LAMK Lahden ammattikorkeakoulu
Lahti University of Applied Sciences

YMPÄRISTÖROBOTTI

LAHDEN
AMMATTIKORKEAKOULU
Tekniikan ala
Tietotekniikka
Tietokone-elektroniikka
Opinnäytetyö
Syksy 2016
Miki Siltakoski

Lahden ammattikorkeakoulu
Tietotekniikka

SILTAKOSKI, MIKI: Ympäristörobotti

Tietokone-elektroniikan opinnäytetyö, 54 sivua, 36 liitesivua

Syksy 2016

TIIVISTELMÄ

Työn tarkoituksena on esitellä ympäristörobottien toimintaa ja niiden kehitystä nykyhetkeen. Työssä käydään läpi nykyaikana olevia ympäristörobotteja ja niiden toimintaa ympäristöntutkimustyössä.

Tarkemmin perehdytään maassa liikkuviin ympäristörobotteihin ja niiden keräämiin tietoihin, joita käytetään ympäristön tutkimustyössä. Laitteistosta esitellään erilaisia ympäristörobottiratkaisuja, joihin sisältyvät virran säästö, kestävyys, toiminnan varmuus, itsenäinen toiminta sekä anturit ja tiedonsiirto. Opinnäytetyössä esitellään myös käytännön esimerkki langattomasti toimivasta ympäristörobotista, joka kerää ympäristöstä ja ilmanlaadusta tietoa ja välittää sitä käyttäjälleen.

Asiasanat: robotti, ympäristörobotti, tutkimus, piirilevysuunnittelu, H-silta, mikro-ohjain

Lahti University of Applied Sciences
Degree Programme in Information Technology

SILTAKOSKI, MIKI: Robot for environmental monitoring

Bachelor’s thesis in computer electronics, 54 pages, 36 appendices

Autumn 2016

ABSTRACT

This thesis deals with robots used in environmental monitoring. The work examines robots that were used in the past and are used at present for environmental monitoring.

The thesis focuses more specifically on ground moving robots and the information which they gather, to be used for environmental research. This work analyzes different kinds of hardware that offer robot solutions for environmental research. The features analyzed are power saving, durability, reliability, independent operation and used sensors and information transfer protocols. The thesis also presents a practical example of a wirelessly operating robot for environmental monitoring, which collects information about the terrain and air quality and conveys it to the user.

Keywords: robots, robot for environmental monitoring, research, PCB design, motor control, microcontroller

SISÄLLYS

1 JOHDANTO

Opinnäytetyön tarkoituksena on tutustua robotteihin yleisesti sekä niiden kehitykseen ja robottien toimintaan käytännössä teollisuudessa ja tutkimustyössä. Tarkoituksena on selvittää robottien käyttöä ympäristön tutkimustyössä sekä vaatimuksia ympäristöroboteille niiden käyttöympäristöissään. Opinnäytetyössä perehdytään lähinnä maastossa toimiviin robotteihin ja ympäristönmittaustekniikoihin ja yleisesti robottien tekniseen puoleen.

Tarkoituksena on myös toteuttaa käytännön esimerkki osittain itsenäisesti toimivasta ympäristörobotista, joka toimii mikrokontrollerilla, käyttäen antureita esteiden ja ilmanlaadun seuraamiseen sekä liikkumisen ohjaamiseen. Toteutettavan robotin on määrä toimia langattomasti, ja sitä on voitava ohjata etänä. Toteutuksen päätavoitteena on saada robotti toimimaan tunnin ajan langattomasti, liikkumaan paikasta toiseen omilla avuillaan, mittaamaan ilmanlaatua ja tallentamaan saadut mittaustiedot, nauhoittamaan videokuvaa tai ottamaan maastokuvia sekä sillä on oltava etäkäyttöön soveltuva käyttöliittymä.

2 ROBOTIIKKA YLEISESTI

Termi robotti on muodollisesti määritelty ISO 8373:2012 (International Standard of Organization) -standardissa seuraavasti: ohjelmoitava, monitoiminen ympäristöä manipuloiva laite tai kone, joka on suunniteltu siirtämään materiaaleja, osia tai erillisiä laitteita käyttäen ohjelmoituja liikkeitä suorittaakseen sille osoitettuja tehtäviä. Roboteille on olemassa muitakin muodollisia määrittelyjä, mutta kaikissa perusajatus on sama; robotti on laite, joka voidaan ohjelmoida suorittamaan itsenäisesti erilaisia tehtäviä. (Saha 2008, 5.)

Robotit on tyypillisesti luokiteltu teollisiin, ei-teollisiin sekä erikoistarkoituksiin soveltuviin alaluokkiin. Teollisuusrobotilla tarkoitetaan yleiskäyttöön sopivaa laitetta, joka pystyy suorittamaan yksinkertaisia tai vähäistä taitoa vaativia työtehtäviä, kuten hitsaamista, työstöä ja kokoamista. Erikoistarkoituksiin soveltuviksi roboteiksi kutsutaan robotteja, jotka suorittavat monimutkaisia tai tarkoitukseltaan ainutlaatuisia tehtäviä. Kärjistettynä esimerkkinä erikoistarkoituksiin soveltuvasta robotista voisi olla avaruudessa toimiva robotti, joka kykenee korjaamaan itsenäisesti viallisen osan satelliitista. Tyypillisesti erikoistarkoituksiin soveltuvia robotteja luokitellaan alaluokkiin seuraavasti: automaattisesti ohjautuvat ajoneuvot, ihmisen liikkeitä simuloivat robotit sekä rinnakkaisesti yhdessä toimivat robotit. (Saha 2008, 6 - 7.)

Teollisuudessa robotteja käytetään usein suorittamaan likaisia, paljon toistoa vaativia, vaarallisia tai ihmiskeholle liian tarkkoja tai vaikeita tehtäviä. Ihmisiä korvataan usein roboteilla vasta jos siihen on nähtävissä järkeviä syitä kuten taloudelliset syyt, vaaralliset olosuhteet tai kun tarvitaan suurta tehokkuutta ja nopeutta. Lyhyissä sekä vähää toistoa vaativissa töissä robotti on yleensä tarpeeton, ihmisen ollessa joustavampi suorittamaan kyseinen tehtävä. (Saha 2008, 8 - 9.)

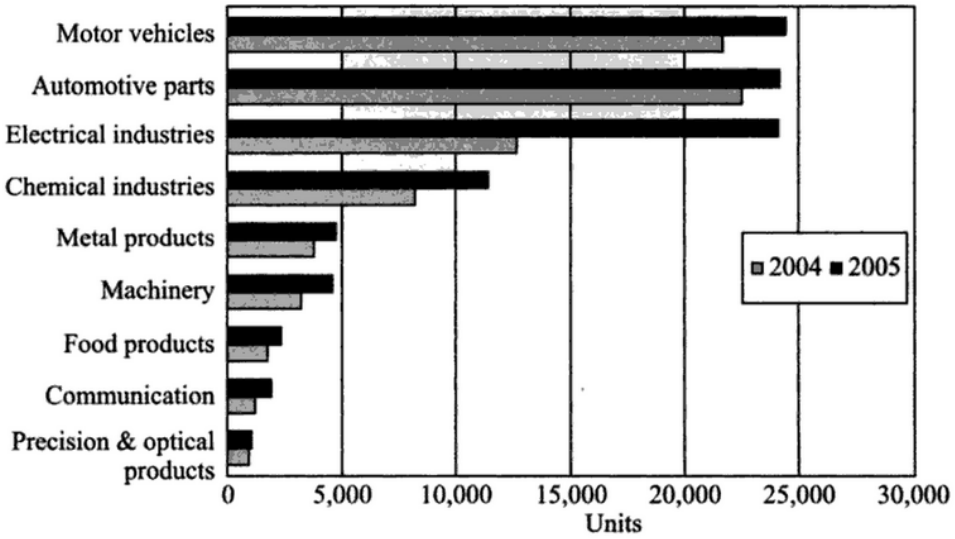


Fig. 1.9 Usage of industrial robots during 2004–05
[Source: http://www.ifr.org/statistics/keyData2005.htm]

KUVIO 1. Robottien käyttö eri aloilla vuosina 2004 ja 2005 (Saha 2008, 10)

Robotteja käytetään maailmassa eniten ajoneuvoteollisuudessa, kuitenkin tietotekniikka- ja elektroniikkateollisuuden kasvaessa kyseisillä aloilla on merkittävästi nouseva kysyntä myös roboteille (KUVIO 1). (Saha 2008, 10.)

3 ROBOTTIEN KEHITYS

> *Jos jokainen työkalu tekisi itse oman työnsä pelkällä käskyllä, jos sukkula osaisi kutoa ja plektra soittaa kädettä, ei työntekijöitä tarvittaisi (Aristoteles 384 - 322 eaa.)*

Vaikka nykypäivänä sana robotti herättääkin useilla ajatuksen tulevaisuudesta ja korkeasta teknologiasta, ajatus roboteista juurtaa juurensa kuitenkin kauas historiaan jopa neljän vuosituhannen päähän. Ihmiskunnan kehittäessä ajansaatossa selityksiä luonnon ilmiöille käyttäen hyväkseen matematiikkaa, fysiikkaa ja geometriaa, joita voitiin soveltaa käytännön työtehtävien ja ongelmien ratkaisussa, kasvoi käsitys myös roboteista. Tämä käsitys ei kuitenkaan ollut konkreettinen käsitys itsenäisesti toimivasta koneesta vaan lähinnä ajatus siitä, että työt voitaisiin mahdollisesti tehdä ilman ihmistä. Yhteisöjen kehittyessä ihmiset kehittivät parempia työkaluja ja teknisiä ratkaisuja käyttäen monimutkaisempia yhdistelmiä perinteisistä työkaluista hyödyntäen matematiikkaa ja fysiikkaa. Tätä voidaan pitää esihistoriallisena lähtökohtana ja alkuperänä robottien kehitykselle. (Nocks 2007, 3-5.)

Myöhemmin robotteja on kehitetty moniin erilaisiin käyttötarkoituksiin ja monille eri aloille kuten teollisuuden aloille hitsaukseen, maalaamiseen, kokoamiseen ja työstämiseen, lääketieteen aloille kuntoutukseen ja kirurgiaan, puolustusvoimille turvallisuus- ja puolustustehtäviin sekä tieteen aloille valtamerten ja avaruuden kartoittamiseen ja tutkimiseen. (Nocks 2007, 6.)

3.1 Robottien kehittyminen teollisuudessa

1950-luvulla George C. Devol pyrki kehittämään ensimmäisen teollisuudessa toimivan robotin, Unimaten. Hän perusti Unimation Robotics Companyn Yhdysvalloissa vuonna 1958 yhdessä Joseph F. Engelberger kanssa. Ensimmäinen Unimate-robotti oli käytössä 1960-luvulla General Motorsin ajoneuvotehtaalla, jossa se toimi osana valumetalli osien valmistusta pudottaen kuumia ajoneuvon osia

jäähdytysnesteeseen, josta ne saatiin linjastolla eteenpäin työntekijöille jälkityöstettäviksi. Kyseisen robotin ansiosta linjaston työntekijöiden ei tarvinnut enää käsitellä metallivalimosta tulevia kuumia osia. (Saha 2008, 2.)

Modernit robotit jatkoivat kehitystään 1960- ja 1970-luvuilla. Vuonna 1963 kehitettiin ensimmäinen raajavammautuneita auttava robottikäsivarsi, Rancho Arm. Robottikäsivarsien kehitystä jatkettiin puolivuosikymmentä ja vuonna 1969 niitä alettiin tuoda kuluttajille. Myöhemmin tietokonejärjestelmien kehittyessä robottikäsivarsista saatiin paljon tarkempia ja niitä voitiin käyttää teollisuudessa hoitamaan kokoamistehtäviä. Robottikäsivarsien menestys johti teollisuusrobottien kehityksen vilkastumiseen, ja 1980-luvulla useat robotiikkaa kehittävät yritykset saivat suuria investointeja ajoneuvoalan yrityksiltä. (RobotWorx 2015.)

3.2 Kehitys ympäristön tutkimustyössä

Teollisuusrobottien kehittyessä yhä monimutkaisemmiksi, kehittyneemmiksi ja tarkemmiksi laitteiksi, joilla saatettiin manipuloida ympäristöä monella eri tavalla, kuten liikkumaan paikasta toiseen, siirtämään esineitä, kerätä näytteitä erilaisilla antureilla, tallentamaan ja lähettämään tietoa ja toimimaan itsenäisesti pitkiä aikoja ilman huoltotoimenpiteitä, heräsi tiedemiehillä ajatus kehittää robotteja ympäristön kartoittamiseen ja tutkimiseen. Alkuperäinen suunnitelma ympäristöroboteille oli käyttää niitä tutkimaan alueita, joissa vallitsee vaikeat ilmasto- tai maasto-olosuhteet. Ensimmäisiä ympäristörobotteja käytettiin lähinnä meri- ja rannikkoalueilla, joissa saattoi vallita ihmiselle vaaralliset olosuhteet, joten oli järkevää käyttää etänä toimivia robotteja keräämään näiltä alueilta tietoja ja näytteitä. Alueilta saattoi myös olla hyvin vähän edeltävää tietoa, joten etärobottien käyttö oli taloudellisempaa kuin lähettää tutkimusryhmä ennen robottien esitiedon keräämistä. (Dunbabin & Marques 2012, 28.)

KUVA 1. Suomen merivoimien käytössä oleva REMUS100-ympäristörobotti (Wikipedia 2014)

Valtamerten kartoitukseen ja tutkimiseen kehitettiin useita erilaisia ympäristörobotteja kuten REMUS (KUVA 1) tai AutoSub, joiden yhtäjaksoinen toiminta-aika oli 10 tunnin molemmin puolin. Niissä oli integroituna erinäisiä laitteita, joihin kuului muun muassa paikallistamisjärjestelmä (GPS), jolla saatettiin asettaa robotille tehtävään kuuluva reitti tai ohjata sitä etänä kartan avulla liikkumaan paikasta toiseen. Akkujen kehityksen myötä toiminta-aika parani huomattavasti ja kylmissäkin olosuhteissa saatettiin suorittaa jopa vuorokauden mittaisia tehtäviä. (Dunbabin & Marques 2012, 28.)

Vaikka suurin osa ensimmäisistä ympäristöroboteista olikin tarkoitettu lähinnä valtamerten ja rannikkoseutujen tutkimiseen, oli olemassa myös ilma- ja maastoympäristöissä toimivia robotteja. Ilmassa toimivalla robotilla Aerosonde UAV saatettiin tutkia ilmastoa laajalla alalla. Vuonna 1998 kyseinen robotti lensi itsenäisesti Atlantin valtameren halki eli yli kolmen tuhannen kilometrin matkan. Maaympäristöön soveltuvilla roboteilla saatettiin tutkia kohteita epätasaisilla tai tasaisilla, mutta laajoilla alueilla. Epätasaisille alueille soveltuvista roboteista tunnetuin oli NASA:n

sponsoroima Dante, kahdeksalla jalalla kävelevä robotti, jolla pystyttiin tutkimaan tulivuoren aluetta. Tasaisella maastolla oli olemassa robotteja, jotka kulkivat pyörillä. Ne saattoivat ottaa näytteitä aavikolta etänä pitkienkin matkojen päästä. Kuitenkin nämä tutkimukset olivat pääasiassa vielä kokeiluja, mutta niistä saatiin tärkeää tietoa maa- ja ilmastorobottien toimintavarmuudesta ja tarkkuudesta niiden edelleen kehittämistä varten. (Dunbabin & Marques 2012, 29.)

4 NYKYAIKAISET SOVELLUKSET

Nykyään ympäristörobotteja käytetään laajalti vesistöjen, maa-alueiden sekä ilmaston tutkimustyössä. Roboteilla on tärkeä rooli myös ulkoavaruuden tutkimuksissa. Robottien kehitysvaiheiden myötä on pystytty ratkaisemaan useita haasteita koskien robottien toimintavarmuutta, huoltovapautta ja käyttötarkkuutta. Kuitenkin tutkimusmenetelmien ja -olosuhteiden kehittyessä roboteilta vaaditaan yhä enemmän. (Dunbabin & Marques 2012, 29.)

4.1 Energiatehokkuus

Nykyaikaiset ympäristörobotit käyttävät usein liikkumiseen ja lisälaitteiden ohjaukseen sähkö- ja polttomoottoreita. Kuitenkin toiminta-aika riippuu paljolti käytettävissä olevan polttoaineen, akkukapasiteetin ja aurinkopaneelien määrästä. Tästä syystä energiatehokkuus pitkäaikaisesti itsenään toimivilla ympäristöroboteilla on äärimmäisen tärkeää.

Valtamerillä ajelehtivat ympäristörobotit on saatu hyödyntämään aallokon liikettä, jolla saadaan luotua sivuttaista liikettä käyttämällä veden alla toimivaa siipeä. Vedenalaisen siiven lisäksi usein käytetään aurinkopaneeleja, joilla saadaan virtaa käyttölaitteille ja antureille sekä parannettua toiminta-aikaa. Veden päällä seilaavat ympäristörobotit voivat parhaimmillaan saavuttaa jopa yli neljän vuoden toiminta-ajan.

Ilmassa toimivia ympäristörobotteja (UAV) koskevat suurimmat haasteet energiatehokkuudessa, koska ne eivät voi pysähtyä säästääkseen energiaa tai ladatakseen akkuja. Usein ilmassa toimivat robotit hyödyntävät aurinkovoimaa, jonka avulla on saatu äärimmäisen kevyet pienoiskoneet lentämään Marsissa hyödyntäen pelkästään aurinkopaneeleja. Kuitenkin aurinkopaneeleja on mahdollista lisätä robottiin vain rajoitettu määrä ja usein vaaditaan lisäksi raskaita ja energiaa kuluttavia tutkimus- ja kuvauslaitteita, jotka heikentävät lennon pituutta. Aurinkovoiman hyödyntämistä on pyritty kehittämään muuttamalla

aerodynamiikan avulla lentoliikettä siten että auringosta saataisiin mahdollisimman suuri hyöty. (Dunbabin & Marques 2012, 29.)

4.2 Tietojen keräys

Ympäristörobotteja käytetään muun muassa kemikaali- ja kaasuvuotojen paikantamiseen. Robottien käyttö säästää aikaa ja resursseja. Sen vuoksi haara ympäristörobottien kehittäjistä on keskittynyt havaitsemaan ilmakehän saastekeskittymiä, vedenalaisia saastelähteitä tai hydrotermisiä purkaumia. Näiden saastelähteiden paikallistamiseen voidaan käyttää erilaisia menetelmiä. Voidaan toteuttaa saastuneiden alueiden kartoitusprosessi, jossa koko saastunut alue kartoitetaan ja näin ollen saadaan tietoa mahdollisista keskittymistä. Toinen vaihtoehto on reaaliaikainen paikannusprosessi, jossa saasteen lähdettä etsitään seuraamalla saastevuotoa niin kauan kunnes sen lähde löydetään. Robottien keräämää tietoa käytetään algoritmeissa, jotka antavat suuntaa mahdolliselle saastelähteelle. Voidaan luoda esimerkiksi kaksiulotteinen kaasun jakaumakartta, josta selviää kaasun leviäminen tuulen ja vallitsevien ilmasto-olosuhteiden vaikutuksesta.

Veden saastumista voidaan mitata esimerkiksi kaasuanturilla, joka mittaa vedestä haihtuvien metaanikaasukuplien kokoa. Saatu tieto voidaan kartoittaa ja käyttää sen havainnollistamiseen tutkittuja menetelmiä siitä, kuinka saaste leviää ja ilmenee erilaisissa vesistöissä. Ympäristörobottien on havaittu voivan suorittaa pienen ja erittäin suuren mittakaavan mittauksia. Kuitenkin robottien ottamat näytteet eivät ole aivan yhtä tarkkoja kuin miehitetyn mittausoperaation näytteet, mikä johtuu robottien rajoitetusta autonomiasta. (Dunbabin & Marques 2012, 30-31.)

5 YMPÄRISTÖROBOTIN SUUNNITTELU JA TOTEUTUS

Opinnäytetyön ympäristörobotin suunnittelu ja toteutus vaiheessa robotille esitetään seuraavat päätavoitteet:

1. robotin liikkumisen langaton toteuttaminen, johon kuuluu eteen ja taakse liikkuminen sekä kääntyminen
2. automaattisen liikkumisen toteuttaminen
3. lämpötilan, ilman kosteuden sekä häkäarvojen kerääminen ja näiden tietojen tallettaminen muistiin
4. kääntyvän kameran kuvan välitys käyttäjälle langattomasti
5. käyttäjän käyttöliittymän toteuttaminen ja ohjaus langattomasti tietokoneella ja radio-ohjaimella.

Robotin liikkuminen toteutetaan sähkömoottoreilla, jotka pyörittävät robotin sivuilla olevia telahihnoja. Tavoitteena on saada robotti liikkumaan eteen, taakse ja kääntymään mahdollisimman tarkasti. Robotin liikkumisnopeutta täytyy myös tavoitteen mukaan voida muuttaa. Liikkumista tulee ohjaamaan ohjelmoitava Mbed-mikro-ohjainkortti ja moottoreiden ohjaus toteutetaan H-sillalla, joka suunnitellaan erilliselle moottorinohjauskortille. Robotista tehdään langaton, joten sen virtalähteeksi asennetaan ladattava akku. Akulta otetaan virta kaikkiin robotin komponentteihin, mistä syystä robotille suunnitellaan teholähde, josta saadaan käyttöjännitteet kaikille laitteille.

Automaattinen liikkuminen toteutetaan kahdella robotin etuosaan asennettavalla estetunnistimella, jotka tunnistavat mahdollisen esteen, jolloin robotin on tarkoitus kääntyä ja väistää mainittu este tai pysähtyä, mikäli edessä on suurempi este kuten rakennuksen seinä. Automaattinen liikkuminen toteutetaan ohjelmoitavalla Mbed-mikro-ohjainkortilla.

Lämpötilan, ilman kosteuden sekä häkäarvojen kerääminen toteutetaan kahdella erillisellä anturilla. Kaasuanturina tulee toimimaan Figaro 822 ja ilman kosteus- ja lämpötila-anturina Honeywellin HIH6030. Anturit

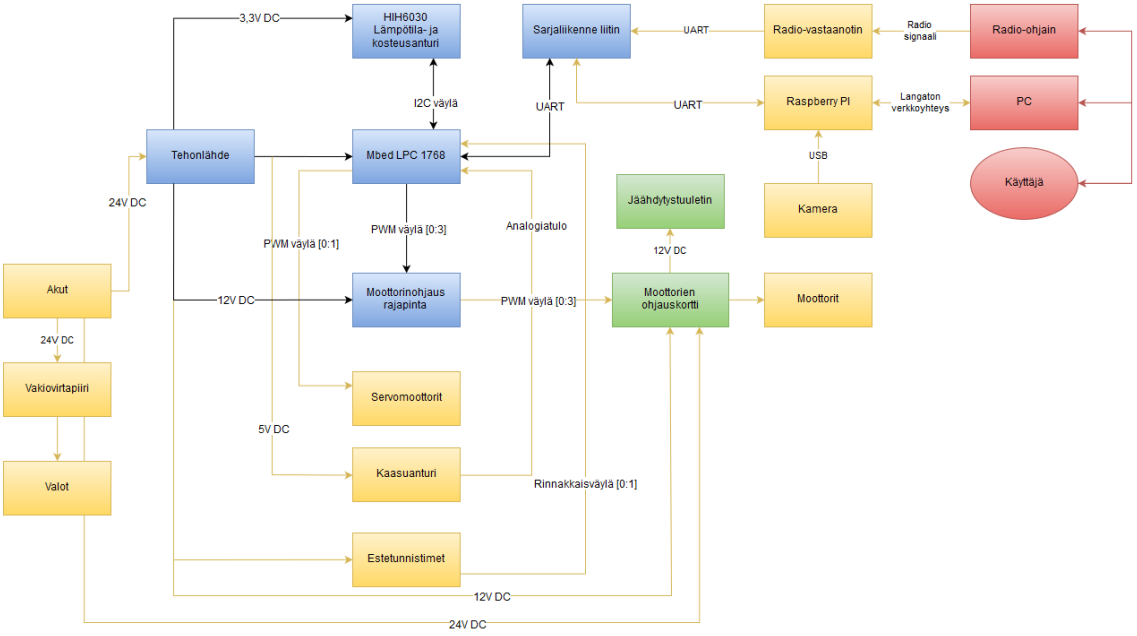
yhdistetään robotin Mbed-mikro-ohjaimeen, joka välittää niiden antamat tiedot eteenpäin langattomasti käyttäjälle. Tiedot tallennetaan myös robotissa olevan tietokoneen muistiin myöhempää tarkastelua varten.

Robottiin asennettavaksi kameraksi tulee tietokoneen verkkokamera, jota käännetään kahdella servomoottorilla. Servomoottorit asennetaan siten, että kameran suuntaa voidaan muuttaa pysty- ja vaakasuunnassa. Kameran kuva välitetään robotin tietokoneen avulla käyttäjän tietokoneelle langattoman verkkoyhteyden välityksellä. Kameran kuva pitää olla nähtävissä myös hämärällä, joten robottiin asennetaan valot.

Käyttäjälle on tavoitteena toteuttaa kaksi erillistä käyttöliittymää, joista toinen on kauko-ohjain ja toinen kotitietokone eli PC. Tärkein tavoite on saada robottiin yhteys molemmilla käyttöliittymillä langattomasti. Käyttöliittymien tulee olla myös yksinkertaiset ja selkeät sekä niistä tulee löytyä mahdollisuus ohjata robottia, muuttaa sen nopeutta ja suunnata siinä olevaa kameraa. Tietokonekäyttöliittymästä pitää pystyä myös lukemaan ja tallentamaan robotin keräämät anturitiedot. Tietokonekäyttöliittymä toteutetaan asentamalla robottiin Raspberry PI, Linux-tietokone, josta tehdään langattoman verkkoyhteyden tukiasema. Tukiaseman kautta robottiin saadaan langaton verkkoyhteys, jonka kautta käyttäjälle lähetetään robotin kameran kuvaa sekä antureiden tietoja. Toinen käyttöliittymä eli kauko-ohjain suunnitellaan ja toteutetaan itse. Siihen asennetaan myös Mbed mikro-ohjainkortti, joka lähettää radiolähettimen välityksellä robottiin käyttäjän valitsemat ohjauskomennot. Radio-ohjain tehdään akku- tai paristokäyttöiseksi sekä siihen asennetaan tarvittavat painikkeet robotin ohjaamista varten.

Robotin koostuessa useista eri kokonaisuuksista tehdään sille suunniteltaessa lohkokaavio (KUVIO 2), josta ilmenee eri komponenttikokonaisuuksien yhteistoiminta ja tietoliikenne. Lohkokaaviossa on keltaisilla laatikoilla merkitty erillisiä laitteita, siniset laatikot ovat mikro-ohjainkortin komponentteja, vihreät moottorien ohjauskortin komponentteja ja punaiset ovat käyttöliittymälaitteita. Nuolet

kuvaavat tietoliikennettä ja virrankulkua eri komponenttien välillä. Keltaiset nuolet kuvaavat johtimia, mustat piirilevyvetoja ja punaiset käyttäjän syötteitä.



KUVIO 2. Ympäristörobotin laitteiston lohkokaavio

5.1 Robotin runko

Ympäristörobotin runko on tässä opinnäytetyössä valmis aihio, jonka on valmistanut Lahden ammattikorkeakoulun TKE04-ryhmä. Runko on valmistettu teräslevystä. Siinä on paikat akuille, ohjauspiireille, kameralle, moottoreille ja teloille. Rungon keskiosaan kiinnitetään piirilevyt pinoten ne päällekkäin. Moottorit kiinnitetään robotin etuosaan, jossa ne pyörittävät telan hihnapyöriä. Telahihna kiristetään nostamalla se hihnapyörien alle. Robotin takaosaan asennetaan kytkimet virransyötölle akuilta, latauspistoke sekä kauko-ohjauksen valintakytkin. Robotin etuosassa toimivat kaasu-anturi, estetunnistimet sekä robotin kamera ja sen suuntaa liikuttavat servomoottorit.

Robotin kaksi sähkömoottoria ovat 24 V:n tasajännitemoottorit valmiilla välityksillä. Moottorit ovat hitsauslaitteen langansyöttöpumpusta, ja niillä on kohtalaisen hidas pyörimisnopeus, mutta korkea vääntömomentti, josta

on hyötyä sekä alhaisen virran kulutuksensa vuoksi että riittävä vääntömomentti raskaan robotin liikuttamista varten eikä siksi välityssuhdetta tarvitse muuttaa ylimääräisillä mekaanisilla ratkaisuilla. Moottoreiden maksimiteho on yhteensä noin 100 W. Robotin teloina toimivat ajoneuvon moottorin nokkapyörää pyörittävät jakohihnat, jotka ovat sekä erittäin kulutusta kestävät sekä venymättömät. Hihnat on ohjattu hihnan ohjauspyörillä kulkemaan robotin molemmilla sivuilla.

Elektroniikka sijoitetaan lähelle robotin moottoreita eli sen etuosaan. Tarvittavat piirilevyt pinotaan käyttämällä kierretappeja. Lämpösuunnittelun vuoksi alemmaksi levyksi sijoitetaan mikrokontrollerikortti, jonka päälle sijoitetaan mottoreiden ohjauskortti, joka jäähdytetään aktiivisesti tietokoneen 3,6 W:n laitetuulettimella, jonka halkaisija on 80 mm. Piirikorttien virrat tulevat robotin takaosassa olevilta akuilta johtimilla korttien peräosassa oleviin liittimiin. Korttien yhteistoiminta toteutetaan johtimilla ja liittimillä. Radiovastaanotin sijoitetaan mahdollisimman ylös robotin keskiosaan, koska näin saadaan mahdolliset ylimääräiset häiriöt moottoreista ja virtajohtimista vältettyä. Radiovastaanottimen signaalijohtimet kierretään signaalimaidensa kanssa häiriöriskin pienentämiseksi.

Robotin akuiksi valitaan kaksi 12 V:n suljettua lyijyakkua, joiden kapasiteetti on 17 Ah. Lyijyakut sopivat tähän käyttötarkoitukseen mainiosti, koska ne ovat sekä edullisia että pitkäkestoisia. Akut kytketään sarjaan, jotta saadaan moottoreiden käyttämä 24 V:n tasajännite. Päävirtajohdin suojataan 15 A:n sulakkeella. Akut ladataan sarjassa, jotta niiden kennojen latausvirta saadaan pidettyä samana, näin vältytään rasittamasta kennoja tai akkuja epätasaisesti, mikä parantaa akkujen käyttöikää. Koska akkuja käytetään pitkiä aikoja, niiden latausjännitteeksi on sopivaa valita korkeampi jännite, jolloin vältytään lyijyn sulfidinoitumiselta, mikä voi johtaa akkujen kiteytymiseen. Korkeampi latausjännite lyhentää myös akkujen latausaikaa sekä parantaa akkujen varauksen kestoa. Latausjännite valitaan yleensä väliltä 2,40 - 2,45 V

kennoa kohden, eli tässä tapauksessa 14,4 V akkua kohden eli yhteensä 28,8 V.

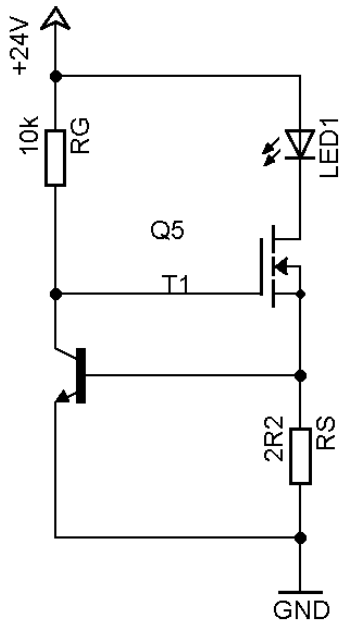
Robottiin asennetaan neljät LED-valaisimet eteen ja taakse. LED-valot kytketään sarjaan tehohäviön pienentämiseksi. Edessä olevat valaisimet ovat 350 mA:n vihreät ja takana 15 mA:n punaiset LED-valaisimet. Edessä olevien tehokkaiden LED-valaisimien virransyöttö toteutetaan vakiovirtapiirillä, jonka virransyöttö ei muutu riippumatta akkuihin varatusta jännitteestä. Vakiovirtapiiri toteutetaan transistori ratkaisulla (KUVIO 3), jossa korkeasta käyttöjännitteestä (24 V) johtuva tehohäviö ohjataan kanavatransistoriin, joka jäähdytetään passiivisesti jäähdytyselementillä.

$$I_f = \frac{V_{BE}}{R_S} = \frac{0{,}7\text{V}}{2{,}2\Omega} \approx 0{,}318\text{A} \qquad \text{(KAAVA 1.)}$$

Maahan vedettävän vastuksen (R_S) arvoksi valitaan 2R2, jolloin LED-valojen läpi kulkevaksi virraksi saadaan laskettua (KAAVA 1) noin 320mA. Kanavatransistorissa oleva tehohäviö saadaan laskemalla (KAAVA 2):

$$P_{Qmax} = \left(V_{DD} - V_f - V_{BE}\right) \cdot I_f = (28V - 13{,}7V - 0{,}7V) \cdot 0{,}318A = 4{,}3W$$

(KAAVA 2.)



KUVIO 3. Vakiovirtapiirin kytkentä

Takana olevien LED-valojen läpi kulkeva virta rajoitetaan etuvastuksella. Käyttöjännite takana oleville LED-valoille otetaan jännitereguloidusta 12 V:n tasajännitteestä, jolloin akkujen muuttuva varaus ei vaikuta LED-valojen läpi kulkemaan virtaan ja niiden kirkkauteen. Etuvastus mitoitetaan laskemalla (KAAVA 3):

$$R = \frac{V_{DD} - V_f}{I_f} = \frac{12V - 6{,}4V}{0{,}015A} \approx 373\Omega \qquad \text{(KAAVA 3.)}$$

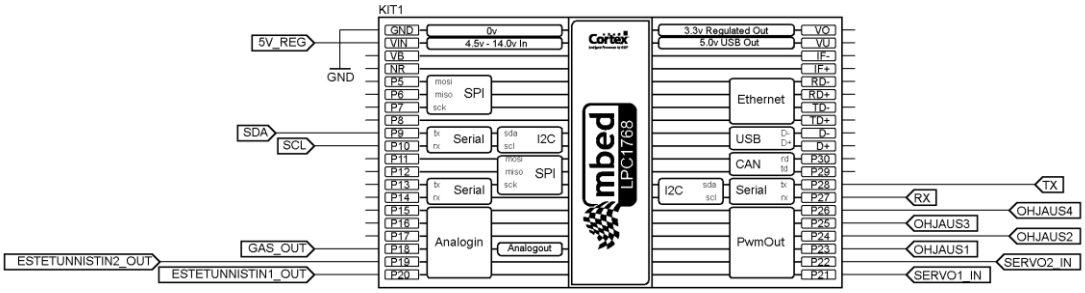
Valitaan 390Ω:n etuvastus jolloin läpikulkeva virta on noin 14mA.

5.2 Mikro-ohjainkortti

Opinnäytetyön käytännön puolessa käytetään ohjelmoitavaa mikro-ohjainta ympäristörobotin ohjaamiseen ja antureiden tietojen keräämiseen. Mikro-ohjain ohjaa robotin keskeisimpiä toimintoja, eli sen liikkumista, kameran suuntaamista, antureiden tiedon keräämistä ja vastaanottaa käskyjä radio-vastaanottimen ja Raspberry PI:n kautta sekä lähettää antureiden keräämän tiedon Raspberry PI:lle tallennettavaksi. Mikro-ohjainkorttiin kuuluvat mikro-ohjain, tehonlähde, kosteus- ja lämpötila-anturi, moottoreidenohjauksen rajapinta sekä liitännät antureille, tiedonsiirtoväylälle sekä lisälaitteille. Mikro-ohjainkortissa oleva mikro-ohjain ohjelmoidaan C++-ohjelmointikielellä. Piirilevy suunnitellaan CadSoft Eagle 5.7.0 –ohjelmalla ja jyrsitään kuparilevylle HPGL-piirilevyjyrsimellä sekä kalustetaan käsin.

Mikro-ohjaimeksi valitaan valmis Mbed LPC 1768 –kortti jossa on 32-bittinen ARM Cortex-M3 -mikro-ohjain, jonka kellotaajuus on 96MHz, 512kt flash-pohjaista ohjelmamuistia, 32kt RAM muistia, useita I/O-liitäntöjä sekä sisäänrakennettu ohjelmoija. Ohjelmointi suoritetaan käyttämällä Mbedin selainpohjaista kääntäjää. Mbedistä löytyy I/O portteja monenlaisiin käyttötarkoituksiin. Mbed toimii robotissa tietokoneena, joka voidaan ohjelmoida toteuttamaan robotille asetettujen toimintojen ylläpitäminen ja tehtävien suoritus. Se toimii 5V:n käyttöjännitteellä, ja siihen kytketään

lämpötila- ja kosteusanturi, kaasuanturi, estetunnistimet, servomoottorit, sarjaliikenneväylä sekä moottorienohjausrajapinta (KUVIO 4).



KUVIO 4. Mbed mikro-ohjaimen kytkentä

Käyttöjännitteet mikro-ohjainkortille otetaan robotin kahdesta sarjaan kytketystä 12 V:n suljetusta lyijyakusta. Tulojännite mikro-ohjaimelle on siis 24 – 29 V:n tasajännite riippuen akkujen varauksesta.

TAULUKKO 1. Komponenttien käyttöjännitteet ja virrankulutus

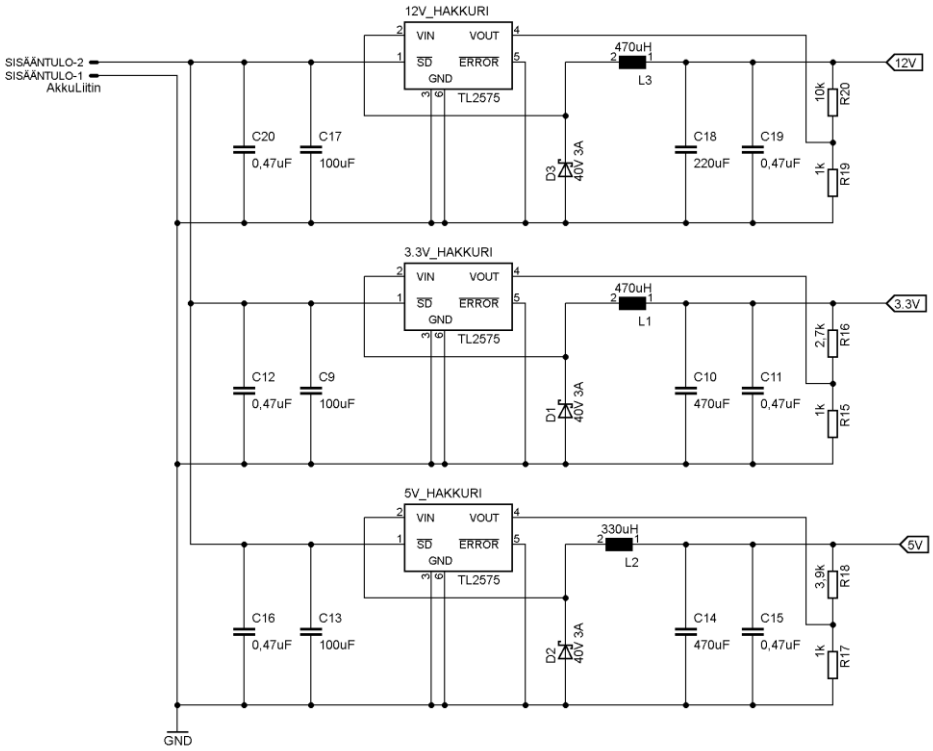
Komponentti	Käyttöjännite (DC)	Maksimi käyttövirta (mA)
Mikro-ohjain (Mbed)	5V	50
Lämpötila- ja kosteusanturi	3,3V	<1
Kaasuanturi	5V ja 3,3V	132
Estetunnistimet	12V	70
Servomoottorit	5V	380
Jäähdytystuuletin	12V	300
Radio-vastaanotin	3,3V	8
Moottoreidenohjauspiirit	12V	200
Moottoreiden ohjausrajapinta	12V	<1

Tarvittavat tehonlähteet mikro-ohjainkortin eri komponenteille ja lisälaitteille ovat siis 3,3 V:n, 5 V:n ja 12 V:n tasajännitteet (TAULUKKO 1). Pääsuoto on toteutettava hakkuriteholähteillä, koska tulojännitteen ollessa yli 24 V pelkkien jänniteregulaattoreiden käyttäminen aiheuttaisi ylisuuren tehohäviön ja johtaisi niiden ylikuumenemiseen. Maksimiantokuorman on oltava vähintään 600 mA. Hakkuripiireiksi (KUVIO 5) valitaan säädettävät TL2575-piirit, joiden maksimiantokuorma on 1 A. Hakkureiden antama

jännite tasataan jänniteregulaattoreilla. Hakkurit suunnitellaan datalehden ohjeiden mukaisesti ja passiivikomponentit valitaan datalehden taulukoiden arvoista. Hakkureiden perään kytkettävien jänniteregulaattorien (KUVIO 6) tulojännite tulee olla aina lähtöjännitettä korkeampi, mistä syystä hakkuripiirien antojännitteet jätetään noin voltin korkeammiksi kuin tarvittavat käyttöjännitteet. Hakkuripiirien antojännitteen säätö tehdään jännitteen jaolla ja halutut vastusarvot saadaan laskemalla (KAAVA 4):

$$R_2 = R_1\left(\frac{V_{OUT}}{V_{REF}} - 1\right), jossa\ V_{REF} = 1{,}23V \qquad \text{(KAAVA 4.)}$$

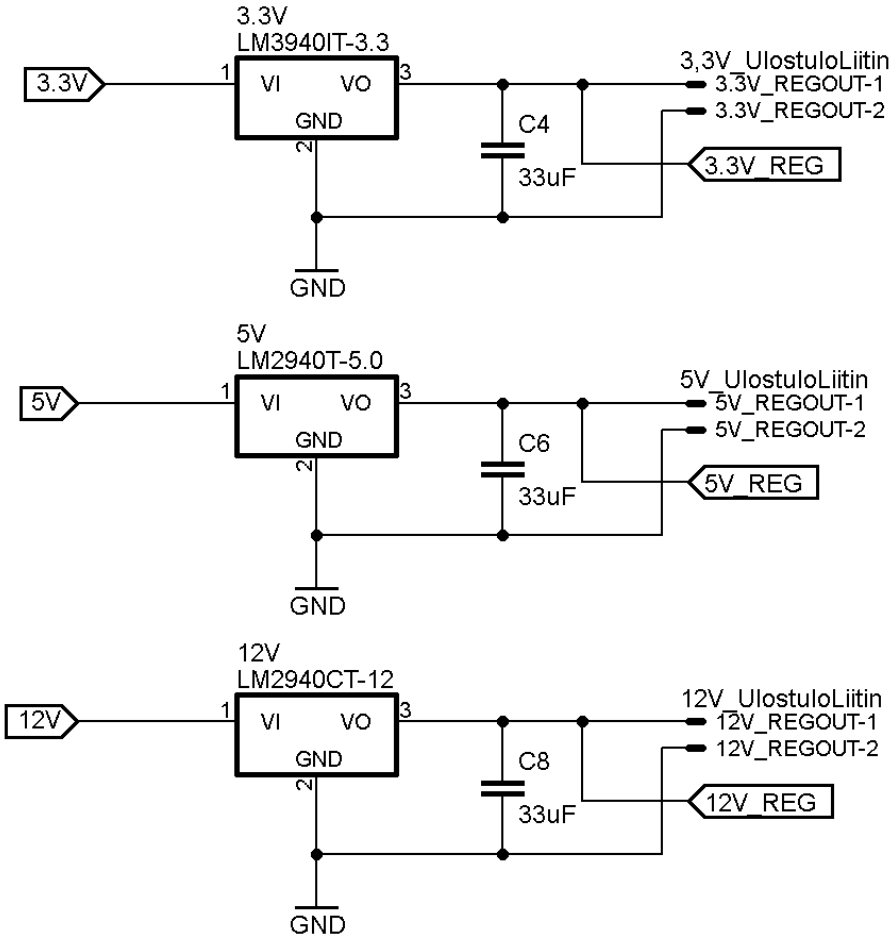
Kaavasta lasketuiden arvojen mukaan vastusarvoiksi valitaan 2,7 kΩ:n, 3,9 kΩ:n ja 10 kΩ:n vastukset, joilla hakkuripiirit antavat lähtöjännitteiksi 4,5 V, 6 V ja 13,5 V.



KUVIO 5. Hakkuripiirien kytkentä

Jänniteregulaattoreiden tulokondensaattori jätetään kokonaan pois, koska regulaattorit kalustetaan heti hakkureiden lähtökondensaattoreiden

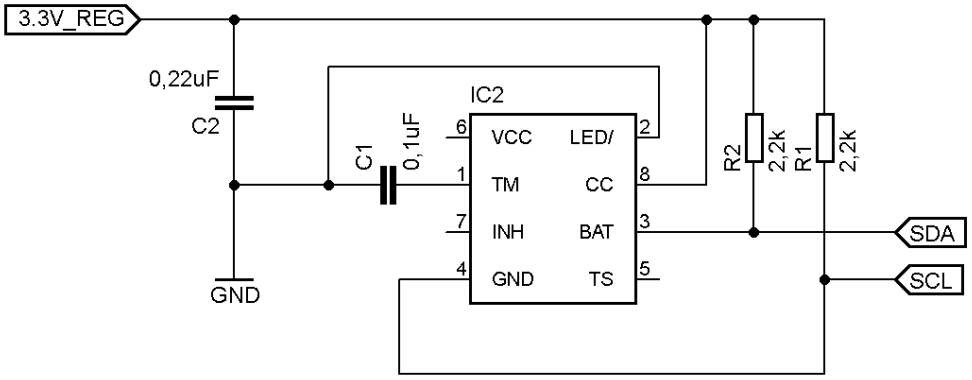
perään. Vaikka jänniteregulaattoreissa oleva tehohäviö jääkin pieneksi (alle 1 W), ne jäähdytetään silti varmuudeksi passiivisilla jäähdytyselementeillä.



KUVIO 6. Jänniteregulaattorien kytkentä

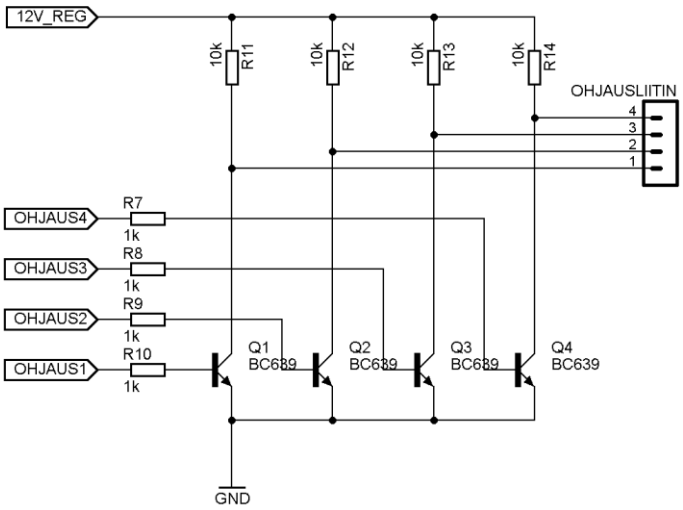
Kosteus- ja lämpötila-anturiksi valitaan Honeywellin HIH6000-sarjan anturi, joka mittaa sekä kosteutta että lämpötilaa. Anturi toimii 3,3 V:n tasajännitteellä, ja siinä on I^2C tietoliikenneliitäntä. Anturin tarkkuus on ±0,5 °C ja ±4,5 kosteusprosenttia sekä analogiadigitaalimuuntimen resoluutio 14 bittiä, mikä on varsin riittävä tähän käyttötarkoitukseen. I^2C väylän SDA ja SCL on vedettävä vastuksella 3,3 V:n käyttöjännitteeseen. Vastuksiksi valitaan I^2C väylällä melko standardit 2,2 kΩ:n vastukset (KUVIO 7). Anturilla on myös hälytysominaisuus, jonka avulla voitaisiin vilkuttaa esimerkiksi LED-valoa lämpötilan tai ilman kosteuden

muuttuessa, mutta tämä ominaisuus jätetään käyttämättä eikä hälytystoimintoa siksi kytketä. Anturin tiedonsiirto toimii 200 kHz:n taajuudella ja sen I^2C osoite on tavun mittainen (00100111b).



KUVIO 7. Lämpötila- ja kosteusanturin kytkentä

Moottoreiden ohjaukselle on toteutettava rajapinta, koska moottorinohjauspiirit ohjaavat yleiskäyttöisiä kanavatransistoreita mistä johtuen pelkkä Mbediltä lähtevän pulssinleveysmodulaation logiikkataso ei ole riittävä. Tästä syystä toteutetaan transistorikytkentä (KUVIO 9), jossa Mbedin moottorinohjauslähdöillä ohjataan etuvastuksien kautta npn-signaalitransistorien (BC639) kantaa. Transistorien kollektorit on kytketty vastuksella 12 V:n tasajännitteeseen ja emitterit maahan. Näin saadaan toteutettua 12 V:n logiikkataso, joka on myös samalla invertoiva, mutta sillä ei ole tässä tapauksessa piirin toiminnan kannalta merkitystä, kunhan se huomioidaan ohjelmoinnissa.



KUVIO 8. Moottoreiden ohjausrajapinta

Koska Mbed on mikrokontrolleri, sillä ei ole vaadittavaa suorituskykyä ja käyttöliittymää, joita vaadittaisiin kameran kuvan näyttämiseen. Mbed-kortilla toteutetaan tietoliikenne Raspberry PI:n sekä radio-ohjaimen Mbed-kortin kanssa. Tietoliikenne toteutetaan sarjaliikenteellä (UART). Koska radio-vastaanottimen tiedonsiirtonopeus on maksimissaan 20 kb/s, valitaan sarjaliikenteen baudinopeudeksi 19200. Käytettävä väylä, joko Raspberry PI tai radio-ohjain valitaan robotin taakse asennettavasta kytkimestä.

5.2.1 Piirilevy

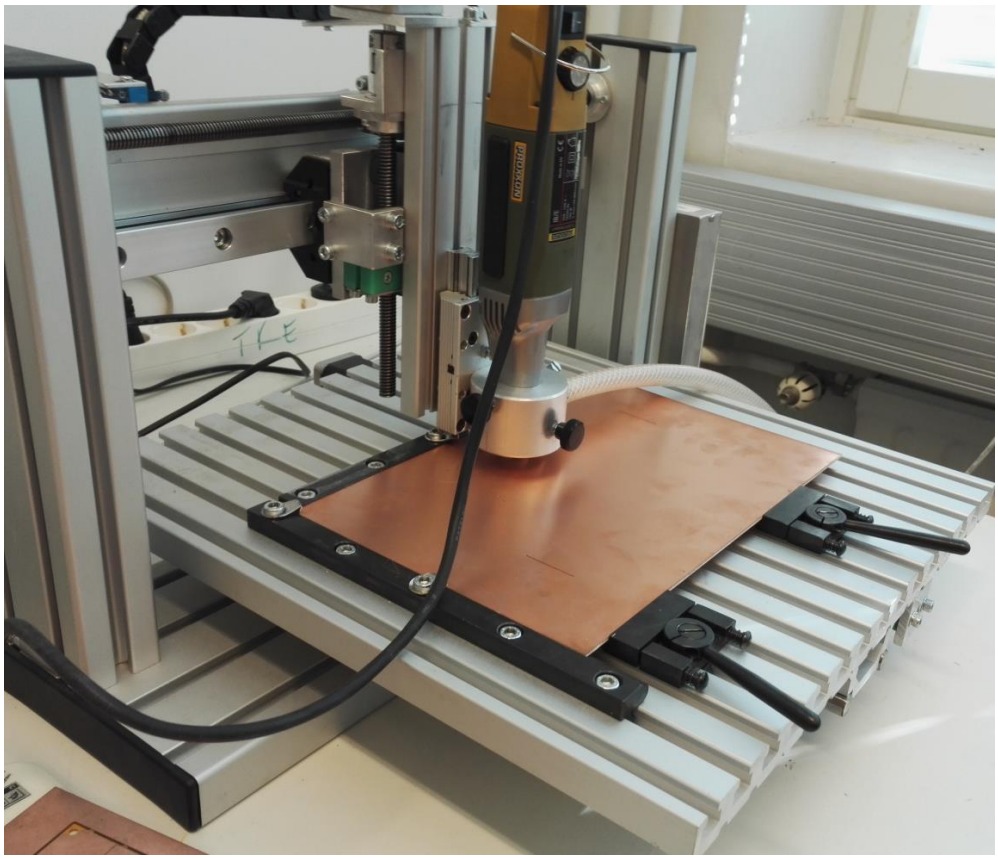
Piirilevyn suunnitteluun käytetään CadSoft Eagle 5.7.0 – ohjelmaa, jossa piirretään aluksi levyn piirikaavio (LIITE 1: luku 1.1), minkä jälkeen itse piirilevy piirretään. Piirilevylle asetetaan tiettyjä ehtoja sijoituksensa kannalta. Piirilevyn koko on oltava 155 x 100mm, siinä on oltava neljä reikää kiinnitystä varten 145 x 85mm:n etäisyydellä, perässä on oltava liitin akulta tuleville käyttöjännitteille ja kaikki muut liittimet on sijoitettava sen etupäähän helppoa laitteiden liittämistä varten.

Piirilevysuunnittelussa on otettava huomioon komponenttien looginen sijoittelu, jossa tulee huomioida tulevat levyn ja passiivijäähdyttimien

kiinnitysreiät. Rei’ille ja jäähdytyselementeille on varattava tilaa, eikä reikien alta tai kiinnikkeiden etäisyydessä saa vetää vetoja. Vetoja vedettäessä tulojännitteet tuodaan heti liitintä lähellä oleville hakkureille, minkä jälkeen jännitereguloidut käyttöjännitteet tuodaan komponenteille suunnassa peräpäästä etupäähän. Komponentit sijoitetaan suunnilleen levyn keskiosaan, sekä Mbedin ohjelmointiliitin suunnataan ulospäin levystä helppoa ohjelmointia ja testausta varten. Käyttöjännitevedot tuodaan mahdollisimman suoraan komponenteille, jotta vältytään turhalta häiriöriskiltä. Koska piirilevy suunnitellaan yksipuoliseksi, on signaalivedot ja komponentit suunniteltava tarkkaan, jotta vältytään ylimääräisten ilmajohtimien vetämiseltä. Piirilevyssä käytetään täyttökuparia, joka kytketään maahan; näin saadaan vähennettyä häiriöriskiä sekä ilmajohtimien vetämistä (LIITE 1: luku 1.2).

Piirilevy jyrsitään kuparilevylle HPGL-piirilevyjyrsimellä (KUVA 2) ja kalustetaan käsin. Piirilevyä jyrsittäessä on otettava huomioon reikien poraus ja vetojen paksuus ja valittava niitä vastaavat terät. Kuparin poistoa ei suoriteta maadoittamattomilta alueilta vaan kaikki alueet maadoitetaan ilmajohtimilla. Samalla varmistetaan maadoituksen kattavuus ja se, ettei pitkiä virtasilmukoita ole.

Levy kalustetaan käsin. Ensin tehdään maadoitus- ja vetotarkistukset yleismittarilla ja korjataan mahdolliset virheet. Tarkistuksen jälkeen tehdään ilmajohdinjuotokset. Tämän jälkeen komponentit juotetaan aloittaen pintaliitoskomponenteista. Lopuksi asennetaan kaikki läpijuotettavat komponentit, joista ensimmäisinä jänniteregulaattorit jäähdytyselementteineen. Kalustuksen ollessa valmis kaikki vedot ja juotosten eheys tarkistetaan uudelleen yleismittarilla tinasiltojen ja kylmäjuotosten varalta. Virheitä ei ilmene, ja valmis piirikortti voidaan siirtää testausvaiheeseen.



KUVA 2. Mikro-ohjainkortin piirilevyn valmistus

5.2.2 Ohjelmointi

Mbed-kortin tarkoitus on ohjata robotin keskeisimpiä ominaisuuksia, joihin kuuluvat robotin liikkumisen ohjaus, kameran suunnan ohjaus ja anturien tiedon kerääminen. Koska robottia tulee voida ohjata langattomasti, pitää ohjelmoida myös tietoliikenne radio-vastaanottimen ja Raspberry PI -kortin tiedonsiirroille. Ohjelmoitavia kokonaisuuksia ovat siis seuraavat:

1. moottoreiden ohjauksen toteutus, johon sisältyy automaattiohjaus
2. kameran suuntaa liikuttavien servomoottoreiden ohjauksen toteutus
3. antureilta tulevan tiedon kerääminen
4. tietoliikenne, joka sisältää robotin käskykannan ja anturien tiedon lähettämisen Raspberry PI-kortille.

Mbedille voidaan kirjoittaa olio-pohjaista koodia, joten jokaiselle edellä mainitulle kokonaisuudelle tehdään omat olionsa, jolloin pääohjelman

kirjoitus on mahdollisimman yksinkertainen ja testaus virheenkorjauksineen helpottuu.

Moottoreiden ohjausrajapinta on neljäpinninen, jotka on kytketty moottoreiden ohjauspiireihin siten, että pinneillä 23 ja 24 ohjataan vasenta moottoria ja pinneillä 25 ja 26 ohjataan oikeaa moottoria. Rajapinta Mbedin ja moottorinohjauspiirien välillä on invertoiva, joten se on otettava huomioon myös ohjelmassa. Ohjelmalla toteutetaan kaksi funktiota, omansa molemmille moottoreille. Funktioilla pitää pystyä muuttamaan moottorin pyörimissuuntaa ja nopeutta, jotta robotin kääntyminen ja peruuttaminen voidaan toteuttaa. Sähkömoottori saadaan pyörimään ajamalla sen napojen välille jännite. Moottorin pyörimissuunta saadaan muuttumaan, mikäli sen läpi ajettavien jännitteiden napaisuus vaihdetaan päinvastaiseksi. Sähkömoottorin jarrutus tapahtuu kytkemällä sen molemmat navat maihin, eli moottorin yli oleva jännite on silloin 0 V. Koska moottoreita ohjataan kanavatransistoreilla, voidaan moottorin pyörimisnopeutta säätää moottorin positiiviseen napaan ajettavan pulssin hyötyaikaa muuttamalla. Hyötyajan muutoksella voidaan siis korottaa tai laskea moottorin napojen välillä olevaa jännitettä, jolloin sen käyttämä teho pienenee ja pyörimisnopeus hidastuu. Mbed-kortissa on pulssinleveysmoduloidut lähdöt, mikä helpottaa ohjelman toteuttamista. Pulssilla on oltava myös taajuus, joka valitaan kohtalaisen korkeaksi eli 20kHz. Moottorinohjausfunktioista on tarkoitus tehdä mahdollisimman yksinkertaiset, joten toteutetaan funktiot, joilla on vain yksi argumentti, joka kuvaa moottorin nopeutta. Moottorin pyörimissuuntaa saadaan muutettua käyttämällä negatiivista nopeuden arvoa. Eteenpäin liikkumiseksi valitaan positiivinen nopeuden arvo ja taaksepäin liikkumista kuvaa negatiivinen arvo. Jotta funktio olisi mahdollisimman yksinkertainen, nopeus skaalataan asteikolle miinus sadasta sataan, jossa nopeus lähempänä nollaa tarkoittaa hitaampaa moottorin pyörimisnopeutta ja nolla jarrua. Koska rajapinta Mbedin ja moottoreiden ohjauspiirien välillä on invertoiva, Mbediltä lähtevän pulssin positiivinen pulssisuhde vastaakin todellisuudessa ohjauspiirillä negatiivista pulssisuhdetta, joten lähtevän

pulssin pulssisuhde täytyy siis funktiossa kääntää negatiiviseksi (KUVIO 9).

```
// value in speed (-100 - 100) (negative speed = reverse movement)
int motor_control::runMotorRight(int speed)
{

    int returnValue;
    float motorSpeed;

    if (speed >= -100 && speed <= 100) {

        returnValue=1;

        if (speed == 0) {
            _P2.write(1);
            _P3.write(1);
        } else if (speed > 0) {
            motorSpeed = 1 - (float(speed)/100);
            _P2.write(1);
            _P3.write(motorSpeed);
        } else if (speed < 0) {
            motorSpeed = 1 + (float(speed)/100);
            _P3.write(1);
            _P2.write(motorSpeed);
        }

    } else
        returnValue=0;

    return returnValue;
}
```

KUVIO 9. Oikean sähkömoottorin ohjausfunktio, jossa _P2 ja _P3 vastaavat Mbedin fyysisiä pinnejä 25 ja 26

Kameran suuntaa ohjaavat servomoottorit on kytketty Mbedin pinneihin 21 ja 22, jotka ovat pulssinleveysmodulaatiolähtöjä. Servomoottoria ohjataan ajamalla sen piirille tietyin väliajoin tietyn ajan pituinen pulssi. Pulssin pituus vastaa servolle määriteltyä suuntaa. Robotin servomoottoreiden pulssintiheys pitää olla 20 ms ja pulssinleveys tulee olla väliltä 0,75 ms – 2,25 ms, joka servomoottorinsuuntana vastaa 0 – 180 astetta. Ohjelman tulee pystyä muuttamaan tarkasti kameran suuntaa pysty- ja vaakatasossa kattaen servomoottorien koko kääntymisalan. Servomoottoreita halutaan ohjata mahdollisimman yksinkertaisesti, joten kummallekin servomoottorille tehdään omat ohjausfunktionsa. Koska pulssinleveys aikana on hiukan epämääräinen ja vaikeasti hallittava käsite, funktio muuttaa sille annetun argumentin eli suunnan asteina (0 – 180 astetta) servomoottorin ohjainpiirille lähtevään pulssinleveysmuotoon (KUVIO 10).

```
int servo_control::positionServo1(int degrees) // value in degrees (0 - 180)
{

    int returnValue;
    float servoPosition;

    if (degrees >= 0 && degrees <= 180) {

        returnValue=1;

        // convert servo degrees to pulsewidth (0.75-2.25ms)
        servoPosition = float(( 75 + ((0.8)*float(degrees)) ) / 100000);

        _SERVO_1.pulsewidth(servoPosition);

    } else
        returnValue=0;

    return returnValue;
}
```

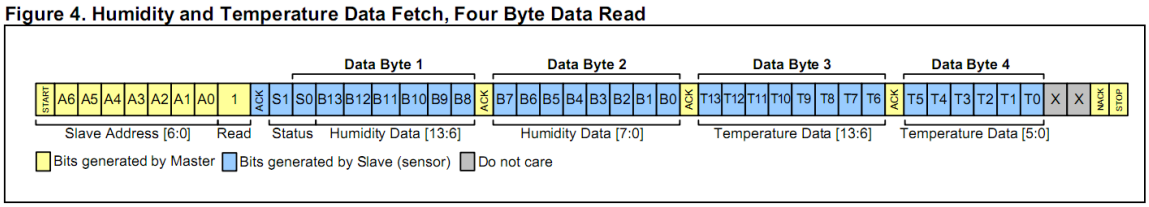
KUVIO 10. Servomoottorin ohjausfunktio, jossa _SERVO_1 vastaa Mbedin fyysistä pinniä 21

Kameran suunnan ohjaukselle toteutetaan myös automaattitoimintoa varten yksinkertainen funktio, jolla kamera kääntyy jatkuvasti hitaasti koko servomoottorin kääntymisalan puolelta toiselle. Tämä on helppo toteuttaa yksinkertaisten silmukoiden ja Mbedin ajastimien avulla (LIITE 2: luku 1.5.1).

```
int sensor::measurementRequest(void)
{

    int returnValue;

    this->start();

    if (this->write(0x27<<1)) { // write device address to I2C (27h)

        this->stop();

        this->start();

        if (this->write(0x27<<1)) {

            returnValue = this->read(0x27<<1, cmd, 4);
        }
    }

    return returnValue;
}
```

KUVIO 11. Lämpötila- ja kosteusanturin tiedon lukeminen I2C-väylältä

Robotilla on tietojen keräämistä varten kosteus- ja lämpötila-anturit, joiden tiedot kerätään käyttämällä niiden tukemia tiedonsiirto-ominaisuuksia. Antureiden keräämät tiedot lähetetään aina eteenpäin Raspberry PI:lle. Lämpötila- ja kosteusanturin lähettämä tieto luetaan I^{2}C-väylältä lähettämällä anturin osoite tieto väylälle (KUVIO 11). Osoitteensa saatuaan anturi lähettää 4 tavua tietoa, jossa ensimmäiset kaksi tavua sisältävät kosteustiedon ja kaksi viimeistä tavua lämpötilatiedon (KUVIO 12).



KUVIO 12. Sensorin tiedon lähetys I^{2}C-väylällä (Honeywell Techincal Note 2012, 2)

Saatu tieto tulee muuttaa bittimuodostaan luettaviksi lämpötila- ja kosteusarvoiksi. Anturin datalehdestä saadaan kaavat (KUVIO 13 ja KUVIO 14), joilla 14-bittinen tieto muunnetaan oikeiksi arvoiksi. 14-bittinen jono kuvaa kymmenjärjestelmän vastaavaa lukua, vastaava luku muutetaan ohjelmassa float-arvoksi. Koska ohjelmassa koko neljätavuinen luku luetaan char-taulukkoon, täytyy siitä erotella kosteuden ja lämpötilan arvot.

Equation 1: Humidity Conversion Function

$$\text{Humidity (\%RH)} = \frac{\text{Humidity Output Count}}{(2^{14} - 2)} \times 100\%$$

KUVA 13. Ilman kosteustiedon muutosfunktio (Honeywell Techincal Note 2012, 3)

Equation 2: Temperature Conversion Function

$$\text{Temperature (°C)} = \frac{\text{Temperature Output Count}}{(2^{14} - 2)} \times 165 - 40$$

KUVA 14. Lämpötilatiedon muutosfunktio (Honeywell Techincal Note 2012, 3)

Kosteusarvo luetaan siten, että ensin ensimmäisestä tavusta erotellaan pois statusbitit, minkä jälkeen se siirretään tavun verran vasemmalle. Toisen tavun tiedot saadaan sen perään or-operaatiolla. Tämän jälkeen voidaan luku sijoittaa edellä mainittuun kaavaan ja tulokseksi saadaan ilman kosteusprosentti. Lämpötila saadaan vastaavanlaisesti, mutta luku täytyy siirtää kaksi bittiä oikealle, koska datalehden mukaan kaksi viimeistä bittiä jätetään huomioimatta (KUVIO 15).

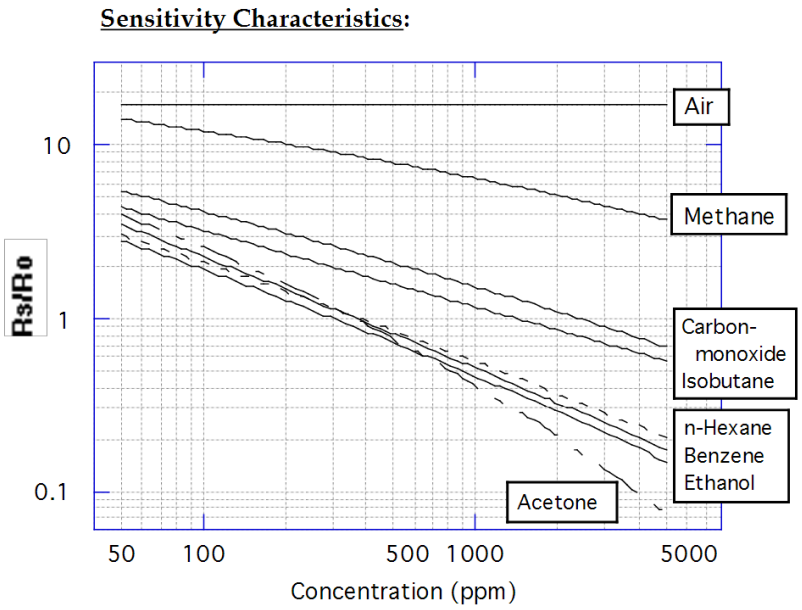
```
float sensor::readHumidity(void)
{
    float humiRH;
    measurementRequest();

    // convert humidity data bits to float value
    float humiRaw = ((cmd[0] & 0x3f) << 8) | (cmd[1] & 0xff);
    humiRH = (humiRaw/16382)*100;   // convert data value to humidity (percent)

    return humiRH;
}

float sensor::readTemperature(void)
{
    float temperature;
    measurementRequest();

    // convert temperature data bits to float value
    float tempRaw = ( (cmd[2] << 8) | cmd[3] ) >> 2 ;

    // convert data value to temperature (degrees)
    temperature = ((tempRaw/16382)*165) - 40;

    return temperature;
}
```

KUVIO 15. Lämpötila- ja kosteusarvojen muuntaminen, jossa anturilta saatu bittijono muunnetaan luettaviksi arvoiksi

Robotissa olevan kaasuanturin tieto luetaan analogiatulosta, jossa Mbedin analogiadigitaalimuunnin muuntaa tulevan jännitteen arvon digitaaliseksi arvoksi, jossa kaasuanturikytkennän tuottama 0-3,3 volttia vastaa digitaalilukua 0-1. Kaasuanturikytkennän vastusten ollessa tiedossa, voidaan kaasuanturin tämänhetkinen suhteellinen resistanssi laskea (KAAVA 4):

$$\frac{R_S}{R_O} = \frac{\left(\frac{V_C}{V_{R_L}} - 1\right) \cdot R_L}{R_O} = \frac{\left(\frac{3{,}3V}{analogiatulon\ arvo[V]} - 1\right) \cdot 10000\Omega}{4400\Omega}$$

(KAAVA 4.)



KUVIO 16. Suhteellisen resistanssin muutos riippuen ilman kaasupitoisuudesta (Figaro 2009, 1)

Suhteellinen resistanssi voidaan sijoittaa kaasuanturin datalehden kuvaajalle (KUVIO 16), jolloin voidaan ilman häkäpitoisuus laskea kuvaajasta saadulla kaavalla (KAAVA 5):

$$hiilimonoksidin\ konsentraatio\ ilmassa\ (ppm) = \frac{7472773}{\left(\frac{R_S}{R_O}^{2,28} \cdot 2882\right)}$$

(KAAVA 5.)

Tietoliikennettä varten robotille ohjelmoidaan käskykanta, jonka avulla tunnistetaan käyttäjän antamat komennot. Komennoiksi valitaan robotin ja kameran liikutus, nopeuden muuttaminen ja automaattiohjauksen käynnistys. Antureiden tiedon haku tehdään määrätyin väliajoin, jolloin isäntälaite pyytää robotilta käskyllä lähettämään antureiden tiedot. Käskykannasta on tehtävä selkeä, muttei liian yksinkertainen, jotta mahdollisilta häiriösignaaleilta etenkin radiovastaanottimen kanssa vältyttäisiin. Käskykannaksi valitaan merkkijonot, joiden ilmaisemiseen käytetään string-muuttujaa. Merkkijonoiksi valitaan käskyä kuvaavia

sanoja esimerkiksi ”forward” tai ”right”, joilla robottia liikutetaan eteenpäin tai käännytään oikealle (TAULUKKO 2).

TAULUKKO 2. Robotin käskykanta

Käsky	Toiminta	Käsky	Toiminta
forward	liikkuminen eteenpäin	decspeed	nopeuden hidastus
backward	peruuttaminen	incturn	kääntymisnopeuden lisäys
left	kääntyminen vasemmalle	decturn	kääntymisnopeuden hidastus
right	kääntyminen oikealle	rstspeed	nopeuksien palautus
stop	pysäytys	getdata	anturitiedon lähetyspyyntö
auto	automaattiohjaus		
horizontal	vaakaservon liikutus		
vertical	pystyservon liikutus		
incspeed	nopeuden lisäys		

Käsky luetaan sarjaliikenneväylältä char-taulukkoon (KUVIO 17), joka muunnetaan string-muuttujaksi, koska string-muuttujalle voidaan käyttää yksinkertaisia C++-kielen kirjastossa olevia funktioita, joilla vältytään ylimääräiseltä koodin kirjoittamiselta. Antureiden tiedot lähetetään sarjaliikenneväylälle myös merkkijono muodossa (KUVIO 18). Näin tiedon luku helpottuu vastaanottavassa päässä.

```
string serial_command::readSerial(void){

    char tempString [15];
    string returnValue;

    if (this->readable()) {

        this->gets(tempString,15); // read string from serial bus

        returnValue=string(tempString);

    } else
        returnValue="";

    return returnValue;
}
```

KUVIO 17. Sarjaliikenneväylän luku

```
int serial_command::writeSerial(string serialOutput)
{

    int returnValue;

    if (this->writeable()) {
        this->puts(serialOutput.c_str()); // write data to serial bus
        returnValue = 1;
    } else
        returnValue = 0;

    return returnValue;
}
```

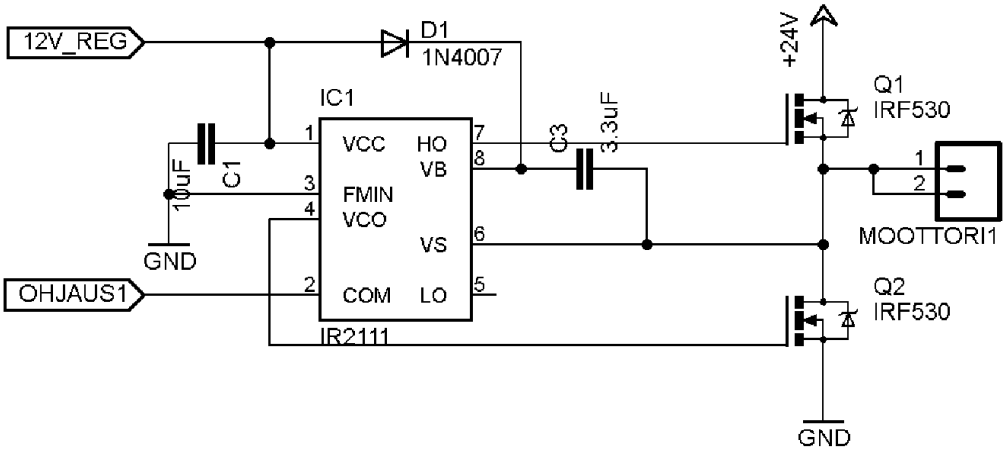
KUVIO 18. Sarjaliikenneväylän lähetys

Pääohjelma vastaa kaikkien funktioiden yhteispelistä. Alustusten jälkeen pääohjelma toimii silmukassa, jossa luetaan aina aluksi komento pyytämällä serial_command-luokan funktiota get_command, jonka jälkeen toimitaan saadun komennon mukaan ja pyydetään toimintaa edellyttävän luokan funktioita. Pääohjelmassa toiminnot vastaavat käytännössä edellä mainittua käskykantaa. Esimerkiksi ”forward”-komennolla toteutetaan eteenpäin liikkuminen ja niin edelleen (LIITE 2: luku 1.1).

5.3 Moottoreiden ohjauskortti

Moottoreidenohjauspiiri on H-silta, jossa puolisiltaohjaimilla, IR2111, ohjataan kahta kanavatransistoria. Kun kaksi puolisiltaa liitetään yhteen, saadaan siis H-silta, joka moottoriin kytkettynä mahdollistaa virran suunnanmuutoksen moottorin läpi. Molemmille moottoreille toteutetaan oma H-silta, jolloin mahdollistetaan myös robotin kääntymisen edellytykset (LIITE 1: luku 2.1). Korttia ohjataan jo edellä mainituilla neljäkanavaisella ohjausliittimellä, jossa jokainen pinni ohjaa aina yhtä puolisiltaohjainta. Puolisiltaohjain toimii siten, että se nostaa ylös aina toisen lähdöistään, jotka on kytketty N-tyypin kanavatransistorien (IRF530) hiloihin, riippuen sisään tulevan loogisen signaalin tasosta. Puolisillassa toisen kanavatransistorin nielu on kytketty suoraan akun positiiviseen napaan ja toisen lähde akun negatiiviseen napaan. Näitä kutsutaan sillan ala- ja yläpuoliksi (HIGH ja LOW). Moottorin toinen navoista on kytketty jäljelle jääviin nieluun ja lähteeseen. Toinen puolisilloista kytketään samoin

tavoin, mutta moottorin toiseen napaan. Moottorin ollessa kytkettynä kanavatransistoreihin tällä tavoin, voidaan siis loogisella ohjaussignaalilla ohjata minkä kanavatransistorien läpi kulkee akun kuormavirta, mikä mahdollistaa moottorin napaan ajettavaksi joko negatiivisen tai positiivisen akun navan (KUVIO 19). Käytännössä tämä tarkoittaa sitä, että kytkennällä voidaan tasavirtamoottorin läpikulkevan virran suuntaa, eli moottorin pyörimissuuntaa, muuttaa Mbedin digitaalisignaalilla. Hyödyllisemmäksi kytkennän tekee vielä se, että kun ohjaussignaalina käytetään pulssia, jonka hyötyaikaa voidaan muuttaa, saadaan moottorin läpi olevaa tehollisjännitettä muutettua, mikä mahdollistaa myös moottoreiden pyörimisnopeuden säätämisen.



KUVIO 19. Puolisiltakytkentä, jossa liitin (MOOTTORI1) vastaa moottorin toista napaa

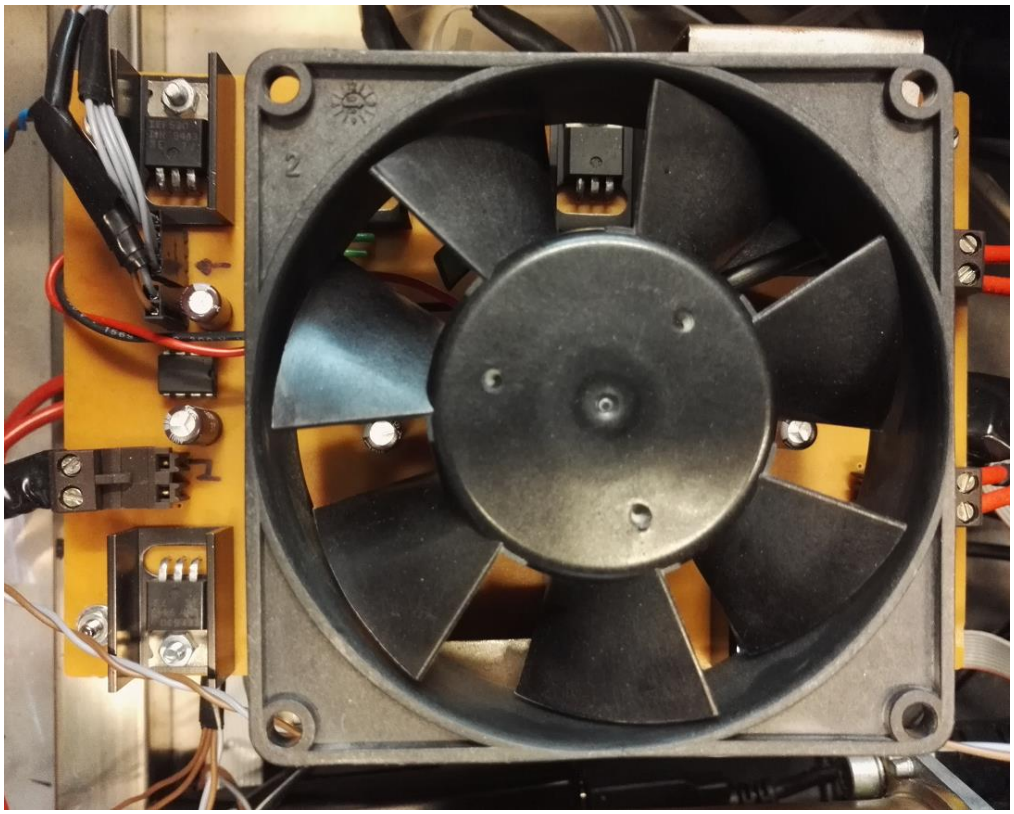
Piirilevyn suunnitteluun käytetään CadSoft Eagle 5.7.0 –ohjelmaa, jossa suunnitellaan levyn piirikaavio ja piirilevykuva. Piirilevylle asetetaan samat ehdot kuin aikaisemmin esitellylle mikro-ohjainkortille. Levyt pinotaan robotin rungon kiinnikkeisiin, mistä syystä piirilevyn koko on oltava sama 155 x 100mm, sekä siinä on oltava neljä reikää kiinnitystä varten 145 x 85mm:n etäisyydellä, perässä on oltava liittimet akulta tuleville käyttöjännitteille, tehonlähteestä tulevalle 12 V:n tasajännitteelle ja moottoreidenohjausliittimelle. Piirilevyn etupäässä tulee olla kaksi liitintä moottoreille.

Piirilevysuunnittelussa on otettava tärkeimpänä huomioon komponenttien sijoittelu. Kanavatransistoreille asennetaan jäähdytyselementit, joiden kiinnitysreiät tulee ottaa huomioon vetoja tehtäessä.

Reititystä tehtäessä tulee huomioida kanavatransistorien kuumeneminen sekä tarvittava virran saanti. Vetojen paksuus moottoreille kulkevissa vedoissa tulee olla vähintään 1,5 mm. Nämä vedot reititetään mahdollisimman kaukana signaalivedoista häiriöriskien välttämiseksi. Tarvittavat virta- ja bootstrap-kondensaattorit sijoitetaan mahdollisimman lähelle puolisiltaohjaimia. Koska piirilevy on oltava vain yksipuolinen, on kaikki vedot ja komponenttien sijoittelu suunniteltava tarkkaan, jotta vältytään ylimääräisten ilmajohtimien vetämiseltä. Piirilevyssä käytetään täyttökuparia, joka kytketään maahan, näin saadaan vähennettyä häiriöriskiä sekä ilmajohtimien vetämistä (LIITE 1: luku 2.2).

Levy jyrsitään HPGL-piirilevyjyrsimellä sekä kalustetaan käsin. Jyrsitylle levylle tehdään tavanomaiset maadoitus- ja vetotarkistukset yleismittarilla ja korjataan mahdolliset virheet. Tarkistuksien jälkeen tehdään ilmajohdinjuotokset. Tämän jälkeen kaikki komponentit juotetaan ja testataan vielä vetojen ja juotosten eheys. Suurempia virheitä ei ilmene, joten valmis piirikortti saadaan testausvaiheeseen.

Kuumenevia kanavatransistoreita varten lisätään levyn päälle aktiivinen laitetuuletin, joka estää rasituksessa kuumenevien komponenttien ylikuumenemisen (KUVA 3).



KUVA 3. Moottorien ohjauskortti robotin runkoon asennettuna

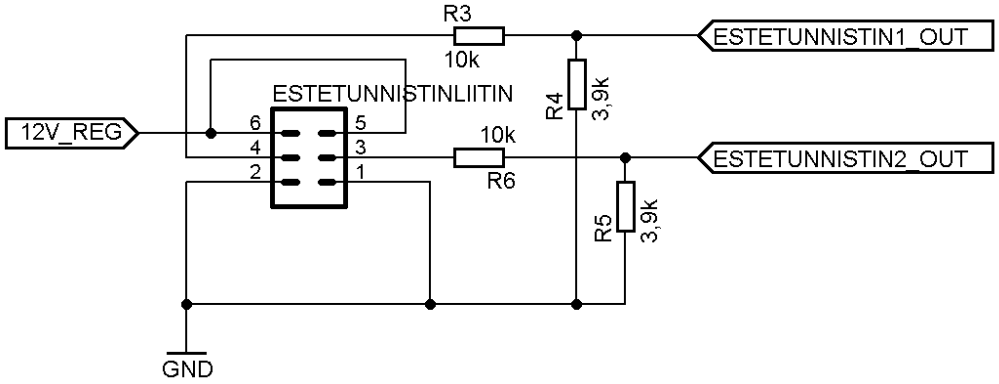
5.4 Lisälaitteet

Robottiin asennettavat lisälaitteet ovat estetunnistimet, servomoottorit, kaasuanturi ja radio-vastaanotin. Lisälaitteet kytketään mikro-ohjainkorttiin johtimilla.

Estetunnistimina toimivat S41-2-C-P pnp -anturit. Pnp-anturi toimii samalla periaatteella kuin tyypillinen pnp-transistori. Kun este havaitaan, päästävät ne käyttöjännitteensä läpi tulopinnille. Estetunnistimet toimivat 12 V:n tasajännitteellä, joka saadaan robotin tehonlähteestä. Estetunnistimet kytketään mikro-ohjainkorttiin liittimellä, jossa ovat käyttöjännitepinnit sekä tulopinni (KUVIO 20). Koska tunnistimien käyttöjännite on 12 V, tulee tulojännite rajoittaa Mbedille sopivaksi eli 3,3 V:iin. Tämä toteutetaan jännitteen jaolla (KAAVA 6):

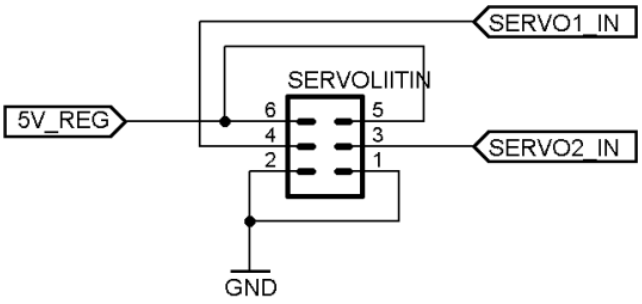
$$V_{OUT} = V_{IN}\left(\frac{R_4}{R_4+R_3}\right) = 12V \cdot \left(\frac{3{,}9k\Omega}{3{,}9k\Omega + 10k\Omega}\right) \approx 3{,}367V$$

(KAAVA 6.)



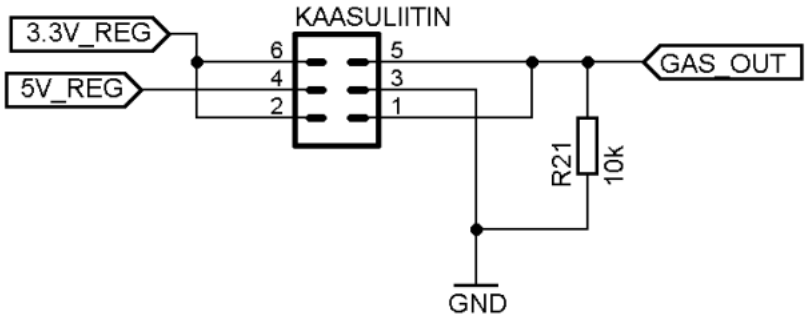
KUVIO 20. Estetunnistimien kytkentä

Robotissa kameran suuntaa muuttavat servomoottorit ovat mallia Parallax Standard Servo. Servomoottorit toimivat 5 V:n tasajännitteellä, joka otetaan mikro-ohjainkortin tehonlähteestä. Servomoottoreita ohjataan pulssinleveysmodulaatiolla, jossa pulssin leveys on 0,75 - 2,25 ms. Leveys vastaa servomoottorin kääntymistä asteina 0 - 180. Pulssin aikaväli on oltava 20 ms. Servomoottorit kytketään mikro-ohjainkorttiin johtimella (KUVIO 21).



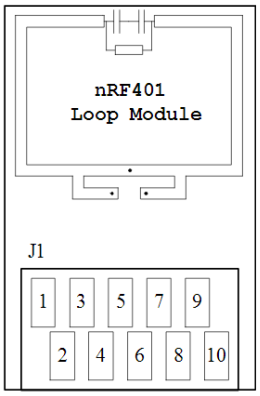
KUVIO 21. Servomoottorien kytkentä

Kaasuanturi on Figaro TGS822. Anturi toimii hehkuttamalla hehkulankaa, johon ilma ajautuu siinä olevan suojaritilän läpi. Anturissa on vastuslanka, jonka resistanssi muuttuu ilman kaasupitoisuuden mukaan. Vastuslangan resistanssi saadaan laskettua jännitteen jaon avulla. Hehkulanka kuumennetaan 5 V:n tasajännitteellä. Vastuslankaan ajetaan 3,3 V:n jännite. Kaasuanturi kytketään johtimella mikro-ohjainkorttiin ja asennetaan robotin etupäähän (KUVIO 22).



KUVIO 22. Kaasuanturin kytkentä

Radiovastaanottimeksi valitaan Nordicin nRF401 (KUVIO 23), joka on valmis radiovastaanotinmoduuli kehä-antennilla. Moduuli toimii 3,3 V:n käyttöjännitteellä. Moduuli käytännössä korvaa yhden galvaanisen digitaalisignaalin radiosignaalilla, joten sitä voidaan käyttää suoraan UART-sarjaliikenteen kanssa.



Pin	Name	Description
1	TXEN	Transmit enable TXEN = "1" ⇒ Transmit mode TXEN = "0" ⇒ Receive mode
2	PWR_UP	Power on/off PWR_UP = "1" ⇒ Power up (Operating mode) PWR_UP = "0" ⇒ Power down (Standby mode)
3	GND	Ground (0V)
4	CS	Channel selection CS="0" ⇒ 433.92MHz (Channel#1) CS="1" ⇒ 434.33MHz (Channel#2)
5	GND	Ground (0V)
6	DIN	Data input
7	GND	Ground (0V)
8	DOUT	Data output
9	GND	Ground (0V)
10	VDD	Power supply (+3-5V DC)

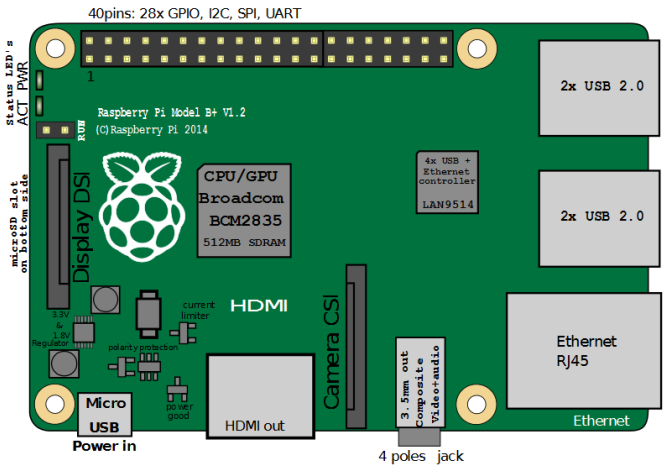
Figure 2.1. **nRF401** Loop Module pin placement and description

KUVIO 23. Radiovastaanottimen kytkentä (Nordic 2000, 2)

5.5 Raspberry PI

Raspberry PI (KUVIO 24) tietokonetta käytetään etäyhteyden muodostamisessa käyttäjän tietokoneen kanssa. Raspberry PI vastaa robotissa antureiden tietojen tallettamisesta sen muistikortille, langattoman

verkkoyhteyden tukiaseman luomisesta ja kameran kuvan ottamisesta sekä sen välittämisestä käyttäjän tietokoneelle. Raspberry PI -korttiin yhdistetään USB-verkkokamera, USB-langatonverkkokortti sekä Mbediltä tuleva sarjaliikenne liitin. Tehonlähteenä käytetään 5 V:n hakkuriteholähdettä, joka ottaa virtansa akuilta. Hakkuriteholähdettä ei suunnitella tai valmisteta itse vaan tässä käytetään Murata Powerin valmistamaa OKI-78SR-5/1.5-W36C-hakkuripiiriä. Raspberry PI -tietokoneeseen ohjelmoidaan ajettava ohjelma Python-ohjelmointikielellä.

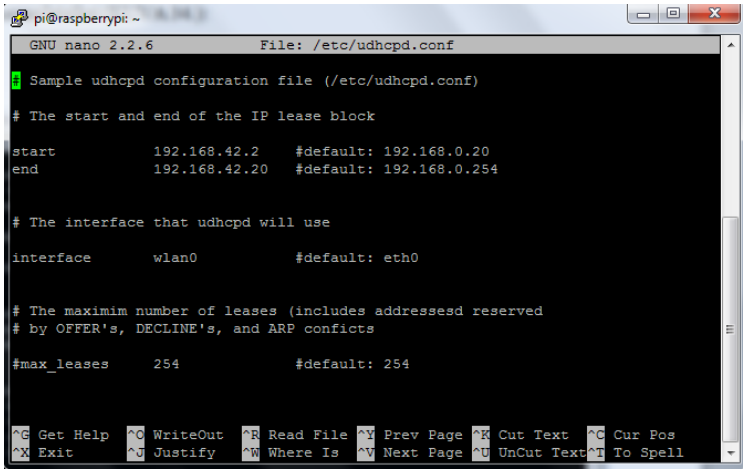


KUVIO 24. Raspberry PI -tietokone ja sen liitännät (Wikipedia 2015)

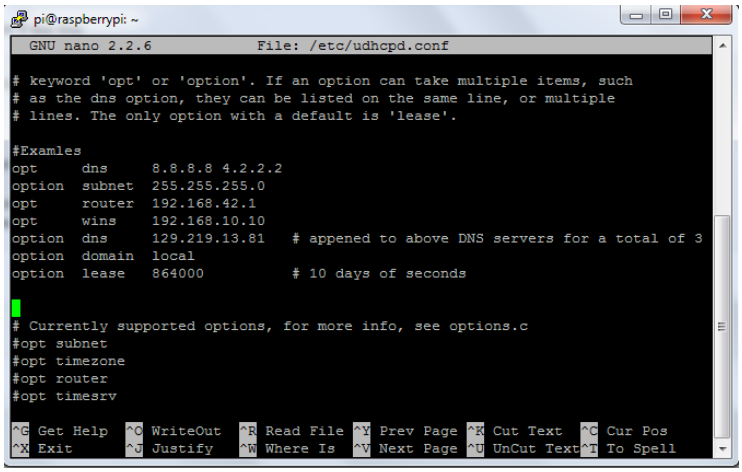
Raspberry PI -malleja on useita, ja tähän toteutukseen valitaan Raspberry PI 2 Model B –kortti. Siinä on neliytiminen ARM Cortex-A7 -mikroprosessori, jonka kellotaajuus on 900MHz, 8Gt ohjelmamuistia microSD-muistikortilla, johon asennetaan myös käyttöjärjestelmä, 1Gt RAM muistia, 21 yleiskäyttöistä I/O liitäntää, UART-sarjaliikenneliitäntä, neljä USB-porttia sekä grafiikkasuoritin ja useita muita hyödyllisiä liitäntöjä, joita ei kuitenkaan tässä esitellä tai käytetä. Ohjelmointi suoritetaan itse kortilla kirjoittamalla Python koodia tiedostoon, joka voidaan suorittaa ilman ylimääräistä kääntämistä.

Raspberry PI -korttiin asennetaan Raspbian Linux-käyttöjärjestelmä, jonka versio on Wheezy. Käyttöjärjestelmä asennetaan muistikortille. Asennuksen jälkeen päivitetään käyttöjärjestelmä ja laitepaketit uusimpiin versioihin. Jotta Raspberry saadaan toimimaan tukiasemana, siihen

liitetään USB-langatonverkkokortti. Tämän jälkeen täytyy muutamia asetuksia muuttaa ja ladata ohjelmapaketit. Ohjelmapaketeista tarvitaan hostapd, joka mahdollistaa tukiasemana toimimisen. Tämän jälkeen täytyy DHCP-asetuksia muuttaa. Asetuksiin muutetaan aliverkon IP-osoitteet, joita tukiasema antaa siihen kytketyille laitteille sekä reitittimen eli itse tukiaseman osoite. Asetukset muutetaan seuraavanlaisiksi (KUVIO 25 ja KUVIO 26):



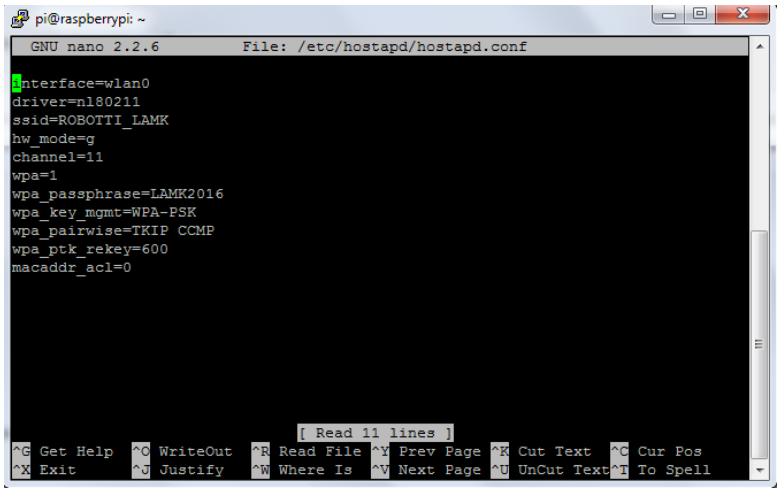
KUVIO 25. DHCP-asetukset



KUVIO 26. DHCP-asetukset

DHCP-asetusten muuttamisen jälkeen täytyy Raspberry PI -korttiin kytketyn langattomanverkkokortin IP-osoite muuttaa äsken vaihdettuun 192.168.42.1. Laitteen käynnistysasetukset muutetaan siten, että osoite pysyy staattisena käynnistäessä. Lopuksi hostapd:n asetukset muutetaan

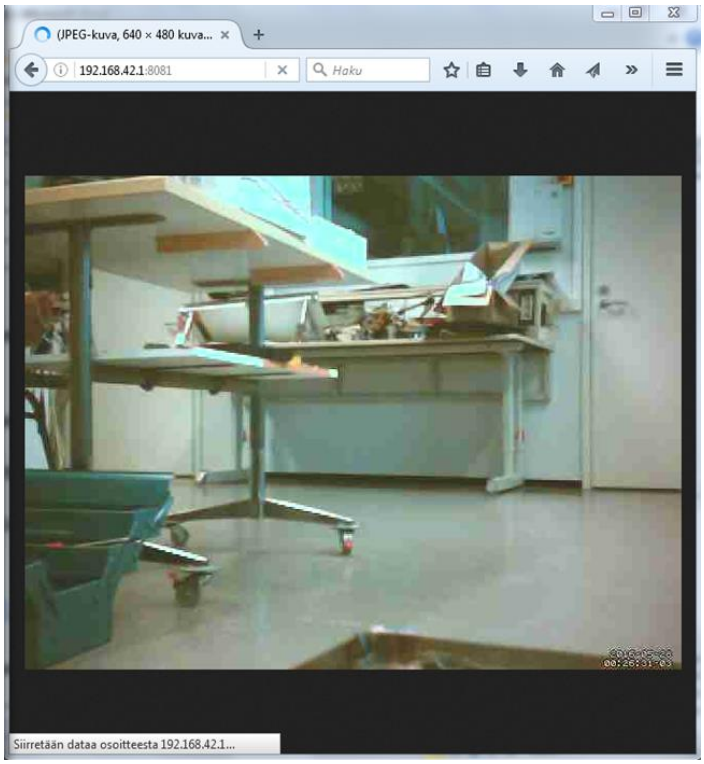
halutun mukaisiksi. Asetuksissa valitaan tukiaseman nimi ja verkon salasana. Salasanaksi valitaan LAMK2016 ja tukiaseman nimeksi ROBOTTI_LAMK (KUVIO 27).



KUVIO 27. Hostapd-tukiaseman asetukset

Tukiasema voidaan nyt käynnistää. Mutta jotta tukiasema käynnistyy aina laitteen käynnistyessä, lisätään hostapd Raspberry PI:n käynnistäessä ajettavaksi ohjelmaksi.

Kameran kuvan näyttämiseen käytetään motion-ohjelmaa, joka täytyy myös ladata ja asentaa Raspberry:yn. Ohjelman lataamisen jälkeen sen asetuksia muutetaan, jotta kuva olisi mahdollisimman reaaliaikainen. Lähetettävän kuvan tarkkuudeksi valitaan 640 x 480 kuvapistettä ja taajuudeksi 60 kuvaa sekunnissa. Kuva myös pakataan noin kolmasosan tarkkuudella ja laadusta tingitään myös kolmasosa. Videokuva näytetään verkkoselainikkunassa portilla 8081, eli kun yhteys on tukiasemaan kytkettynä, videokuva näkyy verkko-osoitteessa 192.168.42.1:8081 (KUVIO 28). Raspberry PI:n kytkettävänä kamerana käytetään USB HD -verkkokameraa. Kuitenkaan teräväpiirtolaatuun ei verkon välityksellä motion-ohjelmalla päästä, koska motion lähettää JPEG-suoratoistokuvaa, mikä pakkaa hyvin paljon lähetetyn kuvan laatua. Myöskään tällä ei päästä koskaan täysin reaaliaikaiseen videokuvaan ja lähetyksessä voidaan havaita noin puolen sekunnin viive.



KUVIO 28. Kameran videokuva selainikkunassa

Raspberry PI:n käyttöliittymä ohjelmoidaan Python-ohjelmointikielellä. Tarkoituksena on ohjelmoida käyttäjän näppäimistösyötteiden tulkinta, tiedonsiirron rajapinta Mbedille, mikä sisältää robotin käskykannan sekä robotin antureiden keräämän tiedon tallentaminen tiedostoon. Ohjelman ja ohjelmoinnin käyttöliittymänä toimii Windows-tietokoneella Putty-ohjelma, joka käyttää ssh-yhteyttä. Käyttöliittymään saadaan yhteys langattoman lähiverkon kautta Raspberry PI:n IP-osoitteella 192.168.42.1.

Näppäimistösyötteiden tulkintaan käytetään keskeytyksiin perustuvaa silmukkaa, jossa silmukka aika ajoin pyyhkäisee tietyin väliajoin ajettavan koodin, jossa ajetaan ohjelman muut toiminnot. Mikäli käyttäjä antaa näppäinsyötteen, ohjelma siirtyy rakenteeseen, jossa tehdään syötetylle näppäimelle määrätty komento. Esimerkiksi jos painetaan nuolinäppäin ylöspäin, ohjelma lähettää Mbedille ”forward”-käskyn eli käskyn liikkua eteenpäin.

Tiedonsiirron rajapintana sarjaliikenne toteutetaan robotin rajapinnan mukaan, eli baudinopeus pidetään samana sekä komennot lähetetään

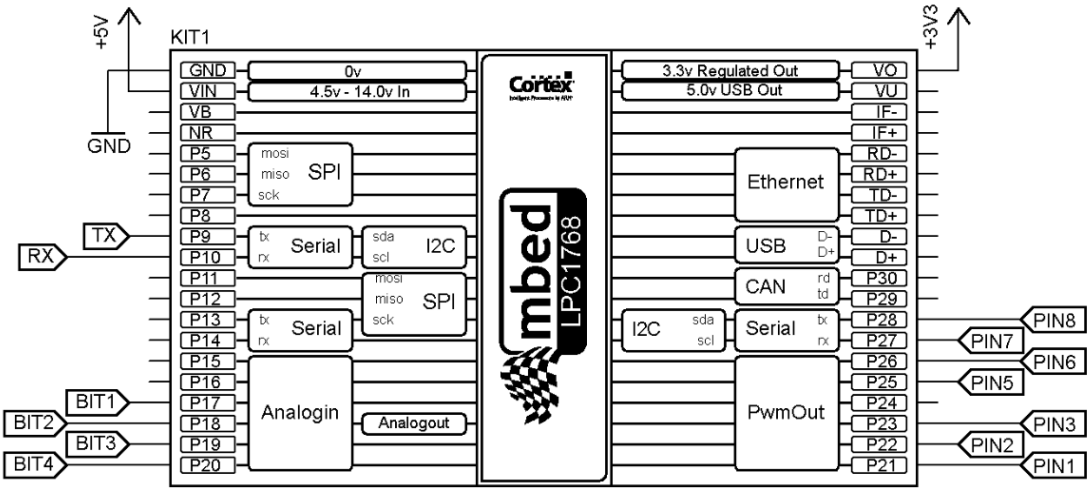
UART-sarjaliikenteellä string-muodossa, mikä vähentää häiriöriskiä ja parantaa ohjelman tulkintaa.

Aina tietyin väliajoin näppäimistöntulkinta silmukassa pyyhkäistään erillinen koodi, jossa huolehditaan robotin antureiden tiedon tallennuksesta. Ensin lähetetään sarjaliikenneväylään Mbedille pyyntö, minkä jälkeen Mbed lähettää antureiden tiedot. Tiedot tallennetaan erilliseen tekstitiedostoon. Antureiden tiedot tallennetaan muodossa: lämpötila, kosteusprosentti ja hiilimonoksidin pitoisuus. Tiedon perään lisätään kellonaika sekä päivämäärä tietojen myöhempää tarkastelua varten (LIITE 2: luku 3.1).

5.6 Radio-ohjain

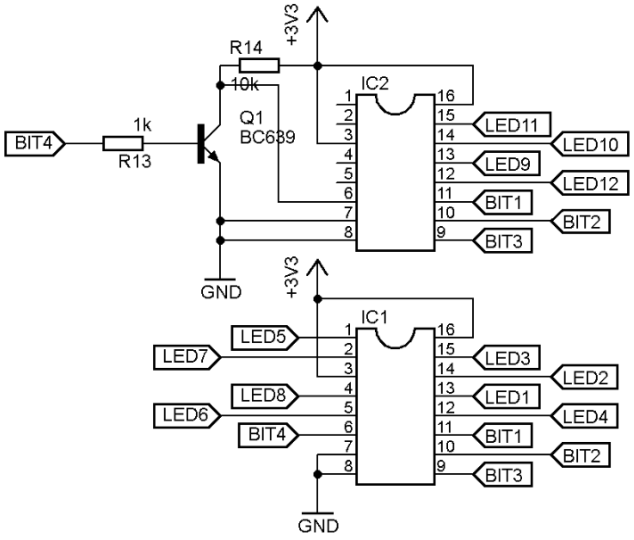
Radio-ohjaimeen tulee mikro-ohjain, näppäimistö käyttäjän syötteitä varten, LED-valot helpottamaan käyttöä, radiolähetin sekä tehonlähde 9 V:n paristoa varten. Radio-ohjaimen piirilevyt jyrsitään HPGL-piirilevyjyrsimellä ja kalustetaan käsin. Mikro-ohjain ohjelmoidaan C++-ohjelmointikielellä.

Radio-ohjain toteutetaan samalla mikro-ohjaimella kuin robotti eli Mbedillä. Mbediin kytketään 4x3 eBlocks™ matriisinäppäimistö, jonka on valmistanut Matrix Multimedia. Näppäimistössä on kaksitoista painiketta, jotka ovat 1-9, 0, tähti ja risuaita. Näppäimistö liitetään Mbedin piirilevyyn 9-pinnisellä DSUB9-liittimellä. Piirilevyyn suunnitellaan myös kaksitoista LED-valoa jokaista näppäintä varten sekä liitännät käyttöjännitteelle ja radiolähettimelle (KUVIO 29). Radiolähettimenä toimii sama, edellä mainittu, nRF401-moduuli.



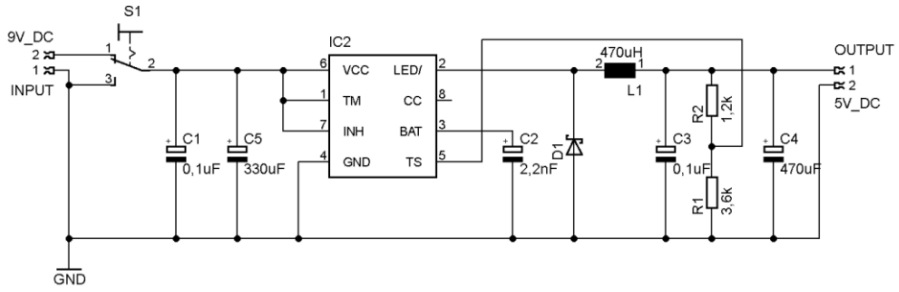
KUVIO 29. Mbedin kytkentä

LED-valoja ohjataan 74HCT4051N-digitaali-multiplexereillä (KUVIO 30). Multiplexerit on kytketty Mbediin 4-bittisellä rinnakkaisväylällä.



KUVIO 30. LED-valojen ohjauspiirit

Jännitelähteeksi suunnitellaan hakkuriteholähde, joka ottaa virran 9 V:n paristosta (KUVIO 31).



KUVIO 31. Radio-ohjaimen teholähteen kytkentä

Piirilevylle asetetaan ehdot valmiin näppäimistön levyn mukaan. Kaksi valmistettavaa piirilevyä pinotaan näppäimistön alapuolelle ja kiinnitetään näppäimistön kiinnitysreikiin. Piirilevyn koon on oltava LED-valojen näkymistä varten 130 x 55mm ja siinä on oltava neljä reikää kiinnitystä varten 80 x 40mm:n etäisyydellä ja siinä on oltava liittimet radiolähettimelle sekä tehonlähteen johtimille. Alemman piirilevyn, eli tehonlähteen tulee olla 95 x 55mm ja siinäkin on oltava neljä kiinnitysreikää sekä keskellä neljä reikää pariston kiinnitystä varten sekä perässä kytkin virran katkaisua varten (LIITE 1: luku 3.2 ja luku 3.5).

Radio-ohjaimelle ohjelmoidaan käyttäjän näppäinsyötteiden tulkitseminen, LED-valojen näyttäminen helpottaakseen käyttäjää ja tietoliikenne rajapinta vastaamaan robotin käskykantaa. Koska näppäimistö koostuu kahdestatoista painikkeesta, ohjelmoidaan ohjaimelle kaksi tilaa, joista toisessa robottia ohjataan ja toisella sen nopeutta on mahdollista muuttaa. Näppäimistön painikkeet valitaan loogisesti, eli ristikossa olevat painikkeet vastaavat ohjausta tai nopeuden lisäämistä tai hidastamista (TAULUKKO 3). Ohjelmoitavia kokonaisuuksia ovat siis seuraavat:

1. painikesyötteiden tulkitseminen
2. LED-valojen näyttö
3. sarjaliikenne, joka sisältää robotin käskykannan
4. tilan vaihto ja painikkeiden toiminnan valitseminen.

TAULUKKO 3. Näppäimistösyötteiden kartoitus

Painike	Toiminta (ohjaustila)	Painike	Toiminta (nopeustila)
2	liikkuminen eteenpäin	2	nopeuden lisäys
8	peruuttaminen	8	nopeuden hidastus
4	kääntyminen vasemmalle	6	kääntymisnopeuden lisäys
6	kääntyminen oikealle	4	kääntymisnopeuden hidastus
5	pysäytys	5	nopeuksien palautus
1	vaakaservon liikutus	risuaita	tilan vaihto (molemmat tilat)
3	pystyservon liikutus	tähti	automaattiohjaus (molemmat tilat)

Painikesyötteiden tulkitsemiseen käytetään Mbed-sivuilta löytyvää valmista koodia, joka todetaan hyväksi käyttötarkoitukseensa. Koodin on kirjoittanut Dimiter Kentri, ja se on vapaasti käytettävissä (LIITE 2: luku 2.2.1).

LED-merkkivaloille ohjelmoidaan funktio, joka sytyttää merkkivalon riippuen painetusta näppäimestä. Mikäli ohjaustila on valittuna, merkkivalo pysyy päällä niin kauan kuin painiketta pidetään pohjassa, kun taas nopeustilassa merkkivalo välähtää vain kerran. Nopeustilassa ollessa nopeutta muutetaan aina pykälän ylös- tai alaspäin kerran painettaessa. Ohjaustilassa ollessa robotti liikkuu painettuun suuntaan niin kauan kun painiketta pidetään pohjassa. Automaattiohjauksen ollessa kytkettynä merkkivaloja sytytetään järjestyksessä puolelta toiselle. Merkkivalojen avulla käyttäjä saa tiedon siitä, missä tilassa robottia tällä hetkellä ohjataan (LIITE 2: luku 2.3.1).

Sarjaliikenne rajapinnasta tehdään robotin rajapinnan mukainen, eli baudinopeus pidetään samana sekä komennot lähetetään radiolähettimen kautta string-muodossa, mikä vähentää häiriöriskiä sekä helpottaa käskyn tulkintaa ohjelmoitaessa (LIITE 2: luku 2.4.1).

Tilanvaihto toteutetaan funktiolla, jossa pääohjelma pyytää käytettävää tilaa riippuen valitusta tilasta. Painettu painike tulkitaan tilafunktiossa ja toimitaan painetun painikkeen mukaan. Esimerkiksi, jos ohjaustilassa painetaan painiketta kaksi, lähetetään robotille käsky ”forward”, eli

ohjataan robotti liikkumaan eteenpäin. Funktiossa on myös lippumuuttuja, jonka ylösnostaminen tarkoittaa sitä, että käyttäjä haluaa muuttaa tilaa. Tilan muutos palautetaan pääohjelman muuttujaan (LIITE 2: luku 2.5.1).

Pääohjelma toimii alustusten jälkeen silmukassa, jossa haetaan painettu painike, minkä jälkeen pyydetään tilanvaihtoluokan funktioita ja toimitaan painetun painikkeen mukaan. Mikäli automaattiohjaus on kytkettynä päälle, pääohjelma odottaa niin kauan kunnes automaattiohjaus kytketään taas pois päältä. Pääohjelma huolehtii myös merkkivalojen sytytyksestä ja tilanvaihdon tulkinnasta (LIITE 2: luku 2.1).

5.7 Laitetestaus ja mittaukset

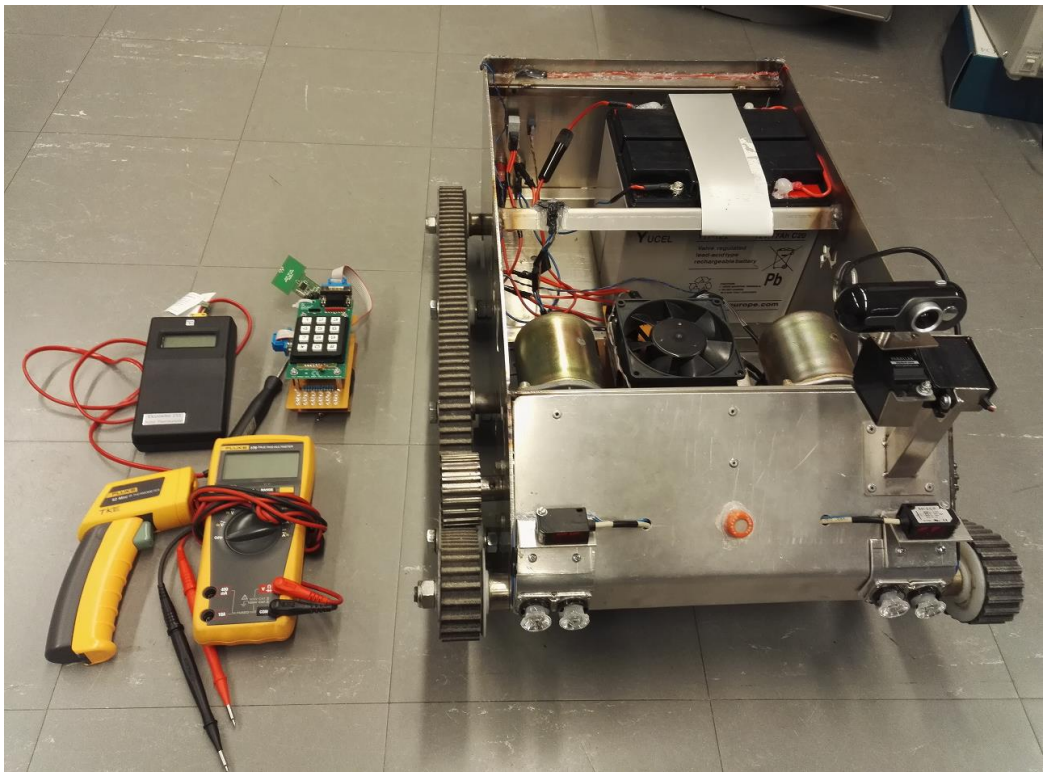
Ympäristörobotin kokonaisuus testataan sille alun perin mainittujen osa-alueiden mukaisesti. Testaukset suoritetaan laboratorio-olosuhteissa, jossa käytetään mittalaitteita tarkemman mittaustuloksen saavuttamiseksi. Lähtökohtana on, että robottiin ei kosketa, sitä ladata, sammuteta tai siirretä testauksen aikana. Näin saadaan mahdollisimman tarkat tulokset vaadituin kriteerein. Tehtävät testikokonaisuudet ovat seuraavat:

1. tunnin mittainen toimivuus- ja käyttöaikatestaus
2. liikkuvuuden ja signaalietäisyyden testaus.

5.7.1 Mittaukset

Ensimmäiseen testikokonaisuuden tarkoitus on mitoittaa robotin käyttöaikaa ja toimintavarmuutta pitkäjaksoisessa käytössä. Ennen varsinaista testiä varmistetaan onnistumisen mahdollisuudet alustavilla mittauksilla. Ensin mitataan robotin kuluttamaa kokonaisvirtaa liikkuessaan ja paikalla ollessaan. Ennen testausta robotin akut ladataan täyteen. Testikokoonpanossa robotin telahihnat ovat ilmassa, mistä syystä tarkkaa virrankulutusta ei saada mitattua, mutta virhemarginaaliksi arvioidaan noin 20 %. Vaikka virhemarginaali onkin melkoisen suuri, ei sillä ole suurta merkitystä, koska tarkoitus on vain kartoittaa tunnin yhtäjaksoisen

käyttöajan saavutettavuutta. Virrankulutus mitataan suoraan akun päävirtajohtimesta oskilloskoopin virtamittauspäällä. Ennen varsinaista testiä robotin akut ladataan jälleen täyteen. Testin aikana käytetään sekä radio-ohjainta että PC-etäohjausta. Testissä tarkkaillaan mahdollisia tietoliikennehäiriöitä ja yhteyden katkoksia. Mittausten aikana tarkkaillaan myös robotin antureiden antamien tietojen eheyttä ja paikkansapitävyyttä. Robotin komponenttien lämpötilaa ja akkujen jännitettä mitataan mittalaitteilla. Testauslaitteistoon kuuluvat yleismittari FLUKE 179, kannettava tietokone HP ProBook 4330s, lämpötilamittarit Elcometer 213 ja FLUKE 62 Mini. Elcometer 213 on kannettava lämpötilamittari johdollisella mittapäällä, jolla voidaan tehdä lämpötila mittauksia painamalla se haluttuun komponenttiin. FLUKE 62 Mini on kannettava, etäkäyttöinen infrapunasäteilyyn perustuva lämpötilamittari. Testauskokoonpanoon kuuluvat mittalaitteiden lisäksi robotti ja sen radio-ohjain (KUVA 4). Testaus suoritetaan sisätiloissa. Robotin komponenttien lämpötilaa mitataan testauksessa kymmenen minuutin väliajoin. Akkujen varaus mitataan ennen testausta, sen puolessa välissä ja lopussa. Robotin mittaamat anturiarvot tallentuvat automaattisesti sen Raspberry PI-tietokoneeseen, ja niitä tulkitaan testin suorituksen jälkeen. Mitatut arvot kirjataan ylös mittauspöytäkirjaan. Testaukseen kuuluvat myös etäkäytön ja robotin liikkumisen arviointi.



KUVA 4. Yhden tunnin käyttöaika- ja toimintavarmuustestauksen kokoonpano

Toisena testauksena tehdään robotin liikkuvuuden ja signaalin etäisyyden mittaus. Testissä liikkuvuutta tarkkaillaan käyttäjän näkökulmasta, eli tarkastellaan kuinka herkästi robotti vastaa annettuihin komentoihin. Tarkasteltavana ovat myös telahihnojen ja moottoreiden toiminta eli aiheutuuko pienistä epätasaisuuksista robotin kulkemiselle ongelmia. Signaalin etäisyyden mittaus toteutetaan asettamalla robotti kulkemaan eteenpäin niin kauan kunnes signaali käyttöliittymän ja robotin välillä katkeaa kokonaan. Etäisyys mitataan mittanauhalla, minkä jälkeen robotti asetetaan tätä etäisyyttä hiukan lähemmäs ja arvioidaan tämän maksimietäisyyden toimintaa ja mahdollisia tietoliikenteessä ilmeneviä häiriöitä. Etäisyys mitataan erikseen sekä PC-etäyhteydellä että radio-ohjaimella. Testaus suoritetaan tilantarpeen vuoksi ulkotiloissa.

5.7.2 Mittaustulokset

Alustavassa virrankulutusmittauksessa saadaan oskilloskoopilla mitatut tulokset (TAULUKKO 4).

TAULUKKO 4. Alustavan virrankulutustestauksen tulokset

Tilanne	Virrankulutus (A)
Moottorit käy suurimmalla nopeudella	2,16
Moottorit käy pienimmällä nopeudella	3,52
Moottorit pysäytettynä	0,6

Virrankulutuksen pohjalta lasketaan teoreettiset maksimi- ja minitoiminta-ajat (KAAVA 7 ja KAAVA 8).

$$t_{max} = \frac{(akun\ kapasiteetti\ \times virhe\ \%)}{(kulutus\ \times virhe\ \%)} = \frac{(17Ah \cdot 0{,}9)}{(0{,}6A \cdot 1{,}2)} \approx 21h$$

(KAAVA 7.)

$$t_{min} = \frac{(akun\ kapasiteetti\ \times virhe\ \%)}{(kulutus\ \times virhe\ \%)} = \frac{(17Ah \cdot 0{,}9)}{(3{,}5A \cdot 1{,}2)} \approx 3{,}6h$$

(KAAVA 8.)

Tunnin käyttöaika- ja toimintavarmuustestauksessa saadaan lämpötilamittareilla ja yleismittarilla mitatut mittaustulokset (TAULUKKO 5).

TAULUKKO 5. Käyttöaika- ja toimintavarmuustestauksen mittaustulokset

Aika	Lämpötila ˚C (ohjauskomponentti)	Lämpötila ˚C (vakiovirtapiiri)	Akkujen varaus (V)
0 min	24,6	24,6	26,92
10 min	26,2	30,2	
20 min	27,6	32,2	
30 min	26,4	31,0	25,35
40 min	33,2	31,6	
50 min	33,8	32,4	
60 min	32,8	33,2	25,27

Käyttöaikatestausta tehtäessä ei havaita lainkaan tiedonsiirron häiriöitä. Etäyhteys langattomalla verkkoyhteydellä toimii moitteettomasti testauksen aikana. Pisin etäisyys tietokoneen ja robotin välillä testauksen aikana on korkeintaan noin kymmenen metriä. Radio-ohjaimen käytössä ei myöskään havaita ongelmia ja tietoliikenneyhteys robottiin toimii ilman häiriöitä. Tietokoneeseen välitetty kameran kuva tulee karkeasti arvioiden noin puolen sekunnin viiveellä. Kamerakuvan huomataan myös pätkivän aika ajoin. Antureiden tiedot tallentuvat testissä ilman suurempia virheitä Raspberry PI -tietokoneeseen. Usean tuhannen mittaustiedon mukana on mittausvirheestä tai tiedonsiirrosta johtuneita vääriä tuloksia noin kymmenkunta. Robotin välittämä kamerakuva toimii noin puolen sekunnin viiveellä ja on hiukan epäselvän laatuinen, mutta riittävä ympäristön tarkkailuun.

Liikkuvuustestauksessa robotin liikkeessä ei ole huomattavissa ongelmia. Pienellä epätasaisuudella ei huomata olevan ongelmia. Robotti kykenee ylittämään ainakin noin kahden senttimetrin korkean esteen. Radio- ja verkkoyhteys toimii ongelmitta. Signaalin kantaman testauksessa langaton verkkoyhteys oli saavutettavissa noin 20 metrin etäisyydellä ja radio-ohjainyhteys noin 30 metrin etäisyydellä. 30 metrin etäisyydellä huomataan tiedonsiirron ongelmia radio-ohjaimella. Langaton verkkoyhteys ei ole yhdistettävissä enää yli 20 metrin etäisyydellä, joten sen toimintaa ei voida tällä etäisyydellä testata. Toimintaetäisyytenä voidaan pitää noin 20:tä metriä.

6 YHTEENVETO

Opinnäytetyön tarkoituksena oli tutustua ympäristörobotteihin ja niiden tekniseen puoleen. Tätä tutkittiin niiden kehityksen kautta aina nykyhetkeen asti, jossa opittiin lähinnä syitä, mistä robotit saivat alun perin alkunsa ja miten niitä on lähdetty myöhemmin kehittämään muihinkin tarkoituksiin, kuten ympäristön tutkimiseen, ja mihin suuntaan ympäristörobotteja tulevaisuudessa pyritään kehittämään. Esimerkiksi syy siihen, miksi ympäristön tutkimiseen käytetään nykyään robotteja, johtuu siitä, että tutkimuskohteet ovat muuttuneet muutamissa vuosikymmenissä paljon hankalammiksi olosuhteiltaan ja taloudellisten syiden vuoksi. Tutkittavat ympäristöt ovat siirtyneet rannikoilta arktisiin oloihin ja syvävesistöihin sekä ulkoavaruuteen. Taloudellisesti robotit vähentävät miehistön tarvetta, suuria miehistönkuljetus ajoneuvoja ja pienentävät projektin laajuutta muun muassa turvallisuusbudjetin pienentyessä henkilöriskien vähentyessä robottien ansiosta.

Opinnäytetyössä selvitettiin myös mittaustekniikkaan liittyviä teknisiä ominaisuuksia ja sitä, mitä vaatimuksia ne asettavat, jota tutkimalla opittiin mittauslaitteiden kehityksestä varhaisista nykyisiin. Nykyään lähes kaikkien ympäristörobottien suurin haaste piilee energiatehokkuudessa, joka juurtaa käyttöaikojen harppauksesta muutamista tunneista aina vuosiin asti. Suurin osa nykyaikana toimivista ympäristöroboteista hyödyntää omavaraista energian tuotantoa, eikä käyttöenergia enää ole rajattu pelkästään polttoaineen tai akkujen määrään. Tutkimalla energiantuotanto järjestelmiä opittiin myös nykyään käytettyjä energian tuotantomalleja, kuten aurinkoenergia.

Käytännön esimerkin avulla opittiin syventämään teoriaosuudessa opittuja asioita. Koska mitään valmista ohjetta tai esimerkkiä ei ollut juurikaan löydettävissä, tuli soveltaa tässä työssä opittuja ja ymmärrettyjä asioita. Siksi robotille asetettiin aluksi tavoitteet, joihin pyritään vastaamaan ja jotka teknisesti voidaan itsenäisesti toteuttaa.

Robotin suunnittelu- ja valmistusvaiheissa opittiin yleisen elektroniikan suunnittelua, kuten tasavirtamuuntimien ja tiedonsiirtoväylien kytkentää, virran ja jännitteen rajoitusta, piirilevyn suunnittelua ja kalustusta sekä sähkömoottorien ohjausta. Robotin toteutusvaiheessa opittiin ohjelmointia, joista tärkeimpänä olio-pohjaisen koodin kirjoittamista, ja Linux-tietokoneen ominaisuuksia ja asentamista. Opittiin myös Python-ohjelmointikielellä koodaamista.

Robotin testausvaiheessa päästiin näkemään, toteutuiko robotille alun perin asetetut tavoitteet. Testaustulosten perusteella tunnin mittainen toiminta on täysin suoritettavissa ja saaduista mittaustuloksista päätellen voitaisiin arvioida jopa kolmen tunnin toiminta-ajan olevan mahdollista saavuttaa ilman robotin lisäkehittämistä. Antureiden keräämä tieto on ehyttä, eikä mittausvirheitä testauksissa juurikaan ilmennyt. Tiedonsiirto ei signaalinkantamalla testauksissa katkeillut tai aiheuttanut epävakaata robotin toimintaa. Robotin liikkuminen oli sulavaa, eikä ongelmia moottoreissa tai telahihnoissa ilmennyt. Lämpötilat ohjauskomponenteissa tai vakiovirtapiirissä eivät nousseet liikaa, mikä mahdollistaa pitkäaikaisen käytön ilman ylikuumenemisen riskiä. Koska mitään häiriöitä ei testauksissa juurikaan huomattu, olisi robotti käytännössä valmis ympäristössä käytettäväksi. Kuitenkin testauksissa huomattiin parannettavia asioita, kuten kameran kehno kuvanlaatu ja viive, tukiaseman toimimisen epävarmuus ja lyhyehkö signaalin kantama. Näitä asioita voitaisiin parantaa käyttämällä Raspberry PI:llä parempaa kuvanvälitysohjelmaa tai suoraan levylle kytkettävää kameraa, jolla saavutettaisiin kevyempi suoritinkäyttö. Tukiaseman toimintavarmuutta voitaisiin ohjelmallisesti parantaa. Signaalin kantaman parannus onnistuisi asentamalla verkkokorttiin tai radio-ohjaimeen suurempi antenni ja antenninvahvistuspiiri. Vaihtoehtoisesti voitaisiin robottiin toteuttaa 4G-yhteys, mikä mahdollistaisi lähes rajattoman toimintasäteen. Vaikka 4G-yhteys olisi teknisesti täysin toteutettavissa, vaatisi se kuitenkin maksullisen operaattorisopimuksen tekemistä.

Lisäkehittäminen robotille on hyvin mahdollista. Uusien antureiden lisääminen mahdollistaisi robotille todellisia käyttökohteita. Antureita voisi olla tuulennopeutta ja suuntaa mittaava anturi, sademääräanturi ja ilmanpaineanturi, jolloin robottia voitaisiin käyttää liikkuvan sääaseman ominaisuudessa. Tämä toki vaatisi myös 4G-yhteyden toteuttamisen sekä robotin suojaamisen kokonaan kosteudelta. Paikannuslaitteen (GPS) asentaminen mahdollistaisi esimerkiksi tontin tai kentän pinta-alan mittaamisen. Kuitenkaan robottia ei ole valmistettu maastossa liikkumiseen tai testattu siellä, joten vaikeakulkuisen alueen mittaaminen vaatisi robotin liikkumislaitteiston muuttamista, mikä voisi olla esimerkiksi suuret kumipyörät, joita liikutettaisiin telahihnan välityksellä, jolloin epätasaisessa maastossa liikkuminen paranisi huomattavasti.

LÄHTEET

Dunbabin, M. & Marques, L. 2012. Robotics for Environmental Monitoring. IEEE ROBOTICS & AUTOMATION MAGAZINE 3/2012, 24 - 39.

Figaro. 2009. TGS 822 - for the detection of Organic Solvent Vapors [viitattu 22.8.2016]. Saatavissa: http://www.figarosensor.com/products/822pdf.pdf

Honeywell Technical Note. 2012. I^2C Communication with the Honeywell HumidIcon™ Digital Humidity/Temperature Sensors [viitattu 15.7.2016]. Saatavissa: http://sensing.honeywell.com/index.php?ci_id=142171

Nocks, L. 2007. The Robot: The Life Story of a Technology. Westport, CT: Greenwood Press.

Nordic. 2000. PRODUCT SPECIFICATION nRF401 Loop Kit [viitattu 22.8.2016]. Saatavissa: http://www.moodle2.tfe.umu.se/pluginfile.php/26024/mod_resource/content/1/nRF401-LOOP_KITrev1_0.pdf

RobotWorx. 2015. The history of industrial robots [viitattu 14.11.2015]. Saatavissa: https://www.used-robots.com/education/the-history-of-industrial-robots

Saha, S. 2008. Introduction to Robotics. New Delhi: Tata McGraw-Hill Publishing Company Limited.

Wikipedia. 2014. REMUS 100 used by Finnish Navy [viitattu 16.1.2016]. Saatavissa: https://en.wikipedia.org/wiki/REMUS_(AUV)#/media/File:REMUS_100_Merivoimien_vuosip%C3%A4iv%C3%A4_2014_01.JPG

Wikipedia. 2015. Location of connectors and main ICs on Generation 1 + revision 1.2 and Generation 2 [viitattu 13.9.2016]. Saatavissa: https://upload.wikimedia.org/wikipedia/commons/c/ca/Raspberry_Pi_B%2B_rev_1.2.svg

LIITTEET

LIITE 1 YMPÄRISTÖROBOTIN PIIRIKAAVIO- JA PIIRILEVYKUVALIITE

LIITE 2 YMPÄRISTÖROBOTIN KOODILIITE

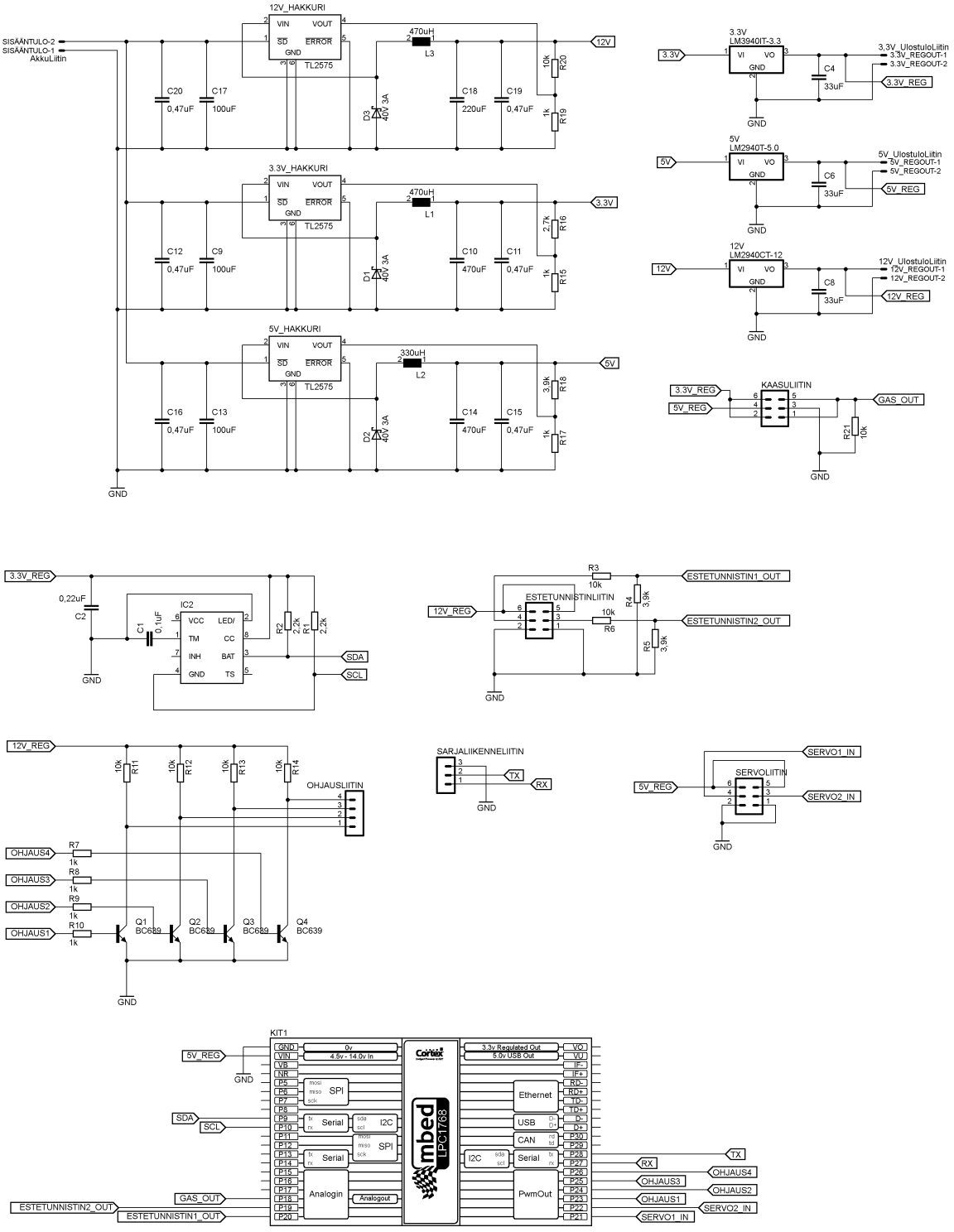
YMPÄRISTÖROBOTIN PIIRIKAAVIO- JA PIIRILEVYKUVALIITE

LIITE 1

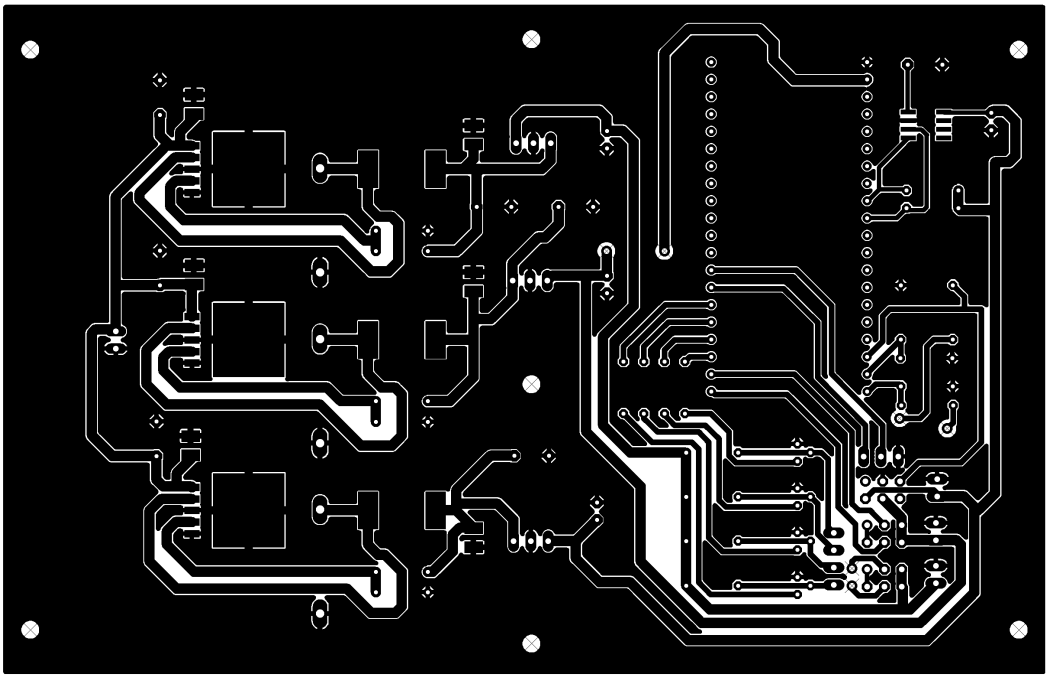
SISÄLLYS

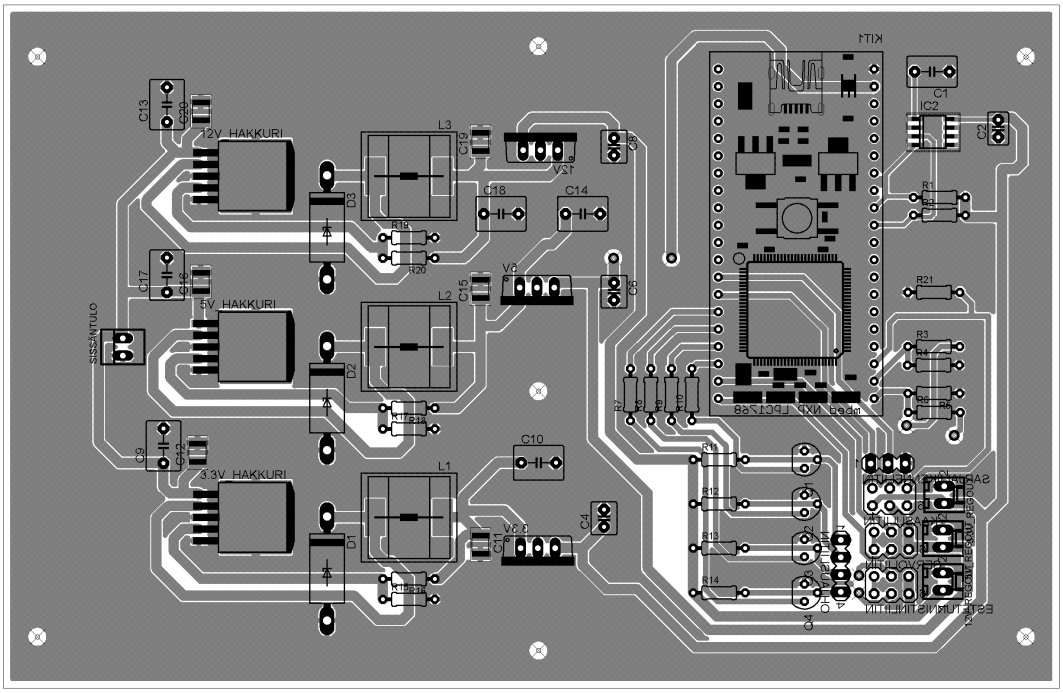
1 MIKRO-OHJAINPIIRILEVY

1.1 Piirikaavio



1.2 Piirilevykuvat



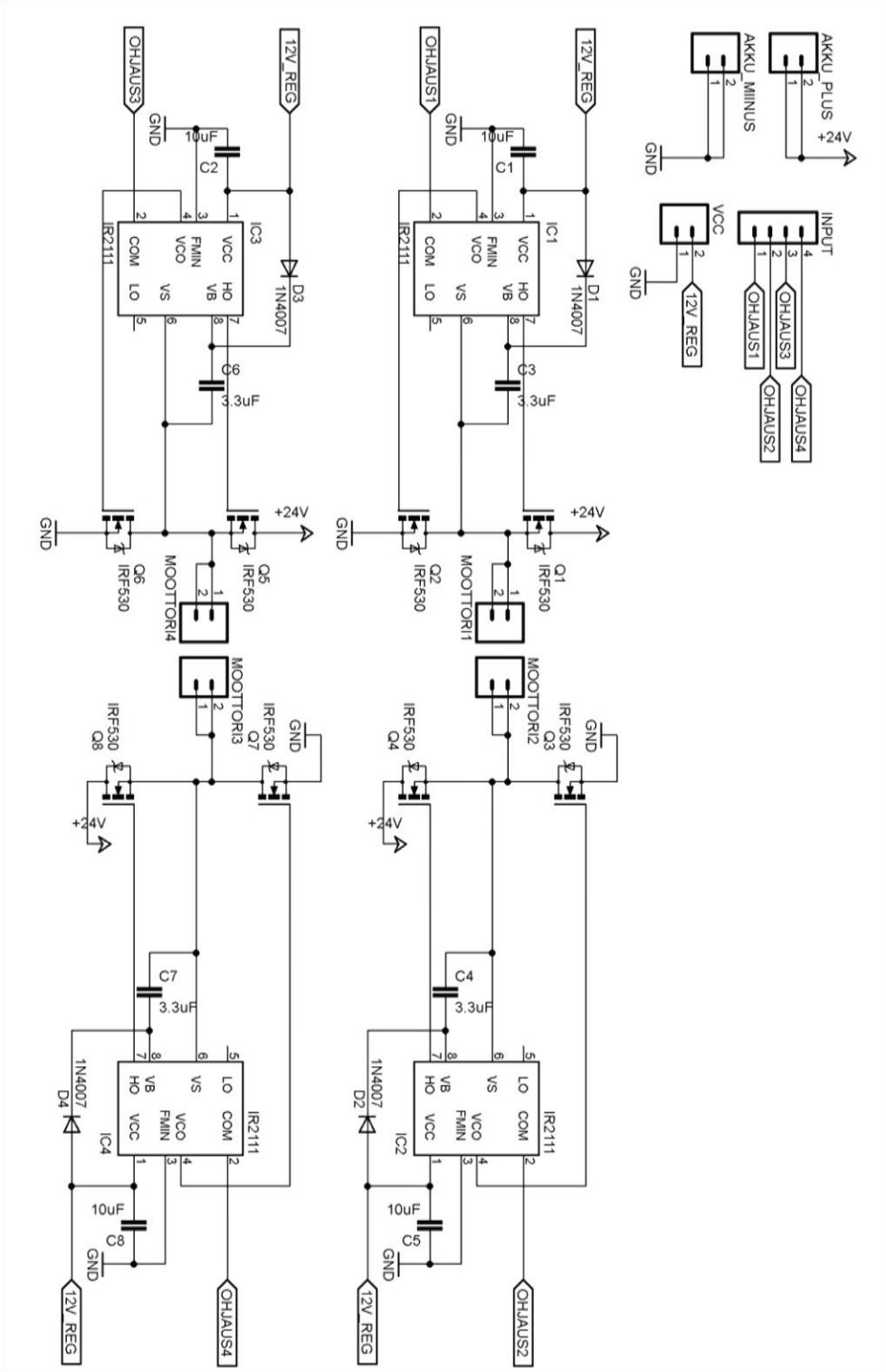


1.3 Komponenttiluettelo

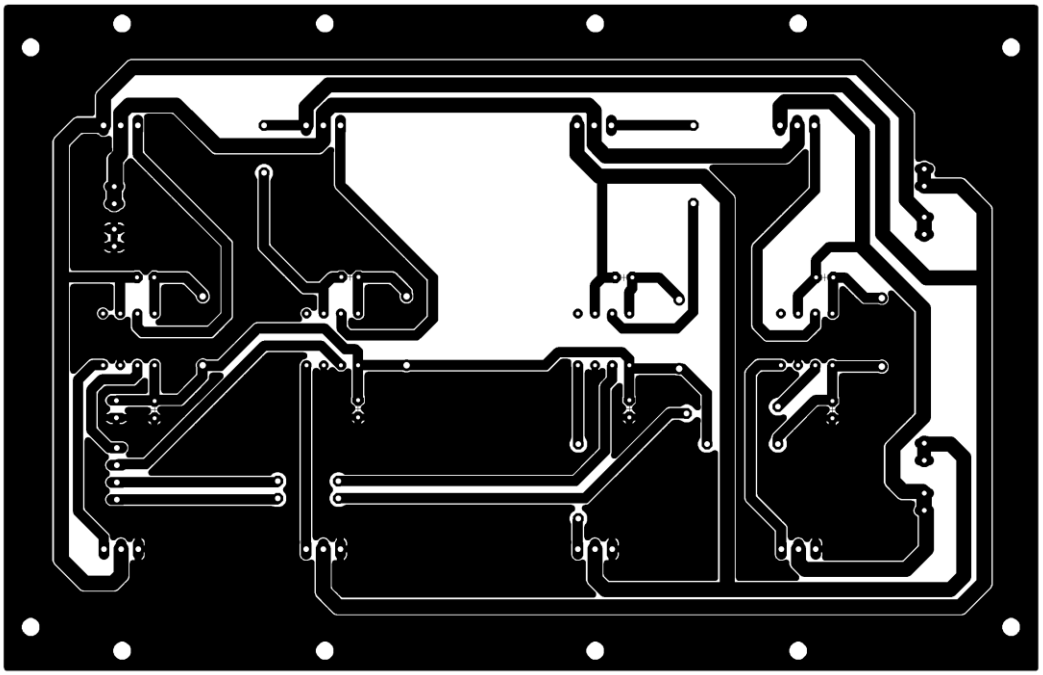
Nimi	Komponentti	Nimi	Komponentti
3.3V	LM3940IT-3.3 (TO-220)	R2	(Metal-film) 2,2k/0,6W/1%
3.3V_HAKKURI	TL2575 (TO-263-5)	R3	(Metal-film) 10k/0,6W/1%
3.3V_REGOUT	LIITINRIMA UROS 2x2,54mm	R4	(Metal-film) 3,9k/0,6W/1%
5V	LM2940T-5.0 (TO-220)	R5	(Metal-film) 3,9k/0,6W/1%
5V_HAKKURI	TL2575 (TO-263-5)	R6	(Metal-film) 10k/0,6W/1%
5V_REGOUT	LIITINRIMA UROS 2x2,54mm	R7	(Metal-film) 1k/0,6W/1%
12V	LM2940CT-12 (TO-220)	R8	(Metal-film) 1k/0,6W/1%
12V_HAKKURI	TL2575 (TO-263-5)	R9	(Metal-film) 1k/0,6W/1%
12V_REGOUT	LIITINRIMA UROS 2x2,54mm	R10	(Metal-film) 1k/0,6W/1%
C1	WIMA (Film-cap) 0,1uF/250V	R11	(Metal-film) 10k/0,6W/1%
C2	WIMA (Film-cap) 0,22uF/50V	R12	(Metal-film) 10k/0,6W/1%
C4	(Al E-cap) 33uF/63V	R13	(Metal-film) 10k/0,6W/1%
C6	(Al E-cap) 33uF/63V	R14	(Metal-film) 10k/0,6W/1%
C8	(Al E-cap) 33uF/63V	R15	(Metal-film) 1k/0,6W/1%
C9	(Al E-cap) 100uF/63V	R16	(Metal-film) 2,7k/0,6W/1%
C10	(Al E-cap) 470uF/25V	R17	(Metal-film) 1k/0,6W/1%
C11	(Ceramic-cap) 0,47uF/35V	R18	(Metal-film) 3,9k/0,6W/1%
C12	(Ceramic-cap) 0,47uF/35V	R19	(Metal-film) 1k/0,6W/1%
C13	(Al E-cap) 100uF/63V	R20	(Metal-film) 10k/0,6W/1%
C14	(Al E-cap) 470uF/63V	R21	(Metal-film) 10k/0,6W/1%
C15	(Ceramic-cap) 0,47uF/35V	SARJALIIKENNE	LIITINRIMA UROS 3x2,54mm
C16	(Ceramic-cap) 0,47uF/35V	SERVOLIITIN	LIITINRIMA UROS 3x2x2,54mm
C17	(Al E-cap) 100uF/63V	SISÄÄNTULO	JOHDINLIITIN 2x5mm^2
C18	(Al E-cap) 220uF/25V		
C19	(Ceramic-cap) 0,47uF/35V		
C20	(Ceramic-cap) 0,47uF/35V		
D1	1N5822 40V/3A		
D2	1N5822 40V/3A		
D3	1N5822 40V/3A		
ESTETUNNISTIN	LIITINRIMA UROS 3x2x2,54mm		
IC2	HIH6030 (SOIC-8)		
KAASULIITIN	LIITINRIMA UROS 3x2x2,54mm		
KIT1	MBED NXP LPC1768		
L1	WE-PD 470uH/1,5A (shielded)		
L2	BOURNS 330uH/1,4A (shielded)		
L3	WE-PD 470uH/1,5A (shielded)		
OHJAUSLIITIN	LIITINRIMA UROS 4x2,54mm		
Q1	BC639 (TO-92)		
Q2	BC639 (TO-92)		
Q3	BC639 (TO-92)		
Q4	BC639 (TO-92)		
R1	(Metal-film) 2,2k/0,6W/1%		

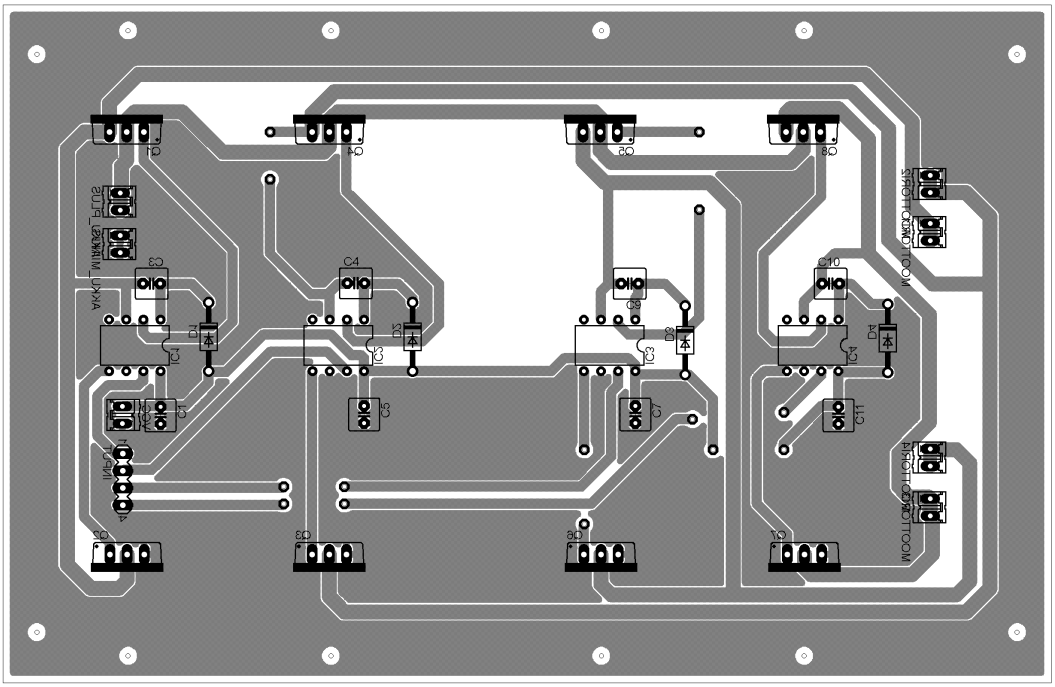
2 MOOTTOREIDEN OHJAUSPIIRILEVY

2.1 Piirikaavio



2.2 Piirilevykuvat



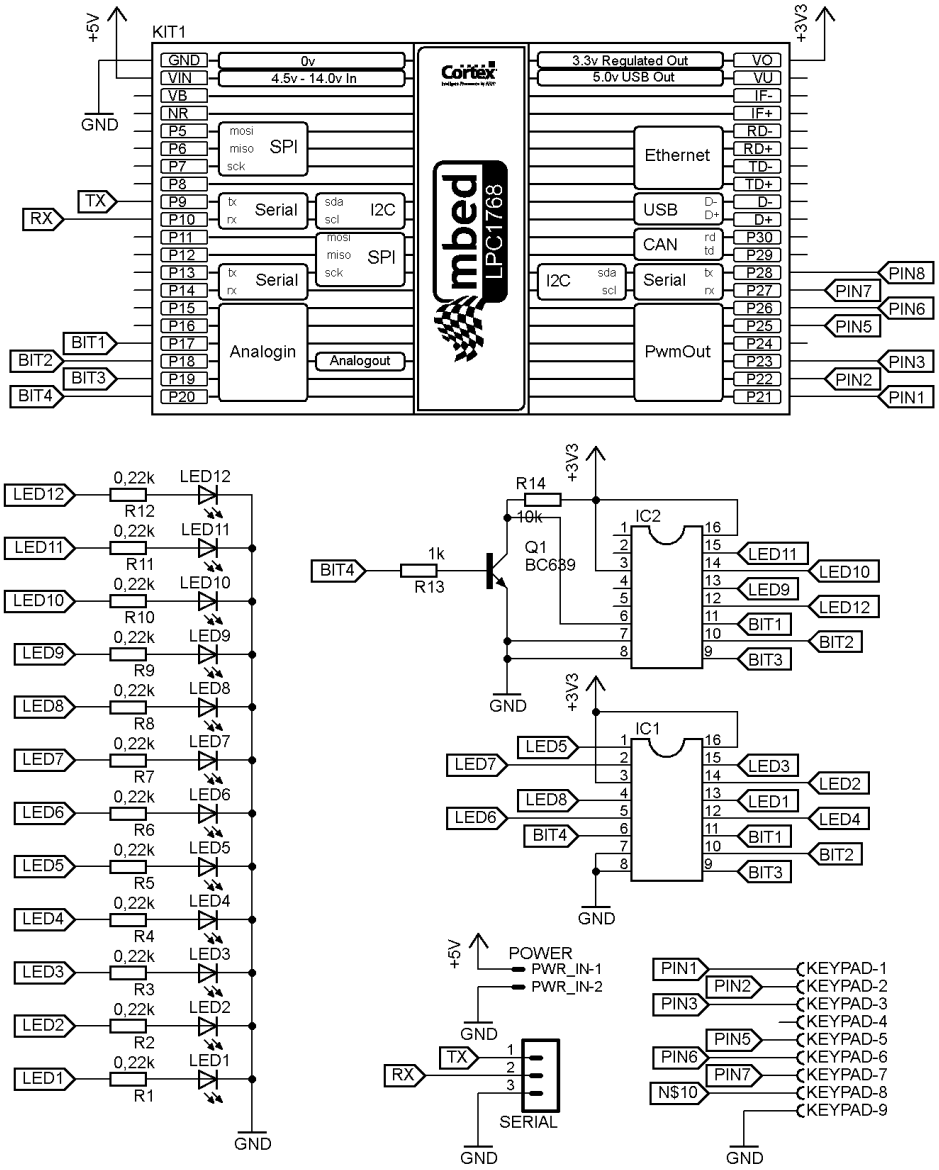


2.3 Komponenttiluettelo

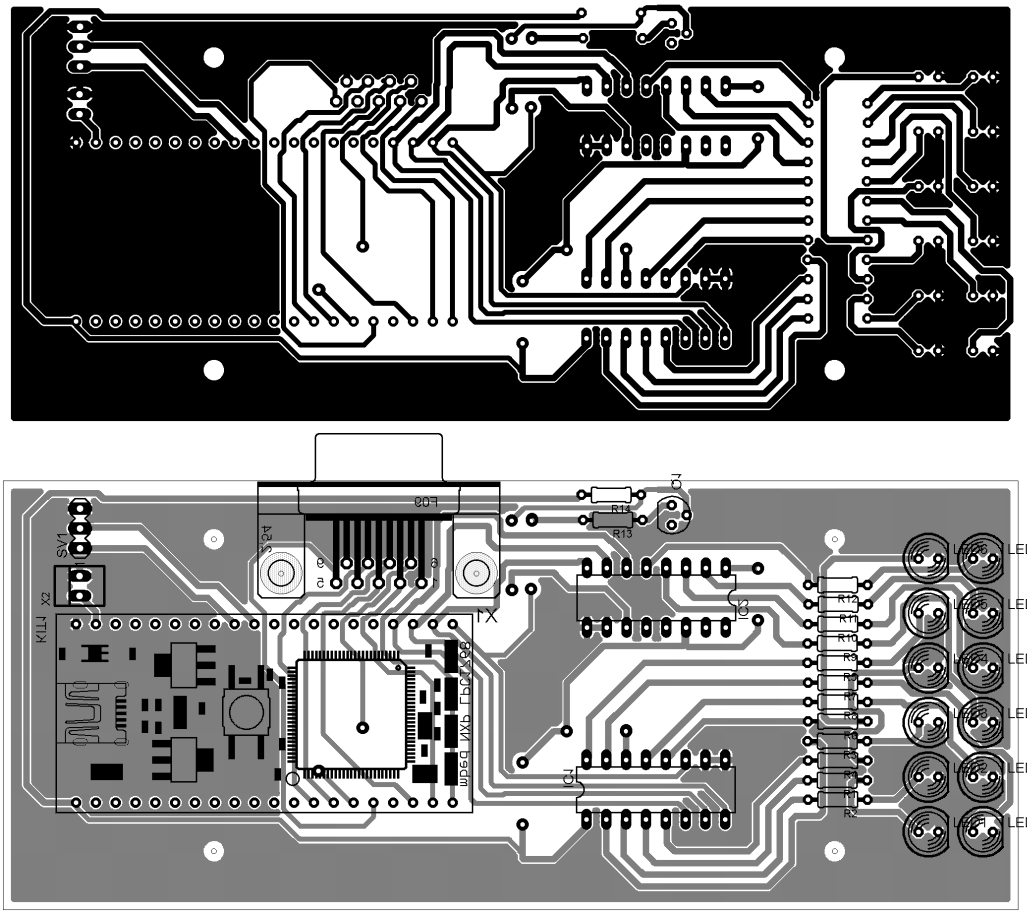
Nimi	**Komponentti**
AKKU_MIINUS	JOHDINLIITIN 2x5mm^2
AKKU_PLUS	JOHDINLIITIN 2x5mm^2
C1	(Al E-cap) 10uF/63V
C2	(Al E-cap) 10uF/63V
C3	(Al E-cap) 3,3uF/63V
C4	(Al E-cap) 3,3uF/63V
C5	(Al E-cap) 10uF/63V
C6	(Al E-cap) 3,3uF/63V
C7	(Al E-cap) 3,3uF/63V
C8	(Al E-cap) 10uF/63V
D1	1N4007
D2	1N4007
D3	1N4007
D4	1N4007
IC1	IR2111 (DIP-8)
IC2	IR2111 (DIP-8)
IC3	IR2111 (DIP-8)
IC4	IR2111 (DIP-8)
INPUT	LIITINRIMA UROS 4x2,54mm
MOOTTORI1	JOHDINLIITIN 2x5mm^2
MOOTTORI2	JOHDINLIITIN 2x5mm^2
MOOTTORI3	JOHDINLIITIN 2x5mm^2
MOOTTORI4	JOHDINLIITIN 2x5mm^2
Q1	IRF530 (TO-220)
Q2	IRF530 (TO-220)
Q3	IRF530 (TO-220)
Q4	IRF530 (TO-220)
Q5	IRF530 (TO-220)
Q6	IRF530 (TO-220)
Q7	IRF530 (TO-220)
Q8	IRF530 (TO-220)
VCC	LIITINRIMA UROS 2x2,54mm

3 RADIO-OHJAIMEN PIIRILEVYT

3.1 Piirikaavio



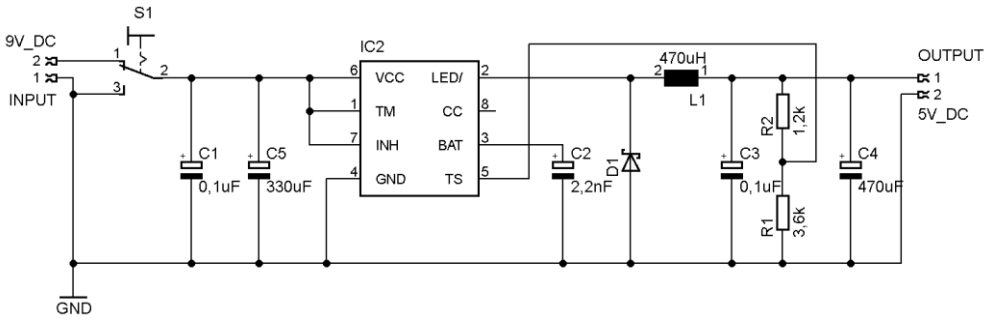
3.2 Piirilevykuvat



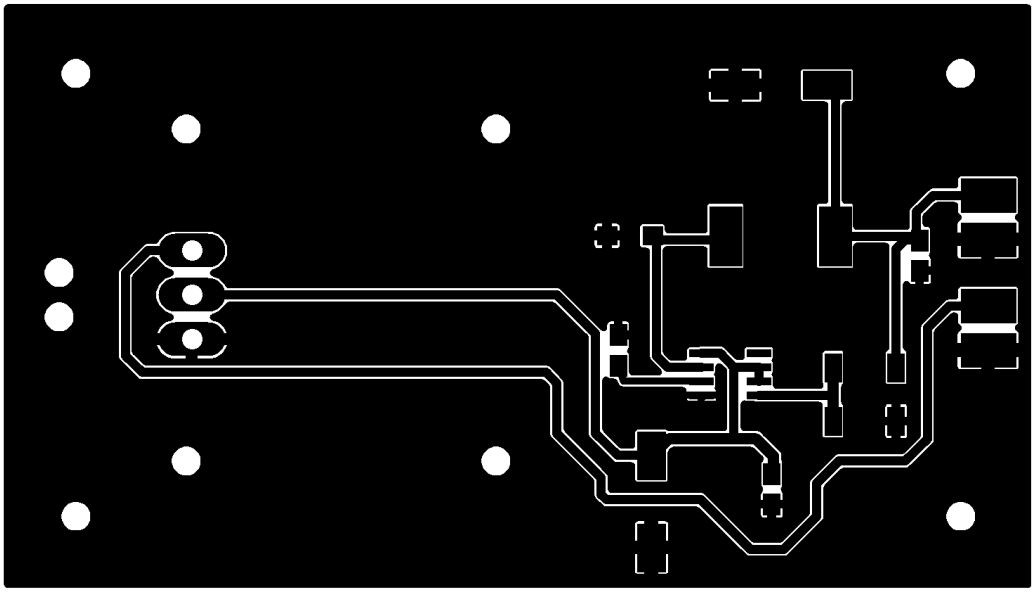
3.3 Komponenttiluettelo

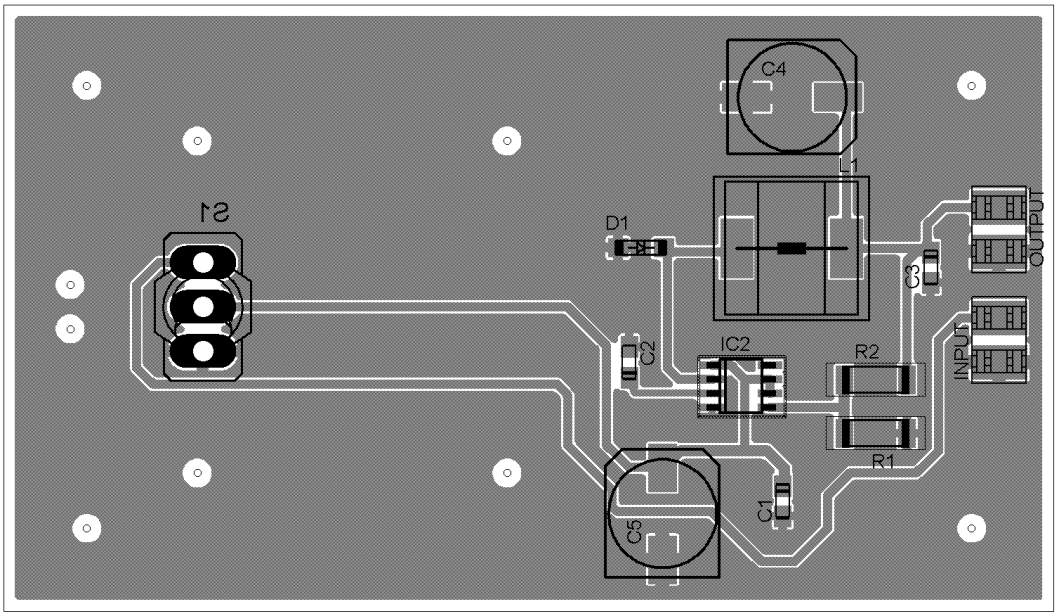
Nimi	Komponentti
IC1	74HCT4051N (DIP-16)
IC2	74HCT4051N (DIP-16)
KEYPAD	LIITIN NAARAS (DSUB9F)
KIT1	MBED NXP LPC1768
LED1	EL383VGC LED 5mm
LED2	EL383VGC LED 5mm
LED3	EL383VGC LED 5mm
LED4	EL383VGC LED 5mm
LED5	EL383VGC LED 5mm
LED6	EL383VGC LED 5mm
LED7	EL383VGC LED 5mm
LED8	EL383VGC LED 5mm
LED9	EL383VGC LED 5mm
LED10	EL383VGC LED 5mm
LED11	EL383VGC LED 5mm
LED12	EL383VGC LED 5mm
PWR_IN	LIITINRIMA UROS 2x2,54mm
Q1	BC639 (TO-92)
R1	(Metal-film) 0,22k/0,6W/1%
R2	(Metal-film) 0,22k/0,6W/1%
R3	(Metal-film) 0,22k/0,6W/1%
R4	(Metal-film) 0,22k/0,6W/1%
R5	(Metal-film) 0,22k/0,6W/1%
R6	(Metal-film) 0,22k/0,6W/1%
R7	(Metal-film) 0,22k/0,6W/1%
R8	(Metal-film) 0,22k/0,6W/1%
R9	(Metal-film) 0,22k/0,6W/1%
R10	(Metal-film) 0,22k/0,6W/1%
R11	(Metal-film) 0,22k/0,6W/1%
R12	(Metal-film) 0,22k/0,6W/1%
R13	(Metal-film) 1k/0,6W/1%
R14	(Metal-film) 10k/0,6W/1%
SERIAL	LIITINRIMA UROS 3x2,54mm

3.4 Teholähdepiirilevyn piirikaavio



3.5 Teholähdepiirilevyn piirilevykuvat





3.6 Teholähdepiirilevyn komponenttiluettelo

Nimi	**Komponentti**
C1	(Ceramic-cap) 0,1uF/25V
C2	(Ceramic-cap) 2,2nF/25V
C3	(Ceramic-cap) 0,1uF/25V
C4	(Al E-cap) 470uF/36V
C5	(Al E-cap) 330uF/36V
D1	B130-13-F 30V/1A
IC2	NCP3063 (SOIC-8)
INPUT	JOHDINLIITIN
L1	470uH/0,9A (shielded)
OUTPUT	JOHDINLIITIN
R1	(Metal-film) 3,6k/0,6W/1%
R2	(Metal-film) 1,2k/0,6W/1%
S1	KYTKIN

YMPÄRISTÖROBOTIN KOODILIITE

LIITE 2

SISÄLLYS

1 YMPÄRISTÖROBOTIN C++ LÄHDEKOODI MBED NXP LPC 1768 MIKRO-OHJAINKORTILLE

1.1 main.cpp

```
#include "mbed.h"
#include "MOTOR_CONTROL.h"
#include "SERVO_CONTROL.h"
#include "SERIAL_COMMAND.h"
#include "SENSOR.h"
#include <string>
#include <sstream>

motor_control movement(p23,p24,p25,p26); // movement control pins (pin1, pin2, pin3, pin4)
servo_control servo(p21,p22);   // servo control pins (servo1, servo2)
serial_command dataBus(p28,p27); // serial bus (tx, rx)

// sensor bus (I2C (sda, scl), analog input (gas sensor), digital output (pnp1, pnp2))
sensor sensorBus(p9,p10,p18,p20,p19);

DigitalOut led1(LED1); // signal led (led1)
Timer moveTimer;

int moveSpeed = 75; // default movement speed = 75
int turnSpeed = 50; // default turn speed = 50

int main()
{
    string commandInput;

    float humidity, temperature, gas;
    string dataOutput;

    dataBus.initSerial(); // initialize serial communication (8N1, 19200 baudrate)
    movement.initMotors(); // initialize motors (20kHz pulse frequency)
    servo.initServos(); // initialize servos (20ms pulserate)
    sensorBus.initI2C(); // initialize I2C bus

    moveTimer.start();

    while(1) {

        commandInput=dataBus.getCommand();

        if (commandInput == "NULL") {

            // invalid or empty command

        } else {

            led1 = led1 ^ 1;

            if (commandInput == "forward") {

                // move forwards
                movement.runMotorLeft(moveSpeed);
                movement.runMotorRight(moveSpeed);
                moveTimer.reset();

            } else if (commandInput == "backward") {

                // move backwards
                movement.runMotorLeft(moveSpeed * -1);
                movement.runMotorRight(moveSpeed * -1);
                moveTimer.reset();

            } else if (commandInput == "left") {
```

```
    // turn left
    movement.runMotorLeft(turnSpeed * -1);
    movement.runMotorRight(turnSpeed);
    moveTimer.reset();

} else if (commandInput == "right") {

    // turn right
    movement.runMotorLeft(turnSpeed);
    movement.runMotorRight(turnSpeed * -1);
    moveTimer.reset();

} else if (commandInput == "stop") {

    // brake
    movement.runMotorLeft(0);
    movement.runMotorRight(0);
    moveTimer.reset();

} else if (commandInput == "auto") {

    // automatic movement
    movement.runAutomatic(sensorBus.readPnp());
    moveTimer.reset();

} else if (commandInput == "horizontal") {

    // track horizontal servo
    servo.trackServo0();


} else if (commandInput == "vertical") {

    // track vertical servo
    servo.trackServo1();

} else if (commandInput == "incspeed") {

    // increase movement speed (max. 95)
    moveSpeed+=5;
    if (moveSpeed > 95)
        moveSpeed = 95;

} else if (commandInput == "decspeed") {

    // decrease movement speed (min. 50)
    moveSpeed-=5;
    if (moveSpeed < 50)
        moveSpeed = 50;

} else if (commandInput == "incturn") {

    // increase turn speed (max. 75)
    turnSpeed+=5;
    if (turnSpeed > 75)
        turnSpeed = 75;

} else if (commandInput == "decturn") {

    // decrease turn speed (min. 50)
    turnSpeed-=5;
    if (turnSpeed < 50)
        turnSpeed = 50;

} else if (commandInput == "rstspeed") {

    // reset speed to default (move 75, turn 50)
    moveSpeed = 75;
    turnSpeed = 50;

} else if (commandInput == "getdata") {

    // send sensor data
```

```
                humidity=sensorBus.readHumidity();
                temperature=sensorBus.readTemperature();
                gas=sensorBus.readGas();

                std::ostringstream oss;
                oss << temperature << " " << humidity << " " << gas;
                dataOutput = oss.str();

                dataBus.writeSerial(dataOutput);

            }

        }

        if (moveTimer.read() > 0.75) {

            // in case of no signal for 0.75s
            movement.runMotorLeft(0);
            movement.runMotorRight(0);
            moveTimer.reset();
        }
    }

}
```

1.2 MOTOR_CONTROL

1.2.1 MOTOR_CONTROL.cpp

```
#include "MOTOR_CONTROL.h"
#include "mbed.h"

/* INTERFACE BETWEEN H-BRIDGE AND MICROCONTROLLER IS INVERTING (1=0V, 0=12V)
*****************************************************************************
GND = 0V ( = 1 )             | pulse              | M1: (1) | (2) | M2: (1) | (2)
-----------------------------+--------------------+---------+-----+---------+-----
ctrl_pin1                    | ctrl_pin2          |   +     | GND |         |
-----------------------------+--------------------+---------+-----+---------+-----
ctrl_pin2                    | ctrl_pin1          |  GND    |  +  |         |
-----------------------------+--------------------+---------+-----+---------+-----
ctrl_pin3                    | ctrl_pin4          |         |     |   +     | GND
-----------------------------+--------------------+---------+-----+---------+-----
ctrl_pin4                    | ctrl_pin3          |         |     |  GND    |  +
*****************************************************************************/

motor_control::motor_control(PinName motor_p0, PinName motor_p1, PinName motor_p2, PinName
motor_p3):_P0(motor_p0),_P1(motor_p1),_P2(motor_p2),_P3(motor_p3)
{
}

void motor_control::initMotors(void)
{
    // 20kHz pulse frequency
    _P0.period(0.00005);
    _P1.period(0.00005);
    _P2.period(0.00005);
    _P3.period(0.00005);
}

// value in speed (-100 - 100) (negative speed = reverse movement)
int motor_control::runMotorLeft(int speed)
{

    int returnValue;
    float motorSpeed;

    if (speed >= -100 && speed <= 100) {
```

```
        returnValue=1;

        if (speed == 0) {
            _P0.write(1);
            _P1.write(1);
        } else if (speed > 0) {
            motorSpeed = 1 - (float(speed)/100);
            _P0.write(1);
            _P1.write(motorSpeed);
        } else if (speed < 0) {
            motorSpeed = 1 + (float(speed)/100);
            _P1.write(1);
            _P0.write(motorSpeed);
        }

    } else
        returnValue=0;

    return returnValue;
}

// value in speed (-100 - 100) (negative speed = reverse movement)
int motor_control::runMotorRight(int speed)
{

    int returnValue;
    float motorSpeed;

    if (speed >= -100 && speed <= 100) {

        returnValue=1;

        if (speed == 0) {
            _P2.write(1);
            _P3.write(1);
        } else if (speed > 0) {
            motorSpeed = 1 - (float(speed)/100);
            _P2.write(1);
            _P3.write(motorSpeed);
        } else if (speed < 0) {
            motorSpeed = 1 + (float(speed)/100);
            _P3.write(1);
            _P2.write(motorSpeed);
        }

    } else
        returnValue=0;

    return returnValue;
}

void motor_control::runAutomatic(int sensorInput)
{

    if (sensorInput == 0b00) {
        runMotorLeft(50);
        runMotorRight(50);
    } else if (sensorInput == 0b11) {
        runMotorLeft(0);
        runMotorRight(0);
    }

    if (sensorInput == 0b01 || sensorInput == 0b10) {

        if (sensorInput == 0b01) {
            runMotorLeft(-50);
            runMotorRight(50);
        } else if (sensorInput == 0b10) {
            runMotorLeft(50);
            runMotorRight(-50);
        }
    }
```

```
}
```

1.2.2 MOTOR_CONTROL.h

```
#ifndef MOTOR_CONTROL
#define MOTOR_CONTROL


#include "mbed.h"

class motor_control
{
public:

    motor_control(PinName motor_p0, PinName motor_p1, PinName motor_p2, PinName motor_p3);
    void initMotors(void);
    int runMotorLeft(int speed);
    int runMotorRight(int speed);
    void runAutomatic(int sensorInput);

protected:

    PwmOut _P0;
    PwmOut _P1;
    PwmOut _P2;
    PwmOut _P3;

};

#endif
```

1.3 SENSOR

1.3.1 SENSOR.cpp

```
#include "SENSOR.h"
#include "mbed.h"

sensor::sensor(PinName sda, PinName scl, PinName gasIn, PinName pnp0, PinName pnp1): I2C
(sda, scl), _GASIN(gasIn), _PNP0(pnp0), _PNP1(pnp1)
{
}

void sensor::initI2C(void)
{
    this->frequency(i2cFreq); // I2C frequency 200kHz
}

int sensor::measurementRequest(void)
{

    int returnValue;

    this->start();

    if (this->write(0x27<<1)) { // write device address to I2C (27h)

        this->stop();

        this->start();

        if (this->write(0x27<<1)) {
```

```
            returnValue = this->read(0x27<<1, cmd, 4);
        }
    }

    return returnValue;
}

float sensor::readHumidity(void)
{
    float humiRH;
    measurementRequest();

    // convert humidity data bits to float value
    float humiRaw = ((cmd[0] & 0x3f) << 8) | (cmd[1] & 0xff);
    humiRH = (humiRaw/16382)*100;   // convert data value to humidity (percent)

    return humiRH;
}

float sensor::readTemperature(void)
{
    float temperature;
    measurementRequest();

    // convert temperature data bits to float value
    float tempRaw = ( (cmd[2] << 8) | cmd[3] ) >> 2 ;

    // convert data value to temperature (degrees)
    temperature = ((tempRaw/16382)*165) - 40;

    return temperature;
}


int sensor::fetchDataRaw(void)
{

    int returnInt;
    measurementRequest();

    returnInt = (int)((cmd[0] << 24) | (cmd[1] << 16) | (cmd[2] << 8) | cmd[3]);
    return returnInt;
}

float sensor::readGas(void)
{

    float returnValue;
    float tempValue;

    tempValue = _GASIN.read()*3.3; // read gas-sensor input voltage

    // convert voltage to gas-sensor resistance
    tempValue = ((3.3/tempValue - 1) * 10000)/4400;
    tempValue = pow((double)tempValue,(double)2.28);

    // convert gas-sensor resistance to carbonmonoxide percentange (ppm)
    returnValue = (double)(7472773) / (tempValue * 2882 );

    return returnValue;

}

int sensor::readPnp(void)
{

    int returnValue;
    int tempValue=0b00;

    if (_PNP0 == 1) // read pnp0 value
        tempValue |= 0b01;

    if (_PNP1 == 1) // read pnp1 value
```

```
        tempValue |= 0b10;

    returnValue = tempValue;

    return returnValue;

}
```

1.3.2 SENSOR.h

```
#ifndef SENSOR
#define SENSOR


#include "mbed.h"
#define i2cFreq 200000

class sensor : public I2C{
public:

    sensor(PinName sda,PinName scl,PinName gasIn, PinName pnp0, PinName pnp1);
    void initI2C(void);
    float readTemperature(void);
    float readHumidity(void);
    int fetchDataRaw(void);
    char cmd[4];
    int measurementRequest(void);
    float readGas(void);
    int readPnp(void);

protected:

    AnalogIn _GASIN;
    DigitalIn _PNP0, _PNP1;
    int dataFetch();
};

#endif
```

1.4 SERIAL_COMMAND

1.4.1 SERIAL_COMMAND.cpp

```
#include "SERIAL_COMMAND.h"
#include "mbed.h"
#include <string>
#include <sstream>

serial_command::serial_command(PinName tx, PinName rx): Serial (tx, rx) {
}

void serial_command::initSerial(void){
    this->baud(19200); // 19200
    this->format(8,SerialBase::None,1) ; // 8N1
    }

string serial_command::readSerial(void){

    char tempString [15];
    string returnValue;

    if (this->readable()) {

        this->gets(tempString,15); // read string from serial bus
```

```
        returnValue=string(tempString);

    } else
        returnValue="";

    return returnValue;
}

string serial_command::getCommand(void){

    string returnValue;
    string tempString;

    tempString=readSerial();

    // convert serial string to command

    if (tempString == "") {
        returnValue = "NULL";
    } else {
        if (tempString.find("forward") != tempString.npos)
            returnValue = "forward";
        else if (tempString.find("backward") != tempString.npos)
            returnValue = "backward";
        else if (tempString.find("left") != tempString.npos)
            returnValue = "left";
        else if (tempString.find("right") != tempString.npos)
            returnValue = "right";
        else if (tempString.find("stop") != tempString.npos)
            returnValue = "stop";
        else if (tempString.find("auto") != tempString.npos)
            returnValue = "auto";
        else if (tempString.find("horizontal") != tempString.npos)
            returnValue = "horizontal";
        else if (tempString.find("vertical") != tempString.npos)
            returnValue = "vertical";
        else if (tempString.find("incspeed") != tempString.npos)
            returnValue = "incspeed";
        else if (tempString.find("decspeed") != tempString.npos)
            returnValue = "decspeed";
        else if (tempString.find("incturn") != tempString.npos)
            returnValue = "incturn";
        else if (tempString.find("decturn") != tempString.npos)
            returnValue = "decturn";
        else if (tempString.find("rstspeed") != tempString.npos)
            returnValue = "rstspeed";
        else if (tempString.find("getdata") != tempString.npos)
            returnValue = "getdata";
        else
            returnValue="NULL";
    }
    return returnValue;
}

int serial_command::writeSerial(string serialOutput)
{

    int returnValue;

    if (this->writeable()) {
        this->puts(serialOutput.c_str()); // write data to serial bus
        returnValue = 1;
    } else
        returnValue = 0;

    return returnValue;
}
```

1.4.2 SERIAL_COMMAND.h

```
#ifndef SERIAL_COMMAND
#define SERIAL_COMMAND


#include "mbed.h"
#include <string>

class serial_command : public Serial {
public:

    serial_command(PinName tx, PinName rx);
    void initSerial(void);
    string getCommand(void);
    string readSerial(void);
    int writeSerial(string serialOutput);

};

#endif
```

1.5 SERVO_CONTROL

1.5.1 SERVO_CONTROL.cpp

```
#include "SERVO_CONTROL.h"
#include "mbed.h"

servo_control::servo_control(PinName servo_0, PinName
servo_1):_SERVO_0(servo_0),_SERVO_1(servo_1){
}

// default servo values
int servo1Degrees = 179,
    servo2Degrees = 75;
int servo1Direction = 0,
    servo2Direction = 0;

void servo_control::initServos(void)
{
    _SERVO_0.period(0.020); // 20ms pulserate
    _SERVO_1.period(0.020);
    positionServo0(servo1Degrees);
    positionServo1(servo2Degrees);
}

int servo_control::positionServo0(int degrees) // value in degrees (0 - 180)
{

    int returnValue;
    float servoPosition;

    if (degrees >= 0 && degrees <= 180) {

        returnValue=1;

        // convert servo degrees to pulsewidth (0.75-2.25ms)
        servoPosition = float(( 75 + ((0.8)*float(degrees)) ) / 100000);

        _SERVO_0.pulsewidth(servoPosition);

    } else
        returnValue=0;

    return returnValue;
```

```
}

int servo_control::positionServo1(int degrees) // value in degrees (0 - 180)
{

    int returnValue;
    float servoPosition;

    if (degrees >= 0 && degrees <= 180) {

        returnValue=1;

        // convert servo degrees to pulsewidth (0.75-2.25ms)
        servoPosition = float(( 75 + ((0.8)*float(degrees)) ) / 100000);

        _SERVO_1.pulsewidth(servoPosition);

    } else
        returnValue=0;

    return returnValue;
}

void servo_control::trackServo0(void) // servo tracking function
{
    servoTimer.start();

    if (servo1Degrees < 180 && servo1Direction == 0 && servoTimer.read() > 0.01) {
        positionServo0(servo1Degrees);
        servoTimer.reset();
        servo1Degrees++;
        if (servo1Degrees >= 180)
            servo1Direction=1;
    } else if (servo1Degrees > 0 && servo1Direction == 1 && servoTimer.read() > 0.01) {
        positionServo0(servo1Degrees);
        servoTimer.reset();
        servo1Degrees--;
        if (servo1Degrees <= 0)
            servo1Direction=0;
    }

}

void servo_control::trackServo1(void) // servo tracking function
{

    servoTimer.start();

    if (servo2Degrees < 180 && servo2Direction == 0 && servoTimer.read() > 0.01) {
        positionServo1(servo2Degrees);
        servoTimer.reset();
        servo2Degrees++;
        if (servo2Degrees >= 180)
            servo2Direction=1;
    } else if (servo2Degrees > 0 && servo2Direction == 1 && servoTimer.read() > 0.01) {
        positionServo1(servo2Degrees);
        servoTimer.reset();
        servo2Degrees--;
        if (servo2Degrees <= 0)
            servo2Direction=0;
    }
}
```

1.5.2 SERVO_CONTROL.h

```
#ifndef SERVO_CONTROL
#define SERVO_CONTROL


#include "mbed.h"
```

```
class servo_control {
public:

    Timer servoTimer;
    servo_control(PinName servo_0, PinName servo_1);
    void initServos();
    int positionServo0(int degrees);
    int positionServo1(int degrees);
    void trackServo0(void);
    void trackServo1(void);

protected:

    PwmOut _SERVO_0;
    PwmOut _SERVO_1;
};

#endif
```

2 YMPÄRISTÖROBOTIN RADIO-OHJAIMEN C++ LÄHDEKOODI MBED NXP LPC 1768 MIKRO-OHJAINKORTILLE

2.1 main.cpp

```
#include "mbed.h"
#include "SERIAL_COMMAND.h"
#include "KEYPAD.h"
#include "MODE_CONTROL.h"
#include "LED_CONTROL.h"
#include <string>

led_control ledBus(p17, p18, p19, p20);
Keypad keypad(p21,p22,p23,p24,p25,p26,p27,p28);
mode_control modeSwap;

int mode = 1;
int autoMovement = 0;

int main()
{
    ledBus.initLeds();
    modeSwap.initMode();

    char key;
    int state;
    int temp;
    string commandOutput;

    while(1) {

        key = keypad.getKey();

        state=key;

        if (state != temp) {
            if (state == 12 && autoMovement == 0) {
                mode=modeSwap.useMode(mode, key, 1);
                ledBus.lightLed(key);
            } else if (state == 10) {
                autoMovement=autoMovement^1;
            } else if (mode == 2 && autoMovement == 0) {
                modeSwap.useMode(mode, key, 0);
                ledBus.lightLed(key);
            }

            temp = state;
        } else if (mode == 2 && autoMovement == 0) {
            ledBus.lightLed(13);
        }

        if (key != KEY_RELEASED) {

            if (mode == 1 && autoMovement == 0) {
                modeSwap.useMode(mode, key, 0);
                ledBus.lightLed(key);
            }


        } else {
            if (autoMovement == 1) {
                modeSwap.useMode(mode, 20, 0);
                ledBus.knightRider();
            } else if (mode == 1) {
                modeSwap.useMode(mode, 5, 0);
                ledBus.lightLed(13);
            }
        }
```

```
        wait_ms(10);

    }
}
```

2.2 KEYPAD

2.2.1 KEYPAD.cpp

```
/*              Copyright (c) 2010 Dimiter Kentri

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.

*/

#include "mbed.h"
#include "KEYPAD.h"

using namespace mbed;

Keypad::Keypad(PinName col1, PinName col2, PinName col3, PinName col4, PinName
row1,PinName row2, PinName row3, PinName row4): _col1(col1),
    _col2(col2),_col3(col3),_col4(col4),_rows(row1,row2,row3,row4)    {

}

int Keypad::getKeyIndex(){
    for(int k=0; k<4; k++) {
            _rows = 0;
    }
    for(int i=0; i<4; i++) {
            _rows = 1 << i;
        for(int j=0; j<4;j++){
            if (_col1)
                return 1+(i*4);
            else if (_col2)
                return 2+(i*4);
               else if (_col3)
                   return 3+(i*4);
            else if (_col4)
                return 4+(i*4);
        }
    }
    return -1;
}

char Keypad::getKey()
{
    int k = getKeyIndex();
    if(k != -1)
        return  keys[k-1];
```

```
    else
        return 0;
}
```

2.2.2 KEYPAD.h

```
/*             Copyright (c) 2010 Dimiter Kentri

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.

*/

#ifndef KEYPAD_H
#define KEYPAD_H

#include "DigitalIn.h"
#include "BusOut.h"


namespace mbed{

const char NO_KEY = '\0';
#define KEY_RELEASED NO_KEY



const int keys[16] = {1,2,3,'A',
                      4,5,6,'B',
                      7,8,9,'C',
                      10,11,12,'D'};

class Keypad{
    public:
        Keypad(PinName col1, PinName col2, PinName col3, PinName col4, PinName
row1,PinName row2, PinName row3, PinName row4);
        int getKeyIndex();
        char getKey();

    protected:
        DigitalIn _col1,_col2,_col3,_col4;
        BusOut _rows;
};

}
#endif
```

2.3 LED_CONTROL

2.3.1 LED_CONTROL.cpp

```
#include "LED_CONTROL.h"
#include "mbed.h"

Timer ledTimer;

led_control::led_control(PinName pin1, PinName pin2, PinName pin3, PinName pin4): BusOut
(pin1, pin2, pin3, pin4)
{
}

int ledMask = 1;
int direction;

void led_control::initLeds(void)
{
    ledTimer.start();

    for (int ii=0; ii<3; ii++) {
        while(ledTimer < 0.2) {
            for (int i = 0; i<12; i++) {
                this->write(i);
            }

        }
        this->write(13);
        wait(0.2);
        ledTimer.reset();
    }

    ledTimer.stop();
}


void led_control::lightLed(int ledNumber)
{
    this->write(ledNumber-1);
}

void led_control::knightRider(void)
{

    ledTimer.start();

    if (ledMask == 1) {
        direction = 0;
    } else if (ledMask == 6) {
        direction = 1;
    }
    if (direction == 0 && ledTimer > 0.05) {
        ledTimer.reset();
        lightLed(ledMask);
        ledMask++;
    } else if (direction == 1 && ledTimer > 0.05) {
        ledTimer.reset();
        lightLed(ledMask);
        ledMask--;
    }
}
```

2.3.2 LED_CONTROL.h

```
#ifndef LED_CONTROL
```

```
#define LED_CONTROL

#include "mbed.h"

class led_control: public BusOut
{
public:

    led_control(PinName pin1, PinName pin2, PinName pin3, PinName pin4);
    void knightRider(void);
    void initLeds(void);
    void lightLed(int ledNumber);

};

#endif
```

2.4 SERIAL_COMMAND

2.4.1 SERIAL_COMMAND.cpp

```
#include "SERIAL_COMMAND.h"
#include "mbed.h"
#include <string>

serial_command::serial_command(PinName tx, PinName rx): Serial (tx, rx) {
}

void serial_command::initSerial(void){
    this->baud(19200); // Serial bus baud rate
    this->format(8,SerialBase::None,1) ; // 8 bits, parity, 1 stop bits
    }

int serial_command::writeSerial(string serialOutput)
{

    int returnValue;

    if (this->writeable()) {
        this->puts(serialOutput.c_str());
        returnValue = 1;
    } else
        returnValue = 0;

    return returnValue;
}
```

2.4.2 SERIAL_COMMAND.h

```
#ifndef SERIAL_COMMAND
#define SERIAL_COMMAND

#include "mbed.h"
#include <string>

class serial_command : public Serial {
public:

    serial_command(PinName tx, PinName rx);
    void initSerial(void);
    int writeSerial(string serialOutput);

};
```

```
#endif
```

2.5 MODE_CONTROL

2.5.1 MODE_CONTROL.cpp

```
#include "MODE_CONTROL.h"
#include "SERIAL_COMMAND.h"
#include "mbed.h"
#include <string>

serial_command serialBus(p9, p10);

void mode_control::initMode(void)
{
    serialBus.initSerial();
}

int mode_control::useMode(int usingMode, int inputKey, int flag)
{

    int returnValue;
    int tempValue;

    if (usingMode == 1 && flag == 0)
        movementMode(inputKey);
    else if (usingMode == 2 && flag == 0)
        speedMode(inputKey);
    else if (flag == 1) {
        tempValue=specialMode(inputKey);

        if (tempValue == 1) {
            if (usingMode == 1)
                returnValue = 2;
            else if (usingMode == 2)
                returnValue = 1;
        }

    }


    return returnValue;

}

void mode_control::movementMode(int inputKey)
{

    switch(inputKey) {
        case 2:
            // move forwards
            serialBus.writeSerial("forward");
            break;
        case 4:
            // move left
            serialBus.writeSerial("left");
            break;
        case 6:
            // move right
            serialBus.writeSerial("right");
            break;
        case 8:
            // move backwards
            serialBus.writeSerial("backward");
            break;
        case 5:
            // brake
```

```
            serialBus.writeSerial("stop");
            break;
        case 1:
            // move horizontal servo
            serialBus.writeSerial("horizontal");
            break;
        case 3:
            // move vertical servo
            serialBus.writeSerial("vertical");
            break;
        case 20:
            // automatic movement
            serialBus.writeSerial("auto");
            break;
    }
}

void mode_control::speedMode(int inputKey)
{

    switch(inputKey) {

        case 2:
            // increase movement speed
            serialBus.writeSerial("incspeed");
            break;
        case 4:
            // decrease turn speed
            serialBus.writeSerial("decturn");
            break;
        case 6:
            // increase turn speed
            serialBus.writeSerial("incturn");
            break;
        case 8:
            // decrease movement speed
            serialBus.writeSerial("decspeed");
            break;
        case 5:
            // reset speed
            serialBus.writeSerial("rstspeed");
            break;

    }

}

int mode_control::specialMode(int inputKey)
{

    int returnValue = 0;

    switch(inputKey) {

        case 10:
            // automatic movement on
            // increase movement speed
            // serialBus.writeSerial("auto");
            break;
        case 12:
            // switch mode
            returnValue = 1;
            break;
    }

    return returnValue;
}
```

2.5.2 MODE_CONTROL.h

```
#ifndef MODE_CONTROL
#define MODE_CONTROL

#include "mbed.h"
#include <string>

class mode_control{
public:

    int useMode(int usingMode, int inputKey, int flag);
    void initMode(void);
    void movementMode(int inputKey);
    void speedMode(int inputKey);
    int specialMode(int inputKey);

};

#endif
```

3 YMPÄRISTÖROBOTIN PYTHON LÄHDEKOODI RASPBERRY PI 2 KORTILLE

3.1 event_handler.py

```
import select
import pty
import os
import fcntl
import tty
import termios
import serial
import time
import sys
import subprocess

def __select( iwtd, owtd, ewtd, timeout=None):

   if timeout is not None:
       end_time = time.time() + timeout
   while True:
       try:
           return select.select(iwtd, owtd, ewtd, timeout)
       except select.error:
           err = sys.exc_info()[1]
           if err.args[0] == errno.EINTR:

               if timeout is not None:
                   timeout = end_time - time.time()
                   if timeout < 0:
                       return([], [], [])
           else:

               raise

STDIN_FILENO=pty.STDIN_FILENO
STDOUT_FILENO=pty.STDOUT_FILENO
string_type=bytes
sys.stdout.write(string_type())
sys.stdout.flush()
buffer = string_type()
mode = tty.tcgetattr(STDIN_FILENO)
tty.setraw(STDIN_FILENO)

try:
               port = serial.Serial("/dev/ttyAMA0", baudrate=19200,
                              parity=serial.PARITY_NONE,
                              stopbits=serial.STOPBITS_ONE,
                              bytesize=serial.EIGHTBITS,
                              timeout=0.1)

               read = 0
               auto = 0
               localtime = 0
               timer0 = 0
               ascii_value = 0

               while True:

                              timer0=timer+1

                              if timer0 > 10:
                                             localtime =
                                             time.asctime(time.localtime(time.time()))
                                             port.write("getdata");
                                             read = port.read(11)
                                             port.flushInput()
                              if len(read) != 0 or read == "-40.00 0.00 0":
```

```
                                        with open ('testfile.txt', 'a') as f:
                                                f.write("%s %s\n" % (read,
                                                localtime))
                                        timer0 = 0

                                r, w, e = __select([STDIN_FILENO], [], [],timeout=0.001)
                                if STDIN_FILENO in r:

                                        data=os.read(STDIN_FILENO, 1)
                                        ascii_value=ord(data[0])

                                if ascii_value == 3:
                                        break
                                if ascii_value == 65:
                                        port.write("forward")
                                if ascii_value == 68:
                                        port.write("left")
                                if ascii_value == 67:
                                        port.write("right")
                                if ascii_value == 66:
                                        port.write("backward")
                                if ascii_value == 71:
                                        port.write("stop")
                                if ascii_value == 97:
                                        port.write("auto")
                                if auto == 0:
                                        print ("auto on")
                                        auto = 1
                                elif auto == 1:
                                        print ("auto off")
                                        auto = 0
                                if ascii_value == 45:
                                        port.write("horizontal")
                                if ascii_value == 43:
                                        port.write("vertical")

finally:
                tty.tcsetattr(STDIN_FILENO, tty.TCSAFLUSH, mode)
```