Niklas Zuban

City Car Concept -projektin mittariston käyttöliittymä ja toiminta

Metropolia Ammattikorkeakoulu
Koneinsinööri (AMK)
Kone- ja tuotantotekniikka
Insinöörityö
6.6.2012

ALKUSANAT

Tämä insinöörityö tehtiin Metropolia Ammattikorkeakoulun Eerikinkadun Koneautomaatiolaboratoriossa. Kiitän erityisesti laboratorion henkilökuntaa ja asistenttejä, erityisesti Pasi Yrjölää, Ville Nopria ja Pauli Nevalaista. Lisäksi haluan kiittää vielä ohjaavaa opettajaa tekniikan tohtori Jari Savolaista hänen antamastaan mahdollisuudesta ja tuesta toteuttaa kyseinen projekti. Erityiskiitokset kuuluvat myös työssä mainittavien autoprojektien vetäjille Sami Ruotsalaiselle ERA-projektista ja Harri Santamalalle Kaupunkiauto-projektista.

Helsingissä 6.6.2012

Niklas Juhani Zuban

Tekijä Otsikko Sivumäärä Aika	Niklas Zuban City Car Concept -projektin mittariston käyttöliittymä ja toiminta 34 sivua + 4 liitettä 6.6.2012
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Kone- ja tuotantotekniikka
Suuntautumisvaihtoehto	Koneautomaatio
Ohjaaja	Lehtori Jari Savolainen, Metropolia Ammattikorkeakoulu

Tässä insinöörityössä tehtiin virtuaalinen mittaristo ERA-sähköautoon ja Metropolia Ammattikorkeakoulun Kestävän kehityksen kaupunkiauto -projektiin. Tarkoituksena oli luoda mittaristo molempien autojen tarpeisiin. Toisessa autossa mittaristo on ensimmäinen laatuaan, ja toisessa sen on tarkoitus korvata alkuperäinen.

Mittaristo rakennettiin täysin tyhjältä pöydältä. Sen visuaalinen ilme perustuu kokonaan työn tekijän omiin mielikuviin siitä, miltä ajoneuvon mittariston tulisi näyttää. Rajoittavana tekijänä olivat vain tekijän taiteelliset ja tekniset taidot.

Mittariston toiminta perustuu CAN-väylältä luettavaan tietoon ja sen kirjoittamiseen näytölle käyttäjän ymmärtämään muotoon. Osa tiedoista esitetään kuvien ja osa numeroiden avulla. Tärkeintä suunnittelussa ja toteutuksessa oli valita olennaisimmat tiedot päänäytölle ja vähemmän olennaiset tiedot piilotettuihin lisäpalkkeihin. Näiden piilotettujen palkkien on tarkoitus tulla käyttäjälle näkyviin pyydettäessä eli esimerkiksi tiettyä painiketta painettaessa. Tärkeää työssä oli luoda ohjelma, joka kykenee muuntautumaan helposti sekä sähköauton että diesel-käyttövoimalla toimivan auton tarpeisiin. Kaupunkiauton valmistumisen edetessä on tärkeää, ettei ohjelmaa tarvitse radikaalisti muuttaa, jotta sen saisi sopimaan eri ajoneuvoon. Koska toisen ajoneuvon valmistuminen kestää huomattavan kauan, keskityttiin tässä työssä lopulta saamaan mittaristosta toimiva versio toiseen autoon, kuitenkin huomioimalla toisen auton tarpeet.

Avainsanat	Mittaristo, sähköauto, CAN-väylä

Author	Niklas Zuban
Title	Car meters user interface and operation of City Car Concept -project
Number of Pages	34 pages + 4 appendices
Date	6 June 2012
Degree	Bachelor of Engineering
Degree Programme	Mechanical Engineering
Specialisation option	Machine Automation
Instructor(s)	Jari Savolainen, D.Sc (tech.)

The goal of this thesis was to create a virtual instrument panel for ERA and City Car Concept projects, which both are car design projects of Metropolia University of Applied Sciences. The aim was to create the instrument panel for different needs of the projects. In one project the panel will be the first of its kind and in the other it will replace the original one. The project was very challenging and some parts of the original plan had to be left outside of it.

The instrument panel was created from the scratch and the exterior was based on author's views on how a panel of a car should look like. The only restrictions were author's creative and technological skills.

The functionality of the instrument panel is based on the information received from CAN-bus, which is displayed on screen so that the driver understands it. The most important thing of design phase was to choose the most important information to main view and less essential information to hidden side bars. These bars would become visible when requested by the driver by pressing a button inside the cockpit.

Important part of the project was to design the program code that would easily transform to the needs of an electric car and diesel powered car. It is essential to be able to compile the program code to Concept City Car as it progresses without changing it much. Because one car will not be complete in a long time, the focus will be on the other, but the needs of both will be notified.

Keywords	Instrument panel, electric car, CAN-bus

Sisällys

Lyhenteet

ARM	Advanced RISC Machines. 32-bittinen mikroprosessoriarkkitehtuuri, jonka kehitti Acorn 1980-luvun puolivälissä.
C++	Ohjelmistokieli. Sen kehitti Bjarne Stroustrup 1980-luvulla. Uusin standardi on vuodelta 2011 (C++11).
CAN	Controller-Area Network. Teollisuudessa yleisesti käytössä oleva sarjaliikennekommunikointijärjestelmä.
ERA	Electric Race About. Metropolia Ammattikorkeakoulun tuottama sähkökäyttöinen urheiluautoprototyyppi.
Qt	Nokian omistama kehitysympäristö ja käyttöliittymäkirjasto.
SD	Secure Digital. Yksi yleisimmistä muistikorttityypeistä.
SVG	Scalable Vector Graphics. Avoin kaksiulotteinen vektorikuvastandardi.
VDO	Continentalin tytäryhtiö, joka valmistaa ajoneuvojen osia ja tarvikkeita.

1 Johdanto

Tämä insinöörityö on osa Metropolia Ammattikorkeakoulun kestävän kehityksen kaupunkiauto -projektia. Projektin lopullisena tarkoituksena on luoda toimiva konseptiauto kaupunkikäyttöön. Auton valmistus sisältää useita eri osakokonaisuuksia. Tässä työssä perehdytään mittariston käyttöliittymään, ohjelmistoon ja laitteistoon. Tarkoituksena on luoda ohjelma, joka näyttää ajoneuvon tarpeelliset tiedot näytöllä perinteisen mittariston tavoin sekä tarvittaessa kykenee kertomaan käyttäjälle myös muita tietoja. Insinöörityön ohjelmointiosuus toteutetaan Qt-ohjelmointiympäristöllä ja sitä käytetään Ubuntu-käyttöjärjestelmässä.

Myös Metropolian edellinen projektiauto ERA (Electric Race About) on kiinnostunut mittariston uusimisesta. ERAssa mittariston tulee toimia sen vanhalla autokäyttöön suunnitellulla PC:llä, jonka käyttöjärjestelmänä toimii Microsoft Windows CE 5.

Tässä insinöörityössä keskitytään mittariston käyttöliittymän ja toimivuuden luomiseen. Mittariston käyttöliittymän suunnittelu on sekä ohjelmallista että osittain myös graafista. Toimivuuden kannalta työssä käsitellään ohjelmakoodin lisäksi myös fyysisiä komponentteja. Työ on rajattu mittariston kehittämiseen, eikä työssä huomioida muita autoon tulevia pc- tai elektroniikkakomponentteja. Yhteistyötä sen sijaan tehdään samaan aikaan auton ajotietokonetta kehittävän ryhmän kanssa projektien samankaltaisten osakokonaisuuksien kohdalla.

Lähtöidea oli käyttää konseptiajoneuvossa perinteisen mittariston tilalla kämmentietokoneelle ohjelmoitua virtuaalista mittaristoa. Tämän idean pohjalta tässä työssä kehitettiin ja suunniteltiin mittaristo ajoneuvoon.

2 Projektin lähtökohdat

City Car Concept -projekti on osa Metropolia Ammattikorkeakoulun kestävän kehityksen kaupunkiauto -projektia. Auton suunnittelu on aloitettu vuonna 2010 ja valmista autoa on tarkoitus esitellä Geneven automessuilla maaliskuussa 2015. Autoa suunnitellaan useissa eri tiimeissä, näistä mainittakoon sähkösuunnittelu-, muotoilu-, korin valmistus- ja ohjelmistoryhmä.

ERA-projektin kohdalla taas lähtöpiste on hieman erilainen. Ajoneuvo on jo valmis ja sitä on esitelty monilla eri messuilla. Auton vanha mittaristo on alun perin tehty Microsoftin Visual Studio -ohjelmistolla ja sen visuaalinen olemus on hieman karkea. Tätä visuaalista ilmettä haluttiin ERAssa uusia, ja siksi mittaristoa suunniteltiin myös siihen.

Haastavuutta projektiin luo ajoneuvojen erilaisuudet ja täysin eri lähtöpisteet. Kaupunkiauton kohdalla ongelmia luo auton keskeneräisyys. Se on vasta niin sanotusti alku tekijöissään, eikä juuri mitään ole vielä lyöty lukkoon. ERA taas on täysin sähköllä kulkeva jo valmis ajoneuvo, ja sen mittaristoon eivät sovi kaikki diesel-polttoainetta käyttävien autojen vaatimat mittarit.

Erilaisuuksistaan huolimatta samaa mittaripohjaa halutaan sen valmistuttua käyttää molemmissa autoissa.

3 Ohjelmointi ja laitteisto

Mittariston vaatimuksina oli alun perin luoda täysin käyttäjän hallinnoima graafinen ratkaisu, jossa eri mittareita voisi piilottaa, siirtää, suurentaa ja pienentää. Mittareiden siirtämistoiminnosta luovuttiin melko nopeasti, koska sen toteuttaminen kahteen autoon, joissa on erilaiset komponentit, on erittäin haastavaa. Lisäksi auto, johon tämä mahdollisuus alun perin haluttiin, ei ole vielä läheskään valmis ja tämän työn kannalta on tärkeää saada tietoa mittariston toiminnasta oikeassa ajoneuvossa.

Tämän päätöksen jälkeen kaupunkiauto siirtyi hieman taka-alalle tässä työssä ja alettiin keskittyä enemmän ERAn tarpeisiin. Kaupunkiautoa ei kuitenkaan unohdettu, ja kaikki kaupunkiauton yksilölliset ominaisuudet on helppo lisätä mittaristoon jälkikäteen.

Ajoneuvojen eroavaisuuksista esimerkkinä esille voidaan nostaa hyvin tavallisesti mittaristoon sijoitettu kierroslukumittari, joka korvataan sähköautossa tehon kulkusuuntaa ja määrää ilmoittavalla mittarilla. Tehon kulkusuunnalla tässä tarkoitetaan sitä, että kulkeeko teho auton akusta moottorille vai toimivatko auton moottorit generaattoreina ja tuottavat tehoa takaisin akkuun.

Lopullisena ratkaisuna ERAn vanhan mittariston pohjalta lähdettiin siis luomaan uutta käyttäjäystävällisempää ratkaisua. Kaupunkiauto-projektissa syntynyttä ideaa ei hylätty tässäkään autossa, vaan erillinen enemmän informaatiota sisältävä vaihtoehtoinen ikkuna lisättiin myös tähän autoon.

3.1 Suunnittelu

Mittariston suunnittelu lähti liikkeelle kaupunkiauton laitteiston valinnasta, jotta ohjelmaa voidaan suunnitella myös sille. Valmiina laitteena oli käytössä Acerin valmistama Android-käyttöjärjestelmää ja ARM-prosessoritekniikkaa käyttävä A500-mallinen laite. Laitteen mitat vaikuttivat sopivilta ajoneuvon muotoiluun, mutta käyttöjärjestelmän suojauksien takia laite hylättiin. Uudeksi laitteeksi valittiin myös Acerin valmistama kämmentietokone mallia Iconia W500 (kuva 1), joka on edellisen mallin kanssa lähes identtinen ominaisuuksiltaan. W500-mallin etuna on vain sen erilainen prosessori, joten sillä voidaan ajaa Microsoftin Windows -käyttöjärjestelmiä. Koska Windows-käyttöjärjestelmä on todella raskas nykyisille kämmentietokoneille, päädyttiin siinä ajamaan Ubuntu-käyttöjärjestelmää.

Acer Iconia W500:n tarkempia teknisiä tietoja ovat 2 GB:n järjestelmä muisti ja 32 GB:ä tallennustilaa. Suorittimen lisäksi laitteessa on erillinen grafiikkakiihdytin ja 10.1 tuuman näyttö, jonka pikselitarkkuus on 1280 vaakasuuntaan ja 800 pystysuuntaan. [1.]

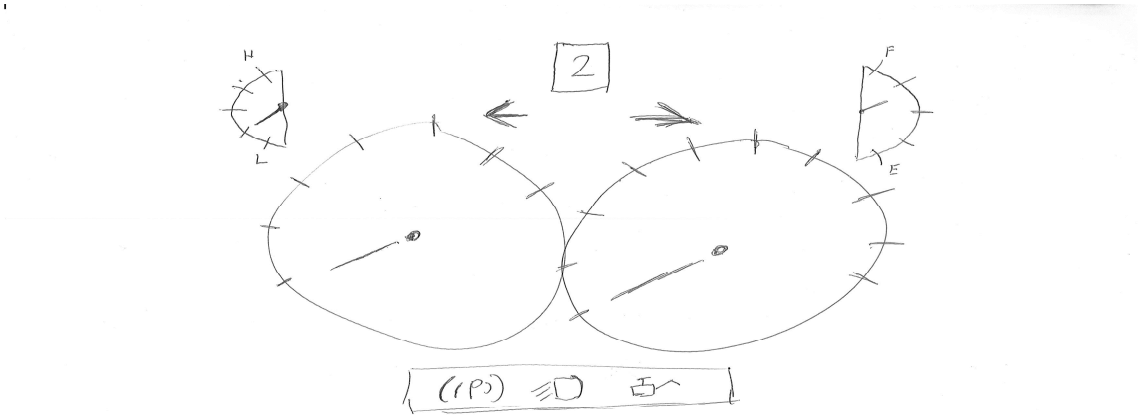
Kuva 1. Acer Iconia W500 kämmentietokone.

ERAssa laitteiston virkaa ajaa ajoneuvokäyttöön suunniteltu PC (kuva 2). PC:n valmistaja on VDO ja käyttöjärjestelmänä laitteessa on Windows CE 5. [2, s. 3.] PC:n näyttö on seitsemän tuumaa ja näytön tarkkuus on 800 kertaa 480 pikseliä. Laitteessa on 32 MB:n järjestelmän suoritusmuisti ja SD-korttipaikka ohjelmien varastointiin. Laite tukee vain tavallisia SD-kortteja kahteen GB:in saakka. Lisäksi laitteessa on oma CAN-väylä, joten lopullisessa ohjelmassa tulee olla tuki myös laitteen omalle CAN-portille.

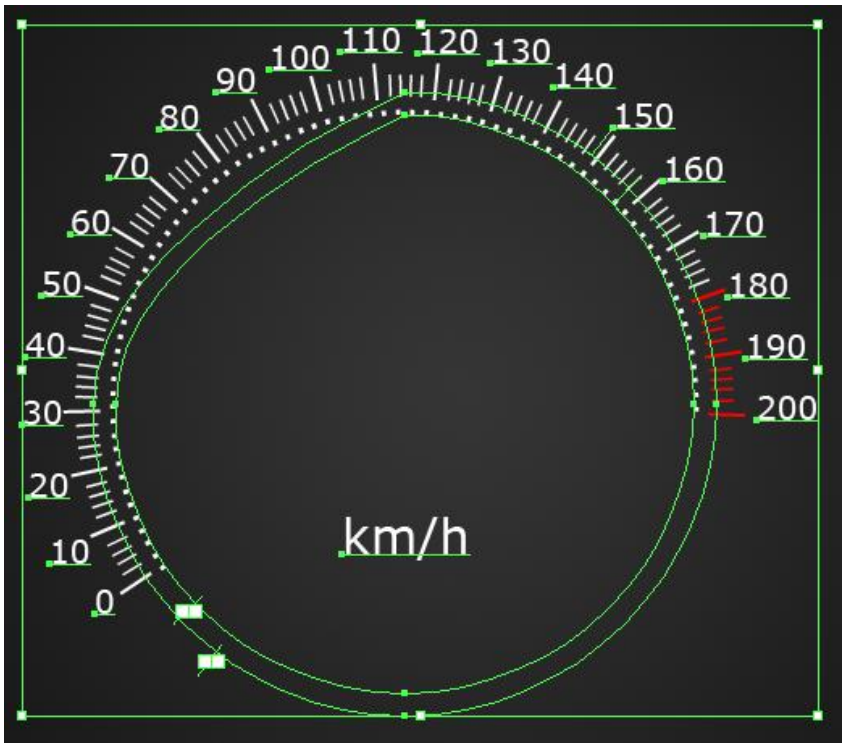
Toisena mahdollisuutena oli käyttää ERAan jo valmiiksi hankittua vaihtoehtoista laitetta, jonka on ollut tarkoitus korvata nykyisellään käytössä oleva laite. Tämä uudempi PC on ominaisuuksiltaan hyvin samanlainen kuin edeltäjänsä. Uuden laitteen käyttöjärjestelmänä toimii kuitenkin Ångström-linux, joka on avoimen lähdekoodin -projekti.

Kuva 2. ERAn mittaristo PC.

Kun laitteistopuoli oli mietitty, alettiin mittaristoa hahmotella paperille. Paperille piirrettiin hyvin karkeasti, minkä muotoisia ja minkälaista informaatiota sisältäviä kohteita ruudulla tullaan lopulta esittämään (kuva 3). Näiden kuvien pohjalta aloitettiin digitaalisten kuvien suunnittelu Adoben Photoshop-kuvankäsittelytyökalulla. Ensimmäinen luonnos perustui lähinnä vain eri kirjainyhdistelmistä luotuun kaarevaan mittaritauluun. Se ei kuitenkaan toiminut, koska kaarevan tekstin näyttäminen ohjelmassa vääristi kuvaa niin, ettei siitä saanut enää selvää (kuva 4).



Kuva 3. Ensimmäisiä hahmotteluja mittariston ulkoasusta paperille.

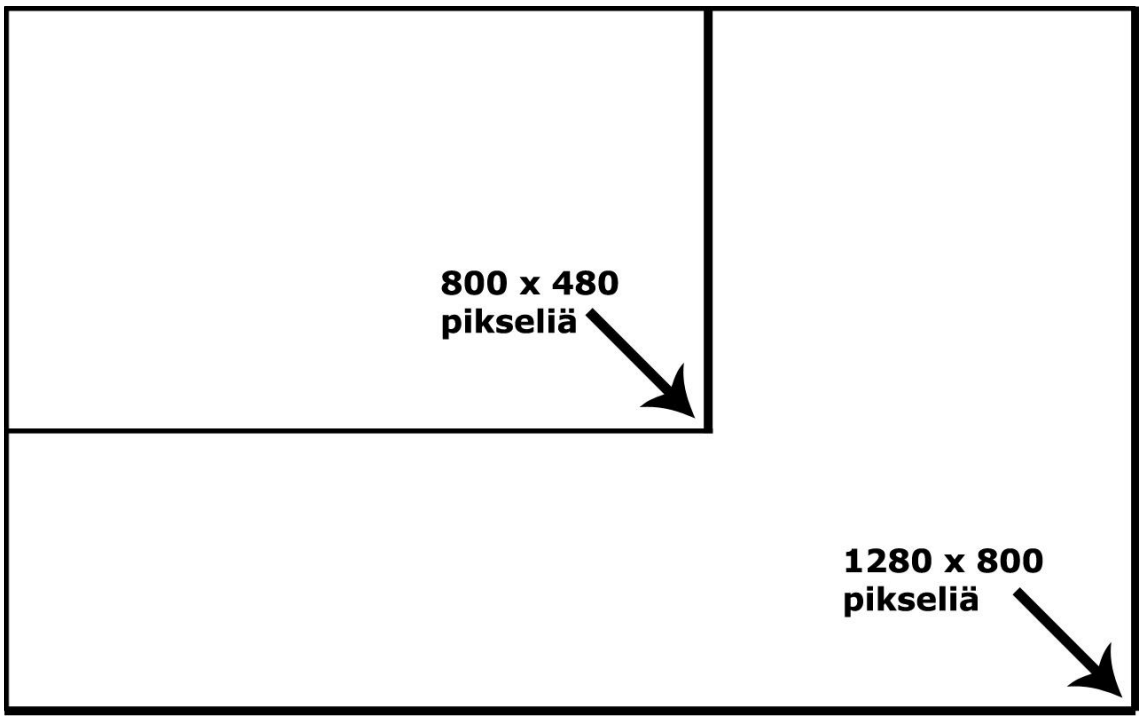


Kuva 4. Ohjelmassa nopeutta ilmaisevat viivat vääristyivät siten, että ne siirtyivät seuraamaan ympyrän vasemmasta yläkulmasta radaltaan poikkeavaa ääriviivaa, joka näkyy kuvassa vihreänä.

3.2 Ohjelmointi

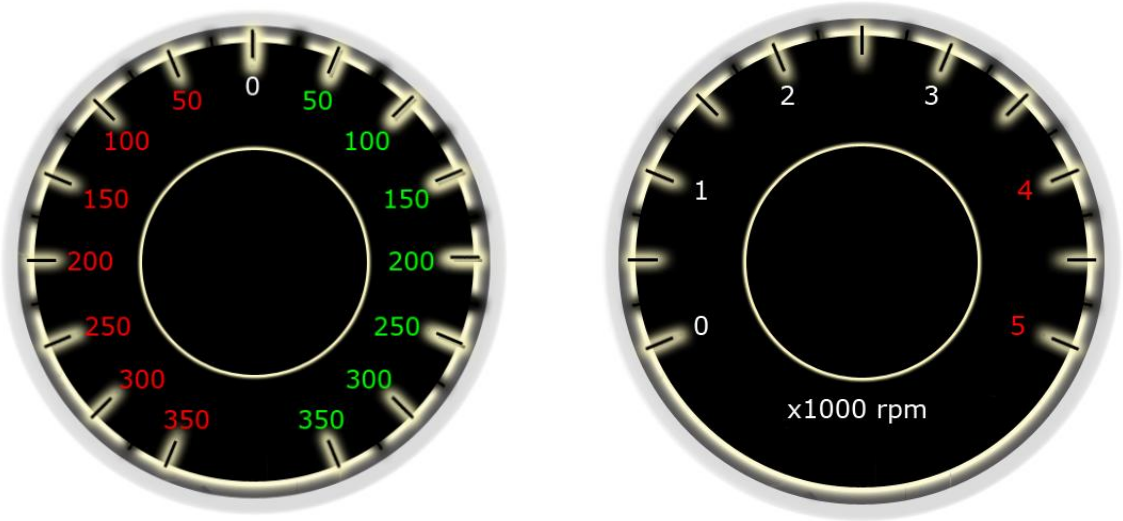
Tärkeintä ohjelmoinnin aloituksessa oli päättää, kuinka mittarit tullaan esittämään ohjelmassa. Ulkoasun määrittämät muotoilut, kuten pyöreät mittarit ja laatikkomaiset merkkivalot, päätettiin toteuttaa ulkoisina kuvina, joita vain esitetään käyttäjälle. Lisäksi haluttiin, että käyttäjä voi tarvittaessa vaihtaa mittaristonäkymästä enemmän tietoa sisältävään listamaiseen yhteenvetotilaan. Tämä toteutusmalli autoista tutusta mittaristotyylistä kelpasi myös ERA-projektiin. Näkymä, jossa on lisää informaatiota, on tärkeä, koska etenkin prototyyppivaiheessa kehittäjät tarvitsevat paljon tietoa auton tilasta.

Aluksi kun mittaristoa alettiin ohjelmoida, huomattiin että autojen näyttöjen tarkkuudet olivat erikokoiset (kuva 5). Tämä ongelma ratkaistiin sillä, että kuvat tehtiin vektorigrafiikkana ja mittaristo skaalautuu aina käynnistyksen yhteydessä kulloinkin vallitsevan näytöntarkkuuden mukaan.



Kuva 5. Näyttöjen erilaiset resoluutiot on hahmoteltu päällekkäin.

Toisena ongelmana havaittiin, että autoihin tulee erilaisia mittareita (kuva 6). Tämän ongelman kanssa tultiin siihen tulokseen, että autosta riippuen valitaan tietynlainen mittari tiettyyn paikkaan. Yleisimmät mittarit, kuten nopeus-, polttoaine-, akun varaus - ja moottorin lämpötila -, pysyvät luonnollisesti samanlaisina molemmissa ajoneuvoissa.



Kuva 6. Kaksi erilaista mittaria, joiden sijainti on sama riippuen ajoneuvosta.

Mittaristo rakennetaan käyttöjärjestelmässä suoritettavaksi prosessiksi, joka hakee, analysoi ja näyttää informaatiota. Tämä informaatio on peräisin ajoneuvon moottorin ja hallintalaitteiden ohjausjärjestelmistä. Ajoneuvoihin on rakennettu CAN-väylä, joka kuljettaa anturien tuottamaa dataa ajoneuvojen ohjausjärjestelmille ja PC-laitteille. Lisäksi nämä ohjausjärjestelmät tuottavat käskyjä ja neuvoja saamastaan informaatiosta ja lähettävät nämä ohjeet takaisin toimilaitteille. Mittariston on tarkoitus sijoittua ryhmältään niin sanottuihin passiivisiin laitteisiin, jotka vain lukevat väylälle kirjoitettua tietoa ja tässä tapauksessa näyttävät sen graafisesti käyttäjälle. Mittariston ei ole tarkoitus antaa käskyjä toimilaitteille, kuten vaihteistolle tai ohjaustehostimelle. Vaikka mittariston saaman datan avulla olisikin helppoa laskea tarvittavat toimenpiteet, niin jätetään se ajoneuvon moottorinohjauslaitteiden suoritettavaksi.

3.3 CAN-väylä

CAN-väylä on automaatioväylä, joka koostuu parikaapelista. Parikaapelin kierretyistä johtimista käytetään nimitystä high ja low. Kaapelien normaalit jännitteet ovat 3,5 V ja 1,5 V. Kun väylällä halutaan siirtää tietoa, jännite-eroja muutetaan. Esimerkiksi jos väylälle halutaan kirjoittaa arvo yksi, molempien johdinten jännitteeksi asetetaan 2,5 V. CAN-väylän johdinten päätevastusten arvo on 120 Ω. [3.]

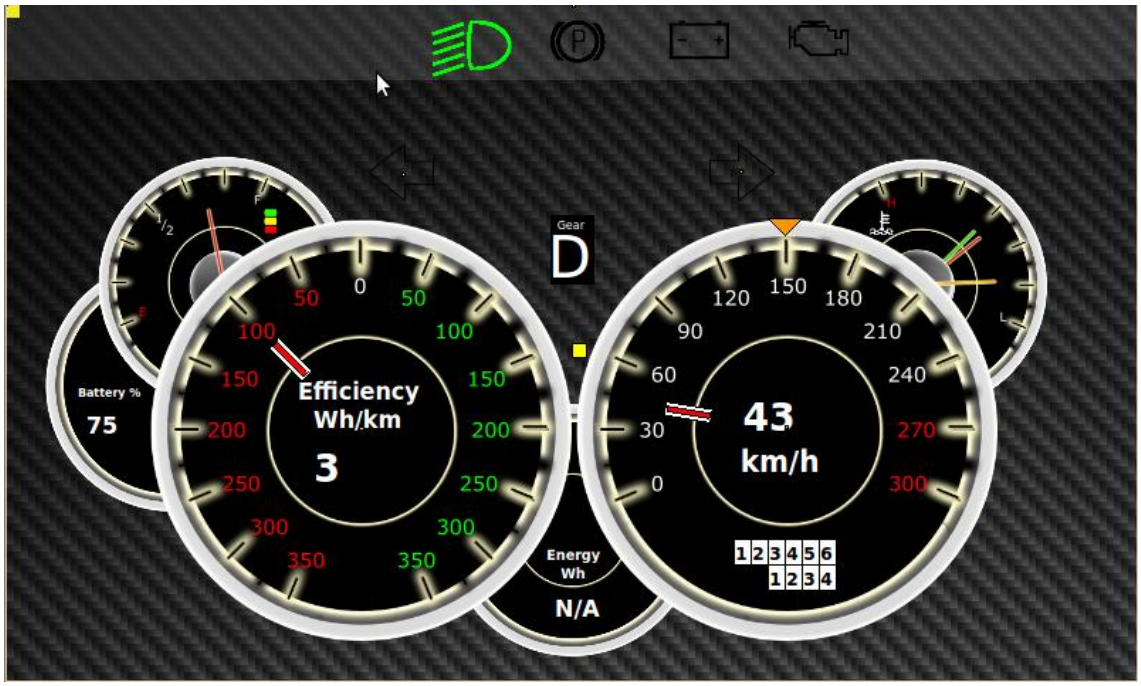
CAN-väylä on otettu käyttöön jo vuonna 1983, jolloin saksalainen ajoneuvoelektroniikan yritys Bosch käytti sitä ajoneuvojen ABS-jarruissa ja moottorinohjauksessa. CAN-väylän vikasietoisuuden vuoksi se on edelleen käytössä monissa erilaisissa sovelluksissa, kuten raskaissa ajoneuvoissa mutta myös terveydenhuollon ja lääketeollisuuden laitteissa. [3.]

Väylään kytketyt laitteet keskustelevat keskenään kirjoittamalla ja lukemalla väylää. Jokainen laite päättää viestin tunnisteen perusteella kuuluko viesti sille. [3.]

4 Käyttöliittymä ja sen rakentuminen

Käyttöliittymän peruselementit tulevat perinteisistä ajoneuvojen mittareista. Ajoneuvoja koskevat Tieliikenneturvallisuusviraston ja Euroopan unionin direktiivit sekä lait ja asetukset määräävät tiettyjä toimintoja ja ominaisuuksia mittaristoon. Pakollisina ominaisuuksina mainittakoon nopeusmittari ja matkamittari. Ilman kaikkia pakollisia ominaisuuksia ajoneuvoa ei voida katsastaa tieliikennekäyttöön (kuva 7).

Pakollisten ominaisuuksien lisäksi ajoneuvoon haluttiin paljon lisäominaisuuksia, kuten virankulutusmittari sekä lämpötilamittarit auton eri komponenteista kuten moottoreista, moottorinjäähdyttimistä, inverttereistä ja akuista. Osa näiden arvoista sijoitettiin informaationäytön puolelle, ja osa taas samaan mittariin, jossa on monta neulaa osoittamassa kunkin erillisen laitteen lämpötilaa. Koska lämpötilojen normaaliarvot poikkeavat hyvin paljon toisistaan, esimerkiksi akun lämpötilan normaaliarvo on 55 °C, kun taas moottorien lämpötilat ajossa ovat normaalisti lähes 200 °C. Nämä erilaiset arvot ja niiden asteikot tuli sijoittaa samalle alueelle siten, että on niin sanottu kylmä alue, normaalialue ja kuuma-alue. Normaalialueen keskikohdassa on se piste johon kukin neula osoittaa silloin, kun kyseinen komponentti näyttää tavallista lämpötila-arvoaan kuormitettuna.



Kuva 7. Mittaristo kokonaisuudessaan

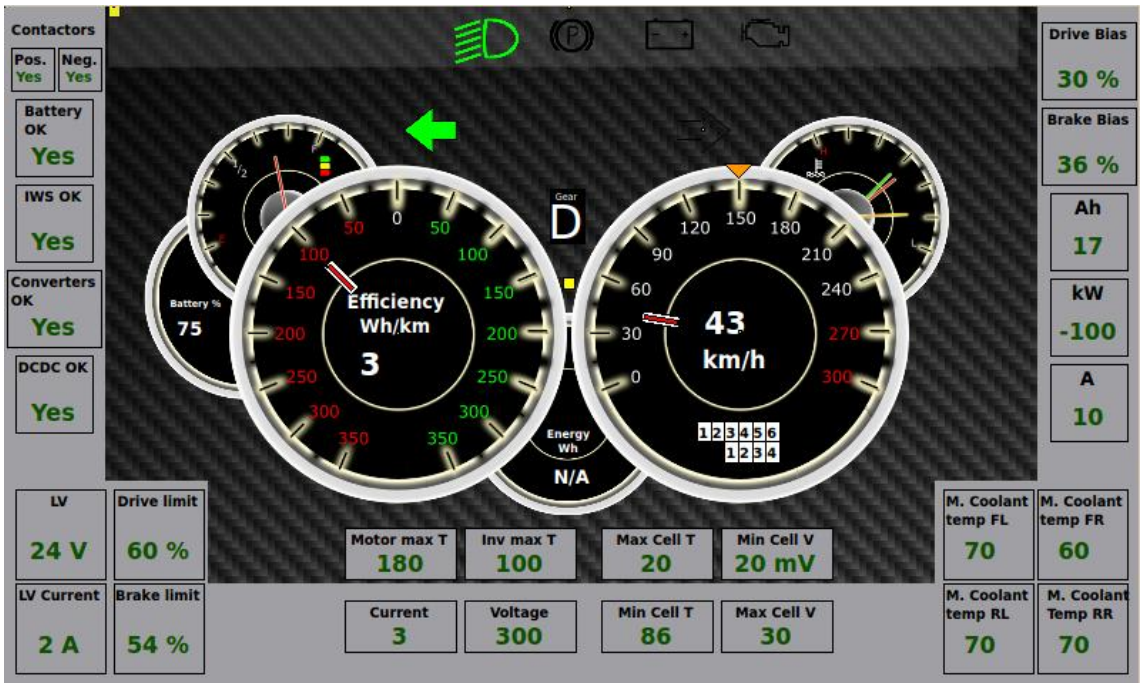
Ajoneuvon nopeus tuli ilmoittaa sekä graafisena analogisena mittarina että digitaalisena arvona. Tämä toteutettiin siten, että piirrettiin pyöreä mittaritaulu, johon asetettiin nopeuden ilmaisevat kohdistimet ja nopeusarvo. Pyöreän mittarin keskelle sijoitettiin tyhjä tila, johon digitaalinen arvo kirjoitetaan. Lisäksi aivan mittarin alalaitaan sijoitettiin katsastettavaan moottoriajoneuvoon pakolliseksi määrätty matkamittari, joka ilmaisee kuudella numerollaan, matkan joka autolla on kokonaisuudessaan ajettu.

Perinteisen moottorin pyörimisnopeuden ilmaisevan mittarin tilalle asetettiin tehonkulutusmittari, joka ilmaisee moottorien yhteenlasketun tehon kulutuksen tai tuoton. Tämän mittarin keskelle taas asetettiin numeerinen arvo ilmaisemaan ajoneuvon tehokkuutta eli, sitä kuinka paljon ajoneuvo kuluttaa tehoa kilometriä kohden. Tämä arvo ilmaisee kuinka suuren määrän energiaa ajoneuvo kuluttaa kilometrin aikana, kun yhden W:n tehon oletetaan kestävän tunnin ajan. Esimerkiksi jos laite kykenee toimimaan yhdellä W:lla tunnin ajan, kuluttaa se siis yhden Wh:n tunnissa.

Viimeisenä päämittarina toimii akun varauksen ilmoittava mittari. Mittarin neula osoittaa viisiportaisella asteikolla, kuitenkin täysin portaattomasti akun todellista varaustilaa tyhjän ja täyden välillä. Lisäksi tarkempi yhden desimaalin takkuinen numeerinen arvo ilmoitetaan aivan mittarin vieressä.

Mittariston ylälaitaan asetettiin läpikuultava palkki, jonka päälle sijoitettiin ajoneuvon varoitus- ja muut valot. Varoitusvaloina lueteltakoon moottorin vika-, käsijarru- ja käynnistyksenestovalo. Muina valoina taas on ajovaloista, kaukovaloista, ja sumuvaloista ilmoittavat kuvakkeet.

Viimeisimpänä käyttöliittymäkomponenttina on lisänäyttö (kuva 8), joka sisältää ylimääräistä tietoa jota ei aina tarvita. Lisänäyttö kutsutaan ruudulle ajoneuvoon sijoitetulla napilla joka antaa käskyn CAN-väylälle. Tällä käskyllä lisänäyttö tuodaan esiin tai kadotetaan näkyvistä. Lisänäyttö sisältää muun muassa virrankulutus, jännite ja lämpötilatietoja.



Kuva 8. Mittaristo, kun lisäpalkki on tuotu esiin.

4.1 Grafiikan näyttäminen näytöllä

Työssä CAN-väylältä saatu informaatio, kuten ajoneuvon nopeus, näytetään käyttäjälle graafisesti. Graafiseen kuvien luomiseen, näyttämiseen ja manipulointiin käytetään Qt-ohjelmiston QGraphicsView-widgetiä, joka on grafiikkaa sisältävä käyttöliittymäkomponentti. Työssä QGraphicsView-widgetti lukittiin koko ruudun alkupisteeseen (0,0). Tämä oli tärkeää, jotta näkymä pysyisi aina paikallaan ja komponenttien asettelu olisi helpompaa. Päänäkymän keskipiste asetettiin QGraphicsViewin keskipisteeseen. Tämä aiheutti sen että päänäkymän piste (0,0) sijaitsee QGraphicsViewin keskipisteessä (400,240). Näillä tiedoilla QGraphicsViewin ja päänäkymän liikuttelu toisiinsa nähden olisi mahdollista, mutta sen toteuttaminen toimivaksi veisi liikaa aikaa.

Esimerkiksi QGraphicsViewin koordinaatisto voitaisiin lukita ja päänäkymää voitaisiin siirtää tasomaisessa koordinaatistossa neljään suuntaan. Tällöin käyttäjä kokisi päänäkymän liikkuvan paikasta toiseen, kun päänäkymän keskipiste vain vaihtaisi paikkaansa. Mikäli näin tehtäisiin, tulisi ottaa huomioon, että kaikki komponentit säilyvät näkyvissä eikä mikään katoa näkyvän koordinaatiston ulkopuolelle. Tätä huomioitaessa tulee myös ryhmien liikuttelu mahdollistaa. Tämä tarkoittaisi myös sitä että käyttäjä kykenisi muokkaamaan näkymää sijaintien kannalta haluamakseen.

4.1.1 QGraphicsView-ikkuna

QGraphicsView eli grafiikannäyttöikkuna tarjoaa ominaisuudet päänäytön visualisoimiseen (esimerkkikoodi 1). Grafiikkanäyttöikkuna kutsuu päänäyttöä, joka voidaan näyttää joko kokonaisuudessaan tai vain osittain. Ikkunalle voidaan asettaa tietty koko, tai se voidaan jättää määrittämättä. Mikäli kokoa ei määritetä, saattaa se muuttua ohjelman aikana. Grafiikkanäyttöikkuna tukee vieritys-ominaisuutta, joten päänäkymää voi halutessaan liikutella sen sisällä kaikkiin suuntiin kaksiulotteisessa koordinaatistossa. Myös sisään- ja ulospäinliike onnistuu, skaalaamalla ikkunan kokoa. Tämä on kuitenkin parempi toteuttaa itse päänäytössä. [4.]

```
QGraphicsScene paanakyma;
paanakyma.addText("Hei, kaikki!");

QGraphicsView grafiikkanaytto(&paanakyma);
grafiikkanaytto.show();
```

Esimerkkikoodi 1. Päänäytön tuonti QGraphicsViewhin.

4.1.2 Päänäkymä

Päänäkymä eli scene ei itsessään näytä ruudulle mitään, se vain kokoaa visuaalisia komponentteja. Päänäytölle kerrotaan vain mitä sen tulee näyttää. Kun Grafiikkanäyttöikkuna pyytää päänäyttöä ilmentymään ruudulle, kaikki komponentit, jotka se tuntee, näytetään. Päänäyttö auttaa selvittämään tiettyjen komponenttien sijainnin ruudulla hyvin nopeasti koska, se ei säilö juuri muuta tietoa kuin elementin ja paikan. Näin ollen paikkatieto löytyy päänäytöstä muutamassa millisekunnissa, vaikka ruudulla olisi miljoonia komponentteja.

Päänäkymä on loputon kaksiulotteinen avaruus pisteitä, jolle on määritetty alkupiste. Tämä alkupiste on päänäkymän keskipiste, ellei sitä toisin määritetä. Määrittämällä alkupiste ja loppupiste, jolloin avaruus rajoitetaan tietylle välille, siirtyy keskipiste joksikin nelikulmion kulmaksi, riippuen loppupisteen koordinaattien etumerkeistä. (esimerkkikoodi 2). [5.]

```
scene->setSceneRect(0,0,screenresolution-
>bottomRight().x(),screenresolution->bottomRight().y());
```

Esimerkkikoodi 2. Päänäkymän koon määrittäminen.

4.1.3 Ryhmät

QGraphicsItemGroup koostaa erillisiä komponentteja yhteen ja kykenee käyttäytymään kuten vain yksi komponentti. Tällä tarkoitetaan sitä, että yhteen ryhmään voidaan lisätä monia erillisiä grafiikkaelementtejä ja niille voidaan tehdä manipulaatioita kuten yhdelle ainoalle elementille. Esimerkiksi vektorikuva ja pikselikuva voidaan liittää toisiinsa ja niitä voidaan pyörittää vain antamalla niiden ryhmälle komento pyörähtää tietty asteluku.

Ryhmien tarkoitus on myös helpottaa ohjelman lukua, kun aivan jokaista erillistä elementtiä ei lisätä päänäkymään erikseen. Kun elementit on ensin sijoitettu ryhmään, voidaan vain pelkkä ryhmä lisätä päänäkymään ja kaikki sen sisältämät elementit tulevat näkyviin (esimerkkikoodi 3). [6.]

```
gear_Group->addToGroup(gear_Rect);
gear_Group->addToGroup(gear_text);
gear_Group->addToGroup(gear_status_text);
```

Esimerkkikoodi 3. Ryhmään lisääminen.

4.1.4 Vektorielementti

QGraphicsSVGItem on tapa käsitellä vektorikuvia Qt:ssa. Vektorikuva renderöidään ensin QSVGRenderiin, josta sitä voidaan jakaa usealle elementille. Näin toimittaessa ohjelma tulee toteuttaa siten, että molemmat sekä elementti että renderi elävät yhtä kauan. Mikäli renderi tuhotaan ennen elementtiä, menettää elementti kuvansa.

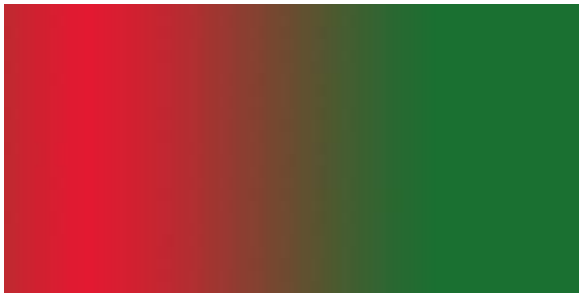
Yksi vektorikuva voi sisältää monta eri elementtiä, kuten vaikka kaikki 52 korttipakan korttia. Siksi jokainen erillinen kortti kuvan sisällä sisältää elementtikohtaisen tunnisteen. Tätä tunnistetta käyttämällä QGraphicsSVGItem voi valita, minkä kortin se näyttää korttipakasta. Lisäksi tunnistetta voidaan vaihtaa kesken ohjelman, jolloin kuva vaihtuu (esimerkkikoodi 4). [7.]

```
QSvgRenderer *renderi = new QSvgRender-
er(QLatin1String("korttipakka.svg"));
     QGraphicsSvgItem *musta = new QGraphicsSvgItem();
     QGraphicsSvgItem *punainen   = new QGraphicsSvgItem();

     musta->setSharedRenderer(renderer);
     musta->setElementId(QLatin1String("risti_assa"));

     punainen->setSharedRenderer(renderer);
     punainen->setElementId(QLatin1String("hertta_kuningas"));
```

Esimerkkikoodi 4. Vektorielementti ja renderöinti.

4.1.5 Nelikulmio- ja monikulmioelementti

QGraphicsRectItem muodostaa nelikulmion, jonka alkupiste ja loppupiste voidaan määrittää. QGraphicsPolygonItem taas muodostaa monikulmion, jolla voi olla lukematon määrä sivuja ja kulmia. Molemmille elementeille voidaan asettaa väritys sekä ääriviivoille että niiden sisälle. Lisäksi on mahdollista luoda QPainter, jolla voidaan luoda erilaisia erikoisvärityksiä kuten liukuväritys. Liukuvärityksellä tarkoitetaan värin vaihtumista toiseen, kun tietty piste ohitetaan (kuva 9). [8.]



Kuva 9. Nelikulmion liukuväritys.

4.2 Ohjelman rakenne

Graafinen ohjelma rakentuu päänäytöstä eli scenestä. GraphicsView-ikkuna näyttää vain päänäytön. Päänäyttö rakennetaan pienemmistä ryhmistä eli groupeista. Ryhmät lisätään päänäkymään jolloin niistä tulee niin sanotusti päänäkymän lapsia (esimerkkikoodi 5). Tällainen rakennemalli mahdollistaa sen, että yksi lapsista voidaan tuhota, se voi kuolla tai se voidaan korvata ilman että koko päänäyttö pitää luoda uudestaan. Ryhmät koostuvat vielä pienemmistä palikoista, kuten QGraphicsSvgItem ja QGraphicsRectItem.

```
/// mainwindow.cpp – CPP-tiedosto
// ryhmien päänäkymään lisäys
scene->addItem(mittaristo_->SpeedOMeter_Group);

/// mittaristo.cpp – CPP-tiedosto
// ryhmään lisäys
SpeedOMeter_Group->addToGroup(SpeedOMeter_BG);
```

Esimerkkikoodi 5. Ryhmän luonti ja päänäyttöön lisääminen.

Itse ohjelma on rakennettu siten, että tämä yksi päänäkymä koostuu oikeasti kahdesta näkymästä, mutta toinen niistä on piilotettu. Tämä piilotettu näkymä paljastetaan vain tietyllä komennolla, joksi on suunniteltu fyysistä nappia autossa. Tällä napilla nostetaan tietty lippu CAN-väylällä, ja kun ohjelma saa lipun nostosta tiedon paljastaa se piilotetun näkymän käyttäjälle. Kun lippu lasketaan alas, piilottaa ohjelma lisänäkymän.

Ohjelman toiminta koostuu kahdesta erillisestä säikeestä. Säikeet suorittavat erillisiä toimintoja, kuitenkin siten että pääsäikeellä on mahdollisuus puuttua orjasäikeen prosessin suoritukseen niin halutessaan. Orjasäikeen on tarkoitus lukea CAN-väylää ja kirjoittaa tietoa niin sanottuun puskuriin, jolla tarkoitetaan tiedon väliaikaista säilytystilaa. Pääsäie kykenee hakemaan puskurista tarvitsemansa tiedon, mutta se ei muokkaa siellä mitään. Pääsäie taas kirjoittaa hakemansa informaation käyttäjälle luettavaan muotoon näytölle.

4.2.1 Perintä

Työssä pääsäie perii sekä orjasäikeen että myös grafiikkaa luovat luokat mittariston ja lisäinformaation. Perintä tapahtuu yksinkertaisesti lisäämällä perittävän luokan tieto perijälle ja luomalla olio ohjaamaan liikennettä näiden luokkien välillä. Tällaisessa C++ ohjelmoinnissa perijä eli vanhempi tuntee perittävän eli lapsen, mutta lapsella ei ole valtaa vanhemman sisältämiin funktioihin. Tämä myös estää turhaa moninkertaistusta ohjelmassa sekä helpottaa laajan kokonaisuuden hallintaa.

Tuntemisella tarkoitetaan, että lapsen julkisiksi määrittelemät funktiot ovat täysin avoimia myös vanhemmalle. Toki lapsella on myös mahdollisuus pitää funktioita niin sanotusti yksityisinä, jolloin vanhemmalla ei ole oikeutta suorittaa niitä. [9. s. 137.]

4.2.2 Päänäkymän luominen

Päänäkymä siis luodaan ohjelman pääluokassa. Päänäkymässä aliluokan tietoja kutsutaan käyttämällä osoitinta kertomaan ohjelmalle, mistä kyseinen komponentti haetaan (esimerkkikoodi 6).

```
/// mainwindow.h - Header tiedosto
#include "scene/mittaristo.h"
public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
   mittaristo *mittaristo_;


/// mainwindow.cpp - CPP-tiedosto
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow),
{
    mittaristo_ = new mittaristo();
   scene->addItem(mittaristo_->SpeedOMeter_Group);
}
```

Esimerkkikoodi 6. Ensin header-tiedostoon luodaan osoite aliluokkaan eli luokka peritään. Toiseksi osoitteesta tehdään toimiva ottamalla se käyttöön cpp-tiedostossa. Nyt osoitetta voidaan käyttää noutamaan aliluokassa muodostettuja ryhmiä.

Suurimmalla osalla ohjelman komponenteista on oma ryhmänsä, joten tämä toiminto toistetaan jokaista ryhmää varten uudestaan. Eri luokassa luotuja ryhmiä varten tulee luoda oma osoitteensa.

4.3 Grafiikan käyttö

Tässä työssä käytetty kuvat ja kuvakkeet sisältävät sekä vektori- että pikseligrafiikkaa. Molempia grafiikoita käytetään sekä päällekkäin että erikseen. Suurimpana esimerkkinä voidaan ottaa nopeusmittari jonka taustakuva on piirretty vektorina ja nopeutta ilmaiseva nuoli on piirretty pikseleinä ohjelman sisäisesti. Nopeusmittarin päälle on asetettu myös tekstitietoa (kuva 10). Tekstitietoa sisältävät kuvakkeet luodaan Qt:n sisäisellä luokalla QGraphicsSimpleTextItem, johon voidaan kirjoittaa näppäimistöltä löytyviä merkkijonoja. Nämä merkkijonot ovat tässä ohjelmassa muotoa QString, ja niihin tuodaan informaatio numeerisista muuttujista.

Kuva 10. Ohjelman nopeusmittari. Oranssi kolmio kuvaa vakionopeudensäätimen ei aktiivista arvoa. Mikäli arvo olisi aktiivinen muuttuisi kolmion väri vihreäksi, ja ajoneuvo pyrkisi pitämään tämän nopeuden.

4.3.1 Vektorigrafiikka

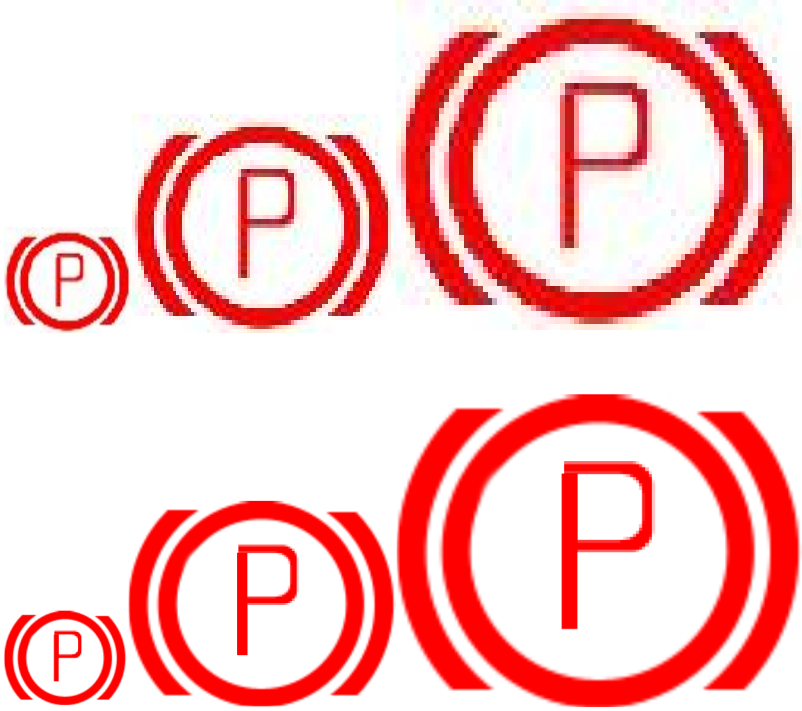
Vektorigrafiikka on tietokoneella tuotettua kuvaa, joka perustuu elementteihin, joista kuva lopulta koostuu. Elementit ovat hyvin yksinkertaisia, kuten suoria, nelikulmioita, ympyröitä, ellipsejä tai monikulmioita. Elementit sisältävät matemaattista ja metadata-tyyppistä tietoa. Esimerkiksi suora sisältää paikkatietoina lähtöpisteen ja loppupisteen sekä metadatana värikoodin ja mahdollisia muotoilullisia määrittelyjä. Tällaisia elementtejä yhdisteltäessä muodostuu lopulta kuva, jota voidaan suurentaa loputtomasti ilman, että kuvan tarkkuus muuttuu alkuperäisestä.

4.3.2 Pikseligrafiikka

Pikseligrafiikka toisin kuin vektorigrafiikka menettää tarkkuuttaa kuvaa lähennettäessä (kuva 11), koska pikseligrafiikka koostuu pikselikoordinaateista, jotka sisältävät värikoodin. Kun pikseleitä on tarpeeksi vierekkäin sekä pysty- että vaakasuunnassa, muodostuu kuva. Näin ollen kuvaa lähennettäessä kaksinkertaiseksi joudutaan yhden pikselin värikoodi kopioimaan kolmeen viereiseen pikseliin. Kun kuvaa on suurennettu tarpeeksi, ei ihminen kykene enää erottamaan kuvasta haluttuja muotoja ja kuvan tarkoitus häviää. Pikseligrafiikkaa voidaan joko piirtää tai luoda ohjelman sisäisesti. Piirtäen pikseligrafiikkasta on helpompi luoda näyttävää ja yksityiskohtaista. Pikseligrafiikka piirtäen luodaan samalla lailla kuin vektorigrafiikka, paitsi että se tallennetaan pikselimuottoon. Ohjelmassa luodut pikseligrafiikat taas vaativat huomattavan määrän informaatiota ennen kuin niistä muodostuu edes perinteisiä kuvioita, kuten kolmioita (esimerkkikoodi 7), neliöitä tai suorakulmiota.

```
/// mittaristo.cpp - CPP-tiedosto

QPolygonF Triangle;
        Triangle.append(QPointF(*scale_*20.,0));
        Triangle.append(QPointF(0.,*scale_*20));
        Triangle.append(QPointF(*scale_*-20.,0));
         Triangle.append(QPointF(*scale_*20.,0));


cruise_pointer = new QGraphicsPolygonItem(Triangle);

cruise_pointer->setPos(0*(*scale_),-247*(*scale_));
cruise_pointer->setBrush(QColor::fromRgb(255,150,0));
cruise_pointer->setZValue(15);
cruise_pointer->setOpacity(1);

cruise_Group->addToGroup(cruise_pointer);

/// mittaristo.h - header tiedosto

QGraphicsItemGroup *cruise_Group;
QGraphicsPolygonItem *cruise_pointer;
QGraphicsRectItem *cruise_Anchorpoint;


/// mainwindow.cpp – CPP-tiedosto

// kutsutaan mittaristo näkyviin scenessä

scene->addItem(mittaristo_->cruise_Group);
```

Esimerkkikoodi 7. Näin monta riviä ohjelmaa vaaditaan, että luodaan yksivärinen kolmio ruudulle tiettyyn paikkaan.



Kuva 11. Pikseligrafiikan ja vektorigrafiikan ero kuvaa lähennettäessä

4.3.3 Tekstigrafiikka

Tekstigrafiikka luodaan kokonaan ohjelman sisäisesti, ja tämän vuoksi sen sisältöä on helppo muokata. Tekstigrafiikka on merkkijonoja tuotuna näytölle kuvan muodossa. Esimerkkinä voidaan katsoa, kuinka teksti syntyy näytölle Word-tekstinkäsittelyohjelmassa. Jokainen merkki sisältää tiedon ympäröivästä kehyksestään. Kun merkkejä on monta, niiden kehykset asetetaan vierekkäin siten, että aiemman merkin loppulaita on päällekkäin seuraavan merkin aloituslaidan kanssa. Näin lopulta syntyy haluttu sana, numeroyhdistelmä tai vain jono merkkejä, joiden yhteenlaskettu pituus tiedetään. Tässä ohjelmassa tekstigraafiset kuvat tuli sijoittaa aina tiettyihin koordinaatteihin keskitetysti, joten merkkijonojen pikselien lukumäärällä x- ja y-suunnassa oli merkitystä. Kun tiedettiin tilan ja merkkijonon pituus pikseleinä voitiin merkkijono keskittää haluttuun tilaan (esimerkkikoodi 8).

```
/// mittaristo_lisapalkki.h – header-tiedosto
QGraphicsSimpleTextItem *motor_temp_max_text;


/// mittaristo_lisapalkki.cpp – CPP-tiedosto
motor_temp_max_text = new QGraphicsSimpleTextItem(0,0);
    motor_temp_max_text->setText("Motor max T");
    motor_temp_max_text->setFont(font1);
    motor_temp_max_text->scale(1.2*(*scale_),1.2*(*scale_));
    qreal motor_temp_max_text_width =
1.2*(*scale_)*motor_temp_max_text->boundingRect().bottomRight().x();
    motor_temp_max_text->setPos((130*(*scale_)-
motor_temp_max_text_width)/2,5*(*scale_));
    motor_temp_max_text->setOpacity(100);
    motor_temp_max_text->setBrush(textbrush);
    motor_temp_max_text->setZValue(7);


/// mainwindow.cpp – CPP-teidosto
qint16 motortemp_max_ = canDataEra_->getDataMotorTempMax();
QString motor_text_;
motor_text_.setNum(motortemp_max_);
mittaristo_lisapalkki_->motor_temp_max_status_text-
>setText(motor_text_);
```

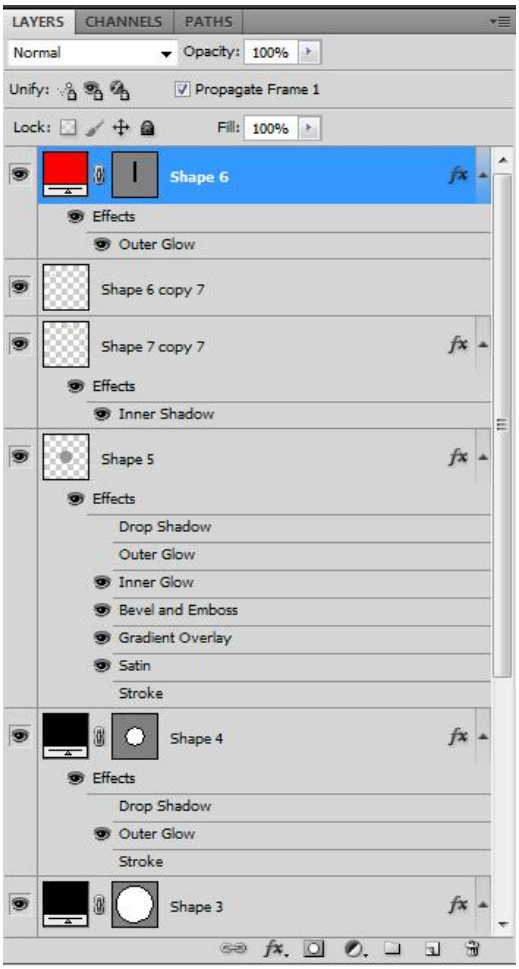
Esimerkkikoodi 8. QgraphicsSimpleTextItemin luonti sekä informaation muokkaaminen numeroista merkkijonoksi.

5 Vektorikuvan luonti

Tämän ohjelman kaikki vektorigrafiikka on piirretty työn tekijän toimesta Adoben tarjoamilla työkaluilla Photoshop CS5 ja Illustartor CS5. Ensin kuvat piirrettiin Photoshopin tarjoamilla graafisen suunnittelun työkaluilla. Nämä kuvat sisälsivät layereitä (kuva 12) eli tasoja, jotka päällekkäin aseteltuna muodostivat halutun näköisen kuvan.

Vektorikuvia tämän tyyppiseen ohjelmaan luotaessa tulee ne suunnitella siten, että niiden pyörimiskeskipiste on tiedossa. Pyörimiskeskipiste ei aina ole sama kuin kuvan keskipiste. Tässä ohjelmassa mittariston neulakomponentit sisältävät eri pyörimiskeskipisteen, kuin mikä on niiden oikea keskipiste. Pyörimiskeskipiste on siis se piste jonka mukaan kuvan on tarkoitus pyöriä, ettei se lähde katsojan silmin pyörimään ulkopuolisen pisteen mukaan. Toisin sanottuna pieni virhe pyörimiskeskipisteen asettamisessa aiheuttaa vain pienen värinän liikkeessä, mutta suurempi virhe saa kuvan siirtymään paikaltaan.

Tasoista koostuva valmis kuva tuodaan Adobe Illustartor -ohjelmaan jossa kaikki tasot yhdistetään yhdeksi ryhmäksi. Yhdistämisen jälkeen kuva käännetään vektorimuotoon tallentamalla se SVG 1.1 -standardin mukaiseen muotoon. Nyt kuvaa voi loitontaa ja lähentää ilman sen tarkkuuden menettämistä.



Kuva 12. Erään kuvan tasopuu.

6 Tiedon kulku väylältä mittaristoon

Ajoneuvon anturien ja toimilaitteiden jakama informaatio kulkee CAN-väylää pitkin tarvittavaan paikkaan autossa. Esimerkiksi kaasupolkimen asentotieto kuljetetaan joissain autoissa väylää pitkin moottorinohjausyksikköön joka, arvon saatuaan käskee moottoria ottamaan enemmän polttoainetta tai sähkötehoa riippuen ajoneuvon käyttövoimasta. Mittaristo-ohjelma on ajoneuvossa osana väylää, mutta se vain lukee muiden käskyjä ja ilmoituksia. Nämä informaatiopalaset se ohjaa näytölle käyttäjän nähtäväksi.

6.1 Säikeiden välinen kommunikointi

Tässä työssä säikeet suorittavat erityyppisiä prosesseja. Pääsäie on niin sanotusti visuaalista grafiikkaa tuottava säie, kun taas orjasäie tuottaa informaation, jota grafiikalla pyritään näyttämään käyttäjälle.

Orjasäie lukee ja varastoi väylällä liikkuvaa informaatiota puskuriin. Mikäli tieto on puskurissa liian kauan, se tuhotaan ja korvataan uudella, vaikka sitä ei olisi ehditty lukea pääsäikeeseen. Pääsäie lukee puskuria tietyin väliajoin. Osaa tiedoista luetaan useammin kuin toisia, esimerkiksi lämpötila-arvon kasvaminen sopivalla määrällä kuten asteella kestää huomattavasti pitempään kuin ajoneuvon liikenopeuden muutos yhdellä kilometrillä tunnissa. Tärkeää on, että kaikki informaatio on sopivan ajantasaista. Koska autossa on hyvin paljon tietoa jota täytyy lukea, pitää hitaammin päivittyviä tietoja lukea harvemmin, jotta nopeammin päivittyvät tiedot pysyvät ajan tasalla.

CAN-väylällä liikkuvien tietojen suurin sallittu koko on 64 bittiä. Viestit erotellaan toisistaan tunnisteiden avulla. Jokainen laite lisää viestiinsä oman tunnisteen, jonka avulla viestit kulkeutuvat oikeaan paikkaan. Koska suurin osa tiedoista ei tarvitse koko 64 bittiä, on niitä yhdistelty siten, että kyseinen raja ei kuitenkaan ylity. Nämä yhdistelyt ovat valmiiksi tiedossa ennen ohjelmointia, joten ohjelmoijan ei tarvitse kuin asettaa ohjelma aloittamaan yksittäisen tiedon luku tietystä pisteestä ja lopettamaan se toiseen tiettyyn pisteeseen. Kaiken kaikkiaan informaatio voi olla hyvinkin lyhyttä kuten vain 1 bitin kokoista tai se saattaa viedä kaikki 64 bittiä. Esimerkkeinä voidaan mainita, että yhden bitin kokoisia tietoja ovat yleensä kaikki päällä- ja pois-tiedot. Kaikkein suurimpia ja eniten bittejä vaativia tietoja ovat fysikaaliset analogiset tiedot kuten jännitteen tai lämpötilan arvo.

6.2 Toteutus

CAN-väylän lukeminen tapahtuu ohjelman toisessa säikeessä tietyin väliajoin. Tätä säiettä kutsutaan orjasäikeeksi, koska se suorittaa toiminnallisuuksia, joita ensimmäinen eli pääsäie tarvitsee toimiakseen. Koska pääsäie ei yksin kykene lukemaan väylää ja päivittämään informaatiota ruudulle tarpeeksi nopeasti, on sille luotu apulaiseksi orjasäie, joka suorittaa pääsäikeen kanssa samanaikaisesti toimintoja. Eri säikeet voivat siis prosessorista riippuen suorittaa toimintojaan samanaikaisesti. Esimerkiksi mikäli prosessori sisältää useita eri fyysisiä ytimiä, voidaan jokaisessa ytimessä suorittaa yhtä säiettä koko ajan. Lisäksi useat prosessorimallit kykenevät jakamaan fyysisen ytimensä kahdeksi virtuaaliseksi ytimeksi. Tällöin jokaisessa virtuaaliytimessä voidaan suorittaa yhtä säiettä. Tällainen menettely mahdollistaa säikeiden määrän kaksinkertaistamisen fyysisten ytimien mukaan.

Koska tiedossa oli mittaristoa suorittavien laitteiden suorituskyvyn rajallisuus, toteutettiin ohjelma siten, että siinä on vain kaksi säiettä. Kahden säikeen suorittaminen yksiytimisellä virtuaaliytimiä tukemattomalla prosessorilla on mahdollista ilman, että suorituskykyä oleellisesti menetetään.

Säikeet toteutettiin siten, että pääsäie perii orjasäikeen. Tällä tavoin pyrittiin keventämään ohjelman järjestelmän suorituskyvyn kulutusta, sillä nyt pääsäie kykenee tarttumaan orjasäikeen informaatioon ilman, että orjasäie lähettää sitä pääsäikeelle.

Pääsäiettä suoritettaessa siinä kutsutaan funktioita toisesta säikeestä. Nämä funktiot sisältävät vain informaation halutusta laitteesta tai anturista, ja palauttavat sen pääsäikeeseen. Pääsäikeessä tieto otetaan vastaan paikalliseen muuttujaan, joka muokkaa arvoa haluttuun yksikköön ja asettaa sen tietokoneen näytölle näkyviin analogisen näköisenä mittarina tai digitaalisena numeerisena arvona.

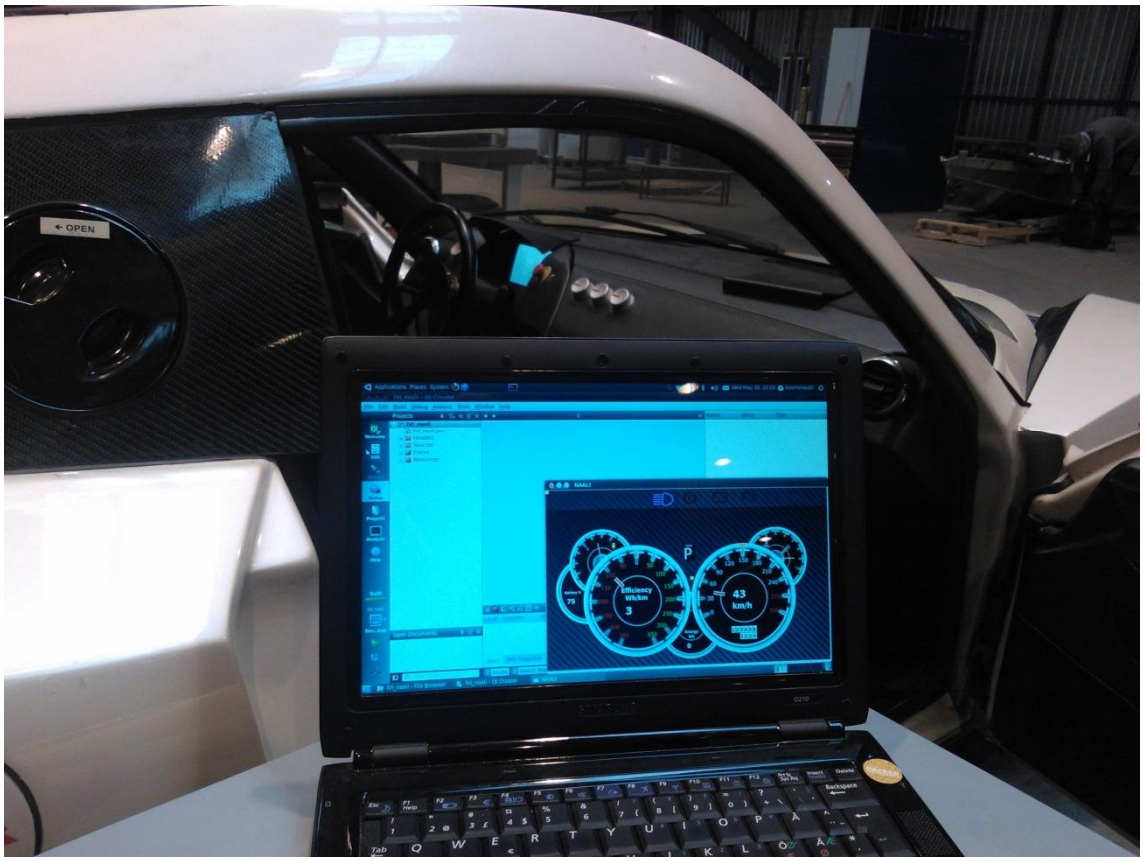
7 Mittariston testaus ja käyttöönotto

7.1 Testaus

Mittariston testaus aloitettiin siten, että toiselta tietokoneelta lähetettiin CAN-viestejä jotka vastaavat auton ohjainlaitteiden lähettämiä. Näiden viestien avulla testattiin saadaanko tieto CAN-väylältä mittaristoon. Ensitestauksella kaikki tuntui toimivan hienosti, eikä suuria korjauksia tehty.

Ensitestauksen jälkeen, lähdettiin mittaristoa kokeilemaan autossa (kuva 13). Kun tietokone oli kytketty ajoneuvon CAN-väylään, huomattiin, että väylää luettaessa ohjelma käsittelee vain uusimman viestin 50 millisekunnin välein. Tämän vuoksi suurin osa viesteistä jäi lukematta eikä mittaristo toiminut halutulla tavalla.

Pikaisten korjausten jälkeen mittaristoa testattiin ajoneuvossa vielä uudelleen. Nyt ongelmana oli että datan parsinta tehtiin väärin ja tieto ei päätynyt perille mittaristoon oikean laisena.



Kuva 13. Tietokone autoon liitettynä.

7.2 PC:n asetukset ja valmistelu

ERAn vanhaan PC-laitteeseen yritettiin tuoda uutta mittaristoa, mutta se ei onnistunut. Qt:lla Ubuntussa luotua mittaristo-ohjelmaa ei saatu käännettyä sellaiseen muotoon että Windows CE 5.0 olisi sitä voinut lukea.

ERAn toiseen PC-laitteeseen mittaristo saatiin toimimaan. Ohjelmaan ei kuitenkaan ehditty rakentamaan tukea laitteen CAN-väylän ajurille, vaan se saatiin käännettyä Ångström-käyttöjärjestelmälle ja ARM-prosessorille ymmärrettävään muotoon (kuva 14).

Kuva 14. Mittaristo käännettynä Ångström-käyttöjärjestelmälle ja ARM-prosessorille.

7.2.1 Ubuntun asetukset

Ubuntu-käyttöjärjestelmään, jolla ohjelma luotiin jouduttiin asentamaan niin sanottu ristiinkääntäjä (cross-compiler). Tämä kääntäjä kykenee kääntämään ohjelman toiselle prosessorityypille ja käyttöjärjestelmälle sopivaksi. Ensiksi varmistuttiin, että uusin Qt Creator -työkalu oli asennettu tietokoneelle. Toiseksi Ångströmin virallisilta internetsivulta ladattiin kohdelaitteelle sopiva työkaluketju (toolchain), jonka avulla Qt:n kirjastot käännettiin kohdelaitteelle sopiviksi (ohje 1). Kirjastojen kääntäminen voi kestää jopa tunteja. Kirjastojen valmistuttua uusi kääntäjä asetetaan Qt Creatoriin (kuva 15). Tämän avulla ohjelmasta saadaan käännettyä toimiva versio kohdelaitteelle. [10.]

Lataa viimeisin Qt Creator tietokonneele.
Lataa sopiva Ångström-toolchain ja pura se komennolla:

```
sudo tar –xvj –C / -f [ladatun tiedoston nimi]
```

Lataa Qt:n vapaan lähdekoodin kirjastot sulautetuille laitteille ja pura haluaamaasi polkuun:

```
tar –xvzf qt-everywhere-opensource-src-X.X.X.tar.gz
```

Luo uusi make.conf –tiedosto:

```
cp –R [latauskansio]/mkspecs/qws/linux-arm-g++/ [latauskansio]/mkspecs/qws/linux-DM3730-g++/
```

Korvaa uuden tiedoston tiedot seuraavilla:

```
#
# qmake configuration for building with arm-linux-g++
#
include(../../common/g++.conf)
include(../../common/linux.conf)
include(../../common/qws.conf)
# modifications to g++.conf
#Toolchain
#Compiler Flags to take advantage of the ARM
architecture
QMAKE_CFLAGS_RELEASE =   -O3 -march=armv7-a -
mtune=cortex-a8 -mfpu=neon -mfloat-abi=softfp
QMAKE_CXXFLAGS_RELEASE = -O3 -march=armv7-a -
mtune=cortex-a8 -mfpu=neon -mfloat-abi=softfp
QMAKE_CC = /usr/local/angstrom/arm/arm-angstrom-linux-
gnueabi/bin/gcc
QMAKE_CXX = /usr/local/angstrom/arm/arm-angstrom-linux-
gnueabi/bin/g++
QMAKE_LINK = /usr/local/angstrom/arm/arm-angstrom-linux-
gnueabi/bin/g++
QMAKE_LINK_SHLIB = /usr/local/angstrom/arm/arm-angstrom-
linux-gnueabi/bin/g++
# modifications to linux.conf
QMAKE_AR = /usr/local/angstrom/arm/arm-angstrom-linux-
gnueabi/bin/ar cqs
QMAKE_OBJCOPY = /usr/local/angstrom/arm/arm-angstrom-
linux-gnueabi/bin/objcopy
QMAKE_STRIP = /usr/local/angstrom/arm/arm-angstrom-
linux-gnueabi/bin/strip
load(qt_config)
```

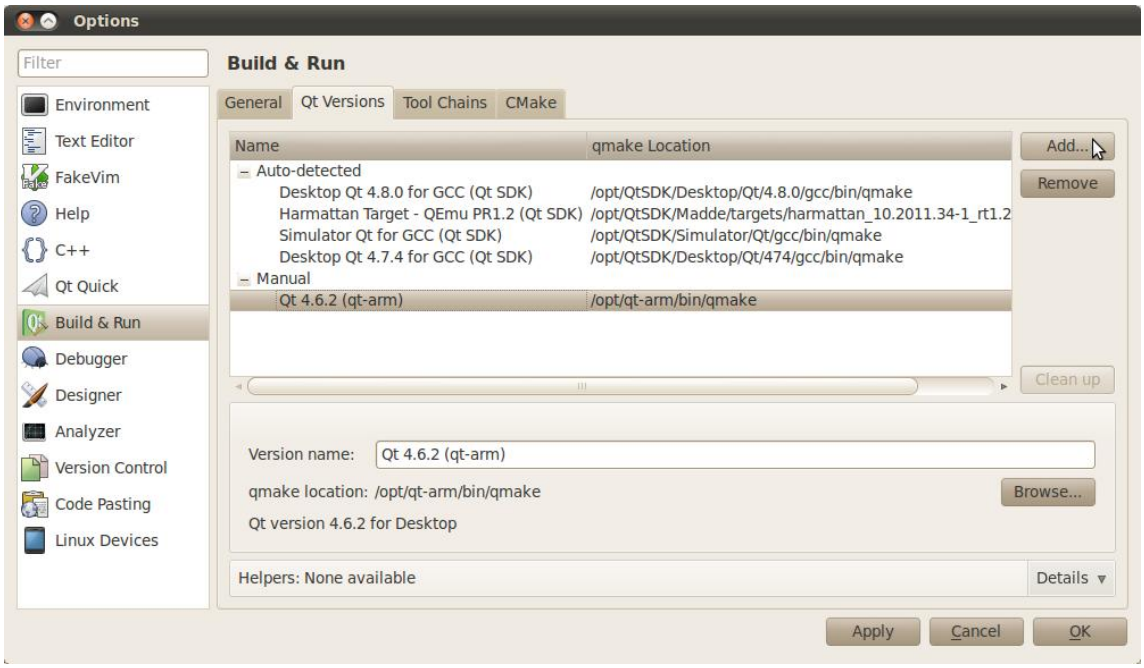
Latauskansiossa suorita seuraava kometonasetusten määrittämiseksi:

```
./configure -opensource -confirm-license -prefix
/opt/qt-arm -no-qt3support -embedded arm -little-
endian -xplatform qws/linux-DM3730-g++ -qtlibinfix
E
```

Viimeiseksi kirjastot käännetään ja asennetaan:

```
make
make install
```

Ohje 1. Valmistuslaitteen valmistelu ohjelman kääntämistä varten.



Kuva 15. Kääntäjän asettaminen Qt Creatoriin.

7.2.2 Kohdelaitteen asetukset

Qt vaatii kirjastojen sijaitsevan samassa paikassa niin kohdelaitteessa, kuin valmistuslaitteessakin, joten kohdelaitteeseen täytyy luoda samaan polkuun saman niminen kansio, johon kopiodaan kaikki käännetyt kirjastot. Tätä toimenpidettä varten kannattaa varata Ext3-tiedostojärjestelmällä alustettu muistitikku, koska muille tiedostojärjestelmille ei voida kopioida Linuxin linkitettyjä tiedostoja. Toiseksi kohdelaitteelle kerrotaan mistä kirjastot löytyvät, eli sen järjestelmään lisätään tieto polusta, mihin kirjastot on sijoitettu. Viimeisenä toimenpiteenä varmistutaan, että kaikki tarvittavat laajennukset on ladattu ja asennettu kohdelaitteeseen. Kun kaikki asetukset ovat molemmissa laitteissa kunnossa, voidaan ohjelma tuoda kohdelaitteeseen ja suorittaa siinä (ohje 2). [10.]

Koska kirjastot käännettiin valmistuslaitteessa polkuun /opt/qt-arm luodaan sama polku kohdelaitteeseen:

```
cd  /
mkdir opt/
cd opt/
mkdir qt-arm/
cd qt-arm/
mkdir lib/
```

Kopioidaan valmistuslaitteen /opt/qt-arm/lib kansion kaikki tiedostot kohdelaitteen samaan polkkun.

Seuraavaksi /etc kansiossa sijaitsevaan teidostoon lisätään kirjastojen osoite:

```
PATH=”/opt/qt-arm/lib/:….
```

Päivitetään laitteen kirjastomanageri ja asennetaan tarvittavat lisäosat:

```
opkg update
opkg install libstdc++6
opkg install libpng12-0
```

Kun kaikki on valmista valmistuslaitteessa käännetty ohjelma voidaan tuoda kohdelaitteeseen ja suorittaa komennolla:

```
./[ohjelman nimi] -qws
```

Ohje 2. Kohdelaitteen astukset, jotta ohjelmaa voidaan suoritaa.

7.2.3 Ohjelman automaattinen käynnistys

Ohjelma voidaan asettaa käynnistymään aina järjestelmän käynnistyksen yhteydessä. Tämä on myös toivottua, kun tietokonetta käytetään mittaristona ajoneuvossa. Ångström-käyttöjärjestelmässä ohjelma asetetaan käynnistymään järjestelmän käynnistymisen kanssa saman aikaisesti, luomalla ohjelmistolle käynnistysskripti ja lisäämällä skriptin polku inittab-tiedostoon (ohje 3). Inittab-tiedosto luetaan aina käynnistyksen yhteydessä, jolloin skripti käynnistää halutun ohjelman. Lisäksi inittab-tiedosto varmistaa myös sen, että ohjelma suoritetaan uudelleen, mikäli se jostain syystä keskeytyy. [11.]

Luo uusi tiedosto ja anna sille nimeksi autorun.sh.

Lisää seuraavat rivit tiedostoon:

```
#!/bin/sh
export TSLIB_TSEVENTTYPE=INPUT
export TSLIB_CONSOLEDEVICE=none
export TSLIB_FBDEVICE=/dev/fb0
export TSLIB_TSDEVICE=/dev/input/event1
export TSLIB_CALIBFILE=/etc/pointercal
export TSLIB_CONFFILE=/etc/ts.conf
export TSLIB_PLUGINDIR=/usr/lib/ts
export QWS_MOUSE_PROTO=tslib:/dev/input/event1
export TSLIB_TSDEVICE=/dev/input/touchscreen0
export
QWS_KEYBOARD=linuxinput:/dev/input/event2:keymap=fr.qmap
export QWS_SIZE=800x600
export QWS_DISPLAY="linuxfb:mmHeight=85:mmWidth=145"
export HOME=/home/root
cd /home/root
#./[ohjelman nimi] -qws < /dev/null 2>&1 1>/dev/ttySAC0
./[ohjelman nimi] -qws < /dev/null 2>&1 | tee -a lo-
gall.txt 2&>1 1>/dev/ttySAC0
```

Siirrä tiedosto kansioon:

```
 /usr/bin/
```

Lisää seuraava rivi /ect/inittab –tiedostoon:

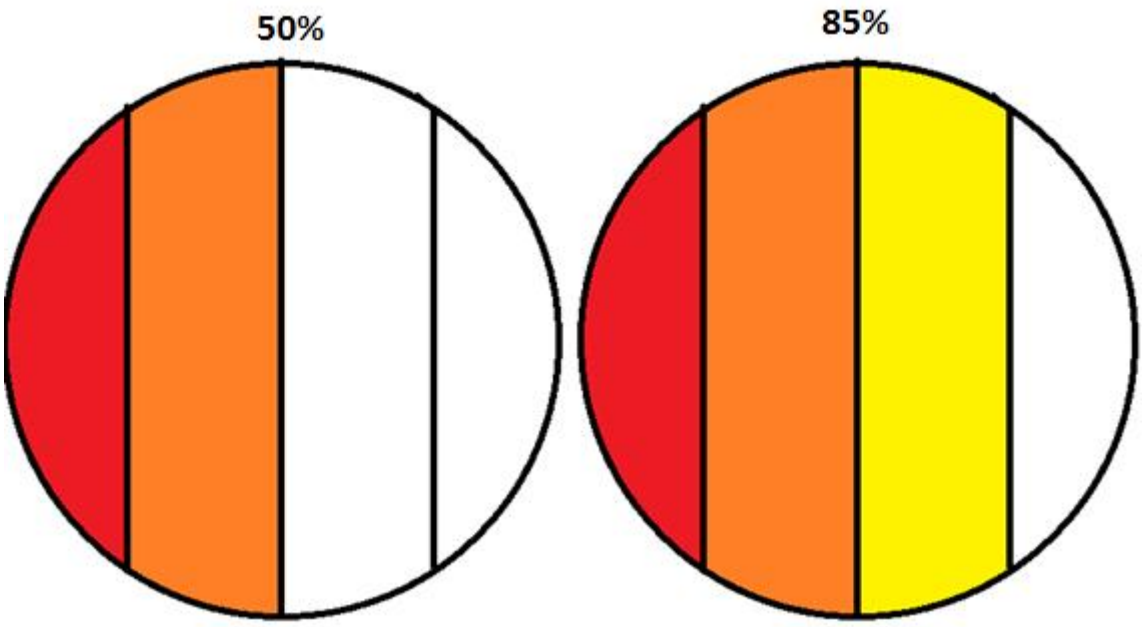
```
k:2345:respawn:/usr/bin/autorun.sh
```

Ohje 3. Käynnistysskriptin luominen ja järjestelmään asetus.

8 Yhteenveto

Insinöörityön tuloksena saatiin lähestulkoon toimiva pohja kaupunkiauton mittaristotarpeita varten, sekä uudistettu mittaristo jo mainetta niittäneen ERA-sähköauton käyttöön. Tämän insinöörityön pohjalta voi uusi henkilö jatkaa mittariston kehittämistä, ja muokata sen ulkoasua ja toimintaa haluamakseen. Tähän projektiin haasteita loi erityisesti kahden erilaisen järjestelmän mahdollisimman hyvä yhteensovittaminen. Uusien tietojen tuominen CAN-väylältä pyrittiin tekemään mahdollisimman helpoksi luomalla yleinen informaation käsittelijäluokka, joka ei välitä keneltä tieto tulee vaan siirtää saapuneet tiedot tunnuksen perusteella haluttuun paikkaan.

Mittariston ulkoasu on tämän yön tekijän näkökulma, siitä miltä mittariston tulisi näyttää. Ulkoasua on helppo muokata haluamakseen vain muuttamalla kuvaketta. Myös uudenlaisia informaation esitystapoja voi kuka tahansa keksiä, esimerkiksi polttoaineen määrä tai akun varaus voidaan esittää myös suoralla, käyrällä tai ympyrällä joiden muodostamia muotoja täytetään funktioiden perusteella. Esimerkiksi vaikka ympyrän sisältöä voisi värittää tietyllä värillä polttoaineen määrän prosentuaalisen arvon mukaan (kuva 16).



Kuva 16. Eräs tapa kuvaamaan käyttövoiman määrää.

Vaikka aivan kaikkea ei saatukaan tämän työn puitteissa toimimaan, saatiin valmiiksi hyvä pohja tulevaa kehitystä varten. Kaupunkiautoon mittaristoa vietäessä uudella ohjelmoijalla on haasteenaan korvata ERAn mittaristoon sovitetut arvot dieselkäyttövoimaiseen autoon sopivilla. Muina haasteina on luoda täysin käyttäjän hallinnoima visuaalinen käyttöliittymä, jossa mittariston paikkaa voisi vaihtaa vain kosketukseen tai hiiren liikkeeseen perustuvalla tekniikalla. Nykyisessä ohjelmassa mittariston paikat ovat vaihdettavissa, jos koordinaatteja ohjelman sisällä muokataan.

Tämän työn antamaa pohjaa voidaan mahdollisesti käyttää myös muihin samankaltaisiin projekteihin, joko kokonaisuudessaan tai vain osittain. Ohjelman voi ladata Metropolia Wiki -sivustolta osoitteesta:

https://wiki.metropolia.fi/display/koneautomaatio/Mittaristo+IVI_naali

Lähteet

1 Acer Inc:n tekniset tiedot Iconia W500. Verkkojulkaisu, http://us.acer.com/ac/en/US/content/model/LE.RK602.047 luettu 21.5.2012

2 VDO:n ViewGate product presentation, Verkkojulkaisu, www.vdonl.nl/Downloads.aspx?id=644&start=1 luettu 21.5.2012

3 CiA:n (Can in Autmation) CAN protocol-julkaisu. Verkkojulkaisu, http://www.can-cia.de/index.php?id=systemdesign-can-protocol luettu 23.5.2012

4 Qt-Centerin QGraphicsView Class Reference. Verkkojulkaisu, http://qt-project.org/doc/qt-4.8/qgraphicsview.html luettu 25.5.2012

5 Qt-Centerin QGraphicsScene Class Reference. Verkkojulkaisu, http://qt-project.org/doc/qt-4.8/qgraphicsscene.html luettu 25.5.2012

6 Qt-Centerin QGraphicsItemGroup Class Reference. Verkkojulkaisu, http://qt-project.org/doc/qt-4.8/qgraphicsitemgroup.html luettu 25.5.2012

7 Qt-Centerin QGraphicsSvgItem Class Reference. Verkkojulkaisu, http://qt-project.org/doc/qt-4.8/qgraphicssvgitem.html luettu 25.5.2012

8 Nokia Qt-docin QGraphicsRectItem Class Reference. Verkkojulkaisu, http://doc.qt.nokia.com/4.7-snapshot/qgraphicsrectitem.html luettu 25.5.2012

9 Michael J. Young, 1998, Mastering Visual C++ 6, Amerikan Yhdysvallat, San Francisco: Sybex Inc.

10 Trey Weaverin blogi. Verkkojulkaisu, http://treyweaver.blogspot.fi/2010/10/setting-up-qt-development-environment.html luettu 4.6.2012

11 Bill's Mini2440 Forum, application auto start. Verkkojulkaisu, http://billforums.station51.net/viewtopic.php?f=5&t=55 luettu 4.6.2012

mainwindow.h - tiedosto

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QDebug>
#include <QObject>
#include <math.h>
#include <QTimer>

#include "can/canbus_era/candataera.h"
#include "scene/mittaristo.h"
#include "scene/mittaristo_lisapalkki.h"

namespace Ui {
class MainWindow;
}
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
    QRect *screenresolution;
    double *scale_xy_;

    bool i;
    QTimer refresh_rate;
    QTimer testitimer;
    QTimer *flasher_timer;

    CanDataEra *canDataEra_;
    mittaristo *mittaristo_;
    mittaristo_lisapalkki *mittaristo_lisapalkki_;

    /// MUUTTUJIA
    bool speedtrigger;
    quint8 old_speed;
    qint32 old_rpm;
    qint16 old_temp;
    qint16 old_inv_temp;
    quint16 old_gas;
    qint8 old_cell_temp;
    bool fl;
    bool fl2;
    bool fl3;
    bool flt;
    bool flt2;
    bool flt3;
    bool flash_check;
    double scale_x_;
    double scale_y_;
    QGraphicsScene *scene; public slots:
```

```
 // mittareiden ohjausslotit
    void ControlSpeed();
    void Controlrpm();
    void ControlGas();
    void ControlTemp();
    void ControlFlasherclock();
    void ControlFlashers();
    void Control_Lights_F_R();
    void WarningLights_Control();
    void ControlExtraMeters_mittaristo();
    void Controll_lisapalkki_values();
    void testislot();
private:
    Ui::MainWindow *ui;
private slots:
protected:
};
#endif


// MAINWINDOW_H
```

Mittaristo.h - tiedosto

```
#ifndef MITTARISTO_H
#define MITTARISTO_H
#include <QtSvg/QtSvg>

class mittaristo
{
    //Q_OBJECT
public:
    explicit mittaristo(QRect *screenresolution = 0, double *scale_xy_
= 0);
    /// Mittaristo NÃ¤kymÃ¤
    // lisaruudut
    QSvgRenderer *tyhja_mittari_ilm_kesk_renderer;
    QSvgRenderer *tyhja_mittari_renderer;
    QGraphicsSvgItem *tyhja_BG;
    QGraphicsSvgItem *tyhja2_BG;
    QGraphicsSvgItem *tyhja_puolikas_BG;
    QGraphicsItemGroup *lisa_group;
    QGraphicsRectItem *tyhja1_Anchorpoint;
    QGraphicsRectItem *tyhja2_Anchorpoint;
    QGraphicsSimpleTextItem *battery_percentage_text;
    QGraphicsSimpleTextItem *battery_percentage_text2;
    QGraphicsSimpleTextItem *energy_text;
    QGraphicsSimpleTextItem *energy_text2;
    // gear
    QGraphicsItemGroup *gear_Group;
    QGraphicsRectItem *gear_Rect;
    QGraphicsSimpleTextItem *gear_text;
    QGraphicsSimpleTextItem *gear_status_text;
    // Nopeusmittari
    QSvgRenderer *SpeedOMeter_Renderer;
    QGraphicsItemGroup *SpeedOMeter_Group;
    QGraphicsItemGroup *SpeedOMeter_Pointer_Group;
    QGraphicsSvgItem *SpeedOMeter_BG;
    QGraphicsRectItem *SpeedOMeter_Pointer;
    QGraphicsRectItem *SpeedOMeter_Pointer2;
    QGraphicsRectItem *SpeedOMeter_Anchorpoint;
    QGraphicsRectItem *SpeedOMeter_Anchorpoint2;
    QGraphicsSimpleTextItem *SpeedOMeter_text;
    QGraphicsSimpleTextItem *SpeedOMeter_text2;
    // cruise
    QGraphicsItemGroup *cruise_Group;
    QGraphicsPolygonItem *cruise_pointer;
    QGraphicsRectItem *cruise_Anchorpoint;
    // Kierroslukumittari
    QSvgRenderer *rpm_Renderer;
    QGraphicsItemGroup *rpm_Group;
    QGraphicsItemGroup *rpm_Pointer_Group;
    QGraphicsRectItem *rpm_Pointer;
    QGraphicsRectItem *rpm_Pointer2;
    QGraphicsSvgItem *rpm_BG;
    QGraphicsRectItem *rpm_Anchorpoint;
    QGraphicsRectItem *rpm_Anchorpoint2;
    QGraphicsSimpleTextItem *rpm_text2;
    QGraphicsSimpleTextItem *rpm_text;
```

```
// Lampotilamittari
QSvgRenderer *Temp_Renderer;
QGraphicsItemGroup *Temp_Group;
QGraphicsItemGroup *Temp_Pointer_Group;
QGraphicsSvgItem *Temp_BG;
QGraphicsSvgItem *Temp_Pointer;
QGraphicsSvgItem *Temp_Pointer2;
QGraphicsSvgItem *Temp_Pointer3;
QGraphicsRectItem *Temp_Anchorpoint;
QGraphicsRectItem *Temp_Anchorpoint2;
QGraphicsRectItem *Temp_Anchorpoint3;
QGraphicsRectItem *Temp_Anchorpoint4;
// Polttoainemittari
QSvgRenderer *Gas_Renderer;
QGraphicsItemGroup *Gas_Group;
QGraphicsItemGroup *Gas_Pointer_Group;
QGraphicsSvgItem *Gas_BG;
QGraphicsSvgItem *Gas_Pointer;
QGraphicsRectItem *Gas_Anchorpoint;
QGraphicsRectItem *Gas_Anchorpoint2;
// Neula
QSvgRenderer *Neula_Renderer;
QSvgRenderer *Neula_Renderer2;
QSvgRenderer *Neula_Renderer3;
QGraphicsItemGroup *Neula_Group;
QGraphicsSvgItem *Neula_BG;
QGraphicsSvgItem *Neula_Knob;
QGraphicsSvgItem *Neula_Pointer;
QGraphicsRectItem *Neula_Anchorpoint;
// Keskipiste
QGraphicsItemGroup *center_Group;
QGraphicsRectItem *center_Anchorpoint;
QGraphicsRectItem *corner_Anchorpoint;
/// mittariston varoitusvalot
// vilkku
QSvgRenderer *Vilkku_Renderer;
QSvgRenderer *Vilkku_off_Renderer;
QGraphicsItemGroup *Vilkku_Group;
QGraphicsItemGroup *Vilkku_off_Group;
QGraphicsItemGroup *Vilkku_Group2;
QGraphicsItemGroup *Vilkku_off_Group2;
QGraphicsSvgItem *Vilkku_BG;
QGraphicsSvgItem *Vilkku_off_BG;
QGraphicsSvgItem *Vilkku_BG2;
QGraphicsSvgItem *Vilkku_off_BG2;
QGraphicsRectItem *Vilkku_Anchorpoint;
QGraphicsRectItem *Vilkku_Anchorpoint2;
//Bensavalo
QSvgRenderer *Nafta_Renderer;
QGraphicsItemGroup *Nafta_Group;
QGraphicsSvgItem *Nafta_BG;
QGraphicsRectItem *Nafta_Anchorpoint;
//LÃ¤mpÃ¶tila valo
QSvgRenderer *temp_light_renderer;
QSvgRenderer *temp_light_high_renderer;
QGraphicsItemGroup *temp_light_Group;
QGraphicsSvgItem *temp_light_BG;
QGraphicsSvgItem *temp_light_high_BG;
QGraphicsRectItem *temp_light_Anchorpoint;
```

```
    //VAroitusvalot ja tekstit
    QGraphicsItemGroup *car_warnings_group;
    QGraphicsSimpleTextItem *warning_text;
    QGraphicsRectItem *warning_Anchorpoint;
    QGraphicsRectItem *upbarlights_Anchorpoint;
    QSvgRenderer *handbrake_Renderer;
    QGraphicsSvgItem *handbrake_BG;
    QGraphicsRectItem *UpBar_BG;
    QSvgRenderer *UpBarLights_renderer;
    QGraphicsSvgItem *frontlight;
    QGraphicsSvgItem *foglight;
    QGraphicsSvgItem *battery;
    QGraphicsSvgItem *motor_warning;
    QGraphicsSvgItem *oil_warning;
    QGraphicsSvgItem *immobilizer_warning;
    // matkamittari
    QGraphicsItemGroup *odo_group;
    QGraphicsSimpleTextItem *odo_1;
    QGraphicsSimpleTextItem *odo_2;
    QGraphicsSimpleTextItem *odo_3;
    QGraphicsSimpleTextItem *odo_4;
    QGraphicsSimpleTextItem *odo_5;
    QGraphicsSimpleTextItem *odo_6;
    QGraphicsRectItem *odo_Anchorpoint;
    QGraphicsRectItem *odo_1_Rect;
    QGraphicsRectItem *odo_2_Rect;
    QGraphicsRectItem *odo_3_Rect;
    QGraphicsRectItem *odo_4_Rect;
    QGraphicsRectItem *odo_5_Rect;
    QGraphicsRectItem *odo_6_Rect;
    //trippimittari
    QGraphicsItemGroup *trip_group;
    QGraphicsSimpleTextItem *trip_1;
    QGraphicsSimpleTextItem *trip_2;
    QGraphicsSimpleTextItem *trip_3;
    QGraphicsSimpleTextItem *trip_4;
    QGraphicsRectItem *trip_Anchorpoint;
    QGraphicsRectItem *trip_1_Rect;
    QGraphicsRectItem *trip_2_Rect;
    QGraphicsRectItem *trip_3_Rect;
    QGraphicsRectItem *trip_4_Rect;
signals:
private:
    QRect *screenresolution_;
    double *scale_;

public slots:

};
#endif // MITTARISTO_H
```

mittaristo_lisapalkki.h

```
#ifndef MITTARISTO_LISAPALKKI_H
#define MITTARISTO_LISAPALKKI_H
#include <QtSvg/QtSvg>
class mittaristo_lisapalkki
{
public:
    explicit mittaristo_lisapalkki(QRect *screenresolution = 0, double
*scale_xy_ = 0);
    QGraphicsRectItem *left_Bar;
    QGraphicsRectItem *right_Bar;
    QGraphicsRectItem *bottom_Bar;
    QGraphicsRectItem *kaakko_Block;
    QGraphicsRectItem *lounas_Block;
    /// Anchorpointit
    QGraphicsRectItem *luode_Anchorpoint;
    QGraphicsRectItem *koilinen_Anchorpoint;
    QGraphicsRectItem *etela_Anchorpoint;
    QGraphicsRectItem *kaakko_Anchorpoint;
    QGraphicsRectItem *lounas_Anchorpoint;
    /// itemit
    QGraphicsItemGroup *lisa_palkki;
    QGraphicsItemGroup *vasenlaita;
    QGraphicsItemGroup *oikealaita;
    QGraphicsItemGroup *alalaita;
    // oikea laita
    QGraphicsRectItem *motor_coolant_temp_FL;
    QGraphicsRectItem *motor_coolant_temp_FR;
    QGraphicsRectItem *motor_coolant_temp_RL;
    QGraphicsRectItem *motor_coolant_temp_RR;
    QGraphicsSimpleTextItem *motor_coolant_temp_FL_text;
    QGraphicsSimpleTextItem *motor_coolant_temp_FL_status_text;
    QGraphicsSimpleTextItem *motor_coolant_temp_FR_text;
    QGraphicsSimpleTextItem *motor_coolant_temp_FR_status_text;
    QGraphicsSimpleTextItem *motor_coolant_temp_RL_text;
    QGraphicsSimpleTextItem *motor_coolant_temp_RL_status_text;
    QGraphicsSimpleTextItem *motor_coolant_temp_RR_text;
    QGraphicsSimpleTextItem *motor_coolant_temp_RR_status_text;
    // vasen laita
    QGraphicsRectItem *Drive_Bias_Rect;
    QGraphicsSimpleTextItem *Drive_Bias_text;
    QGraphicsSimpleTextItem *Drive_Bias_status_text;
    QGraphicsRectItem *Brake_Bias_Rect;
    QGraphicsSimpleTextItem *Brake_Bias_text;
    QGraphicsSimpleTextItem *Brake_Bias_status_text;
    QGraphicsRectItem *Ah_Rect;
    QGraphicsSimpleTextItem *Ah_text;
    QGraphicsSimpleTextItem *Ah_status_text;
    QGraphicsRectItem *kW_Rect;
    QGraphicsSimpleTextItem *kW_text;
    QGraphicsSimpleTextItem *kW_status_text;
    QGraphicsRectItem *A_Rect;
    QGraphicsSimpleTextItem *A_text;
    QGraphicsSimpleTextItem *A_status_text;
    QGraphicsRectItem *Pos_contactor_Rect;
```

```
    QGraphicsSimpleTextItem *contactor_text;
    QGraphicsSimpleTextItem *Pos_contactor_text;
    QGraphicsSimpleTextItem *Pos_contactor_status_text;
    QGraphicsRectItem *Neg_contactor_Rect;
    QGraphicsSimpleTextItem *Neg_contactor_text;
    QGraphicsSimpleTextItem *Neg_contactor_status_text;
    QGraphicsRectItem *Battery_OK_Rect;
    QGraphicsSimpleTextItem *Battery_OK_text;
    QGraphicsSimpleTextItem *Battery_OK_status_text;
    QGraphicsRectItem *IWS_OK_Rect;
    QGraphicsSimpleTextItem *IWS_OK_text;
    QGraphicsSimpleTextItem *IWS_OK_status_text;
    QGraphicsRectItem *Converters_OK_Rect;
    QGraphicsSimpleTextItem *Converters_OK_text;
    QGraphicsSimpleTextItem *Converters_OK_status_text;
    QGraphicsRectItem *DCDC_OK_Rect;
    QGraphicsSimpleTextItem *DCDC_OK_text;
    QGraphicsSimpleTextItem *DCDC_OK_status_text;
    QGraphicsRectItem *LV_Rect;
    QGraphicsSimpleTextItem *LV_text;
    QGraphicsSimpleTextItem *LV_status_text;
    QGraphicsRectItem *drive_limit_Rect;
    QGraphicsSimpleTextItem *drive_limit_text;
    QGraphicsSimpleTextItem *drive_limit_status_text;
    QGraphicsRectItem *LV_current_Rect;
    QGraphicsSimpleTextItem *LV_current_text;
    QGraphicsSimpleTextItem *LV_current_status_text;
    QGraphicsRectItem *brake_limit_Rect;
    QGraphicsSimpleTextItem *brake_limit_text;
    QGraphicsSimpleTextItem *brake_limit_status_text;
    // alalaita
    QGraphicsRectItem *motor_temp_max_Rect;
    QGraphicsSimpleTextItem *motor_temp_max_text;
    QGraphicsSimpleTextItem *motor_temp_max_status_text;
    QGraphicsRectItem *inverter_temp_max_Rect;
    QGraphicsSimpleTextItem *inverter_temp_max_text;
    QGraphicsSimpleTextItem *inverter_temp_max_status_text;
    QGraphicsRectItem *cell_temp_max_Rect;
    QGraphicsSimpleTextItem *cell_temp_max_text;
    QGraphicsSimpleTextItem *cell_temp_max_status_text;
    QGraphicsRectItem *cell_voltage_min_Rect;
    QGraphicsSimpleTextItem *cell_voltage_min_text;
    QGraphicsSimpleTextItem *cell_voltage_min_status_text;
    QGraphicsRectItem *current_Rect;
    QGraphicsSimpleTextItem *current_text;
    QGraphicsSimpleTextItem *current_status_text;
    QGraphicsRectItem *voltage_Rect;
    QGraphicsSimpleTextItem *voltage_text;
    QGraphicsSimpleTextItem *voltage_status_text;
    QGraphicsRectItem *cell_temp_min_Rect;
    QGraphicsSimpleTextItem *cell_temp_min_text;
    QGraphicsSimpleTextItem *cell_temp_min_status_text;
    QGraphicsRectItem *cell_voltage_max_Rect;
    QGraphicsSimpleTextItem *cell_voltage_max_text;
    QGraphicsSimpleTextItem *cell_voltage_max_status_text;
private:
    QRect *screenresolution_;
    double *scale_;
};
#endif // MITTARISTO_LISAPALKKI_H
```

Luokkadiagrammi