Tampereen yliopisto

Nikita Melentyev

ROBOTTIKÄSIVARREN TARKKUUDEN MÄÄRITTÄMINEN

Tekniikan ja luonnontieteiden tiedekunta
Kandidaatintyö
Syyskuu 2019

TIIVISTELMÄ

Nikita Melentyev: Robottikäsivarren tarkkuuden määrittäminen
Kandidaatintyö
Tampereen yliopisto
Tekniikan kandidaatin tutkinto-ohjelma
Syyskuu 2019

Robotti on hyvin monimutkainen systeemi ja sen ohjaaminen, varsinkin autonomisella tasolla, sitoo useita osajärjestelmiä. Robotilla on oltava joukko sensoreita, joilla se voi saada tietoa niin omasta kuin ympäristön tilasta. On oltava myös joukko osia, jotka mahdollistavat robotin liikkumisen, sekä työkaluja, joilla robotti saa aikaan muutoksia ympäristössään. Lisäksi robotti tarvitsee prosessorin, joka käsittelee sensoridataa ja ohjaa koko robotin toimintaa. Haasteena on saada nämä osajärjestelmät toimimaan yhdessä annetun tehtävän suorittamiseksi.

Eräs robottityyppi on robottikäsivarret. Nimitys tulee robotin kyvystä suorittaa ihmiskäden kaltaisia toimintoja. Tälläisilla roboteilla on lukuisia sovelluksia mm. teollisuudessa ja usein tarkkuus nousee tärkeäksi robottikäsivarren ominaisuudeksi. Tarkkuutta voidaan ajatella kahdesta näkökulmasta. Toistotarkkuus kuvaa robotin kykyä palata samaan asentoon ja paikoitustarkkuus puolestaan kuvaa saavutetun asennon poikkeavuutta tavoiteasennosta.

Robotin toiminta määritetään ohjelmoimalla prosessori. On olemassa useita ympäristöjä, jotka tarjoavat eritasoista tukea robottien ohjelmointiin. Matalan tason ohjelmointi on robotin yksittäisten moottorien ohjelmointia ja korkeammille tasoille siirtyessä ohjelmointi muuttuu abstraktimmaksi. Korkean tason ohjelmoinnissa ympäristö hoitaa itse matalan tason haasteet ja käyttäjä voi keskittyä osajärjestelmien ohjaamiseen ja käytännön toiminnan suunnitteluun. ROS on eräs ohjelmointiympäristö, joka tarjoa monipuoliset työkalut robottien ohjelmointiin, kehitykseen ja simulointiin. Työssä on perehdytty kyseisen ympäristön toimintaan ja yleisimpiin työkaluihin.

Työn tavoitteena oli määrittää erään robottikäsivarren tarkkuus. Tähän tarkoitukseen luotiin ohjelma käyttäen ROS-ympäristöä. Testiohjelma ohjasi robottia ennalta määrättyihin asentoihin ja keräsi epäsuorasti paikoitusdataa. Saadun datan avulla saatiin määritettyä robotin toisto- ja paikoitustarkkuus, sekä päädyttiin tulokseen, että työssä käytetty robotti kannattaa kalibroida paremman tuloksen saavuttamiseksi.

Avainsanat: Robottikäsivarsi, robotin ohjelmointi, robotin tarkkuus, ROS-ympäristö

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

ROS	Robot Operating System
URDF	Unified Robot Description Format
PID-säädin	Proportional-integral-derivative-säädin
GPS	Global Positioning System
EN-standardi	Eurooppalainen standardi
ISO-standardi	Kansainvälinen standardi (International Standard Organization)

1. JOHDANTO

Robottien käyttö on jatkuvassa kasvussa alalla kuin alalla. Robotit mahdollistavat työskentelyn ihmiselle ulottumattomissa paikoissa ja usein ovat ihmistä tehokkaampia erilaisten ominaisuuksien kohdalla. Sellaisia ovat mm. nopeus, voima, kestävyys ja tarkkuus.

Teollisuudessa robotteja käytetään esineiden liikuttamiseen, valmisteiden työstämiseen sekä monenlaisiin kokoonpanotehtäviin. Lääketieteessä täysin autonomisten robottien käyttö ei toistaiseksi ole yleistä, mutta kehityksen alla on monenlaisia sovelluksia esimerkiksi kirurgiassa, sairauksien diagnosoinnissa ja laboratoriotyössä. Autonomisten robottien sijaan käytetään tietokoneavusteisia manipulaattoreita, jotka esimerkiksi mahdollistavat pienemmän haavan leikkauksessa, takaavat monipuolisimmat ja tarkemmat liikkeet sekä vaimentavat operaattorin käsien tärinää näin lisäten liikkeiden vakautta. Logistiikassa ja farmakologiassa automaattiset varastot hoitavat pakkausten varastoinnin ja noudon, jolloin työntekijä voi keskittyä paremmin muuhun työhön, kuten asiakaspalveluun. [1]

Jo tässä vaiheessa robotteja kehitetään mitä erilaisimpiin käyttökohteisiin ja näin olleen niiden käyttötavat ja toimintamenetelmät vaihtelevat suuresti. Tässä työssä keskitytään käden liikkeitä matkiviin robottikäsivarsiin. Robottikäsivarsien liikkeiden tarkkuus nousee tärkeäksi ominaisuudeksi monessa käyttökohteessa.

Ennen kuin voidaan puhua robotista millään tasolla, robotti on ohjelmoitava. Tähän tarkoitukseen on olemassa useita ohjelmistoja ja järjestelmiä, jotka tarjoavat monipuolisia työkaluja niin yksittäisen robotin liikkeiden ja käytöksen suunnitteluun, kuin useasta robotista koostuvien systeemien suunnitteluun, testaamiseen ja integrointiin. Tässä työssä keskitytään Robot Operating System – ympäristöön, perehdytään sen toimintaperiaatteisiin sekä käytetään sitä robottikäsivarren ohjelmointiin.

Luku kaksi käsittää työn teoreettista taustaa. Siinä avataan robottia käsitteenä, sekä perehdytään robotin ominaisuuksiin esimerkkien kautta. Lisäksi luvussa kaksi kerrotaan enemmän ROS:sta ja sen lisäosista, tarkkuudesta robotin näkökulmasta, sekä esitetään tulosten käsittelyn kannalta olennaisia matemaattisia käsitteitä ja kaavoja. Luvussa kolme kerrotan tarkemmin suoritetusta tutkimuksesta, sekä esitetään yksityiskohtaisesti

työssä käytettyjä menetelmiä robotin tarkkuuden määrittämiseksi. Tämän jälkeen esitetään yhteenveto tarkkuusmittauksen tuloksista luvussa neljä. Luvussa viisi käydään läpi tulosten käsittelyä ja luvussa kuusi esitetään päätelmiä ja yhteenveto suoritetusta tutkimuksesta.

2. TEOREETTINEN TAUSTA

Työn tarkoituksena oli perehtyä robottien ohjelmointiin, tarkkuuteen sekä tutkia erään robotin tarkkuutta käytännössä. Työssä kehitettiin ohjelma, jonka avulla robotin tarkkuus saataisi määritettyä. Ohjelmointi suoritettiin käyttäen ROS-ympäristöä ja ohjelmointikielenä C++:aa.

Tässä luvussa käsitellään tutkimuksen taustatietoja. Avataan muun muassa robottia käsitteenä ja fyysisenä oliona, sekä käydään läpi tutkimuksessa käytettävän robotin rakennetta ja valmistajan lupaamia ominaisuuksia. Tutustutaan tarkemmin Robot Operating Systemiin, sen toimintaperiaatteisiin sekä käyttökohteisiin. Lisäksi käsitellään työssä käytettäviä kirjastoja, jotka pohjautuvat ROS-järjestelmään. Lopuksi käsitellään työn matemaattista taustaa ja pohditaan, mistä robotin liikkumistarkkuus koostuu.

2.1 Robotin tarkastelu

Käsitettä ”robotti” on käytetty ensimmäisen kerran Karen Capekin näytelmässä vuonna 1921. Tšekin kielellä *robota* tarkoittaa etymologisesti orjaa tai työläistä eli viittaa jonkinlaiseen pakkotyöhön.

Teollisuudessa ja tieteessä yksikäsitteistä määritelmää robotille ei ole. Eurooppalainen standardi EN775/1992 määrittää manipuloivan teollisuusrobotin automaattisesti ohjautuvaksi uudelleenohjelmoitavaksi monikäyttölaitteeksi, jolla on useampia vapausasteita ja joka on asennettu liikkuvalle tai kiinteälle alustalle. Puolestaan *Robotics Institute of America* ei ota kantaa alustaan, mutta vaatii robotilta muiden yllämainittujen ominaisuuksien lisäksi kyvyn tehdä havaintoja ja reagoida niiden mukaisesti. Eräs määrittely voi olla, että robotti on olio, jolla on fyysinen vaikutus ympäristöönsä, joka muokkautuu robotin havaintokyvyn ja ympäristön muutoksen myötä. [2]

Yleisesti ottaen voidaan sanoa, että robotti on älykäs kone, jolla on jonkinlainen itsetietoisuus ja joka on ohjelmoitu tekemään annettua tehtävä automaattisesti, kuitenkin ottaen muuttuvan ympäristönsä huomioon. Robotin tieto ympäristöstä ja itsestään tulee pääasiassa antureista ja kameroista, joiden määrä voi tarkoituksesta riippuen vaihdella. Jo yksittäisen anturin dataa on mahdollista käyttää robotin tilan muuttamiseen. Esimerkkinä tästä valoanturin avulla robotin suuntaaminen valon suuntaan tai valosta poispäin. Toisaalta on mahdollista yhdistää useammasta anturista saatava data ja työstää se muuksi käytön kannalta hyödylliseksi dataksi. Esimerkiksi robottikäsivarren tapauksessa

yksittäisten servomoottoreiden asematiedoista ja robotin akselien pituuksista voidaan epäsuorasti laskea robotin työkalun asema koordinaatistosta. Tällaista koordinaattien määritystapaa hyödynnetään myös tässä tutkimuksessa.

Robottikäsivarsi on yksi käytetyimpiä konetyyppejä teollisuudessa. Rakenteeltaan robottikäsivarsi muodostuu akseleista (engl. *link*) joita yhdistävät nivelet (engl. *joint*). Nivel mahdollistaa kahden akselin liikkumisen toisensa nähden. Tyypillisesti nivelet ovat kiertoniveliä (engl. *revolute joint*), jotka mahdollistavat saranan tapaan rotaatioliikkeen, tai lineaarisia niveliä (engl. *prismatic joint*), joiden ansiosta translaatioliike ja suhteellinen liike ovat mahdollisia. Nivelten määrän kasvaessa myös robotin vapausasteiden eli mahdollisten liikesuuntien määrä kasvaa. Tavallisesti teollisuusroboteilla on kuusi vapausastetta. Tämä tarkoittaa käytännössä sitä, että robotti voi lähestyä työkalullaan, rajatapauksia lukuun ottamatta, jokaista toiminta-alueen pistettä mistä tahansa suunnasta. Kuudella vapausasteella on etua esimerkiksi viiteen, kun ympäristössä esiintyy esteitä tai kun työstämisen pitää tapahtua tietyssä kolmiulotteisessa kulmassa. [3]

Käyttämällä erilaisia nivelyhdistelmiä sekä nivelten kiinnitystapoja saadaan aikaiseksi lukuisia geometrioita robottikädelle. Robotin rakenne määritellään ensimmäisen kolmen nivelen perusteella, mikäli kyseessä on kinemaattinen rakenne, jossa nivelet eivät muodosta silmukoita keskenään. [3] Tutkimuksessa käytetyn robotin rakennetta tarkastellaan myöhemmin.

Robottien liikkumisen yhteydestä puhutaan usein kinematiikasta. Kinematiikka on tieteen ala, joka tarkastelee liikettä huomioimatta itse liikkeen alkuperää. Sen avulla voidaan tarkastella ja suunnitella robotin asentoja sekä siirtymistä asennosta toiseen. Kun halutaan ratkaista robotin työkalun sijainti toiminta-avaruudessa robotin asennon kautta, puhutaan kinemaattisesta ongelmasta. Puolestaan robottia ohjelmoidessa halutaan usein päinvastainen tieto eli missä asennossa robotin kuuluu olla, kun halutaan saavuttaa tietty piste. Silloin on kyse käänteisestä kinemaattisesta ongelmasta. [3]

Tässä työssä tullaan puhumaan paljon robotin tarkkuudesta. Robottikäsivarren kohdalla se tarkoittaa kykyä saavuttaa haluttu työavaruuden piste mahdollisimman tarkasti. Tarkkuutta käsitellään myöhemmin tarkemmin. Teollisuuden sovelluksissa robottikäsivarret saavuttavat jopa 0,01 millimetrin tarkkuuden useamman kilogramman käsittelykyvyllä. Tässä työssä käytetty robotti ei kykene ihan samanlaisiin tuloksiin ja jää noin 2,5 – 5,0 millimetrin tasolle, joka tälle käyttötarkoitukselle ja hintaluokalle on aika hyvä tulos. [4]

Teollisuusrobottien tarkkuus määritellään soveltuvien standardien mukaan. Standardi ISO 9283:1998 määrittää, että tarkkuutta arvioidessa on käytettävä maksivirhettä ja ajon aikana on käytettävä maksimikuormaa sekä maksimiliikkumisnopeutta. Näiden lisäksi

virralliseen tarkkuusmittaukseen kuuluu joukko määräyksiä liittyen itse mittausasetelmaan. Lyhyesti esitettynä seuraavien ehtojen pitää täyttyä:

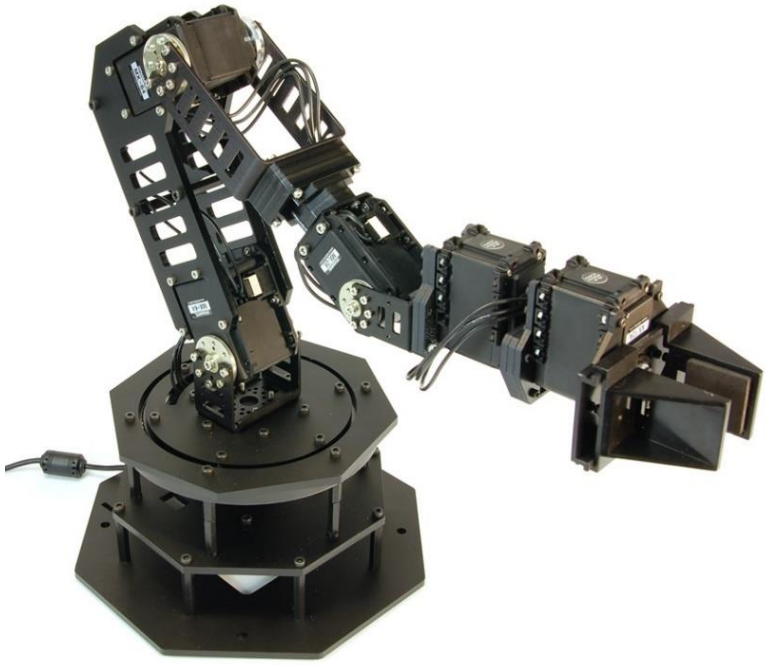
1. on suoritettava lämmitysajo
2. robotti on käskettävä kolmeen eri asentoon
3. on käytettävä kahta kameraa ja robottiin kiinnitettyä opista kohdistinta.

Ensin on siis suoritettava lämmitysajo, kunnes koneisto (moottorit ja vaihteisto) saavuttaa stabiilin lämpötilan ympäristön ollessa 71 F (21.7 °C) asteen lämpötilassa. Sitten robotti on käskettävä kolmeen asentoon. Käskyjen on oltava keskenään samantyylisiä. Viimeinen ehto määrittää, että mittauksessa on käytettävä kahta kameraa ja robottiin kiinnitettyä kohdistinta tai muita vastaavia työkaluja, joilla tarkkuus voidaan mitata ulkoisesti. [5]

Tässä työssä tarkkuusmittaus suoritettiin hieman kevyemmillä ehdoilla. Robotille suoritettiin mittausohjelman testiajo useampaan kertaan ennen varsinaista datan tallentamista lämmityksenä. Robotin asema varmistettiin ensimmäisellä testikierroksella ulkoisesti rullamitan avulla ja arvioitiin silmämääräisesti robotin käyttäytymistä mittauksessa. Laskuissa käytetyt koordinaatit ovat epäsuorasti määritettyjä robotin kinematiikan avulla. Mittauksen kulusta puhutaan myöhemmin tarkemmin.

2.2 WidowX Robot Arm

Tutkimuksessa käytetty robotti on Trossen Roboticsin kehittämä Interbotix-tuoteperheeseen kuuluva WidowX Robot Arm Mark II. Robotti voidaan nähdä kuvassa 1. Robotti koostuu tornimaisesta alustasta, kolmesta akselista, sekä kuudesta servomoottorista. Robotilla on viisi vapausastetta, joka mahdollistaa hyvin monipuoliset liikkeet. Työkaluna toimii tarttuja, jonka maksimiväli yhden pehmustekerroksen kanssa on 32 mm. On myös mahdollista luopua yhdestä vapausasteesta mikä valmistajan mukaan lisää tarkkuutta ja luonnollisesti käsittelykykyä. [6] Kyseessä on siis tarttujan kiertoliikkeen poistaminen. Robotti koottiin ensimmäistä kertaa, joten suoritettava mittaus oli samalla robotin testiajamista.



Kuva 1. *Tutkimuksessa käytetty robotti WidowX Robot Arm Mark II. [7]*

Valmistajan sivulta ei löydy tarkkaa tietoa kyseisen robotin tarkkuudesta. Tarkkuusominaisuuksista kuitenkin mainitaan, että niiden taso on korkea [6]. Interbotix-tuoteperheen analysointi osoitti, että samaan hintaluokkaan kuuluvien robottikäsien tarkkuus vaihtelee 5 mm:stä 2,5 mm:iin. On kuitenkin huomioitavaa, että vertailun kohteena olleiden robottien käsittelykyky oli huomattavasti pienempi, sekä robottien osat olivat uudempia verrattuna tutkittavana olevaan robottiin. WidowX:n sekä vertailun kohteena olleiden robottien ominaisuudet löytyvät taulukosta 1. Lisäksi taulukosta huomataan, että moottoreiden määrä ei kerro suoraan vapausasteiden määrää, vaan moottoreita voidaan käyttää myös rinnakkain. [8]

Taulukko 1. *WidowX:n vertailu tuoteperheen muihin samankaltaisiin roboteihin.*

Malli	WidowX	ReactorX 200	ReactorX 150	PincherX150
Ulottuma	37 cm	55 cm	45 cm	45 cm
Kiertokulma	360°	360°	360°	360°
Tarkkuus	-	2,5 mm	2,5 mm	5,0 mm
Käsittelykyky	500 g	150 g	100 g	50 g
Servojen lkm.	6	7	6	8
Vapausasteita	5	5	5	5

Perusrakenteeltaan robotti kuuluu kiertonivelisiin robotteihin. Robotin kaikki nivelet ovat siis kiertoniveliä. Tällainen rakenne muistuttaa hyvin paljon ihmisen kättä. Robotti kykenee yksinkertaisiin ihmiskäden kaltaisiin toimintoihin. Se pystyy tarttumaan kappaleisiin ja simuloimaan ranteen kiertoa, sekä liikeradoiltaan on hyvin samanlainen ihmiskäden kanssa.

Robotin toimintaa ohjaa Arbotix Robocontroller – niminen mikrokontrolleri. Mikrokontrolleri mahdollistaa useiden digitaalisten ja analogisten ulkoisten sensoreiden liittämisen osaksi robottia. Adruino-soveltuvuus mahdollistaa myös hyvin matalatasoisen ohjelmoinnin, mutta tässä työssä käytetään kuitenkin eri ohjelmointiympäristöä. Mikrokontrolleri mahdollistaa niin autonomisen toiminnan tallennetun ohjelman mukaan, kuin toiminnan ohjaamisen tietokoneen avulla USB-liitännän kautta. Robotin kauko-ohjaaminenkin on mahdollista mikrokontrolleria varten kehitetyn ohjaimen avulla. [9]

Robotin liikkumista mahdollistavat DYNAMIXEL-servomoottorit. Robotissa on käytetty kolmenlaisia moottoreita ja niiden tärkeimmät ominaisuudet löytyvät taulukosta 2. Kaikista robotin servoista saa tiedon niiden asennosta, lämpötilasta, sekä tulo- ja kuormajännitteestä. MX-sarjan moottorit toimivat PID-ohjauksella. Puolestaan AX-sarjan moottoreiden säätö tapahtuu niin sanotulla Compliance-ohjauksella, jolloin servo myötää tavoiteasennossa, mikäli siihen kohdistuu tietty voima. Tämä soveltuu hyvin tarttujan ohjaamiseen. Käden liikkeistä vastaavien moottoreiden MX-64 ja MX-28 pienin mahdollinen kiertokulma on 0,088° eli ainakin lähtökohtaisesti moottoreiden resoluutio mahdollistaa hyvinkin pienet liikkeet. [10][11]

Taulukko 2. *Työssä käytetyn robotin servomoottoreiden tärkeimmät ominaisuudet.*

Malli	**MX-64**	**MX-28**	**AX-12A**
Toimintakulma	360°	360°	300°
Asemasensori	Magneettinen enkooderi	Magneettinen enkooderi	Potentiometri
Resoluutio	0,088°	0,088°	0,29°
Compliance/PID	PID	PID	Compliance
Määrä	2	2	2

2.3 Robot Operating System

Pohjimmillaan robotti on tietokone, jolla on kyky analysoida ympäristöään ja olla vuorovaikutuksessa sen kanssa. Havaintojen perusteella robotti kykenee tekemään muutoksia ympäristöönsä ja omaan tilaansa. Esimerkiksi saadessaan tiedon kappaleen koosta ja koordinaateista, robotti voi siirtää kyseisen kappaleen paikasta toiseen.

Koska robotti on tietokone, se vaatii konekielellä kirjoitetun koodin toimiakseen. Näin olleen robotin ohjelmointi on prosessina hyvin samanlainen ohjelmistokehityksen kanssa. Tarvitaan ympäristö, joka tukee yhtä tai useampaa robottialustaa (engl. *robot platform*) sekä kykenee kääntämään ohjelmoijan tuottamaa koodia konekielelle. Robottien ohjelmointiin on kehitetty useita ympäristöjä, jotka sisältävät kehitysprosessia helpottavia ja mahdollistavia työkaluja ja kirjastoja. Robottialustojen sekä sovelluskohteiden variaatio tekee kuitenkin hyvin haastavaksi yleisen, monikäyttöisen ja yksinkertaisen ympäristön luomisen. [12]

Robot Operating System eli lyhyesti ROS on tutkimuksessa käytetty ympäristö. UNIX-pohjaisilla tietokonealustoilla toimiva ROS on avoin, laajan valikoiman sovelluskehyksiä (engl. *framework*) sisältävä systeemi. Se on yhteensopiva useiden robottialustojen kanssa ja sen päämääränä on helpottaa monimutkaisten robottijärjestelmien kehittämistä. ROS:n perustoimintaperiaate perustuu robotin toiminnan hajottamiseen prosesseihin (engl. *node*) sekä näiden prosessien väliseen kommunikointiin. Hajottaminen on myös mahdollista jopa useiden tietokoneiden välillä eli prosessit voivat sijaita jopa fyysisesti eri paikoissa, mikäli käyttötarkoitus sitä vaatii. Prosessi ylläpitää esimerkiksi sensoreiden tai servomoottoreiden dataa, käsittelee robotin tilaa, koordinaatteja tai esimerkiksi liikeratojen suunnittelua. Myös yksittäinen ajettava ohjelma on prosessi. Jako selkeisiin kokonaisuuksiin parantaa projektin selkeyttä ja ylläpitoa, sekä mahdollistaa prosessien uudelleenkäytön esimerkiksi toisella robottialustalla vähentämällä näin työn määrää. [13]

Kun puhutaan robottikehityksestä projektina, puhutaan useista aihekokonaisuuksista ja asiantuntijoista niiden takana. Esimerkiksi robotilla voi olla liikkuva alusta ja lisäksi useita toiminnallisia päätteitä. Robotti voi hyödyntää GPS-navigointia, kameroita, sensoreita ja tarvitsee hyvin koodatun ohjelman pystyäkseen toimimaan autonomisesti ilman ihmisen jatkuvaa valvontaa. Realistisesti ajateltuna, harvalla laitoksella on resurssit kaikkien yllä mainittujen tarpeiden kattamiseksi. Tällaisissa tapauksissa ROS:n avoimuus nousee hyvin tärkeään asemaan, sillä se mahdollistaa jonkun toisen kirjoittaman prosessin tai sovelluskehyksen hyödyntämisen omassa projektissa. ROS:n eräänlaisena tavoitteena onkin kannustaa alan ammattilaisia yhteistyöhön prosessien kehityksen parissa. [13]

Oletuksena ROS tulee ilman graafista käyttöliittymää. Kaikki käskyt, toiminnot ja työkalut aktivoidaan konsolilta. Oletusversio sisältää kuitenkin laajan valikoiman työkaluja muun muassa visualisointiin, testaukseen ja debuggaukseen, prosessien hallintaan, liikeratojen suunnitteluun jne. Lisäksi mainittakoon, että ROS:n prosessit voivat kommunikoida keskenään esimerkiksi RPC-tyylillä eli etäproseduurikutsulla palvelujen (engl. *service*)

välityksellä, asynkronisella datasiirrolla niin kutsutuilla aihealueilla (engl. *topic*) tai hakemalla dataa parametripalvelimelta (engl. *parameter server*). ROS mahdollistaa niin reaaliaikaisen ohjelmoinnin kuin valmiiden ohjelmien siirtämisen alustoille. [13]

Jatkossa keskitytään tutkimuksessa käytettyihin menetelmiin, työkaluihin ja toimintoihin. Tutkimustarkoitukseen kehitetyn ohjelman lähdekoodi löytyy liitteestä 2. Yleisesti ottaen ROS:n yhteen robottiin ja yhteen toimintoon liittyvät prosessit ryhmitellään niin sanottuihin pakkauksiin (engl. *package*). Tarkkuusmittauksia varten oli luotu uusi pakkaus, jonka pohjarakenteeksi otettiin WidowX:n valmistajan luoma testiajopakkaus. Pakkaus sisälsi robotin kannalta tarpeelliset toiminnot kuten robotin käynnistyksen, visualisoinnin sekä simuloinnin.

Visualisointityökalu RViz eli ROS Visualizer simuloi robotin 3D-mallin ruudulle. Robotin ollessa kytkettynä tietokoneeseen (puhutaan online-tilasta), ruudulla näkyvä malli käyttäytyy robotin mukaisesti. Halutessa robotin asentoa voidaan muuttaa manuaalisesti ruudulla näkyvien säätimien avulla tai valitsemalla tietty asento, mikäli sellainen on tallennettuna. Offline-tilassa eli kun robottia ei kytketä tietokoneeseen, RViz simuloi oikean robotin käytöstä ja toimii hyvänä lähtökohtana oman ohjelman testaamiseen. On myös mahdollista simuloida ympäristön elementtejä, kuten liikutettavia kappaleita tai esteitä. Tämä onnistuu niin manuaalisesti koodaamalla, kuin autonomisesti ulkoisten kameroiden tai sensoreiden avulla.

Robotin kinemaattisiin ongelmiin on kehitetty MoveIt-niminen työkalu. MoveIt pohjautuu ROS:n viestintämenetelmiin ja käyttää mm. yllä mainittua RViz-työkalua sekä ROS:n robottitiedostojen formaattia URDF. Lyhyesti selitettynä URDF-tiedosto sisältää robotin toiminnan kannalta olennaisia tietoja, kuten robotin mittoja, nivelien ominaisuuksia jne. Kyseistä aihetta ei kuitenkaan käsitellä sen tarkemmin tässä työssä. MoveIt on siis työkalu liikkeiden suunnitteluun, robotin akseleiden törmäysten tarkasteluun, sekä robotin ja ympäristön kolmiulotteiseen vuorovaikutuksen tarkasteluun. Tutkimuksessa käytetty ohjelma käyttää melko paljon MoveItin kirjastoja robotin liikuttamiseen kolmiulotteisessa koordinaatistossa. [14]

Tarkkuusmittausten ohjelma käyttää asynkronista viestintää. Muun muassa robotin tavoitekoordinaattien asettaminen sekä vertailu käyttää hyväksi ROS:n tf2-nimistä transformointiin tarkoitettua kirjastoa. Koordinaatit haetaan aihealueella, johon julkaistaan robotin geometriaan liittyvää tietoa. Asynkronisuus näkyy mm. siinä, että prosessi hakee viimeisimmän tiedon robotin toiminnallisen päätteen sijainnista eikä tietoa haeta servomoottoreilta välittömästi prosessin pyynnöstä. Koodinaattien päivitys tapahtuu kuitenkin jatkuvasti, joten viimeisin tieto on myös hyvin pitkälle ajantasainen. [13]

2.4 Tarkkuus ja työn matemaattinen tausta

Tutkimuksesta saatava data muodostuu kolmiulotteisten koordinaattien joukosta eli pisteistä, joita tutkimuksessa käytetty robotti saavutti tarkkuusmittauksessa. Tulokset ryhmiteltiin robotille annettujen tavoitepisteiden mukaan. Tarkoituksena oli määrittää robotin tarkkuus saadun datan perusteella.

Robotin tarkkuus koostuu kolmesta osatekijästä. Yksi niistä on avaruudellinen resoluutio. Tässä tutkimuksessa siihen ei varsinaisesti keskitytty, mutta tällainen asia on hyvä ottaa huomioon, kun puhutaan teollisuusrobottien tarkkuuden tasosta. Robotin kaikki potentiaaliset asennot muodostavat tietynlaisen joukon pisteitä. Näiden pisteiden määrä on teoriassa rajallinen, mutta tarpeeksi isolla resoluutiolla asia ei ole silmin havaittavissa. Syynä avaruudelliseen resoluutioon ovat esimerkiksi servomoottoreiden mittauslaitteiston näytteenottonopeus, moottoreiden pienin mahdollinen liike, tai moottoreissa olevien hammasrattaiden väljäliike. [15]

Toinen osatekijä on toistotarkkuus (engl. *repeatability*). Se ilmaisee, kuinka tarkka systeemi on sisäisesti. Toistotarkkuus siis ilmaisee, kuinka täsmällisesti robotti kykenee palaamaan tiettyyn asentoon. Tuloksissa on aina läsnä satunnaisvirhe, mutta merkitystä on hajonnan määrällä. Pienempi hajonta tarkoittaa sisäisesti tarkempaa systeemiä. Toistotarkkuuden arviointia varten on ensin määritettävä koordinaateittain datajoukon keskiarvo. Keskiarvon kaava on

$$\mu = \frac{1}{N} * \sum_{i=1}^{N} x_i, \tag{1}$$

jossa μ on keskiarvo, N on joukon koko sekä x_i on joukon yksittäinen arvo. Määrittämällä datajoukon keskiarvo x-, y- ja z-koordinaateille saadaan joukon keskipiste. Vertailemalla keskipisteen ja saavutettujen pisteiden etäisyyksiä, saadaan hajonta. Pisteen etäisyys keskipisteestä lasketaan kaavalla

$$l_i = \sqrt{(X_s - \bar{X})^2 + (Y_s - \bar{Y})^2 + (Z_s - \bar{Z})^2}, \tag{2}$$

jossa l_i on etäisyys, (X_s, Y_s, Z_s) ovat saavutetun pisteen koordinaatit ja $(\bar{X}, \bar{Y}, \bar{Z})$ ovat keskipisteen koordinaatit. Etäisyyksistä lasketaan myös keskiarvo kaavalla (1). Tämän jälkeen voidaan määrittää joukon otoskeskihajonta kaavalla

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N} (l_i - \bar{l})^2}{N - 1}}, \tag{3}$$

jossa σ on otoskeskihajonta, l_i on kaavalla (2) saatu pisteen etäisyys ja $\bar{l}$ etäisyyksien keskiarvo, sekä N on joukon koko. Oletettavasti datapisteet ovat normaalijakautuneita, jolloin toistotarkkuus voidaan määrittää kaavalla

$$RP = \bar{l} + 3 * \sigma, \tag{4}$$

jossa RP on toistotarkkuus, $\bar{l}$ on keskietäisyys ja σ on keskihajonta. Kolmenkertainen keskihajonta kattaa tarkkuusvälin, jolla robotti löytää asennon 99,7%:ssa tapauksista. [5]

Kolmas ja viimeinen tarkkuuden tekijä on robotin paikoitustarkkuus (engl. *accuracy*), joka kertoo systemaattisen virheen esiintyvyydestä. Tässä tapauksessa analysoidaan, kuinka lähelle tavoitepistettä robotti pääsee. Tavoitepistettä vertaillaan datajoukon keskipisteeseen, joka määritettiin x-, y- ja z-koordinaateille kaavalla (1). Systemaattinen poikkeama on keskipisteen ja tavoitepisteen välinen etäisyys, joka voidaan määrittää kaavalla (2). Tässäkin tapauksessa, mitä pienempi poikkeama on, sitä parempi on tarkkuus. [15]

Robotille ominaisen laskentatavan myötä koordinaatit määritetään kohdassa, johon käsivarsi loppuu ja josta työkalu, eli tarttuja, alkaa. Näin olleen mahdollinen systemaattinen kulmapoikkeamasta aiheutuva virhe robotin koordinaateissa on suurempi leukojen kohdalla kuin käsivarren päässä. Kyseistä poikkeama ei kuitenkaan erikseen määritetä tässä työssä.

3. TUTKIMUSMENETELMÄT

Tässä luvussa käsitellään menetelmiä, joilla määritettiin WidowX Robot Arm – robottikäsivarren tarkkuus. Aluksi kerrotaan tutkimuksen toteutuksesta yleisesti, sekä tekijöistä, jotka vaikuttivat menetelmän valintaan. Sitten käydään läpi tulosten käsittelyprosessia ja lopuksi esitetään muutama arvio mittaustuloksiin liittyen.

3.1 Tutkimusjärjestelyt

Tutkimuksessa robotti oli oletettu jäykkärakenteiseksi eli muodonmuutoksia akseleissa ei otettu huomioon. Lisäksi asema-antureista saatava tieto ajateltiin hyvin tarkkana eikä antureiden tarkkuutta otettu tuloksissa huomioon.

Tutkimusta varten oli valittu joukko asentoja, tai tarkemmin päätepisteitä, joihin robotin oli tarkoitus osoittaa. Päätepisteiden suunnittelussa pyrittiin ottamaan huomioon kääntymiskulman, päätepisteiden välisen etäisyyden, sekä alustan ja toimilaitteen välisen etäisyyden vaikutusta tarkkuuteen. Pisteitä lähestyttiin kuitenkin niin, että robotin tarttuja oli vaakatasossa. Robotti ohjelmoitiin suunnittelemaan itse reitti mahdollisimman lähelle päätepistettä määritetyllä toleranssilla. Kaikki päätepisteet olivat sallitulla alueella ja oli ennalta tarkastettu, että robotti ylettää niihin.

Tarkkuutta mitattiin iteratiivisesti. Robotti ohjelmoitiin liikkumaan kolmiulotteisesta pisteestä toiseen ja saavutettuaan piste otettiin robotin koordinaatit talteen. Yksi näytteenottokierros sisälsi jokaisen pisteen täsmälleen kerran ja pisteet käytiin läpi aina samassa järjestyksessä. Saadakseen enemmän vertailukelpoista dataa, robotti toisti kyseisen sekvenssin 10 kertaa.

Ensimmäisen kierroksen aikana ohjelma varmisti robotin liikuttamisen käyttäjältä, mahdollistaakseen myös tarkkuusmittauksen käsin. ROS tulkitsee koordinaatteja metreissä ja tutkimuksessa käytetyn robotin avaruuden origo sijaitsee pöydän tasolla, täsmälleen alustan keskipisteessä. Koordinaatin ja origon välisen etäisyyden määrittäminen ei siis vaatinut erillisiä toimenpiteitä vaan oli mitattavissa suoraan.

Tutkimuksen alkuperäisen suunnitelman mukaan tarkoituksena oli kiinnittää robotin tarttujaan tussi sekä ohjelmoida robotti jättämään merkkejä millimetripaperille. Kuitenkin testiajon aikana todettiin, että kyseisen menetelmän tuottamaa jälkeä on vaikea analysoida. Tussin tekemät jäljet eli lyhyet viivat osuivat noin 5 neliömillimetrin kokoiselle alueelle ja

täten olivat vaikeasti erotettavissa silmin. Lisäksi kyseinen menetelmä mahdollisti mittaamisen vain x-y – tasossa ja korkeuden z vaihtelu jäisi kokonaan pois tarkastelusta.

Lopullisessa menetelmässä tussin jättämä jälki korvattiin digitaalisella jäljellä. Niin kuin on aiemmin mainittu, robotti on tietyllä tasolla tietoinen omasta tilasta ja tässä tapauksessa hyödyksi voitiin käyttää tieto tarttujan kolmiulotteisesta sijainnista robotin työavaruudessa. Tällä menetelmällä tarkkuutta mitattiin epäsuorasti, mutta servomoottoreiden mittauslaitteiston avulla tuottamat koordinaattiarvot olivat hyvin vertailukelpoisia alkuperäiseen tavoitepisteeseen. Lisäksi tällä menetelmällä pystyttiin hyödyntämään koko robotin avaruutta pisteiden valinnassa.

3.2 Paikoitus- ja toistotarkkuuden määrittäminen

Tarkkuusmittauksesta vastaava ohjelma tallensi digitaaliset jäljet erilliseen tekstitiedostoon ryhmitellen jäljet kierroksittain luettavuuden parantamiseksi. Jatkoanalysointia varten data siirrettiin Microsoft Exceliin.

Data koostui saavutettujen pisteiden x-, y- ja z-koordinaateista. Koordinaatit oli ryhmitelty kierroksittain tavoitepisteen mukaan. Saatiin siis 6 datajoukkoa, joissa jokaisessa oli 10 pistealkiota analysoitavana. Laskenta suoritettiin jokaiselle datajoukolle erikseen. Ensimmäisena määritettiin joukkojen keskipisteet, sillä keskipisteen koordinaatteja tarvittiin niin toisto-, kuin paikoitustarkkuuden määrittämisessä.

Perusajatuksena on, että keskipisteen ympärillä on normaalijakautunut satunnaisvirheen aiheuttama perusjoukko pisteitä. Mittaustulokset muodostavat otoksen kyseisestä perusjoukosta ja jokainen saavutettu piste puolestaan osuu tietyn matkan päähän keskipisteestä. Määrittämällä saavutetun pisteen ja keskipisteen välinen etäisyys jokaiselle otoksen pisteelle saadaan joukko etäisyyksiä. Näiden etäisyyksien keskiarvon ja otoskeskihajonnan avulla saadaan määritettyä keskimääräinen poikkeama keskipisteestä. Koska perusjoukko on normaalijakautunut, otoskeskihajonta σ kerrottuna 3:lla antaa alueen, jolle alkiot sijoittuvat 99,7% ajasta.

Paikoitustarkkuuden arviointiin käytettiin keskipisteen ja tavoitepisteen välistä etäisyyttä. Mikäli keskipiste poikkea tavoitepisteestä, robotin toiminnassa on jokin jatkuva virhetekijä. Määritetyissä arvoissa voi esiintyä pientä hajontaa, mutta olemme kiinnostuneita poikkeamien maksimiarvosta niin paikoitus- kuin toistotarkkuuden kohdalla.

ROS laskee reitin haluttuun pisteeseen robotin kinematiikan mukaan. Reitin suunnitteluvaiheessa sallitaan tietty poikkeama tavoitteesta suunnittelun onnistumiseksi. Sallittu

poikkeama vaikuttaa luonnollisesti hajontaan. Kokeellisen testaamisen kautta määritettiin pienin mahdollinen poikkeama, jolla reittisuunnittelu olisi luotettava onnistumisen kannalta ja samanaikaisesti tarpeeksi tarkka.

Suunnittelutoleransseiksi asetettiin ±0,02 radiaania kulmapoikkeamalle niveltä kohden ja ±0,001 metriä kokonaisuudessaan poikkeamalle annetusta pisteestä. Näissä rajoissa robotin on siis teoriassa löydettävä jokin sallittu asento. Käytännössä yllämainittujen lisäksi tulokseen vaikuttavat myös robotin ominaisuudet, joten lopullisen tarkkuuden oletettiin olevan noin 3,5 – 6,0 mm.

4. TARKKUUSMITTAUKSEN SUORITTAMINEN

Ennen mittausta tehdyn havainnon mukaan, robotti oli perusasennossa hieman kallellaan. Silmämääräisesti arvioituna toinen ja kolmas servomoottori eivät olleet ihan keskiasennossa.

Ohjelman ensimmäisellä kierroksella, manuaalisia mittauksia suorittaessa todettiin, että robotin paikannustieto todella kohdistuu pisteeseen, josta tarttuja alkaa. Toisin sanoin kyseinen kohta on 5:nen servomoottorin kiinnitysalustan keskipisteessä eli poikkeaa noin 5 senttimetrin verran tarttujan syvemmästä kohdasta.

Robotin viidennen ja kuudennen servomoottorin välisen akselin asento ei pysynyt ihan kohtisuorassa vaan tarttujan puolelta oli hieman matalampi. Tämä saattoi vaikuttaa mittaustuloksiin ja on todennäköisesti seurausta samasta ilmiöstä, mikä aiemmin huomattiin.

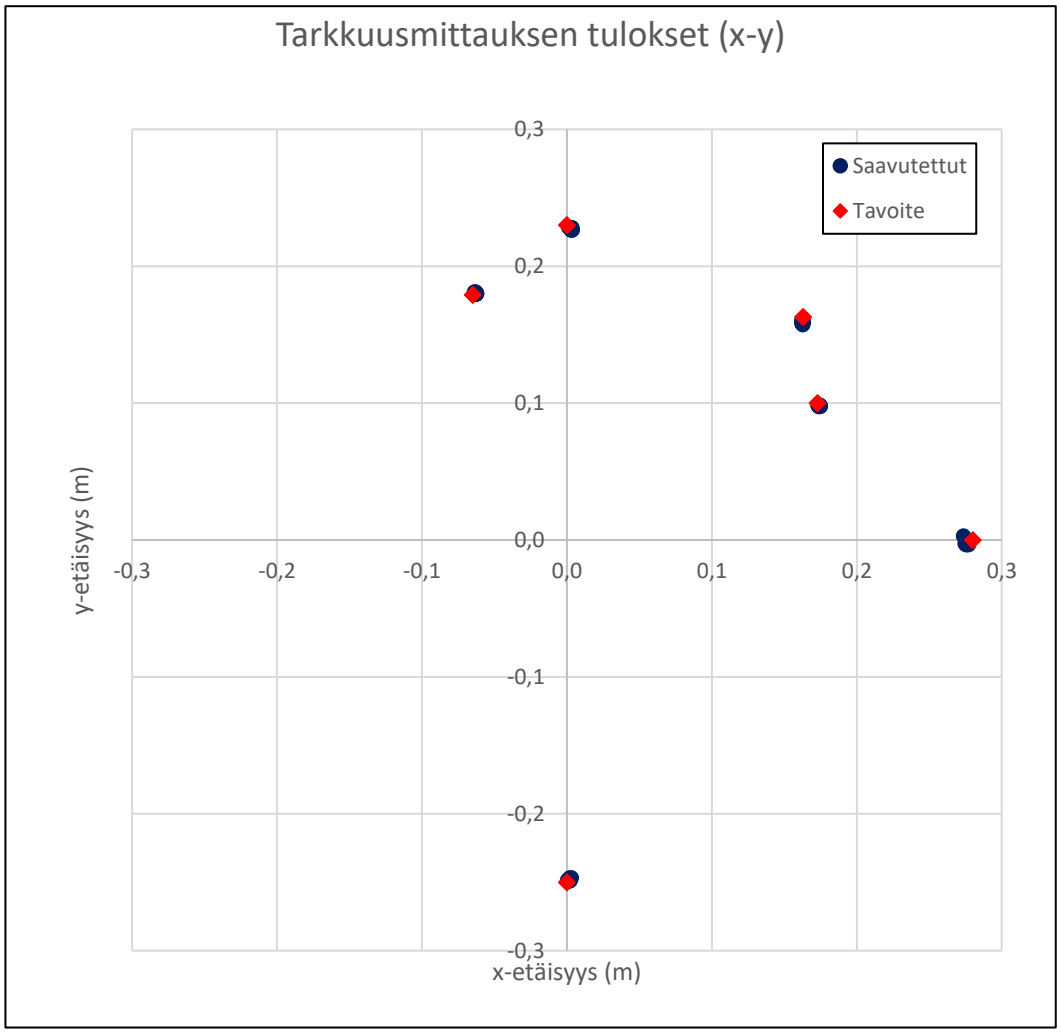
Robotin tavoitepisteiden koordinaatit löytyvät taulukosta 3. Taulukon pisteet numeroidaan niiden esiintymisjärjestyksen mukaan ja jatkossa pisteet nimetään järjestystä vastaavalla numerolla.

Taulukko 3. *Tavoitepisteiden koordinaatit numeroituna.*

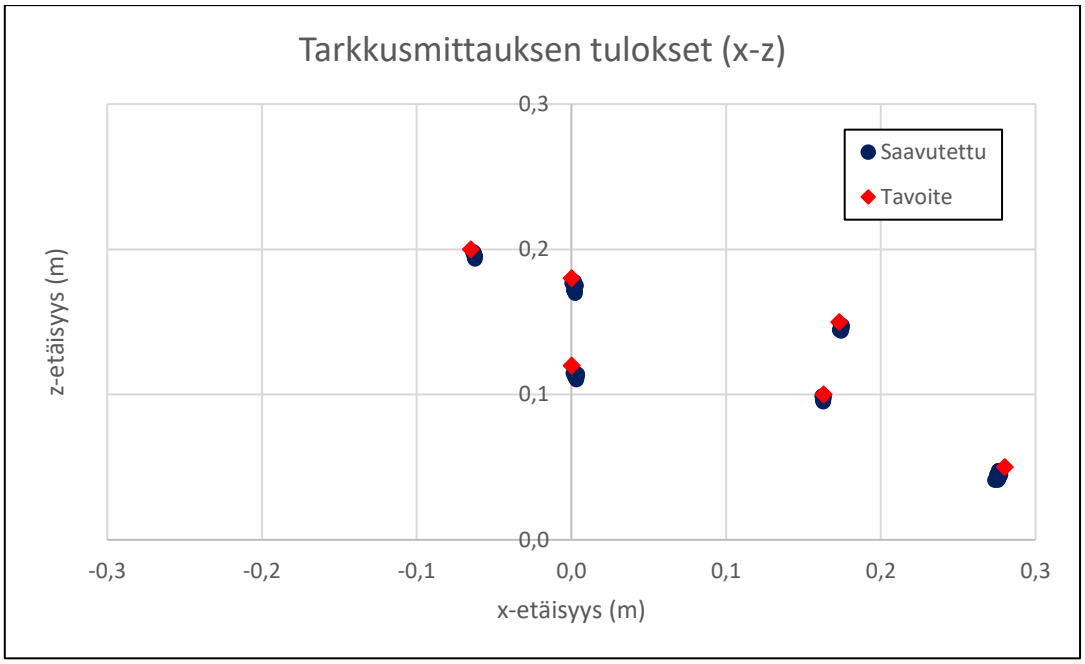
Piste	Koordinaatit		
	x (m)	y (m)	z (m)
1	0,280	0,000	0,050
2	0,173	0,100	0,150
3	0,163	0,163	0,100
4	0,000	0,230	0,120
5	-0,065	0,179	0,200
6	0,000	-0,250	0,180

Pisteiden kolmiulotteisen luonteen takia tarkkuusmittauksen tulosten yhteenveto on laitettu kahteen kuvaajaan. Kuvassa 2 tulokset näkyvät ylhäältäpäin eli x-y - suunnassa ja kuvassa 3 puolestaan sivustapäin x-z - suunnassa. Koordinaatisto vastaa robotin koordinaatistoa siten, että z-akseli kulkee robotin tornin keskipisteen läpi.

Datamäärän vuoksi, mitatut pisteiden koordinaatit löytyvät liitteestä 1. Johtuen saavutettujen pisteiden pienestä hajonnasta, kuvien 2 ja 3 mittasuhteessa saavutetut pisteet menevät lähes kokonaan päällekkäin. Tuloksista kuitenkin huomaa pienen poikkeavuuden tavoitearvosta. Lisäksi huomataan, että joukko pyrkii tavoitteesta poispäin aina tulosuuntaan ja pidemmät matkat aiheuttavat suuremman poikkeaman z-koordinaatissa. Näiden kuvien tarkoituksena on näyttää suuntaa antava pisteiden jakautuminen.



Kuva 2. *Yhteenveto tarkkuusmittauksen tuloksista ylhäältäpäin katsottuna. Origo on robotin alustan keskipisteessä.*



***Kuva 3**. Yhteenveto tarkkuusmittauksen tuloksista sivusta katsottuna. Z-akseli kulkee robotin alustan keskipisteen kautta.*

Liitteessä 1 olevat tulokset ovat taulukoitu tavoitepisteiden 1-6 mukaisesti. Jokaiselle tavoitepisteelle on myös luotu kuvaaja resoluutiolla 10 mm x 10 mm, joka näyttää paljon tarkemmin saavutettujen pisteiden hajonnan.

5. TARKKUUDEN MÄÄRITTÄMINEN

Yksi mittaustulos hylättiin karkeana virheenä. Karkea virhe tuli 1. tavoitepisteen kohdalla 6. kierroksella. Laskujen kannalta oli tarpeeksi dataa, joten virheellä ei ollut suurempaa vaikutusta.

Jokaiselle datajoukolle määritettiin keskipiste käyttäen kaavaa (1). Saadut arvot löytyvät taulukosta 4. Arvoista huomataan poikkeavuus taulukon 3 arvoista eli tuloksista löytyy systemaattinen virhe.

Taulukko 4. *Yhteenveto datajoukkojen keskipisteistä. Vrt. taulukkoon 3.*

Piste	Koordinaatit		
	x (m)	y (m)	z (m)
1	0,276	-0,002	0,045
2	0,174	0,098	0,146
3	0,163	0,159	0,097
4	0,003	0,227	0,112
5	-0,063	0,180	0,195
6	0,002	-0,248	0,174

Toistotarkkuuden arviointi päätettiin jakaa kahteen tapaukseen huomattavan suuren z-suuntaisten arvojen poikkeavuuden takia. Ensimmäisessä tapauksessa huomioon otettiin vain x-y – suuntaiset koordinaatit. Toisessa puolestaan otettiin huomioon myös z-suuntainen koordinaatti. Kummassakin tapauksessa jokaiselle saavutetulle pisteelle määritettiin etäisyys joukon keskipisteestä kaavalla (2). Saaduille tuloksille määritettiin keskiarvo kaavalla (1) sekä hajonta kaavalla (3). Pistekohtaiset toistotarkkuudet laskettiin puolestaan kaavalla (4). Ensimmäisen tapauksen tulokset löytyvät taulukosta 5 ja puolestaan toisen tapauksen tulokset taulukosta 6. Kumpaakin taulukkoa luetaan samalla tavalla.

Taulukko 5. *Saavutettujen pisteiden etäisyydet keskipisteistä, etäisyyksien keskiarvot, hajonnat sekä toistotarkkuudet tavoitepisteittäin x-y - suunnassa.*

Kierros	l_1 (m)	l_2 (m)	l_3 (m)	l_4 (m)	l_5 (m)	l_6 (m)
1	0,0016	0,0006	0,0003	0,0024	0,0011	0,0004
2	0,0013	0,0005	0,0006	0,0007	0,0015	0,0006
3	0,0004	0,0003	0,0019	0,0017	0,0014	0,0012
4	0,0009	0,0006	0,0015	0,0003	0,0004	0,0005
5	0,0005	0,0005	0,0006	0,0003	0,0006	0,0017
6	-	0,0003	0,0005	0,0012	0,0001	0,0010
7	0,0017	0,0006	0,0002	0,0008	0,0003	0,0014
8	0,0003	0,0009	0,0008	0,0011	0,0006	0,0010
9	0,0010	0,0011	0,0012	0,0003	0,0006	0,0005
10	0,0012	0,0005	0,0005	0,0012	0,0004	0,0011
μ (m)	0,0010	0,0006	0,0008	0,0010	0,0007	0,0010
σ (m)	0,0005	0,0002	0,0006	0,0007	0,0005	0,0004
μ+3σ (m)	0,0025	0,0013	0,0025	0,0030	0,0021	0,0022

Taulukko 6. *Saavutettujen pisteiden etäisyydet keskipisteistä, hajonnat sekä toistotarkkuudet avaruudessa. Vrt. taulukko 5.*

Kierros	l_1 (m)	l_2 (m)	l_3 (m)	l_4 (m)	l_5 (m)	l_6 (m)
1	0,0016	0,0009	0,0017	0,0035	0,0011	0,0009
2	0,0028	0,0012	0,0009	0,0011	0,0032	0,0046
3	0,0028	0,0014	0,0027	0,0039	0,0020	0,0027
4	0,0017	0,0023	0,0025	0,0003	0,0008	0,0028
5	0,0016	0,0007	0,0020	0,0003	0,0017	0,0030
6	-	0,0003	0,0014	0,0023	0,0005	0,0032
7	0,0032	0,0006	0,0013	0,0009	0,0021	0,0016
8	0,0024	0,0017	0,0008	0,0022	0,0025	0,0033
9	0,0039	0,0018	0,0019	0,0008	0,0017	0,0023
10	0,0012	0,0021	0,0010	0,0012	0,0004	0,0017
μ (m)	0,0024	0,0013	0,0016	0,0017	0,0016	0,0026
σ (m)	0,0009	0,0007	0,0006	0,0013	0,0009	0,0011
μ+3σ (m)	0,0051	0,0033	0,0035	0,0055	0,0043	0,0058

Systemaattista virhettä tarkasteltiin tarkemmin määrittämällä keskipisteen poikkeama tavoitteesta. Taulukossa 7 esitetään siis paikoitustarkkuuden arvot niin x-y – suunnassa kuin koko avaruudessa.

Taulukko 7. *Keskipisteiden poikkeamat tavoitteesta kaksi- ja kolmiulotteisessa tapauksessa.*

Piste	Etäisyys (m)	
	$S_{kok.}$	S_{x-y}
1	0,0071	0,0048
2	0,0046	0,0022
3	0,0052	0,0042
4	0,0089	0,0040
5	0,0053	0,0026
6	0,0065	0,0030

6. YHTEENVETO JA PÄÄTELMÄT

Työn tarkoituksena oli määrittää WidowX Robot Arm – robotin paikoitus- ja toistotarkkuus, sekä perehtyä robotin ohjelmointiin ROS-ympäristössä. Näihin tavoitteisiin päästiin. Yhteenveto tarkkuusmittauksen tuloksista löytyy taulukosta 8.

Taulukko 8. *Yhteenveto tarkkuusmittauksen tuloksista. Tarkkuus tarkoittaa maksimipoikkeamaa toiminnan aikana.*

Toistotarkkuus (m)		Paikoitustarkkuus (m)	
x-y	kok.	x-y	kok.
0,003	0,006	0,005	0,009

Robotin tarkkuutta määritellessä päädyttiin tulosten jakamiseen kahteen tapaukseen. Tulos oli selvästi tarkempi, kun z-suuntaista koordinaattia ei otettu huomioon. Tässä tapauksessa saadut tulokset vastaavat hyvinkin aiemmin mainittua arviota. Kappaleiden siirrossa x-y – suuntaisilla koordinaateilla on enemmän merkitystä, sillä kappale mitä todennäköisemmin sijoitetaan jonkinlaiselle kiinteälle alustalle. 3 millimetrin toistotarkkuus on tämän tasoiselle robotille oikein hyvä tulos. Toisaalta taas, robotti toimii kolmiulotteisessa työympäristössä ja tehtävät saattavat vaatia myös tämänhetkistä tarkempaa kokonaistarkkuutta. Tulos kokonaistarkkuudelle sellaisenaan esittää teoreettisen pallomaisen alueen. Todellisuudessa kuitenkin jakauma on enemmänkin sellaisen ellipsoidin muotoinen, jonka puoliakselien mitat ovat x-y – suunnassa 3 mm ja z – suunnassa 6 mm.

Saadut tulokset voivat viestiä servomoottoreiden kalibroinnin tarpeesta. Niin kuin on huomattu mittauksien alkuvaiheessa, robotti on valmiiksi hieman kallellaan, kun se käsketään perusasentoon. Perusasennossa kaikkien servomoottoreiden kuuluisi olla keskiasennossa ja poikkeama siitä kertoo moottorin epäkeskisyydestä. Servomoottoreiden epäkeskinen asento saattaisi myös selittää, miksi pienempi kulmapoikkeaman suunnittelutoleranssi oli mahdollinen simulaatiossa mutta ei oikeassa ajossa. Suunnittelutoleranssia täytyi kasvattaa kaksinkertaiseksi liikesuunnittelun onnistumiseksi.

ROS osoittautui hyvin kattavaksi ja laajaksi ympäristöksi. Sen käyttö antaa hyvin laajat mahdollisuudet robotin suunnitteluun ja kehittämiseen ja kaiken lisäksi robotin vaihtuminen ei vaikuta juuri ollenkaan ohjelmointiprosessin. Kääntöpuolena on tietenkin se, että yksittäistä robottia ei pysty ohjelmoimaan matalalla tasolla. WidowX:n kohdalla se kostautui lähinnä koordinaatistossa. ROS:n kirjastot mahdollistavat helppokäyttöisen asentojen suunnittelun roboteille, joilla on 6 vapausastetta. Tutkimuksessa käytetyllä robotilla

vapausasteita oli vain 5, joten jokaisessa asennossa piti asettaa tarkka orientaatio suunnittelun onnistumiseksi. WidowX:n tapauksessa esimerkiksi sylinterikoordinaatisto olisi helpompi, jossa olisi x- ja y-koordinaatit käden asennon määrittämiseen ja α-kulma ensimmäisen nivelen asennon määrittämiseen alustan nähden.

LÄHTEET

[1] Robotics in Healthcare - Get Ready!, The Medical Futurist, 2016, Saatavissa: https://medicalfuturist.com/robotics-healthcare (viitattu 5.2019).

[2] Spyros G. Tzafestas, Introduction to Mobile Robot Control, Elsevier, 2013.

[3] Reza N. Jazar, Theory of Applied Robotics: Kinematics, Dynamics, and Control (2nd Edition), Spronger, 2007, s. 1-22.

[4] ABB Robotics, Robotics product range, Saatavissa: https://search-ext.abb.com/library/Download.aspx?DocumentID=9AKK1074920494&LanguageCode=en&DocumentPartId=&Action=Launch (viitattu 1.9.2019)

[5] Robotic Gripper Repeatability Definition and Measurement, Robotiq, Saatavissa: https://blog.robotiq.com/bid/36551/Robotic-Gripper-Repeatability-Definition-and-Measurement (viitattu 11.9.2019)

[6] WidowX Robot Arm, Trossen Robotix, Saatavissa: https://www.trossenrobotics.com/widowxrobotarm (viitattu 5.2019).

[7] Kuva WidowX-robotista, Trossen Robotix, Saatavissa: https://www.trossenrobotics.com/Shared/Images/Product/WidowX-Robot-Arm-Kit-Mark-II/widowx-a.jpg (viitattu 11.9.2019).

[8] Interbotix, Trossen Robotix, Saatavissa: https://www.trossenrobotics.com/m/interbotix (viitattu 5.2019).

[9] ArbotiX-M Robocontroller, Trossen Robotix, Saatavissa: https://www.trossenrobotics.com/p/arbotix-robot-controller.aspx (viitattu 22.9.2019).

[10] DYNAMIXEL Guide, Trossen Robotix, Saatavissa: https://learn.trossenrobotics.com/projects/159-trossen-robotics-dynamixel-guide.html (viitattu 22.9.2019).

[11] DYNAMIXEL AX-12-servomoottorin dokumantaatio, Robotis, Saatavissa: http://support.robotis.com/en/techsupport_eng.htm#product/dynamixel/ax_series/dxl_ax_actuator.htm (viitattu 22.9.2019).

[12] About ROS, Saatavissa: https://www.ros.org/about-ros/ (viitattu 11.9.2019).

[13] ROS Documentation – Introduction, Saatavissa: http://wiki.ros.org/ROS/Introduction (viitattu 11.9.2019).

[14] Frequently Asked Questions, MoveIt, Saatavissa: https://moveit.ros.org/documentation/faqs/ (viitattu 11.9.2019).

[15] Industrial Robotics, Engineer On A Disc, Saatavissa: http://engineeronadisk.com/V2/book_integration/engineeronadisk-14.html (viitattu 11.9.2019).

LIITE 1 TARKKUUSMITTAUKSEN TULOKSET

Piste 1			
Kierros	**x (m)**	**y (m)**	**z (m)**
1	0,2773	-0,0034	0,0449
2	0,2755	-0,0013	0,0423
3	0,2761	-0,0021	0,0476
4	0,2761	-0,0034	0,0462
5	0,2758	-0,0030	0,0463
6	0,2736	0,0029	0,0411
7	0,2761	-0,0009	0,0476
8	0,2763	-0,0025	0,0424
9	0,2755	-0,0034	0,0410
10	0,2748	-0,0025	0,0449

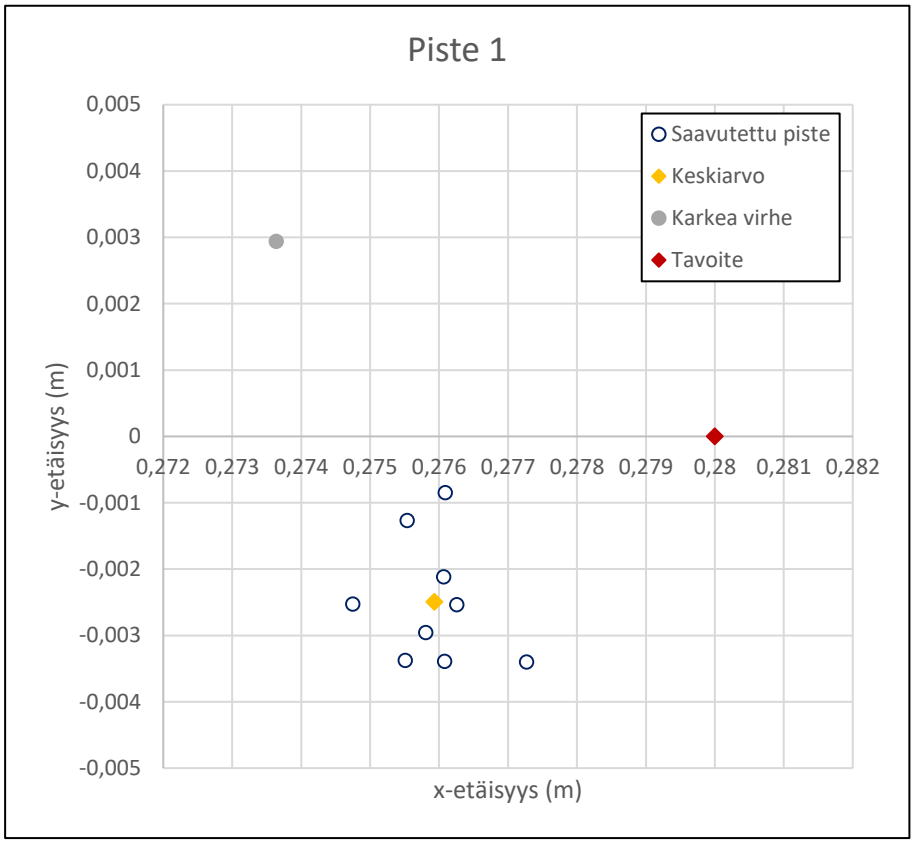
Piste 2			
Kierros	**x (m)**	**y (m)**	**z (m)**
1	0,1747	0,0983	0,1466
2	0,1737	0,0984	0,1449
3	0,1738	0,0982	0,1473
4	0,1743	0,0987	0,1437
5	0,1746	0,0983	0,1455
6	0,1741	0,0978	0,1459
7	0,1741	0,0976	0,1456
8	0,1749	0,0976	0,1474
9	0,1732	0,0987	0,1444
10	0,1738	0,0978	0,1479

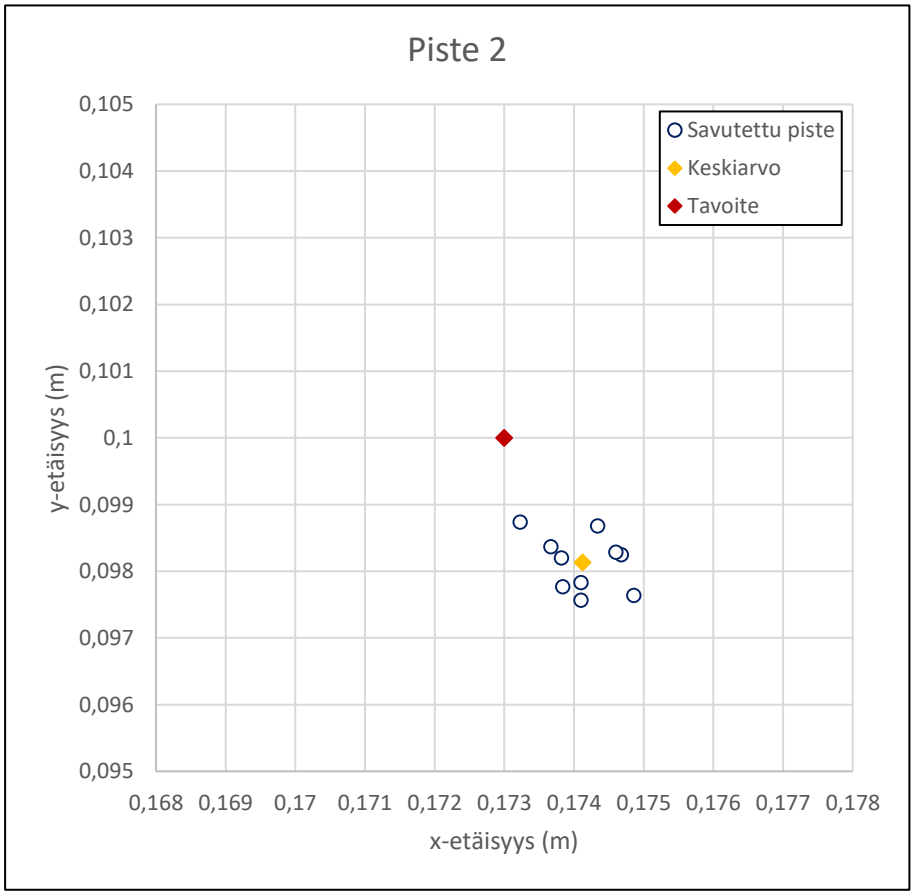
Piste 3			
Kierros	**x (m)**	**y (m)**	**z (m)**
1	0,1625	0,1590	0,0954
2	0,1627	0,1582	0,0964
3	0,1621	0,1607	0,0989
4	0,1627	0,1573	0,0951
5	0,1622	0,1592	0,0990
6	0,1623	0,1584	0,0984
7	0,1627	0,1588	0,0957
8	0,1624	0,1580	0,0973
9	0,1633	0,1598	0,0985
10	0,1629	0,1584	0,0962

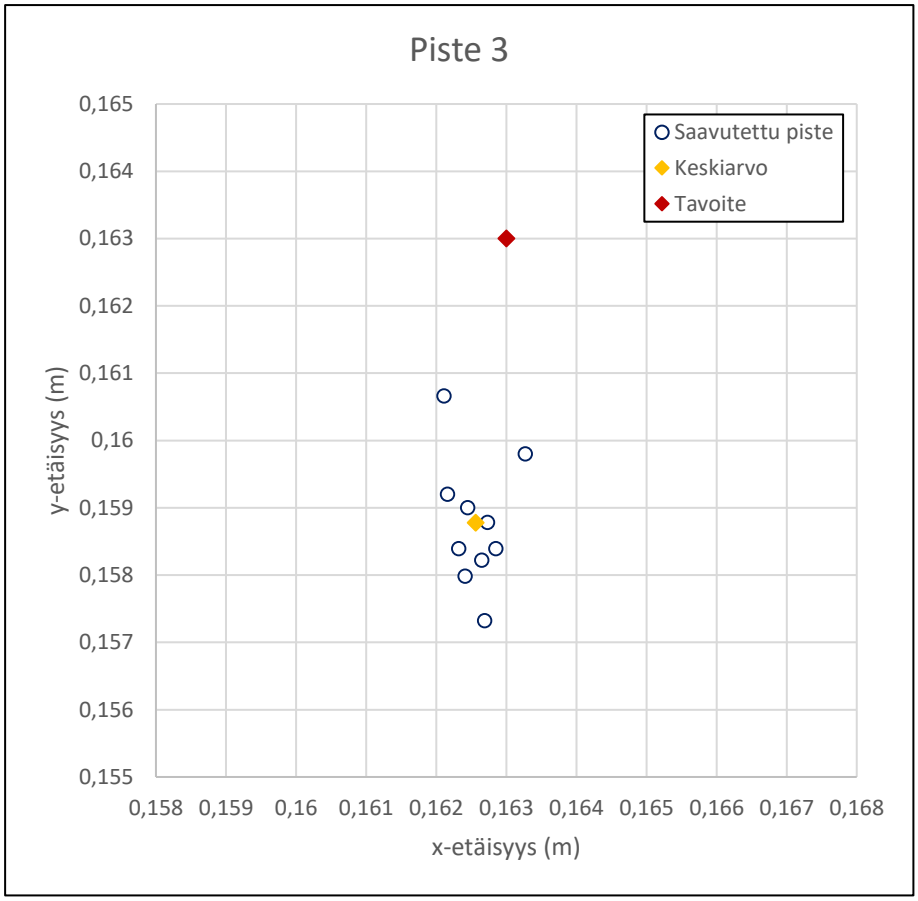
Piste 4			
Kierros	**x (m)**	**y (m)**	**z (m)**
1	0,0011	0,2286	0,1146
2	0,0028	0,2268	0,1130
3	0,0018	0,2285	0,1156
4	0,0031	0,2278	0,1122
5	0,0032	0,2277	0,1121
6	0,0038	0,2265	0,1140
7	0,0035	0,2281	0,1125
8	0,0031	0,2263	0,1102
9	0,0028	0,2275	0,1113
10	0,0021	0,2281	0,1123

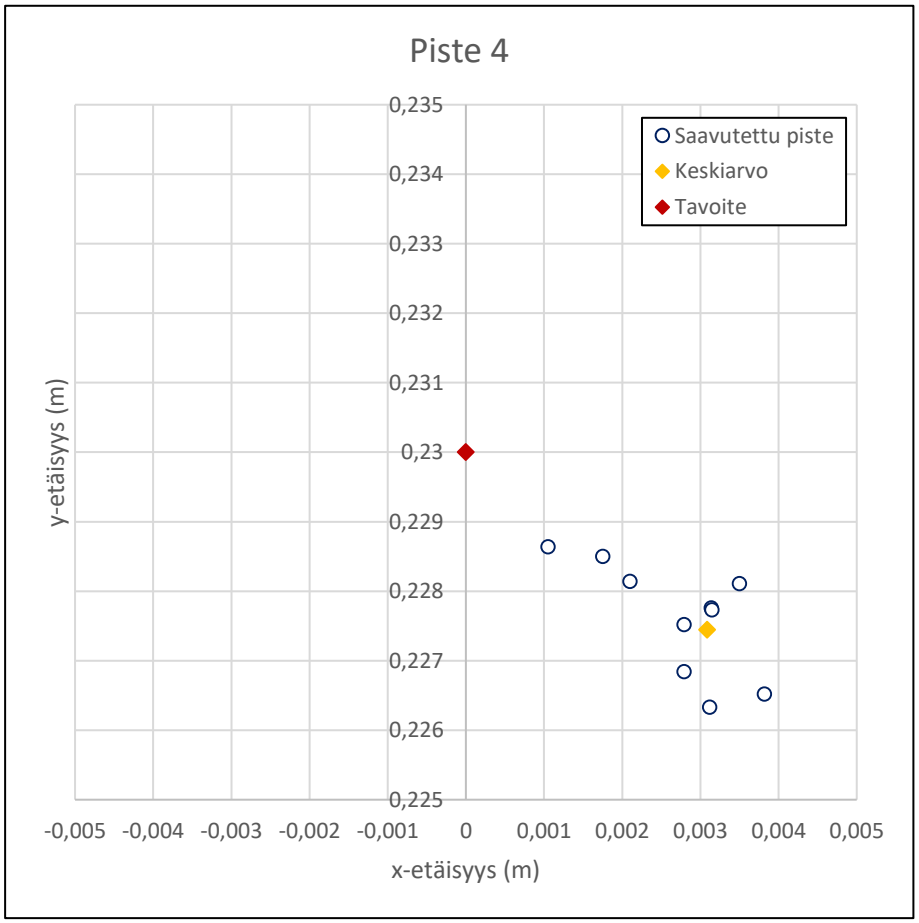
Piste 5			
Kierros	**x (m)**	**y (m)**	**z (m)**
1	-0,0630	0,1812	0,1957
2	-0,0638	0,1811	0,1983
3	-0,0634	0,1813	0,1970
4	-0,0625	0,1798	0,1947
5	-0,0627	0,1795	0,1939
6	-0,0626	0,1801	0,1959
7	-0,0624	0,1803	0,1934
8	-0,0630	0,1807	0,1978
9	-0,0631	0,1806	0,1970
10	-0,0623	0,1801	0,1954

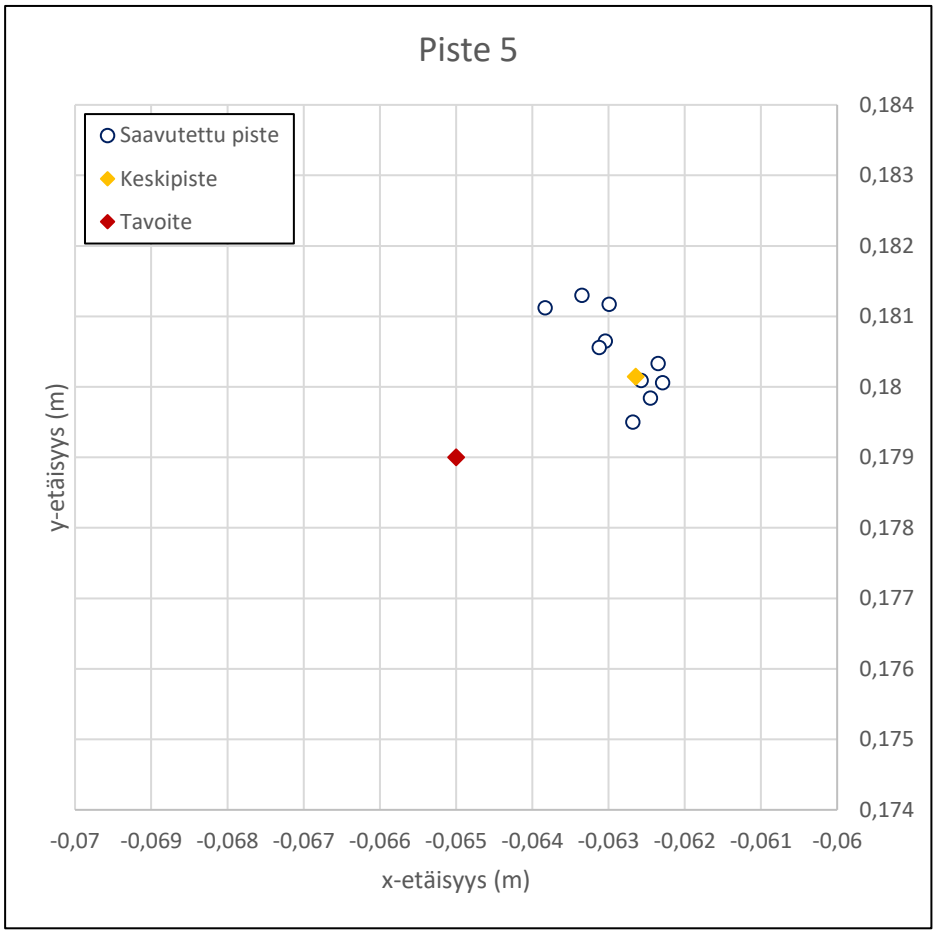
Piste 6			
Kierros	**x (m)**	**y (m)**	**z (m)**
1	0,0023	-0,2474	0,1750
2	0,0023	-0,2483	0,1696
3	0,0015	-0,2489	0,1718
4	0,0023	-0,2483	0,1715
5	0,0004	-0,2483	0,1767
6	0,0019	-0,2488	0,1713
7	0,0030	-0,2468	0,1750
8	0,0019	-0,2488	0,1774
9	0,0015	-0,2479	0,1720
10	0,0019	-0,2467	0,1755

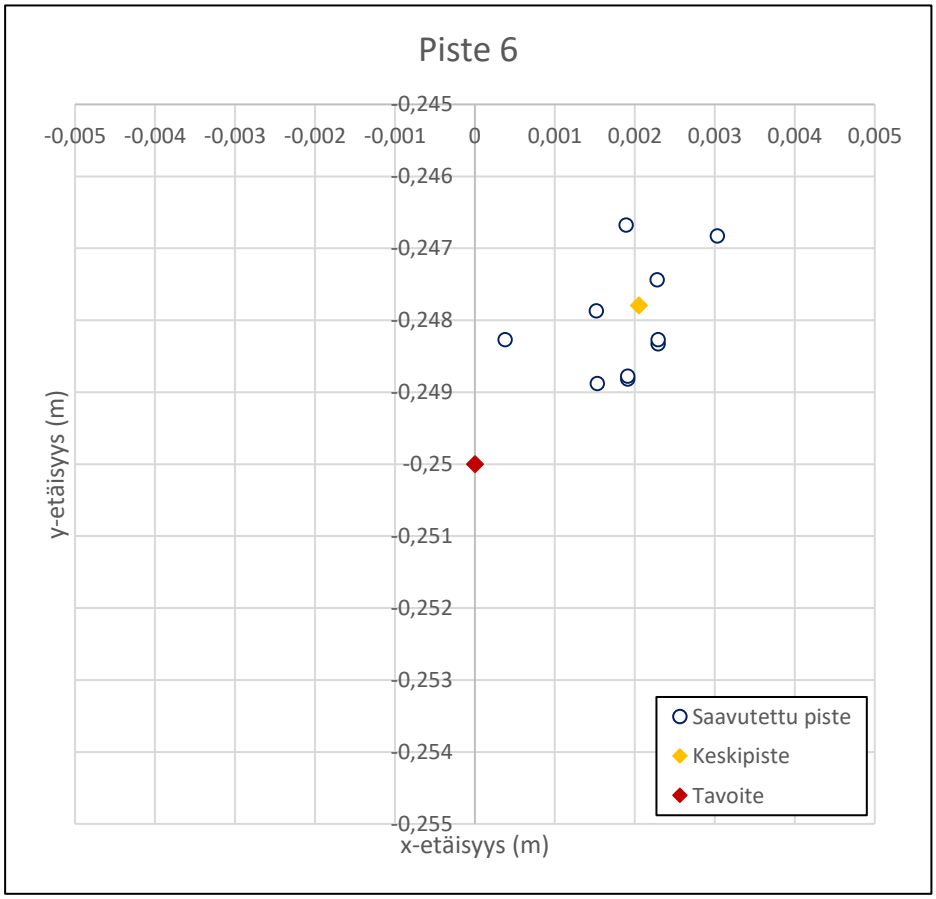

Piste 1
Saavutettu piste
Keskiarvo
Karkea virhe
Tavoite
y-etäisyys (m)
x-etäisyys (m)
0,005
0,004
0,003
0,002
0,001
0
-0,001
-0,002
-0,003
-0,004
-0,005
0,272 0,273 0,274 0,275 0,276 0,277 0,278 0,279 0,28 0,281 0,282


Piste 2
Savutettu piste
Keskiarvo
Tavoite
y-etäisyys (m)
x-etäisyys (m)
0,105
0,104
0,103
0,102
0,101
0,1
0,099
0,098
0,097
0,096
0,095
0,168 0,169 0,17 0,171 0,172 0,173 0,174 0,175 0,176 0,177 0,178


Piste 3
Saavutettu piste
Keskiarvo
Tavoite
y-etäisyys (m)
x-etäisyys (m)
0,165
0,164
0,163
0,162
0,161
0,16
0,159
0,158
0,157
0,156
0,155
0,158 0,159 0,16 0,161 0,162 0,163 0,164 0,165 0,166 0,167 0,168


Piste 4
Saavutettu piste
Keskiarvo
Tavoite
y-etäisyys (m)
x-etäisyys (m)
0,235
0,234
0,233
0,232
0,231
0,23
0,229
0,228
0,227
0,226
0,225
-0,005 -0,004 -0,003 -0,002 -0,001 0 0,001 0,002 0,003 0,004 0,005


Piste 5
Saavutettu piste
Keskipiste
Tavoite
y-etäisyys (m)
x-etäisyys (m)
0,184
0,183
0,182
0,181
0,18
0,179
0,178
0,177
0,176
0,175
0,174
-0,07
-0,069
-0,068
-0,067
-0,066
-0,065
-0,064
-0,063
-0,062
-0,061
-0,06


Piste 6
Saavutettu piste
Keskipiste
Tavoite
y-etäisyys (m)
x-etäisyys (m)
-0,005
-0,004
-0,003
-0,002
-0,001
0
0,001
0,002
0,003
0,004
0,005
-0,245
-0,246
-0,247
-0,248
-0,249
-0,25
-0,251
-0,252
-0,253
-0,254
-0,255

LIITE 2 MITTAUKSESSA KÄYTETTY KOODI

```
1.  #include <ros/ros.h>
2.  #include <tf/tf.h>
3.
4.  // MoveIt!
5.  #include <moveit/planning_scene_interface/planning_scene_interface.h>
6.  #include <moveit/move_group_interface/move_group_interface.h>
7.
8.  // TF2
9.  #include <tf2_geometry_msgs/tf2_geometry_msgs.h>
10.
11. // other
12. #include <fstream>
13.
14. // TODO: rewrite main function
15. // TODO: writing coords into separate file
16. // TODO: rewrite poses for sequence
17.
18. // Moves arm at 3rd joint
19. void liftArm(moveit::planning_interface::MoveGroupInterface &arm, double angle)
    {
20.   const robot_state::JointModelGroup* joint_model_group = arm.getCurrentState()
    ->getJointModelGroup("widowx_arm");
21.
22.   moveit::core::RobotStatePtr current_state = arm.getCurrentState();
23.   std::vector<double> joint_group_positions;
24.   current_state = arm.getCurrentState();
25.   current_state->copyJointGroupPositions(joint_model_group, joint_group_posi-
    tions);
26.
27.   joint_group_positions[3] -= angle;
28.   arm.setJointValueTarget(joint_group_positions);
29.   arm.move();
30. }
31.
32. // Calibrate orientation to one after ground calibration
33. void correctOrientation(moveit::planning_interface::MoveGroupInterface &arm){
34.   double roll, pitch, yaw;
35.
36.   tf2::Quaternion q;
37.   tf2::convert(arm.getCurrentPose().pose.orientation, q);
38.   tf2::Matrix3x3(q).getRPY(roll, pitch, yaw);
39.
40.
41.   liftArm(arm, M_PI/2 - pitch);  // radians
42.
43.   ROS_INFO("Corrected orientation by %0.2f", M_PI/2 - pitch);
44. }
45.
46. // Leaving mark in the target location, including correct approach and re-
    treat motions
47. void createMark(geometry_msgs::Pose target_pose, moveit::planning_interface::Mo
    veGroupInterface &arm, std::ofstream &logFile){
48.
49.   // calculating new orientation based on current pose
50.   double offset = std::atan2(arm.getCurrentPose().pose.position.y, arm.getCur-
    rentPose().pose.position.x);
51.   double y = target_pose.position.y;
52.   double x = target_pose.position.x;
53.
54.   tf2::Quaternion q, q_rot;
```

```
55.   tf2::convert(arm.getCurrentPose().pose.orientation, q);
56.   q_rot.setRPY(0, 0, std::atan2(y, x) - offset);
57.   q = q_rot*q;
58.   q.normalize();
59.
60.   tf2::convert(q, target_pose.orientation);
61.
62.   // crete mark
63.   arm.setPoseTarget(target_pose);
64.   arm.move();
65.
66.   correctOrientation(arm);
67.
68.   // create log entry
69.   logFile << "x: " << arm.getCurrentPose().pose.position.x;
70.   logFile << " y: " << arm.getCurrentPose().pose.position.y;
71.   logFile << " z: " << arm.getCurrentPose().pose.position.z << std::endl;
72.
73.   ROS_INFO("Created digital mark at [%0.5f, %0.5f,, %0.5f]", arm.getCurrent-
    Pose().pose.position.x, arm.getCurrentPose().pose.position.y, arm.getCurrent-
    Pose().pose.position.z);
74. }
75.
76. // Taking pen
77. void placePen(moveit::planning_interface::MoveGroupInter-
    face &arm, moveit::planning_interface::MoveGroupInterface &gripper){
78.
79.   std::string c = "";
80.   std::cout << "Reattach pen? (y/n): ";
81.   std::cin >> c;
82.
83.   if(c == "n"){
84.     return;
85.   }
86.   c = "";
87.
88.   geometry_msgs::Pose pen_pose = arm.getCurrentPose().pose;
89.
90.   pen_pose.position.x = 0.25;
91.   pen_pose.position.y = 0.0;
92.   pen_pose.position.z = 0.05;
93.   arm.setPoseTarget(pen_pose);
94.   arm.move();
95.
96.   correctOrientation(arm);
97.
98.   gripper.setJointValueTarget("gripper_joint", 0.030);
99.   gripper.move();
100.
101.  while(c != "y"){
102.    std::cout << "Pen set? (y/n): ";
103.    c == "";
104.    std::cin >> c;
105.  }
106.
107.  gripper.setJointValueTarget("gripper_joint", 0.010);
108.  gripper.move();
109.
110.  pen_pose.position.z += 0.08;
111.  arm.setPoseTarget(pen_pose);
112.  arm.move();
113.}
114.
115.// Calibrate ground orientation and return calibrated orientation
116.tf2::Quaternion calibrateGround(moveit::planning_interface::MoveGroupInter-
    face &arm){
```

```
117.   tf2::Quaternion q;
118.   geometry_msgs::Pose pose = arm.getCurrentPose().pose;
119.
120.   arm.setNamedTarget("ground_calibration_pose");
121.   arm.move();
122.
123.   tf2::convert(pose.orientation, q);
124.
125.   ROS_INFO("Ground orientation calibrated");
126.
127.   return q;
128. }
129.
130. // Creating target points for precision test
131. std::vector<std::tuple<double, double, double>> createTargetPoints(){
132.
133.   std::vector<std::tuple<double, double, double>> target_points;
134.   std::tuple<double, double, double> coords;
135.
136.   for (int i = 0; i < 6; i++){
137.     if ( i == 0) coords = std::make_tuple(0.28, 0.0, 0.05);
138.     if ( i == 1) coords = std::make_tuple(0.173, 0.10, 0.15);
139.     if ( i == 2) coords = std::make_tuple(0.163, 0.163, 0.10);
140.     if ( i == 3) coords = std::make_tuple(0.0, 0.23, 0.12);
141.     if ( i == 4) coords = std::make_tuple(-0.065, 0.179, 0.20);
142.     if ( i == 5) coords = std::make_tuple(0.0, -0.25, 0.18);
143.
144.     ROS_INFO("Created tar-
    get point [%0.3f, %0.3f, %0.3f]", std::get<0>(coords), std::get<1>(coords), std
    ::get<2>(coords));
145.
146.     target_points.push_back(coords);
147.   }
148.   return target_points;
149. }
150.
151. // Converting target points into target poses
152. std::vector<geometry_msgs::Pose> convertPointsToPoses(std::vector<std::tu-
    ple<double, double, double>> &points){
153.   std::vector<geometry_msgs::Pose> target_poses;
154.
155.   for (std::tuple<double, double, double> coords : points){
156.     geometry_msgs::Pose target_pose;
157.
158.     target_pose.position.x = std::get<0>(coords);
159.     target_pose.position.y = std::get<1>(coords);
160.     target_pose.position.z = std::get<2>(coords);
161.
162.     target_poses.push_back(target_pose);
163.   }
164.   return target_poses;
165. }
166.
167. int main(int argc, char** argv)
168. {
169.   ros::init(argc, argv, "widowx_arm_precision_test");
170.   ros::NodeHandle nh;
171.   ros::AsyncSpinner spinner(1);
172.   spinner.start();
173.
174.   // robot settings
175.   moveit::planning_interface::MoveGroupInterface arm("widowx_arm");
176.   moveit::planning_interface::MoveGroupInterface gripper("widowx_gripper");
177.   arm.setPlannerId("RRTConnectkConfigDefault");
178.   arm.setGoalPositionTolerance(0.001);
179.   arm.setGoalOrientationTolerance(0.02);
```

```
180.   gripper.setMaxVelocityScalingFactor(0.1);
181.   arm.setMaxVelocityScalingFactor(1.0);
182.
183.   // vector for target locations
184.   std::vector<std::tuple<double, double, double>> target_points = createTar-
    getPoints();
185.   std::vector<geometry_msgs::Pose> target_poses = convertPointsToPoses(tar-
    get_points);
186.
187.   arm.setNamedTarget("default_pose");
188.   arm.move();
189.
190.   // log file
191.   std::ofstream logFile;
192.   logFile.open("/home/crimatorre/widowx_arm/src/wid-
    owx_arm_pick_place/data/servo_position_log.txt");
193.   logFile << std::setprecision(5) << std::fixed;
194.
195.   int loops = 1;
196.   std::string s = "";
197.   std::cout << "Enter number of loops to execute: ";
198.   std::cin >> loops;
199.
200.   for (int i = 0; i < loops; i++){
201.
202.     logFile << "Loop " << i+1 << ":" << std::endl;
203.
204.     // measure relative pose at first round
205.     if (i == 0){
206.       for (geometry_msgs::Pose pose : target_poses){
207.         createMark(pose, arm, logFile);
208.     std::cout << "Continue (enter any key): ";
209.     std::cin >> s;
210.     s = "";
211.       }
212.
213.     } else {
214.       for (geometry_msgs::Pose pose : target_poses){
215.         createMark(pose, arm, logFile);
216.       }
217.
218.     }
219.
220.     logFile << std::endl;
221.
222.     ROS_INFO("Sequence progress %i/%i.", i+1, loops);
223.   }
224.   ROS_INFO("Test completed.");
225.
226.   return 0;
227. }
```