LAPPEENRANNAN TEKNILLINEN YLIOPISTO
LUT School of Energy Systems
LUT Kone

# ARDUINO-POHJAISEN MOOTTORINOHJAUSJÄRJESTELMÄN TOIMINTA

# OPERATION OF ARDUINO BASED ENGINE CONTROL UNIT

Lappeenrannassa 2.11.2018
Iikka Martikainen
Tarkastaja: TkT Heikki Handroos
Ohjaaja: DI Teemu Priha

**ALKUSANAT**

Tahdon kiittää kandidaatintyöni ohjaajaa diplomi-insinööri Teemu Prihaa neuvoista ja tuesta työn loppuun saattamisessa.

*Iikka Martikainen*

Iikka Martikainen

Lappeenrannassa 2.11.2018

**TIIVISTELMÄ**

Lappeenrannan teknillinen yliopisto
LUT Energiajärjestelmät
LUT Kone

Iikka Martikainen

**ARDUINO-POHJAISEN MOOTTORINOHJAUSJÄRJESTELMÄN TOIMINTA**

Kandidaatintyö
2018

55 sivua, 24 kuvaa, 2 taulukkoa ja 3 liitettä

Tarkastaja: TkT Heikki Handroos
Ohjaaja: DI Teemu Priha

Hakusanat: moottorinohjainjärjestelmä, Arduino -mikro-ohjain

Tämän opinnäytetyön tavoitteena on selvittää Arduino -mikro-ohjaimeen perustuvan moottorinohjausjärjestelmän toimintaperiaate. Moottorinohjausjärjestelmien toimintaan yleisesti perehdytään kirjalliskatsauksen keinoin ja Arduino-pohjaisen moottorinohjausjärjestelmän toimintaa tutkitaan elektroniikkakaavioita ja ohjelmiston koodia hyväksikäyttäen. Lisäksi moottorinohjaimen toimintaa tutkitaan käytännössä asentamalla se moottoripyörään.

Moottorinohjainjärjestelmän toiminnan todetaan olevan melko suoraviivaista ja moottorinohjausjärjestelmän olevan edullinen ja harrastelijalle kiinnostava vaihtoehto. Avoimen lähdekoodin vuoksi Arduino-pohjainen moottorinohjausjärjestelmä mahdollistaa järjestelmän jatkokehittelyn esimerkiksi autourheilun jokamiesluokan käyttöön.

## ABSTRACT

Lappeenranta University of Technology
LUT School of Energy Systems
LUT Mechanical Engineering

Iikka Martikainen

**OPERATION OF ARDUINO BASED ENGINE CONTROL UNIT**

Bachelor's thesis

2018

55 pages, 24 pictures, 2 tables and 3 appendices

Examiner: D.Sc. (Tech.) Heikki Handroos
Supervisor: M.Sc. (Tech.) Teemu Priha

Keywords: Engine Control Unit (ECU), Arduino micro controller

The aim of this paper is to figure out the working principle of an engine control unit (ECU) that is based on an Arduino micro controller. Basics of ECU operation are studied from literature and working principle of Arduino-based ECU is studied from electrical diagrams and program code related to the mentioned ECU. In addition, the operation of Arduino-based ECU is experienced in practice by installing the ECU on a motorcycle.

Operating principle of the ECU is found to be relatively simple and the ECU is affordable and interesting alternative for an enthusiast. Being open source project, the ECU is good subject for further development, for example to be used in low budget racing series.

**ALKUSANAT**
**TIIVISTELMÄ**
**SISÄLLYSLUETTELO**
**SYMBOLILUETTELO**


1 JOHDANTO ..... 8

2 MENETELMÄT ..... 10

3 ISKUMÄNTÄMOOTTORIN TOIMINTA ..... 11

4 ELEKTRONIIKAN PERUSKÄSITTEITÄ ..... 14

5 MOOTTORINOHJAUKSEN TOIMINTA ..... 16

5.1 Moottorin käyntitilan anturointi ..... 16

5.2 Polttoainesuihkutus ..... 19

5.3 Elektroninen sytytys ..... 20

5.4 Moottorinohjausyksikkö ..... 21

6 SPEEDUINO-MOOTTORINOHJAUSJÄRJESTELMÄ ..... 23

6.1 Speeduino-moottorinohjausjärjestelmän komponentit ..... 23

6.1.1 Arduino Mega 2560 ..... 23

6.1.2 Speeduino V0.3 -piirilevy ..... 24

6.1.3 Induktiivisen pyörimisnopeusanturin signaalinkäsittelypiiri ..... 25

6.2 Speeduino-moottorinohjausjärjestelmän toiminta ..... 26

6.2.1 Speeduino-ohjelmiston suorittamat toiminnot ..... 26

6.2.2 Signaalinkäsittely Speeduino V0.3 -piirilevyllä ..... 28

7 SPEEDUINO-MOOTTORINOHJAUKSEN ASENTAMINEN ..... 35

7.1 Speeduino V0.3.7 -piirilevyn kokoaminen ..... 35

7.2 Moottoripyörään tarvittavat muutokset ..... 36

7.2.1 Polttoainejärjestelmään tehdyt muutokset ..... 36

7.2.2 Sytytykseen tehdyt muutokset ..... 42

7.2.3 Anturointiin tehdyt muutokset ..... 43
7.2.4 Johtosarjaan tehdyt muutokset ..... 44
7.3 Speeduino-ohjelmiston asennus ja moottorin arvojen asettaminen ..... 46
7.4 Kokemuksia Speeduino-moottorinohjauksesta ja sen asentamisesta ..... 50
8 JOHTOPÄÄTÖKSET ..... 52
LIITTEET
LIITE I: Otteita Speeduino-ohjelmiston koodista.
LIITE II: Projektipyörään lisätyn johtosarjan yksinkertaistettu kytkentäkaavio
LIITE III: Muutostöihin hankitut komponentit hintoineen.

**SYMBOLILUETTELO**

| | |
|---|---|
| CDI | Capacitor Discharge Ignition, kapasitiivinen sytytysjärjestelmä |
| CLT | Coolant Temperature, Jäähdytysnesteen lämpötila |
| ECU | Engine Control Unit, Moottorinohjausjärjestelmä |
| IAT | Intake Air Temperature, imuilman lämpötila |
| LED | Light-Emitting Diode, hohtodiodi |
| MAP | Manifold Air Pressure, ilmanpaine imusarjassa |
| NTC | Negative Temperature Coefficient |
| PTC | Positive Temperature Coefficient |
| RC-piiri | Resistance Capacity -piiri. |
| RAM | Random Access Memory |
| ROM | Read Only Memory |
| TPS | Throttle Position Sensor |

# 1 JOHDANTO

Suomalaiset matkustavat selkeän enemmistön matkoistaan henkilöautoilla (Liikennevirasto, 2018, s. 8), joten autoilulla on suuri vaikutus liikkumisen kokonaisympäristövaikutuksiin. Liikennevälineiden ympäristövaikutuksia pyritään vähentämään tulevaisuudessa sähkö-, hybridi- ja vaihtoehtoisia polttoaineita käyttävillä ajoneuvoilla (Genta et al, 2014, s. 516). Liikenteessä on kuitenkin edelleen runsaasti perinteisiä polttoaineita käyttäviä ajoneuvoja sekä harrastajia, joita kiinnostaa ajoneuvojensa ominaisuuksien ja rakenteen muokkaaminen alkuperäisestä poikkeavaksi. Jälkiasennettavan moottorinohjainyksikön avulla voidaan tavallisesta bensiinikäyttöisestä ajoneuvosta tehdä flexfuel-ajoneuvo, jolloin voidaan käyttää E85-etanolipolttoainetta.

Erilaisia vaihtoehtoja moottorinohjauksen päivitykseen vertaillaan Heikki Kososen opinnäytteessä ”Moottorinohjauksen valinta” (2011). Moottorinohjauksen tuomia suorituskykyyn liittyviä etuja käsitellään Paavo Töytärin kandidaatintyössä ”Ottomoottorin suorituskyvyn parantaminen sähköisillä järjestelmillä autourheilun jokamiesluokassa” (2015) ja erään moottorinohjaimen asentamista Markus Kakon insinöörityössä ”Megasquirt-moottorinohjaimen rakennus ja asennus” (2013). Speeduino-moottorinohjauksen rakennetta tai toimintaa ei käsitellä edellä mainituissa opinnäytetöissä lainkaan.

Speeduino-moottorinohjausjärjestelmä on täysin avoimen lähdekoodin moottorinohjausjärjestelmä, joka rakentuu Arduino mikro-ohjaimen ympärille (Speeduino). Avoin lähdekoodi ja esimerkiksi Megasquirt-järjestelmää edullisempi hinta tekevät Speeduino-moottorinohjauksesta houkuttelevan vaihtoehdon. Taulukossa 1 on vertailtu Speeduino- ja Megasquirt-moottorinohjainjärjestelmien hintoja. Hinnat eivät ole suoraan vertailukelpoisia, mutta molempien mallistoista on pyritty valitsemaan mahdollisimman hyvin toisiaan vastaavat mallit ja tärkeimmät eroavaisuudet ominaisuuksissa on merkitty vertailuun.

*Taulukko 1. Speeduino- ja Megasquirt-moottorinohjausjärjestelmien hintavertailua*

| | Speeduino | Megasquirt |
|---|---|---|
| | Käyttövalmiit moottorinohjaimet | |
| versio | MX5 Plug n Play | MX5 plug-n-play |
| Ostopaikka | Speeduino store | Autotune.fi |
| hinta (9.10.2018) | 240$ (n. 210€) | 889 € |
| puutteet | Ei nakutuksen tunnistusta<br>Ei ilmastoinnin ohjausta | |
| | edullisimmat moottorinohjaimet | |
| versio | v0.4 Assembled | MS1 v2.2 rakennussarja |
| Ostopaikka | Speeduino store | Autotune.fi |
| hinta (9.10.2018) | 160$ (n. 140€) | 273 € |
| puutteet | Ei koteloa | Vain 2 kanavaa polttoainesuuttimille |

Kandidaatintyöni tavoitteena oli selvittää Arduino mikro-ohjaimeen pohjautuvan moottorinohjauksen toimintaa. Selvitin, mitä moottorinohjaus tarvitsee toimiakseen, kuinka Speeduino-moottorinohjausjärjestelmä toimii, kuinka se soveltuu harrastekäyttöön ja mitä muunnos kaasutinkäyttöisestä elektroniseen suihkutukseen vaatii. Käytännön kokemuksia moottorinohjauksesta saatiin asentamalla Speeduino-moottorinohjainjärjestelmä oheislaitteineen moottoripyörään, jossa oli alun perin nelisylinterinen kaasutintoiminen nelitahtimoottori yksinkertaisella elektronisella sytytyksellä. Koska iskumäntämoottoreita on useita eri tyyppejä, rajasin aihettani koskettamaan ainoastaan vapaasti hengittäviä nelitahtisia nelisylinterisiä rivimoottoreita monipistesuihkutuksella ja hukkakipinäsytytyksellä. Speeduino-moottorinohjaimen asennuksen jälkeen moottoripyörän moottori vastaa edellä mainittua kuvausta ja kyseistä moottorityyppiä esiintyy runsaasti myös henkilöautoissa.

# 2 MENETELMÄT

Moottorinohjausjärjestelmien toimintaan tutkittiin kirjallisuuskatsauksena merkittyihin lähteisiin perustuen. Lähteet on etsitty LUT-finna -palvelulla joko fyysisinä teoksina tai verkkolähteinä. Suoraan Speeduino-moottorinohjausjärjestelmään liittyvissä asioissa on käytetty lähteenä Speeduino-projektin verkkosivuja, sillä tieteellisiä lähteitä kyseisestä moottorinohjausjärjestelmästä ei löydetty.

Speeduino-moottorinohjaimen toimintaan perehdyttiin avoimessa jaossa olevien ohjelmistokoodin sekä elektroniikkakaavion perusteella. Speeduino-ohjelmiston toimintaa selvitettiin pääasiassa ohjelmakoodiin kirjoitettujen kommenttien perusteella. Signaalinkäsittelyä Speeduino-piirilevyllä tutkittiin elektroniikkakaaviota tulkitsemalla, Moottorialan sähköoppi (Juhala et al, 2005) -kirjaa hyödyntäen.

Käytännön kokemuksia Speeduino-moottorinohjausjärjestelmästä haettiin asentamalla moottorinohjausjärjestelmä moottoripyörään. Asennusprosessi on kuvattu mahdollisimman tarkasti myöhemmin opinnäytetyössä. Kaikki asennusta varten tehdyt hankinnat dokumentoitiin ja taulukoitiin todellisten kustannusten laskemiseksi.

# 3 ISKUMÄNTÄMOOTTORIN TOIMINTA

Moottorit ovat energianmuuntimia, joiden tehtävä on muuttaa esimerkiksi sähkö- tai kemiallista energiaa mekaaniseksi energiaksi, useimmiten pyörimisliikkeeksi. Nimensä mukaisesti polttomoottorissa kemiallinen energia muutetaan palamisen kautta mekaaniseksi energiaksi. Vaikka sähkömoottorien käyttö on yleistymässä, polttomoottori on vielä toistaiseksi tavallisin auton voimalähde. (Nieminen, 2005, s. 60) Polttomoottoreita on olemassa useita eri tyyppejä, mutta tässä opinnäytetyössä keskitytään nelitahtisiin iskumäntämoottoreihin kipinäsytytyksellä.

Iskumäntämoottorin toiminta perustuu ilma-polttoaineseoksen syklittäiseen palamiseen sylinterissä, mikä pakottaa männän edestakaiseen liikkeeseen. Kiertokangen ja kampiakselin avulla edestakainen liike muutetaan pyöriväksi liikkeeksi. Nelitahtimoottorin toiminta on jaettu neljään eri vaiheeseen: imu-, puristus-, työ- ja poistotahtiin. Kukin tahti on puolikkaan kampiakselin pyörähdyksen mittainen, jolloin yksi työkierto tapahtuu kahden täyden kierroksen aikana. Nelitahtimoottorissa kaasujen vaihtoa hallitaan imu- ja pakoventtiileillä. (Reif, 2015, s. 8–9)

Tehokas palaminen sylinterissä vaatii polttoaineen ja ilman syöttämistä tarkalla seossuhteella, koska liian vähän polttoainetta sisältävä seos ei syty ja liian paljon polttoainetta aiheuttaa ylimääräisiä saasteita ja polttoaineenkulutusta (Reif, 2015, s. 10). Täten ottomoottorin toiminta on parhaillaan tietyllä optimaalisella ilma-polttoaineseossuhdealueella, mutta moottoreita käytetään laajalla pyörimisnopeus- ja kuormitusalueella, mikä asettaa haasteita ilma-polttoaineseoksen muodostavalle laitteistolle (Pitkänen, 2000, s. 5). Pitkään käytössä olleet kaasuttimet poistuivat ajoneuvokäytöstä, koska niillä ei pystytä riittävän tarkasti huomioimaan käyntiolosuhteita eikä siten saada aikaan tarkkaa seossuhdetta. Polttoaineen suihkutus alkoi yleistyä viime vuosituhannen loppupuolella, kun liikenteen päästöihin alettiin kiinnittää huomiota. Polttoainesuihkutuksen tarkemmat säätömahdollisuudet sallivat esimerkiksi katalysaattorien käytön. (Nieminen, 2005, s. 114–115)

Ottomoottorin toiminta perustuu polttoaineen sytyttävään kipinään, joka käynnistää palamisen sylinterissä. Moottorin toiminnan takaamiseksi sytytysjärjestelmän on kyettävä tuottavaan riittävästi korkeajännitteistä energiaa sytytystulpan kipinää varten ja kipinä on laukaistava oikea-aikaisesti. Aluksi käytettiin magneettosytytystä, mutta tekniikan kehittyessä sytytykseen käytettyjä mekaanisia ratkaisuja korvattiin elektronisilla vastineilla. Induktiivisessa sytytysjärjestelmässä sytytyspuolan tehtävänä on varastoida kipinän muodostamiseen tarvittava sähköenergia ja muuttaa akkujännite riittävän korkeaksi jännitteeksi kipinän muodostamiseksi. (Reif, 2015, s. 136) Toisin kuin induktiivisessa sytytysjärjestelmässä, pienissä moottoreissa yleisesti käytetty kapasitiivinen sytytysjärjestelmä (Capacitive Discharge Ignition, CDI) varastoi kipinän muodostamiseen tarvittavan energian kondensaattoriin, josta se sytytyshetkellä puretaan sytytyspuolan läpi sytytystulpalle. Sytytyspuolaa käytetään siis muuntajana nostamaan kondensaattorilta purkautuva jännite riittävän korkeaksi kipinän muodostamiseen. Kondensaattoria ladataan normaalia käyttöjännitettä korkeammalla jännitteellä ja purkautumista ohjataan sähköisellä signaalilla. (STMicroelectronics, 2004, s. 1-2) Kapasitiivisen sytytysjärjestelmän etuihin induktiivisiin järjestelmiin verrattuna kuuluu lyhyempi latautumisaika, joka mahdollistaa tehokkaan kipinän myös korkeilla kierroksilla. Toisaalta kipinän kesto on lyhyempi, mikä hankaloittaa polttoaineseoksen sytyttämistä alhaisella kuormalla. (Agarwal, 2018)

Korkeajännitteinen sytytysenergia jaetaan oikean sylinterin sytytystulpalle joko pyörivällä virranjakajalla tai elektronisilla ratkaisuilla, joissa käytetään useampia sytytyspuolia. Tulpanjohdot nimensä mukaisesti johtavat korkeajännitteisen energian sytytystulpille. Tulpanjohdot on suunniteltu kestämään korkeaa jännitettä, mutta niissä tapahtuvien häviöiden vuoksi johtimien pituus on syytä rajoittaa minimiin. Sytytystulppa puolestaan synnyttää kipinän elektrodiensa välissä ja käynnistää polttoaineseoksen palamisen sylinterissä. (Reif, 2015, s. 159–162, 178)

Koska sytytyshetkestä seoksen täydelliseen palamiseen kuluu tietty aika, sytytyksen ajankohta vaikuttaa suuresti sylinterissä muodostuvaan palamispaineeseen ja siten moottorin tehoon. Suurin teho moottorista saadaan, kun palamispaine on suurimmillaan männän juuri ohitettua ylimmän kohtansa. Liian myöhäisellä sytytyksellä teho kärsii ja liian aikainen sytytys aiheuttaa nakutusta. Nakutus on tilanne, jossa palaminen sylinterissä alkaa liian aikaisin, palaminen on hallitsematonta ja palotilan paine vaihtelee äkillisesti. Nakutus on

moottorille haitallista, sillä moottorin komponentteja ei ole suunniteltu kestämään nakutuksen aiheuttamia paineiskuja. Sytytysajankohtaa ennen männän yläasentoa kutsutaan sytytysennakoksi ja sitä kuvataan usein sytytyshetkeä vastaavan kampiakselikulman ja männän yläasennon kampiakselikulman erotuksena. Moottorin pyörintänopeus ja kuormitus vaikuttavat optimaaliseen sytytysajankohtaan. Erilaiset kuormitustilanteet vaikuttavat sylinterissä vallitsevaan paineeseen, joka puolestaan vaikuttaa palamisen nopeuteen. Paineen nousu kasvattaa palamisen nopeutta, eli vastaavasti sytytysajankohtaa tulee siirtää myöhemmäksi. Moottorin pyörimisnopeuden kasvaessa mäntä saavuttaa ja ohittaa ylimmän asentonsa nopeammin, joten pyörimisnopeuden kasvaessa sytytyksen ennakkoa tulee kasvattaa, jotta palotilan paine on suurimmillaan optimaalisella hetkellä. (Nieminen, 2007, s. 203–212)

# 4 ELEKTRONIIKAN PERUSKÄSITTEITÄ

Resistanssi kuvaa aineen ominaisuutta vastustaa sähkövirran kulkua. Resistanssin yksikkönä käytetään ohmia (Ω). Resistanssi riippuu sähkövirtaa johtavan kappaleen pituudesta, poikkipinta-alasta ja resistiivisyydestä. Resistiivisyys on kullekin aineelle ominainen, lämpötilasta riippuva suure. Riippuen aineesta lämpötilan kasvattaminen voi kasvattaa tai vähentää aineen resistiivisyyttä. (Juhala et al, 2005, s. 14–15) Vastuksiksi kutsutaan elektroniikan rakenneosia, joilla rajoitetaan sähkövirran kulkua. Vastuksen resistanssi voi olla kiinteä, voimakkaasti riippuvainen lämpötilasta tai valosta. Lisäksi valmistetaan potentiometrejä, joissa sähkövirran kulkema matka ja siten vastuksen resistanssi vaihtelevat potentiometrin säätimen asennon mukaan. (Juhala et al, 2005, s. 48–49)

Kondensaattori toimii virtapiirissä lyhytaikaisena varastona sähköenergialle. Kytkettäessä jännitelähteeseen, varautuu kondensaattori jännitelähteen kanssa samaan jännitteeseen. Kun kondensaattori irrotetaan jännitelähteestä, pysyy se samassa jännitteessä, kunnes varaus purkautuu joko kondensaattorin napojen yhdistyttyä tai hitaasti sähkövuotojen myötä. Kondensaattorin kykyä varata itseensä sähköä ilmoitetaan yksikössä faradi (F). (Juhala et al, 2005, s. 53–55)

Yhdistämällä vastuksen ja kondensaattorin sarjaan, saadaan Resistance Capacity (RC) -piiri. Napojen välisen jännitteen muuttuessa, pyrkii kondensaattori latautumaan ja purkautumaan saavuttaakseen saman jännitteen. Latautumisen ja purkautumisen sähkövirrat kulkevat vastuksen kautta, jossa tapahtuu jännitehäviötä, mikä alentaa kondensaattorin napojen välistä jännite-eroa ja hidastaa kondensaattorin varautumista. (Juhala et al, 2005, s. 56–58)

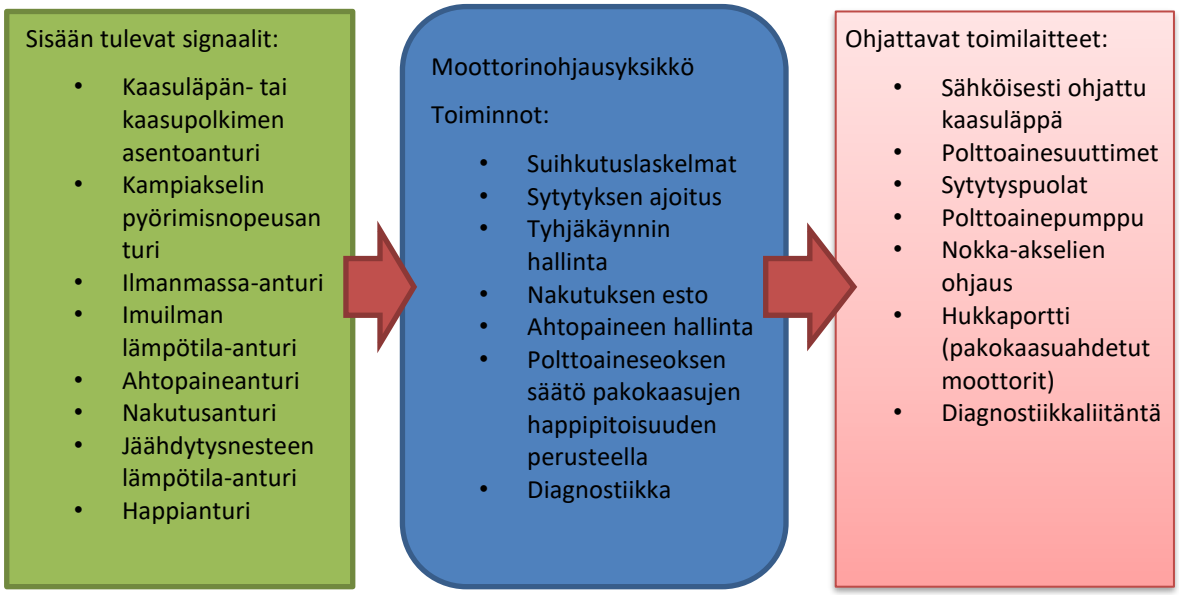
Diodit ovat puolijohdekomponentteja, joita voidaan hyödyntää esimerkiksi tasasuuntaamiseen, valon tuottamiseen tai jännitteen vakaimena. Tasasuuntausdiodi päästää sähkövirtaa lävitseen vain yhteen suuntaan, joten sillä voidaan estää tasavirran läpipääsy tiettyyn suuntaan. Suuntaa, jossa diodi sallii virran, kutsutaan päästösuunnaksi ja suuntaa, jossa diodi estää virran, estosuunnaksi. Läpilyöntidiodit, joita myös zenerdiodeiksi kutsutaan, toimivat päästösuunnassa kuten tasasuuntausdiodi. Estosuunnassa läpilyöntidiodit muuttuvat tietyllä jännitteellä johtavaksi ja tätä jännitettä kutsutaan

zenerjännitteeksi. Jännitteen laskiessa zenerjännitteen alapuolelle estää läpilyöntidiodi jälleen virran kulun. Light Emitting Diode (LED) eli hohtodiodi on kuin tasasuuntausdiodi, mutta kytkettynä päästösuuntaan se säteilee käytetylle puolijohdemateriaalille ominaista valon aallonpituutta. Aallonpituus voi olla näkyvän valon tai infrapunavalon alueella. (Juhala et al, 2005, s. 58–64)

Transistori on puolijohde, jolla on kolme napaa: emitteri (E), kanta (B) ja kollektori (C). Transistoreita voidaan käyttää esimerkiksi kytkiminä, jolloin kantaan johtamalla pieni virta saadaan emitteriltä kollektorille kulkemaan suurempi virta ja vastaavasti kantaan pienen jännitteen kytkemällä saadaan kytkettyä suurempi jännite emitteriltä kollektorille. Transistoreilla on haastavaa saada aikaan täydellisesti suljettua kytkintä, sillä vaikka kanta olisi jännitteetön vuotaa transistori hieman sähköä emitteriltä kollektorille. Lisäksi transistoria voidaan käyttää esimerkiksi vahvistimena tai jännitteenvakaimena. (Juhala et al, 2005, s. 66–75)

# 5 MOOTTORINOHJAUKSEN TOIMINTA

Iskumäntämoottorin toimintaa ohjataan kontrolloimalla seoksenmuodostusta, kaasujen ohjausta ja sytytystä. Aikaisemmin ohjausta tehtiin mekaanisilla ratkaisuilla. Esimerkiksi sytytysennakkoa säädettiin keskipako- ja alipainesäätimellä virranjakajassa. (Nieminen, 2007, s. 203) Nykyiset moottorinohjausjärjestelmät tulkitsevat antureilta saamiaan tietoja ja tietojen perusteella valitsevat tilanteeseen sopivat ohjausarvot, jotka lähetetään edelleen moottorin käyntiä ohjaaville toimilaitteille. Moderneissa järjestelmissä käynninohjauksessa voidaan huomioida moottorin käynnin lisäksi myös ympäristön ja auton muiden järjestelmien tila. (Nieminen, 2007, s. 189) Kuvassa 1 on esitelty Bosch Motronic -moottorinohjausjärjestelmän käyttämiä anturi- ja toimilaitetyyppejä.

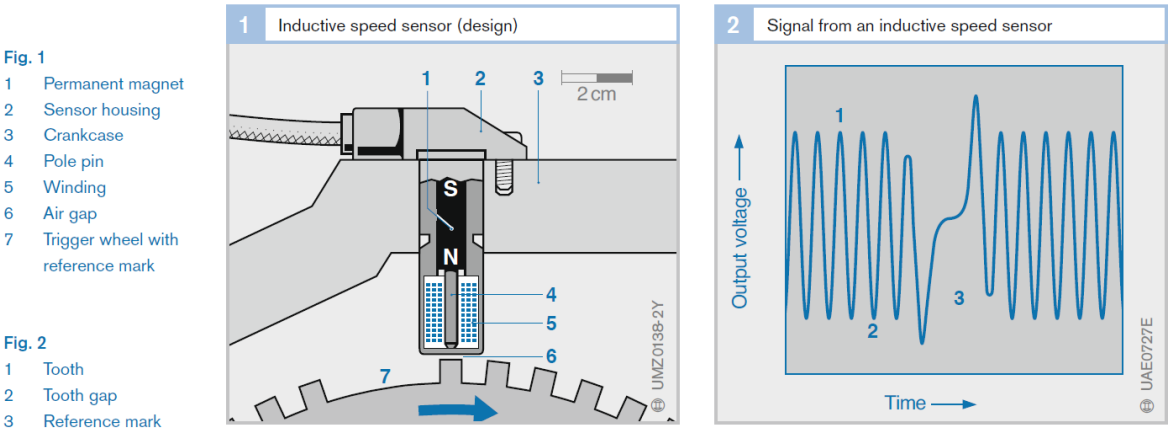



**Kuva 1.** Esimerkki moottorinohjauksen käyttämistä komponenteista. (Mukaillen: Reif, 2015, s. 213).

## 5.1 Moottorin käyntitilan anturointi

Toimiakseen oikein moottorinohjauksen tulee saada tietoa kampiakselin pyörimisnopeudesta sekä sen asennosta. Tiedon keräämiseen käytetään yleisesti vauhtipyörälle asennettua metallista hammaskehää, jossa on tietyssä kohdassa yhden tai useamman hampaan levyinen aukko. Induktiivinen-, Hall- tai optinen anturi antaa jännitesignaalin hampaan ohittaessa anturin. Saamastaan jännitedatasta

moottorinohjaustietokone laskee signaalin tiheyden perusteella pyörimisnopeuden ja signaaleissa olevan aukon perusteella kampiakselin asennon. (Nieminen, 2007, s. 209) Kuvassa 2 induktiivisen pyörimisnopeusanturin rakenne, sekä kuvaaja sen tuottamasta jännitteestä




**Kuva 2.** Induktiivisen pyörimisnopeusanturin rakenne, sekä sen tuottama signaali (Reif, 2015, s. 236).

Nelitahtimoottorissa ei voida pelkän kampiakselin asennon perusteella kertoa, onko tietty sylinteri puristus- vai poistotahdissa. Jotta voitaisiin käyttää erillisiä sytytyspuolia ja sekventiaalista polttoaineen suihkutusta, täytyy moottorinohjaustietokoneen tietää tarkalleen kunkin sylinterin tahti. Ongelma voidaan ratkaista asentamalla nokka-akselille edellä kuvatun kaltainen asentoanturi. Nokka-akseli pyörähtää yhden kierroksen kampiakselin pyörähtäessä kaksi kierrosta. Tällöin nokka- ja kampiakselin asema-antureiden tiedon perusteella voidaan kertoa tarkalleen missä vaiheessa kukin sylinteri on. (Reif, 2015, s. 238).

Koska lämpötila vaikuttaa ilman tiheyteen, on moottorin ja imuilman lämpötiloilla suuri merkitys oikean ilma-polttoaineseossuhteen löytämiselle, joten lämpötilat mitataan erillisillä antureilla ja viestitään moottorinohjaustietokoneelle (Nieminen, 2007, s. 210). Anturin toiminta perustuu vastuselementtiin, jonka resistanssi riippuu lämpötilasta. Vastuksia on kahta tyyppiä: NTC (Negative Temperature Coefficient) ja PTC (Positive Temperature Coefficient). Yleisemmän NTC tyyppisen vastuksen resistanssi tippuu lämpötilan noustessa ja PTC tyyppisen vastuksen resistanssi vastaavasti nousee. Vastuselementti kytketään osaksi mittausvirtapiiriä, jonka avulla moottorinohjaus voi laskea sensorin vastuksen ja tulkita täten sensorin kokeman lämpötilan. Vastus on asennettu suojaavan kuoren sisään ja vastusten

ulkomuoto vaihtelee runsaasti käyttökohteen mukaan. Imuilman ja moottorin lämpötilan lisäksi voidaan tarpeen mukaan erillisillä antureilla mitata myös moottoriöljyn tai polttoaineen lämpötilaa. (Reif, 2015, s. 235)

Jotta moottoriin saadaan tarkasti syötettyä juuri oikea määrä polttoainetta, täytyy pyörimisnopeuden ja lämpötilojen lisäksi tietää moottorin kuormitus, jota kuvaa hyvin moottoriin kulkeva ilmamäärä. Ilmamäärää voidaan arvioida useilla eri keinoilla. Vanhemmissa järjestelmissä käytettiin imukanavaan sijoitettua jousikuormitettua läppää, joka avautui ilmavirran vaikutuksesta. Läppä oli kytketty pontentiometriin, jonka resistanssi on riippuvainen läpän asennosta. Näin ollen resistanssi riippuu moottoriin virtaavan ilman tilavuusvirrasta. Ilmamäärää voidaan arvioida myös mittaamalla imusarjan painetta tai ilmamassa-anturilla. (Nieminen, 2005, s. 115–117) Imusarjan painetta mitataan erityisillä paineantureilla. Paineanturin ohuen kalvon toisella puolella on tyhjiö ja sen toiselle puolelle vaikuttava ilmanpaine saa kalvon taipumaan, aiheuttaen kalvoon asennettujen vastuselementtien resistanssin muutoksen. Resistanssin muutoksen avulla voidaan laskea paine imusarjassa. Ilmamäärämittarin toiminta perustuu ilmavirran jäähdyttävään ominaisuuteen. Osa imukanavan ilmavirrasta ohjataan kulkemaan lämpövastuksen ja lämpösensitiivisten vastusten ohi. Lämpötilan ja sitä kautta lämpösensitiivisten vastusten resistanssin muutoksen perusteella voidaan laskea ilmamäärä. (Reif, 2015, s. 240, 244)

Kaasuläpän asennon tunnistamiseen käytetään pontentiometriä. Läpän kiertymäkulma vaikuttaa potentiometrin resistanssiin, josta tietokone tulkitsee kaasuläpän asennon. Tietoa käytetään muun muassa tyhjäkäynnin ja voimakkaan kiihdytyksen tunnistamiseen, jolloin seokseen voidaan lisätä hieman ylimääräistä polttoainetta tehon lisäämiseksi. (Nieminen, 2005, s. 119)

Ilma-polttoaineseoksen palamista ja pakokaasujen koostumusta seurataan happitunnistimilla (nk. Lambda-anturi). Anturi kertoo pakokaasujen happipitoisuudesta ja siten palamisen onnistumisesta. Tiedon avulla voidaan tehdä korjauksia polttoaineen syöttöön, jotta palaminen sylintereissä olisi mahdollisimman lähellä haluttua. (Nieminen, 2005, s. 120) Yksinkertaisten happitunnistimien jännite muuttuu merkittävästi ainoastaan optimaalista ilma-polttoaine –seosta vastaavan jäännöshappipitoisuuden ($\lambda=1$) alueella. Täten voidaan vain todeta, onko pitoisuus optimaalisen alueen ylä- vai alapuolella. Rakenteeltaan hieman

monimutkaisempi laajakaistahappianturi pystyy tuottamaan tarkempaa tietoa pakokaasun happipitoisuudesta, sillä sen tuottama jännite muuttuu epälineaarisesti jäännöshappipitoisuuden funktiona. (Reif, 2015, s. 248–253) Kuvassa 3 Bosch LSU4.9 laajakaistahappianturi.

**Kuva 3.** Bosch LSU4.9 laajakaistahappianturi

## 5.2 Polttoainesuihkutus

Polttoaineen suihkutus tarkoittaa polttonesteen sumuttamista imusarjaan tai suoraan sylinteriin. Järjestelmä mahdollistaa aikaisempia kaasutinratkaisuja paremman polttoaineen sekoittumisen, annostelun ja tarkemman säätelyn moottorin toimintaan. Ensimmäiset suihkutuslaitteet 50-luvulla muistuttivat dieselmoottorien ruiskutuslaitteistoja, mutta ympäristönsuojelun nostettua päätään 60- ja 70-luvuilla kehitys polttoaineen suihkutuksessa vauhdittui. Nykyisellään käytetään monipistesuihkutusta, jossa kullekin sylinterille on oma polttoainesuuttimensa ja palotapahtumaa voidaan kontrolloida yksilöllisesti. Käytetään myös suuripaineisia suorasuihkutusjärjestelmiä, joissa polttoaine sumutetaan suoraan sylinteriin. (Nieminen, 2005, s. 114–115) Tästä eteenpäin käsitellään vain monipistesuihkutusta imusarjaan.

Suuttimien suihkutus voidaan ajoittaa monella eri tavalla monipistesuihkutteisessa moottorissa (Nieminen, 2005, s. 114–115). Suihkutuksen ajoittuminen kampiakselin

kulmaan verrattuna on tärkeä tekijä polttoaineenkulutuksen ja pakokaasujen koostumuksen muodostumiseen. Suuttimia voidaan käyttää yhtäaikaisesti, jolloin riittävän hyvän ilma-polttoaine-seoksen aikaansaamiseksi polttoaineannos suihkutetaan kahdessa osassa: yksi osa kerran kampiakselin pyörähdyksen aikana. Tällä menetelmällä kaikilla sylintereillä jää eripituinen aika polttoaineen höyrystymiselle suihkutuksen ja imuventtiilin aukeamisen välissä. Ryhmäsuihkutuksessa suuttimet on jaettu ryhmiin, jotka suihkuttavat koko polttoaineannoksen kerralla, yhden kerran moottorin työkierron aikana. Suihkutusajankohta on hieman tarkemmin valittavissa, mutta edelleen polttoaineen höyrystymiseen käytettävissä oleva aika ei ole sama sylintereiden välillä. Sekventiaalisessa suihkutuksessa polttoaine suihkutetaan jokaiselle sylinterille erikseen ja täten suihkutus voidaan aloittaa jokaiselle sylinterille optimaalisella ajankohdalla. (Reif, 2015, s. 100–101)

Suihkutusmoottoreissa polttoainepumppu toimittaa polttoaineen suodattimen läpi polttonesteen jakoputkelle ja edelleen suuttimille. Jakoputken paine tulee pitää tasaisena ja tähän tarkoitukseen voidaan käyttää järjestelmäpaineventtiilillä, joka palauttaa osan jakoputkelle saapuvasta polttoaineesta säiliöön. (Nieminen, 2005, s. 124-125) Polttoaineen suihkutusjärjestelmä tarvitsee siis pumpun, joka kykenee tuottamaan suuremman tilavuusvirran, kuin kaasutintoiminen moottori tarvitsee (Nieminen, 2005, s. 102). Jakoputkesta säiliöön palaava polttoaine lämpenee moottorin läheisyydessä ja täten säiliössä olevan polttoaineen lämpötila nousee. Lämpötilan nousu aiheuttaa muun muassa polttoaineen höyrystymistä säiliössä. Ongelman lievittämiseksi järjestelmäpaineventtiili voidaan siirtää lähemmäksi polttoainesäiliötä, jolloin ylimääräinen polttoaine ei kierrä moottoritilan kautta. Paluukierto ja mekaanisen järjestelmäpaineventtiili voidaan myös korvata järjestelmällä, jossa moottorinohjaus tarkkailee jakoputkessa vallitsevaa painetta ja ohjaa polttoainepumppua tuottamaan vain tarvittavan paineen. Tällöin polttoainepumppu pumppaa vain tarvittavan määrän polttoainetta, eli pumppu voidaan mitoittaa pienemmäksi ja se kuluttaa vähemmän energiaa. Lisäksi haluttaessa polttoaineen painetta voidaan säädellä moottorin käyntitilan mukaan. Eli moottorin kuormitustilan ollessa suuri, voidaan painetta lisätä ja vastaavasti pienellä kuormalla painetta alentaa. (Reif, 2015, s. 76–77)

### 5.3 Elektroninen sytytys

Moottorinohjausjärjestelmän saamien anturitietojen ja elektroniikan avulla voidaan sytytys säätää tarkemmin moottorin tilaa vastaavaksi kuin aikaisemmilla mekaanisilla järjestelmillä.

Tämä johti parempaan käyntiin, matalampaan kulutukseen ja vähentyneisiin päästöihin. Aluksi elektronisissa järjestelmissä käytettiin virranjakajaa ohjaamaan kipinä oikean sylinterin sytytystulpalle, mutta myöhemmin virranjakajan tarve poistui kaksoiskipinä- ja yksittäiskipinäsytytyspuolien käyttöönoton myötä. (Nieminen, 2007, s. 208–209, 214)

Kaksoiskipinäsytytyspuolassa yhteen sytytyspuolaan on liitetty kaksi tulpanjohtoa saman käämin eri päihin ja näin ollen yhdestä puolasta saadaan kaksi kipinää yhtäaikaisesti. Kytkennät tehdään siten, että kipinät annetaan yhtäaikaisesti työtahdin alussa olevalle sekä niin sanottu ”hukkakipinä” poistotahdin lopussa olevalle sylinterille. Nelisylinterisessä moottorissa tarvitaan siis kaksi kaksoiskipinäsytytyspuolaa, jotka voivat olla rakennettu samaan koteloon. Yksittäiskipinäpuolassa jokaiselle sylinterille on oma sytytyspuola, joka on asennettu mahdollisimman lähelle sytytystulppaa, jopa suoraan sen päälle. Rakenne mahdollistaa sytytyksen säätämisen sylinterikohtaisesti ja poistaa häiriöherkät suurjännitejohtimet. Järjestelmä vaatii kuitenkin tiedon kunkin sylinterin tahdista ja siksi asema-anturin myös nokka-akselille. (Nieminen, 2007, s. 216–217)

## 5.4 Moottorinohjausyksikkö

Moottorinohjauksentietokoneen piirilevy ja tarvittavat elektroniikkakomponentit ovat asennettu suojaavan kotelon sisään. Sensorit ja toimilaitteet kiinnitetään moottorinohjaukseen monipinnisellä liittimellä tai useammalla liittimellä. Prosessointiyksikössä on yhdelle mikrosirulle rakennettu sisään tulevien ja ulos lähtevien signaalikanavien lisäksi muun muassa ajastimia sekä Random Access Memory (RAM) ja Read-Only Memory (ROM) -muisteja. Prosessointiyksikön muistiin on tallennettu tarvittavat algoritmit, joiden avulla moottorinohjauksen prosessointiyksikkö laskee antureilta saatujen signaalien perusteella toimilaitteille oikeat ohjausarvot. Mikrosirun ROM-muistille on tallennettu binäärimuodossa prosessointiyksikköä ohjaava koodi, jota prosessointiyksikkö suorittaa komento kerrallaan. RAM-muistiin tallennetaan hetkellisesti tarvittavia arvoja, sillä RAM-muisti tyhjenee, jos se ei saa sähkövirtaa. (Reif, 2015, s. 254–256)

Moottorinohjaustietokone saa sekä analogisia että digitaalisia signaaleita antureilta. Jotta moottorinohjauksen prosessointiyksikkö voi käsitellä analogisia signaaleita, tulee ne ensiksi muuntaa digitaalisiksi. Esimerkiksi nollan ja viiden voltin jännitteen välillä vaihtelevan

analogista signaalia voidaan tulkita noin tuhannen askelen tarkkuudella. Digitaalisia signaaleita moottorinohjauksen prosessointiyksikkö pystyy lukemaan ilman muunnoksia. Induktiivisten pyörimisnopeusanturien aaltomainen signaali muutetaan prosessointiyksikön luettavaksi suorakaideaaltosignaaliksi erillisellä piirillä moottorinohjausyksikössä. (Reif, 2015, s. 254–255)

# 6 SPEEDUINO-MOOTTORINOHJAUSJÄRJESTELMÄ

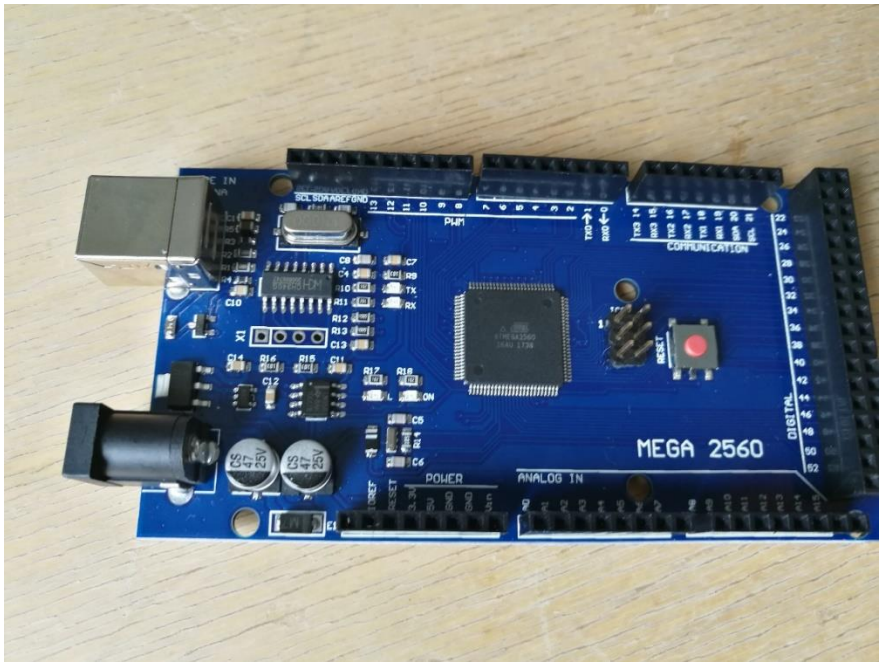
Speeduino on avoimen lähdekoodin projekti, jossa tavoitteena on luoda edullinen, helposti lähestyttävä laajoilla ominaisuuksilla varustettu moottorinohjausyksikkö. Moottorinohjausyksiköllä on mahdollista ohjata 1-4, 6- ja 8- sylinterisiä moottoreita ja ominaisuuksiin kuuluu muun muassa seoksen rikastaminen kiihdytyksessä, moottorin ollessa kylmä tai käynnistyksen jälkeen. Järjestelmällä voidaan myös esimerkiksi asettaa kierrosrajoitin, ohjata polttoainepumpun tai säätyvien nokka-akseleiden toimintaa ja tehdä ajoneuvosta polttoaineen koostumusta aistiva flexfuel-ajoneuvo. Lisäksi Tunerstudio-ohjelmistoa käyttämällä voidaan kerätä dataa moottorin toiminnasta ja happianturilta saatua tietoa hyväksikäyttämällä säätää ajoneuvon polttoainesuihkutuskarttoja. (Speeduino, overview, 2018)

## 6.1 Speeduino-moottorinohjausjärjestelmän komponentit

Speeduino-moottorinohjausyksikkö koostuu pääasiassa Arduino -mikro-ohjaimesta sekä tähän liitettävästä Speeduino-piirilevystä. Lisäominaisuuksia varten järjestelmään voidaan liittää muita komponentteja kuten Induktiivisen pyörimisnopeusanturin signaalinkäsittelypiiri induktiivisien pyörimisnopeusanturien käyttämistä varten. (Speeduino Manual, 2017, s. 11)

### 6.1.1 Arduino Mega 2560

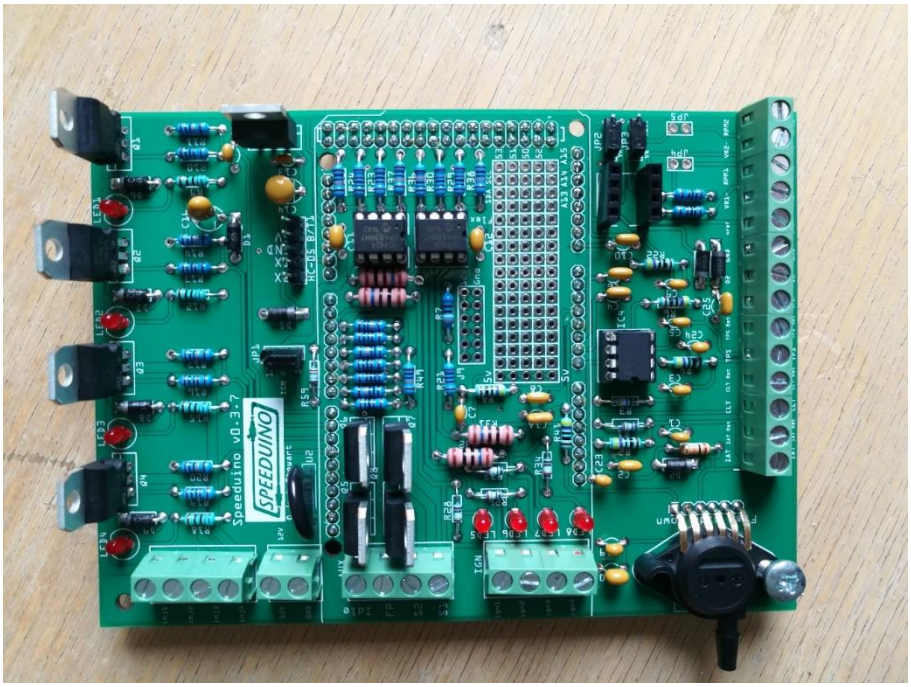
Arduino Mega 2560 on ATmega2560 mikrosiruun perustuva mikro-ohjain. Ohjaimessa on 54 digitaalista tulo- tai lähtöliitäntää, joista viidessätoista voidaan käyttää pulssinleveysmodulaatiota, ja lisäksi 16 analogista sisääntuloa. Ohjelmoitavaa muistia mikro-ohjaimessa on 256 kilobittiä. (Arduino, 2018) Mikro-ohjaimelle tallennetaan Speeduino-ohjelmisto, jonka komentoja mikro-ohjain toteuttaa. Kuvassa 4 projektissa käytetty kopio Arduino Mega 2560 mikro-ohjaimesta.




**Kuva 4.** Kopio Arduino Mega 2560 mikro-ohjaimesta

### 6.1.2 Speeduino V0.3 -piirilevy

Kuvassa 5 Speeduino V0.3.7 -piirilevy valmiiksi koottuna.




**Kuva 5.** Speeduino V0.3.7 -piirilevy elektroniikkakomponentit asennettuna.

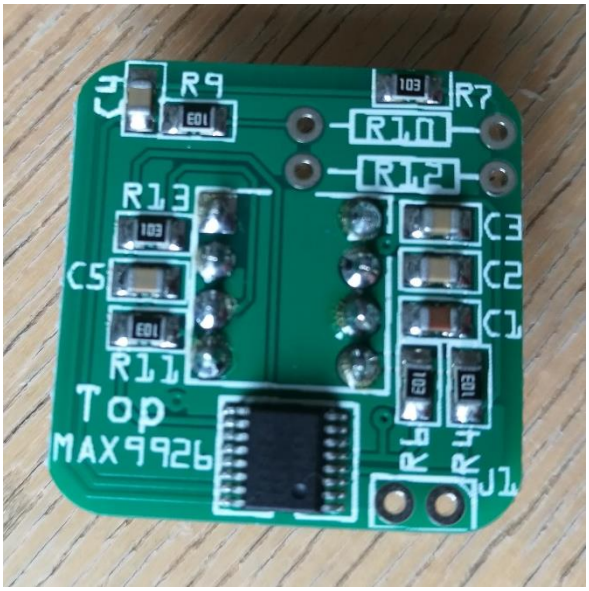
Piirilevyllä ovat seuraavat toiminnot (Speeduino Manual, 2017, s. 17–18)

- neljä kanavaa polttoainesuuttimille
- neljä kanavaa sytytyspuolille
- täysin suojatut sisääntulokanavat jäähdytysnesteen- ja imuilman lämpötila-antureille sekä kaasuläpän asentoanturille ja happianturille
- paikat imusarjan paineanturille ja pyörimisnopeuden VR-signaalinkäsittely-yksikölle

Piirilevy on moottorinohjainjärjestelmän kookkain komponentti ja muut komponentit liittyvät siihen. Kooltaan piirilevy on elektroniikkakomponenttien kera n. 14 x 10 x 3,5 cm.

### 6.1.3 Induktiivisen pyörimisnopeusanturin signaalinkäsittelypiiri

Induktiiviset pyörimisnopeusanturit tuottavat sinimuotoista signaalia, joka täytyy muuntaa suorakaideaalloksi, jotta se on moottorinohjauksen luettavissa. Signaalinkäsittelyyn Speeduino-moottorinohjaimen kanssa on tarkoitettu MAX9926 signaalinkäsittelijä, joka muuttaa jopa kahden induktiivisen anturin pyörimisnopeusdatan oikeaan muotoon. Tällöin voidaan käyttää induktiivista anturia sekä kampi- että nokka-akselilla. (Speeduino, VR conditioner, 2018). Kuvassa 6 MAX9926-signaalinkäsittelijä, jota käytettiin induktiivisten pyörimisnopeusanturien signaalien käsittelyyn.




**Kuva 6.** Induktiivisen pyörimisnopeusanturin signaalinkäsittelypiiri

## 6.2 Speeduino-moottorinohjausjärjestelmän toiminta

Tässä moottorinohjaimen toimintaa käsitellään mikro-ohjaimen suorittaman ohjelmiston, että mikro-ohjaimeen liitettävän Speeduino-piirilevyn suorittaman signaalinkäsittelyn kannalta. Tarkoituksena on antaa käsitys toiminnan perusteista.

### 6.2.1 Speeduino-ohjelmiston suorittamat toiminnot

Speeduino-moottorinohjausjärjestelmän ohjelmiston toiminta on selitetty Speeduino-projektin kotisivujen code overview -osiossa seuraavasti: Ohjelmiston toiminta on jaettu jatkuvasti suoritettavaan pääsilmukkaan sekä alifunktioihin, joita kutsutaan tarvittaessa. Pääsilmukka lukee analogiset signaalit ja tarkastaa onko moottori käynnissä. Lukemien tietojen perusteella ohjainyksikkö määrittelee olosuhteisiin sopivat arvot mm. polttoaineventtiilien aukioloajoille ja ajoittaa polttoaineen suihkutuksen sekä kunkin sylinterin sytytyksen. (Speeduino, Code overview)

Liitteeseen 1 on kerätty Speeduino-ohjelmiston koodista olennaisia kohtia, opinnäytetyön laajuus huomioiden. Liitteestä 1 tai itse alkuperäistä ohjelmistoa lukemalla voidaan huomata, että Speeduino-ohjelmiston pääsilmukassa kutsutaan aliohjelmia, jotka suorittavat eri sensoreilta saatavan signaalin tulkitsemisen. Riippuen anturista, signaalit luetaan esimerkiksi neljä tai 15 kertaa sekunnissa. Speeduino-moottorin ohjaimen kotisivujen Code overview kohdassa signaalien luennan taajuutta on perusteltu anturien hitaudella. Signaalit eivät muutu niin nopeasti, että niitä olisi perusteltua lukea tiheämmin. (Speeduino, Code overview) Liitteestä 1 voidaan lukea sensorien lukemiseen liittyvät toimenpiteet, mutta yksinkertaistettuna ohjelmisto lukee kyseiseen porttiin tulevan analogisignaalin digitaalisena, tallentaa tiedon välimuuttujaan ja mikro-ohjaimen muistiin tallennetun kalibrointitaulukon avulla muuttaa tiedon oikeaan muotoon tai yksikköön. Tarvittaessa melua ja virheellisiä signaaleja suodatetaan pois tarkistamalla, että mitattu arvo on sallituissa rajoissa.

Kun pyörimisnopeusanturi havaitsee hampaan, keskeyttää ohjelmisto pääsilmukan suorittamisen ja kutsuu trigger -alifunktiota. Riippuen asetuksissa ilmoitetusta ajoituskehän tyypistä alifunktio ohjautuu oikealle decoder-vaihtoehdolle. Tässä tapauksessa alifunktio triggerPri_missingTooth kutsutaan, jolloin se tallentaa välimuuttujaan sen hetkisen ajan. Nykyisen ja edellisen hampaan välinen aika lasketaan vähentämällä sen hetkisestä ajasta

edellisen hampaan havaitsemisen aika. Mikäli edellisen hampaan näkemisestä on liian lyhyt aika, tulkitaan signaali meluksi ja jätetään huomiotta. Muutoin tallennetaan hammasvälin aika talteen ja vertaillaan aikaa kahden edellisen hampaan välillä kuluneeseen aikaan. Hampaiden välisen ajan perusteella lasketaan kampiakselin hetkellinen pyörimisnopeus. Mikäli aika viimeisimpien hampaiden välillä on huomattavasti suurempi kuin edellinen (eli nykyisen ja edellisen hampaan välissä on aukko), tulkitsee ohjelma nykyisen hampaan olevan aukon jälkeen ensimmäinen hammas ja tallentaa nykyisen ajanhetken ajaksi, jolloin ensimmäinen hammas ohitti anturin. Mikäli ohjelma ei tulkitse hampaiden välissä olleen aukkoa, merkitään nykyinen hammas tavalliseksi hampaaksi ja tallennetaan hampaan ohittamisen aika.

Ohjelma myös laskee anturin ohittaneiden hampaiden lukumäärää ja nollaa luvun aina, kun se havaitsee aukon jälkeen ensimmäisen hampaan. Kun tiedetään anturin ohittaneiden ja hampaiden kokonaismäärä ajoituspyörällä voidaan myöhemmin laskea kampiakselin asento viimeisimmän hampaan ohitushetkellä. Asentotiedon tarkkuus riippuu kampiakselille sijoitettujen hampaiden lukumäärästä. Esimerkiksi, kun hampaita on 36, voidaan asento tietää ainoastaan tarkkuudella $\frac{360°}{36} = 10°$. Mikäli nokka-akselilla on pyörimisnopeusanturi, myös sen havaittua hampaan pääsilmukka keskeytyy. Riippuen nokka-akselille sijoitetusta hammaspyörästä ja asetusten valinnoista, ohjataan toiminta oikealle alifunktiolle, joka käsittelee ja tallettaa hampaan ohitukseen liittyvät tiedot.

Tämän jälkeen ohjelmisto laskee suihkutettavan polttoaineen määrän sekä sytytysennakon moottorinohjaimelle syötetyistä taulukoista käyntinopeuden ja imusarjassa vallitsevan paineen perusteella. Alifunktioilla huomioidaan muiden tekijöiden, kuten imuilman- ja jäähdytysnesteen lämpötilan vaikutus tarvittavaan polttoainemäärään. Alifunktioilta saadut kertoimet yhdistetään yhdeksi korjauskertoimeksi ja lasketaan polttoainesuuttimille oikea polttoainemäärä. Kun otetaan vielä huomioon moottorinohjaimelle syötetyt moottorin ja polttoainesuuttimien ominaispiirteet voidaan laskea suuttimien avaamisen käytetyn jännitesignaalin pituus, jolla saavutetaan oikea polttoaineannos. Sytytysennakkoa varten on omat alifunktionsa, jotka korjaavat taulukoissa olevia arvoja ulkoisten olosuhteiden mukaan.

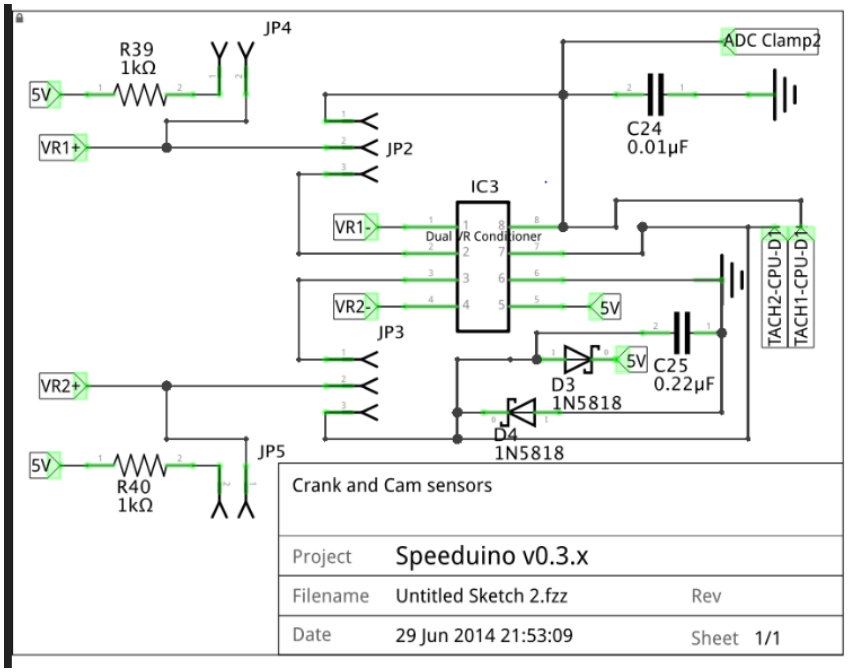
Kun tiedetään kampiakselin asento ja pyörimisnopeus sekä tarvittava pulssinpituus polttoaineen suihkuttamiseksi voidaan arvioida oikea ajankohta suihkutuksen aloittamiselle

ja lopettamiselle. Vaikka kampiakselin asento tiedettäisiin esimerkiksi kymmenen asteen tarkkuudella, voidaan edellisen hampaan ohittamisesta kuluneen ajan ja pyörimisnopeuden perusteella arvioida kampiakselin asentoa tarkemmin kuin kymmenen asteen tarkkuudella. Suihkutuksen aloitus ajoitetaan siten, että kaikki polttoaine ehditään suihkuttaa ennen imuventtiilin avautumista. Koska kampiakselin asento voidaan tietää vain rajatulla tarkkuudella, täytyy arvioida aika, joka kuluu, kunnes saavutetaan oikea kampiakselin asento suihkutuksen aloittamiselle. Asetetaan niin sanottu aikataulutus, jonka mukaan polttoainesuuttimen avaaminen suoritetaan. Kun lähestytään oikeaa kulmaa ja havaitaan lisää hampaita, aikataulutusta päivitetään ajantasaisilla kampiakselin asento- ja pyörimisnopeustiedoilla. Suuttimille syötetään aiemmin laskettu olosuhteisiin sopivan pituinen pulssi. Riippuen valitusta suihkutusjärjestyksestä suuttimia avataan joko yksitellen tai pareina.

Kuten polttoaineensuihkutukselle, tehdään myös sytytykselle oma aikataulutuksensa. Induktiivisia sytytyspuolia käytettäessä tulee huomioida puolien varautumiseen tarvittava aika, ennen kuin kipinä voidaan laukaista. Moottorinohjaimelle on siis asetettu puolien varaamiseen tarvittava aika ja varaaminen aloitetaan ennen kuin saavutetaan sytytykselle laskettu kampiakselin asento. Kampiakselin saavuttaessa oikean asennon laukaistaan kipinä. Riippuen sytytyspuolista lataaminen aloitetaan syöttämällä jännitettä ja kipinä laukeaa jännitteen katketessa, tai päinvastoin.

### 6.2.2 Signaalinkäsittely Speeduino V0.3 -piirilevyllä

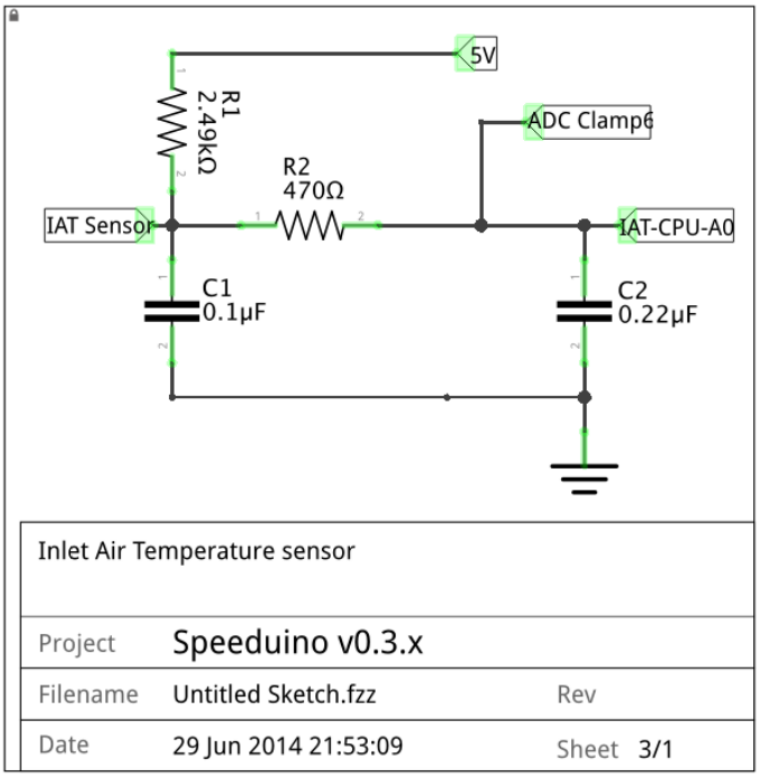
Piirilevyllä on kaksi kanavaa pyörimisnopeusantureita varten, jolloin voidaan käyttää erillisiä antureita esimerkiksi kampi- ja nokka-akselilla. Kuvassa 7 esitetty pyörimisnopeusanturin signaalinkäsittelypiiri.




**Kuva 7.** Elektroniikkakaavio pyörimisnopeusanturien signaalinkäsittelypiiristä. (Leike Speeduino v0.3.6_schematic-kaaviosta)

Erilaisia kokoonpanoja varten piirilevyllä on hyppylankakytkentöjä (kuvassa 7 esimerkiksi JP2), joilla voidaan ohjata signaalien kulkua piirilevyllä. Speeduino Manual (2017, s. 20-21) kehottaa kytkemään JP2 kytkennän navat kaksi ja kolme, kun käytetään induktiivista pyörimisnopeusanturia tai hall-anturia, joka tuottaa muuta kuin viiden voltin suorakaideaaltoa. Kaaviosta näemme, että tällöin VR1+ signaali ohjautuu induktiivisen pyörimisnopeusanturin signaalinkäsittelypiirin liitäntään, jota on merkitty kaaviossa tunnuksella IC3. Induktiivisen pyörimisnopeusanturin toinen johdin kytketään porttiin VR1-, josta signaali menee suoraan signaalinkäsittelypiirille. Signaalinkäsittelypiiri muuttaa saamansa signaalin 0-5V suorakaideaalloksi, joka ohjataan eteenpäin mikro-ohjaimelle. Toisen kanavan osalta kaaviosta löytyy samat liitännät ja toiminnot. Mikro-ohjaimelle syötettävien signaalien jännite on rajattu nollan ja viiden voltin väliin ratkaisulla, joka perustuu zenerdiodeihin. Jännitteen ylittäessä 5V jännite pääsee purkautumaan zenerdiodin läpi maahan ja vastaavasti jännitteen laskiessa liian alas positiivinen jännite purkautuu zenerdiodin läpi virtapiiriin.

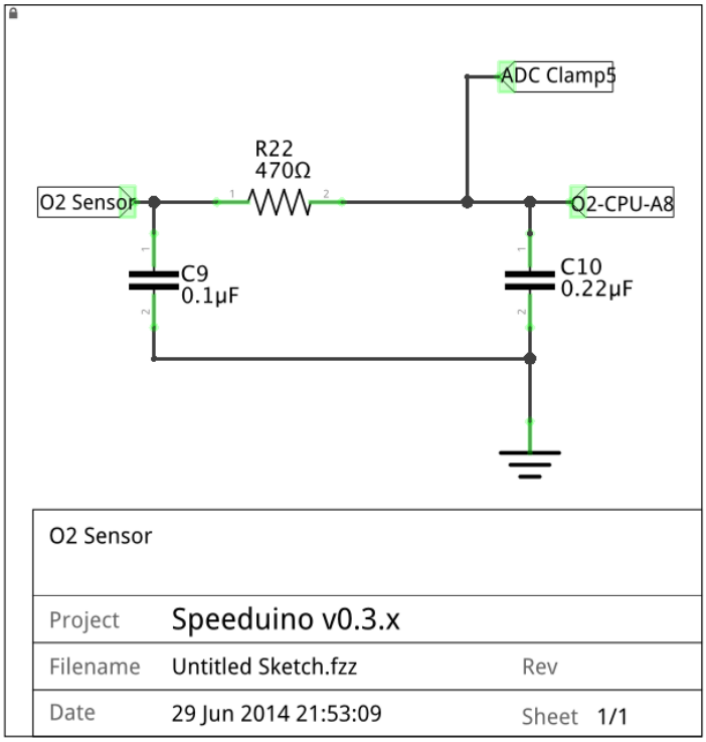
Kuvassa 8 on esitetty imuilman lämpötila-anturin signaalinkäsittelypiiri.




**Kuva 8.** Elektroniikkakaavio imuilman lämpötila-anturien signaalinkäsittelypiiristä. (Leike Speeduino v0.3.6_schematic-kaaviosta)

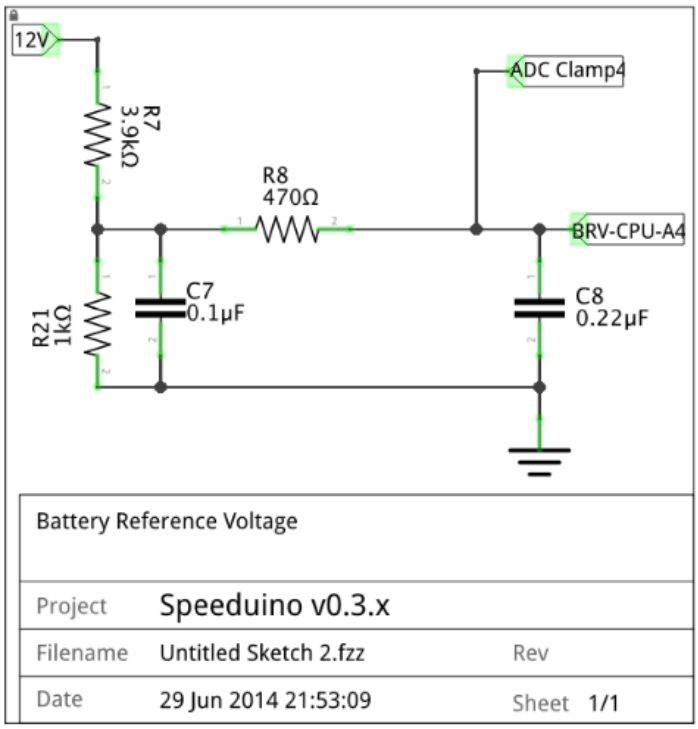
Imuilman ja jäähdytysnesteen lämpötila-antureiden signaalinkäsittely Speeduino-piirilevyllä on identtistä, joten molempien toiminta käsitellään yhtäaikaisesti. Sensorien toinen napa on maadoitettu ja toinen on kytketty IAT Sensor tai vastaavasti CLT Sensor kytkentään. 5V jännite syötetään vastuksen läpi risteykseen, josta jännite jakaantuu lämpötila-anturin ja mikro-ohjaimen suuntiin. Risteyskohdan jännite on syötettävää jännitettä vastuksen R1 jännitehäviön verran pienempi ja jännitehäviö vastuksessa R1 riippuu sen läpi kulkevan sähkövirran suuruudesta. Syötettävän jännitteen ollessa vakio virtapiirissä kulkevan sähkövirran suuruuteen vaikuttaa virtapiirin kokonaisresistanssi, johon anturin resistanssi osaltaan vaikuttaa. Koska anturin resistanssi riippuu lämpötilasta, myös jännite risteyskohdassa riippuu anturin kokemasta lämpötilasta. Kondensaattori C1 tasaa jännitevaihteluja ja täten hillitsee jännitteen heittelyä mittauksen aikana, lisäksi vastus R2 ja kondensaattori C2 muodostavat RC-piirin, joka edelleen tasaa jännitevaihteluita. Mikro-ohjaimelle menevä signaali rajataan nollan ja viiden voltin väliin Clamp -piirissä mikro-ohjaimen suojelemiseksi.

Happianturin, imusarjan paineanturin ja kaasuläpän asentoanturin signaalit käsitellään lähes samalla tapaa. Eroavaisuutena on, että anturit tuottavat 0-5V analogisen signaalin, jolloin jännitteenjakoa ei tarvita. Jännitteenvaihtelua kontrolloidaan ja mikro-ohjainta suojataan samanlaisilla kytkennöillä kuin lämpötila-antureiden vastaavissa virtapiireissä. Kuvassa 9 happianturin signaalinkäsittelypiiri.




**Kuva 9.** Elektroniikkakaavio jäännöshappianturin signaalinkäsittelypiiristä. (Leike Speeduino v0.3.6_schematic-kaaviosta)

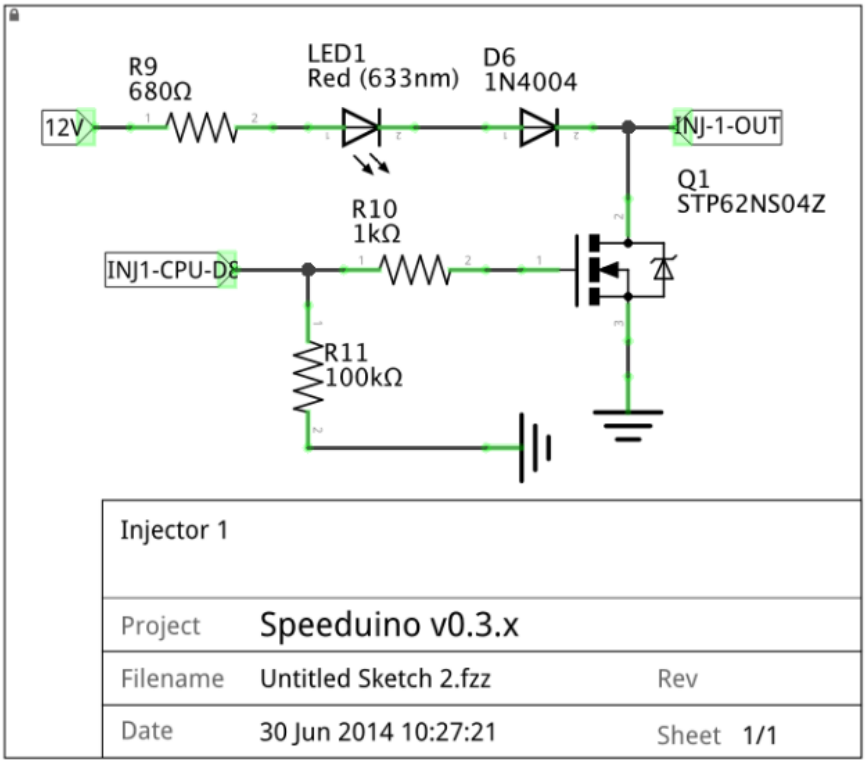
Ajoneuvon käyttöjännitettä käsitellään myös hieman samankaltaisella virtapiirillä. Kuvassa 10 käyttöjännitettä tarkkaileva signaalinkäsittelypiiri.




**Kuva 10.** Elektroniikkakaavio käyttöjännitteen signaalinkäsittelypiiristä. (Leike Speeduino v0.3.6_schematic-kaaviosta)

Käyttöjännite kulkee kahden vastuksen läpi nollapotentiaaliin ja näiden välistä otetaan haara moottorinohjausyksikölle. Virtapiiri toimii kuten lämpötila-antureiden vastaava virtapiiri, mutta vastuksien resistanssit ovat vakiot ja käyttöjännitteen vaihtelu aiheuttaa muutoksen mitattavaan jännitteeseen. Jännitteenvaihtelu ja mikro-ohjaimen suojaus on hoidettu kuten aikaisemmissakin virtapiireissä.

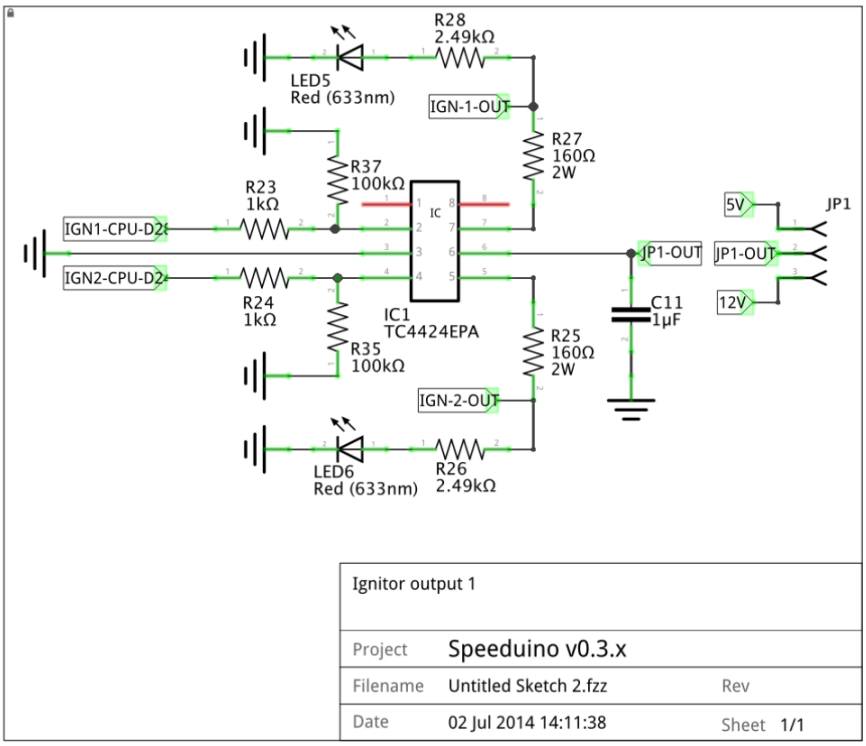
Mikro-ohjain tuottaa ohjaussignaalit neljälle polttoainesuutinkanavalle. Kukin signaali haaroittuu vastuksen läpi MOSFET-transistorin hilan kannalle, sekä toisen vastuksen läpi nollapotentiaaliin. Jännite hilassa saa puolijohteen johtavaksi ja virta polttoainesuuttimelta pääsee kulkemaan nollapotentiaaliin. Samalla virta pääsee kulkemaan vastuksen, hohtodiodin, diodin ja transistorin läpi nollapotentiaaliin, saaden hohtodiodin loistamaan, mikä kertoo moottorinohjausyksikön käyttäjälle kyseisen polttoainesuutinkanavan olevan käytössä. Hilan ollessa jännitteetön transistori ei johda sähköä kollektorilta emitterille, polttoainesuutin on kiinni ja hohtodiodi ei loista. Kuvassa 11 esitetty polttoainesuuttimen ohjaamiseen tarkoitettu piiri.




**Kuva 11.** Elektroniikkakaavio polttoainesuuttimen ohjauspiiristä. (Leike Speeduino v0.3.6_schematic-kaaviosta)

Piirilevyllä on neljän polttoainesuutinkanavan lisäksi neljä oheislaitteille, kuten polttoainepumppu tai jäähdyttimen tuuletin, tarkoitettua kanavaa, joiden toiminta vastaa polttoainesuutinkanavia. Oheislaitteiden tapauksessa virtapiiristä puuttuu hohtodiodin haara kokonaisuudessaan, jolloin käytön aikana piirilevystä ei voi nähdä onko kanava käytössä.

Hyppylankakytkennällä JP1 valitaan sytytyksessä käytettävän ohjausjännitteen suuruus joko viiteen tai 12 volttiin. Piirilevyllä on neljä sytytyskanavaa, joissa on käytetty kahdelle kanavalle yhteistä mikropiiriä, joka toimii kahden MOSFET-transistorin tavoin. Kuten polttoainesuuttimien tapauksessa mikro-ohjaimelta saatu signaali yhdistää transistorin navat ja valittu jännite yhdistyy IGN-OUT liitäntään sekä vastuksen ja hohtodiodin kautta nollapotentiaaliin. Kuvassa 12 esitetty kahden sytytyskanavan ohjauksesta huolehtiva piiri.




**Kuva 12.** Elektroniikkakaavio sytytyspuolien ohjauspiiristä. (Leike Speeduino v0.3.6_schematic-kaaviosta)

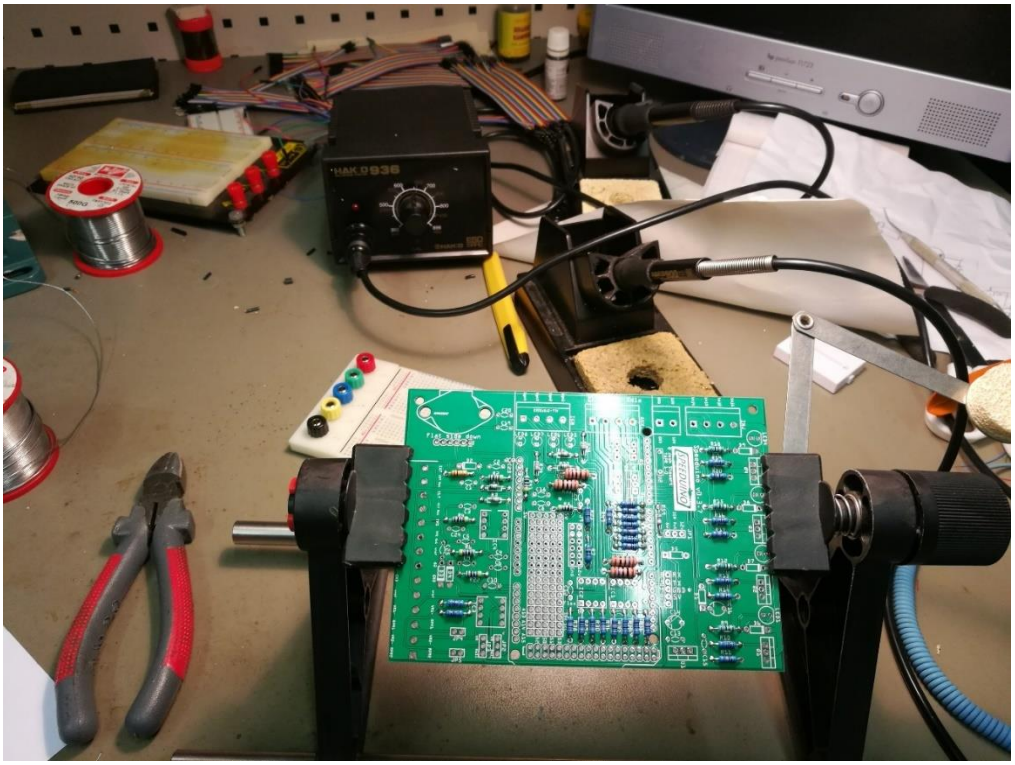
# 7 SPEEDUINO-MOOTTORINOHJAUKSEN ASENTAMINEN

Speeduino-moottorinohjausjärjestelmän toimintaa testattiin käytännössä asentamalla se vuosimallin 1989 Honda CBR600F malliseen moottoripyörään (myöhemmin projektipyörä). Projektipyörässä on nelisylinterinen nelitahtinen rivimoottori ja alun perin projektipyörässä oli neljäkurkkuinen kaasutin sekä kapasitiivinen sytytysjärjestelmä (Honda, 1987, 1-4). Projektipyörään ei ole olemassa suoraan polttoainesuihkutuksen tai ohjelmoitavan sytytyksen komponentteja, joten muutostyö tuli tehdä soveltaen muihin moottoreihin tarkoitettuja osia ja valmistamalla osia itse. Asentaminen oli tarkoituksena tehdä siten, että tarvittaessa voitaisiin helposti palata alkuperäiseen rakenteeseen, mikäli moottorinohjauksen toiminta ei tyydytä tai projektipyörän myyminen tulisi ajankohtaiseksi.

Muutokseen tarvittavia osia valittaessa käytettiin apuna Markus Kakon opinnäytetyötä *MegaSquirt-moottorinohjaimen rakennus ja asennus*. Aina kun mahdollista valittiin uusien merkkiosien sijasta käytettyjä tai tarvikeosia kustannusten hillitsemiseksi.

## 7.1 Speeduino V0.3.7 -piirilevyn kokoaminen

Kirjoitushetkellä V0.3.7 -piirilevyä voi ostaa virallisesta Speeduino-verkkokaupasta ainoastaan elektroniikkakomponentit asentamattomana, jolloin ostajan on huolehdittava erillisenä myytävän komponenttilajitelman asentamisesta ja juottamisesta piirilevylle. Komponentit tulivat selkeästi eriteltyinä ja merkittyinä. Pakkauksen mukana tuli myös osaluettelo ja komponenttien paikat piirilevyllä ovat selkeästi merkittyjä piirroksin, kirjaimin ja numeroin. Komponentit asennettiin pujottamalla johtimet piirilevyllä olevista rei’istä ja juottamalla ne kiinni säädettävän juotosaseman sekä -tinan avulla. Lisäksi piirilevylle juotettiin liitännät sähköjohtimille sekä Arduino 2560 Mega -mikro-ohjaimelle. Kuvassa 13 Speeduino V0.3.7. -piirilevy elektroniikkakomponenttien asennusvaiheessa.



**Kuva 13.** Speeduino V0.3.7- piirilevy kokoamisvaiheessa.

## 7.2 Moottoripyörään tarvittavat muutokset

Elektronisen moottorinohjauksen asentaminen projektipyörään ja luotettavan toiminnan varmistaminen vaatii muutoksia polttoainejärjestelmään, sytytykseen, sähkökytkentöihin sekä anturointiin, muun muassa happianturin lisäämisen pakoputkeen. Muutokseen tarvittavia osia valittaessa käytettiin apuna Markus Kakon opinnäytetyötä *MegaSquirt-moottorinohjaimen rakennus ja asennus*.

### 7.2.1 Polttoainejärjestelmään tehdyt muutokset

Kaasuttimet täytyi korvata polttoainesuuttimilla ja kaasuttimien mukana poistuneet kaasuläpät ja imusarja tuli korvata muulla ratkaisulla. Erilaisia ratkaisuvaihtoehtoja pohdittiin esimerkiksi yksilöllisen alumiinisen imusarjan valmistamisesta hitsaamalla, mutta lopulta päädyin tilaamaan käytetyn Honda CBR600F4i moottoripyörään tarkoitetun neljäkurkkuisen kaasuläppärungon. Läppärunkoon on integroitu tyhjäkäyntiventtiilin koneisto, kaasuläpän asentotunnistin ja sen mukana tuli myös polttoainesuuttimet, polttonesteenjakoputki sekä järjestelmäpaineventtiili. Läppärunko muistuttaa rakenteeltaan hyvin paljon alkuperäisiä kaasuttimia, sillä molemmissa on rinnakkain neljä pyöreää kanavaa, joiden läpi ilma virtaa moottoriin. Sen sijaan, että kaasuttimessa kuristuksen

kohdalla laskevan ilmanpaine imee polttoainetta kohokammiosta, kaasuläppärungoissa ei ole kuristusta vaan sähköisesti ohjatut polttoainesuuttimet suihkuttavat polttoaineen imukanavaan.

Koska valittua kaasuläppärunkoa ei päästy fyysisesti mittaamaan ennen ostopäätöstä, eikä luotettavaa tietoa mitoista löydetty, luotettiin saman moottoripyörävalmistajan uudemman sukupolven vastaavan mallin läppärungon sopivan riittävän hyvin projektipyörään. Tilatun läppärungon mittaaminen paljasti, että moottorin ja läppärungon ilmakanavat olivat erikokoisia sekä kahden keskimmäisen välisissä etäisyyksissä oli noin 19mm ero. Läppärungon ja moottorin imukanavien yhteensovittamiseen tarvittiin ratkaisu. Alkuperäisessä rakenteessa kaasuttimen ja moottorin liityntä oli toteutettu kumisilla holkeilla ja näiden ympärille laitetuilla kiristimillä, joten sain idean valmistaa silikonisia yhdyntäkappaleita läppärungon ja moottorin välille. Yhdyntäkappaleissa olisi sopivat halkaisijat sekä läppärungolle, että moottorin imukanaville ja nämä olisivat epäkeskeisesti toistensa suhteen, jolloin myös kanavien sijainnin aiheuttamat ongelmat saadaan ratkaistua. Kuvassa 14 Solidworks-ohjelmistolla 3D-mallinnettu yhdyntäkappale.

**Kuva 14.** 3D-malli yhdyskappaleesta.

Myös perinteisempiä valmistusmenetelmiä harkittiin, mutta päädyin 3D-tulostamaan muovista muotit, joihin kappaleet voitaisiin valaa. Osoittautui vaikeaksi löytää materiaalia, joka sietäisi riittävästi polttoainetta, öljyä ja korkeaa lämpötilaa sekä olisi joustavaa ja harrastelijan valettavissa. Ehdot täyttävä polyuretaani löytyi ja suunnitelman kanssa jatkettiin eteenpäin.

Alkuperäisistä kaasuttimista, kumisista holkeista ja hankitusta läppärungoista otettujen mittojen perusteella mallinnettiin 3D-mallit tarvittavista yhdyskappaleista. Käyttämällä hyväksi Solidworks-ohjelmiston valmiita muotinsuunnittelutoimintoja mallinnettiin kolmiosainen muotti. Muotista tehtiin kolmiosainen, koska kaksi osaa ympäröivät valettavan tuotteen ulkopuolelta ja yksi osa jää valettavan osan sisään ja täten määrittää kappaleen sisäpuolisen muodon. Kuvassa 15 muovista tulostetut valumuotit. Vasemmalla käyttövalmiiksi koottuna ja oikealla kolmessa osassa.

**Kuva 15.** 3D-tulostetut valumuotit.

Kappaleen sisäpuolisten muotojen monimutkaisuuden vuoksi sisäosaa ei saisi kokonaisena valmiista kappaleesta ulos, joten sisäosa tuli rikkoa valetun kappaleen vapauttamiseksi. Muotit tulostettiin PLA-muovista 3D-tulostimella. Kaksiosainen ulkomuotti tulostettiin

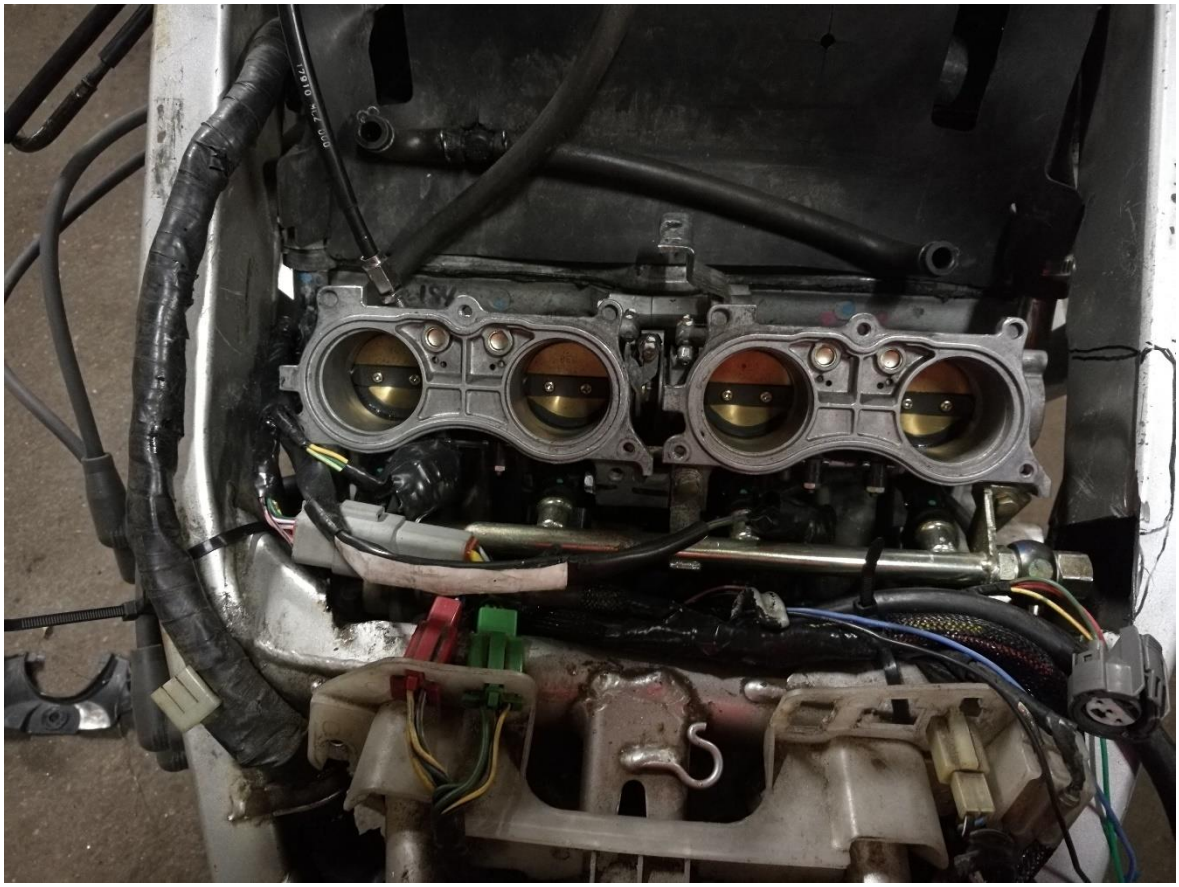
uudelleen käytettävän luonteensa vuoksi kestämään uudelleen käyttöä, kun taas muotin sisäosat (yksi valettavaa kappaletta kohden) tulostettiin mahdollisimman heikkorakenteiseksi muotin rikkomisen helpottamiseksi.

Muottien valukappaletta koskettavat pinnat käsiteltiin valukappaleen irrottamista helpottavalla aineella, minkä jälkeen ulkomuotit puristettiin yhteen kierretankojen ja mutterien avulla. Sisämuotti asetettiin paikalleen ja liikkumattomuus varmistettiin teipillä. Muotin täyttöaukkoon taiteltiin kartongista täyttösuppilo, jonka avulla muotit täytettiin valitulla UR3450 polyuretaanilla. Muottien annettiin vuotaa hieman yli ja suppiloon jätettiin ylimäärä polyuretaania, jottei kappaleisiin jäisi suuria ilmakuplia. Polyuretaanin kovettua ulkomuotit purettiin, sisämuotti rikottiin valukappaleen sisään sekä valusaumoja siistittiin tarpeen mukaan. Kuvassa 16 kaksi valmista valettua yhdyskappaletta.

**Kuva 16.** Valmiita yhdyskappaleita.

Yhdyskappaleiden valu onnistui odotettua paremmin ja lopputuotteet sopivat mittojensa puolesta hyvin kaasuläppärunkojen ja sylinterikannessa olevien imukanavien yhdistämiseen. Läppärunkoja sovitettaessa törmättiin kuitenkin ongelmaan, jota ei osattu ennalta odottaa. Läppärungoissa oleva polttoaineen jakotukki ei mahtunut projektipyörän rungon sisään,

joten runkoon jouduttiin tekemään lovia asentamisen mahdollistamiseksi. Kuvassa 17 kaasuläppärungot asennettuna projektipyörään rungon loveamisen jälkeen.



**Kuva 17.** Läppärungot asennettuna projektipyörään.

Projektipyörän alkuperäinen kaasuttimien polttoainetarpeen tyydyttämiseen tarkoitettu polttoainepumppu ei riitä täyttämään polttoainesuuttimien tarpeita ja polttoainejärjestelmästä puuttuu kokonaan paineventtiilin vaatima paluulinja polttoainesäiliöön. Alkuperäinen polttoainepumppu sijaitsi polttoainelinjassa polttoainesäiliön ja kaasuttimien välissä. Esimerkiksi Honda CBR600F4i moottoripyörässä polttoainesuihkutukseen tarkoitettu pumppu on sijoitettu polttoainesäiliöön ja siksi ei sovellu suoraan projektipyörään. Polttoainelinjaan asennettavan ja riittävän pienituottoisen pumpun löytäminen osoittautui hankalaksi, sillä suurin osa tarjolla olevista pumpuista oli tarkoitettu autojen virityskäyttöön. Liian suuri tuotto aiheuttaa polttoaineen ylimääräistä kiertoa järjestelmässä samalla lämmittäen polttoainetta. Pahimmillaan pumpun tuotto voi olla niin suuri, ettei järjestelmäpaineventtiili kykene palauttamaan riittävästi polttoainetta säiliöön ja paine jakoputkessa nousee hallitsemattomasti. Ongelma ratkaistiin hankkimalla

tasavirtamoottorinohjain, joka säätää pumpulle syötettävää jännitettä saamansa pulssimuotoisen signaalin perusteella. Moottorinohjainyksiköltä syötetään signaalia tasavirtamoottorinohjaimelle, joten polttoainepumpun tuottoa voidaan ohjata moottorin käyntitilan, esimerkiksi pyörimisnopeuden perusteella. Koska ilman polttoaineen paineanturia ei voida kontrolloida painetta polttoaineen jakoputkessa, säilytettiin alkuperäinen järjestelmäpaineventtiili. Tässä tapauksessa polttoainepumpun tuotto tulee säätää siten, että kaikissa käyttötilanteissa se on hieman suurempi kuin tarvittaisiin. Silloin järjestelmäpaineventtiilillä on mahdollisuus toimia tarkoitetulla tavalla ja paine polttoaineen jakoputkessa ei pääse laskemaan liian alas.

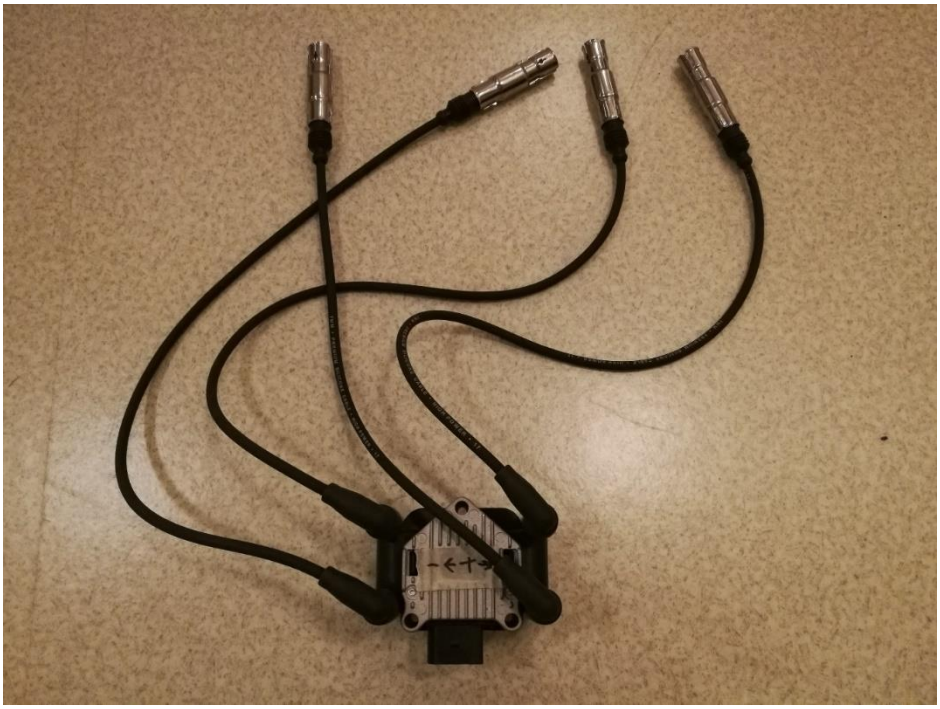
Polttoainepumpuksi valittiin polttoainelinjaan asennettava pumppu, jonka tuotto vastaisi tarvetta ja jossa on imupuolella 12mm liitäntä sekä painepuolella M18x1,5 kierreliitäntä. Pumppua on käytetty muun muassa BMW-merkkisissä henkilöautoissa varaosakoodilla 16 12 1 150 201. Polttoainesuodattimeksi pyrittiin valitsemaan mahdollisimman pienikokoinen korkeapaineiseen polttoainelinjaan sopiva suodatin. Valitussa suodattimessa on 8mm liitännät, jotka sopivat pumpun ja polttoaineen jakoputken väliseen linjaan. Vastaavaa suodatinta käytetään muun muassa Peugeot 106 -mallisessa henkilöautossa. Kuvassa 18 polttoainesuihkutuksen vaatimat muutokset polttoainejärjestelmään. Komponentit alhaalta lähtien: polttoainepumppu, polttoainesuodatin ja kaasuläppärunko, jossa polttoainesuuttimet ja järjestelmäpaineventtiili. Järjestelmäpaineventtiililtä lähtee polttoaineen paluulinja kuvasta ulos oikealle.

**Kuva 18.** Päivitetty polttoainejärjestelmä.

### 7.2.2 Sytytykseen tehdyt muutokset

Moottoripyörän alkuperäinen kapasitiivinen sytytysjärjestelmä ei sisällä liitäntöjä ulkoiselle ohjaukselle, joten sen ohjaaminen ulkoisella moottorinohjausjärjestelmällä ei onnistu yksinkertaisesti. Alkuperäinen sytytysjärjestelmä päädyttiin korvaamaan edullisella induktiivisella sytytysjärjestelmällä, joka on yhteensopiva Speeduino moottorinohjausjärjestelmän kanssa. Koska induktiivisessa sytytysjärjestelmässä sytytyspuolaa käytetään sytytysenergian varastoimiseen, ja alkuperäisessä kapasitiivisessa sytytysjärjestelmässä sytytyspuolia käytetään vain jännitteen kasvattamiseen, ei ole syytä olettaa projektipyörän sytytyspuolien toimivan osana uutta sytytysjärjestelmää. Siispä koko sytytysjärjestelmä korvattiin ja sytytyspuoliksi valittiin Markus Kakon esimerkin mukaisesti kaksoiskipinäsytytyspuolat, joita ohjataan suoraan moottorinohjaimella. Neljäsylinteriselle moottorille tarkoitettu sytytyspuolapaketti sisältää kaksi kaksoiskipinäsytytyspuolaa koteloituna samaan rakenteeseen. Alkuperäiset sytytystulpanjohdot eivät olleet

yhteensopivia uusien sytytyspuolien kanssa, joten nekin korvattiin uusilla. Kuvassa 19 käytetty kaksoiskipinäsytytyspuolat sekä sytytystulpanjohdot.



**Kuva 19.** Kaksoiskipinäsytytyspuola sytytysjohtimineen.

### 7.2.3 Anturointiin tehdyt muutokset

Projektipyörän kampiakselilla oli jo valmiina induktiivinen pyörimisnopeusanturi, jota alkuperäinen sytytysjärjestelmä käytti. Päätin käyttää tätä alkuperäistä anturia kampiakselin pyörimisnopeuden mittaamiseen.

Moottoripyörän alkuperäinen sytytys- tai polttoainejärjestelmä eivät huomioineet toiminnassaan imuilman tai moottorin lämpötilaa, vaan esimerkiksi kylmäkäynnistyksiä auttamaan kuljettajan oli käytettävä seoksen rikastamiseen erillistä vipua ohjaustangossa. Imuilman lämpötilaa mittaamaan valittiin Markus Kakon esimerkin mukaan avorakenteinen anturi ja moottorin lämpötilaa mitattiin jäähdytysjärjestelmässä valmiiksi olevalla anturilla.

Toimiakseen tehokkaasti moottorinohjaus tarvitsee tietoa moottorin kuormitustilasta ja kaasuläpän asennosta. Kuormituksen mittaustarpeeseen valittiin suoraan moottorinohjausyksikön piirilevylle asennettava paineanturi MPX4250, joka mittaa imusarjassa vallitsevaa painetta. Anturi on suositeltu käytettäväksi Speeduino-

moottorinohjausyksikön kanssa ja ostettiin moottorinohjausyksikön hankinnan yhteydessä Speeduino-verkkokaupasta. Paineanturi kytkettiin letkun välityksellä yhteyteen imusarjan kanssa. Kaasuläpän asentoa mitattiin läppärungossa valmiiksi olevalla alkuperäisellä anturilla.

Koska projektipyörässä ei koskaan ole ollut elektronista moottorinohjausta, oli käynninohjaukseen tarvittavat arvot lähdettävä kokeellisesti hakemaan ilman pohjatietoja. Jotta saadaan riittävästi tietoa moottorin käynnistä ja palamistapahtumasta sylintereissä oli tarkka pakokaasujen happipitoisuuden mittaaminen elinehto turvallisten käyntiarvojen saavuttamiselle. Tähän tarkoitukseen ostettiin Bosch LSU4.9 -laajakaistahappianturi ja sen toimintaa ohjaamaan 14point7 –yrityksen valmistama Spartan 2 ohjausyksikkö laajakaistahappianturille. 14point7 kertoo sivuillaan, että ohjausyksikkö on suunniteltu nimenomaan jälkiasennettavia moottorinohjausyksiköitä ajatellen, sillä se antaa useimmille mittareille ja moottorinohjausyksiköille yhteensopivaa lineaarista analogista signaalia (14point7). Kuvassa 20 14point7 Spartan 2 -laajakaistahappianturin ohjausyksikkö.

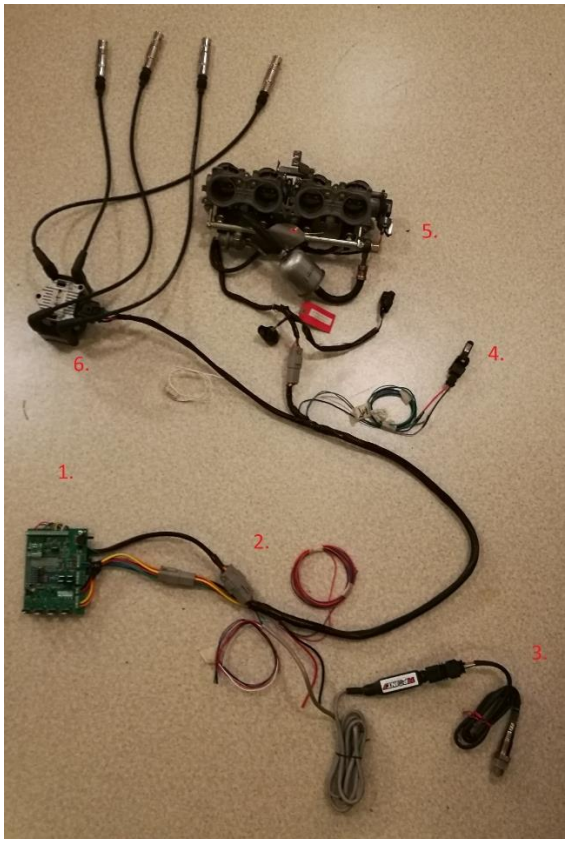

**Kuva 20.** 14point7 Spartan 2 -laajakaistahappianturin ohjausyksikkö

### 7.2.4 Johtosarjaan tehdyt muutokset

Projektipyörän alkuperäinen johtosarja ei sisältänyt tarvittavia johtimia tai liittimiä lisääntyneille antureille ja toimilaitteille, joten päätin alkuperäisen johtosarjan rinnalle lisätä

tarvittavat johtimet ja liittimet erillisenä johtosarjana. Jättämällä alkuperäinen johtosarja koskemattomaksi mahdollistettiin alkuperäiseen sytytysjärjestelmään palaaminen.

Kuvassa 21 lähes valmis johtosarja koesovitettuna anturien, sytytyspuolien ja kaasuläppärungon kanssa. Kohdassa 1. Speeduino-moottorinohjausyksikkö, kohdassa 2. johtosarjan liityntä Speeduino-moottorinohjaimeen ja moottoripyörän alkuperäiseen johtosarjaan, kohdassa 3. jäännöshappianturi ohjainyksikköineen, kohdassa 4. imuilman lämpötila-anturi, kohdassa 5. kaasuläppärunko polttoainesuuttimineen ja kaasuläpän asentoantureineen, kohdassa 6. kaksoiskipinäpuolat.




**Kuva 21.** Johtosarja ja komponentit koesovituksessa.

Haltech-moottorinohjausjärjestelmän keskustelupalstalta saadun tiedon mukaan kyseisen moottorinohjaimen johtosarjassa käytetään mitaltaan 14 AWG johdinta suuttimien ja sytytyspuolien suurivirtaisia johtimia varten sekä antureiden ja muiden pienivirtaisia signaalijohtimia varten mitaltaan 20 AWG johdinta (Haltech). Tilasin edellä mainitun kokoisia silikonipäällysteisiä johtimia useissa eriväreissä, jotta eri tarkoituksiin tulevat johtimet olisi helpompi tunnistaa ja nimetä. Liittimiksi valitsin Deutsch DT-liittimien kopiot,

kustannusten rajoittamiseksi. Tiettyihin kohteisiin (mm. sytytyspuola, suuttimet, anturit) täytyi käyttää alkuperäisiä liittimiä, sillä kyseisissä komponenteissa liitin on osa runkoa eikä siksi vaihdettavissa. Liitteessä 2 on esitetty projektipyörään tehdyn johtosarjan kytkentäkaavio yksinkertaistetussa muodossa, sekä tarkempi erittely käytetyistä johtimista.

### 7.3 Speeduino-ohjelmiston asennus ja moottorin arvojen asettaminen

Ennen moottorinohjausyksikön käytön aloittamista on Arduino -mikro-ohjaimelle asennettava Speeduino-laiteohjelmisto (englanniksi Firmware). Viimeisin virallinen laiteohjelmisto on vapaasti ladattavissa Speeduino-moottorinohjausjärjestelmän kotisivuilta. Lataamisen jälkeen laiteohjelmisto käännettiin Arduino IDE -ohjelmistolla binäärimuotoiseksi, minkä jälkeen laiteohjelmisto asennettiin mikro-ohjaimelle. (Speeduino manual, 2017, s. 1-4)

Alkuarvojen asettamiseen moottorinohjaimelle ja moottorinohjaimen ohjelmoimiseen käytin TunerStudio 3.0.28 -ohjelmistoa. TunerStudio-ohjelmistolla luodaan projektitiedosto, johon ladataan Speeduino-moottorinohjaimen yhdistämiseen tarvittavat asetukset, minkä jälkeen voidaan yhdistää moottorinohjaimeen. Yhteyden muodostamisen jälkeen TunerStudio-ohjelmistolla voidaan syöttää moottorinohjaimelle tiedot moottorin ominaisuuksista, kalibroida antureita, sekä seurata moottorinohjaimen toimintaa reaaliaikaisesti. Ohjelmiston avulla säädetään suihkutettavan polttoaineen määrän ja sytytyksen taulukoita.

Sytytys- ja polttoaineensuihkutusjärjestelmien käyttöönotto suoritettiin kahdessa vaiheessa, mikä teki ongelmanaiheuttajien etsimisestä helpompaa. Ensiksi projektipyörä saatettiin käyntikuntoiseksi alkuperäisen polttoainejärjestelmän ja induktiivisen sytytysjärjestelmän kanssa, minkä jälkeen lisättiin polttoaineensuihkutus.

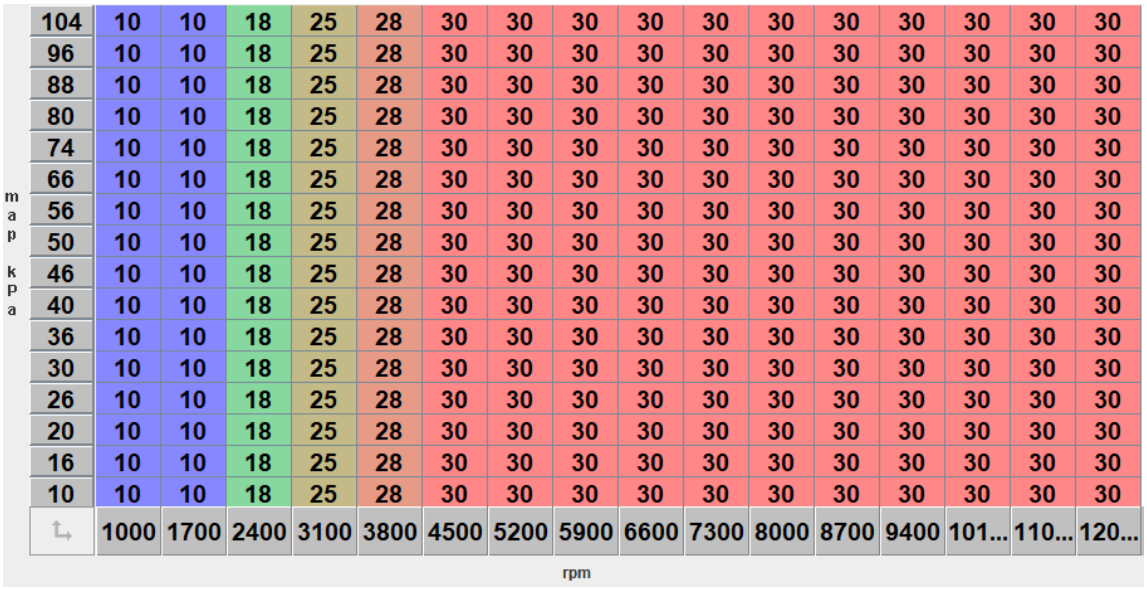
Moottorin sytytyksen ajoittamiseen käytettiin ajoituslamppua. Ajoituslamppu on työkalu, joka yhdistetään sytytysjohtimeen ja joka väläyttää valoa, kun virta kulkee johtimessa. Osoittamalla välkkyvällä valolla moottorin ajoitusmerkintöjä saadaan selville sytytyksen ajoitus. Moottoria käytettiin alkuperäisellä sytytysjärjestelmällä eri käyntinopeuksilla ja kullakin pyörimisnopeudella tarkastettiin pyörimisnopeutta vastaava sytytysennakko ajoitusmerkintöjen ja ajoituslampun avulla. Koska alkuperäinen järjestelmä anturoi

pelkästään pyörimisnopeutta, ovat sytytysennakon arvot riippumattomia moottorin käyntitilan muista suureista. Mittauksia tehtiin neljä pyörimisnopeusalueen alapäässä, koska tarkoituksena oli saada moottori vain toimintakuntoon, yläkierrosalueen sytytys säädetään kohdalleen myöhemmin. Taulukossa 2 kirjattuna pyörimisnopeudet ja niitä vastaavat sytytysennakon arvot.

*Taulukko 2. Mitattuja sytytysennakon arvoja alkuperäisellä sytytysjärjestelmällä*

| Pyörimisnopeus (rpm) | Sytytysennakko (astetta) |
|---|---|
| 1500 | 10 |
| 2000 | 10 |
| 3000 | 25 |
| 4000 | 28 |

Projektipyörään asennettiin Speeduino-moottorinohjausjärjestelmä johtosarjoineen, sekä induktiiviset kaksoiskipinäsytytyspuolat. Johtosarja kytkettiin alkuperäisen johtosarjan jännite- ja pyörimisnopeusanturin liitäntöihin. Sytytystulpat irrotettiin sylintereistä, kytkettiin sytytysjohtimiin ja asetettiin projektipyörän runkoa vasten siten, että sytytystulpat pääsivät maadoittumaan ja kipinät olivat nähtävissä. Moottoria pyöritettiin käynnistysmoottorilla ja samalla ajoituslampun avulla tarkastettiin sytytyksen ajoitus. TunerStudio-ohjelmiston avulla muutettiin sytytyksen pyörimisnopeusanturilta saatavan signaalin ja sytytysajankohdan välistä kulmaa, kunnes ajoitusmerkinnät täsmäsivät moottoria pyöritettäessä. Sytytystulpat asennettiin takaisin paikoilleen ja moottoripyörä käynnistettiin kiinteällä kymmenen asteen sytytysennakolla. Taulukon 2 avulla luotiin alustava sytytyskartta, jonka toimintaa testattiin käynnistämällä moottori. Kuvassa 22 taulukon 2 perusteella luotu sytytysennakkokartta. X-akselilla pyörimisnopeus 1000-12000 rpm ja Y-akselilla imusarjan absoluuttinen paine 10-104 kPa, sytytysennakon arvot merkitty asteina.




**Kuva 22.** Käytetty sytytysennakkokartta

Kaasuläppärungot asennettiin polyuretaanista valettujen yhdyskappaleiden avulla alkuperäisten kaasuttimien tilalle ja myös muu polttoainejärjestelmä korvattiin. Polttoaineen paluulinjan letkun toinen pää laitettiin polttoainesäiliön täyttöaukosta sisään ja polttoainejärjestelmän toimintaa testattiin kytkemällä polttoainepumpun johtimet suoraan akun napoihin. Polttoainetta alkoi virrata paluulinjasta polttoainesäiliöön ja ilmenneet vuotokohdat korjattiin.

Koska mitään pohjatietoja soveltuvista polttoainekartoista ei ollut, käynnistämiseen soveltuvaa polttoaineseosta lähdettiin hakemaan kokeilemalla. TunerStudio-ohjelmiston avulla muutettiin polttoainekartan arvoja laajalla alueella ja muutoksien jälkeen moottoria yritettiin käynnistää. Noin kymmenen yrityksen jälkeen moottori saatiin käynnistymään, kun käytettiin kuvassa 24 näkyvää polttoainekarttaa ja kaasuläppiä raotettiin hieman. Moottorin käynti ei ollut käyntiäänen perusteella tasaista, mutta käynnistyminen on hyvä lähtökohta myöhemmille säädöille. Kuvassa 23 moottorin arvojen asettamiseen käytetty koeasetelma.


**Kuva 23.** Ensimmäisiin käynnistyksiin käytetty koeasetelma

Kuvassa 24 käynnistämiseen käytetty polttoainekartta. X-akselilla pyörimisnopeus 1000-12000 rpm ja Y-akselilla imusarjan absoluuttinen paine 10-104 kPa, polttoaineen arvot merkitty täytösasteena prosentteina

| map kpa | 1000 | 1200 | 1300 | 1400 | 2000 | 2800 | 3600 | 4500 | 5200 | 5500 | 5800 | 6200 | 6500 | 6800 | 6900 | 12000 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 100 | 40 | 40 | 40 | 40 | 40 | 40 | 93 | 93 | 93 | 93 | 93 | 93 | 93 | 93 | 93 | 95 |
| 96 | 40 | 40 | 40 | 40 | 40 | 40 | 93 | 90 | 90 | 90 | 90 | 91 | 92 | 93 | 93 | 93 |
| 90 | 40 | 40 | 40 | 40 | 40 | 40 | 90 | 88 | 86 | 86 | 86 | 87 | 88 | 89 | 89 | 90 |
| 86 | 40 | 40 | 40 | 40 | 40 | 40 | 88 | 86 | 84 | 83 | 83 | 83 | 84 | 85 | 85 | 86 |
| 76 | 40 | 40 | 40 | 40 | 40 | 40 | 82 | 80 | 78 | 76 | 76 | 76 | 77 | 78 | 78 | 78 |
| 70 | 40 | 40 | 40 | 40 | 40 | 40 | 78 | 76 | 74 | 72 | 72 | 72 | 73 | 74 | 74 | 74 |
| 66 | 40 | 40 | 40 | 40 | 40 | 40 | 73 | 72 | 70 | 68 | 68 | 69 | 69 | 70 | 70 | 71 |
| 60 | 40 | 40 | 40 | 40 | 40 | 40 | 69 | 68 | 66 | 64 | 65 | 65 | 66 | 66 | 67 | 67 |
| 56 | 40 | 40 | 40 | 40 | 40 | 40 | 65 | 63 | 62 | 61 | 61 | 61 | 62 | 63 | 63 | 63 |
| 50 | 40 | 40 | 40 | 40 | 40 | 40 | 61 | 59 | 58 | 57 | 57 | 58 | 58 | 59 | 59 | 59 |
| 46 | 40 | 40 | 40 | 40 | 40 | 40 | 56 | 55 | 54 | 53 | 54 | 54 | 54 | 55 | 55 | 55 |
| 40 | 40 | 40 | 40 | 40 | 40 | 40 | 52 | 51 | 50 | 50 | 50 | 50 | 51 | 51 | 51 | 52 |
| 36 | 40 | 40 | 40 | 40 | 40 | 40 | 48 | 47 | 46 | 46 | 46 | 47 | 47 | 47 | 48 | 48 |
| 30 | 40 | 40 | 40 | 40 | 40 | 40 | 44 | 43 | 42 | 42 | 43 | 43 | 43 | 44 | 44 | 44 |
| 26 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 39 | 39 | 39 | 39 | 39 | 40 | 40 | 40 | 40 |
| 16 | 40 | 40 | 40 | 40 | 40 | 40 | 37 | 36 | 35 | 34 | 33 | 33 | 33 | 33 | 33 | 33 |

rpm

**Kuva 24.** Käytetty polttoainekartta

### 7.4 Kokemuksia Speeduino-moottorinohjauksesta ja sen asentamisesta

Speeduino-piirilevyn kokoaminen oli hyvillä välineillä helppoa, koska komponentit ja niiden paikat olivat selkeästi merkittyjä. Halutessaan voi ostaa myös piirilevyn, johon suurin osa komponenteista on jo valmiiksi juotettuna. Piirilevyllä tapahtuvaan signaalinkäsittelyyn ei tarvitse perehtyä, mikäli moottorinohjaimen tarjoamat perusominaisuudet riittävät.

Arduino Mega 2560 mikro-ohjaimelle asennettavan Speeduino ohjelmiston asentaminen oli suoraviivaista projektin kotisivuilta löytyvän dokumentaation avulla ja halutessaan mikro-ohjaimen voi ostaa valmiiksi ohjelmiston kanssa Speeduino verkkokaupasta. Itse ohjelmiston toimintaan ei tavallisen käyttäjän tarvitse perehtyä, vaan kaikki tarvittavat asetukset voidaan tehdä TunerStudio-ohjelmiston kautta.

Koska Speeduino-moottorinohjausjärjestelmä ei sovi suoraan mihinkään moottoriin, vaatii se käyttäjältään tuntemusta käyttämiensä anturien, sytytyspuolien ja polttoainesuuttimien toiminnasta. Käytettyjen komponenttien tyypit vaikuttavat niin johdotukseen kuin moottorinohjaimelle TunerStudio-ohjelmiston välityksellä syötettäviin arvoihin. Moottorinohjaimen asentamisen tueksi on moottorinohjaimen kotisivuilta saatavissa englanninkielisiä oppaita, jotka auttavat muutamien komponenttien kanssa, mutta näistä on apua vain rajatuissa tapauksissa.

Helpointa on asentaa Speeduino-moottorinohjausjärjestelmän moottoriin, jossa on jo entuudestaan elektroninen sytytys ja polttoaineen suihkutus sekä josta löytyy kattavasti tietoa moottorinohjaimelle syötettävistä arvoista sekä alustavia sytytysennakko- ja polttoainekarttoja. Tällöin tarvitsee ainoastaan sovittaa moottorinohjainyksikkö olemassa olevaan johtosarjaan, asettaa arvot sekä kartat ja käynnistää moottori.

Koko projektin ajalta kerättiin talteen kuitit tehdyistä komponenttihankinnoista ja näistä on koostettu liitteeseen 3 taulukointi eri komponenteista ja niiden hinnoista, työkalujen hankintoja ei ole huomioitu. Taulukossa 3 on tiivistettynä liitteen 3 tärkeimmät tiedot.

*Taulukko 3. Projektipyörän muutoksen aikana tehdyt hankinnat*

| | Hinta (€) | Arvioitu kustannus (€) | Erotus (€) |
|---|---|---|---|
| **Moottorinohjausjärjestelmä** | **178,27** | **170** | **-8,27** |
| **Sensorit** | **132,21** | **175** | **42,79** |
| **Polttoainejärjestelmä** | **136,14** | **150** | **13,86** |
| **Sytytysjärjestelmä** | **66,25** | **50** | **-16,25** |
| **Tarvikkeita (mm. johdot, liittimet)** | **138,03** | **100** | **-38,03** |
| **Postikulut, Alv ja tullaus** | 82,41 | | |
| Summa | 733 | 645 | **-88** |

Vaikka kuluja pyrittiin vähentämään aina kun mahdollista, kokonaisuutena projektiin kului merkittävä määrä rahaa. Kokonaissummasta kuitenkin alle neljännes kului itse moottorinohjainyksikköön, joten projektipyörän tapauksessa suurin kustannus muodostui muiden järjestelmien päivittämisestä moottorinohjauksen edellyttämälle tasolle.

Kuitenkin itse Speeduino-moottorinohjausjärjestelmän käyttäminen oli kivutonta ja kaikkia sen tarjoamia ominaisuuksia ei edes ehditty tämän opinnäytetyön puitteissa käsittelemään tai kokeilemaan. Projektipyörän tapauksessa moottorinohjaimen asentaminen oli kuitenkin työläs prosessi, koska suoraan sopivia komponentteja ei löytynyt ja tietoa oli saatavilla hyvin niukasti. Sopivien komponenttien etsiminen, yhteensovittaminen ja johtosarjan suunnitteleminen sekä sytytysennakko- ja polttoainekarttojen luominen täytyi tehdä yrityksen ja erehdyksen kautta ja siksi ne veivät selkeästi eniten aikaa.

# 8 JOHTOPÄÄTÖKSET

Speeduino-moottorinohjausjärjestelmä koostuu melko yksinkertaisista osista ja sen toimintaan tutustuminen on helppoa avoimen lähdekoodin vuoksi. Avoin lähdekoodi tekee Speeduino-moottorinohjainjärjestelmästä helposti muokattavan, niin ohjelmiston, kuin elektroniikkakomponenttien osalta. Peruskäyttäjän ei kuitenkaan tarvitse tietää syvällisesti mitä moottorinohjaimen sisällä tapahtuu ja Speeduino onkin innokkaalle harrastelijalle houkuttava tapa tutustua moottorinohjaimien toimintaan ja niillä saavutettaviin ominaisuuksiin. Edullisen hintansa vuoksi ohjelmoitavan moottorinohjaimen asentaminen pienenkin budjetin projektiin on mahdollista ja jopa suositeltavaa.

Opinnäytetyössään ”Ottomoottorin suorituskyvyn parantaminen sähköisillä järjestelmillä autourheilun jokamiesluokassa” (2015) Paavo Töytäri käsitteli moottorinohjausjärjestelmän tuomia etuja autourheiluluokassa, jossa autoon käytettävät budjetit ovat hyvin rajallisia. Mielestäni Speeduino-moottorinohjausjärjestelmä sellaisenaan tai ylimääräisistä ominaisuuksista karsitulla piirilevyllä ja koodilla, voisi olla ominaisuuksiltaan ja hinnaltaan oikein kilpailukykyinen vaihtoehto autourheilun jokamiesluokkaan. Jokamiesluokan käyttöön räätälöityjen edullisten moottorinohjausyksikköjen valmistaminen on helppoa ja niiden myynnillä saattaa olla kaupallista potentiaalia.

Asennusprosessin työläyden vuoksi tässä opinnäytteessä tyydyttiin saattamaan moottorinohjausjärjestelmä ainoastaan käyntikuntoiseksi. Tämän vuoksi muun muassa kaasuläpän asentoanturi, jäännöshappianturi ja lämpötila-anturit jätettiin kytkemättä, sillä ne eivät olleet käynnistämiselle välttämättömiä ja olisivat vaatineet huomattavan määrän lisätyötä komponenttien sovittamisessa. Projektipyörän keskeneräisyyden vuoksi en nähnyt myöskään järkeväksi lähteä hakemaan esimerkiksi hyviä asetuksia laajemmalle kierrosalueelle, joten kokemuksia Speeduino-moottorinohjausjärjestelmästä arkikäytössä ei saatu.

Opinnäytetyön tekeminen oli mielenkiintoinen ja opettavainen prosessi. Opin paljon moottorinohjausjärjestelmien toiminnasta, elektroniikasta sekä polyuretaanista tuotteiden valamisesta muovista 3D-tulostettuun muottiin. Laajuutensa puolesta aiheesta olisi voinut

tehdä vaikka kaksi opinnäytetyötä: ensimmäisen opinnäytteen keskittyessä Speeduino-moottorinohjaimen toimintaan ja toisen syventyessä moottorinohjaimen asennukseen sekä käytön hyötyihin ja haittoihin. Tämän opinnäytteen puitteissa ei käsitelty Speeduino-moottorinohjainjärjestelmän käyttöä ja asennus rajattiin vain yhteen tapaukseen, joten Speeduino-moottorinohjausjärjestelmän käytettävyydessä ja mahdollisuuksissa on vielä paljon tutkimattomia osa-alueita.

LÄHTEET

Agarwal, T. Capacitor Discharge Ignition (CDI) Working Principle, Its Advantage and Disadvantage. Elprocus -verkkosivusto. Viitattu 22.6.2018. Saatavilla: https://www.elprocus.com/capacitor-discharge-ignition-cdi-system-working/

Arduino. Arduino Mega 2560 rev3 -tuotesivu. Viitattu 30.3.2018. Saatavilla: https://store.arduino.cc/arduino-mega-2560-rev3

Genta, G. Morello, L. Cavallino, F. Filtri L. 2014. The Motor Car - Past, Present and Future. Torino: Politecnico di Torino, Department of Mechanical and Aerospace Engineering.
Saatavilla: https://link-springer-com.ezproxy.cc.lut.fi/book/10.1007%2F978-94-007-8552-6#about

Haltech-moottorinohjausjärjestelmän keskustelupalsta. Viitattu 9.6.2018
Saatavilla: http://forums.haltech.com/viewtopic.php?t=11005

Honda Motor CO., LTD., Service publications Office. 1987. Honda CBR600F Shop Manual.

Juhala, Matti. & Lehtinen, Arto. & Suominen, Matti. & Tammi, Kari. 2005. Moottorialan sähköoppi. 8. painos. Jyväskylä, Gummerus kirjapaino Oy. ISBN 951-9155-19-8

Liikennevirasto. 2018. Henkilöliikennetutkimus 2016. Liikenneviraston tilastoja 1/2018. Helsinki. ISBN 978-952-317-513-6
Saatavilla: https://julkaisut.liikennevirasto.fi/pdf8/lti_2018-01_henkiloliikennetutkimus_2016_web.pdf

Nieminen, Simo. 2005. auto.car, Auton rakenne 1, Moottori ja tehonsiirto. 1. painos, Helsinki, WSOY. ISBN 951-0-26999-9

Nieminen, Simo. 2008. Auton sähkölaitteet. 1. painos, Helsinki, WSOY. ISBN 978-951-0-26997-8

Nieminen, Simo. 2003. Auton sähkötekniikka. 5-6. painos, Helsinki, WSOY. ISBN 951-0-22307-7

Pitkänen, Jorma. 2000. Ottomoottorin polttoaineen syöttölaitteet. Espoo, Teknillinen korkeakoulu. ISBN 951-22-4869-7

Reif, Konrad. 2015. Gasoline Engine Management - Systems and Components. Springer Vieweg, Wiesbaden. ISBN 978-3-658-03964-6
Saatavilla: https://link-springer-com.ezproxy.cc.lut.fi/book/10.1007%2F978-3-658-03964-6#about

Speeduino, code overview. Viitattu 7.9.2018
Saatavilla: https://speeduino.com/wiki/index.php/Code_overview

Speeduino Manual. 2017. Saatavilla: https://speeduino.com/Speeduino_manual.pdf

Speeduino-projektin kotisivut. Viitattu 29.3.2018
Saatavilla: https://speeduino.com/wiki/index.php/Speeduino

Stone, Richard & Ball, Jeffrey K. 2004. Automotive Engineering Fundamentals. Warrendale, SAE. ISBN: 0-7680-0987-1

STMicroelectronics. 2004 AN819 Application note, Capacitive discharge ignition. Viitattu 22.6.2018. Saatavilla: https://www.st.com/content/ccc/resource/technical/document/application_note/af/27/67/e5/fe/17/48/95/CD00003947.pdf/files/CD00003947.pdf/jcr:content/translations/en.CD00003947.pdf

14point7. Spartan Lambda Controller 2 –tuotesivu. Viitattu 29.3.2018. Saatavilla: https://www.14point7.com/products/spartan-lambda-controller-2

Liite I

OTTEITA SPEEDUINO-OHJELMISTON KOODISTA

Ohjelmiston versio: 12/2017

Koko ohjelmisto saatavilla: https://speeduino.com/wiki/index.php/Firmware_History

Luettu Arduino IDE 1.8.4 ohjelmistolla.

Saatavilla: https://www.arduino.cc/en/Main/Software

Tekstin kopioinnin vuoksi rivitys tai rivinumerointi ei ole täysin yhdenmukainen alkuperäisen tekstin kanssa.

**Sensorien luku**

Pääsilmukassa:

speeduino.ino

```
1.  //***Perform sensor reads*** //----------------------------------------------------------------
    ----------------------------------------
2.  readMAP();
3.  if (BIT_CHECK(LOOP_TIMER, BIT_TIMER_15HZ)) //Every 32 loops
4.  {
5.      BIT_CLEAR(TIMER_mask, BIT_TIMER_15HZ);
6.      readTPS(); //TPS reading to be performed every 32 loops (any faster and it can upset th
    e TPSdot sampling time)
```

****

speeduino.ino

```
1.  } //The IAT and CLT readings can be done less frequently (4 times per second)
2.  if (BIT_CHECK(LOOP_TIMER, BIT_TIMER_4HZ)) {
3.      BIT_CLEAR(TIMER_mask, BIT_TIMER_4HZ);
4.      readCLT();
5.      readIAT();
6.      readO2();
7.      readBat();
```

****

Sensors.ino:

```
1.  static inline void instanteneousMAPReading() {
```

```
    unsigned int tempReading; //Instantaneous MAP readings
    #
    if defined(ANALOG_ISR_MAP) tempReading = AnChannel[pinMAP - A0];#
    else tempReading = analogRead(pinMAP);
    tempReading = analogRead(pinMAP);#
    endif //Error checking
    if ((tempReading >= VALID_MAP_MAX) || (tempReading <= VALID_MAP_MIN)) {
        mapErrorCount += 1;
    } else {
        mapErrorCount = 0;
    } //During startup a call is made here to get the baro reading. In this case, we can't apply the ADC filter
    if (initialisationComplete == true) {
        currentStatus.mapADC = ADC_FILTER(tempReading, ADCFILTER_MAP, currentStatus.mapADC);
    } //Very weak filter
    else {
        currentStatus.mapADC = tempReading;
    } //Baro reading (No filter)
    currentStatus.MAP = fastMap10Bit(currentStatus.mapADC, configPage1.mapMin, configPage1.mapMax); //Get the current MAP value
    if (currentStatus.MAP < 0) {
        currentStatus.MAP = 0;
    } //Sanity check
}
static inline void readMAP() {
        unsigned int tempReading; //MAP Sampling system
        switch (configPage1.mapSample) {
            case 0: //Instantaneous MAP readings
                instanteneousMAPReading();
                break;
    }
}
```

****

```
void readTPS() {
    currentStatus.TPSlast = currentStatus.TPS;
    currentStatus.TPSlast_time = currentStatus.TPS_time;#
    if defined(ANALOG_ISR) byte tempTPS = fastMap1023toX(AnChannel[pinTPS - A0], 255); //Get the current raw TPS ADC value and map it into a byte
    #
    else analogRead(pinTPS);
    byte tempTPS = fastMap1023toX(analogRead(pinTPS), 255); //Get the current raw TPS ADC value and map it into a byte
    #
    endif
currentStatus.tpsADC = ADC_FILTER(tempTPS, ADCFILTER_TPS, currentStatus.tpsADC); //Check that the ADC values fall within the min and max ranges (Should always be the case, but noise can cause these to fluctuate outside the defined range).
    byte tempADC = currentStatus.tpsADC; //The tempADC value is used in order to allow TunerStudio to recover and redo the TPS calibration if this somehow gets corrupted
    if (currentStatus.tpsADC < configPage1.tpsMin) {
        tempADC = configPage1.tpsMin;
    } else if (currentStatus.tpsADC > configPage1.tpsMax) {
        tempADC = configPage1.tpsMax;
    }
    currentStatus.TPS = map(tempADC, configPage1.tpsMin, configPage1.tpsMax, 0, 100); //Take the raw TPS ADC value and convert it into a TPS% based on the calibrated values
    currentStatus.TPS_time = currentLoopTime;
}
```

*****

```
void readCLT() {
    unsigned int tempReading;#
    if defined(ANALOG_ISR) tempReading = fastMap1023toX(AnChannel[pinCLT - A0], 511); //Get the current raw CLT value
    #
    else tempReading = analogRead(pinCLT);
    tempReading = fastMap1023toX(analogRead(pinCLT), 511); //Get the current raw CLT value

    #
    endif currentStatus.cltADC = ADC_FILTER(tempReading, ADCFILTER_CLT, currentStatus.cltADC);
    currentStatus.coolant = cltCalibrationTable[currentStatus.cltADC] - CALIBRATION_TEMPERATURE_OFFSET; //Temperature calibration values are stored as positive bytes. We subtract 40 from them to allow for negative temperatures
}
void readIAT() {
    unsigned int tempReading;#
    if defined(ANALOG_ISR) tempReading = fastMap1023toX(AnChannel[pinIAT - A0], 511); //Get the current raw IAT value
    #
    else tempReading = analogRead(pinIAT);
    tempReading = fastMap1023toX(analogRead(pinIAT), 511); //Get the current raw IAT value

    #
    endif currentStatus.iatADC = ADC_FILTER(tempReading, ADCFILTER_IAT, currentStatus.iatADC);
    currentStatus.IAT = iatCalibrationTable[currentStatus.iatADC] - CALIBRATION_TEMPERATURE_OFFSET;
}
```

*****

```
void readO2() {
        unsigned int tempReading;#
        if defined(ANALOG_ISR) tempReading = fastMap1023toX(AnChannel[pinO2 - A0], 511); //Get the current O2 value.
        #
        else tempReading = analogRead(pinO2);
        tempReading = fastMap1023toX(analogRead(pinO2), 511); //Get the current O2 value.
        #
        endif currentStatus.O2ADC = ADC_FILTER(tempReading, ADCFILTER_O2, currentStatus.O2ADC);
        currentStatus.O2 = o2CalibrationTable[currentStatus.O2ADC];
    }

void readBat() {
    unsigned int tempReading;#
    if defined(ANALOG_ISR) tempReading = fastMap1023toX(AnChannel[pinBat - A0], 245); //Get the current raw Battery value. Permissible values are from 0v to 24.5v (245)
    #
    else tempReading = analogRead(pinBat);
    tempReading = fastMap1023toX(analogRead(pinBat), 245); //Get the current raw Battery value. Permissible values are from 0v to 24.5v (245)
    #
    endif currentStatus.battery10 = ADC_FILTER(tempReading, ADCFILTER_BAT, currentStatus.battery10);
```

```
}
```

*****

**Decoderit:**

(util.ino, ennen pääsilmukkaa, ohjaa triggers/getrpm/jne. missing tooth versioihin)

```
void initialiseTriggers() {
    byte triggerInterrupt = 0; // By default, use the first interrupt
    byte triggerInterrupt2 = 1;#
    if defined(CORE_AVR) switch (pinTrigger) { //Arduino Mega 2560 mapping
        case 2:
            triggerInterrupt = 0;
            break;
        case 3:
            triggerInterrupt = 1;
            break;
        case 18:
            triggerInterrupt = 5;
            break;
        case 19:
            triggerInterrupt = 4;
            break;
        case 20:
            triggerInterrupt = 3;
            break;
        case 21:
            triggerInterrupt = 2;
            break;
        default:
            triggerInterrupt = 0;
            break; //This should NEVER happen
    }#
    else triggerInterrupt = pinTrigger;#
    endif#
    if defined(CORE_AVR) switch (pinTrigger2) { //Arduino Mega 2560 mapping
        case 2:
            triggerInterrupt2 = 0;
            break;
        case 3:
            triggerInterrupt2 = 1;
            break;
        case 18:
            triggerInterrupt2 = 5;
            break;
        case 19:
            triggerInterrupt2 = 4;
            break;
        case 20:
            triggerInterrupt2 = 3;
            break;
        case 21:
            triggerInterrupt2 = 2;
            break;
        default:
            triggerInterrupt2 = 0;
            break; //This should NEVER happen
    }#
    else triggerInterrupt2 = pinTrigger2;#
```

```
    endif pinMode(pinTrigger, INPUT);
    pinMode(pinTrigger2, INPUT);
    pinMode(pinTrigger3, INPUT); //digitalWrite(pinTrigger, HIGH);
    detachInterrupt(triggerInterrupt);
    detachInterrupt(triggerInterrupt2); //Set the trigger function based on the decoder in the config
    switch (configPage2.TrigPattern) {
        case 0: //Missing tooth decoder
            triggerSetup_missingTooth();
            trigger = triggerPri_missingTooth;
            triggerSecondary = triggerSec_missingTooth;
            getRPM = getRPM_missingTooth;
            getCrankAngle = getCrankAngle_missingTooth;
            triggerSetEndTeeth = triggerSetEndTeeth_missingTooth;
            if (configPage2.TrigEdge == 0) {
                attachInterrupt(triggerInterrupt, trigger, RISING);
            } // Attach the crank trigger wheel interrupt (Hall sensor drags to ground when triggering)
            else {
                attachInterrupt(triggerInterrupt, trigger, FALLING);
            }
            if (configPage2.TrigEdgeSec == 0) {
                attachInterrupt(triggerInterrupt2, triggerSec_missingTooth, RISING);
            } else {
                attachInterrupt(triggerInterrupt2, triggerSec_missingTooth, FALLING);
            }
            break;

              ...

        default:
            trigger = triggerPri_missingTooth;
            getRPM = getRPM_missingTooth;
            getCrankAngle = getCrankAngle_missingTooth;
            if (configPage2.TrigEdge == 0) {
                attachInterrupt(triggerInterrupt, trigger, RISING);
            } // Attach the crank trigger wheel interrupt (Hall sensor drags to ground when triggering)
            else {
                attachInterrupt(triggerInterrupt, trigger, FALLING);
            }
            break;
    }
}
```

*****

## Kun näkee hampaan:

```
Each decoder must have the following 4 functions(Where xxxx is the decoder name):
* triggerSetup_xxx - Called once from within setup() and configures any required variables
* triggerPri_xxxx - Called each time the primary(No.1) crank / cam signal is triggered(Called as an interrupt, so variables must be declared volatile)
* triggerSec_xxxx - Called each time the secondary(No.2) crank / cam signal is triggered(Called as an interrupt, so variables must be declared volatile)
* getRPM_xxxx - Returns the current RPM, as calculated by the decoder
* getCrankAngle_xxxx - Returns the current crank angle, as calculated b the decoder
```

*****

Puuttuva hammas decoder:

decoders.ino

```
1.  /*Name: Missing tooth wheelDesc: A multi-
    tooth wheel with one of more 'missing' teeth. The first tooth after the missing one is cons
    idered number 1 and isthe basis for the trigger angleNote: This does not currently support
    dual wheel (ie missing tooth + single tooth on cam)*/
2.  void triggerSetup_missingTooth() {
3.      triggerToothAngle = 360 / configPage2.triggerTeeth; //The number of degrees that passes
     from tooth to tooth
4.      if (configPage2.TrigSpeed == 1) {
5.          triggerToothAngle = 720 / configPage2.triggerTeeth;
6.      } //Account for cam speed missing tooth
7.      triggerActualTeeth = configPage2.triggerTeeth - configPage2.triggerMissingTeeth; //The
    number of physical teeth on the wheel. Doing this here saves us a calculation each time in
    the interrupt
8.      triggerFilterTime = (int)(1000000 / (MAX_RPM / 60 * configPage2.triggerTeeth)); //Trigg
    er filter time is the shortest possible time (in uS) that there can be between crank teeth
    (ie at max RPM). Any pulses that occur faster than this time will be disgarded as noise
9.      secondDerivEnabled = false;
10.     decoderIsSequential = false;
11.     checkSyncToothCount = (configPage2.triggerTeeth) >> 1; //50% of the total teeth.
12.     toothLastMinusOneToothTime = 0;
13.     toothCurrentCount = 0;
14.     toothOneTime = 0;
15.     toothOneMinusOneTime = 0;
16.     MAX_STALL_TIME = (3333 UL * triggerToothAngle * (configPage2.triggerMissingTeeth + 1));
     //Minimum 50rpm. (3333uS is the time per degree at 50rpm)
17. }
18. void triggerPri_missingTooth() {
19.     curTime = micros();
20.     curGap = curTime - toothLastToothTime;
21.     if (curGap >= triggerFilterTime) //Pulses should never be less than triggerFilterTime,
    so if they are it means a false trigger. (A 36-
    1 wheel at 8000pm will have triggers approx. every 200uS)
22.     {
23.         toothCurrentCount++; //Increment the tooth counter
24.         addToothLogEntry(curGap); //if(toothCurrentCount > checkSyncToothCount || currentSt
    atus.hasSync == false)
25.         { //Begin the missing tooth detection //If the time between the current tooth and t
    he last is greater than 1.5x the time between the last tooth and the tooth before that, we
    make the assertion that we must be at the first tooth after the gap
26.             if (configPage2.triggerMissingTeeth == 1) {
27.                 targetGap = (3 * (toothLastToothTime - toothLastMinusOneToothTime)) >> 1;
28.             } //Multiply by 1.5 (Checks for a gap 1.5x greater than the last one) (Uses bit
    shift to multiply by 3 then divide by 2. Much faster than multiplying by 1.5)
29.             else {
30.                 targetGap = ((toothLastToothTime - toothLastMinusOneToothTime)) * 2;
31.             } //Multiply by 2 (Checks for a gap 2x greater than the last one)
32.             if ((toothLastToothTime == 0) || (toothLastMinusOneToothTime == 0)) {
33.                 curGap = 0;
34.             }
35.             if ((curGap > targetGap) || (toothCurrentCount > triggerActualTeeth)) {
36.                 if (toothCurrentCount < (triggerActualTeeth) && (currentStatus.hasSync == t
    rue)) {
37.                     currentStatus.hasSync = false;
38.                 } //This occurs when we're at tooth #1, but haven't seen all the other teet
    h. This indicates a signal issue so we flag lost sync so this will attempt to resync on the
```

```
    next revolution. //This is to handle a special case on startup where sync can be obtained and the system immediately thinks the revs have jumped: //else if (currentStatus.hasSync == false && toothCurrentCount < checkSyncToothCount ) { triggerFilterTime = 0; }
39.                 else {
40.                     toothCurrentCount = 1;
41.                     revolutionOne = !revolutionOne; //Flip sequential revolution tracker
42.                     toothOneMinusOneTime = toothOneTime;
43.                     toothOneTime = curTime;
44.                     currentStatus.hasSync = true;
45.                     currentStatus.startRevolutions++; //Counter
46.                     triggerFilterTime = 0; //This is used to prevent a condition where serious intermitent signals (Eg someone furiously plugging the sensor wire in and out) can leave the filter in an unrecoverable state
47.                     toothLastMinusOneToothTime = toothLastToothTime;
48.                     toothLastToothTime = curTime;
49.                     triggerToothAngleIsCorrect = false; //The tooth angle is double at this point
50.                 }
51.             } else { //Filter can only be recalc'd for the regular teeth, not the missing one.
52.                 setFilter(curGap);
53.                 toothLastMinusOneToothTime = toothLastToothTime;
54.                 toothLastToothTime = curTime;
55.                 triggerToothAngleIsCorrect = true;
56.             }
57.         } //EXPERIMENTAL!
58.         if (configPage1.perToothIgn == true) {
59.             uint16_t crankAngle = ((toothCurrentCount - 1) * triggerToothAngle) + configPage2.triggerAngle;
60.             doPerToothTiming(crankAngle);
61.         }
62.     }
63. }
64. void triggerSec_missingTooth() { //TODO: This should really have filtering enabled on the secondary input.
65.     revolutionOne = 1;
66. }
67. uint16_t getRPM_missingTooth() {
68.     uint16_t tempRPM = 0;
69.     if ((currentStatus.RPM < currentStatus.crankRPM)) {
70.         if (toothCurrentCount != 1) {
71.             if (configPage2.TrigSpeed == 1) {
72.                 tempRPM = crankingGetRPM(configPage2.triggerTeeth / 2);
73.             } //Account for cam speed
74.             else {
75.                 tempRPM = crankingGetRPM(configPage2.triggerTeeth);
76.             }
77.         } else {
78.             tempRPM = currentStatus.RPM;
79.         } //Can't do per tooth RPM if we're at tooth #1 as the missing tooth messes the calculation
80.     } else {
81.         if (configPage2.TrigSpeed == 1) {
82.             tempRPM = (stdGetRPM() * 2);
83.         } //Account for cam speed
84.         else {
85.             tempRPM = stdGetRPM();
86.         }
87.     }
88.     return tempRPM;
89. }
90. int getCrankAngle_missingTooth(int timePerDegree) { //This is the current angle ATDC the engine is at. This is the last known position based on what tooth was last 'seen'. It is only accurate to the resolution of the trigger wheel (Eg 36-1 is 10 degrees)
91.     unsigned long tempToothLastToothTime;
```

```
92.     int tempToothCurrentCount;
93.     bool tempRevolutionOne; //Grab some variables that are used in the trigger code and ass
    ign them to temp variables.
94.     noInterrupts();
95.     tempToothCurrentCount = toothCurrentCount;
96.     tempToothLastToothTime = toothLastToothTime;
97.     tempRevolutionOne = revolutionOne;
98.     interrupts();
99.     int crankAngle = ((tempToothCurrentCount - 1) * triggerToothAngle) + configPage2.trigge
    rAngle; //Number of teeth that have passed since tooth 1, multiplied by the angle each toot
    h represents, plus the angle that tooth 1 is ATDC. This gives accuracy only to the nearest
    tooth. //Estimate the number of degrees travelled since the last tooth}
100.            long elapsedTime = (micros() - tempToothLastToothTime); //crankAngle += DIV_ROUN
    D_CLOSEST(elapsedTime, timePerDegree);
101.            if (elapsedTime < SHRT_MAX) {
102.                crankAngle += div((int) elapsedTime, timePerDegree).quot;
103.            } //This option is much faster, but only available for smaller values of elapsed
    Time
104.            else {
105.                crankAngle += ldiv(elapsedTime, timePerDegree).quot;
106.            } //crankAngle += uSToDegrees(elapsedTime); //Sequential check (simply sets whet
    her we're on the first or 2nd revoltuion of the cycle)
107.            if (tempRevolutionOne) {
108.                crankAngle += 360;
109.            }
110.            if (crankAngle >= 720) {  crankAngle -= 720;
111.          } else if (crankAngle > CRANK_ANGLE_MAX) {
112.                crankAngle -= CRANK_ANGLE_MAX;
113.            }
114.            if (crankAngle < 0) {
115.                crankAngle += CRANK_ANGLE_MAX;
116.            }
117.            return crankAngle;
118.        }
119.        void triggerSetEndTeeth_missingTooth() {
120.                ignition1EndTooth = ((ignition1EndAngle - configPage2.triggerAngle) / trigge
    rToothAngle) - 1;
121.                if (ignition1EndTooth > configPage2.triggerTeeth) {
122.                    ignition1EndTooth -= configPage2.triggerTeeth;
123.                }
124.                if (ignition1EndTooth <= 0) {
125.                    ignition1EndTooth -= configPage2.triggerTeeth;
126.                }
127.                if (ignition1EndTooth > triggerActualTeeth) {
128.                    ignition1EndTooth = triggerActualTeeth;
129.                }
130.                ignition2EndTooth = ((ignition2EndAngle - configPage2.triggerAngle) / trigge
    rToothAngle) - 1;
131.                if (ignition2EndTooth > configPage2.triggerTeeth) {
132.                    ignition2EndTooth -= configPage2.triggerTeeth;
133.                }
134.                if (ignition2EndTooth <= 0) {
135.                    ignition2EndTooth -= configPage2.triggerTeeth;
136.                }
137.                if (ignition2EndTooth > triggerActualTeeth) {
138.                    ignition2EndTooth = triggerActualTeeth;
139.                }
140.                ignition3EndTooth = ((ignition3EndAngle - configPage2.triggerAngle) / trigge
    rToothAngle) - 1;
141.                if (ignition3EndTooth > configPage2.triggerTeeth) {
142.                    ignition3EndTooth -= configPage2.triggerTeeth;
143.                }
144.                if (ignition3EndTooth <= 0) {
145.                    ignition3EndTooth -= configPage2.triggerTeeth;
146.                }
```

```
                if (ignition3EndTooth > triggerActualTeeth) {
                    ignition3EndTooth = triggerActualTeeth;
                }
                ignition4EndTooth = ((ignition4EndAngle - configPage2.triggerAngle) / triggerToothAngle) - 1;
                if (ignition4EndTooth > configPage2.triggerTeeth) {
                    ignition4EndTooth -= configPage2.triggerTeeth;
                }
                if (ignition4EndTooth <= 0) {
                    ignition4EndTooth -= configPage2.triggerTeeth;
                }
                if (ignition4EndTooth > triggerActualTeeth) {
                    ignition4EndTooth = triggerActualTeeth;
                }
```

****

decoders.ino

```
static inline void addToothLogEntry(unsigned long toothTime) { //High speed tooth logging history
        toothHistory[toothHistoryIndex] = toothTime;
        if (toothHistoryIndex == (TOOTH_LOG_BUFFER - 1)) {
            if (toothLogRead) {
                toothHistoryIndex = 0;
                BIT_CLEAR(currentStatus.status1, BIT_STATUS1_TOOTHLOG1READY);
                toothLogRead = false; //The tooth log ready bit is cleared to ensure that we only get a set of concurrent values.
            }
        } else {
            toothHistoryIndex++;
        }
    }
    /*As nearly all the decoders use a common method of determining RPM (The time the last full revolution took)A common function is simplerdegreesOver is the number of crank degrees between tooth #1s. Some patterns have a tooth #1 every crank rev, others are every cam rev.*/ //static inline uint16_t stdGetRPM(uin16_t degreesOver)
static inline uint16_t stdGetRPM() {
    uint16_t tempRPM = 0;
    if (currentStatus.hasSync == true) {
        if ((currentStatus.RPM < currentStatus.crankRPM) && (currentStatus.startRevolutions == 0)) {
            tempRPM = 0;
        } //Prevents crazy RPM spike when there has been less than 1 full revolution
        else if ((toothOneTime == 0) || (toothOneMinusOneTime == 0)) {
            tempRPM = 0;
        } else {
            noInterrupts();
            revolutionTime = (toothOneTime - toothOneMinusOneTime); //The time in uS that one revolution would take at current speed (The time tooth 1 was last seen, minus the time it was seen prior to that)
            interrupts(); //if(degreesOver == 720) { revolutionTime / 2; }
            tempRPM = (US_IN_MINUTE / revolutionTime); //Calc RPM based on last full revolution time (Faster as /)
            if (tempRPM >= MAX_RPM) {
                tempRPM = currentStatus.RPM;
            } //Sanity check
        }
    } else {
        tempRPM = 0;
```

```
    }
    return tempRPM;
}
```

****

**pääsilmukka:**

pyöriikö kone

speeduino.ino

```
previousLoopTime = currentLoopTime;
currentLoopTime = micros();
unsigned long timeToLastTooth = (currentLoopTime - toothLastToothTime);
if ((timeToLastTooth < MAX_STALL_TIME) || (toothLastToothTime > currentLoopTime)) //Check how long ago the last tooth was seen compared to now. If it was more than half a second ago then the engine is probably stopped. toothLastToothTime can be greater than currentLoopTime if a pulse occurs between getting the lastest time and doing the comparison
{
    currentStatus.RPM = currentStatus.longRPM = getRPM(); //Long RPM is included here
    FUEL_PUMP_ON();
    fuelPumpOn = true; //Not sure if this is needed.
} else { //We reach here if the time between teeth is too great. This VERY likely means the engine has stopped
    currentStatus.RPM = 0;
    currentStatus.PW1 = 0;
    currentStatus.VE = 0;
    toothLastToothTime = 0;
    toothLastSecToothTime = 0; //toothLastMinusOneToothTime = 0;
    currentStatus.hasSync = false;
    currentStatus.runSecs = 0; //Reset the counter for number of seconds running.
    secCounter = 0; //Reset our seconds counter.
    currentStatus.startRevolutions = 0;
    toothSystemCount = 0;
    secondaryToothCount = 0;
    MAPcurRev = 0;
    MAPcount = 0;
    currentStatus.rpmDOT = 0;
    AFRnextCycle = 0;
    ignitionCount = 0;
    ignitionOn = false;
    fuelOn = false;
    if (fpPrimed == true) {
        digitalWrite(pinFuelPump, LOW);
        fuelPumpOn = false;
    } //Turn off the fuel pump, but only if the priming is complete
    disableIdle(); //Turn off the idle PWM
    BIT_CLEAR(currentStatus.engine, BIT_ENGINE_CRANK); //Clear cranking bit (Can otherwise get stuck 'on' even with 0 rpm)
    BIT_CLEAR(currentStatus.engine, BIT_ENGINE_WARMUP); //Same as above except for WUE
    BIT_CLEAR(currentStatus.engine, BIT_ENGINE_RUN); //Same as above except for RUNNING status
    BIT_CLEAR(currentStatus.engine, BIT_ENGINE_ASE); //Same as above except for ASE status
    //This is a safety check. If for some reason the interrupts have got screwed up (Leading to 0rpm), this resets them. //It can possibly be run much less frequently.
    initialiseTriggers();
```

****

```
1.  //Begin the fuel calculation //Calculate an injector pulsewidth from the VE
2.  currentStatus.corrections = correctionsFuel();
3.  lastAdvance = currentStatus.advance; //Store the previous advance value
4.  if (configPage1.algorithm == LOAD_SOURCE_MAP) //Check which fuelling algorithm is being use
    d
5.  { //Speed Density
6.      currentStatus.VE = get3DTableValue( & fuelTable, currentStatus.MAP, currentStatus.RPM);
     //Perform lookup into fuel map for RPM vs MAP value
7.      currentStatus.advance = get3DTableValue( & ignitionTable, currentStatus.MAP, currentSta
    tus.RPM) - OFFSET_IGNITION; //As above, but for ignition advance
8.  } else { //Alpha-N
9.      currentStatus.VE = get3DTableValue( & fuelTable, currentStatus.TPS, currentStatus.RPM);
     //Perform lookup into fuel map for RPM vs TPS value
10.     currentStatus.advance = get3DTableValue( & ignitionTable, currentStatus.TPS, currentSta
    tus.RPM) - OFFSET_IGNITION; //As above, but for ignition advance
11. }
12. currentStatus.PW1 = PW(req_fuel_uS, currentStatus.VE, currentStatus.MAP, currentStatus.corr
    ections, inj_opentime_uS);
13. currentStatus.advance = correctionsIgn(currentStatus.advance);
```

*****

```
1.  //****************************************************** //How fast are we going? Need to
     know how long (uS) it will take to get from one tooth to the next. We then use that to est
    imate how far we are between the last tooth and the next one
2.  //We use a 1st Deriv accleration prediction, but only when there is an even spacing between
     primary sensor teeth
3.  //Any decoder that has uneven spacing has its triggerToothAngle set to 0
4.  if (secondDerivEnabled && toothHistoryIndex >= 3 && currentStatus.RPM < 2000) //toothHistor
    yIndex must be greater than or equal to 3 as we need the last 3 entries. Currently this mod
    e only runs below 3000 rpm
5.  //if(true)
6.  {
7.  //Only recalculate deltaV if the tooth has changed since last time (DeltaV stays the same u
    ntil the next tooth)
8.  //if (deltaToothCount != toothCurrentCount)
9.      {
10.         deltaToothCount = toothCurrentCount;
11.         int angle1, angle2; //These represent the crank angles that are travelled for the l
    ast 2 pulses
12.         if (configPage2.TrigPattern == 4) { //Special case for 70/110 pattern on 4g63
13.             angle2 = triggerToothAngle; //Angle 2 is the most recent
14.             if (angle2 == 70) {
15.                 angle1 = 110;
16.             } else {
17.                 angle1 = 70;
18.             }
19.         } else if (configPage2.TrigPattern == 0) { //Special case for missing tooth decoder
     where the missing tooth was one of the last 2 seen
20.             if (toothCurrentCount == 1) {
21.                 angle2 = 2 * triggerToothAngle;
22.                 angle1 = triggerToothAngle;
23.             } else if (toothCurrentCount == 2) {
24.                 angle1 = 2 * triggerToothAngle;
25.                 angle2 = triggerToothAngle;
26.             } else {
27.                 angle1 = angle2 = triggerToothAngle;
28.             }
29.         } else {
30.             angle1 = angle2 = triggerToothAngle;
```

```
        }
        long toothDeltaV = (1000000 L * angle2 / toothHistory[toothHistoryIndex]) - (100000 L * angle1 / toothHistory[toothHistoryIndex - 1]);
        long toothDeltaT = toothHistory[toothHistoryIndex]; //long timeToLastTooth = micros() - toothLastToothTime;
        rpmDelta = (toothDeltaV << 10) / (6 * toothDeltaT);
    }
    timePerDegree = ldiv(166666 L, (currentStatus.RPM + rpmDelta)).quot; //There is a small amount of rounding in this calculation, however it is less than 0.001 of a uS (Faster as ldiv than / )
} else { //If we can, attempt to get the timePerDegree by comparing the times of the last two teeth seen. This is only possible for evenly spaced teeth
    if (triggerToothAngleIsCorrect == true && toothLastToothTime > toothLastMinusOneToothTime) {
        noInterrupts();
        unsigned long tempToothLastToothTime = toothLastToothTime;
        unsigned long tempToothLastMinusOneToothTime = toothLastMinusOneToothTime;
        uint16_t tempTriggerToothAngle = triggerToothAngle;
        interrupts();
        timePerDegree = (unsigned long)(tempToothLastToothTime - tempToothLastMinusOneToothTime) / tempTriggerToothAngle;
    } else {
        long rpm_adjust = ((long)(micros() - toothOneTime) * (long) currentStatus.rpmDOT) / 1000000; //Take into account any likely accleration that has occurred since the last full revolution completed
        timePerDegree = ldiv(166666 L, currentStatus.RPM + rpm_adjust).quot; //There is a small amount of rounding in this calculation, however it is less than 0.001 of a uS (Faster as ldiv than / )
    }
} //Check that the duty cycle of the chosen pulsewidth isn't too high.
unsigned long pwLimit = percentage(configPage1.dutyLim, revolutionTime); //The pulsewidth limit is determined to be the duty cycle limit (Eg 85%) by the total time it takes to perform 1 revolution
if (CRANK_ANGLE_MAX_INJ == 720) {
    pwLimit = pwLimit * 2;
} //For sequential, the maximum pulse time is double (2 revolutions). Wouldn't work for 2 stroke... //Apply the pwLimit if staging is dsiabled and engine is not cranking
if ((!BIT_CHECK(currentStatus.engine, BIT_ENGINE_CRANK)) && configPage11.stagingEnabled == false) {
    if (currentStatus.PW1 > pwLimit) {
        currentStatus.PW1 = pwLimit;
    }
} //Calculate staging pulsewidths if used
if (configPage11.stagingEnabled == true) { //Scale the 'full' pulsewidth by each of the injector capacities
    uint32_t tempPW1 = ((unsigned long) currentStatus.PW1 * staged_req_fuel_mult_pri) / 100;
    if (configPage11.stagingMode == STAGING_MODE_TABLE) {
        uint32_t tempPW3 = ((unsigned long) currentStatus.PW1 * staged_req_fuel_mult_sec) / 100; //This is ONLY needed in in table mode. Auto mode only calculates the difference.
        byte stagingSplit = get3DTableValue( & stagingTable, currentStatus.MAP, currentStatus.RPM);
        currentStatus.PW1 = ((100 - stagingSplit) * tempPW1) / 100;
        if (stagingSplit > 0) {
            currentStatus.PW3 = (stagingSplit * tempPW3) / 100;
        } else {
            currentStatus.PW3 = 0;
        }
    } else if (configPage11.stagingMode == STAGING_MODE_AUTO) {
        currentStatus.PW1 = tempPW1; //If automatic mode, the primary injectors are used all the way up to their limit (COnfigured by the pulsewidth limit setting) //If they exceed their limit, the extra duty is passed to the secondaries
        if (tempPW1 > pwLimit) {
            uint32_t extraPW = tempPW1 - pwLimit;
            currentStatus.PW1 = pwLimit;
```

```
            currentStatus.PW3 = ((extraPW * staged_req_fuel_mult_sec) / staged_req_fuel_mult_pri) + inj_opentime_uS; //Convert the 'left over' fuel amount from primary injector scaling to secondary
        } else {
            currentStatus.PW3 = 0;
        } //If tempPW1 < pwLImit it means that the entire fuel load can be handled by the primaries. Simply set the secondaries to 0
    } //currentStatus.PW3 = 2000; //Set the 2nd channel of each stage with the same pulseWidth
    currentStatus.PW2 = currentStatus.PW1;
    currentStatus.PW4 = currentStatus.PW3;
} //If staging is off, all the pulse widths are set the same (Sequential adjustments will be made below)
else {
    currentStatus.PW2 = currentStatus.PW3 = currentStatus.PW4 = currentStatus.PW1;
} // Initial state is for all pulsewidths to be the same (This gets changed below)
```

****

Polttoainesuihkutuksen ajoitus

```

//****************************************************************************************** //BEGIN INJECTION TIMING //Determine next firing angles
if (!configPage1.indInjAng) {
    configPage1.inj4Ang = configPage1.inj3Ang = configPage1.inj2Ang = configPage1.inj1Ang;

} //Forcing all injector close angles to be the same.
int PWdivTimerPerDegree = div(currentStatus.PW1, timePerDegree).quot; //How many crank degrees the calculated PW will take at the current speed
injector1StartAngle = configPage1.inj1Ang - (PWdivTimerPerDegree); //This is a little primitive, but is based on the idea that all fuel needs to be delivered before the inlet valve opens. See http://www.extraefi.co.uk/sequential_fuel.html for more detail
if (injector1StartAngle < 0) {
    injector1StartAngle += CRANK_ANGLE_MAX_INJ;
}
if (injector1StartAngle > CRANK_ANGLE_MAX_INJ) {
    injector1StartAngle -= CRANK_ANGLE_MAX_INJ;
} //Repeat the above for each cylinder
switch (configPage1.nCylinders) { //2 cylinders
    case 2:
        injector2StartAngle = (configPage1.inj2Ang + channel2InjDegrees - (PWdivTimerPerDegree));
        if (injector2StartAngle > CRANK_ANGLE_MAX_INJ) {
            injector2StartAngle -= CRANK_ANGLE_MAX_INJ;
        }
        if (injector2StartAngle < 0) {
            injector2StartAngle += CRANK_ANGLE_MAX_INJ;
        }
        break; //3 cylinders
    case 3:
        injector2StartAngle = (configPage1.inj2Ang + channel2InjDegrees - (PWdivTimerPerDegree));
        if (injector2StartAngle > CRANK_ANGLE_MAX_INJ) {
            injector2StartAngle -= CRANK_ANGLE_MAX_INJ;
        }
        if (injector2StartAngle < 0) {
            injector2StartAngle += CRANK_ANGLE_MAX_INJ;
        }
        injector3StartAngle = (configPage1.inj3Ang + channel3InjDegrees - (PWdivTimerPerDegree));
        if (injector3StartAngle > CRANK_ANGLE_MAX_INJ) {
```

```
            injector3StartAngle -= CRANK_ANGLE_MAX_INJ;
        }
        if (injector3StartAngle < 0) {
            injector3StartAngle += CRANK_ANGLE_MAX_INJ;
        }
        break; //4 cylinders
    case 4:
        injector2StartAngle = (configPage1.inj2Ang + channel2InjDegrees - (PWdivTimerPerDegree));
        if (injector2StartAngle > CRANK_ANGLE_MAX_INJ) {
            injector2StartAngle -= CRANK_ANGLE_MAX_INJ;
        }
        if (injector2StartAngle < 0) {
            injector2StartAngle += CRANK_ANGLE_MAX_INJ;
        }
        if (configPage1.injLayout == INJ_SEQUENTIAL) {
            injector3StartAngle = (configPage1.inj3Ang + channel3InjDegrees - (PWdivTimerPerDegree));
            if (injector3StartAngle > CRANK_ANGLE_MAX_INJ) {
                injector3StartAngle -= CRANK_ANGLE_MAX_INJ;
            }
            if (injector3StartAngle < 0) {
                injector3StartAngle += CRANK_ANGLE_MAX_INJ;
            }
            injector4StartAngle = (configPage1.inj4Ang + channel4InjDegrees - (PWdivTimerPerDegree));
            if (injector4StartAngle > CRANK_ANGLE_MAX_INJ) {
                injector4StartAngle -= CRANK_ANGLE_MAX_INJ;
            }
            if (injector4StartAngle < 0) {
                injector4StartAngle += CRANK_ANGLE_MAX_INJ;
            }
            if (configPage3.fuelTrimEnabled) {
                unsigned long pw1percent = 100 + (byte) get3DTableValue( & trim1Table, currentStatus.MAP, currentStatus.RPM) - OFFSET_FUELTRIM;
                unsigned long pw2percent = 100 + (byte) get3DTableValue( & trim2Table, currentStatus.MAP, currentStatus.RPM) - OFFSET_FUELTRIM;
                unsigned long pw3percent = 100 + (byte) get3DTableValue( & trim3Table, currentStatus.MAP, currentStatus.RPM) - OFFSET_FUELTRIM;
                unsigned long pw4percent = 100 + (byte) get3DTableValue( & trim4Table, currentStatus.MAP, currentStatus.RPM) - OFFSET_FUELTRIM;
                if (pw1percent != 100) {
                    currentStatus.PW1 = (pw1percent * currentStatus.PW1) / 100;
                }
                if (pw2percent != 100) {
                    currentStatus.PW2 = (pw2percent * currentStatus.PW2) / 100;
                }
                if (pw3percent != 100) {
                    currentStatus.PW3 = (pw3percent * currentStatus.PW3) / 100;
                }
                if (pw4percent != 100) {
                    currentStatus.PW4 = (pw4percent * currentStatus.PW4) / 100;
                }
            }
        } else if (configPage11.stagingEnabled == true) {
            PWdivTimerPerDegree = div(currentStatus.PW3, timePerDegree).quot; //Need to redo this for PW3 as it will be dramatically different to PW1 when staging
            injector3StartAngle = configPage1.inj3Ang - (PWdivTimerPerDegree); //This is a little primitive, but is based on the idea that all fuel needs to be delivered before the inlet valve opens. See http://www.extraefi.co.uk/sequential_fuel.html for more detail
            if (injector3StartAngle < 0) {
                injector3StartAngle += CRANK_ANGLE_MAX_INJ;
            }
            if (injector3StartAngle > CRANK_ANGLE_MAX_INJ) {
                injector3StartAngle -= CRANK_ANGLE_MAX_INJ;
```

```
            }
            injector4StartAngle = injector3StartAngle + (CRANK_ANGLE_MAX_INJ / 2); //Phase this either 180 or 360 degrees out from inj3 (In reality this will always be 180 as you can't have sequential and staged currently)
            if (injector4StartAngle < 0) {
                injector4StartAngle += CRANK_ANGLE_MAX_INJ;
            }
            if (injector4StartAngle > CRANK_ANGLE_MAX_INJ) {
                injector4StartAngle -= CRANK_ANGLE_MAX_INJ;
            }
        }
        break; //5 cylinders
    case 5:
        injector2StartAngle = (configPage1.inj2Ang + channel2InjDegrees - (PWdivTimerPerDegree));
                if (injector2StartAngle > CRANK_ANGLE_MAX_INJ) {
                    injector2StartAngle -= CRANK_ANGLE_MAX_INJ;
                }
                injector3StartAngle = (configPage1.inj3Ang + channel3InjDegrees - (PWdivTimerPerDegree));
                if (injector3StartAngle > CRANK_ANGLE_MAX_INJ) {
                    injector3StartAngle -= CRANK_ANGLE_MAX_INJ;
                }
                injector4StartAngle = (configPage1.inj4Ang + channel4InjDegrees - (PWdivTimerPerDegree));
                if (injector4StartAngle > CRANK_ANGLE_MAX_INJ) {
                    injector4StartAngle -= CRANK_ANGLE_MAX_INJ;
                }
                injector5StartAngle = (configPage1.inj1Ang + channel5InjDegrees - (PWdivTimerPerDegree));
                if (injector5StartAngle > CRANK_ANGLE_MAX_INJ) {
                    injector5StartAngle -= CRANK_ANGLE_MAX_INJ;
                }
                break; //6 cylinders
            case 6:
                injector2StartAngle = (configPage1.inj2Ang + channel2InjDegrees - (PWdivTimerPerDegree));
                if (injector2StartAngle > CRANK_ANGLE_MAX_INJ) {
                    injector2StartAngle -= CRANK_ANGLE_MAX_INJ;
                }
                injector3StartAngle = (configPage1.inj3Ang + channel3InjDegrees - (PWdivTimerPerDegree));
                if (injector3StartAngle > CRANK_ANGLE_MAX_INJ) {
                    injector3StartAngle -= CRANK_ANGLE_MAX_INJ;
                }
                break; //8 cylinders
            case 8:
                injector2StartAngle = (configPage1.inj2Ang + channel2InjDegrees - (PWdivTimerPerDegree));
                if (injector2StartAngle > CRANK_ANGLE_MAX_INJ) {
                    injector2StartAngle -= CRANK_ANGLE_MAX_INJ;
                }
                injector3StartAngle = (configPage1.inj3Ang + channel3InjDegrees - (PWdivTimerPerDegree));
                if (injector3StartAngle > CRANK_ANGLE_MAX_INJ) {
                    injector3StartAngle -= CRANK_ANGLE_MAX_INJ;
                }
                injector4StartAngle = (configPage1.inj4Ang + channel4InjDegrees - (PWdivTimerPerDegree));
                if (injector4StartAngle > CRANK_ANGLE_MAX_INJ) {
                    injector4StartAngle -= CRANK_ANGLE_MAX_INJ;
                }
                break; //Will hit the default case on 1 cylinder or >8 cylinders. Do nothing in these cases
            default:
```

```
                break;
        }
```

*****

Sytytysennakon laskenta

```
//****************************************************************************************
****** //| BEGIN IGNITION CALCULATIONS
if (currentStatus.RPM > ((unsigned int)(configPage2.HardRevLim) * 100)) {
    BIT_SET(currentStatus.spark, BIT_SPARK_HRDLIM);
} //Hardcut RPM limit
else {
    BIT_CLEAR(currentStatus.spark, BIT_SPARK_HRDLIM);
} //Set dwell //Dwell is stored as ms * 10. ie Dwell of 4.3ms would be 43 in configPage2. T
his number therefore needs to be multiplied by 100 to get dwell in uS
if (BIT_CHECK(currentStatus.engine, BIT_ENGINE_CRANK)) {
    currentStatus.dwell = (configPage2.dwellCrank * 100);
} else {
    currentStatus.dwell = (configPage2.dwellRun * 100);
}
currentStatus.dwell = correctionsDwell(currentStatus.dwell);
int dwellAngle = uSToDegrees(currentStatus.dwell); //Convert the dwell time to dwell angle
based on the current engine speed //Calculate start angle for each channel //1 cylinder (Ev
eryone gets this)
ignition1EndAngle = CRANK_ANGLE_MAX_IGN - currentStatus.advance;
ignition1StartAngle = ignition1EndAngle - dwellAngle; // 360 - desired advance angle - numb
er of degrees the dwell will take
if (ignition1StartAngle < 0) {
    ignition1StartAngle += CRANK_ANGLE_MAX_IGN;
} //This test for more cylinders and do the same thing
switch (configPage1.nCylinders) { //2 cylinders
    case 2:
        ignition2EndAngle = channel2IgnDegrees + CRANK_ANGLE_MAX_IGN - currentStatus.advanc
e;
        ignition2StartAngle = ignition2EndAngle - dwellAngle;
        if (ignition2StartAngle > CRANK_ANGLE_MAX_IGN) {
            ignition2StartAngle -= CRANK_ANGLE_MAX_IGN;
        }
        break; //3 cylinders
    case 3:
        ignition2EndAngle = channel2IgnDegrees + CRANK_ANGLE_MAX_IGN - currentStatus.advanc
e;
        ignition2StartAngle = ignition2EndAngle - dwellAngle;
        if (ignition2StartAngle > CRANK_ANGLE_MAX_IGN) {
            ignition2StartAngle -= CRANK_ANGLE_MAX_IGN;
        }
        ignition3EndAngle = channel3IgnDegrees + CRANK_ANGLE_MAX_IGN - currentStatus.advanc
e;
        ignition3StartAngle = channel3IgnDegrees + 360 - currentStatus.advance - dwellAngle
;
        if (ignition3StartAngle > CRANK_ANGLE_MAX_IGN) {
            ignition3StartAngle -= CRANK_ANGLE_MAX_IGN;
        }
        break; //4 cylinders
    case 4:
        ignition2EndAngle = channel2IgnDegrees + CRANK_ANGLE_MAX_IGN - currentStatus.advanc
e;
        ignition2StartAngle = ignition2EndAngle - dwellAngle;
        if (ignition2StartAngle > CRANK_ANGLE_MAX_IGN) {
```

```
            ignition2StartAngle -= CRANK_ANGLE_MAX_IGN;
        }
        if (ignition2StartAngle < 0) {
            ignition2StartAngle += CRANK_ANGLE_MAX_IGN;
        }
        if (configPage2.sparkMode == IGN_MODE_SEQUENTIAL) {
            ignition3EndAngle = channel3IgnDegrees + CRANK_ANGLE_MAX_IGN - currentStatus.advance;
            ignition3StartAngle = ignition3EndAngle - dwellAngle;
            if (ignition3StartAngle > CRANK_ANGLE_MAX_IGN) {
                ignition3StartAngle -= CRANK_ANGLE_MAX_IGN;
            }
            ignition4EndAngle = channel4IgnDegrees + CRANK_ANGLE_MAX_IGN - currentStatus.advance;
            ignition4StartAngle = ignition4EndAngle - dwellAngle;
            if (ignition4StartAngle > CRANK_ANGLE_MAX_IGN) {
                ignition4StartAngle -= CRANK_ANGLE_MAX_IGN;
            }
        } else if (configPage2.sparkMode == IGN_MODE_ROTARY) {
            if (configPage11.rotaryType == ROTARY_IGN_FC) {
                byte splitDegrees = 0;
                if (configPage1.algorithm == LOAD_SOURCE_MAP) {
                    splitDegrees = table2D_getValue( & rotarySplitTable, currentStatus.MAP / 2);
                } else {
                    splitDegrees = table2D_getValue( & rotarySplitTable, currentStatus.TPS / 2);
                } //The trailing angles are set relative to the leading ones
                ignition3EndAngle = ignition1EndAngle + splitDegrees;
                ignition3StartAngle = ignition3EndAngle - dwellAngle;
                if (ignition3StartAngle > CRANK_ANGLE_MAX_IGN) {
                    ignition3StartAngle -= CRANK_ANGLE_MAX_IGN;
                }
                if (ignition3StartAngle < 0) {
                    ignition3StartAngle += CRANK_ANGLE_MAX_IGN;
                }
                ignition4EndAngle = ignition2EndAngle + splitDegrees;
                ignition4StartAngle = ignition4EndAngle - dwellAngle;
                if (ignition4StartAngle > CRANK_ANGLE_MAX_IGN) {
                    ignition4StartAngle -= CRANK_ANGLE_MAX_IGN;
                }
                if (ignition4StartAngle < 0) {
                    ignition4StartAngle += CRANK_ANGLE_MAX_IGN;
                }
            }
        }
        break; //5 cylinders
    case 5:
        ignition2EndAngle = channel2IgnDegrees + CRANK_ANGLE_MAX_IGN - currentStatus.advance;
        ignition2StartAngle = ignition2EndAngle - dwellAngle;
        if (ignition2StartAngle > CRANK_ANGLE_MAX_IGN) {
            ignition2StartAngle -= CRANK_ANGLE_MAX_IGN;
        }
        if (ignition2StartAngle < 0) {
            ignition2StartAngle += CRANK_ANGLE_MAX_IGN;
        }
        ignition3EndAngle = channel3IgnDegrees + CRANK_ANGLE_MAX_IGN - currentStatus.advance;
        ignition3StartAngle = ignition3EndAngle - dwellAngle;
        if (ignition3StartAngle > CRANK_ANGLE_MAX_IGN) {
            ignition3StartAngle -= CRANK_ANGLE_MAX_IGN;
               }
               ignition4EndAngle = channel4IgnDegrees + CRANK_ANGLE_MAX_IGN - currentStatus.advance;
```

```
                ignition4StartAngle = ignition4EndAngle - dwellAngle;
                if (ignition4StartAngle > CRANK_ANGLE_MAX_IGN) {
                    ignition4StartAngle -= CRANK_ANGLE_MAX_IGN;
                }
                ignition5StartAngle = channel5IgnDegrees + CRANK_ANGLE_MAX - currentStatus.advance - dwellAngle;
                if (ignition5StartAngle > CRANK_ANGLE_MAX_IGN) {
                    ignition5StartAngle -= CRANK_ANGLE_MAX_IGN;
                }
                break; //6 cylinders
            case 6:
                ignition2EndAngle = channel2IgnDegrees + CRANK_ANGLE_MAX_IGN - currentStatus.advance;
                ignition2StartAngle = ignition2EndAngle - dwellAngle;
                if (ignition2StartAngle > CRANK_ANGLE_MAX_IGN) {
                    ignition2StartAngle -= CRANK_ANGLE_MAX_IGN;
                }
                ignition3EndAngle = channel3IgnDegrees + CRANK_ANGLE_MAX_IGN - currentStatus.advance;
                ignition3StartAngle = ignition3EndAngle - dwellAngle;
                if (ignition3StartAngle > CRANK_ANGLE_MAX_IGN) {
                    ignition3StartAngle -= CRANK_ANGLE_MAX_IGN;
                }
                break; //8 cylinders
            case 8:
                ignition2EndAngle = channel2IgnDegrees + CRANK_ANGLE_MAX_IGN - currentStatus.advance;
                ignition2StartAngle = ignition2EndAngle - dwellAngle;
                if (ignition2StartAngle > CRANK_ANGLE_MAX_IGN) {
                    ignition2StartAngle -= CRANK_ANGLE_MAX_IGN;
                }
                ignition3EndAngle = channel3IgnDegrees + CRANK_ANGLE_MAX_IGN - currentStatus.advance;
                ignition3StartAngle = ignition3EndAngle - dwellAngle;
                if (ignition3StartAngle > CRANK_ANGLE_MAX_IGN) {
                    ignition3StartAngle -= CRANK_ANGLE_MAX_IGN;
                }
                ignition4EndAngle = channel4IgnDegrees + CRANK_ANGLE_MAX_IGN - currentStatus.advance;
                ignition4StartAngle = ignition4EndAngle - dwellAngle;
                if (ignition4StartAngle > CRANK_ANGLE_MAX_IGN) {
                    ignition4StartAngle -= CRANK_ANGLE_MAX_IGN;
                }
                break; //Will hit the default case on 1 cylinder or >8 cylinders. Do nothing in these cases
            default:
                break;
        } //If ignition timing is being tracked per tooth, perform the calcs to get the end teeth
        //This only needs to be run if the advance figure has changed, otherwise the end teeth will still be the same
        if ((configPage1.perToothIgn == true) && (lastAdvance != currentStatus.advance)) {
            triggerSetEndTeeth();
        }
```

******

Aikataulutus polttoaineen suihkutukselle

```
//****************************************************************************************** //| BEGIN FUEL SCHEDULES
```

```
2.  //Finally calculate the time (uS) until we reach the firing angles and set the schedules //
    We only need to set the shcedule if we're BEFORE the open angle
3.  //This may potentially be called a number of times as we get closer and closer to the openi
    ng time
4.  //Determine the current crank angle
5.  int crankAngle = getCrankAngle(timePerDegree);
6.  if (crankAngle > CRANK_ANGLE_MAX_INJ) {
7.      crankAngle -= 360;
8.  }
9.  if (fuelOn && !BIT_CHECK(currentStatus.status1, BIT_STATUS1_BOOSTCUT)) {
10.     if (currentStatus.PW1 >= inj_opentime_uS) {
11.         if ((injector1StartAngle <= crankAngle) && (fuelSchedule1.Status == RUNNING)) {
12.             injector1StartAngle += CRANK_ANGLE_MAX_INJ;
13.         }
14.         if (injector1StartAngle > crankAngle) {
15.             setFuelSchedule1(((unsigned long)(injector1StartAngle - crankAngle) * (unsigned
     long) timePerDegree), (unsigned long) currentStatus.PW1);
16.         }
17.     }
18.     /*-----------------------------------------------------------------------------------
    ---
    -        | A Note on tempCrankAngle and tempStartAngle:        |   The use of tempCrankAngl
    e/tempStartAngle is described below. It is then used in the same way for channels 2, 3 and
    4 on both injectors and ignition        |   Essentially, these 2 variables are used to real
    ign the current crank angle and the desired start angle around 0 degrees for the given cyli
    nder/output        |   Eg: If cylinder 2 TDC is 180 degrees after cylinder 1 (Eg a standard
     4 cylidner engine), then tempCrankAngle is 180* less than the current crank angle and
        |       tempStartAngle is the desired open time less 180*. Thus the cylinder is being tr
    eated relative to its own TDC, regardless of its offset        |        |   This is done to
     avoid problems with very short of very long times until tempStartAngle.        |   This wi
    ll very likely need to be rewritten when sequential is enabled        |-------------------
    ------------------------------------------------------------------        */
19.     if ((channel2InjEnabled) && (currentStatus.PW2 >= inj_opentime_uS)) {
20.         tempCrankAngle = crankAngle - channel2InjDegrees;
21.         if (tempCrankAngle < 0) {
22.             tempCrankAngle += CRANK_ANGLE_MAX_INJ;
23.         }
24.         tempStartAngle = injector2StartAngle - channel2InjDegrees;
25.         if (tempStartAngle < 0) {
26.             tempStartAngle += CRANK_ANGLE_MAX_INJ;
27.         }
28.         if ((tempStartAngle <= tempCrankAngle) && (fuelSchedule2.Status == RUNNING)) {
29.             tempStartAngle += CRANK_ANGLE_MAX_INJ;
30.         }
31.         if (tempStartAngle > tempCrankAngle) {
32.             setFuelSchedule2(((unsigned long)(tempStartAngle - tempCrankAngle) * (unsigned
    long) timePerDegree), (unsigned long) currentStatus.PW2);
33.         }
34.     }
35.     if ((channel3InjEnabled) && (currentStatus.PW3 >= inj_opentime_uS)) {
36.         tempCrankAngle = crankAngle - channel3InjDegrees;
37.         if (tempCrankAngle < 0) {
38.             tempCrankAngle += CRANK_ANGLE_MAX_INJ;
39.         }
40.         tempStartAngle = injector3StartAngle - channel3InjDegrees;
41.         if (tempStartAngle < 0) {
42.             tempStartAngle += CRANK_ANGLE_MAX_INJ;
43.         }
44.         if ((tempStartAngle <= tempCrankAngle) && (fuelSchedule3.Status == RUNNING)) {
45.             tempStartAngle += CRANK_ANGLE_MAX_INJ;
46.         }
47.         if (tempStartAngle > tempCrankAngle) {
48.             setFuelSchedule3(((unsigned long)(tempStartAngle - tempCrankAngle) * (unsigned
    long) timePerDegree), (unsigned long) currentStatus.PW3);
49.         }
```

```
    }
    if ((channel4InjEnabled) && (currentStatus.PW4 >= inj_opentime_uS)) {
        tempCrankAngle = crankAngle - channel4InjDegrees;
        if (tempCrankAngle < 0) {
            tempCrankAngle += CRANK_ANGLE_MAX_INJ;
        }
        tempStartAngle = injector4StartAngle - channel4InjDegrees;
        if (tempStartAngle < 0) {
            tempStartAngle += CRANK_ANGLE_MAX_INJ;
        }
        if ((tempStartAngle <= tempCrankAngle) && (fuelSchedule4.Status == RUNNING)) {
            tempStartAngle += CRANK_ANGLE_MAX_INJ;
        }
        if (tempStartAngle > tempCrankAngle) {
            setFuelSchedule4(((unsigned long)(tempStartAngle - tempCrankAngle) * (unsigned long) timePerDegree), (unsigned long) currentStatus.PW4);
        }
    }
    if (channel5InjEnabled) {
        tempCrankAngle = crankAngle - channel5InjDegrees;
        if (tempCrankAngle < 0) {
            tempCrankAngle += CRANK_ANGLE_MAX_INJ;
        }
        tempStartAngle = injector5StartAngle - channel5InjDegrees;
        if (tempStartAngle < 0) {
            tempStartAngle += CRANK_ANGLE_MAX_INJ;
        }
        if (tempStartAngle <= tempCrankAngle && fuelSchedule5.schedulesSet == 0) {
            tempStartAngle += CRANK_ANGLE_MAX_INJ;
        }
        if (tempStartAngle > tempCrankAngle) { //Note the hacky use of fuel schedule 3 below
            /*            setFuelSchedule3(openInjector3and5,                     ((unsigned long)(tempStartAngle - tempCrankAngle) * (unsigned long)timePerDegree),
     (unsigned long)currentStatus.PW1,                     closeInjector3and5
       );*/
            setFuelSchedule3(((unsigned long)(tempStartAngle - tempCrankAngle) * (unsigned long) timePerDegree), (unsigned long) currentStatus.PW1);
        }
    }
}
*****
```

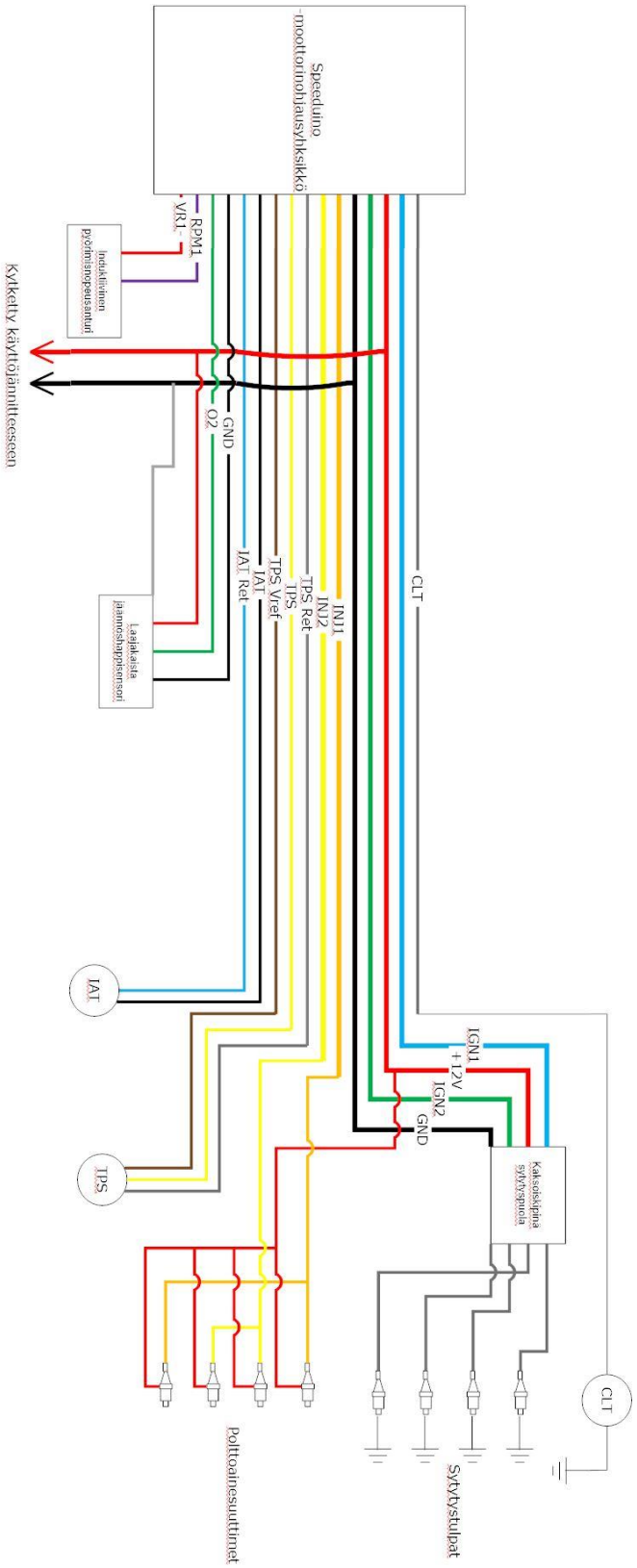
Aikataulutus sytytysajankohdalle

```
//****************************************************************************************** //| BEGIN IGNITION SCHEDULES
//Likewise for the ignition
//fixedCrankingOverride is used to extend the dwell during cranking so that the decoder can trigger the spark upon seeing a certain tooth. Currently only available on the basic distributor and 4g63 decoders.
if (configPage2.ignCranklock && BIT_CHECK(currentStatus.engine, BIT_ENGINE_CRANK) && (decoderHasFixedCrankingTiming == true)) {
    fixedCrankingOverride = currentStatus.dwell * 3;
} else {
    fixedCrankingOverride = 0;
} //Perform an initial check to see if the ignition is turned on (Ignition only turns on after a preset number of cranking revolutions and: //Check for any of the hard cut rev limits being on
if (currentStatus.launchingHard || BIT_CHECK(currentStatus.spark, BIT_SPARK_BOOSTCUT) || BIT_CHECK(currentStatus.spark, BIT_SPARK_HRDLIM) || currentStatus.flatShiftingHard) {
    if (configPage1.hardCutType == HARD_CUT_FULL) {
        ignitionOn = false;
    } else {
```

```
        curRollingCut = ((currentStatus.startRevolutions / 2) % maxIgnOutputs) + 1;
    } //Rolls through each of the active ignition channels based on how many revolutions have taken place
} else {
    curRollingCut = 0;
} //Disables the rolling hard cut //if(ignitionOn && !currentStatus.launchingHard && !BIT_CHECK(currentStatus.spark, BIT_SPARK_BOOSTCUT) && !BIT_CHECK(currentStatus.spark, BIT_SPARK_HRDLIM) && !currentStatus.flatShiftingHard)
if (ignitionOn) { //Refresh the current crank angle info //ignition1StartAngle = 335;
    crankAngle = getCrankAngle(timePerDegree); //Refresh with the latest crank angle
    if (crankAngle > CRANK_ANGLE_MAX_IGN) {
        crankAngle -= 360;
    }
    if ((ignition1StartAngle > crankAngle) && (curRollingCut != 1)) {
        /*            long some_time = ((unsigned long)(ignition1StartAngle - crankAngle) * (unsigned long)timePerDegree);            long newRPM = (long)(some_time * currentStatus.rpmDOT) / 1000000L;            newRPM = currentStatus.RPM + (newRPM/2);            unsigned long timePerDegree_1 = ldiv( 166666L, newRPM).quot;            unsigned long timeout = (unsigned long)(ignition1StartAngle - crankAngle) * 282UL;            */
        if (ignitionSchedule1.Status != RUNNING) {
            setIgnitionSchedule1(ign1StartFunction, ///((unsigned long)(ignition1StartAngle - crankAngle) * (unsigned long)timePerDegree),
                degreesToUS((ignition1StartAngle - crankAngle)), currentStatus.dwell + fixedCrankingOverride, ///((unsigned long)((unsigned long)currentStatus.dwell* currentStatus.RPM) / newRPM) + fixedCrankingOverride,
                ign1EndFunction);
        }
    }
    /*        if( (ignitionSchedule1.Status == RUNNING) && (ignition1EndAngle > crankAngle) && configPage2.StgCycles == 0)        {          unsigned long uSToEnd = 0;          ONLY ONE OF THE BELOW SHOULD BE USED (PROBABLY THE FIRST):          *********          if(ignition1EndAngle > crankAngle) { uSToEnd = fastDegreesToUS( (ignition1EndAngle - crankAngle) ); }          else { uSToEnd = fastDegreesToUS( (360 + ignition1EndAngle - crankAngle) ); }        *********          uSToEnd = ((ignition1EndAngle - crankAngle) * (toothLastToothTime - toothLastMinusOneToothTime)) / triggerToothAngle;          *********          refreshIgnitionSchedule1( uSToEnd + fixedCrankingOverride );        }        */
    tempCrankAngle = crankAngle - channel2IgnDegrees;
    if (tempCrankAngle < 0) {
        tempCrankAngle += CRANK_ANGLE_MAX_IGN;
    }
    tempStartAngle = ignition2StartAngle - channel2IgnDegrees;
    if (tempStartAngle < 0) {
        tempStartAngle += CRANK_ANGLE_MAX_IGN;
    } {
        unsigned long ignition2StartTime = 0;
        if (tempStartAngle > tempCrankAngle) {
            ignition2StartTime = degreesToUS((tempStartAngle - tempCrankAngle));
        } //else if (tempStartAngle < tempCrankAngle) { ignition2StartTime = ((long)(360 - tempCrankAngle + tempStartAngle) * (long)timePerDegree); }
        else {
            ignition2StartTime = 0;
        }
        if ((ignition2StartTime > 0) && (curRollingCut != 2)) {
            setIgnitionSchedule2(ign2StartFunction, ignition2StartTime, currentStatus.dwell + fixedCrankingOverride, ign2EndFunction);
        }
    }
    tempCrankAngle = crankAngle - channel3IgnDegrees;
    if (tempCrankAngle < 0) {
        tempCrankAngle += CRANK_ANGLE_MAX_IGN;
    }
    tempStartAngle = ignition3StartAngle - channel3IgnDegrees;
    if (tempStartAngle < 0) {
        tempStartAngle += CRANK_ANGLE_MAX_IGN;
    } //if (tempStartAngle > tempCrankAngle)
```

```
    {
        long ignition3StartTime = 0;
        if (tempStartAngle > tempCrankAngle) {
            ignition3StartTime = degreesToUS((tempStartAngle - tempCrankAngle));
        } //else if (tempStartAngle < tempCrankAngle) { ignition4StartTime = ((long)(360 - tempCrankAngle + tempStartAngle) * (long)timePerDegree); }
        else {
            ignition3StartTime = 0;
        }
        if ((ignition3StartTime > 0) && (curRollingCut != 3)) {
            setIgnitionSchedule3(ign3StartFunction, ignition3StartTime, currentStatus.dwell + fixedCrankingOverride, ign3EndFunction);
        }
    }
    tempCrankAngle = crankAngle - channel4IgnDegrees;
    if (tempCrankAngle < 0) {
        tempCrankAngle += CRANK_ANGLE_MAX_IGN;
    }
    tempStartAngle = ignition4StartAngle - channel4IgnDegrees;
    if (tempStartAngle < 0) {
        tempStartAngle += CRANK_ANGLE_MAX_IGN;
    } //if (tempStartAngle > tempCrankAngle)
    {
        long ignition4StartTime = 0;
        if (tempStartAngle > tempCrankAngle) {
            ignition4StartTime = degreesToUS((tempStartAngle - tempCrankAngle));
        } //else if (tempStartAngle < tempCrankAngle) { ignition4StartTime = ((long)(360 - tempCrankAngle + tempStartAngle) * (long)timePerDegree); }
        else {
            ignition4StartTime = 0;
        }
        if ((ignition4StartTime > 0) && (curRollingCut != 4)) {
            setIgnitionSchedule4(ign4StartFunction, ignition4StartTime, currentStatus.dwell + fixedCrankingOverride, ign4EndFunction);
        }
    }
    tempCrankAngle = crankAngle - channel5IgnDegrees;
    if (tempCrankAngle < 0) {
        tempCrankAngle += CRANK_ANGLE_MAX_IGN;
    }
    tempStartAngle = ignition5StartAngle - channel5IgnDegrees;
    if (tempStartAngle < 0) {
        tempStartAngle += CRANK_ANGLE_MAX_IGN;
    } //if (tempStartAngle > tempCrankAngle)
    {
                long ignition5StartTime = 0;
                if (tempStartAngle > tempCrankAngle) {
                    ignition5StartTime = degreesToUS((tempStartAngle - tempCrankAngle));
                } //else if (tempStartAngle < tempCrankAngle) { ignition4StartTime = ((long)(360 - tempCrankAngle + tempStartAngle) * (long)timePerDegree); }
                else {
                    ignition5StartTime = 0;
                }
                if (ignition5StartTime > 0) {
                    setIgnitionSchedule5(ign5StartFunction, ignition5StartTime, currentStatus.dwell + fixedCrankingOverride, ign5EndFunction);
                }
            }
       } //Ignition schedules on
       } //Has sync and RPM
       } //loop()
```

PROJEKTIPYÖRÄÄN LISÄTYN JOHTOSARJAN YKSINKERTAISTETTU KYTKENTÄKAAVIO



Alla olevassa taulukossa esitetty johtosarjassa käytettyjen johtimien värit, materiaalit, paksuudet ja tehtävät.

# Johdotus

| Merkintä speeduinossa | väri | Suojamateriaali | Johtimen paksuus (AWG) | Liittimessä | Tehtävä |
|---|---|---|---|---|---|
| IAT | musta | silikoni | 22 | 12P DT | imuilman lämpötila-anturi |
| IAT Ret | sininen | silikoni | 22 | 12P DT | imuilman lämpötila-anturi |
| CLT | valkoinen | silikoni | 22 | 12P DT | Moottorin lämpötila-anturi |
| TPS | keltainen | silikoni | 22 | 12P DT | Kaasuläpän asentoanturi |
| TPS Ret | Harmaa | silikoni | 22 | 12P DT | Kaasuläpän asentoanturi |
| TPS Vref | ruskea | silikoni | 22 | 12P DT | Kaasuläpän asentoanturi |
| O2 | vihreä | muovi | 22 | 12P DT | Jäännöshappianturi, signaali |
| GND | musta | muovi | 22 | 12P DT | Jäännöshappianturi, maadoitus |
| VR1- | punainen | silikoni | 22 | 12P DT | pyörimisnopeusanturi |
| RPM1 | violetti | silikoni | 22 | 12P DT | pyörimisnopeusanturi |
| | | | | | |
| 12V | punainen | silikoni | 14 | 6P DT | Jännite, negatiivinen napa |
| GND | musta | silikoni | 14 | 6P DT | Jännite, positiivinen napa |
| INJ1 | oranssi | silikoni | 14 | 6P DT | Polttoaineen suihkutuskanava 1 |
| INJ2 | keltainen | silikoni | 14 | 6P DT | Polttoaineen suihkutuskanava 2 |
| IGN1 | sininen | silikoni | 14 | 6P DT | sytytyskanava 1 |
| IGN2 | vihreä | silikoni | 14 | 6P DT | sytytyskanava 2 |

Liite 3

MUUTOSTÖIHIN HANKITUT KOMPONENTIT HINTOINEEN

Alla taulukoituna asennusprojektin aikana hankitut komponentit ja tarvikkeet hintoineen. Projektin kustannuksia arvioitiin etukäteen vuoden 2018 alussa ennen hankintojen suorittamista.

| | hankittu | Hinta (€) | Arvioitu kustannus (€) | Erotus (€) |
|---|---|---|---|---|
| | | | | |
| **ECU** | | | | |
| arduino | | | 20 | |
| | MEGA 2560 R3 | 6,99 | | |
| | HC-06 blutooth module | 3,03 | | |
| | usb to ttl | 0,87 | | |
| speeduino | | | | |
| | v0.3 + MAP + VR conditioner | 167,38 | 150 | **-17,38** |
| | | | | |
| **yht** | | **178,27** | **170** | **-8,27** |
| | | | | |
| **sensorit** | | | | |
| lämpötila | | 9,64 | 10 | **0,36** |
| WBO2 | 14point7 + bosch LSU 4.9 | 122,57 | 165 | **13,81** |
| | | | | |
| **yht** | | **132,21** | **175** | **42,79** |
| | | | | |
| **polttoaine** | | | | |
| läppärunkopaketti | | 77,02 | 130 | **52,98** |
| johtosarja läppärungoille | | 22,75 | | **-22,75** |
| pumppu | | 31,56 | 20 | **-11,56** |
| suodatin | | 4,81 | | **-4,81** |
| | | | | |
| | | | | |
| **yht** | | **136,14** | **150** | **13,86** |
| | | | | |
| **sytytys** | | | | |
| puolat | | 31,27 | 50 | **18,73** |
| johdot | | 34,98 | | **-34,98** |
| | | | | |
| **yht** | | **66,25** | **50** | **-16,25** |
| | | | | |

| | | | | |
|---|---|---|---|---|
| **tarvikkeita** | | | 100 | |
| | hobbyking (14awg johdot) | 27,51 | | |
| | DT 2x 6P, 5x 4P , 12P | 21,62 | | |
| | Ebay 9kpl 2m 20awg | 17,06 | | |
| | Sytytyspuolan liitin | 12,69 | | |
| | USB-jatkojohto | 2,95 | | |
| | EV1 liitin | 6,2 | | |
| | Letkuja ym. | 50 | | |
| | | | | |
| **yht** | | **138,03** | **100** | **-38,03** |
| | | | | |
| **postikulut +alv** | | | | |
| | Autovaraosa 24 osa1 | 37,64 | | |
| | Autovaraosa 24 osa2 | 16,15 | | |
| | WBO2 tullaus | 28,62 | | |
| | | | | |
| **yht** | | 82,41 | | |
| | | | | |
| summa | | 733 | 645 | **-88** |