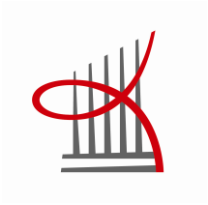


KAUKO-OHJAUS LANGATTOMASSA LÄHIVERKOSSA

Aripekka Vorne

Opinnäytetyö
Toukokuu 2014
Tietotekniikka
Sulatetut järjestelmät ja elektroniikka

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Sulautetut järjestelmät ja elektroniikka

ARIPEKKA VORNE:
Kauko-ohjaus langattomassa lähiverkossa

Opinnäytetyö 66 sivua, joista liitteitä 23 sivua
Toukokuu 2014

Tämän opinnäytetyön tarkoituksena oli korvata kauko-ohjattavan auton radio-ohjaus langattomalla lähiverkkoyhteydellä, mikä käytännössä tarkoitti auton radiovastaanottimen korvaamista Raspberry Pi -tietokoneella, joka sopi työn tarkoituksiin erinomaisesti pienen kokonsa ja ominaisuuksiensa ansiosta. Tässä raportissa esitellään Raspberry Piin lisäksi muut työssä käytetyt laitteet ja ohjelmistot sekä käydään läpi työtä varten kirjoitetut ohjelmistokoodit vaihe vaiheelta. Työn varsinaisina tavoitteina, jotka kaikki täytettiin, oli oppia Python-ohjelmointikieltä, tutustua työskentelyyn Linux-ympäristössä sekä luoda toimiva sovellus.

Kauko-ohjattavan auton ohjauksen ja nopeudensäädön toiminta perustui pulssinleveysmodulaatioon, jolla ohjattiin auton moottoreita. Autossa oli kaksi moottoria, yksi servomoottori ohjausta varten ja yksi tasasähkömoottori liikkumiseen. Työn haasteena oli luoda moottoreille lähetetyt ohjaussignaalit niin, että ne vastaisivat radiovastaanottimen autolle lähettämiä signaaleita. Tässä opinnäytetyöraportissa esitellään kaksi vaihtoehtoista tapaa luoda nuo signaalit. Ensimmäinen tapa oli käyttää Raspberry Pi -tietokonetta ja siihen asennettua RPIO-Python-moduulia, ja toinen oli Raspberry Piin luku- ja kirjoitusominaisuuksien laajentaminen kytkemällä siihen Arduino Mega 2560 -kehitysalusta.

Komentojen välitys Raspberry Pi -tietokoneelle toteutettiin lähiverkossa näkyvän verkkosivun kautta, johon sai yhteyden kirjoittamalla Internet-selaimen osoiteriviin Raspberry Piin IP-osoitteen. Verkkosivu ja komentojen välitys toteutettiin Python-ohjelmointikieleen perustuvalla Flask Microframework -palvelinohjelmistolla.

RPIO-moduulilla toteutetun version eduksi voidaan laskea sen pieni tilan tarve. Käyttäjäkokemuksen ja kauko-ohjattavan auton toiminnan kannalta Arduino-kehitysalustalla toteutettu versio oli kuitenkin parempi, sillä sen toiminta oli lähempänä auton alkuperäistä toimintaa.

Asiasanat: kauko-ohjaus, langaton lähiverkko, Raspberry Pi, pulssinleveysmodulaatio

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Embedded Systems and Electronics

ARIPEKKA VORNE:
Remote Control in Wireless Local Area Network

Bachelor's thesis 66 pages, appendices 23 pages
May 2014

The objective of this thesis was to make a radio controlled car work with a wireless local area network connection. In practice this meant replacing the radio receiver of the car with a Raspberry Pi computer, which was well suited for the needs of this project, thanks to its small size and capabilities. This report includes background information on the Raspberry Pi and on the other hardware and software that were used in this project, and explains step by step all the programming code that was written. Overall the goals of this thesis were to learn to use Python programming language, to become familiar with working in a Linux environment and to create a working application. All of these goals were met.

The electronic controls of the RC-car were based on pulse width modulation, which was used to control the motors of the car. The car had two motors, one servo motor that was responsible for the steering and one DC motor that provided the momentum. The challenging part of the project was recreating the control signals for the motors so that they would match those that were originally created in the radio receiver of the car. This thesis proposes two alternative ways to create those signals: the first is by using only the Raspberry Pi and a Python programming module called RPIO, and the second is by extending the I/O capabilities of the Raspberry Pi by connecting a Arduino Mega 2560 development board to the Raspberry Pi.

The control commands were send to the Raspberry Pi via a website, which was accessible in the local area network by typing the IP-address of the Raspberry Pi to a web browser. The website, along with relaying the received commands forward, were implemented by using Flask Microframework, which is a server software based on a Python programming language.

The system using the RPIO-module had an advantage of requiring little space. However, the version that used the Arduino development board provided a better user experience, mainly because its operation was closer to the original.

Key words: remote control, wireless local area network, Raspberry Pi, pulse width modulation

SISÄLLYS

LYHENTEET JA TERMIT

DMA	Direct Memory Acces. Ominaisuus, jonka avulla laitteiston osat voivat olla suoraan yhteydessä järjestelmän muistiin.
GPIO	Genral Purpose I/O. GPIO-pinni on laajasti ohjelmoitavissa oleva I/O-pinni.
HTML	HyperText Markup Language. Verkkosivujen luonnissa yleisesti käytetty kieli.
HTTP	HyperText Transfer Protocol. HTTP on muun muassa Internet-selaimien käyttämä tiedonsiirtoprotokolla.
I/O	Input/output, eli sisääntulo/ulostulo. I/O-pinni on piirilevyn liitin, jonka kautta voidaan suorittaa luku- tai kirjoitusoperaatiota.
IP-osoite	Internetin protokollaosoite, joka on jokaiselle Internetiin tai lähiverkkoon kytketylle laitteelle yksilöllinen numerosarja.
MOSFET	Metal-oxide-semiconductor field-effect transistor. MOSFET on yleisesti käytetty kanavatransistori.
PWM	Pulse width modulation, eli pulssinleveysmodulaatio.
SSH	Secure Shell. SSH on tiedonsiirtoprotokolla, jota käytetään salatussa tietoliikenteessä.
USB	Universal Serial Bus. USB on sarjaväyläarkkitehtuuri, jota käytetään yleisesti oheislaitteiden kytkemisessä tietokoneeseen sekä mobiililaitteiden akun latauksessa.
Wi-Fi	Tavaramerkki, jota käytetään usein WLAN-termin synonyymina.
WLAN	Wireless Local Area Network, eli langaton lähiverkko. Toimii 2,4 GHz:n taajuusalueella.

1 JOHDANTO

Elektroniikan tee-se-itse-harrastelijoiden määrä on viimevuosina kasvanut halpojen ja helppokäyttöisten laitteiden myötä, joista esimerkkeinä mainittakoon Arduino-kehitysalustat ja Raspberry Pi -minitietokone, jotka mahdollistavat kaikenlaiset projektit aina LED-valojen vilkutuksesta kulunvalvontajärjestelmiin. Näiden laitteiden menestyksen seurauksena markkinoille on hiljattain tullut myös kalliimpia ja tehokkaampia sulautettuihin järjestelmiin suunniteltuja laitteita, kuten Intelin NUC-tietokoneet, joista löytyy muun muassa pöytäkoneissa käytettyjä tehokkaita prosessoreita.

Tämän opinnäytetyön tavoitteena oli saada kauko-ohjattavan auton radiovastaanotin korvattua Raspberry Pi -tietokoneella ja näin ollen mahdollistaa ohjaus lähiverkossa WLAN-yhteyden välityksellä. Työn aihe valikoitui pääasiassa sen perusteella, että se havainnollistaisi hyvin laitteiston ja ohjelmiston välistä vuorovaikutusta, eli sulautettua järjestelmää.

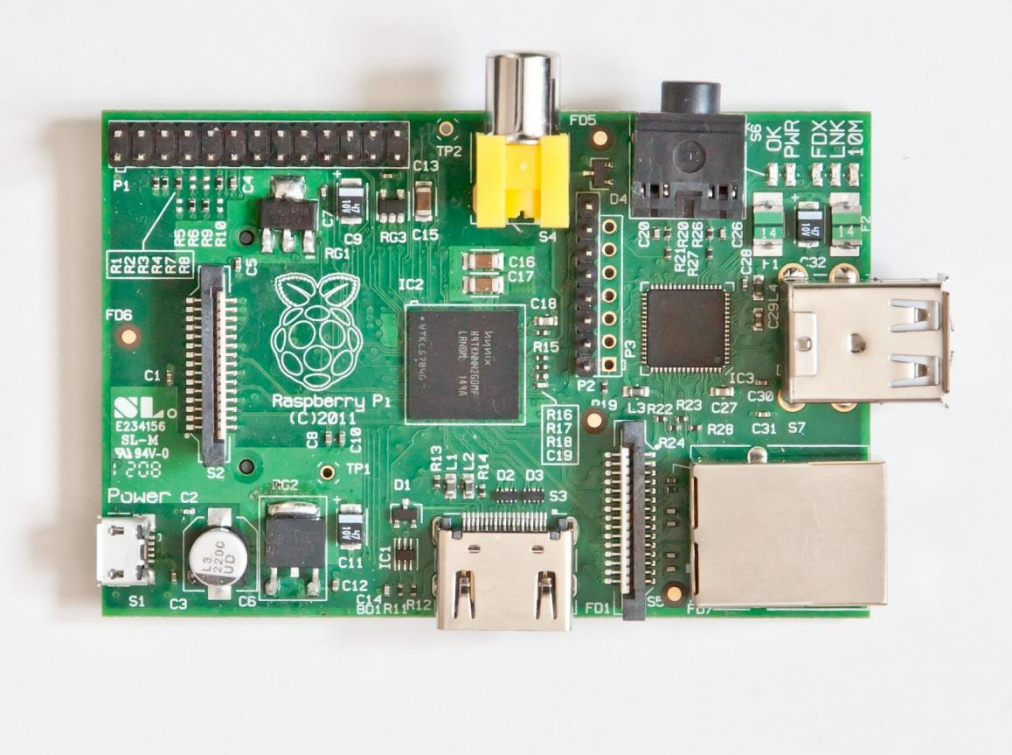
Työn alusta asti oli selvää, että työ tulisi olemaan erittäin ajoituskriittinen, eli koko järjestelmän toiminta perustuisi laitteiston ja ohjelmiston kykyyn luoda erinäisiä signaaleita vähintään kymmenen mikrosekunnin tarkkuudella. Lisäksi käytetty ohjelmointiympäristö ja -kieli olivat ennestään tuntemattomia, mikä lisäsi työn haastavuutta entisestään; tämä oli kuitenkin tarkoituksenmukaista, sillä yksi työn tavoitteista oli uuden oppiminen ja osaamisen laajentaminen.

Näiden seikkojen seurauksena työ oli pitkälti tutkimus- ja kehitysprojekti, ja kehitysvaihetta onkin haluttu nostaa esiin tässä raportissa esittelemällä kaksi vaihto-ehtoista ratkaisua, joista toinen toteutettiin pelkällä Raspberry Pi -tietokoneella ja RPIO-nimisellä Python-ohjelmointimoduulilla, ja toinen laajentamalla Raspberry Piin ominaisuuksia kytkemällä siihen Arduino Mega 2560 -kehitysalusta.

2 RASPBERRY PI -TIETOKONE

2.1 Historia

Raspberry Pi on vuonna 2012 julkaistu, alun perin lapsille ja nuorille suunnattu, luottokortin kokoinen tietokone (kuva 1). Idea tietokoneeseen syntyi vuonna 2006, kun Ebden Upton, Rob Mullins, Jack Lang ja Alan Mycroft Cambridgen yliopiston tietokonelaboratoriosta huolestuivat aloittavien tietotekniikan opiskelijoiden alati heikentyvistä taidoista (Raspberrypi.org. 2014a).



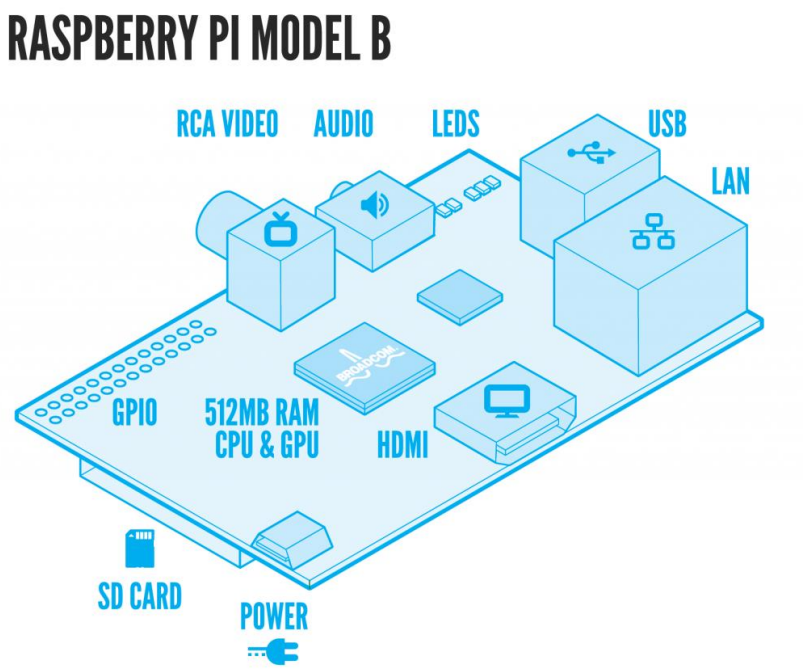
KUVA 1. Rasberry Pi Model B (Readwrite.com. 2014)

Nykyään Raspberry Piin oikeuksia hallinnoi Raspberry Pi Foundation -hyväntekeväisyyssäätiö (Raspberrypi.org. 2014b).

2.2 Piirilevy

Raspberry Pi on yhden piirilevyn järjestelmä, eli "system on chip", joka perustuu Broadcomin BCM2835-piiriin. Kuvassa 2 on esitetty Raspberry Piin B-mallin piirilevyn

oleelliset liittimet ja komponentit. A-mallissa on yksi USB-portti vähemmän, eikä siinä ole Ethernet-liitintä ja muistia siinä on puolet B-mallista, eli 256 megatavua. Käyttövirran piirilevy saa micro-USB -portin kautta ja B-mallin virrankulutus on tyypillisesti 700 – 1000 mA, joten tietokoneiden tavallinen USB 2.0 portti ei riitä, koska niiden ulostulo on rajattu puoleen ampeeriin. Virtalähteenä on suositeltavaa käyttää älypuhelimistakin tuttua USB-verkkovirtalaturia tai erillisellä virtalähteellä varustettua USB-hubia. (Raspberrypi.org. 2014b)



KUVA 2. Raspberry Piin B-mallin liittimet (Raspberrypi.org. 2014b)

Raspberry Piissä on 700 MHz:n ARM1176JZF-S -prosessori, joka vastaa suorituskyvyltään suurin piirtein Intelin 300 MHz:n Pentium 2 -prosessoria, mutta paremmalla graafisella suorituskyvyllä (Raspberrypi.org. 2014b). Suoritusteho riittää HD-tasoisen kuvan toistamiseen ongelmitta ja esimerkiksi Quake3- tai Minecraft-peliin.

Työn kannalta tärkeää piirilevyssä olivat sen GPIO-pinnit, joiden avulla Raspberrylla luodut signaalit saataisiin välitettyä kauko-ohjattavalle autolle. Pinnejä on 26 kappaletta ja niiden GPIO-numerointi vaihtelee piirilevyn revision-numerosta riippuen (kuva 3). Ohjelmoidessa toiminnallisuutta Raspberryn I/O-pinneihin on mahdollista käyttää pinnien fyysistä numerointia; useimmat ohjelmointikirjastot ja -moduulit kuitenkin tukevat paremmin GPIO-numerointia, joten piirilevyn revision-numero on hyvä olla tiedossa.

KUVA 3. Raspberry Piin I/O-pinnit (Elinux.org. 2014a)

Revision-numeron saa tietää, kun syöttää Linux-terminaaliin komennon "cat /proc/cpuinfo". Komennon antamisen jälkeen terminaaliin ilmestyy prosessorin tiedot, josta löytyy rivi "Revision". Rivillä ilmoitettu koodiluku ei kuitenkaan kerro suoraan revision-numeroa, vaan lukua pitää verrata esimerkiksi taulukkoon 1. Työssä käytetyn Raspberryn koodiluku oli "0003", joten kyseessä oli B-mallin Revision 1.0.

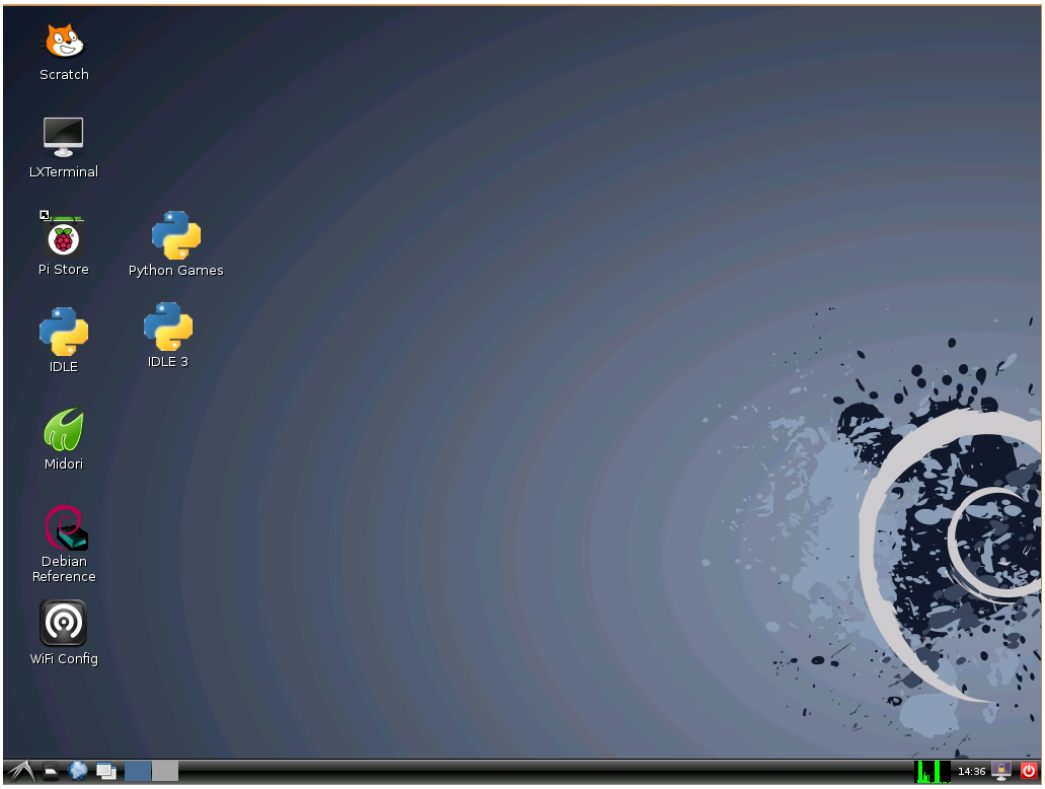
TAULUKKO 1. Raspberry Pi revision (Elinux.org. 2014b)

Revision	Release Date	Model	PCB Revision	Memory	Notes
Beta	Q1 2012	B (Beta)	?	256MB	Beta Board
0002	Q1 2012	B	1.0	256MB	
0003	Q3 2012	B (ECN0001)	1.0	256MB	Fuses mod and D14 removed
0004	Q3 2012	B	2.0	256MB	(Mfg by Sony)
0005	Q4 2012	B	2.0	256MB	(Mfg by Qisda)
0006	Q4 2012	B	2.0	256MB	(Mfg by Egoman)
0007	Q1 2013	A	2.0	256MB	(Mfg by Egoman)
0008	Q1 2013	A	2.0	256MB	(Mfg by Sony)
0009	Q1 2013	A	2.0	256MB	(Mfg by Qisda)
000d	Q4 2012	B	2.0	512MB	(Mfg by Egoman)
000e	Q4 2012	B	2.0	512MB	(Mfg by Sony)
000f	Q4 2012	B	2.0	512MB	(Mfg by Qisda)

3 RASPBIAN-KÄYTTÖJÄRJESTELMÄ

Yleisin Raspberry Piissä käytetty käyttöjärjestelmä on Debian "Wheezy" -Linux-jakelupakettiin perustuva Raspbian, jonka voi ladata ilmaiseksi muun muassa Raspberry Piin virallisilta sivuilta. Raspbianin asemaa Raspberry Piin "de facto -käyttöjärjestelmänä" vahvistaa myös siihen perustuvien käyttöjärjestelmien määrä, joista esimerkkeinä mainittakoon Moebius, Occidentalis ja OpenELEC (Elinux.org. 2014c). Ensimmäinen versio Raspbianista julkaistiin kesäkuussa 2012 ja käyttöjärjestelmän kehitys jatkuu yhä.

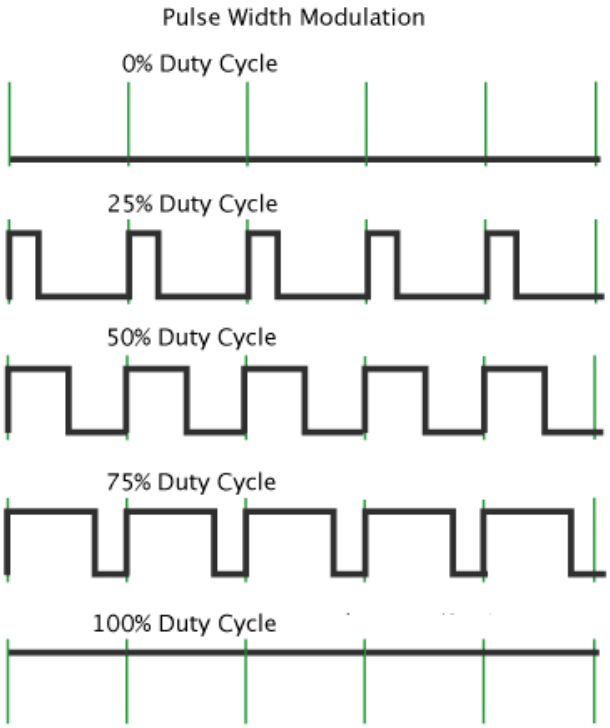
Raspbianin käyttöliittymä perustuu perinteisen komentorivin, eli Linux-terminaalin, lisäksi LXDE-työpöytäympäristöön (kuva 4). LXDE soveltuu hyvin Raspberryyn, koska se vie vähemmän muistia ja prosessoritehoa kuin useammat muut työpöytäympäristöt, tinkimättä kuitenkaan ominaisuuksista (Linux.fi. 2013). Raspbianin monipuolisuutta tukee myös sen kattava ohjelmistokirjasto, joka sisältää yli 35 00 ohjelmistopakettia, jotka ovat helposti asennettavissa tarpeen mukaan (Rasbian.org). Suurin osa näistä paketeista on käännetty Raspberry-yhteensopiviksi suoraan Debianilta, mutta joukossa on myös useita Raspberrya varta vasten kehitettyjä ohjelmistopaketteja.



KUVA 4. LXDE-työpöytäympäristö

4 KAUKO-OHJATTAVAN AUTON TOIMINTA

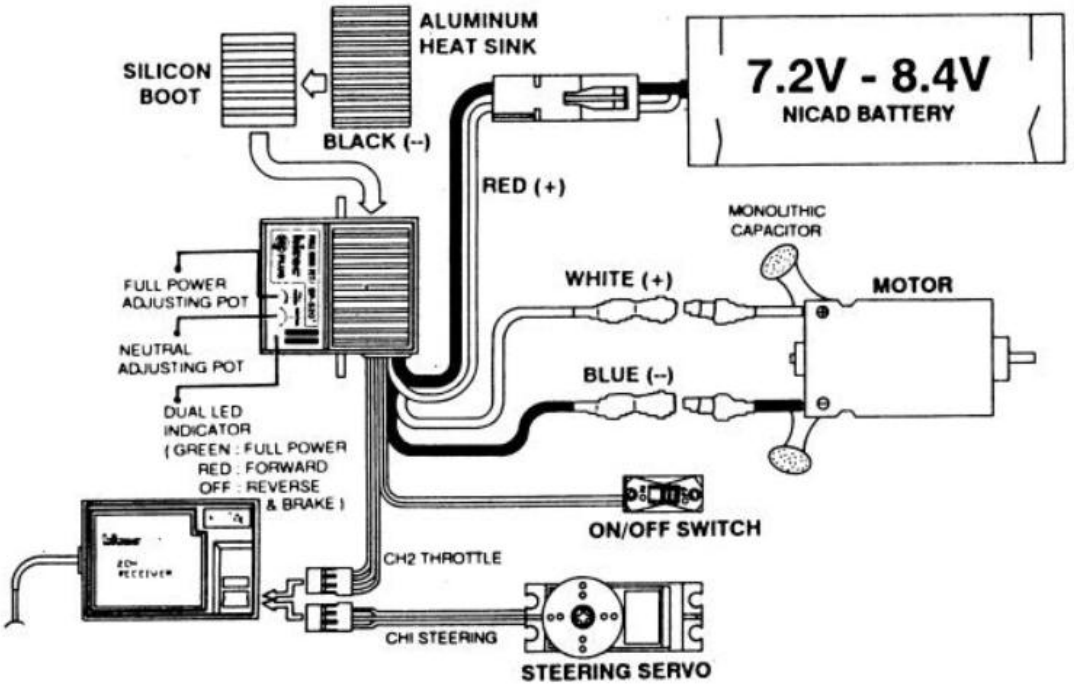
Pulssinleveysmodulaatio, eli PWM, on yleisin tapa ohjata servomoottoreita ja sitä on mahdollista hyödyntää myös muunlaisten tasasähkömoottoreiden ohjauksessa. PWM-signaali on kanttiaaltoa, jossa signaalin ylhäälläoloaika määrittää kohdelaitteen toiminnan. Jaksonajan ja ylhäälläoloajan suhde ilmoitetaan prosenttiluvulla, jota kutsutaan pulssisuhteeksi (kuva 5). Jaksonaika useimmissa servomoottoreissa on noin 20 ms, mutta pulssisuhdetta tärkeämpää laitteen toiminnan kannalta on usein varsinaisen pulssin kesto, eli leveys, joten jaksonajan ei tarvitse olla viimeisen päälle määritetty.



KUVA 5. PWM-signaali ja pulssisuhde (Hirzel, muokattu)

PWM-ohjauksessa käytetään usein 3-johtoista kytkentää, jossa yksi johto on maa, toinen käyttöjännite ja kolmas PWM-signaali. 4-johtoisia PWM-kytkentöjä näkee esimerkiksi tietokoneiden PWM-ohjatuissa laitetuulettimissa, joissa neljäs johto palauttaa takometrin signaalin emolevylle tuulettimen kierrosnopeuden saamiseksi. PWM-signaalia voidaan moottorinohjauksen lisäksi hyödyntää muun muassa LED-valojen ohjauksessa, jossa pulssisuhde määrittää valojen kirkkauden sekä audiosignaalin luomisessa (Shirriff. 2009).

Työssä käytetty kauko-ohjattava auto oli malliltaan Kyosho EP Ultima ST, jossa PWM-signaalien tuottamisesta vastasi Kyoshon Perfex AM 2ch -radiovastaanotin, joka sai virtansa Hitecin SP-520 PLUS -nopeudensäädinyksiköltä tulevasta PWM-johdotuksen käyttöjännitejohdosta (kuva 6). Tämä sama jännite välitettiin vastaanottimen kautta myös ohjauksesta vastaavan servomoottorin käyttöjännitteeksi. Servomoottorista poiketen, työnnöstä vastaavan tasasähkömoottorin ohjaus ei tapahtunut suoraan PWM-johdoilla sen suuren virrantarpeen takia, vaan signaalilla ohjattiin nopeudensäätimen MOSFET:ia, joka kontrolloi suoraan akulta moottorille menevää virtaa.



KUVA 6. Kauko-ohjattavan auton kytkennät (Hitec)

Käyttöjännitteen arvoksi mitattiin Fluken PM3380B-oskilloskoopillla 5,8 volttia ja PWM-signaalin huippujännitteeksi 3,3 volttia. PWM-ohjaussignaalin 3,3 voltin jännitteen ansiosta signaali oli teoriassa mahdollista luoda suoraan Raspberryn I/O-pinneistä, sillä niiden ulostulojännite on rajattu kyseiseen arvoon – pois lukien 5 V -pinnit, jotka ovat suoraan yhteydessä piirivilevyn microUSB-liittimeen.

Radiovastaanottimen tuottaman PWM-signaalin jaksonajaksi mitattiin oskilloskoopilla 18,0 ms ja pulssin leveyden havaittiin vaihtelevan välillä 1,0–2,0 ms (liite 1). Pulssin leveyden vaikutus DC-moottoriin ja servomoottoriin on esitetty taulukossa 2.

TAULUKKO 2. PWM-signaalin vaikutus moottoreihin.

Pulssin kesto	DC-moottori	Servomoottori
1,0 ms	Täydellä teholla taaksepäin	Äärivasemmalla
1,5 ms	Seis	Keskiasennossa
2,0 ms	Täydellä teholla eteenpäin	Äärioikealla

Molempien moottoreiden ohjauksessa 1,5 millisekunnin kestoinen pulssi vastasi siis lepotilaa. Taulukossa 2 esitetyt arvot ovat kuitenkin ideaalisia ja vastaavat signaaleja, joita syntysi mikäli radio-ohjaimen trim-säätimet olisivat keskiasennoissa. Trimmauksella, eli hienosäädöllä, säädetään ohjaimen kaasuliipaisimen ja ohjauspyörän lepotiloja vastaavien pulssien kestoja. Esimerkiksi DC-moottorin lepotilan pulssi täytyi säätää kestämään 1,55 ms, koska ilman tätä hienosäätöä auto olisi kulkenut oletuksena taaksepäin. DC-moottorin lepotilan pulssin leveyttä tulisi ensisijaisesti säätää nopeudensäädinyksikön potentiometreista, mutta mittaushetkellä ei ollut saatavilla tarvittavia työkaluja kyseiseen toimenpiteeseen.

Oskilloskooppimittausten tulokset voitiin vahvistaa kytkemällä moottorit HP E3611A -teholähteeseen ja HP 8110A -pulssigeneraattoriin, joilla onnistuneesti imitoitiin auton oikeita ohjaussignaaleita. Teholähteen virranrajoituksen avulla voitiin samalla todeta ohjausservon vaativan vähintään 500 mA virran toimiakseen kunnolla. Testauksen yhteydessä varmistettiin ohjauksen toimivan myös signaaleilla, joiden jaksonajat olivat 10, 16, 20 ja 30 millisekuntia.

5 PYTHON -OHJELMOINTIKIELI

Työn ohjelmistokoodit kirjoitettiin pääasiassa Python ohjelmointikielellä, jonka perusteet ovat helposti opittavissa, ja koska se on esimerkiksi C-kieltä korkeampitasoisempi ohjelmointikieli, on se myös suhteellisen helppolukuinen aloittelijoille, kuten kuvan 7 yksinkertaisesta esimerkistä on havaittavissa. Ohjelmointikielen tasoilla tarkoitetaan kuinka läheisesti kyseinen kieli kommunikoi laitteiston kanssa; assemblyn tapaiset matalan tason kielet kommunikoivat lähes suoraan laitteiston kanssa, kun taas Pythonilla ja muilla korkean tason kielillä kommunikaatio tapahtuu useamman vaiheen kautta. Käytännössä korkean tason abstraktimmat kielet vaihtavat matalan tasojen kielien tarkemman resurssienhallinnan ja nopeamman suoritusajan käyttäjäystävällisyyteen.

```cpp
//C++

#include <iostream>

int main()
{
    std::cout << "Hello World!";
    return 0;
}
```

```python
#Python

print('Hello World!')
```

KUVA 7. Hello world -vertailu

Yleisesti käytettyjen C- ja C++-kielien kaarisulkeista ja puolipisteistä poiketen Python-koodin rakenne perustuu sisennyksiin ja rivinvaihtoihin, mikä saattaa aluksi hämmentää C-kielestä Pythoniin siirtyvää ohjelmoijaa.

Python on niin sanottu tulkittu kieli, eli koodia ei tarvitse kääntää konekielelle luontivaiheessa, vaan se käännetään ennen sen suoritusta, mikä mahdollistaa Python-koodin kirjoittamisen millä tahansa tavallisella tekstinkäsittelyohjelmalla, kun tallentaessa lisää perään tiedostopäätteen ".py". Tämä seurauksena Python-sovellusten suoritusaika on tosin hieman pidempi verrattuna sovelluksiin, joissa käytetty ohjelmointikieli on vaatinut koodin kääntämistä konekielelle jo luontivaiheessa. Tämän script-kielimäisyyden ansiosta Python soveltuukin erinomaisesti projekteihin, jotka edellyttävät sovelluksen kehitystä nopealla aikataululla. (Maruch & Maruch. 2006. 7–8)

Pythonin kehitti Guido van Rossum 80-luvun lopulla ja ensimmäinen versio siitä julkaistiin vuonna 1991. Nimensä ohjelmointikieli on saanut brittiläiseltä Monty Python -komediaryhmältä (Maruch & Maruch. 2006. 8–9). Tällä hetkellä Pythonista on käytössä kaksi versiota, 2.7 ja 3.4. Python 2:n kehitystyön päättymisestä huolimatta, on sen käyttö edelleen suositeltavaa, mikäli aikoo käyttää kolmannen osapuolen ohjelmistoja ja moduuleita, koska osa niistä ei ole vielä yhteensopivia Python 3:n kanssa (Python.org).

6 FLASK MICROFRAMEWORK -PALVELINOHJELMISTO

Vastaanottaakseen komentoja toiselta tietokoneelta täytyi Raspberryyn asentaa palvelinohjelmisto. Vaihtoehtoja oli useita, kuten Django ja Apache, joiden käyttö kuitenkin tuntui hieman ylilyönniltä työn vaatimuksiin nähden. Työssä päädyttiin käyttämään Flask-ohjelmistoa, joka oli kevyt ja soveltui erinomaisesti käyttötarkoitukseen, eli Raspberryn I/O-pinnien hallintaan lähiverkon kautta. Flaskin suurin etu oli kuitenkin se, että palvelin toimi yhden Python-tiedoston kautta, mikä teki muun työssä käytetyn Python-koodin integroinnista helppoa.

Flaskin vahvuus ilmenee mikro-sanasta ohjelman nimessä, sillä Flask ei sisällä oletuksena mitään ylimääräistä, kuten tietokanta- tai lomakejärjestelmää, vaan se on käyttäjän tarpeista riippuen laajennettavissa. Käyttäjä voi näin ollen muokata palvelimestaan juuri sellaisen kuin haluaa, tarvitsemillaan kolmannen osapuolen ohjelmistoilla. (Flask. 2014a)

Flask-sovellusten tiedostorakenne on kuvan 8 mukainen. "App" on sovellus kansio, jossa voi olla varsinaisen Pythonilla toteutetun Flask-sovelluksen lisäksi muu sovelluksessa käytetty ohjelmakoodi sekä pakolliset kansiot "static" ja "templates". Templates-kansioon tulee verkkosivun muodostamiseen käytetyt HTML-tiedostot, kun taas static-kansioon tulee puolestaan kaikki HTML-sivupohjien käyttämät tiedostot, kuten css-muotoilutiedostot, JavaScript-tiedostot, kuvat ja niin edelleen.

```
/App
    /static
        .css
        .js
        .jpg
    /templates
        .html
    flaskApp.py
    script1.xx
    script2.yy
```

KUVA 8. Flask-sovelluksen tiedostorakenne

Komentojen välitys Raspberrylle toteutettaisiin siis verkkosivun kautta, kirjoittamalla Internet-selaimen osoiteriviin Raspberryn IP-osoite. Verkkosivun luomiseen Flask käyttää apunaan Jinja2-sivupohjamoottoria, jonka avulla html-tiedostosta ja siihen linkitetystä sisällöstä, kuten kuvista, muodostetaan varsinainen verkkosivu.

Ensimmäinen versio Flaskista julkaistiin huhtikuussa 2010 ja siitä on nykyään saatavilla Python 3 -yhteensopiva versio, mutta Python 2 -version käyttö on edelleen suositeltavaa (Flask. 2014b).

7 RASPBERRY PIIN KÄYTTÖNOTTO

Jotta työssä luotu sovellus toimisi, täytyy Raspberry asentaa toimintakuntoiseksi. Ensimmäisenä Raspberryssa käytettävään SD-muistikorttiin, jonka on hyvä olla kooltaan ainakin 8 GB, asennetaan Rasbian-käyttöjärjestelmä asettamalla kortti toisen tietokoneen muistikortinlukijaan tai ulkoiseen USB-kortinlukijaan. Käyttöjärjestelmän voi ladata esimerkiksi Raspberry Pi -säätiön sivuilta (Raspberrypi.org. 2014c) ja asennuksessa noudattaa sivustolla annettuja ohjeita (Raspberrypi.org. 2014d).

Käyttöjärjestelmän asennuksen jälkeen Raspberryyn voi liittää WLAN-sovittimen ja virtajohdon sekä kytkeä Ethernet-johdolla reitittimeen. Ylimääräistä näyttöä, hiirtä tai näppäimistöä ei tarvita, koska Raspbianissa on oletuksena päällä SSH-palvelin, johon voi muodostaa yhteyden lataamalla toiselle koneelle esimerkiksi PuTTY-sovelluksen osoitteesta "http://www.putty.org/". SSH-yhteyden muodostamisen jälkeen PuTTY toimi etäkäytettävänä Linux-terminaalina. PuTTY:n vaatiman Raspberryn IP-osoitteen saa selville lataamalla koneelle esimerkiksi Fing-sovelluksen osoitteesta "http://www.overlooksoft.com/fing". Tarkemmat ohjeet etäyhteyden muodostamiseen löytää vaikkapa ModMyPiin blogista (Modmypi.com. 2013). Kun etäyhteys on muodostettu ja Raspberylle on annettu kiinteä lähiverkon IP-osoite, on aika vaihtaa langattomaan lähiverkkoyhteyteen.

Langattoman yhteyden voi muodostaa helposti etätyöpöytäyhteyden avulla, käyttäen esimerkiksi Windowsin Remote Desktop Connection -ohjelmaa, sillä Raspbian-käyttöjärjestelmän työpöydältä löytyy valmiina pikakuvake WiFi Config -ohjelmalle, jonka avulla yhteyden muodostaminen sujuu vaivatta. Työssä käytetty WLAN-sovitin ei vaatinut erillisten ajureiden asennusta, joten tehtäväksi jäi ainoastaan syöttää Raspberryyn langattoman verkon SSID-tunnus ja salasana. Etätyöpöytäyhteyttä varten Raspberryyn täytyy asentaa esimerkiksi xrdp-ohjelmisto kirjoittamalla Linux-terminaaliin komento "sudo apt-get install xrdp".

Käyttöjärjestelmän ja Internet-yhteyden toimiessa, asennetaan Raspberryyn työn kannalta oleelliset ohjelmistopaketit ja ohjelmointimoduulit. Asennukset suoritetaan kirjoittamalla Linux-terminaaliin seuraavat komennot: "sudo apt-get install python-pip", "sudo pip install flask" ja "sudo pip install RPIO"

Etäyhteyden ansiosta työn pystyi tekemään kokonaan tehokkaammalta Windows-tietokoneelta. Windows-koneelle ladattiin PuTTY:n lisäksi WinSCP-FTP-ohjelma tiedostojen siirtoa varten, sekä Notepad++-tekstinkäsittelyohjelma, jolla kirjoitettiin kaikki työn koodi, koska mitään työssä käytetyistä koodeista ei ollut tarvetta kääntää.

8 FLASK-SOVELLUS

Työn keskipisteenä toimi Python-sovellus "flaskRPIO" (liite 2), jonka kautta tapahtui verkkosivun toiminta sekä ohjauskomentojen välitys signaaleita tuottaville ohjelmille.

Ensimmäisenä flaskRPIO-ohjelmassa kutsutaan kyseissä koodissa käytettyjä moduuleita import-komennolla (kuva 9). Moduulin yksittäisiä luokkia voidaan kutsua komennolla "from <moduuli> import <luokka>"; kuvan 9 tapauksessa flask-moduulista tuodaan siis luokat *Flask*, *render_template*, *request* ja *jsonify*. Ensimmäisen rivin teksti "-*- coding: utf-8 -*-" on pakollinen, mikäli koodissa tai sen kommenteissa käyttää "ääkkösiä", sillä Python 2 tukee oletuksena vain tavallista ASCII-merkistöä.

```
# -*- coding: utf-8 -*-
#Kutsutaan koodissa käytettyjä kirjastoja ja niiden luokkia.
from flask import Flask, render_template, request, jsonify
import subprocess
```

KUVA 9. Moduulien tuonti Flask-sovellukseen

Seuraavaksi alustetaan Flask-sovellus (kuva 10). Alustus tapahtuu luomalla flask-olio, jonka nimi on tässä tapauksessa "app". Sovelluksen koostuessa vain kyseisestä tiedostosta, ja tiedostorakenteen ollessa luvussa 6 (kuva 8) esitetyn mukainen, voidaan sulkujen sisään kirjoittaa vain "__name__". Tämän jälkeen luodaan funktio verkkosivun ulkoasun luontia varten. Funktion paluuarvona oleva render_template-luokka kutsuu Jinja2-sivupohjamoottoria luomaan verkkosivun HTML-tiedostosta, jonka nimi on annettu luokalle parametrina. HTML-tiedostojenhan tulee siis sijaita templates-nimisessä kansiossa sovelluskansion sisällä. Komento "@app.route()" määrittää funktion URL-osoitteen. Kauttaviiva sulkujen sisällä tarkoittaa, että funktio suoritetaan aina verkkosivun latautuessa.

```
#Alustetaan Flask-sovellus
app = Flask(__name__)

#Määritellään verkkosivun luonnissa käytettävä html-sivupohja,
#joka sijaitsee sovelluskansion templates-kansiossa.
@app.route('/')
def index():
    return render_template('index.html')
```

KUVA 10. Flask-sovelluksen alustus

Verkkosivun kannalta pakollisten koodien jälkeen voidaan luoda auton ohjauskomentojen välityksestä vastaavat funktiot "Steering" ja "Throttle". Pythonin funktiot vastaavat toiminnaltaan C-kielen aliohjelmia. Aluksi funktioille määritellään niiden URL-osoite aiemmin esitettyyn tapaan, minkä jälkeen voidaan luoda itse funktio. Kuvassa 11 on esitetty servomoottorin ohjauskomentoja välittävä funktio. Varsinaisten signaaleiden luonnista vastaavat erilliset Python-sovellukset, joita kutsutaan subprocess-moduulin avulla. Args1-muuttujassa on määritelty kutsuttavan sovelluksen tyyppi ja tiedostopolku lista muodossa. Proc1-oliolla puolestaan kutsutaan args1-muttujassa määriteltyä sovellusta subprocess-moduulin Popen-luokan avulla. Tietojen välitys tälle "aliohjelmalle" mahdollistetaan antamalla Popen-luokan stdin-parametrille arvo *subprocess.PIPE*.

```
#Servomoottorin ohjauskomentoja vastaanottava funktio.
#App.route:lla määritetään nimi,
#jolla funktiota voi kutsua html-sivupohjassa.
@app.route('/_Steering')
def Steering():

    #Määritellään kutsuttavan ohjelman tyyppi ja sijainti
    #ja kutsutaan sitä.
    args1 = ['python', '/home/pi/flaskRPIO/SteeringRPIO.py']
    proc1 = subprocess.Popen(args1, stdin=subprocess.PIPE)

    #Vastaanotetaan verkkosivulta muuttuja "s",
    #joka sisältää ohjauskomennon.
    s = request.args.get('s')

    #Lähetetään saatu komento "aliohjelmalle".
    #Lähetettävän muuttujan täytyy olla string-muodossa.
    proc1.communicate(s)

    #Palautetaan saatu komento takasin verkkosivulle
    #debuggausta varten.
    return jsonify(steerFlask = s)
```

KUVA 11. Steering-funktio

Aliohjelman käynnistämisen jälkeen sijoitetaan muuttujaan *s* ohjauskomento (kuva 11), joka haetaan verkkosivulta flask-moduulin request-luokan avulla. Suluissa on määritelty verkkosivulta haettava muuttujan nimi. Tämän jälkeen muuttuja lähetettään aliohjelmalle subprocess.Popen().communicate() -komennon avulla, jonka viimeisten sulkujen sisään kirjoitetaan lähetettävä arvo tai muuttuja, jonka täytyy olla string-muodossa. Lopuksi verkkosivulta saatu arvo palautetaan takaisin jsonify-luokan avulla, mitä käytettään varmistamaan tiedonvälityksen toimivuus verkkosivun ja Flask-sovelluksen välillä. Tasasähkömoottorin ohjauskomentojen välitys (kuva 12) tapahtuu samalla periaatteella kuin aiemmin esitetyn servomoottorin ohjauskomentojen välityskin.

```
#Tasasähkömoottorin ohjauskomentoja vastaanottava funktio.
#Toimii samalla periaatteella kuin Steering-funktio.
@app.route('/_Throttle')
def Throttle():

    args2 = ['python', '/home/pi/flaskRPIO/ThrottleRPIO.py']
    proc2 = subprocess.Popen(args2, stdin=subprocess.PIPE)

    t = request.args.get('t')
    proc2.communicate(t)

    return jsonify(thrtFlask = t)
```

KUVA 12. Throttle-funktio

Lopuksi sovellus käynnistetään (kuva 13). Host-parametrin arvolla *"0.0.0.0"* palvelin asetetaan koko lähiverkon saataville. Port-parametrilla määritetään palvelimen käyttämä TCP-portti, jonka kannatta olla 80, sillä se käyttää oletuksena HTTP-tiedonsiirtoprotokollaa. Debug-parametrin arvon ollessa *True* osaa palvelin käynnistää itsensä uudelleen koodin muuttuessa, mikä on hyödyllistä sovellusta kehitettäessä. Debugille on kuitenkin suotavaa antaa arvo *False* lopullisessa sovelluksessa, etenkin jos palvelin tulee sijaitsemaan ei-niin-yksityisessä verkossa, sillä debug-työkalu antaa sovelluksen käyttäjille mahdollisuuden suorittaa palvelinkoneella Python-komentoja, jotka voivat olla enemmän tai vähemmän haitallisia (Flask. 2014c). Mikäli app.run() -komennolle ei anna mitään parametreja, käynnistyy sovellus niin, että palvelin näkyy pelkästään isäntälaitteelle.

```
#Käynnistetään sovellus toimimaan Raspberryn IP-osoitteessa.
if __name__ == '__main__':
    app.run(
        host="0.0.0.0",
        port=int("80"),
        debug=False
    )
```

KUVA 13. Flask-sovelluksen käynnistys

Sovelluksen sai käynnistettyä kirjoittamalla Linux-terminaaliin komennon "sudo python /home/pi/flaskRPIO/flaskRPIO.py". Sovellusta voi käyttää kirjoittamalla Internet selaimen osoiteriviin Raspberryn IP-osoitteen. Raspberrylla osoiteriviin voi myös kirjoittaa "0.0.0.0" – tai "localhost:5000", mikäli app.run() -komento on jätetty tyhjäksi.

9 HTML-SIVUPOHJA

Käyttäjäkokemuksesta työssä on vastuussa HTML-koodi (liite 3), joka on siis yhtä kuin verkkosivun sisältö, ja css-muotoilutiedostot, joilla voidaan muokata verkkosivun ulkoasua. Sivustolla on kaksi liukusäädintä, joista toinen vastaa kauko-ohjattavan auton kaasusta ja toinen ohjauksesta. Nämä interaktiiviset ominaisuudet toteutettiin käyttämällä JavaScript-kirjastoa nimeltä jQuery, joka on Pythonin tavoin helposti opittavissa, ja jolla on kattava virallinen dokumentaatio.

Html-tiedoston alussa on head-osio, eli "header", jossa alustetaan verkkosivun sisältö. Kuvassa 14 on nähtävissä koodin alku, jossa on määritetty sivuston nimi ja käytetty merkistö.

```
<!DOCTYPE html>
<html lang="fi">
    <head>

        <!-- Sivun nimi, näkyy välilehdessä -->
        <title>Control Center</title>

        <!-- Sivuston merkistö -->
        <meta charset="UTF-8"/>
```

KUVA 14. HTML-tiedoston alku

Seuraavaksi kutsutaan sivupohjassa käytettäviä script- ja muotoilutiedostoja (kuva 15). JQuery.ui.all.css -tiedostoa tarvitaan sivun liukusäädinten ulkoasua varten ja Grid 960 -järjestelmän 960_16_col.css -tiedostoa on käytetty hyväksi sivun sisällön asettelussa. Verkkosivun varsinainen ulkonäkö on määritelty style.css -tiedostossa. Lisäksi kutsutaan liukusäätimien toiminnan kannalta oleellisia jQuery-kirjastoja. Teksti "{{ url_for{'static', filename=' '} }}" on Flask-sovelluksen tapa linkittää sisältöä sivupohjaan. Tiedostonimen perään tulee halutun tiedoston sijainti static-kansiosta alkaen. Koska style.css -tiedoston sisältö ei varsinaisesti vaikuta sovelluksen toimintaan, ei sitä tässä raportissa käydä läpi. Koodi on kuitenkin kokonaisuudessaan nähtävissä liitteessä 4.

```
<!-- Css-muotoilutiedostot -->
<!-- jQuery teema slidereita ja 960 asettelua varten (http://960.gs/) -->
<!-- style.ccs sisältää sivun ulkoasun muotoilut -->
<link rel=stylesheet type=text/css href="{{ url_for('static', filename='jq/themes/base/jquery.ui.all.css') }}">
<link rel=stylesheet type=text/css href="{{ url_for('static', filename='960_16_col.css') }}">
<link rel=stylesheet type=text/css href="{{ url_for('static', filename='style.css') }}">

<!-- jQuery-kirjastot -->
<script type="text/javascript"  src="{{ url_for('static', filename='jq/jquery-1.10.2.js') }}"></script>
<script type="text/javascript"  src="{{ url_for('static', filename='jq/ui/jquery-ui.js') }}"></script>
```

KUVA 15. Verkkosivun tarvitsemien tiedostojen linkitys

Headerin lopussa alustetaan jQuerylla vielä käytetyt liukusäätimet. Kuvassa 16 on esitetty servomoottorin ohjauskomennoista vastaavan säätimen alustus. Value-arvolla säädin asetetaan tiettyyn aloitusarvoon sen sijaan, että säädin olisi minimiarvossaan sivuston latautuessa. "Slide" määrittää mitä tapahtuu, kun säädintä liikutetaan, tässä tapauksessa säätimen arvo tallennetaan tunnisteeseen *amount1*.

```
<!-- Liukusäätimien alustus -->
<script>
    $(function() {
    $( "#sliderSteering" ).slider({ // Säätimen nimi
    range: "min",
    value: 0, // Oletusarvo
    min: -4, // Säätimen ääriarvot
    max: 4,
    slide: function( event, ui ) { // Liutettaessa otetaan säätimen senhetkinen arvo
        $( "#amount1" ).val( ui.value );  // Sijoitetaan arvo tunnisteeseen amount1
    }
    });
    $( "#amount1" ).val( $( "#sliderSteering" ).slider( "value" ) );
    });
</script>
```

KUVA 16. Vaakasuoran liukusäätimen alustus

Toistetaan sama myös kaasun ohjauskomennoista vastaavalle säätimelle (kuva 17).

```
        <script>
            $(function() {
            $( "#sliderThrottle" ).slider({
            orientation: "vertical",
            range: "min",
            value: 0,
            min: -4,
            max: 4,
            slide: function( event, ui ) {
                $( "#amount2" ).val( ui.value );
            }
            });
            $( "#amount2" ).val( $( "#sliderThrottle" ).slider( "value" ) );
            });
        </script>

    </head>
```

KUVA 17. Pystysuoran liukusäätimen alustus

Headerin jälkeen tulee body-osio, jossa on verkkosivun näkyvä sisältö. Ensimmäisenä luodaan pääsisältöalue/pääkehysalue, jonka tunnisteeksi on annettu *main*. Teksti "class="container_16"" tarkoittaa, että kyseessä on Grid 960 -järjestelmän pääkehysalue, joka on nimensä mukaisesti 960 pikseliä leveä. Myöhempien div-sisältöalueiden class-tekstiä seuraava "grid" määrittää alueen leveyden. Grid-pystysarakkeita on yhteensä 16, joten yksi grid on oletuksena 50 pikseliä leveä ja sillä on 5 pikselin levyiset marginaalit molemmin puolin. Tekstillä "Push" div-sisältöaluetta voidaan työntää oikealle annetun määrän gridejä, "pull" vastaavasti vetää sisältöaluetta vasemmalle. Ensimmäisenä kehysalueeseen lisättiin tasasähkömoottoria ohjaava liukusäädin, joka näin ollen tuli oletuksena kehysalueen vasempaan reunaan.

```
<!-- Verkkosivun sisältö-->
<body>
    <div id="main" class="container_16">

        <!-- Sisältöalue, johon säädin sijoitetaan -->
        <div id="verticalSlider" class="grid_1 push_1">
            <!-- Pystysuuntainen kaasusäädin vasemassa reunassa -->
            <div id="sliderThrottle"></div>
        </div>
```

KUVA 18. Pystysuoran liukusäätimen lisäys

Seuraavaksi HTML-koodiin on kirjoitettu ominaisuus Raspberryn välittämän kamerakuvan suoratoistoa varten (kuva 19), josta on kerrottu tarkemmin lisää luvussa 11.1.

```
<div id="content" class="grid_12 push_1">
    <!-- Sisältöalue kamerakuvalle -->
    <div id="cameraFeed" class="grid_8">
        <!-- Stream -->
        <object data="http://192.168.1.67:9000/?action=stream" width=100%>
            <!-- Jos kamera-streamia ei löydy, ladataan place holder -kuva, jos sitäkään ei löydy kirjoitetaan virheviesti -->
            <img src="{{ url_for('static', filename='img/camera.png') }}" alt="Stream not found" class="backup")>
        </object>
        <!-- Erityisesti mobillilaitteiden jotkin kevyet selaimet eivät tue objectia, joten mikäli käyttä moista
             niin alla olevan rivin voi poistaa kommenteista ja yllä olevat 3 riviä kommentoida pois. -->
        <!-- <img style="width: 100%;" src="http://192.168.1.67:9000/?action=stream" alt="Stream not found"> -->
    </div>
```

KUVA 19. Kamerakuvan suoratoisto

Kuvan toistoa content-sisältöalueessa seuraa ControlInfo-sisältöalue (kuva 20), joka sisältää liukusäädinten senhetkiset arvot, joiden perässä näkyy puolestaan Flask-sovelluksen funktioiden paluuarvot.

```
    <div id="controlInfo" class="grid_7 push_2">
        <!-- Säätimien senhetkiset arvot -->
        <input type="text" id="amount2"/> |
        <input type="text" id="amount1"/> =
        <!-- Flask-sovelluksen paluuarvot -->
        <span id="thrt">T</span>|<span id="steer">S</span>
    </div>
</div>
```

KUVA 20. Liukusäädinten arvojen näyttäminen

.

Viimeisenä main-sisältöalueeseen lisätään ohjauksen vaakasuora liukusäädin (kuva 21), jolloin se sijoittuu sivun alareunaa, sillä se ei leveytensä takia enää mahtuisi edellisten elementtien viereen.

```
<!-- Sisältöalue, johon säädin sijoitetaan -->
<div id="horizontalSlider" class="grid_9 push_3">
    <div id="sliderSteering"></div>
</div>
```

KUVA 21. Vaakasuoran liukusäätimen sijoitus

Tiedoston lopussa on jQuery-koodit säätimien arvojen lähetystä varten. Kuvassa 22 on nähtävissä servomoottorin ohjauskomennon lähetys. Säätimen, jonka tunniste on *sliderSteering*, pysähtyessä suoritetaan koodi, joka kutsuu Flask-sovelluksen Steering-funktiota. Funktiolle lähetetään säätimen sen hetkinen arvo, tallennettuna muuttujaan *s*. Tämän jälkeen verkkosivulle palautetaan funktion paluuarvo muuttujassa *steerFlask*, jonka arvo sijoitetaan tunnisteeseen *steer*.

```
<!-- Liukusäätimen pysähtyessä lähetetään sen arvo palvelimelle -->
<!-- Servomoottori -->
<script>
    $( "#sliderSteering" ).on( "slidestop", function( event, ui ) {
        $.getJSON('/_Steering', { // Kutsuttavan funktion URL (app.route)
            // sijoitetaan lähetettävään muuttujaan (s) säätimen arvo
            s: $('input[id="amount1"]').val(),
        }, function(data) {
            // Sijoitetaan funktion paluuarvo muttujaan #steer
            $("#steer").text(data.steerFlask);
        });
        return false;
    });
</script>
```

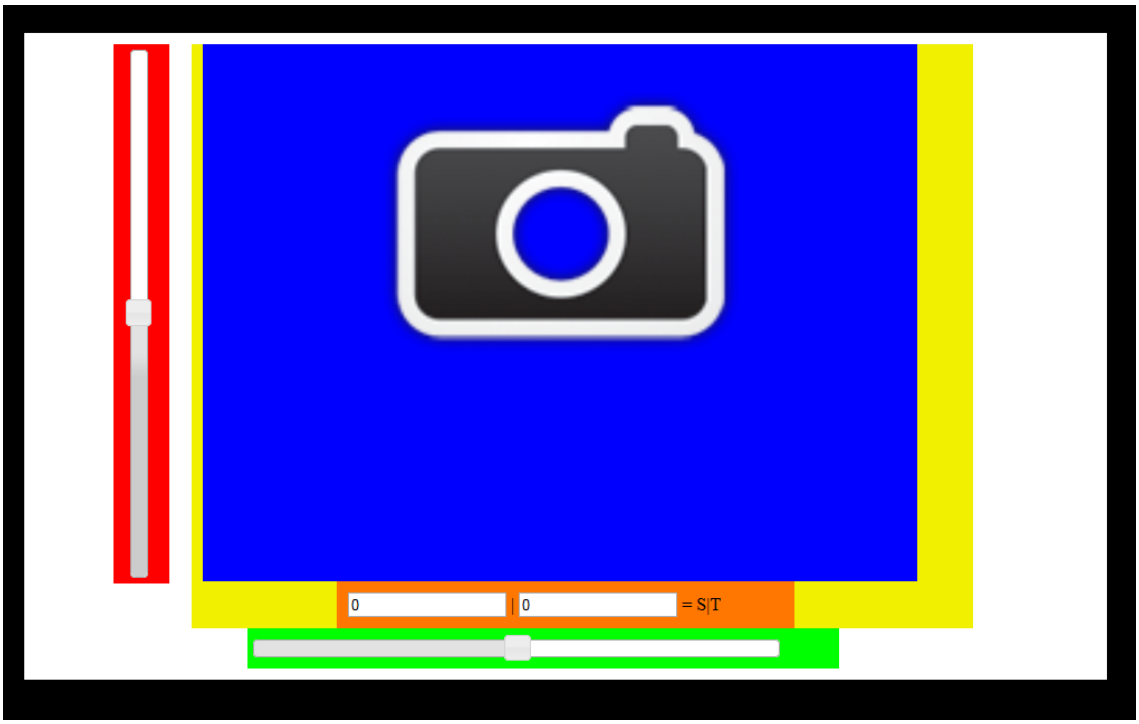
KUVA 22. Servomoottorin ohjauskomentojen lähetys

Lopuksi koodi toistetaan tavalliseen tapaan tasasähkömoottoria varten (kuva 23).

```
            <!-- Tasasähkömoottori -->
            <script>
                $( "#sliderThrottle" ).on( "slidestop", function( event, ui ) {
                    $.getJSON('/_Throttle', {
                        t: $('input[id="amount2"]').val(),
                    }, function(data) {
                        $("#thrt").text(data.thrtFlask);
                    });
                    return false;
                });
            </script>

        </div>
        <div class="clear"></div>
    </body>
</html>
```

KUVA 23. Tasasähkömoottorin ohjauskomentojen lähetys

Kehitystyön loppupuolella verkkosivu oli kuvan 24 kaltainen. Div-alueiden taustavärit määriteltiin style.css -tiedostossa erivärisiksi niiden sijoittelun helpottamiseksi.



KUVA 24. Verkkosivun kehitysversio

10 OHJAUSSIGNAALIN LUONTI

10.1 RPIO-moduuli

Kauko-ohjattavalle autolle välitettävien ohjaussignaaleiden luomiseen käytettiin RPIO-Python-moduulia, joka laajentaa Raspbian-käyttöjärjestelmän mukana tulevan RPi.GPIO-moduulin toimintaa. Tärkein RPIO-moduulin ominaisuus työn kannalta oli sen PWM-luokka, joka mahdollisti jopa 15 PWM-signaalin tuottamisen eri GPIO-pinneistä. Vakionahan Raspberryssa on vain yksi varsinainen PWM-pinni, GPIO-18, jonka signaali luodaan laitteistolla. Ohjelmistolla luotu signaali on puolestaan yksinkertaisimmillaan vain I/O-pinnien ulostulon vaihtelua HIGH- ja LOW-tilojen välillä ohjelmistokoodin avulla, mikä ei ole kovin tehokasta eikä tarkkaa.

RPIO-moduulin tuottama PWM-signaali on laitteistolla ja ohjelmistolla tuotettujen signaalien välimuoto, sillä moduuli käyttää hyväkseen DMA-ominaisuutta, eli Direct Memory Accessia. DMA:n avulla laitteiston osat voivat olla suoraan yhteydessä järjestelmän muistiin, eli toisin sanoen tehdä luku- ja kirjoitusoperaatioita ilman prosessorin toimimista välikätenä. Tämän ominaisuuden ansiosta säästyy prosessoritehoa ja PWM-signaaleita on mahdollista luoda jopa yhden mikrosekunnin tarkkuudella.

Tässä luvussa käydään läpi servomoottorin ohjauksesta vastaava Python-sovellus "SteeringRPIO" (liite 5), joka on lähes identtinen tasasähkömoottorin ohjauksesta vastaavan koodin (liite 6) kanssa, pois lukien DMA-kanavien ja GPIO-pinnien eri numeroinnit.

Koodin alussa kutsutaan tavalliseen tapaan siinä käytettyjä moduuleita ja niiden luokkia (kuva 25).

```
# -*- coding: utf-8 -*-
import sys, time
from RPIO import PWM
```

KUVA 25. Moduulien tuonti steeringRPIO- ja throttleRPIO-sovelluksiin

Tämän jälkeen alustetaan PWM-signaalit (kuva 26). Vaikka molemmille moottoreille onkin omat sovelluksensa täytyy signaaleita luoda kaksi kappaletta, koska kun moduu-

lin toimintaa mitattiin Fluken PM3380B-oskilloskoopilla (liite 7), havaittiin signaalin ajoitusten poikkeavan yhdellä aktiivisella DMA-kanavalla huomattavasti annetuista arvoista. Kahdella aktiivisella kanavalla signaalien ajoitukset olivat puolestaan kaksiskertaa annettuja arvoja suuremmat, mutta kunhan arvot jaettiin kahdella, olivat signaalit erittäin tarkkoja. PWM.init_channel -komennolla määritetään signaalin käyttämä DMA-kanava, joka voi olla välitä 0–14, ja signaalin jaksonaika mikrosekunneissa jaettuna kahdella.

```
#Alustetaan PWM-signaalit DMA-kanaville.
#Kanava 2 on turha ja sitä käytetään vain siksi,
#että signaaleista saataisiin tarkkoja;
#ajoitukset täytyy tosin jakaa kahdella.
PWM.setup()
PWM.init_channel(0, subcycle_time_us=10000)
PWM.init_channel(2, subcycle_time_us=10000)
```

KUVA 26. PWM-signaaleiden alustus

Signaaleiden alustuksen jälkeen vastaanotetaan flaskRPIO-sovelluksen lähettämä ohjauskomento, joka sijoitetaan muuttujaan *input* (kuva 27). Muuttujan sisältöä verrataan sen mahdollisiin arvoihin, jotka ovat siis samat kuin verkkosivun liukusäätimessä. Vertailun perusteella var1-muuttujaan sijoitetaan PWM-signaalin pulssin kesto mikrosekunneissa. Signaalin kesto on jaettu tavalliseen tapaan kahdella, mutta myös kymmenellä, PWM.setup() -komento nimittäin sisältää parametrin "pulse_width_incr", jonka arvo on vakiona kymmenen. Tätä jakajaa on tosin mahdollista muuttaa kirjoittamalla PWM.setup() -komennon sulkeiden sisään esimerkiksi "pulse_width_incr=1" tai "pulse_width_incr=50", jolloin signaaleita on mahdollista luoda vastaavasti yhden tai 50 mikrosekunnin tarkkuudella.

```
#Määritellään pulssin kesto saadun
#parametrin perusteella.
#Pulssin kesto on var1-muuttujan arvo
#kerrottuna PWM.setup:ssa määritellyn
#pulse_incr_us:n arvolla, joka on vakiona 10 us.
input = sys.stdin.readline()
var1 = 0

if(input == '-4'): #Vasemalle 4
    var1 = 55
elif(input == '-3'): #Vasemalle 3
    var1 = 60
elif(input == '-2'): #Vasemalle 2
    var1 = 65
elif(input == '-1'): #Vasemalle 1
    var1 = 70
elif(input == '0'): #Keskellä
    var1 = 75
elif(input == '1'): #Oikealle 1
    var1 = 80
elif(input == '2'): #Oikealle 2
    var1 = 85
elif(input == '3'): #Oikealle 3
    var1 = 90
elif(input == '4'): #Oikealle 4
    var1 = 95
```

KUVA 27. Vastaanotetun ohjauskomennon vertailu

Pulssin keston määrityksen jälkeen luodaan varsinainen PWM-signaali, mihin käytetään komentoa "PWM.add_channel_pulse()" (kuva 28). Komennon sulkeiden sisään kirjoitetaan käytettävä DMA-kanava, käytettävä GPIO-pinni, pulssin alkamishetki, joka täytyy myös jakaa kahdella ja kymmenellä, sekä pulssin kesto. Komennon luoma signaali on hyvin lyhytkestoinen, joten sitä pidennetään ajastetulla while-silmukalla. Silmukasta saa ajastetun, kun ennen silmukkaa tallennetaan muuttujaan systeemikellon aika ja jokaisella silmukan suorituskierroksella otetaan uusi aika, jota verrataan alkuperäiseen aikaan. Aikojen erotuksen ylittäessä tietyn ajan silmukan suoritus keskeytetään. Servon ohjaus ei vaadi yhtä pitkäkestoisia signaaleita kuin tasasähkömoottorin ohjaus.

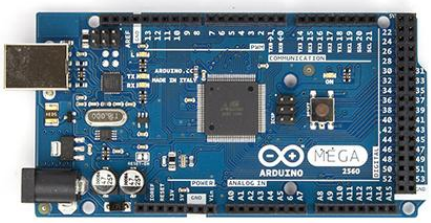
```
#Luodaan ajastettu while-silmukka signaalin luontia varten.
#PWM.add_channel_pulse(DMA-kanava, GPIO-pinni,
#pulssin alkuhetki / incr, pulssin kesto / incr)
oldTime = time.time()
while True:
    PWM.add_channel_pulse(0, 17, 0, var1)
    PWM.add_channel_pulse(2, 21, 0, 100)
    newTime = time.time()
    if((newTime - oldTime) > 0.2):
        break
```

KUVA 28. Signaalin luonti ajastetulla while-silmukalla

Ideaalinen tilanne, joka olisi vastannut kauko-ohjattavan auton alkuperäistä toimintaa, olisi ollut kaksi päättymätöntä while-silmukkaa, jotka olisi suoritettu rinnakkain. Pulssien kestojen muuttaminen "lennosta", eli kesken näiden silmukoiden suorituksen, osoittautui kuitenkin mahdottomaksi. Silmukoiden ajastuksen seurauksena auton liikuttaminen eteen- ja taaksepäin vaati liukusäätimen jatkuvaa painelua verkkosivulla, minkä etuna voidaan tosin pitää sitä, että verkkoyhteyden katketessa auto ei karkaa niin kuin se karkaisi jatkuvan signaalin tapauksessa.

10.2 Arduino Mega2560 -kehitysalusta

Ohjaussignaalit on mahdollista toteuttaa myös käyttämällä toista piirilevyä, joka laajentaa Raspberryn I/O-ominaisuuksia. Tässä vaihtoehtoisessa ratkaisussa käytettiin hyväksi Arduino Mega 2560 -kehitysalustaa (kuva 29), joka perustuu nimensä mukaisesti Atmelin ATMega2560-mikrokontrolleriin, ja joka tukee PWM-signaalin luomista useasta I/O-pinnistään. PWM-signaalien luontia Arduinolla ryhdyttiin tutkimaan ja kehittämään, kun hetken aikaa työn taustatukimusta tehdessä näytti siltä, ettei pelkällä Raspberry Piillä olisi ollut mahdollista tuottaa tarpeeksi montaa ja tarkkaa PWM-signaalia.



KUVA 29. Arduino Mega 2560 (Arduino)

Komentojen välitys Raspberrylta Arduinolle tapahtui USB-kaapelia pitkin, jonka kautta Arduino sai myös käyttövirtansa. Raspberryn puolella koodeja muutettiin niin, että ohjauskomennoiden luonnista vastaavista sovelluksista poistettiin while-silmukat ja RPIO-moduulin komennot. Käytännössä tämä tarkoitti vastaanotettavan muuttujan vertailun muokkaamista niin, että vertailun tuloksen perusteella sarjaväylään kirjoitettiin yhden tavun mittainen komento, eli yksi merkki. Arduino-versio kopioitiin luonnollisesti ensin omaan kansioonsa, joten Flask-sovelluksessa (liite 8) olevia tiedostopolkuja täytyi myös muuttaa.

Kuvassa 30 on esimerkkinä servomoottorin ohjauksesta vastaavien komentojen lähetys (liite 9. 1). Tasasähkömoottorille koodi on lähes sama, mutta siinä lähetettävät merkit ovat aakkosten kirjaimet k–s (liite 9. 2). Merkkien kirjoittaminen sarjaväylään vaati erillisen serial-moduulin, joka asennettiin kirjoittamalla Linux-terminaaliin komento "sudo apt-get python-serial". Serial-olion luonnissa näkyvä teksti "/dev/ttyACM0" tarkoittaa Raspberry Piin B-mallin ylempää USB-porttia.

```
# -*- coding: utf-8 -*-
import sys, serial
from RPIO import PWM

#Alustetaan USB-yhteys.
#serial.Serial(USB-portti, baud)
ser = serial.Serial('/dev/ttyACM0', 9600)

input = sys.stdin.readline()

elif(input == '-4'): #vasemalle 4
    ser.write('a')
elif(input == '-3'): #vasemalle 3
    ser.write('b')
elif(input == '-2'): #vasemalle 2
    ser.write('c')
elif(input == '-1'): #vasemalle 1
    ser.write('d')
elif(input == '0'): #keskellä
    ser.write('e')
elif(input == '1'): #oikealle 1
    ser.write('f')
elif(input == '2'): #oikealle 2
    ser.write('g')
elif(input == '3'): #oikealle 3
    ser.write('h')
elif(input == '4'): #oikealle 4
    ser.write('i')
```

KUVA 30. SteeringArduino.py

Arduino-kehitysalustan ohjelmointi vaatii erillisen kehitysympäristön, jonka voi ladata ilmaiseksi Arduinon verkkosivuilta. Koodi ladataan kehitysalustalle USB-kaapelin kautta. Ennen kuin aloittaa koodin kirjoittamisen, kannattaa asetuksista valita oikea Arduino-kehitysalusta ja COM-portti, johon USB-kaapeli on kytketty. Koodin kirjoitus tapahtuu C-kielen kaltaisella kielellä, joka on suhteellisen helposti opittavissa pelkästään tutkimalla ohjelmasta löytyviä esimerkkisovelluksia.

Jotta kehitysalustaan ladattava koodi (liite 10) toimisi halutulla tavalla, täytyi siihen lisätä setup-osioon komento, jolla muutettiin suoraan ATMega2560-piirin ajoitusrekistereitä (kuva 31). Kehitysalustan PWM-pinnien kellopulssin taajuus oli oletuksena pinnistä riippuen joko 490 Hz tai 980 Hz. Rekisteriä muuttamalla taajuudeksi saatiin noin 61 Hz pinneihin 4 ja 13, jolloin signaalin jaksonaika oli 16,4 ms, kun se oletustaajuudella olisi ollut noin 1 ms.

```
void setup()
{
  TCCR0B = TCCR0B & 0b11111000 | 0x05;
  /*
    http://playground.arduino.cc/Main/TimerPWMCheatsheet
    ---------------------------------------------
    timer 0 (controls pin 13, 4);
    ---------------------------------------------
    Setting     Divisor         Frequency
    0x01            1               62500
    0x02            8               7812.5
    0x03            64              976.5625
    0x04            256             244.140625
    0x05            1024            61.03515625

    TCCR0B = TCCR0B & 0b11111000 | <setting>;
  */
  Serial.begin(9600);
}
```

KUVA 31. ATMegan ajoitusrekisterin muuttaminen

PWM-signaali luodaan analogWrite-komenolla (kuva 32), vaikka käytössä onkin digitaaliset I/O-pinnit. PWM-signaalin tarkkuus määritellään jakamalla pinnin kellotaajuuden määrittämä jaksonaika 256:lla, mikä tässä tapauksessa tarkoitti sitä, että pulssin kesto voitiin määrittää 64 mikrosekunnin tarkkuudella. RPIO-moduulista poiketen Arduinon tuottama PWM-signaali on jatkuvaa ja näin ollen sisältää vaaran auton karkaamisesta verkkoyhteyden katketessa.

```
void loop()
{
  if (Serial.available() > 0) {
    //luetaan tavu sarjaväylästä
    char input = Serial.read();

    //ohjaus
    if(input == 'a'){ //vasemalle 4
      analogWrite(steer, 16); //1088
       Serial.println("left4");
    }
    else if(input == 'b'){ //vasemmalle 3
      analogWrite(steer, 18); //1216
      Serial.println("left3");
    }
```

KUVA 32. PWM-signaalin luonti Arduinolla

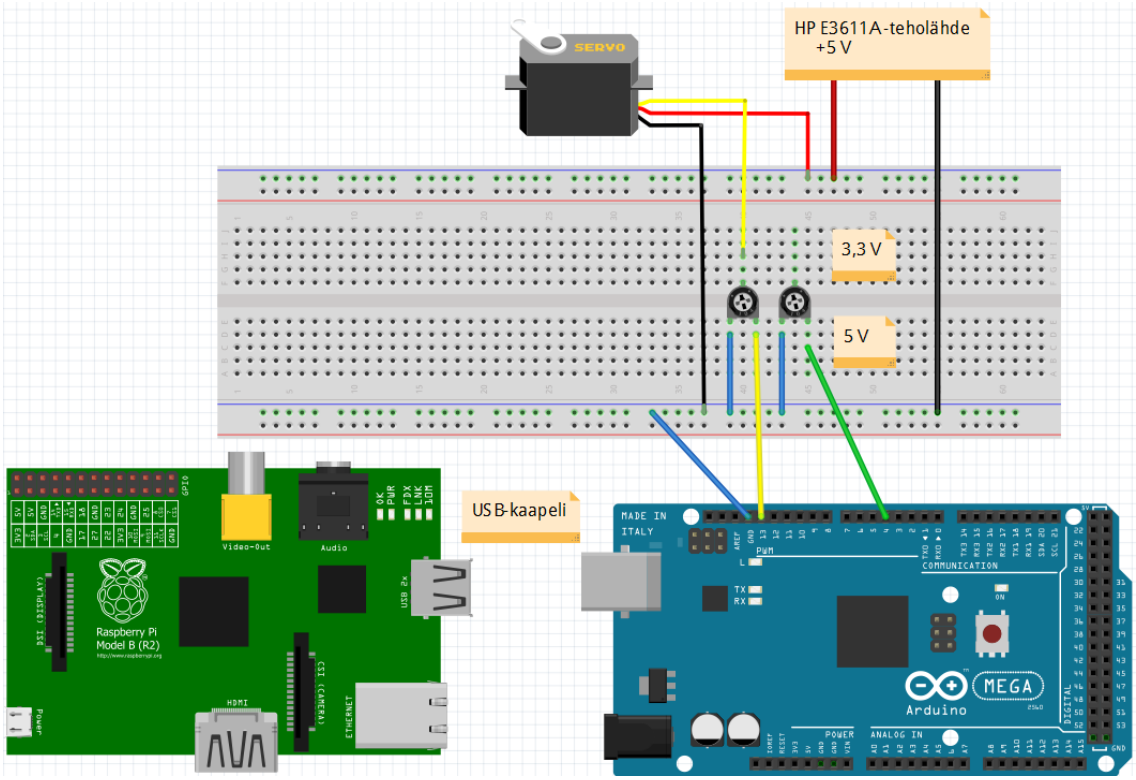
Arduinon koodin lisättiin myös Serial.println() -komento jokaisen vertailun tulokseen (kuva 32). Println-komento tulostaa sarjaväylään sulkujen sisään kirjoitetun tekstin ja lisää perään rivinvaihdon merkin "\r" ja uuden rivin aloituksen merkin "\n". Sarjaväylän lukemista varten Arduino-version Flask-sovellukseen lisättiin serial-moduulin kutsuminen ja serial-olion luonti samaan tapaan kuin steeringArduino-sovelluksessa sekä komento "print(ser.readline())" molempien subprocess-moduulin communicate-komentojen perään. Readline-komento lukee merkkejä, eli tavuja, sarjaväylästä rivinvaihtoon asti, kun pelkkä read-komento puolestaan lukisi vain yhden tavun.

Arduinonkaan toiminta ei ollut täysin virheetöntä, sillä kehitysalustan tuottamia signaaleita mitatessa (liite 11) havaittiin, että komento "analogWrite(13, 1) ei tuottanutkaan 64 mikrosekunnin kestoista pulssia pinnistä 13, vaan pulssin kesto oli 128 µs, joten signaalin jaksonajan jakajaa laskiessa täytyi noudattaa kaavaa 1.

$$Jakaja = \frac{pulssi(us)}{\frac{16400\ us}{256}} - 1 \quad (1)$$

On myös huomioitava, että Arduinon I/O-pinnien ulostulojännite on 5 V, eikä sellaisenaan sovi auton moottoreiden ohjaukseen, joka vaatii alhaisemman 3,3 V jännitteen. Yksikertainen ratkaisu tähän on käyttää yhden kilo-ohmin potentiometreja ja säätää ne niin, että ulostulojännite alenee 3,3 volttiin. Jännitettä on myös mahdollista laskea kytkemällä diodeja sarjaan, esimerkiksi kahdella 1N4007-diodilla, joiden kynnysjännite on 0,7 V, jännitteen voi pudottaa 5 voltista 3,6 volttiin. Kuvassa 33 on nähtävissä Arduino

Mega 2560 ja Raspberry kytkettyinä toisiinsa, potentiometrit ja erillinen servomoottori, jota käytettiin Arduinon toimintaa mitatessa.



KUVA 33. Arduino-testikytkentä

Työstä toteutettiin myös kolmas versio, jossa hyödynnettiin Raspberry Piin I2C-väylään kytkettyä servo-ohjainpiiriä. Piirin luomat signaalit olivat mittaustulosten perusteella kuitenkin epätarkkoja annettuihin arvoihin nähden, ja sovellus toimi tiedonsiirtoa lukuun ottamatta lähes samalla periaatteella kuin Arduinokin, joten kyseinen toteutus jätettiin tästä raportista pois.

11 JATKOKEHITYS

11.1 Kuvan välitys verkkosivulle

Kuten HTML-koodista oli jo havaittavissa, on sovellukseen sisällytetty mahdollisuus Raspberryn kameramoduulin kuvan suoratoistoa varten. Suoratoisto toteutettiin Miguel Motan ohjeiden (Mota, 2014) mukaan. Valitettavasti kuvan toisto toimi heikoimmillaan vain viisi sekuntia ja parhaimmillaan noin 25 minuuttia ennen kuin Raspberry verkkoyhteyksineen "kaatui". Syy tähän epävakauteen ei selvinnyt.

Suoratoisto olisi toimiessaan parantanut koko sovelluksen käyttäjäkokemusta huomattavasti, sillä se olisi mahdollistanut auton ohjaamisen näköyhteyden ulkopuolella, etenkin jos kamera olisi sijoitettu erikseen ohjattavalle alustalle.

11.2 Ohjaus Internetin välityksellä

Auton ohjaus on mahdollista toteuttaa lähiverkkoyhteyden sijaan myös Internet-yhteydellä. Yksinkertaisin ja käyttökelpoisin vaihtoehto Internet-ohjauksen toteuttamiseen on korvata WLAN-sovitin USB-porttiin liitettävällä mobiililaajakaistalla. Tämä kuitenkin vaatii joko kiinteän IP-osoitteen, jollaisia nykyään harvemmin myönnetään, tai dynaamisen IP-osoitteen nimipalvelun.

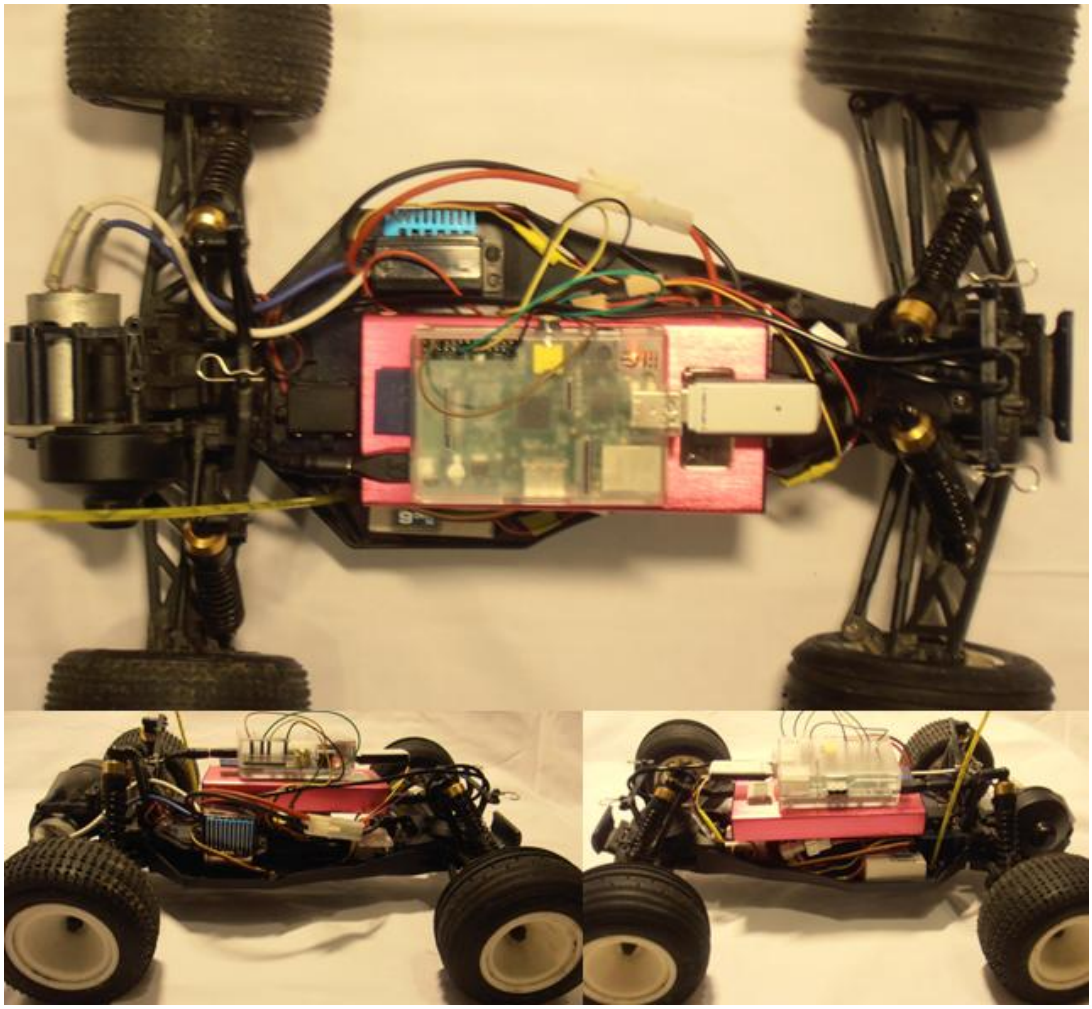
11.3 GPS-paikannus

Järjestelmään olisi myös mahdollista lisätä GPS-paikannus, jonka avulla olisi mahdollista nähdä auton kulkema reitti, viimeinen tunnettu sijainti sekä ehkä myös määrittää kuinka pitkän matkaa auton haluaa kulkevan eteen- tai taaksepäin.

12 LOPPUTULOS

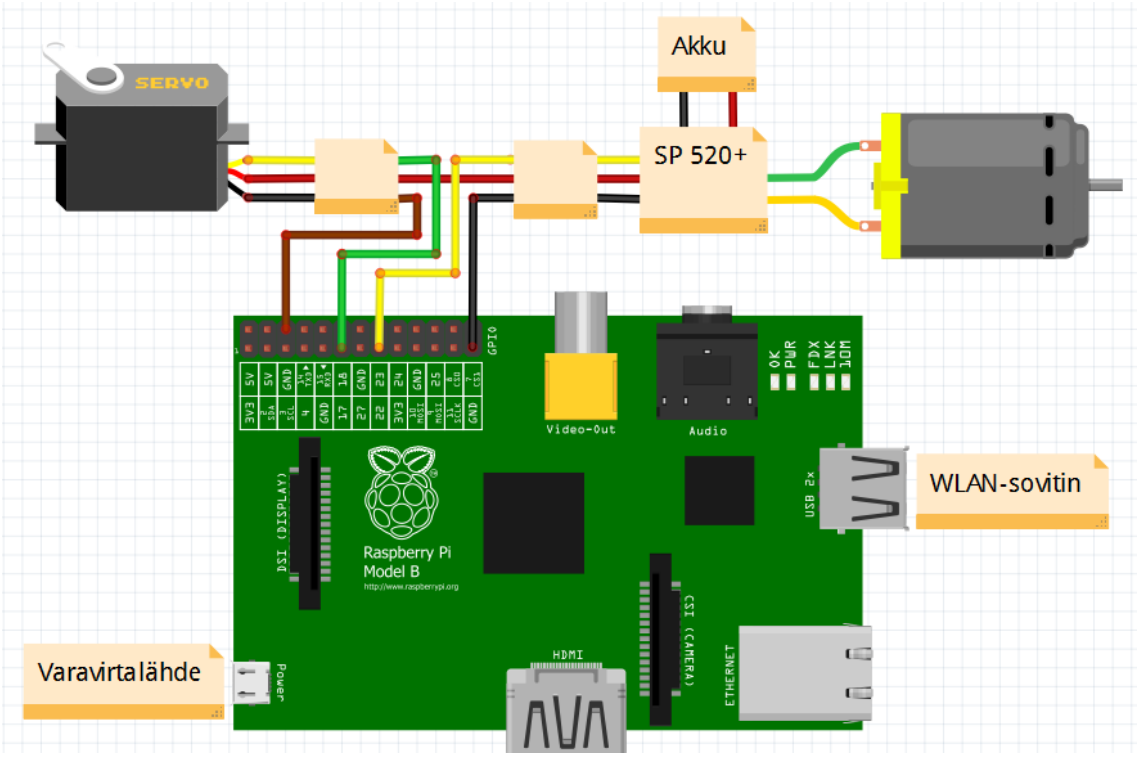
Sovelluksen toimivuus testattiin periaatteessa jo eri lähteistä tuotettuja PWM-signaaleita mitatessa, sillä ohjelmistokoodit kirjoitettiin näiden mittaustulosten perusteella. Lopputestaus koostuikin pääasiassa eri komponenttien sijoittamisesta kauko-ohjattavan auton sisään ja ohjauskomentoja vastaavien pulssien hienosäädöstä.

Raspberryn virtalähteenä toimi mobiililaitteille tarkoitettu 11 Ah:n varavirtalähde, jossa oli kaksi USB-porttia, joista toisen virtarajoitus oli 1 A ja toisen 2,1 A, ja josta pitäisi teoriassa riittää virtaa Raspberrylle noin 11–12 tunniksi täydellä latauksella. Varavirtalähde sijoitettiin alustalle auton oman akun päälle ja Raspberry puolestaan varavirtalähteen päälle (kuva 35). Kaikki lisätyt osat ovat irrotettavissa niin, että auton voi palauttaa takasin radio-ohjattavaksi. Auton kori mahtui juuri ja juuri paikalleen laitteiden sijoittamisen jälkeen.



KUVA 35. Langattomalla lähiverkkoyhteydellä ohjattava Kyosho EP Ultima ST

RPIO-moduulilla toteutetun järjestelmän lopullinen kytkentä oli siis kuvan 36 mukainen. Testauksen aikana havaittiin, että eteenpäin liikkuminen oli sujuvampaa kuin taaksepäin liikkuminen, joka vaatisi todennäköisesti hieman pidempikestoiset signaalit.

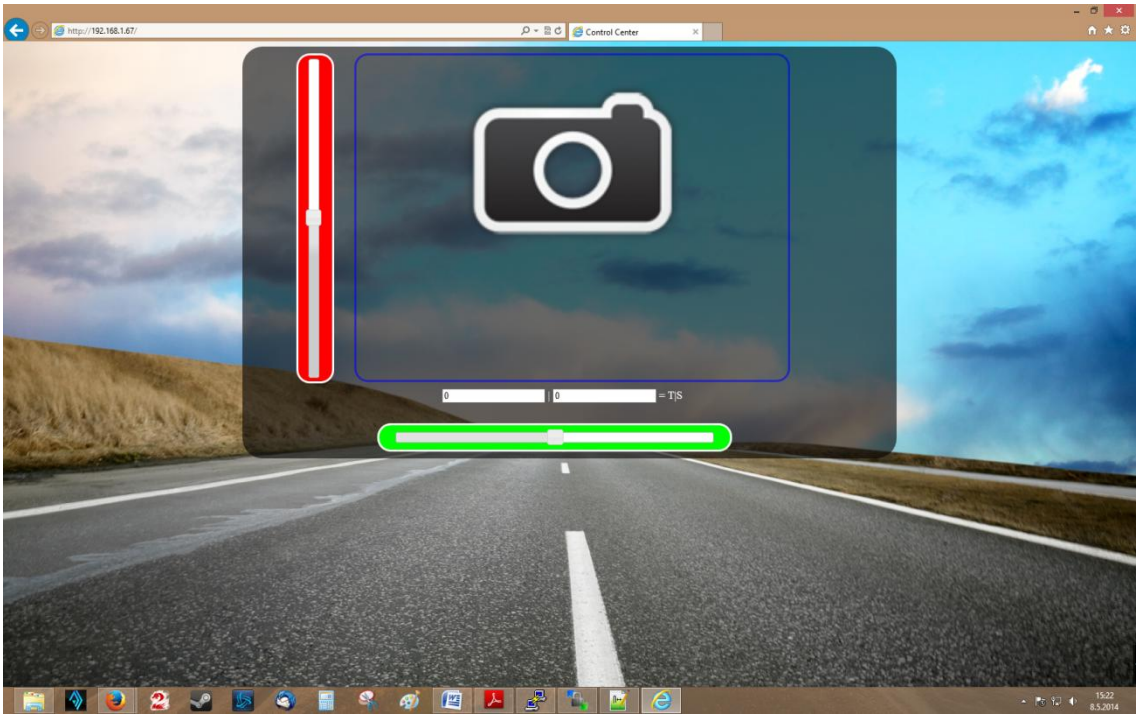


KUVA 36. RPIO-järjestelmän kytkentä

Arduino-kehitysalustalla toteutettua järjestelmää ei sijoitettu auton korin sisään tilan puutteen takia, joten sitä ei testattu ulkoilmassa. Arduino-version heikkous oli kuitenkin myös sen vahvuus, sillä RPIO-version purskittainen liike ei tuntunut kovinkaan luontevalta verrattuna Arduinon tuottamaan jatkuvaan ja tasaiseen liikkeeseen. Arduinolla toteutettu järjestelmä onkin näistä kahdesta versiosta kehityskelpoisempi, mutta vaatisi jonkinlaisen WLAN-yhteyden tarkistuksen tietyin väliajoin ilman, että järjestelmän ohjausviiveet kasvaisivat merkittävästi.

Lopullista sovellusta varten verkkosivun ulkoasua muutettiin myös kehitysversion sateenkaarenkirjavista laatikoista hieman hillitymmäksi (kuva 37). Verkkosivu testattiin toimivaksi Mozilla Firefox 29 ja Internet Explorer 10 -selaimilla sekä Nintendo 3DS XL ja Nokia Lumia 920 -laitteiden selaimilla. Sovelluksen tiedostokokoa olisi ollut mahdollista pienentää erittelemällä jQuery-kansiosta vain liukusäätimien luontiin tarvitut tiedostot. Koko sovellus vei noin 11 MB levytilaa, josta jQuery-kansion osuus oli 10

MB. Tallennustilan kannalta kyseessä kuitenkaan ollut mikään suuri sovellus, kun ottaa huomioon, että muistikortilla oli työn valmistuessa vielä yli 5 GB vapaata tallennustilaa.



KUVA 37. Lopullinen verkkosivu

Kuvassa 38 on nähtävissä ote molempien sovellusversioiden toiminnasta Linux-terminaalista nähtynä. Kuvassa ylempänä on RPIO-versio ja alempana Arduino-versio.

```
add_channel_pulse: channel=0, gpio=17, start=0, width=75
add_channel_pulse: channel=2, gpio=21, start=0, width=100
add_channel_pulse: channel=0, gpio=17, start=0, width=75
add_channel_pulse: channel=2, gpio=21, start=0, width=100
shutting down dma channel 0
clear_channel: channel=0
shutting down dma channel 2
clear_channel: channel=2
192.168.1.88 - - [08/May/2014 16:09:12] "GET /_Steering?s=0 HTTP/1.1" 200 -
```

```
192.168.1.88 - - [08/May/2014 16:36:01] "GET /_Steering?s=2 HTTP/1.1" 200 -
right3

192.168.1.88 - - [08/May/2014 16:36:02] "GET /_Steering?s=3 HTTP/1.1" 200 -
right4

192.168.1.88 - - [08/May/2014 16:36:03] "GET /_Steering?s=4 HTTP/1.1" 200 -
middle

192.168.1.88 - - [08/May/2014 16:36:05] "GET /_Steering?s=0 HTTP/1.1" 200 -
```

KUVA 38. Sovellukset Linux-terminaalista nähtynä.

13 POHDINTA

Kaiken kaikkiaan työ täytti kaikki sille asetetut tavoitteet, eli sitä tehdessä tuli opittua Python-ohjelmointikielen perusteita sekä tutustuttua työskentelyyn Linux-ympäristössä, mutta ennen kaikkea työn tuloksena oli toimiva sovellus, mikä olisi ollut lähes mahdotonta ilman tehtyjä oskilloskooppimittauksia ja niiden tuloksia.

Langattoman lähiverkon käytön tuoma etu alkuperäiseen radio-ohjaukseen verrattuna on mahdollisuus ohjata useampaa laitetta samalla alueella ilman, että kaikille laitteille täytyy määrittää oma taajuutensa, koska radio-ohjaukseen verrattuna järjestelmä on lähes immuuni häiriösignaaleille, mutta toisaalta WLAN-järjestelmän ohjausviiveet saattavat tuottaa ongelmia tietyissä tilanteissa. Mahdollisia käyttökohteita sovellukselle viihdekäytön lisäksi voisivat olla muun muassa kiinteistöjen tai muiden kohteiden etätarkastus, kunhan kamerakuvan suoratoiston saa toimimaan luotettavasti. Järjestelmästä olisi myös mahdollista tehdä täysin itsenäinen korvaamalla kauko-ohjaus erilaisilla sensoreilla.

Työ toteutettiin olemassa olevia laitteita hyödyntäen, mutta jälkiviisautena työn kannalta parempi valinta ohjattavaksi laitteeksi olisi ollut robottialusta askel- tai vaihteistomoottoreilla, jotka olisivat tehneet eteen- ja taaksepäin liikkumisesta paremmin hallittavaa. Nykyinen järjestelmä vaatii reilusti tilaa ympärilleen suhteellisen suuritehoisen tasasähkömoottorin takia, mutta servomoottorin ohjaukseen sovellus sopi kuitenkin erinomaisesti.

LÄHTEET

Arduino. Mega2560. Luettu 03.05.2014
http://arduino.cc/en/Main/arduinoBoardMega2560

Elinux.org. 2014a. Raspberry Pi Low-level Peripherals. Luettu 15.04.2014
http://elinux.org/RPi_Low-level_peripherals

Elinux.org. 2014b. Raspberry Pi Hardware History. Luettu 16.04.2014
http://elinux.org/RPi_HardwareHistory

Elinux.org. 2014c. Raspberry Pi Distributions. Luettu 16.04.2014
http://elinux.org/RPi_Distributions

Flask. 2014a Foreword. Luettu 19.04.2014
http://flask.pocoo.org/docs/foreword

Flask. 2014b. Advanced Foreword. Luettu 19.04.2014
http://flask.pocoo.org/docs/advanced_foreword

Flask. 2014c. Quickstart. Luettu 03.05.2014
http://flask.pocoo.org/docs/quickstart/

Hirzel, T. PWM. Luettu 19.04.2014
http://arduino.cc/en/Tutorial/PWM

Hitec. SP-520 Manual. Käyttöohje. Tallennettu 11.04.2014
http://hitecrcd.com/files/ManualSP520.pdf

Linux.fi. 2013. LXDE. Luettu 11.04.2014.
http://linux.fi/wiki/LXDE

Maruch, A. & Maruch, S. 2006. Python for dummies. Yhdysvallat: John Wiley & Sons.

Modmypi.com. 2013. Remotely Accessing the Raspberry Pi via SSH – Console Mode
Luettu 04.05.2014
https://www.modmypi.com/blog/tutorials/remotely-accessing-the-raspberry-pi-via-ssh-console-mode

Mota, M. 2014. Raspberry Pi camera board video streaming. Luettu 30.04.2014
http://www.miguelmota.com/blog/raspberry-pi-camera-board-video-streaming/

Python.org. FAQ. Luettu 19.04.2014
https://docs.python.org/2.7/faq/general.html

Raspberrypi.org. 2014a. About us. Luettu 04.04.2014
http://www.raspberrypi.org/about/

Raspberrypi.org. 2014b. FAQ. Luettu 04.04.2014
http://www.raspberrypi.org/help/faqs/

Raspberrypi.org. 2014c. Downloads. Luettu 04.05.2014
http://www.raspberrypi.org/downloads/

Raspberrypi.org. 2014d. Installing images. Luettu 04.05.2014
http://www.raspberrypi.org/documentation/installation/installing-images/

Raspbian.org. Raspbian. Luettu 16.04.2014
http://www.raspbian.org/

Readwrite.com. 2014. Raspberry Pi: Everything You Need To Know.
Luettu 18.04.2014
http://readwrite.com/2014/01/20/raspberry-pi-everything-you-need-to-know

Shirriff, K. 2009. Secrets of Arduino PWM. Luettu 19.04.2014
http://www.righto.com/2009/07/secrets-of-arduino-pwm.html

LIITTEET

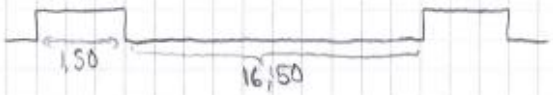
Liite 1. Mittauspöytäkirja, kauko-ohjattavan auton toiminta

1 (3)

Mittauspöytäkirja 10.3.2014 Arjpekka Verne

Kauko-ohjattavan auton toiminta

Ohjausservon signaalit jakson aika 18,0 ms



Lepotila ~1,50 ms
äärioikea ~2,00 ms
äärivasen ~1,05 ms

max trim. oikealle

Lepotila 1,74 ms
oikea 2,27 ms raja 2,00 ms
vasen 1,23

max trim vasemmalle

Lepotila 1,35 ms
oikea 1,89 ms
vasen 0,85 ms raja 1,00 ms

Käyttöjännite 5,83 V (punainen)
Pulssin amplitudi 3,3 V (keltainen)

Fluke PM3380B 100 MHz Combiscope

Hewlett Packard 8110A Pulse- / Pattern Generator
Hewlett Packard E3611A DC Power Supply
5V riittävä käyttöjännite. Virtarajoitus vähintään 300mA servolle (enkaat ilmassa)
Vasen 1,00ms Keskellä 1,50ms Oikea 2,00ms 500mA naatalaskeen

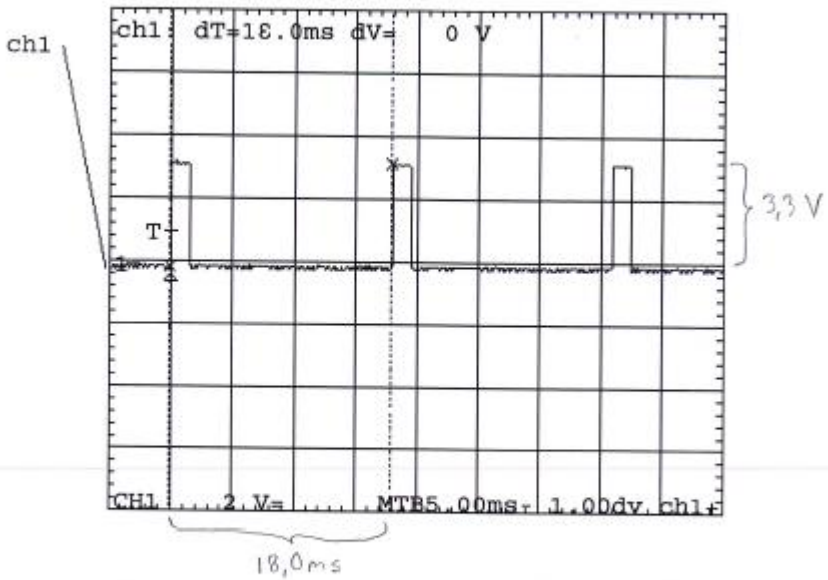
"Kaasun" signaalit samat kuin ohjauksessa 1,0 - 2,0 ms 18,0 ms
Neutraali 1,505ms - 1,555ms 1,500ms peruuttaa!

Jaksonajalla ei suurta merkitystä 20,0 ms toimii yhtä hyvin.
16,0 ms
10,0 ms
30,0 ms

(jatkuu)

Kauko-ohjattavan auton PWM-signaalin jaksonaika

PM3380B

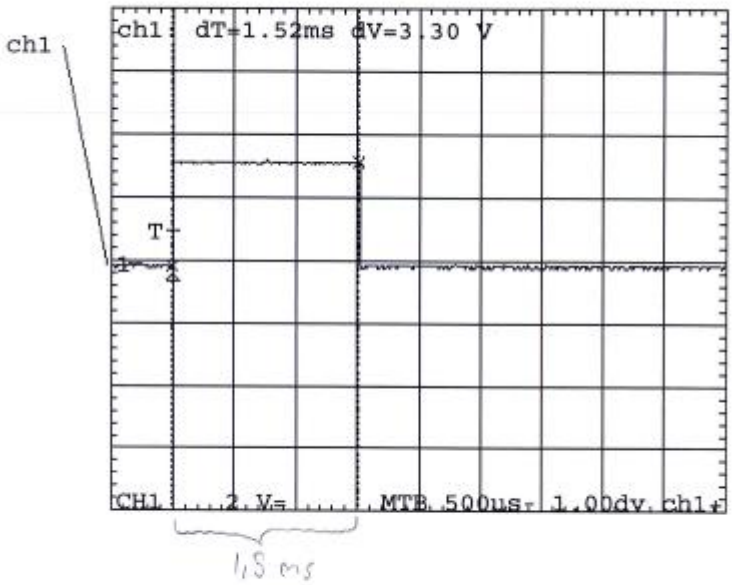


Y/Div:	Timebase:	TRACE	Trigger time & date	
2.00 V	5.00ms	ch1	13:40:31:29	10-03-2014

Time of hardcopy: 13:40:31 10-03-2014

PWM-signaalin "neutraali" pulssi

PM3380B

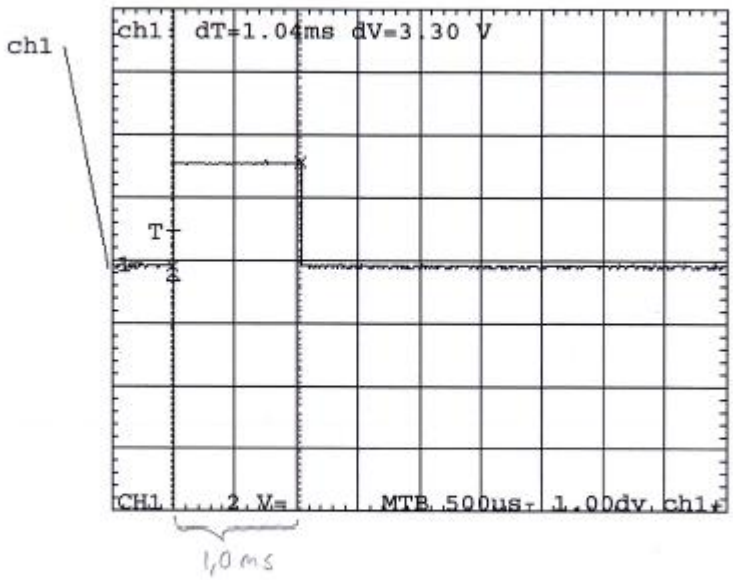


Y/Div:	Timebase:	TRACE	Trigger time & date	
2.00 V	500us	ch1	13:32:36:99	10-03-2014

Time of hardcopy: 13:32:37 10-03-2014

PWM-signaalin minimipulssi (vasen/taakse)

PM3380B

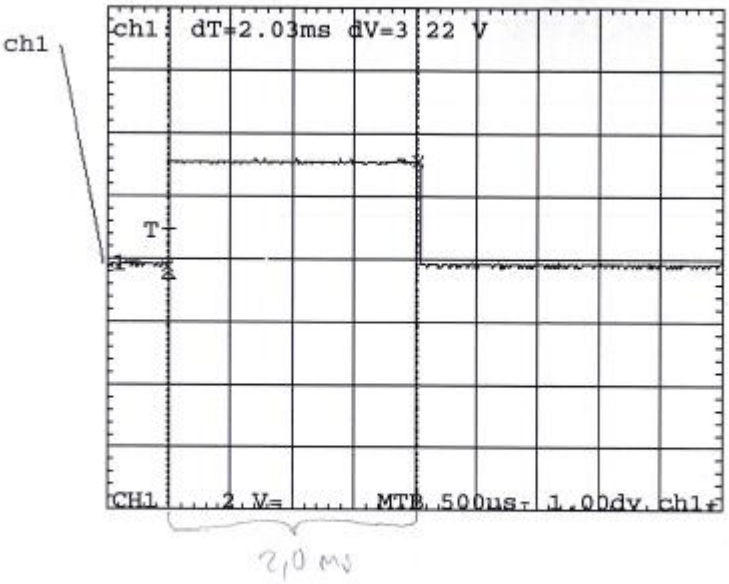


Y/Div:	Timebase:	TRACE	Trigger time & date
2.00 V	500us	ch1	13:38:20:04 10-03-2014

Time of hardcopy: 13:38:20 10-03-2014

PWM-signaalin maksimipulssi (oikea/eteen)

PM3380B



Y/Div:	Timebase:	TRACE	Trigger time & date
2.00 V	500us	ch1	13:36:45:22 10-03-2014

Time of hardcopy: 13:36:45 10-03-2014

Liite 2. RPIO-version Flask-sovelluksen lähdekoodi

flaskRPIO.py

```
# -*- coding: utf-8 -*-
#Kutsutaan koodissa käytettyjä kirjastoja ja niiden luokkia.
from flask import Flask, render_template, request, jsonify
import subprocess

#Alustetaan Flask-sovellus
app = Flask(__name__)

#Käynnistetään kamera-stream
#args3 = ['sh', '/home/pi/flaskRPIO/start_stream.sh']
#proc3                    =                    subprocess.Popen(args3,
stdout=subprocess.PIPE).communicate()[0]

#Määritellään verkkosivun luonnissa käytettävä html-sivupohja,
#joka sijaitsee sovelluskansion templates-kansiossa.
@app.route('/')
def index():
    return render_template('index.html')

#Servomoottorin ohjauskomentoja vastaanottava funktio.
#App.route:lla määritetään nimi,
#jolla funktiota voi kutsua html-sivupohjassa.
@app.route('/_Steering')
def Steering():

    #Määritellään kutsuttavan ohjelman tyyppi ja sijainti
    #ja kutsutaan sitä.
    args1 = ['python', '/home/pi/flaskRPIO/SteeringRPIO.py']
    proc1 = subprocess.Popen(args1, stdin=subprocess.PIPE)

    #Vastaanotetaan verkkosivulta muuttuja "s",
    #joka sisältää ohjauskomennon.
    s = request.args.get('s')

    #Lähetetään saatu komento "aliohjelmalle".
    #Lähetettävän muuttujan täytyy olla string-muodossa.
    proc1.communicate(s)

    #Palautetaan saatu komento takasin verkkosivulle
    #debuggausta varten.
    return jsonify(steerFlask = s)
```

(jatkuu)

```
#Tasasähkömoottorin ohjauskomentoja vastaanottava funktio.
#Toimii samalla periaatteella kuin Steering-funktio.
@app.route('/_Throttle')
def Throttle():

     args2 = ['python', '/home/pi/flaskRPIO/ThrottleRPIO.py']
     proc2 = subprocess.Popen(args2, stdin=subprocess.PIPE)

     t = request.args.get('t')
     proc2.communicate(t)

     return jsonify(thrtFlask = t)

#Käynnistetään sovellus toimimaan Raspberryn IP-osoitteessa.
if __name__ == '__main__':
    app.run(
        host="0.0.0.0",
        port=int("80"),
        debug=False
    )
```

Liite 3. HTML-sivupohjan lähdekoodi

index.html

```
<!DOCTYPE html>
<html lang="fi">
     <head>

          <!-- Sivun nimi, näkyy välilehdessä -->
          <title>Control Center</title>

          <!-- Sivuston merkistö -->
          <meta charset="UTF-8"/>

          <!-- Css-muotoilutiedostot -->
          <!-- jQuery teema slidereita ja 960 asettelua varten
(http://960.gs/) -->
          <!-- style.ccs sisältää sivun ulkoasun muotoilut -->
          <link rel=stylesheet type=text/css href="{{ url_for('static',
filename='jq/themes/base/jquery.ui.all.css') }}">
          <link rel=stylesheet type=text/css href="{{ url_for('static',
filename='960_16_col.css') }}">
          <link rel=stylesheet type=text/css href="{{ url_for('static',
filename='style.css') }}">

          <!-- jQuery-kirjastot -->
          <script type="text/javascript"   src="{{    url_for('static',
filename='jq/jquery-1.10.2.js') }}"></script>
          <script type="text/javascript"   src="{{    url_for('static',
filename='jq/ui/jquery-ui.js') }}"></script>

          <!-- Liukusäätimien alustus -->
          <script>
               $(function() {
               $( "#sliderSteering" ).slider({ // Säätimen nimi
               range: "min",
               value: 0, // Oletusarvo
               min: -4, // Säätimen ääriarvot
               max: 4,
               slide: function( event, ui ) { // Liutettaessa otetaan
säätimen senhetkinen arvo
                    $( "#amount1" ).val( ui.value );   // Sijoitetaan
arvo tunnisteeseen amount1
               }
               });
               $( "#amount1" ).val( $( "#sliderSteering" ).slider(
"value" ) );
               });
          </script>

          <script>
               $(function() {
               $( "#sliderThrottle" ).slider({
```

(jatkuu)

```
                orientation: "vertical",
                range: "min",
                value: 0,
                min: -4,
                max: 4,
                slide: function( event, ui ) {
                    $( "#amount2" ).val( ui.value );
                }
                });
                $( "#amount2" ).val( $( "#sliderThrottle" ).slider(
"value" ) );
                });
          </script>

     </head>

     <!-- Verkkosivun sisältö-->
     <body>
          <div id="main" class="container_16">

               <!-- Sisältöalue, johon säädin sijoitetaan -->
               <div id="verticalSlider" class="grid_1 push_1">
                    <!-- Pystysuuntainen kaasusäädin vasemmassa reunassa
-->
                    <div id="sliderThrottle"></div>
               </div>

               <div id="content" class="grid_12 push_1">
                    <!-- Sisältöalue kamerakuvalle -->
                    <div id="cameraFeed" class="grid_8">
                         <!-- Stream -->
                         <object
data="http://192.168.1.67:9000/?action=stream" width=100%>
                              <!-- Jos kamera-streamia ei löydy, lada-
taan place holder -kuva, jos sitäkään ei löydy kirjoitetaan virhevies-
ti -->
                              <img src="{{ url_for('static', file-
name='img/camera.png') }}" alt="Stream not found" class="backup")>
                         </object>
                         <!-- Erityisesti mobillilaitteiden jotkin kevy-
et selaimet eivät tue objectia, joten mikäli käyttä moista
                              niin alla olevan rivin voi poistaa kom-
menteista ja yllä olevat 3 riviä kommentoida pois. -->
                         <!--      <img      style="width:      100%;"
src="http://192.168.1.67:9000/?action=stream" alt="Stream not found">
-->
                    </div>

                    <div id="controlInfo" class="grid_7 push_2">
                         <!-- Säätimien senhetkiset arvot -->
                         <input type="text" id="amount2"/> |
                         <input type="text" id="amount1"/> =
                         <!-- Flask-sovelluksen paluuarvot -->
                         <span                  id="thrt">T</span>|<span
id="steer">S</span>
                    </div>
               </div>

               <!-- Sisältöalue, johon säädin sijoitetaan -->
               <div id="horizontalSlider" class="grid_9 push_3">
```

```
                    <div id="sliderSteering"></div>
                </div>

                <!-- Liukusäätimen pysähtyessä lähetetään sen arvo pal-
velimelle -->
                <!-- Servomoottori -->
                <script>
                    $( "#sliderSteering" ).on( "slidestop", function(
event, ui ) {
                        $.getJSON('/_Steering', { // Kutsuttavan funk-
tion URL (app.route)
                            //  sijoitetaan  lähetettävään  muuttujaan
(s) säätimen arvo
                            s: $('input[id="amount1"]').val(),
                        }, function(data) {
                            // Sijoitetaan funktion paluuarvo muttu-
jaan #steer
                            $("#steer").text(data.steerFlask);
                        });
                        return false;
                    });
                </script>

                <!-- Tasasähkömoottori -->
                <script>
                    $( "#sliderThrottle" ).on( "slidestop", function(
event, ui ) {
                        $.getJSON('/_Throttle', {
                            t: $('input[id="amount2"]').val(),
                        }, function(data) {
                            $("#thrt").text(data.thrtFlask);
                        });
                        return false;
                    });
                </script>

            </div>
            <div class="clear"></div>

        </body>
</html>
```

Liite 4. CSS-muotoilutiedoston lähdekoodi

style.css

```
body
{
     background-image: url(../static/img/Highway.jpg);
     /*lähde: http://www.tlxtransport.com/wp-content/uploads/
       2011/07/Highway-to-Heaven.jpg*/
     background-size: 100%;
     background-color: #000000;
    background-repeat:no-repeat;
     /*määrittelemättömän fontin paksuus*/
    font-weight: normal;
     color: #FFFFFF; /*...ja väri*/
}

/*kaikki 960-järjestelmää käyttävät divit sisältävät
     automaattisesti "position: relative;"*/

/*pääkehysalue*/
#main
{
     /*taustanväri, backup*/
     background-color: rgb(0, 0, 0);
     /*läpinäkyvä tausta*/
     background-color: rgba(0, 0, 0, 0.6);
     padding-bottom: 10px;
     padding-top: 10px;
     padding-left: 10px;
     border-radius: 30px; /*kulmien pyöristys*/
     top: 20px; /*irrotetaan yläreunasta*/
}

/* sis. camera ja info*/
#content
{}

/*kaasusäätimen kehys*/
#verticalSlider
{
     background-color: #FF0000;
     border-style: solid;
     border-width: medium;
     border-color: #FFFFFF;
     border-radius: 20px;
     height: relative;
     padding-top: 10px;
     padding-left: 10px;
}

/*kaasusäädin*/
#sliderThrottle
{
     left: 5px;
     bottom: 5px;
     height: 470px;
}
```

(jatkuu)

```
/*ohjaussäätimen kehys*/
#horizontalSlider
{
     background-color: #00FF00;
     border-style: solid;
     border-width: medium;
     border-color: #FFFFFF;
     border-radius: 20px;
     padding-bottom: 10px;
     padding-top: 10px;
     margin-top: 20px;
}

/*ohjaussäädin*/
#sliderSteering
{
     width: 470px;
     margin-left: auto;
     margin-right: auto;
}

/*kamerakuvan kehys*/
#cameraFeed
{
     width: 640px;
     height: 480px;
     border-style: solid;
     border-width: medium;
     border-color: rgb(0, 0, 255);
     border-color: rgba(0, 0, 255, 0.6);
     border-radius: 20px;
}

/*säädinarvojen kehys*/
#controlInfo
{
     padding-bottom: 10px;
     padding-top: 10px;
     padding-left: 10px;
}

/*keskitetään "error-kuva"*/
img.backup
{
     width: 50%;
     display: block;
     margin-left: auto;
     margin-right:auto;
}
```

Liite 5. RPIO-version servomoottorin ohjauksen lähdekoodi

steeringRPIO.py

```
# -*- coding: utf-8 -*-
import sys, time
from RPIO import PWM

#Alustetaan PWM-signaalit DMA-kanaville.
#Kanava 2 on turha ja sitä käytetään vain siksi,
#että signaaleista saataisiin tarkkoja;
#ajoitukset täytyy tosin jakaa kahdella.
PWM.setup()
PWM.init_channel(0, subcycle_time_us=10000)
PWM.init_channel(2, subcycle_time_us=10000)

#Määritellään pulssin kesto saadun
#parametrin perusteella.
#Pulssin kesto on var1-muuttujan arvo
#kerrottuna PWM.setup:ssa määritellyn
#pulse_incr_us:n arvolla, joka on vakiona 10 us.
input = sys.stdin.readline()
var1 = 0

if(input == '-4'): #Vasemalle 4
     var1 = 55
elif(input == '-3'): #Vasemalle 3
     var1 = 60
elif(input == '-2'): #Vasemalle 2
     var1 = 65
elif(input == '-1'): #Vasemalle 1
     var1 = 70
elif(input == '0'): #Keskellä
     var1 = 75
elif(input == '1'): #Oikealle 1
     var1 = 80
elif(input == '2'): #Oikealle 2
     var1 = 85
elif(input == '3'): #Oikealle 3
     var1 = 90
elif(input == '4'): #Oikealle 4
     var1 = 95

#Luodaan ajastettu while-silmukka signaalin luontia varten.
#PWM.add_channel_pulse(DMA-kanava, GPIO-pinni,
#pulssin alkuhetki / incr, pulssin kesto / incr)
oldTime = time.time()
while True:
     PWM.add_channel_pulse(0, 17, 0, var1)
     PWM.add_channel_pulse(2, 21, 0, 100)
     newTime = time.time()
     if((newTime - oldTime) > 0.2):
          break
```

Liite 6. RPIO-version tasasähkömoottorin ohjauksen lähdekoodi

throttleRPIO.py

```
# -*- coding: utf-8 -*-
import sys, time
from RPIO import PWM

#Alustetaan PWM-signaalit DMA-kanaville.
#Kanava 2 on turha ja sitä käytetään vain siksi,
#että signaaleista saataisiin tarkkoja;
#ajoitukset täytyy tosin jakaa kahdella.
PWM.setup()
PWM.init_channel(1, subcycle_time_us=10000)
PWM.init_channel(2, subcycle_time_us=10000)

#Määritellään pulssin kesto saadun
#parametrin perusteella.
#Pulssin kesto on var1-muuttujan arvo
#kerrottuna PWM.setup:ssa määritellyn
#pulse_incr_us:n arvolla, joka on vakiona 10 us.
input = sys.stdin.readline()
var1 = 0

if(input == '-4'): #Taakse 4
     var1 = 55
elif(input == '-3'): #Taakse 3
     var1 = 60
elif(input == '-2'): #Taakse 2
     var1 = 65
elif(input == '-1'): #Taakse 1
     var1 = 70
elif(input == '0'): #Seis
     var1 = 77
elif(input == '1'): #Eteen 1
     var1 = 80
elif(input == '2'): #Eteen 2
     var1 = 85
elif(input == '3'): #Eteen 3
     var1 = 90
elif(input == '4'): #Eteen 4
     var1 = 95

#Luodaan ajastettu while-silmukka signaalin luontia varten.
#PWM.add_channel_pulse(DMA-kanava, GPIO-pinni,
#pulssin alkuhetki / incr, pulssin kesto / incr)
oldTime = time.time()
while True:
     PWM.add_channel_pulse(1, 22, 0, var1)
     PWM.add_channel_pulse(2, 21, 0, 100)
     newTime = time.time()
     if((newTime - oldTime) > 0.3):
          break
```

Liite 7. Mittauspöytäkirja, RPIO-moduulin toiminta

Mittauspöytäkirja 7.4.2014 Aripekka Vorne

RPIO - Python-moduulin toiminta

Fluke PM3380B 100 MHz Combiscope

subcycle_time_us = 16000 = 18,0 ms jakso.
add_channel_pulse(1, 22, 0, 125) = 1,5 ms pulssi

...pulse(..120) = 1,35 ms
pulse(130) = 1,45 ms
140 = 1,55 ms
150 = 1,67 ms
160 = 1,79 ms
170 = 1,90 ms
180 = 2,02 ms
175 = 1,96 ms

110 = 1,22 ms
100 = 1,12 ms
90 = 1,00 ms
95 = 1,06 ms

Kahdella pulssilla jakson aika kaksinkertaistuu ja myös pulssin kesto

Parittomat tuhat-luvut subcycle_us_time:sa eivät toimi
(9000, 11000, 15000 jne.)

95 = 1,90 ms

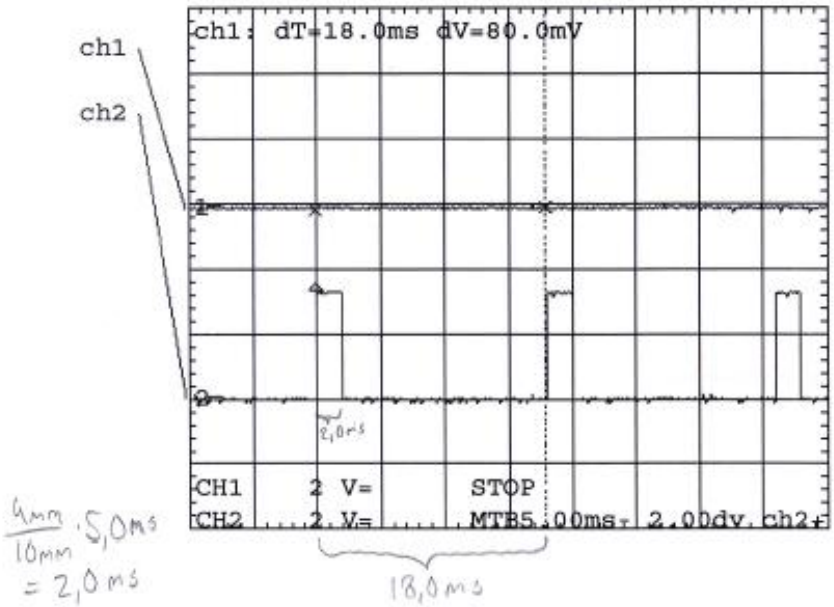
55 = 1,10 ms 50 mahdollinen

```
from RPIO import PWM

PWM.setup()
PWM.init_channel(0, subcycle_time_us=10000)
PWM.init_channel(1, subcycle_time_us=10000)

while True:
    PWM.add_channel_pulse(0, 21, 0, 55)
    PWM.add_channel_pulse(1, 22, 0, 95)
```

(jatkuu)

RPI0: Yksi aktiivinen DMA-kanava

PM3380B



Y/Div:	Timebase:	TRACE	Trigger time & date	
2.00 V	5.00ms	ch1	12:55:19:19	07-04-2014
2.00 V	5.00ms	ch2	12:55:19:19	07-04-2014

Time of hardcopy: 12:56:04 07-04-2014

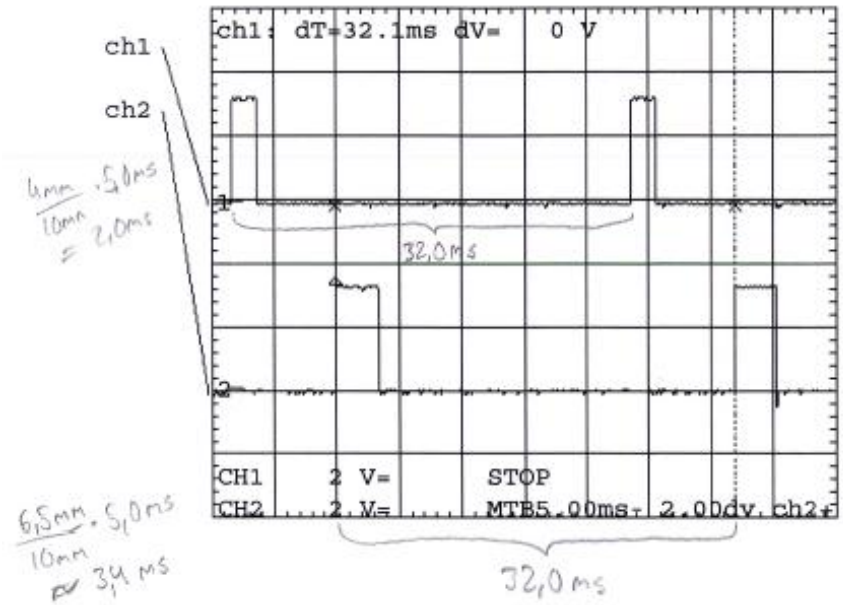
CH2:

init_channel(1, 16000) → 16,0ms ≠ 18,0ms

pulse(1, 21, 0, 170) → 1,7ms ≠ 2,0ms

RPIO: Kaksi aktiivista DMA-kanavaa

PM3380B



Y/Div:	Timebase:	TRACE	Trigger time & date	
2.00 V	5.00ms	ch1	13:02:27:54	07-04-2014
2.00 V	5.00ms	ch2	13:02:27:54	07-04-2014

Time of hardcopy: 13:02:33 07-04-2014

CH1:

init_channel(0, 16000) → 16,0ms = 32,0ms / 2

pulse(0, 22,0, 100) → 1,0ms = 2,0ms / 2

CH2:

init_channel(1, 16000) → 16,0 ms = 32,0ms / 2

pulse(1, 21,0, 170) → 1,7ms = 3,4ms / 2

Liite 8. Arduino-version Flask-sovelluksen lähdekoodi

1 (2)

flaskArduino.py

```
# -*- coding: utf-8 -*-
#Kutsutaan koodissa käytettyjä kirjastoja ja niiden luokkia.
from flask import Flask, render_template, request, jsonify
import subprocess, serial

ser = serial.Serial('/dev/ttyACM0', 9600)

#Käynnistetään kamera-stream
#args3 = ['sh', '/home/pi/flaskRPIO/start_stream.sh']
#proc3=subprocess.Popen(args3,stdout=subprocess.PIPE).communicate()[0]

#Alustetaan Flask-sovellus
app = Flask(__name__)

#Määritellään verkkosivun luonnissa käytettävä html-sivupohja,
#joka sijaitsee sovelluskansion templates-kansiossa.
@app.route('/')
def index():
    return render_template('index.html')

#Servomoottorin ohjauskomentoja vastaanottava funktio.
#App.route:lla määritetään nimi,
#jolla funktiota voi kutsua html-sivupohjassa.
@app.route('/_Steering')
def Steering():

    #Määritellään kutsuttavan ohjelman tyyppi ja sijainti
    #ja kutsutaan sitä.
    args1 = ['python', '/home/pi/flaskArduino/steeringArduino.py']
    proc1 = subprocess.Popen(args1, stdin=subprocess.PIPE)

    #Vastaanotetaan verkkosivulta muuttuja "s",
    #joka sisältää ohjauskomennon.
    s = request.args.get('s')

    #Lähetetään saatu komento "aliohjelmalle".
    #Lähetettävän muuttujan täytyy olla string-muodossa.
    proc1.communicate(s)

    print(ser.readline())

    #Palautetaan saatu komento takasin verkkosivulle
    #debuggausta varten.
    return jsonify(steerFlask = s)

#Tasasähkömoottorin ohjauskomentoja vastaanottava funktio.
#Toimii samalla periaatteella kuin Steering-funktio.
@app.route('/_Throttle')
def Throttle():

    args2 = ['python', '/home/pi/flaskArduino/throttleArduino.py']
    proc2 = subprocess.Popen(args2, stdin=subprocess.PIPE)
```

(jatkuu)

```
        t = request.args.get('t')
        proc2.communicate(t)

        print(ser.readline())

        return jsonify(thrtFlask = t)

#Käynnistetään sovellus toimimaan Raspberryn IP-osoitteessa.
if __name__ == '__main__':
    app.run(
        host="0.0.0.0",
        port=int("80"),
        debug=False
    )
```

Liite 9. Arduino-version ohjauskomentojen välityksen lähdekoodi

steeringArduino.py

```
# -*- coding: utf-8 -*-
import sys, serial

#Alustetaan USB-yhteys.
#serial.Serial(USB-portti, baud)
ser = serial.Serial('/dev/ttyACM0', 9600)

input = sys.stdin.readline()

if(input == '-4'): #vasemmalle 4
     ser.write('a')
elif(input == '-3'): #vasemmalle 3
     ser.write('b')
elif(input == '-2'): #vasemmalle 2
     ser.write('c')
elif(input == '-1'): #vasemmalle 1
     ser.write('d')
elif(input == '0'): #keskellä
     ser.write('e')
elif(input == '1'): #oikealle 1
     ser.write('f')
elif(input == '2'): #oikealle 2
     ser.write('g')
elif(input == '3'): #oikealle 3
     ser.write('h')
elif(input == '4'): #oikealle 4
     ser.write('i')
```

(jatkuu)

throttleArduino.py

```
# -*- coding: utf-8 -*-
import sys, serial

#Alustetaan USB-yhteys.
#serial.Serial(USB-portti, baud)
ser = serial.Serial('/dev/ttyACM0', 9600)

input = sys.stdin.readline()

if(input == '-4'): #vasemmalle 4
     ser.write('k')
elif(input == '-3'): #vasemmalle 3
     ser.write('l')
elif(input == '-2'): #vasemmalle 2
     ser.write('m')
elif(input == '-1'): #vasemmalle 1
     ser.write('n')
elif(input == '0'): #keskellä
     ser.write('o')
elif(input == '1'): #oikealle 1
     ser.write('p')
elif(input == '2'): #oikealle 2
     ser.write('q')
elif(input == '3'): #oikealle 3
     ser.write('r')
elif(input == '4'): #oikealle 4
     ser.write('s')
```

Liite 10. Arduino Mega 2560 -kehitysalustaan ladattu koodi

flaskArduinoPWM.ino

```
int steer = 4;
int throttle = 13;

void setup()
{
  TCCR0B = TCCR0B & 0b11111000 | 0x05;
  /*
    http://playground.arduino.cc/Main/TimerPWMCheatsheet
    ---------------------------------------------
    timer 0 (controls pin 13, 4);
    ---------------------------------------------
    Setting    Divisor   Frequency
    0x01       1         62500
    0x02            8         7812.5
    0x03            64        976.5625
    0x04       256       244.140625
    0x05       1024           61.03515625

    TCCR0B = TCCR0B & 0b11111000 | <setting>;
  */
  Serial.begin(9600); //määritetään baudi nopeus
  analogWrite(steer, 23);
  analogWrite(throttle, 23);
}

void loop()
{
  if (Serial.available() > 0) {
    //luetaan tavu sarjaväylästä
    char input = Serial.read();

    //ohjaus
    if(input == 'a'){ //vasemalle 4
      analogWrite(steer, 16); //1088
       Serial.println("left4");
    }
    else if(input == 'b'){ //vasemmalle 3
      analogWrite(steer, 18); //1216
      Serial.println("left3");
    }

    else if(input == 'c'){ //vasemalle 2
      analogWrite(steer, 20); //1344
      Serial.println("left2");
    }

    else if(input == 'd'){ //vasemalle 1
      analogWrite(steer, 21); //1408
      Serial.println("left1");
    }
    else if(input == 'e'){ //keskellä
      analogWrite(steer, 23); //1536
      Serial.println("middle");
    }
```

(jatkuu)

```
else if(input == 'f'){ //oikealle 1
  analogWrite(steer, 24); //1600
  Serial.println("right1");
}

else if(input == 'g'){ //oikealle 2
  analogWrite(steer, 25); //1664
  Serial.println("right2");
}

else if(input == 'h'){ //oikealle 3
  analogWrite(steer, 27); //1792
  Serial.println("right3");
}

else if(input == 'i'){ //rightrightrightoikealle 4
  analogWrite(steer, 29); //1920
  Serial.println("right4");
}

//kaasu
else if(input == 'k'){ //taakse 4
  analogWrite(throttle, 16);
  Serial.println("back4");
}

else if(input == 'l'){ //taakse 3
  analogWrite(throttle, 18);
  Serial.println("back3");
}

else if(input == 'm'){ //taakse 2
  analogWrite(throttle, 20);
  Serial.println("back2");
}

else if(input == 'n'){ //taakse 1
  analogWrite(throttle, 21);
  Serial.println("back1");
}

else if(input == 'o'){ //seis
  analogWrite(throttle, 23);
  Serial.println("stop");
}

else if(input == 'p'){ //eteen 1
  analogWrite(throttle, 24);
  Serial.println("forward1");
}

else if(input == 'q'){ //eteen 2
  analogWrite(throttle, 25);
  Serial.println("forward2");
}

else if(input == 'r'){ //eteen 3
  analogWrite(throttle, 27);
  Serial.println("forward3");
}
```

```
    else if(input == 's'){ //eteen 4
      analogWrite(throttle, 29);
      Serial.println("forward4");
    }

    else{
      analogWrite(steer, 23);
      analogWrite(throttle, 23);
      Serial.println("Error!");
    }
  }
}
```

Liite 11. Mittauspöytäkirja, Arduino Mega 2560 -kehitysalustan toiminta

Mittauspöytäkirja 29.4.2014 Aripekka Vorne

Fluke PM 3384B 100 MHz Combiscope

Arduino PWM – Mega2560

TCCR0B = TCCR0B & 0b11111000 | 0x05;

Jaksonaika: 16,4 ms ~ 61 Hz

analogWrite(13,1) → 128 µs pulssi

$\frac{16400}{256} = 64 \Rightarrow 1 = 2$

analogWrite(13,2) → 192 µs pulssi

analogWrite(13,3) → 257 µs

$\frac{1550}{64} = 24$

analogWrite(13,24) → 1,60 ms

analogWrite(13,23) → 1,54 ms

analogWrite(13,22) → 1,48 ms

$\frac{1000}{64} - 1 = 14,6$

analogWrite(13,14) → 960 µs

analogWrite(13,15) → 1,02 ms

analogWrite(13,16) → 1,09 ms

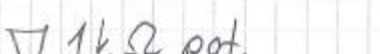
$\frac{2000}{64} - 1 = 30,25$

analogWrite(13,30) → 1,99 ms

analogWrite(13,29) → 1,93 ms

analogWrite(13,28) → 1,87 ms

MIN NEUTRAALI MAX
15 17 23 51 29



1kΩ pot.

o 5V → 3,3V