TALLINNA TEHNIKAÜLIKOOL
INSENERITEADUSKOND
Virumaa kolledž

# Robotmanipulaatori loomine ROS-i raamistikul

## Creation of a robotic manipulator based on the ROS control system

TELEMAATIKA- JA ARUKAD SÜSTEEMID ÕPPEKAVA LÕPUTÖÖ

Üliõpilane: Jevgeni Kostenko
Üliõpilaskood: 182726
Juhendaja: Sergei Pavlov

Kohtla-Järve, 2021

# SISUKORD

# EESSÕNA

Lõputöö teema „Robotmanipulaatori loomine ROS-i raamistikul” sõnastas lektor Sergei Pavlov, lähtudes minu tööst õppeaines „Robottehnilised süsteemid” (RAA0520). Õppeine kaitsmisel esitasin robotmanipulaatori projekti koos kuue vabadusastmega (inglise keeles *6 Degrees of Freedom, 6DoF*). Tehtud tööd otsustati edasi arendada ehk robotmanipulaatori arendamist lõputööna jätkata.

Robotmanipulaator on robotkäsi, mis koosneb kuuest samm-mootorist, robotkäest, küünarnukist, randmest, viiest liigendist ja pöörlevast alusest. Projekti arendamine ja modelleerimine viiakse läbi iseseisvalt, modelleerimine, detailide planeerimine ja komponentide valik toimub nullist. Robotmanipulaatori eripäraks on see, et tema peamised kereosad valmistatakse plastikust 3D-printimise abil.

Projekti olulisteks osadeks on: odav omahind (võrreldes sarnaste projektidega, mis on avatud lähtekoodiga), universaalsed mehhanismid (korduvad), moodulsus (mehhanismide sõltumatus), kulumiskindlus (laagrid, hammasrataste määrimine), täpsus (samm-mootorid), vaba juhtimistarkvara (ROS, STM32CubeIDE, Linux OS ja muud kogud).

# LÜHENDITE JA TÄHISTE LOETELU

**DIY** – (*Do It Yourself*) – omatehtud, ("tee seda ise"), kasutusel ka akronüümina DIY, on eri ringkondades ja subkultuurides laialt levinud mõiste.

**ROS** – (*Robot Operating System*) – on tarkvararaamistik robotite kasutamiseks, integreerimiseks ja arendamiseks.

**RViz** (*ROS Visualization*) – ROS-i programm, mis võimaldab visualiseerida erinevaid sõnumeid, mudeleid ja tähiseid.

**API** (*Application Programming Interface*) ehk rakendusliides võimaldab suhtlust erinevate tarkvarakomponentide vahel kasutades eeldefineeritud eeskirju.

**STM32** – *STMicroelectronics*'i toodetud 32-baidiste mikrokontrollerite perekonnast.

**SDK** (*Software Development Kit*) ehk arendustarkvara, kogum erinevatest tööriistadest, mille abil luuakse tarkvara kindla platvormi jaoks. Tarkvarakomponentide vahel kasutatakse eeldefineeritud eeskirju.

**OS** – baasoperatsioonisüsteem Linuxi perekonnast.

**URDF** (*Unified Robot Description Format*) - on XML-i spetsifikatsioon roboti kirjeldamiseks.

# SISSEJUHATUS

Käesoleval ajal robotite teema on saamas lahutamatuks osaks meie igapäevasest elust Maal. Inimese loodud keskkond — tehiskeskkond — on juba valmis asendama inimtööd (sh rutiinset), samal ajal tõstes tootlikkust, täpsust ja tehtud töö kvaliteeti.

Antud lõputöös vaadeldakse võimalust luua robotmanipulaator oma kätega ja kasutusele võtta ROS-i raamistikul põhinev avatud lähtekoodiga tööstuslik juhtimissüsteem. ROS-i süsteem on välja töötatud 2007. aastal Stanfordi ülikooli tehisintellekti laboris nimetusega *switchyard* [1]. ROS pakub standardseid operatsioonisüsteemi teenuseid nagu riistvara abstraktsioon, seadmete madalatasemeline kontroll, tihti kasutatavate funktsioonide rakendamine, protsessidevaheliste sõnumite edastamine ja paketihaldus.

Omatehtud robotmanipulaatori projekteerimise väljakutse seisneb nii mehaanikadetailide valmistamises kui ka robotmanipulaatori juhtimiseks vajaliku tarkvara loomises.

Lõputöö eesmärk on programmjuhtimise valmislahenduste kasutamise võimaluste kirjeldamine. Samuti on vaatluse all robotmanipulaatori detailide loomine ja valmistamine 3D-printimise abil. Antud projekti oluline osa on ka sellise tõstejõuga robotmanipulaatori loomine, mis suudaks teisaldada kuni 1-kiloseid ja üle selle kaaluvaid esemeid.

# 1. ROBOTMANIPULAATORI MEHAANILINE OSA

Uurimuse plaan jagatakse mitmeks punktiks:

- Projekti mehaanilise osa sarnaste olemasolevate lahenduste analüüs ja uue lahenduse optimaalne valik. Vaatluse all on robotmanipulaatori üksikasjad — kere materjal, pöörlevate sõlmede laagrite tüübid, ajami tüübid ning mehaanilise jõuülekande ja mikrokontrollerite tüüp.
- Tarkvaratööriistade analüüs robotmanipulaatori 3D-detailide loomiseks.
- Robotmanipulaatori juhtimise tarkvara analüüs.
- ROS-süsteemi kasutuselevõtmine ja seadistamine.

## 1.1 Olemasolevate lahenduste analüüs

Ülevaade olemasolevatest avalikest allikatest pärit lahendustest pakub palju omatehtud (inglise keeles *DIY*) robotmanipulaatoreid. Selle projekti lahenduste valiku peamiseks määravaks teguriks on 3D-printimise abil valmistatud robotmanipulaator, kuna see tehnoloogia võimaldab luua keerulisi mehhanisme ilma spetsiaalsete materjalide töötlemise seadmeteta (näiteks, ilma selliste masinateta nagu treipink, puurpink, freespink jne). Kaasaegsed materjalid, mida kasutatakse robotmanipulaatori detailide trükkimisel, võimaldavad samuti saavutada robotmanipulaatori kandejõuks vajalikku paindlikkust ja tugevust.

TOP 10 analoogset avalikku allikatest pärit projekti [vt Tabel 1] [2]:

*Tabel 1 TOP 10 omatehtud (DIY) robotmanipulaatorit*

| **Nimetus** | **3D printimine** | **Kontroller** | **X-vabadusastet** | **Kandevõime, kg** | **Hind EUR** |
|---|---|---|---|---|---|
| BCN3D Moveo | Jah | Arduino | 4DOF | 0.5 | DIY |
| Thor by AngelLM | Jah | Arduino Mega | 6DOF | 0.75 | DIY |
| EEZYbotARM MK2 | Jah | Mini Maestro USB Servo | 4DOF | 0.5 | DIY |
| Zortrax Arm | Jah | Arduino Mega | 5DOF | 0.1 | DIY |
| LittleArm | Jah | Arduino Uno | 3DOF | 0.25 | DIY |
| **Hoelldorfer** | **Jah** | **Arduino** | **6DOF** | **2** | **DIY** |
| OWI Arm Edge | Jah | Arduino | 5DOF | 0.1 | 42 |
| MeArm | Jah | Raspberry Pi | 4DOF | 0.25 | 87 |
| UFactory uArm | Ei | Arduino MEGA 2560 | 4DOF | 0.5 | 749 |
| RBX1 Remix | Jah | Raspberry Pi | 6DOF | 0.8 | 749,99 |

Nagu tabelist nähtub, on ainuke parameetrite poolest sobiv robotmanipulaator **Hoelldorfer**, kuna sellel on vajalik kandejõud (nagu ülalpool kirjeldatud, üks kilogramm). Nimetatud robotmanipulaatorit on üksikasjalikult uuritud ja tehtud kindlaks ajami rihmade erinevat tüüpi olemasolu. 3D-printimise kasutamise võimaluse abil on tarvis vähendada erinevate tootjate roboti komponentide kasutamist, kärpides kulusid ja sõltuvust kõrvalistest komponentidest. Samuti selgitati välja juhtimises kasutatud mikrokontrollerid, mis on tänaseks moraalselt aegunud ja millel puudub sisseehitatud funktsionaalsus tarkvara ja riistvara ühendamiseks.

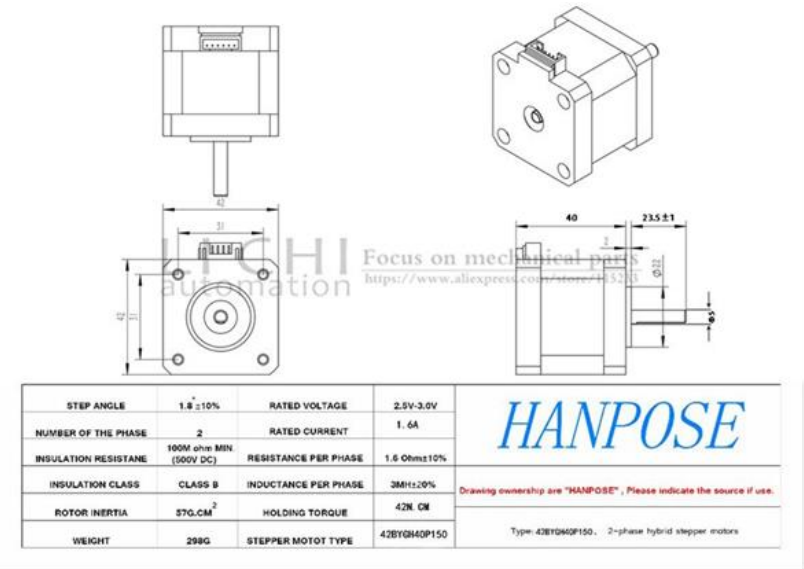
## 1.2 Optimaalse lahenduse kindlaksmääramine

**Olemasolevate lahenduste analüüsimisel määrati kindlaks järgmised materjalide valiku kriteeriumid ja komponendid robotmanipulaatori loomisel:**

- Vähendada kõrvaliste tootjate roboti komponentide kasutamist
- Kere ja ajami mehhanismide valmistamisel kasutada 3D-printimist
- Kulumiskindluse suurendamiseks ja sujuvaks liikumiseks kasutada pöörlevates sõlmedes laagreid
- Täpse asukoha kindlaks määramiseks kasutada samm-mootoreid
- Kasutada kaasaegset mikrokontrollerit, millel on riistvara ja tarkvara võimalus samm-mootorite juhtimiseks

## 1.3 Roboti komponentide kindlaksmääramine

**Mootorid** [vt Joonis 1] – robotmanipulaatori pöörlemistelgede põhimootoriteks valiti samm-mootorid NEMA 17 [3]:

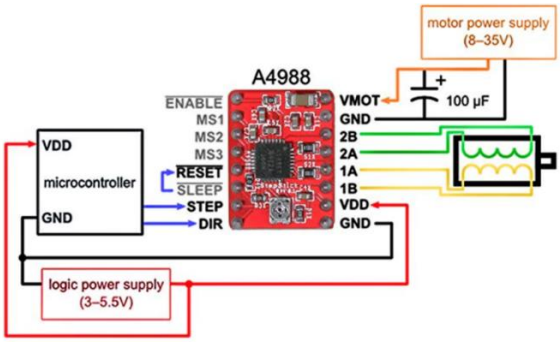



*Joonis 1 Samm-mootor NEMA 17*

Seda tüüpi mootoreid kasutatakse laialdaselt 3D-printerite loomisel. Reduktori kaudu mootori kasutamisel saadakse järgmine suhe liigendite vahel:

- Joint 1 (Liigend) – 61:1
- Joint 2 (Liigend) – 140:1
- Joint 3 (Liigend) – 72:1
- Joint 4 (Liigend) – 38:1
- Joint 5 (Liigend) – 60:1
- Joint 6 (Liigend) – 1:1

**Draiverid** [vt Joonis 2] – mootorite juhtimiseks otsustati kasutada HR4988-draiverit (modifikatsioon A4988) [4]:




*Joonis 2 Samm-mootori draiver*

**Mikrokontrolleri draiverid** [vt Joonis 3] – draiverite juhtimiseks otsustati kasutada STM32F407VGTx ARM [5], CPU ARM Cortex-M4, FPU MPU 168 MHz


*Joonis 3 STM32 samm-mootori mikrokontroller*

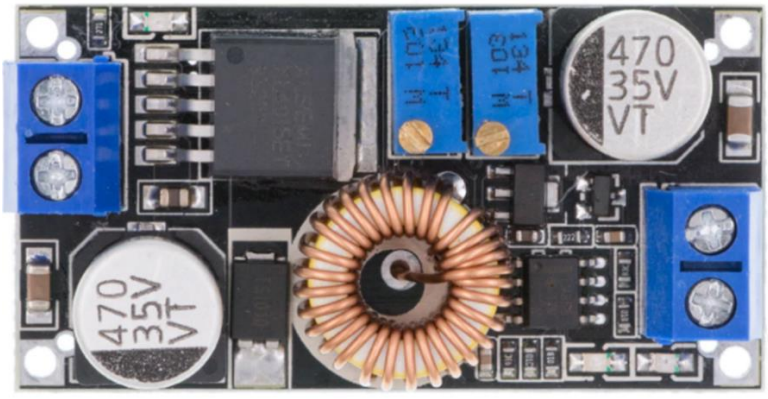
**Käigu mehaanilised tõkised** [vt Joonis 4] – robotmanipulaatori liikumise piiramiseks otsustati kasutada mehaanilisi tõkiseid [6]:


*Joonis 4 Samm-mootori käigu mehaaniline tõkis*

**Toiteallikas** [vt Joonis 5] – toiteks otsustati kasutada toiteplokki 12V 20A [7]:


*Joonis 5 Robotmanipulaatori elektroonikaseadmete toiteallikas*

**Pinget alandav muundur** [vt Joonis 6] – pinge alandamiseks 12V toiteallikast 5V kontrolleriteni otsustati kasutada alandavaid muundureid XL4015[8] :




*Joonis 6 Pinget alandav muundur samm-mootorite ja mikrokontrollerite jaoks*

**Raspberry Pi 3 Model B** [vt Joonis 7] – välise juhtseadmena otsustati kasutada *Raspberry Pi 3*-e [9]:

*Joonis 7 Mikrokontroller ROS juhtimiseks ja visualiseerimiseks*

**Laagrid, plastik** [vt Joonis 8] – pöörlevates sõlmedes otsustati kasutada allnimetatud laagreid.

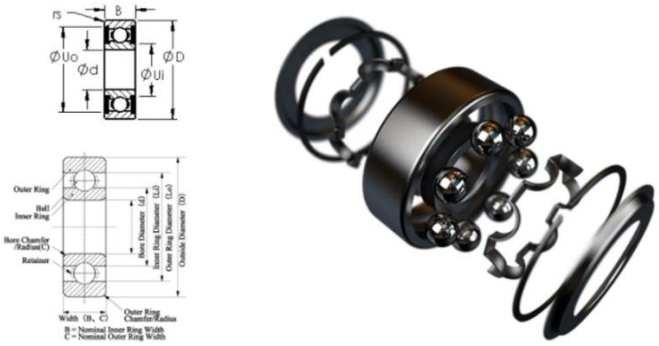
Aluse pöörlemiseks (**Liigend 1**) otsustati kasutada koonuslaagrit [10]:

*Joonis 8 Koonuslaager robotmanipulaatori Liigendi 1 jaoks*

Sõlmede (**Liigend 2, Liigend 3, Liigend 4, Liigend 5**) pöörlemiseks otsustati kasutada kuullaagrit [vt Joonis 9] [11]:


*Joonis 9 Robotmanipulaatori laager sõlmede (Liigend 2, Liigend 3, Liigend 4, Liigend 5) pöörlemiseks*

**Liigend 6** sõlme pöörlemiseks otsustati kasutada laagrit, mille põhielementideks on kerakujulised teraskuulid [12] [vt Joonis 10]:




*Joonis 10 Robotmanipulaatori laager Liigend 6 (haaratsi) jaoks*

**Planetaarülekandeks** otsustati kasutada kuullaagreid [vt Joonis 11] [13]:


*Joonis 11 Robotmanipulaatori laagrid planetaarülekandeks*

**3D-printeri hõõgniit (*filament*)** – põhiliste detailide trükkimiseks otsustati kasutada **PLA-d** (polülaktiid) plastikut. Hammasrataste printimiseks otsustati kasutada **ASA-t** (mis koosneb akrüülnitriilist, butadieenist ja stüreenist) ja **PETG-ed** (polüetüleen) plastikut.

# 2. ROBOTMANIPULAATORI 3D-MODELLEERIMINE

## 2.1 3D-modelleerimise platvormide analüüs

Robotmanipulaatori 3D-mudeli projekteerimiseks on hulk tarkvarariistu. Analüüsimisel selgus, et professionaalse 3D-toote modelleerimise jaoks sobivad järgmised kandidaadid [vt Tabel 2]:

*Tabel 2 Tarkvaratooted robotmanipulaatori 3D-projekteerimiseks*

| **Nimetus** | **Luba** |
|---|---|
| FreeCAD | Tasuta |
| Autodesk Fusion 360 | 1-aastane prooviversioon |
| AutoCad | 30-päevane prooviversioon |
| **SOLIDWORKS** | **30-päevane prooviversioon** |

Lõpptulemusena valiti tarkvaratooteks SolidWorks, sest mudeleid on võimalik eksportida ROS-i keskkonda URDF-i formaadis.

## 2.2 Projekteerimine ja mudeli arendus

Robotmanipulaatori kujundamise protsess on nii huvipakkuv kui ka küllalt rohket pingutust nõudev, kuna nõuab kannatlikku lähenemist tööprotsessile. Kuna töö autor ei ole sarnaste projektidega varem kokku puutunud, siis robotmanipulaatori projekteerimine ja mudeli arendus osutus tema jaoks tõsiseks väljakutseks.

### 2.2.1 Projekteerimise probleemide analüüs

**Esimeseks** lahendust vajavaks probleemiks projekteerimisel oli küsimus, millele toetuda – kas võtta eeskujuks juba avalikes allikates olev projekt või kui seda pole, siis millised mõõdud etaloniks valida?

**Teiseks** probleemiks oli küsimus, milliseid teiste tootjate robotikomponente kasutada, et säilitada optimaalne tugevuse ja suuruse suhe?

**Kolmandaks** küsimuseks oli töö autori sellise projekti arendamise kogemuse puudumine. Millised on arenduse tähtajad? Kuidas kõik kokku siduda? Kuidas esitada robotmanipulaatori lõppversiooni? Milliseid materjale ja valemeid üksikute mehaaniliste sõlmede arvutuste jaoks kasutada?

Kõik eelpool loetletud probleemid üheskoos nõudsid otsustavaid tegusid, et liikuda seatud eesmärgi poole.

### 2.2.2 Projekteerimise probleemide lahendamine

**Esimese küsimuse** lahendus osutus piisavalt lihtsaks. Ilmnes, et inimese käe näol on juba ammu olemas valmis näited — õlaliiges, küünarliiges, haare, liikmete vaheline õõs—, mis väga hästi sobivad selle projekti jaoks. Sellest tulenevalt saadi käe suuruse järgi ettekujutus tulevase robotmanipulaatori mõõtmetest. Robotmanipulaatori arendust alustati kõige olulisemast sõlmest — *Base-Link*'ist.

**Teise küsimuse** lahendus tuli sõidukite valdkonnast (sõidukite remont ja moderniseerimine on autori hobi). Teadmised selles valdkonnas aitasid autoril jõuda selgusele, millise tugevuse ja suurusega teiste tootjate robotite komponente valida.

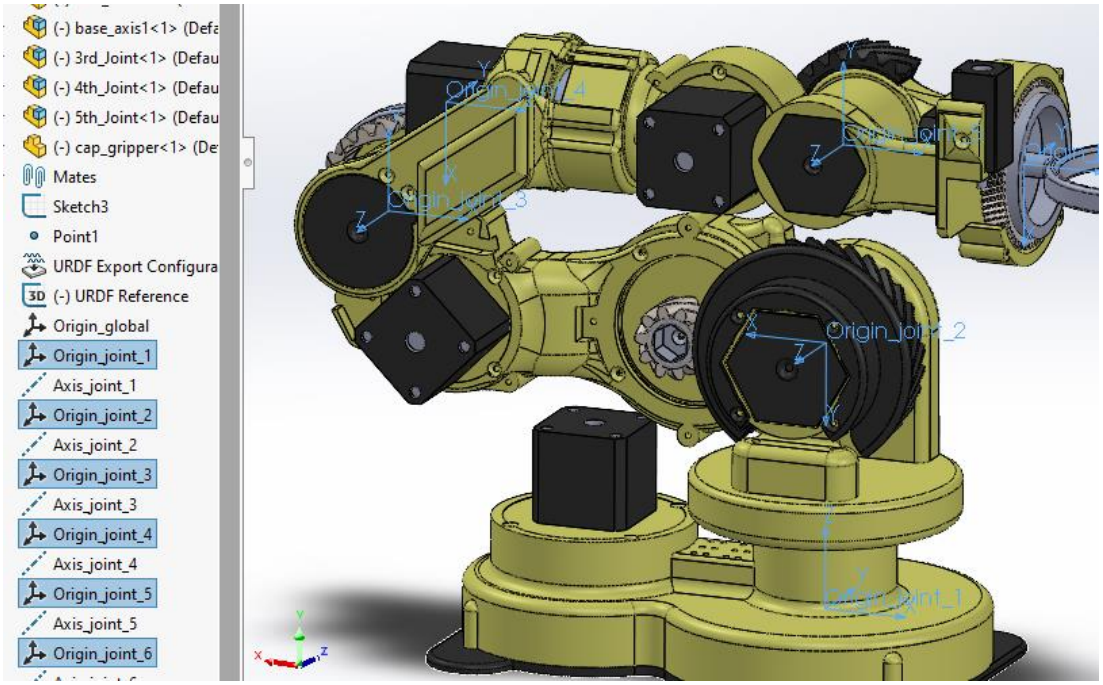
Autori tähelepanekud on näidanud, et mida lihtsam on konstruktsioon, seda töökindlam ja lihtsam hooldada, kuid nõrkade sõlmede kaasajastamine nõuab vähem muudatusi üldises konstruktsioonis. 3D-printimise tehnoloogia korral pole vaja kasutada näiteks rihmülekannet, vaid on võimalik luua hammasratas. Tugevuse ja suuruse õigeks suhteks on vaja kinni pidada ka tugevate jäikate roiete üleehitusest. Sellel etapil otsustati alustada kahte tüüpi planetaarülekannete väljatöötamist — esimene tüüp maksimaalse võimsussuhtega, teine tüüp minimaalsete mõõtmetega. Selle tulemusena loodi planetaarülekanded mõõtmetega 80 mm välisläbimõõdu (ülekandearv 1:36) ja 60 mm välisläbimõõduga (ülekandearv 1:28).

Projekteerimisel oli vaja samuti arvestada iga sõlme kaalukoormusega. Tehti kindlaks, et kogu roboti mass toetub Liigendile 1 ja nõuab pöörlemise ajal minimaalset lõtku ja maksimaalset kulumiskindlust. Selleks valiti koonusrull-laager, mis talub suuri koormusi. Ülejäänud teljed (v.a Liigend 4 ja Liigend 6) pöörlevad vertikaalselt ja see võimaldab kasutada väiksemaid laagreid, näiteks jalgratta omasid, mis taluvad kuni 200-kilost koormust. Ülejäänud telgede jaoks konstrueeriti tavapärased kuullaagrid võimalikult väikeste mõõtmetega, kuna võimsuskoormus neile ei ole suur.

**Kolmanda küsimuse** puhul selgus, et puudub valem, mis võimaldaks kiiresti kavandada robotmanipulaatori lõppjärku ning kompenseerida autori kogenematust sellise projekti arendamisel. Samas sai selgeks, et pärast esimese ja teise etapi läbimist, kus töötati välja planetaarülekanne, leiti lahendus edasiliikumiseks ja tugikonstruktsiooni järgmise lüli ehitamiseks.

## 2.3 3D-mudeli eksport ROS-i keskkonda

ROS-i süsteem toetab 3D-mudelite importimist **URDF-i** (Unified Robot Description Format) formaadis. Tarkvaratoote robotmanipulaatori 3D-projekteerimiseks võimaldab raalprojekteerimise programm Solidworks eksportida URDF-i mudeleid, kasutades laiendit "*SolidWorks to URDF Exporter*" [14]. Selleks on vaja iga paaritussõlme jaoks seadistada X, Y, Z koordinaatide süsteem, et ROS-i süsteem saaks ruumis liikumise arvutamiseks määrata sõlmede seosed, nende suurused ja nendevahelised kaugused [vt Joonis 12].



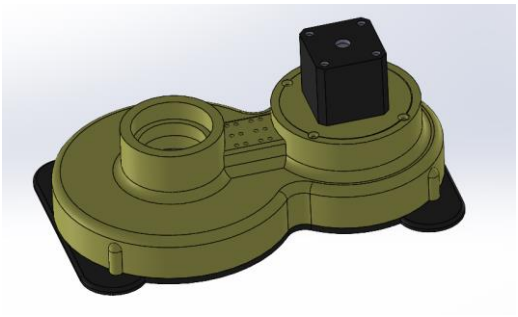

*Joonis 12 URDF formaadi robotmanipulaatori koordinaatsüsteem (ROS-i integratsioonimudel)*

### 2.3.1 3D-mudeli sõlmede kokkupanek

Robotmanipulaatori 3D-mudeli kokkupanemisel on väga oluline, et liitmiseks oleksid valmis montaažisõlmed. Iga sõlm ja tema liide on tulevikus mudeli ROS-i süsteemi integreerimisel võtmetähtsusega sõlmede suuruste (*Link*) ja liidete telgede vahekauguste (*Joint*) arvutamisel.

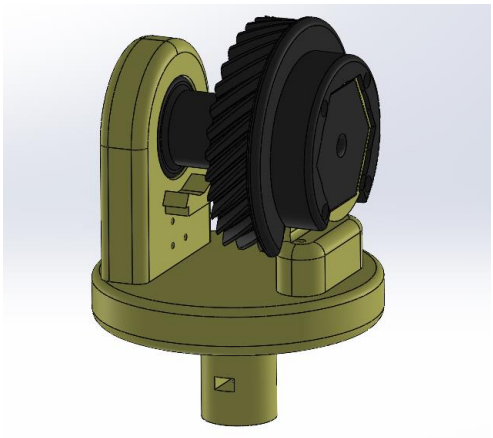
Valmistati 7 *Link*-sõlme.

Link 1 [vt Joonis 13]:



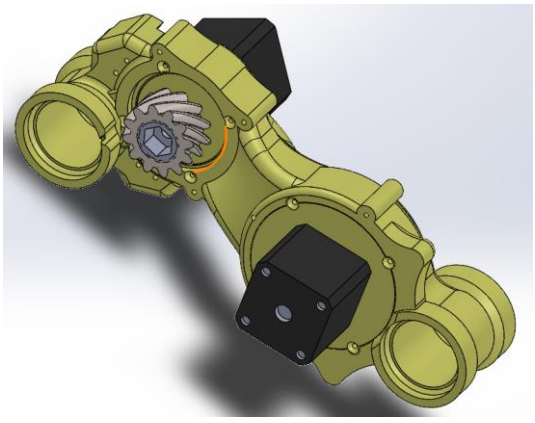
*Joonis 13 Robotmanipulaatori Link 1 URDF-i projekti jaoks*

*Link* 2 [vt Joonis 14]:



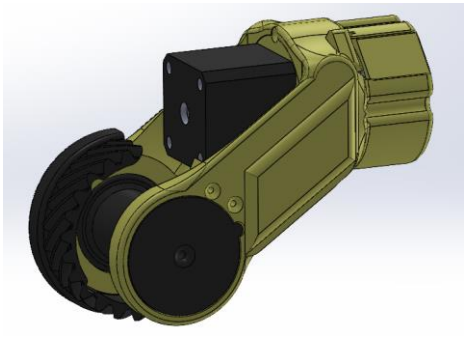
*Joonis 14 Robotmanipulaatori Link 2 URDF-i projekti jaoks*

*Link 3* [vt Joonis 15]:



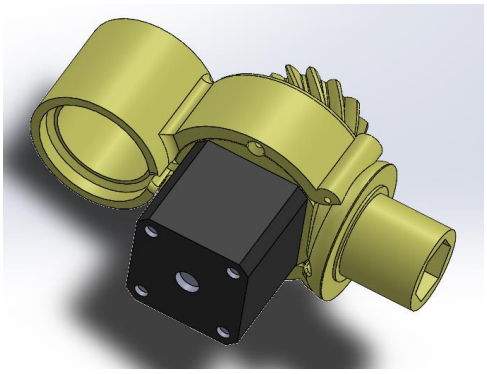
*Joonis 15 Robotmanipulaatori Link 3 URDF-i projekti jaoks*

*Link* 4 [vt Joonis 16]:



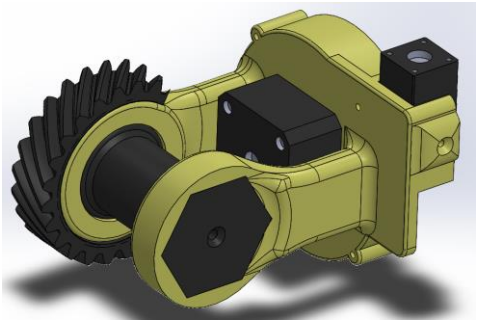
*Joonis 16 Robotmanipulaatori Link 4 URDF-i projekti jaoks*

*Link* 5 [vt Joonis 17]:



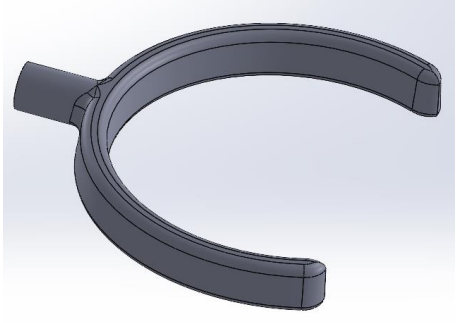
*Joonis 17 Robotmanipulaatori Link 5 URDF-i projekti jaoks*

*Link* 6 [vt Joonis 18]:



*Joonis 18 Robotmanipulaatori Link 6 URDF-i projekti jaoks*

*Link* 7 [vt Joonis 19]:

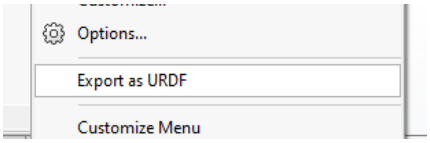


*Joonis 19 Robotmanipulaatori Link 7 (haarats) URDF-i projekti jaoks*

### 2.3.2 Mudeli eksport URDF-projekti

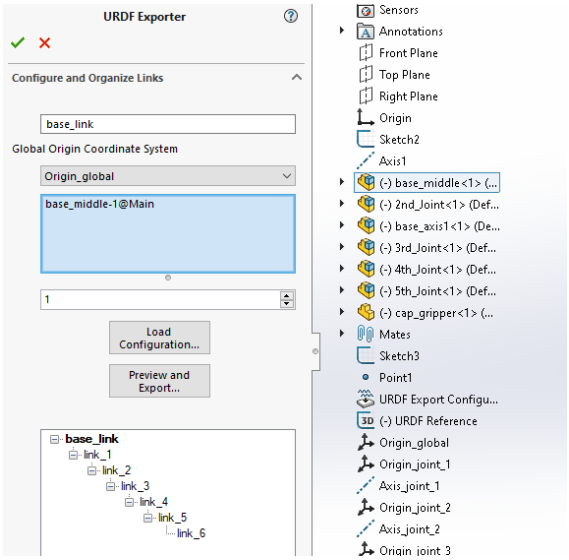
Pärast kõigi *Link*-sõlmede liitmist piki nende pöörlemistelgi (kasutades laiendit „SolidWorks to URDF Exporter") on mudel valmis URDF-i projekti eksportimiseks.

Selleks kuvatakse laiendi dialoogiaken [vt Joonis 20] – Tools -> Export as URDF:



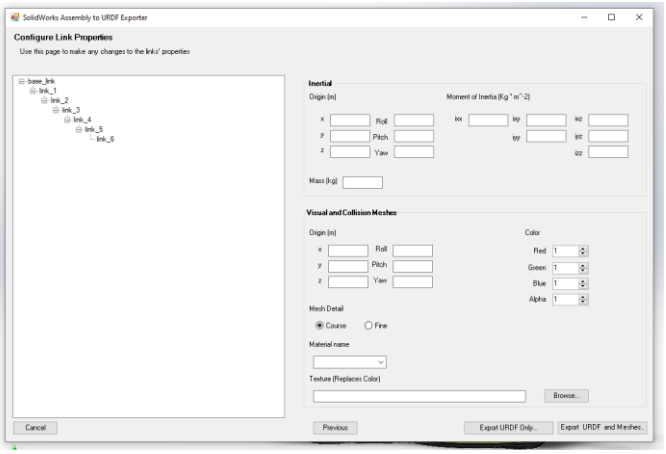

*Joonis 20 URDF-i eksportija kontekstimenüü*

Märgitakse *Link*-sõlmed ja nende nimed [vt Joonis 21]:




*Joonis 21 URDF-i eksportija sätted robotmanipulaatori jaoks*

Vajutatakse nuppu [vt Joonis 22] – "Preview and Export..." -> "Export URDF and Meshes"




*Joonis 22 URDF-i eksportija viimane samm eksportimiseks robotmanipulaatori mudeli eksportimiseks*

Sellel etapil on robotmanipulaatori mudel valmis eksportimiseks URDF-i projekti nimega **"robot_arm_mumm"** selle edasiseks integreerimiseks ROS-i keskkonda.

# 3. ROS-I SÜSTEEMI KASUTUSELEVÕTMINE JA SEADISTAMINE

ROS (*Robot Operating System*) — robotite operatsioonisüsteem — on funktsionaalne ökosüsteem robotite programmeerimiseks. ROS töötati algselt välja 2007. aastal.

ROS pakub standardseid operatsioonisüsteemi teenuseid nagu riistvara abstraktsioon, seadmete madalatasemeline kontroll, tihti kasutatavate funktsioonide rakendamine, protsessidevaheliste sõnumite edastamine ja paketihaldus. ROS põhineb graafide arhitektuuril, kus andmetöötlus toimub sõlmedes, mis saavad omavahel sõnumeid vastu võtta ja edastada. Raamatukogu on keskendunud Unixi-laadsetele süsteemidele (mille hulgas on ka *Ubuntu Linux*).

Seade, millele paigaldatakse baasoperatsiooni- ja ROS-i süsteem, on mikrokontroller *Raspberry Pi 3B*.

## 3.1 ROS-i versiooni valik

Kuna väline juhtseade *Raspberry Pi 3B* toetab operatsioonisüsteemi *Ubuntu Server 20.04* (koodnimega *Focal*) paigaldamist ja ROS toetab seda operatsioonisüsteemi koos viimase versiooniga *Noeticu*, siis valiti aluseks ROS-i versioon.

### 3.1.1 Baasoperatsioonisüsteemide analüüs ROS-i jaoks

Kuna ROS on metaoperatsioonisüsteem (raamistik), vajame selle käitamiseks Linuxi perekonna baasoperatsioonisüsteemi. Operatsioonisüsteemi valikul tehti praktiline valik järgmiste ROS-i toetatud operatsioonisüsteemide vahel [15]:

- ***Raspbian OS x64*** [16], koodnimi: ***Buster –*** *ROS version:* ***Kinetic***
- ***Ubuntu Mate 18.04*** [17], koodnimi: ***Bionic*** – *ROS version:* ***Melodic***
- ***Ubuntu Server 20.04.3 LTS*** [18], koodnimi: ***Focal*** – *ROS version:* ***Noetic***

Praktilise analüüsi põhjal tehti kindlaks, et ROS-i versioonid [19] - *Kinetic* ja *Melodic* on vananenud ja nende funktsionaalsus on piiratud robotmanipulaatori URDF-i mudelite integreerimiseks ROS-i keskkonda. Suureks puuduseks oli ka põhiliste operatsioonisüsteemide *Raspbian OS x64* ja *Ubuntu Mate 18.04* jõudlusvõime kokkupanekul *Kinetic*- ja *Melodic*-versioonidega ning *ROS Noetic*-versiooni tugi on võimalik ainult *Ubuntu Server 20.04-s* (*Focal*).

### 3.1.2 Baasoperatsioonisüsteemi kindlaksmääramine

Pärast ebaõnnestunud katseid integreerida robotmanipulaatori mudel baasoperatsioonisüsteemide (OS) *Raspbian OS x64* ja *Ubuntu Mate 18.04* baasil ROS-i keskkonda, leiti eksperimentaalselt optimaalne lahendus OS-i valimiseks – *Ubuntu Server 20.04*.

OS-i installimiseks kasutati juhiseid ametlikult veebilehelt [20].

Kuna *Ubuntu Server 20.04* paigaldatakse vaikimisi ilma graafilise liideseta, pole *Raspbian OS x64* ja *Ubuntu Mate 18.04* jaoks saadaval ühtegi sõltuvat graafilist paketti nagu redaktorid, mängud ja muu sisseehitatud tarkvara. Sellest tuleneb ka aluseks oleva OS-i suurenenud jõudlus ja süsteemi stabiilsus. Kuid ROS nõuab töötamiseks graafilist kesta.

### 3.1.3 Baasoperatsioonisüsteemi seadistamine

Graafilise kesta paigaldamiseks testiti 3 kergete pakettide varianti:

- *xubuntu-desktop*
- *lubuntu-desktop*
- *mate-ubuntu-desktop*

Eksperimentaalse meetodiga tehti kindlaks, et kuigi kahel esimesel variandil on kerged kestad, mis toetavad kuvahaldureid *gdm3*, *lightdm* ja *sddm*, on need praktikas ebastabiilsed ning tarbivad rohkem mälu ja protsessori aega kui *mate-ubuntu-desktop*.

Paketi *mate-ubuntu-desktop* repositooriumi (elektrooniline varamu, kus säilitatakse sinna üles laaditud materjale) paigaldamiseks kasutati käsku:

```
sudo apt install mate-ubuntu-desktop
```

Riistvara korrektseks tööks ja videodraiverite toeks on vaja lisada faili ka seadistuste konfiguratsioon /boot/config.txt [21]:

```
dtoverlay=vc4-kms-v3d
gpu_mem=128
arm_control=0x200
hdmi_edid_file=1
hdmi_mode=57
```

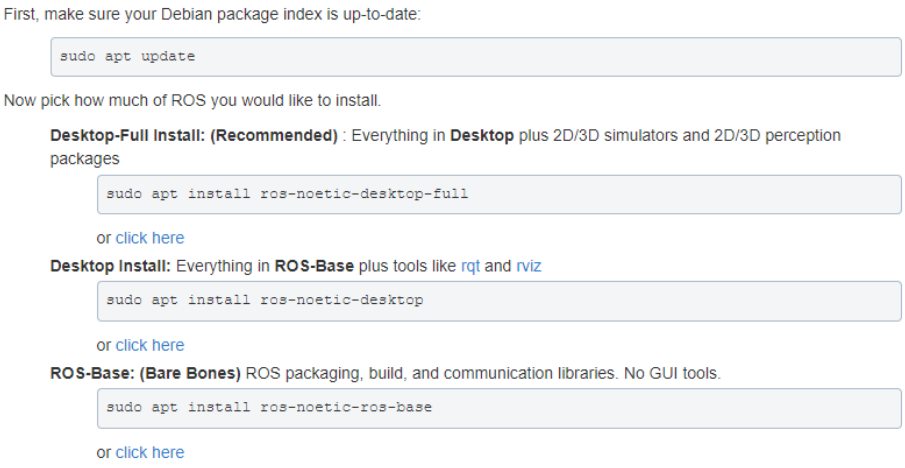
Nüüd on süsteem ***ROS Noetic Ninjemys*** paigaldamiseks valmis.

## 3.2 ROS-i paigaldamine ja seadistamine

ROS-i paigaldamiseks kasutati juhiseid ametlikult veebilehelt [22].

ROS-i süsteemi paigaldamisel vastavalt juhistele on ainsaks paranduseks see, et enne täissüsteemi paigaldamist (*Desktop-Full Install*) tuleb esmalt paigaldada baaskonfiguratsioon (*ROS-Base*) ja alles seejärel täiskonfiguratsioon.

Määratud paigaldusjärjestus on soovitatav teha vastavalt juhistele [vt Joonis 23]:




*Joonis 23 ROS-i paigaldamise juhis Ubuntu 20.04-le*

Kuid praktikas katkeb täispaketi paigaldusprotsess lahendamata seoste tõttu. Ja **alles siis**, kui paigaldatakse esmalt baaspakett ***ROS-Base***, on kogu paketi edasine paigaldamine edukas.

### 3.2.1 Nõutavate lisalaiendite paigaldamine

3D-mudeli rakendamiseks ROS-i süsteemi on vaja paigaldada ka lisalaiendid – *MoveIt* [23], *Rviz* [24] ja *ROSSERIAL* [25]

*MoveIt*-laiendi paigaldamise käsk:

```
sudo apt install ros-noetic-moveit ros-noetic-moveit-planners-ompl ros-noetic-moveit-resources-prbt-moveit-config ros-noetic-moveit-ros ros-noetic-moveit-setup-assistant
```

*Rviz*-laiendite paigaldamise käsk:

```
sudo apt install ros-noetic-rviz*
```

*ROSSERIAL*-laiendite paigaldamise käsk:

```
sudo apt install ros-noetic-rosserial*
```

Sellel etapil on baasoperatsioonisüsteem ja ROS-i süsteem täielikult paigaldatud ning valmis robotmanipulaatori olemasoleva 3D-mudeli rakendamiseks ning *ROS Core Master*-ist positsioneerimisandmete edastamiseks mikrokontrollerisse *STM32 ROSSERIAL*-i protokolli järgi.

## 3.3 3D-mudeli import ROS-i keskkonda

Robotmanipulaatori 3D-mudeli importimiseks käivitatakse konfiguratsiooni assistent.

Selleks, et initsialiseerida ROS-i töökeskkonda, luuakse kaust „ros_ws", kuhu koondatakse projekt kõigi programmi puutuvate lisadega.

Tööruumi ettevalmistavad käsud:

```
source /opt/ros/noetic/setup.bash
```

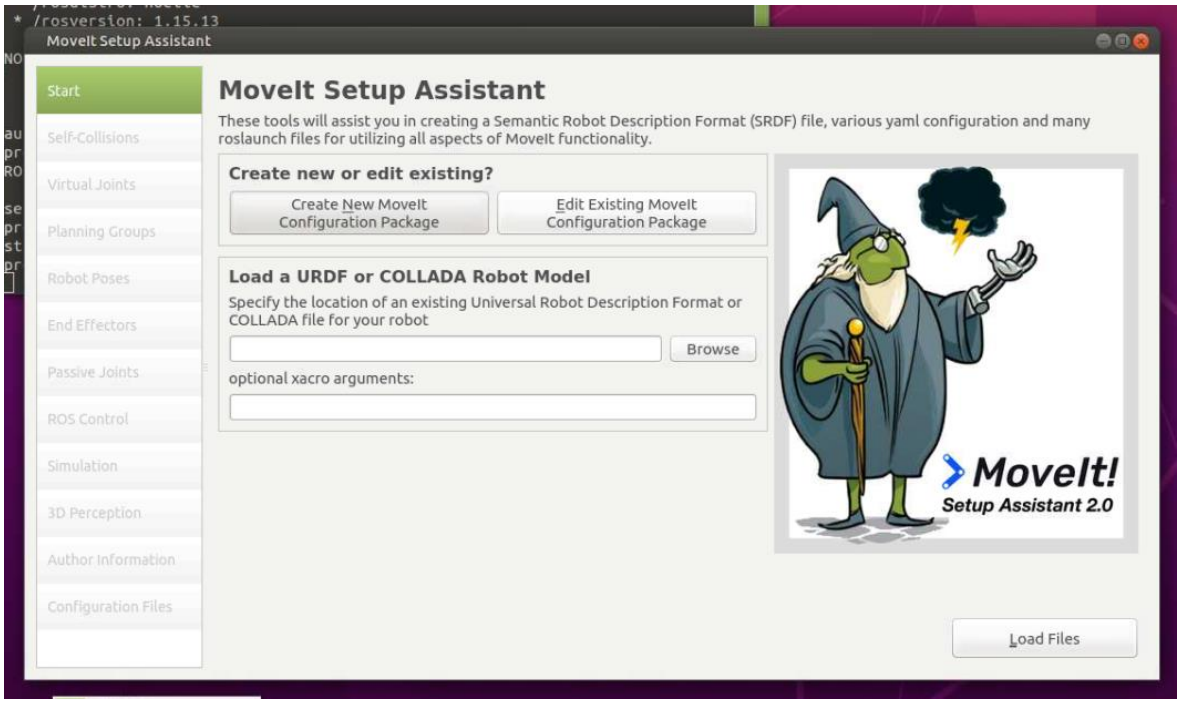
```
mkdir -p ~/ros_ws && cd ~/ros_ws
```

```
catkin_init_workspace && mkdir -p src && catkin_make
```

ROS-i tööruumi ettevalmistamise järel kopeeritakse robotmanipulaatori URDF-i projektifailid *SolidWorks*-is väljatöötatud kausta „src".

Käsk *ROS MoveIt* impordi meistri käivitamiseks [vt Joonis 24]:

```
roslaunch moveit_setup_assistant setup_assistant.launch
```




*Joonis 24 MoveIt'i paigaldamise Setup Assistant URDF-projekti seadistamiseks*

URDF-i konfiguratsiooni paigaldamiseks *ROS MoveIt*-i abil kasutati juhiseid ametlikult veebilehelt [26].

URDF-i projekti seadistamise järel saadakse robotmanipulaatori konfiguratsiooni lisapakett „**robot_arm_mumm_config"**, mis on valmis *ROS Rviz*-i visualiseerimise keskkonnas käivitamiseks.

### 3.3.1 ROS Rviz-i visualiseerimise keskkonna käivitamine

Saadud konfiguratsioonipakett „robot_arm_mumm_config“ sõltub varem loodud URDF-i projektist „robot_arm_mumm”, mis salvestab robotmanipulaatori mudelid, mistõttu on väga oluline luua eraldi tööruum iseseisvate projektide või sama mudeliga projektide jaoks.

„Robot_arm_mumm_config”-i konfiguratsioonikaustas „launch” luuakse käivitusfailid “demo.launch”. Nendest tehakse koopia edasiste muudatuste jaoks ja sisestatakse järgmine kopeerimiskäsk:

```
cd ~/ros_ws/src/robot_arm_mumm_config/launch && cp demo.launch arm.launch
```

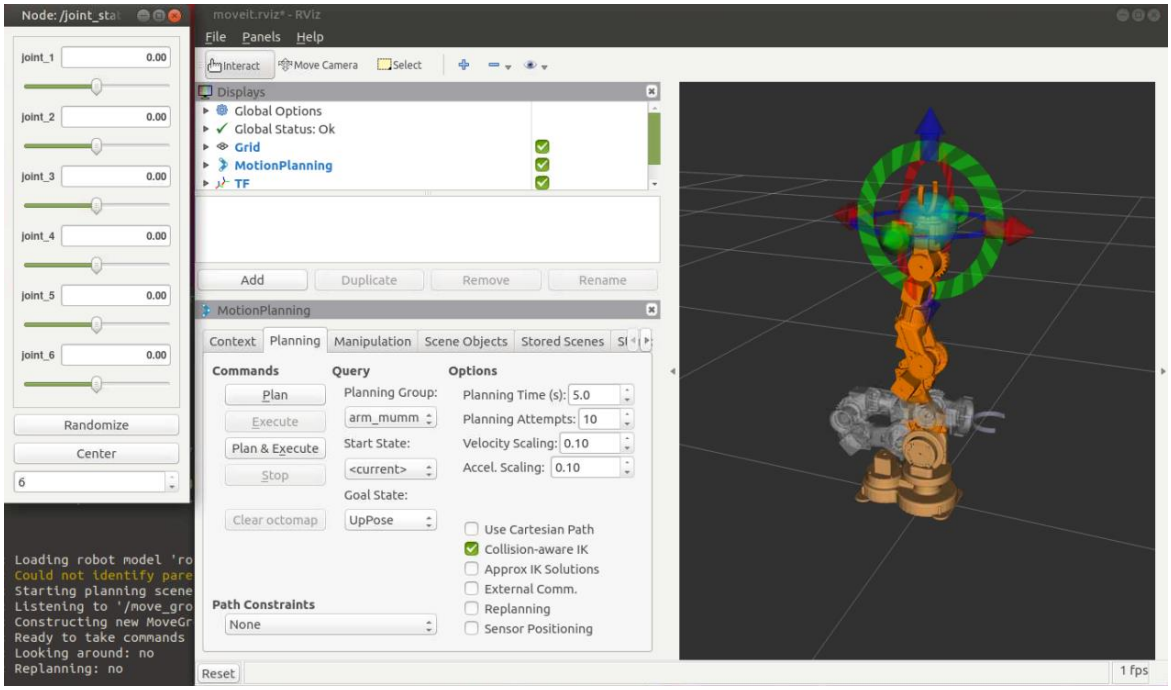
Tehakse kopeeritud failis järgmised muudatused, et aktiveerida üksikute *Link*-sõlmede visuaalne juhtimine:

```
sed -i 's/"use_gui" default="false"/"use_gui" default="true"/g' arm.launch
```

Samuti tuleb edaspidi luua robotmanipulaatori kontrollerite juhtimispaketid, mis sõltuvad loodud URDF-i projekti konfiguratsioonist.

Visuaalkeskkonna *Rviz* käivitamiseks kasutatakse käsku:

```
roslaunch robot_arm_mumm_config arm.launch
```




*Joonis 25 Robotmanipulaatori visuaalne juhtimine ROS Rviz-i poolt*

Sellel etapil on robotmanipulaatori 3D-mudel juhtimiseks valmis [vt Joonis 25].

## 3.4 ROS-i sõlmede interaktsioon

Peale Linuxi baasoperatsioonisüsteemi seadistamist, ROS-i paigaldamist, sõltuvate pakettide seadistamist ja robotmanipulaatori 3D-mudeli lõimimist ROS-i keskkonda jõutakse kõige tähtsama, ROS-i süsteemi sõlmede interaktsiooni arhitektuurini [27].

Üheks peamistest ROS-i eesmärkidest on hõlbustada ROS-i sõlmede vahelist suhtlust. Need sõlmed kujutavad endast käivitatavat koodi. Koodi saab paigutada täielikult ühte arvutisse või jagada arvutite vahel või arvutite ja robotite vahel. Sellise arhitektuuri eeliseks on see, et iga sõlm saab juhtida ühte süsteemi lõiku.

Näiteks, üks sõlm saab pildistada kaamerast ja saata pildi töötlemiseks teisele ROS-i sõlmele. Pärast pilditöötlust võib teine sõlm saata teate kolmandale robotmanipulaatori juhtsõlmele, lähtudes pilditöötlusest.

Põhiline ROS-i sõlmede interaktsiooni põhimõte on sõnumite saatmine ja vastuvõtmine. Sõnumid on rühmitatud eraldi teemadesse, mida nimetatakse *ROS Topic*'iks. ROS-i sõlm saab postitada sõnumeid üksikule *ROS topic*'ile või tellida *ROS Topic*'u sõnumite vastuvõttu.

### 3.4.1 ROS-sõlmed

ROS-i sõlmed (*ROS node*) kujutavad endast teatavate protsesside esitusi [28], mis sooritavad konkreetseid ülesandeid või arvutusi. Iga sõlme jaoks eraldatakse baasoperatsioonisüsteemis eraldi protsess ja registreerimine *ROS Master*-i põhisõlmes, et suhelda teiste süsteemi sõlmedega. Iga ROS-i sõlm on sõltumatu ja suhtleb teiste sõlmedega sisemise *ROS API* abil, registreerides sündmused *ROS Master*-i teemas.

ROS-i sõlmede arhitektuuri tugevus seisneb selles, et suurema ülesande saab lahutada eraldi alamülesanneteks, mis täidavad iseseisvalt konkreetset rolli üldises süsteemis.

Näiteks, robotmanipulaatori juhtimise saab jagada eraldi sõltumatuteks alamülesanneteks, millest igaüks jälgib konkreetset sündmust/arvutust. Ülesanded täidetakse asünkroonselt sisendteabe alusel. Üks ROS-i sõlm võtab kaamerast vastu pilte roboti liikumisest sõltuva sagedusega, edastades saadud pildid teisele sõlmele, mis omakorda tunneb ära pildil olevad objektid ja edastab keskkonna koordinaadid kolmandale sõlmele. Kolmas sõlm arvutab roboti trajektoori ja saadab teate neljandale sõlmele, mis arvutab välja mootoritele genereeritud impulsside arvu.

Sellises süsteemis ei tea iga ROS-i sõlm teistes sõlmedes toimuvatest protsessidest ning tegeleb teatud tüüpi sõnumite vastuvõtmise, töötlemise ja saatmisega teistele sõlmedele. Samas ei mõjuta ühe sõlme koodi muutmine teise sõlme tööd.

### 3.4.2 ROS-i teemad

Kui ROS-i sõlm suhtleb teiste sõlmedega, on vaja ühist andmevahetuse "nimelist siini". Need "siinid" nimetatakse ROS-i teemaks (*ROS Topic*) [29]. ROS-i teemal on oma sõnumitüüp ja ROS-i teemas avaldatud teabel võib olla ainult üks ROS-i teate tüüp. Kui tüübid ei ühti, ei saa tellija ROS-i teemaga ühendust luua.

Kuna ROS-i sõlm saab teavet saata või vastu võtta, jagunevad sellised sõlmed kahte tüüpi:

- *ROS node Publisher* – avaldab teavet
- *ROS node Subscriber* – saab teavet

Üks ROS-i sõlm võib teavet nii vastu võtta kui ka avaldada. Selline lähenemine näitab selgelt, et ROS põhineb graafide arhitektuuril, kus andmetöötlus toimub sõlmedes, mis saavad omavahel sõnumeid vastu võtta ja edastada.

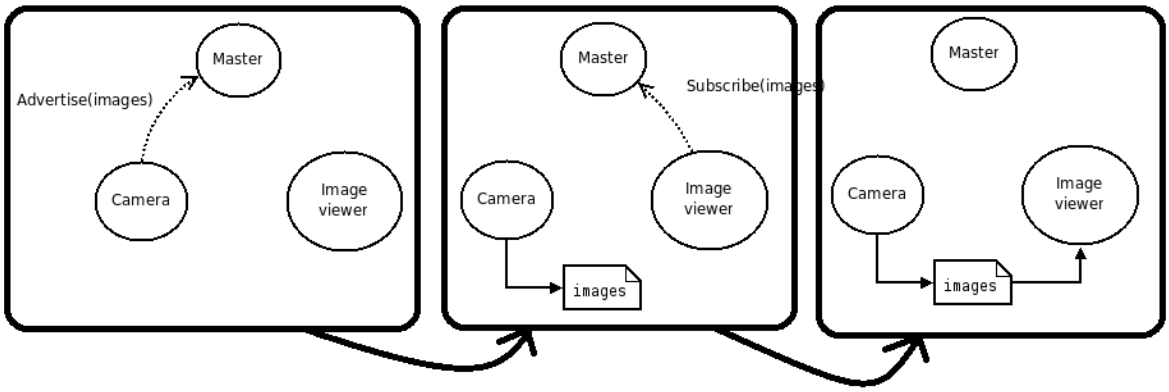
### 3.4.3 ROS-i teated

ROS-i teated on määratletud sõnumitüübi järgi [30]. Iga teate tüüp põhineb sisseehitatud (baas-) andmetüüpidel. Sisseehitatud andmetüüp põhineb 14 primitiivsel andmetüübil. See on väga sarnane *C++*-struktuuridega, kus kõrgetasemelised andmetüübid põhinevad primitiivsetel andmetüüpidel.

Näiteks, baastüübil *std_msgs:: String* on andmeväli *data* ja selle tüübiks on *std::string C++*-st.

### *3.4.4 ROS Master*

*ROS Master* on sõltumatu ROS-i sõlm, mis haldab kõiki ROS-i süsteemi teabetellimusi ja väljaandeid [31]. *ROS Master* pakub ROS-i süsteemi sõlmedele nime andmise ja registreerimisteenuseid. Sõlmede vahelise ühenduse loob *ROS Master*.




*Joonis 26 ROS Master Publisher ja Subscriber näidis*

Ülaltoodud näitest [vt Joonis 26] nähtub tellimuse registreerimise ja sõnumite avaldamise järjekord kahe sõlme Kaamera (*Camera*) ja Pildivaatur (*Image_viewer*) vahel:

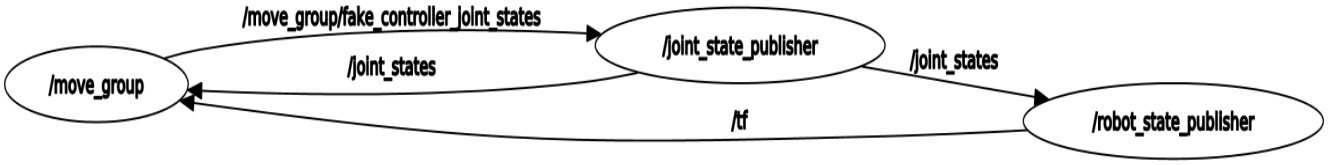
- Kaamera (*Camera*) teatab *ROS Master*'ile, et ta postitab pilte ROS-i teemasse pildid (*images*), kuid kuna tellijaid ei ole, siis andmeid ROS-i teemas ei avaldata
- Pildivaatur (*Image_viewer*) saadab tellimistaotluse ROS-i teemale pildid (*images*)
- Nüüd, kui ROS-i teemal piltidel (*images*) on nii avaldaja (*Publisher*) kui ka tellija (*Subscriber*), teavitab *ROS Master* teineteise olemasolust kaamerale ja pildivaaturile, et nad saaksid hakata teineteisele sõnumeid saatma

## 3.5 *ROS Rviz*-i teemade struktuur URDF-i projekti jaoks

URDF-i projekti konfiguratsiooni saamise järel roboti visuaalseks juhtimiseks on vaja koostada vajalike ROS-i teemade loend ja nendevaheliste sõnumite interaktsiooni skeem.

### 3.5.1 Rviz-i teemade struktuuri analüüs

ROS-i süsteemil on sisseehitatud sõlmede visualiseerimise tööriistad. Üheks tööriistaks ROS-i süsteemis käesolevate teemade vaatamiseks on utiliit *rqt_graph*.




*Joonis 27 Robotmanipulaatori ROS-i teemade skeem Rviz-is*

Käesolevast graafist [vt Joonis 27] nähtub, et avaldaja *„/robot_state_publisher"* avaldab sõnumeid teemas *"/tf"*, millele on tellitud *„/move_group"*.

Kõigi teemade sõnumite uurimisel tehti kindlaks, et tegelikke positsioneerimisandmeid avaldab teemas *"/tf"* ainult *"/robot_state_publisher"* ja teemas *"/move_group/fake_controller_joint_states"* avaldatakse simulatsiooniandmed funktsionaalsuse demonstratsiooniks.

ROS-i süsteemi teemade loendi kuvamiseks kasutatakse käsku:

```
rostopic list
```

Teemas *„/tf"* kasutatava sõnumi tüübi määramiseks kasutatakse käsku:

```
rostopic info /tf
```

Seejärel saadakse  informatsiooni silumiseks (*debugging*):

```
Type: tf2_msgs/TFMessage
Publishers:
 * /robot_state_publisher (http://ros:45425/)
Subscribers:
 * /move_group (http://ros:40135/)
 * /rviz_ros_2001_2403723421050923995 (http://ros:35865/)
```

### 3.5.2 Robotmanipulaatori liigeste sõlmede liikumise andmete moodustamine

Füüsiliste sõlmede liikumiseks on vaja moodustada positsioneerimisandmete vastuvõtt Rviz-i keskkonna robotmanipulaatori virtuaalsest juhtimisruumist ning edastada töödeldatud andmed riistvarajuhtimise mikrokontrolleri STM32 sõlme. Algselt saadud andmed ei ole riistvaralisele juhtimisele edastamiseks valmis. Esiteks on tarvis töödelda „tooreid" andmeid ja arvutada füüsiliste sõlmede jaoks ettenähtud piirangud.

Nagu oli eelnevalt määratletud, on positsioneerimisandmeid avaldav ROS-i teema *"/tf"* sõnumitüüp *"tf2_msgs/TFMessage"* [32]. Nimetatud tüübil on järgmine struktuur: [vt Lisa 4]

```
transforms:
  -
    header:
      seq: 0
      stamp:
        secs: 1637612814
        nsecs: 656158208
      frame_id: "link_1"
    child_frame_id: "link_2"
    transform:
      translation:
        x: 0.0
        y: 0.0
        z: 0.025
      rotation:
        x: 0.0
        y: 0.0
        z: 0.0
        w: 1.0
  -
  ...
```

Antud tüüp koosneb andmemassiivist *"transforms"*, mis hoiab endas kõigi sõlmede liikumise koordinaate koos ajatempliga. Robotmanipulaatori sõlmede pöörlemise määramiseks on vaja järgmisi välju:

- teisendus. pöörlemine (*transform. rotation*) -> x, y, z, w
- päis (*header*) -> ajatempel (*stamp*), frame_id

Kuna algandmed *"transform.rotation -> x, y, z, w"* on kvaternioon, luuakse funktsioon, mis teisendab kvaterniooni radiaanideks. Radiaanide väärtust on nüüd mugav kasutada liigendsõlmede pöörlemise arvutamiseks.

```
def joint_trackval(joint):
    orientation_list = [joint.x, joint.y, joint.z, joint.w]
    return euler_from_quaternion(orientation_list)
```

Funktsioon aktsepteerib objekti *Joint*-tüüpi väljadega x, y, z, w.

## 3.6 Ühendus STM32-mikrokontrolleriga

Pärast täielikku seadistamist, ROS-i juhtpakettide paigaldamist ja välise *Raspberry Pi 3B*-kontrolleri visuaalse juhtimise käivitamist jääb üle konfigureerida sild, et ühendada STM32-mikrokontrolleri samm-mootorite välisjuhtimine ja riistvaraline juhtimine.

Selle jaoks on ROS-i süsteemis välja töötatud spetsiaalne protokoll nimega *ROSSERIAL* [33]. *ROSSERIAL* on protokoll standardsete seeriaviisliste ROS-i teadete pakendamiseks ja mitme ROS-i teema paljundamiseks märgiseadme (nt jadapordi või võrgupesa) kaudu.

Antud protokolli eesmärk on otsene punkt-punkt side jadapordi kaudu erinevate mikrokontrollerite integreerimiseks ROS-i süsteemi. Peamised teegid selle protokolli kasutamiseks on kirjutatud *Pythonis*-e ja *C++* keeles.

### 3.6.1 Teekide ettevalmistamine STM32 jaoks

STM32-mikrokontrolleritele ROS-i teekide loomiseks laetakse alla skriptvara ametlikust ROS-i hoidlast [34].

Skripti käivitamiseks peab olema eelinstallitud *ROS + Python* süsteemid, antud juhul väline juhtkontroller *Raspberry Pi 3B*, millele on installitud *Ubuntu 20.04* baasoperatsioonisüsteem.

Teekide loomiseks sisestatakse järgmised käsud:

```
cd ~/ros_ws/ && mkdir -p Inc && mkdir -p Src
```

```
rosrun rosserial_stm32 make_libraries.py .
```

Pärast seda genereeritakse kaustas “Inc” kõik STM32-mikrokontrollerite jaoks vajalikud ROS-i teegi failid.

### 3.6.2 Teekide kasutuselevõtmine STM32-mikrokontrolleritesse

ROS-i teekide käitamiseks STM32-mikrokontrolleritel on eeltingimuseks *C++*-projektide tugi. Kuna STM32 jaoks mõeldud *STM32CubeIDE*-platvormide ametlik tarnija on sellise võimaluse pakkunud (kuid peamised arendus- ja süsteemiteegid on kirjutatud *C* keeles), on *C++*- ja ROS-i projektide integreerimine võimalik väikeste koodimuudatustega. Selleks kasutati *STM32CubeIDE* arendaja ametliku veebilehe juhiseid. [35]

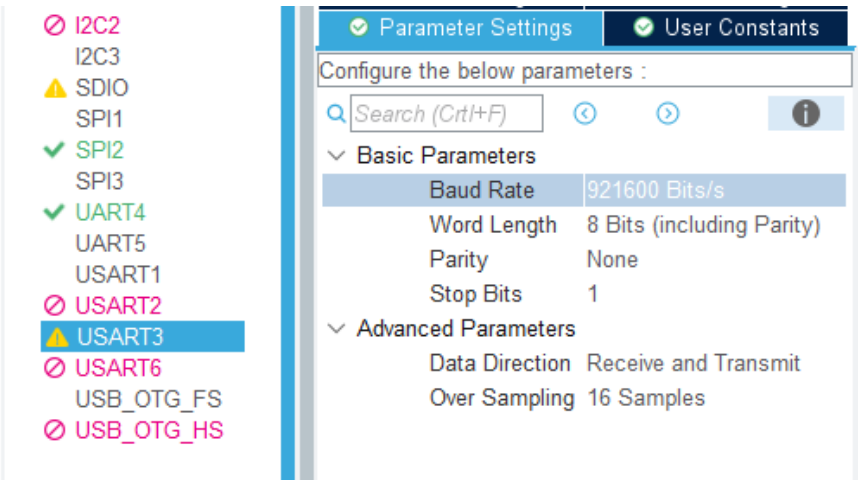
ROS-i teegi funktsionaalsus STM32-mikrokontrollerite jaoks on piiratud ja see ei ole ROS-i hostmasinas kasutatavate teekide koopia. Näiteks, klassi *"ros :: NodeHandle"* objektil puudub sisseehitatud *spin()*-funktsioon (mis on hostversioonil saadaval) ja puuduvad ka paljud geomeetria arvutamise funktsioonid, mis nõuab lisafunktsioonide kirjutamist sissetulevate ja väljaminevate sõnumite töötlemiseks *ROSSERIAL*'i protokolli abil.

### 3.6.3 STM32-mikrokontrolleritega ühendamise seadistamine

ROSSERIAL kasutab jadaporti kaudu sõnumite ühendamisel ja edastamisel kiirust 57600 baiti/s. Eksperimentaalse meetodiga tehti kindlaks, et see kiirus on piisav ühenduse loomiseks ja primitiivsete sõnumite edastamiseks sagedusega mitte üle 3 Hz. Uurimisel oli samuti kindlaks tehtud, et stabiilne ühendus on parim kiirusel 921600 baiti/s. Samuti tuleb seadistada sisendi (*Input*) ja väljundi (*Output*) lõikepuhver suurusega 2048 *ROSSERIAL*-i protokolli jaoks ROS-i teegi failis *„ros.h"*. [vt Lisa 3]

```
typedef NodeHandle_<STM32Hardware, 25, 25, 2048, 2048> NodeHandle;
```

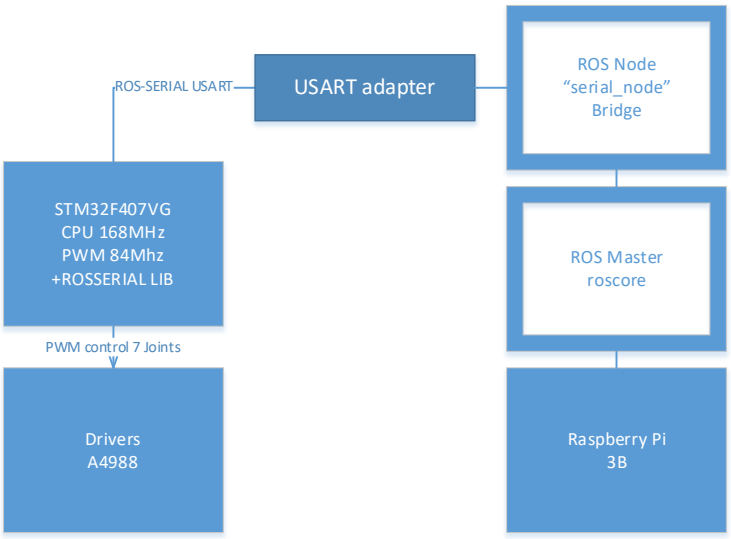
Riistvara seadistamiseks kasutatakse *STMCubeMX*-i utiliiti [vt Joonis 28] [36]:




*Joonis 28 STM32CubeMX seadistus USART ühenduse jaoks*

### 3.6.4 *ROS Serial Bridge*-i vastastikune toime

*Raspberry Pi 3B*-hostseadme ja kliendi STM32-mikrokontrolleri vahelise ühenduse tagamiseks *ROSSERIAL*-i protokolli kasutades on vaja käivitada ROS-i sõlm *"serial_node"*, mis on ROS-i meistri käivitatud hostseadme ja kliendi seadme vaheline proksiserver [vt Joonis 29].




*Joonis 29 ROSSERIAL Bridge-i sõlm ROS Master-i ja STM32 Client vahel*

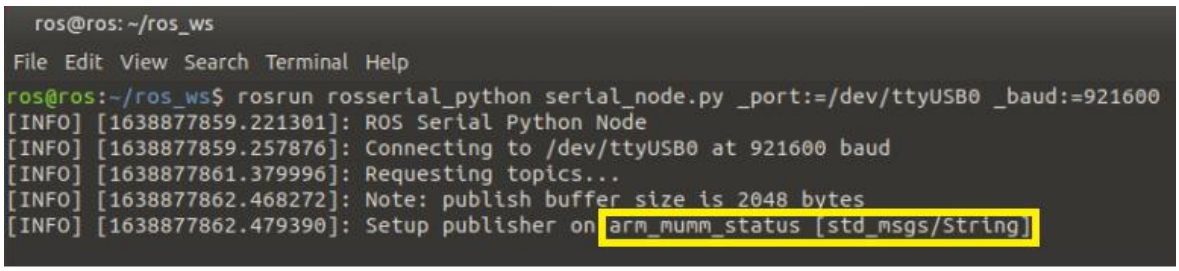
### 3.6.5 Ühenduse loomine

*"Serial_node"* käivitamiseks hostseadme *Raspberry Pi 3B (ROS Master)* poolel ja ühenduse loomiseks kliendi STM32-mikrokontrolleriga kasutatakse järgmist käsku:

```
rosrun rosserial_python serial_node.py _port:=/dev/ttyUSB0 _baud:=921600
```

Sel juhul kasutatakse ühendamiseks jadapordi USB-adapterit */dev/ttyUSB0* ja määratakse edastuskiirus, mis on võrdne STM32-seadmes määratud kiirusega – 921600 baiti/s.

Eduka ühenduse korral saadakse järgmine tulemus [vt Joonis 30]:




*Joonis 30 ROSSERIAL-i eduka ühenduse tulemus*

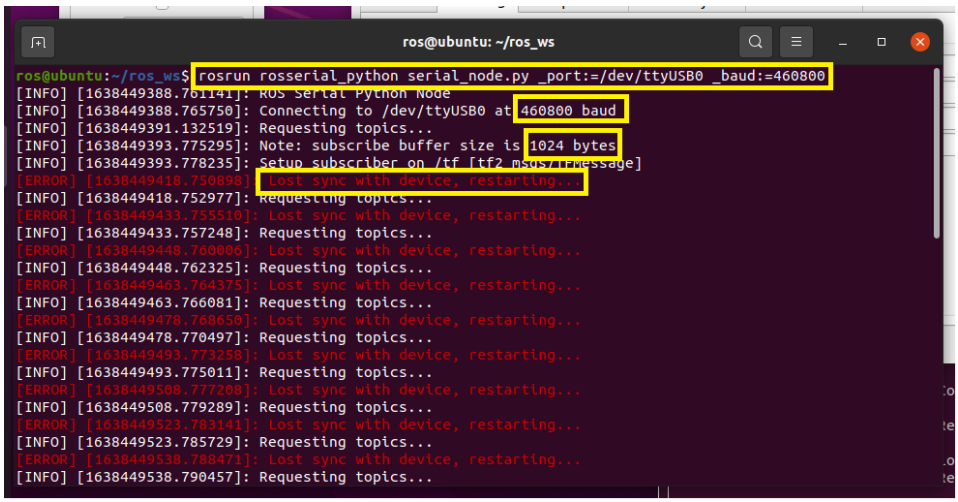
### 3.6.6 Ühendusküsimused

Hostseadme *Raspberry Pi 3B* ja STM32-mikrokontrolleri vahelise ühenduse loomisel ei olnud pikka aega võimalik stabiilset ühendust saada. Ametlikke ROS-i jadateeke kasutati nii hostseadme kui ka STM32-klientseadme jaoks. Kõik kasutatud seadmed olid töökorras. Kõik ühenduse seaded oli õigesti konfigureeritud.

Ühenduse loomise esimesel etapil käivitati STM32-mikrokontroller ühenduse otsimise režiimis ROS-i abonendi (*ROS Subscriber*) sõlmena, millel oli ühendus ROS-i teemaga *„/tf*". Silumiseks ja STM32-mikrokontrolleri küljel oleva ühenduse olemasolu visuaalseks määramiseks loodi lähtekoodis kahevärvilise LED-valgussignaali abil hoiatus ühenduse olemasolu või puudumise kohta. Samuti loodi teadaanne konsooli terminaliga ühenduse olemasolu kohta. Ühenduse loomise teisel etapil *Raspberry Pi 3B*-süsteemi hostipoolel käivitati sõlm *ROS Serial "serial_node"* koos vajalike pordi ja kiiruse seadistustega. Ühenduse olemasolu või puudumise silumiseks sõlmes on terminalikonsoolis juba sisseehitatud teavitusseadmed.

Mõlema seadme käivitamise järel saadi edukas ühendus koos teabe ja valgusmärguandega. Stabiilne ühendus kestis aga vaid 5 kuni 15 sekundit. 5–15 sekundi pärast ühendus katkes, millest andis teada konsooli infoväljund ja valgusmärguanne.

Probleemile kiirlahenduse leidmine ei õnnestunud. Vea kõrvaldamine  sünkroonimise ega taaskäivitamisega (*Lost sync with device, restarting*) ei olnud võimalik [vt Joonis 31]. Katsed luua ühendus teise arvutiga, kuhu oli paigaldatud sama versioon ROS-ist, USART-adapter ja olemasoleva robotmanipulaatori koopia, andsid tulemuseks sama vea. Seega vea võimalus *Raspberry Pi 3B*-seadme operatsioonisüsteemis oli välistatud.




*Joonis 31 Vigade kuvamine virtuaalmasina ROSSERIAL-i ühenduse sünkroonimise võimatuse kohta*

### 3.6.7 Ühendusküsimuste lahendamine

Selle jaoks, et teha kindlaks, millises piirkonnas võivad tekkida tõrked, mis põhjustavad ühenduse vea, otsustati jagada ülesanne rühmadesse.

- Probleemid füüsilises ühenduses (juhtmed)
- Probleemid ülekandeseadmetega (adapteritega)
- Tarkvaraprobleemid hostsüsteemis
  - Ülekandekiirusega seotud probleemid
  - Probleemid *ROSSERIAL*-i protokolli utiliidis *„rosserial_python"*
- STM32-kliendisüsteemi tarkvaraprobleemid
  - Ülekandekiirusega seotud probleemid
  - Lõikepuhvri mälu eraldamise probleemid
  - ROS Serial teekide probleemid

Seadmete **füüsilise ühendamise** probleemide väljaselgitamiseks asendati juhtmed erinevat tüüpi, pikkuse ja ristlõikega ning põimunud juhtmetega. Juhtmete asendamine ei mõjutanud kuidagi ühendustulemust.

Jadapordi riistvara on ühenduse stabiilsuse seisukohalt samuti väga oluline lüli. Proovitud sai erinevat tüüpi adaptereid, mida oli eelnevalt kontrollitud teiste seadmete peal ja mis ei põhjustanud rikkeid. **Ülekandeseadmete** väljavahetamine ei mõjutanud kuidagi ühenduse stabiilsust.

Võimaliku rikke kõrvaldamiseks **hostsüsteemis** käivitati *ROSSERIAL*-i *„rosserial_python"*-sõlmed erinevatel kiirustel. Testimisel tehti kindlaks, et stabiilse ühenduse kestus sõltub edastuskiirusest — mida aeglasem on kiirus, seda lühem on stabiilne ühendus. Aga kuna kiiruste muutmisel oli vaja muuta ühenduse kiirust nii host- kui ka kliendisüsteemides, siis ei saanud rikke põhjus olla üksnes hostsüsteemis. Märgates, et kõige stabiilsem ühendus toimub kiirusel 921600 baiti/s, otsustati sel kiirusel jätkata ühendusvea otsimist.

Hostsüsteemis testiti teist tüüpi *ROSSERIAL*-i sõlme *"rosserial_server"*. See sõlmetüüp erineb *"rosserial_python"*-ist selle poolest, et see on kirjutatud *C++* keeles ning sellel on parem jõudlus ja stabiilsus, kuid sellel on vähem silumisvõimalusi — see ei väljasta teavet konsooliterminali ühenduse kohta ega veateateid. Uurides ühendust selle utiliidiga, ei leitud muutusi ühenduse stabiilsuses.

STM32-**klientseadme** võimalike tõrgete uurimiseks oli juba kindlaks tehtud, et kiirus mõjutab ühenduse stabiilsust. Seejärel otsustati uurida, kuidas mõjutab *ROSSERIAL*-i protokolli lõikepuhvri suuruse muutmine. Lõikepuhvri suurendamisel 1024 baidilt 2048 baidile oli märgata ühenduse kestuse väikest muutust, kuid lõikepuhvri suuruse veelgi

suurem muutus ei mõjutanud ühenduse stabiilsust. Sellel etapil kontrolliti võimalike rikete avastamiseks kõiki sõlmi. Nüüd jäi õhku võimalus, et viga peitub STM32-mikrpkontrollerit teenindavas ROS-i teegis.

### 3.6.8 Stabiilse ühenduse lahendus

Pärast ametlikust ROS-i allikast paigaldatud teegi võimalike ühendustõrgete uurimist ei leitud stabiilse ühenduse probleemile lahendust. Ka sellistele ühendusvigadele avatud lähtekoodiga lahenduse otsimine ei andnud tulemusi.

Nüüd otsustati STM32-mikrokontrollerite jaoks leida alternatiivne teek, mis kasutab *ROSSERIAL*-i protokolli.

Pärast mitmete lahenduste uurimist ROS-i teekide kolmandate autorite juures leiti STM32-perekonna kontrollerite jaoks optimaalne variant, F4 versioon, mis põhineb *ROS Serial for STM32* teegi algsel versioonil [37].

Seadmete ühendamise sammude kordamisel *ROSSERIAL*-i protokollil kiirusega 921600 baiti/s loodi edukalt ühendus. Eksperimentaalsed uuringud on kinnitanud püsiühenduse stabiilsust.

Teegi muudetud versiooni lähtekoodi uurimisel leiti põhjus, miks ROS-i teegi algversioon stabiilselt ei töötanud. Algse teegi lõikepuhvri haldamise funktsioonidel oli vigu sissetulevate sõnumite arvutamisel ja töötlemisel.

## 3.7 Elektroonika sõlmede vastastikune toime

Robotmanipulaatori süsteemi võib jagada kolmeks põhiliseks plokiks:

1. Toiteallikad

2. Mikrokontrollerite juhtimissüsteemid

3. Mehhaanilised ajamisõlmed

### 3.7.1 Tarvitatav võimsus

Toiteploki projekteerimisel oli määratud kõikide elementide maksimaalne tarvitatav võimsus ning valitud väikse võimsusvaruga toiteplokk.

Iga elemendi tarvitatava võimsuse arvutus:

- 1x Raspberry Pi 3B – 730 mA
- 1x STM32F407VGTx – 140 mA
- 7x A4988 – 50 mA
- 7x NEMA 17 – 1.7 A
- 2x XL4015 – 100 mA

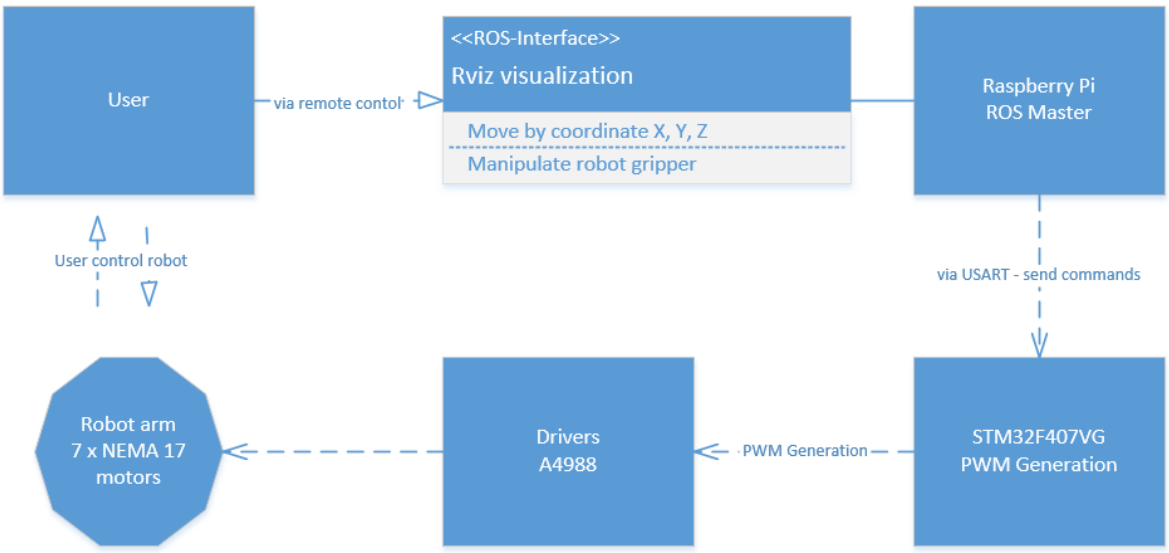
Üldine tarvitatav võimsus on 13320 mA = 13.3 A. Seejuures toiteploki võimsus on 20 A. Toiteploki pinge on 12 V. Mikrokontrolleri toiteks vajalik pinge on 5 V. Pinge muundamiseks 12 voldist mikrokontrollerite jaoks vajalikule 5 voldile on kasutatud pingemuundurit *XL4015*.

### 3.7.2 Kasutajaga interaktsiooni loogika

Robotmanipulaatori juhtimiseks peab kasutaja paigaldama baasoperatsioonisüsteemi kaugjuhtimise ROS-i süsteemi käivitamise ja visuaalse juhtimisinteraktsiooni jaoks. *Raspberry Pi 3B*-seadme hostsüsteemil on interaktsiooni jaoks paigaldatud krossplatvormi toetusega *VNC* server. [38]

Vajalike positsioonide konfigureerimise jaoks on kasutatud utiliiti *MoveIt Setup Assistant*. Visuaalse robotmanipulaatori otsejuhtimise jaoks kasutatakse utiliiti Rviz.

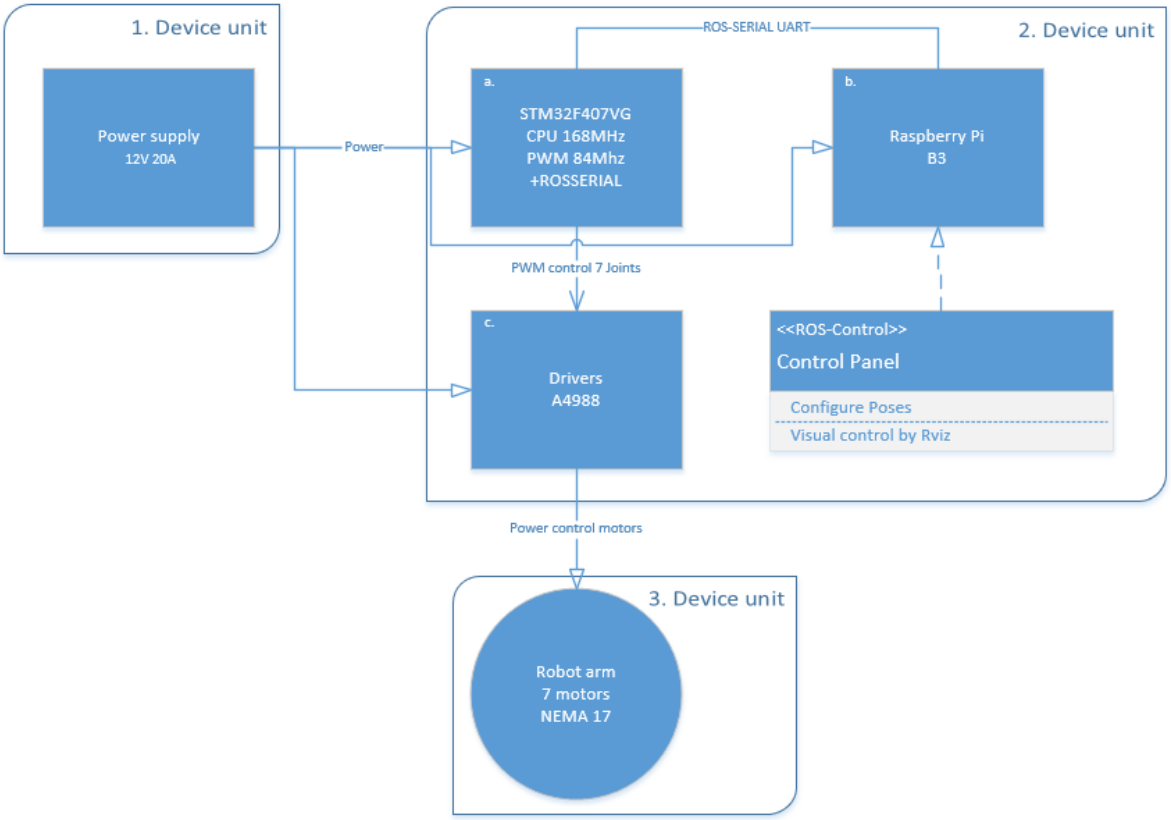
Üldine kasutaja ja robotmanipulaatori interaktsiooni skeem [vt Joonis 32]:




*Joonis 32 Kasutaja ja robotmanipulaatori interaktsioon*

### 3.7.3 Elektroonikasõlmede vastastikuse toime loogika

Allpool on toodud kõikide elektroonikasõlmede vastastikuse toime üldine skeem [vt Joonis 33].




*Joonis 33 Elektroonikasõlmede vastastikune toime*

Ülaltoodud skeemil on esialgselt projekteeritud [*"1. Device unit"*] ja [*"2. Device unit"*] moodulite ühendamine. Projekti realiseerimise käigus jäeti toiteploki paigaldamine ühel füüsilisel moodulil ära, kuna toiteplokk tekitas häireid (elektromagneetiline kiirgus) samm-mootorite draiveritele. Tulevikus on kavas nende moodulite ühendamine häirekindlates sektsioonides.

Loogikaskeem [vt Joonis 33] kirjeldab [*2. Device unit*], et *Raspberry Pi 3B*-d juhitakse arvutilt kaudselt või klaviatuuri ja arvutihiire abil lokaalselt. Rviz-i abil häälestatakse robotmanipulaatori asendi. Rviz-i kujundatud positsioneerimise andmed saadetakse STM32-mikrokontrollerile läbi ROSSERIAL-i protokolli. STM32-mikrokontroller genereerib PWM-signaali draiveritele A4988 saadud Rviz-i andmete baasil.

Draiverid A4988 juhivad robotmanipulaatori samm-mootorite pöörlemist [*3. Device unit*].

Kõik mikroskeemide moodulid saavad toidet toiteplokilt 12 V 20 A [*1. Device unit*].

# 4. PROGRAMMIDRAIVERI LOOMINE

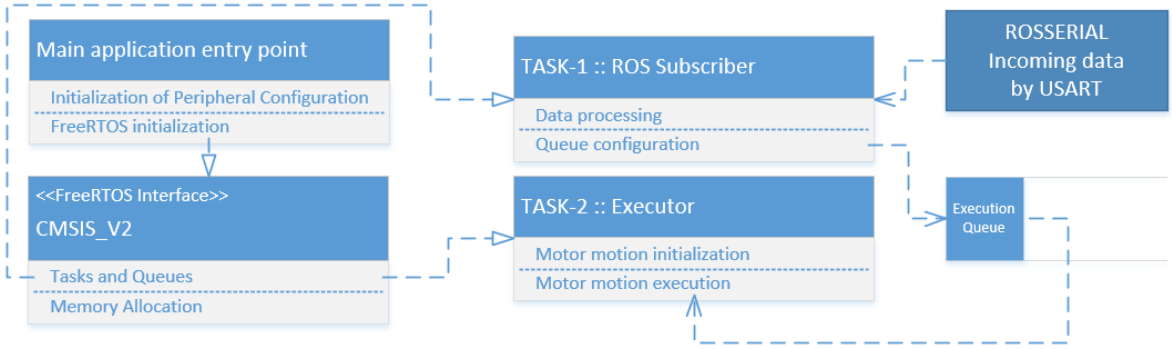
## 4.1 Draiveri rakenduse arhitektuur

Iga robotmanipulaatori mehhaanilisel sõlmel on HR4988-moodul — draiveriga juhitav samm-mootori ajam. Mooduldraiverit juhitakse STM32-kontrolleri genereeritud PWM-signaaliga [39]. [vt Lisa 1; Joonis 33]

PWM-signaali genereerimise jaoks kasutusel oleval STM32-mikrokontrolleril on 14 PWM-generaatorist sõltumatut signaali. Käesolevas projektis kasutatakse ainult 7 PWM-generaatorit 7 samm-mootori jaoks.

Sageduste ja väljundportide häälestust teostatakse programmikeskkonna *STMCubeMX* [vt Lisa 2] abil. Samuti aktiveeriti selles keskkonnas operatsioonisüsteem *FreeRTOS* [40] ja häälestati voojagamise parameetrid üksikülesannete jaoks.

Töötati välja ka samm-mootorite juhtimise programmidraiverite arhitektuur interaktsiooni jaoks ROS-i süsteemiga [vt Joonis 34]. Nende draiverite abil saadud ruumis positsioneerimise andmed muundatakse robotmanipulaatori sõlmede samm-mootorite liikumiseks.




*Joonis 34 Draiveri rakenduse arhitektuur robot-manipulaatori jaoks*

Ülaltoodud skeemis "Joonis 34" näitab robotmanipulaatori mehhaaniliste sõlmede juhtimise programmidraiveri arhitektuuri. *FreeRTOS Interface'i* abil loodi kaks sõltumatut sihtülesannet — TASK-1 ja TASK-2. TASK-1 töötleb ROS-i süsteemilt saabunud sisendsõnumeid ROSSERIAL-i protokolli järgi ning moodustab mehhaaniliste sõlmede teisaldamise positsioonide järjekorra [Execution Queue]. TASK-2 töötleb järjekorra [Execution Queue] andmeid ning genereerib vastavalt nendele PWM-signaali.

## 4.2 ROSSERIAL-i sisend-sõnumite töötlemine

FreeRTOS liidese loodud sihtülesanne TASK-1 sisendsõnumite töötlemiseks STM32-mikrokontrolleril ROSSERIAL-i protokolli järgi on suletud tsükli protsessi analoog, mis ei mõjuta teisi sihtülesandeid üldises süsteemis. [vt Lisa 3]

ROS-i teemaga (*topic*) "/tf" ühendamise jaoks loodi abonent *ROS Subscriber*:

```
ros::Subscriber<tf2_msgs::TFMessage> joints_subscriber("/tf", &CallBack);
```

Antud abonent täidab *CallBack*-funktsiooni *„tf2_msgs::TFMessage*"-tüüpi sõnumite saamisel. *CallBack*-funktsioonis toimub saadud andmete massiivi töötlemine ning roboti asendi muutumise kontroll võrreldes eelmiste väärtustega. Kui mingi *Joint*-sõlme positsioonide väärtused on muutunud, siis uued positsioonide väärtused lisatakse täitmise järjekorda *„[Execution Queue]"* TASK-2 voo jaoks. [vt Lisa 3]

Sõltumatu sihtülesanne TASK-2 jälgib järjekorra olekut ning järjekorras uute elementide tekkimisel töötleb neid elemente. Elemendi struktuuris on märgitud *Joint*-sõlme number, eelmine positsioon, uus positsioon, eelmine ajamärgis ja uus ajamärgis. Elemendi struktuuri andmete põhjal toimub samm-mootori nihke (impulssite arv) ning perioodi kiirenduse (signaali laius) arvutus ajamärgiste vahel. Saadud impulssite arvu ja signaali laiuse arvutused täidetakse PWM-signaali genereerimise erifunktsioonis. Peale teisaldamise täitmist vastavalt elemendi andmetele eemaldatakse täidetud element järjekorrast.

Tulevikus on kavas luua *ROS Publisher*, mis saadab *ROS Master*-isse andmed roboti positsioneerimise kohta.

# KOKKUVÕTE

Antud lõputöö käigus loodi 3D-modelleerimise ja 3D-printimise tehnoloogiate abil robotmanipulaator. Selle juhtimiseks ruumis loodi RSOS Rviz-i abil roboti URDF-i mudel. Nende kokkupanek võimaldab roboti liikumise planeerimist ja teisaldamise koordinaatide edastamist jadaporti abil. Töö tulemusena täideti roboti modelleerimise täistsükkel — mehhaanilise osa ja selle komponentide planeerimisest kuni robotmanipulaatori side- ja juhtimistarkvara arendamiseni.

Lõputöö eesmärk oli robotmanipulaatori väljatöötamine ning varustamine tööstusliku juhtimissüsteemi ROS avatud lähtekoodiga. Projekti peamiseks komponendiks on juhtimisploki autonoomsus mitme mikrokontrolleri paigaldamisega ühes moodulis. Antud eesmärgi saavutamiseks tuli ROS-i süsteemi põhjalikult uurida ning viia ellu kõik projektiosad.

Töö tulemuseks on valmis juhtimisseade, mille saab tööle panna eemalt arvuti abil või kohapeal klaviatuuri ja arvutihiirt kasutades.

Tehtud tööd hinnates võib öelda, et tulemus jätab väga hea mulje. Töö käigus saadud kogemuse tulemusena on autoril nüüd olemas ettekujus ning järeldused projekti arhitektuurist ning samuti sellest, mida tuleb arvesse võtta ja muuta robotmanipulaatori järgmises versioonis.

Tulevikus on kavas viia täielikult ellu robotmanipulaatori mehhaaniliste liigendsõlmede (*Joint*-sõlmede) juhtimise programmidraiverite arendamine projekteeritud arhitektuuri abil.

# SUMMARY

Currently, the topic of robots is becoming an integral part of human life and is associated with the technologies of the future. The purpose of this thesis is to create a robotic manipulator using 3D modeling and 3D printing technologies.

The title of the topic describes the technology used to control robots in this work - "Creation of a robotic manipulator based on the ROS control system".

The graduation thesis is composed of four chapters, each of them dealing with different aspects:

1. Description of the mechanical part of the robotic manipulator, analysis and selection of components
2. Creation of a 3D-model of a robotic manipulator
3. Implementation and setup of the ROS control system
4. Creation of the architecture of the software driver for interaction with ROS

During the dissertation, a robotic manipulator was created using 3D modeling and 3D printing technology. A URDF-model of a robot based on ROS Rviz was created to control a robotic manipulator in a coordinate system. The assembly allows you to plan the movement of the robot and transmit the coordinates of the movement through the serial port.

As a result of the work, a full cycle of robot development was completed - from planning the mechanical part and its components, to developing software for communication and control of the hardware of the robotic manipulator.

The result of the work is a ready-made control device through a computer remotely or using a keyboard and mouse locally. In the future, it is planned to fully implement the development of software drivers for controlling the mechanical Joint-units of the robotic-manipulator, according to the designed architecture.

Dissertation author: Jevgeni Kostenko

# KASUTATUD KIRJANDUSE LOETELU


1. ROS (Robot Operating System) [Online]
https://en.wikipedia.org/wiki/Robot_Operating_System (01.12.2021)

2. Top 10 DIY & 3D Printed Robot Arms [Online]

https://all3dp.com/2/10-best-robot-arms-to-3d-print-or-buy/ (01.12.2021)

3. NEMA 17 Stepper Motor [Online]

https://aliexpress.ru/item/32830931647.html (01.12.2021)

4. HR4988 Stepper Motor Drivers [Online]

https://aliexpress.ru/item/32906506347.html (01.12.2021)

5. Microcontroller STM32F407VGTx [Online]
https://www.keil.com/dd2/stmicroelectronics/stm32f407vgtx/ (01.12.2021)

6. Endstop Mechanical [Online]

https://aliexpress.ru/item/32956547677.html (01.12.2021)

7. Power Supply [Online]

https://aliexpress.ru/item/32911389359.html (01.12.2021)

8. Power Converter [Online]

https://aliexpress.ru/item/1084552308.html (02.12.2021)

9. Raspberry Pi Official [Online]

https://www.raspberrypi.org/products/raspberry-pi-3-model-b (02.12.2021)

10. Bearing 30*47*12 mm [Online]

https://aliexpress.ru/item/885585348.html (02.12.2021)

11. Bearing 52*40*7 mm [Online]

https://aliexpress.ru/item/32899281440.html (02.12.2021)

12. Bearing 8*22*7 [Online]

https://aliexpress.ru/item/32342893985.html (02.12.2021)

13. Bearing 5*10*4 [Online]

https://aliexpress.ru/item/32830761829.html (02.12.2021)

14. SolidWorks to URDF Exporter [Online]

http://wiki.ros.org/sw_urdf_exporter (02.12.2021)

15. Ubuntu OS List for Raspberry Pi [Online]

https://ubuntu.com/download/raspberry-pi (02.12.2021)

16. Raspberry Pi OS with desktop [Online]
https://downloads.raspberrypi.org/raspios_armhf/images/raspios_armhf-2021-11-08/2021-10-30-raspios-bullseye-armhf.zip (04.12.2021)

17. Ubuntu Mate 18.04 for Raspberry Pi 3 [Online]

https://ubuntu-mate.community/t/i-need-download-ubuntu-mate-18-04/23495 (04.12.2021)

18. Ubuntu Server 20.04 Official Image [Online]
https://ubuntu.com/download/raspberry-pi/thank-you?version=20.04.3&architecture=server-arm64+raspi (04.12.2021)

19. ROS Distributions [Online]

http://wiki.ros.org/Distributions (04.12.2021)

20. Ubuntu Server Installation Instruction [Online]

https://ubuntu.com/tutorials/how-to-install-ubuntu-on-your-raspberry-pi#1-overview (04.12.2021)

21. Raspberry Pi Configuration [Online]

https://elinux.org/RPiconfig (04.12.2021)

22. Ubuntu install of ROS Noetic [Online]
http://wiki.ros.org/noetic/Installation/Ubuntu (04.12.2021)

23. Meta package that contains all essential package of MoveIt. [Online]
https://moveit.ros.org/ (05.12.2021)

24. 3D visualization tool for ROS [Online]

http://wiki.ros.org/rviz (05.12.2021)

25. Metapackage for core of rosserial. [Online]

http://wiki.ros.org/rosserial (05.12.2021)

26. MoveIt Setup Assistant Tutorial [Online]
http://docs.ros.org/en/hydro/api/moveit_setup_assistant/html/doc/tutorial.html (05.12.2021)

27. ROS nodes, topics, and messages [Online]
https://subscription.packtpub.com/book/hardware_and_creative/9781788479592/1/ch01lvl1sec13/ros-nodes-topics-and-messages (05.12.2021)

28. ROS Nodes [Online]

http://wiki.ros.org/Nodes (05.12.2021)

29. ROS Topics [Online]

http://wiki.ros.org/Topics (05.12.2021)

30. ROS Messages [Online]

http://wiki.ros.org/Messages (05.12.2021)

31. ROS Master [Online]

http://wiki.ros.org/Master (06.12.2021)

32. ROS Message type tf2_msgs/TFMessage [Online]
http://docs.ros.org/en/melodic/api/tf2_msgs/html/msg/TFMessage.html (07.12.2021)

33. ROS Serial Protocol [Online]

http://wiki.ros.org/rosserial (08.12.2021)

34. ROS Serial Repository for STM32 microcontrollers [Online]
https://github.com/yoneken/rosserial_stm32 (08.12.2021)

35. How to configure STM32CubeIDE to support C++ development [Online]
https://community.st.com/s/question/0D50X0000At0EJ3/how-to-configure-stm32cubeide-to-support-c-development (09.12.2021)

36. STM32Cube initialization code generator [Online]
https://www.st.com/en/development-tools/stm32cubemx.html (10.12.2021)

37. ROSserial on STM32F4 [Online]

https://github.com/xav-jann1/rosserial_stm32f4 (11.12.2021)

38. VNC Server for Raspberry Pi 3B remote control [Online]
https://www.realvnc.com/en/connect/download/vnc/raspberrypi/ (12.12.2021)

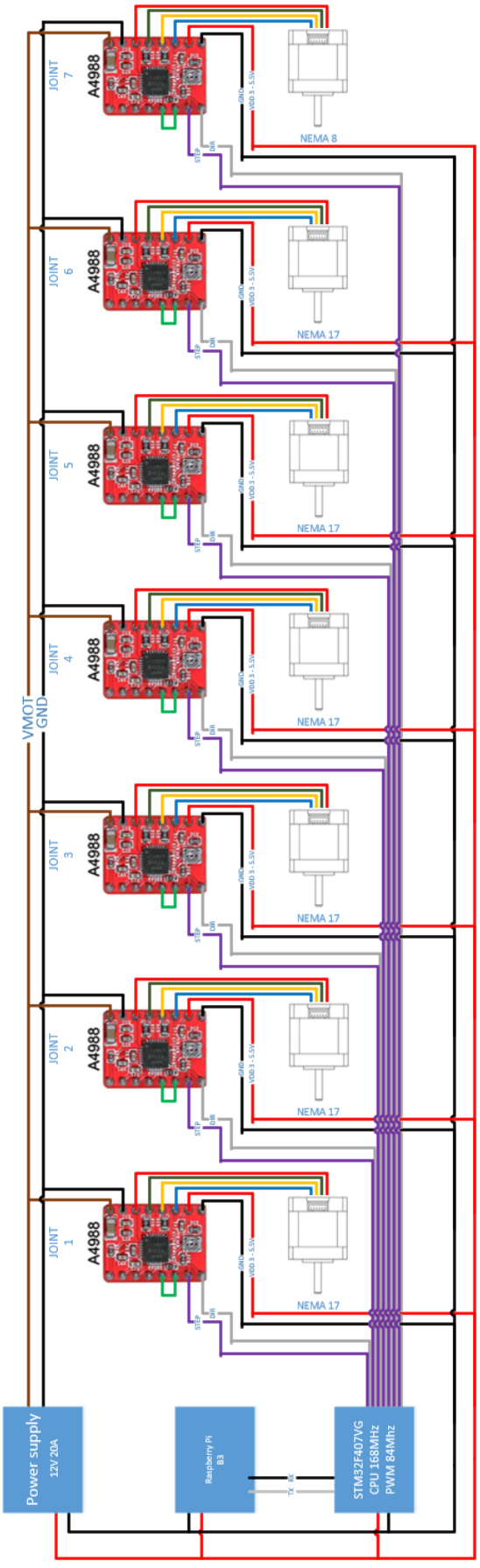
39. Pulse-width modulation [Online]

https://en.wikipedia.org/wiki/Pulse-width_modulation (14.12.2021)

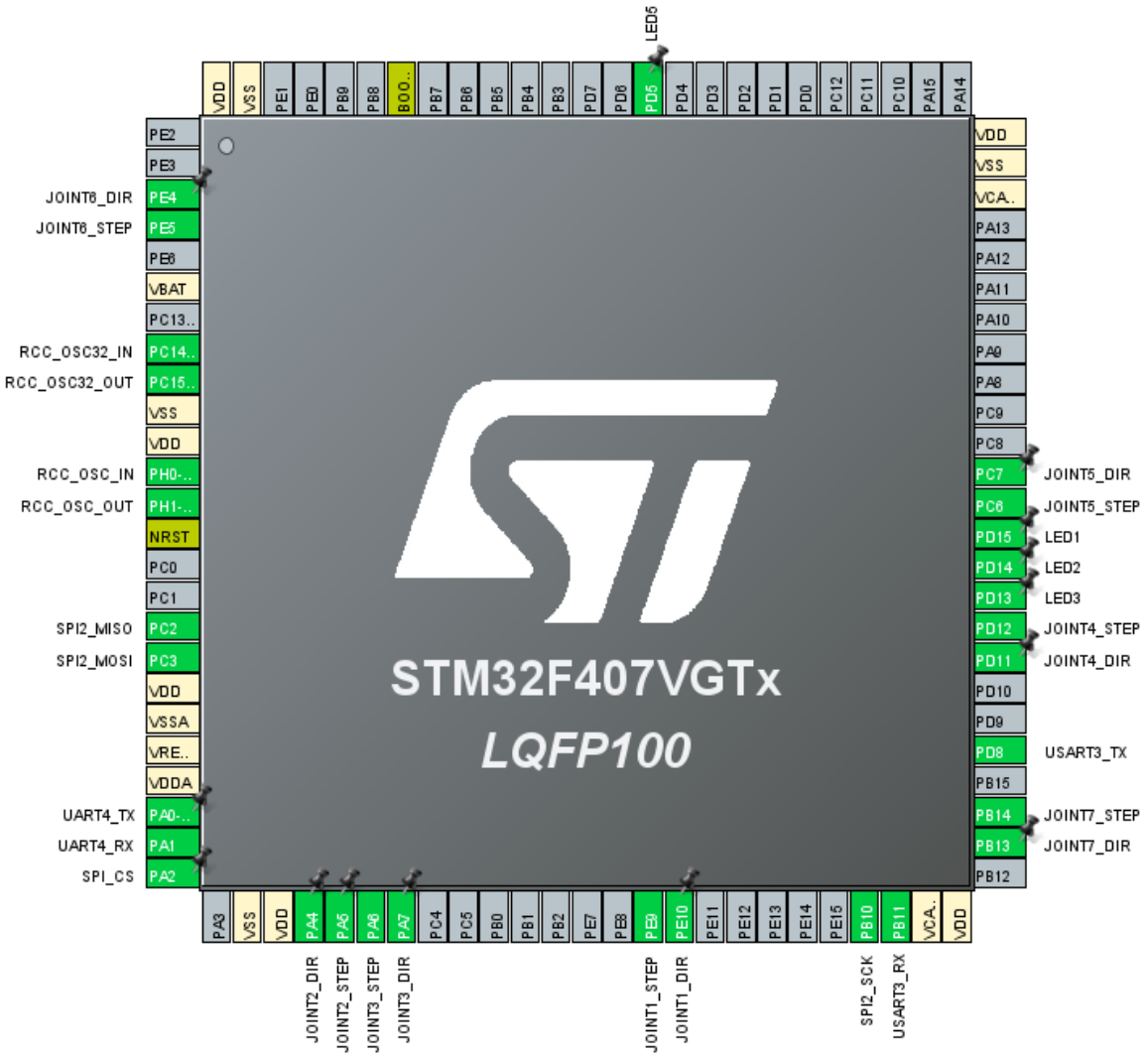
40. Real-time operating system for microcontrollers [Online]
https://www.freertos.org/ (14.12.2021)

# LISAD

**Lisa 1**: Juhi juhtlülitus.



**Lisa 2**: STMCubeMX tarkvarakeskkonna signaali väljundi sageduste ja portide seadistamine.



**Lisa 3**: FreeRTOS TASK-1 design

File "**main.c**" – initialization of TASK-1:

```
void StartDefaultTask(void *argument)
{
  /* USER CODE BEGIN 5 */
       /* Infinite loop */
       setup();
       for (;;) {
              loop();
       }
  /* USER CODE END 5 */
}
```

File "**ros.h**" – configuration of ROSSERIAL:

```
#ifndef _ROS_H_
#define _ROS_H_

#include "STM32Hardware.h"
#include "ros/node_handle.h"

namespace ros {
       typedef NodeHandle_<STM32Hardware, 25, 25, 1024, 1024> NodeHandle;
}
#endif
```

File "**RobotArm.cpp**" – configuration of TASK-1:

```
#include <RobotArm.h>
#include <ros.h>
#include <ros/node_handle.h>
#include <ros/subscriber.h>
#include <ros/duration.h>
#include <ros/msg.h>
#include <ros/publisher.h>
#include <ros/time.h>
#include <ros/service_client.h>
#include <ros/service_server.h>
#include <tf2_msgs/TFMessage.h>
#include "classes/Joint.h"
#include <cmath>
#include <std_msgs/String.h>
#include "stm32f4xx_hal_uart.h"
#include "main.h"
#include <string>
#include <sstream>
#include <iomanip>

using namespace std;
extern UART_HandleTypeDef huart4;
extern UART_HandleTypeDef huart3;
UART_HandleTypeDef ROSSERIAL_PORT = *&huart4;
```

```
ros::NodeHandle nh;

std_msgs::String str_msg;
ros::Publisher chatter("arm_mumm_status", &str_msg);

void joint_data_callback(const tf2_msgs::TFMessage &data);
ros::Subscriber<tf2_msgs::TFMessage> joints_subscriber("/tf",
&joint_data_callback);

const double PI = 3.14159265358979238463;

double round_up(double value, int decimal_places) {
	const double multiplier = std::pow(10.0, decimal_places);
	return std::ceil(value * multiplier) / multiplier;
}


Joint J1("J1");
Joint J2("J2");
Joint J3("J3");
Joint J4("J4");
Joint J5("J5");
Joint J6("J6");

struct EulerRollPitchYawAngles {
	double roll = 0.0, pitch = 0.0, yaw = 0.0;
};

struct Quaternion {
	double w = 0.0, x = 0.0, y = 0.0, z = 0.0;
};

EulerRollPitchYawAngles rpy_angles;
EulerRollPitchYawAngles& Euler2RollPitchYaw(Quaternion &q) {
	// roll
	double sinroll_cospitch = 2 * (q.w * q.x + q.y * q.z);
	double cosroll_cospitch = 1 - 2 * (q.x * q.x + q.y * q.y);
	rpy_angles.roll = std::atan2(sinroll_cospitch, cosroll_cospitch);
	// pitch
	double sinpitch = 2 * (q.w * q.y - q.z * q.x);
	if (std::abs(sinpitch) >= 1)
		rpy_angles.pitch = std::copysign(M_PI / 2, sinpitch);
	else
		rpy_angles.pitch = std::asin(sinpitch);
	// yaw
	double sinyaw_cospitch = 2 * (q.w * q.z + q.x * q.y);
	double cosyaw_cospitch = 1 - 2 * (q.y * q.y + q.z * q.z);
	rpy_angles.yaw = std::atan2(sinyaw_cospitch, cosyaw_cospitch);
	return rpy_angles;
}
Quaternion quaternion;
EulerRollPitchYawAngles& JointTrackVal(Joint &joint) {
	quaternion.x = joint.X;
	quaternion.y = joint.Y;
	quaternion.z = joint.Z;
	quaternion.w = joint.W;
	return Euler2RollPitchYaw(quaternion);
}
void InitTfSubscriber() {
```

```
	//:::::::::::::::::::::
	J1.LowerLimit = 3.14;
	J1.UpperLimit = 3.14;
	//:::::::::::::::::::::
	J2.LowerLimit = -3.14;
	J2.UpperLimit = 1.0;
	//:::::::::::::::::::::
	J3.LowerLimit = -3.14;
	J3.UpperLimit = -0.2;
	//:::::::::::::::::::::
	J4.LowerLimit = -3.14;
	J4.UpperLimit = 3.14;
	//:::::::::::::::::::::
}


void setup() {
	nh.initNode();
	nh.getHardware()->init();
	nh.subscribe(joints_subscriber);
}

void loop() {
	nh.spinOnce();
	uint16_t d = 10;
	if (nh.connected()) {
		HAL_GPIO_WritePin(GPIOD, LED1_Pin, GPIO_PIN_SET);
		osDelay(d / 2);
	} else {
		HAL_GPIO_WritePin(GPIOD, LED2_Pin, GPIO_PIN_SET);
		osDelay(d / 2);
	}
	HAL_GPIO_WritePin(GPIOD, LED1_Pin, GPIO_PIN_RESET);
	HAL_GPIO_WritePin(GPIOD, LED2_Pin, GPIO_PIN_RESET);
	HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
	osDelay(d / 2);
}

EulerRollPitchYawAngles rpy;
bool hasChanges = false;
double_t rounded = 0.0, current_track_val = 0.0;
Joint tmp("TMP");
geometry_msgs::TransformStamped joint_1;
geometry_msgs::TransformStamped joint_2;
geometry_msgs::TransformStamped joint_3;
geometry_msgs::TransformStamped joint_4;
geometry_msgs::TransformStamped joint_5;
geometry_msgs::TransformStamped joint_6;
tf2_msgs::TFMessage::_transforms_type *transform_arr;

void joint_data_callback(const tf2_msgs::TFMessage &data) {
	transform_arr = data.transforms;
	joint_1 = transform_arr[0];
	joint_2 = transform_arr[1];
	joint_3 = transform_arr[2];
	joint_4 = transform_arr[3];
	joint_5 = transform_arr[4];
	joint_6 = transform_arr[5];
```

```
// ---------------------------
// JOINT 1
tmp.SetTransform(joint_1);
if (tmp.X != J1.X || tmp.Y != J1.Y || tmp.Z != J1.Z || tmp.W != J1.W) {
	J1.SetTransform(joint_1);
	// Limits and track range
	rpy = JointTrackVal(J1);
	double_t track_val_J1 = rpy.yaw;
	J1.LastTrackValue = track_val_J1;
	rounded = round_up(J1.LastTrackValue, 2);
	if (J1.LastTrackValueRounded != rounded) {
		J1.LastTrackValueRounded = rounded;
		hasChanges = true;
	}
}
// ---------------------------

// ---------------------------
// JOINT 2
tmp.SetTransform(joint_2);
if (tmp.X != J2.X || tmp.Y != J2.Y || tmp.Z != J2.Z || tmp.W != J2.W) {
	J2.SetTransform(joint_2);
	// Limits and track range
	rpy = JointTrackVal(J2);
	current_track_val = abs(rpy.pitch);
	if (J2.W < 0 and rpy.pitch < 0)
		current_track_val = abs(rpy.pitch);
	else if (J2.W > 0 and rpy.pitch < 0)
		current_track_val = PI / 2 + (PI / 2 - abs(rpy.pitch));
	else if (J2.W > 0 and rpy.pitch > 0 and rpy.yaw < PI)
		current_track_val = (-(PI / 2 + (PI / 2 - abs(rpy.pitch))));
	else
		current_track_val = (-rpy.pitch);

	double_t track_val_J2 = (-current_track_val);
	J2.LastTrackValue = track_val_J2;
	rounded = round_up(J2.LastTrackValue, 2);
	if (J2.LastTrackValueRounded != rounded) {
		J2.LastTrackValueRounded = rounded;
		hasChanges = true;
	}
}
// ---------------------------
// JOINT 3
tmp.SetTransform(joint_3);
if (tmp.X != J3.X or tmp.Y != J3.Y or tmp.Z != J3.Z or tmp.W != J3.W) {
	J3.SetTransform(joint_3);
	// Limits and track range
	rpy = JointTrackVal(J3);
	double_t track_val_J3 = 0.0;
	current_track_val = 0;

	if (rpy.yaw <= 0)
		current_track_val = PI + rpy.yaw;
	else
		current_track_val = (-(PI - rpy.yaw));

	track_val_J3 = current_track_val;
	J3.LastTrackValue = track_val_J3;
```

```
        rounded = round_up(J3.LastTrackValue, 2);
        if (J3.LastTrackValueRounded != rounded) {
            J3.LastTrackValueRounded = rounded;
            hasChanges = true;
        }
    }
    // ---------------------------
    // JOINT 4
    tmp.SetTransform(joint_4);
    if (tmp.X != J4.X or tmp.Y != J4.Y or tmp.Z != J4.Z or tmp.W != J4.W) {
        J4.SetTransform(joint_4);
        // Limits and track range
        rpy = JointTrackVal(J4);
        current_track_val = 0;
        if (rpy.pitch <= 0 and rpy.roll < 0)
            current_track_val = rpy.pitch;
        else if (rpy.pitch < 0 and rpy.roll > 0)
            current_track_val = (-(PI / 2 + (PI / 2 + rpy.pitch)));
        else if (rpy.pitch > 0 and rpy.roll < 0 and rpy.yaw < 0)
            current_track_val = rpy.pitch;
        else if (rpy.pitch > 0 and rpy.roll > 0 and rpy.yaw > 0)
            current_track_val = (PI / 2 + (PI / 2 - rpy.pitch));
        else
            current_track_val = rpy.pitch;

        double_t track_val_J4 = (-current_track_val);
        J4.LastTrackValue = track_val_J4;
        rounded = round_up(J4.LastTrackValue, 2);
        if (J4.LastTrackValueRounded != rounded) {
            J4.LastTrackValueRounded = rounded;
            hasChanges = true;
        }
    }
    // ---------------------------
    // JOINT 5
    tmp.SetTransform(joint_5);
    if (tmp.X != J5.X or tmp.Y != J5.Y or tmp.Z != J5.Z or tmp.W != J5.W) {
        J5.SetTransform(joint_5);
        // Limits and track range
        rpy = JointTrackVal(J5);
        current_track_val = 0;
        if (rpy.pitch < 0 and rpy.roll > 0 and rpy.yaw > 0)
            current_track_val = (-(rpy.pitch + PI / 2));
        else if (rpy.pitch > 0 and rpy.roll > 0 and rpy.yaw > 0)
            current_track_val = (-(PI / 2 + rpy.pitch));
        else if (rpy.pitch < 0 and rpy.roll < 0 and rpy.yaw < 0)
            current_track_val = PI / 2 + rpy.pitch;
        else if (rpy.pitch > 0 and rpy.roll < 0 and rpy.yaw < 0)
            current_track_val = PI / 2 + rpy.pitch;

        double_t track_val_J5 = current_track_val;
        J5.LastTrackValue = track_val_J5;
        rounded = round_up(J5.LastTrackValue, 2);
        if (J5.LastTrackValueRounded != rounded) {
            J5.LastTrackValueRounded = rounded;
            hasChanges = true;
        }
    }
    // ---------------------------
```

```
        // JOINT 6
        tmp.SetTransform(joint_6);
        if (tmp.X != J6.X or tmp.Y != J6.Y or tmp.Z != J6.Z or tmp.W != J6.W) {
              J6.SetTransform(joint_6);
              // Limits and track range
              rpy = JointTrackVal(J6);
              current_track_val = 0;
              if (rpy.pitch > 0 and rpy.roll < 0 and rpy.yaw < 0)
                    current_track_val = (-rpy.pitch);
              else if (rpy.pitch > 0 and rpy.roll > 0 and rpy.yaw > 0)
                    current_track_val = (-(PI / 2 + (PI / 2 - rpy.pitch)));
              else if (rpy.pitch < 0 and rpy.roll < 0 and rpy.yaw < 0)
                    current_track_val = (-rpy.pitch);
              else if (rpy.pitch < 0 and rpy.roll > 0 and rpy.yaw > 0)
                    current_track_val = (PI / 2 + rpy.pitch) + PI / 2;

              double_t track_val_J6 = current_track_val;
              J6.LastTrackValue = track_val_J6;
              rounded = round_up(J6.LastTrackValue, 2);
              if (J6.LastTrackValueRounded != rounded) {
                    J6.LastTrackValueRounded = rounded;
                    hasChanges = true;
              }

        }
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_SET);
        // ---------------------------

        if (hasChanges) {
              HAL_GPIO_WritePin(GPIOD, LED2_Pin, GPIO_PIN_SET);
              hasChanges = false;
        }
}
```

**Lisa 4**: ROS-i teema "/tf" andmete töötlemine Python rospy.Subscriber abil.

```
#!/usr/bin/env python

import rospy
from tf2_msgs.msg import TFMessage
from tf.transformations import *
import math
import numpy as np

class Joint:
    def __init__(self):
        self.x = None
        self.y = None
        self.z = None
        self.w = None
        self.lower_limit = None
        self.upper_limit = None
        self.last_track_value = 0.0
        self.last_track_value_rounded = 0.0

    def set_rotation(self, joint):
        self.x = joint.transform.rotation.x
        self.y = joint.transform.rotation.y
        self.z = joint.transform.rotation.z
        self.w = joint.transform.rotation.w


# Declaring Joint classes and his limits
#:::::::::::::::::::::
J1 = Joint()
J1.lower_limit = -3.14
J1.upper_limit = 3.14
#:::::::::::::::::::::
J2 = Joint()
J2.lower_limit = -3.14
J2.upper_limit = 1.0
#:::::::::::::::::::::
J3 = Joint()
J3.lower_limit = (-math.pi)
J3.upper_limit = -0.2
#:::::::::::::::::::::
J4 = Joint()
J4.lower_limit = -3.14
J4.upper_limit = 3.14
#:::::::::::::::::::::
J5 = Joint()
```

```
#:::::::::::::::::::::
J6 = Joint()
#:::::::::::::::::::::


def joint_trackval(joint):
    orientation_list = [joint.x, joint.y, joint.z, joint.w]
    return euler_from_quaternion(orientation_list)


def callback(data):
    global roll, pitch, yaw
    transform_arr = data.transforms
    # get array values
    joint_1 = transform_arr[0]
    joint_2 = transform_arr[1]
    joint_3 = transform_arr[2]
    joint_4 = transform_arr[3]
    joint_5 = transform_arr[4]
    joint_6 = transform_arr[5]
    print("callback OK")
     # init joint instances values
    # --------------------------
    # JOINT 1
    tmp_joint_1 = Joint()
    tmp_joint_1.set_rotation(joint_1)
    if tmp_joint_1.x != J1.x or tmp_joint_1.y != J1.y or tmp_joint_1.z !=
J1.z or tmp_joint_1.w != J1.w:
        J1.set_rotation(joint_1)
        # Limits and track range
        (roll, pitch, yaw) = joint_trackval(J1)
        track_val_J1 = yaw
        J1.last_track_value = track_val_J1
        rounded = round(J1.last_track_value, 2)
        if (J1.last_track_value_rounded != rounded):
            print("J1 track:%2.10f %2.4f -> %2.4f" % (track_val_J1,
J1.last_track_value_rounded, rounded))
            J1.last_track_value_rounded = rounded
    # --------------------------
    # JOINT 2
    tmp_joint_2 = Joint()
    tmp_joint_2.set_rotation(joint_2)
    if tmp_joint_2.x != J2.x or tmp_joint_2.y != J2.y or tmp_joint_2.z !=
J2.z or tmp_joint_2.w != J2.w:
        J2.set_rotation(joint_2)
        # Limits and track range
        (roll, pitch, yaw) = joint_trackval(J2)
        current_track_val = abs(pitch)
        if J2.w < 0 and pitch < 0:
```

```
            current_track_val = abs(pitch)
        elif J2.w > 0 and pitch < 0:
            current_track_val = math.pi / 2 + (math.pi / 2 - abs(pitch))
        elif J2.w > 0 and pitch > 0 and yaw < 3.14:
            current_track_val = (-(math.pi / 2 + (math.pi / 2 -
abs(pitch))))
        else:
            current_track_val = (-pitch)

        track_val_J2 = (-current_track_val)
        J2.last_track_value = track_val_J2
        rounded = round(J2.last_track_value, 2)
        if (J2.last_track_value_rounded != rounded):
            print("J2 track:%2.10f %2.4f -> %2.4f" % (track_val_J2,
J2.last_track_value_rounded, rounded))
            J2.last_track_value_rounded = rounded
    # --------------------------
    # JOINT 3
    tmp_joint_3 = Joint()
    tmp_joint_3.set_rotation(joint_3)
    if tmp_joint_3.x != J3.x or tmp_joint_3.y != J3.y or tmp_joint_3.z !=
J3.z or tmp_joint_3.w != J3.w:
        J3.set_rotation(joint_3)
        # Limits and track range
        (roll, pitch, yaw) = joint_trackval(J3)
        track_val_J3 = 0
        current_track_val = 0
        if(yaw <= 0):
            current_track_val = math.pi + yaw
        else:
            current_track_val = (-(math.pi - yaw))

        track_val_J3 = current_track_val
        J3.last_track_value = track_val_J3
        rounded = round(J3.last_track_value, 2)
        if (J3.last_track_value_rounded != rounded):
            print("J3 track:%2.10f %2.4f -> %2.4f" % (track_val_J3,
J3.last_track_value_rounded, rounded))
            J3.last_track_value_rounded = rounded
    # --------------------------
    # JOINT 4
    tmp_joint_4 = Joint()
    tmp_joint_4.set_rotation(joint_4)
    if tmp_joint_4.x != J4.x or tmp_joint_4.y != J4.y or tmp_joint_4.z !=
J4.z or tmp_joint_4.w != J4.w:
        J4.set_rotation(joint_4)
        # Limits and track range
        (roll, pitch, yaw) = joint_trackval(J4)
        current_track_val = 0
```

```
        if(pitch <= 0 and roll < 0):
            current_track_val = pitch
        elif (pitch < 0 and roll > 0):
            current_track_val = (-(math.pi / 2 + (math.pi / 2 + pitch)))
        elif (pitch > 0 and roll < 0 and yaw < 0):
            current_track_val = pitch
        elif (pitch > 0 and roll > 0 and yaw > 0):
            current_track_val = (math.pi / 2 + (math.pi / 2 - pitch))
        else:
            current_track_val = pitch

        track_val_J4 = (-current_track_val)
        J4.last_track_value = track_val_J4
        rounded = round(J4.last_track_value, 2)
        if (J4.last_track_value_rounded != rounded):
            print("J4 track:%2.10f %2.4f -> %2.4f" % (track_val_J4,
J4.last_track_value_rounded, rounded))
            J4.last_track_value_rounded = rounded
    # --------------------------
    # JOINT 5
    tmp_joint_5 = Joint()
    tmp_joint_5.set_rotation(joint_5)
    if tmp_joint_5.x != J5.x or tmp_joint_5.y != J5.y or tmp_joint_5.z !=
J5.z or tmp_joint_5.w != J5.w:
        J5.set_rotation(joint_5)
        # Limits and track range
        (roll, pitch, yaw) = joint_trackval(J5)
        current_track_val = 0
        if(pitch < 0 and roll > 0 and yaw > 0):
            current_track_val = (-(pitch + math.pi / 2))
        elif (pitch > 0 and roll > 0 and yaw > 0):
            current_track_val = (-(math.pi / 2 + pitch))
        elif (pitch < 0 and roll < 0 and yaw < 0):
            current_track_val = math.pi / 2 + pitch
        elif (pitch > 0 and roll < 0 and yaw < 0):
            current_track_val = math.pi / 2 + pitch

        track_val_J5 = current_track_val
        J5.last_track_value = track_val_J5
        rounded = round(J5.last_track_value, 2)
        if (J5.last_track_value_rounded != rounded):
            print("J5 track:%2.10f %2.4f -> %2.4f" % (track_val_J5,
J5.last_track_value_rounded, rounded))
            J5.last_track_value_rounded = rounded
    # --------------------------
    # JOINT 6
    tmp_joint_6 = Joint()
    tmp_joint_6.set_rotation(joint_6)
```

```
    if tmp_joint_6.x != J6.x or tmp_joint_6.y != J6.y or tmp_joint_6.z !=
J6.z or tmp_joint_6.w != J6.w:
        J6.set_rotation(joint_6)
        # Limits and track range
        (roll, pitch, yaw) = joint_trackval(J6)
        current_track_val = 0
        if(pitch > 0 and roll < 0 and yaw < 0):
            current_track_val = (-pitch)
        elif (pitch > 0 and roll > 0 and yaw > 0):
            current_track_val = (-(math.pi / 2 + (math.pi / 2 - pitch)))
        elif (pitch < 0 and roll < 0 and yaw < 0):
            current_track_val = (-pitch)
        elif (pitch < 0 and roll > 0 and yaw > 0):
            current_track_val = (math.pi / 2 + pitch) + math.pi / 2

        track_val_J6 = current_track_val
        J6.last_track_value = track_val_J6
        rounded = round(J6.last_track_value, 2)
        if (J6.last_track_value_rounded != rounded):
            print("J6 track:%2.10f %2.4f -> %2.4f" % (track_val_J6,
J6.last_track_value_rounded, rounded))
            J6.last_track_value_rounded = rounded
    # --------------------------


def listener():
    rospy.init_node('tf_listener', anonymous=True)
    rospy.Subscriber("/tf", TFMessage, callback)
    print ("Listener started...")
    rospy.spin()


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