Jere Kurvinen

Arduino-pohjainen Toyota-ajoneuvon diagnostiikkaväylän lukulaite

Opinnäytetyö

Insinööri

Sähkö- ja automaatiotekniikka

2020

Kaakkois-Suomen ammattikorkeakoulu

Tekijä/Tekijät	Tutkintonimike	Aika
Jere Kurvinen	Insinööri (AMK)	marraskuu 2020
Opinnäytetyön nimi Arduino-pohjainen Toyota-ajoneuvon diagnostiikkaväylän lukulaite		45 sivua 9 liitesivua
Toimeksiantaja Jere Kurvinen		
Ohjaaja Jyrki Liikanen		

Tiivistelmä

Työssä suunniteltiin ja toteutettiin vuoden 1997 Toyota Starlet -ajoneuvon diagnostiikkaväylään yhteensopiva Arduino-pohjainen lukija, jonka avulla konehuoneeseen sijoitettujen antureiden tietoja voidaan seurata langattomasti puhelimeen tehdystä sovelluksesta. Lukijan avulla voidaan paikantaa vikoja ja arvioida moottorin sekä antureiden kuntoa. Ennen lukijan suunnittelua ja toteutusta otettiin selvää diagnostiikkaliittimen tyypistä, antureilta saatavista tiedoista ja liikennöinnin muodosta.

Lukija valmistettiin Arduino-kehitysalustaa ja siihen liitettävää bluetooth-moduulia hyödyntäen. Lukijan valmistuksessa pyrittiin yksinkertaisuuteen ja työvaiheiden hyvään dokumentointiin, jotta lukijaa voidaan jatkossakin kehittää. Komponenttien valinnassa pyrittiin ottamaan huomioon niiden toiminnallisuus, saatavuus ja edullisuus.

Valmistunut lukija saavutti sille asetetut tavoitteet. Suunnitteluvaiheessa onnistuttiin valitsemaan oikeat menetelmät ja komponentit, jotta laitteen valmistaminen on yksinkertaista ja edullista. Valmiilla lukijalla onnistuttiin lukemaan diagnostiikkaväylästä saatavia tietoja ja näyttämään ne puhelimen sovelluksessa bluetooth-yhteyden kautta. Ohjelmasta onnistuttiin koostamaan helposti muokattava, ja se voidaan ladata Arduinoon ilman muutoksia.

Asiasanat

Arduino, ajoneuvo, diagnostiikka, kehitysalusta, anturi

Author (authors)	Degree	Time
Jere Kurvinen	Bachelor of Engineering	November 2020
Thesis title		
Arduino based Toyota vehicle diagnostic port reader		45 pages 9 pages of appendices
Commissioned by		
Jere Kurvinen		
Supervisor		
Jyrki Liikanen		

Abstract

The objective of the thesis was to design and implement Arduino-based diagnostic port reader compatible with the 1997 Toyota Starlet vehicle, which allows wirelessly monitoring sensors in the engine room using phone. The reader can be used to locate faults and determine health of the engine and sensors. Before designing and implementing the reader, the type of diagnostic connector, information from the sensors and form of communication was clarified.

The reader was developed using the Arduino development platform and bluetooth module connected to it. Manufacturing was aimed at simplicity and good documentation of used materials and technics, so the reader could be developed even further. In the selection of components, the aim was to consider functionality, availability and affordability.

Goals of the reader were achieved. Right methods and materials were able to be chosen so the device is simple and inexpensive to make. The reader was able to read information from the diagnostic port and display it in phone's application wirelessly via bluetooth connection. Easy-to-edit program was achieved, and it can be downloaded to the Arduino without any modifications.

Keywords

Arduino, development platform, diagnostics, sensor, vehicle

SISÄLLYS

TERMIT JA LYHENTEET

A/D	Analoginen/Digitaalinen.
+B	DLC-liittimen ulostulojännitteen nasta.
BTDC	Before Top-Dead-Center, ennen yläkuolokohtaa.
CC	Creative Commons tekijänoikeuslisenssi.
COM	Communication Port, tiedonsiirtoportti.
DLC	Diagnostic Link Connector, Toyotan diagnostiikka-liitin.
E1	DLC-liittimen maadoitus nasta.
ECT	Engine Coolant Temperature, moottorin jäähdytys-nesteen lämpötila [°C].
ECU	Engine Control Unit, moottorin ohjausyksikkö.
EN	Enable, HC-05 moduulin EN nasta.
FDM	Fused filament fabrication, yleinen 3D-tulostus-teknologia.
GND	Ground, Arduinon maadoitus nasta.
GPIO	General-Purpose Input/Output, yleiskäyttöinen tulo / lähtö.
IAC	Idle Air Control, tyhjäkäynnin ilmanohjaus.
IDE	Integrated Development Environment, integroitu ke-hitysympäristö.
ICSP	In Circuit Serial Programming, piirin sarjaohjelmointi.
IGN	Ignition Timing Angle, sytytysennakko [°].

INJ	Injector Pulse Width, Polttoainesuuttimen pulssin leveys [ms].
I/O	Input/Output, tulo / lähtö.
IoT	Internet of Things, esineiden internet.
LSB	Least Significant Bit, vähiten merkittävä bitti.
MAC	Media Access Control, verkossa yksilöivä osoite.
MAP	Manifold Absolute Pressure, imusarjan absoluuttinen paine [kPa]
MSB	Most Significant Bit, merkittävin bitti.
OBD	On-Board Diagnostic, ajoneuvon sisäinen diagnostiikka.
PLA	Polyactide, 3D-tulostamisessa yleisesti käytettävä polyaktidi muovi.
PNG	Portable Network Graphics, häviötön bittikarttagrafiikan tallennusformaatti.
PWM	Pulse-Width Modulation, pulssinleveysmodulaatio.
RPM	Rounds Per Minute, kierrosta minuutissa [rpm].
RX	Receive, sarjaliikenteen vastaanottava nasta.
SMD	Surface Mount Device, pintaliitoskomponentti.
SPD	Speed, ajoneuvon nopeus [km/h].
SPP	Serial Port Profile, sarjaportin emulointi langattoman yhteyden yli.
STL	Stereolithography, stereolitografia, 3D-tulostuksessa käytetty tiedostotyyppi.

TDCL	Toyota Diagnostic Communication Link, Toyota sarjaliikenneprotokolla.
TE2	DLC-liittimen TE2 nasta, testitilan aktivoimiseen.
TPS	Throttle Position Sensor, kaasuläpän asento anturi.
TX	Receive, sarjaliikenteen lähettävä nasta.
USART	Universal Synchronous/Asynchronous Receiver-Transmitter, sarjaliikenteen lähetys- ja vastaanottopiiri.
USB	Universal Serial Bus, sarjaväyläarkkitehtuuri oheislaitteiden liittämiseksi tietokoneeseen.
VCC	IC power-supply pin, HC-05 moduulin syöttöjännitteen nasta.
VF1	DLC-liittimen VF1 nasta, sarjaliikenteen lähettämiseen.
VIN	Input Voltage, Arduinon jänniteregulaattoriin menevän syöttöjännitteen nasta.
XML	Extensible Markup Language, merkintäkieli tiedonvälitykseen järjestelmien välillä.
XYZ	Cartesian coordinate system, karteesinen koordinaatisto.

1 JOHDANTO

Tässä opinnäytetyössä suunnitellaan ja toteutetaan laite, jonka avulla voidaan lukea vuoden 1997 Toyota Starlet -ajoneuvon diagnostiikkaväylästä saatavia tietoja. Lukijan avulla voidaan paikantaa vikoja ja arvioida moottorin sekä antureiden kuntoa. Laite tulee tehdä Arduino-kehitysalustaa hyödyntäen, ja tietojen lukemisen lisäksi laitteen täytyy lähettää ne langattomasti bluetooth-yhteyden kautta Android-sovellukseen. Lukijan valmistuksessa pyrittiin yksinkertaisuuteen ja työvaiheiden hyvään dokumentointiin, jotta lukijaa voidaan jatkossakin kehittää. Komponenttien valinnassa pyrittiin ottamaan huomioon niiden toiminnallisuus, saatavuus ja edullisuus.

Työn alussa selvitetään, mitä Arduino-kehitysalustat ovat ja mitä niiden valinnassa kannattaa ottaa huomioon. Tämän jälkeen selvitetään ajoneuvossa olevan diagnostiikkajärjestelmän toimintaa sekä tietoliikennettä. Sen jälkeen käsitellään työssä käytettävät komponentit ja menetelmät sekä valmistetaan lukija vaiheissa. Lopuksi pohditaan, miten työn tavoitteissa onnistuttiin ja mitä jatkossa kannattaisi kehittää.

2 KEHITYSALUSTA

Kehitysalusta on kokonaisuus, johon kuuluu fyysisen piirilevyn lisäksi ohjelmointiympäristö eli IDE. Arduino on ensimmäinen laajalle levinnyt, edullinen, avoimen CC-lisenssin omaava elektroniikan kehitysalusta. Arduinon suunnitelmat, kytkentäkaaviot ja lähdekoodi on avoimesti kaikkien käytettävissä, joka tekee siitä erinomaisen alustan niin aloittelijoille kuin kokeneemmille kehittäjille. Arduino-kehitysalustoja on saatavissa useissa eri hinta- ja kokoluokissa, ja projektiin sopivan alustan valinta tapahtuu haluttujen toiminnollisuuksien perusteella. [1.]

Arduinon kehitys alkoi viiden opiskelijan voimin 2000-luvun alussa Italiassa Ivrea Interaction Design -instituutiossa. Ensimmäinen Arduino-alusta esiteltiin 2005, ja sen tarkoituksena oli auttaa opiskelijoita kehittämään elektronisia prototyyppejä ilman aiempaa kokemusta elektroniikan ohjelmoinnista. Suosion

kasvaessa, Arduinoa alettiin kehittää yhä vaativampiin kohteisiin, kuten 3D-tulostimiin, automaatio- ja IoT-järjestelmiin. Arduino on edelleen suosituin kehitysalusta harrastelijoiden ja jopa joidenkin suuryritysten keskuudessa [2.]

Lähes jokainen Arduino sisältää Atmel Corporation -yhtiön valmistaman mikro-ohjaimen eli mikrokontrollerin. Mikrokontrolleri sisältää suorittimen lisäksi muisti- ja liityntälohkoja tehden siitä tietokoneeseen verrattavan laitteen. Sen tehtävä on säilyttää koostettua ohjelmakoodia ja suorittaa käyttäjän antamia käskyjä. Moneen Arduinoon lisäksi lisätty tulo- ja lähtöportteja eli I/O:ta, AD-muunnin, sarjaliikenne ja ajastimia, joiden käyttöönotto tapahtuu ohjelmoinnilla. [3, s. 6.]

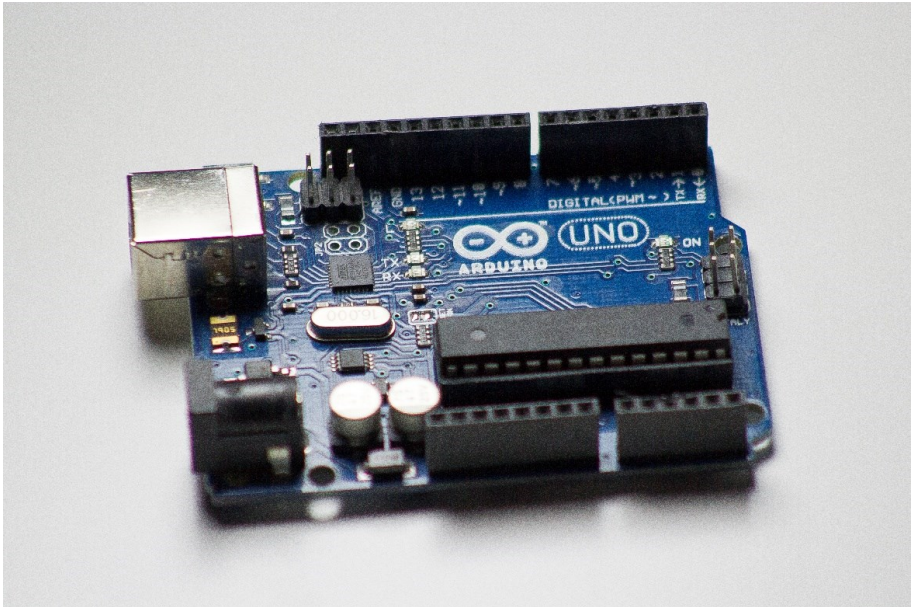
Useimmiten mikrokontrollerin ohjelmointi tehdään C- tai Assembly-kielellä erillisen ICSP-sarjaliikenneohjelmointilaitteen kautta. Arduinon sisältämän mikrokontrollerin ohjelmointia on yksinkertaistettu siten, että tehtaalla siihen on valmiiksi ohjelmoitu alkulatausohjelma, joka mahdollistaa mikrokontrollerin ohjelmoinnin tavallisen USB-liitynnän avulla, ilman erillistä ohjelmointilaitetta. [3, s. 6.]

Arduino-kehitysalustaa valittaessa on hyvä ottaa huomioon mikrokontrollerin nopeus, muistin määrä, suorittimen tehokkuus ja virrankulutus. Useiden prosessin yhtäaikainen suorittaminen tarvitsee tehokkaan suorittimen, mutta samalla virrankulutus kasvaa. Muistin määrän tarve kasvaa, mitä enemmän suoritettavaa ohjelmakoodia tai graafisia kuvia halutaan säilyttää. Valinnassa kannattaa ottaa huomioon myös kehitysalustan koko, suosio, ohjelmointikieli ja GPIO-nastojen määrä tarpeen mukaan. [4.]

Suositumman kehitysalustan käyttäjämäärät ovat suurempia, ja sen käyttöönottoon löytyy helpommin ohjeita sekä valmiita ohjelmia. Arduinolla on hyvin aktiivinen harrastajayhteisö, joka ylläpitää ohjeita ja julkaisee uusia sovelluksia päivittäin. Pelkästään Arduino-foorumille rekisteröityneitä käyttäjiä on yli miljoona [5]. Käyttäjällä onkin mahdollisuus hyödyntää olemassa olevia ohjelmia tai kehittää ohjelma kokonaan itse. Avoimen lähdekoodin ansiosta myös kehitysalustan itse kasaaminen on mahdollista.

2.1 Arduino Uno R3

Arduinon suosituin kehitysalusta Uno R3 (kuva 1) sisältää kaksi Atmelin AVR-tuoteperheen mikrokontrolleria [6]. Tärkeimpänä ATmega328P, joka on käyttäjän ohjelmoitavissa. Sen tehtävä on säilyttää ja suorittaa koostettu ohjelma. Toissijaista ATmega16U2-mikrokontrolleria käytetään käyttöliittymänä USB-kaapelin ja USART-sarjaliikenteen välillä, jonka takia erillistä USB-sarjaliikennemuunninta ei tarvita. Arduino Uno on helpoin tapa lähteä tutustumaan Arduino-kehitysalustoihin, mutta lopullisiin projekteihin se on monesti liian suurikokoinen, jonka vuoksi on kehitelty pienempiä kehitysalustoja. [3, s. 6.]

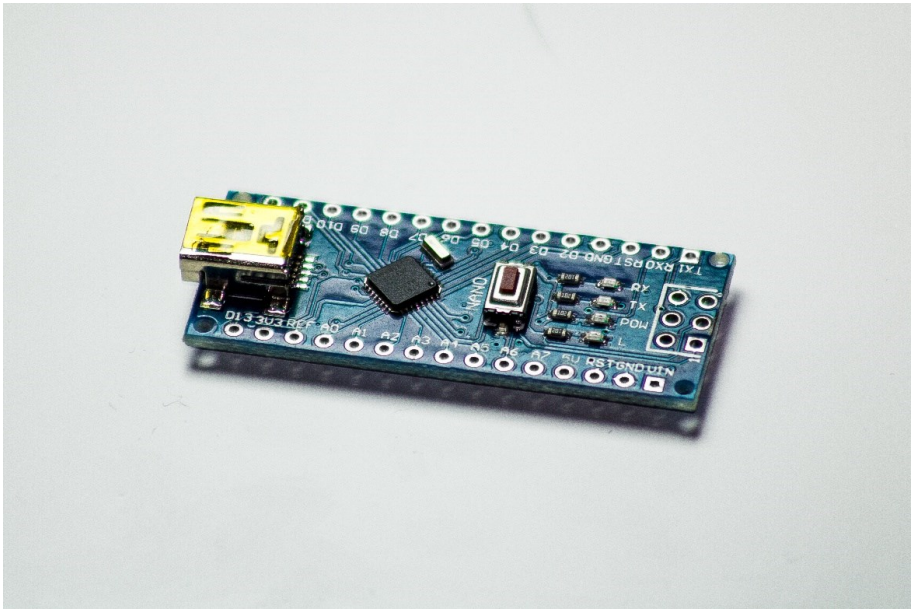


Kuva 1. Arduino Uno R3 kehitysalusta

2.2 Arduino Nano V3.0

Arduino Nano V3.0 (kuva 2.) on lähes Uno R3:a vastaava, mutta huomattavasti pienempi tehden siitä erinomaisen valinnan lopullisiin projekteihin. Nano on kooltaan 45 x 18 mm ja sisältää 16 MHz ATmega328-mikrokontrollerin, johon on yhdistetty 8 analogista sisäänmenoa ja 14 digitaalista GPIO-nastaa, joista 6 antaa PWM-signaalia. Nanossa käytetään pienempikokoisia SMD-pintaliitoskomponentteja, ja suurempikokoiset liittimet on korvattu pienemmillä, mutta toiminnaltaan vastaavilla. Esimerkiksi ohjelmointi tehdään pienimpikokoisemman Mini USB -liittimen kautta perinteisen Type-B-liittimen sijaan. [5.]

Nanossa ATmega16U2 on korvattu hiukan kalliimmalla, mutta kestävämmällä FT232RL-USB-sarjaliikennemuuntimella [7]. Sarjaliikennemuunnin on ominaisuuksiltaan rajoittuneempi, sillä Arduinoa ei voida ohjelmoida näkymään erityyppisenä USB-laitteena, kuten vaikkapa näppäimistönä tai peliohjaimena, vaan se näkyy tietokoneella ainoastaan COM-sarjaliikenneporttina. Toimiakseen se tarvitsee oikeanlaiset Windows-ajurit, eikä välttämättä toimi plug and play -tyyppisesti. Ajurit tulevat kuitenkin IDE-ohjelmiston asennuspaketin mukana. [8.]



Kuva 2. Arduino Nano V3.0 -kehitysalusta

2.3 Arduino IDE

Arduino IDE on Arduinon ohjelmointiin tarkoitettu ohjelmointiympäristö, joka on ilmaiseksi ladattavissa Arduinon kotisivuilta. IDE on yhteensopiva C- ja C++-ohjelmointikielien kanssa, ja se voidaan asentaa Windows-, Linux- ja Mac OS X -käyttöjärjestelmiin. Asennus Windows-käyttöjärjestelmään kannattaa tehdä asennustiedoston avulla, joka asentaa Arduinon kanssa keskusteluun tarvittavat USB-ajurit. [9.] Asennuksen jälkeen ohjelma voidaan käynnistää, ja Arduino liittää USB-kaapelilla tietokoneeseen.

IDE:llä luodaan ohjelma eli luonnos, jossa määritellään Arduinon tekemät toiminnot. Luonnos ladataan Arduinon ROM- eli lukumuistiin USB-liitännän

kautta, joten mikrokontrollerin ei tarvitse olla jatkuvasti yhdistettynä tietokoneeseen. Ohjelma voidaan ladata Arduinoon etukäteen, jonka jälkeen se voidaan asentaa paikoilleen ja hoitaa virransyöttö VIN-nastaan. Arduinoon voidaan tallentaa vain yksi ohjelma kerrallaan, ja vanha luonnos korvautuu aina uudella. Arduinon ajatus on helpottaa ohjelmointiin ja elektroniikkaan tutustumista, joten sen ohjelmointikieli on yritetty pitää mahdollisimman yksinkertaisena. IDE tarkastaa ohjelman oikeellisuuden ennen luonnoksen lataamista mikrokontrolleriin. Mikäli se on puutteellinen, IDE näyttää korostetusti väärinkirjoitetun ohjelmapätkän. [10.]

IDE-ohjelmiston valmiit ohjelmakirjastot selkeyttävät Arduinon ohjelmoimista. Kirjastoja voidaan hakea ”Sketch”-välilehden ”Include Libraries” -painikkeen alta löytävästä ”Library Manager” -ikkunasta. Tämän kautta kirjastoja voidaan hakea ja asentaa. Ohjelmaan ne tuodaan #include tagilla, jonka perään kirjoitetaan tuotavan kirjaston nimi. Tärkein on Wiring-niminen, IDE-ympäristöön sisäänrakennettu kirjasto, joka tekee yleisten GPIO-nastojen operoinnin yksinkertaisemmaksi. Wiring-ohjelmat kirjoitetaan C++-kielellä, ja minimaalisin ohjelma vaatii ainoastaan kaksi funktiota. [3, ss. 13 – 18.]:

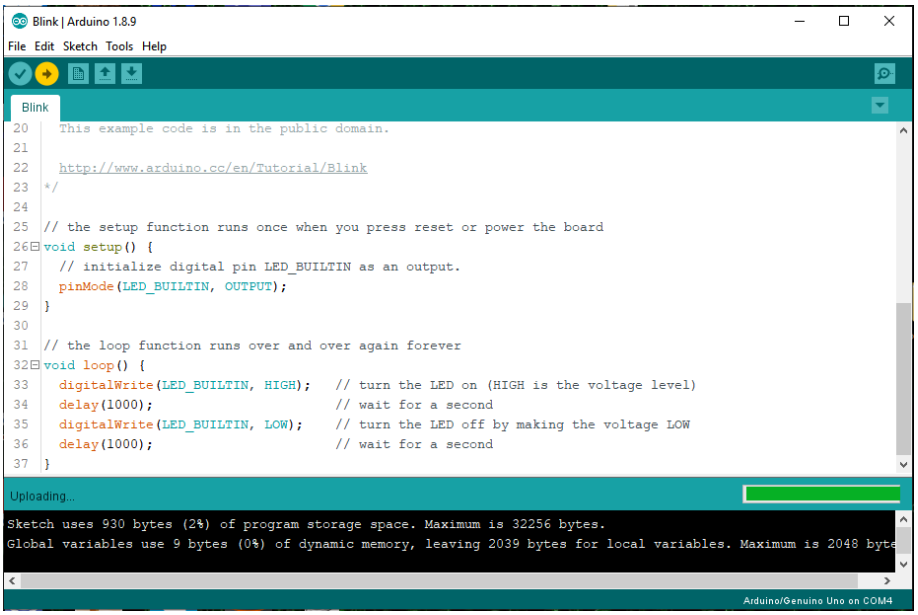
- setup() funktio suoritetaan ainoastaan kerran ohjelman alussa. Funktion sisällä voidaan määritellä muuttujien alkuasetukset.
- loop() funktio suoritetaan jokaisella ohjelmakierrolla yhä uudelleen, niin kauan, kunnes Arduino sammutetaan tai resetoidaan. Yksinkertaisissa ohjelmissa riittää, että toiminnot tehdään loop() funktion sisällä.

Näiden kahden funktion lisäksi voidaan määritellä omia funktioita, joita kutsutaan loop() funktiossa halutulla hetkellä. Omien funktioiden sisään voidaan kirjoittaa ohjelmakoodia, joka suoritetaan kerran, kun funktiota kutsutaan. [3, ss. 13 – 18.]

Ennen funktioita määritellään muuttujat. Muuttujia voidaan määritellä erityyppisiksi, esimerkiksi boolean-, volatile- ja unsigned long -määritteillä. Boolean määrittää muuttujan niin, että sen arvo voi olla 0–1 välillä. Unsigned long -määritystä käytetään numeromuuttujien määrittelyyn, ja se voi varastoida 32 bittiä pitkiä numeroita. Volatile-määrityksellä arvojen kirjoittaminen ja

luku tehdään Arduinon RAM-muistiin, joka lisää lukujen luetettavuutta, kun niitä kirjoitetaan ”interrupt”-toiminnolla. [11.]

Arduinon toimintaa voidaan testata IDE-ympäristöön valmiiksi tehdyillä esimerkkikoodeilla. ”File”-välilehden ”Examples”-valikosta voidaan hakea esimerkiksi blink-ohjelma (kuva 3.), joka vilkuttaa Arduinoon sisäänrakennettua LE-Diä sekunnin välein. ”Tools”-välilehdeltä haetaan ”Board:”-painikkeesta ohjelmoitavan alustan tyyppi, kuten vaikkapa Arduino Nano. ”Port”-painikkeesta tulee valita oikea COM-portti, johon Arduinon USB-kaapeli on kytketty. Ohjelmakoodi voidaan koostaa ”Sketch”-välilehden ”Verify/Compile”-painikkeesta tai pikanäppäimillä ”CTRL+R”. Arduinon ohjelmointi tehdään samasta valikosta ”Upload”- tai ”CTRL+U”-pikanäppäimillä, joka lataa ohjelman Arduinoon. [3, ss. 13 – 18.]



Kuva 3. Arduino IDE-ohjelmointiympäristö

3 AJONEUVON DIAGNOSTIIKKA

3.1 Moottorinohjausyksikkö

Moottorinohjausyksikkö eli ECU on ajoneuvosta löytyvä tietokoneen kaltainen elektroninen laite, joka ohjaa polttomoottorin tapahtumia optimaalisen suorituskyvyn ja taloudellisuuden saavuttamiseksi. ECU vastaanottaa dataa lukui-

sista ajoneuvoon sijoitetuista antureista ja ohjaa datan perusteella auton toimilaitteita. Liitteessä 1 on esiteltynä Toyota 4E-FE -moottorin ECU-ohjauksen kytkentäkaavio ja siihen liitetyt anturit ja toimilaitteet. Tyypillisin ohjattava toimilaite on polttoaineen suihkutusjärjestelmä, joka huolehtii polttoaineen toimittamisesta sylintereille. [12.]

3.2 OBD-järjestelmät

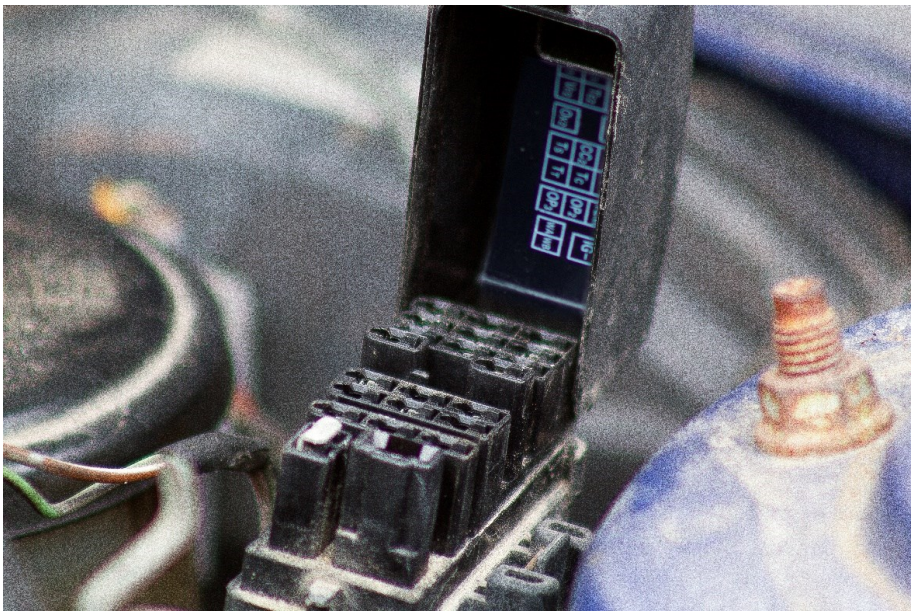
Moottorinohjausyksiköiden yleistyessä myös vikojen diagnosointia haluttiin parantaa. Ajoneuvoihin lisättyä itsediagnostiikkajärjestelmää eli OBD-järjestelmää voidaan käyttää moottorinohjausyksikön tallentamien tietojen lukemiseen. Modernit OBD-toteutukset käyttävät nopeaa tiedonsiirtoporttia ja standardoituja vikakoodeja. Vanhoista ajoneuvoista löytyvä OBD-I on valmistajakohtainen standardi, joka voi erota suuresti nykyaikaisemmasta OBD-II standardista, käytettyjen osoitteiden tai tiedonsiirtoprotokollien osalta. [13.]

Ajoneuvoihin, jotka sisältää OBD-II standardin diagnostiikan, on helppo löytää edullisia ja toimivia, diagnostiikan lukulaitteita [13]. Vanhemman OBD-I-liitännän lukemiseen tarvitaan kuitenkin erikseen kyseiseen automerkkiin ja malliin soveltuva lukija. Mallikohtaisten lukijoiden saatavuus on heikkoa, ja hinnat voi helposti nousta useisiin satoihin euroihin. OBD-II-standardisoitu liityntä tuli pakolliseksi Suomessa myytäviin ajoneuvoihin vuonna 2000, joten sitä vanhemmista löytyy useimmiten ainoastaan OBD-I. [14.]

3.3 Data Link Connector

Toyotan ennen vuotta 2000 valmistamista autoista löytyy useimmiten OBD-I-järjestelmä. Toyotan 4E-FE-moottorin kytkentäkaaviosta (ks. liite 1) löytyvä X1-niminen liityntä viittaa konehuoneesta löytyvään Data Link Connector lyh. DLC-liittimeen. Liitin sijaitsee yleensä konepellin alla, kuljettajan puolella olevan sulakerasian lähistöllä, lähellä iskunvaimentimen kiinnityskohtaa. Liittimen nastoista voidaan lukea antureiden antamia arvoja jännitteinä sekä ECU:n antamia tietoja jännitepulsseina. Liittimen malli ja siinä olevien nastojen toiminnallisuudet vaihtelevat ajoneuvon mallin, valmistusvuoden ja moottorityypin mukaan. [15.] Vuoden 1997 Toyota Starletista löytyi DLC1-tyyppinen (kuva 4.)

22-nastainen liitin. Työssä käytin liittimestä löytyviä +B-, E1-, TE2- ja VF1-nastoja.



Kuva 4. Toyota Starlet 1997 DLC1-liitin

3.4 Toyota Diagnostic Communication Link

Toyota Diagnostic Communication Link lyh. TDCL on joistakin DLC-liittimistä löytyvä tietoliikenneväylä, joka lisää reaaliaikaisen moottorin anturitietojen lähettämisen sarjaliikenteenä eteenpäin. Kaikista Toyotan ajoneuvoista väylää ei löydy, ja helpoin tapa sen olemassaolon selvittämiseen on tarkistaa DLC-liittimen kannen sisäpuolelta TE2-merkinnän löytyminen, joka indikoi sarjaliikenteen olemassaolon. TE2-nastan yhdistäminen E1-nastaan antaa ECU:lle käskyn siirtyä testitilaan. Testitilassa ECU lähettää antureilta saamiaan tietoja reaaliaikaisesti VF1-nastan kautta sarjaliikenteenä. [15.]

Sarjaliikenne koostuu biteistä ja niiden muodostamasta bittijonosta. TDCL:n lähettämä bittijono alkaa 16 bittiä pitkällä lepotiedolla, jota seuraa 4 bittiä pitkä OBD-tunnus. Tunnuksen jälkeen tulee kaksitoista kappaletta 11 bittisiä sanoja, jotka sisältävät antureilta luetut tiedot. Tiedot on kehystetty sanoihin, jotta vastaanottava laite tietää, milloin varsinainen data alkaa. Sanan alkaminen on määritetty Start-bitillä, joka on looginen nolla, ja loppuminen on määri-

tetty kahdella Stop-bitillä, jotka molemmat ovat loogisia ykkösiä. Näiden välissä on kahdeksan data bittiä, jotka sisältävät anturilta saadut arvot binääri-eli 2-järjestelmämuodossa. [16, s. 1.]

Taulukossa 1 on esimerkki, miltä yksittäinen sana voi näyttää. Start- ja stop-bittien välissä olevat kahdeksan databittiä muodostavat binäärisen luvun, jonka maksimiarvo desimaalissa on 255. Ensimmäistä eli suurimman arvon kuvaavaa bittiä kutsutaan MSB-bitiksi ja viimeistä pienimmän arvon omaavaa LSB-bitiksi. LSB-bitti määrittää, onko luku parillinen vai pariton. Binääriluku muutetaan desimaaliluvuksi kertomalla kaikki numerot kahden potensseilla oikealta vasemmalle suuruusjärjestyksessä ja laskemalla tulot yhteen. [17.]

Esimerkkiluvun 10101011 muuntaminen desimaaliksi:

$$1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 215$$

Taulukko 1. Kehyksen sisältämät bittitiedot

START	1 (MSB)	2	3	4	5	6	7	8 (LSB)	STOP	STOP
0	1	0	1	0	1	0	1	1	1	1

Koska databittejä on vain kahdeksan ja desimaalinen arvo rajoittuu 255:een, ei kaikkien antureiden tietoja voida tuoda oikeassa muodossa, vaan niihin täytyy jälkikäteen käyttää skaalauksia. Kun luetaan moottorin kierrosnopeus ja saadaan arvoksi 60, tulee se kertoa korjauskertoimella 25, joten todelliseksi kierrosnopeudeksi saadaan 1500 rpm. Ajoneuvon nopeus, imusarjan paine ja binäärimuuttujat voidaan tuoda sellaisinaan, ilman skaalausta, sillä niiden arvo on yleensä alle 255. [16, s. 1.]

Kolmestatoista sanasta yhdeksän on tunnettuja. Luettaviin tietoihin kuuluu mm. suuttimen pulssin leveys, tyhjäkäynnin ilmanohjauksen venttiilin asento, moottorin kierrosnopeus, imusarjan absoluuttinen paine, moottorin jäähdytysnesteen lämpötila, kaasuläpän asento ja ajoneuvon nopeus. Taulukossa 2 on tunnettujen sanojen lyhenteet, skaalaukset ja yksiköt. Viimeiset 0x11 ja 0x12 sanaa sisältää yhdistetysti ECUn antamia 0–1 tietoja eri antureilta ja kytkimiltä, joiden tunnetut bitit on lisätty taulukkoon bittien numerojärjestyksen mukaisesti vasemmalta oikealle ja tuntemattomat bitit on jätetty välistä. [16, s. 2.]

Taulukko 1. Sanojen merkitykset [16, ss. 1 – 2.]

Sana	Merkitys	Yksikkö	Skaalaus
0x01	INJ	mS	$X/10$
0x02	IGN	$°\ BTDC$	$X - 90$
0x03	IAC	%	$Yhtälö\ 1.$
0x04	RPM	rpm	$X \cdot 25$
0x05	MAP	$kPa\ Abs.$	X
0x06	ECT	V	$Taulukko\ 4.$
0x07	TPS	%	$X/2$
0x08	SPD	km/h	X
0x11.1	Kylmärikastus	$1 = päällä$	$0 - 1$
0x11.2	Lämmitys	$1 = päällä$	$0 - 1$
0x12.1	Käynnistys	$1 = päällä$	$0 - 1$
0x12.2	Tyhjäkäyntiventtiili	$1 = kiinni$	$0 - 1$
0x12.3	A/C lämmitys	$1 = päällä$	$0 - 1$
0x12.7	Vikakoodit	$1 = Ei\ koodeja$	$0 - 1$

IAC-anturilta tieto tulee venttiiliä ohjaavan servomoottorin askelmäärinä, josta venttiilin asento prosentteina voidaan laskea yhtälön 1 mukaisesti [16, s. 1].

$$IAC = \frac{IAC_{step}}{125} \cdot 100 = 100\ \% \tag{1}$$

jossa IAC Venttiilin asento [%]

IAC_{step} Venttiilin servomoottorin askelmäärä [-]

ECT-anturilta tieto tulee 0,3–4,3 V jännitteenä, josta lämpötila voidaan laskea celsiusasteina taulukon 3 kaavojen mukaisesti [16, s. 2].

Taulukko 2. Moottorin lämpötilan laskeminen jännitteestä

Jännitealue	Laskukaava
3,4 – 4,3 V	$T_{ECT} = -20 + (4{,}3 - V_{ECT}) \cdot 22{,}22$
2,4 – 3,4 V	$T_{ECT} = 0 + (3{,}4 - V_{ECT}) \cdot 20$
1,5 – 2,4 V	$T_{ECT} = 20 + (2{,}4 - V_{ECT}) \cdot 22{,}22$
0,9 – 1,5 V	$T_{ECT} = 40 + (1{,}5 - V_{ECT}) \cdot 33{,}33$
0,5 – 0,9 V	$T_{ECT} = 60 + (0{,}9 - V_{ECT}) \cdot 50$
0,3 – 0,5 V	$T_{ECT} = 80 + (4{,}3 - V_{ECT}) \cdot 100$

TDCL lähettää datan 1,25 s välein n. 100 baudin tiedonsiirtonopeudella [15]. Baudi kertoo, kuinka nopeasti signaali voi muuttua sekunnin aikana. Kyseisen 100 baudin nopeudella sekunnissa voidaan havaita 100 bitin muutosta, ja kun sekunti jaetaan baudeilla, saadaan yhden bitin päällä oloajaksi 10 mS. Kyseessä on hyvin hidas sarjaliikenne, sillä nykyiset siirtonopeudet vaihtelevat 9600–115200 baudin välillä. [18.] 100 baudin nopeus vaikuttaa kuitenkin riittävältä ajoneuvon vikojen diagnosointiin.

4 KOMPONENTTIEN VALINTA

4.1 Arduino

Aluksi ajatuksena oli valita Arduino Uno R3 -kehitysalusta minimaalisen konfiguraatiomahdollisuuden takia. Uno R3 eroaa muista Arduino-kehitysalustoista siten, että ohjelmoitava mikrokontrolleri on irrotettavissa, eikä sitä ole tinattu kiinteästi suoraan piirilevyyn, kuten vaikkapa Arduino Nano -kehitysalustassa. Mikrokontrolleri voidaan ensin ohjelmoida kehitysalustan avulla, ja sen jälkeen siirtää se erilliselle piirilevylle. Tällöin piirilevylle luodaan ns. minimaalinen konfiguraatio ja muiden komponenttien sijoittelu on vapaampaa, mutta suuri osa kehitysalustan ominaisuuksista jää puuttumaan. [19.]

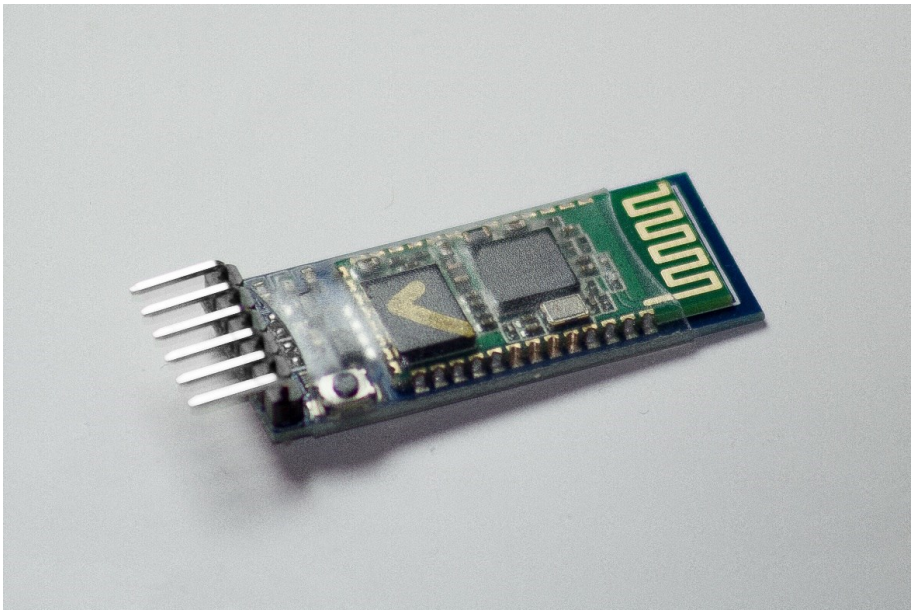
Päädyin kuitenkin valitsemaan Arduino Nanon, sillä halusin lopulliseen lukijaan mahdollisuuden uudelleen ohjelmointiin USB-portin avulla. Arduino Nano

on toiminnaltaan lähes UNOa vastaava, eikä tehosta tarvitse tinkiä, sillä molemmat hyödyntävät samaa ATmega328-mikrokontrolleria ohjelman suorittamiseen. Työssä tarvittavien nastojen määrä on vähäinen, sillä TDCL-sarjaliikenteen lukemiseen tarvitaan yksi ja HC-05-moduulille kaksi GPIO-nastaa.

4.2 HC-05-bluetoothmoduuli

Edulliset kuluttajille suunnatut ajoneuvojen OBD-lukulaitteet analysoivat ja lähettävät tiedot bluetoothin kautta puhelimeen. Puhelimeen asennettava sovellus toimii käyttäjän ja lukulaitteen välisenä käyttöliittymänä, ja erillinen näyttö, painikkeet yms. voidaan jättää lukijasta, jotta siitä saadaan mahdollisimman kompakti ja edullinen. Langattoman yhteyden ansiosta lukija voidaan jättää diagnostiikkaporttiin kiinni, eikä erillisiä johtimia tarvita näytön tuomiseksi kojelaudalle. Puhelimen kanssa tapahtuvaa keskustelua varten valitsin Arduinon kanssa yhteensopivan HC-05-bluetoothmoduulin (kuva 5.).

HC-05-moduulilla voidaan luoda langaton SPP-sarjaportti, joka simuloi RS-232-sarjaliikennettä, jossa liikenne siirtyy yksi bitti kerrallaan sarjamuotoisena, hyvin samaan tapaan kuin TDCL-sarjaliikenteessä. Moduulia voidaan käyttää isäntä- tai orjatyyppisenä ja +4 dBm lähetysteholla voidaan saavuttaa 10 m kantama. [20.]



Kuva 5. HC-05-bluetooth-moduuli

HC-05-moduulin VCC-nastan käyttöjännite voi olla 3,6–6 V väliltä, ja se voidaan ottaa Arduinon 5V nastasta. Looginen keskustelu tulee kuitenkin toteuttaa 3,3 V jännitteellä, ja tulee tehdä logiikkatason siirto. Arduinon lähettämä 5 V signaali TX/D7-nastasta viedään jännitteenjakajan kautta moduulin RX-nastaan. Jakaja toteutetaan 1 kΩ ja 2 kΩ arvoisilla vastuksilla, jotka kytketään sarjaan ja niiden välistä saadaan sopiva 3,3 V jännite. Arduino on suunniteltu tulkitsemaan 3–5 V jännite loogiseksi ykköseksi, joten jännitteen jakamista ei tarvita, kun moduuli lähettää dataa TX-nastasta Arduinon RX/D8-nastaan. [21.]

HC-05 tuli konfiguroida, eri määrittää sille nimi ja tunnusluku, jota käytetään, kun puhelin halutaan yhdistää moduulin kanssa. Konfigurointia varten moduulissa on pieni nappi, joka yhdistää EN- ja VCC-nastat. Nappia pohjassa pitämällä, moduulin käynnistyessä, siirtyy moduuli AT-komentotilaan, jossa voidaan lähettää komentoja Arduinon kautta. Kun nappia ei paineta käynnistyessä, siirtyy moduuli automaattisesti DATA-tiedonsiirtotilaan ja se voidaan havaita puhelimella. AT-komentotilassa moduulin LED-valo vilkkuu hitaasti n. kerran sekunnissa ja DATA-tilassa nopeasti n. 5 kertaa sekunnissa. [20.]

Konfigurointia varten käytin Arduino Uno -kehitysalustaa ja erillistä ohjelmaa (ks. liite 2/1), jossa komennot lähetetään Arduino IDE -sarjamonitorin kautta. Kun Arduinoa ohjelmoidaan ja seurataan USB-portin kautta, vie se käytännössä Arduinon ainoan sarjaliikenneportin itselleen. SoftWareSerial-kirjasto mahdollistaa sarjaliikenteen lähes mistä tahansa Arduinon nastoista [18]. Ohjelman alussa määritetään RX/D8- ja TX/D7-nastat, joita käytetään moduulin kanssa kommunikointiin. Moduulin sarjaliikenteen siirtonopeus AT-tilassa on 38400 baudia. [20.]

Taulukossa 4. on moduulin konfiguroinnissa käytetyt AT komennot. AT-komentojen yleisin muoto on AT+KOMENTO ja vastaus OK tai error. Moduulin kytkennän ja toiminnan varmistin käskyllä AT, jonka vastaus OK kertoo moduulin toimivan. Seuraavaksi muutin DATA-tilassa käytettävän sarjaliiketeen tiedonsiirtonopeuden vakio 9600 baudista 38400 baudiin, jotta jatkossa mo-

duulin konfigurointi ja tiedonsiirto voidaan tehdä muuttamatta nopeutta ohjelmakoodiin. Lopuksi muutin puhelimelle näkyvän nimen ja yhdistäessä annettavan PIN-koodin. Vastaukseksi komentoihin sain OK, joka tarkoittaa, että tiedot on onnistuneesti kirjoitettu moduulin Flash-muistille. [20.]

Taulukko 3. Käytettyjä AT-komentoja

Komento	Toiminto	Vastaus
AT	Yhteyden testaaminen.	OK
AT+UART?	Tiedonsiirtonopeuden kysely.	9600,0,0
AT+NAME?	Moduulin nimen kysely.	HC-05
AT+PSWD?	PIN-koodin kysely.	1234
AT+UART=38400,0,0	Tiedonsiirtonopeuden uudelleenmääritys.	OK
AT+NAME=ToyotaOBD	Moduulin nimen uudelleenmääritys.	OK
AT+PSWD=2507	PIN-koodin uudelleenmääritys.	OK

Puhelimen bluetoothilla voidaan nyt havaita ToyotaOBD-niminen yhteys, ja yhdistäminen tehdään syöttämällä määritetty PIN-koodi. Yhdistämisen jälkeen moduulin LED-valo vilkkuu nopeasti 2 kertaa viiden sekunnin välein. Tietojen lähettämisen testaamista varten latasin puhelimeen Google Play -kaupasta ”Serial Bluetooth Terminal” -sovelluksen [23]. Sovelluksesta voidaan lähettää tekstiä, joka on luettavissa sarjamonitorista ja moduulin toimiessa puhelimesta lähetetty teksti näkyy sarjamonitorissa.

4.3 LM2596-jännitteensäädin

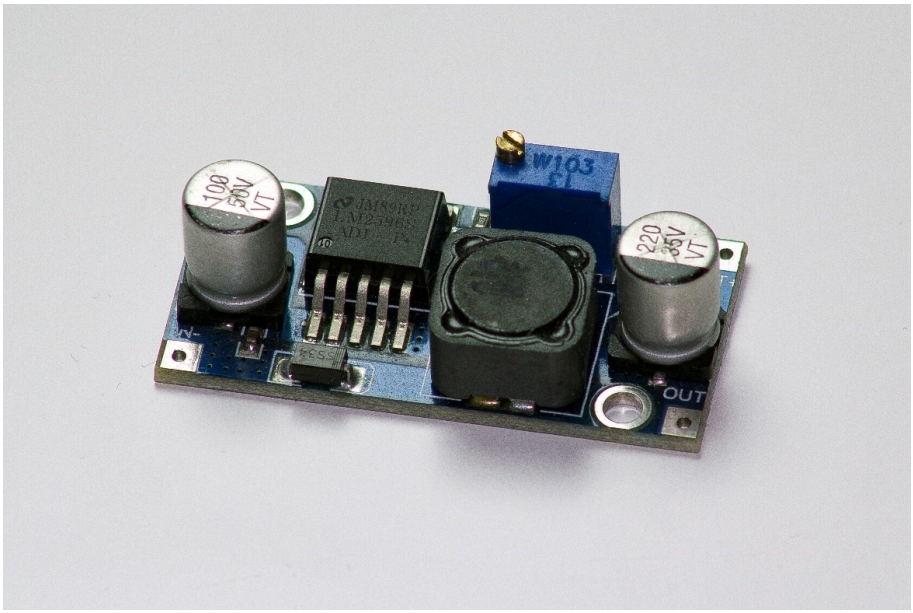
Arduino Nano voidaan sähköistää VIN-nastasta, johon voidaan kytkeä 7–12 V reguloimaton jännite [24]. Arduinossa on sisäänrakennettuna LM1117- lineaarinen regulaattori, joka on automaattisesti säätyvistä vastuksista koostuva jännitteenjakaja. Se säätyy tulojännitteen ja kuorman mukaan muodostaen tasaisen lähtöjännitteen. Ylimääräinen jännite muutetaan vastusten avulla ns. hukkalämmöksi, joka aiheuttaa regulaattorin kuumenemista, ja korkeilla jännitteillä jäähdytyssiilin käyttäminen on suositeltavaa. [25.]

DLC-liittimen +B-nastasta saadaan akun 14,4 V jännite auton ollessa käynnissä. Tätä jännitettä on hyvä hyödyntää Arduinon sähköistykseen, sillä se

katkeaa, kun auto sammutetaan, joten lukija ei turhaan kuluta ajoneuvon akkua ollessaan pysäköitynä. 14,4 V jännite on kuitenkin lähellä Arduinon LM1117-regulaattorin maksimijänniterajoja, ja korkean jännitteen pitkäaikainen käyttö saattaisi aiheuttaa pysyvää vahinkoa regulaattorille sekä Arduinolle, joten tuli miettiä vaihtoehtoinen ja turvallinen ratkaisu Arduinon sähköistämiseen. [25.]

Ratkaisuni oli käyttää LM2596-buck-hakkuritekniikkaan perustuvaa jännitteen alentavaa muunninta (kuva 6.). Kyseessä on valmis piirilevy, joka sisältää LM2596-mikropiirin, kelan, jännitteentasaus kondensaattorit ja 10 kΩ trimmerin jännitteen säätämiseen. Muuntimeen voidaan syöttää +IN-nastaan 4–40 V jännite ja trimmeristä säätämällä se voidaan laskea halutulle 5 V tasolle [26]. Hakkuripiiri katkoo jännitettä suurella taajuudella transistorin avulla, joten hyötysuhde on lineaarista regulaattoria parempi, koska hakkuripiiri on joko johtavassa tilassa tai pois päältä, eikä siinä ole suurta resistanssia tuottamassa lämpöhäviöitä [27].

Hakkurin +OUT antama jännite kannattaa kytkeä Arduinon VIN-nastaan, jotta se kulkee LM1117-regulaattorin kautta ja Arduinoon suunnitellut suojausominaisuudet säilyvät, kuten virranrajoitus, terminen sammutus ja resetoitava sulake. Trimmeriä säätämällä sain jännitteen LM1117-regulaattorin suositetulle 7 V tasolle, ja regulaattori laskee sen mikrokontrollerin tarvitsemalle 5 V tasolle. Arduinon 5V nastasta voidaan LM1117-regulaattorin jännitettä hyödyntää myös HC-05-moduulin sähköistämiseen. [24.]

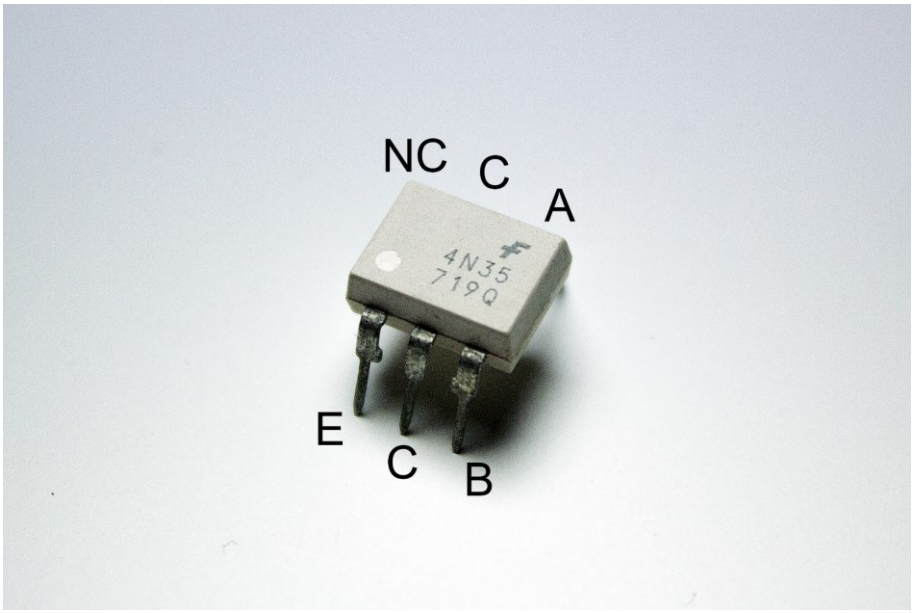


Kuva 6. LM2596 buck-hakkuri

4.4 4N35-optoerotin

Arduino tulkitsee 0–3 V jännitteen loogiseksi nollaksi ja 3–5 V jännitteen loogiseksi ykköseksi, eikä sitä ole suunniteltu kestämään suurempia jännitteitä, joten Arduinolle ei voida syöttää ajoneuvosta saatavaa 0–14,4 V signaalia ilman sen galvaanista erottamista. Galvaanisessa erottamisessa estetään tasavirran kulkeminen kahden eri virtapiirin komponenttien välillä. Erottamisella voidaan myös ehkäistä häiriöiden ja jännitepiikkien kulkeutumista mikrokontrollerille. [28.]

Signaalin erottamisen toteutin 4N35-optoerottimella (kuva 7.), jossa virtapiirien erotus tehdään valoa hohtavan infrapuna-LEDin ja valolle herkän NPN-fototransistorin avulla. Optoerottimen ensiöpuolelle syötetty jännite saa LEDin hohtamaan valoa, joka saa toisiopuolelle kytketyn fototransistorin johtamaan sähköä. Arduinon 5V nasta kytketään fototransistorin kollektoriin (C) ja emitteri (E) kytketään 15 kΩ vastuksen kautta Arduinon GND-nastaan eli maihin. Emitterin ja vastuksen välistä Arduinon D2-nastaan saadaan nyt 0–5 V pulssisignaali. [29.]



Kuva 7. 4N35-optoerotin

Fototransistorin LEDin kynnysjännite on 1,3 V, ja sitä suuremmilla jännitteillä riski LEDin hajoamiseen kasvaa [19]. 14,4 V signaalin kytkemiseksi jännite tuli laskea sopivalle tasolle etuvastuksen avulla. LEDin katodivirran ollessa 15 mA sain vastuksen arvoksi 873,3 Ω, joten valitsin arvoa lähimmäksi osuvan 1 kΩ vastuksen, joten ledille menee n. 13 mA virtaa. Käytin samanarvoista vastusta myös sarjaliikenteen pulssin indikoivan ledin kytkennässä. [29.]

LEDin etuvastuksen arvo voidaan määrittää yhtälöstä 2.

$$R = \frac{(U_{VF1} - U_F)}{I_F} \tag{2}$$

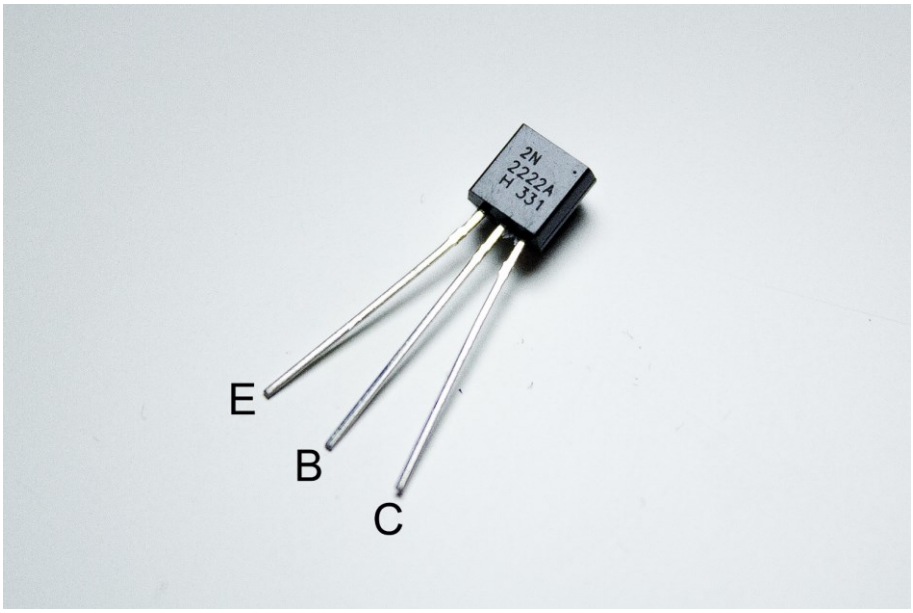
jossa	R	etuvastuksen arvo	[Ω]
	U_{VF1}	sarjaliikenteen jännite	[V]
	U_F	LEDin kynnysjännite	[V]
	I_F	katodivirta	[mA]

4.5 2N2222A-transistori

Aluksi kokeilin signaalin viemistä suoraan optoerottimelle, mutta signaalin antama virta ei kuitenkaan saanut LEDiä hohtamaan. Vahvistin signaalia erilli-

sellä NPN-transistorikytkennällä. Kytkinkäytössä ja digitaalisten signaalien siirrossa transistorista hyödynnetään sulku- ja kyllästystilat. Sulkutilassa transistori ei johda sähköä, ja sitä voidaan verrata avoimeen kytkimeen. Kyllästystilassa transistori johtaa sähköä kollektorilta (C) emitterille (E). VF1-nastan heikko virta johdetaan transistorin kannalle (B), joka saa transistorin johtamaan. [30.] Transistorin johtaessa +B-nastan 14,4 V jännite pääsee kulkemaan pulssia indikoivan LEDin ja optoerottimen lävitse.

Transistoria valittaessa tuli ottaa huomioon sen jännitteen- ja virrankesto. Jännitteenkesto U_{CE} saadaan suoraan kytkentäjännitteestä. Transistorin tulee siis kestää vähintään VF1 nastan 14,4 V. Virrankesto, eli kollektorin lävitse kulkema virta, I_C saadaan laskemalla kuormana olevien LEDien virta yhteen, tässä tapauksessa I_C on n. 30 mA. Pienitehoista transistoria valitessa arvoja kannattaa pyöristää reilusti ylöspäin, joten valitsin 2N2222A NPN -transistorin (kuva 8.), jonka U_{CE} on 50 V ja I_C 800 mA. [31.]



Kuva 8. 2N2222A-transistori

Seuraavaksi tuli selvittää kantavirta. Kantaan tuodulla ohjausvirralla säädellään kollektorivirran suuruutta. Näiden virtojen suhteesta voidaan laskea transistorin virtavahvistuskerroin h_{FE}. Pienitehoisilla transistoreilla se on tyypillisesti 100–300 [30]. Valitsemani 2N2222A-transistorin virtavahvistuskerroin on

100, kun kollektorivirta on 1 – 150 mA väliltä [31]. Kantavirran suuruus on oltava vähintään 0,30 mA, jotta saavutetaan täysi 30 mA kollektorivirta. Jos tätä ei saavuteta, osa kuormitukselle tarkoitetusta tehosta häviää lämpönä transistorissa. Koska kyseessä on pienitehoinen kytkentä ja transistoria käytetään kytkimenä, voi kantavirran mitoittaa esim. nelinkertaiseksi (1,2 mA), jotta transistori pysyy varmasti kyllästetyssä tilassa. [30.]

Kantavirran suuruus voidaan määrittää yhtälöstä 3.

$$I_B = \frac{I_C}{h_{FE}} \tag{3}$$

jossa	h_{FE}	virtavahvistuskerroin	[-]
	I_B	kantavirta	[mA]
	I_C	kollektorivirta	[mA]

Kantavirran rajoittamiseksi tuli laskea sopiva kantavastuksen arvo. Arvoa laskettaessa myös kantaan tuotu jännite vaikuttaa virran suuruuteen [30]. VF1-nastan 14,4 V jännitteestä vähennetään transistorin kannan ja emitterin välissä olevan piidiodin kynnysjännite $U_{BE(sat)}$ 0,6 V. Kantavastuksen arvoksi sain 11,5 kΩ virran ollessa 1,2 mA. Valitsin arvoa lähimmäksi osuvan 12 kΩ vastuksen ja varmistin mitoituksen oikeellisuuden laskemalla kantavirran $I_B = \frac{14{,}4\ V - 0{,}6\ V}{12\ \mathrm{k\Omega}} = 1{,}15\ mA$. Vastuksessa häviävä teho $P = (14{,}4\ V - 0{,}6\ V) \cdot 1{,}15\ mA = 0{,}012\ W$, joten tehonkestoltaan 0,25 W vastus riittää hyvin.

Kantavastuksen arvo voidaan määrittää yhtälöstä 4.

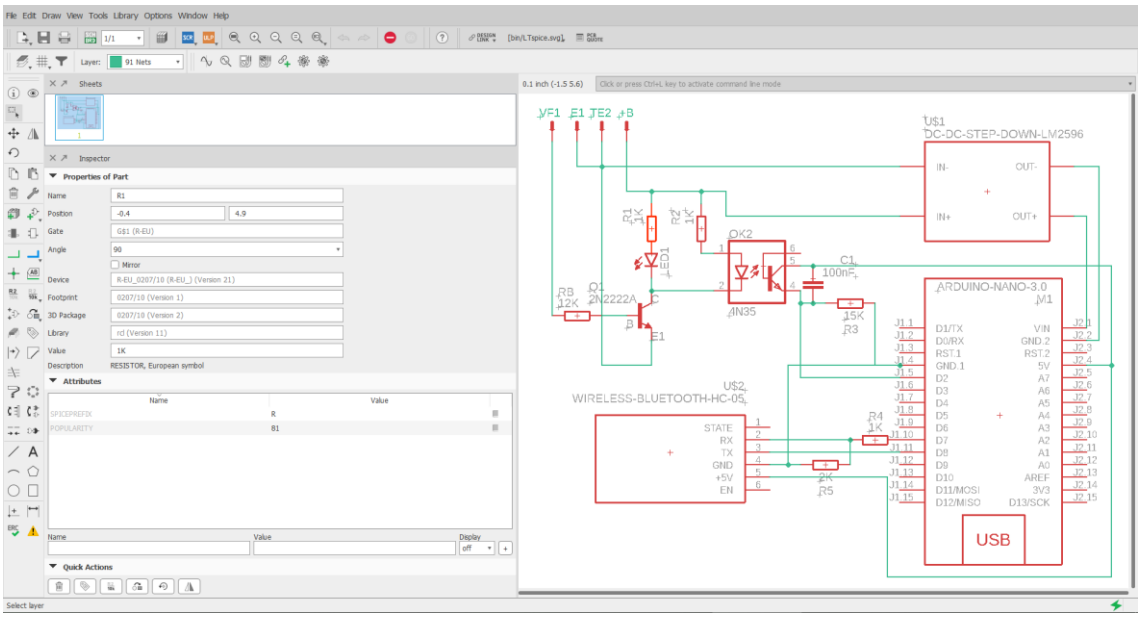
$$R_B = \frac{(U_{VF1} - U_{BE(sat)})}{I_B} \tag{4}$$

jossa	I_B	kantavirta	[mA]
	R_B	kantavastuksen arvo	[kΩ]
	U_{BE}	diodin kynnysjännite	[V]
	U_{VF1}	sarjaliikenteen jännite	[V]

5 LUKIJAN VALMISTUS

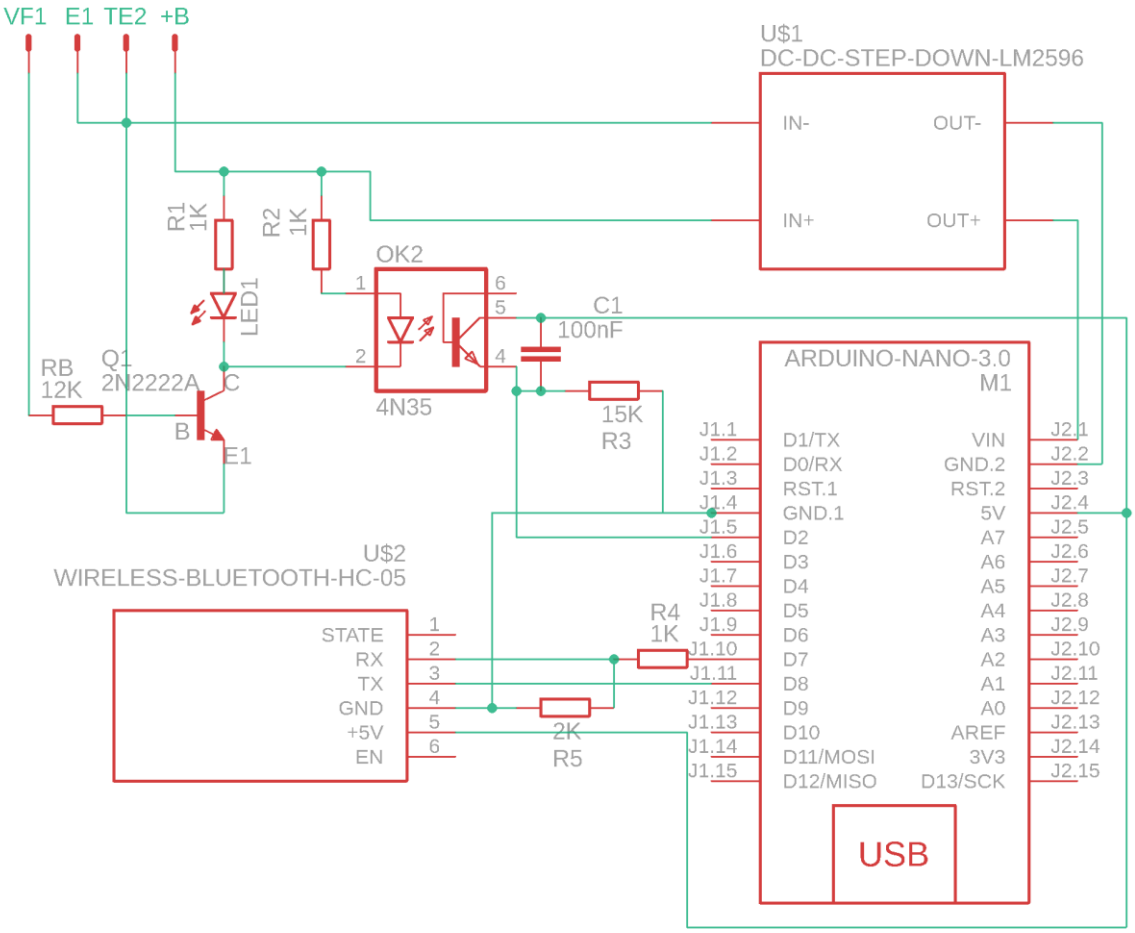
5.1 Kytkentäkaavio

Valittujen komponenttien jälkeen aloitin kytkentäkaavion piirtämisen. Kokeilin Fritzing-ohjelmaa, mutta huomasin sen olevan enemmän koekytkentäalusta-kuvien piirtämiseen tarkoitettu. Päädyin ilmaiseen Autodesk Eagle -ohjelmaan (kuva 9). Ilmaisversio sisältää kaksi kytkentä- ja signaalitasoa sekä 80 cm^2 kokoisen alueen suunnitteluun [32]. Ohjelmaan voi ladata ilmaiset komponenttikirjastot, jotka sisältävät symbolit mm. Arduinolle, HC-05- ja LM2596-moduuleille [33]. Kytkentäkaavion piirtäminen aloitetaan ”File”-välilehden ”New”-painikkeesta, jonka alta valitaan ”Schematic”.



Kuva 9. Autodesk Eagle

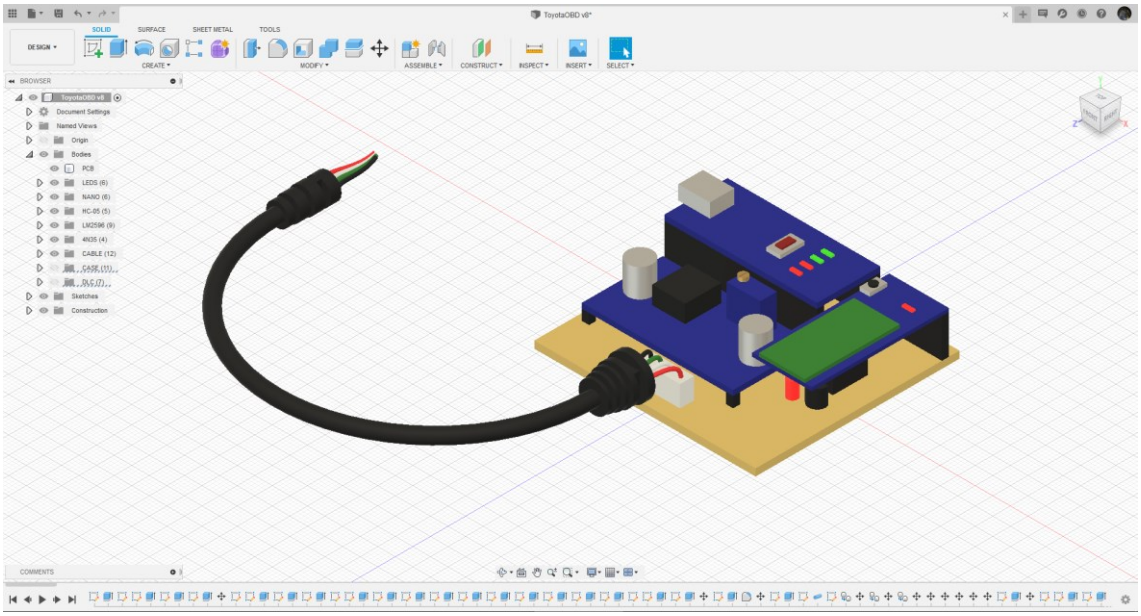
Ladatun .lbr-tiedostopäätteisen kirjaston vieminen ohjelmaan onnistuu helpoiten tuplaklikkaamalla ladattua tiedostoa. Kirjastosta ”Add to Schematic” -painikkeella symbolit viedään kytkentäkaaviokuvaan. Peruskomponentit, kuten vastukset, transistorit ja kondensaattorit, löytyivät Eaglen mukana tulevista kirjastoista, ja ne voidaan lisätä ”Add part” -painikkeella. ”Value”- ja ”Label”-parametreilla komponenteille sai määritettyä oikeat arvot ja nimet. Sijoittelin komponentit sopiville paikoille niin, että johdottaminen onnistuu siististi ”Net”-työkalulla.



Kuva 10. Kytkentäkaavio

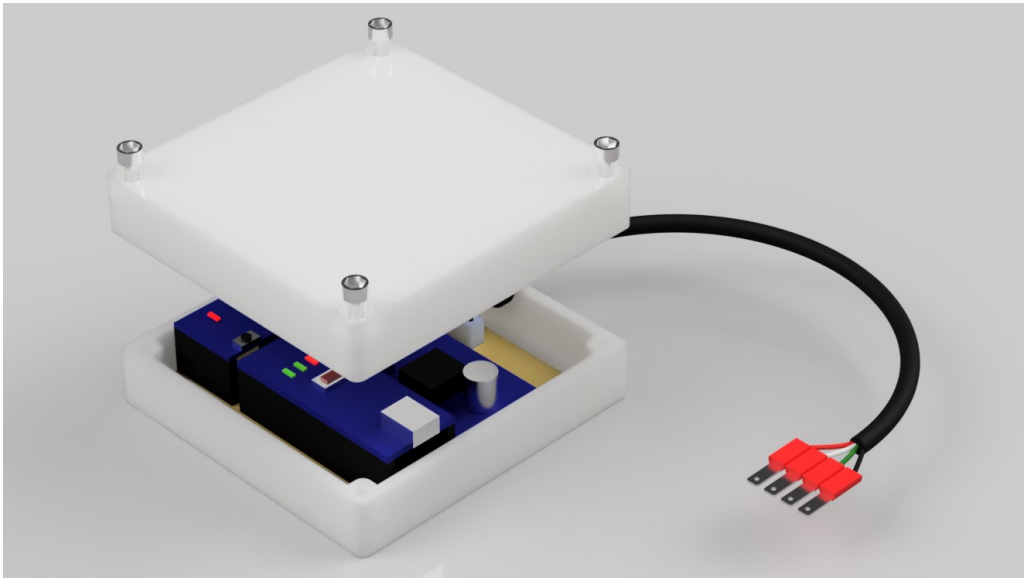
5.2 Lukijan kokoonpanomalli

Piirilevystä mallinsin suurpiirteisen 3D-mallin Fusion 360 -ohjelmalla (kuva 11.). Fusion 360 on vuodeksi ilmainen, mutta sitä ei saa käyttää kaupalliseen tarkoitukseen, ja samaan tapaan Eaglen kanssa kuvien piirtäminen on rajoitettu kahteen tasoon ja 80 cm^2 alueeseen [34]. Mallinnusta varten otin työntömitalla mittoja käytettävistä komponenteista ja määrittelin kytkentälevyn kooksi 54 mm x 64 mm. Mietin sopivat paikat Arduinolle ja moduuleille mahdollisimman kompaktin piirilevyn luomiseksi. HC-05-moduuli nousee hieman ilmaan kytkentäriman vuoksi, joten sen alle sijoitin pienempiä komponentteja.



Kuva 11. Autodesk Fusion 360 ja piirilevyn kokoonpanomalli

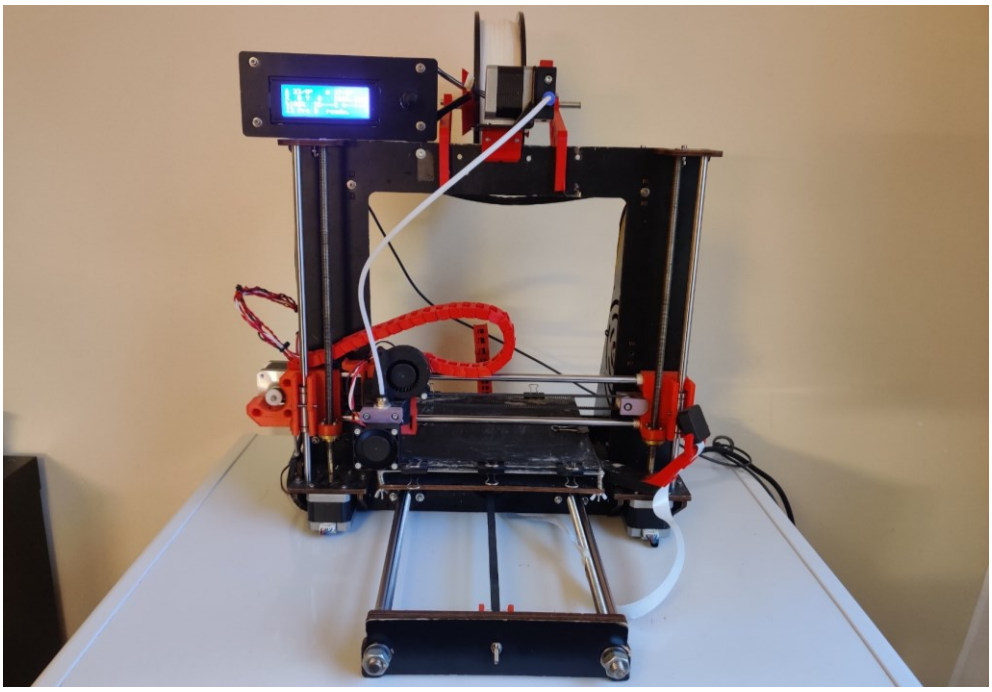
Piirilevymallin ympärille suunnittelin kotelon, joka suojaa kosteudelta, lialta ja mekaanisilta iskuilta. Kotelosta sain kompaktin 71 x 71 x 26 mm. Kotelo koostuu kahdesta puolikkaasta, jotka kiinnitetään toisiinsa neljällä M3-kuusiokoloruuvilla. Pohjassa on piirilevylle tehdyt kiinnityshakaset ja kannessa upotukset M3-kuusiokoloruuveille. USB-kaapelin vedonpoistolle tehty kolo pitää kaapelin tukevasti kiinni kotelossa. Yhdistin piirilevymallin ja kotelon Fusion 360 -ohjelmassa ja tarkistin osien sopivuuden. Yhdistettyä mallia voi käyttää kokoonpanomallina (kuva 12.).



Kuva 12. Lukijan kokoonpanomalli

5.3 Kotelon valmistus

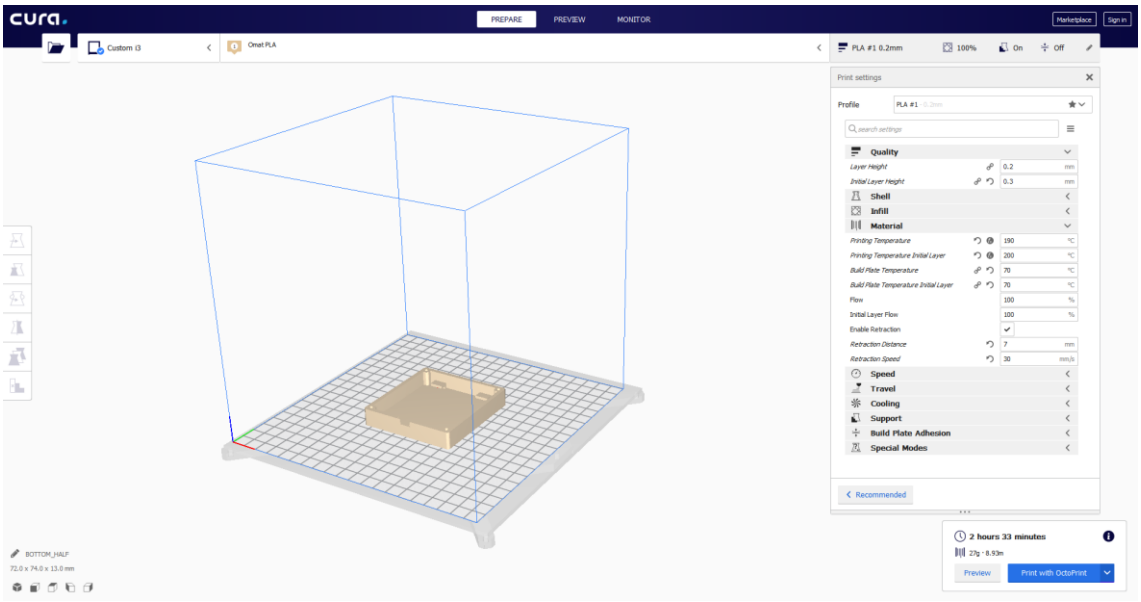
Mallinnetun kotelon valmistin FDM-tekniikkaan perustuvalla Geeetech i3 -3D-tulostimella (kuva 13.), jossa tulostusmateriaali sulatetaan ja pursotetaan askelmoottorin avulla suuttimen lävitse ohueksi 0,4 mm paksuksi nauhaksi, joka ladotaan 0,2 mm korkuisiksi kerroksiksi muodostaen kappaleen. Askelmoottoreita käytetään myös suuttimen ja tulostusalustan liikuttamiseen XYZ-suunnissa ja niiden ohjaamisen hoitaa Arduino Mega 2560 -pohjainen GT2560-piirilevy. [35.]



Kuva 13. 3D-tulostin

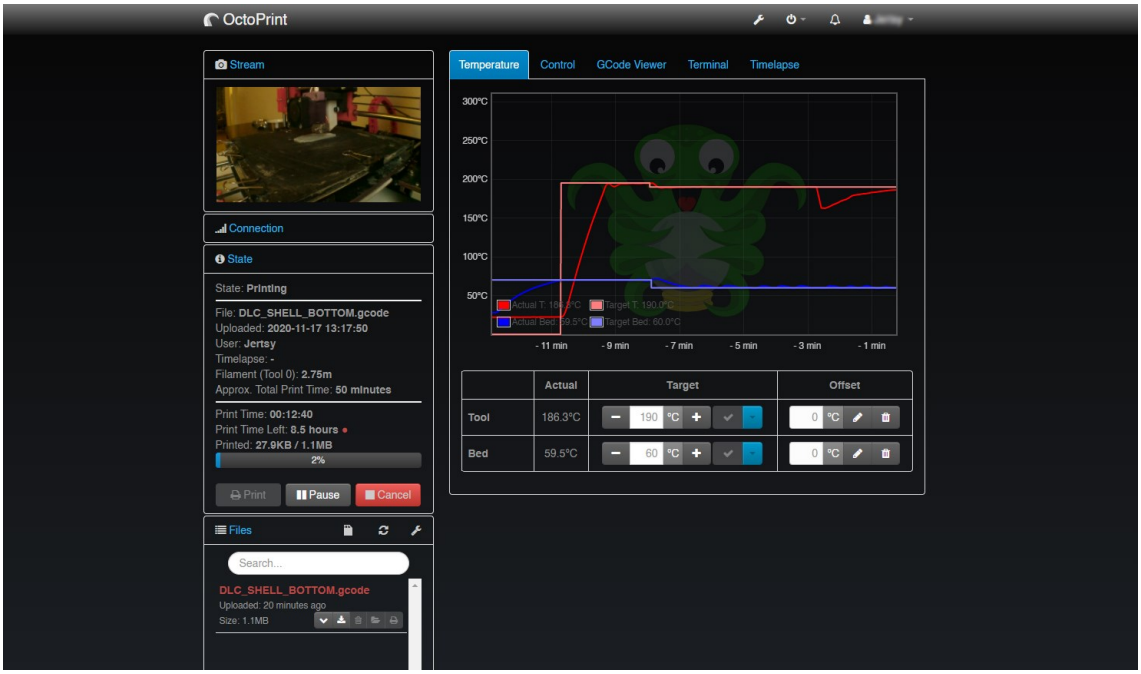
Muovia valitessa kannattaa ottaa huomioon sen kestävyys eri olosuhteissa. UV-säteily sekä korkeat ja matalat lämpötilat vaikuttavat muovin kestävyyteen heikentävästi. Kotelo sijoittuu ajoneuvon konehuoneeseen, joten se on piilossa UV-säteilyltä eikä siihen kohdistu mekaanista rasitusta. Ainoastaan kovat pakkaset voi vaikuttaa muovin kestävyyteen, joten valitsin tulostusmateriaaliksi läpikuultavan PLA-muovin. PLA on uusiutuvista raaka-aineista valmistettu biohajoava tulostusmateriaali, joka on helposti tulostettavaa ja kestävää. Visuaalisesti hyvä lopputulos on helppo saavuttaa ilman ylimääräistä jälkikäsittelyä, kuten hiomista tai maalaamista. [36.]

Fusion 360 -ohjelmasta mallinnetut kappaleet sai suoraan STL-muodossa Ultimaker Cura 4.0 slicer -ohjelmaan (kuva 14.). Slicer-ohjelma muodostaa halutujen tulostusparametrien perusteella G-code-tiedoston, jonka avulla tulostimen Arduino ohjaa askelmoottoreita sekä suuttimen ja tulostusalustan lämmitysvastuksia. Curassa määrittelin PLA-materiaalin tulostuslämpötilaksi suositellun 190 °C. Tulostusalustanlämpötilaksi määritin 70 °C ja ennen tulostamista pyyhin alustan isopropanolilla, jotta kappaleen ensimmäinen kerros tarttuu tiukasti kiinni.



Kuva 14. Cura 4.0

Tulostimeen voidaan G-code-tiedostot siirtää joko USB-kaapelilla tai muistikortilla. Tulostimeeni olen kuitenkin lisännyt Raspberry Pi 3 -minitietokoneen OctoPi-käyttöjärjestelmällä, joka lisää OctoPrint web -käyttöliittymän tulostimen ohjaukseen (kuva 15.). Curasta ”Print with Octorprint” lähettää G-code-tiedoston OctoPrint-käyttäliittymään, josta sen tulostus voidaan aloittaa. Tulostinta voidaan ohjata ja seurata reaaliaikaisesti Raspberryyn liitetyn kameran avulla. [37.]



Kuva 15. Octoprint-käyttöliittymä

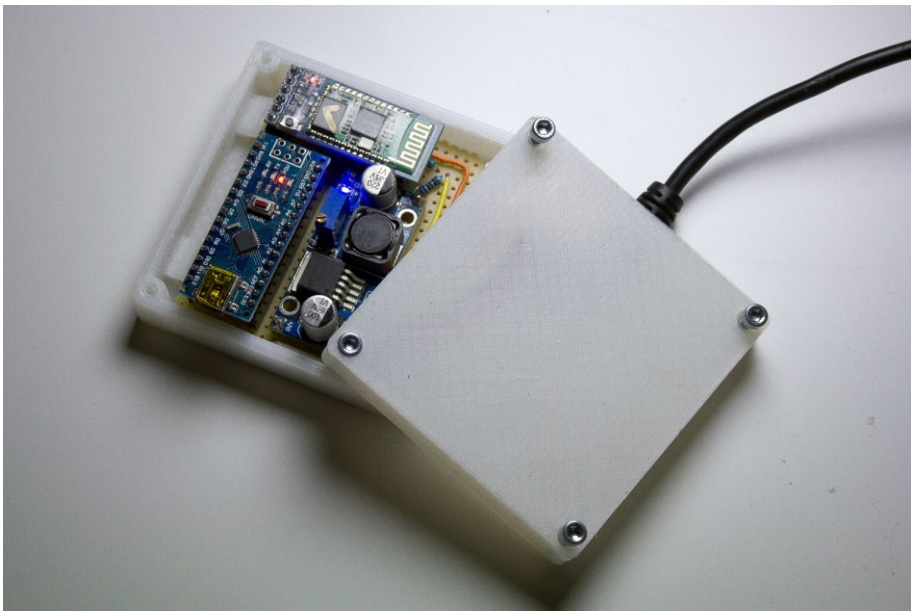
5.4 Lukijan kokoonpano

Piirilevymallin ja kytkentäkaavion avulla tein kytkennät koekytkentälevylle. Koekytkentälevyssä on valmiit juotosliuskat, joita hyödyntämällä voidaan tehdä komponenttien välisiä kytkentöjä tinaamalla. Pienempien komponenttien sijoittelun pyrin tekemään niin, että kytkentöjen tekeminen onnistuu mahdollisimman lyhyitä reittejä pitkin. Tinasin erillisiä hyppylankoja komponenttien välille, jos kytkentöjä ei voinut tehdä juotosliuskoja hyödyntämällä.

Arduinossa ja HC-05-moduulissa oli valmiit piikkirimat, joita varten tinasin piirilevyyn liitosrungot. Optoerottimelle tinasin 6-napaisen IC-kannan ja loput komponentit tinasin suoraan kytkentälevyyn. DLC-liittimen korkeampi jännite tuodaan kauimmaksi Arduinosta mahdollisten häiriöiden minimoimiseksi. Kytkentälevyn kiinnitin koteloon tehtyjen kiinnittimien sekä kahden ruuvin avulla. Valmis piirilevy kotelossaan (kuva 16.).

DLC-liittimeen vietävä kaapeli on peräisin vanhasta USB-kaapelista, joka on sopivan mittainen, sisältää vedonpoiston ja tarvittavat neljä johdinta. Punainen johdin viedään +B-nastaan, musta E1-, valkoinen TE2- ja vihreä VF1-nastaan.

Johtimien päihin puristin DLC-liittimen nastoihin sopivat 2,8 mm levyiset latta-liittimet. Aluksi tarkoituksena oli tehdä kotelo myös lattaliittimien ympärille, mutta tarkemmin ajateltuna kannattaa muut DLC-liittimen nastat pitää paljaana, jotta niitä voidaan hyödyntää, jos ajoneuvoa tarvitsee vikadiagnosoida tarkemmin.



Kuva 16. Valmis piirilevy kotelossaan

5.5 Arduino-ohjelma

Toyota-ajoneuvojen moottorin käyntitietojen lukemiseen on saatavilla Windows-pohjainen ohjelma, joka keskustelee ajoneuvon moottorinohjausyksikön kanssa tietokoneen COM RS232 -sarjaliikenneportin kautta [38]. Sarjaliikenneporttia ei juurikaan nähdä nykyaikaisissa tietokoneissa, ja on käytettävä erillistä USB-adapteria tai asennettava tietokoneeseen sarjaliikenteen mahdollistava lisäkortti. Windows-ohjelmaan perustuen on Arduinolle kirjoitettu yhteensopiva ohjelma [39], jota muokkasin omaan käyttötarkoitukseen sopivammaksi (ks. liite 2).

Ohjelman alussa määritetään sarjaliikennekirjasto bluetoothmoduulia varten #include <SoftwareSerial.h> -komennolla ja määritetään siihen tarvittavat nastat SoftwareSerial BTSerial(8, 7); -komennolla. #define-komennoilla voidaan määrittää nimet nastojen muuttujille, joiden perusteella on helpompi ymmärtää, mitä arvoja missäkin kohtaa koodia käsitellään. Esimerkiksi #define

LED_PIN 13 tarkoittaa, että Arduinon nastaan 13 voidaan ohjelmassa viitata LED_PIN-nimellä. Setup()-funktiossa määritetään sarjaliikenteen nopeus, virheiden seuranta sekä nastojen toiminta tulo- ja lähtöliitäntöinä.

Sarjaliikenteen lukeminen toteutetaan Arduinon ”Interrupt”-toimintoa hyväksikäyttäen. Toimintoa kannattaa käyttää, koska sen avulla voidaan lukea sarjaliikennettä ja samaan aikaan tehdä loop() funktiossa muita komentoja [40]. Sarjaliikenteestä luetut bitit kirjoitetaan OBDdataIDX-muuttujaan, josta ne puretaan Switch-komponentin avulla helpommin ymmärrettäviin sanoihin. Switch-komponentin sisään kirjoitetuilla case-tapahtumilla voidaan määrittää tiettyjen ohjelmapätkien suorittaminen, kun case-ehto toteutuu. Sanojen sisältämät arvot skaalataan case-ehtojen sisällä, ja skaalatut arvot voidaan lukea ohjelman alussa määritetyistä muuttujista, kuten OBD_INJ tai OBD_RPM. Sarjaliikenteen toiminnan testaamiseksi Arduinon sisäänrakennettu LED on määritetty vilkkumaan pulssin mukaan. [39.]

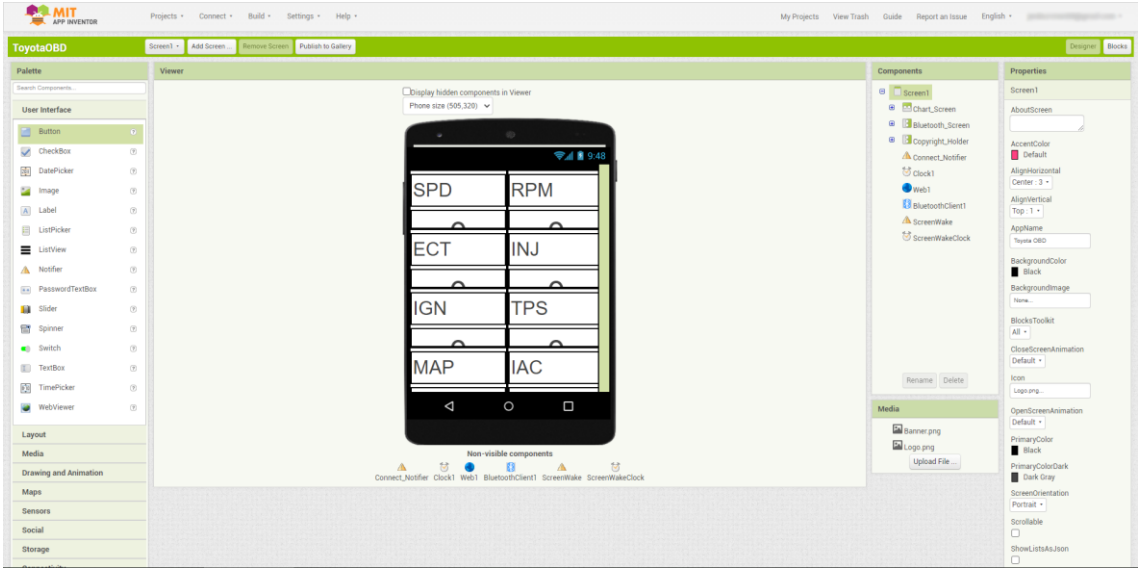
Loop()-funktion alussa kutsutaan SendBluetoothData()-funktiota, jolla saadaan skaalatut arvot lähetettyä puhelimeen. Jos-funktiolla kysellään, onko moduulin sarjaliikenne käytettävissä, ja kun lauseke toteutuu, luetaan sarjaliikenteestä vastaanotettu kirjain. Case-ehtojen avulla voidaan määrittää tietyn kirjaimen saapuessa tietyn ohjelmanpätkän suorittaminen. Tässä tapauksessa Android-sovelluksen lähettämä ”r”-kirjain käskee Arduinoa lähettämään kaikki arvot XML-muodossa bluetoothsarjaliikenteen kautta puhelimeen. Eri kirjaimia määrittelemällä ohjelmaan ja sovellukseen voitaisiin kehittää lisää toimintoja.

5.6 Android-sovellus

Android-sovellusten kehittämiseen on saatavilla lukuisia ohjelmia, joista virallisin on Googlen kehittämä Android Studio, jossa ohjelmointi tapahtuu Java- tai Kotlin-ohjelmointikielellä [41]. Ohjelmointia voidaan tehdä myös visuaalisesti Web-pohjaisella MIT App Inventor 2 -ohjelmalla. Visuaalisessa ohjelmoinnissa ohjelma tehdään sisäänrakennetuilla ohjelmalohkoilla ja komponenteilla, joita yhdistelemällä saadaan halutut toiminnallisuudet Android-sovellukseen [42]. Sovelluksen kehittäminen aloitetaan ”Projects”-välilehden alta ”Start New Project” [43].

Sovellussuunnittelu onnistuu ”Designer”-ikkunassa (kuva 17.) olevien erilaisten komponenttien avulla, jotka voidaan viedä vetämällä ne ”Palette”-paneelista puhelimen näytön kaltaiseen ”Viewer”-paneeliin. ”Properties”-paneelista komponentteja voidaan mukauttaa muuttamalla niiden parametreja. Komponentteja käsitellään ylhäältä alaspäin, ja sovellus muodostuu puumaisesta rakenteesta. Ensimmäisessä ”Screen”-komponentissa määritin sovellukselle mm. nimen, näytön orientoinnin ja logon. Kuvat ja logot vein ohjelmaan korkearesoluutioisina PNG-tiedostoina, jotka piirsin ilmaisella Inkscape-vektorigrafiikkaohjelmalla .

”HorizontalArrangement”- ja ”VerticalArrangement”-komponentteja yhdistelemällä sain sopivan sijoittelun kuville, teksteille ja numeroille. Tekstit ja muuttuvat arvot näytetään ”Label”-komponentin avulla. ”Connectivity”-paneelista lisäsin ”BluetoothClient”- ja ”WebViewer”-komponentit sovellukseen, joiden avulla bluetoothkommunikointi ja arvojen tulkinta Arduinosta onnistuu. ”Timer”-komponentilla pystyin määrittelemään erilaisia ajastuksia ohjelmalohkoille. Komponenttien toiminnallisuus voidaan määritellä ”Blocks”-välilehdellä.



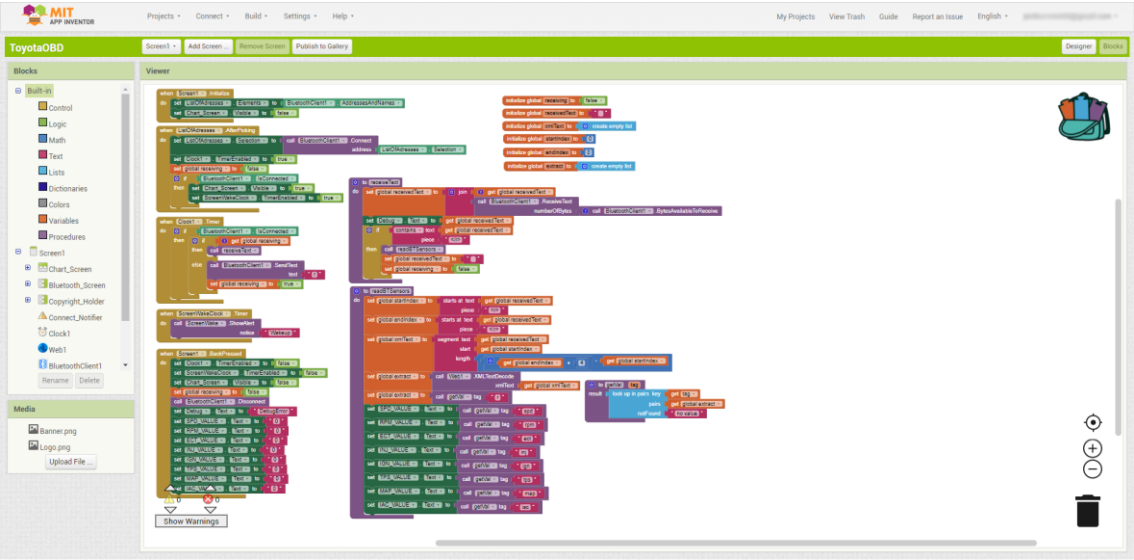
Kuva 17. MIT App Inventor 2 “Designer” -ikkuna

”Blocks”-ikkunassa (kuva 18.) ohjelmoidaan näytöllä näkyvien komponenttien toiminnallisuudet ohjelmalohkoilla (ks. liite 3). Ensimmäisenä määritin bluetoothyhteyden muodostamisen ”Screen1.Initialize”-ohjelmalohkolla, joka

hakee etusivulla näkyvään ”ListOfAdresses”-listaan vapaana olevien bluetoothlaitteiden MAC-osoitteet sekä nimet. Bluetoothosoitetta painamalla siirrytään ”ListOfAdresses.AfterPicking” lohkoon, joka käskee puhelimen bluetoothia ottamaan yhteyden lukijaan. Yhteyden onnistuessa sovellus näyttää siirtyy Chart_Screen-näyttöön, josta antureiden arvot voidaan lukea.

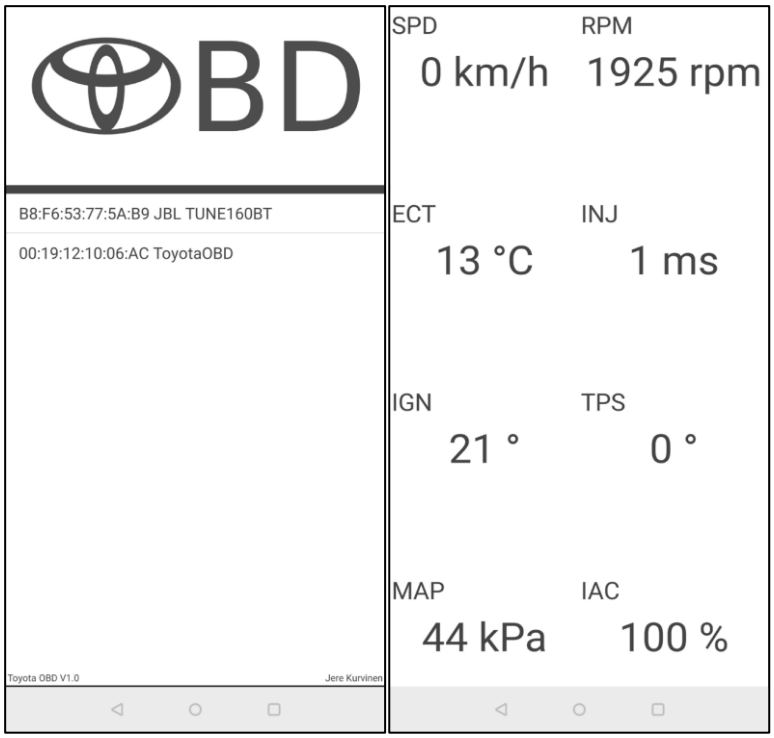
Niin kauan, kun bluetoothyhteys on muodostettuna, sovellus lähettää Clock1.Timer-lohkon mukaisesti ”r”-kirjaimen Arduinolle, joka käskee Arduinoa lähettämään dataa. ”ScreenWakeClock.Timer” pitää huolen siitä, että puhelin ei lukkiudu dataa seurattaessa. Puhelimen takaisin-painiketta painettaessa suljetaan bluetoothyhteys ja asetetaan kaikki muuttujat lähtötilaan. ”Receive-Text”-lohko odottaa Arduinolta tulevaa dataa. Jos data sisältää viimeisen </r>-tagin, se kutsuu ”readBTSensors”-lohkoa, joka lukee Arduinolta tulleet tiedot.

Samassa ”readBTSensors”-lohkossa tapahtuu arvojen muuttaminen XML-muodosta selkeämpään muotoon ”XMLTextDecode”-lohkolla. <r> - </r> -tagien välissä olevat tiedot puretaan ja kirjoitetaan sovelluksessa näkyviin ”Label”-komponentteihin. Ensimmäinen <r>-tagi kertoo sovellukselle datan alkamisesta ja </r> datan loppumisesta. Antureiden arvoille määritetyt tagit puretaan sovelluksessa näkyviin ”Label”-komponentteihin.



Kuva 18. MIT App Inventor 2 “Blocks” -ikkuna

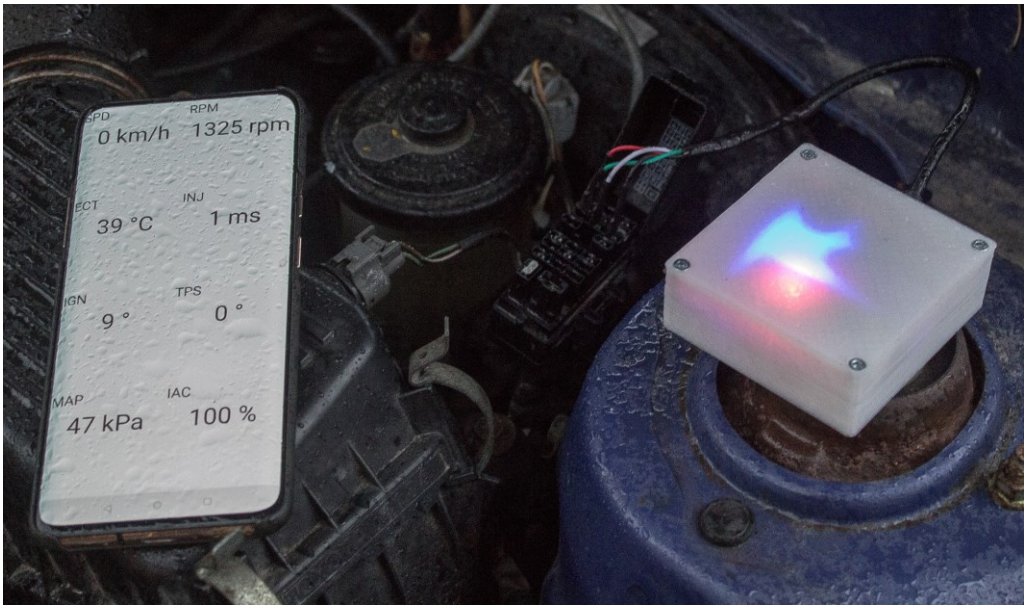
Valmiin sovelluksen koostaminen tehdään “Designer”-ikkunan ”Build”-painikkeesta, josta voidaan valita sovelluksen asennustiedoston tallennuspaikka. Helpoin tapa on käyttää ”provide QR code for .apk), joka avaa QR-koodi-ikkunan. Puhelimen QR-koodinlukijasovelluksessa voidaan avata koodi, josta aukeaa latauslinkki sovellukseen. Suoraan Android ei anna asentaa ennestään tuntemattomasta lähteestä ladattua sovellusta, vaan se pyytää käyttäjää muuttamaan asetuksia ja hyväksymään asennuksen. Asennuksen jälkeen sovellus voidaan käynnistää ja testata sen toimintaa (kuva 19.).



Kuva 19. Kuvakaappaukset Android-sovelluksesta

6 POHDINTA

Kokonaisuudessaan opinnäytetyölle asetetut tavoitteet saavutettiin. Työssä onnistuttiin valmistamaan Toyota-ajoneuvon diagnostiikkaväylän lukemiseen soveltuva Arduino-pohjainen lukulaite. Valmistunut lukija lähettää datan langattomasti bluetoothmoduulin avulla Android-sovellukseen, josta ajoneuvon antureita voidaan monitoroida reaaliaikaisesti. Valmistuneella lukijalla onnistuttiin havaitsemaan, että ajoneuvon tyhjäkäynnin kierrosnopeus 1325 rpm on suositeltua 900 rpm huomattavasti korkeampi (kuva 20.).



Kuva 20. Antureiden monitorointia lukijalla

Lukijan haluttiin olevan mahdollisimman yksinkertainen ja edullisesti toteutettavissa. Tavoitteiden saavuttamiseksi tuli suunnitteluvaiheessa valita oikeat materiaalit ja menetelmät. Valitut komponentit olivat edullisia, ja niitä on saatavissa suomalaisista elektroniikkaa myyvistä kaupoista. Kokonaiskustannuksiltaan lukijalle tuli n. 30 € hintaa. Valitut ohjelmat ovat ilmaiseksi kaikkien saatavilla, ja niiden opetteluun löytyy runsaasti materiaalia verkosta.

Lukijan toteutuksessa onnistuttiin hyvin, ja valmistus tapahtui osissa. Lopputuloksena on kattava kuvaus käytetyistä komponenteista ja työvaiheista. Kaikkia vaiheita ei välttämättä tarvitse toteuttaa. Esimerkiksi 3D-tulostetun kotelon voi korvata valmiillakin kotelolla. Selkeä kytkentäkaavio ja laitteen kokoonpanomalli helpottaa laitteen kokoamista, ja ohjelmakoodi on helposti muokattavissa, sillä sarjaliikenteen lukeminen tapahtuu erillisissä funktioissa, joten koodin lisääminen esimerkiksi loop()-funktioon onnistuu helposti.

Työssä valmistunutta lukijaa voidaan vielä kehittää monella tapaa. Piirilevy olisi hyvä teettää piirilevyvalmistajalla, jotta vältytään mahdollisilta kytkentävirheiltä kasausvaiheessa. Antureilta saatujen tietojen oikeellisuudesta ja arvojen skaalauksista voisi tehdä lisää tutkimusta. TDCL-väylästä saadaan myös enemmän dataa ulos, kuten esimerkiksi antureiden bittitiedot, jotka voisi lisätä sovellukseen. Myös ennalta tuntemattomia sanoja voitaisiin selvittää ja

Android-sovelluksen ulkoasua parantaa. Sovellukseen voisi myös lisätä hälytyksiä esimerkiksi liian korkealle lämpötilalle.

Työn aihe oli entuudestaan melko tuntematon, eikä aiempaa kokemusta ollut kuin Arduinon perusteista. Arduinon turvallinen kytkeminen ajoneuvon diagnostiikkaan vaatii monien asioiden huomioon ottamista. Myös uusien ohjelmien opetteluun ja sisäistämiseen meni huomattava määrä aikaa. Opinnäytetyöprosessi syvensi tietämystäni sarjaliikenteestä, Arduino-laitteiden toiminnasta, piirilevysuunnittelusta ja ohjelmoinnista. Lisäksi Arduinoon liittyvä käsitteistö ja lähteet tulivat hyvin tutuiksi.

LÄHTEET

1. Arduino. Introduction. WWW-dokumentti. 2020. Saatavilla: https://www.arduino.cc/en/guide/introduction [viitattu 2.8.2020].

2. Arduino. About Us. WWW-dokumentti. 2020. Saatavilla: https://www.arduino.cc/en/Main/AboutUs [viitattu 2.8.2020].

3. Blum, J. Exploring Arduino: Tools and Techniques for Engineering Wizardry, Indiana: John Wiley & Sons, Inc. 2013.

4. Gopinath, S. How to Choose the Right Arduino Board for Your Project. WWW-dokumentti. Päivitetty 21.3.2018. Saatavilla: https://maker.pro/arduino/tutorial/how-to-choose-the-right-arduino-board-for-your-project [viitattu 8.1.2020].

5. Arduino. Arduino Forum. WWW-dokumentti. Päivitetty 4.9.2020. Saatavilla: https://forum.arduino.cc/. [viitattu 4.9.2020].

6. Arduino. UNO-TH_Rev3e_sch. PDF-dokumentti. Päivitetty 6.3.2019. Saatavilla: https://content.arduino.cc/assets/UNO-TH_Rev3e_sch.pdf [viitattu 30.8.2020].

7. Guadalupi, A. NanoV3.3_sch. PDF-dokumentti. Päivitetty 23.1.2019. Saatavilla: https://content.arduino.cc/assets/NanoV3.3_sch.pdf [viitattu 30.8.2020].

8. Lady, A. Arduino UNO FAQ.. WWW-dokumentti. Päivitetty 12.1.2013. Saatavilla: https://learn.adafruit.com/arduino-tips-tricks-and-techniques/arduino-uno-faq [viitattu 30.8.2020].

9. Arduino. Arduino IDE. WWW-dokumentti. Päivitetty 16.11.2020. Saatavilla: https://www.arduino.cc/en/software [viitattu 16.11.2020].

10. Arduino. Enviroment. WWW-dokumentti. Päivitetty 5.2.2018. Saatavilla: https://www.arduino.cc/en/Guide/Environment [viitattu 12.6.2020].

11. Arduino. Variables. WWW-dokumentti. Päivitetty 11.14.2020. Saatavilla: https://www.arduino.cc/reference/en/ [viitattu 11.14.2020].

12. Autowiki. Moottorinohjausyksikkö. WWW-dokumentti. Päivitetty 30.5.2017. Saatavilla: http://www.autowiki.fi/index.php/Moottorinohjausyksikk%C3%B6 [viitattu 14.7.2020].

13. Heikkilä, J. EOBD, OBD2 ja valmistajakohtaiset protokollat. WWW-dokumentti. Päivitetty 20.1.2007. Saatavilla: http://www.obd.fi/index.php?option=com_content&task=view&id=32&Itemid=2 [viitattu 1.11.2020].

14. Wikipedia. ELM327. WWW-dokumentti. Päivitetty 2.9.2020. Saatavilla: https://en.wikipedia.org/wiki/ELM327 [viitattu 14.7.2020].

15. Toyota Kgbconsulting. OBD-1 Serial Interface. WWW-dokumentti. Päivitetty 1.11.2020. Saatavilla: http://toyota.kgbconsulting.ca/wiki/OBD-1_Serial_Interface [viitattu 1.11.2020].

16. Yotatech. OBD-I Protocol Description. PDF-dokumentti. Päivitetty 10.2.2020. Saatavilla: https://www.yotatech.com/forums/attachments/f116/100816d1418667254-successful-93-obd-reading-obd-i-protocol-description.pdf [viitattu 10.2.2020].

17. Wikipedia. Binäärijärjestelmä. WWW-dokumentti. Päivitetty 31.5.2020. Saatavilla: https://fi.wikipedia.org/wiki/Bin%C3%A4%C3%A4rij%C3%A4rjestelm%C3%A4 [viitattu 16.4.2020].

18. Jimblom. Serial Communication. WWW-dokumentti. Päivitetty 1.11.2020. Saatavilla: https://learn.sparkfun.com/tutorials/serial-communication/all [viitattu 1.11.2020].

19. Arduino. Arduino To Breadboard. WWW-dokumentti. Päivitetty 2.8.2020. Saatavilla: https://www.arduino.cc/en/Tutorial/ArduinoToBreadboard [viitattu 2.8.2020].

20. Iteadstudio. Serial Port Bluetooth Module (Master/Slave): HC-05. WWW-dokumentti. Päivitetty 24.3.2017. Saatavilla: https://www.itead.cc/wiki/Serial_Port_Bluetooth_Module_(Master/Slave)_:_HC-05 [viitattu 23.10.2020].

21. Currey, M. Arduino With HC-05 Bluetooth Module in Slave Mode. WWW-dokumentti. Päivitetty 9.27.2014. Saatavilla: http://www.martyncurrey.com/arduino-with-hc-05-bluetooth-module-in-slave-mode/ [viitattu 13.9.2020].

22. Arduino. SoftwareSerial Library. WWW-dokumentti. Päivitetty 24.12.2020. Saatavilla: https://www.arduino.cc/en/Reference/softwareSerial [viitattu 13.9.2020].

23. Morich, K. Serial Bluetooth Terminal. WWW-dokumentti. Päivitetty 27.8.2020. Saatavilla: https://play.google.com/store/apps/details?id=de.kai_morich.serial_bluetooth_terminal&hl=fi [viitattu 23.10.2020].

24. Arduino. Arduino Nano. WWW-dokumentti. Päivitetty 11.10.2020. Saatavilla: https://store.arduino.cc/arduino-nano [viitattu 11.10.2020].

25. Texas Instruments. LM1117. PDF-dokumentti. Päivitetty 11.10.2020. Saatavilla: https://www.ti.com/lit/ds/symlink/lm1117.pdf [viitattu 11.10.2020].

26. Texas Instruments. LM2596. PDF-dokumentti. Päivitetty 1.2.2020. Saatavilla: https://www.ti.com/lit/ds/symlink/lm2596.pdf [viitattu 12.10.2020].

27. Texas Instruments. Buck Converter Integrated Switch. WWW-dokumentti. Päivitetty 17.10.2020. Saatavilla: https://www.ti.com/power-management/non-isolated-dc-dc-switching-regulators/step-down-buck/buck-converter-integrated-switch/overview.html [viitattu 17.10.2020].

28. Electronics Notes. Photocouplers, Opto-couplers & Opto-isolators. WWW-dokumentti. Päivitetty 6.10.2020. Saatavilla: https://www.electronics-notes.com/articles/electronic_components/transistor/what-is-a-photocoupler-optocoupler-optoisolator.php [viitattu 6.10.2020].

29. Vishay Intertechnology. 4N35. PDF-dokumentti. Päivitetty 1.1.2019. Saatavilla: https://www.vishay.com/docs/81181/4n35.pdf [viitattu 7.10.2020].

30. Huhtama, K. Transistorin käyttö kytkimenä. WWW-dokumentti. Päivitetty 1.11.2020. Saatavilla: https://huhtama.kapsi.fi/ele/index.php?si=ml26.sis [viitattu 1.11.2020].

31. ON Semiconductor. 2N2222A. PDF-dokumentti. Päivitetty 14.2.2015. Saatavilla: http://web.mit.edu/6.101/www/reference/2N2222A.pdf [viitattu 11.13.2020].

32. Autodesk Inc,. Eagle. WWW-dokumentti. Päivitetty 9.11.2020. Saatavilla: https://www.autodesk.com/products/eagle/overview [viitattu 9.11.2020].

33. DIYmodules. DIY Modules Library for EAGLE PCB Design Software. WWW-dokumentti. Päivitetty 9.11.2020. Saatavilla: https://www.diymodules.org/eagle [viitattu 9.11.2020].

34. Autodesk Inc,. Fusion 360. WWW-dokumentti. Päivitetty 8.11.2020. Saatavilla: https://www.autodesk.com/products/fusion-360/personal [viitattu 8.11.2020].

35. Varotsis, A. Introduction to FDM 3D printing. WWW-dokumentti. Päivitetty 18.10.2020. Saatavilla: https://www.3dhubs.com/knowledge-base/introduction-fdm-3d-printing [viitattu 18.10.2020].

36. 3d Matter. FDM 3D printing materials compared. WWW-dokumentti. Päivitetty 18.10.2020. Saatavilla: https://www.3dhubs.com/knowledge-base/fdm-3d-printing-materials-compared/ [viitattu 18.10.2020].

37. Häußge, G. OctoPrint. WWW-dokumentti. Päivitetty 10.9.2020. Saatavilla: https://octoprint.org/ [viitattu 10.9.2020].

38. Chem407. ToyotaOBD Flagship implementation. WWW-dokumentti. Päivitetty 2008. Saatavilla: http://www.carina-e.ru/viewtopic.php?f=6&t=1145 [viitattu 9.8.2020].

39. GadgetFreak. Reading OBD(1) data from Toyota Corolla 1992. WWW-dokumentti. Päivitetty 3.5.2014. Saatavilla: https://forum.arduino.cc/index.php?topic=237539.0 [viitattu 9.8.2020].

40. Gammon, N. Interrupts. WWW-dokumentti. Päivitetty 8.1.2012. Saatavilla: http://gammon.com.au/interrupts [viitattu 11.14.2020].

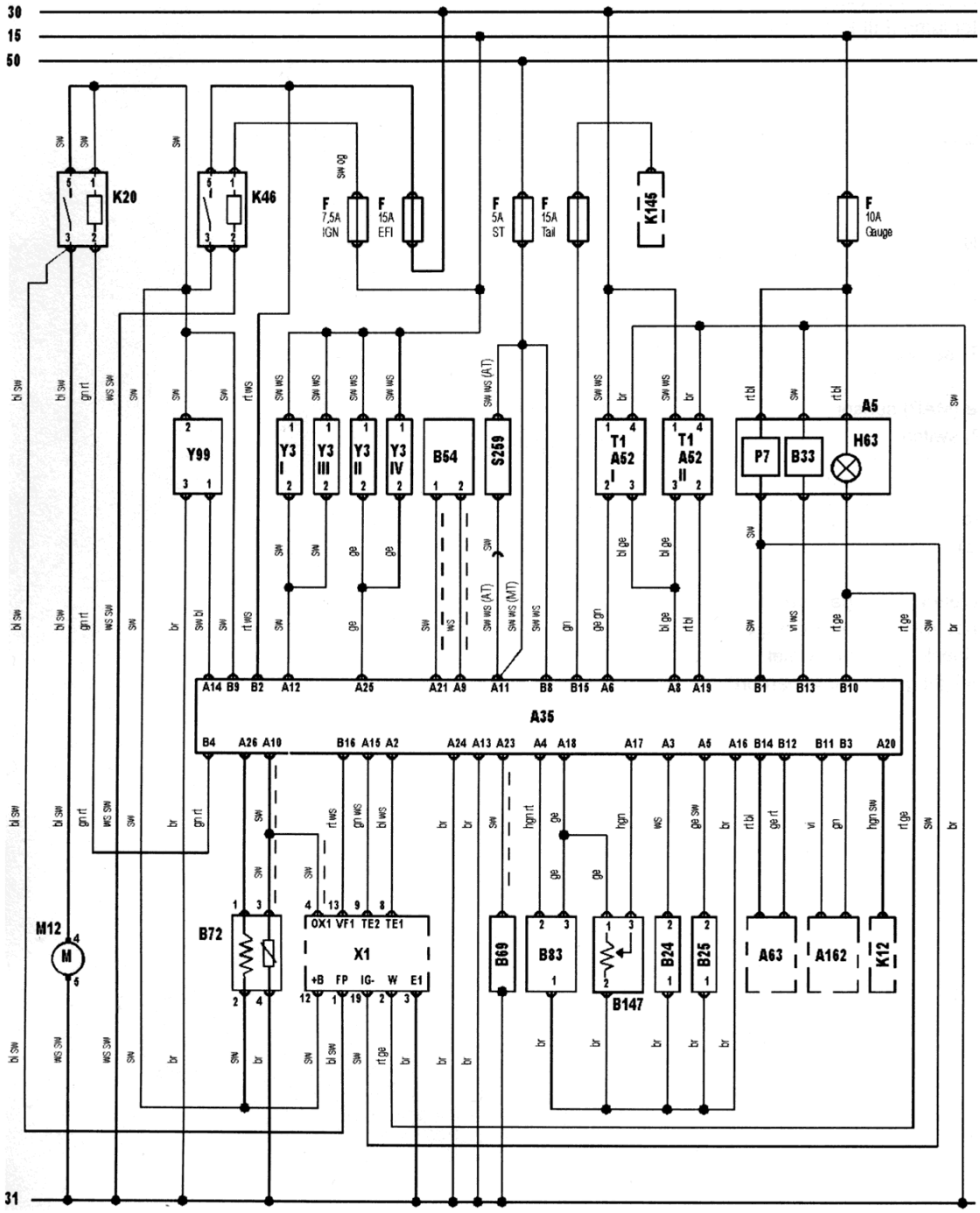
41. Google Android. Meet Android Studio. WWW-dokumentti. Päivitetty 2020. Saatavilla: https://developer.android.com/studio/intro [viitattu 3.8.2020].

42. Massachusetts Institute of Technology. About Us. WWW-dokumentti. 2020. Saatavilla: https://appinventor.mit.edu/explore/about-us [viitattu 3.8.2020].

43. Massachusetts Institute of Technology. Mit App Inventor. WWW-dokumentti. 2020. Saatavilla: http://ai2.appinventor.mit.edu/ [viitattu 16.2.2020].

KUVALUETTELO

Liite 1/1



A63	AC control module	Y99	Idle air control (IAC) valve
30	Battery +	A52	Ignition amplifier
31	Battery -	T1	Ignition coil
B54	Crankshaft position (CKP) sensor	15	Ignition switch - ignition ON
X1	Data link connector (DLC)	50	Ignition switch - start signal
A35	Engine control module (ECM)	A162	Immobilizer control module
K46	Engine control relay	Y3	Injector
K12	Engine coolant blower motor relay	B25	Intake air temperature (IAT) sensor
B24	Engine coolant temperature (ECT) sensor	B69	Knock sensor (KS)
H63	Engine malfunction indicator lamp (MIL)	B83	Manifold absolute pressure (MAP) sensor
M12	Fuel pump	S259	Park/neutral position (PNP) switch
K20	Fuel pump relay	K145	Side/tail lamps relay
F	Fuse	P7	Tachometer
B72	Heated oxygen sensor (HO2S)	B147	Throttle position (TP) sensor

Liite 2/1

```
#include <SoftwareSerial.h> // Lisätään sarjaliikenne kirjasto

SoftwareSerial BTSerial(8, 7); // Sarjaliikenteen määritys nastoihin (8 =
RX, 7 = TX)

void setup()
{
  Serial.begin(9600);           // Arduinon sarjaliikenteen siirtonopeus.
  Serial.println("AT:");
  BTSerial.begin(38400);        // HC-05 vakio sarjaliikenteen siirtonopeus
AT tilassa.
}

void loop()
{
  if (BTSerial.available())     // Vastausten luku moduulilta ja näyttäminen
Arduinon sarjamonitorissa.
  Serial.write(BTSerial.read());

  if (Serial.available())      // Komentojen luku sarjamonitorista ja lähe-
tys moduulille.
  BTSerial.write(Serial.read());
}
```

Liite 2/2

```
#include <SoftwareSerial.h> // Lisätään sarjaliikenne kirjasto

SoftwareSerial BTSerial(8, 7); // Bluetooth sarjaliikenteen määritys nas-
toihin (8 = RX, 7 = TX)

#define ENGINE_DATA_PIN 2 // VF1 nasta
#define ENGINE_DATA_INT 0  // Interrupt määritys
#define LED_PIN 13 // Arduinon sisäänrakennettu LED

// Nimet käytettäville TDCL sanoille
#define OBD_INJ 1 // Suuttimen pulssin leveys (INJ)
#define OBD_IGN 2 // Sytytysennakko (IGN)
#define OBD_IAC 3 // Tyhjäkäynnin ilmanohjaus (IAC)
#define OBD_RPM 4 // Moottorin kierrosnopeus (RPM)
#define OBD_MAP 5 // Imusarjan absoluuttinen paine (MAP)
#define OBD_ECT 6 // Moottorin jäähdytysnesteen lämpötila (ECT)
#define OBD_TPS 7 // Kaasuläpän asento (TPS)
#define OBD_SPD 8 // Nopeus (SPD)

// Sarjaliikenteen sanojen lukuun tarvittavien muuttujien määritys
#define DEBUG_OUTPUT true
#define  TOYOTA_MAX_BYTES  24
volatile uint8_t ToyotaNumBytes, ToyotaID;
volatile uint8_t ToyotaData[TOYOTA_MAX_BYTES];
volatile uint16_t ToyotaFailBit = 0;
boolean OBDConnected;
unsigned long OBDLastSuccessPacket;

// Setup funktion määritys
void setup() {
  BTSerial.begin(38400); // Bluetoothin sarjaliikenne nopeus
  Serial.begin(38400);   // Arduino sarjamonitoroinnin nopeus
  if (DEBUG_OUTPUT) {
    Serial.println("System started");}
  // setup input and output pins
  pinMode(ENGINE_DATA_PIN, INPUT); // Määritetään VF1 nasta sisäänmenoksi
  pinMode(LED_PIN, OUTPUT); // Määritetään LED nasta ulostuloksi
  attachInterrupt(ENGINE_DATA_INT, ChangeState, CHANGE); // Interrupt toi-
minto sarjaliikenteen lukemiseen
  OBDConnected=false;
}


// Loop funktion määritys
void loop() {

  SendBluetoothData(); // Kutsutaan SendBluetoothData funktiota

  // if found bytes
  if (ToyotaNumBytes > 0)  {
    if (DEBUG_OUTPUT) { debugdataoutput();  }
     // set last success
     OBDLastSuccessPacket = millis();
     // set connected to true
     OBDConnected = true;
     // reset the counter.
     ToyotaNumBytes = 0;
  } // end if (ToyotaNumBytes > 0)
  // if found FAILBIT and debug
  if (ToyotaFailBit > 0 && DEBUG_OUTPUT )  { debugfaildataoutput(); }
  //check for lost connection
```

Liite 2/3

```
 if (OBDLastSuccessPacket + 3500 < millis() && OBDConnected) {
     // set OBDConnected to false
     OBDConnected = false;
  } // end if lost connection

}

void SendBluetoothData(){ // Datan lähetys bluetoothilla funktio
  if (BTSerial.available()) // Jos-sarjaliikenne käytettävissä
  {
    int inByte = BTSerial.read(); // Luetaan vastaanotetut tiedot
    switch ((char)inByte)         // Vastaanotettu tavu muutetaan kirjai-
meksi
    {
case 'r': // Kirjaimen ollessa r, suoritetaan alla oleva funktio.
BTSerial.println("<r>");
BTSerial.print("<spd>");
BTSerial.print(int(getOBDdata(OBD_SPD)));
BTSerial.print(" km/h");
BTSerial.println("</spd>");
BTSerial.print("<rpm>");
BTSerial.print(int(getOBDdata(OBD_RPM)));
BTSerial.print(" rpm");
BTSerial.println("</rpm>");
BTSerial.print("<ect>");
BTSerial.print(int(getOBDdata(OBD_ECT)));
BTSerial.print(" °C");
BTSerial.println("</ect>");
BTSerial.print("<inj>");
BTSerial.print(int(getOBDdata(OBD_INJ)));
BTSerial.print(" ms");
BTSerial.println("</inj>");
BTSerial.print("<ign>");
BTSerial.print(int(getOBDdata(OBD_IGN)));
BTSerial.print(" °");
BTSerial.println("</ign>");
BTSerial.print("<tps>");
BTSerial.print(int(getOBDdata(OBD_TPS)));
BTSerial.print(" °");
BTSerial.println("</tps>");
BTSerial.print("<map>");
BTSerial.print(int(getOBDdata(OBD_MAP)));
BTSerial.print(" kPa");
BTSerial.println("</map>");
BTSerial.print("<iac>");
BTSerial.print(int(getOBDdata(OBD_IAC)));
BTSerial.print(" %");
BTSerial.println("</iac>");
BTSerial.println("</r>");
break;
}}}

// Sarjaliikenteen luku
void ChangeState()
{
  static uint8_t ID, EData[TOYOTA_MAX_BYTES];
  static boolean InPacket = false;
  static unsigned long StartMS;
  static uint16_t BitCount;

  int state = digitalRead(ENGINE_DATA_PIN);
```

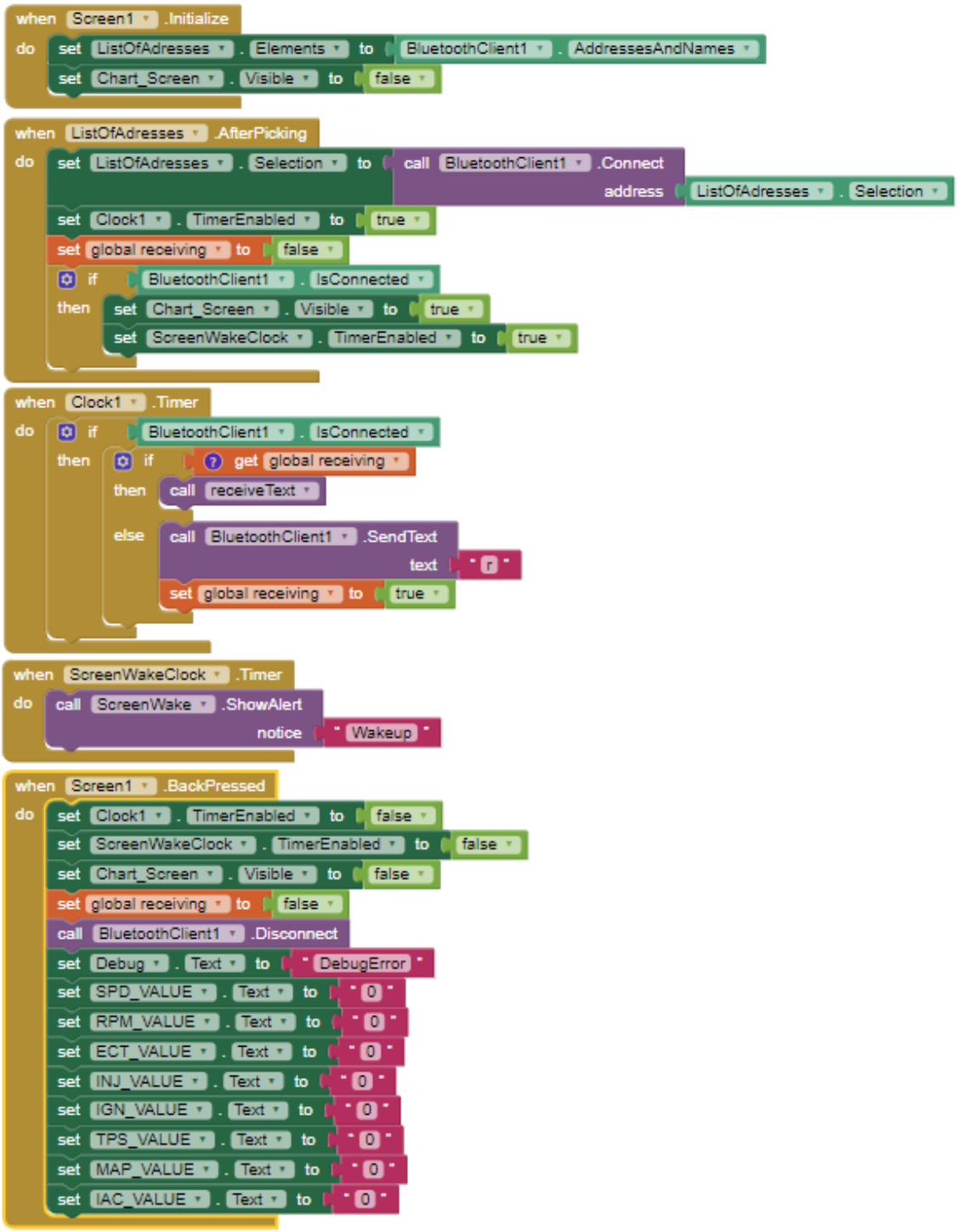
Liite 2/4

```
 digitalWrite(LED_PIN, state); // LED päällä, kun sarjaliikenteen bitti 1.

  if (InPacket == false)  {
    if (state == HIGH)   {
      StartMS = millis();
     }   else   { // else  if (state == HIGH)
       if ((millis() - StartMS) > (15 * 8))   {
          StartMS = millis();
          InPacket = true;
          BitCount = 0;
       } // end if  ((millis() - StartMS) > (15 * 8))
    } // end if  (state == HIGH)
  }  else   { // else  if (InPacket == false)
    uint16_t bits = ((millis() - StartMS)+1 ) / 8;  // The +1 is to cope
with slight time errors
    StartMS = millis();
    // process bits
    while (bits > 0)  {
      if (BitCount < 4)  {
        if (BitCount == 0)
          ID = 0;
        ID >>= 1;
        if (state == LOW)  // inverse state as we are detecting the change!
          ID |= 0x08;
      }   else    { // else    if (BitCount < 4)
        uint16_t bitpos = (BitCount - 4) % 11;
        uint16_t bytepos = (BitCount - 4) / 11;
        if (bitpos == 0)      {

          // Start bit, should be LOW
           if ((BitCount > 4) && (state != HIGH))  { // inverse state as we
are detecting the change!
              ToyotaFailBit = BitCount;
              InPacket = false;
              break;
           } // end if ((BitCount > 4) && (state != HIGH))

        }  else if (bitpos < 9)  { //else TO  if (bitpos == 0)

           EData[bytepos] >>= 1;
           if (state == LOW)  // inverse state as we are detecting the
change!
              EData[bytepos] |= 0x80;

       } else { // else if (bitpos == 0)

          // Stop bits, should be HIGH
          if (state != LOW)  { // inverse state as we are detecting the
change!
            ToyotaFailBit = BitCount;
            InPacket = false;
            break;
          } // end if (state != LOW)

          if ( (bitpos == 10) && ((bits > 1) || (bytepos ==
(TOYOTA_MAX_BYTES - 1))) ) {
            ToyotaNumBytes = 0;
            ToyotaID = ID;
            for (int i=0; i<=bytepos; i++)
              ToyotaData[i] = EData[i];
            ToyotaNumBytes = bytepos + 1;
```

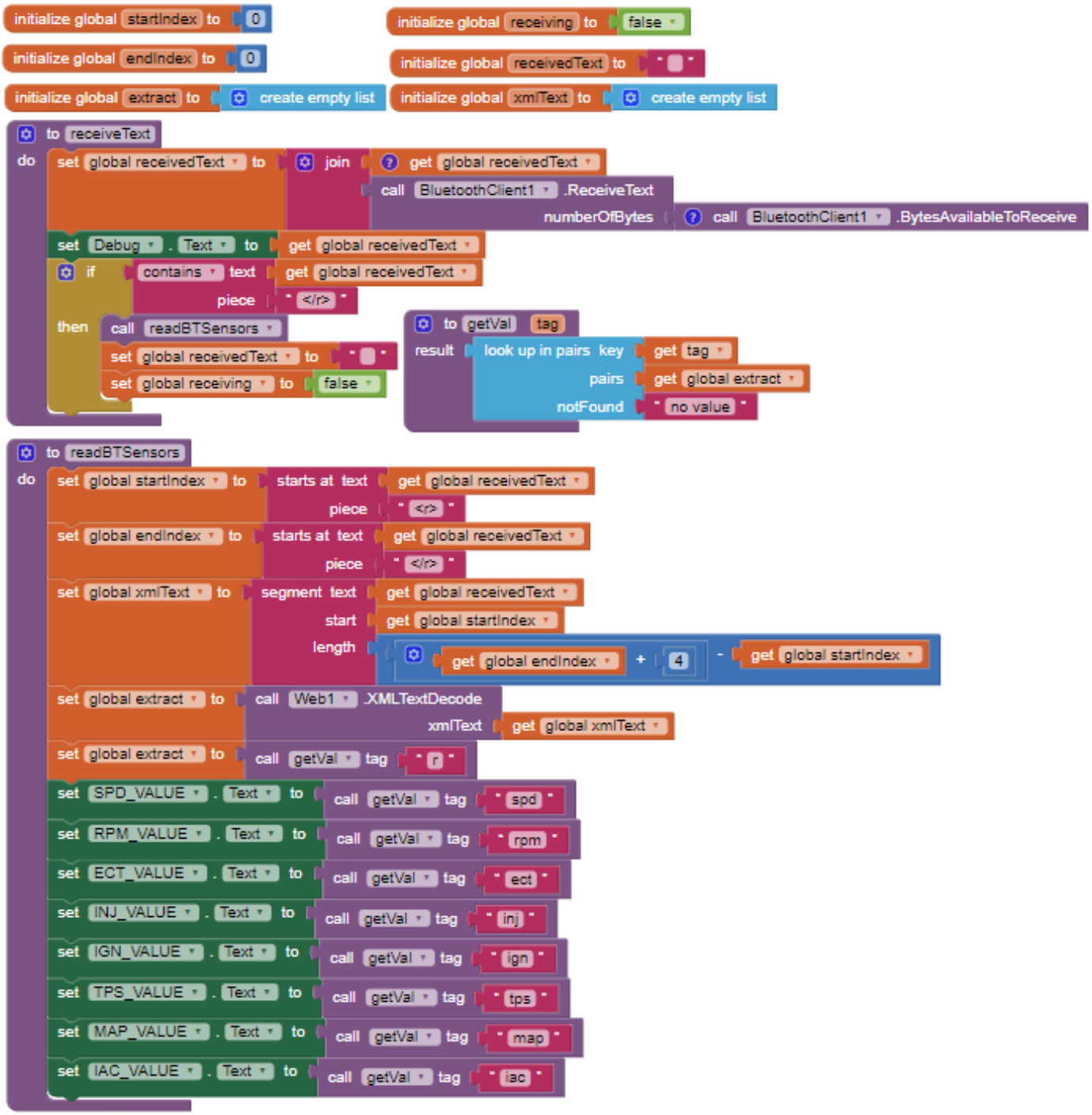
```
            if (bits >= 16)  // Stop bits of last byte were 1's so detecting
for next packet
              BitCount = 0;
            else  {
              ToyotaFailBit = BitCount;
              InPacket = false;
            }
            break;
          }
        }
      }
      ++BitCount;
      --bits;
    } // end while
  } // end (InPacket == false)
} // end void change

// GET DATA FROM OBD
float getOBDdata(int OBDdataIDX) {
 // define return value
  float returnValue;
  switch (OBDdataIDX) {
    case 0:// UNKNOWN
        returnValue = ToyotaData[0];
        break;
    case OBD_INJ: //  Injector pulse width (INJ) - in milisec
        returnValue = ToyotaData[OBD_INJ]/10;
        break;
    case OBD_IGN: // Ignition timing angle (IGN) - degree- BTDC
        returnValue = ToyotaData[OBD_IGN]-90;
        break;
    case OBD_IAC: //Idle Air Control (IAC) - Step # X = 125 = open 100%
        returnValue = ToyotaData[OBD_IAC]/125*100;
         break;
     case OBD_RPM: //Engine speed (RPM)
         returnValue = ToyotaData[OBD_RPM]*25;
         break;
     case OBD_MAP: //Manifold Absolute Pressure (MAP) - kPa Abs
         returnValue = ToyotaData[OBD_MAP];
          break;
    case OBD_ECT: // Engine Coolant Temperature (ECT)
          if (ToyotaData[OBD_ECT] >= 244)
            returnValue = ((float)(ToyotaData[OBD_ECT] - 244) * 10.0) +
132.0;
          else if (ToyotaData[OBD_ECT] >= 238)
            returnValue = ((float)(ToyotaData[OBD_ECT] - 238) * 4.0) +
103.0;
          else if (ToyotaData[OBD_ECT] >= 228)
            returnValue = ((float)(ToyotaData[OBD_ECT] - 228) * 2.1) +
80.0;
          else if (ToyotaData[OBD_ECT] >= 210)
            returnValue = ((float)(ToyotaData[OBD_ECT] - 210) * 1.11) +
60.0;
          else if (ToyotaData[OBD_ECT] >= 180)
            returnValue = ((float)(ToyotaData[OBD_ECT] - 180) * 0.666) +
40.0;
          else if (ToyotaData[OBD_ECT] >= 135)
            returnValue = ((float)(ToyotaData[OBD_ECT] - 135) * 0.444) +
20.0;
          else if (ToyotaData[OBD_ECT] >= 82)
```

```
           returnValue = ((float)(ToyotaData[OBD_ECT] - 82) * 0.377) + 0.0;
          else if (ToyotaData[OBD_ECT] >= 39)
            returnValue = ((float)(ToyotaData[OBD_ECT] - 39) * 0.465) + (-
20.0);
          else if (ToyotaData[OBD_ECT] >= 15)
            returnValue = ((float)(ToyotaData[OBD_ECT] - 15) * 0.833) + (-
40.0);
          else
            returnValue = ((float)ToyotaData[OBD_ECT] * 2.0) + (-70.0);
          break;
    case OBD_TPS: // Throttle Position Sensor (TPS) - DEGREE
          returnValue = ToyotaData[OBD_TPS]/2;
          break;
    case OBD_SPD: // Speed (SPD) - km/h
          returnValue = ToyotaData[OBD_SPD];
          break;
    case 9:// UNKNOWN
         returnValue = ToyotaData[9];
         break;
    case 10:// UNKNOWN
         returnValue = ToyotaData[10];
         break;
    case 11:// FLAG #1
         returnValue = ToyotaData[11];
         break;
    case 12:// FLAG # 2
         returnValue = ToyotaData[12];
         break;
    default: // DEFAULT CASE (in no match to number)
      // send "error" value
      returnValue =  9999.99;
   } // end switch
  // send value back
  return returnValue;
} // end void getOBDdata

// Debug
void debugdataoutput() {
   // output to Serial.
    Serial.print("ID=");
    Serial.print(ToyotaID);
    for (int i=0; i<ToyotaNumBytes; i++)
    {
      Serial.print(", ");
      Serial.print(ToyotaData[i]);
    }
     Serial.println(".");
} // end void

void debugfaildataoutput() {
        Serial.print("FAIL = ");
        Serial.print(ToyotaFailBit);
        if (((ToyotaFailBit - 4) % 11) == 0)
        Serial.print(" (StartBit)");
        else if (((ToyotaFailBit - 4) % 11) > 8)
        Serial.print(" (StopBit)");
        Serial.println(".");
        ToyotaFailBit = 0;
} // end void
```

Liite 3/1


when Screen1 .Initialize
do set ListOfAdresses . Elements to BluetoothClient1 . AddressesAndNames
set Chart_Screen . Visible to false
when ListOfAdresses .AfterPicking
do set ListOfAdresses . Selection to call BluetoothClient1 .Connect
address ListOfAdresses . Selection
set Clock1 . TimerEnabled to true
set global receiving to false
if BluetoothClient1 . IsConnected
then set Chart_Screen . Visible to true
set ScreenWakeClock . TimerEnabled to true
when Clock1 .Timer
do if BluetoothClient1 . IsConnected
then if get global receiving
then call receiveText
else call BluetoothClient1 .SendText
text "r"
set global receiving to true
when ScreenWakeClock .Timer
do call ScreenWake .ShowAlert
notice "Wakeup"
when Screen1 .BackPressed
do set Clock1 . TimerEnabled to false
set ScreenWakeClock . TimerEnabled to false
set Chart_Screen . Visible to false
set global receiving to false
call BluetoothClient1 .Disconnect
set Debug . Text to "DebugError"
set SPD_VALUE . Text to "0"
set RPM_VALUE . Text to "0"
set ECT_VALUE . Text to "0"
set INJ_VALUE . Text to "0"
set IGN_VALUE . Text to "0"
set TPS_VALUE . Text to "0"
set MAP_VALUE . Text to "0"
set IAC_VALUE . Text to "0"

Liite 3/2


initialize global startIndex to 0
initialize global endIndex to 0
initialize global extract to create empty list
initialize global receiving to false
initialize global receivedText to
initialize global xmlText to create empty list
to receiveText
do
set global receivedText to join get global receivedText
call BluetoothClient1 .ReceiveText
numberOfBytes call BluetoothClient1 .BytesAvailableToReceive
set Debug . Text to get global receivedText
if contains text get global receivedText
piece </r>
then call readBTSensors
set global receivedText to
set global receiving to false
to getVal tag
result look up in pairs key get tag
pairs get global extract
notFound no value
to readBTSensors
do
set global startIndex to starts at text get global receivedText
piece <r>
set global endIndex to starts at text get global receivedText
piece </r>
set global xmlText to segment text get global receivedText
start get global startIndex
length get global endIndex + 4 - get global startIndex
set global extract to call Web1 .XMLTextDecode
xmlText get global xmlText
set global extract to call getVal tag r
set SPD_VALUE . Text to call getVal tag spd
set RPM_VALUE . Text to call getVal tag rpm
set ECT_VALUE . Text to call getVal tag ect
set INJ_VALUE . Text to call getVal tag inj
set IGN_VALUE . Text to call getVal tag ign
set TPS_VALUE . Text to call getVal tag tps
set MAP_VALUE . Text to call getVal tag map
set IAC_VALUE . Text to call getVal tag iac