TALLINNA TEHNIKAÜLIKOOL
INSENERITEADUSKOND
Elektroenergeetika ja mehhatroonika instituut

TÖÖSTUSLIKE MANIPULAATORITE JA LIIKURROBOTITE SIDUMINE AVATUD LÄHTEKOODIGA TARKVARAPLATVORMIGA ROS

INTEGRATION OF INDUSTRIAL MANIPULATORS AND MOBILE ROBOTS WITH OPEN-SOURCE SOFTWARE PLATFORM ROS

BAKALAUREUSETÖÖ

Üliõpilane: Andre Talvoja
/nimi/

Üliõpilaskood: 211256AAAB

Juhendaja: Madis Lehtla, vanemlektor
/nimi, amet/

Kaasjuhendaja: Tanel Jalakas, vanemteadur
/nimi, amet/

Tallinn 2021

AUTORIDEKLARATSIOON

Olen koostanud lõputöö iseseisvalt.
Lõputöö alusel ei ole varem kutse- või teaduskraadi või inseneridiplomit taotletud.
Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

"..18..." ..mai..... 2021..

Autor: .....
/ allkiri /

Töö vastab bakalaureusetööle esitatud nõuetele

"..18....." .....mai..... 2021…..

Juhendaja: .....
/ allkiri /

Töö vastab bakalaureusetööle esitatud nõuetele

"..18....." .....mai..... 2021…..

Kaasjuhendaja: .....
/ allkiri /

Kaitsmisele lubatud

"....."..... 202… .

Kaitsmiskomisjoni esimees .....
/ nimi ja allkiri /

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina Andre Talvoja

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose **Liikurrobotite ja tööstuslike manipulaatorite sidumine robotite juhtimiseks ettenähtud tarkvaraplatvormiga ROS ning tööks vajaliku juhendmaterjalide koostamine**,

mille juhendaja on Madis Lehtla ja kaasjuhendaja on Tanel Jalakas

1.1 reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

1.2 üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.

3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

_18.05.2021_____ (*kuupäev*)

[1] *Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.*

LÕPUTÖÖ LÜHIKOKKUVÕTE

Autor: Andre Talvoja *Lõputöö liik:* Bakalaureusetöö

Töö pealkiri: Tööstuslike manipulaatorite ja liikurrobotite sidumine avatud lähtekoodiga tarkvaraplatvormiga ROS

Kuupäev: 18.05.2021 65 *lk (lõputöö lehekülgede arv koos lisadega)*

Ülikool: Tallinna Tehnikaülikool

Teaduskond: Inseneriteaduskond

Instituut: Elektroenergeetika ja mehhatroonika instituut

Töö juhendaja(d): Madis Lehtla, Tanel jalakas

Töö konsultant (konsultandid):

Sisu kirjeldus:

Töö eesmärgiks oli koostada juhendid algajaile koos sobivate näidetega tööstuslike manipulaatorite ja liikurrobotite sidumiseks avatud lähtekoodiga tarkvaraplatvormiga ROS. Töö käigus valmis näidis, mida saab õppeprotsessis demonstreerida koos Tallinna Tehnikaülikooli tootmise automatiseerimise õppelaboris paikneva Mitsubishi MELFA tööstusliku manipulaatoriga.

Märksõnad: Haridusrobootika, tööstusrobotid, mobiilsed manipulaatorid, tarkvara, vahevara, programmeerimiskeskkonnad, robotite koostöö, tootmise automatiseerimine, täiturmehhanismid, liigendkäed, liikumise juhtimine, trajektoori planeerimine, simulatsioon, animatsioon, riist-tarkvara integreerimine, kinemaatika, positsioonjuhtimine, ROS, tööstuslik manipulaator, praktikumijuhend.

ABSTRACT

Author: Andre Talvoja *Type of the work:* Bachelor Thesis

Title: Integration of industrial manipulators and mobile robots with open-source software platform ROS

Date: 18.05.2021 65 *pages (the number of thesis pages including appendices)*

University: Tallinn University of Technology

School: School of Engineering

Department: Department of Electrical Power Engineering and Mechatronics

Supervisor(s) of the thesis: Madis Lehtla, Tanel Jalakas

Consultant(s):

Abstract:

The aim of the work was to compile instructions for beginners with suitable examples for connecting industrial manipulators and mobile robots to the open-source software platform ROS. In the course of the work, a sample was prepared, which can be demonstrated in the study process together with the Mitsubishi MELFA industrial manipulator located in the production automation study laboratory of Tallinn University of Technology.

Keywords: Education Robotics, Industrial Robots, Mobile Manipulation, Software, Middleware and Programming Environments, Robot Collaboration, Factory Automation, Actuation Mechanisms, Articulated arms, Motion Control, Path Planning, Simulation, Animation, Hardware-Software Integration, Kinematics, Positioning Control, ROS, Industrial Manipulator, Laboratory Instruction Manual.

LÕPUTÖÖ ÜLESANNE

Lõputöö teema:	**Liikurrobotite ja tööstuslike manipulaatorite sidumine robotite juhtimiseks ettenähtud tarkvaraplatvormiga ROS**
Lõputöö teema inglise keeles:	
Üliõpilane:	**Andre Talvoja, 211256AAAB**
Eriala:	
Lõputöö liik:	**bakalaureusetöö**
Lõputöö juhendaja:	**Madis Lehtla**
Lõputöö kaasjuhendaja: (ettevõte, amet ja kontakt)	**Tanel Jalakas**
Lõputöö ülesande kehtivusaeg:	**2021 kevadsemester**
Lõputöö esitamise tähtaeg:	**18.05.2021**

_____ _____ _____

Üliõpilane (allkiri) Juhendaja (allkiri) Õppekava juht (allkiri)

_____

Kaasjuhendaja (allkiri)

2. Töö eesmärk

Töö eesmärgiks on juhendite koostamine algajaile koos sobiva näitega.

3. Lahendamisele kuuluvate küsimuste loetelu:

Näite koostamiseks vajaliku teoreetilise materjali kogumine.

Tallinna Tehnikaülikooli laboris NRG-423 paikneva Mitsubishi MELFA roboti integreerimine süsteemiga ROS. Algõppeks vajalike praktiliste tööde juhendite välja töötamine.

4. Lähteandmed

Kasutatakse Interneti vahendusel kättesaadavaid kirjandusallikaid ja IEEEXplore digitaalraamatukogu kaudu kättesaadavaid artikleid.

5. Uurimismeetodid

Iseseisev uurimistöö koos tarkvarakomponentide praktilise katsetamisega. Ülesannete modelleerimine, simulatsioon ja katsed robotikontrolleritega.

6. Graafiline osa

Joonised, pildid, kinemaatilised skeemid.

7. Töö struktuur

Eessõna, Lühendite ja tähiste loetelu
Sissejuhatus
Avatud lähtekoodiga robotite juhtimistarkvara komponentidest (ülevaade)
Robotite juhtimisplatvormi valik ja selle olulisemad komponendid (ROS tutvustus)
Robotite integreerimise näited ja nendega liideste tarkvara (Mitsubishi MELFA ja liikurrobotid)
Praktiliste tööde kirjeldus, Praktiliste tööde juhendid
Kokkuvõte
Kasutatud kirjanduse loetelu, Lisad

Lõputöö konsultandid

Tanel Jalakas, Tallinna Tehnikaülikooli vanemteadur.

10. Töö etapid ja ajakava

ROS platvormiga tutvumine **15.11.2020**
Teoreetilise kirjanduse läbitöötamine **20.11.2020**
NRG-423 paikneva Mitsubishi MELFA roboti ja ROS integreerimine **15.12.2020**
Õppeks vajaliku baasinfo ja tutvustava materjali koostamine **01.01.2021**
Praktiliste tööde väljatöötamine ja juhendite koostamine **01.02.2021**
Juhendajale läbilugemiseks saatmine **01.03.2021**
Paranduste sisseviimine **15.03.2021**
Juhendajale teiseks läbilugemiseks saatmine **01.04.2021**
Töö lõplik versioon valmis **18.04.2021**

SISUKORD

EESSÕNA

Liikurrobotite juhtimiseks on tänapäeval üha enam kasutusel mitmesugused avatud lähtekoodiga tarkvaraplatvormid, sh ROS. Tallinna tehnikaülikooli elektroenergeetika ja mehhatroonika instituudis arendatakse mitmeid unikaalseid liikurroboteid (sh laev Nymo jt), samuti on kohapeal olemas mitmesugused tööstuslikud manipulaatorid, mille puhul on oluline tarkvaraplatvormi standardiseeritus ja ühilduvus. Robotid on osaks ka mitmetest teistest instituudi töötajate osalusel arendatavatest süsteemidest, nt betooniprinter.

Töö raames uuriti vabavaralisi ja tasuta kättesaadavaid robotite juhtimiseks ettenähtud tarkvarakomponente. Uue tarkvara kaasamine võimaldab kaasajastada ning mitmekesistada õppetöö sisu robotitehnika, tööstusautomaatika ning mitmesugustes süsteemide modelleerimise ja juhtimistarkvaraga seotud õppeainetes. Samuti laienevad võimalused robotite rakendamiseks tööstusautomaatikas ja teaduses.

Soovin tänada lõputöö juhendajat, vanemlektor Madis Lehtlat lõputöö idee eest. Samuti abi eest lõputöö teostamiseks vajaliku info kogumisel ning töö praktilisel teostusel. Samuti kaasjuhendajat, vanemteadur Tanel Jalakat igakülgse abi eest tööks vajalike materjalide kogumisel ning töö käigus esile kerkinud probleemide lahendamisel.

LÜHENDITE JA TÄHISTE LOETELU

3D – kolmemõõtmeline (ingl k three-dimensional)
CMake – avatud lähtekoodil põhinev programmipakettide koosteprogramm
DOF – liikuvusastmete arv (ingl k degrees of freedom)
MRPT – mobiilse roboti programmeerimise tarkvarapakett (ingl k Mobile Robot Programming Toolkit)
P – pöördenurk algasendist ümber Y telje (ingl k Pitch)
R – pöördenurk algasendist ümber x telje (ingl k Roll)
ROS – roboti operatsioonisüsteem (ingl k Robot Operating System)
RViz – kolmemõõtmeline robotite visualiseerimisrakendus tarkvarapaketist ROS
SLAM – samaaegse positsioneerimise ja kaardistamise meetod (ingl k Simultaneous Location and Mapping).
STL – stereolitograafias levinud failivorming objekti pindade kirjeldamiseks kolmemõõtmelisel võrgustikul (ingl k stereolithography)
TCP – edastusohje protokoll (ingl k transmissioon control protocol)
TCP/IP – internetiprotokollistik (ingl k internet protocol)
tf – koodinaatide teisendus erinevate koordinaatteljestike vahel (ingl k transformation frame)
URDF – universaalne roboti kirjeldamise formaat (ingl k Universal Robot Description Format)
WSL – Windowsi Linux alamsüsteem (ingl k Windows Linux Subsystem)
Y – pöördenurk algasendist ümber z telje (ingl k Yaw)
YARP – tarkvarapakett nimega „jälle üks roboti platvorm" andurite, protsessorite ja ajamite ühendamiseks robotites (ingl k Yet Another Robot Platform)
Qt – Ettevõtte QT ja QT-projekti poolt arendatav graafilise liidese elementide teek rakendusprogrammide kasutajaliideste loomiseks
Rqt - Qt-põhine liides ROS kasutajaliideste loomiseks
UDP – kasutajadatagrammi protokoll (ingl k user datagram protocol).

SISSEJUHATUS

Robotitehnika on tänapäeval muutunud kiirelt arenevaks ja samas ka igapäevaelus vältimatuks tehnikavaldkonnaks. Selle arengu kiirenemise üks olulisemaid põhjustajaid on kindlasti unifitseerimine. Robootika algusaegadel oli tavaks, et iga robot konstrueeriti unikaalsetest riistvarakomponentidest, mis nõudis alati ka samavõrd unikaalse juhtimistarkvara loomist. Selle asemel, et endiselt iga uue roboti projekteerimisel alustada nii riist- kui tarkvara uuesti loomisest, korduvkasutatakse üha rohkem varem loodud elemente. Selline lähenemine võimaldab roboti väljatöötamise protsessi oluliselt kiirendada ja lihtsustada, mis omakorda muudab uute robotite tootmise soodsamaks ning teeb need üha laiemale kogukonnale kättesaadavamaks.

Sellise lähenemispõhimõtte heaks näiteks on erinevad vahevarad, mida kasutatakse robotite ja kasutaja, või roboti ja seda juhtiva programmi vahel. Üheks niisuguse vahevara näiteks on ROS (*Robot Operating System*). Teiste analoogsete vahevarade seas on ROS levikule hiiglasliku tõuke andnud asjaolu, et tegu on avatud lähtekoodiga ja tasuta levitatava tarkvaraga. See on tekitanud ROS tarkvaraplatvormile tugeva kogukonna, kes seda toetavad ja arendavad, integreerides seda pidevalt uute lisaseadmetega ja täiendades olemasolevaid süsteeme. Erinevate uuringute andmetel eeldatakse ROSiga seotud robootikaturu kasvu keskmiselt 9,6% aastas perioodil 2021-2026. [1]

Käesoleva lõputöö esimene osa käsitleb ROS süsteemi ajalugu ja analoogseid tarkvarapakette. Teine osa käsitleb, ROS vahevara erinevaid osi ja nende omavahelisi seoseid ja andmevahetust väliste. Kolmas osa kirjeldab põhilisi probleeme, mis esinevad ROS rakendamisel liikurrobotite juhtimiseks. Neljas osa käsitleb Tallinna Tehnikaülikooli tootmise automatiseerimise laboris (ruumis NRG-423) paikneva Mitsubishi MELFA roboti integreerimist süsteemiga. Integreerimiseks vajaliku tarkvara paigaldust ja sideprogrammile puuduvate vahelülide loomist ning paketi häälestamist. Lõputöö viimane osa sisaldab õppeks kasutatava praktilise programmi väljatöötamist.

1. PROBLEEMI ANALÜÜS

Avatud lähtekoodiga tarkvaraplatvormide (sh Linux ja ROS) populaarsus soodsa, töökindla ja kasutajasõbraliku robotijuhtimise platvormina kasvab kiiresti. Üha enam tuleb turule tööstuslikus kasutuses sobivaid robotsüsteeme, mis kasutavad tööks avatud lähtekoodiga tarkvaraplatvorme ja -komponente. See omakorda loob uusi väljakutseid õppeasutustele sh kõrgkoolidele eelnimetatud komponentide tutvustamiseks õppeprotsessis. Käesoleva lõputöö eesmärk on arendada võimalusi robotite ja robootikaplatvormide õpetamisel ning rakendamisel. Selle eelduseks on tööks vajalike tehniliste abimaterjalide (sh juhendite jt teavikute) läbitöötamine. Tallinna tehnikaülikooli laboris oleva tööstusliku manipulaatori integreerimine avatud lähtekoodiga tarkvarakeskkonda ROS ja selle visualiseerimisrakendusse RViz võimaldab tutvustada selle rakendamist tööstuslike manipulaatorite juhtimisel.

1.1 Avatud lähtekoodiga platvormi ROS arengulugu

Kuigi nimi ROS tuleneb lühendina roboti operatsioonisüsteemist pole see eraldiseisev operatsioonisüsteem, vaid tarkvararaamistik mõne olemasoleva operatsioonisüsteemi (nt Linux) peal. Seega koosneb see pakett erinevatest programmidest, arendustööriistadest, andmebaasidest ja elementidest, mis võimaldavad erinevatel roboti osadel üksteisega suhelda. Baastarkvarana on enamasti kasutusel mõni kogukonna poolt ametlikult toetatud Linuxi tuumal põhinevatest operatsioonisüsteemidest (enamlevinud on Ubuntu ja Debian). Lisaks Debiani põhistele Linux-süsteemidele on eelkompileeritud paigalduspaketid olemas ka Microsoft Windowsile ja Arch Linuxile. [2]

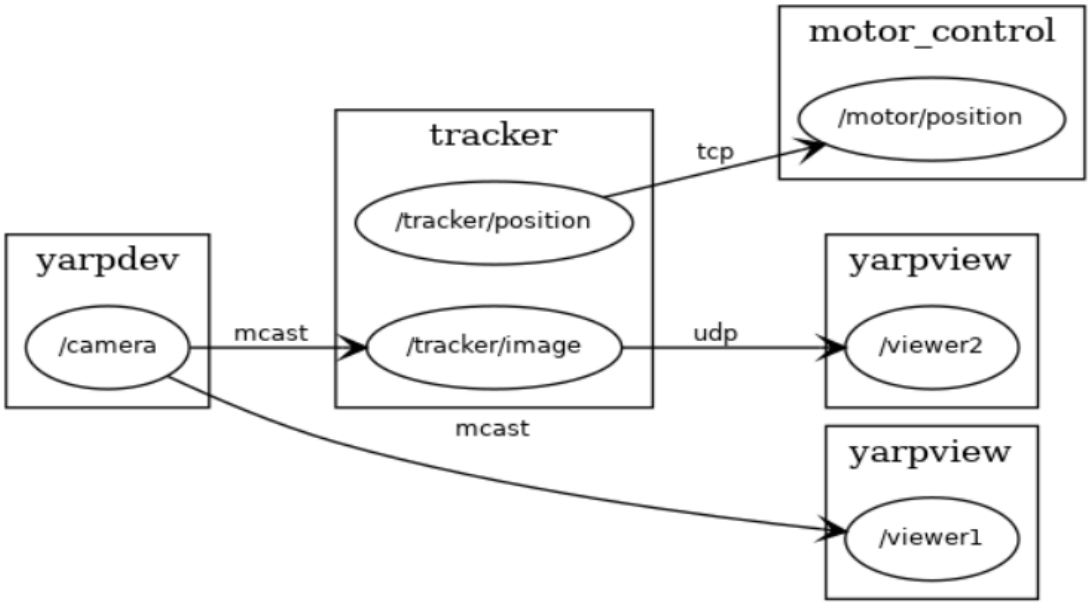
Tarkvaraplatvormi arenduse alguseks võib lugeda aastat 2006, kui kaks Stanfordi ülikooli tudengit puutusid kokku robootikatarkvara taaskasutusprobleemiga ja proovisid seda lahendada. Iga uue roboti ehitamisel kulus enamik tööaega samade põhielementide taasloomisele, jättes seetõttu vähem aega uute funktsioonide arendamiseks. [3]. Esimene ametlik ROS distributsioon, ROS Mango Tango, avaldati aastal 2009. [3] Alates teisest versioonist on kõiki ROS distributsioone nimetatud kilpkonnade järgi. Aasta 2021 seisuga uuendatakse neid sama intervalliga kui Ubuntu operatsioonisüsteemi pikaajalise toega versioone.

Tarkvaraplatvormi ülesehitus rajaneb õhukese struktuuri põhimõttele, mille eesmärgiks on vältida üksikuid ressursimahukaid programmiosi. [3] See muudab tarkvara edaspidise arendamise, veaotsingu ja üksikute funktsioonide lisamise kergemaks.

1.2 Teised analoogsed tarkvaraplatvormid

1.2.1 Tarkvaraplatvorm YARP

Avatud lähtekoodiga robootikaplatvormidest üks enimkasutatavaid lahendusi on ka YARP (*Yet Another Robot Platform*). See on aastal 2002 alustatud ja eelkõige inimest imiteerivate robotite (humanoidrobotite) juhtimiseks mõeldud tarkvararaamistik. YARP on modulaarne ja kirjutatud programmeerimiskeeles C++. YARP projektide koostamiseks kasutatakse programmipakettide koosteprogrammi CMake. See on avatud lähtekoodiga ja platvormist sõltumatu tarkvara kompileerimise automatiseerimiseks. Robottikaplatvormi YARP põhielementideks on protsessid. Konkreetsed protsessid võivad olla seotud mingite sisend- või väljundseadmetega või näiteks mingit protseduuri teostava vahelüliga. YARPi üks eelistest on see, et protsessid ei pea asuma samas seadmes või isegi samas arvutivõrgus. Protsessil võib olla üks või mitu porti, millistest igaühe suhtlusviis, protokoll jne on eraldi määratav. Ka võib protsess saata ühe pordi kaudu infot mitmele teiste protsesside portidele. Andmed ja sõnumid, mida portide vahel vahetatakse võivad olla erinevatel kujudel. Selleks, et YARP neid käsitleda saaks tuleb need panna YARP-i jaoks sobivasse vormi ehk pudelisse. [4]



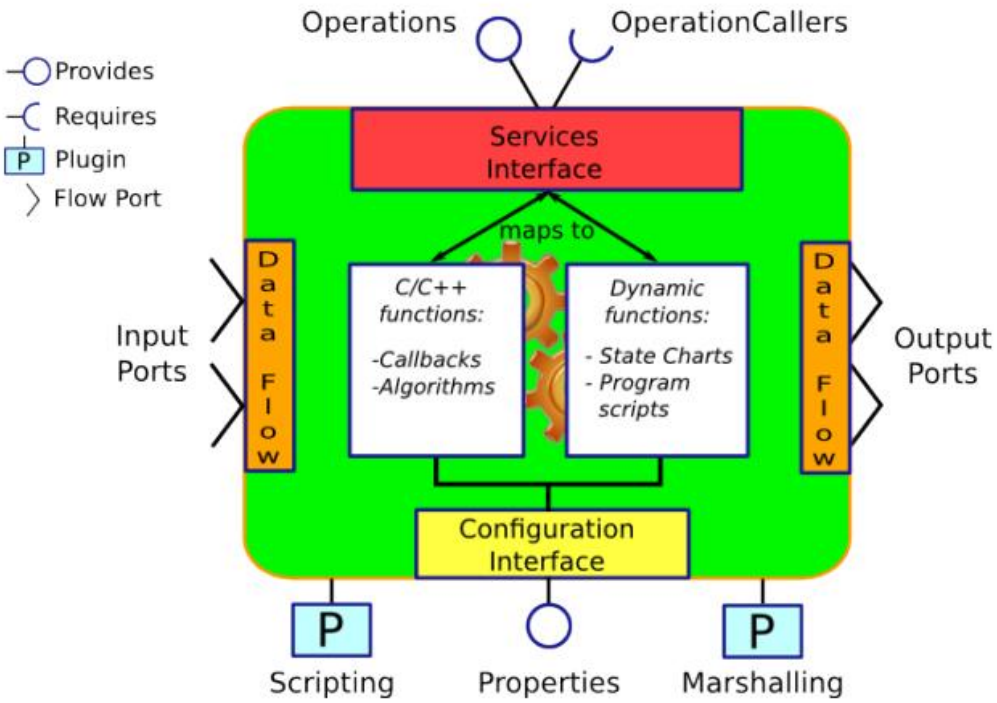
Joonis 1. YARP andmevoo liikumise põhimõtteskeem. [4]

Joonisel on näidatud ühe YARP süsteemi põhimõtteskeem. Kaamerast võetakse *multicast* protokolliga videovoog, mida võtavad vastu protsess *tracker* ja ka otse visualiseerimistarkvara. Multicast on protokoll, mis võimaldab saata andmevoogu korraga mitmele seadmele. Protsess *tracker* analüüsib videovoogu ja selle infost lähtuvalt juhib roboti ajamit ning saadab samuti voo programmi Yarpview. Kuna videovoo puhul ei ole reaalajas tagasiside vajalik, siis nende puhul kasutatakse tagasisideta protokolle multicast ja udp, aga mootori juhtimiseks tagasisidega andmesideprotokolli tcp.

Pakett YARP ja ROS pole konkurendid vaid neid kasutatakse sageli koos. Nii paketil YARP kui ka paketil ROS on funktsioone, mis on teisest paremini arendatud ja selline sümbioos nende kooskasutamisel võib robotil soovitava tulemi saavutamist kiirendada. Alates paketi YARP versioonist 2.3.4 on ühendus ROSiga YARPi integreeritud. See tähendab, et paktist YARP saab suhelda otse ROSi teemade, teenuste ja parameetrite serveriga. Pakett YARP on olemas operatsioonisüsteemidele Windows, Linux ja macOS jaoks. [5]

1.2.2 Tarkvaraplatvorm OROCOS

Tarkvaraplatvorm Orocos (*Open robot Control software*) on Euroopa Liidu toel aastal 2001 algatatud projekt. Orocos-e loomise eesmärk ja arhitektuur on mõnevõrra sarnane YARPi ja ROSiga. Iga andur, seade jms on süsteemi jaoks eraldi moodul. Samuti hakati kogu projekti arendama põhjusel, et hoida kokku aega mis kuluks iga järgmise roboti puhul samade programmilõikude uuesti ja uuesti kirjutamisele. Orcos-e peamiseks programmeerimiskeeleks on C++ ja pakettide kompileerimisel kasutatakse sarnaselt YARP-iga tarkvara koosteprogrammi CMake. Kuna Orcos on suunatud eeskätt roboti reaalajas juhtimisele maksimaalselt väikeste viidetega, siis puudub selles ROSiga sarnane ülemsõlm. Selle asemel suhtlevad kõik moodulid omavahel otse, tagades nii võimalikult lühikese viite reaalajasides. [6]



Joonis 2. Paketi Orocos andmevoo liikumise põhimõtteskeem [7]

Paketil Orcos on samuti olemas liidestuspaketid platvormile ROS. Rakendades pakette Orcos ja ROS omavahelises sümbioosis on võimalik luua roboteid, millel on informatsioonile kiiret reageerimist nõudvatele moodulitele suhtluseks Orcos ning vähem ajakriitiliste süsteemi osadega suhtluseks ROS. Orocos nõuab kasutajalt rohkem programmeerimisoskust ning selle seadistamine võib algaja jaoks olla päris keeruline. Sel põhjusel pole Orcos praegu veel saavutanud ROSi sarnast populaarsust.

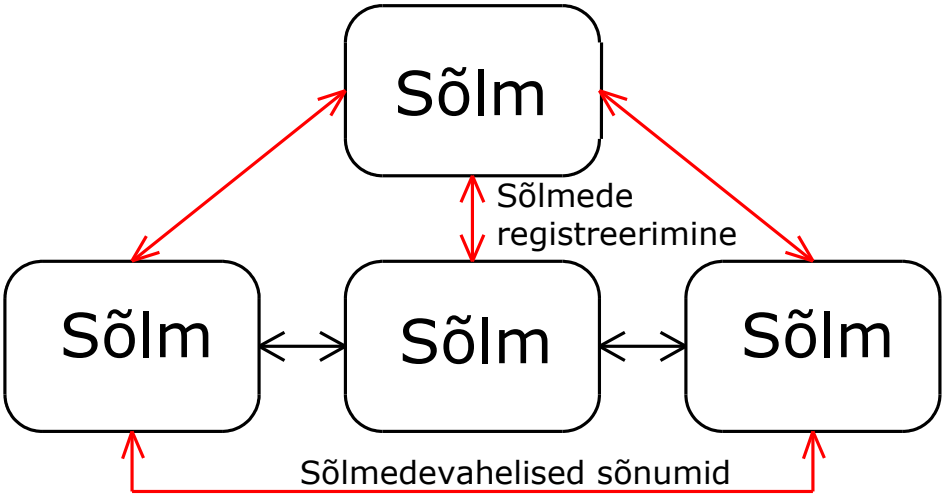
1.2.3 Muud avatud lähtekoodiga komponendid ja paketid

Erinevaid avatud lähtekoodil põhinevaid projekte, vahevarasid, ja robootikaplatvorme on veel mitmeid. Enamus neist on aga praeguseks hakanud kas populaarsuselt ja funktsionaalsuselt ROSile alla jääma, või on leidnud sellega mingi sümbioosi. Näiteks ROSiga umbes samaaegselt alustatud Player/Stage projekti roboti käitumise visualiseerimiseks mõeldud tarkvarapakett Gazebo on aktiivselt edasi arendatav ja pakub sellisena teiste hulgas ka ROSile olulist lisandväärtust.

2. TARKVARAPLATVORMI ROS ELEMENDID

2.1 Tarkvaramoodulid ehk sõlmed

Iga sõlm platvormil ROS on mingi kindla otstarbega programm. Otstarve on reeglina igal sõlmel erinev ja see võib pakkuda ka teistele sõlmedele mingit teenust. Sõlme ülesandeks võib olla näiteks sidepidamine mingi anduri või täituruga, side konkreetse andmesideprotokollile vastavalt teiste seadmetega, mingi süsteemi komponendi või parameetri jälgimine (nt manipulaatori liikumise jälgimine), parameetrite väärtuste teisenduste (nt kinemaatiliste teisenduste) teostamine vms. [8]



Joonis 3. ROS sõlmede vaheline suhtlus.

Sõlmedevaheline suhtlus toimub tellitud (subscribed) teemade (topic) kuulamise ja oma väljundandmete teemadesse postitamise (publishing) kaudu. Info, mida sõlm teemadest kuulab või teemadesse postitab võib olla mingi positsioon, millegi väärtus, või ka näiteks videokaamera voog.

2.2 Liidermoodul ehk ülemsõlm

Liidersõlm (master) juhib süsteemi komponentide vahelist infovahetust. Iga ROSi süsteemis töötav sõlm peab enda olemasolust ROSi teavitama. Selleks iga sõlm registreerib ennast ülemsõlme juures, ning informeerib ülemsõlme seda ka sellest, mis sõlmede infot soovib kuulata.

Ülemsõlm omakorda hoiab infot sõlmede kohta, registreerib sõlmed, nende nimed ja teemad millega need seotud on. Samuti haldab parameetrite serverit. Selline lahendus võimaldab samas süsteemis mitme samanimelise sõlme olemasolu, mis omavahel ei suhtle.

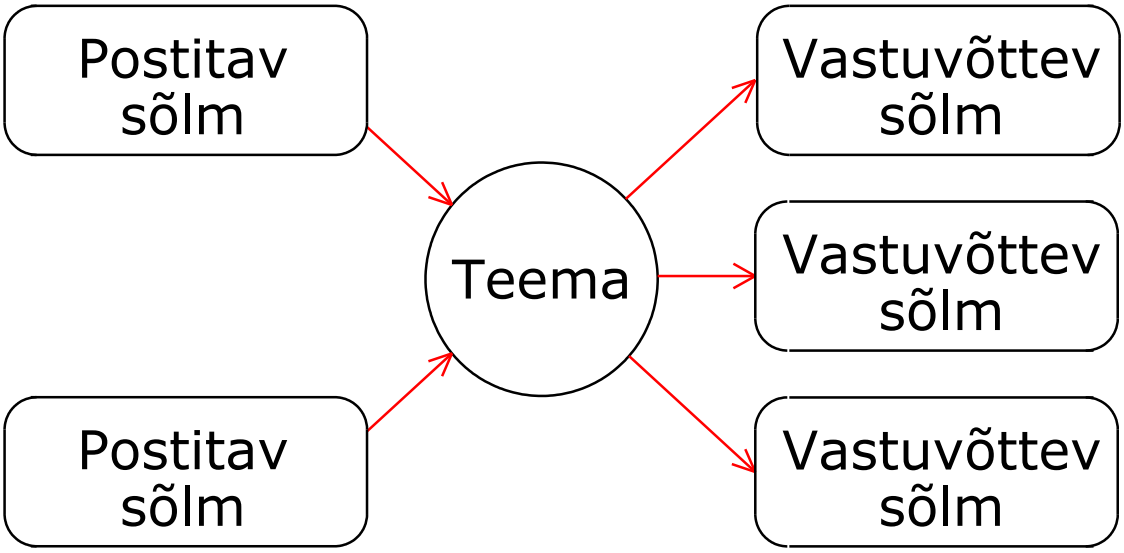
Ülemsõlm käivitatakse tavaliselt käsuga *roscore,* mis käivitab ülemsõlme, parameetrite serveri ja sündmuste logimiseks kasutatava /rosout teema. [9] Praktilistes rakendustes on tavaks kasutada (faililaiendiga „launch") käivitusfaili. Sinna faili saab kirjutada lisaks erinevaid seadeid ja parameetreid, mida oleks ebamugav iga kord käsurealt sisestada. Lisaks käivitatakse käivitusfailiga auromaatselt ka *roscore* käsklus, kui ülemsõlm ei tööta. Käivitusfaile käivitatakse vormingus *roslaunch paketi_nimi sõlme_nimi*.

```
roslaunch ros_melfa Rviz.launch
```

Joonis 4. Käivitusfaili käivitamise koodinäide.

2.3 Andmevahetusteemad

Teemad (*topic*) on infokanalid mida sõlmed kuulavad või millesse infot postitavad. [10] Sama teema postitajate ja kuulajate arv ei ole kuidagi piiratud, ning sama teema postitajad ja kuulajad ei pea olema teadlikud sellest milline sõlm veel seda teemat jälgib või sellesse infot saadab. Seda, kas konkreetsel sõlmel on õigus konkreetsesse teemasse infot saata, kontrollib ülemsõlm.



Joonis 5. Sõlmede suhtlus sama ROSi teema vahendusel.

2.4 Andmevahetuse sõnumid

Infot mida ROS teemade kaudu vahetatakse, nimetatakse sõnumiteks. Selleks, et nii andmeid vastuvõttev kui postitav pool saaks vahetatavatest sõnumitest alati ühtmoodi aru, tuleb sõnumivorming eelnevalt kokku leppida. Seda tehakse teadete failis (faililaiend „mgs") määratavate sõnumidefinitsioonidega.

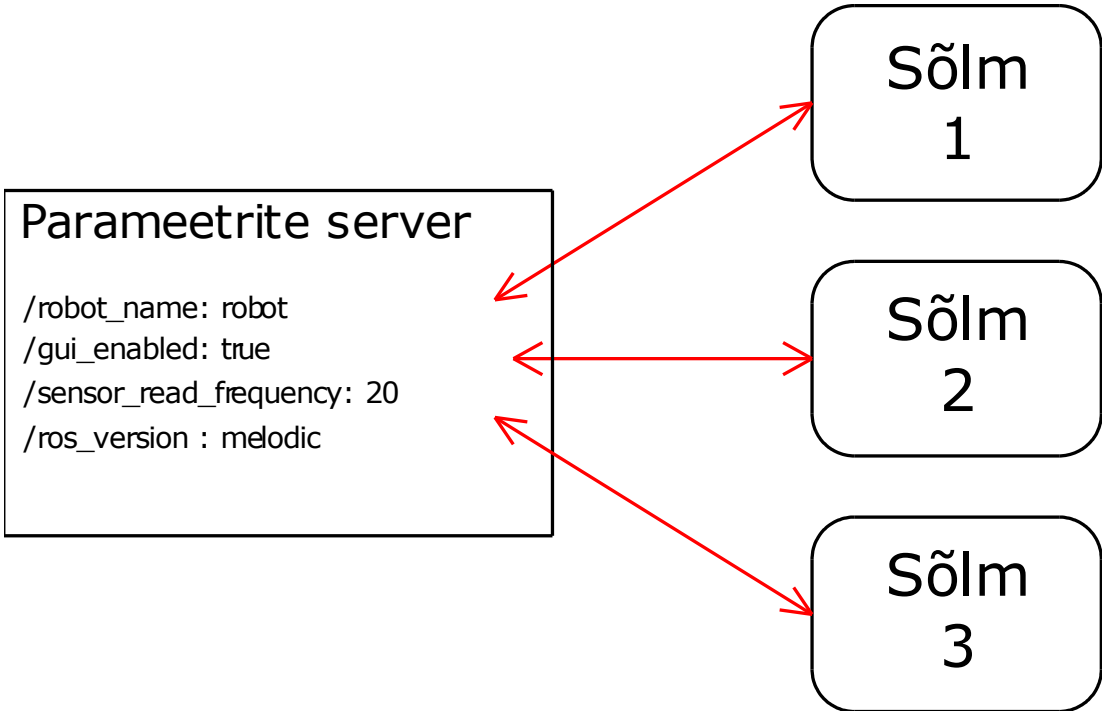
```
Header header

string[] name
float64[] position
float64[] velocity
float64[] effort
```

Joonis 6. sõnumivormingu näide. [11]

Joonisel toodud näites on kirjeldatud liigendi asendi teema „joint_states" sõnumivorming. Selle vormingu järgi tuleb sellesse teemasse postitamiseks või selle kuulamiseks määrata elemendi (nt roboti liigendi) nimi, mis tuleb anda tekstina ja 64 bitiliste, numbriliste suurustena, samuti asend (position), kiirus (velocity) ja mõju (effort). Nendest iga väärtuse puhul on tegu nimekirjaga (list). Komaga eraldatult võib sel viisil edasi anda hulga liigendite nimesid või asendite, kiiruste või mõjude sh jõudude või pöördemomentide väärtusi.

2.5 Parameetrite server

Süsteemi staatiliste parameetrite haldamiseks on kasutusel parameetrite server, mis käivitatakse automaatselt ülemsõlme poolt. Parameetrite serveris hoitakse süsteemi üldisi väärtusi ja ROS keskkonna parameetreid, aga ka robotipõhiseid parameetreid. Näiteks roboti kirjeldus, roboti nimi ja mõnedelt sensoritelt info küsimise sagedus, liigendite omavahelised paiknemised jms. [12] Roboti juhtimisel, või ka näiteks visualiseerides laetakse roboti kinemaatiline mudel parameetrite serverisse. Sealt saab iga sõlm, mis mingisugust infot vajab, vastavaid parameetreid küsida.



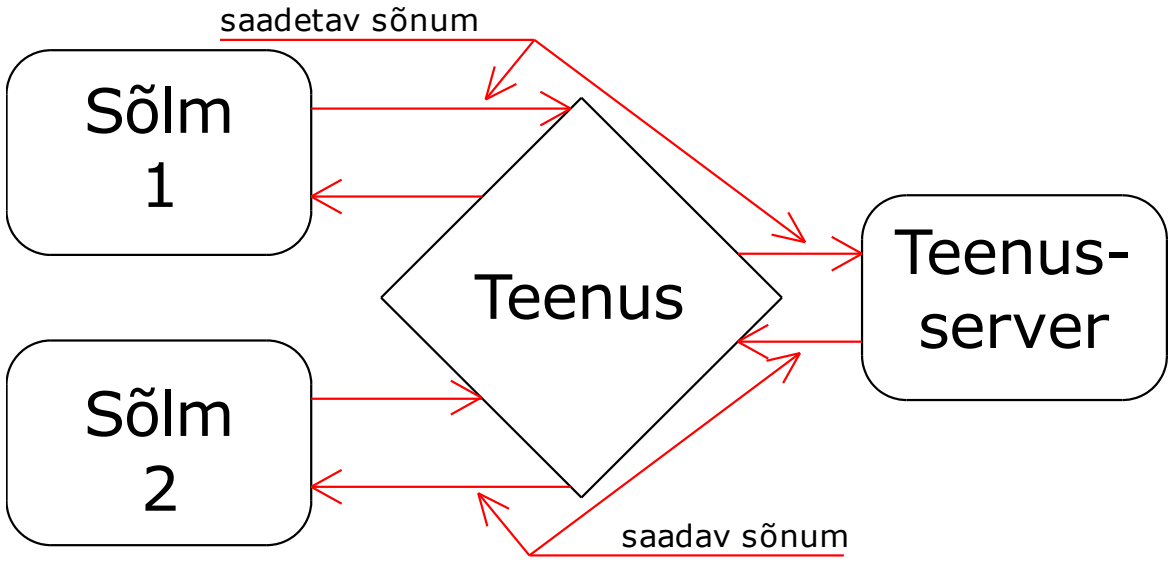
Joonis 7. ROS parameetrite serveri suhtlus.

Parameetrite serverisse sisestatud väärtused ei ole sõlmepõhised ja sõlme sulgemisel jäävad parameetrid serveris aktiivseks kuni nende sulgemiseni.

2.6 Teenused

Võrreldes lihtsate sõlmede ühesuunalise ja ilma tagasisideta infovahetusega teemade kaudu toimub teenussõlmede puhul suhtlus klient-server arhitektuuri põhimõtteid järgides. See tähendab, et teenusesse infot saatev või küsiv sõlm jääb alati ootama vastust teenusserverilt. Sealjuures ei pea infot saatev ja vastuvõttev sõlm olema üksteisest teadlikud ning ühe teenusega võivad infot vahetada mitu sõlme. Küll aga saab konkreetset teenust ja seda pakkuvat serverit korraga süsteemis olla ainult üks.

Teenuseid on hea kasutada näiteks süsteemi osade oleku kontrollimiseks, lülitamiseks jne.



Joonis 8. ROS teenuse suhtlus.

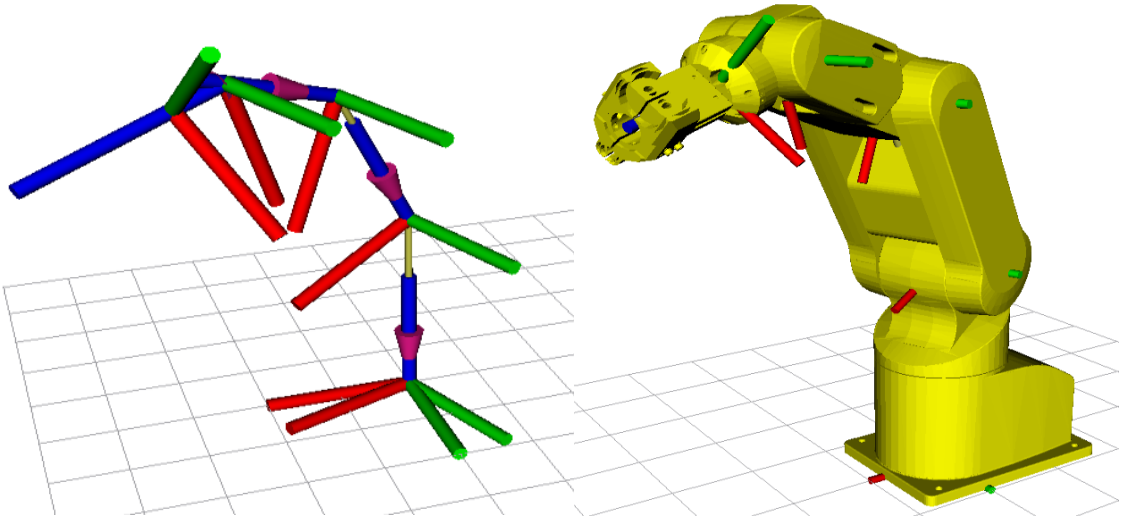
Teenuse parameetritena määratakse teenuse nimi ning teenusele saadetav ja sellelt saadava sõnumi nimi. Sarnaselt teemadele peavad nende sõnumivormingud olema eelnevalt määratud, et nii sõnumi saatja kui vastuvõtja saaksid sellest samamoodi aru. [13]

2.7 Projektihaldussüsteem Catkin

Kõiki ROSi pakette paigaldatakse projektide haldussüsteemiga Catkin. See on oma sisemiselt ehituselt avatud lähtekoodil põhinev pakettide koosteprogramm CMake ning sellele ROSiga sobitamiseks juurde lisatud tööriistad. Catkin on vajalik selleks, et erinevates programmeerimiskeeltes sh C++, Pythonis jt kirjutatud programme käsitleks ROS ühe tervikuna. Catkin loob paigaldusel failide struktuuri, kontrollib ja vajadusel paigaldab konkreetse projekti jaoks vajalikud seosepaketid (dependencies). Catkin haldab korraga kõiki töökeskkonnas olevaid projekte. ROS kasutamisel käsuviibalt on vajalik, et ROS sõlmed käivitatakse samuti Catkini töökeskkonna peakataloogist, milleks on enamasti kasutaja kodukaustas paiknev „catkin_ws“ [14]

2.8 Visualiseerimisrakendus RViz

Roboti simulatsiooniks ja visualiseerimiseks on ROS tarkvararaamistikus olemas rakendus RViz. See võimaldab, võttes aluseks mõne keskkonna või roboti suhtes fikseeritud asukoha, simuleerida roboti käitumist keskkonnas. Samuti on võimalik importida teisi roboti andurite ja mõõturite mõõteandmeid, olekuid, liigendite omavaheliste paiknemiste muutumist, jms.

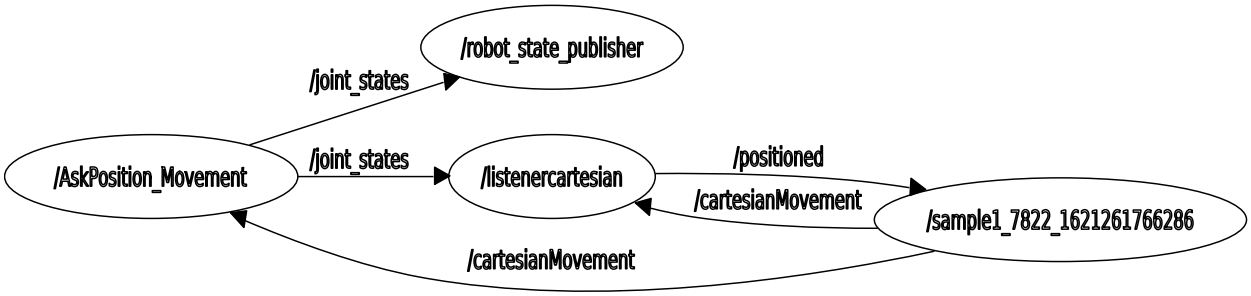


Joonis 9. Mitsubishi MELFA RV-2AJ koordinaatteljestike ja samas asendis roboti kujutis visualiseerimis-rakenduses RViz.

Visualiseerimisrakendus RViz kuvab kasutajale roboti olekut graafilises vormis, seetõttu on see suurepärane abivahend ka süsteemi vigade otsimisel ja programmide silumisel tarkvaraarendusprotsessis. Visualiseerimisrakendust saab käivitada sisestades operatsioonisüsteemi käsuviibalt „rosrun rviz rviz".

2.9 Graafiline pakettide käivitus ja silumisrakendus Rqt

Rakendusprogramm Rqt on graafilise kasutajaliidese teegi Qt baasil loodud ROSi graafiline kasutajaliides. See võimaldab kasutajal ühe programmi kaudu ja visuaalse tagasisidega jälgida töötavaid sõlmi, teenuseid, teemasid või muid elemente ja nendevahelisi seoseid, postitada sõlmedele infot ja ka seda pealt kuulata jpm. Oma lihtsasti mõistetava ja ülevaatliku liidese tõttu on Rqt väga kasulik süsteemist vigade otsimisel ehk tarkvara silumisel.



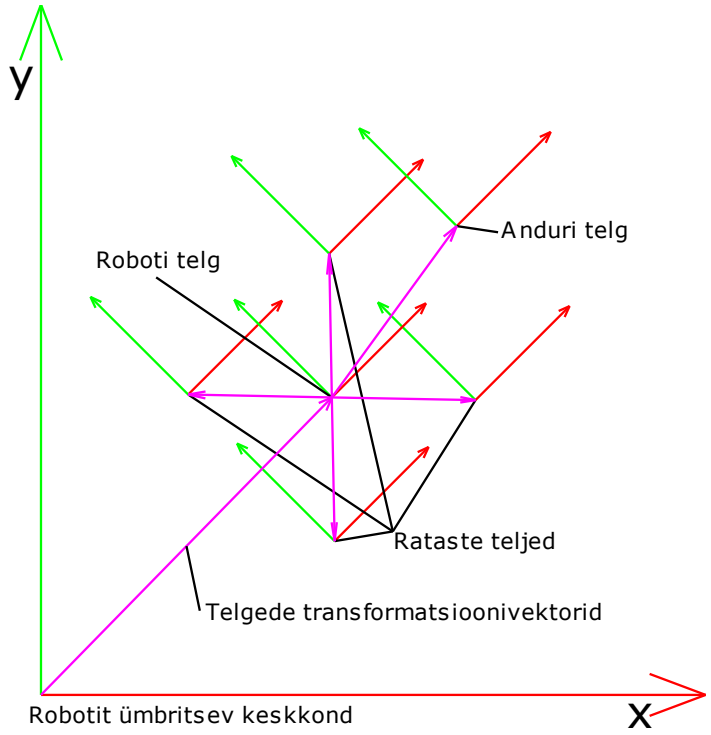
Joonis 10. Näide Rqt keskkonnas paketi ros_melfa sõlmede vahelise suhtluse visualiseerimisest.

Näites on väljavõte Rqt keskkonna funktsioonist Rqt Graph. Rqt Graph kuvab kõik hetkel aktiivsed sõlmed ja nende vahel infovahetuseks kasutatavad teemad. Samuti infovahetuse suunad. Rqt saab käsurealt avada sisestades „rqt".

2.10 Kinemaatilised teisendused

Robotitel on mitmeid liikuvaid mehhanisme sh erinevaid andureid ja täiturseadmeid, mis ei paikne üksteise suhtes fikseeritud asukohas. Seetõttu on sageli vaja teha koordinaatide ja asendite teisendusi ühest koordinaatsüsteemist või teljestikust teise koordinaatsüsteemi või teise teljestikku.

Näiteks liikurrobotil on enda asukoha teadmiseks ja manööverdamiseks vaja teada oma liikuvate mehhanismide nt rataste ja andurisüsteemide paiknemist ja suunda. Selleks valitakse üks roboti nullpunkt, mille suhtes saab kõikide teiste seadmete asukoha määrata. Sealjuures tekib igale liikuvale mehhanismile asukoha koordinaadid ja asendi kvaternioon roboti baasteljestiku suhtes. ROS navigatsioonipaketis on teisenduste (transformations) arvutamiseks objekt *tf*. Selle abil on võimalik igal ajahetkel välja arvutada kõikide roboti detailide telgede paiknemine ruumis ja anda seejärel see info teistele sõlmedele kasutamiseks, mis seda vajavad.



Joonis 11. Liikurroboti mehhanismide kinemaatilised teisendused. [15]

Joonisel on kujutatud nelja ratta ja ühe anduriga liikurroboti koordinaatteljestike teisendused ümbritseva keskkonnas suhtes. Objekt Tf nõuab, et igal teljel oleks määratud taustsüsteem ehk vanem (parent) ehk teljestik mille suhtes arvutusi teostatakse. Ühel vanemal võib ROS kontseptsiooni järgi olla mitu nn last, aga teljestikke, mis omavahel moodustavad suletud ringi, olla ei tohi. [16] Rööpkinemaatikaga ahelad (nt delta kolmjalg tüüpi manipulaatoritel vms) tuleb rööpahelad siduda muude tarkvaraliste vahenditega väljaspool URDF faili.

Lisaks konkreetse teljestiku asukohale ristkoordinaatides on vaja määrata selle asend. Asendite arvutamiseks kasutab ROS kvaternioone (quaternion). Kvaternioonid on neljakomponendilised kompleksarvud. Hüperkompleksarvude (kvaternioonide) ja pööramismaatriksite kasutamise eeliseks asendite määramisel võrreldes nurkadega määratud asendiga on asjaolu, et see väldib nn „teljestike lukustumist" (gimbal lock) arvutuste teostamisel. [17]

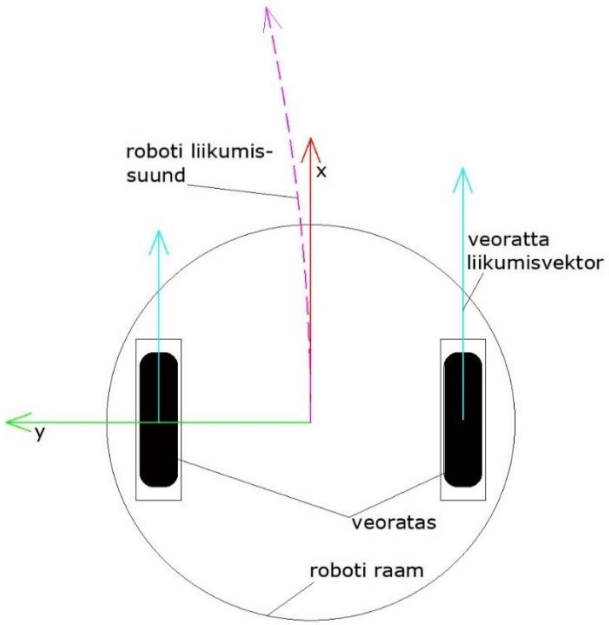
3. LIIKURROBOTITE JUHTIMINE

3.1 Ülevaade

Tarkvaraplatvorm ROS populaarsus liikurrobotite ehitamisel tuleneb suurest hulgast valmiskomponentidest enamike liikuvate robotiliikide jaoks. Seda nii tasapinnalises keskkonnas manöövreid tegevate robotite jaoks kui ka kolmemõõtmeliselt liikuvatele robotitele, nagu erinevad lendavad või näiteks ka allveerobotid. Ka tavapäraselt kahemõõtmelises maailmas liikuvate robotite puhul on erinevaid juhtimis- ja liikumispõhimõtteid palju. Kasutatakse erinevaid rataste arve ja konfiguratsioone, aga ka roomikuid või üldse kõndivaid liikumisi.

3.2 Ratastega veokite juhtimine

Lihtsamates keskkondades saab rakendada ratastega veokeid. Väikeste mõõtmetega, peamiselt õppeotstarbeks ehitatavatel robotitel kasutatakse kõige rohkem kahe vedava rattaga sõiduki või veoki diferentsiaaljuhtimist st roolimehhanismi ei kasutata ja pööramist juhitakse rataste kiiruste erinevusega. [18] Selle eelis on, et samad ajamid, millega muudetakse roboti liikumise kiirust on kasutuses ka roboti suuna muutmisel. Selline lahendus hoiab kokku ajamite arvult, aga ka roboti maksumuselt, programmi keerukuselt jms.



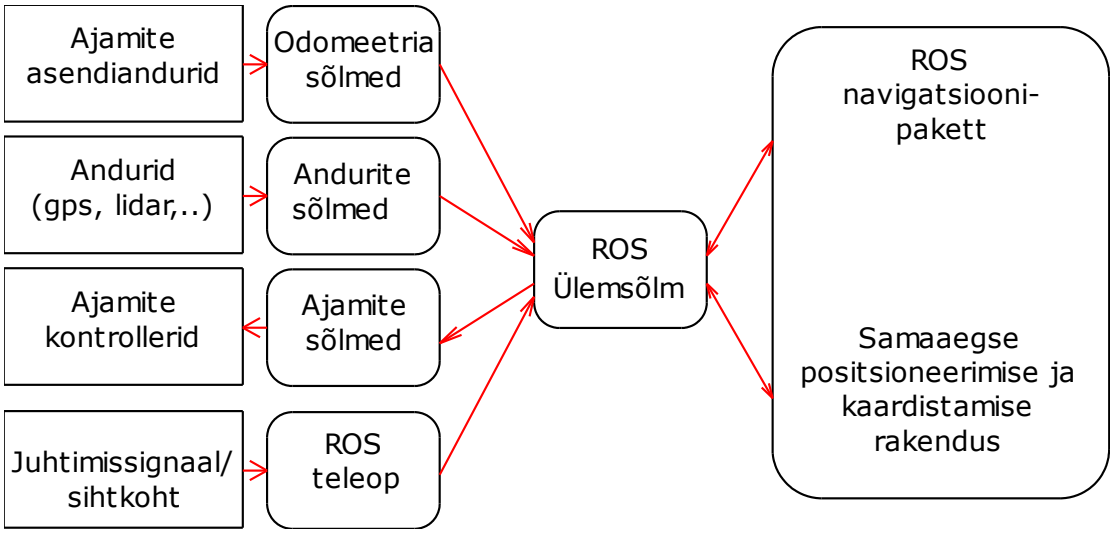
Joonis 12. Kahe erineva kiirusega vedava rattaga roboti liikuminediferentsiaaljuhtimisel. [19]

Sellist kinemaatikamudelit kasutatakse ka paljude kodumajapidamises kasutatavate robotite, nagu näiteks robottolmuimejate ja robotniidukite puhul.

3.3 Liikurroboti navigatsioonisüsteem

Liikurroboti üks põhilisemaid funktsioone, mida on vaja saavutada, on liikumine seda ümbritsevas keskkonnas. Tavaliselt saavutatakse see selliselt, et sisestatakse robotisse keskkonna kaart, või lastakse robotil see ise andurite abil luua. Seejärel positsioneeritakse robot kaardil keskkonna suhtes õigesse punkti ning sihtkoha info sisestamisel on roboti eesmärk sinna liikuda. Seejuures on oluline mitte ära eksida, mitte millelegi otsa liikuda ja mitte navigeerimisel kuhugi kinni jääda, kust robot iseseisvalt välja ei saaks. [20]

Nende ülesannete täitmiseks on olemas ROS Navigational Stack (ROS navigatsioonipakett).



Joonis 13. Liikurroboti navigatsioonisüsteemiga sidumise põhimõtteskeem.

Seejuures nii andurite kui ajamitega suhtlevad sõlmed täidavad ainult vastava info vahetamise funktsiooni ning kogu analüüs ja planeerimine teostatakse navigatsioonipaketis.

3.4 Liikurroboti positsioneerimisviisid

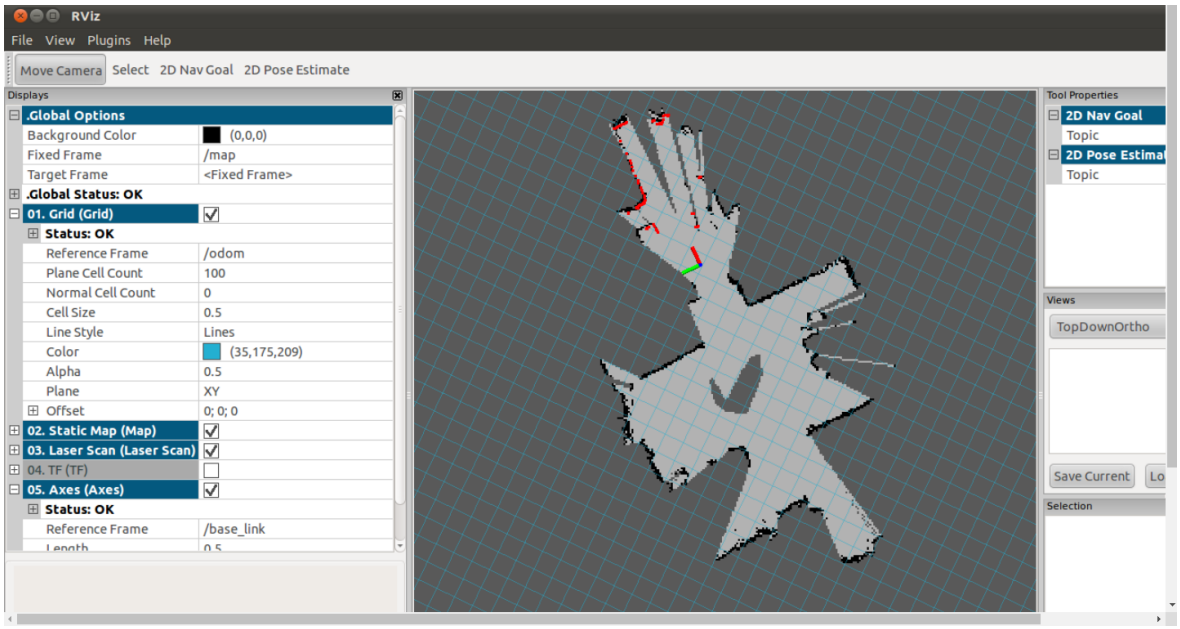
Kahe vedava rattaga sõidukite navigatsioonipakett on mõeldud kahemõõtmeliselt liikuvate ja holonoomiliselt juhitavate liigenditega robotite juhtimiseks. Holonoomilisteks liigenditeks nimetatakse liigendeid, mis on juhitavad kõigi oma olemasolevate liikuvusastemete (degrees of freedom, DOF) ulatuses. Mõiste võttis

esimesena kasutusele Heinrich Hertz aastal 1894. Selle paketi eesmärk on reaalajas odomeetria signaalide ja teiste andurite info kogumine, kogutud info analüüsimine ja selle põhjal robotile liikumiseks vajalike käskluste väljastamine.

Robotit on võimalik navigeerida mitmel erineval viisil. Kõige lihtsamal juhul kasutatakse roboti manööverdamiseks ainult operaatori poolt reaalajas antavaid signaale. See viis on kõige ebaefektiivsem, kuna sel juhul sõltub roboti suutlikkus ainult operaatori suutlikkusest ja on sellega piiratud. Teine võimalus on, et laaditakse kogu piirkonna staatiline kaart robotisse, robot positsioneeritakse kaardile ning antakse talle ette sihtkoht või rada, milleni robot kaarti lugedes püüab jõuda, või mida järgida. Selle navigeerimisviisi miinuseks on, et staatilised kaardid ei arvesta tihti dünaamiliste muutustega keskkonnas, mis võib tekitada robotile liikumisel takistusi, millega robot ei suuda iseseisvalt hakkama saada. [21] Veelgi parema tulemuse saab, kui kasutada SLAM (*simultaneous localisation and mapping*), ehk samaaegse positsioneerimise ja kaardistamisega juhtimist. Sellisel juhul robot reaalajas kaardistab enda ümbritsevat keskkonda, ning loob sellest oma kaardiserverisse kaardi kõikide takistuste ja nende kaugusega anduri keskpunkti suhtes. Enim kasutatav seade sellise kaardi loomiseks on LIDAR (*laser imaging, detection and ranging*). Lidar on oma olemuselt pöörlevale alusele paigaldatud laserkaugusmõõtja. [22] Navigatsioonipaketiga on võimalikud väga paljud ereinvad navigeerimisviisid, mille jaoks on olemas valmiskujul tarkvarakomponendid. Seetõttu kasutatakse enamasti ROSiga navigeerimisel mitte ühte neist eraldi, vaid mõnda kombinatsiooni.

3.5 Samaaegne asukoha määramine ja kaardistamine

Navigatsioonipaketis on olemas ka samaaegse asukoha määramise ja kaardistamise rakendus nimega *gmapping*. Selle eesmärk on koguda laser-skaneerimisseadmelt ehk lidarilt (light detection and ranging) või sügavuskaameralt punktide pilv, milles iga punkti kaugus on ajahetkes mõõdetud lähtudes anduri asukohast, ja joonistada saadud info alusel robotit ümbritsevast keskkonnast kahemõõtmeline kaart. Navigeerimisel jäetakse kõik seni mõõdetud punktide asukohad meelde ning liidetakse pidevalt lisanduvate punktidega. Nii tekib reaalses mõõtkavas kaart, mida on võimalik kasutada navigeerimisel ja takistuste vältimisel.



Joonis 14. Navigeerimisel koostatud kaart visualiseerimiskeskkonnas RViz. [23]

Liikuvate robotite kasutamise populaarsuse kohta võib julgelt öelda, et see on alles tõusuteel. Täna ei ole nende arendamisel põhiülesandeks enam mitte sisendite ja väljundite vaheline suhtlus, vaid veel parema, täpsema ja töökindlama ümbritsevat keskkonda mõistva, iseõppiva, ja automaatselt õigeid seoseid loova tarkvara välja töötamine.

4. MANIPULAATORI INTEGREERIMINE SÜSTEEMI

4.1 Tegevuskava

Tööstuslik manipulaator MELFA RV-2AJ on väikese võimsusega viie teljega liigendkäe tüüpi manipulaator. Käe maksimaalse siruulatusega 440mm ja tõstejõuga 2 kg on see mõeldud kasutamiseks eelkõige väikest tõstejõudu aga suurt täpsust vajavates rakendustes. Töö eesmärgiks oli siduda tootmise automatiseerimise laboris paiknev manipulaator tarkvaraplatvormiga ROS koos näidisprogrammiga selle juhtimiseks.

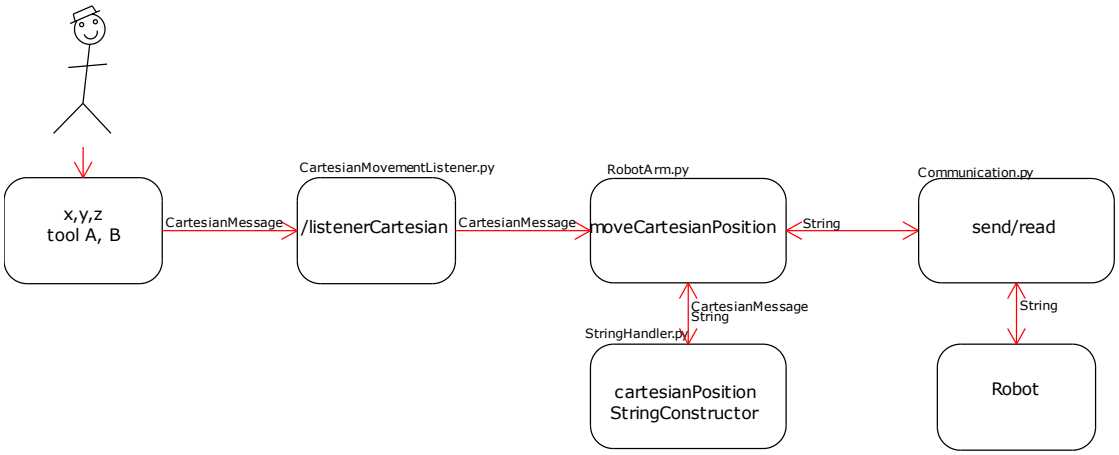
Töö esimeseks ülesandeks on tarkvara paigaldamine ja seadistamine. Paigaldamiseks on vajalik arvutisse eelnevalt paigaldada ka tarkvaraplatvormi ROS toetav operatsioonisüsteem. Selles osa sai töös katsetatud järgmisi variante:

1. Paigaldada juhtarvutisse Linux operatsioonisüsteem. Kuna see oleks reaalsuses nõudnud lisaarvuti paigaldamist ja hilisemat haldamist, siis esimene variant jäi kõrvale.
2. Paigaldada juhtarvutisse WSL (Windows Subsystem for Linux) Ubuntu operatsioonisüsteemiga. Selle variandi kasutamine oleks olnud mugavaim. Kahjuks selgus katsetuste käigus, et valitud juhtarvutis paiknevale Windowsile ei olnud võimalik paigaldada WSL versiooni, mis oleks lubanud alamsüsteemile jagada arvuti jadaporti. Jadaport on aga vajalik RV-2AJ juhtimiseks. Seega sai lõpliku lahendusena Linux operatsioonisüsteemi virtuaalmasinas.
3. Paigaldada juhtarvutisse Oracle Virtual manager ja selle alla omakorda Ubuntu. Oracle Virtual manageri alla sai paigaldatud Ubuntu 18.04 ja sellele omakorda ROS.

Töö käigus sai katsetatud ROS versioone Melodic Morenia ja Noetic Ninjemys.

Avatud lähtekoodiga side Mitsubishi robotitega MATLABi keskkonnas on lahendanud ühe esimesena Martin Melouni poolt Tšehhi tehnikaülikoolist Prahas oma töös „Mitsubishi Melfa Robot Control Toolbox“. [24] Tarkvara on küll esialgselt mõeldud robotite juhtimiseks MATLAB keskkonnast, kuid selle edasiarenduseks ja üheks esimeseks Pythoni keelseks kaugjuhtimise liideseks Mitsubishi robotitele võib lugeda paketti ros_mitsubishi, mille arendajateks olid Federal University of Ouro Preto üliõpilased Alexandre Magno de S Thiago Filho, Paulo Henrique dos Santos ja robotiklubi Rodetas Robô Clube. See võimaldab käsurealt kaudu saata liikumiskäske RV-2AJ robotile ja küsida robotilt selle positsiooni, mis seejärel tagastatakse tekstina kasutajale terminaliprogrammi. [25] Mainitud lahenduses salvestab moveCartesianPosition

funktsioon vastu võetud väärtused soovitud positsioonina ning kasutades cartesianPositionStringConstructor funktsiooni koostab operaatori poolt antud positsioonidest robotile sobivas vormingus andmejada. Läbi sidefunktsioondie saadetakse see jada robotile ning hakatakse pidevalt robotilt tsüklis positsiooni pollides võrdlema vastuseks tulevaid väärtusi ette antud positsiooniga. Kui need on saavutatud ja robot on manöövri lõpetanud, siis algab protsess uuesti või kui CartesianMovement teemas uut sõnumit ei ole, siis jääb ootele. Samasugune protsess on võimalik ka liigendi pöördenurkadega positsioneerimisel. Sel juhul positsioneeritakse robot viie telje nurkade ja kiiruse infoga, muu osa protsessist on põhimõttelt identne ristkoordinaadistikus positisoneerimisega.



Joonis 15. Paketi „ros_mitsubishi" RV-2AJ ja ROS suhtluse andmevoo diagramm. [25]

Kahjuks puuduvad sellel lahendusel sideprotokollist ja robotist tulenevad ajalised kitsendused ja info esitatakse ROSi teistele komponentidele sobimatus vormingus (tekst ja kraadid nurgaühikutena). See tarkvara on mõeldud ainult roboti positsioneerimiseks, aga mitte visualiseerimiseks. Seetõttu suhtleb robot korraga ainult kas ristkoordinaadistikus või liigendite nurkade kaudu. Kuna tööstuslikke manipulaatoreid reaalsetes rakendustes positsioneeritakse reeglina ristkoordinaadistikus, aga RViz keskkonnas visualiseerimiseks on vaja anda nurkade info radiaanides, siis nimetatud tarkvara olemasoleval kujul roboti samaaegse juhtimise ja visualiseerimise jaoks rakendatav pole. Samuti puudub eelmainitud paketis võimalus luua programm millega samaaegselt robotit positsioneerida ja kasutada haaratsit ning erinevaid roboti sisendeid ja väljundeid.

4.2 Manipulaatori visualiseerimine Rviz keskkonnas

Tööstuslikke manipulaatoreid positsioneeritakse töös enamasti ristkoordinaadistikus ehk Descartes`i koordinaadistikus. Samas Mitsubishi MELFA RV-2AJ manipulaatorit on võimalik positsioneerida lisaks ka liigendite nurkade abil, silindrilises koordinaadistikus ja tööriista koordinaadistikus. Seega võiks loodav programm arvestada kõigi nende võimalustega. Arvestades asjaolu, et erinevate positsioneerimisviiside jaoks kasutatakse erinevaid sõnumiformaate ja et robot ei nõua juhtimisel pidevalt sama koordinaatsüsteemi kasutamist, on mõistlik, et roboti juhtimisprogrammis saaks kasutada vajadusel erinevaid koordinaatsüsteeme.

Et saavutada seda, ning et samaaegselt robotile ette antava programmi täitmisega toimuks ka RViz keskkonnas visualiseerimine, tuleks kõikide positsioonide ja teiste käskude sisestamiseks teha ainult üks käivitusfail. Nii on roboti juhtimine kasutajale ülevaatlikum ja vajalike muudatuste tegemine samuti lihtsam. Soovitatav on ka, et uue positsiooni etteandmine toimuks alles siis, kui robot on eelmise positsiooni saavutanud kuid sujuvamate liikumistrajektooride puhul võib uusi punkte robotile ette anda ka liikumise ajal. Füüsilise roboti positsioneerimiseks on seega vajalik luua tagasiside selle tegeliku asendi kohta.

4.3 Tarkvarapaketi koostamine

4.3.1 Kataloogistruktuuri loomine

Paketi loomiseks tuleb esmalt käsuviibalt valida aktiivseks kaustaks kasutaja profiili all paikenv kaust catkin_ws (projektihaldussüsteei Catkin töökataloog). Selleks sisestada käsuviibale cd ./catkin_ws. Uue paketi loomiseks tuleb seejärel sisestada catkin_create_pkg koos loodava paketi nimega, näiteks *catkin_create_pkg paki_nimi*.

Seejärel tuleb sisestada kõik ROSi rakendused, millest loodava paketi töö sõltuma hakkab. Antud näiteks oli nendeks *message_generation, message_runtime, std_msgs* ja *rospy.* Nendest esimene on vajalik paketi kompileerimisel teatefailide (msg) loomiseks ja teine sõnumite haldamiseks paketi sõlmede käivitamisel. Std_msgs on ROS teek, mis sisaldab erinevate sõnumite ja andmevormingute definitsioone (sh. märgijada tüüp String, tõeväärtustüüp Bool ja mitmesugused arvulised andmetüübid). Rospy on vajalik juhul, kui ROSi pakett on kirjutatud Pythoni programmeerimiskeeles. Kogu käsk näeb näiteks välja järgmine:

```
catkin_create_pkg ros_melfa message_generation message_runtime std_msgs rospy
```

Joonis 16. Catkin paketi loomise koodinäide.

Selle rea tulemusel luuakse kataloogipuu tühja lähtekoodi /src kataloogiga ning failidega CMakelists ja package.xml

4.3.2 Paketisiseste seoste määramine

Fail CMakelists määrab mis sõnumeid, teenuseid ja lisafaile peab projektihaldussüsteem kompileerima. [26] Antud juhul on vaja kasutada kahte sõnumiformaati. Ühte ristkoordinaatsüsteemi, ja teist liigendite nurkade kaudu positsioneerimiseks. Selle jaoks peab lisama sõnumifailide defineerimise sektsiooni vastavate sõnumifailide (msg) nimed

```
# Generate messages in the 'msg' folder
add_message_files(
  FILES
    CartesianMessage.msg
    JointMessage.msg
```

Joonis 17. Failis CMakelists määratud sõnumite seosed.

Mõlemad sõnumifailid (msg) on võetud *ros_mitsubishi* [25] paketist ja määravad teatega edastatava andmevormingu. See sobib ainult RV2AJ robotile, kuueteljelisel RV1A tuleb lisada veel üks nurk (c) tööriista asendi määramiseks ja lineaartelje olemasolul ka lineaartelje asendi (L1) etteandmiseks või küsimiseks.

```
float32 x
float32 y
float32 z
float32 a
float32 b
float32 speed
```

Joonis 18. CartesianMessage.msg fail.

Roboti RV2AJ ristkoordinaatpositsiooni sõnum koosneb kuuest 32 bitisest arvust. Ristkoordinaadid, tööriista asendi koordinaadid ja kiirus.

```
float32 j1
float32 j2
float32 j3
float32 j5
float32 j6
float32 speed
```

Joonis 19. JointMessage.msg fail.

Viieteljelise liigendkäe (RV-2AJ robotil telg nr 4 puudub) positsioneerimisel liigendite nurkade järgi koosneb sõnum samuti kuuest 32 bitisest arvust. Antud on viis telge ning kiirus.

4.3.3 Paketiväliste seoste määramine

Paketifailis Package.xml määratakse paketi üldised seaded. Paketi nimi, versioon, haldaja info, litsents jms. Kui mõni seos jäi paketi loomisel määramata, siis saab siia faili neid juurde kirjutada. Samuti juhul kui paketile midagi lisatakse ja lisatud osal on mõni uus paketiväline seos.

```
<buildtool_depend>catkin</buildtool_depend>
<build_depend>message_generation</build_depend>
<build_depend>rospy</build_depend>
<build_depend>std_msgs</build_depend>
<build_depend>message_runtime</build_depend>
<build_export_depend>rospy</build_export_depend>
<build_export_depend>std_msgs</build_export_depend>
<exec_depend>message_runtime</exec_depend>
<exec_depend>rospy</exec_depend>
<exec_depend>std_msgs</exec_depend>
```

Joonis 20. Paketifailis Package.xml määratud paketivälised seosed.

4.4 Andmevahetussõlme ülesehitus

4.4.1 Python programmitekstide vorming

Kuna rakenduse ülejäänud sõlmede ega RViziga sobivaid valmiskomponente ei õnnestunud leida, siis lähtudes eelnevast sai otsustatud luua täiesti uus ROSi pakett kõigi vajalike komponentidega. Paketi programmeerimiskeeleks valiti keel Python3, mis on vajalik tööks keskkonnaks ROS2.

ROSi sõlmeprogrammi kirjutamisel määratakse esmalt ära sõlme failivorming. Tuleb ära määrata Pythoni interpretaatori asukoht, viited Python teekide asukohale ja kasutatava

tekstifaili märgikodeering (eestikeelsete kommentaaride puhul kasutatakse enamasti UTF-8 märgikodeeringut). Pythoni faili alguses peaks välja nägema järgnev.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

Joonis 21. Python interpretaatori ja märgikodeeringu defineerimine programmifaili alguses.

4.4.2 Sõlmeüleste seoste ja staatiliste parameetrite määramine

Funktsiooniga *os.getcwd()* saab süsteemilt vastuseks arvuti /catkin_ws kataloogi asukoha. Sellisel viisil kataloogipuu kirjeldamine lihtsustab edaspidi erinevatele kataloogisisestele failidele suunamist.

```
import os
path = os.getcwd() + '/src/ros_melfa/src'
```

Joonis 22. Töökataloogi viide ROS sõlme programmifailis.

4.4.3 Muud olulised üdkasutatavad teegid

Funktsioon *time* võimaldab tekitada ajalisi viiteid ja intervalle. Selleks tuleb programmi algusesse lisada järgnev rida.

```
import time
```

Joonis 23. Ajaliste viidetega seotud teegi viide programmikoodi alguses.

ROS ülene teek *rospy* tuleb importida alati, kui ROSi sõlme kood on kirjutatud Python programmeerimiskeeles. C++ puhul tuleks kasutada tarkvara roscpp.

```
import rospy
```

Joonis 24. Rospy teegiviide ROS sõlmeprogrammi algusosas.

Numpy on Pythoni matemaatikapakett. Seda on vaja erinevate keerulisemate matemaatiliste tehete teostamisel. Objekt Tf on vajalik koordinaatide teisendusteks roboti erinevate koordinaatteljestike vahel. Seda saab kasutada antud juhul 4x4 teisendusmaatriksite (pööramis- ja nihutusmaatriksite), kvaternioonide ja Euleri nurkade teisendisteks.

```
import numpy as np  # Maatriksarvutused
import tf       # Kinemaatilised teisendused
```

Joonis 25. Objektide Numpy ja tf viited ROS sõlmes.

Erinevad sõnumiformaadid, mida on plaanis kasutada, tuleb samuti importida. Antud juhul on vaja kasutada mõlema positsioneerimisviisi sõnumeid, *JointState* sõnumit RViz keskkonda visualiseerimiseks ning sõnumiformaadid number, andmejada, ning pealkiri, mis on leitavad ROS standardsete sõnumite teegist.

```
from ros_melfa.msg import JointMessage, CartesianMessage
from sensor_msgs.msg import JointState
from std_msgs.msg import Header, String, Float64
```

Joonis 26. Sõnumite vormingu ja andmetüüpide viited ROS sõlmes.

4.4.4 Sideprotokollide teegid

Erinevad tööstusrobotid kasutavad sideks erinevaid sideprotokolle, levinud on Etherneti põhised sideprotokollid sh Modbus TCP, aga ka lihtsal RS232 jadaliidesel põhinevad ühendused. Paljud protokollid eeldavad ka spetsiaalset lisatarkvara roboti kontrolleri poolelt (firmware options). Erinevate tootjate robotitel kasutatakse erinevaid sideprotokolle. Näiteks RV2AJ ja RV1A roboteid kasutatakse lisaks RS232 sideprotokollile ka Etherneti võrgukaardi ja TCP/IP protokolliga. Objekti *„Communication" saaks* üle võtta ros_mitsubishi paketist juhul kui kasutatakse ainult ühte kindlat jadaporti. Sõltuvalt sellest, kas kasutatakse füüsilist RS232 jadaporti või USB üleminekut tuleb valida õige pordiaadress ja vastavalt teine variant koodist välja kommenteerida.

```
import Communication
#comm_channel=Communication.Communication('/dev/ttyS0')
comm_channel=Communication.Communication('/dev/ttyUSB0')
```

Joonis 27. Sideteegi Communication viite näide ROSi sõlme programmikoodis.

4.4.5 Roboti parameetrite määramine kinemaatikamudelis

Roboti parameetrid on sõltuvad konkreetsest robotist. Et saaks teostada roboti positsiooni arvutusi pööramismaatriksitena (rotation matrix), tuleb sisestada staatiliste parameetritena roboti telgede andmed 4x4 teisendusmaatriksitena.

```
# Defineerime Mitsubishi MELFA RV-2AJ teljed ruumis maatriksitena
TDH1 = np.matrix('1 0 0 0; 0 0 1 0; 0 -1 0 300; 0 0 0 1');
TDH2 = np.matrix('0 1 0 0; -1 0 0 -250; 0 0 1 0; 0 0 0 1');
TDH3 = np.matrix('1 0 0 160; 0 1 0 0; 0 0 1 0; 0 0 0 1');
TDH5 = np.matrix('0 0 1 0; 1 0 0 0; 0 1 0 0; 0 0 0 1');
TDH6 = np.matrix('1 0 0 0; 0 1 0 0; 0 0 1 72; 0 0 0 1');
```

Joonis 28. Roboti telgede kirjeldus 4x4 maatriksitena.

Lisaks tuleb määrata kõikidele muudele süsteemis kasutatavatele muutujatele algväärtuseks *None.* Nii ei teki veaolukorda kui vastav muutuja veel sündmuste ohjurites defineeritud pole.

```
TDH_tool = None
active_cmd_string = None
active_cart_message = None
active_jointmessage = None
query_state = 0
```

Joonis 29. Roboti muutujate määramine koodis.

4.4.6 Märgistringide jada robotile saatmine

Andmejadade robotile saatmiseks tuleb teha funktsioon, mis suunab andmejada valitud kommunikatsioonikanali kaudu robotile.

```
#märgijadade massiivi saatmine
def SendCommandList(CmdStr):
    comm_channel.send(CmdStr)
```

Joonis 30. Sidefunktsiooni määramine.

4.4.7 Vahendusprogramm käskluste saatmiseks robotile läbi ROSi teadete

Kuna haaratsi käsitsemiseks ja sisendite juhtimiseks tuleb robotile edastatavatele käskudele ette lisada käsklus *EXEC*, tuleb kirjutada vastav funktsioon, mis eristab need käsutüübid ning lisab vastava rea. Alamprogramm exec_command käivitatakse sündmuste ohjuri (event handler) poolt ROSi teate vastuvõtul.

```
def exec_command(CmdStr):
    time.sleep(0.1)
    commands = []
    commands.append('CNTLON')

    if ('HOPEN' in CmdStr) or ('HCLOSE' in CmdStr) or ('M_OUT' in CmdStr):
        print ('IO or Gripper command')
        comm_channel.readAll()
        cmd1='EXEC' # haaratsikäskudele lisada ette EXEC
        commands.append(cmd1 + str(CmdStr))
    else:
        comm_channel.readAll()
        commands.append(str(CmdStr))

    commands.append('CNTLOFF')
    SendCommandList(commands)
    time.sleep(0.1)
```

Joonis 31. MELFA robotitele käskluste saatmine keeles Python.

4.4.8 Andmejadade koostamine MELFA robotite positsioneerimiseks

Programm lähtub eeldusest et roboti on eelnevalt positsioneeritud paketiga FESTO Ciros (varasema nimega Cosirop). Andmete saatmine toimub sarnasel vormis nagu MATLABi Melfa Robot Control Toolbox paketis.[24] Robot loeb jadasiinilt positsioone, mis on esitatud andmejada kujul. Kuna mujal koodis tehakse koordinaatidega tehteid eraldi, siis robotile positsiooni saatmiseks tuleb needa andmejadad koordinaatidest luua. Koodi struktuur on sarnane mõlema positsioneerimisviisi korral.

```
def SendJointPositioningCommand(j1, j2, j3, j5, j6, speed) :
    JointPositionString = '(' + str(j1) + ',' + str(j2) + ',' + str(j3) + ', 0.0,' +
    str(j5) + ',' + str(j6) + ')'
    commands = []
    commands.append('CNTLON')
    commands.append('EXECJOVRD ' + str(speed))
      commands.append('EXECJCOSIROP = ' + JointPositionString)
      commands.append('EXECMOV JCOSIROP')
      commands.append('CNTLOFF')
      SendCommandList(commands)
```

Joonis 32. Positsiooni andmejada koostamise näide koodis.

4.4.9 Positsioonide küsimine MELFA robotist

Robotist on vaja positsioneerimisel reaalajas küsida roboti hetke asukoht, et seda võrrelda soovitud asukohaga. Samuti on positsioone vaja küsida RViz-i visualisatsiooni jaoks.

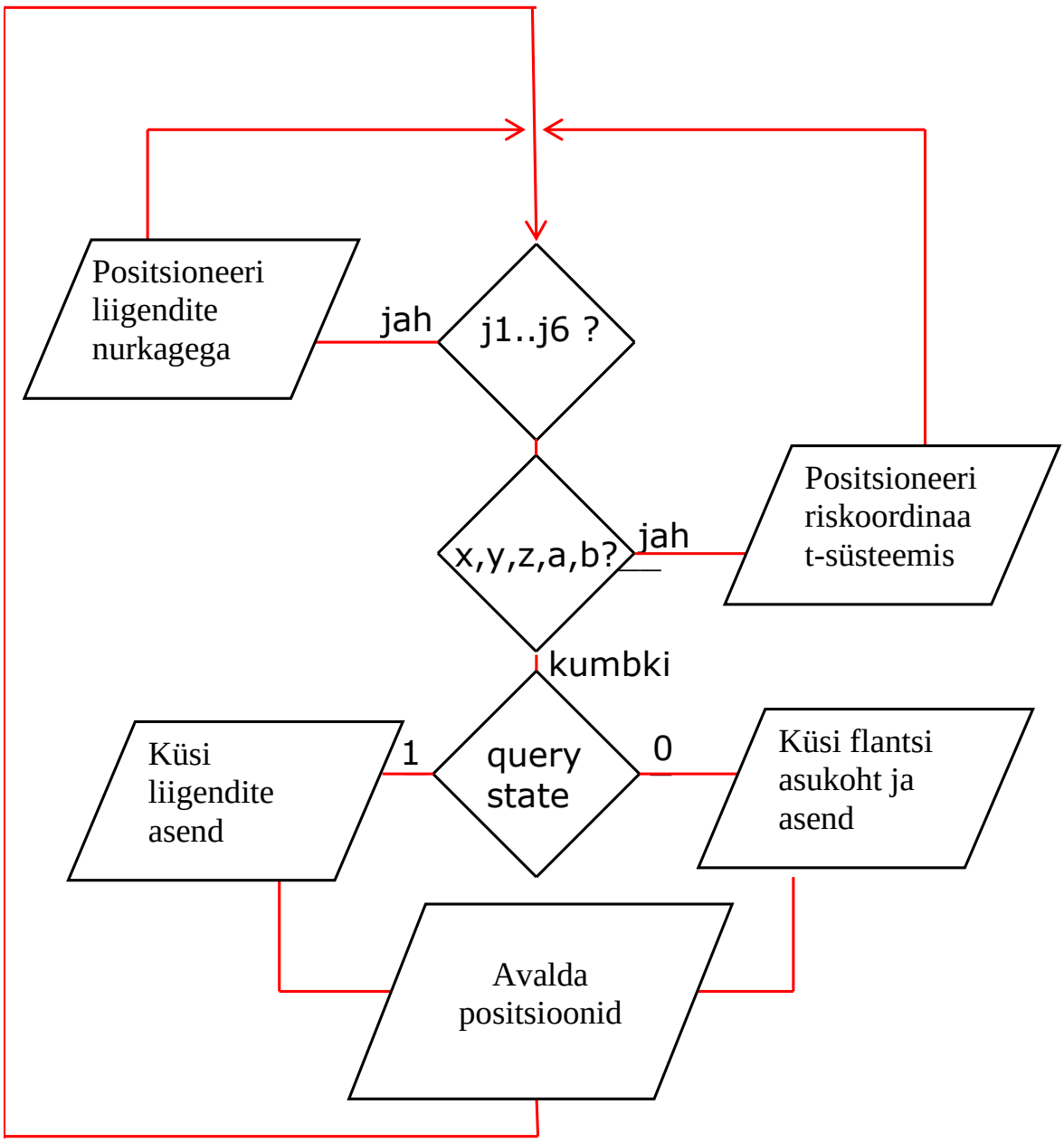
```
def GetJointPosition() :
    exec_command('JPOSF')
    return comm_channel.readUntil('\n', 100)

def GetCartesianPosition() :
    exec_command('PPOSF')
    return comm_channel.readUntil('\n', 100)
```

Joonis 33. Positsioonide küsimise funktsioonid koodis.

4.4.10 Tegevusohjuri funktsioon

Tegevusohjuri funktsiooni eesmärk on tuvastada, kas soovitakse saata robotile juhtimiskäsklust, ning kui seda parajasti ei tehta, siis küsima robotilt positsiooni ja postitama seda */joint_states* teemasse, mida RViz kuulab. Samuti */cartesian_state* teemasse, et süsteemis oleks korraga olemas roboti positsioon mõlemas variandis.

Selline käitumismall on vajalik, kuna jadasiini side robotiga saab toimida korraga ainult ühes suunas. Samuti positsiooni küsimisel peab seda tegema mõlemas positsioneerimissüsteemis, kuna operaatori poolt ette antud positsioonid võivad olla emmas-kummas süsteemis.



Joonis 34. Tegevusohjuri funktsiooni plokkdiagramm.

Kuna ROS kasutab nurkade väärtustena radiaane, aga Mitsubishi MELFA kraade, siis tuleb */joint_states* teemasse postitamiseks enne robotilt saadud kraadid konverteerida radiaanideks.

```
#Teisendame liigendite asendite nurgad kraadidest radiaanidesse
    gamma1 = np.deg2rad(gamma[0])
    gamma2 = np.deg2rad(gamma[1])
    gamma3 = np.deg2rad(gamma[2])
    gamma5 = np.deg2rad(gamma[4])
    gamma6 = np.deg2rad(gamma[5])
```

Joonis 35. Roboti liigendite nurkade teisendamine kraadidest radiaanidesse Pythoni koodis.

4.5 Positsiooni tagasiside ja kontrolli sõlm

Et roboti positsioneerimisel ja robotile käskude edastamisel ei tekiks olukorda, kus robotile saadetakse uus sihtkoht enne kui see on jõudnud eelmise tegevuse lõpetada, on vajadus luua veel üks sõlm. Selle sõlme ülesanne on operaatori poolt loodava programmi poolt postitatud positsioonide kuulamine ja selle, ning eelmises punktis kirjeldatud robotiga suhtlemise jaoks loodud sõlme vahel info vahetamine.
Selliselt on võimalik tekitada olukord, kus robotile ei saadeta enne uut positsiooni, kuni eelmise positsiooni väärtused on saanud võrdseks roboti asukoha väärtustega.

```
    if pos_command and pos_state:
      # Võrreldakse kahe samatüübilise andmestruktuuri elemente
      # Kiirust ei võrrelda
     while pos_command and pos_state and (pos_state.x != pos_command.x
or pos_state.y != pos_command.y or pos_state.z != pos_command.z or
pos_state.a != pos_command.a or pos_state.b != pos_command.b):

     #print pos_state, pos_command
      z.data=False
      pub_pos_OK.publish(z)

     print ("Positioned OK")
      pos_command=None
      z.data=True
     pub_pos_OK.publish(z)
     print_position()
```

Joonis 36. Ristkoordinaatides juhtimisprogrammist etteantud positsiooni võrdlus robotist saadud tagasisidega sõlmes MovementListener.py.

Kood ootab kuni väärtused on saanud võrdseks, siis postitatakse teemasse */positioned* sõnum „True" ning kuvatakse kasutajale tagasisideks „Positioned OK". Seda teemat kuulab roboti jaoks kirjutatav programm.

Samalaadne funktsioon on ka liigendi nurkade võrdluse kohta. Mõlemat võrdlust teostatakse samal. Võrdlemiseks võetakse andmejadast võrreldavad parameetrid välja. See on vajalik seetõttu, et andmejada sisaldab ka kiiruse infot, aga seda ei saa võrrelda.

Erinevus töö aluseks võetud ros_mitsubishi paketiga on eelkõige selles, et kogu võrdlus käib postituste jälgimise (kuulamise) ja postitamise teel. See võimaldab samal ajal info jagamist kõikide süsteemi osadega, mis seda vajavad. Sõlm kuulab kokku nelja teemat.

```
def listener():
        rospy.init_node('listenercartesian', anonymous=False)

        rospy.Subscriber("cartesianMovement", CartesianMessage, command_callback,
        queue_size=1)
        rospy.Subscriber("cartesian_state", CartesianMessage, state_callback, queue_size=1)

        rospy.Subscriber("jointMovement", JointMessage, joint_command_callback,
        queue_size=1000)
        rospy.Subscriber("joint_states", JointState, joint_states_callback, queue_size=1000)

        timer = rospy.Timer(rospy.Duration(0.1), timer_callback)

        rospy.spin()

        timer.shutdown()
```

Joonis 37. Jälgitavate ehk kuulatavate sõnumite defineerimine, näide failist MovementListener.py.

4.6 Kinemaatikamudeliga seotud ruumiline mudel

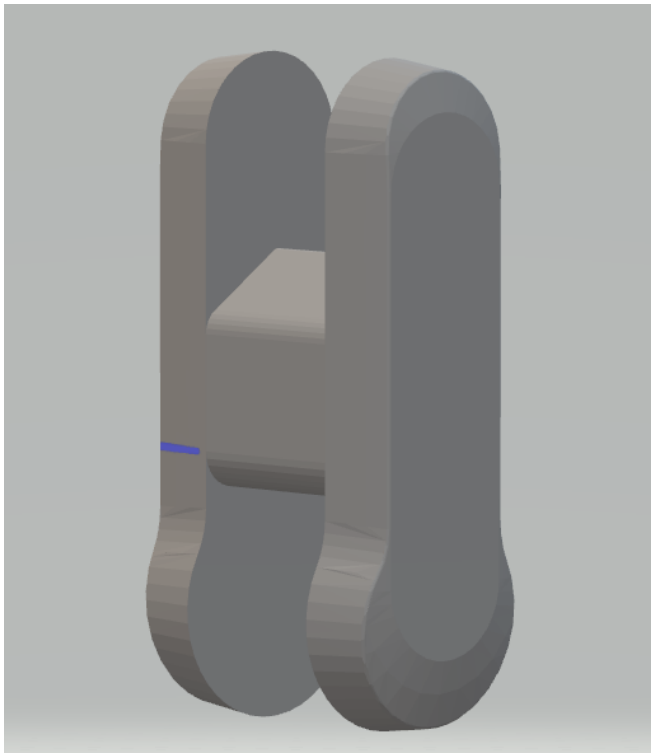
Süsteemis kasutatakse URDF (Universal Robot Description Format) standardile vastavat XML-vormingus mudelit. [27]

Mudel on vajalik roboti kirjeldamiseks virtuaalkeskkonnas. Seda formaati kasutab enamus vabavaralisi ja ka osa tasulisi robotite virtuaalkeskkonnas visualiseerimiseks loodud programme. Sellega määratakse roboti paiknemine ruumis, roboti teljed ja nende võimalikud pööramissuunad. [28] Samuti saab URDF mudeliga anda robotile visualiseerimiseks reaalse kuju ning määrata roboti omavaheliste detailide ja vajadusel ka ümbritseva keskkonna võimalikud kokkupõrkeparameetrid. URDF fail tuleb kirjutada XML-kirjelduskeeles.

```
<?xml version="1.0"?>
<robot name="RV-2AJ">
   <link name="world"/>
   <link name="base">
      <visual>
         <geometry>
    <mesh filename="package://ros_melfa/mesh/L1_J1_base.stl" scale="0.001 0.001 0.001"/>
         </geometry>
         <origin xyz="0 0 -0.023" rpy="0 0 0"/>
         <material name="grey">
            <color rgba="1 1 0 1"/>
         </material>
   </visual>
```

Joonis 38. RV-2AJ aluse kirjeldus URDF failis.

Roboti nimi on vajalik määrata selleks, et süsteem suudaks robotit kindlalt eristada. Ilma robotile nime määramata visualisatsioon ei tööta. Seejärel tuleb igale teljele või muule elemendile määrata nimi. Antud juhul robot algab liigendiga „world" mis on sisestatud selleks, et siduda robot ruumis kindla punktiga. Ilma seda tegemata on võimalik, et erinevad programmid kuvavad sama roboti erinevasse punkti. Kuna antud juhul õnnestus hankida reaalse RV-2AJ roboti kolmemõõtmelised kujutised .stl kujul, siis on kasutatud neid. Kui sellist võimalust ei ole, siis saab ka elemente kirjeldada läbi geomeetriliste kujundite.



Joonis 39. Roboti RV-2AJ üksiku lüli 3D kujutis.

Roboti iga element on eraldi .stl fail, mis paikneb ruumis mingisuguses punktis. Seetõttu tuleb kõik roboti detailide kujutised keerata ja/või pöörata ruumis õigesse asukohta. Telgede asukohana määratakse punkt ruumis, kus need peavad paiknema roboti liigendite nullasendi korral. Elementidele värvi määramine ei ole kohustuslik, aga see võimaldab robotit hiljem teistest keskkonna elementidest eristada. Saab määrata ka selliseid parameetreid nagu näiteks telgede mass, aga antud juhul ei ole see vajalik.

Kokkupõrkeinfo annab RViz-ile visualiseerimisel andmed selle kohta, et mis roboti detailid ei tohi omavahel kokku puutuda. Seda infot on võimalik samuti määrata läbi geomeetriliste kujundite. Nii saab jätta detailide ümber kaitsetsooni ja vältida paremini juhuslikku vigastamist. Antud töös on piisav kui kasutada ka kokkupõrkeinfona telgede kujutisi.

```
<collision>
 <geometry>
  <mesh filename="package://ros_melfa/mesh/L1_J1_base.stl" scale="0.001 0.001 0.001"/>
 </geometry>
 <origin xyz="0 0 -0.023" rpy="0 0 0"/>
</collision>
```

Joonis 40. Roboti RV-2AJ aluse kokkupõrkegeomeetri info URDF failis.

Sarnaselt tuleb kirjeldada ka kõik teised teljed. Kui kasutada RViz poolt loodavaid geomeetrilisi kujundeid, siis saab detailid paigutada kohe liigendite suhtes õigesti. Kasutades imporditavaid kolmemõõtmelisi kujundeid peab aga iga telje puhul eraldi seda vastavalt paigutama. Seejuures väärtused x, y, ja z määravad elemendi paiknemise ruumis telgedel, ja väärtused r, p ja y näitavad detaili pööramist ümber x,y ja z telje. Roboti kinemaatilise mudeli tähtsaimad osad on liigendid. Liigendite parameetritega määratakse ära roboti kinemaatika telgede paiknemised, liigenditevahelised suhted, liikumissuunad ja ulatused. Lisaks saab määrata ka parameetreid nagu inerts, liigendite kiirus jms.

```
<joint name="fix" type="fixed">
   <parent link="world"/>
   <child link="base"/>
   <origin xyz="0 0 0" rpy="0 0 0"/>
</joint>

<joint name="J1" type="revolute">
   <parent link="base"/>
   <child link="waist"/>
   <origin xyz="0 0 0" rpy="0 0 0"/>
   <limit lower="-2.618" upper="2.618" effort="1000" velocity="100"/>
   <axis xyz="0 0 1"/>
</joint>
```

Joonis 41. RV-2AJ liigendi J1 info URDF failis.

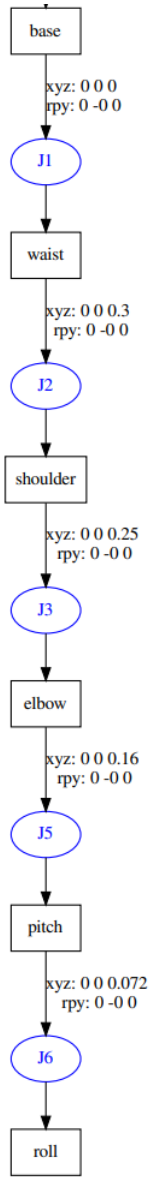
Sarnaselt telgedega tuleb määrata esimene liigend fikseeritult maailmaga seotuks. Selline liigend ei ole tegelikult liigend ja RViz ka ei käsitle seda kui liigendit, vaid kui jäika lüli. Teise liigendi(J1) alguspunkt peab olema sama, mis on Mitsubishi MELFA RV-2AJ kinemaatikamudelis J1 asukohaks. Liigendi tüüp on pöördliigend. Pöördliigendil on võimalik määrata pööramise ulatus mõlemas suunas nullpunktist radiaanides. Samuti tuleb neid parameetreid määrates määrata ka liigendi kiirus ja telge liigutav jõud. [29]

```
<joint name="J2" type="revolute">
  <parent link="waist"/>
  <child link="shoulder"/>
  <origin xyz="0 0 .3" rpy="0 0 0"/>
  <limit lower="-1.047" upper="2.094" effort="1000" velocity="100"/>
  <axis xyz="0 1 0"/>
</joint>

<joint name="J3" type="revolute">
   <parent link="shoulder"/>
   <child link="elbow"/>
   <origin xyz="0 0 .25" rpy="0 0 0"/>
   <limit lower="-1.92" upper="2.094" effort="1000" velocity="100"/>
   <axis xyz="0 1 0"/>
</joint>

<joint name="J5" type="revolute">
    <parent link="elbow"/>
    <child link="pitch"/>
    <origin xyz="0 0 .16" rpy="0 0 0"/>
    <limit lower="-1.57" upper="1.57" effort="1000" velocity="100"/>
    <axis xyz="0 1 0"/>
</joint>

<joint name="J6" type="revolute">
    <parent link="pitch"/>
    <child link="roll"/>
    <origin xyz="0 0 .072" rpy="0 0 0"/>
    <limit lower="-3.49" upper="3.49" effort="1000" velocity="100"/>
    <axis xyz="0 0 1"/>
</joint>
```

Joonis 42. RV-2AJ liigendite info URDF failis.

RViz-i kõik pikkuse mõõdud on meetrites. Seetõttu on vajalik URDF faili koostades konverteerida kõik mõõdud meetritesse ja ka näiteks .stl failide suurust vähendada. Iga liigendi puhul tuleb määrata temast hierarhias allpool ja ülevalpool olevad teljed. Liigendi paiknemine ruumis ning vajadusel pööramine ümber mõne oma telje. Samuti ka see, ümber mis telgede on võimalik liigendit pöörata.

Kui URDF faili koostamine on õnnestunud, siis on võimalik lasta ROSil koostada roboti lihtsustatud kinemaatikamudel nii nagu ROS seda mõistab. See näitab ära roboti kõik elemendid ja nende omavahelised seosed.



Joonis 43. Programmiga Graphiz koostatud RV-2AJ lihtsustatud kinemaatikaskeem.

Urdf failist graafilise kinemaatikaseemi tegemiseks tuleb käsurealt sisestada käsk kujul *rosrun urdf_parser urdf_to_graphiz Melfa.urdf*.

4.7 Paketi käivitusfailid

URDF mudeli käivitamiseks RViz keskkonnas tuleb koostada ka käivitusfail „.launch“. Käivitusfaili faili näol on tegu ROS jaoks ette antavate tegevuste ja nende järjekorra kirjeldusega. Sellega võib käivitada lisaks ka muid sõlmi või nende gruppe. Anda esmaseid parameetreid jms. Käivitusfail kirjutatakse samuti XML keeles. [30]

```
<?xml version="1.0"?>
<launch>
   <node name="AskPosition_Movement" pkg="ros_melfa" type="AskPosition_Movement.py"
output="screen" required="false" />
   <node name="MovementListener" pkg="ros_melfa" type="MovementListener.py"
output="screen" launch-prefix="gnome-terminal -e" required="false" />
   <arg name="model" />
   <param name="robot_description" command="cat '$(find
ros_melfa)/urdf/Melfa_multitool.urdf'" />
   <param name="use_gui" value="true"/>
   <node name="joint_state_publisher" pkg="ros_melfa" type="joint_state_publisher">

   </node>
   <node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher" />
   <node name="rviz" pkg="rviz" type="rviz" args="-d $(find ros_melfa)/rviz/Melfa.rviz"
required="true" />
</launch>
```

Joonis 44. RV-2AJ RViz käivitusfail.

Käivitusfaili kasutamise eelis on ka see, et kui faili käivitamisel ülemsõlm ei tööta, siis see käivitatakse. Samuti ROS parameetrite server ja logi. Kuna sõlmed AskPositionMovement ja MovementListener on vaja iga kord käivitada, siis on kasutajale mugavam, kui see on tehtud automaatseks. Sealjuures AskPositionMovementi käivitame samas aknas ning MovementListeneri jaoks teeme uue terminali akna. Roboti programmi siia nimekirja lisada ei ole mõtet ja seega see tuleb avada eraldi. Lisaks käsib see .launch fail ROSil laadida URDF mudel ROS parameetrite serverisse ning käivitada sõlmed joint_state_publisher ning robot_state_publisher. Viimaks tuleb käivitada RViz rakendus eelnevale seadistatud parameetritega.

4.8 Roboti juhtimisprogrammi koostamine

Kasutaja peab roboti korrektseks positsioneerimiseks vastavalt soovitavatele tegevustele looma programmi. Kuna RV-2AJ positsioneerimine toimub roboti baaskoordinaadistikus, siis tuleb seda programmi koordinaatide sisestamisel arvestada. See tähendab, et iga koordinaat ja positsioon kirjeldab roboti nullpunkti ja roboti pea

vahelist suhet. Tööriista keskpunkti järgi positsioneerides tuleks vastav ümberarvutus programmi sisse viia.
Programmi koostamiseks tuleb esmalt sisestada kõik kasutatavad koordinaadid massiivi.

```
P1  = CartesianMessage(215, -80, 300, 90, 180,  10)
J2 = JointMessage(  0,  0,  60, 0, 0,   100)
```

Joonis 45. Kasutaja poolt sisestatavate koordinaatide vorming programmis.

Seejärel tuleb sisestada positsioone postitavasse funktsiooni kõik positsioonid ja muud käsklused nende soovitava toimumise järjekorras. Iga positsiooni järel tuleb kasutada käsklust check_pos(), et tagada, et robot ei anna enne uut käsklust, kuni manipulaator on eelmisesse positsiooni jõudnud.

```
pubAbsXYZ.publish(P3)
   check_pos()
```

Joonis 46. Positsiooni näide roboti programmis.

Funktsioon postitab positsiooni olenevalt selle iseloomust ühte kolmest teemast. Need on /cartesianMovement, /jointMovement või /command_strings. Tagasiside saamiseks kuulab see teemat /positioned, kuhu annab peale positsioneerimist tõese väärtuse kontrolli ja tagasisidega tegelev sõlm.

```
pubAbsXYZ = rospy.Publisher('cartesianMovement', CartesianMessage, queue_size=1)
pubAbsJ = rospy.Publisher('jointMovement', JointMessage, queue_size=1)
pubCmdStr = rospy.Publisher('command_strings', String, queue_size=1)

rospy.Subscriber("positioned", Bool, positioned_callback, queue_size=1)
```

Joonis 47. Postitaja ja vastuvõtja näide roboti programmis.

Lisaks positsioneerimisele saab kasutada ka käsklusi haaratsi avamiseks, sulgemiseks, ning süsteemis kasutusel olevate väljundite juhtimiseks. Kuna nende puhul ei ole võimalik saada manipulaatorilt tagasisidet, tuleb need sisestada programmi koos ajalise viitega. See on vajalik, kuna liikudes ühest positsioonist teise hakkaks muidu robot liikuma enne käskluse teostamist.

```
# Start lambi sisselülitamine juhtpaneelil
    pubCmdStr.publish('M_OUT(0)=1')
    time.sleep(2.0)

# Kaane väljastamine detailimagasinist
    pubCmdStr.publish('M_OUT(9)=1')
    time.sleep(2.0)

# Haaratsi juhtimiskäske ei saa anda manipulaatori liikumise ajal
    pubCmdStr.publish('HCLOSE 1')
    time.sleep(3.0)
```

Joonis 48. Väljundite juhtimise näited roboti programmis.

Kasutaja ülesandeks on käsitsi või muul viisil positsioneerides leida kõik manipulaatorile vajalikud positsioonid ning kasutades neid ja teisi ette antud käskluste vorme koostada robotile programm, mis täidaks mõnda katsestendi tegevustest. Täpsed tegevused saab ette anda töö käigus. Koostatud pakett annab kasutajale reaalajas tagasiside positsioneerimise õnnestumise kohta ning visualiseerib roboti liikumist ka keskkonnas RViz.

KOKKUVÕTE

Käesoleva lõputöö eesmärk oli koostada robootika algõppes kasutatav materjal, mis võimaldaks anda ülevaate tarkvaraplatvormist ROS. Selleks on töö esimeses osas kirjeldatud selle tarkvaraplatvormi ajalugu ja teisi tähtsamaid sarnaseid lahendusi. Töö teine osa käsitleb ROS peamisi komponente ja nende kirjeldust ning omavahelisi seoseid. Töö kolmas osa annab lühikese ülevaate ROS rakendamisest liikurrobotitel ja sellega seonduvatest erisustest. Neljandas osas on käsitletud tarkvaraplatvormi integreerimist Elektroenergeetika ja mehhatroonika instituudi tootmise automatiseerimise õppelaboris paikneva Mitsubishi tööstusliku manipulaatoriga ja laboristendi juhtimiseks vajaliku tarkvara väljatöötamist. Loodud on võimalus manipulaatori ja robotsüsteemi juhtimiseks koos samaaegse visualiseerimisega. Samuti on koostatud näidisprogramm, mida on võimalik kasutada ROS teemaliste laboratoorsete tööde läbi viimiseks.

Lõputöö võib lugeda õnnestunuks, kuna kõik põhiülesanded, mis sai töö alustamisel seatud, said täidetud. Süsteemi üldiseid põhimõtteid kirjeldav osa annab lühida, aga ülevaatliku osa süsteemi põhilistest elementidest. Töö praktiline osa võimaldab juhtida robotit ja robotsüsteemi komponente ROS tarkvaraplatvormiga reaalsetes rakendustes.

Töö võimalike edasiarendustena võiks näha selle lõputöö käigus koostatud paketi kohandamist mõne teise ülikoolis paikneva tööstusroboti jaoks. Samuti saab visualiseerimise ja juhtimisliidest edasi arendada interaktiivseks tööks ROSi MoveIt programmiga.

SUMMARY

The goal of this thesis was to develop entry-level materials for study of software platform ROS. First part of thesis deals with the history of ROS and various similar projects. Second part of thesis deals with major components of ROS. Describes them and various connections between them. Third part describes, on a simple example, moving robots with ROS and main difficulties and differences involved. Last part of the work was the integration of ROS with Mitsubishi industrial manipulator, located at Industrial automation study laboratory of the Department of electrical power engineering and mechatronics at Tallinn University of Technology. This part also involved the development of software for control of laboratory set-up. This integration also included possibility for simultaneous real-time visualisation and control of the manipulator together with entire robot station. An example program code created is applicable in study laboratory for control of manipulator and robot station components with ROS software platform in real application.

The thesis can be considered successful, because all the main goals that were achieved. The descriptive part gives short but concise overview of the basic principles of ROS and its components. Practical section has created ability to use ROS on real life applications.

Possible developments of this work I would see an adaptation of this package to some other industrial robot at the University. The visualisation and control interface can be further developed for interactive work with ROS MoveIt program.

KASUTATUD KIRJANDUS

[1] L. Wood, „"Robot Operating System Market - Growth, Trends, COVID-19 Impact, and Forecasts (2021-2026)"," ResearchAndMarkets.com, 2021.

[2] Open Source Robotics Foundation, Inc., „ROS Platforms," Open Source Robotics Foundation, Inc, 05 02 2021. [Võrgumaterjal]. Available: http://wiki.ros.org/melodic/Installation/Platforms. [Kasutatud 18 05 2021].

[3] *https://www.theconstructsim.com/history-ros/.*

[4] YARP, „YARP ports," Mai 2021. [Võrgumaterjal]. Available: http://www.yarp.it/git-master/note_ports.html. [Kasutatud 18 05 2021].

[5] YARP, „Yarp with ROS," Mai 2021. [Võrgumaterjal]. Available: https://www.yarp.it/git-master/yarp_with_ros.html. [Kasutatud 18 05 2021].

[6] T. I. H. B. S. J. R. S. a. m. o. c. Peter Soetens, „OROCOS and ROS," 20 02 2009. [Võrgumaterjal]. Available: https://www.orocos.org/node/1117. [Kasutatud 18 05 2021].

[7] G. Borghesan, „Introduction to Orocos," 2020.

[8] D. T. L. H. Carol Fairchild, ROS Robotics By Example, Packt, 2016.

[9] Open Source Robotics Foundation, Inc, „ROS master," Open Source Robotics Foundation, Inc, 17 10 2015. [Võrgumaterjal]. Available: http://wiki.ros.org/rosmaster. [Kasutatud 18 05 2021].

[10] Robotics Back-End, „What is ROS topic?," Robotics Back-End, 2021. [Võrgumaterjal]. Available: https://roboticsbackend.com/what-is-a-ros-topic/. [Kasutatud 18 05 2021].

[11] Open Source Robotics Foundation, Inc, „JointState Message," Open Source Robotics Foundation, Inc, 01 2021. [Võrgumaterjal]. Available: http://docs.ros.org/en/api/sensor_msgs/html/msg/JointState.html. [Kasutatud 18 05 2021].

[12] Robotics Back-End, „What is ROS parameter?," Robotics Back-End, 05 2021. [Võrgumaterjal]. Available: https://roboticsbackend.com/what-is-a-ros-parameter/. [Kasutatud 18 05 2021].

[13] Robotics Back-End, „What is a ROS Service?," Robotics Back-End, 05 2021. [Võrgumaterjal]. Available: https://roboticsbackend.com/what-is-a-ros-service/. [Kasutatud 18 05 2021].

[14] Open Source Robotics Foundation, Inc, „A Brief History of Catkin," Open Source Robotics Foundation, Inc, 2014. [Võrgumaterjal]. Available: https://catkin-tools.readthedocs.io/en/latest/history.html. [Kasutatud 18 05 2021].

[15] *https://husarion.com/tutorials/ros-tutorials/3-simple-kinematics-for-mobile-robot/.*

[16] Open Source Robotics Foundation, Inc, „Adding a frame to tf," Open Source Robotics Foundation, Inc, 04 2021. [Võrgumaterjal]. Available: http://wiki.ros.org/tf/Tutorials/Adding%20a%20frame%20%28C%2B%2B%29. [Kasutatud 18 05 2021].

[17] D. J. Wyss-Gallifent, „Gimbal lock," University of Maryland, 05 2021. [Võrgumaterjal]. Available: http://www.math.umd.edu/~immortal/MATH431/lecturenotes/ch_gimballock.pdf. [Kasutatud 18 05 2021].

[18] G. McComb, „WAYS TO MOVE YOUR ROBOT," 2014.

[19] *https://www.sciencedirect.com/science/article/abs/pii/S0957415808000512.*

[20] R. Yehoshua, „October 2016 ROS Lecture 7 ROS navigation stack," 2016.

[21] Gaitech , „Gaitech EDU," Gaitech, 2016. [Võrgumaterjal]. Available: https://edu.gaitech.hk/turtlebot/map-navigation.html. [Kasutatud 18 05 2021].

[22] R. K. Megalingam, „ROS based Autonomous Indoor Navigation," 2018.

[23] D. R. Hessmer, „2d SLAM with ROS and Kinect," Aprill 2011. [Võrgumaterjal]. Available: http://www.hessmer.org/blog/2011/04/10/2d-slam-with-ros-and-kinect/. [Kasutatud 18 05 2021].

[24] T. P. Martin Meloun, „Inverse Kinematics for a General 6R Manipulator," Czech Technical University, Prague, 2013.

[25] P. H. d. S. R. R. C. Alexandre Magno de S Thiago Filho, „ROS communnication with robot Mitsubish Melfa RV-2AJ," Universidade Federal de Ouro, 15 12 2018. [Võrgumaterjal]. Available: https://github.com/alexandremstf/ros_mitsubishi. [Kasutatud 01 02 2021].

[26] JetBrains s.r.o., „jetbrains.com," JetBrains s.r.o., 08 Märts 2021. [Võrgumaterjal]. Available: https://www.jetbrains.com/help/clion/cmakelists-txt-file.html. [Kasutatud 18 05 2021].

[27] C. Z. M. K. Roman Szewczyk, Automation 2017, Warsaw, 2017.

[28] Open Source Robotics Foundation, Inc, „wiki.ros.org," Open Source Robotics Foundation, Inc, 2021. [Võrgumaterjal]. Available: http://wiki.ros.org/urdf/Tutorials/Create%20your%20own%20urdf%20file. [Kasutatud 18 Mai 2021].

[29] „ABB IRB1200 ROS support," 12 Märts 2020. [Võrgumaterjal]. Available: https://github.com/ros-industrial/abb_experimental/tree/kinetic-devel/abb_irb1200_support/urdf. [Kasutatud 21 05 2021].

[30] Open Source Robotics Foundation, Inc, „ROS answers," Open Source Robotics Foundation, Inc, 25 Juuli 2014. [Võrgumaterjal]. Available: https://answers.ros.org/question/187815/how-do-i-display-my-own-urdf-in-rviz/. [Kasutatud 18 05 2021].

[31] 02 2019. [Võrgumaterjal]. Available: https://www.bccresearch.com/market-research/engineering/robotics.html.

[32] [Võrgumaterjal]. Available: https://www.theconstructsim.com/history-ros/.

[33] B. G. K. C. J. F. T. F. J. L. E. B. R. W. A. N. Morgan Quigley, „ROS: an open-source Robot Operating System," Computer Science Department, Stanford University, Stanford, CA, 2009.

[34] Open Source Robotics Foundation, Inc, „JointState Message," Open Source Robotics Foundation, Inc, 13 01 2021. [Võrgumaterjal]. Available: http://docs.ros.org/en/api/sensor_msgs/html/msg/JointState.html. [Kasutatud 18 05 2021].

[35] Open Source Robotics Foundation, Inc, „ROS Melodic Morenia," Open Source Robotics Foundation, Inc, 14 08 2018. [Võrgumaterjal]. Available: http://wiki.ros.org/melodic. [Kasutatud 18 05 2021].

LISAD

Lisa 1 Tarkvarapaketi paigaldamine

Järgnev juhend on väikeste muudatustega rakendatav ROS Melodic Morenia või Noetic Ninjemys paigaldamiseks mitmetes Debiani põhistes LINUX operatsioonisüsteemides sh Ubuntu. Käsud tuleb täita mitte ainult administraatori kasutaja poolt vaid ka administraatoriõigustes, lisades ridade ette käsu „sudo" (see on eriti oluline Ubuntu puhul).

ROS paigaldus
Lisa ROS allikate nimekirja:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

Lisa serveri võti:

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

Uuenda kõik tarkvaraallikad:

```
sudo apt update
```

Paigalda ROS:

```
sudo apt install ros-noetic-desktop-full
```

või kui soovitakse vanemat versiooni, siis ros-melodic-desktop-full.

Paljud komponendid on kirjutatud programmeerimiskeeles Python ja vajavad tööks selle teeke. Paigaldusprotsess on palju kiirem kui eelnevalt on olemas üldised tarkvaraarenduses kasutatavad paketid sh automake, build-essential, binutils, gcc, g++, cmake, doxygen, python3, libboost-all-dev, qt5-assistant, python3-nuphy, python3-automat, python3-opencv, ignition-tools, libignition-common3 ja mitmed teised nendega seotud paketid.

Paigalda ROS jaoks vajalikud lisad:

```
sudo apt install python3-rosinstall python3-rosinstall-generator python3-wstool
```

vanemas versioonis vastavalt: sudo apt install python-rosinstall python-rosinstall-generator python-wstool

```
sudo apt install python3-rosdep
```

vanemas versioonis vastavalt: sudo apt install python-rosdep

Rosdep on ROS pakettide paigalduse abivahend, mis paigaldab iga paketiga koos automaatselt muud toimiseks vajalikud lisapakid.
Tee rosdep algseadistus:

```
sudo rosdep init
rosdep update
```

Kasutamiseks tuleb paigaldada ka projektifailide halduse süsteem Catkin.

Catkin-i paigaldus

Tee tavakasutajale catkin-i jaoks kataloog(tavaliselt kasutatakse nime catkin_ws):

```
mkdir -p ~/catkin_ws/src
```

Mine kataloogi:

```
cd ~/catkin_ws/
```

Loo catkin kataloogisüsteem:

```
catkin_make
```

Käsuviibalt kasutamise lihtsustamiseks tuleks lisada vajalikud viited tavakasutaja faili bashrc. Skript bashrc, käivitatakse iga kord kui kasutaja avab uue terminali akna. Nii leitakse käsurealt käsku sisestades süsteemis kiiremini ROSiga seotud komponendid:

```
echo "source /opt/ros/noethic/setup.bash" >> ~/.bashrc
```

või vanemas versioonis echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc

Et ROS kasutamine oleks tulevikus lihtsam on siinkohal mõistlik kohe lisada viide ka catkinile Ubuntu bashrc faili(bashrc on skript, mis käivitatakse iga kord kui kasutaja avab uue terminali akna). Nii oskab süsteem ROSiga seotud käskudele alati õigesti vastata:

```
echo 'source ~/catkin_ws/devel/setup.bash' >> ~/.bashrc
```

Peale seda tuleb taaskäivitada tavakasutaja bash :

```
exec bash
```

Alternatiiviks eelnevale on peale iga uue akna avamist sisestada järgmised käsud:

```
cd ~/catkin_ws
```

```
source devel/setup.bash
```

Mitsubishi Melfa ROS liidesepaketi paigaldus

Antud ROS pakett on koostatud ROS Melodic-u jaoks.

Salvesta projekt oma Catkin-i src kataloogi: /catkin_ws/src

Liigu Catkin-i töölauakataloogi:

```
cd ~/catkin_ws
```

Kompileeri pakett:

```
catkin_make
```

See pakett kasutab jadaporti ühendumiseks. Jadaport on aga süsteemi vaikeseadistustes tavakasutajale keelatud.

Kasutajale „kasutaja" jadapordi ligipääsu alaliseks lisamiseks:

```
sudo adduser kasutaja dialout
```

Kasutaja ajutiseks lisamiseks:

```
sudo chown :kasutaja /dev/ttyUSB0
```

Lisa 2 Programmikood

Lisa 2.1 Roboti sideprogrammi fail Askposition_Movement.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import os
path = os.getcwd() + '/src/ros_melfa/src'

import sys
sys.path.insert(0, path)

import time

import Communication
#comm_channel=Communication.Communication('/dev/ttyS0')
comm_channel=Communication.Communication('/dev/ttyUSB0')

import rospy

import numpy as np  # Matriksarvutused

import tf       # Kinemaatilised teisendused

from tf.transformations import euler_from_quaternion
from ros_melfa.msg import JointMessage, CartesianMessage
from sensor_msgs.msg import JointState
from std_msgs.msg import Header, String, Float64

# Robotist andmete küsimisel kasutatavad liigendite nimed
joint_names = ['J1', 'J2', 'J3', 'J5', 'J6']

# Defineerime Mitsubishi MELFA RV-2AJ teljed ruumis maatriksitena
TDH1 = np.matrix('1 0 0 0; 0 0 1 0; 0 -1 0 300; 0 0 0 1');
TDH2 = np.matrix('0 1 0 0; -1 0 0 -250; 0 0 1 0; 0 0 0 1');
TDH3 = np.matrix('1 0 0 160; 0 1 0 0; 0 0 1 0; 0 0 0 1');
TDH5 = np.matrix('0 0 1 0; 1 0 0 0; 0 1 0 0; 0 0 0 1');
TDH6 = np.matrix('1 0 0 0; 0 1 0 0; 0 0 1 72; 0 0 0 1');

TDH_tool = None

active_cmd_string = None
active_cart_message = None
active_jointmessage = None

query_state = 0

#märgijadade massiivi saatmine
def SendCommandList(CmdStr):
    comm_channel.send(CmdStr)

def exec_command(CmdStr):
    time.sleep(0.1)
    commands = []
    commands.append('CNTLON')

    if ('HOPEN' in CmdStr) or ('HCLOSE' in CmdStr) or ('M_OUT' in CmdStr):
        print ('IO or Gripper command')
        comm_channel.readAll()
        cmd1='EXEC' # haaratsikäskudele lisada ette EXEC
        commands.append(cmd1 + str(CmdStr))
```

```
    else:
        comm_channel.readAll()
        commands.append(str(CmdStr))

    commands.append('CNTLOFF')
    SendCommandList(commands)
    time.sleep(0.1)

def SendJointPositioningCommand(j1, j2, j3, j5, j6, speed) :
    JointPositionString = '(' + str(j1) + ',' + str(j2) + ',' + str(j3) + ', 0.0,' + str(j5) + ','
+ str(j6) + ')'
    commands = []
    commands.append('CNTLON')
    commands.append('EXECJOVRD ' + str(speed))
    commands.append('EXECJCOSIROP = ' + JointPositionString)
    commands.append('EXECMOV JCOSIROP')
    commands.append('CNTLOFF')
    SendCommandList(commands)

def SendCartesianPositioningCommand(x, y, z, a, b, speed) :
    CartesianPositionString = '(' + str(x) + ',' + str(y) + ',' + str(z) + ',' + str(a) + ',' +
str(b) + ', 0.000)(6,0)'
    commands = []
    commands.append('CNTLON')
    commands.append('EXECJOVRD ' + str(speed))
    commands.append('EXECPCOSIROP = ' + CartesianPositionString)
    commands.append('EXECMOV PCOSIROP')
    commands.append('CNTLOFF')
    SendCommandList(commands)

def GetJointPosition() :
    exec_command('JPOSF')
    return comm_channel.readUntil('\n', 100)

def GetCartesianPosition() :
    exec_command('PPOSF')
    return comm_channel.readUntil('\n', 100)

cartesianPositionDict=None
jointPositionDict=None

def timer_callback(event):
    global active_cmd_string
    global active_cart_message
    global active_jointmessage
    global query_state
    global cartesianPositionDict
    global jointPositionDict

    if active_cmd_string:
        print ("Executing command: ", active_cmd_string)
        exec_command(active_cmd_string)
        active_cmd_string = None

    elif active_cart_message:
        print ("Positioning using cartesian")

        SendCartesianPositioningCommand(active_cart_message.x, active_cart_message.y,
active_cart_message.z, active_cart_message.a, active_cart_message.b,
active_cart_message.speed)

        active_cart_message = None
        time.sleep(0.2)
```

```
    elif active_jointmessage:
        print ("Positioning using joint angles")
        SendJointPositioningCommand(active_jointmessage.j1, active_jointmessage.j2,
active_jointmessage.j3, active_jointmessage.j5, active_jointmessage.j6,
active_jointmessage.speed)
        active_jointmessage = None
        time.sleep(0.2)

    else:
        print ("Updating the model")
        pubJ = rospy.Publisher('joint_states', JointState, queue_size=10)
        pubC = rospy.Publisher('cartesian_state', CartesianMessage, queue_size=10)

        if query_state == 0:
            str111 = str(GetJointPosition())
            strArr = str111.split(';')
            strLst = strArr[1:12:2];
            strLst[3]='0.0'
            gamma = [float(i) for i in strLst]
            print (gamma)
            query_state = 1

        if query_state == 1:
            str222 = str(GetCartesianPosition())
            strArr = str222.split(';')
            strLst = strArr[1:10:2];
            print (strLst)
            Pxyz = [float(i) for i in strLst]

            query_state = 0

        if gamma[0:2] and gamma[4:5]:
            print (jointPositionDict)

            #Teisendame liigendite asendite nurgad kraadidest radiaanidesse
            gamma1 = np.deg2rad(gamma[0])
            gamma2 = np.deg2rad(gamma[1])
            gamma3 = np.deg2rad(gamma[2])
            gamma5 = np.deg2rad(gamma[4])
            gamma6 = np.deg2rad(gamma[5])

            position = JointState()

            position.header = Header()
            position.header.stamp = rospy.Time.now()

            position.name = joint_names
            position.position = [gamma1, gamma2, gamma3, gamma5, gamma6]
            position.velocity = []
            position.effort = []
            pubJ.publish(position)


            #Koostame pööramismaatriksid
            Rz1 = tf.transformations.rotation_matrix(gamma1, (0,0,1) );
            Rz2 = tf.transformations.rotation_matrix(gamma2, (0,0,1) );
            Rz3 = tf.transformations.rotation_matrix(gamma3, (0,0,1) );
            Rz5 = tf.transformations.rotation_matrix(gamma5, (0,0,1) );
            Rz6 = tf.transformations.rotation_matrix(gamma6, (0,0,1) );

            #Liigendite asendid maatriksitena korrutatult
            P1 = np.dot (Rz1, TDH1);       #P1 = Rz1*TDH1
            P2 = np.dot(np.dot(P1, Rz2), TDH2);     #P2 = P1*Rz2*TDH2
            P3 = np.dot(np.dot(P2, Rz3), TDH3);
```

```
            P5 = np.dot(np.dot(P3, Rz5), TDH5);
            P6 = np.dot(np.dot(P5, Rz6), TDH6);

            #if TDH_tool:
            #   P_tool = np.dot(P6, TDH_tool);

            #Konverteerime maatriksi kvaterniooniks
            Q6 = tf.transformations.quaternion_from_matrix(P6)
            (R,P,Y) = euler_from_quaternion(Q6)
            b_r = np.rad2deg(R)

            a = gamma1 + np.rad2deg(P)
            print ("Tool0 euler angles (R,P,Y):", b_r, np.rad2deg(P), np.rad2deg(Y))

            # märgikorrektsioon ?!
            if (b_r<0):
               b_r=-b_r

            # x, y, z, a, b, 0
            cartesian_from_model = CartesianMessage(P6.item(0,3), P6.item(1,3),
P6.item(2,3), a, b_r, 0)
            cartesian_from_robot = CartesianMessage(Pxyz[0], Pxyz[1], Pxyz[2], Pxyz[3],
Pxyz[4], 0)
            pubC.publish(cartesian_from_robot)
            print ("Model:", cartesian_from_model.x, cartesian_from_model.y,
cartesian_from_model.z, cartesian_from_model.a, cartesian_from_model.b)
            print ("Robot:", cartesian_from_robot.x, cartesian_from_robot.y,
cartesian_from_robot.z, cartesian_from_robot.a, cartesian_from_robot.b)

def command_string_callback(dat):
   global active_cmd_string
   active_cmd_string = dat.data

def cartesian_positioning_callback(dat):
   global active_cart_message
   active_cart_message = dat

def joint_positioning_callback(dat):
   global active_jointmessage
   active_jointmessage = dat

def talker():
   rospy.init_node('joint_state_publisher', anonymous=False)

   # Teated roboti juhtimisprogrammist
   rospy.Subscriber("command_strings", String, command_string_callback, queue_size=1)
   rospy.Subscriber("cartesianMovement", CartesianMessage,
cartesian_positioning_callback, queue_size=1)
   rospy.Subscriber("jointMovement", JointMessage, joint_positioning_callback,
queue_size=1)

   timer = rospy.Timer(rospy.Duration(0.1), timer_callback)
   rospy.spin()
   timer.shutdown()



if __name__ == '__main__':
    try:
       talker()
    except rospy.ROSInterruptException:
        pass
```

Lisa 2.2 Roboti liikumise jälgimisprogrammi fail MovementListener.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import os
path = os.getcwd() + '/src/ros_melfa/src'

import sys
sys.path.insert(0, path)

#Kraadidest radiaanideks teisendamiseks
import numpy as np

import rospy
from ros_melfa.msg import CartesianMessage, JointMessage
from sensor_msgs.msg import JointState

from std_msgs.msg import Bool, Float64, Header

import time # meetodile 'now()'

pos_command = None
pos_state = None

joint_pos_command = None
joint_pos_state = None

def print_position():
    if joint_pos_state:
        print ("Joints:", np.degrees(joint_pos_state.position))
    if pos_state:
        print ("State (Tool0): ", pos_state.x, pos_state.y, pos_state.z, pos_state.a,
pos_state.b)


def timer_callback(event):
    global pos_command
    global pos_state

    global joint_pos_command
    global joint_pos_state

    z=Bool()
    pub_pos_OK = rospy.Publisher('positioned', Bool, queue_size=1)

    if pos_command and pos_state:
        # Võrreldakse kahe samatüübilise andmestruktuuri elemente
        # Kiirust ei võrrelda
        while pos_command and pos_state and (pos_state.x != pos_command.x
or pos_state.y != pos_command.y or pos_state.z != pos_command.z or
pos_state.a != pos_command.a or pos_state.b != pos_command.b):

            z.data=False
            pub_pos_OK.publish(z)

        print ("Positioned OK")
        pos_command=None
```

```
        z.data=True
        pub_pos_OK.publish(z)
        print_position()


    if joint_pos_command and joint_pos_state:

        while joint_pos_command and joint_pos_state and
(joint_pos_state.position[0] != joint_pos_command.position[0] or
joint_pos_state.position[1] != joint_pos_command.position[1] or
joint_pos_state.position[2] != joint_pos_command.position[2] or
joint_pos_state.position[3] != joint_pos_command.position[3] or
joint_pos_state.position[4] != joint_pos_command.position[4]):

            z.data=False
            pub_pos_OK.publish(z)

        print ("Positioned OK")
        joint_pos_command=None
        z.data=True
        pub_pos_OK.publish(z)
        print_position()


def state_callback(data):
    global pos_state

    if not joint_pos_command:
        pos_state = data

def command_callback(data):
    global joint_pos_command
    global pos_command
    pos_command = data
    joint_pos_command = None
    print ("\nCommand (Tool0): ", pos_command.x, pos_command.y,
pos_command.z, pos_command.a, pos_command.b)


def joint_states_callback(data):
    global joint_pos_state
    joint_pos_state = data

def joint_command_callback(data):
    global pos_command
    global joint_pos_command

    if not pos_command:
        # Soovitud asend, millisse soovitakse positsioneerida
        joint_pos_command = JointState()
        joint_pos_command.header = Header()
        joint_pos_command.header.stamp = rospy.Time.now()
        joint_pos_command.name = ['J1', 'J2', 'J3', 'J5', 'J6']
        #teisendame käsuga saadud nurgad kraadidest radiaanidesse
        angles = [data.j1, data.j2, data.j3, data.j5, data.j6]

        gamma1 = np.deg2rad(angles[0])
```

```
        gamma2 = np.deg2rad(angles[1])
        gamma3 = np.deg2rad(angles[2])
        gamma5 = np.deg2rad(angles[3])
        gamma6 = np.deg2rad(angles[4])

        joint_pos_command.position = np.array([gamma1, gamma2, gamma3,
gamma5, gamma6], np.float64)

        joint_pos_command.velocity = []
        joint_pos_command.effort = []
        pos_command = None
        print ("\nJoint command: ", angles)


def listener():
    rospy.init_node('listenercartesian', anonymous=False)

    rospy.Subscriber("cartesianMovement", CartesianMessage, command_callback,
queue_size=1)
    rospy.Subscriber("cartesian_state", CartesianMessage, state_callback,
queue_size=1)

    rospy.Subscriber("jointMovement", JointMessage, joint_command_callback,
queue_size=1000)
    rospy.Subscriber("joint_states", JointState, joint_states_callback,
queue_size=1000)

    timer = rospy.Timer(rospy.Duration(0.1), timer_callback)

    rospy.spin()

    timer.shutdown()



if __name__ == '__main__':
    listener()
```

Lisa 2.3 Roboti juhtimisprogrammi fail roboti_programm.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import os
path = os.getcwd() + '/src/ros_melfa/src'

import sys
sys.path.insert(0, path)

# Roboti juhtimisprogammi ülesanne on liigutada roboti manipulaatorit erinevate sihtasendite vahel
# kasutades selleks teadete saatmist ristkoordinaatides või liigendite koordinaatides
# Teate saatmise protseduuride parameetrid on järgnevad:
#   x, y, z, a, b, speed
#   j1, j2, j3, j5, j6, speed
# Käesolevas näites toimub ristkoordinaatidega postitsioneerimine roboti baaskoordinatsüsteemis.
# Robotit on võimalik positsiooneerida ka tööriista koorindaatsüsteemis
# või tööobjekti suhtes baaskoordinaatsüsteemis

# Teate saatmise protseduurid CartesianMessage ja JointMessage plokeerivad programmi töö seini kuni
# etteantud positsioon on saavutatud (või etteantud aeg möödunud).
import rospy
from ros_melfa.msg import CartesianMessage
from ros_melfa.msg import JointMessage


#Ajaliste viivituste etteandmine programmis, seda läheb vaja ka haaratsi juhtimisel, sest seda robotikontroller ei juhi
import time

from std_msgs.msg import String, Bool

# Cartesian targets

#robot.moveCartesianPosition(P1)
P1  = CartesianMessage(215, -80, 300, 90, 180,  10)
P2  = CartesianMessage(215, -80, 325, 90, 180,  10)
P3  = CartesianMessage(215, -30, 325, 90, 180,  10)
P4  = CartesianMessage(215, -80, 325, 90, 180,  10)
P5  = CartesianMessage(215, -80, 350, 90, 180,  10)
P6  = CartesianMessage(215, -30, 350, 90, 180,  10)
P7  = CartesianMessage(215, -80, 350, 90, 180, 100)
P8  = CartesianMessage(215, -80, 300, 90, 180, 100)
P9  = CartesianMessage(215, -30, 300, 90, 180, 100)
P10 = CartesianMessage(215,  20, 300, 90, 180, 100)
P11 = CartesianMessage(215,  20, 350, 90, 180, 100)
P12 = CartesianMessage(215,  45, 325, 90, 180, 100)
P13 = CartesianMessage(215,  70, 350, 90, 180, 100)
P14 = CartesianMessage(215,  70, 300, 90, 180, 100)
```

```
# Joint targets
J2 = JointMessage(  0,  0,  60, 0, 0,      100)
J3 = JointMessage(  0,  90,  0, 0, 0,      50)
J4 = JointMessage(  0,  90, -90, 0, 0,     50)
J5 = JointMessage(  0,  90, -90, 90, 0,    50)
J6 = JointMessage(  0,  90, -90, -90, 0,   50)
J7 = JointMessage(  0,  90, -90, 45, 0,    50)
J8 = JointMessage(-150, 90, -90, 45, 0,    50)
J9 = JointMessage(-150, 45, -90, 45, 0,    50)

J0_calib_pos = JointMessage(0, 0, 0, 0, 0,  50)

J0_home_pos = JointMessage(0, 45, 90, 45, 0,   50)
Jhome1 = JointMessage(0, 45, 90, 45, 45,   50)
Jhome2 = JointMessage(0, 45, 90, 45, 90,   50)
Jhome3 = JointMessage(0, 45, 90, 45, -45,  50)
Jhome4 = JointMessage(0, 45, 90, 45, -90,  50)

posOK=Bool(False)

def positioned_callback(data):
    global posOK
    posOK = data.data

def check_pos():
    global posOK
    time.sleep(0.1)
    while (posOK!=True):
        time.sleep(0.2)
    posOK=False


def talker():
    global posOK
    pubAbsXYZ = rospy.Publisher('cartesianMovement', CartesianMessage,
queue_size=1)
    pubAbsJ = rospy.Publisher('jointMovement', JointMessage, queue_size=1)
    pubCmdStr = rospy.Publisher('command_strings', String, queue_size=1)

    rospy.Subscriber("positioned", Bool, positioned_callback, queue_size=1)

    rospy.init_node('sample1', anonymous=True)

    # Start lambi sisselülitamine juhtpaneelil
    pubCmdStr.publish('M_OUT(0)=1')
    time.sleep(2.0)

    # Kaane väljastamine detailimagasinist
    pubCmdStr.publish('M_OUT(9)=1')
    time.sleep(2.0)
    pubCmdStr.publish('M_OUT(9)=0')
    time.sleep(2.0)

    #Roboti liigutamise eeltingimuseks on et servomootorid peavad olema
käivitatud - kõik manipulaatorid annavad vastasel juhul vea.
    pubCmdStr.publish ('SRVON')
    time.sleep(5.0)
```

```
    pubAbsXYZ.publish(P1)

    #rospy.wait_for_message('positioned', Bool, timeout=None) #, timeout=10
    check_pos()

    pubAbsXYZ.publish(P2)
#    check_pos()

    time.sleep(3.0)

    # Haaratsi juhtimiskäske ei saa anda manipulaatori liikumise ajal
    pubCmdStr.publish('HCLOSE 1')
    time.sleep(3.0)

    pubCmdStr.publish('HOPEN 1')
    time.sleep(3.0)

    pubAbsJ.publish(J0_home_pos)
    check_pos()

    pubAbsJ.publish(Jhome1)
    check_pos()
    pubAbsJ.publish(Jhome2)
    check_pos()
    pubAbsJ.publish(Jhome3)
    check_pos()
    pubAbsJ.publish(Jhome4)
    check_pos()

    pubAbsXYZ.publish(P3)
    check_pos()
    pubAbsXYZ.publish(P4)
    check_pos()
    pubAbsXYZ.publish(P5)
    check_pos()
    pubAbsXYZ.publish(P6)
    check_pos()
    pubAbsXYZ.publish(P7)
    check_pos()
    pubAbsXYZ.publish(P8)
    check_pos()
    pubAbsXYZ.publish(P9)
    check_pos()
    pubAbsXYZ.publish(P10)
    check_pos()
    pubAbsXYZ.publish(P11)
    check_pos()
    pubAbsXYZ.publish(P12)
    check_pos()
    pubAbsXYZ.publish(P13)
    check_pos()
    pubAbsXYZ.publish(P14)
    check_pos()

    pubAbsJ.publish(J0_calib_pos)
    check_pos()
```

```
    pubAbsJ.publish(J2)
    check_pos()
    pubAbsJ.publish(J3)
    check_pos()
    pubAbsJ.publish(J4)
    check_pos()
    pubAbsJ.publish(J5)
    check_pos()
    pubAbsJ.publish(J6)
    check_pos()
    pubAbsJ.publish(J7)
    check_pos()
    pubAbsJ.publish(J8)
    check_pos()
    pubAbsJ.publish(J9)
    check_pos()
    pubAbsJ.publish(J0_calib_pos)
    check_pos()
    pubAbsJ.publish(J0_home_pos)
    check_pos()

    time.sleep(2.0)

    # Start lambi väljalülitamine juhtpaneelil
    pubCmdStr.publish('M_OUT(0)=0')
    time.sleep(3.0)

    pubCmdStr.publish ('SRVOFF')
    time.sleep(2.0)


if __name__ == '__main__':
    talker()
```