Jarmo Seisko

# OBD-etälukulaite

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Ajoneuvo- ja kuljetustekniikka

Insinöörityö

15.12.2017

Metropolia

| | |
|---|---|
| Tekijä(t)<br>Otsikko<br><br>Sivumäärä<br>Aika | Jarmo Seisko<br>OBD-etälukulaite<br><br>25 sivua + 1 liite<br>15.12.2017 |
| Tutkinto | Insinööri (AMK) |
| Koulutusohjelma | Ajoneuvo- ja kuljetustekniikka |
| Suuntautumisvaihtoehto | Ajoneuvosähkötekniikka |
| Ohjaaja(t) | Lehtori Vesa Linja-aho, Metropolia Ammattikorkeakoulu |

Tämä insinöörityö tehtiin osana startup-yritys Servoped Oy:n tuotekehitystä. Työn tavoitteena oli suunnitella ja toteuttaa laite, jolla voidaan kommunikoida henkilöajoneuvon kanssa käyttäen ajoneuvon OBD-väylää. Laitteeseen haluttiin myös etäohjauksen mahdollisuus.

Työssä käydään läpi suunnittelun eri vaiheet ja valmiille laitteelle asetetut vaatimukset. Haasteena työssä oli suunnitella laite siten, että sillä kerättävää dataa voitaisiin käyttää hyvällä hyötysuhteella.

Varsinainen ohjelmiston suunnittelu suoritettiin Arduino-ohjelmointiympäristössä, jota työssä käytetty Hologram Dash -mikrokontrolleri tukee. Ajoneuvon kiihtyvyystietoja tallennettiin erillisellä kiihtyvyys anturilla, joka on kytketty Hologram Dash -mikrokontrolleriin.

Lopputuloksena syntyi toimiva OBD-etälukulaite, joka on muokattavissa ja laitteen konfiguraatiota voidaan muuttaa etäohjauksella. Laitteen keräämä data lähetään ensin Amazon-palvelimelle, josta data siirretään Heroku-palvelinympäristöön, jossa data tarjotaan asiakkaille räätälöidyssä muodossa.

| | |
|---|---|
| Avainsanat | OBD, CAN-väylä, etäohjaus, IoT, autot, pilvipalvelut |

| | |
|---|---|
| Author(s)<br>Title<br><br>Number of Pages<br>Date | Jarmo Seisko<br>Remote controlled OBD-device<br><br>25 pages + 1 appendix<br>15 Dec 2017 |
| Degree | Bachelor of Engineering |
| Degree Programme | Automotive engineering |
| Specialisation option | Automotive electrical engineering |
| Instructor(s) | Vesa Linja-aho, Senior Lecturer, Metropolia UAS |

This engineering thesis was carried out as part of product development at the startup company Servoped Oy. The aim of this thesis was to design and implement a device that can communicate with a passenger vehicle using the OBD-connection of the vehicle. Remote connection to the device was implemented.

This thesis examines the various stages of designing a new product and the requirements for the finished product. The challenge was to determine such specifications for the device to be able to provide useful data.

The actual software design was carried out in the Arduino programming environment which is supported by the Hologram Dash -microcontroller. Vehicle acceleration data was recorded with a separate acceleration sensor connected to the Hologram Dash -microcontroller.

The result was a fully functional OBD-device that is customizable and remote controlled. The data collected by the OBD-device is first sent to Amazon-server and then routed to Heroku-server which is where the data is offered to customers in appropriate format.

| | |
|---|---|
| Keywords | OBD, CAN bus, remote control, IoT, cars, cloud services |

## Sisällys

## Lyhenteet

| | |
|---|---|
| OBD-II | Onboard Diagnostics II |
| CAN | Controller Area Network |
| IoT | Internet of Things |
| PID | Parameter identification number |
| EOBD | European Onboard Diagnostics |
| I/O | Input/Output |
| ISO | International Organization for Standardization |

# 1 Johdanto

Ajoneuvotekniikka on kehittynyt paljon viimeisen vuosikymmenen aikana, ja uusissa autoissa on ohjainlaitteita enemmän kuin koskaan. Nykypäivän ajoneuvossa ohjainlaitteiden määrä voi ylittää jo sadan kappaleen rajan. Ohjainlaitteet liittyvät toisiinsa lähes kaikissa uusissa ajoneuvoissa CAN-väylällä, joka on standardoitu ISO-standardilla.

Ajoneuvojen digitalisoitumisen lisäksi myös uudet palvelut, kuten yhteiskäyttöajoneuvot (CarSharing) ja Uber tekevät tuloaan. Servoped Oy haluaa tuoda asiakkailleen mahdollisuuden saada etänä tiedon siitä, mitä ajoneuvossa tapahtuu juuri nyt. Koska saatavilla olevien ohjainlaitteiden määrä on kasvanut niin suureksi, tämä tieto voi tarkoittaa eri asioita monille eri asiakkaille. Esimerkkejä käyttötavoista ovat ajoneuvon lukituksen etäohjaus, sijaintitiedot ja vikakooditiedot. Laitteen tulee myös toimia lähtökohtaisesti kaikissa Euroopassa myytävissä ajoneuvoissa.

Työn tarkoituksena oli suunnitella ja tuottaa OBD-etälukulaite, jota Servoped Oy tulee käyttämään tuotekehityksessään apuna tuottamaan sisältöä ajoneuvolta saatavasta datasta. Servoped Oy:n vaatimuksena oli suunnitella ja tuottaa laite, joka on mahdollisimman pieni, muokattavissa sekä kannattavasti toteutettavissa. Laitteelta saatava data analysoidaan palvelinympäristössä, jossa se tarjotaan myös asiakkaalle sopivassa muodossa. Laite on siis myös IoT-laite eli esineiden internet -laite.

# 2 CAN-väylä ja OBD-diagnostiikka

CAN-väylä on yleisin ajoneuvokäytössä oleva väyläprotokolla, jota voidaan käyttää tarjoamaan OBD-diagnostiikka palveluja. OBD-diagnostiikka oli ensimmäinen standardoitu ajoneuvokäyttöön tarkoitettu käyttöliittymä, joka otettiin käyttöön Kalifornian osavaltiossa Yhdysvalloissa vuonna 1991. Melko pian tämän jälkeen kehitettiin uusi standardi OBD-II, joka tuli Yhdysvalloissa voimaan vuonna 1996. (OBD-II Background.)

## 2.1 OBD-diagnostiikka

OBD eli ”onboard diagnostics” on standardoitu liitäntä sekä tietorajapinta henkilö ajoneuvoissa. OBD:n kautta voidaan keskustella ajoneuvon moottorinohjainlaitteen kanssa ja saada esimerkiksi katsastuksessa käytettäviä tietoja. Standardoinnin takia tiedot ovat saatavissa kaikista ajoneuvoista. OBD-diagnostiikka syntyi, kun tarve seurata ajoneuvojen käytönaikaisia päästöjä lisääntyi. Vikatilanteissa polttomoottoreiden haitalliset päästöt saattavat moninkertaistua ja tästä syystä ajoneuvoihin kehitettiin OBD-diagnostiikka, jotta ajoneuvot pystyvät valvomaan itseään. (Mikä on OBD?.)

### 2.1.1 OBD-II- ja EOBD-standardit

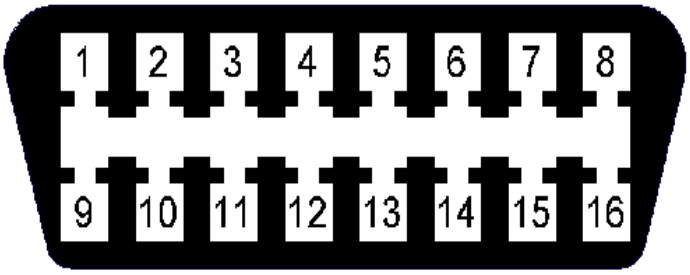
OBD-II-standardin mukaan moottorin ohjausyksikkön täytyy tarjota määrätyt vianmääritystiedot niitä pyydettäessä. SAE J1979 -standardissa on määritelty, kuinka menetellään erilaisten vianmääritystietojen pyytämiseksi ja parametrit, jotka tulee olla saatavilla moottorinohjainlaitteelta. Parametreja kutsutaan parametri-id:llä (PID), jotka on myös määritelty SAE J1979:ssä. Standardissa on määritelty vain vähimmäisvaatimukset, joten halutessaan auton valmistajat voivat lisätä omia parametrejään. OBD-II-standardin tarkoitus on tarjota ajoneuvosta reaaliaikaisia tietoja erityisesti ympäristöpäästöihin liittyvistä säätösuureista sekä näihin liittyviä vikakoodeja. (Mikä on OBD?.)

EOBD (European On Board Diagnostic) on eurooppalainen versio OBD-II:sta ja sisältää samat asiat, mutta EOBD tuli voimaan vasta vuonna 2001 bensiiniajoneuvoille ja 2003 dieselajoneuvoille. EU direktiivin 98/69/EC mukaan 1.1.2000 jälkeen tyyppihyväksyttyjen ajoneuvojen tulee olla EOBD-yhteensopivia. Kaikkien vuodesta 2001 lähtien myytyjen uusien ajoneuvojen täytyy olla EOBD-yhteensopivia. (Mikä on OBD?.)

### 2.1.2 OBD-II-liitin ajoneuvossa

SAEJ1962-standardi määrittelee OBD-II-diagnoosiliitännän, josta on kaksi standardoitua laitteistoliitäntää, tyyppi A ja tyyppi B. Molemmat näistä ovat tyypiltään naaraspuolisia, 16-nastaisia liittimiä. A-tyypin liitin on suunniteltu ajoneuvoihin, joiden käyttöjännite on 12 volttia, ja B-tyypin liitin on ajoneuvoille, joiden käyttöjännite on 24 volttia. Liitimet eroa-

vat toisistaan siinä, että tyypin A liitintä ei voi kytkeä B-tyypin liittimen pistokkeesen erilaisen keskiuran vuoksi. B-tyypin liitin on myös merkitty sinisellä värillä korkeammasta käyttöjännitteestä johtuen. Liittimestä tarvitaan vain 2 pinniä CAN-väylän kytkemiseen, mutta yleisimmät OBD-II-testerit ottavat myös käyttöjännitteensä OBD-II-liittimestä (kuva 1). (OBD-II Connector.)




| | |
|---|---|
| 1 | *Valmistajan määritettävissä* |
| 2 | **J-1850 -väylien signaalin +** |
| 3 | *Valmistajan määritettävissä* |
| 4 | **Ajoneuvon maataso** |
| 5 | **Signaalien maataso** |
| 6 | **CAN High (J-2284 & ISO 15765-4)** |
| 7 | **K-johdin (ISO 9141-2 & ISO 14230-4)** |
| 8 | *Valmistajan määritettävissä* |
| 9 | *Valmistajan määritettävissä* |
| 10 | **J-1850 -väylien signaalin -** |
| 11 | *Valmistajan määritettävissä* |
| 12 | *Valmistajan määritettävissä* |
| 13 | *Valmistajan määritettävissä* |
| 14 | **CAN Low (J-2284 & ISO 15765-4)** |
| 15 | **L-johdin (ISO 9141-2 & ISO 14230-4)** |
| 16 | **Ajoneuvon käyttöjännite +** |

Kuva 1. OBD-liitin ja pinnijärjestys (OBD-II Connector.)

### 2.1.3 OBD-II-protokollat ja -moodit

OBD-II-standardissa on määritelty viisi eri protokollaa, jolla voidaan olla yhteydessä moottorinohjainlaitteeseen. Yleensä ajoneuvoissa on kuitenkin käytössä vain yksi näistä, eikä muille protokollille varattuja pinnejä ole asennettu ollenkaan OBD-II-liittimeen. Ylivoimaisesti yleisin eurooppalaisissa ajoneuvoissa käytössä oleva protokolla on Robert Boschin kehittämä CAN-protokolla. (OBD-II Background.)

OBD-II-standardin mukaisia protokollia ovat

- SAE J1850 PWM (pulssinleveys modulaatio)
- SAE J1850 VPW (muuttuva pulssinleveys)
- ISO 9141-2 ( K- ja L-johtimet)
- ISO 14230 KWP2000
- ISO 15765 CAN.

Vuodesta 2008 lähtien ISO 15765 -CAN-protokolla on ollut saatavilla lähes kaikissa ajoneuvoissa, joten on projektin laajuuden kannalta mielekästä keskittyä vain CAN-protokollaan. ISO 15031 -standardissa on määritelty OBD-II-protokollalla käytettävät moodit. Eri moodeilla voidaan järjestelmällisesti pyytää moottorinohjainlaitteelta tietoja eri tiloista (kuva 2).

| | |
|---|---|
| 01 | Reaaliaikaiset diagnoositiedot |
| 02 | Freeze Frame |
| 03 | Vikamuistin luku |
| 04 | Vikamuistin tyhjennys |
| 05 | Lambda-anturin testaus |
| 06 | Osajärjestelmien itsetestaus, ajoittainen valvonta |
| 07 | Osajärjestelmien itsetestaus, jatkuva valvonta, vikamuisti |
| 08 | Merkkikohtaiset testit |
| 09 | Ajoneuvon tunnistetiedot |
| 0A | Vikakoodit, jotka poistuvat vain korjaamalla vika |

Kuva 2. OBD-II-moodit

1. Reaaliaikaiset diagnoositiedot, joita ovat esimerkiksi moottorin pyörintänopeus ja pakokaasun lämpötila. Laskennalliset suureet kuten suihkutusmäärä sekä suihkutusaika. Yleisiä tietoja ajoneuvon kokoonpanosta, esimerkiksi, onko ajoneuvo varustettu manuaali- vai automaattivaihteistolla.

2. Vian ilmenemishetkellä vallinneet olosuhteet tallentuvat Freeze Frame -moodiin. Näistä tiedoista voi olla arvokasta apua vian selvittämisessä. Yleensä tallennetaan moottorin pyörintänopeus, ulkoilman lämpötila, jäähdytysnesteen lämpötila ja ajonopeus.

3. Vikamuisti sellaisille vioille, jotka sytyttävät moottorin vikavalon. Tässä moodissa on myös tyhjennystoiminto kyseisille vioille.

4. Tämä moodi tyhjentää vikamuistit sekä Freeze Frameen tallentuneet tiedot. Tyhjennys poistaa myös Lambdan mitta-arvot, jotka löytyvät moodista 5.

5. Lambda-antureiden testaus- ja raja-arvojen tulostustoiminnot.

6. Osajärjestelmien tulostukset toiminnoista, jotka eivät kuulu OBD-II-järjestelmän jatkuvan valvonnan piiriin.

7. Osajärjestelmien tulostukset toiminnoista, jotka kuuluvat OBD-II-järjestelmän jatkuvan valvonnan piirin. Vikamuisti vioille, jotka eivät sytytä moottorin vikavaloa.

8. Varattu autonvalmistajan merkkikohtaisiin toimintoihin.

9. Ajoneuvon VIN-numero sekä CIN- ja CVN-koodit.

10.Vikakoodit, jotka voidaan poistaa vain korjaamalla vika.

(Bosch 2003: 12–13; Juhala ym. 2005: 50.)

### 2.1.4 OBD-II PID-tunnisteet

PID-tunnisteet ovat SAE J1979 -standardissa määriteltyjä tunnisteita, jolla ajoneuvoilta voidaan kysyä haluttuja tietoja. Ajoneuvot palauttavat vastauksen kyselyyn ensimmäisessä moodissa 1–4 bitin pituisilla vastauksilla hex-muodossa. Joidenkin PID-tunnisteiden arvot vaativat vielä laskutoimenpiteitä, jotta päästään todelliseen arvoon. Nämä laskutoimenpiteet löytyvät myös SAE J1979 -standardista. Tässä työssä käytettiin pääosin vain moodin 1 PID-tunnisteita, joihin ajoneuvo vastaa tällä hetkellä vallitsevilla arvoilla (kuva 3). (SAE J1979.)

| PID (hex) | PID (Dec) | Data bytes returned | Description |
|---|---|---|---|
| 00 | 0 | 4 | PIDs supported [01 - 20] |
| 01 | 1 | 4 | Monitor status since DTCs cleared. (Includes malfunction indicator lamp (MIL) status and number of DTCs.) |
| 02 | 2 | 2 | Freeze DTC |
| 03 | 3 | 2 | Fuel system status |
| 04 | 4 | 1 | Calculated engine load |
| 05 | 5 | 1 | Engine coolant temperature |
| 06 | 6 | 1 | Short term fuel trim—Bank 1 |
| 07 | 7 | 1 | Long term fuel trim—Bank 1 |
| 08 | 8 | 1 | Short term fuel trim—Bank 2 |
| 09 | 9 | 1 | Long term fuel trim—Bank 2 |
| 0A | 10 | 1 | Fuel pressure (gauge pressure) |
| 0B | 11 | 1 | Intake manifold absolute pressure |
| 0C | 12 | 2 | Engine RPM |
| 0D | 13 | 1 | Vehicle speed |
| 0E | 14 | 1 | Timing advance |
| 0F | 15 | 1 | Intake air temperature |
| 10 | 16 | 2 | MAF air flow rate |
| 11 | 17 | 1 | Throttle position |
| 12 | 18 | 1 | Commanded secondary air status |
| 13 | 19 | 1 | Oxygen sensors present (in 2 banks) |
| 14 | 20 | 2 | Oxygen Sensor 1<br>A: Voltage<br>B: Short term fuel trim |

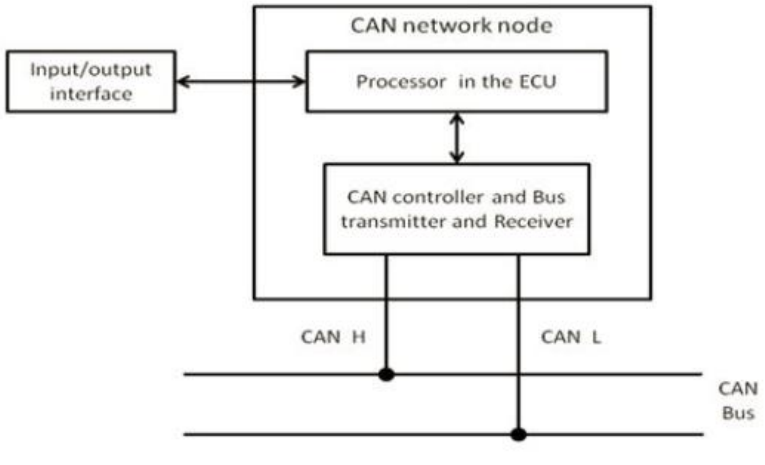
Kuva 3. OBD-II PID-tunnisteet moodissa 1 (SAE J1979.)

## 2.2 Controller Area Network (CAN) -väylä

Robert Bosch aloitti CAN-väylän kehittämisen jo vuonna 1983, joten ihan tuoreesta keksinnöstä ei ole kyse. Ensimmäiseen tuotanto autoon CAN-väylä pääsi vuonna 1988, jolloin se löytyi BMW:n 8-sarjalaisesta. Bosch on julkaissut viimeisimmän päivityksen CAN-protokollaan vuonna 1991. Sittemmin ISO on julkaissut standardit ISO 11898-1 ja ISO 11898-2. Kansainvälisten ISO-standardien myötä CAN-väylä voitiin ottaa mailmanlaajuisesti ajoneuvokäyttöön. Tarve CAN-väylälle syntyi, kun turvallisuuteen liittyvät järjestelmät alkoivat kehittyä ja sama tieto piti saada monelle ohjainlaitteelle samaan aikaan (kuva 4). (SAE J1979.)

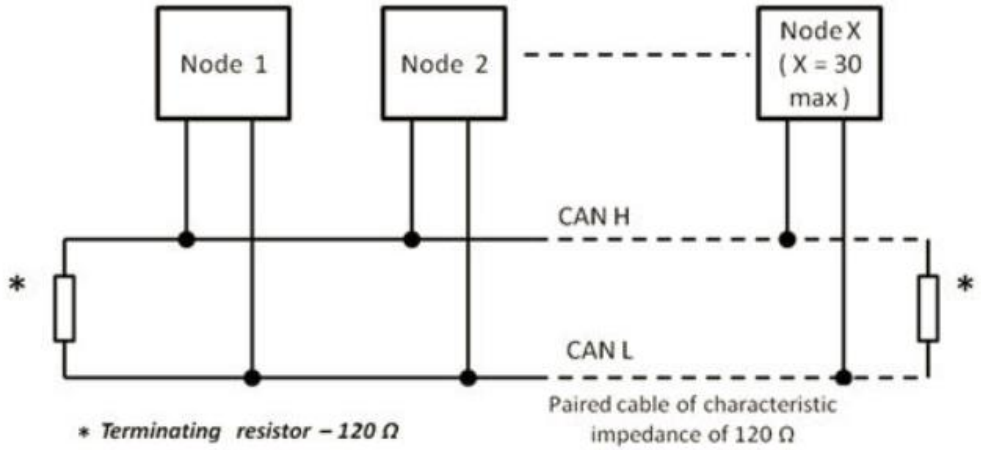
### Toimintaperiaate

CAN-väylän fyysinen rakenne koostuu kahdesta kierretystä kuparijohtimesta, joita kutsutaan CAN H- ja CAN L -johtimiksi. Johtimet on kierretty toisiinsa, jotta ajoneuvossa mahdollisesti esiintyvät sähkömagneettiset häiriöt indusoituvat varmasti molempiin johtimiin ja näin ne kumoavat toisensa (kuva 6). Väyläjohtimien päät ovat liitettyinä toisiinsa 120 ohmin vastuksilla, jotta vältyttäisiin viestien heijastumiselta. CAN-väylä on rakenteeltaan modulaarinen, eli ohjainlaitteita voidaan lisätä tai poistaa väylästä. Tämä tarkoittaa myös sitä, että kaikki väylällä kulkevat viestit ovat saatavilla jokaiselle väylään liitetylle ohjainlaitteelle (kuva 5). (Basics of Controller Area Network (CAN) bus – Part 1.)




Kuva 4. CAN-ohjainlaitteen rakenne (Basics of Controller Area Network (CAN) bus – Part 1.)

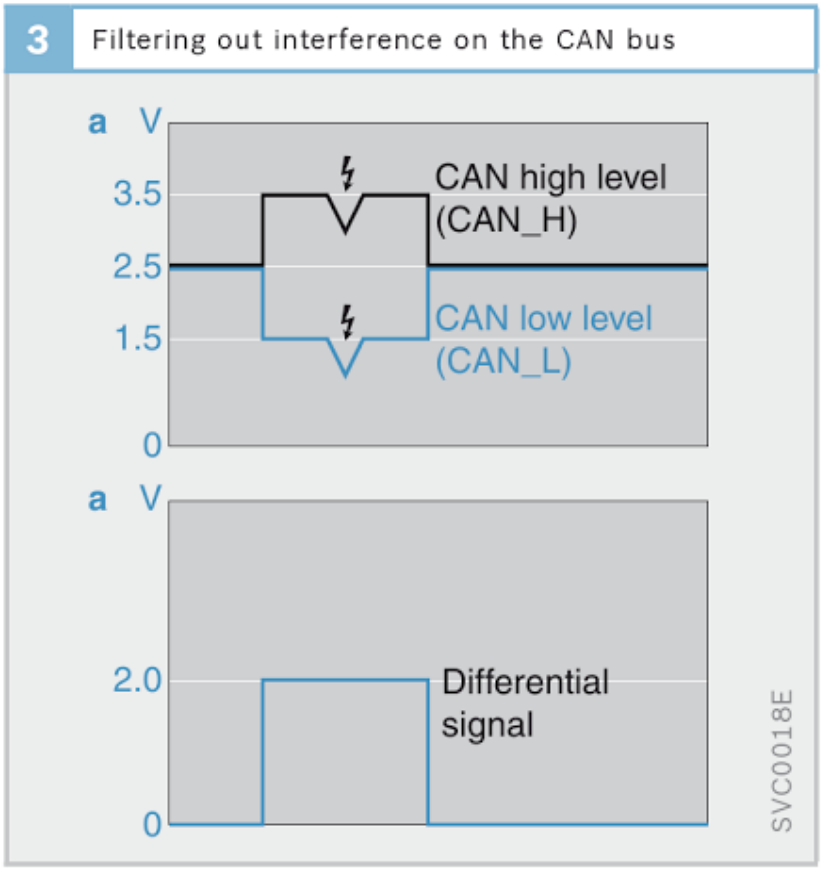
Ohjainlaitteiden rakenne koostuu yleensä mikrokontrollerista, CAN-ohjaimesta ja CAN-tulkista (Kuva 4). Mikrokontrolleri toimii varsinaisena ohjainlaitteena ja siihen voidaan liittää sensoreita sekä käyttölaitteita, joiden toimintaa hallitaan. Kun mikrokontrolleri haluaa lähettää tai vastaanottaa viestejä CAN-väylältä, se lähettää käskyn CAN-ohjaimelle, joka puolestaan muuttaa viestin jännitetasoiksi jotka CAN-tulkki sitten välittää CAN-väylälle. (Basics of Controller Area Network (CAN) bus – Part 1.)




Kuva 5. CAN-ohjainlaitteiden kytkentä CAN-väylään (Basics of Controller Area Network (CAN) bus – Part 1.)

Viestien lähettäminen CAN-väylälle tapahtuu jännitetasoa muuttamalla ennalta määriteltyjen jännitetasojen välillä. Jännitetasoja kutsutaan dominoivaksi ja resessiiviseksi jännitetasoksi ja kahden johtimen jännitetasoja muutetaan vastakkaisiin suuntiin niin, että resessiivinen taso on molemmilla johtimilla noin 2,5 volttia (kuva 6).

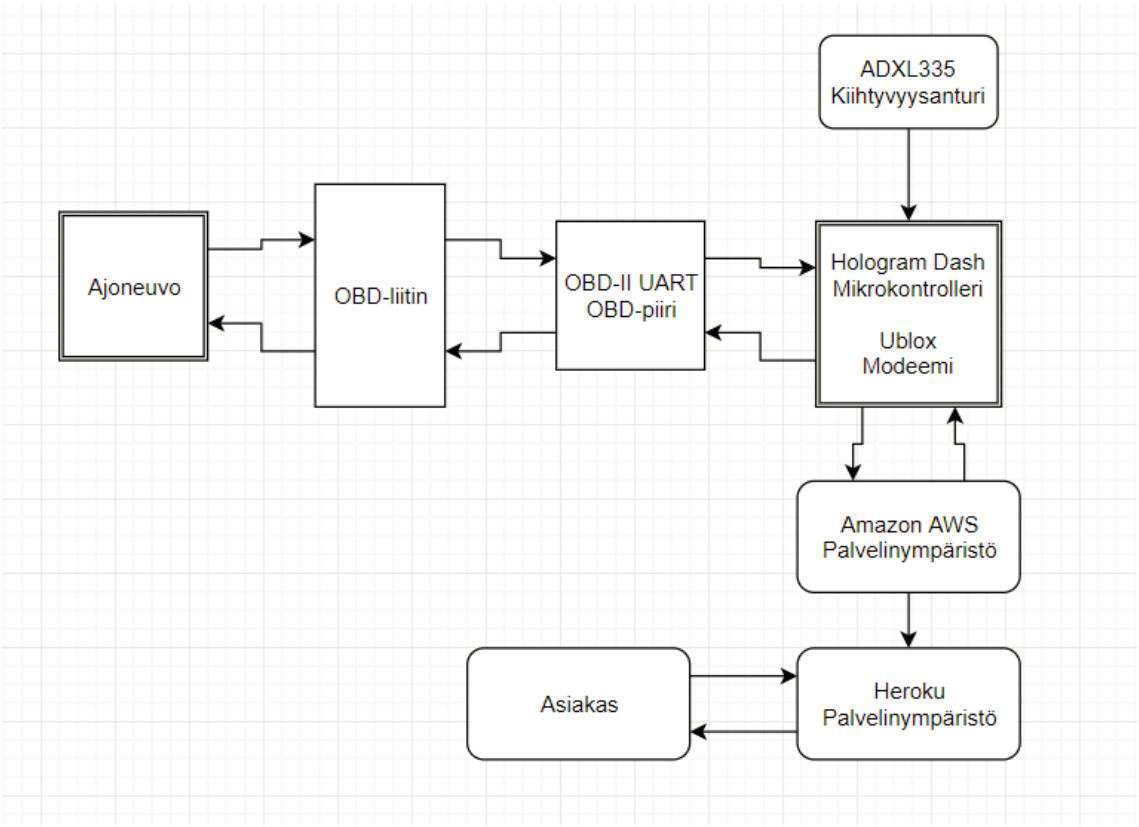
Tämän tilan CAN-tulkki tulkitsee tarkoittavaan loogista 0-tilaa. Looginen 1-tila saavutetaan muuttamalla molempien CAN-johtimien jännitettä 1 voltilla vastakkaisiin suuntiin. Tällöin CAN-johtimien välistä voidaan mitata 2 voltin jännite (kuva 6). (Mencher 2007: 94.)




Kuva 6. CAN-signaali (Mencher 2007: 94.)

# 3 OBD-etälukulaitteen elektroniikka

Heti projektin alkaessa kävi selväksi, että täysin Servoped Oy:n käyttötarkoitukseen soveltuvaa laitetta ei ollut markkinoilla, joten sellainen täytyi tuottaa itse. Uuden elektroniikkalaitteen suunnittelu on haastavaa, ja tätä työtä helpottaaksemme käytettiin valmiita kehitystarkoitukseen suunniteltuja komponentteja, joita yhdistelemällä päästiin haluttuun lopputulokseen. Laitteen toiminta on esitetty toimintakaaviossa (kuva 7).




Kuva 7. OBD-etälukulaitteen toimintakaavio

## 3.1 Toimintaperiaate

Laitteen toiminta rakentuu mikrokontrollerin ympärille, joka käskee muita komponentteja toimimaan halutulla tavalla ja oikeaan aikaan. Mikrokontrolleria voidaankin ajatella esimerkiksi ajoneuvon kuljettajana, joka aina loppukädessä päättää mitä tehdään, vaikka ajoneuvo voikin suorittaa myös itsenäisesti tehtäviä, kuten esimerkiksi pitämällä nopeuden vakionopeudensäätimelle asetetussa arvossa.

Ennen kuin tiedonkeruu voi alkaa, on tunnistettava, missä tilassa ajoneuvo juuri sillä hetkellä on. Ajoneuvolta halutaan kerätä tietoja vain silloin, kun ajoneuvo on käytössä. Tästä seuraa tarve tunnistaa milloin OBD-etälukulaitteen tulisi olla päällä. Ratkaisu tähän ongelmaan löytyi tarkkailemalla ajoneuvon jännitetasoa, jota voidaan lukea OBD-liittimen pinnistä 16 (kuva 1). Henkilöajonevoissa on käytössä 12 voltin sähköjärjestelmä, ja kun ajoneuvon moottori on käynnissä, luettava jännite nousee laturin ansiosta yli 14 volttiin. Samalla periaatteella OBD-etälukulaite asetetaan virransäästötilaan, kun ajoneuvo ei ole käynnissä, jotta ajoneuvon akku ei tyhjentyisi.

Hologram Dash tukee jo itsessään jännitteen lukemista, joten tämän ominaisuuden lisäämiseen ei tarvittu lisäkomponentteja. Jännitetasoa seuraamalla voimme päätellä, milloin on oikea aika aloittaa sekä lopettaa tiedonkeräys. Kun on tunnistettu ajoneuvon olevan käynnissä, mikrokontrolleri lähettää aloituskäskyt muille komponenteille, jonka jälkeen dataa kerätään kerran sekunnissa ja se lähetetään palvelimelle. Samalla kun data lähetetään palvelimelle, palvelin lähettää myös paluuviestin, jolla voidaan etänä hallita OBD-etälukulaitteen toimintaa.

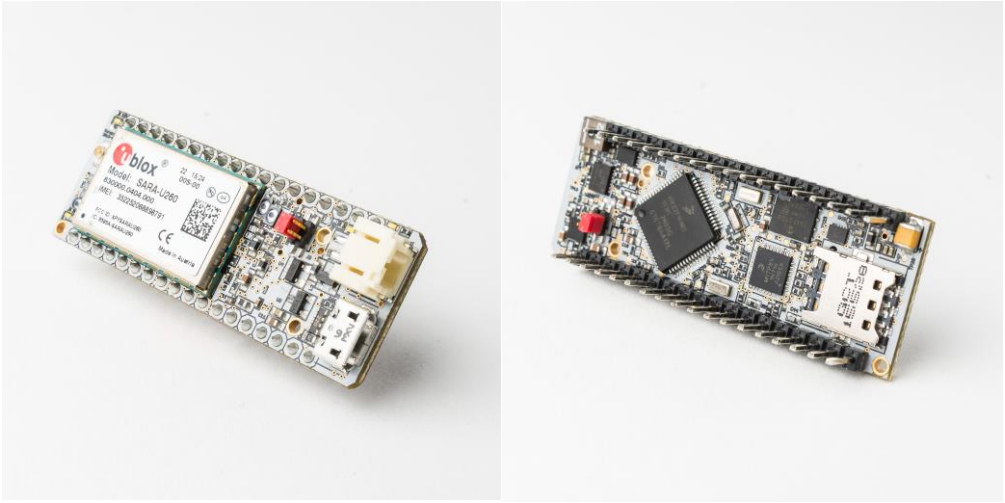
Palvelimella vastaanotettu data jäsennetään ja siitä rakennetaan algoritmien avulla esimerkiksi kuljettajan ajokäyttäytymistä arvioiva profiili. Ajoneuvosta kerätään myös mahdolliset vikakoodit, jolloin ajoneuvon kuntoa on mahdollista seurata myös etänä.

## 3.2 Mikrokontrolleri

Mikrokontrollerilla tarkoitetaan pienelle piirilevylle rakennettua laitetta tai sirua, joka sisältää yleensä prosessorin, muistia sekä ohjelmoitavan syöttölaitteiden liitännän. Mikrokontrollereja kutsutaan usein myös sulautetuiksi järjestelmiksi, sillä osat on koottu hyvin pienelle piirilevylle tai sirulle. Käyttökohteita ovat lähes kaikki mekaaniset tai sähköiset laitteet, joiden toimintaa täytyy hallita. Mikrokontrolleri ohjelmoidaan tekemään sille suunniteltu työ, ja se voi myös muuttaa toimintaansa mukautumalla ympäristöstä sille annetun tiedon perusteella. Esimerkiksi ajoneuvoissa kaikki ohjainlaitteet käyttävät mikrokontrollereja. (Microcontrollers.)

Hologram Dash

Hologram Dash -mikrokontrolleri valittiin projektiin, sillä se on avoimen lähdekoodin esineiden internetalusta eli siis täysin muokattavissa. Lisäksi sen pieni fyysinen koko oli edellytys tuotteen onnistumiselle, sillä ajoneuvon OBD-liitäntäpistoke sijaitsee yleensä ahtaassa tilassa ja laitteemme tulee olla kiinni aina, kun ajoneuvo liikkuu. Muita valintaperusteita olivat täysi muokattavuus, riittävän nopea internetyhteys sekä etäpäivitysmahdollisuus. Hologram Dash laitteessa on Ubloxin valmistama 4g-modeemi (kuva 8). (Hologram Dash Datasheet.)



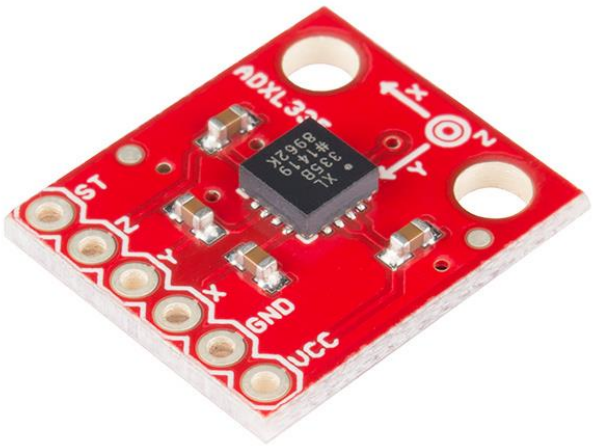
Kuva 8. Hologram Dash -mikrokontrolleri (Hologram Dash: Cellular Dev Kit + Global Data Plan.)

Lisäksi Hologram Dash tukee natiivisti Arduino-ohjelmointikieltä, eli ohjelmoimiseen voidaan käyttää suoraan Arduino IDE -ympäristöä. Arduino on maailmanlaajuisesti suosittu kehittämisympäristö, jonka käyttäjiä löytyy harrastelijoista ammattilaisiin asti. Vaikka Hologram tarjoaa Dashin mukana sen omaa verkkopalvelua sekä sim-korttia, toimii Hologram Dash myös muiden operaattoreiden sim-korteilla. (Hologram Dash Datasheet.)

## 3.3 ADXL335-kiihtyvyysanturi

Ajoneuvon tarkkaan seuraamiseen tarvitaan myös kiihtyvyysanturi. Monessa ajoneuvossa on itsessään jo kiihtyvyysanturointi, sillä niitä käytetään muun muassa luistonesto- ja ajonvakautusjärjestelmissä. Ajoneuvon omien kiihtyvyysantureiden dataa ei kuitenkaan ole standardoidusti saatavilla, joten liittämällä erillisen kiihtyvyysanturin OBD-etälukulaitteeseen saamme kaikista ajoneuvoista kiihtyvyysdataa.

Kolmiakselinen ADXL335-kiihtyvyysanturi valittiin pienen koon sekä helpon käytettävyyden takia. ADXL335 tarjoaa kaikkien kolmen akselin kiihtyvyystiedot erillisinä pinneinä, joista ne mitataan jännitetasoina (kuva 9). Jännitetasonmittaus sopii tähän työhön erittäin hyvin, sillä Hologram Dash pystyy mittaamaan jännitetasoja ilman lisämuutoksia. (ADXL 335.)



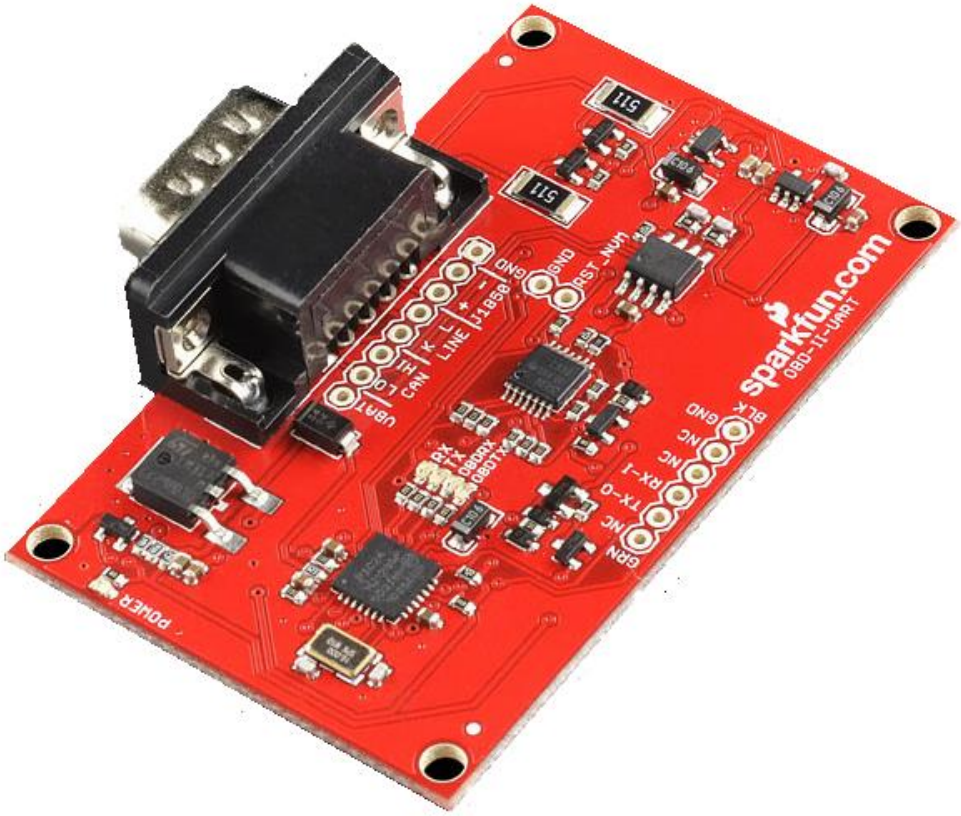

Kuva 9. Kolmiakselinen ADXL 335 -kiihtyvyysanturi (ADXL 335.)

## 3.4 OBD-II UART -piiri

Tähän työhön valittiin Sparkfun:n valmistama OBD-II UART -piiri, joka on yhteensopiva laajasti käytössä olevaan ELM327-komentosarjaan. Valinta oli helppo, sillä laitteen voi kytkeä suoraan ajoneuvon OBD-porttiin mukana tulleen johdon avulla eikä muutoksia

piirilevyyn tarvita. Piiri saa myös käyttöjännitteensä ajoneuvon liittimestä, joten riittää, että kytketään vain maadoitus sekä RX- ja TX-pinnit, jolla mikrokontrolleri kommunikoi OBD-piirin kanssa sarjamuotoisella yhteydellä. (OBD-II UART.)

OBD-piiri on myös riittävän pieni kooltaan, jotta se mahtuu koteloon ilman fyysisiä muutoksia (kuva 10). Toiminnan kannalta tärkeää oli, että OBD-piiri kykenee pyytämään ajoneuvolta tietoja vähintään kerran sekunnissa. Tämä OBD-piiri tukee myös useiden parametrien pyytämistä kerralla, mikä helpottaa tietojen käsittelyä suuresti.



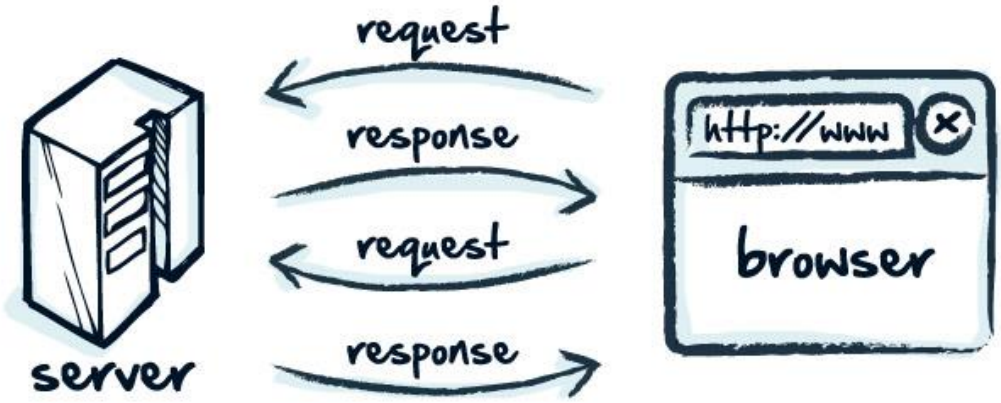

Kuva 10. OBD-II UART -piiri (OBD-II UART.)

# 4 Palvelinympäristö

Palvelimella tarkoitetaan tietokonetta, jonka tehtävä on palvella muita laitteita, jotka tekevät sille pyyntöjä. Näitä muita laitteita kutsutaan tässä yhteydessä asiakkaiksi. Palvelimia käytetään tyypillisesti tiedonkäsittelyssä, tiedonsiirrossa, verkkosivujen tarjonnassa, pelipalveluissa sekä sovelluspalveluissa. (Servers in computer networking.)

Nykypäivänä useimmat palvelimet toimivat pyyntö-vastausmallilla, joka tarkoittaa, että asiakas tekee ensin pyynnön palvelimelle, joka sitten suorittaa tehtävän ja palauttaa vastauksen takaisin asiakkaalle. Näin tapahtuu esimerkiksi verkkosivua käytettäessä, jolloin verkkoselain tekee palvelimelle pyyntöjä käyttäjän toiminnan perusteella (kuva 11).

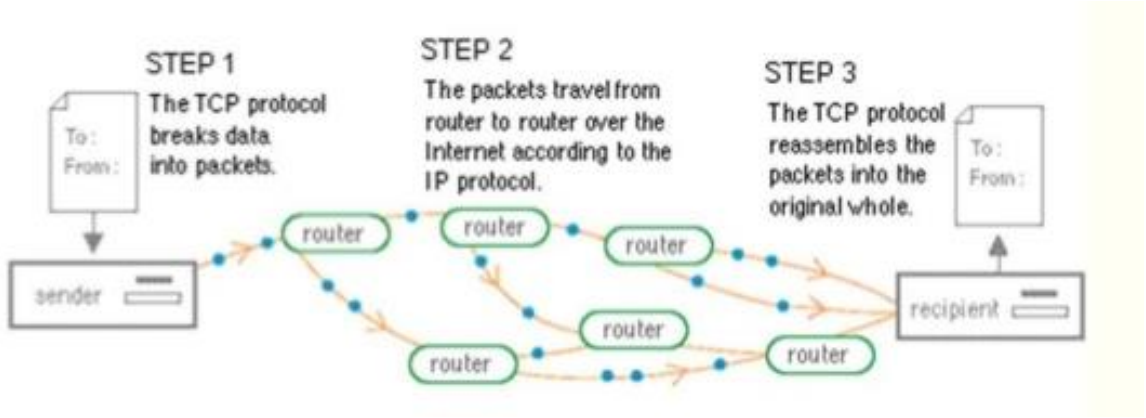
Vastaavasti Heroku-palvelinympäristössä toimiva palvelu tuottaa loppukäyttäjälle sisältöä OBD-etälukulaitteen keräämistä tiedoista. Kun käyttäjä tekee pyynnön nähdäkseen esimerkiksi vikakoodit lähettää verkkoselain pyynnön palvelimelle, joka hakee vikakoodit tietokannasta ja sitten palauttaa vastauksen verkkoselaimelle.




Kuva 11. Pyyntö-vastausmalli (Web Development – The Basics.)

## 4.1 TCP/IP-protokolla

TCP/IP-protokolla on yleisin käytössä oleva protokolla, joka koostuu IP-protokollasta ja TCP-protokollasta. IP-protokolla on alemman tason protokolla, joka vastaa laitteiden osoitteista sekä pakettien reitityksestä. TCP-protokolla on yleisin IP-protokollan päällä käytetty protokolla ja sen tehtävä on muodostaa yhteys laitteiden välille, sekä huolehtia datapakettien lähettämisestä. IETF-standardointiorganisaatio vastaa TCP/IP-protokollaperheen standardoinnista ja hyväksynnöistä. TCP-protokolla vastaa siis yhteyden muodostamisesta ja IP-protokolla datapakettien lähettämisestä sekä vastaanottamisesta (kuva 12). OBD-etälukulaitteen keräämät tiedot lähetettiin palvelimelle myös käyttäen TCP/IP-protokollaperhettä. (TCP/IP.)




Kuva 12. TCP/IP-protokolla (TCP/IP.)

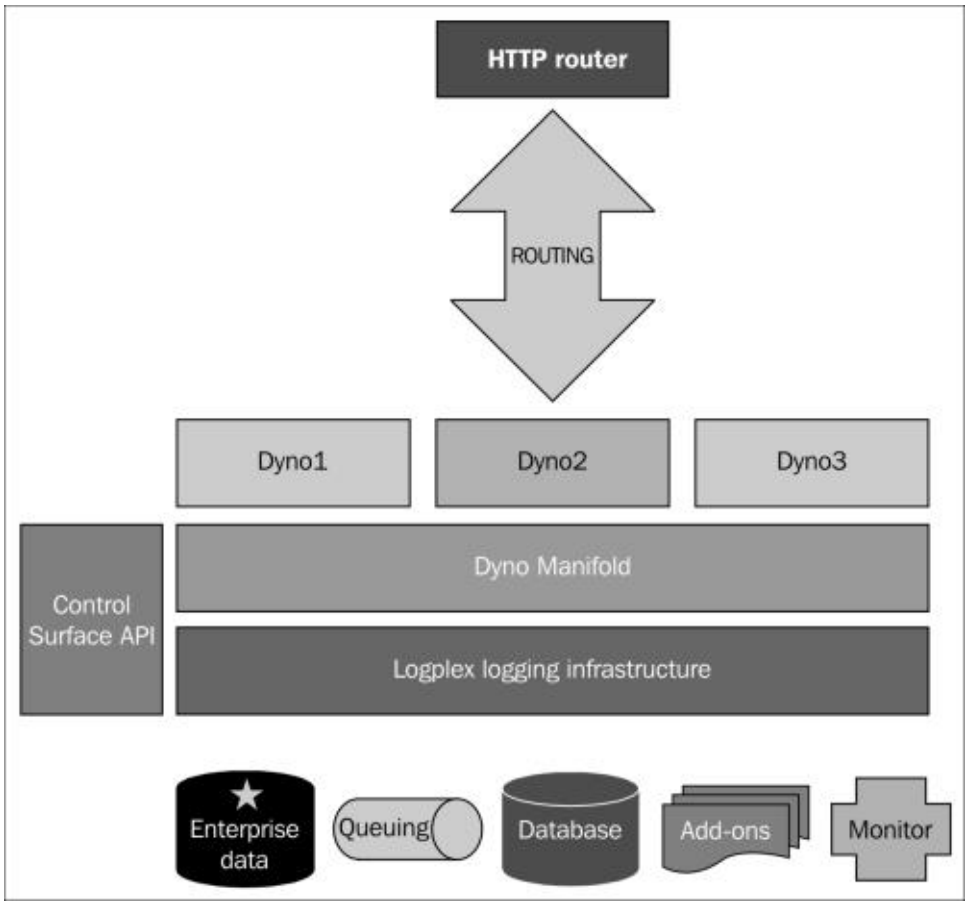
## 4.2 Amazon AWS

Amazon AWS on Amazon.com:n perustama pilvipalvelu, joka tarjoaa yrityksille IT-palveluita verkossa. AWS on verkkoympäristö, jonne yritykset voivat siirtää omien palvelintensa sekä palveluidensa toiminnan. Palvelua markkinoidaan erityisesti sen helpolla skaalattavuudella, joka mahdollistaa yrityksen nopeaa kasvua. Muita etuja ovat palveluiden toimintavarmuus, sillä AWS:n palvelinverkko on erittäin laaja. Tiedot ovat myös turvassa, sillä tiedot varmuuskopioidaan automaattisesti, eikä tietoja pääse häviämään, vaikka yksittäinen palvelin hajoaisikin.

Tässä työssä Amazon AWS -pilvipalvelua käytettiin reitittimenä, joka vastaanotti OBD-etälukulaitteen lähettämät tiedot käyttäen TCP/IP-protokollaperhettä. Tietojen vastaanottamisen jälkeen AWS lähetti tiedot edelleen Heroku-palvelin ympäristöön, jossa tiedot tarjotaan asiakkaille heidän toiveidensa mukaan.

## 4.3 Heroku

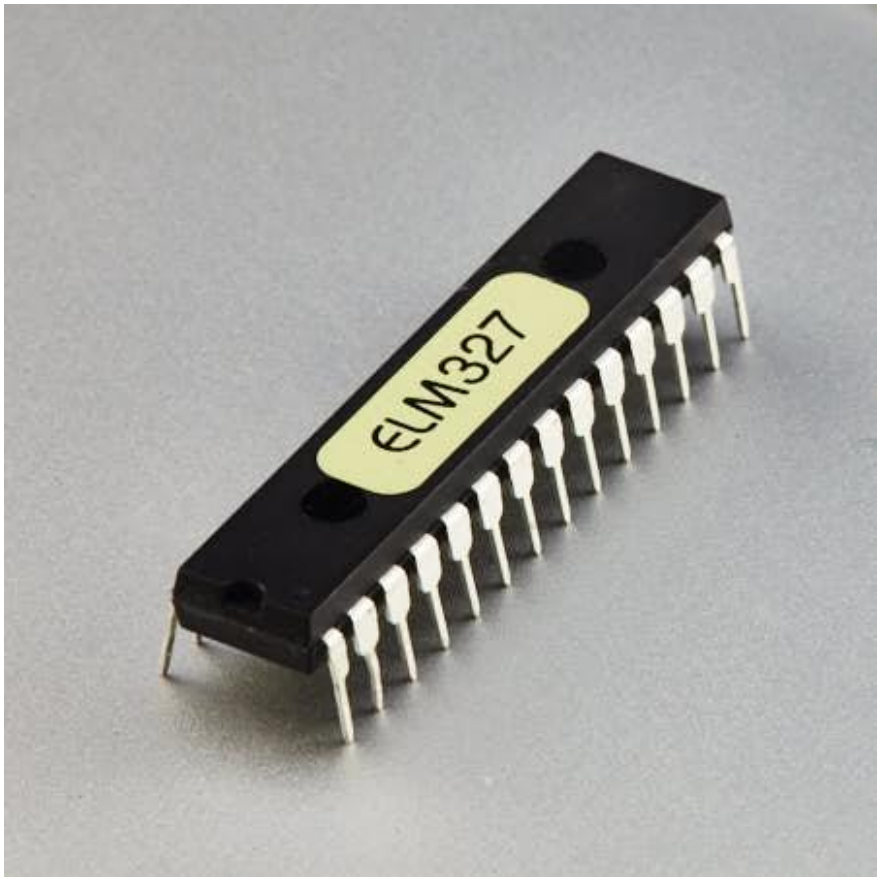
Heroku on vuonna 2007 perustettu verkkopalvelu, joka mahdollistaa useiden ohjelmointikielien käytön. Heroku tukemia kieliä ovat Ruby, Java, Node.js, Scala, Clojure, Python, PHP sekä Go. Heroku on suunniteltu niin, että jokaisella asiakkaalla on oma verkko-osoite, johon http-viestit ohjataan. Yhden verkko-osoitteen alla voi kuitenkin toimia monia ohjelmia, joita kutsutaan dynoiksi. Etuna tässä järjestelyssä on se, että dynoja voidaan kirjoittaa eri ohjelmointikielillä, mutta ne voivat silti toimia yhdessä saman verkko-osoitteen alla (kuva 13). (An overview of Heroku's architecture.)




Kuva 13. Heroku-palvelinrakenne (An overview of Heroku's architecture.)

## 4.4 ELM327

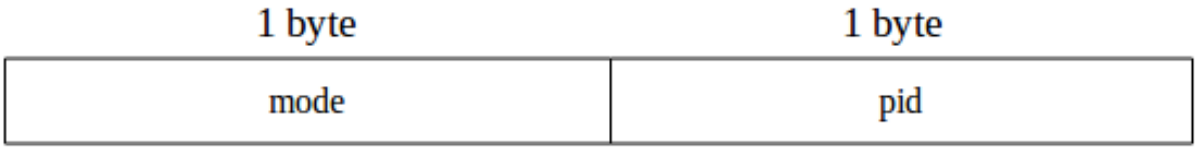
ELM327 on ELM Electronicsin valmistama mikrokontrolleri, joka on suunniteltu tuottamaan helppokäyttöinen käyttöliittymä ajoneuvon OBD-liitäntään (kuva 14). ELM327-piirin kanssa voidaan keskustella sarjamuotoisella tietoliikenneyhteydellä ja se voidaan muodostaa myös USB:n, WIFIn tai Bluetoothin välityksellä. ELM327-piiri osaa automaattisesti tunnistaa ajoneuvossa käytössä olevan OBD-protokollan, ja se tukee kaikkia SAE J1979 -standardissa määriteltyjä protokollia. (ELM 327 v2.2.)




Kuva 14. ELM327-piiri (ELM 327 v2.2.)

ELM 327 -komentosarja perustuu Dennis Hayesin vuonna 1981 kehittämään AT-komentokieleen. Komentosarja koostuu lyhyistä komentokehotteista, jotka alkavat ”AT”-kirjaimilla. Komentokehotteilla pyydetään ajoneuvolta SAE J1979 -standardissa määriteltyjen PID-tunnisteiden perusteella. (ELM 327 v2.2.)

Tietoliikenteen vähentämiseksi ELM 327-piiriltä voidaan pyytää PID-tietoja lisäämättä kometokehotteeseen ”AT”-etuliitettä. Tällöin ELM 327-piirille lähetettävä viesti koostuu OBD-moodista ja PID:stä, josta tietoja halutaan (kuva 15).

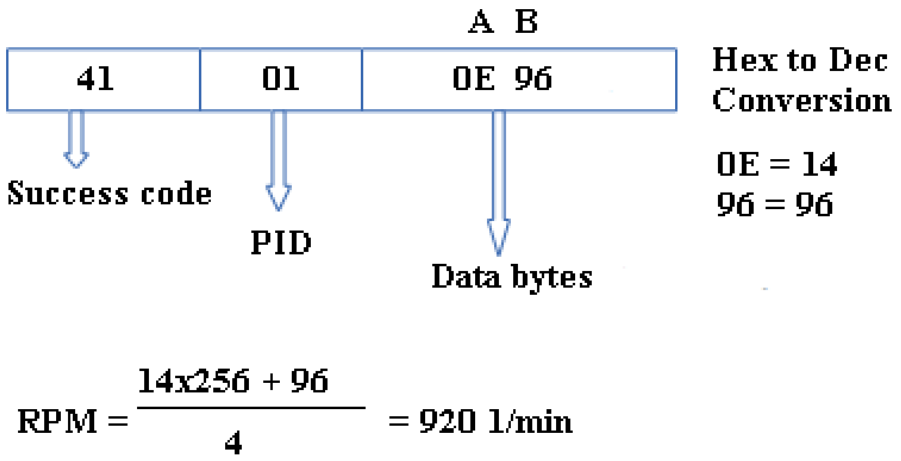



Kuva 15. ELM 327:n PID-pyyntö (SAE J1979.)

Ajoneuvon tämänhetkistä moottorin pyörintänopeutta kysyttäessä valitaan moodi ”01” sekä sitä vastaava PID-tunniste, joka on ”0C”. Lähetettäväksi viestiksi saadaan siis ”010C”. SAE J1979 -standardissa on myös määritelty, missä muodossa ajoneuvon on palautettava vastaus tiettyä PID-tunnistetta kysyttäessä. (SAE J1979.)

Moottorin pyörintänopeuden kohdalla vaatimus on, että ajoneuvo palauttaa onnistuneesta kyselystä kaksi bittiä, joista ensimmäistä merkitään kirjaimella ”A” ja toista kirjaimella ”B” (kuva 16). Vastaus voi sisältää enintään 4 bittiä tietoa. (SAE J1979.)

Ajoneuvo vastaa kyselyyn ensin ilmoittamalla, että kysely on onnistunut ”41”, ja tämän jälkeen seuraa kysytty tunniste sekä varsinainen vastaus hex-muodossa (kuva 16). Tämän jälkeen päästäksemme oikeaan arvoon moottorin pyörintänopeuden kohdalla, täytyy bitti ”A” kertoa 256:lla, tulokseen lisätä bitti ”B” ja tämän jälkeen jakaa vielä 4:llä. (SAE J1979.)




Kuva 16. ELM 327:n PID-vastaus RPM kyselyyn (OBD Scanner using ELM327.)

ELM 327 -komentosarjalla on myös mahdollista pyytää ajoneuvolta useita PID-tunnisteita yhdellä komentokehotteella, mutta tämä toiminallisuutta ei vaadita standardissa, joten kaikki ajoneuvovalmistajat eivät tue ominaisuutta. Jos ajoneuvovalmistaja kuitenkin tukee kyseistä ominaisuutta, se vähentää huomattavasti tarvittavaa tietoliikennettä, kun jopa viisi PID-tunnistetta voidaan kysyä yhdellä komentokehotteella. (ELM 327 v2.2.)

## 4.5 Hologram Dash -pääohjelma

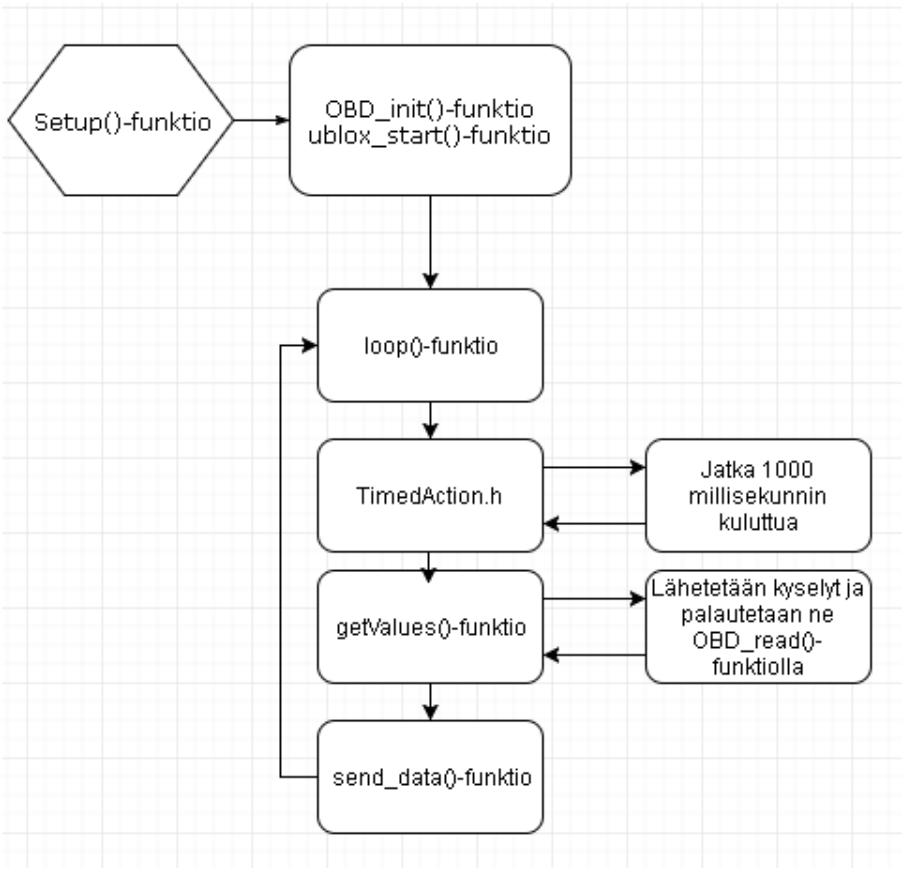
Hologram Dash -laitteen ohjelmoinnissa täytyi ottaa huomioon erityisesti järjestys, jossa koodia suoritetaan, jotta laite pystyy toimimaan, vaikka ajoneuvo ei hetkellisesti vastaisikaan pyyntöihin. Lisäksi aina OBD-etälukulaitteen käynnistyessä täytyy tunnistaa mitä OBD-protokollaa kyseinen ajoneuvo käyttää.

Käynnistyessään Hologram Dash suorittaa ensimmäiseksi setup()-funktion (kuva 18), jossa määritellään muiden komponenttien toimintaa ja annetaan esimerkiksi modeemille palvelimen tiedot, jotta myöhemmin ajoneuvolta luettava data voidaan lähettää oikein.

Setup()-funktion jälkeen siirrytään suorittamaan loop()-funktiota (kuva 18). Tavallisesti loop()-funktiota suoritetaan loputtomiin, kunnes laitteelle annetaan lopetuskomento tai sen virta katkaistaan. OBD-etälukulaitteen halutaan suorittavan eri tehtäviä eri aikoina, joten saman koodin suorittaminen loputtomasti uudestaan ei tullut kyseeseen.

Ongelma ratkaistiin käyttämällä TimedAction.h-kirjastoa, joka mahdollistaa eri funktioiden suorittamisen halutulla ajanhetkellä. Käytännössä tämä mahdollistaisi myös sen, että dataa voitaisiin kerätä ajoneuvolta jatkuvasti, mutta lähettäminen tapahtuisi kollektiivisesti tietyin väliajoin. Tässä työssä päätettiin kuitenkin lähettää data heti, kun se on kerätty. Data kerätään ajoneuvolta ensin kutsumalla getValues()-funktiota 1000 millisekunnin välein ja se lähetetään heti perään send_data()-funktiolla, koska sitä kutsutaan myös 1000 millisekunnin välein.

Loop()-funktion sisällä ensimmäisenä olevalla getValues()-funktiolla (kuva 18) lähetetään sarjayhteyden avulla pyynnöt ensin OBD-piirille, joka välittää kyselyt ajoneuvolle ja palauttaa sitten vastauksen käyttäen sarjayhteyttä. Kyselyt lähetetään Serial1.println-komennolla, mutta vastaukset täytyy kuitenkin lukea erillisellä OBD_read()-funktiolla (kuva 18). OBD-piiri odottaa ajoneuvolta vastausta jokaiseen kyselyyn 200 millisekunnin ajan, minkä jälkeen se siirtyy seuraavaan käskyyn. OBD_read()-funktio myös lopettaa itsensä noin 200 millisekunnin päästä, jos OBD-piiri ei vastaa.




Kuva 18. Hologram Dash -pääohjelman logiikkakaavio

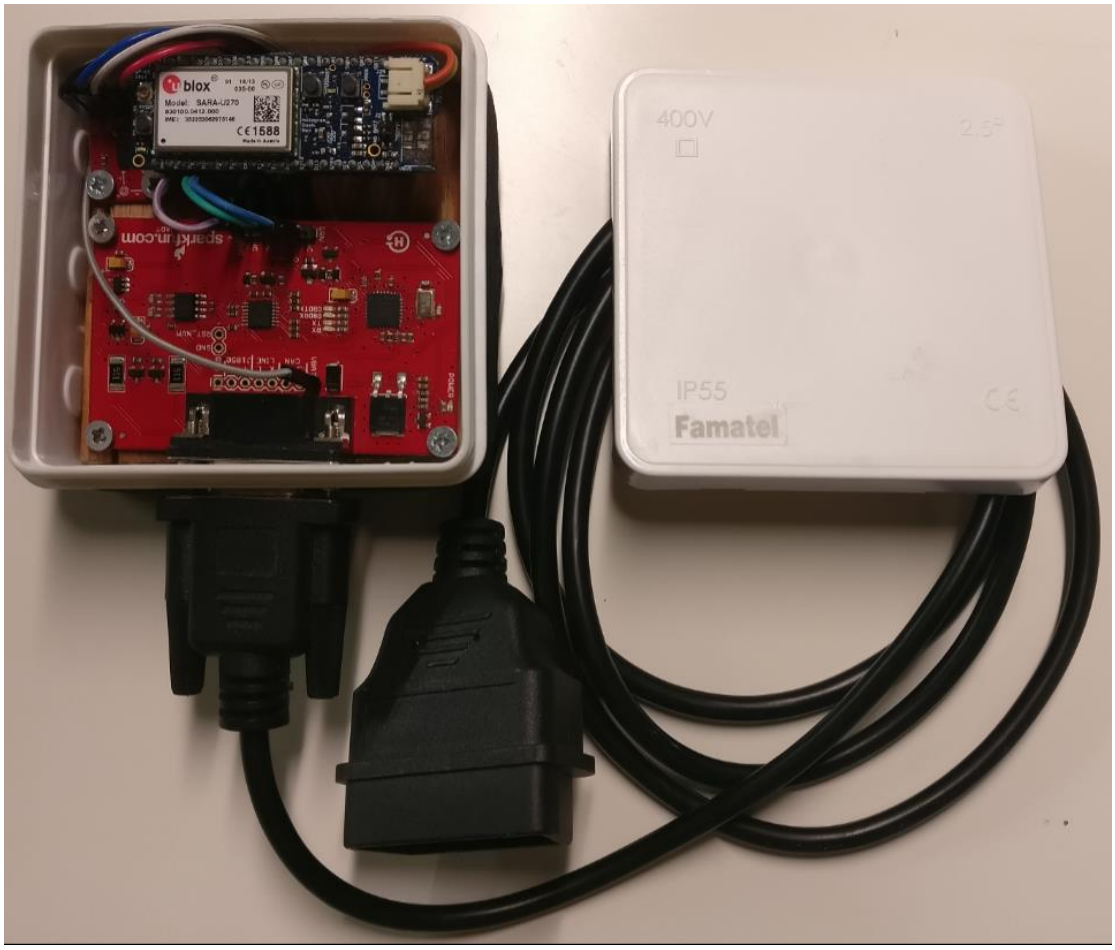
GetValues()-funktiolla on tarkoitus tallentaa ajoneuvolta saadut tiedot hetkeksi OBD-etälukulaitteen omaan muistiin, josta ne voidaan seuraavalla send_data()-funktiolla sitten lähettää palvelimelle. GetValues()-funktio tallentaa aivan ensiksi ajanhetken, jotta palvelimella voidaan tietää, milloin tiedot on kerätty, vaikka tiedonsiirrossa esiintyisi viiveitä. Tämän jälkeen funktio kerää ensin ajoneuvolta saatavat tiedot ja sen jälkeen pyytää ADXL335-kiihtyvyysanturilta kyseisellä hetkellä vallitsevat kiihtyvyydet ja päivittää ne niille nimettyihin muuttujiin.

Send_data()-funktiolla kerätään getValues()-funktion tallentamat tiedot ja lisätään ne yhteen merkkijonoon, jonka modeemi lähettää palvelimelle (kuva 18). Selvyyden vuoksi tässä funktiossa data lähetetään muodossa ”muuttuja1=arvo1&muuttuja2=arvo2…”, vaikka tiedot voitaisiin lähettää myös pelkkinä desimaalilukuina, kun tiedetään, missä järjestyksessä arvot tulevat.

Ubloxin modeemit on suunniteltu toimimaan AT-komentokehotteilla, ja käytettäessä TCP/IP-rajapintaa täytyy myös tietää, montako merkkiä lähetettävä merkkijono sisältää. Tästä syystä merkkijono rakennetaan monesta osasta ja tietyssä järjestyksessä. Ensiksi merkkijono obdString tyhjennetään ja sen jälkeen siihen lisätään järjestyksessä aiemmin ajoneuvolta saadut muuttujat. Kun muuttujat on lisätty, lasketaan varsinaisen viestin pituus. Tämän jälkeen modeemille lähetettävä viesti kootaan AT-komennosta, varsinaisen viestin pituudesta sekä varsinaisesta viestistä. Pääohjelma löytyy kokonaisuudessaan liitteestä 1.

# 5 Yhteenveto

Projekti oli onnistunut, sillä Servoped Oy sai OBD-etälukulaitteen avulla todistettua yrityksen toimintasuunnitelman toimivaksi sekä hyvän lähtökohdan tuotteen jatkokehitykselle. Työ oli haastava, koska siinä yhdistyi ajoneuvotekniikan, elektroniikka suunnittelun ja ohjelmoinnin osa-alueet. Tutkimustyötä ja kokeilua tarvittiin myös paljon, sillä läheskään kaikkiin ongelmiin ei löytynyt vastauksia, koska vastaavia laitteita ei ole laajasti saatavilla. Laite saatiin mahtumaan sähkökytkentärasiaan, jossa sitä on helppo kantaa mukana (kuva 22).




Kuva 22. OBD-etälukulaite

## Lähteet


ADXL 335. 2014. Verkkoaineisto. Sparkfun. Https://www.sparkfun.com/products/9269. Luettu 29.10.2017.

An overview of Heroku's architecture. 2017. Verkkoaineisto. Safari Books Online. Https://www.safaribooksonline.com/library/view/heroku-cloud-application/9781783550975/ch01s07.html. Luettu 8.11.2017.

Basics of Controller Area Network (CAN) bus – Part 1. 2015. Verkkoaineisto. Automobile Electrical Systems. Https://autoelectricalsystems.wordpress.com/2015/11/10/basics-of-controller-area-network-can-bus-part-1/. Luettu 8.11.2017.

ELM 327 v2.2. 2017. Verkkoaineisto. ELM Electronics. Https://www.elmelectronics.com/ic/elm327/?v=f0aa03aaca95. Luettu 8.11.2017.

Hologram Dash: Cellular Dev Kit + Global Data Plan. 2017. Verkkoaineisto. Hologram. Https://angel.co/projects/216659-hologram-dash-cellular-dev-kit-global-data-plan. Luettu 7.11.2017.

Hologram Dash Datasheet. 2017. Verkkoaineisto. Hologram. Https://hologram.io/docs/reference/dash/datasheet/. Luettu 11.7.2017.

Juhala, M.; Lehtinen, A.; Suominen, M. & Tammi, K. 2005. Moottorialan sähköoppi. Jyväskylä: Gummerus kirjapaino Oy.

Mencher, Bernhard. 2007. Automotive Electrics, Automotive Electronics. E-kirja. Robert Bosch GmbH.

Microcontrollers. 2017. Verkkoaineisto. Future Electronics. Http://www.futureelectronics.com/en/Microcontrollers/microcontrollers.aspx. Luettu 11.7.2017.

Mikä on OBD? 2015. Verkkoaineisto. Elekma Oy. Https://www.elekma.com/mika_on_obd. Luettu 20.9.2017.

OBD-II Background. 2011. Verkkoaineisto. B&B Electronics. Http://www.obdii.com/background.html. Luettu 11.7.2017.

OBD-II Connector. 2011. Verkkoaineisto. B&B Electronics. Http://www.obdii.com/connector.html. Luettu 11.7.2017.

OBD Scanner using ELM327. 2017. Verkkoaineisto. DeepThoughts Engineering. Http://dthoughts.com/blog/2014/11/06/obd-scanner-using-elm327/. Luettu 29.10.2017.

OBD-II UART. 2014. Verkkoaineisto. Sparkfun. Https://www.sparkfun.com/products/9555. Luettu 29.10.2017.

SAE J1979. 2002. Verkkoaineisto. Society of Automotive Engineers. Https://law.resource.org/pub/us/cfr/ibr/005/sae.j1979.2002.pdf. Luettu 14.12.2017.

Servers in computer networking. 2017. Verkkoaineisto. Lifewire. Https://www.lifewire.com/servers-in-computer-networking-817380. Luettu 29.11.2017.

TCP/IP. 2015. Verkkoaineisto. Giordano Bellini. Http://atelier.inf.unisi.ch/~dalsat/sai/projects/2015/html/internet/tcp_ip.html. Luettu 8.11.2017.

Web Development – The Basics. 2017. Almir Mesic. Verkkoaineisto. Https://www.almirmesic.com/tag/learning-2/. Luettu 7.11.2017.

## Hologram Dash -ohjelmakoodi

```
#include <Time.h>
#include <TimeLib.h>
#include <TimedAction.h>
#include <ADXL335.h>

int Time1, rpm1, speed1, load1, pos1, temp1, lenght1;
String socket1String = String("\"AT+USOWR=0,");
String obdString;
String send1String;


int string_width;
const int pin_x = A0;
const int pin_y = A1;
const int pin_z = A2;
const float aref = 3.3;
int i = 0;
float x;
float y;
float z;

ADXL335 accel(pin_x, pin_y, pin_z, aref);

char send_string;
char rxData[32];
char rxWord1[32];
char rxWord2[32];
char rxIndex=0;

void setup()
{
 HologramCloud.enterPassthrough();
 Serial.begin(115200);
 OBD_init();
 ublox_start();
}

void OBD_init(void)
{
 Serial.println("ATZ");  //Reset the OBD-II-UART
 //Wait for a bit before starting to send commands after the reset.
 delay(1000);
 OBD_read();
 delay(20);
 Serial.println("ATE0");  //Set autodetect protocol
 delay(20);
 OBD_read();
 delay(20);
 Serial.flush();
```

```
}

void ublox_start()
{
 SerialSystem.write("AT+CGACT=1,2\n"); //Activate stored profile 1
 delay(1000);
 SerialSystem.write("AT+UPSDA=0,3\n"); //Activate method 0
 delay(1000);
 SerialSystem.write("AT+USOCR=6\n"); //Activate socket connection type
 delay(1000);
 SerialSystem.write("AT+USOCO=0,\"52.11.90.112\",9500\n"); //Set Socket IP @
52.11.90.112 & Port 9500
 delay(1000);
}

void OBD_read(void)
{
  int a=0;
  char c;
  do{
    a++;
   if(Serial2.available() > 0)
    {
     c = Serial2.read();
     if((c!= '>') && (c!='\r') && (c!='\n')) //Keep these out of our buffer
      {
       rxData[rxIndex++] = c; //Add whatever we receive to the buffer
      }
    }
  }while((c != '>') && (a<100)); //The ELM327 ends its response with this char so
when we get it we exit out.
  rxData[rxIndex++] = '\0';//Converts the array into a string
  rxIndex=0; //Set this to 0 so next time we call the read we get a "clean buffer"
}

void getValues()
{
  //Save time for this dataset!
  Time1 = now();

  //Ask for current RPM
 Serial.println("010C");
 OBD_read();
 rpm1 = ((strtol(&rxData[6],0,16)*256)+strtol(&rxData[9],0,16))/4;

  //Ask for current speed
 Serial.println("010D");
 OBD_read();
 speed1 = strtol(&rxData[6],0,16);
```

```
 //Ask for current engine load
Serial.println("0104");
 OBD_read();
 load1 = strtol(&rxData[6],0,16);

 //Ask for current throttle position
Serial.println("0111");
 OBD_read();
 pos1 = (strtol(&rxData[6],0,16))*(100/255);

 //Ask for current coolant temperature
Serial.println("0105");
 OBD_read();
 temp1 = (strtol(&rxData[6],0,16))-40;

 accel.update(); //Update ADXL335 values
 x = accel.getX(); //Get X-axis values
 y = accel.getY(); //Get Y-axis values
 z = accel.getZ(); //Get Z-axis values


void send_data()

 //Empty obdString!
 obdString = "";

 //Compile string & calculate string lenght!
 obdString += ",\"/?Time1=";
 obdString += Time1;
 obdString += "&Rpm=";
 obdString += rpm1;
 obdString += "&Speed=";
 obdString += speed1;
 obdString += "&Load=";
 obdString += load1;
 obdString += "&Tpos=";
 obdString += pos1;
 obdString += "&Temp=";
 obdString += temp1;
 obdString += "&X=";
 obdString += x;
 obdString += "&Y=";
 obdString += y;
 obdString += "&Z";
 obdString += z;
 obdString += "\"\n";
```

```
  lenght1 = obdString.length();
  String socket1String = String("\"AT+USOWR=0,");

  //Complete string!!
  send1String += socket1String;
  send1String += lenght1;
  send1String += obdString;

  const char *cstr = send1String.c_str();
 SerialSystem.write(cstr);
}

TimedAction getValuesAction = TimedAction(1000,getValues);
TimedAction send_dataAction = TimedAction(1000,send_data);

void loop()
{
 getValuesAction.check();
 send_dataAction.check();
}
```