Opinnäytetyö (AMK)

Ajoneuvo- ja kuljetustekniikka

2023

Ville Karikoski

OBD-II-vikadiagnoosiohjelmiston luominen Windows-käyttöjärjestelmälle

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

Opinnäytetyö (AMK) | Tiivistelmä

Turun ammattikorkeakoulu

Ajoneuvo- ja kuljetustekniikka

Joulukuu 2023 | 32 sivua

Ville Karikoski

OBD-II-vikadiagnoosiohjelmiston luominen Windows-käyttöjärjestelmälle

Opinnäytetyön aiheena oli luoda OBD-II-vikadiagnoosiohjelmisto, jonka avulla voidaan käsitellä vikakoodeja ja anturitietoja. Ohjelmisto luotiin Windows-käyttöjärjestelmälle ja sen käyttöliittymä perustui Windows Forms -luokkakirjastoon. Kommunikaatio tietokoneen ja CAN-väylän välillä toteutettiin ELM327-mikropiirin avulla. Työn aiheeseen päädyttiin, koska haluttiin harjoitella ohjelmointia hieman laajemman projektin parissa.

Opinnäytetyössä käsiteltiin hieman vikadiagnoosijärjestelmien historiaa. Lisäksi perehdyttiin yleisellä tasolla sarjamuotoiseen tiedonsiirtoon ja tietoliikenneväyliin. Tietoliikenneväylistä tarkemmin tutustutttiin CAN-väylään, koska se oli työn aiheen kannalta oleellisin. Digitaalitekniikkaa käsitellään sen verran, mitä työn sisällön kannalta olisi hyvä ymmärtää. Käytännön osuudessa tutustuttiin ohjelmiston käyttöliittymään ja pääominaisuuksiin ja tarkasteltiin, miten ne saatiin toteutettua.

Ohjelmisto saatiin täyttämään tavoitteiden mukaiset vaatimukset, mutta siihen jäi aikapuutteen vuoksi vielä paljon puutteita ja kehityskohteita, joita avattiin työn lopussa. Ohjelmistoa päästiin testaamaan myös oikean ajoneuvon kanssa ja koe-tilanteen yhteydessä kerätystä datasta saatiin luotua Excel-ohjelmistossa kaavio. Todettiin, että projekti opetti paljon ja antoi uusia näkökulmia tulevaisuudenkin ohjelmointiprojekteihin.

Asiasanat:

CAN-väylä, henkilöautot, ohjelmistokehitys, vikadiagnostiikka

Bachelor’s Thesis | Abstract

Turku University of Applied Sciences

Automotive and Transportation Engineering

December 2023 | 32 pages

Ville Karikoski

Development of OBD-II Diagnostics Software for the Windows Operating System

The goal of this thesis was to develop an OBD-II diagnostics software that could be used to handle fault codes and sensor data. The software was developed for the Windows operating system and its user interface is based on the Windows Forms class library. Communication between the computer and the CAN bus was implemented using the ELM327 microchip. The motivation behind this subject was to gain practical experience in programming by working on a slightly larger project.

The theory part of this thesis gives a brief overview about the history of the diagnostic systems, general aspects of serial data communication, and buses used in data transmission. The CAN bus will be discussed in more detail because it is heavily involved in this project. The theory of digital technology was covered to the extent necessary for understanding the content of the work. The user interface and the main features of the software are discussed in the practical section of this thesis, along with explanations of the methods by which they were implemented.

While the software met the defined requirements, there was room for improvements and further development, which were discussed at the end of the thesis. The software was tested with an actual vehicle and a chart was generated in Excel from the data gathered during the tests. The project was recognized as a valuable learning experience, and it offered new view points that can be applied to future progarmming projects.

Keywords:

CAN bus, fault diagnostics, passenger cars, software development

Sisältö

Liitteet

Kuvat

Kuviot

1 Johdanto

Ajoneuvojen tekniikka on kehittynyt nopealla tahdilla erityisesti viime vuosikymmenten aikana. Suurimpia syitä nopealle kehitykselle ovat muun muassa kovaa tahtia kiristyvät päästötavoitteet ja turvallisuusvaatimukset. Näiden tavoitteiden täyttäminen on pakottanut lisäämään ajoneuvoihin lukuisia elektronisia järjestelmiä, mikä on tuonut mukanaan myös monia uudenlaisia haasteita.

Tämän työn tavoitteena on luoda Windows pohjainen ohjelmisto, jonka avulla voidaan lukea ajoneuvosta yleisimmät vikakoodit ja nollata ne. Lisäksi ohjelmistolla tulisi pystyä lukemaan reaaliaikaista dataa antureilta ja muuntaa se helpommin luettavaksi graafiseen muotoon. Työssä tullaan käsittelemään jonkin verran vikadiagnoosijärjestelmien historiaa, mutta pääpaino on siinä, millä tavalla digitaalinen tiedonsiirto ajoneuvon eri järjestelmien välillä toimii. Aihe on rajattu käsittelemään pääosin henkilöautoja ja sarjamuotoista tiedonsiirtoa.

Tämän opinnäytetyön aihe valikoitu oman mielenkiinnon mukaan ja halusta kehittää ohjelmointitaitoja hieman haastavamman projektin parissa. Ensisijaisesti tavoitteena ei ole julkaista valmistunutta ohjelmistoa, vaan se tehdään puhtaasti oppimismielessä. Lopuksi pohditaankin, miten projektissa onnistuttiin ja, mitä voitaisiin tehdä toisin. Työssä käydään läpi valmiin ohjelmiston ominaisuuksia ja miten ne on saatu toteutettua.

2 Vikadiagnoosijärjestelmät henkilöautoissa

Henkilöautoissa voi esiintyä lähes ääretön määrä erilaisia vikoja. Monet niistä voi olla hyvinkin helppo ja nopea diagnosoida, kuten esimerkiksi ilmeiset mekaaniset viat. Sähköjärjestelmissä esiintyvät viat ovat sen sijaan monesti monimutkaisempia, koska oireet saattavat olla ajoittaisia ja ne voivat myös ilmaantua eri tavoin eri aikoina.

Vikadiagnoosijärjestelmien tarkoituksena on tunnistaa mahdolliset vikatilat, kirjata ne muistiin ja ilmoittaa niistä ajoneuvon käyttäjälle. Tämä helpottaa usein vianetsintää huomattavasti ja voi kertoa ongelmista jo ennen kuin niitä muutoin edes huomaisi. Vikadiagnostiikkajärjestelmien antamat tiedot eivät kuitenkaan aina kerro suoraan vian syytä, vaan niitä pitää tulkita lähtökohtaisesti suuntaa antavasti. Näitä järjestelmiä kutsutaan usein myös englannin kieleen pohjautuvalla termillä OBD *eli On-Board Diagnostic*.

2.1 Standardointi

Vikadiagnoosijärjestelmien yksi päätarkoituksista on ollut valvoa erityisesti päästöjä vähentävien järjestelmien toimintaa. Näihin järjestelmiin liittyy useita eri säännöksiä ja standardeja, jotka määrittelevät muun muassa sen, mitä valvotaan ja millä tavalla. Standardointi mahdollistaa esimerkiksi, että yhdellä laitteella voidaan kätevästi lukea tietoja usean eri valmistajan järjestelmistä.

Tämänlaisista standardikokonaisuuksista tunnetuimpia ovat OBD-I ja OBD-II, joista ensimmäinen perustettiin vuonna 1988 Kaliforniassa. OBD-I-järjestelmä muutti pakolliseksi päästöjä valvovan järjestelmän, joka osaa tallentaa tunnistetun vian ja ilmoittaa siitä kuljettajalle vikavalon avulla. OBD-I oli kuitenkin vielä suhteellisen alkeellinen, eikä se määritellyt esimerkiksi sitä, millä tavalla järjestelmä pitää ajoneuvossa toteuttaa. Myöhemmin vuonna 1994 Kaliforniassa perustettiin myös OBD-II järjestelmä, joka tuli vuonna 1996 pakolliseksi kaikissa Yhdysvalloissa myytävissä uusissa ajoneuvoissa. (Robert Bosch Gmbh 2018, 914.) OBD-II oli vaatimuksiltaan huomattavasti tiukempi ja

määritteli muun muassa diagnostiikkapistokkeen rakenteen, joka on säilynyt samanlaisena nykypäivään asti. Lisäksi määriteltiin myös esimerkiksi käytettävät protokollat eli sen, miten tiedonsiirron tulee tapahtua sähköisten komponenttien välillä.

Euroopassa käytännössä OBD-II:sta vastaava järjestelmä EOBD eli *European OBD* tuli pakolliseksi vuonna 2000 kaikkiin bensiinikäyttöisiin henkilöautoihin ja vuoden 2003 jälkeen myös dieselkäyttöisiin henkilöautoihin (Robert Bosch Gmbh 2018, 917). Standardeja on päivitetty jatkuvasti sen mukaan, miten päästövaatimukset ovat kiristyneet.

2.2 Vikakoodien lukulaitteet

OBD-II ajoneuvoissa vikakoodien lukeminen tapahtuu kytkemällä diagnoosipistokkeeseen siihen tarkoitettu lukulaite. Tällaisia laitteita on tarjolla monenlaisia, joista osa on suunnattu ennemmin ammattilais- ja osa yleiskäyttöön. Ammattikäyttöön tarkoitetetulla laitteilla pääsee tarkastelemaan syvemmin ajoneuvon järjestelmiä ja käsittelemään esimerkiksi merkkikohtaisia vikakoodeja ja muita erityistoimintoja. Ne ovat kuitenkin yleensä huomattavasti kalliimpia kuin yleismalliset lukulaitteet. Lukijat voivat olla esimerkiksi pienikokoisia itsenäisiä laitteita, mutta lukijana voi toimia myös tietokone, johon on asennettu tarkoitukseen soveltuva ohjelmisto.

3 Väylätekniikka

Elektroniikkaa ollaan jo pitkään käytetty laajasti ajoneuvoissa ja sen merkitys on edelleen vain kasvusuuntainen. Moderneissa ajoneuvoissa on lukuisia eri järjestelmiä keskusyksiköineen, joita ovat esimerkiksi moottori, jarrut, turvatyynyt ja mahdolliset lisävarusteet, kuten kaistavahti ja vakionopeudensäädin. Jotta kyseiset järjestelmät tomisivat oikein ja luotettavasti, pitää niiden välisen ja sisäisen kommunikaation olla järjestetty asianmukaisesti. Tämä onnistuu käyttämällä tarkoitukseen suunniteltuja tietoliikenneväyliä.

3.1 Digitaalitekniikan perusteita

Tavu on yksikkö, joka koostuu vaihtelevasta määrästä bittejä riippuen käyttöympäristöstä. Yleisimpiä käytettyjä tavun kokoja ovat esimerkiksi 8 tai 16 bittiä. Käytettävä tavun koko vaikuttaa muun muassa siihen, että miten suuria lukuja yksittäisellä tavulla voidaan ilmaista. Tavu voidaan ajatella binäärilukuna, koska yksittäisen bitin arvo voi olla vain nolla tai yksi. Yleinen käytäntö on kääntää nämä binääriluvut heksadesimaalimuotoon, jolloin esimerkiksi 16-bittisellä tavulla voidaan ilmaista luvut väliltä 0–FFFF, joka on desimaalijärjestelmässä 0–65535. Käytettävissä olevien kokonaislukujen määrä saadaan korottamalla numero kaksi potenssiin sillä luvulla, mikä on luvun ilmaisemiseen käytettävien bittien lukumäärä eli tässä tapauksessa 16. Tavuun voidaan sisällyttää luvulle myös esimerkiksi etumerkki, mikä onnistuu helposti käyttämällä yksi tavun biteistä tähän tarkoitukseen. Jos siis 16-bitin tavusta yksi bitti käytetään etumerkkiin, jää jäljelle vain 15 bittiä itse luvulle, jolloin käytettäväksi lukuväliksi muodostuu -32768–32768 desimaalijärjestelmässä.

3.2 Tietoliikenneväylät

Systeemiä, jonka avulla tietotekniset yksiköt esimerkiksi ajoneuvoissa voivat liikuttaa dataa toistensa välillä, kutsutaan väyläksi. Tiedonsiirto väylässä voi tapahtua joko sarja- tai rinnakkaismuodossa (Juhala ym. 2005, 120). Se, että kumpaa tiedonsiirtotapaa käytetään riippuu siitä, että mitä vaatimuksia väylälle annetaan. Rinnakkaismuotoisessa tiedonsiirrossa dataa kuljetetaan useamman johtimen avulla, mikä mahdollistaa korkeammat tiedonsiirtonopeudet, mutta on monesti kalliimpi toteuttaa (Frenzel 2015, 5). Sarjamuotoinen tiedonsiirto voidaan toteuttaa vain kahdella johtimella ollen näin yleensä halvempi vaihtoehto, mutta toisaalta myös keskimäärin hitaampi (Frenzel 2015, 7).

Väylissä olevat päätepisteet, joita voivat olla esimerkiksi erilaiset anturit ja keskusyksiköt, kutsutaan nimellä solmu. Solmut voivat olla väylässä joko isäntiä (*Master*), joilla on täydet vapaudet lähettää viestejä tai orjia (*Slave*). Orja-tyyppiset solmut pystyvät vastaamaan vain isäntien lähettämiin pyyntöihin. Kommunikaatiotapa eli esimerkiksi viestien rakenne ja tiedonsiirtonopeudet, voivat vaihdella käytettävän protokollan mukaisesti. Väylien käytön yksi suurimmista eduista on se, että niiden ansiosta tarvittavan kaapeloinnin määrä vähenee merkittävästi verrattuna tilanteeseen, jossa väylätekniikkaa ei käytettäisi.

3.3 CAN-väylä

Ajoneuvoissa olevien tietoliikenneväylien pitää pystyä sietämään todella vaihtelevia ja haastavia olosuhteita. Johtimille on vain rajoitetusti tilaa, minkä vuoksi ne ovat lähellä toisiaan ja niiden keskeinen häiriöiden riski kasvaa. Ajoneuvossa olevan väylän tulisi myös pystyä jatkamaan toimintaansa, vaikka jokin verkoston yksittäinen solmu päätyisi virhetilaan. Nämä ja monet muut haasteet huomioon ottaen, kehitettiin erityisesti ajoneuvoihin sopiva CAN-väylä (*Controller Area Network*). CAN-väylän on kehittänyt Rober Bosch GmbH 1980-luvulla ja se on käytössä edelleen niin ajoneuvoissa kuin monissa muissakin sovellutuksissa, joissa tarvitaan häiriöitä sietäviä yhteyksiä (Zhu 2010, 11).

CAN-väylän fyysinen kerros voidaan toteuttaa käyttötarkoituksesta riippuen muutamalla eri tavalla. Kerroksella viitataan standardoidun *OSI-mallin* (*Open Systems Interconnection Reference Model*) mukaiseen järjestelmään, jossa tiedonsiirtoprotokolla jaetaan seitsemään eri kerrokseen (Edwards & Bramante 2009, 45). Toteutustavoista yleisin on nopea eli High-Speed CAN, jossa fyysinen kerros koostuu kahdesta toistensa ympärille kiedotusta kaapelista, joissa on toisistaan hieman poikkeavat jännitteet. Näiden johtimien välistä jännite-eroa tulkitsemalla saadaan selville signaalin arvo (nolla tai yksi) kullakin ajanhetkellä. Kummassakin kaapelissa lähetetään sama viesti yhtäaikaisesti eli kommunikaatio on sarjamuotoista, mutta signaalit ovat ikään kuin toistensa peilikuvia. Tämän tekniikan etuna on se, että kun signaaliin kytkeytyykin ulkopuolisia häiriöitä, se vaikuttaa kumpaankin johtimeen likimain yhtä paljon, jolloin niiden välinen jännite-ero pysyy myös samana ja häiriön vaikutus mitätöityy. (Juhala ym. 2005, 134.) Jokaisen solmun väylän liityntäkohtaan on yleensä lisätty häiriöiden minimoimiseksi myös päätevastukset (Juhala ym. 2005, 135).

Toinen yleisesti käytetty CAN-väylän muoto on hidas eli Low-Speed CAN, jota käytetään yleensä sellaisissa tilanteissa, joissa tarvitaan erityistä viansietokykyä (Zhu, Y. 2010, 14). Hidas CAN toimii hyvin samankaltaisella periaatteella kuin nopeakin, mutta eroja on muun muassa käytetyissä jännitteissä, tiedonsiirtonopeudessa ja kytkentätavassa. Joissain tapauksissa käytetään myös yhden johtimen CAN-väylää, jos häiriönsiedolla ei ole niin suurta merkitystä, mutta tällöin tiedonsiirtonopeudet ovat alhaisempia. Tätä ratkaisua käytetään pääasiassa vain valmistuskustannusten minimoimiseen (Robert Bosch Gmbh 2018, 1459). CAN-väylä ei ole ainoa ajoneuvoissa käytetty väylätekniikka, vaan usein käytetään lisäksi myös muita vaihtoehtoja erilaisissa käyttötarkoituksissa. Näitä ovat muun muassa LIN-väylä ja FlexRay.

3.4 Sarjamuotoinen tiedonsiirto

Sarjamuotoisessa tiedonsiirrossa data eli bitit välitetään nimensä mukaisesti jonossa. Tiedonsiirron onnistumisen kannalta on tärkeää, että lähettäjä ja

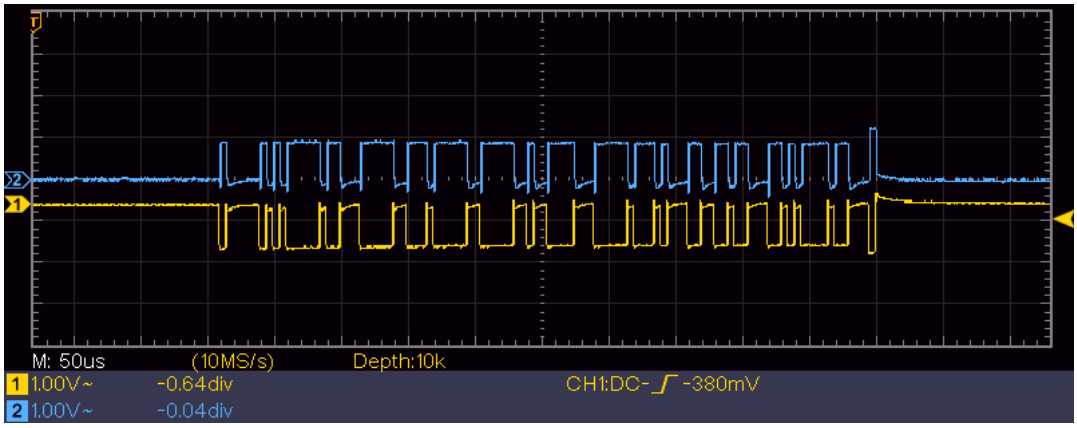
vastaanottaja toimivat samalla taajuudella eli tulkitsevat yksittäisen bitin keston samalla tavalla. Tämä voidaan toteuttaa esimerkiksi käyttämällä erillistä johdinta, jolla lähetetään tahdistussignaali vastaanottajalle. Tahdistaminen on mahdollista toteuttaa myös ilman erillistä kaapelia esimerkiksi lisäämällä viestiin ylimääräisiä bittejä tahdistusta varten, mutta tällöin tiedonsiirtonopeus on kuitenkin ennalta määritettävä. (Frenzel 2015, 19.) Nopeus ilmoitetaan yleensä muodossa bittiä per sekunti.

Vaikka lähettäjän ja vastaanottajan sisäiset kellot olisivatkin säädetty samalle taajuudelle, eivät ne suurella todennäköisyydellä ole täysin synkronoituneita keskenään. Ongelma voidaan ratkaista esimerkiksi lisäämällä viestiin niin sanottu viestikehys. Viestikehyksen rakenne vaihtelee, mutta se sisältää yleensä ainakin aloitus- ja lopetusmerkin, jonka välissä varsinainen viesti on. Sen lisäksi, että kehys mahdollistaa tahdistamisen, voidaan myös varioida muun muassa viestin pituutta lopetusmerkin ansiosta. Kehyksen sisään voidaan lisätä myös esimerkiksi virheentunnistuksen mahdollistavia osioita tai viestin lähettäjän tunnistetietoja. Yksi kehyksen käytön huonosta puolista on se, että viestiin varsinaisesti kuulumattomien bittien lähettämiseen kuluu aikaa, jolloin keskimääräinen tiedonsiirtonopeus laskee (Frenzel 2015, 18). Pitkien viestien lähettämiseen liittyy myös riski siitä, että synkronointi epätahdistuu, jonka vuoksi usein päädytään lähettämään viestit lyhyemmissä osissa.

Yksi yleisesti käytetty tapa tunnistaa tiedonsiirtovirheitä on sisällyttää viestiin niin sanottu pariteettibitti, joka lisätään useimmiten databittien perään. Pariteettibitti kertoo, että onko viestin data-osuudessa pariton vai parillinen määrä tarkkailtavaksi valitun arvoisia bittejä (Heath 2002, 81). Vastaanotin tarkistaa, että pitääkö pariteettibitin arvo paikkansa ja jos ei pidä, tiedetään, että viestissä on todennäköisimmin virhe. Tämä ei kuitenkaan ole erityisen tarkka virheentunnistusmetodi, koska voi olla mahdollista, että esimerkiksi hyvin kohinaisessa ympäristössä voi useammankin kuin yhden bitin arvo muuttua. Aiemmin mainituilla databiteillä tarkoitetaan niitä bittejä, jotka sisältävät viestin varsinaisen sanoman. Databittien lukumäärästä pitää olla lähettäjän ja vastaanottajan välillä yhteisymmärrys ja se ilmaistaan usein tavuina.

4 Viestikehykset CAN-väylässä

Viestikehys kuuluu aiemminkin mainitussa OSI-mallissa siirtoyhteyskerrokseen ja sen rakenne määritellään ISO 11898 -standardissa (Juhala ym. 2005, 130). CAN-väylässä käytetään neljää erilaista viestikehystä riippuen viestin tarkoituksesta, mutta tässä työssä tutkitaan tarkemmin vain laajentamatonta datakehystä. Datakehys on kehyksistä se, jota väylässä varsinaisen datan välittämiseksi käytetään. Erilaisia kehyksiä käytetään, kun halutaan ilmoittaa virheestä, pyytää tietoa tietyltä solmulta tai lisätä viivettä viestien välille. Oskilloskooppikuva datakehyksestä on nähtävissä kuvassa 1.



Kuva 1. CAN datakehys oskilloskoopilla tarkasteltuna. Ylempänä CAN H ja alempana CAN L signaali.

Kuvasta voidaan havaita CAN H ja CAN L välisten signaalien peilikuvamaisuus. Kuvasta nähdään myös, että signaali koostuu likimääräisestä kanttiaallosta. Kehyksen eri osiot eivät ole silmämääräisesti helposti havaittavissa, mutta erilaisia osioita on seitsemän ja niistä ensimmäinen on yksittäinen aloitusbitti, joka on dominantti eli arvoltaan nolla. Aloitus-osiota kutsutaan myös nimellä *Start of Frame.* (Zhu 2010, 16). Viestikehys päättyy lopetus-osioon (*End of Frame*), joka koostuu seitsemästä resessiivisestä bitistä (Zhu 2010, 20).

4.1 Viestien priorisointi

CAN-viestikehyksen alussa on osio, joka sisältää viestin lähettäjän tunnisteen. Tunniste-osio on pituudeltaan 11 bittiä (Juhala ym. 2005, 131). Tunnisteen tarkoituksena on mahdollistaa viestien suodattamisen ja niiden priorisoinnin. Priorisoinnin johdosta tunniste-osiota kutsutaan myös usein kiistelykentäksi (*arbitration field*) (Juhala ym. 2005, 131).

Priorisointi tulee tarpeen sellaisessa tilanteessa, jossa väylään on saapumassa useampia kuin yksi viesti yhtä aikaa. Jos jokin solmu lähettää väylään dominantin bitin, välittyy se jokaiselle väylän solmulle. Jos toinen solmu aloittaa sattumalta lähettämään viestiä samanaikaisesti, tarkastelee se samalla onko väylä samassa tilassa kuin bitti, mitä se yrittää lähettää. Jos solmu havaitsee eroavaisuuden, jää se odottamaan vuoroaan. Voittajaksi määräytyy alhaisimman tunnisteen omaava viesti. (Zhu 2010, 17.) Tunniste-osuuden perässä on niin sanottu RTR (*Remote Transmission Request*) -bitti. Tämän tarkoituksena on tarkentaa, että onko viestin tarkoitus lähettää vai pyytää dataa (Juhala ym. 2005, 131).

4.2 Data-osio

Data-osion pituutta voidaan vaihdella nollasta kahdeksaan tavuun, jotka kukin koostuvat kahdeksasta bitistä.Tavujen lukumäärä ilmaistaan ennen varsinaista dataa neljän bitin pituisella DLC koodilla (*Data Lenght Code*). Tyhjää data-osiota voidaan käyttää esimerkiksi erilaisten prosessien synkronoinnissa (Robert Bosch Gmbh 2018, 1461).

4.3 Virheiden tunnistaminen

Data-osion jälkeen kehyksessä tulee seuraavaksi virheentarkastukseen käytettävä CRC (*Cyclic Reduntacy Code*) -osio, joka on pituudeltaan 15 bittiä (Zhu 2010, 20). Lähettäjä luo CRC-koodin lähetettävien databittien perusteella matemaattisten kaavojen avulla. Vastaanottaja tekee vastaanottamilleen databiteille samat laskutoimenpiteet ja vertaa tuloksia. Jos tulokset poikkeavat, voidaan todeta virheen ilmaantuneen. Tämä virheentunnistusmetodi on huomattavasti tehokkaampi, kuin esimerkiksi stuff-bitit, joita kehyksessä myös käytetään. CRC-osion jälkeen tulee niin sanottu kuittauskenttä (*Ack Field*), jossa ilmoitetaan, että löytyikö viestistä virheitä vai ei (Robert Bosch Gmbh 2018, 1461). Jos virhe löytyi, lähetetään väylään erillinen virheilmoitus.

5 Menetelmät ja välineet

5.1 ELM327

Ohjelmiston toiminnan kannalta on välttämätöntä muodostaa yhteys ajoneuvon CAN-väylään jollakin tavalla. Tässä työssä ratkaisuksi valikoitui Bluetooth-adapteri, joka sisältää Elm Electronicsin valmistaman ELM327-mikropiirin. Kyseisen mikropiirin tehtävänä on kääntää CAN-väylästä tulevat viestit sellaiseen muotoon, että se voidaan vastaanottaa tietokoneen sarjaporteilla ja vastaisuudessa myös kääntää tietokoneen lähettämät viestit CAN-väylälle sopivaksi. Valinnan etuna on se, että ELM327:sta on tehty kattava dokumentaatio, eikä se ole erityisen hintava. ELM327 tarjoaa myös monia hyödyllisiä ominaisuuksia. Se pystyy muun muassa tunnistamaan kohdeajoneuvon protokollan automaattisesti ja sen toimintaa pystyy konfiguroimaan erillisillä AT-etuliitteisillä komennoilla.

5.2 Ohjelmiston testaaminen ja debuggaus

Debuggaus on yleisimmin ohjelmoinnissa käytetty termi, joka tarkoittaa vianetsintää. Ohjelmistoa kehittäessä kohtaa usein lähes väistämättä erilaisia virhetilanteita, joista osa voi johtua vain esimerkiksi yksittäisistä kirjoitusvirheistä. Toisinaan vian löytäminen voi olla myös hyvinkin haastavaa ja aikaa vievää. Tässä työssä ohjelmiston testaaminen ilman lisätarvikkeita olisi vähintäänkin epämukavaa, koska se vaatisi useimmiten läsnäoloa jossakin ajoneuvossa. Työtä tehdessä ajoneuvoa ei saatu käytännöllisesti riittävän lähelle, joten alettiin pohdiskella ratkaisua, joka mahdollistaisi testailun kätevästi kotioloissa.

Tätä varten on kehitetty muutamia valmiita virtuaalisia *ECU* (*Engine Control Unit*) -järjestelmiä, mutta ne olivat pääosin projektin budjettiin nähden liian kalliita. Vaihtoehtoisesti tässä työssä ECU:n korvikkeeksi valikoitui Arduino UNO -mikrokontrolleri. Arduinoon kiinnitettiin myös erillinen Seeed Studion

valmistama CAN-BUS Shield V2 -lisäosa, johon on sisäänrakennettu MCP-2515 ja MCP-2551 -mikropiirit, jotka mahdollistavat kommunikoinnin CAN-väylän kanssa. Lisäksi tähän kokonaisuuteen juotettiin myös OBD-II-naaras-liitin, jotta Bluetooth-adapteri saadaan siihen kätevästi kiinnitettyä. Arduinoon ajettiin Autodesk Instructables -nettisivuilta löytynyt ohjelmisto ”Arduino OBD2 Simulator”, jonka on julkaissut henkilö nimimerkillä mviljoen2 (mviljoen2. n.d.). Ohjelmistokehityksen loppuvaiheessa ohjelmiston toimivuutta testattiin myös oikean henkilöauton kanssa. Testauskohteena käytettiin vuosimallin 2000 Volvo S80 -henkilöautoa, jossa on 2,4-litrainen vapaastihengittävä bensiinimottori.

5.3 Ohjelmointikielen, ympäristön ja käyttöliittymän valinta

Tässä työssä ohjelmointikieleksi valikoitui C# pääosin siksi, että sen käytöstä oli eniten kokemusta. *IDE*:nä (*Integrated Development Environment*), eli ohjelmointiympäristönä käytettiin Microsoft Visual Studio 2022 -ohjelmistoa. Graafisen käyttöliittymän pohjana käytettiin Microsoftin Windows Forms -luokkakirjastoa, joka on ollut saatavilla jo 2000-luvun alkupuolelta asti. Windows Forms on vanha, mutta suhteellisen helppokäyttöinen ja soveltuu käyttötarkoitukseen riittävän hyvin. Windows Formsin valitseminen käyttöliittymäksi rajoittaa ohjelmiston toimivaksi vain Windows-pohjaisella alustalla. Ohjelmisto käyttää Microsoftin avoimen lähdekoodin .Net Framework 4.7.2 -ohjelmistokomponenttikirjastoa.

Formsilla luodut ohjelmistot toimivat pääsääntöisesti tapahtumapohjaisesti eli niiden toiminta ei seuraa jotakin määrättyä yksittäistä ohjelmistokoodia, vaan ohjelmisto jää odottamaan erilaisia tapahtumia, kuten napin painalluksia. Tapahtuman ilmaantuessa ohjelmisto suorittaa esimerkiksi napin painallukseen liittyvän funktion, joka sisältää erityisesti kyseiseen tilanteeseen tarkoitetun ohjelmistokoodin. Tämän jälkeen ohjelmisto palaa taas odottamaan seuraavaa tapahtumaa. Tapahtuman voi laukaista myös esimerkiksi käyttöliittymään upotettu ajastin tai vaikka sarjaporttiin saapunut viesti.

5.4 Signaalin tarkastelu oskilloskoopilla

CAN-väylässä liikkuvia viestejä tutkittiin Owon SDS1102 -merkkisellä oskilloskoopilla. Tämä ei ollut ohjelmiston kehittämisen kannalta välttämätöntä, mutta se auttoi ymmärtämään syvällisemmin tyypillisen CAN-viestin rakennetta ja todensi myös sen, että Arduinolta tulleet viestit olivat sellaisia kuin niiden pitäisikin olla. Oskilloskoopin kanavat kytkettiin CAN H ja CAN L -johtimiin ja jännitealueet säädettiin asentoon 1 volttia/div. Tiedettiin, että tiedonsiirtonopeus on 500 kb/s, jolloin yhden bitin pituus on 2 µs ja jonka perusteella horisontaalisäätö säädettiin 2 µs/div. CAN-väylässä viesti ilmenee vain hyvin lyhyen ajan, jonka vuoksi oskilloskooppi säädettiin liipaisutilaan. Tämä tarkoittaa käytännössä sitä, että oskilloskooppi kaappaa signaalista pysäytetyn näytteen, kun jännite muuttuu määrätyn verran.

Tarkastelu toteutettiin niin, että CAN-väylään lähetettiin pyyntö jäähdytysnesteen lämpötilasta ja vastaus otettiin heksadesimaalimuodossa ylös. Data-osion löytämiseksi oskilloskoopin tallentamaa signaalia alettiin tulkita bitti kerrallaan sen alkupäästä lähtien. Tiedettiin, että kyseessä on perinteinen ei-laajennettu CAN-dataviestikehys, jolloin bittien määrä ennen data-osiota pitäisi olla stuff-bittejä lukuunottamatta vakio. Ylös otettiin data-osiota edeltävä DLC-koodi ja itse data-osio paperille binäärimuodossa. Binääriluvut käännettiin heksadesimaalimuotoon ja niitä verrattiin ohjelmiston vastaanottamiin viesteihin. Kokeiluissa onnistuttiin ja oskilloskoopista tulkitut viestit täsmäsivät ohjelmistossa esiintyneen viestin kanssa. DLC-koodi piti myös paikkansa lähetettyjen datatavujen lukumäärän suhteen. Data-osiota edemmäksi ei viestikehyksessä edetty.

6 Ohjelmiston kehittäminen

6.1 Yhteyden muodostaminen ELM327-adapteriin ja yhteyskokeilut

Ensimmäisenä muodostettiin Bluetooth-yhteys ELM327:n ja tietokoneen välille. Kun yhteys oltiin saatu muodostettua, tarkistettiin, toimiiko niiden välinen kommunikaatio halutulla tavalla. Tarkastus tehtiin käyttäen PuTTY-terminaaliemulaattoria, joka mahdollistaa sarjaporttiyhteyden muodostamisen. Yhteys saatiin muodostettua ELM327:n valmistajan ohjeiden mukaisia parametreja käyttäen, joista tarkemmin myöhemmin. ELM327 tukee erityisiä AT-etuliitteisiä komentoja, joiden avulla sen toimintaa voidaan muokata halutulla tavalla. Kommunikaation testaamiseksi lähetettiin ELM327:an komento ATZ, jonka tarkoituksena on resetoida laitteen asetukset nollille. ELM327 lähettää tästä vastaukseksi mikropiirin nimen ja versionumeron. Myös muutamia muita viestejä lähetettiin ja vastaukset olivat oikeanlaisia, joten voitiin olettaa yhteyden toimivan.

6.2 Uuden projektin luominen ja sarjaporttiyhteyden konfigurointi

Ohjelmistoa varten luotiin tyhjä Windows Forms (.NET Framework) -projekti. Projektiin otettiin käyttöön .NET-luokkakirjastoon kuuluva SerialPort-luokka, jonka avulla sarjaporttiyhteys saadaan konfiguroitua ja muodostettua. SerialPort-luokasta luodaan ohjelmiston käynnistyksen yhteydessä instanssi ja sen konstruktoriin syötettiin tulevan sarjaportin alustava nimi, baudinopeus, pariteetti-, data- ja stoppibittien määrä. Parametrit ovat samat, kuin mitä käytettiin myös yhteyskokeilussa PuTTY-ohjelmistossa. Baudinopeudeksi valittiin 38400 baudia, ei pariteettibittiä, kahdeksan databittiä ja yksi stop-bitti. Sarjaportin nimimuuttujan arvoksi annettiin ”COM1”, mutta annetulla nimellä ei ole suurta merkitystä vielä tässä vaiheessa, koska se korvataan myöhemmin itse ohjelmistossa. On tärkeää ymmärtää, että tässä on kyse tietokoneen ja ELM327:n välisestä sarjaporttiyhteydestä, eikä ajoneuvon ja ELM327:n

välisestä. ELM327 osaa itsenäisesti tunnistaa yleisimmät OBD-protokollat ja hoitaa ajoneuvon kanssa kommunikoinnin yleensä ilman erillisiä toimenpiteitä.

6.3 Käyttöliittymän suunnittelu

Ohjelmiston käyttöliittymän ajateltiin aluksi koostuvan jonkinlaisesta päävalikosta, jossa olisi painikkeet jokaiselle osiolle, kuten vikakoodien lukemiselle ja joilla olisi omat näkymänsä. Tämän kanssa oli jonkin verran haasteita ja aikaa oli rajallisesti, joten päädyttiin mahduttamaan koko ohjelmisto yhdelle näkymälle. Ratkaisu ei välttämättä ollut kaikkein käyttäjäystävällisin, mutta tarvittavat asiat saatiin ikkunalle mahdutettua niin, että kokonaisuus oli kuitenkin suhteellisen järkevä. Käyttöliittymä johon päädyttiin on nähtävissä liitteessä 1.

Käyttöliittymä koostuu kolmesta osiosta, joista keskimmäisenä on tietoikkuna, joka näyttää vastaanotetun datan. Tietoikkunaan tulevat näkyviin myös ELM327:n omat ja ohjelmistoon liittyvät ilmoitukset. Ohjelmisto yrittää aina kääntää tulevan datan ensin luettavampaan muotoon ennen kuin se tulostetaan tietoikkunaan, mutta tulostaa kuitenkin viestin raakamuodossa, jos kääntäminen ei ole mahdolllista. Tietoikkunan alapuolella avulla on tekstin syöttämistä varten tarkoitettu laatikko, johon voidaan kirjoittaa komentoja manuaalisesti ELM327:lle. Ohjelmistoon on sisällytetty myös muutamia komentoja, joiden avulla voidaan esimerkiksi tyhjentää tietoikkuna tekstistä. Komennot alkavat aina ”/” merkillä ja tuettujen komentojen lista voidaan tulostaa näytölle komennolla ”/?”. Komennot kirjoitetaan samaan tekstilaatikkoon, kuin mihin ELM327:llekin tarkoitetut viesti kirjoitettaisiin.

Tietoikkunan vieressä vasemmalla on osio XY-kaaviolle, josta reaaliaikaista dataa voidaan tarkastella visuaalisemmassa muodossa. Kaavion otsikko päivittyy automaattisesti sen mukaan, että mikä on valittu datalähde. Kaavion X-akselille mahtuu data viimeiseltä kahdeltakymmeneltä näytteeltä. Y-akseli skaalautuu automaattisesti tulevan datan perusteella. Zoomausmahdollisuuksia ei ole sisällytetty, eikä historiaa voida itse ohjelmistossa kaaviosta selata, mutta

datan tarkastelu onnistuu myöhemmin esimerkiksi Excelissä lokitiedoston avulla.

Käyttöliittymän oikeaan reunaan on laitettu osio, jossa on muutamia painikkeita ja valintoja ohjelmiston toiminnan ohjaamiseksi. Näistä ylimmäisenä on COM-portin valitsemiseen tarkoitettu yhdistelmäruutu ja yhteyden muodostamispainike. Ohjelmisto hakee saatavilla olevat COM-portit käynnistyksen yhteydessä, mutta uudelleen hakua varten on lisätty oma painike. Alempana on valintanapit sille, halutaanko ajoneuvolta pyytää live-dataa vai halutaanko komentoja lähettää manuaalisesti. Jos live-tila on valittuna, voidaan valintanappien alapuolelta valita siihen liittyvät asetukset. Ensin valitaan se, mitä tietoa halutaan pyytää. Seuraavaksi voidaan valita, halutaanko kaavio näkyviin ja halutaanko tallentaa tiedot lokitiedostoon. Viimeisenä valitaan nopeus, jolla anturitietoja pyydetään. Nopeus on rajoitettu 200 millisekuntiin pyyntöjen välillä virhetilanteiden minimoimiseksi. Live-tilan käynnistämiseksi on lisätty erillinen start/stop-painike.

6.4 Lokitiedostot ja niiden hyödyntäminen Excel-ohjelmistossa

Live-datan seuraaminen esimerkiksi ajotilanteessa on useimmiten mahdotonta ja lailla kiellettyä. Tästä syystä ohjelmistoon päätettiin lisätä ominaisuus, jonka avulla datanäytteet voidaan tallentaa tekstitiedostoon aikaleimattuna. Lokiin on ensimmäiselle riville päätetty lisätä otsikot, jotta tulkinta olisi helpompaa. Aikaleima ja näyte eritellään tiedostossa erotinmerkillä, joksi valittiin sarkain. Erotinmerkin ansiosta esimerkiksi Excel-ohjelmistossa voidaan lokitiedoston sisältö muuntaa suoraan taulukoksi. Taulukon pohjalta voidaan tehdä esimerkiksi kaavio halutulta aikavälilta, joka helpottaa datan tulkintaa. Loki toimii työn kirjoitushetkellä vain live-tilan ollessa käytössä.

6.5 Datan vastaanottaminen

Kun ohjelmisto havaitsee, että sarjaportilta on tullut viesti, se laukaisee SerialPort-luokkaan kuuluvan DataReceived-tapahtuman. Tapahtuma käsitellään tähän tarkoitukseen luodussa tapahtumankäsitteijäfunktiossa (*Event handler*). Tässä funktiossa sarjaportin viesti siirretään erilliselle *string*- eli merkkijonomuuttujalle. DataReceived-tapahtuma ilmenee erillisessä säikeessä, mikä estää esimerkiksi käyttöliittymän elementtien päivittämisen suoraan kyseisestä funktiosta, joten seuraavaksi kutsutaan erillistä delegaattifunktiota, joka suorittaa loput tarvittavat toimenpiteet asianmukaisessa säikeessä (SerialPort.Datareceived Event. n.d). Se, mitä jatkotoimenpiteet ovat, riippuu siitä, että mitä vastaanotettu viesti sisältää ja mitkä asetukset ohjelmistossa on valittuna.

OBD viestien sisältö on standardoitu ISO-15031-5 ja sitä vastaavan SAE-J1979 standardin mukaisesti. Ensin tarkastetaan, onko viesti vastaus johonkin tiedettyyn pyyntöön, mikä onnistuu tarkastelemalla viestin ensimmäistä tavua. Jos luvun ensimmäinen numero on neljä, tarkoittaa tämä viestin olevan vastaus. Toinen luku, joka on heksadesimaaliluku väliltä 1-A kertoo, että mistä OBD-II-palvelusta (*service*) on kyse. Tässä työssä käsiteltävä ohjelmisto yrittää etsiä viestistä vain numeroa 1, joka viittaa sen hetkiseen anturitietoon ja numeroa ”3”, joka kertoo viestin liittyvän vikakoodeihin. Jos viestin alku olisi ”41”, eli vastaus anturitiedon pyyntöön, kuvaisi viestin seuraava merkkipari, että mistä tietolähteestä on kyse. Tämä merkkipari on myös heksadesimaaliluku ja se, että mihin tietolähteeseen kukin arvo viittaa, on määritelty myös SAE-J1979-standardissa. Tätä lukua kutsutaan Parameter ID:ksi, mikä on lyhennettynä PID. Ohjelmisto tukee kirjoitushetkellä vain kuutta eriä PID-arvoa. Jos kyseessä on tuettu PID-arvo, muuntaa ohjelmisto viestin luettavaan muotoon ja siirtää sen sitten omaan muuttujaansa. Riippumatta siitä, onko live-tila päällä vai ei, tulostetaan viesti muunnettuna ohjelmiston tekstiruutuun.

Kaavion ollessa käytössä tulostetaan tekstiruutuun myös aikaleima ja lisätään kaavioon piste. Asetuksesta riippuen tallennetaan kyseinen arvo myös lokiin.

Jos vastaanotettua viestiä ei tunnistettu, tulostetaan se tekstiruutuun alkuperäisessä muodossaan. Tämä mahdollistaa tarvittaessa muidenkin yksittäisten PID-arvojen lukemisen, mutta vaatii manuaalisen käännöksen. Tämän ansiosta myös ELM327:n omat viestit saadaan nähtäville.

6.6 Datan konversio luettavaan muotoon

Kun viestin alkuosa on tunnistettu ja PID löytyy tuettujen listalta, siirtyy viesti funktioon, jossa se käännetään luettavaksi. Viestin informaatio-osuuden pituus tavuina vaihtelee riippuen siitä, mikä PID on kyseessä. Jokainen tavu sisältää heksadesimaaliluvun ja useimmissa tapauksissa tieto ei ole suoraan tulkittavissa vain kääntämällä luvut desimaalimuotoon, vaan tieto on skaalattu SAE-J1979-standardin mukaisella tavalla. Esimerkiksi jäähdytysnesteen lämpötila ilmaistaan yhdellä tavulla. Yksi tavu, joka koostuu OBD-järjestelmässä kahdeksasta bitistä, mahdollistaa vain kaksinumeroisen positiivisen heksadesimaaliluvun ilmaisemisen, mikä rajoittaa luvun mahdolliset arvot välille 0-FF, joka on desimaalijärjestelmään käännettynä 0-255. Lämpötila annetaan celsiusasteina ja voi olla myös negatiivinen esimerkiksi tilanteessa, jossa auto on seissyt vaikka yön yli pakkasessa. Tästä syystä jäähdytysnesteen arvoon on lisätty neljänkymmenen asteen offset. Tällä tarkoitetaan sitä, että ilmaistu arvo on neljäkymmentä astetta todellisuutta korkeampi ja siitä pitää vähentää kyseinen luku. Tämä toimenpide laskee tietysti myös käytettävissä olevien lukujen ylärajaa yhtä paljon, jolloin Celsiusasteet voidaan ilmaista desimaalijärjestelmässä arvosta -40 arvoon 215.

Toiseksi esimerkiksi otettakoon ilmamassamittarin massavirran arvo. Tämä tieto ilmaistaan kahdella tavulla. Nämä kaksi tavua yhdistetään ensin yhdeksi 16-bittiseksi tavuksi. Kuudestatoista bitistä koostuvan heksadesimaaliluvun avulla voidaan ilmaista arvot väliltä 0-FFFF (0-65535 desimaalijärjestelmässä), joka on sellaisenaan jopa tarpeettoman suuri massavirralle. Jotta koko skaala saataisin käytettyä hyväksi, on massavirta ilmoitettu viestissä muodossa 0,01 g/s. Yksikkömuunnoksen jälkeen mahdolliset arvot ovat siis välillä 0-655,35 g/s.

6.7 Vikakoodien luku ja nollaaminen

Vikakoodeja voidaan pyytää OBD-järjestelmästä lähettämällä CAN-väylään viesti ”03” tai ”07”. Ero näiden välillä on se, että ”07” pyytää koodeja, jotka ovat tallentuneet väliaikaisesti viime ajosyklin aikana ja ”03” sellaisia, jotka ovat tallentuneet järjestelmään pysyvästi. Vastaukseksi tulee numerolla ”43” tai ”47” alkava viesti riippuen valitusta moodista ja, jonka ohjelma sitten tunnistaa ja ohjaa oikeaan funktioon.

Vastausviestin tieto-osuus on kuusi tavua pitkä ja mahdolliset vikakoodit luetaan kahden tavun pareissa. Tämä rajoittaa yhteen vastaukseen mahtuvaksi vain kolme yksittäistä vikakoodia joten, jos koodeja on useampia, jaetaan vastaus useampaan viestiin. Vastauksen tieto-osuus on aina kuusi tavua, vaikka vikakoodeja ei olisikaan ja tällaisessa tapauksessa tyhjät tavuparit ilmaistaan muodossa ”00 00”. Tarkastellaan tilannetta, jossa yksi vastaanotetun viestin vikakoodeista on esimerkiksi ”03 35”. Ensimmäinen numero eli tässä tapauksessa ”0” kertoo, minkä kategorian vikakoodi on kyseessä ja onko koodi merkkikohtainen vai standardin mukainen. Tämä tieto tulkitaan SAE J2012 - standardin mukaisesti yksittäisten bittien perusteella, mutta tässä työssä käytettiin valmiiksi luotua taulukkoa, jossa mahdolliset vaihtoehdot oli yksinkertaisemmin lueteltuna. Taulukko löytyy esimerkiksi ELM327-mikropiirin omasta teknisestä tiedotteesta (ELM327 OBD to RS232 Interpreter n.d., 31).

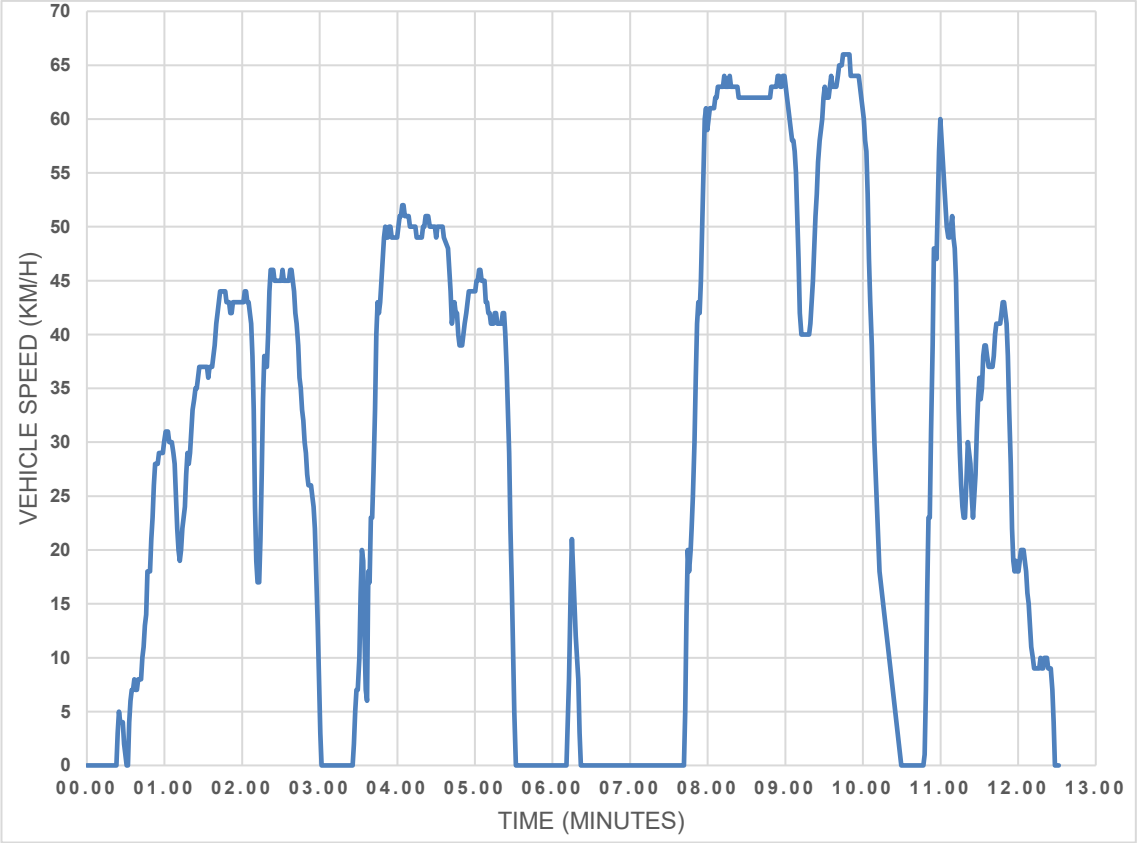
Kategorialla tarkoitetaan, että liittyykö koodi esimerkiksi voimansiirtoon tai alustaan. Tässä kuvitellussa tlanteessa numero ”0” käännetään taulukon mukaisesti muotoon ”P0”, jossa ”P” viittaa koodin kohdistuvan voimansiirtoon ja ”0” kertoo koodin olevan geneerinen eli se on standardin mukainen. Jos arvo olisi ”1”, koodi olisi autonvalmistajan itse määrittelemä. Seuraavaksi tarkastellaan koodin loppuosaa, mikä tässä tapauksessa on ”3 35”. Näistä numeroista ensimmäinen tarkentaa, että mistä järjestelmästä on kyse ja tässä tapauksessa numero ”3” viittaa sytytysjärjestelmään. Loput kaksi numeroa eli ”35” kertovat, että mikä yksittäinen vika on kyseessä. Tässä tapauksessa koodi tarkoittaa vikaa kampiakselin asentotunnistimessa. Koodin lopullinen muoto on

siis P0335 ja ohjelmistossa se tallennettaisiin tällaisena omaan väliaikaiseen muuttujaansa. Ohjelmistoon on liitetty tekstitiedosto, joka sisältää geneeriset vikakoodit selitteineen. Vastaanotetut koodit yritetään tunnistaa tekstitiedoston avulla ja jos selite löytyi, tulostetaan koodit käyttäjälle nähtäväksi selitteineen. Jos selitettä ei löytynyt, tulostetaan vain itse koodit ilman selitettä myöhempää selvitystä varten.

6.8 Ohjelmiston testaaminen ajoneuvon kanssa

Kun ohjelmiston kehityksessä oltiin päästy riittävän pitkälle, päätettiin sitä testata myös oikean henkilöauton kanssa. Auton kanssa lähdettiin noin kuuden kilometrin pituiselle ajolle. Ohjelmistoon laitettiin päälle live-tila ja matkan ajalta päätettiin nauhoittaa ajonopeus lokitiedostoon ja myöhemmin tehdä datasta Excel-kaavio. Ohjelmisto asetettiin lähettämään pyyntöjä ajonopeudesta sekunnin välein.

Ajon jälkeen todettiin, että ohjelmisto toimii niin kuin pitääkin ja lokitiedosto löytyi sille tarkoitetusta kansiosta. Lokitiedoston sisällöstä luotu kaavio on nähtävissä kuviossa 1.



Kuvio 1. Lokitiedostoston avulla luotu kaavio oikeasta ajotilanteesta.

Lokitiedoston sisältö ja kaavio näyttivät asianmukaisilta, eikä silmämääräisesti ollut löydettävissä mitään poikkeamia. Aikaleimojen perusteella tietoja oli vastaanotettu suurin piirtein sekunnin välein niin kuin pitikin, eikä katkoksia löytynyt. Tietojen siirtäminen Exceliin ja kaavion tekeminen onnistui nopeasti. Aikaleimassa aika ilmaistaan kellonajan perusteella, mikä ei kaavion tekemisen kannalta ollut välttämättä paras vaihtoehto. Tämän kaavion kannalta olisi ollut kätevämpää, jos aika olisi lähtenyt nollasta live-tilan käynnistyttyä. Vikakoodien lukua ja niiden nollaamista ei päästy kunnolla kokeilemaan, koska käytetyssä autossa sellaisia ei ollut olemassa.

7 Pohdinta ja kehityskohteet

Työn aikana kehitetty ohjelmisto saatiin täyttämään tavoitteiksi annetut kriteetit ja sitä voidaan käyttää apuna vianetsinnässä OBD-II ajoneuvoissa. Itse ohjelmointiin oli projektin aikana vain rajallisesti aikaa, minkä vuoksi ohjelmistoa kehittäessä jouduttiin tekemään muutamia kompromisseja ja ohjelmistoon jäi myös paljon puutteita ja kehityskohteita.

Ohjelmiston suurimpia puutteita on se, että erilaisia anturitietolähteitä tuetaan vain kuusi kappaletta. Alhainen määrä johtuu enimmäkseen ajanpuutteesta, mutta osittain myös ohjelmiston toteutustavasta, koska samoja tietoja pitäisi syöttää useampaan eri paikkaan. Vikakoodeista pystytään tunnistamaan 999 erilaista geneeristä koodia, mutta merkkikohtaisia ei saatu sisällytettyä. Merkkikohtaisten vikakoodien selitteiden löytäminen on huomattavasti hankalampaa, eikä niiden lisäämisen katsottu olevan tämän projektin kannalta oleellista.

Ohjelmiston kehityskohteena on myös se, että ikkunan kokoa ei voida millään tavalla muuttaa, mikä voisi olla joissain tapauksissa toivottu ominaisuus. Ohjelmistosta voisi myös saada kätevämmän,jos sen kehittäisi Android-pohjalle, koska matkapuhelinta olisi huomattavasti näppärämpi kuljettaa mukana kuin esimerkiksi läppäriä. Live-tilaan liittyvää kaavio-ominaisuutta voisi parannella lisäämällä siihen historian ja mahdolisuuden tehdä siihen erilaisia säätöjä. Tämä voisi mahdollistaa datan tarkemman tarkastelun suoraan ohjelmistosta, eikä tarvitsisi aina käyttää lokitiedostoa.

8 Yhteenveto

Työn aiheena oli OBD-II-vikadiagnostiikkaohjelmiston kehittäminen Windows-käyttöjärjestelmälle. Päädyin aiheesen, koska halusin kehittää ohjelmointitaitojani ja ajattelin, että prosessista voisi saada aikaiseksi myös hyvän opinnäytetyön. Ohjelmoinnille oli aikaa vain rajatusti, mutta se riitti kuitenkin ohjelmiston minimitavoitteiden täyttämiseen.

Teoriaosuuden alussa todettiin, että vikadiagnostiikkajärjestelmät ovat oleellinen osa päästöihin liittyvää valvontaa ja niihin liittyvät vaatimukset on tarkoin määritelty standardein. Standardit päivittyvät jatkuvasti päästövaatimusten kiristyessä. Todettiin myös, että erilaisia vikakoodien lukulaitteita on monenlaisia niin ammatti- kuin yleiskäyttöönkin.

Digitaalitekniikan osalta kerrottiin, että tavu koostuu vaihtelevasta määrästä bittejä. Tavujen sisältö käännetään usein heksadesimaaliluvuksi ja käytettävissä olevien bittien määrä vaikuttaa lukuväliin, joka sen avulla voidaan ilmaista. Kerrottiin, että tiedonsiirto tietoteknisten yksiköiden välillä tiedonsiirto tapahtuu väylien avulla, joissa tiedonsiirto voi olla sarja- tai rinnakkaismuotoista. Käytettävä tiedonsiirron muoto riippuu käyttötarkoituksesta. Sarjamuotoisesta tiedonsiirrosta todettiin, että siinä tieto siirtyy jonomuodossa useimmiten yhtä johdinta pitkin. Sarjamuotoisessa tiedonsiirrossa suurimpia ongelmia ovat muun muassa lähettäjän ja vastaanottajan välinen tahdistus ja virheiden tunnistaminen.

Henkilöautoissa yleisesti käytettävän CAN-väylän kerrottiin olevan hyvä sietämään häiriöitä. Tehokas häiriönsietokyky perustuu muun muassa väylän fyysiseen toteutukseen, jossa sama signaali välitetään kahdessa johtimessa ja tulkitaan niiden välistä jännite-eroa. CAN-väylästä on muutamia erilaisia variaatioita erilaisia käyttökohteita varten. CAN-väylän tiedonsiirrossa käytetään viestikehystä, joka mahdollistaa esimerkiksi tahdistamisen, viestien priorisoinnin ja tehokkaan häiriöntunnistuksen.

Ohjelmistokehityksestä kerrottiin, mitä apuvälineitä käytettiin ja mikä on ohjelmiston pohja. Käytännön osuus alkoi yhteyskokeiluilla ja uuden projektin luomisella. Käytiin läpi ohjelmiston käyttöliittymää, pääominaisuuksia, ja miten ne toteutettiin. Kerrottiin ohjelmiston kokeilusta oikean ajoneuvon kanssa ja esitettiin tilanteesta syntynyt kaavio. Työn lopussa mietitiin, miten onnistuttiin ja minkälaisia puutteita ohjelmistoon jäi.

Opinnäytetyöstä tuli mielestäni onnistunut kokonaisuus, vaikka alussa olikin hieman epävarmuutta vähäisen ohjelmointikokemuksen vuoksi. Ohjelmointivaihe korosti, että se vaatii paljon kärsivällisyyttä, koska muutamia yksittäisiä ongelmia sai ratkoa jopa useita tunteja. Verkosta löytyy paljon tietoa ja apuja ohjelmointiin, mutta niitä pitää silti useimmiten itse soveltaa omaan käyttötarkoitukseen sopivaksi. Palkitsevimpia olivat ne tilanteet, kun itse keksii ratkaisun johonkin ongelmaan. Työn tekeminen opetti paljon ja antoi uusia näkökulmia ja työkaluja mahdollisia tulevaisuuden ohjelmointiprojekteja ajatellen. Opinnäytetyön tekeminen ja siihen liittyvään teoriaan tutustuminen oli myös miellyttävää, koska aihe oli itselle mielenkiintoinen.

Lähteet

Edwards, J. & Bramante, R. 2009. Network Self-Teaching Guide. E-kirja ProQuest Ebook Central kirjapalvelussa. 1., painos. John Wiley & Sons, Incorporated. Vaatii kirjautumisen palveluun. Viitattu 31.10.2023. https://ebookcentral.proquest.com/lib/turkuamk-ebooks/detail.action?docID=433738

ELM327 OBD to RS232 Interpreter. n.d. Tekninen tiedote. elmelectronics.com sivusto. Viitattu 24.10.2023. https://www.elmelectronics.com/DSheets/ELM327DSH.pdf

Frenzel, L. 2015. Handbook of Serial Communications. E-kirja ProQuest Ebook Central kirjapalvelussa. 1., painos. Elsevier Science & Technology. Vaatii kirjautumisen palveluun. Viitattu 2.10.2023. https://ebookcentral.proquest.com/lib/turkuamk-ebooks/detail.action?docID=2189944

Heath, S. 2002. Embedded Systems Design. E-kirja ProQuest Ebook Central kirjapalvelussa. 2., painos. Elsevier Science & Technology. Vaatii kirjautumisen palveluun. Viitattu 15.11.2023. https://ebookcentral.proquest.com/lib/turkuamk-ebooks/detail.action?docID=294113

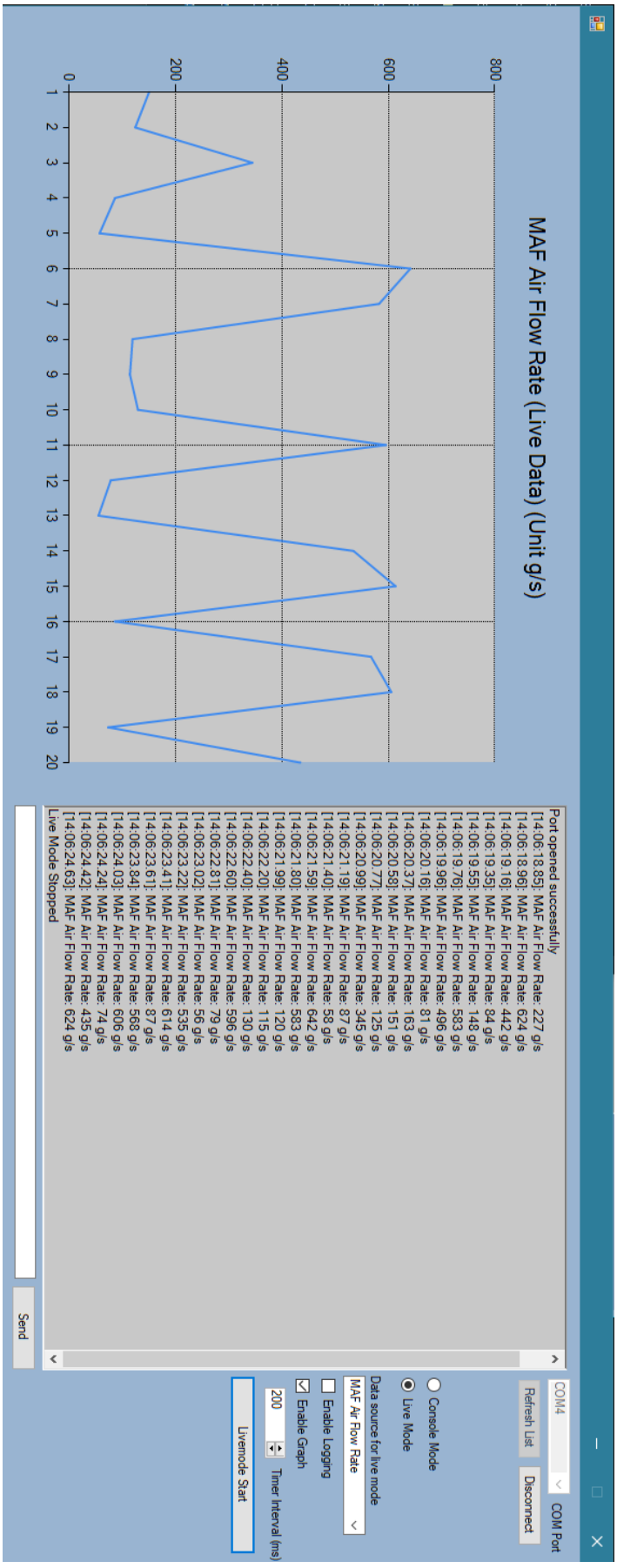
Juhala, M.; Lehtinen, A.; Suominen, M. & Tammi, K. 2005. Moottorialan sähköoppi,8. uudistettu painos. Jyväskylä: Gummerus Kirjapaino Oy

mviljoen2. n.d. Arduino OBD2 Simulator. Artikkeli. Autodesk Instructables sivusto. Viitattu 15.11.2023. https://www.instructables.com/Arduino-OBD2-Simulator/

Robert Bosch GmbH 2018. Automotive Handbook - 10th Edition. Vehicle Physics. Wiley, Chichester

SerialPort.Datareceived Event. n.d. Tiedote. learn.microsoft.com sivusto. Viitattu 1.11.2023. https://learn.microsoft.com/en-us/dotnet/api/system.io.ports.serialport.datareceived?view=netframework-4.7.2

Zhu, Y. 2010. CAN and FPGA Communication Engineering. E-kirja ProQuest Ebook Central kirjapalvelussa. 1., painos. Diplomica Verlag. Vaatii kirjautumisen palveluun. Viitattu 3.10.2023. https://ebookcentral.proquest.com/lib/turkuamk-ebooks/detail.action?docID=660231


MAF Air Flow Rate (Live Data) (Unit g/s)
800
600
400
200
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Port opened successfully
[14:06:18.85] MAF Air Flow Rate: 227 g/s
[14:06:18.96] MAF Air Flow Rate: 624 g/s
[14:06:19.16] MAF Air Flow Rate: 442 g/s
[14:06:19.35] MAF Air Flow Rate: 84 g/s
[14:06:19.55] MAF Air Flow Rate: 148 g/s
[14:06:19.76] MAF Air Flow Rate: 583 g/s
[14:06:19.96] MAF Air Flow Rate: 496 g/s
[14:06:20.16] MAF Air Flow Rate: 81 g/s
[14:06:20.37] MAF Air Flow Rate: 163 g/s
[14:06:20.58] MAF Air Flow Rate: 151 g/s
[14:06:20.77] MAF Air Flow Rate: 125 g/s
[14:06:20.99] MAF Air Flow Rate: 345 g/s
[14:06:21.19] MAF Air Flow Rate: 87 g/s
[14:06:21.40] MAF Air Flow Rate: 58 g/s
[14:06:21.59] MAF Air Flow Rate: 642 g/s
[14:06:21.80] MAF Air Flow Rate: 583 g/s
[14:06:21.99] MAF Air Flow Rate: 120 g/s
[14:06:22.20] MAF Air Flow Rate: 115 g/s
[14:06:22.40] MAF Air Flow Rate: 130 g/s
[14:06:22.60] MAF Air Flow Rate: 596 g/s
[14:06:22.81] MAF Air Flow Rate: 79 g/s
[14:06:23.02] MAF Air Flow Rate: 56 g/s
[14:06:23.22] MAF Air Flow Rate: 535 g/s
[14:06:23.41] MAF Air Flow Rate: 614 g/s
[14:06:23.61] MAF Air Flow Rate: 87 g/s
[14:06:23.84] MAF Air Flow Rate: 568 g/s
[14:06:24.03] MAF Air Flow Rate: 606 g/s
[14:06:24.24] MAF Air Flow Rate: 74 g/s
[14:06:24.42] MAF Air Flow Rate: 435 g/s
[14:06:24.63] MAF Air Flow Rate: 624 g/s
Live Mode Stopped
Send
COM4
COM Port
Refresh List
Disconnect
Console Mode
Live Mode
Data source for live mode
MAF Air Flow Rate
Enable Logging
Enable Graph
200
Timer Interval (ms)
Livemode Start