OULUN AMMATTIKORKEAKOULU

Jussi Pulkkinen

MOBIILIROBOTIN KOORDINAATTIOHJAUS

MOBIILIROBOTIN KOORDINAATTIOHJAUS

Jussi Pulkkinen
Opinnäytetyö
Syksy 2016
Automaatiotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Automaatiotekniikka, tuotantopainotteinen suuntautuminen

Tekijä(t): Jussi Pulkkinen
Opinnäytetyön nimi: Mobiilirobotin koordinaattiohjaus
Työn ohjaaja(t): Tero Hietanen, Tapio Heikkilä
Työn valmistumislukukausi ja -vuosi: Syksy 2016
Sivumäärä: 35 + 1 liitettä

Tämän työn tavoitteena oli luoda toimiva koordinaattiohjaus Probot Oy:n valmistamalle mobiilirobottialustalle Oulun VTT:llä. Ohjauksen koodi toteutettiin käyttäen Qt Creator -ohjelmaa. Ohjelmointikieleksi valittiin C++. Työ oli osa neljän kuukauden mittaista harjoittelua VTT:llä. Lopputuloksena saavutettu ohjelma ajaa robottia haluttuun koordinaattipisteeseen suurpiirteisellä tarkkuudella, ja tuotetta voidaan kehittää pidemmälle lisäämällä anturointia. Keskeisenä osana työtä oli ajettavan trajektorin muodon määrittäminen alkupisteen ja määränpään välille.

Asiasanat: ohjelmointi, robotiikka, trajektorilaskenta

ABSTRACT

Oulu University of Applied Sciences
Automation engineering

Author(s): Jussi Pulkkinen
Title of thesis: Coordinatedrive for a mobile robot
Supervisor(s): Tero Hietanen, Tapio Heikkilä
Term and year when the thesis was submitted: Fall 2016
Pages: 35 + 1 appendices

The goal of this bachelor's thesis was to create a trajectory planner and control code for a mobile robot unit using C++ on Qt Creator. The finished program will take destination coordinates (x, y) and the angle in which the robot should be at the destination as input data and drive the robot to this destination on a calculated trajectory. The thesis was a part of an internship at VTT Oulu.
The finished program's drive accuracy does not have to be very high, since no external sensoring is used in this first stage of the prototype.

Keywords: programming, robotics, trajectory planning

ALKULAUSE

Tämä työ tarjosi minulle sopivasti haasteita muun muassa uuden ohjelmointikielen opettelun kanssa, mutta mielenkiintoinen aihe ja hyvä työilmapiiri motivoivat minua eteenpäin tahdilla, johon olen itse (ja toivottavasti on työnantajakin) hyvin tyytyväinen.

Suuret kiitokset VTT:llä kaikille, jotka minua työssä avustivat. Lisäksi haluan kiittää työni valvojia: VTT:n johtava tutkija Tapio Heikkilää sekä Oamkin tutkintovastaava Tero Hietasta, jonka antaman vinkin ansiosta tämän työn sain.

Suurin kiitos kuuluu vaimolleni Minnalle, joka jaksoi vaikuttaa kiinnostuneelta aina, kun jauhoin hänelle robotiikasta ja trajektorimuodoista.

Oulussa 2.11.2016

Jussi Pulkkinen

SISÄLLYS

SANASTO

BeagleBone	RaspBerry Pin kaltainen, hyvin pieni tietokone.
C++	Yleisesti käytössä oleva, oliopohjainen ohjelmointikieli.
CAN	Ajoneuvotekniikassa runsaasti käytetty väyläratkaisu.
Qt Creator	TrollTechin kehittämä ohjelmointityökalu. Alkuosa lausutaan kuten englanninkielen sana ”cute”.
Trajektori	Termi, jota käytetään kuvaamaan robotille laskettua liikerataa.

1 JOHDANTO

Tämä opinnäytetyö suoritettiin Oulun VTT:lle. Työn tavoitteena oli saada Probot Oy:ltä hankittuun mobiilirobottialustaan koordinaattiohjaus. Tehtävän lähtökohtana oli mobiilirobottialusta, jonka ohjaus tapahtui käyttäen BeagleBone Black -tietokoneeseen USB-johdolla yhdistettyä PlayStation 3 -ohjainta.

Peliohjaimelta luettiin sen vasemman joystick-tatin asentoa, josta otettiin pysty-akselilta nopeuden arvo ja vaaka-akselilta kääntymisarvo. Tavoitteena oli luoda ohjaus, jossa ei käytetä kyseistä pelikonsolin ohjainta, vaan liitetään BeagleBoneen kannettava tietokone. Tässä tietokoneessa suorittuva ohjelma ottaa käyttäjältä input-tietona määränpääpisteen koordinaatit sekä kulman, johon robotin halutaan päätyvän. Ohjelman tulee laskea koordinaattien ja kulman mukaan reitti alku- ja loppupisteen välille.

Ohjelman koodaaminen toteutettiin käyttäen TrollTechin kehittämää Qt Creator -ohjelmistoa ja ohjelmointikielenä käytetään C++:aa.

2 OHJELMOINNIN JA VÄYLIEN TEORIAA

2.1 C++

C++ on oliopohjainen ohjelmointikieli, joka sai alkunsa kun Bell Labsilla työskennellyt Bjarne Stroustrup laajensi perinteistä C-kieltä oliopohjaiseksi. C++ ohjelmointikielenä sai suosiota 90-luvun aikana, ja se standardisoitiin vuonna 1999. Koska C++ on laajennettu perinteisestä C-kielestä, on sen komennoissa ja syntaksissa paljon tuttua aiemmin C-kieltä käyttäneille. Oliopohjaisen ohjelmoinnin ero perinteiseen ohjelmointiin on se, että ohjelma jaetaan olioihin. Ohjelman toiminta perustuu eri olioiden keskinäisiin vuorovaikutuksiin. Perinteisessäkin ohjelmoinnissa ohjelma jaetaan osiin (funktioihin), mutta tämä jako tapahtuu alemmalla kerroksella. Oliopohjaista ohjelmointia suositaan muun muassa reaaliaikajärjestelmien sekä oliopohjaisten tietokantojen ohjelmoinnissa. (1, s. 26–32.)

Qt Creator on Nokian omistama ohjelmistokehitysympäristö. Omistajuus siirtyi Nokialle vuonna 2008, kun Nokia osti Qt Creatorin kehittäjän, Trolltechin. Qt Creator on todella hyvin yhteensopiva eri alustojen ja käyttöjärjestelmien välillä. Tämä yhteensopivuus ja mahdollisuus siirtää ohjelmia alustalta toiseen onkin ollut yksi keskeisistä ajatuksista ohjelman kehityksessä. (2, s. 6–7.)

Qt Creatorin tarjoama dokumentaatio ohjelman ominaisuuksista ja sen sisältämistä valmiista luokista oli todella suuri apu tätä työtä tehdessä ja C++:lla ohjelmointia opetellessa.

2.2 CAN-väylä

Pidän CAN-väylää käsittelevän osion lyhyenä, sillä se ei ole tämän opinnäytetyön osalta keskeinen asia, koska CAN-väyläliikenteestä tässä robotissa huolehtii Probot Oy:n valmiiksi BeagleBonelle ohjelmoimat ajurit.

Controller Area Network eli CAN on varsinkin ajoneuvotekniikassa usein käytetty väyläratkaisu. Alunperin CAN-väylä suunniteltiin juurikin autojen järjestelmiin, jotka vaativat reaaliaikaista tiedonsiirtoa. Nykyisin sitä käytetään yleisesti myös ajoneuvotekniikan ulkopuolella anturi-, mittaus- ja ohjaustiedon siirtämiseen. Parhaiten CAN toimii kun siirrettävät viestit ovat pieniä, kuten yksittäinen anturitieto. Vaikka CAN-väylässä käytetty johdin onkin rakenteeltaan yksinkertainen parikaapeli, ei viesti ole aivan niin helposti luettavissa kuin sellaisissa ratkaisuissa, joissa on/off-viesti kerrotaan joko päästämällä virta läpi tai estämällä se. CAN-viesti koostuu varsinaisen anturi- tai ohjaustiedon lisäksi myös tunnisteosasta sekä osasta, joka kertoo käytettävien tietotavujen määrän kyseisessä viestissä. Koska tunnisteosa erottelee viestit, voidaan samaa kaapelia pitkin lähettää useamman laitteen viestejä. Kohtia, joissa väylään on liitetty laite, kutsutaan solmuiksi ja jokaisella solmulla on oma tunnisteensa. Tämä rakenne vähentää todella huomattavasti tarvittavan johdotuksen määrää. (3, s. 1–4.)

3 MOBIILIROBOTIIKKA JA PAIKANNUS

Mobiilirobotti on erään määritelmän mukaan määritelty koneeksi, joka täyttää kolme vaatimusta. Ensinnäkin koneella tulee olla liikkuvuus sille määritetyllä toiminta-alueella. Toisekseen vaaditun inhimillisen käyttöasteen tulee olla riittävän pieni (eli ollakseen mobiilirobotti tulee laitteen hallita riittävä autonomian määrä). Viimeisenä vaatimuksena on, että laitteella tulee olla jonkinasteinen havainnointikyky (englanninkielinen termi perception) ympäristöstään ja siellä olevista esteistä. Havainnointikyvyn ja riittävän autonomian yhdistelmä mahdollistaa itsenäisesti trajektorin laskemisen ja päivittämisen senhetkisen ajoympäristön mukaan. (4, s. 3.)

Tämän määritelmän valossa on kyseenalaista, voidaanko tässä työssä käytettävää robottia kutsua mobiilirobotiksi, sillä ennen anturointia laite vaatii ihmisen syöttämät koordinaatit eikä havainnointikykyä ole vielä laisinkaan. Siten voidaan sanoa että tässä työssä luodaan pohja, jota voi käyttää oikean, havainnoivan mobiilirobotin luomiseen. Toisaalta termi mobiilirobotti itsessään tarkoittaa sanatarkasti liikkuvaa robottia. Tämä suppeampi määrittely täyttyy myös tämän opinnäytetyön mukaisessa laitteessa.

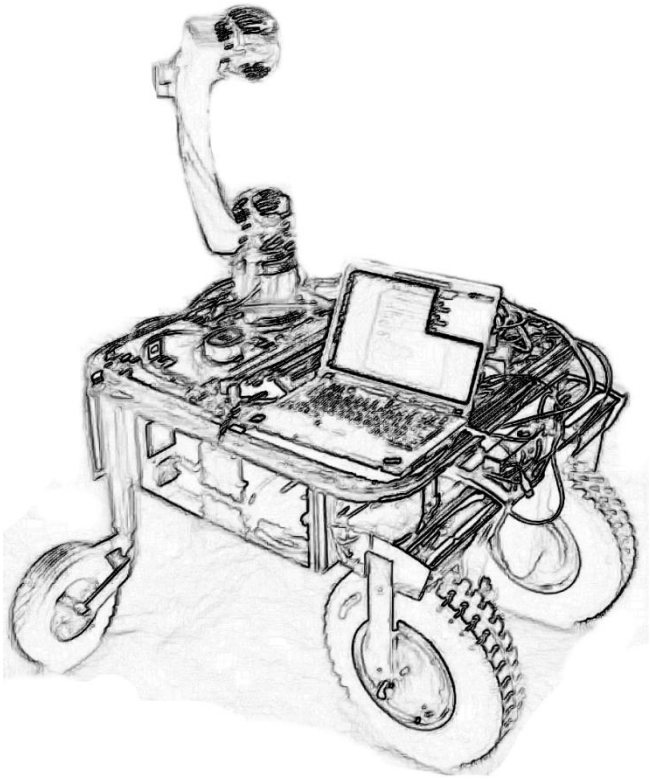
Mobiilirobottien liikkumista kehitettäessä on katsottu paljon luontoon ja tarkemmin siihen, kuinka eläimet liikkuvat. Tähän mennessä on jo kehitetty robotteja, jotka tavoittelevat liikkeillään kävelemistä, hyppimistä ja lentämistä. Varsinkin tasaisella maalla hyvin tehokas tapa on kuitenkin pyörävetoinen liikkuminen, joka ei suoraan ole saanut inspiraatiotaan luonnollisista liikkumismenetelmistä. Pyörien käyttö on myös mekaanisesti hyvin paljon yksinkertaisempaa kuin moninivelisten jalkojen ja askelluksen toteuttaminen keinotekoisesti. (5, s. 13–15.)

3.1 Probotin mobiilirobottialusta

Tässä projektissa käytettävä mobiilirobottialusta liikkuu käyttäen kahta oululaisen Probot Oy:n kehittämää MobilityModule™ 400 -yksikköä, ja lukeutuu näin ollen differentiaaliajorobotteihin. Käytännössä tämä tarkoittaa sitä, että robottia

ohjataan muuttamalla kahden vetävän renkaan pyörimisnopeuksien suhdetta. Alusta on hankittu VTT:lle kehitettäväksi prototyypiksi. MobilityModule-yksikkö sisältää alustan kasteripyöriä suuremman, moottoroidun renkaan, sisäänrakennetun mooottorinohjaimen sekä rungon sisällä kulkevan johdotuksen. Alustaan on kiinnitetty BeagleBone Black -mallinen tietokone, joka hoitaa CAN-rajapintaa ja jossa pyörivät robotin mukana tulleet moottorinohjausohjelmistot. Beagle Bonelta ohjausviesti moottoreille kulkee CAN-väylää pitkin molemmille moottoreille. CAN-väylällä kulkevan viestiliikenteen on koodannut Probot Oy.

Itse alusta koostuu kahden vetävän MobilityModule-yksikön lisäksi rungosta, kahdesta vapaasti pyörivästä renkaasta sekä pienestä, kuuden vapausasteen robottikourasta. Robottikoura ja sen ohjaus ei liity tähän opinnäytetyöhön, joten en paneudu niiden toimintaan tarkemmin. Mobiilirobottialustan rakenne työn alussa sekä Probotin MobilityModule-yksiköt näkyvät kuvan 1 oikeassa alalaidassa.



KUVA 1. Robottialustan alkuperäinen rakenne

Rakennetta muutettiin kuvan 1 tilanteesta myöhemmin niin, että robottitarttuja siirrettiin toiseen päähän alustaa painojakauman parantamiseksi.

3.2 Ajotarkkuuden ja paikannuksen ongelmia

Mobiilirobotiikka ja siinä käytettävä anturointi ovat olleet jo vuosikymmenien ajan hyvin paljon tutkittuja aloja. Luotettavan sijaintitiedon saaminen on edelleen vailla ongelmatonta ratkaisua. Sisätiloissa paikannukseen käytetään usein SLAM-menetelmiä (Simultaneous Localisation And Mapping). Tässä menetelmässä käytetään esimerkiksi laseranturia, joka pyyhkäisee pistepilven ympäriltään ja muodostaa näin kaksiulotteisen kartan ympäröivästä alueestaan. Tämä on hyvin toimiva ratkaisu esteiden havaitsemiseen ja tunnetulla alueella ajamiseen. Mikäli ajoympäristö on tiedossa, voidaan robotin sijaintia määrittää vertaamalla anturin muodostamaa karttaa jo tiedossa olevaan alueen karttaan. Tämä luonnollisesti ei onnistu ajettaessa joko avarissa ulkotiloissa tai tilassa, jonka pohjapiirrosta ei tunneta. (6, s. 1.)

Sisätiloissa robotin paikannukseen on useita muitakin menetelmiä laseranturin lisäksi. Vaihtoehtoja on yksinkertaisten videokamerapaikannuksen ja viivanseurannan lisäksi esimerkiksi akustinen mittaus sekä menetelmä, joka perustuu langattoman verkon signaalin vahvuuden tarkkailuun. Usein anturointitavasta riippumatta käytetään kiinteällä sijainnilla olevia välimajakoita, joilla pyritään estämään virheen kasautumista (toisin sanoen driftaamista). (7, s.3–7.)

Nykytekniikalla GPS-paikannus voidaan saada jopa senttimetrin tarkkuusluokassa. GPS-paikannus kuitenkin toimii luotettavasti vain ulkotiloissa, riittävän selkeällä säällä, sekä silloin kun riittävä määrä satelliitteja on oikeassa asennossa. Nämä rajoitteet ovat helposti havaittavissa esimerkiksi puhelimen GPS-paikannusta käytettäessä. Sisätiloihin siirryttäessä signaali lähes poikkeuksetta vääristyy tai häviää kokonaan. Esimerkiksi sisätiloissa kappaletavaran kuljettamiseen ja poimimiseen tämä ratkaisu ei todellakaan toimi.

Nämä satelliittipaikannuksen rajoittavat tekijät johtuvat siitä, että signaali, jota satelliitit lähettävät, on elektromagneettinen radiosignaali. Tällaisen signaalin

läpäisykyky on varsin rajoitettu, joten maapallon toisella puolella sijaitsevan satelliitin signaalia ei pystytä käyttämään. Useimmiten kuitenkin satelliitteja on riittävästi oikealla alueella. GPS-paikannuksessa lasketaan signaalin matkan kestoa satelliitilta takaisin laitteelle. Tämän vuoksi väliaineet, jotka muuttavat signaalin kulkunopeutta (tai jopa estävät signaalin kulun kokonaisuudessaan), vaikeuttavat tarkan sijainnin mittaamista. Muita virhettä aiheuttavia tekijöitä GPS -paikannuksessa ovat epätarkkuudet ajanmittauksessa, matala- ja korkeapaineen vaihtelut säätilassa, virheet satelliitin fyysisen sijainnin ennustamisessa, sekä paikannuslaitteen rajoitteet. Yleisin paikannuslaitteen rajoite on, ettei kaikkia tarvittavia etäisyyksiä pystytä laskemaan tarkalleen samanaikaisesti, minkä vuoksi syntyy pieniä virheitä ajassa. Hyvinkin pieni virhe ajan määrityksessä aiheuttaa jo metriluokan virheitä kun käsitellään signaalia, joka liikkuu tyhjiössä valon nopeudella. (8, s. 3–7.)

Varsinkin vähäisellä anturoinnilla ilmenee mobiiliroboteilla melko suuria epätarkkuuksia halutun sijainnin ja todellisen sijainnin välillä. Tähän vaikuttavat monet epätarkkuustekijät, joista ensimmäisenä mainittakoon renkaiden ja ajopinnan välinen kitka. Mikäli rengas pääsee pyörähtämään tyhjää (kuten hiekalla tai pölyisellä hallin lattialla), tulee robotin rengasodometriaan nopeasti lisääntyvää virhettä mitatun ja todellisen sijainnin välille. Tämä ongelma ilmenikin projektissa melko alkuvaiheilla, kun yritin määrittää robotin kääntymisominaisuuksia. Mikäli ajettava käännös oli liian tiukka tai alusta liian pölyinen, alkoi vetävistä renkaista toinen tai jopa molemmat pyöriä tyhjää. Jo pienikin sutaisu aiheuttaa huomattavia virheen muutoksia robotin sijaintia jäljittävään rengasodometriaan, jonka tilaa on mahdollista tiedustella CAN-väylää pitkin. Tilanne parani jonkin verran kun robotin rakennetta muutettiin niin, että painojakauma oli enemmän vetävien pyörien päällä (tavaranostoihin tarkoitettu koura oli projektin alussa vapaasti kääntyvien pyörien päässä alustaa). Sutimisen vähentämisen lisäksi tämä toimenpide vähensi kasteripyörien kääntymisen aiheuttamaa fyysistä vastusta. (9, s. 1.)

Moottorien ajossa ilmeni myös jonkin verran epätasapainoa. Mitä ilmeisemmin MobilityModule-yksiköt ja niiden moottoriohjaimet eivät ole täysin tasapainossa keskenään. Tämä näkyi hyvin kulmamuutoksena, kun asetti alustan ajamaan suoraa ajoa useamman metrin matkan, jolloin minkäänlaista kulmamuutosta ei pitäisi ilmetä. Ajoittain robotin kulma muuttui jopa kymmeniä asteita suoran ajon aikana. Alustan mahdollinen epätasaisuus aiheuttaa myös haasteita tähän, mikäli kulkua määritetään yksinomaan renkaiden kulkeman matkan mukaan. Jos robotti ajaa suoran reitin jossa on kuoppa tai kumpare ainoastaan toisen renkaan kohdalla, tapahtuu kulmanmuutosta.

Käännöksiin epätarkkuutta tuovat alustan toisessa päässä olevat vapaasti kääntyvät, jo aiemminkin mainitut, kasterirenkaat. Renkaiden kääntymisen aiheuttamat kitkavoimat vaihtelevat sen mukaan, missä alkuasennossa kääntyvät renkaat ovat. Tämänkin syyn vuoksi rengasodometrian käyttö ei ole kannattavaa. Jos robotti asetettaisiin liikkumaan ennalta määrättyä rataa niin, että tarkkailtaisiin ainoastaan CAN-väylää pitkin tulevaa rengasodometriatietoa, seuraisi hyvin nopeasti virheen kasautumista. Tästä syystä anturointi tulee olemaan suuressa roolissa kun robotti tulee käyttöön. Anturitiedolla voidaan jatkuvasti korjata sijainti- ja kulmatietoa niin, ettei suurempaa virheen kasautumista pääse tapahtumaan.

Mahdollisena kehitysideana alustan rakennetta voisi muuttaa niin, etteivät vetävät renkaat olisi toisessa päässä alustaa vaan asetettuina robotin keskiosaan molemmille puolille, ja kääntyvät renkaat olisivat molemmissa päissä alustaa. Tällainen rakennemuutos auttaisi myös robotin painojakaumaa. Kääntyvät renkaat voisivat hyvin olla hieman irti maasta, jolloin paino olisi jakautunut lähes kokonaan vetäville MobilityModule™ 400 -yksiköille.

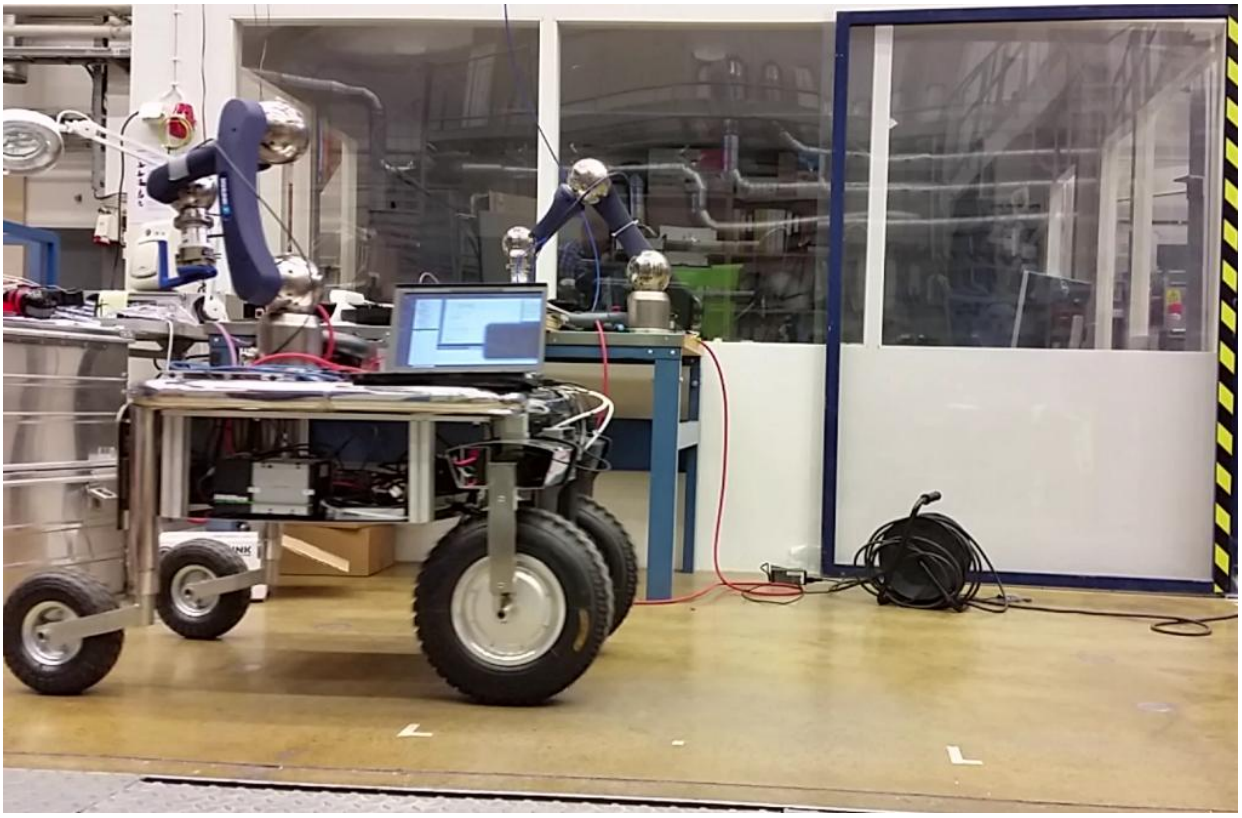
4 OHJAUSKOODIN TOTEUTUS

Ohjausta varten tuli luoda ohjelma, joka ottaa käyttäjältä vastaan tiedon halutusta määränpäästä sekä laskee ja toteuttaa sopivan trajektorin. Qt Creator on ohjelmistona varsin helppokäyttöinen ja varsinkin socket-yhteyden luominen oli todella helppo toteuttaa (verrattuna ratkaisuun, jossa ei ole käytettävissä valmista C++-luokkaa yhteyden muodostamista varten). Käyn tässä luvussa koodista otettujen otteiden avulla läpi ohjelman rakennetta sekä toimintaa.

Ohjelman rakennetta kuvaava kaavio löytyy työn lopusta liitteenä 1.

4.1 Robotin fyysisten ominaisuuksien määrittäminen

Halusin aluksi tarkastella robotin nopeuksia ja kiihtyvyyksiä lyhyissä ajoissa. Tätä varten tein aluksi Qt Creatorilla yksinkertaisen ohjelman, joka ottaa Beaglebone-tietokoneeseen socket-yhteyden ja lähettää sille käyttäjän syöttämän ajokomennon ja ajoajan. Ohjelma lähettää pysähtymiskomennon annetun ajoajan jälkeen. Käytännössä siis korvasin PlayStation-ohjaimen ajokomennon lähettäjänä Lenovo T400 kannettavalla tietokoneella. Tällä ohjelmalla tein ja kuvasin suoria ajoja, joissa tarkastelin alustan kiihtyvyyttä sekä sitä, millaisia SI-järjestelmän nopeuksia eri drive-komennot vastaavat. Seuraavaksi muutin ajosuunnan niin, että vetävät renkaat ovat edessä. Tämä tapahtui yksinkertaisesti muutamalla ajokomennon nopeusosan etumerkki negatiiviseksi. Kuvassa 2 esitetty mittausjärjestely oli yksinkertainen. Asetin maahan teipillä merkit tarkasti 50 cm:n välein ja kuvasin ajoja mahdollisimman hyvin sivusta käyttäen matkapuhelimen kameraa. Kuvan laatu oli 720p 60 kuvalla sekunnissa, joten tarkkuus oli varsin hyvä mittaustarkoitukseen.



KUVA 2. Robotin nopeuden ja kiihtyvyyden testausasetelma.

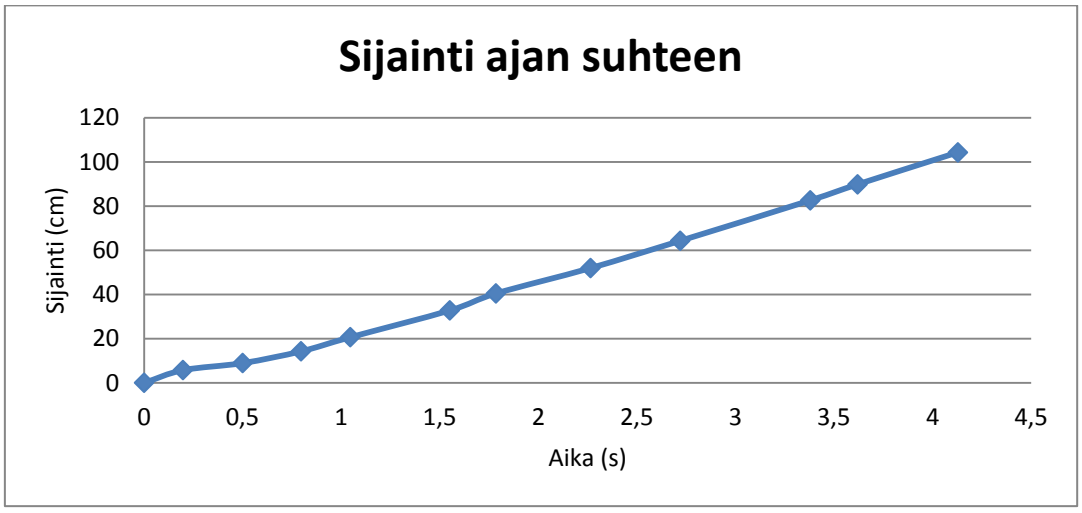
Testiajoissa ilmeni yksi luvussa 2 mainituista ongelmista robotin kanssa. Suora ajo ei mene suoraan, vaan kulman muutosta tapahtuu suurella vaihtelulla (jopa 15 asteen muutos kahden metrin mittaisella suoralla ajolla). Kuten mainitsin, tässä virheessä oli suurta vaihtelua ja myöhemmällä tutkimisella tulin siihen tulokseen, että suurin syyllinen tähän olivat vetämättömät renkaat. Pienikin asentovirhe vapaasti pyörivien renkaiden alkutilanteessa aiheutti robotin kaartumisen asentovirheen suuntaan. Tämän vaikutus pieneni hieman kun robottikoura siirrettiin vetävien renkaiden päähän, jolloin painojakaumasta tuli suotuisampi.

Kuvatusta videosta mittasin robotin liikkumista niin, että pysäytin kuvan Media Player Classic -ohjelmistoa käyttäen (kyseinen ohjelmisto mahdollistaa helposti millisekunnin tarkkuusluokassa olevan aikaleiman käyttämisen) ja otin ylös robotin kulkeman matkan ja ajan. Nämä ovat nähtävillä taulukosta 1.

TAULUKKO 1. Mitatut arvot testiajosta

Aika (ms)	Sijainti (px)
0	0
0,197	27
0,5	42
0,796	67
1,046	97
1,551	154
1,785	190
2,265	244
2,72	302
3,38	388
3,62	422
4,129	490

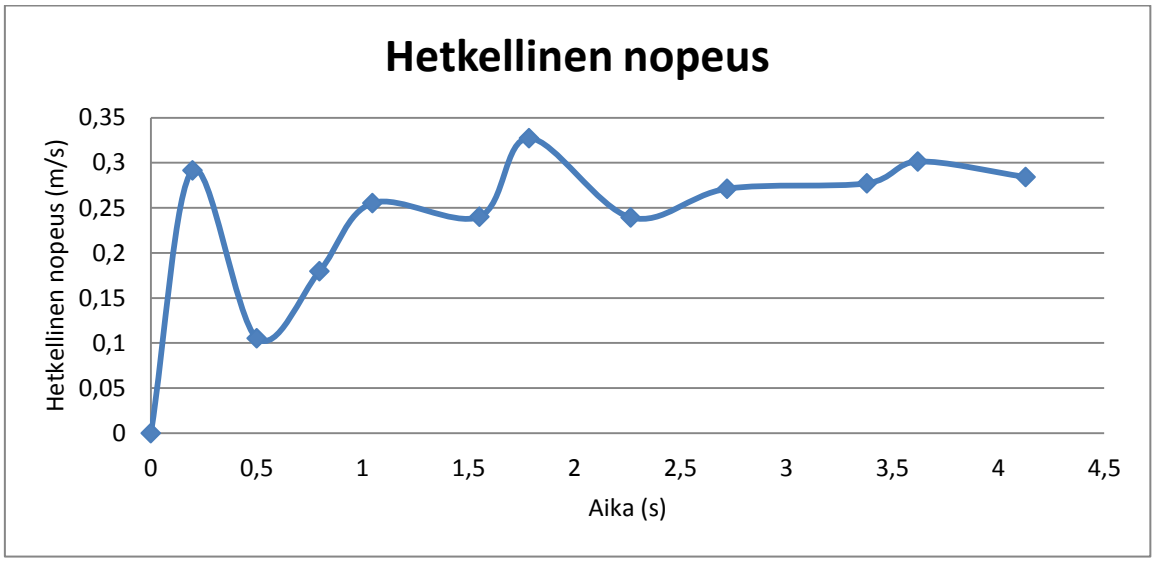
Mittasin ensin matkan pikseleinä ja muutin sen myöhemmin SI-järjestelmää vastaavaksi. Kuvatulta videolta yksi pikseli vastasi noin 2,13 millimetrin matkaa (eli yksi senttimetri oli noin 4,7 pikseliä). Kuvassa 3 on nähtävissä robotin sijainti ajan suhteen.



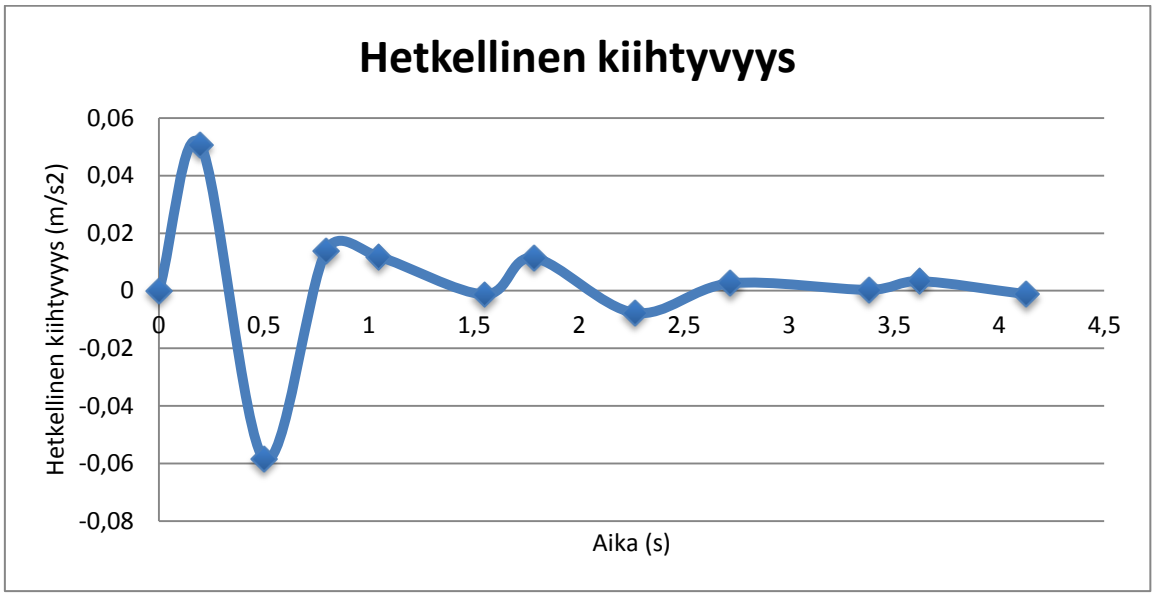
KUVA 3. Robotin sijainti ajan suhteen testiajossa.

Tässä opinnäytetyössä toteutettava robotin kulku on sokeaa (ilman ulkoista

anturointia) joten ajosegmenttien pituudet päätettiin rajata mittaamalla ajetun segmentin kestoa ja näin ollen arvioimalla, vastaako kuljettu matka tai tapahtunut kaarroksen pituus laskettua trajektoria. Mittauksen perusteella robotin kulku on melko tasaista (alun lyhyttä kiihdytystä lukuun ottamatta), joten suurpiirteinen trajektorin ajo voidaan toteuttaa aikaa mittaamalla. Tutkiakseni tarkemmin robotin liikkeen dynamiikkaa tein kuvaajat myös hetkellisen nopeuden mittauksista sekä hetkellisen kiihtyvyyden mittauksista, jotka ovat nähtävissä kuvista 4 ja 5.



KUVA 4. Hetkellinen nopeus mittapisteiden välillä.



KUVA 5. Hetkellinen kiihtyvyys mittapisteiden välillä.

Kuvista 4 ja 5 ilmeni todella hyvin, että alustan sisäisissä moottorinohjaimissa toimii PID-säädin, joka säätää nopeutta asetusarvoon. Itse mitattu ajokoekin vastaa askelvastekoetta, sillä nopeus asetettiin ilman ramppia tavoitearvoon. Videoidussa testissä käytin ajokomentoa ”drive 10 0” eli nopeudella 10 kääntämättä lainkaan. Kuvaajista on varsin helposti pääteltävissä ja laskettavissa, että kyseinen arvo vastaa reaalimaailmassa SI-järjestelmän arvoa 1 km/h, sillä 1 km/h = ~ 0,278 m/s, joka on hyvin lähellä loppuarvoa, johon kuvan 4 käyrä tasaantuu. Samoin kuvan 5 käyrä vastaa hyvin tasoittuvan säätimen käyttäytymisessä sitä, kuinka kovaa säädin pyrkii pienentämään erosuuretta.

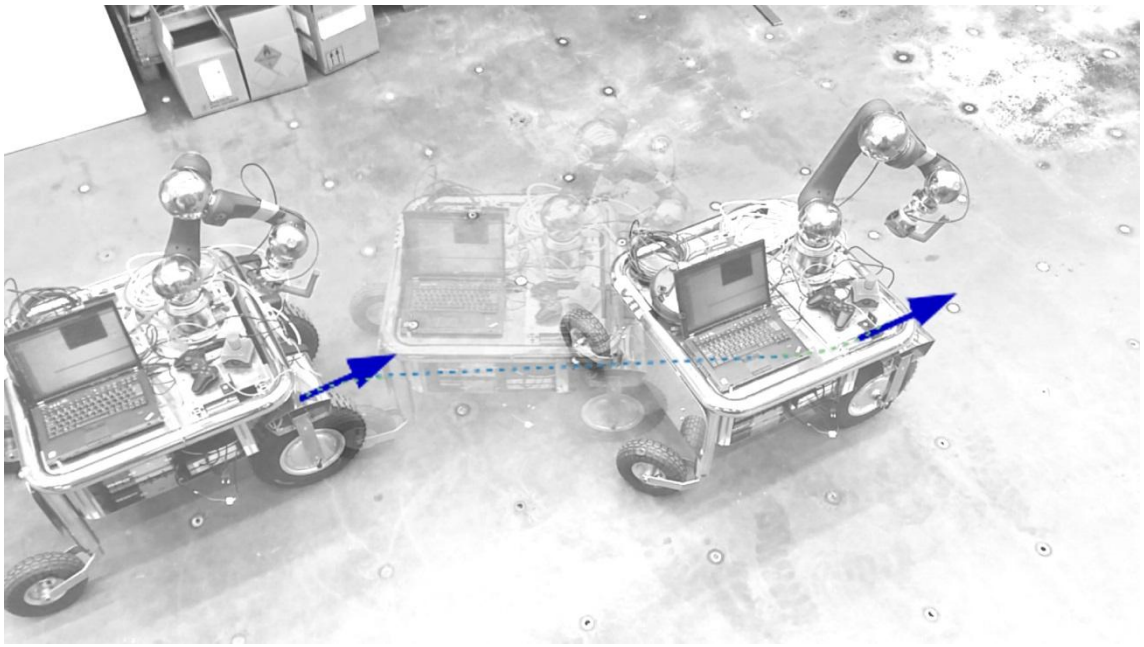
Seuraavaksi testasin kaarroksia. Näitä testejä tein runsaasti, ja sain määritettyä suurpiirteisen kertoimen, joka kertoo kuinka monen asteen suuruista kulmaa mikäkin ajoaika vastaa. Lisäksi tuli määrittää pienin mahdollinen kääntösäde robotille. Kävi ilmi, ettei robotti käänny lainkaan kaikilla ohjauksen komennoilla, sillä vetämättömien renkaiden aiheuttama vastus oli liian suuri. Aluksi määritin robotin kääntösäteeksi noin 40 senttiä (käsivaralla mittaus), mutta sain myöhemmin pienennettyä tätä 28 senttiin, jolloin robotti kääntyy toisen vetävän renkaan ympäri. Tämä vaati oivalluksen siitä, että skaalaus -15 ja 15 välille

tapahtuu vain ja ainoastaan Beaglebonella pyörivän PlayStation-ohjaimen Python-koodissa. Kannettavalta tietokoneelta ajettaessa ei siis ollut mitään estettä käyttää suurempia nopeuden ja ohjauksen komentoarvoja.

Dynamiikan kannalta ongelmaksi muodostuu se, että suoran ajo-osuuden jälkeinen käännös ei ole tasainen ympyräkaari, vaan lyhyt osa kaarteen alussa on loivempaa käännöstä. Tämä johtuu siitä, että toisen moottorin on hidastuttava suoran segmentin ajonopeudesta nollaan, ja reaalimaailmassa mikään hidastaminen ja kiihdyttäminen ei tapahdu välittömästi. Ensimmäisessä kaarteessa tätä ongelmaa ei esiinny, koska toinen rengas pysyy paikallaan koko kaarroksen ajan. Tämän ominaisuuden aiheuttamaa kulman muutosta kompensoitiin muuttamalla toisen kaarroksen ajon pituutta määrittävää kerrointa niin, että kaarros menee hieman pidemmälle. Toki toisen kaarroksen muodon riippuvuus suoran segmentin ajonopeudesta ja renkaan hidastuvuudesta aiheuttaa myös muutosta sijaintiin (vääristymä tasaiseen ympyräkaaren muotoon), mutta nämä muutokset ovat suuruudeltaan niin pieniä, ettei tämän kaltaisessa sokeassa ajossa sitä tarvitse huomioida.

4.2 Trajektorin laskenta

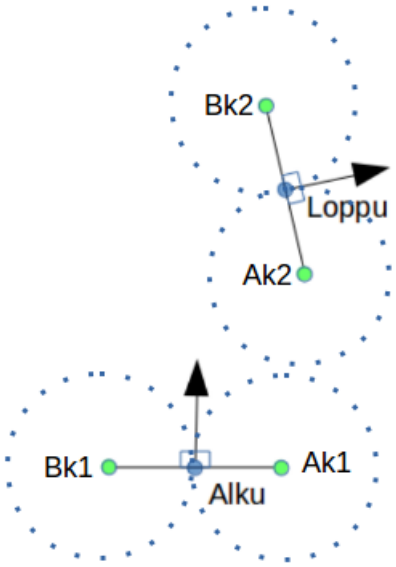
Tämän opinnäytetyön robotti käyttää trajektorinaan rataa, joka koostuu kolmesta segmentistä (kaksi kaarrossegmenttiä ja suora näiden välissä). Kaikki robotin kaarrokset tehdään samalla kääntösäteellä. Lopullisen reitin rakenne on nähtävissä kuvasta 6. Kuva 6 on manipuloitu valmiilla ohjelmalla tehdystä testiajosta kuvatusta videosta niin, että siinä on paremmin nähtävissä robotin kulkema reitti. Tässä ajossa käskettiin robottia kulkemaan pisteeseen (2, -1) kulmaan 0.



KUVA 6. Robotin kulkema reitti testiajossa.

4.2.1 Kaarroskeskikohdat

Ensimmäisenä trajektorille täytyy laskea kaarrosten keskikohdat. Loin koodiin double-tyyppiset muuttujat kaikille mahdollisille kaarroskeskiökoordinaateille, eli kaikille pisteille, jotka ovat kääntösäteen etäisyydellä kohtisuorasti alku- ja loppupisteestä. Kuvassa 7 on esitetty selventävästi mahdollisten kaarroskeskipisteiden määrittäminen.



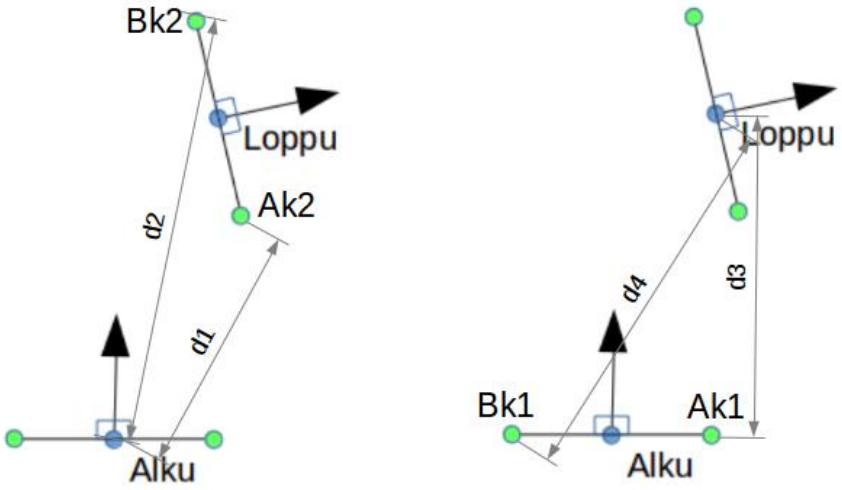
KUVA 7. Mahdolliset kaarroskeskipisteet merkattu kuvaan vihreällä.

Koodissa esiintyvät q-alkuisen toiminnot ovat Qt:n omista kirjastoista löytyviä toimintoja. Esimerkiksi kuvissa 8 ja 9 nähtävä qSqrt() suorittaa neliöjuuren, qPow() nostaa potenssiin ja qSin & qCos suorittavat trigonometrisia funktioita.

```
        double Akx1, Akx2, Aky1, Aky2, Bkx1, Bkx2, Bky1, Bky2;

// Neljän mahdollisen keskiöpisteen koordinaatit


        Akx1 = x1 + (qCos((kulma1 - 90)/180*pi))*r;
        Aky1 = y1 + (qSin((kulma1 - 90)/180*pi))*r;
        Akx2 = x2 + (qCos((kulma2 - 90)/180*pi))*r;
        Aky2 = y2 + (qSin((kulma2 - 90)/180*pi))*r;
        Bkx1 = x1 + (qCos((kulma1 + 90)/180*pi))*r;
        Bky1 = y1 + (qSin((kulma1 + 90)/180*pi))*r;
        Bkx2 = x2 + (qCos((kulma2 + 90)/180*pi))*r;
        Bky2 = y2 + (qSin((kulma2 + 90)/180*pi))*r;
```

KUVA 8. Ote kaarroskeskiöiden laskentakoodista

Kun mahdollisten pisteiden kaikki koordinaatit on määritetty, lasketaan millä pisteillä on lyhin etäisyys vastakkaiseen ratapisteeseen ja valitaan kyseisen pisteet käytettäviksi keskiöiksi. Tätä etäisyyttä selventää kuva 9.



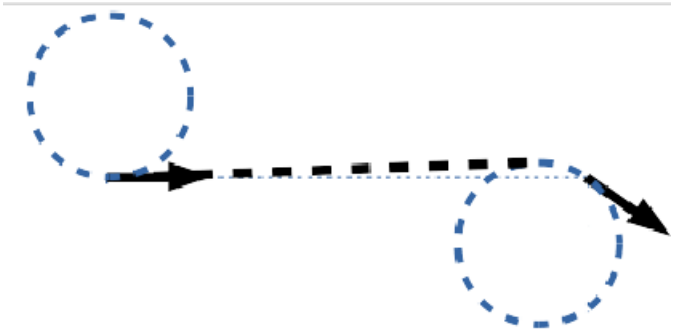
KUVA 9. Kaarroskeskiökoordinaattien etäisyydet.

Koodiote näiden etäisyyksien laskennasta on nähtävissä kuvassa 10.

```
// Seuraavaksi määritetään d1-d4 eli etäisyydet toiseen ratapisteeseen

        double d1, d2, d3, d4;

        d1 = qSqrt(qPow(Akx2,2) + qPow(Aky2,2));
        d2 = qSqrt(qPow(Bkx2,2) + qPow(Bky2,2));
        d3 = qSqrt(qPow((x2-Akx1),2) + qPow(y2-Aky1,2));
        d4 = qSqrt(qPow((x2-Bkx1),2) + qPow(y2-Bky1,2));
```

KUVA 10. Keskiöparien etäisyyksien laskeminen.

Tähän laskentaan sisältyy yksi lukuisista erikoistilanteista, jotka trajektorin laskennassa tulee ottaa huomioon. Tässä erikoistilanteessa d3 ja d4 ovat yhtä pitkät. Tämä aiheutuu ajossa, jossa seuraava ratapiste ei liiku kohtisuorasti lainkaan edelliseen verrattuna ja on esitettynä kuvassa 11.



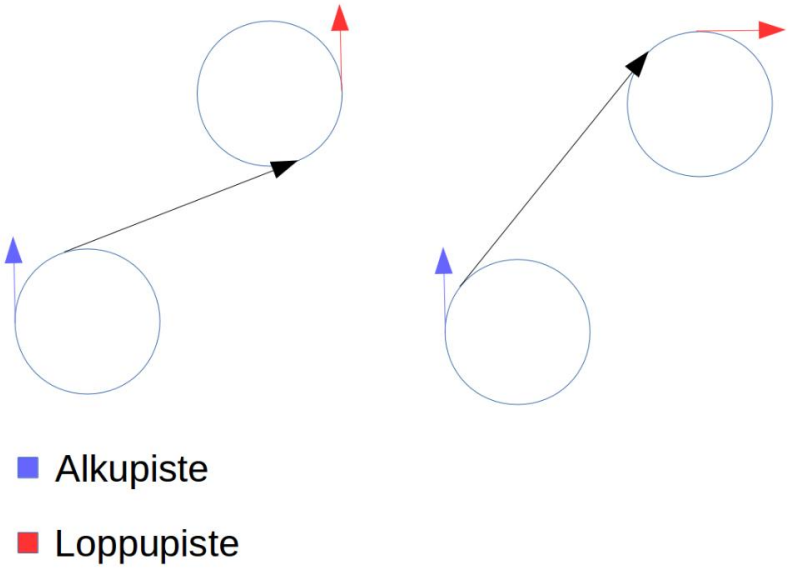
KUVA 11. Kaarroskeskiömäärityksen erikoistilanne.

Tämän tilanteen tunnistaminen ja korjaaminen vaati koodiin lisäkohdan, jossa verrataan alku- ja loppukulmaa. Käy järkeen, että tilanteessa jossa alkukulma on erisuuri kuin loppukulma ja kohtisuoraa sijainnin muutosta ei tapahdu, on rata aina muotoa jossa kaarrokset kääntyvät eri suuntiin (ellei haluta ajaa niin että ensimmäinen kaarros on lähes kokonainen ympyräkierros). Robotti kun ei voi mitenkään kääntää itseään ympyräkaarta pitkin siirtymättä suoralta ajoradalta sivulle. Tämän vuoksi voidaan todeta, että tässä erikoistilanteessa laskuissa käytettävä oikea kaarroskeskiö saadaan valitsemalla kulman muutokseen nähden vastakkaisen puoleinen keskiö.

Kun käytettävien keskiöiden koordinaatit ovat tiedossa, voidaan niistä valita pienin esimerkiksi käyttäen IF-lausejoukkoa. C++ on varsin monipuolinen ohjelmointikieli, ja usein koodattavan tehtävän toteutukseen löytyy todella paljon erilaisia ratkaisutapoja. Pienimmän arvon olisi voinut määrittää myös esimerkiksi taulukoimalla arvot ja tunnistamalla niistä pienimmän.

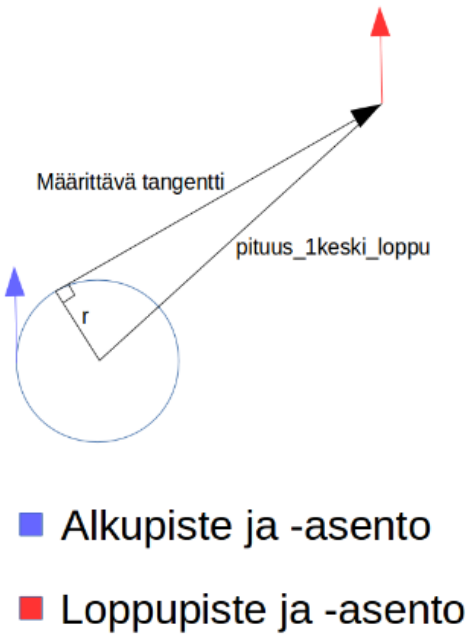
4.2.2 Trajektorin muoto

Trajektorit jaan muodon perusteella kahteen tyyppiin (C-tyyppisiin ja S-tyyppisiin) sen mukaan, kääntyvätkö kaarrossegmentit samaan suuntaan vai eri suuntiin. Tyyppijaottelu on havainnollistettu kuvassa 12.



KUVA 12. Trajektorin tyypin määritys

C-tyyppisessä trajektorissa sekä ensimmäisen kaarrossegmentin että viimeisen kaarrossegmentin kaarrokset ovat samaan suuntaan. S-tyyppisessä kaarrokset sen sijaan ovat erisuuntaiset. Kaarrostyypin määrittäminen tapahtuu ohjelmassa vertaamalla ensimmäisen kaarroksen ympyräkaarelle piirretyn tangentin kulmaa sekä loppusijainnin kulmaa. Ajatus tämän takana selkeytyy alla olevasta kuvasta 13.



KUVA 13. Trajektorin muodon määrittäminen vertaamalla loppukulmaa piirretyn tangentin kulmaan.

Lienee asiallista selventää, ettei kuvaan piirretty tangenttiviiva kuvaa varsinaista liikerataa, vaan ainoastaan auttaa selvittämään tarvitseeko alku- ja loppukaarteiden olla samansuuntaiset vai erisuuntaiset. Tämä trajektorimuodon määrittely tulee tehdä aikaisessa vaiheessa, sillä trajektorin tyyppi vaikuttaa myös ajettavien kaarroksien sekä suoran segmentin pituuksiin kaarroksien suunnan lisäksi.

Seuraavaksi ohjelma tarkistaa kumpaan suuntaan kaarrokset tulee tehdä. Käytin tähän int-tyyppistä muuttujaa nimeltään “suuntamuuttuja”, joka saa arvon 1 tai -1 (1 tarkoittaa että ensimmäinen kaarros tehdään oikealle ja -1 tarkoittaa vastaavasti vasemmalle kääntymistä). Suuntamuuttuja määritetään vertaamalla alku- ja loppupisteen välille piirretyn suoran (reitin kulma) sekä alkutilanteen kulmaa keskenään.

4.2.3 Kulmien suuntaus

Trigonometrisesti lasketuissa kulmissa ilmenee tiettyjä ongelmia trigonometrian jaksollisen luonteen vuoksi. Näinpä jokainen laskettu kulma tulee tarkistaa sen

mukaan, mihin koordinaatiston neljästä sektorista kyseinen piste sijoittuu. Käytän koodissa tästä laskutoimituksesta nimitystä “suuntaaminen”. Esimerkki siitä, miksi suuntaaminen on tarpeellista, voidaan esittää laskemalla reittikulma pisteeseen, joka on x-akselin negatiivisella puolella ja y-akselin positiivisella puolella. Kun lasketaan trigonometrisesti tangetilla reitin kulma, saadaan arvo joka on välillä -90 ↔ 0, kun todellisuudessa reitin kulma on 180 plus tämä negatiivinen kulma. Eli todellinen kulma on välillä 90 ↔ 180.

Suuntaus on joka kerralla toteutettu melko yksinkertaisella, joskin ajoittain sotkuisen näköiseksi paisuneella if-lausejoukolla. Näytteenä otan koodista reittikulman (alkupisteestä loppupisteeseen piirretyn suoran kulma) suuntauksen (esitetty kuvassa 14). Tässä kulma_reitti tarkoittaa kulmaa joka on saatu trigonometrisesti laskemalla y- ja x-koordinaattien muutoksien tangentista. Muuttujat x2 ja y2 tarkoittavat loppupisteen koordinaatteja, ja kulma_r_suunnattu tarkoittaa kulmasta jatkossa käytettävää, korjattua arvoa.

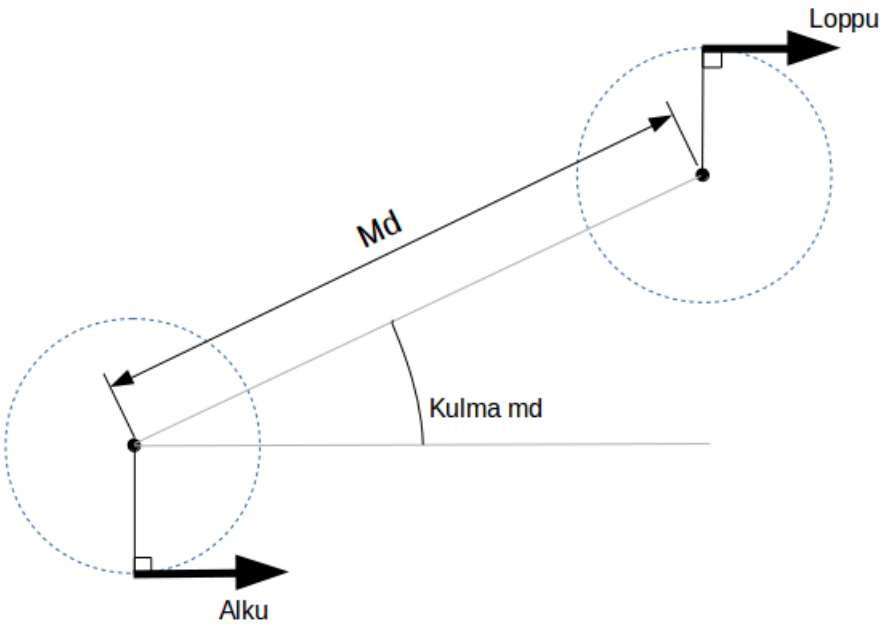
```
if ((x2>=0) & (y2>=0))
   {
      kulma_r_suunnattu = kulma_reitti;
   }

if ((x2<=0) & (y2>0))
   {
      kulma_r_suunnattu = 180 + kulma_reitti;
   }

if ((x2<0) & (y2<=0))
   {
      kulma_r_suunnattu = 180 + kulma_reitti;
   }

if ((x2>=0) & (y2<0))
   {
      kulma_r_suunnattu = 360 + kulma_reitti;
   }
```

KUVA 14. Esimerkki kulman suuntauksesta.

Kulmien suuntaus oli myös suurin virheellisten ratojen aiheuttaja trajektorin laskentaan, tarvittavien IF-lauseiden ja erikoisehtojen vuoksi.

4.2.4 Ajosegmenttien pituudet

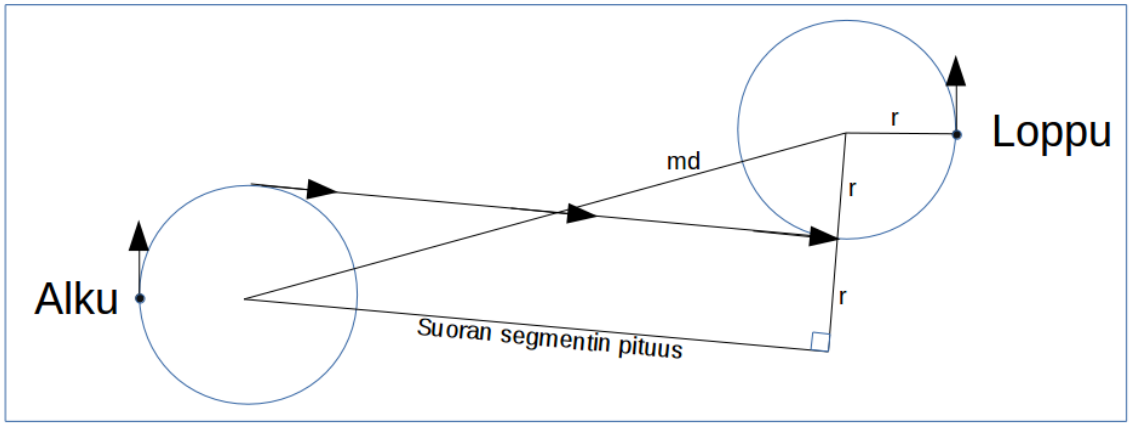
Ajosegmenttien pituuksien laskentaan vaikuttaa kumpaa muotoa laskettava rata on. Mikäli rata on tyyppiä, jossa molemmat kaarrokset tehdään samaan suuntaan, on suoran segmentin mitta sama kuin kaarroskeskipisteiden etäisyys toisistaan (käytin tälle etäisyydelle koodissa muuttujaa nimeltään md eli Mid Distance), mikä on esitetty kuvassa 15.



KUVA 15. Keskiöiden etäisyys ja niiden välisen suoran kulma.

Kaarrossegmenttien pituudet kulmissa ilmoitettuna saadaan melko vaivattomasti laskemalla ensin “md”-suoran suunnattu kulma “kulma_md” ja vertaamalla tätä lähtö- ja loppukulmaan. Kaikessa yksinkertaisuudessaan tarkastellaan kuinka paljon kulman on muututtava ensimmäisen kaarroksen aikana, jotta robottialusta saavuttaa kulman “kulma_md” ja vastaavasti kuinka monta astetta on muututtava edelleen kulmasta “kulma_md”, jotta päädytään haluttuun loppukulmaan.

Eri suuntiin tapahtuvien käännösten radalla trajektorin laskenta on hieman monimutkaisempaa. Kun edellisentyyppisessä trajektorissa suoran segmentin mitta tuli suoraan kaarroskeskiöiden välimatkasta (tangenttipiste molemmilla ympyräkaarilla samassa kohdassa), on tässä tapauksessa suoran segmentin määrittäminen tehtävä niin, että ajattelemme kuvioon suorakulmaisen kolmion. Tämän suorakulmaisen kolmion kateetit ovat suoran segmentin mitta sekä kaksi kertaa kääntösäde ”2r”. Tällöin hypotenuusa on kaarroskeskiöiden välimatka ”md”. Laskenta on esitetty alla kuvassa 16.



Kuva 16. Suoran segmentin pituuden laskeminen S-tyyppiselle trajektorille.

Kulmien laskenta tämäntyyppisessä trajektorissa menee melko vastaavasti kuin kahden samansuuntaisen kaarroksen trajektorillakin, eli tutkimalla alku- ja loppukulmien eroa suoran segmentin suunnattuun kulmaan (lasketaan kuinka paljon kulman on muututtava, jotta saavutetaan haluttu tila).

4.3 Radan ajaminen

Kun trajektorilaskenta on tehty, siirtyy ohjelma ajamaan laskettua rataa. Tässä työssä ei ohjelmaan toteutettu konsolisovellusta monimutkaisempaa käyttöliittymää, vaan navigoinnin toteutus nostettiin ensisijaiseksi teemaksi. Sovellus

pyytää ensin käyttäjältä ajettavan ratapisteen kulman sekä x- ja y-koordinaatit. Tämän jälkeinen ohjelman toiminta on selostettu tässä luvussa.

Ensimmäisenä muodostetaan socket-yhteys BeagleBonen ja ohjelmaamme suorittavan kannettavan tietokoneen välille. Tämä socket-yhteys ottaa käyttöön BeagleBonella aiemmin käytetyn PlayStation-ohjaimen paikan. Ohjelman tulee luoda uusi socket, jotta yhteys voidaan muodostaa. Tämä oli varsin helppoa toteuttaa Qt Creatorin valmiin QTcpSocket-luokan avulla. Varsinainen yhdistäminen tapahtuu koodissa käyttäen riviä ”socket-> connectToHost ("192.168.7.2",54345)”, jossa käytetty IP-osoite ja portti 54345 ovat BeagleBonella ennalta määritettyjä.

Mikäli yhteydessä ilmenee jokin ongelma (toisin sanoen mikäli yhteyden muodostaminen ei onnistu koodissa määritetyn sekuntimäärän sisään), tulostuu käyttäjälle teksti ”Yhteysvirhe” ja ohjelma loppuu. Mikäli yhteys onnistuu, ohjelma lähettää sockettiin QByteArray-muodossa (bittijonona) ensimmäiselle segmentille määritettyä drive-komentoa. BeagleBonella komennot siirtyvät Probot Oy:n tekemien Python-koodattujen ohjelmien käsiteltäväksi ja siitä edelleen CAN-väyliä pitkin moottoreiden ohjaimille. Näillä Probot Oy:n tekemillä ajuriohjelmistoilla ei ole käyttöliittymää,vaan ne suorittuvat itsenäisesti BeagleBonella, johon ei kuulu näyttölaitetta. Ohjelma alkaa ajokomennon lähettämisen jälkeen kutsua 100 ms välein ”trajectory_control"-oliota ja lähettää sille samalla tiedon siitä, kuinka pitkä (ajallisesti) nykyisen segmentin tulisi olla. Mikäli robotti tuotteistetaan joskus markkinoille asti, tulee tämä lähetettävä aikatieto korvata halutulla kulman muutoksella tai suoran segmentin tapauksessa halutulla sijainnin muutoksella, joita tarkkaillaan anturoinnilla.

Trajectory_control-olion idea on, että se laskee kuinka monta kertaa oliota on kutsuttu ja palauttaa 100 millisekunnin välein muuttujan ”operation”. Tämän muuttujan arvo (int-muotoinen kokonaisluku) puolestaan kertoo kutsuneelle ohjelmalle kuinka toimia. Kun tavoitteena oleva segmentin ajopituus on saavutettu, lataa ohjelma ".txt"-tiedostoon tallennetun, seuraavaa segmenttiä kuvaavan da-

tan (segmentin ajallinen pituus sekä sen ajokomento) ja aloittaa uuden segmentin ajamisen vastaavalla tavalla kuin edellisen.

Tässä vaiheessa operation-muuttujan hallinnoimia toimenpiteitä ovat ainoastaan ”0: jatka samaa segmenttiä” tai ”1: siirry seuraavaan segmenttiin”, mutta anturitiedolla tähän on mahdollista lisätä myös toimenpiteitä, kuten esteentunnistus ja sen väistäminen, sekä ajon tarkkuutta ylläpitävä säätö. Tällöin Trajectory_Control-osa kannattaisi toteuttaa esimerkiksi switch-case rakenteella, jolloin uusien ”operation”-toimintojen lisääminen olisi hyvin helppoa ja kompaktimpaa kuin saman toteuttaminen käyttäen IF-lausejoukkoa.

Viimeisen ajosegmentin päätteeksi ohjelma lähettää ”drive 0 0” -komennon, mikä käskee robotin pysähtymään, ja sulkee tämän jälkeen avoinna olevan socket-yhteyden.

5 YHTEENVETO

Työn aiheena oli differentiaaliajoisen mobiilirobotin navigoinnin ohjelmoiminen niin, että robotti osaa liikkua käyttäjän antamaan sijaintiin ja asentoon. Lopputuloksena oli robotti, jonka ohjaus toimi riittävällä tarkkuudella silloiseen anturoinnin tasoon nähden.

Tässä työssä tehty koordinaattiohjaus laskee tarkasti oikein tarvittavat ajomatkat sekä kaarroksien pituudet, mutta näiden tarkka ajaminen on täysin mahdotonta ilman erittäin tarkkaa anturointia (joka robottiin myöhemmin pyritään lisäämään). Jo pienet epätasaisuudet lattiassa aiheuttavat silmin havaittavaa epätarkkuutta ajoon, puhumattakaan siitä epätarkkuudesta, joka seuraisi ulkona, epätasaisella alustalla ajamisesta.

Tässä työssä ajamiseen käytetyt segmenttiajat tulisi siis unohtaa kokonaan, ja käyttää niiden sijaan anturidataa kertomaan tarkasti robotin kulmaa ja sijaintia. Ulkokäytössä suosisin itse RTK-pohjaista satelliittipaikannusta, jonka tarkkuus on riittävä jopa VR (Virtual Reality) -sovelluksissa käytettäväksi.

Mobiilirobotiikka on kehittyvä ala, joka hyppää massiivisesti eteenpäin, kun anturointiin ja sijainnin päivittämiseen liittyvät ongelmat onnistutaan ratkaisemaan yksiselitteisesti.

LÄHTEET

1. Singh, Jaspreet – Kaur, Pinkiparampreet 2008. Object Oriented Programming Using C++. Technical Publications Pune.

2. Rischpater, Ray – Zucker, Daniel 2010. Beginning Nokia Apps Development. Apress.

3. Alanen, Jarmo 2000. CAN - ajoneuvojen ja koneiden sisäinen paikallisväylä. VTT. Saatavissa: http://www.oamk.fi/~eeroko/Opetus/Ohjausjarjestelmat/CAN/CAN-perusteet_AlasenMateriaalia.pdf. Hakupäivä 1.10.2016.

4. Lima, Pedro – Ribeiro, Maria Isabel 2002. Mobile Robotics. Instituto Superior Técnico. Saatavissa: http://users.isr.ist.utl.pt/~mir/cadeiras/robmovel/Introduction.pdf. Hakupäivä 2.8.2016.

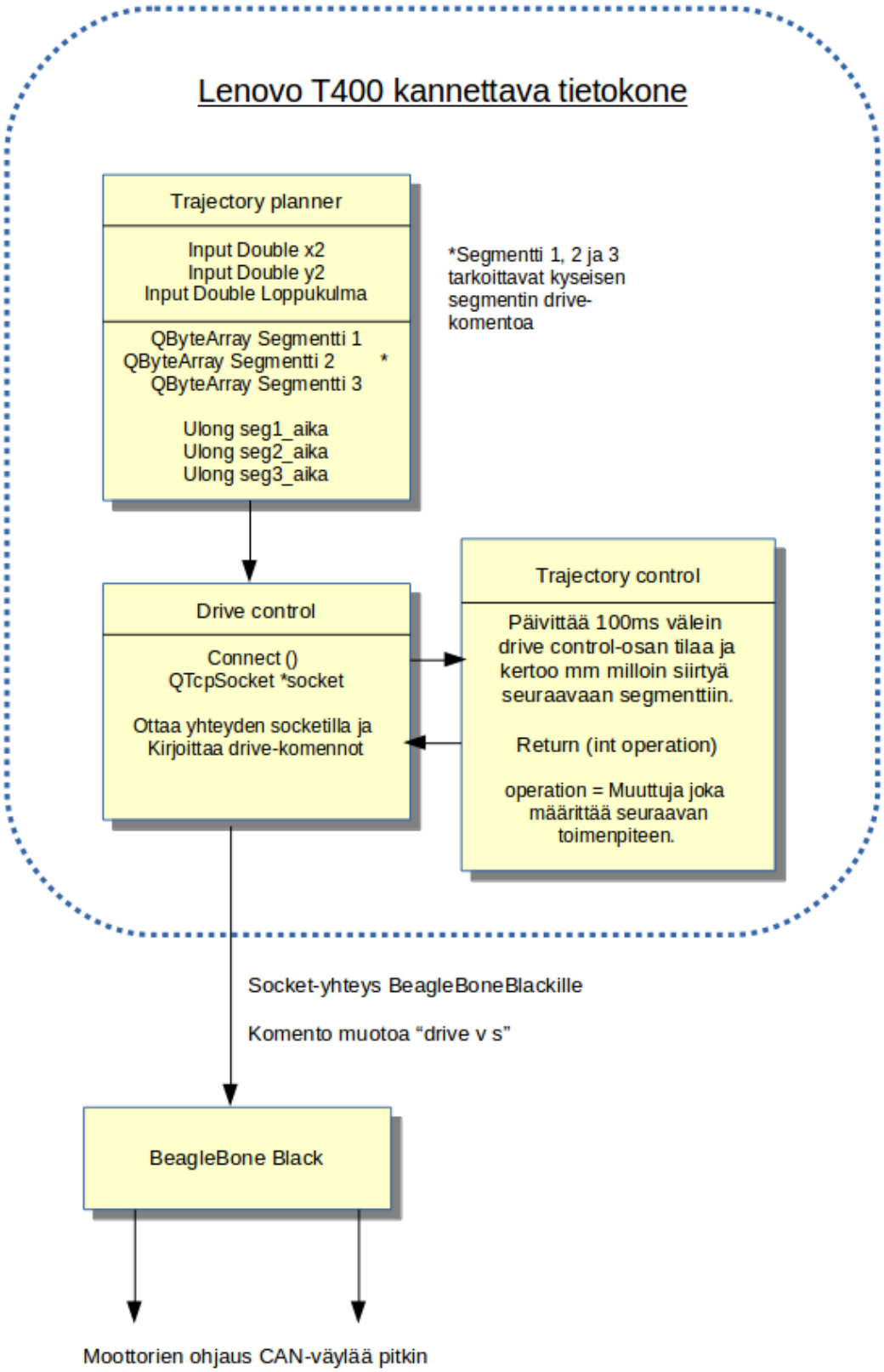
5. Siegwart, Roland – Nourbakhsh, Illah R. 2004. Introduction to Autonomous Mobile Robots. Massachosetts Institute of Technology.

6. Durrant-Whyte, Hugh – Bailey, Tim 2006. Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms. Saatavissa: http://www-personal.acfr.usyd.edu.au/tbailey/papers/slamtute1.pdf. Hakupäivä 14.9.2016.

7. Daniels, Richard C. – Huxford, Robert H. 2001. Using Global Positioning Systems (GPS): How it Works, Limitations, and Some Guidelines for Operation. Washington Department of Ecology. Saatavissa: https://fortress.wa.gov/ecy/publications/publications/0006015.pdf. Hakupäivä 24.8.2016.

8. Laaksonen, Janne 2007. Mobile Robot localization using sonar ranging and WLAN intensity maps. Lappeenranta University of Technology.

9. Tian, Yu – Sarkar, Nilanjan 2013. Control of a Mobile Robot Subject to Wheel Slip. Springer Science+Business Media Dordrecht . Saatavissa: http://research.vuse.vanderbilt.edu/rasl/wp-content/uploads/2014/01/pdf/control%20of%20mobile%20robot.pdf. Hakupäivä 1.9.2016.


Lenovo T400 kannettava tietokone
Trajectory planner
Input Double x2
Input Double y2
Input Double Loppukulma
QByteArray Segmentti 1
QByteArray Segmentti 2 *
QByteArray Segmentti 3
Ulong seg1_aika
Ulong seg2_aika
Ulong seg3_aika
*Segmentti 1, 2 ja 3 tarkoittavat kyseisen segmentin drive-komentoa
Drive control
Connect ()
QTcpSocket *socket
Ottaa yhteyden socketilla ja Kirjoittaa drive-komennot
Trajectory control
Päivittää 100ms välein drive control-osan tilaa ja kertoo mm milloin siirtyä seuraavaan segmenttiin.
Return (int operation)
operation = Muuttuja joka määrittää seuraavan toimenpiteen.
Socket-yhteys BeagleBoneBlackille
Komento muotoa “drive v s”
BeagleBone Black
Moottorien ohjaus CAN-väylää pitkin