointiin.

Siinä vaiheessa kun tietokannasta ei löydy tietoja, sovellus voisi kysyä käyttäjältä, missä ajoneuvossa sovellus on käynnistetty. Käyttäjän syöttämän tiedon mukaan sovellus voisi kiinnittää laitetiedot suoraan tietylle ajoneuvolle, joten vältyttäisiin käsin tehtävältä kiinnittämiseltä. Useasti ensimmäisen käynnistyksen tekevä käyttäjä on tuotantokoneen korjannut huoltomies. Käynnistys tehdään varmistamaan, että kaikki sovellukset toimivat ajoneuvossa. Tällä toimella vältyttäisiin ylläpidollisilta jälkitöiltä.

10.2 Lisäyksiä kerättäviin tietoihin

Kerättäviin tietoihin voisi lisätä tiedon signaalin kohinasta, häiriötasosta sekä pakettien lähetyksen suorituskyvystä. Tietokantaan voisi myös tallentaa ilmastoa koskevat tiedot, kuten ulkoilman lämpötilan, ilmankosteuden ja tiedon sataako vettä tai lunta. Tiedot voisi päivittää erilliseen tietokantatauluun esimerkiksi puolen tunnin välein. Samalla kun haettaisiin mittatietoja langattomista yhteyksistä, voitaisiin viitata säätilatauluun, jolloin voitaisiin tutkia sään vaikutuksia verkkoyhteyksien toimintaan.

10.3 Ajastetut tietokantaoperaatiot

Mitä enemmän tietokantaan talletetaan mittatietoa, sitä hitaammaksi tietokantaoperaatiot menevät. Statistista tietoa kuten verkkokortti- tai tukiasemakohtaisia toiminnallisia keskiarvoja voisi laskea valmiiksi omiin tauluihin ajastettuna, esimerkiksi yöaikaan, jolloin järjestelmällä ei ole käyttäjiä. Tiedon analysointisovellus hakisi valmiiksi lasketut tiedot taulusta ja näyttäisi ne järjestelmän käyttäjille.

10.4 Sähköpostihälytykset

Palvelimelle voisi tehdä palvelinsovelluksen, joka analysoisi kerättyä tietoa. Mikäli esimerkiksi tietyssä ajoneuvossa verkkoyhteys pätkii tarpeeksi tietyn aikajakson sisällä, voisi sovellus lähettää verkon ylläpitäjille ja huoltomiehille siitä sähköpostitse hälytyksen.

Sähköpostihälytys voidaan toteuttaa asentamalla palvelimelle IIS-sähköpostitukipalvelut. Sovellus kirjoittaisi hälytyspostin palvelin C:\InetPub\mail\ -kansioon tekstitiedostona, jonka palvelu lähettää edelleen Ruukilla käytössä olevan Exchange-palvelimen kautta määritetyille henkilöille.

11 YHTEENVETO

Tässä työssä toteutettiin tiedonkeruujärjestelmä, jolla saadaan mitattua jatkuvatoimisesti erilaisiin ajoneuvoihin asennettujen tietokoneiden langattomaan verkkoyhteyteen liittyvät tiedot. Järjestelmä koostuu kolmesta eri osasta, tiedonkeruusovelluksesta, tietokannasta ja tiedon analysointisovelluksesta.

Järjestelmä ohjelmoitiin Microsoft Visual Studio 2010 –kehitysympäristössä C#-ohjelmointikielellä. Kerätyt mittatiedot tallennettiin Microsoft SQL Server 2008 –tietokantaan.

11.1 Käyttöönotto

Järjestelmän käyttöönotto suoritetaan Ruukin Raahen Windows 7 –päivitysprojektin yhteydessä vasta myöhemmin kuluvaa vuotta tai vasta seuraavana vuonna. Luvussa jatkokehitysmahdollisuudet mainittuja asioita on jo osittain toteutettu ja sovelluskehitystyötä jatketaan edelleen. Käyttöönoton jälkeen järjestelmällä pystytään joko korvaamaan tai helpottamaan luvussa 2 esitettyjä ongelmatilanteiden selvittämiseen käytettyjä menetelmiä.

11.2 Järjestelmä hankinnan ja ylläpidon tukena

Järjestelmän tietokantaan keräämiä mittatietoja voidaan tuoda erillisiin sovelluksiin, kuten Excel-taulukkolaskentaohjelmistoon. Mittatietoihin voidaan kohdistaa erilaisia hakuja, joilla saadaan selville esimerkiksi toiminnallisia keskiarvoja.

Esimerkki 1. Tietokannasta haetaan valmistajien A, B ja C verkkokorteilla tehdyt mittaustulokset. Tuloksista lasketaan keskiarvot, joita vertaillaan keskenään. Osoittautuu, että valmistajan B verkkokortti toimii parhaiten. Jatkossa voidaan siis hankkia vain valmistajan B verkkokortteja, koska ne toimivat parhaiten.

Esimerkki 2. Vertaillaan kahden eri ajoneuvon A ja B verkkoyhteyden toimintaa, joissa on täysin samanlainen laitteisto. Tietokannasta haetaan kummankin ajoneuvon mittatiedot, joista lasketaan keskiarvot. Osoittautuu, että ajoneuvon A verkkoyhteys on toiminut paljon paremmin. Tästä voidaan päätellä, että mahdollisesti ajoneuvossa B on huonompi tai rikkinäinen antenni tai antennin kaapeloinnissa on jotain vikaa.

Esimerkki 3. Vertaillaan eri ajuriversioiden A ja B toimintaa. Tietokannasta haetaan tiedot molemmilta ajuriversioilta. Mittatiedoista laskemalla selviää, että ajuriversio B toimii paremmin. Päätetään jatkossa käyttää ajuriversiota B.

On kuitenkin syytä muistaa, että aina kun mittatietoja vertaillaan, tulisi vertailukohteina käyttää samoissa olosuhteissa mitattuja tietoja. Esimerkiksi alue, jolla on kymmenen tukiasemaa hyvin lähekkäin palauttaa varmasti paremmat mittaustulokset (signaalin voimakkuus ja laatu) kuin alue, jolla on harvassa kaksi tukiasemaa. Alueet voidaan tarkistaa tiedon analysointisovelluksesta.

LÄHDELUETTELO

1: Rautaruukki Oyj: Layout-piirros, sisäinen verkko (LIITE 1).

2: Ruukki Metals Oy: Juha Kangasluoma, haastattelu

3: Isoworks Oy: Marko Männistö, haastattelu

4: Bai, Y., 2010. Practical Database Programming. S.29

5: Sempf, B., Sphar, C., Davis, S.R. 2010. C# All-in-one for Dummies. S.12

6. Roshan, P., Leary, J., 2004. 802.11 Wireless LAN Fundamentals. S.217

7. Roshan, P., Leary, J., 2004. 802.11 Wireless LAN Fundamentals. S.150

8. Garmin: About GPS, saatavissa:

http://www8.garmin.com/aboutGPS/

Tiedot haettu 25.11.2011

9. GlobalSat BU-353 GPS-receiver, manual.

10. NMEA: NMEA 0183 Standard:

http://www.nmea.org/content/nmea_standards/nmea_083_v_400.asp

Tiedot haettu 05.12.2011

11. Moghadampour, G., 2011. C# Windows- ja tietokantaohjelmointi. S. 199

12. Microsoft: About WMI:

http://msdn.microsoft.com/en-us/library/windows/desktop/aa384642%28v=vs.85%29.aspx

Tiedot haettu 20.01.2012

13. Microsoft: Overview of the Windows SDK:

http://msdn.microsoft.com/en-us/library/ms717422.aspx

Tiedot haettu 05.02.2012

14. Windows Software Development Kit, manual.

15. Nokia Oyj: Mauno Mattila, haastattelu

16. Codeplex: Managed Wifi API:

http://managedwifi.codeplex.com/

Tiedot haettu 10.11.2011

17. Bai, Y., 2010. Practical Database Programming. S.12

18. Microsoft: Using SQL Server Management Studio:

http://msdn.microsoft.com/en-us/library/ms174173.aspx

Tiedot haettu 07.02.2012

19. Bai, Y., 2010. Practical Database Programming. S.30

20. Deitel, H.M., Deitel, P.J., 2005. Visual C# How to Program. S.599

21. Moghadampour, G., 2011. C# Windows- ja tietokantaohjelmointi. S. 57

22. Moghadampour, G., 2011. C# Windows- ja tietokantaohjelmointi. S. 201

23. Moghadampour, G., 2011. C# Windows- ja tietokantaohjelmointi. S. 124

24. Moghadampour, G., 2011. C# Windows- ja tietokantaohjelmointi. S. 106

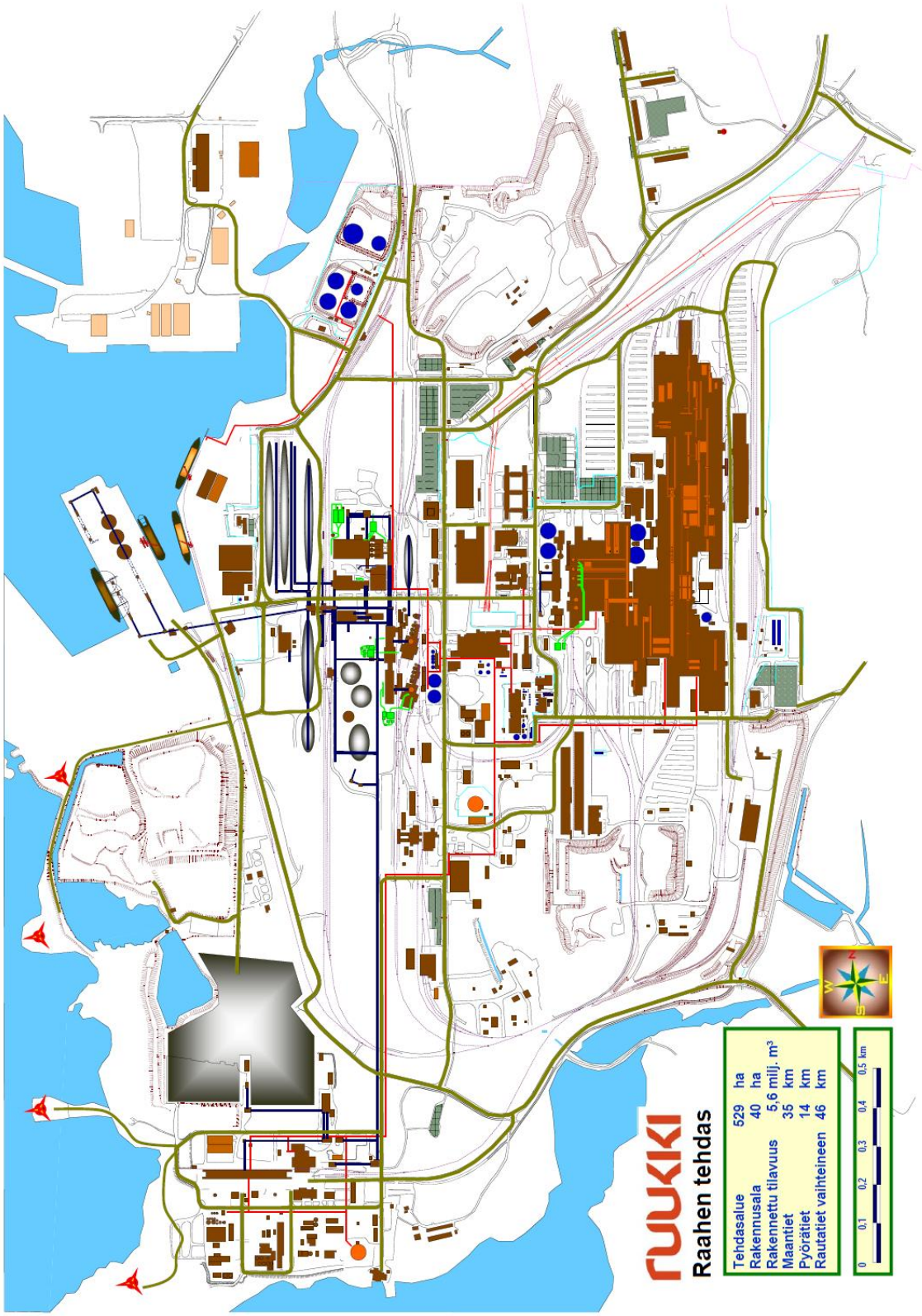
25. Bai, Y., 2010. Practical Database Programming. S.31

26. Microsoft: What Is QoS:
http://technet.microsoft.com/en-us/library/cc757120%28v=ws.10%29.aspx
Tiedot haettu 19.02.2012

27. Microsoft: Windows XP Professional Product Documentation:
http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/ping.mspx?mfr=true
Tiedot haettu 19.02.2012

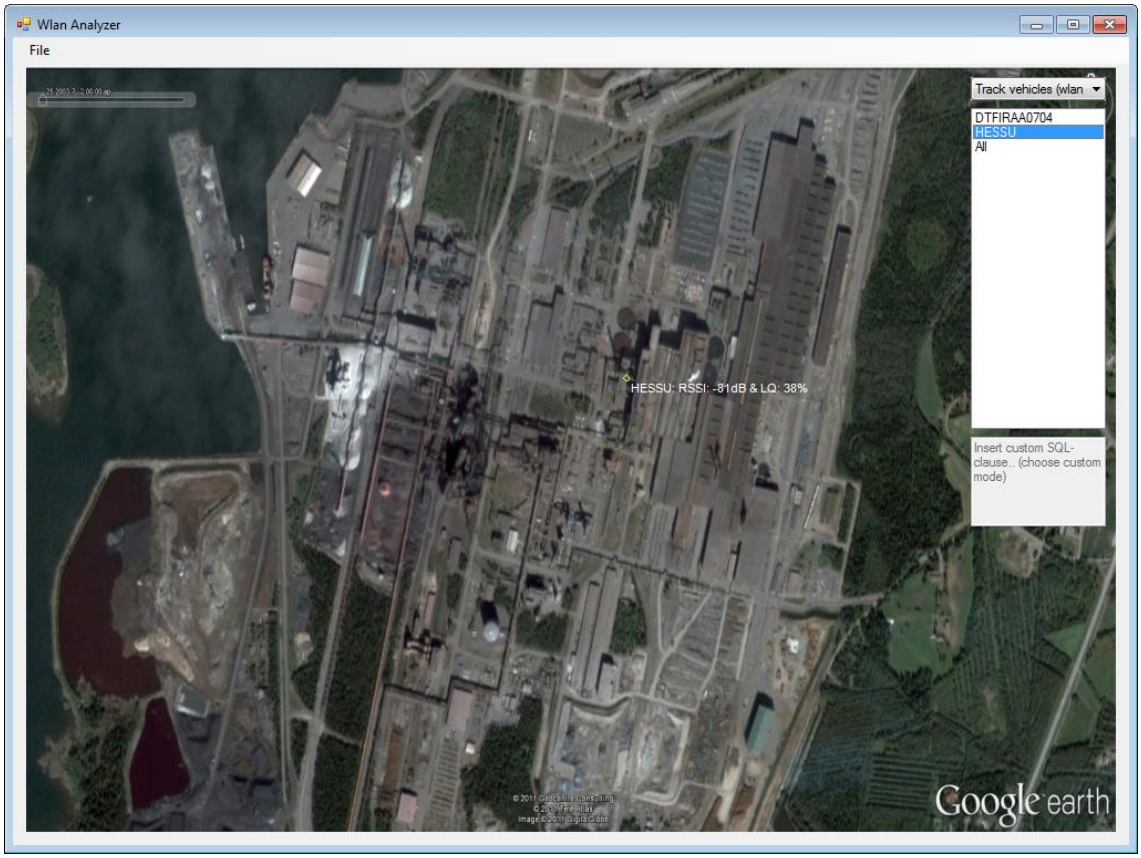
LIITTEET

LIITE 1: Tehdasalueen layout-piirros



LIITE 2: Kuva WLAN-kontrollerit

LIITE 3: Kuvakaappaus tiedon analysointisovelluksen käyttöliittymästä



LIITE 4: Kuvakaappaus tiedonkeruusovelluksen käyttöliittymästä

LIITE 5: Tiedonkeruusovelluksen MainForm-luokan lähdekoodi

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;
using NativeWifi;
using System.Management;
using System.Net.NetworkInformation;

namespace WlanDataLogger
{
    public partial class MainForm : Form
    {
        string[] gps_n = new string[100];
        string[] gps_e = new string[100];
        string[] wlan_ap = new string[100];
        string[] wlan_rssi = new string[100];
        string[] wlan_lq = new string[100];
        bool[] rflag = new bool[100];
        int store_index = 0;
        string devicename;
        string ip;
        string tb_gps_n;
        string tb_gps_e;
        int id;
        WlanClient client = new WlanClient();
        string connectionString = WlanDataLogger.Properties.Resources.connectionString;
        public delegate void InvokeDelegate();
        bool com_port_ok = false;
        string[] comports;
        string SSIDtoScan = "abcdefghijklm";

        public MainForm()
        {
            InitializeComponent();
        }

        private void MainForm_Load(object sender, EventArgs e)
        {
            this.GetComNames("startup");
            this.GetDeviceInfo();
            this.CheckDeviceInfo();
        }

        private void CheckDeviceInfo()
        {
            if (textWlanCard.Text != "" | textDriverDate.Text != "" | textDriverVer.Text != "")
            {
                DataRow[] temp_rows;
                string commandString = @"SELECT * FROM device_data
                WHERE
                Device_name = '" + devicename +
                "' AND Device_ip = '" + ip +
                "' AND Wlan_card = '" + textWlanCard.Text +
                "' AND Wlan_driver_vers = '" + textDriverVer.Text +
                "' AND Wlan_driver_date = '" + textDriverDate.Text +
                "'";
                textMonitor.AppendText("Checking device information from database." + System.Environment.NewLine);
                try
                {
                    SqlDataAdapter da = new SqlDataAdapter(commandString, connectionString);
                    DataTable table = new DataTable();
                    da.Fill(table);
                    temp_rows = table.Select();
                    if (table.Rows.Count == 0)
                    {
                        textMonitor.AppendText("Device information not found, saving device information to database." +
System.Environment.NewLine);
                        this.SaveDeviceInfo();
                    }
                    else
                    {
                        id = (Int32)temp_rows[0][0];
                        textMonitor.AppendText("Device information found, device id " + id + "." + System.Environment.NewLine);
                        Sqltimer.Enabled = true;
                        Wlantimer.Enabled = true;
                        StartTimer.Enabled = false;
                    }
                }
                catch (SqlException ex)
                {
                    StartTimer.Enabled = true;
                    textMonitor.AppendText("Database read failed. Error: " + ex.Message + System.Environment.NewLine);
                }
            }
            else
            {
                textMonitor.AppendText("No proper device information found. Check wlan card connectivity." +
System.Environment.NewLine);
```

```
            }
        }

        private void SaveDeviceInfo()
        {
            try
            {
                using (SqlConnection connection = new SqlConnection(connectionString))
                using (SqlCommand command = connection.CreateCommand())
                {
                    command.CommandText = @"INSERT INTO device_data (Timestamp, Device_name, Device_ip, Wlan_card, Wlan_mac,
Wlan_driver_vers, Wlan_driver_date)
                    VALUES
                    (GetDate(), @Device_name, @Device_ip, @Wlan_card, @Wlan_mac, @Wlan_driver_vers, @Wlan_driver_date)";
                    command.Parameters.AddWithValue("@Device_name", devicename);
                    command.Parameters.AddWithValue("@Device_ip", ip);
                    command.Parameters.AddWithValue("@Wlan_card", textWlanCard.Text);
                    command.Parameters.AddWithValue("@Wlan_mac", textMac.Text);
                    command.Parameters.AddWithValue("@Wlan_driver_vers", textDriverVer.Text);
                    command.Parameters.AddWithValue("@Wlan_driver_date", textDriverDate.Text);
                    Console.WriteLine(connection.ConnectionTimeout);
                    connection.Open();
                    command.ExecuteNonQuery();
                    textMonitor.AppendText("Save ok." + System.Environment.NewLine);
                    Sqltimer.Enabled = true;
                    Wlantimer.Enabled = true;
                    StartTimer.Enabled = false;
                }
            }
            catch (SqlException ex)
            {
                StartTimer.Enabled = true;
                textMonitor.AppendText("Database save failed. Error: " + ex.Message + System.Environment.NewLine);
            }
        }

        private void GetComNames(string state)
        {
            switch (state)
            {
                case "startup":
                    comports = System.IO.Ports.SerialPort.GetPortNames();
                    int comindex = comports.Count();
                    textMonitor.AppendText("Searching for COM ports." + System.Environment.NewLine);
                    if (comindex - 1 > 0) { serialPort.PortName = comports[comindex - 1]; }
                    foreach (string i in comports)
                    {
                        comboCom.Items.Add(i);
                        if (serialPort.PortName == i) { comboCom.SelectedItem = i; }
                        this.SerialStart();
                    }
                    break;
                case "switch":
                    if (comboCom.Items.Count < 2)
                    {
                        textMonitor.AppendText("No serial data received, check GPS-receiver" + System.Environment.NewLine);
                        this.serialPort.Close();
                    }
                    else
                    {
                        textMonitor.AppendText("No serial data received, switching COM port." + System.Environment.NewLine);
                        comboCom.SelectedIndex = (comboCom.SelectedIndex - 1);
                        serialPort.PortName = comboCom.SelectedText;
                        this.SerialStart();
                    }
                    break;
            }
            ComTimer.Enabled = true;
        }

        private void CheckComStatus()
        {
            if (com_port_ok == false)
            {
                this.GetComNames("switch");
            }
            else
            {
                ComTimer.Enabled = false;
            }
        }

        private string FixMac(string mac)
        {
            string fixed_mac = "";
            for (int x = 0; x < mac.Length; x++)
            {
                if (x == 2 | x == 4 | x == 6 | x == 8 | x == 10)
                {
                    fixed_mac = fixed_mac + ":" + mac[x];
                }
                else
                {
                    fixed_mac = fixed_mac + mac[x];
                }
            }
            return fixed_mac;
        }
```

```
        private void GetDeviceInfo()
        {
            textMonitor.AppendText("Searching for device information." + System.Environment.NewLine);
            devicename = System.Net.Dns.GetHostName();
            textName.Text = devicename;
            ip = System.Net.Dns.GetHostByName(devicename).AddressList[0].ToString();
            textIp.Text = ip;
            string wlanCardName = null;
            NetworkInterface[] interfaces = NetworkInterface.GetAllNetworkInterfaces();
            foreach (NetworkInterface adapter in interfaces)
            {
                if (adapter.NetworkInterfaceType.ToString() == "Wireless80211")
                {
                    textMac.Text = this.FixMac(adapter.GetPhysicalAddress().ToString());
                    textWlanCard.Text = adapter.Description;
                    wlanCardName = adapter.Description;
                }
            }
            WqlObjectQuery wqlQuery = new WqlObjectQuery(@"SELECT DriverDate,DriverVersion FROM Win32_PnpSignedDriver WHERE
DeviceName = '" + wlanCardName + "'");
            ManagementObjectSearcher searcher = new ManagementObjectSearcher(wqlQuery);
            foreach (ManagementObject obj in searcher.Get())
            {
                textDriverDate.Text = "" + obj["DriverDate"];
                textDriverVer.Text = "" + obj["DriverVersion"];
            }
        }

        static string GetStringForSSID(Wlan.Dot11Ssid ssid)
        {
            return Encoding.ASCII.GetString(ssid.SSID, 0, (int)ssid.SSIDLength);
        }

        private void GetWlanInfo()
        {
            try
            {
                foreach (WlanClient.WlanInterface wlanIface in client.Interfaces)
                {
                    Wlan.WlanAvailableNetwork[] networks = wlanIface.GetAvailableNetworkList(0);
                    Wlan.WlanBssEntry[] redes = wlanIface.GetNetworkBssList();
                    foreach (Wlan.WlanBssEntry wlanBssEntry in redes)
                    {
                        if (GetStringForSSID(wlanBssEntry.dot11Ssid) == SSIDtoScan)
                        {
                            byte[] macAddr = wlanBssEntry.dot11Bssid;
                            var macAddrLen = (uint)macAddr.Length;
                            var str = new string[(int)macAddrLen];
                            for (int i = 0; i < macAddrLen; i++)
                            {
                                str[i] = macAddr[i].ToString("x2");
                            }
                            string mac = string.Join("", str);
                            textSSID.Text = "" + GetStringForSSID(wlanBssEntry.dot11Ssid);
                            textWlanAP.Text = mac;
                            textRSSI.Text = "" + wlanBssEntry.rssi;
                            textLinkQ.Text = "" + wlanBssEntry.linkQuality;
                        }
                    }
                }
            }
            catch (Exception ex)
            {
                textMonitor.AppendText("Wlan: Error - " + ex.Message + System.Environment.NewLine);
            }

        }

        private void SerialStart()
        {
            try
            {
                serialPort.Open();
                if (serialPort.IsOpen)
                {
                    textMonitor.AppendText("COM Port: Connection ok" + System.Environment.NewLine);
                    comboCom.Enabled = false;
                }
                else
                {
                    textMonitor.AppendText("COM Port: Connection failed" + System.Environment.NewLine);
                }
            }
            catch (Exception ex)
            {
                textMonitor.AppendText("COM Port: Error - " + ex.Message + System.Environment.NewLine);
                this.GetComNames("switch");
            }
        }

        private void Sqltimer_Tick(object sender, EventArgs e)
        {
            this.SqlWrite();
        }

        private void StoreData()
        {
            if (store_index > (gps_n.Count() - 1)) { store_index = 0; }
            gps_n[store_index] = this.textNorth.Text;
```

```
            gps_e[store_index] = this.textEast.Text;
            wlan_ap[store_index] = textWlanAP.Text;
            wlan_rssi[store_index] = textRSSI.Text;
            wlan_lq[store_index] = textLinkQ.Text;
            textMonitor.AppendText("System: Wlan connection failed, stored data " + gps_n[store_index] + " at position " +
store_index + ".");
            textMonitor.AppendText("System: Wlan connection failed, stored data " + gps_e[store_index] + " at position " +
store_index + ".");
            rflag[store_index] = false;
            store_index++;
        }

        private void RestoreData()
        {
            for (int x = 0;x < store_index; x++)
            {
                if (rflag[x] == false)
                {
                    try
                    {
                        using (SqlConnection connection = new SqlConnection(connectionString))
                        using (SqlCommand command = connection.CreateCommand())
                        {
                            command.CommandText = "INSERT INTO wlan_data (ID, Gps_N, Gps_E, Wlan_ap, Wlan_lq, Wlan_rssi) VALUES
(@ID, @Gps_N, @Gps_E, @Wlan_ap, @Wlan_lq, @Wlan_rssi)";
                            command.Parameters.AddWithValue("@ID", id);
                            command.Parameters.AddWithValue("@Gps_N", gps_n[x]);
                            command.Parameters.AddWithValue("@Gps_E", gps_e[x]);
                            command.Parameters.AddWithValue("@Wlan_ap", wlan_ap[x]);
                            command.Parameters.AddWithValue("@Wlan_lq", wlan_lq[x]);
                            command.Parameters.AddWithValue("@Wlan_rssi", wlan_rssi[x]);
                            connection.Open();
                            command.ExecuteNonQuery();
                            textMonitor.AppendText("System: Restored data to database from position " + x + "." +
System.Environment.NewLine);
                            rflag[x] = true;
                        }
                    }
                    catch (SqlException ex)
                    {
                        Console.WriteLine(ex.Message);
                        return;
                    }
                }
            }
        }

        private void SqlWrite()
        {
            if (textNorth.Text != "" && textEast.Text != "")
            {
                try
                {
                    using (SqlConnection connection = new SqlConnection(connectionString))
                    using (SqlCommand command = connection.CreateCommand())
                    {
                        command.CommandText = @"INSERT INTO wlan_data (ID, Timestamp, Gps_N, Gps_E, Wlan_ap, Wlan_lq, Wlan_rssi)
                                                VALUES (@ID, GetDate(), @Gps_N, @Gps_E, @Wlan_ap, @Wlan_lq, @Wlan_rssi)";
                        command.Parameters.AddWithValue("@ID", id);
                        command.Parameters.AddWithValue("@Gps_N", textNorth.Text);
                        command.Parameters.AddWithValue("@Gps_E", textEast.Text);
                        command.Parameters.AddWithValue("@Wlan_ap", textWlanAP.Text);
                        command.Parameters.AddWithValue("@Wlan_lq", textLinkQ.Text);
                        command.Parameters.AddWithValue("@Wlan_rssi", textRSSI.Text);
                        connection.Open();
                        command.ExecuteNonQuery();
                        textMonitor.AppendText("System: Measurements stored to database" + System.Environment.NewLine);
                        this.RestoreData();
                    }
                }
                catch (SqlException ex)
                {
                    Console.WriteLine(ex.Message);
                    this.StoreData();
                }
            }
        }

        private void MainForm_FormClosing(object sender, FormClosingEventArgs e)
        {
            if (serialPort.IsOpen) serialPort.Close();
        }

        private void serialPort_DataReceived(object sender, System.IO.Ports.SerialDataReceivedEventArgs e)
        {
            serialPort.DiscardInBuffer();
            serialPort.DiscardOutBuffer();
            string serialString = serialPort.ReadLine();
            string gpgga = "$GPGGA";
            string questionmark = "?";
            string north = "";
            string south = "";
            for (int x = 0; x < serialString.Length; x++)
            {
                if (serialString[x] == questionmark[0])
                {
                    textNorth.Clear();
                    textEast.Clear();
                    return;
```

```
                }
            }
            if (serialString[0] == gpgga[1] && serialString[1] == gpgga[2] && serialString[2] == gpgga[3] && serialString[3] ==
gpgga[4] && serialString[4] == gpgga[5])
            {
                for (int x = 17; x < 26; x++)
                {
                    north = north + serialString[x].ToString();
                }
                for (int x = 30; x < 39; x++)
                {
                    south = south + serialString[x].ToString();
                }
                com_port_ok = true;
                tb_gps_n = north;
                tb_gps_e = south;
                textNorth.BeginInvoke(new InvokeDelegate(InvokeMethod));
                return;
            }
            else
            {
                return;
            }
        }

        private void InvokeMethod()
        {
            textNorth.Text = tb_gps_n;
            textEast.Text = tb_gps_e;
        }

        private void Wlantimer_Tick(object sender, EventArgs e)
        {
            this.GetWlanInfo();
        }

        private void StartTimer_Tick(object sender, EventArgs e)
        {
            this.CheckDeviceInfo();
        }

        private void ComTimer_Tick(object sender, EventArgs e)
        {
            this.CheckComStatus();
        }

        private void MainForm_Resize(object sender, EventArgs e)
        {
            if (FormWindowState.Minimized == WindowState) { Hide(); }
        }

        private void notifyIcon1_DoubleClick(object sender, EventArgs e)
        {
            Show();
            WindowState = FormWindowState.Normal;
        }
    }
}
```

LIITE 6: Lähdekoodit ja projektitiedostot CD-levyllä