Primjena ROS sustava u vođenju industrijskog robota

Ištuk, Joško

Master's thesis / Diplomski rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture / Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:235:480095

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-16**

Repository / Repozitorij:

Repository of Faculty of Mechanical Engineering and Naval Architecture University of Zagreb

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Joško Ištuk

Zagreb, 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor: Student:

Doc. dr. sc. Tomislav Staroveški, dipl. ing. Joško Ištuk

Zagreb, 2021.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem mentoru doc. dr. sc. Tomislavu Staroveškom na strpljenju, razumijevanju, moralnoj podršci kao i na pristupačnosti i svom vremenu koje je uložio kako bi mi pomogao.

Zahvaljujem prof. dr. sc. Tomi Udiljaku i asistentici Dori Bagarić mag. ing. mech. na velikoj pomoći, potpori i savjetima koji su mi puno pomogli u izradi diplomskog rada.

Veliko hvala mojim roditeljima, ocu Ivici i majci Mirjam, što su vjerovali u mene i u svemu me podržavali tokom mojih godina studija. Hvala i mojim sestrama koje su mi bile podrška kada sam to trebao.

Zahvaljujem prijateljima Tomislavu Zeleniki i Antunu Jakobu Mariću. Prijateljstvo koje smo izgradili jedna je od najljepših stvari koje mi je donio studij.

Veliko hvala mojoj djevojci Ani koja je vjerovala u mene i svojom ljubavlju i potporom doprinijela ovom uspjehu.

Joško Ištuk

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske radove studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment,
inženjerstvo materijala te mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum:	Prilog:
Klasa: 602-04/21-6/1	
Ur. broj: 15-1703-21	

DIPLOMSKI ZADATAK

Student: **JOŠKO IŠTUK** Mat. br.: 0035195298

Naslov rada na hrvatskom jeziku: **Primjena ROS sustava u vođenju industrijskog robota**

Naslov rada na engleskom jeziku: **ROS based control of industrial robot arm**

Opis zadatka:

Industrijski roboti sve se češće primjenjuju u obradi odvajanjem čestica. Najšira primjena je na obratcima složenije geometrije, gdje nisu tražene visoke točnosti i velike sile obrade. Suvremeni trendovi u razvoju ovakvih obradnih sustava usmjereni su povećanju njihove autonomnosti, gdje se integracijom različitih senzora nastoji ostvariti adaptivna regulacija obradnog procesa, s obzirom na karakteristike površine obratka ili alata. Razvoj takvih sustava najčešće je zasnovan na javno dostupnim platformama otvorenog koda (eng. open source), od kojih se najčešće primjenjuje ROS (eng. Robot Operating System).

U radu je potrebno:
1. Opisati ROS sustav s osvrtom na njegovu primjenu u obradnim sustavima.
2. Prilagoditi ROS sustav za vođenje industrijskog robota tipa ABB IRB6640.
3. Testirati sustav u laboratorijskim uvjetima.
4. Dati zaključke rada.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan: 4. ožujka 2021.

Rok predaje rada: 6. svibnja 2021.

Predviđeni datum obrane: 10. svibnja do 14. svibnja 2021.

Zadatak zadao: doc. dr. sc. Tomislav Staroveški

Predsjednica Povjerenstva: prof. dr. sc. Biserka Runje

SADRŽAJ

SADRŽAJ I
POPIS SLIKA III
POPIS TABLICA VI
POPIS KRATICA VII
POPIS OZNAKA VIII
SAŽETAK IX
SUMMARY X
1. UVOD 1
2. ROS – ROBOTSKI OPERACIJSKI SUSTAV 5
2.1. Značajke ROS sustava 5
2.1.1. Prednosti i nedostaci ROS sustava 6
2.1.2. Razlika između ROS-a i OS-a 8
2.2. Povijest ROS sustava 9
2.3. Struktura ROS sustava 12
2.3.1. Razina sustava datoteka 12
2.3.1.1. ROS paketi 14
2.3.1.2. Tipovi ROS poruka 15
2.3.1.3. Tipovi ROS servisa 15
2.3.2. Razina povezivanja procesa 16
2.3.3. ROS razina zajednice 19
2.4. Alati ROS sustava 20
2.4.1. RViz 20
2.4.2. MoveIt! 20
2.4.3. Gazebo 21
2.4.4. ROS–Industrial 22

2.4.4.1. Primjena ROS-a u obradnim sustavima ... 23

3. EKSPERIMENTALNI POSTAV ... 27

3.1. Industrijski robot ABB IRB6640 ... 27

3.2. Kontroler IRC5 ... 31

3.3. Programski paket ABB *RobotStudio* ... 31

4. EKSPERIMENTALNI DIO ... 34

4.1. Instalacija ROS sustava ... 34

4.2. Prilagodba ROS sustava za vođenje industrijskog robota ... 36

4.2.1. ABB IRB6640 support paket ... 36

4.2.1.1. URDF ... 37

4.2.1.2. Razvoj 3D modela za vizualizaciju robota ... 41

4.2.1.3. Razvoj 3D modela za provjeru sudara ... 46

4.2.1.4. Razvoj URDF konfiguracijske datoteke robota ... 48

4.2.2. Razvoj MoveIt! konfiguracijskog paketa robota ... 53

4.3. Realizacija vođenja robota ... 55

4.3.1. Vođenje robota izvedeno u simulaciji ... 56

4.3.1.1. Konfiguracija kontrolera IRC5 ... 56

4.3.1.2. Pokretanje simuliranog robota ... 61

4.3.2. Vođenje stvarnog robota ... 68

5. ZAKLJUČAK ... 69

LITERATURA ... 70

PRILOZI ... 72

POPIS SLIKA

Slika 1 Robot Unimate - prvi industrijski robot [2] 1

Slika 2 Roboti PUMA (lijevo) i ABB IRB (desno) [2] 2

Slika 3 SCARA (lijevo) te delta robot (desno) [3, 4] 3

Slika 4 Mreža računala u ROS sustavu [4] 6

Slika 5 PR - *Personal Robot* [6] 10

Slika 6 Plakat verzije sustava *ROS Groovy Galapagos* (lijevo), razvoj ROS sustava tokom prvih godina (desno) [6] 11

Slika 7 Struktura ROS sustava datoteka 13

Slika 8 Klasična struktura ROS paketa 14

Slika 9 Primjer nekih tipova poruka u ROS-u 15

Slika 10 Struktura ROS grafa procesa 16

Slika 11 Vizualizacija radnog prostora robota pomoću *RViz*-a [6] 20

Slika 12 Simulacija izuzimanja predmeta s hrpe u Gazebu [12] 22

Slika 13 Istraživanje primjene industrijskih robota u Sj. Americi (1997. - 2013.) [14] 24

Slika 14 Usporedba primjene robota u obradnim sustavima danas i u budućnosti [14] 25

Slika 15 Primjer primjene *Scan-N-Plan* tehnologije u procesu poliranja 26

Slika 16 Industrijski robot ABB IRB6640 27

Slika 17 Doseg robota ABB IRB6640 [15] 28

Slika 18 Dimenzije potrebne za razvoj konfiguracijske datoteke i 3D modela robota 30

Slika 19 Kontroler robota 31

Slika 20 Sučelje programa ABB *RobotStudio* 32

Slika 21 Izbor robota u programu *ABB RobotStudio* 33

Slika 22 Sadržaj radnog prostora *catkin_ws* 35

Slika 23 Karakteristike elementa <link> URDF datoteke [6] 37

Slika 24 Karakteristike elementa <joint> URDF datoteke [6] 40

Slika 25 Zglob između članaka 3 i 4 robota ABB IRB 6640 42

Slika 26 Specifikacijsko stablo članka 4 43

Slika 27 Usporedba dijelova *Body.11* i *Body.21* 45

Slika 28 Definiranje točke za translaciju 45

Slika 29 Konačna pozicija članka 4 46

Slika 30 Model članka 4 u programskom paketu *MeshLab* prije prilagodbe 47

Slika 31 Model članka 4 u programskom paketu *MeshLab* nakon prilagodbe 48

Slika 32 Kinematski lanac robota [19] 50

Slika 33 Pozicija koordinatnog sustava članka 1 51

Slika 34 Uspješna vizualizacija modela robota u *RViz-u* 52

Slika 35 *MoveIt! Setup Assistant* 53

Slika 36 Optimizacija provjere sudara u programskom paketu *MoveIt!* 54

Slika 37 Procesi instalirani na virtualni kontroler 58

Slika 38 Signali uspješno postavljeni na virtualni kontroler 59

Slika 39 Signali povezani sa specifičnim izlazima robotskog sustava 60

Slika 40 Ponovno postavljanje virtualnog kontrolera 61

Slika 41 Pozadinski procesi ROS_StateServer i ROS_MotionServer prije povezivanja s ROS sustavom 62

Slika 42 Pozadinski procesi ROS_StateServer i ROS_MotionServer nakon povezivanja s ROS sustavom 63

Slika 43 Programsko sučelje *MoveIt!/RViz* povezano s robotom simuliranim u programu *RobotStudio* 64

Slika 44 Trenutna pozicija robota prikazana u simulaciji u programu *RobotStudio* 64

Slika 45 Robot u programu *RViz* nakon izvršenja trajektorije 65

Slika 46 Robot u programu *RobotStudio* nakon izvršenja trajektorije 65

POPIS TABLICA

Tablica 1 Opis vrste i raspon gibanja po zglobovima 28

Tablica 2 Ograničenja brzine zglobova 29

Tablica 3 Procesi koji se postavljaju na kontroler robota [6] 58

Tablica 4 Signali koji se postavljaju na kontroler robota [6] 59

Tablica 5 Signali i pripadajući izlazi sustava [6] 60

Tablica 6 Učitavanje modula u procese [6] 61

POPIS KRATICA

Kratica	**Opis**
2D	dvodimenzionalan
3D	trodimenzionalan
AMCL	*Adaptive Monte Carlo Localization*
CAD	*Computer-aided Design* – Oblikovanje pomoću računala
DOF	*Degree of Freedom* – stupanj slobode gibanja
OS	*Operating System* – Operacijski sustav
OSRF	*Open Source Robotics Foundation*
PLC	*Programmable Logic Controller* – Programabilni logički sklop
PUMA	*Programmable Universal Machine for Assembly*
ROS	*Robot Operating System* – Robotski operacijksi sustav
ROS-I	*ROS-Industrial*
SLAM	*Simultaneous Localization and Mapping*
SRDF	*Semantic Robot Description Format*
URDF	*Unified Robot Description Format*

POPIS OZNAKA

Oznaka	Jedinica	Opis
p	rad	Kut zakreta oko osi y
r	rad	Kut zakreta oko osi x
x	m	Pomak po osi x
y	m	Pomak po osi y
y	rad	Kut zakreta oko osi z
z	m	Pomak po osi z

SAŽETAK

U ovom diplomskom radu dan je opis procesa prilagodbe ROS sustava za vođenje industrijskog robota tipa ABB IRB6640. Rad je podijeljen na pet dijelova. U uvodu se govori o razvoju industrijske robotike te o potencijalu ROS sustava za primjenu u industriji. Potom se daje opis glavnih koncepata i značajki ROS sustava korištenih za realizaciju zadatka uz osvrt na primjenu ROS sustava u obradnim sustavima. ABB robot, IRC5 kontroler te programski paket *RobotStudio* korišten za simulaciju vođenja robota kratko su opisani u trećem dijelu. U četvrtom, eksperimentalnom dijelu rada prikazan je razvoj i prilagodba ROS sustava od postavljanja sve do uspješne realizacije vođenja robota u laboratoriju. U posljednjem, petom dijelu donose se zaključci rada.

Ključne riječi: ROS, ROS-Industrial, industrijski robot, vođenje robota, platforma otvorenog koda

SUMMARY

This thesis describes the process of adaptation of ROS system for control of the ABB IRB6640 industrial robot arm. The thesis is divided into five sections. Development of industrial robotics along with potential industrial applications of ROS system are discussed in the introductory section of the thesis. The main concepts and features of ROS system used in the thesis are described and the review of the ROS based machining systems is presented in the second section. The ABB robotic arm, the IRC5 controller and *RobotStudio* software are briefly described in the third part of the thesis. The fourth part, that is the experimental part of the thesis presents the development and the adaptation of ROS beginning with the installation of the system leading to the successful control of the industrial robot arm. Concluding remarks are given in the final chapter of the thesis.

Key words: ROS, ROS-Industrial, ABB industrial robot arm, robot control, open-source

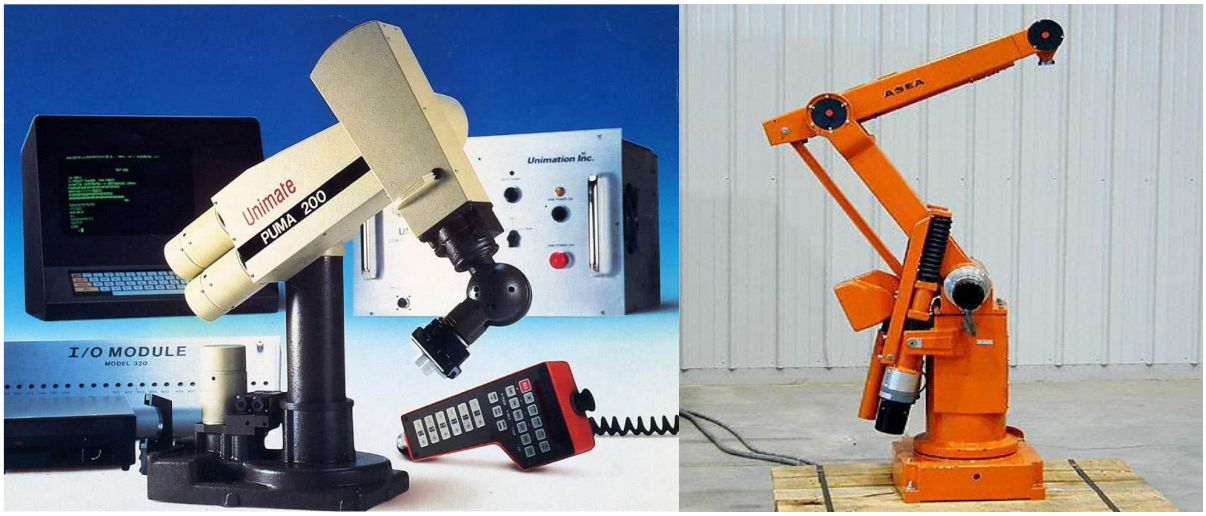
1. UVOD

Industrijska robotika grana je industrije koja se bavi proučavanjem, razvojem i primjenom robotskih sustava u proizvodnji, a posljednjih desetak godina i prilagodbom robotskih sustava za rad na proizvodnoj liniji u direktnom kontaktu i suradnji s čovjekom [1]. Ova grana industrije, kao i industrijski roboti koji su njen osnovni dio, od svojeg nastanka krajem 50-ih godina 20. stoljeća imali su jako veliki utjecaj na razvoj procesa proizvodnje. Izum i razvoj industrijskog robota revolucionarizirao je kompletnu industriju. U nekoliko desetljeća od izuma industrijski roboti prošli su nekoliko razvojnih faza [2].

Početkom industrijske robotike smatra se 1954. godina kada je George Devol zatražio patent za svoj izum koji je bio osnova za razvoj prvog industrijskog robota, robota Unimate (Slika 1). Robot Unimate razvijen je 1961. godine i iste te godine postavljen u tvornici General Motors u Trentonu u SAD-u. Korišten je za vađenje i slaganje gotovih proizvoda iz kalupa nakon postupka tlačnog lijevanja. U nekoliko godina koje su slijedile roboti su se počeli koristiti i u drugim tvornicama za obavljanje jednostavnih, repetitivnih poslova (npr. točkasto zavarivanje šasija automobila). Industrijski roboti prve generacije bili su najčešće hidraulički pogonjeni. Nisu imali nikakav senzorski sustav, a upravljanje se svodilo na izvršavanje naredbi prethodno pohranjenih na magnetnoj traci. Programski kod pohranjuje se u memoriju robota te se naredbe izvršavaju jedna po jedna. Radna okolina robota ove generacije mora biti visoko organizirana kako bi mogli izvršavati zadatke [2, 3].

Slika 1 Robot Unimate - prvi industrijski robot [2]

Pojavom i razvojem najprije PLC-a (eng. *Programmable Logic Controller*) krajem 1960-ih godina, a zatim i mikroprocesora početkom 1970-ih započinje druga faza razvoja industrijskih robota i pojavljuje se druga generacija industrijskih robota. Razvoj ove tehnologije omogućio je veću procesorsku moć što je onda omogućilo akviziciju i obradu podataka iz okoline. Roboti dobivaju prve senzorske sustave preko kojih dobivaju informacije iz okoline, a pomoću jednostavne logike računala ovi roboti imaju mogućnost reagiranja. Na ovaj način dolazi do razvoja regulacije robota s povratnom vezom, a to znači da se roboti mogu eksploatirati u radnoj okolini manjeg stupnja organiziranosti. Isto tako, razvojem i napretkom električnih motora oni se počinju koristiti za pogon robota što je u kombinaciji s mikroprocesorima olakšalo pokretanje i regulaciju robota, a isto tako dovelo i do smanjenja cijene proizvodnje robota. Druga generacija robota ima mogućnost donošenja jednostavnih logičkih odluka, da ili ne. Ovoj generaciji robota pripadaju roboti kao što su PUMA (eng. *Programmable Universal Machine for Assembly*) i prvi ABB-ov robot IRB serije, a o jednom takvom robotu bit će više govora dalje u ovom radu. Roboti su prikazani na slici 2 [2, 3].



Slika 2 Roboti PUMA (lijevo) i ABB IRB (desno) [2]

Treća generacija robota pojavljuje se početkom 1980-ih godina. U ovom razdoblju drastično su porasla ulaganja u robotiku kao i profit od prodaje robota, čak 80% u odnosu na prethodno razdoblje [3]. Roboti se osim u automobilskoj industriji počinju koristiti u nizu drugih industrija na poslovima bojanja, lemljenja, zavarivanja, ali i rukovanja te montaže proizvoda. Na razvoj ove generacije robota, osim većeg ulaganja kapitala najviše su utjecali

veliki napredak računala te pojava interneta. Programiranje robota postaje nešto jednostavnije što znači da im se teoretski tijekom radnog vijeka može mijenjati namjena. Mogu se programirati *on-line*, pomoću privjeska za programiranje robota ili *off-line*, povezivanjem robota i računala, što je omogućilo korištenje naprednijih programskih jezika kao i povezivanje sa CAD (eng. *Computer-Aided Design*) alatima. Upotreba modernijih i procesorski snažnijih računala i mikroprocesora omogućila je lakšu i bržu obradu veće količine podataka sa senzora pa roboti ove generacije imaju mogućnost adaptacije programiranog kretanja s obzirom na nesređeni radni prostor bez programske upute. Dakle, roboti treće generacije osim pamćenja i donošenja jednostavnih logičkih odluka imaju i sposobnost učenja. Razvijaju se dvije nove konfiguracije robota, a to su SCARA robot (eng. *Selective Compliance Assembly Robot Arm*) vidljiv na slici 3 lijevo te delta robot na slici 3 desno [3, 4].

Slika 3 SCARA (lijevo) te delta robot (desno) [3, 4]

Četvrta generacija industrijskih robota predstavlja moderne robote razvijene u 21. stoljeću. Na njihov razvoj, između ostalog, veliki utjecaj ima daljnji razvoj *modernog* računala kao i razvoj umjetne inteligencije. No zanimljivo, na polju industrijske robotike razvoj je značajno usporen u odnosu na prijašnja desetljeća. Osim pojave kolaborativnih robota nema značajnog razvoja, a i integracija kolaborativnih robota u proces proizvodnje još uvijek nije istinski zaživjela. Jedan od razloga ovakvog trenda je zatvorenost proizvođača industrijskih robota. Gotovo svi veći proizvođači vode politiku zatvorenosti kako bi zaštitili svoju tehnologiju i na taj način sačuvali svoje mjesto na tržištu. Proizvođači često za programiranje svojih robota imaju vlastite programske jezike što onemogućuje primjenu programskog koda između dva različita robota. Dakle, potrebno je kao prvo dobro poznavati jezik za pojedinog robota, a onda i ispisati veliku količinu monolitnog koda ukoliko se radi o složenom robotskom sustavu. Nadalje, komunikacijski protokoli razlikuju se od proizvođača do proizvođača, što znači da je suradnja dva robota različitih proizvođača maksimalno otežana. Ovakva zatvorenost tržišta otežava razvoj inovacija što na kraju dovodi do spomenutog trenda [3].

Istovremeno postoji opcija koja nudi potencijalno rješenje ovih problema. Robotski operacijski sustav (eng. *Robot Operating System*) ili skraćeno ROS je alat kojim se želi dati veća sloboda razvojnim inženjerima robotskog softvera, ali i hardvera, da se posvete inovaciji i dodavanju vrijednosti robotskim sustavima, a ne razvoju tih komponenti nanovo za svaki novi sustav ili namjenu. Cilj je dakle, kako slikovito kažu tvorci ROS-a: ″Izbjeći ponovno izmišljanje kotača″ (eng. *″Reinventing the Wheel″*) [4].

2. ROS – ROBOTSKI OPERACIJSKI SUSTAV

Razvoj softvera za robote nije nimalo jednostavan zadatak, naročito u novije vrijeme kada se broj različitih komponenti robotskih sustava, velikog broja različitih proizvođača, sve više povećava. Hardver robota različit je od proizvođača do proizvođača što onemogućava ponovno korištenje istog softvera na dva različita robota, a ako je ponovna upotreba softvera i moguća, to često zahtjeva ulaganje velike količine vremena u prilagodbu programskog koda i integraciju. Drugim riječima, jedan od glavnih problema je manjak standardizacije među komponentama (dijelova robota, senzora i aktuatora) različitih proizvođača. Nadalje, sama količina programskog koda koja je potrebna da bi se ostvarile i najosnovnije funkcionalnosti komponenti je vrlo velika, što čini razvoj softvera kompliciranim. Ako je uz to kod razvijan u jednom dijelu (monolitno) za sve dijelove sustava (driveri, senzori, upravljanje i dr.), traženje i ispravljanje grešaka u kodu (eng. *debugging*) postaje vrlo zahtjevno, a ako se greška pojavi u samo jednom dijelu cijeli sustav neće raditi. Na kraju, u robotici postoje algoritmi koji se često koriste za različite funkcionalnosti sustava poput na primjer lokalizacije u prostoru, vizualizacije ili upravljanja. Bez dijeljenja koda razvojni inženjeri softvera za svaku novu namjenu morali bi razvijati algoritme ispočetka, što predstavlja gubitak vremena, a na kraju i novca [4, 5].

Kao odgovor na ove i druge probleme i poteškoće u razvoju robotskih sustava i kao potencijalno rješenje, 2007. godine počinje razvoj robotskog operacijskog sustava.

2.1. Značajke ROS sustava

Robotski operacijski sustav (ROS) je besplatan i svima dostupan, prilagodljivi okvir za razvoj softvera za robotske sustave. To je zapravo svojevrsna zbirka isprobanih i dokazano pouzdanih alata, biblioteka i konvencija koja za cilj ima što više pojednostaviti razvoj složenih robotskih sustava kako bi ujedno bili i robusni. Glavna ideja pri razvoju ROS sustava je bila omogućiti razvoj softvera koji bi se mogao implementirati u velik broj različitih robotskih sustava uz što manje truda odnosno promjena u programskom kodu. Primjerice jedan laboratorij možda ima stručnjake na području mapiranja prostora i može razviti vrhunski softver za razvoj mapa. Druga pak tvrtka zapošljava vrhunske stručnjake u razvoju sustava autonomne navigacije

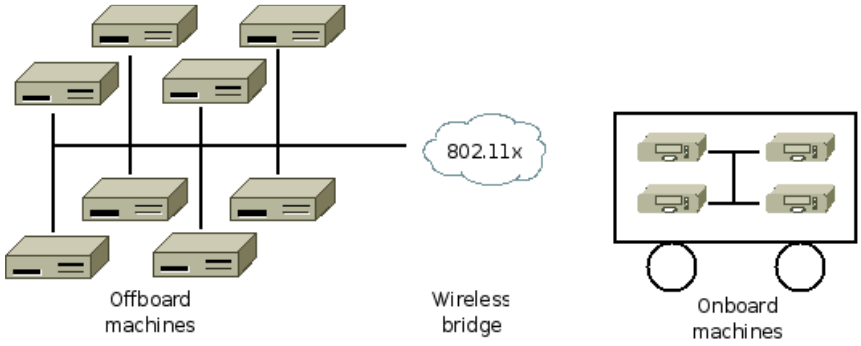
kroz mapirani prostor, a treći laboratorij razvio je vrhunski vizijski sustav. ROS sustav razvijen je upravo kako bi omogućio suradnju između ovakvih skupina, čime bi se ubrzao razvoj inovacija. ROS sustav je isto tako knjižnica provjerenih hardverskih sučelja i upravljačkih programa (eng. *drivera*) za razne komponente robotskih sustava velikog broja različitih proizvođača. Upravljanje robotom pomoću ROS sustava funkcionira na mehanizmu izmjene poruka između procesa, odnosno ROS čvorova. Neki čvorovi objavljuju poruke na neku temu dok se drugi na iste pretplaćuju i tako izmjenjuju informacije [6].

2.1.1. Prednosti i nedostaci ROS sustava

Nekoliko je značajki ROS sustava kojima se ističe i koje ga čine dobrim izborom pri razvoju robotskih sustava, a prema [4, 7] to su:

- **komunikacija između više računala**

  ROS sustav može biti sačinjen od velikog broja procesa (čvorova) na više različitih računala koji su međusobno povezani kako pokazuje slika 4. Među povezanim računalima nema nadređenog računala, već su sva računala povezana sa središnjim sustavom (*roscore masterom)* o čemu će više riječi biti u ovome poglavlju. Komunikacija između procesa ili čvorova jedna je od osnovnih i najkorisnijih značajki ROS sustava.



Slika 4 Mreža računala u ROS sustavu [4]

- **jezična neutralnost programskog koda**

  Kod pisanja programskog koda većina ljudi koja se bavi razvojem ima neki programski jezik koji preferiraju više od ostalih. Iz tog razloga ROS sustav osmišljen je kao jezično neutralan, odnosno komunikacija je moguća između čvorova pisanih u različitim jezicima (C++, Python, Java i dr.). Većina paketa ROS sustava pisana je ipak u jezicima C++ i Python.

- **veliki broj dostupnih alata**

  ROS sustav pruža veliki broj dostupnih alata za traženje i ispravljanje grešaka u kodu (eng. *debugging*), snimanje i reprodukciju podataka, grafički prikaz podataka, generiranje dokumentacije, simulaciju, vizualizaciju i mnoge druge.

- **programska podrška za veliki broj robota, senzora i aktuatora**

  U ROS sustavu dostupni su mnogi upravljački programi i paketi za integraciju hardvera velikog broja različitih proizvođača u ROS sustav. Primjerice robote proizvođača ABB, KUKA, Motoman, Fanuc, Universal Robots i mnogi drugi, ali i visoko kvalitetne senzore i aktuatore.

- **modularnost programskog koda**

  Jedan od problema kod pisanja monolitnog koda, kakav je uobičajen za programiranje robota izvan ROS sustava, je taj da ako se u jednom dijelu koda pojavi greška moguće je da će se cijeli program zaustaviti iako možda greška koja se pojavila ne utječe direktno na izvršavanje nekog drugog dijela tog koda. U ROS sustavu to nije tako. Svaki čvor pisan je kao zaseban program i ako se koji od njih prestane raditi to neće ugroziti cijeli sustav. Ako se takvo što i dogodi ROS nudi odlične alate za ispravljanje grešaka u kodu.

- **besplatan je i otvorenog koda (eng. *open-source*)**

  Kompletan programski kod ROS sustava je javno dostupan. Zbog ove činjenice ROS sustav se razvija vrlo brzo (eventualne greške u kodu brzo se pronalaze i rješavaju) te se vrlo brzo širi i sve više popularizira. Svima je dozvoljena upotreba ROS sustava i svih njegovih komponenata u istraživačke, ali i u komercijalne svrhe.

- **aktivna zajednica**

  Jedan od najkorisnijih aspekata ROS sustava je svakako živa i vrlo aktivna zajednica korisnika širom svijeta koja raste iz godine u godinu. Može se reći i da je smisao ROS sustava njegova zajednica jer su upravo korisnici zaslužni za velik postotak sadržaja dostupnih putem ROS sustava. Vrijedan izvor informacija pri razvoju zadatkom zadanog sustava pronađen je u izvoru [8], što je internetska stranica s pitanjima i odgovorima vezanim uz ROS sustav pokretana od zajednice, a održavana od strane

odgovornih moderatora koji se brinu da je sadržaj na njoj vjerodostojan. Stranica broji više od 13 000 pitanja sa više od 70% odgovorenih, odnosno zatvorenih.

Naravno, ROS sustav ima i nekih nedostataka od kojih su najčešće spominjani sljedeći [5]:

- **složenost sustava**

  Potreban je relativno dug vremenski period kako bi se ovladalo svim aspektima ROS sustava. Neke upute za učenje implementacije nisu prilagođene početnicima i općenito mrežno dostupni materijali na početku početnicima mogu biti kompliciranі.

- **nedostatak sigurnosnih značajki**

  ROS sustav razvijen je da bude otvorenog koda te je prvotno razvijen za upotrebu u istraživačke svrhe. Iz ovih razloga sustavu nisu dodane sigurnosne značajke niti alati. S obzirom da ROS sustav izmjenjuje podatke putem internetske mreže podaci su izloženi hakerskim napadima i nisu sigurni. Mnogi su ROS paketi dijelom nekih komercijalnih sustava, ali rijetko koji pokreće isključivo ROS [9].

- **nedostatak podrške za izvršavanje u stvarnom vremenu (eng. *real-time*)**

  ROS sustav izgrađen je na Linux jezgri koja bez određenih ekstenzija ne podržava izvršavanje u stvarnom vremenu. ROS iz tog razloga, a i zbog činjenice da prvotno nije razvijen za sustave koji to moraju podržavati, nema mogućnost izvršavanja u stvarnom vremenu.

2.1.2. Razlika između ROS-a i OS-a

Operacijski sustav (eng. *Operating System*) je softver koji omogućuje vezu između hardvera i programa, a na taj način i korisnika koji će upravljati tim hardverom. Zadužen je za alokaciju hardverskih resursa poput memorije i procesora procesima kojima su potrebni. Može sadržavati grafičko sučelje – GUI (eng. Graphical User Interface), ali i ne mora. Neki od poznatih operacijskih sustava su Linux, Windows, macOS, Android i dr. [7].

Iako nosi naziv robotski operacijski sustav, ROS sustav nije zapravo operacijski sustav. On se još naziva i meta-operacijskim sustavom jer pruža neke funkcionalnosti operacijskog sustava kao što su apstrakcija hardvera, komunikaciju među procesima te upravljanje paketima, no ipak

ne pruža sve funkcionalnost OS-a. ROS sustav ne može upravljati hardverom sustava na niskoj razini kao što su memorija i procesor [4].

2.2. Povijest ROS sustava

ROS sustav počeo se razvijati kao projekt dvojice doktoranda na sveučilištu *Standford*, Keenana Wyrobeka i Erica Bergera, 2006. godine. Razlog je bio pokušaj rješavanja dva već spomenuta problema koja se pojavljuju pri razvoju sustava u robotici:

1. previše izgubljenog vremena na razvoj sustava koji su već razvijeni
2. premalo vremena posvećenog inovaciji.

Kao pokušaj rješavanja tih problema ovaj dvojac na *Standfordu* pokreće *Standford Personal Robotics Program* koji za cilj ima razvoj softverskog okvira (eng. *framework*) koji bi omogućio komunikaciju između procesa te alata za razvoj programskog koda. Uz to željeli su razviti i robota na kojem bi se sustav mogao testirati, mobilnog robota *Personal Robot* ili skraćeno *PR* prikazan na slici 5. Cilj je bio napraviti 10 takvih robota te ih donirati sveučilištima diljem SAD-a gdje bi softver za robote bio razvijan na njihovom sustavu. Za razvoj od sveučilišta *Standford* dobivaju 50 000$ što koriste kako bi napravili robota, no dobivena sredstva nisu dovoljna za razvoj 10 primjeraka. Kao investitor javlja se tvrtka koja se bavi robotikom, *Willow Garage*, pod čijim pokroviteljstvom kreće pravi razvoj ROS sustava. Scott Hassan, osnivač spomenute tvrtke, prepoznaje projekt ROS sustava kao vrlo važan i uskoro tvrtka odbacuje sve ostale projekte i posvećuje se samo razvoju ROS sustava [10].

Slika 5 PR - *Personal Robot* [6]

Razvoj ROS-a u sklopu tvrtke *Willow Garage*

Tvrtka *Willow Garage* bila je zadužena za razvoj i održavanje ROS sustava nešto više od 6 godina (2007. – 2013.) sve do zatvaranja tvrtke. U tom razdoblju ostvaren je velik napredak u razvoju sustava kao i velika popularizacija sustava u zajednici ljudi koji se bave robotikom što dobro pokazuje slika 6(desno). U ovom razdoblju predstavljena je prva verzija ROS sustava *ROS Mango Tango* još nazvana i *ROS 0.4*. Ova verzija nije bila javno dostupna jer je bila zapravo ispitna verzija sustava. Prva javno dostupna verzija sustava bila je *ROS 1.0*. U godinama koje su uslijedile izašlo je još 6 verzija ROS sustava:*Box Turtle* – 2010.

- *ROS C-Turtle* – 2010.
- *Diamond Back* – 2011.
- *ROS Electric Emys* – 2011.
- *ROS Fuerte Turtle* – 2012.
- *ROS Groovy Galapagos* – 2012.

Slika 6 Plakat verzije sustava *ROS Groovy Galapagos* (lijevo), razvoj ROS sustava tokom prvih godina (desno) [6]

Zanimljivo je da od verzije ROS sustava *Box Turtle* pa sve do danas svaka nova verzija ROS sustava naziva se po nekoj vrsti kornjače te dolazi sa specifičnim plakatom koji prikazuje upravo kornjaču. Plakat za verziju ROS *Groovy Galapagos* prikazan je na slici 6 lijevo. Kornjača je tako postala zaštitnim znakom ROS sustava. Osim nekoliko novih verzija ROS sustava iz ovog razdoblja razvoja za popularizaciju važni su razvoj robota *PR2* 2009. godine kao i robota *Turtlebot* 2011. godine. 2010. godine 11 robota PR2 poslano je fakultetima diljem SAD-a čime je ostvarena prvotna zamisao tvoraca sustava. Ovi roboti bili su isto tako i komercijalno dostupni [10].

Razvoj ROS-a u sklopu udruge *Open Source Robotics Foundation*

2013. godine gašenjem tvrtke *Willow Garage* zadaću razvoja i održavanje ROS sustava preuzima novoosnovana udruga *Open Source Robotics Foundation* (OSRF). ROS sustav nastavio se razvijati jednako dobro i pod novom upravom. Razvijaju se nove verzije ROS sustava [10]:

- *ROS Hydro Medusa* – 2013.
- *ROS Indigo Igloo* – 2014.
- *ROS Jade Turtle* – 2015.
- *ROS Kinetic Kame* – 2016.

- *ROS Lunar Loggerhead* – 2017.
- *ROS Melodic Morenia* – 2018.
- *ROS Noetic Ninjemys* – 2020.

Važno je napomenuti kako je posljednja verzija ROS sustava - *ROS Noetic*, ujedno i zadnja verzija ROS-a koja će biti izdana. Naime, 2018. godine izdana je prva verzija ROS2 sustava. Ovaj sustav nastao je kako bi uklonio neke od nedostataka koji su bili navedeni ranije u radu. ROS2 ima mogućnost izvršavanja u stvarnom vremenu te razvijene sigurnosne značajke što ga čini više pogodnim za komercijalnu primjenu. Uz ove donosi i mnoge druge preinake poput kompatibilnosti s modernijom verzijom programskog jezika Python – Python 3, no u ovom radu neće biti više govora o tome. Osim razvoja sustava ROS2 vrijedno je spomenuti i razvoj nadogradnje ROS sustava za primjenu u industriji, ROS-Industrial, odnosno osnutak konzorcija 2013. godine. Više o ROS-Industrialu (ili kraće ROS-I) bit će više riječi nešto kasnije u ovom poglavlju.

2.3. Struktura ROS sustava

ROS je strukturno podijeljen na tri cjeline, odnosno tri razine [11]:

- Razina sustava datoteka
- Razina povezivanja procesa
- Razina zajednice

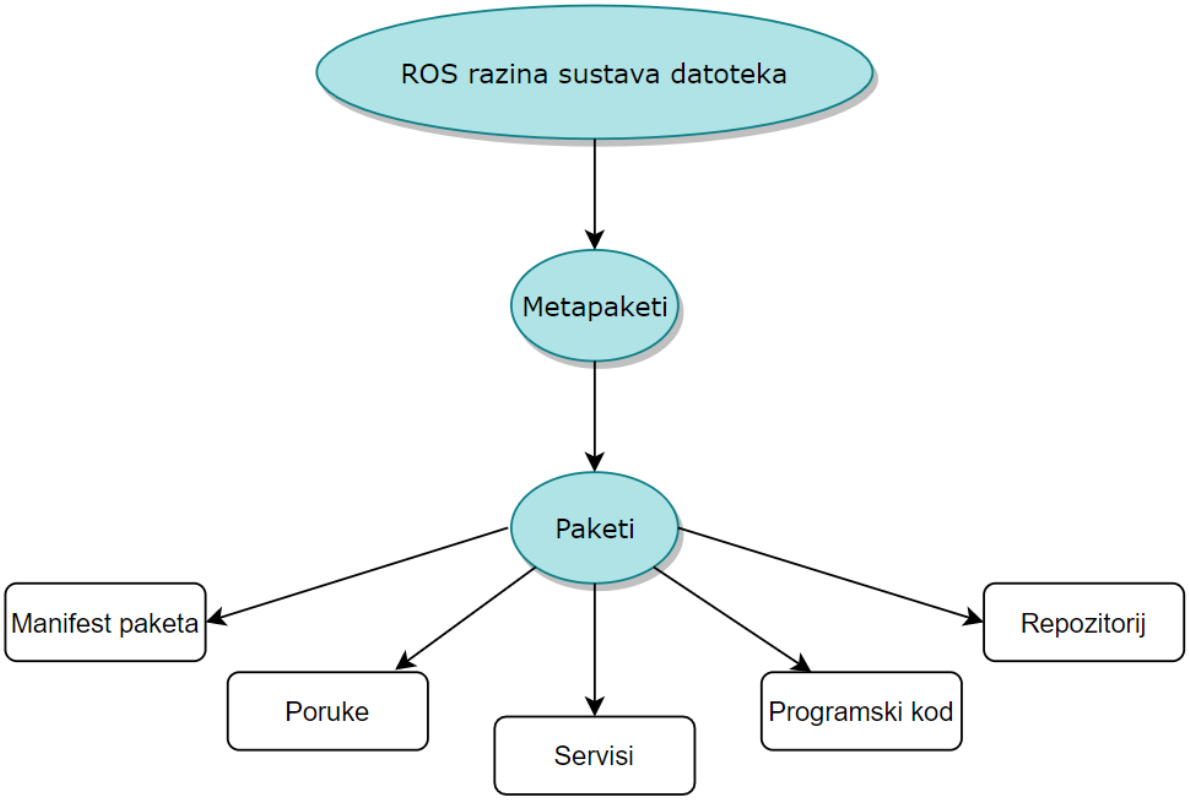
Prva razina predstavlja razinu sustava datoteka. Na ovoj razini definiraju se struktura direktorija i datoteke potrebne za funkcioniranje sustava.

Druga razina je razina povezivanja procesa na kojoj se odvija komunikacija između procesa, odnosno čvorova i povezanih sustava.

Treća razina je razina zajednice. Ova razina presudna je za funkcioniranje ROS-a jer omogućuje njegovo brzo širenje i razvoj. ROS zapravo i postoji zahvaljujući zajednici [11].

2.3.1. Razina sustava datoteka

ROS sustav datoteka (eng. *Filesystem Level*) definira strukturu podataka koji su pohranjeni na disku. Slično kao i OS, ROS je podijeljen na direktorije, a direktoriji sadrže datoteke koje opisuju funkcionalnost direktorija u kojima se nalaze.



Slika 7 Struktura ROS sustava datoteka

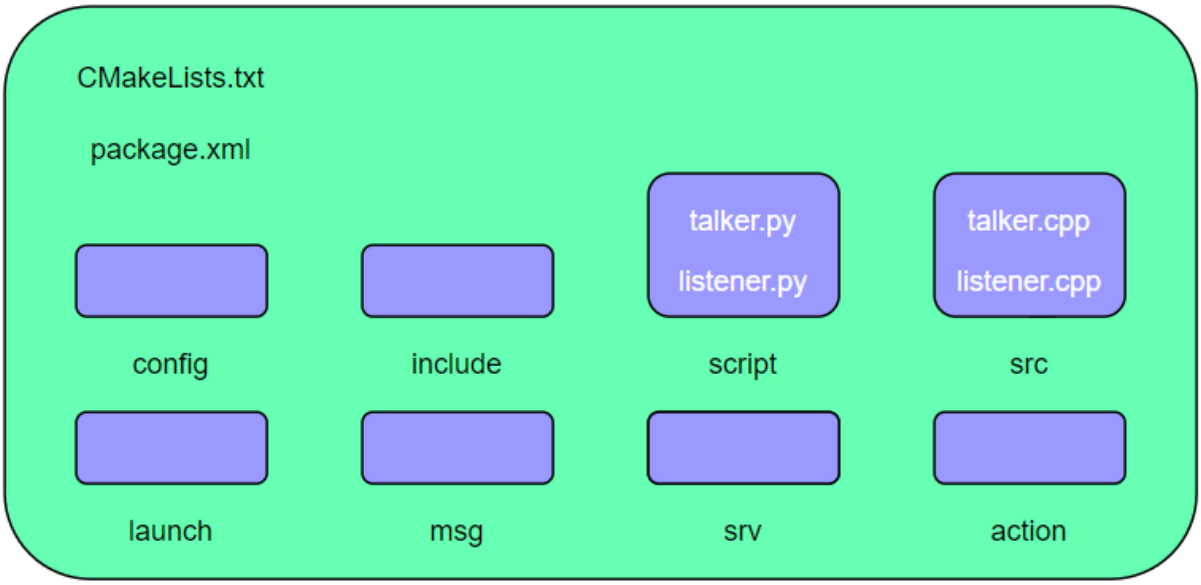
Na slici 7prikazan je dijagram strukture ROS razine sustava datoteka, a čine ju [6, 12]:

- **Paketi** (eng. *Packages*) koji su osnovna jedinica organizacije softvera u ROS sustavu. Mogu sadržavati datoteke za pokretanje ROS čvorova, biblioteke, konfiguracijske datoteke, skripte i druge vrste datoteka.
- **Metapaketi** (eng. *Metapackages*) koji su specijalna vrsta paketa koja samo predstavljaju skupinu više povezanih paketa. Ustvari metapaketi su virtualni paketi koji ne sadrže programski kod niti datoteke koje sadrže obični paketi.
- **Manifest paketa** (eng. *Package Manifest*) je vrsta konfiguracijske datoteke (najčešće naziva *package.xml*) koja sadrži informacije o paketu kao što su ime paketa, verzija, opis, informacije o licenci, funkcijske zavisnosti te informacije o autoru.
- **Tipovi poruka** (eng. *Message (msg) types*) su opisi poruka koji definiraju strukturu podataka poruka koje se šalju u ROS-u.
- **Tipovi servisa** (eng. *Service (srv) types*) su opisi servisa koji definiraju strukturu zahtjeva i odgovora za servise u ROS-u.

- **Repozitoriji** (eng. *Repositories*) predstavljaju skupine paketa koji dijele isti sustav održavanja (*Versioning Control System* - VCS) poput primjerica Git Hub-a. Paketi koji dijele zajednički VCS nazivaju se repozitorijem.

2.3.1.1. ROS paketi

Paketi su osnovna jedinica organizacije softvera u ROS sustavu. Stvaraju se pomoću alata *catkin_create_pkg*. Oni su najniža razina podataka koja se može izgraditi (eng. *build*) u ROS sustavu. Prema dogovoru svi paketi imaju istu strukturu, čiji je primjer grafički prikazan na slici 8.



Slika 8 Klasična struktura ROS paketa

Namjena pojedinih direktorija i datoteka unutar ROS paketa je kako slijedi [5]:

- *config* direktorij sadrži sve konfiguracijske datoteke koje se koriste u ROS paketu. Ovaj direktorij stvara korisnik, a dogovorno se ovaj direktorij uvijek naziva *config* kako bi se dijeljenje, odnosno korištenje dijeljenih paketa što više olakšalo.
- *include/package_name* direktorij sadrži biblioteke koje se koriste unutar paketa.
- *script* direktorij sadrži Python skripte. Na Slika 8 prikazane su dvije skripte *talker.py* i *listener.py*, koje se u ovom koriste kako bi se pokazala mogućnost izmjena poruka u ROS-u.
- *src* direktorij sadrži skripte pisane u programskom jeziku C++.

- *launch* direktorij sadrži datoteke za pokretanje jednog ili više ROS čvorova.
- *msg* direktorij sadrži definicije tipova poruka.
- *srv* direktorij sadrži definicija tipova servisa.
- *action* direktorij sadrži datoteke koje definiraju djelovanje paketa.
- *package.xml* je manifest paketa.
- *CMakeLists.txt* je tekstualna datoteka koja sadrži upute za izgradnju paketa. Sadrži informacije o tome kako i gdje treba prevesti (kompajlirati) te instalirati program.

2.3.1.2. Tipovi ROS poruka

ROS koristi biblioteke za opis poruka kako bi pridružio vrijednosti podacima (porukama) koje ROS čvorovi izmjenjuju. Na ovaj način omogućuje se korisnicima da programski kod pišu u bilo kojem od podržanih programskih jezika, a da čvorovi i dalje mogu međusobno komunicirati. Isto tako na ovaj način omogućuje se automatsko generiranje koda korištenjem ROS alata. Podatci o tipovima poruka pohranjeni su u direktoriju *msg* određenog paketa. Iako ROS nudi velik broj različitih predefiniranih tipova poruka korisnici mogu definirati vlastite tipove poruka koje će izmjenjivati čvorovi [6].

Svaka poruka definirana je u dva dijela: polja i konstante. Polje definira tip podatka koji pojedina poruka šalje. Različiti tipovi podataka mogu biti: bool, string, int64, float32, ali i neki drugi tip poruke definiran od strane programera. Konstante daju imena za određena polja slično je slično imenima za varijable u programskim jezicima. Primjer tipa poruka prikazuje slika 9.

```
ijosko@ji-dstretch-ros:~/catkin_ws$ rosmsg show moveit_msgs/JointLimits
string joint_name
bool has_position_limits
float64 min_position
float64 max_position
bool has_velocity_limits
float64 max_velocity
bool has_acceleration_limits
float64 max_acceleration
```

Slika 9 Primjer nekih tipova poruka u ROS-u

2.3.1.3. Tipovi ROS servisa

ROS servisi također su poput ROS poruka definirani bibliotekom kako bi se olakšala programska višejezičnost u ROS-u. ROS servisi isto kao ROS poruke služe za komunikaciju

između čvorova, ali na nešto drugačiji način. ROS servisi zapravo koriste poruke kako bi ostvarili svojevrsnu komunikaciju na zahtjev (eng. *request/response*). Jedan čvor šalje poruku zahtjeva drugom čvoru, a zatim čeka odgovor. Drugi čvor zatim obrađuje primljene podatke, obrađuje ih i vraća prvom čvoru koji ih prima i komunikacija je završena. Za razliku od komunikacije porukama koja se odvija kontinuirano u nizu informacija ovdje se razmjena podataka odvija samo jednom za jedan zahtjev [5].

Primjer tipa servisa može izgledati ovako:

int64 a

int64 b

---

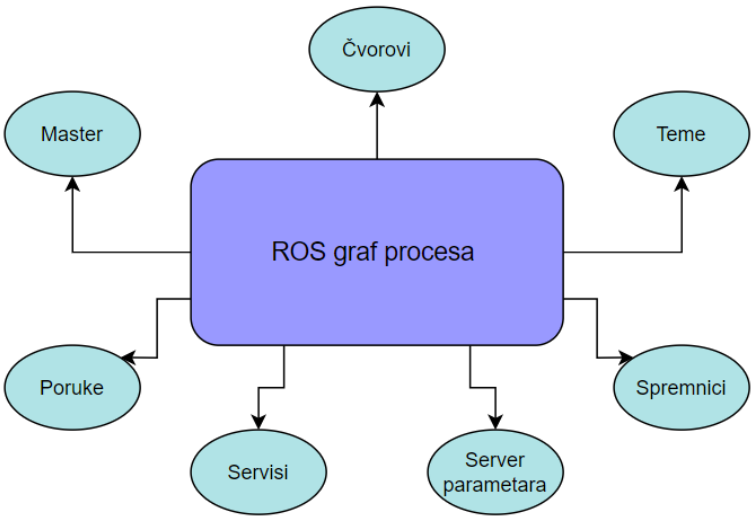
int64 sum,

gdje prvi čvor šalje drugom čvoru dva cijela broja, a i b, koje drugi čvor zbraja i šalje rezultat zbrajanja nazad prvom čvoru.

2.3.2. Razina povezivanja procesa

Razina ROS grafa procesa opisuje partnersku mrežu (eng. *peer-to-peer network*) ROS čvorova koji zajedno obrađuju podatke. Bilo koji čvor može pristupiti mreži i ostvariti interakciju s ostalim čvorovima, vidjeti informacije koje se izmjenjuju, ali i slati informacije [11].

Osnovni pojmovi i koncepti ove razine ROS sustava prikazani na slici 10:



Slika 10 Struktura ROS grafa procesa

- **Čvorovi** (eng. *Nodes*) u ROS-u su procesi koji obrađuju informacije. ROS je osmišljen da bude modularan na vrlo niskoj razini, stoga sustav za vođenje robota obično sačinjava velik broj čvorova. Primjerice, prvi čvor upravlja laserskim skenerom prostora, drugi čvor upravlja pogonom robota, treći čvor upravlja senzorom vibracija, a n-ti čvor generira trajektoriju. Ovakva podjela funkcionalnosti puno je bolje rješenje od jednog programa koji upravlja svim funkcijama sustava jer ako se iz nekog razloga neki čvor isključi to neće nužno kao posljedicu imati zaustavljanje cijelog sustava. Funkcija koju je obavljao ugašeni čvor neće biti izvršena i vrlo lako će se ustanoviti u kojem dijelu sustava je došlo do greške. su zbog svoje jednostavne strukture ROS čvorovi vrlo pogodni za pronalazak i otklanjanje grešaka u kodu. Pri stvaranju ROS čvorova najčešće se koriste biblioteke *roscpp ili rospy* pisane u programskom jeziku Python, odnosno C++ [6].
- **Teme** (eng. *Topics*) u ROS-u mogu se opisati kao sabirnice poruka sa definiranim, jedinstvenim imenom. Može se reći kako su teme zapravo ime kojim se određuje sadržaj poruka. Poruke se u prenose putem sustava čvorova koji objavljuju i čvorova koji se pretplaćuju na neku temu (eng. *publisher/subscriber system*). Kada neki čvor šalje poruku preko teme kaže se da on objavljuje na tu temu, a suprotno kada neki čvor prima poruku putem teme kažemo da se on pretplaćuje na temu. ROS čvorovi koji objavljuju ne znaju koji još čvorovi objavljuju na neku temu niti koji se čvorovi pretplaćuju, dakle stvaranje i korištenje poruka odvojeni su [5].
- **Master** (eng. *Master*) je središnja jedinica sustava koja bilježi imena svih dijelova grafa procesa te osigurava da se čvorovi koji to trebaju mogu pronaći, odnosno izmijeniti informacije. Kada se bilo koji čvor pokrene u ROS-u on će prvo potražiti nadzorni čvor i prijaviti ime čvora. Isto vrijedi i za zaustavljanje bilo kojeg procesa. Na taj način ROS master ima sve detalje o svim pokrenutim čvorovima u ROS sustavu koje koristi kako bi povezivao čvorove. Kada čvor počne objavljivati podatke na neku temu istovremeno će detalje vezane uz tu temu, poput imena i vrste, predati nadzornom čvoru. Nadzorni čvor u tom slučaju provjerava postoji li još neki čvor koji se želi pretplatiti na tu istu temu. Ukoliko je neki čvor pretplaćen na temu, nadzorni čvor će mu proslijediti informacije o čvoru koji objavljuje te tako omogućiti izmjenu podataka. Poslije spajanja dva čvora nadzorni čvor više nema nikakvu kontrolu nad procesima. Nadzorni čvor se

pokreće naredbom *roscore*, a u slučaju sustava distribuiranog na više računala nije potrebno na svakom računalu pokretati novi master već je dovoljno to učiniti jedanput [5].

- **Poruke** (eng. *Messages*) se u ROS sustavu koriste za komunikaciju između dvaju čvorova. Već su ranije navedeni pojedini tipovi poruka i kako se one razlikuju. Poruka je ustvari struktura podataka koja sadrži neku vrijednost i koja se može izmjenjivati između procesa. Neki od standardnih tipova podataka u ROS-u su *integer, floating point, boolean, time* i dr. Moguće je također sastaviti i vlastiti tip poruke [5].
- **Servisi** (eng. *Services*) se u ROS-u koriste kada je neka informacija potrebna u jednom trenutku za razliku od poruka kod kojih je protok informacija kontinuiran. Isto tako teme ne mogu ostvariti ovakav protok podataka jer je jedan čvor ne može istovremeno na istu temu objavljivati i s nje primati poruke. ROS servisi definiraju se kao parovi poruka. Potrebno je u *.srv* datoteci definirati tip podataka zahtjeva i odgovora te spremiti ove datoteke u *srv* direktorij unutar paketa. Kada se ROS servis pokrene jedan čvor postaje poslužitelj (eng. *server*), a drugi klijent koji od servera traži uslugu (eng. *service*) [5].
- **Server parametara** (eng. *Parameter Server*) omogućuje spremanje podataka na centralnoj lokaciji gdje su dostupni svim čvorovima. Svi čvorovi mogu pristupiti podacima, upisivati podatke, mijenjati ih pa i brisati. Najbolja je u server parametara upisivati podatke koji su statički i ne mijenjaju vrijednost vrlo često, primjerice poput konfiguracijskih parametara. Server parametara dio je ROS mastera [6].
- **Spremnici** (eng. *Bags*), odnosno datoteke tipa *.bag* (po čemu su i dobile ime *bags*), koriste se u ROS sustavu za pohranu podataka izmijenjenih ROS porukama. Imaju vrlo važnu ulogu i stoga je razvijen velik broj korisnih alata za spremanje, analizu i vizualizaciju ovih datoteka. Uobičajeno se za stvaranje spremnika koristi alat *rosbag* koji se pretplaćuje na jednu ili više ROS tema i sprema podatke dobivene porukama redoslijedom kojim dolaze. Kasnije se ovi podaci mogu reproducirati na tim istim temama ili čak prepisati drugim temama. Ovakav alat vrlo je koristan u istraživanju i razvoju sustava jer omogućuje pristup, a i reprodukciju očitanja sa senzora [6].

2.3.3. ROS razina zajednice

ROS razina zajednice (eng. *Community Level*) stavlja na raspolaganje resurse koji omogućuju zajednici korisnika ROS sustava izmjenu softvera, ali i znanja i iskustava. Ti resursi su sljedeći [5, 6]:

- **Distribucije** (eng. *Distributions*) su setovi ROS paketa podijeljenih po verzijama sustava. Slične su distribucijama sustava Linux (Debian, Ubuntu, Fedora i dr.). Smisao je olakšati instalaciju i prikupljanje ROS softvera. Kada je distribucija izdana promjene su za nju minimalne i ograničene su na ispravljanje grešaka koda, bez mijenjanja osnovnih paketa.
- **Repozitoriji** (eng. *Repositories*). Razvoj i širenje ROS-a ovisi o mreži repozitorija programskog koda. Potiče se razvoj vlastitih repozitorija ROS paketa svih korisnika.
- **ROS Wiki** glavni je internetski forum za dokumentaciju i informacije o ROS sustavu. Otvoren je za sve, dakle svatko može izraditi korisnički račun i prijaviti se te sudjelovati u stranici vlastitom dokumentacijom, prijavljujući i ispravljajući prepoznate greške u kodu ili pišući upute za druge korisnike.
- **Sustav za prijavu grešaka u kodu** (eng. *Bug Ticket System*) predstavlja veliku prednost zajednice jer što se softver više koristi, veća je vjerojatnost za pronalazak i otklanjanje grešaka što ROS čini stabilnijim.
- **Pretplata na službeni ROS mail** (eng. *Mailing Lists*) olakšava komunikaciju u zajednici. Na ovaj način šire se vijesti o novostima u ROS sustavu, a predstavlja i mjesto pogodno za postavljanje pitanja iskusnijim korisnicima.
- **ROS odgovori** (eng. ROS *Answers*) je internetska stranica namijenjena za pitanja i odgovore na sve što ima veze s ROS sustavom. Vrlo koristan resurs za sve početnike jer se mogu pronaći rješenja iskusnih ROS korisnika.
- **Blog** donosi zajednici novosti uz mnoge slike i videa vezana uz ROS

Mnogi izvori slažu se u tome kako je ROS zajednica zasigurno jedan od najvažnijih aspekata ROS sustava. Bez postojanja aktivne zajednice popularizacija i sve veća upotreba ROS sustava ne bi se dogodila. ROS nije jedini sustav za razvoj robotskih sustava koji se pojavio, ali je svakako najuspješniji. Mnogi kao razlog zašto je to tako vide u dvije stvari. Kao prvo, u

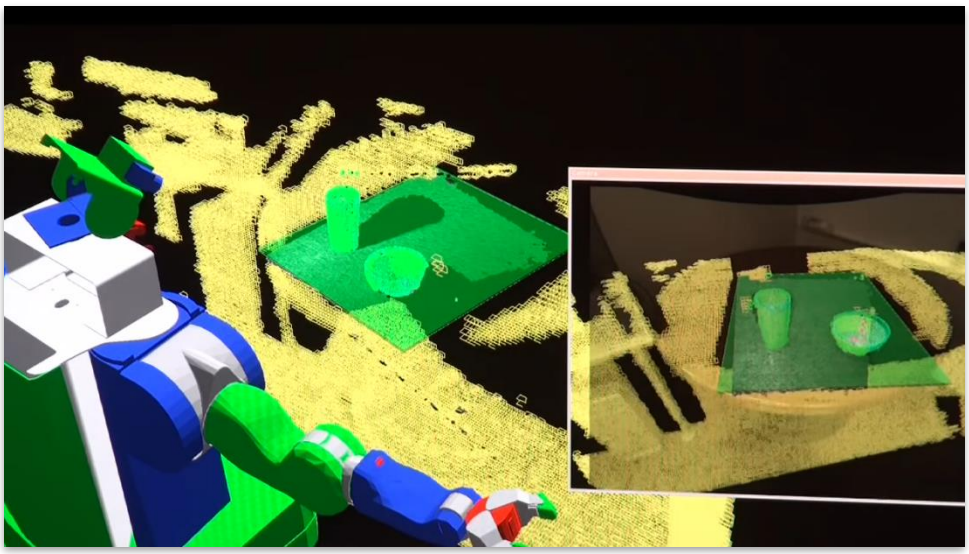
činjenici što je besplatan i otvoren za korištenje svima. Te drugo, i puno bitnije, u snažnoj i aktivnoj zajednici ljudi koji razvijaju i održavaju ROS sustav i njegovih korisnika.

2.4. Alati ROS sustava

2.4.1. RViz

RViz je službeno programsko sučelje za 3D vizualizaciju robota u ROS-u. Pomoću ovog alata moguće je vizualizirati informacije sa raznih senzora robota (laseri, kamere, enkoderi…), tj. u simulaciji vidjeti ono što robot vidi i procesira te kako djeluje u prostoru. *RViz* olakšava razvoj sustava jer omogućuje brzo testiranje i otklanjanje grešaka u kodu pomoću vizualizacije koda.

Podatke je u *RViz* moguće unijeti direktno sa robotskih senzora ili ručno putem sučelja. Koristi se i kod planiranja putanje manipulatora za vizualizaciju razlike stvarne i planirane putanje robota, željene pozicije, ali i predmeta u radnom prostoru robota. Bitno je napomenuti kako *RViz* nije alat za simulaciju robota, već samo vizualizaciju. Naziv *RViz* nastao je kao skraćenica engleskih riječi ***R**obot **Vis**ualization.* Dolazi integriran sa ROS distribucijama. Na slici 11 prikazana je vizualizacija predmeta u radnom prostoru robota [6].



Slika 11 Vizualizacija radnog prostora robota pomoću *RViz*-a [6]

2.4.2. MoveIt!

MoveIt! je osnovni softverski okvir za planiranje trajektorije i manipulacije predmeta u prostoru u ROS sustavu. Integriran je na velikom broju robota uključujući robota *PR2*,

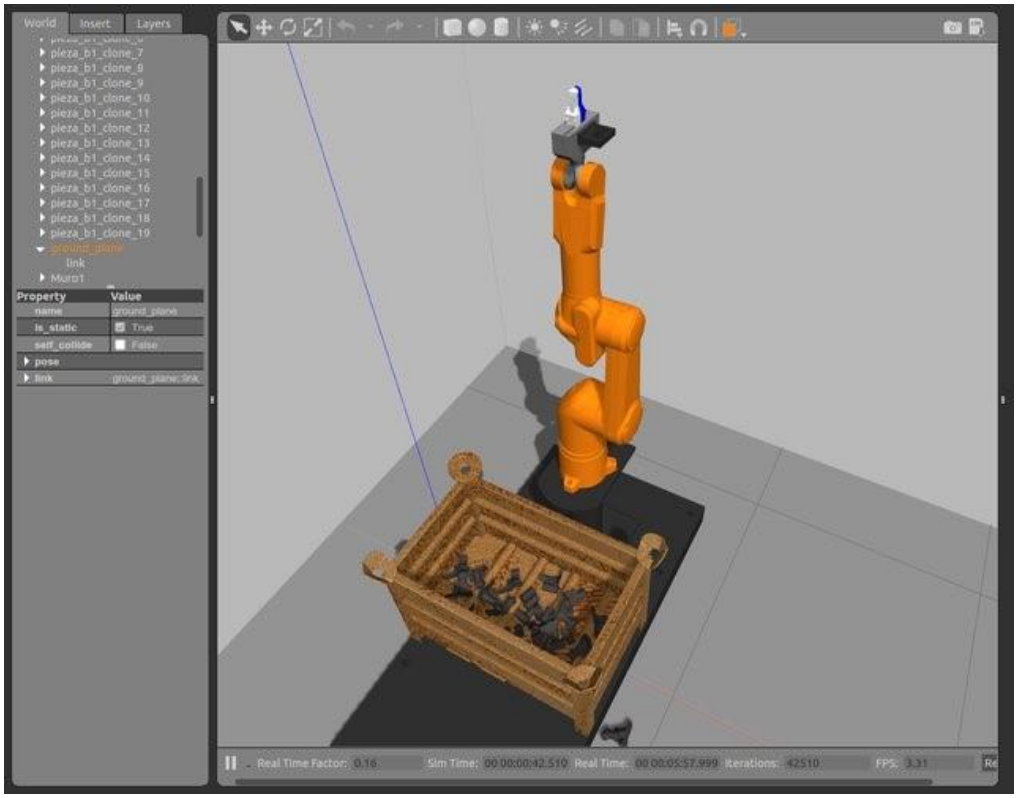
Robonauta (robot instaliran na ISS svemirskoj postaji), a korišten je za vođenje robota i u sklopu ovog rada. Programski kod napisan je u potpunosti u programskom jeziku C++, ali podržava i Python.

Kombinira naprednu kinematiku, planiranje putanje i sposobnost provjere kolizije kako bi isplanirao i izvršio optimalnu putanju na bilo kojem robotskom manipulatoru ili mobilnom robotu. *MoveIt!* je u potpunosti agnostičan prema tipu robota pa ga koriste stotine tvrtki u komercijalne svrhe, ali i mnogi istraživački laboratoriji u istraživanjima kako bi razvili inovativna rješenja na području automatizacije. Osim za planiranje trajektorija koristi se i za planiranje hvatanja i manipulacije predmeta jer omogućuje dodavanje velikog broja predmeta u radni prostor robota. Za vizualizaciju robota i putanja koristi ranije spomenuti alat *RViz*. Prvotno je razvijen kao dio ROS sustava 2013. godine, no kasnije se izdvaja kao zaseban projekt [13].

2.4.3. Gazebo

Gazebo je 3D simulator za simulaciju jednog ili više robota u složenom, unutarnjem ili vanjskom prostoru sa ugrađenim dodacima koji omogućuju izvanrednu simulaciju fizike, kinematike i dinamike robota. Osim upravljanja robota može simulirati i rad senzora pa čak i smetnje na pojedinim senzorima. *Gazebo* je kompatibilan s ROS-om pa je moguće pokretanjem potrebnih čvorova upravljati robotom u simulaciji, spremati podatke sa senzora i hardvera robota.

Jasne su prednosti simulacije stvarnog robota u virtualnom okruženju i danas je simulacija neizostavan alat u razvoju bilo kojeg robotskog sustava. Dobar simulator omogućuje vrlo brzo ispitivanje algoritama i dizajna robota te treniranje sustava umjetne inteligencije koristeći realističan radni prostor. Isto kao i ROS i svi alati navedeni u ovom poglavlju su otvorenog koda i besplatni za sve. Slika 12 prikazuje sučelje *Gazebo* simulatora prilikom simulacije izuzimanja predmeta s hrpe. [12].



Slika 12 Simulacija izuzimanja predmeta s hrpe u Gazebu [12]

2.4.4. ROS–Industrial

ROS–Industrial ili kraće ROS–I je projekt, odnosno program otvorenog koda koji proširuje područje primjene ROS sustava sa istraživačkog rada na sveučilištima i u laboratorijima na industriju i robotski hardver koji se ovdje koristi. Cilj je spojiti u jedno dobre strane ROS sustava i pouzdanost i sigurnost kontrolera industrijskih robota.

Započeo je kao zajednički projekt tvrtki *Yaskawa Motoman Robotics* i *Willow Garage* te istraživačkog laboratorija *Southwest Research Institute* kao pokušaj primjene ROS-a u industrijskom okruženju za automatizaciju proizvodnje. Dio ROS-Industriala je i repozitorij koji uključuje pakete za robotski hardver koji je često u upotrebi (manipulatori, hvataljke i senzori različitih proizvođača). Repozitorij je objavljen na mrežnoj platformi *GitHub* i javno dostupan od 2013. godine. Iste te godine osnovan je i Konzorcij zadužen za što brži razvoj i održavanje ROS-Industriala kroz [14]:

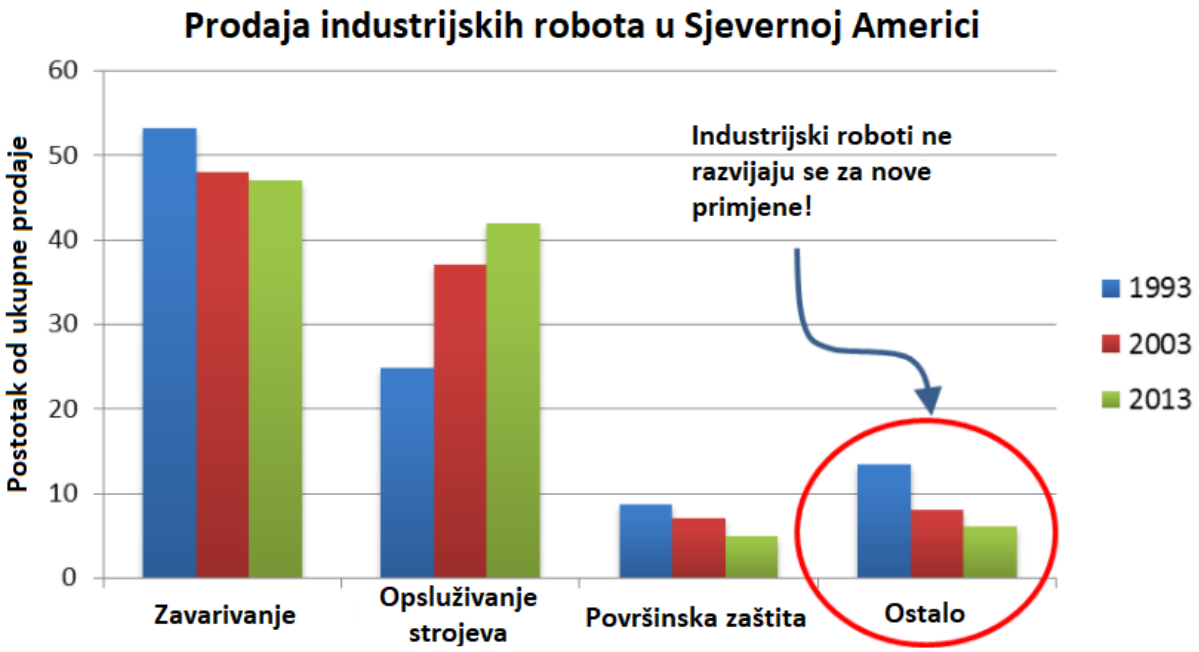
- Određivanje prioriteta razvoja mogućnosti ROS-Industriala s obzirom na zahtjeve zajednice i tržišta

- Uvođenje i primjenu standarda pri razvoju programskog koda za robote kako bi bio robustan i siguran za primjenu u industriji
- Pružanje usluga tehničke podrške i edukacije novih korisnika

Trenutno u svijetu postoje tri konzorcija na tri kontinenta, u Sjevernoj Americi, Europi te Aziji. Konzorcij je neprofitna udruga članova, a da bi se postalo članom Konzorcija potpisuje se ugovor o članstvu te je potrebno doprinijeti financijski u skladu sa razinom članstva. Konzorcij u Europi smješten je u Stuttgartu u Njemačkoj, a vodi ga institut za proizvodno inženjerstvo i automatizaciju *Fraunhofer IPA*. Konzorcij je financijski podržan od Europske komisije projektom *ROSIN*. Sve zajedno u svoja tri dijela Konzorcij ima preko 60 članova koji predstavljaju akademsku zajednicu, *start-up* tvrtke te velike multinacionalne kompanije. Predstavnici dolaze iz gotovo svih djelatnosti: automobilske i svemirske industrije, građevine, poljoprivrede, logistike, elektrotehnike, energetike, zdravstva i obrane, a naravno i proizvodne i robotske industrije [14].

2.4.4.1. Primjena ROS-a u obradnim sustavima

Motivacija za razvoj ROS-Industrial nadogradnje za ROS sustav pojavila se istraživanjem tržišta industrijskih robota u Sjevernoj Americi iz 2013. godine. Naime, iako je prodaja industrijskih robota bila u porastu, područje njihove primjene nije se mijenjalo. Više od 20 godina primjena industrijskih robota bila je ograničena na približno uvijek isti set zadataka – zavarivanje, rukovanje materijalima pri posluživanju strojeva, montažu i površinsku zaštitu. To je tako jer su roboti isplativi za repetitivne i poslove visokog volumena dok za poslove manjeg volumena i veće varijabilnosti baš i ne. No, to ne bi bilo toliko zabrinjavajuće bez činjenice da je unutar tih nešto više od 20 godina u razvoj robotike (najviše u vizijske sustave, mobilne i kolaborativne robote) utrošeno više od milijardu dolara [14].



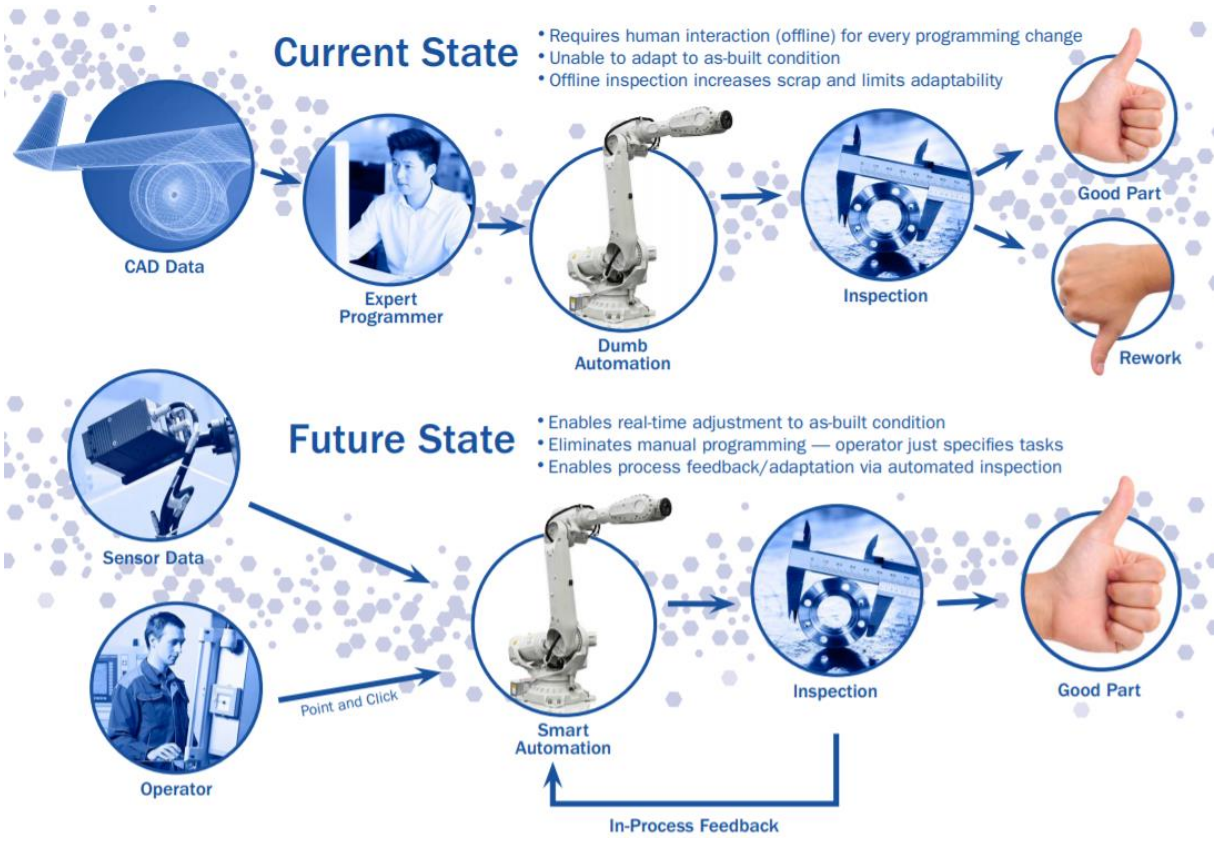
Slika 13 Istraživanje primjene industrijskih robota u Sj. Americi (1997. - 2013.) [14]

Kako pokazuju rezultati istraživanja prikazani na slici 13, unatoč visokim ulaganjima broj različitih primjena robota u industriji je u padu. To je tako jer su sredstva uglavnom bila uložena u istraživanje i razvoj gdje su i razvijene određene inovacije, no iako su roboti hardverski dovoljno fleksibilni za velik broj različitih primjena u automatizaciji procesa velika barijera primjeni inovacija u industriji je softverska arhitektura mnogih industrijskih robota. Jednostavno, kada se uračuna trošak integracije i programiranja robota (za koje je potreban visokokvalificirani kadar), primjena robota u poslovima niskog volumena i visoke varijabilnosti procesa nije ekonomski isplativa. ROS-I potencijalno nudi rješenje upravo na taj problem [14].

Scan-N-Plan tehnologija

Ova tehnologija za sada je glavni pravac razvoja primjene ROS sustava u obradnim sustavima sa nekoliko različitih rješenja koja se baziraju na jednom principu – skeniranja površine koju treba obraditi i na temelju toga generiranju trajektorije robota na kojem se nalazi određeni alat za obradu. *Scan-N-Plan* tehnologija je naziv za skupinu alata koja omogućuje planiranje trajektorije robota u stvarnom vremenu s obzirom na podatke s 3D skenera. Klasično programiranje robota odvija se *online*, programiranjem robota pomoću privjeska za učenje ili *offline*, na računalu, neovisno o fizičkom robotu uz korištenje simulacije i CAD modela obratka. No, niti jedna od ovih metoda nije idealna. Pri *online* programiranju potrebno je robota dovesti

u zahtijevane pozicije te ih umjeriti, a nakon toga napisati program, testirati ga i na kraju optimirati. Sve ovo zahtjeva puno vremena u kojem robot ne može biti u pogonu, što znači zastoj proizvodnje i u konačnici financijski trošak. *Ofline* metoda pak zahtjeva visokokvalificiranu radnu snagu što znači višu cijenu programiranja i proizvodnje, a često je program ponovno potrebno korigirati na stvarnoj robotskoj stanici. Metoda skeniranja i planiranja omogućuje da se veliki dio ovog proces zaobiđe. Ova tehnologija u budućnosti može imati primjenu u adaptivnoj obradi s obzirom na dinamične promjene stanja alata i geometrije obratka, a moguće i adaptaciju procesa obrade u stvarnom vremenu s obzirom na mjerenja određenih senzora (sila, akustičke emisije, temperature…). Slika 14 prikazuje usporedbu sadašnjih i potencijalnih budućih procesa obrade uz korištenje robotskih sustava [14].



Slika 14 Usporedba primjene robota u obradnim sustavima danas i u budućnosti [14]

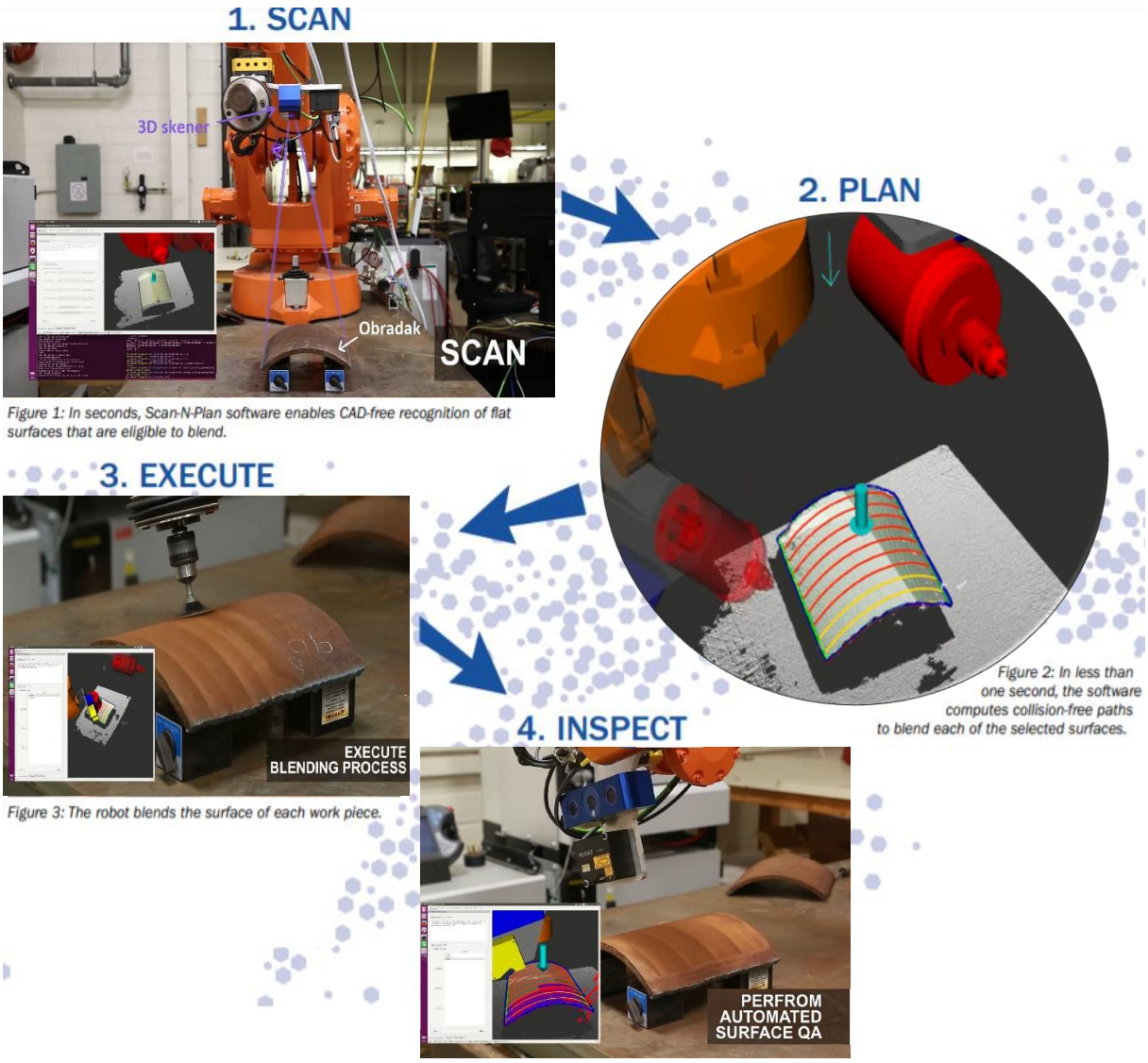


Figure 1: In seconds, Scan-N-Plan software enables CAD-free recognition of flat surfaces that are eligible to blend.

Figure 2: In less than one second, the software computes collision-free paths to blend each of the selected surfaces.

Figure 3: The robot blends the surface of each work piece.

Slika 15 Primjer primjene *Scan-N-Plan* tehnologije u procesu poliranja

Slika 15 prikazuje proces robotske obrade poliranjem uz korištenje tehnologije *Scan-N-Plan*. Mnogi obradni procesi poput primjerice lijevanja ili zavarivanja nakon primarne obrade zahtijevaju dodatnu obradu površine. Postoje mnogi načini završne obrade poput pjeskarenja, brušenja ili poliranja i mnogi se mogu izvoditi ručno od strane čovjeka, no čovjek ručnom obradom nikada neće moći ostvariti ponovljivost stroja. To dovodi do varijacija u kvaliteti proizvoda, a povećava i cijenu proizvodnje. Isto tako cilj je jednog dana čovjeka osloboditi rada u potencijalno opasnim uvjetima koji nisu rijetkost u procesu obrade odvajanjem čestica. Poželjno je naći rješenje koje nudi fleksibilnost ručne obrade s ponovljivošću i sigurnošću strojne, a tehnologija *Scan-N-Plan* upravo je korak u tom smjeru [14].

3. EKSPERIMENTALNI POSTAV

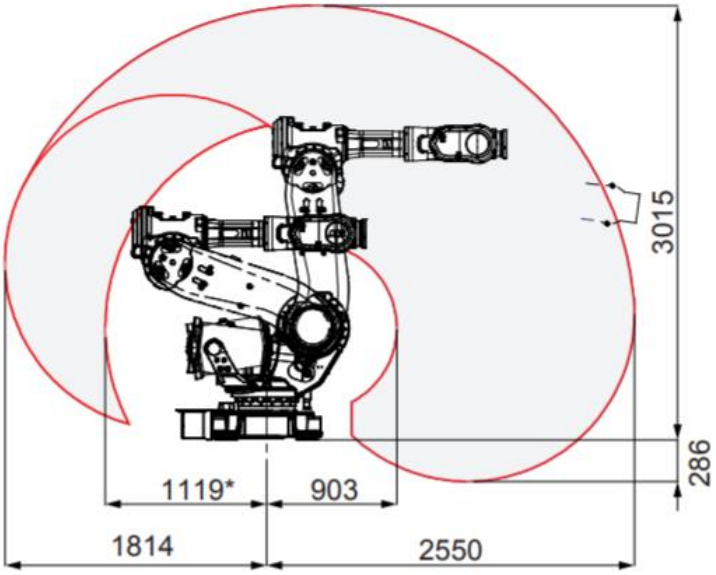
U ovom dijelu rada bit će dan kratki opis i pregled specifikacija koje se tiču robota i kontrolera, a bitne su za razvoj ROS sustava za vođenje industrijskog robota. Isto tako opisat će se programski paket *RobotStudio* u kojem će biti izvršeno vođenje robota pomoću ROS sustava u simulaciji.

3.1. Industrijski robot ABB IRB6640

Cilj ovog diplomskog rada je ostvariti vođenje industrijskog robota tipa ABB IRB6640 235/2.55 prikazanog na Slika 16. Broj 235 u nazivu modela robota govori o maksimalnoj nosivosti u kg, dok broj 2.55 označava doseg robota u m (Slika 17).

Slika 16 Industrijski robot ABB IRB6640

- Najveća nosivost: 235 kg
- Doseg: 2,55 m



Slika 17 Doseg robota ABB IRB6640 [15]

Robot ima šest stupnjeva slobode (šest rotacija) kako je prikazano u tablici 1.

Tablica 1 Opis vrste i raspon gibanja po zglobovima

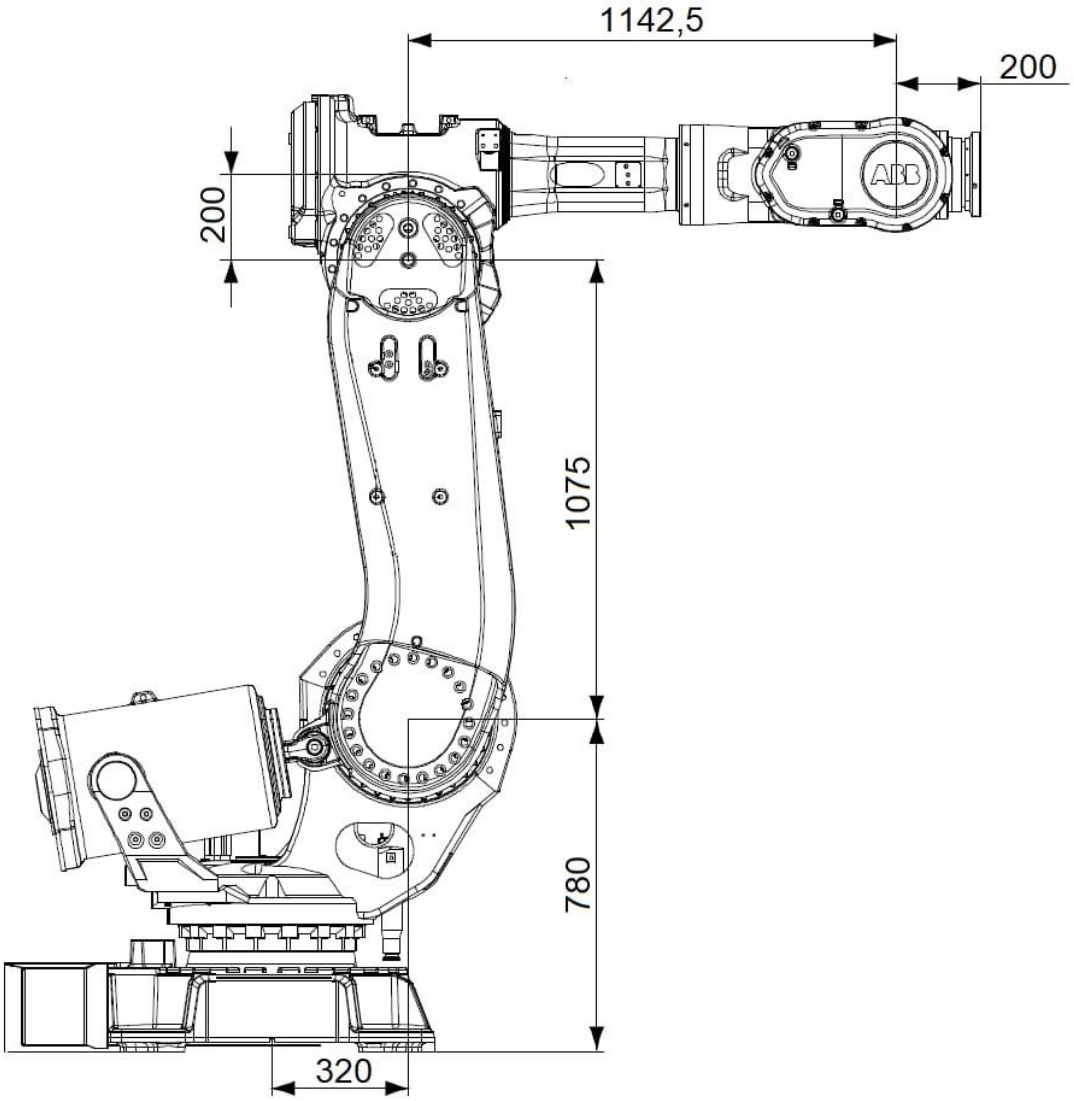
Zglob	Vrsta gibanja	Raspon
1	Rotacija zgloba između postolja i ruke	+170° do -170°
2	Rotacija ručnog zgloba	+85° do - 65°
3	Rotacija ručnog zgloba	+70° do -180°
4	Rotacija šake	+300° do -300°
5	Savijanje šake	+120° do -120°
6	Rotacija prihvatnice	360° do -360°

Tablica 2 Ograničenja brzine zglobova

Zglob	Brzina rotacije
1	100 °/s
2	90 °/s
3	90 °/s
4	170 °/s
5	120 °/s
6	190 °/s

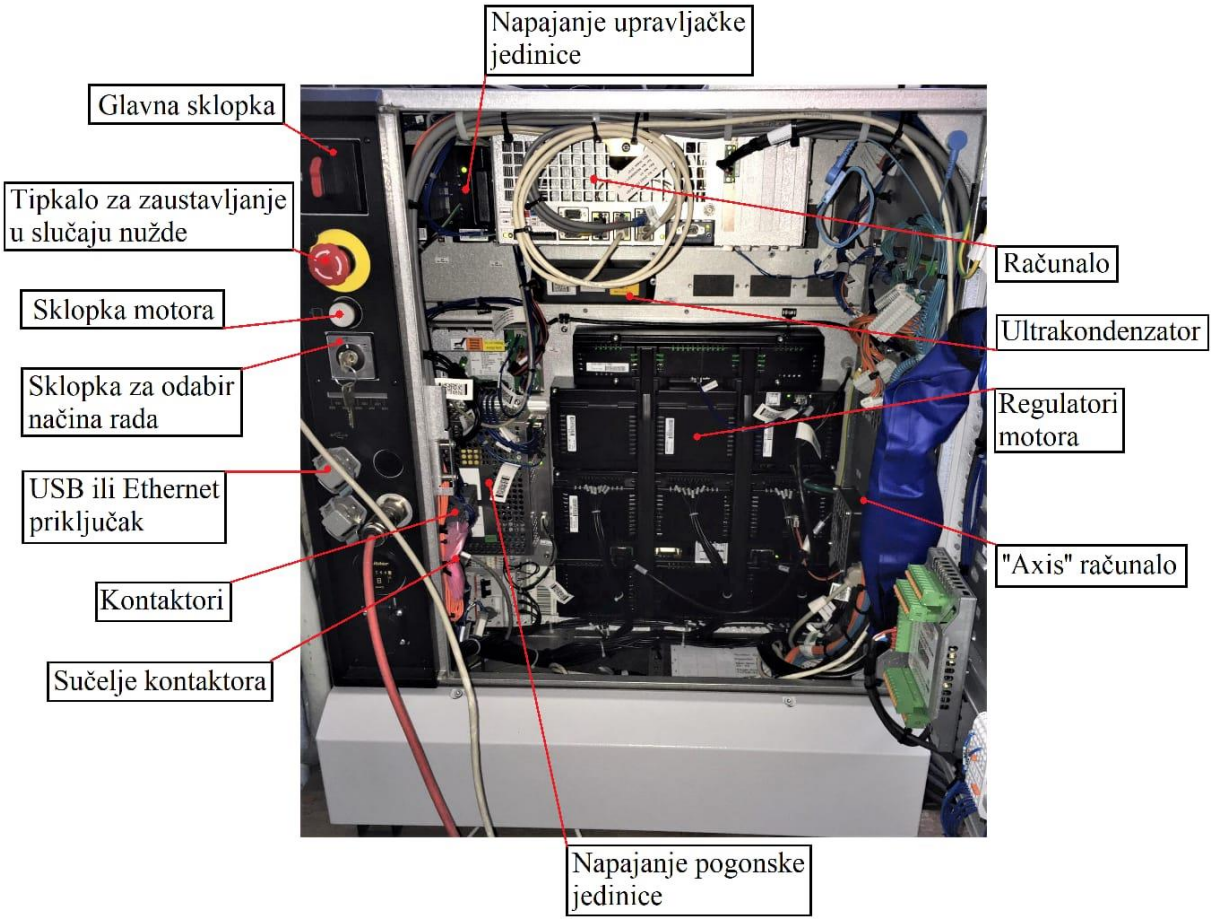
Najveće brzine rotacije pojedinih zglobova dane su u tablici 2.

Prilikom razvoja 3D modela i konfiguracijske datoteke robota korišteni su podaci prikazani slikom 18



Slika 18 Dimenzije potrebne za razvoj konfiguracijske datoteke i 3D modela robota

3.2. Kontroler IRC5

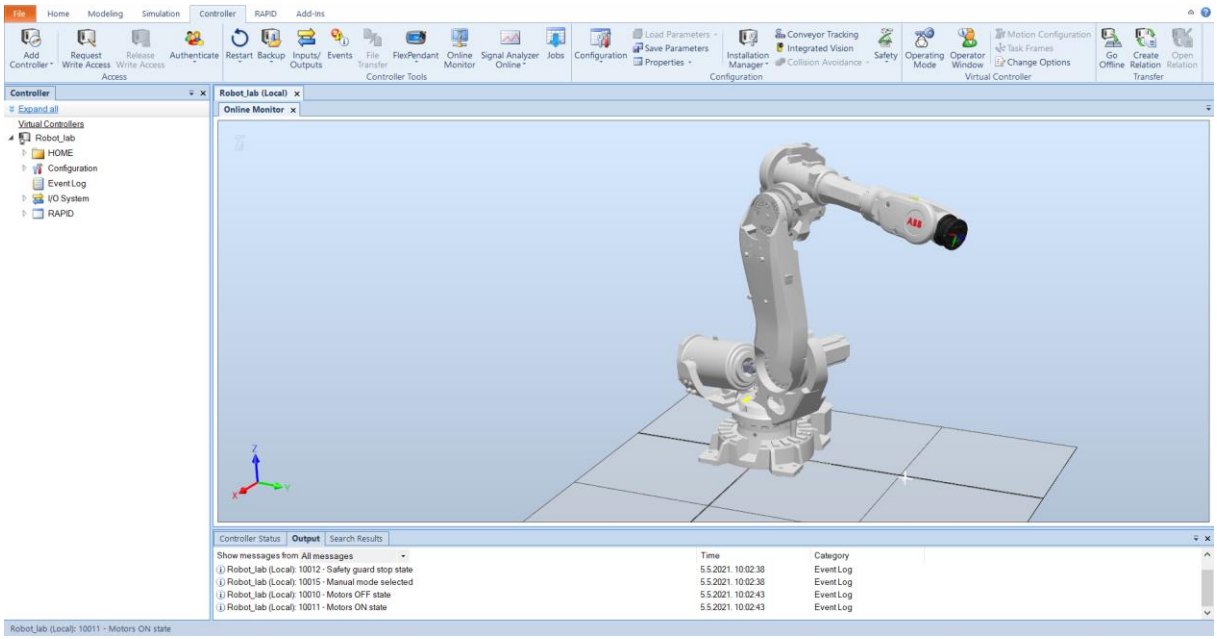


Slika 19 Kontroler robota

Slika 19 prikazuje pojedine komponente kontrolera.

3.3. Programski paket ABB *RobotStudio*

ABB *RobotStudio* je programski paket za modeliranje, simulaciju i *offline* programiranje robota. *Offline* programiranje daje mogućnost da se proces proizvodnje odvija neovisno o procesu programiranja robota. Kako se proces proizvodnje ne zaustavlja isplativost robotskog sustava je veća. Program nudi i opciju uvođenja CAD modela iz drugih programskih paketa za modeliranje kako bi se omogućio vjerni prikaz radnog prostora. Kombinacijom *offline* programiranja i simulacije robota može se proces proizvodnje optimizirati prije nego se implementira na fizičkom sustavu. Programsko sučelje programa vidljivo je slici 20.



Slika 20 Sučelje programa ABB *RobotStudio*

Unutar programa dostupni su modeli ABB robota moderne generacije koja dolazi s IRC5 kontrolerom. Pojedini robot može se izabrati odabirom na tipku *Home* na alatnoj traci te zatim opcije *ABB Library.* Neki od ponuđenih robota prikazani su slikom 21.

Prilikom izrade ovog rada program *RobotStudio* korišten je za simulaciju jer sadrži virtualni kontroler koji je identičan onome na fizičkom robotu. Realizacijom vođenja prvo na simuliranom robotu i kontroleru željela se potvrditi funkcionalnost razvijenog ROS sustava prije implementacije na stvarnom robotskom sustavu kako bi se spriječila mogućnost nastanka materijalne štete ili ozljede.

Slika 21 Izbor robota u programu *ABB RobotStudio*

4. EKSPERIMENTALNI DIO

Kako bi se ostvario konačni cilj ovog diplomskog rada - realizacija vođenja industrijskog robota ABB IRB6640 korištenjem ROS sustava, bilo je potrebno u ROS sustavu razviti odgovarajuće konfiguracijske datoteke robota kao i datoteke za pokretanje ROS čvorova koji će ostvariti komunikaciju s kontrolerom robota. U ovom poglavlju bit će opisani koraci u razvoju sustava od odabira i instalacije distribucije ROS-a sve do pokretanja robota - prvo u simulaciji, u programu *RobotStudio*, a zatim i fizičkog robota u laboratoriju.

4.1. Instalacija ROS sustava

Prvi korak u realizaciji vođenja robota pomoću ROS sustava je naravno instalacija sustava. Bilo je potrebno pažljivo odabrati ROS i Linux verzije distribucija. S obzirom na verziju softvera (*RobotWare*) robotskog kontrolera potrebno je odabrati ispravnu verziju ROS-Industrial paketa robotskog drivera. Nadalje, ROS paketi su kompatibilni s određenim verzijama ROS distribucija koje su u konačnici kompatibilne samo s određenim verzijama distribucija Linuxa. Isto tako potrebno je bilo obratiti pažnju na to koliko još dugo pojedine kompatibilne verzije ROS distribucija imaju podršku u pogledu održavanja sustava. Prema svim navedenim zavisnostima odabrane su sljedeće verzije distribucija: Linux Debian 9 (Stretch) i ROS Melodic Morenia.

Linux Debian Stretch instaliran je na računalo s operacijskim sustavom Windows10 pomoću programskog paketa za virtualizaciju hardvera *Oracle VM VirtualBox*. Nakon toga ROS je instaliran u sustavu Linux Stretch prema detaljnim uputama na službenoj ROS stranici.

Sljedeći važan korak bio je stvaranje te izgradnja ROS radnog prostora (eng. *Workspace*) što je učinjeno izvršavanjem sljedećih naredbi:

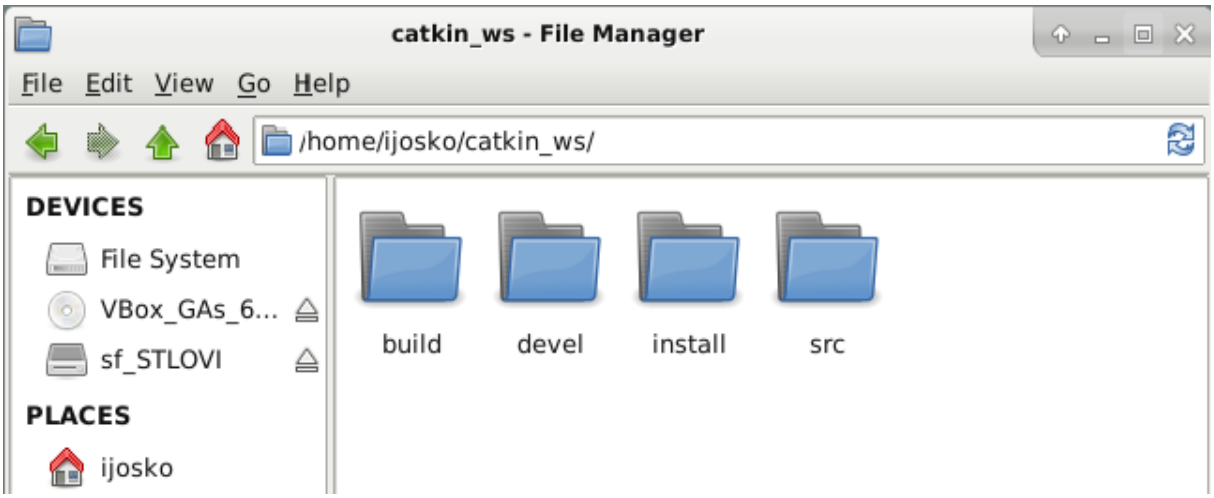
```
$ mkdir -p ~/catkin_ws/src

$ cd ~/catkin_ws/

$ catkin_make
```

Prvo se kreira novi direktorij koji će postati novi radni prostor te, unutar tog direktorija, datoteka *src*. Naziv *catkin_ws* (eng. *catkin workspace*) preuzet je prema službenim uputama, a inače se radni prostor može nazvati proizvoljno. Zatim se naredbom *cd* ulazi u stvoreni

direktorij te se naredbom *catkin_make* gradi (eng. *build*) radni prostor. Unutar radnog prostora može se instalirati i graditi, ali i prilagođavati ili razvijati vlastite ROS pakete. *Catkin* je službeni sustav za izgradnju u ROS-u. Na slici 22 prikazan je sadržaj radnog prostora nakon stvaranja i izgradnje. Može se uočiti kako su stvoreni novi direktoriji *build, devel* te *install*.



Slika 22 Sadržaj radnog prostora *catkin_ws*

Unutar direktorija *devel* nalazi se nekoliko datoteka tipa *.sh*. Prilikom svakog novog pokretanja terminala u Linuxu potrebno je izvršiti naredbu *source devel/setup.bash* koja će dodati potrebne ROS varijable na putanju kako bi se omogućila upotreba ROS naredbi i funkcija.

Kako je već rečeno u drugom poglavlju, ROS-Industrial nudi neke vrlo korisne pakete, poput paketa *Simple Message*, koji definira jednostavnu vezu za izmjenu poruka i odgovarajuće protokole kako bi se ostvarila komunikacija ROS sustava s kontrolerom industrijskog robota. Kako bi se ostvarile sve funkcionalnosti ROS-Industrial programa u vidu realizacije vođenja industrijskog robota potrebno je, nakon instalacije ROS-a i postavljanja radnog prostora, instalirati odgovarajući *Industrial core* paket otvaranjem terminala i upisivanjem naredbe *sudo apt install ros-melodic-industrial-core* [6].

Nadalje, ROS-Industrial repozitorij sadrži metapakete nekoliko proizvođača industrijskih robota kao što su: ABB, Fanuc, Kuka (eksperimentalni paketi), Motoman, Robotiq te Universal Robots. S obzirom da je zadatak bio ostvariti vođenje industrijskog robota proizvođača ABB to je vrlo praktično. Metapaketi svakog od navedenih proizvođača, pa tako i ABB-a, sadrže velik broj paketa za različite verzije robota. Postoje dvije vrste paketa za specifičnu verziju robota: *moveit* konfiguracijski paketi i *support* paketi Metapaket proizvođača ABB nije sadržavao paket za robota ABB IRB6640 235/2.55, no sadržavao je pakete nekih sličnih verzija

robota koje su poslužile kao dobar primjer i bazu za izradu vlastitog paketa. Opis, namjena i sadržaj ovih paketa kao i proces razvoja vlastitih paketa bit će opisan dalje u ovom poglavlju.

4.2. Prilagodba ROS sustava za vođenje industrijskog robota

4.2.1. ABB IRB6640 support paket

Paket ABB IRB6640 dostupan je kao dio metapaketa ABB koji je dostupan u repozitoriju ROS-Industriala. Predstavlja programsku podršku (eng. *support*) za robota ABB IRB6640, no verziju IRB6640-185/280 koja je različita od verzije robota dostupnog u laboratoriju stoga je ROS paket trebalo prilagoditi kako bi se nadalje omogućio daljnji razvoj paketa za realizaciju vođenja industrijskog robota.

Paket sadrži konfiguracijske datoteke, 3D modele i datoteke za pokretanje robota. Datoteke se prema dogovoru u svim ROS paketima raspoređuju u nekoliko direktorija, prema namjeni, kako bi se olakšalo dijeljenje ROS paketa sa zajednicom te korištenje tih istih paketa. Datoteke su u direktorije podijeljene na sljedeći način:

- ***Config*** direktorij sadrži konfiguracijske datoteke robota koje su tipa *.urdf* ili *.xacro*. Ove datoteke osnove su za razvoj sustava vođenja robota i više o tim tipovima datoteka bit će rečeno dalje u poglavlju.
- ***Launch*** direktorij sadrži datoteke za pokretanje koje se koriste za istovremeno pokretanje više ROS čvorova, pozivanje drugih datoteka ili upisivanje parametara u server parametara.
- ***Meshes*** direktorij sadrži 3D modele dijelova robota. Definiranje geometrije u ROS sustavu može, ali i ne mora biti ostvareno učitavanjem 3D modela, a to će biti objašnjeno u nastavku. S obzirom na to ovaj direktorij je opcionalan. Ako postoji najčešće je podijeljen na još dva poddirektorija: *Collision* – u kojem se nalaze 3D modeli robota za provjeru sudara i *Visual* – u kojem se nalaze 3D modeli robota za vizualizaciju. ROS sustav podržava gotovo sve formate 3D, no kao najbolji se preporučaju formati DAE te STL.

4.2.1.1. URDF

URDF ili *Unified Robotics Description Format* je programski jezik na bazi XML-a za označavanje podataka koji se koristi u istraživanju i industriji za modeliranje sustava sastavljenih od više dijelova kao što su roboti. Upravo je ovaj programski jezik osnova za razvoj svih robotskih modela u ROS sustava jer omogućuje vizualizaciju i simulaciju robota u ROS sustavu. Pomoću oznaka (eng. tag) definiraju se određene informacije koje će kasnije biti osnova za vođenje robota poput geometrije dijelova robota, boje, kinematike i dinamike. URDF jezik je programski jezik koji je čitljiv čovjeku, ali i računalu jer je XML format koji ono može jednostavno obraditi. To omogućuje brzi razvoj ROS paketa robota.

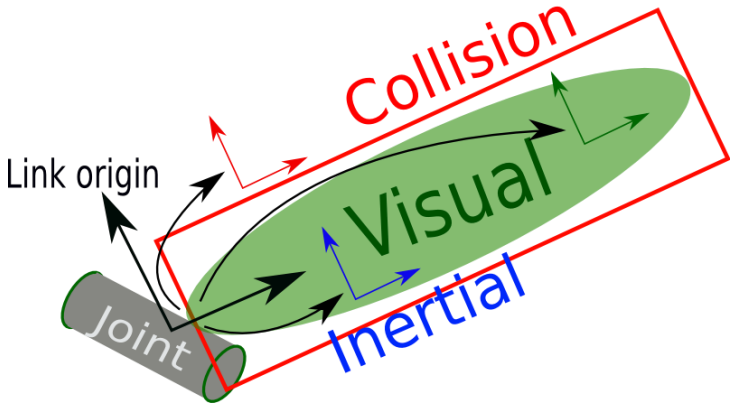
Oznake koje se koriste za modeliranje robota su [6]:

<robot>

Ovaj element je korijen, odnosno svi drugi elementi u definiranju modela robota su mu podređeni. Unutar ovog elementa definira se parametar *name,* tj. ime robota.

<link>

Element *link*, odnosno članak definira kruto tijelo kao dio robota. Unutar ovog elementa najčešće ima više različitih elemenata koji definiraju inercijske karakteristike te karakteristike za vizualizaciju (eng. *visual*) i provjeru sudara članka (eng. *collision*). Slikom 23 shematski su prikazane ove karakteristike. Potrebno je za svaki članak definirati ime kao parametar.



Slika 23 Karakteristike elementa <link> URDF datoteke [6]

<visual>

Unutar ovog elementa različitim podređenim elementima definiraju se karakteristike za vizualizaciju članka robota.

<origin>

Definira poziciju koordinatnog sustava vizualnog elementa u odnosu na koordinatni sustav članka. Definira se vrijednostima pomaka po koordinatnim osima referentnog koordinatnog sustava *x, y* i *z* te kutovima zakreta osi jednih u odnosu na druge *r* (rotacija oko osi *x*), *p* (rotacija oko osi *y*) i *y* (rotacija oko osi *z*)

<geometry>

Definira oblik vizualnog elementa članka. Definira se dodatnim podređenim elementima.

<box>

Vizualno definira članak u obliku kocke. Potrebno je odrediti veličinu kocke postavljanjem vrijednosti parametra *size*. Ishodište kocke je u težištu.

<cylinder>

Vizualno definira članak u obliku valjka. Potrebno je odrediti veličinu valjka postavljanjem vrijednosti *radius* i *lenght* odnosno polumjera osnovice i visine valjka. Ishodište valjka je u težištu.

<sphere>

Vizualno definira članak u obliku kugle. Potrebno je odrediti veličinu kugle postavljanjem vrijednosti *radius* odnosno polumjera osnovice i visine valjka. Ishodište kugle je u težištu.

<mesh>

Vizualno definira oblik članka učitavanjem 3D modela. Potrebno je u parametar *filename* postaviti putanju do 3D modela sačuvanog u memoriji. Moguće je postaviti i parametar *scale* kojim se može skalirati, odnosno povećati ili smanjiti 3D model.

<material>

Definira materijal vizualnog elementa članka. Moguće je ovaj element definirati i izvan nadređenih elementa <link> i <visual>, u glavnom elementu <robot>, što onda omogućava definiranje materijala drugih članaka jednostavno postavljanjem parametra *name*.

<color>

Definira boju materijala vizualnog elementa postavljanjem parametara *r, g, b* i *a* (eng. *red, green, blue, alpha*) u rasponu između 0 i 1.

<collision>

Unutar ovog elementa podređenim elementima definira se oblik članka koji se koristi za detekciju sudara članaka međusobno ili sa predmetima u okolini. Oblik članka za detekciju sudara može se razlikovati od vizualnog oblika članka što je često i slučaj. Oblik članka za detekciju sudara definira se što jednostavnijim, odnosno sa što manjim brojem površina kako bi se smanjilo opterećenje procesora prilikom izračunavanja putanja i time smanjilo vrijeme izvršavanja programa. Element <collision> definira se podređenim elementima koji su identični kao i kod elementa <visual>.

<inertial>

Unutar ovog elementa definiraju se inercijska svojstva članka.

<origin>

Ovim elementom moguće je mijenjati poziciju težišta s obzirom na koordinatni sustav članka. Definira se jednakim parametrima kao i kod elementa <visual>.

<mass>

Definira masu članka postavljanjem vrijednosti parametra *value*.

<inertia>

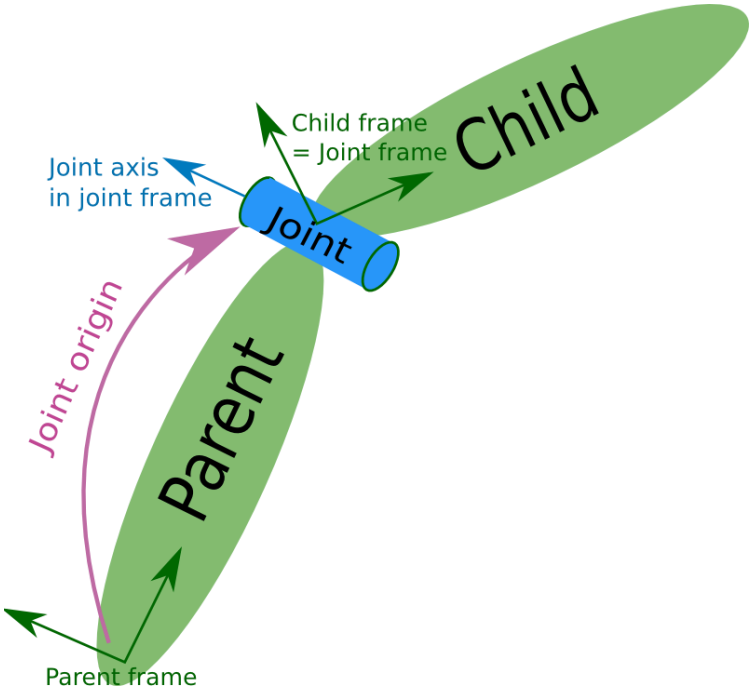
Definira matricu [3x3] momenata inercije članka. Obzirom da je matrica momenata inercije simetrična postavlja se samo šest elemenata matrice iznad dijagonale i to postavljanjem parametara i_{xx}, i_{xy}, i_{xz}, i_{yy}, i_{yz} te i_{zz}.

<joint>

Element *joint*, odnosno zglob određuje kinematiku i dinamiku zgloba između članaka ili između članka i okoline te ujedno može definirati i granice pokretljivosti zglobova robota. Obavezno se unutar ovog elementa moraju postaviti parametri *name* za ime zgloba te parametar *type* koji određuje vrstu zgloba. Zglobovi u URDF datoteci mogu biti određeni kao [6]:

- **Nepomični** (eng. *fixed*) – zapravo nije zglob jer je nepomičan, tj. nema ni jedan stupanj slobode gibanja (eng. *Degree of Freedom*, DOF). Nije potrebno definirati nikakve dodatne podređene elemente kod ovog tipa zgloba ili veze.

- **Rotacijski** (eng. *revolute*) – rotacija oko jedne osi u rasponu definiranom gornjom i donjom maksimalnom vrijednošću.
- **Kontinuirano rotacijski** (eng. *continuous*) – rotacija oko jedne osi bez ograničenja
- **Translacijski** (eng. *prismatic*) – translacijski pomak duž jedne osi uz raspon kretanja ograničen gornjom i donjom maksimalnom vrijednošću.
- **Planarni** (eng. *Planar*) – ovaj zglob dozvoljava kretanje po ravnini okomitoj na određenu os (2 DOF).
- **Plutajući** (eng. *Floating*) – ovaj zglob dozvoljava gibanje u prostoru (6 DOF).



Slika 24 Karakteristike elementa <joint> URDF datoteke [6]

<parent>

Definira članak u zglobu koji se naziva roditelj (eng. *parent*) postavljanjem imena tog članka u parametar *link*. (Slika 24)

<child>

Definira članak u zglobu koji se naziva djetetom (eng. *child*) postavljanjem imena tog članka u parametar *link*. (Slika 24)

<origin>

Definira transformaciju iz koordinatnog sustava članka roditelja u koordinatni sustav članka djeteta. Definira se jednakim parametrima kao i kod elementa <visual>.

<axis>

Definira os u koordinatnom sustavu zgloba oko koje rotiraju zglobovi definirani kao rotacijski ili duž koje je omogućeno kretanje translacijskih zglobova. Isto tako predstavlja normalu na ravninu po kojoj se mogu gibati zglobovi definirani kao planarni. Os se definira postavljanjem vrijednosti parametara *x, y* i *z* (npr. (1, 0, 0) označava rotaciju ili translaciju po osi *x*) Za zglobove definirane kao nepomični ili plutajući nije potrebno podesiti ovaj element.

<limit>

Definira ograničenja kretanja zgloba postavljanjem vrijednosti sljedećih parametara:

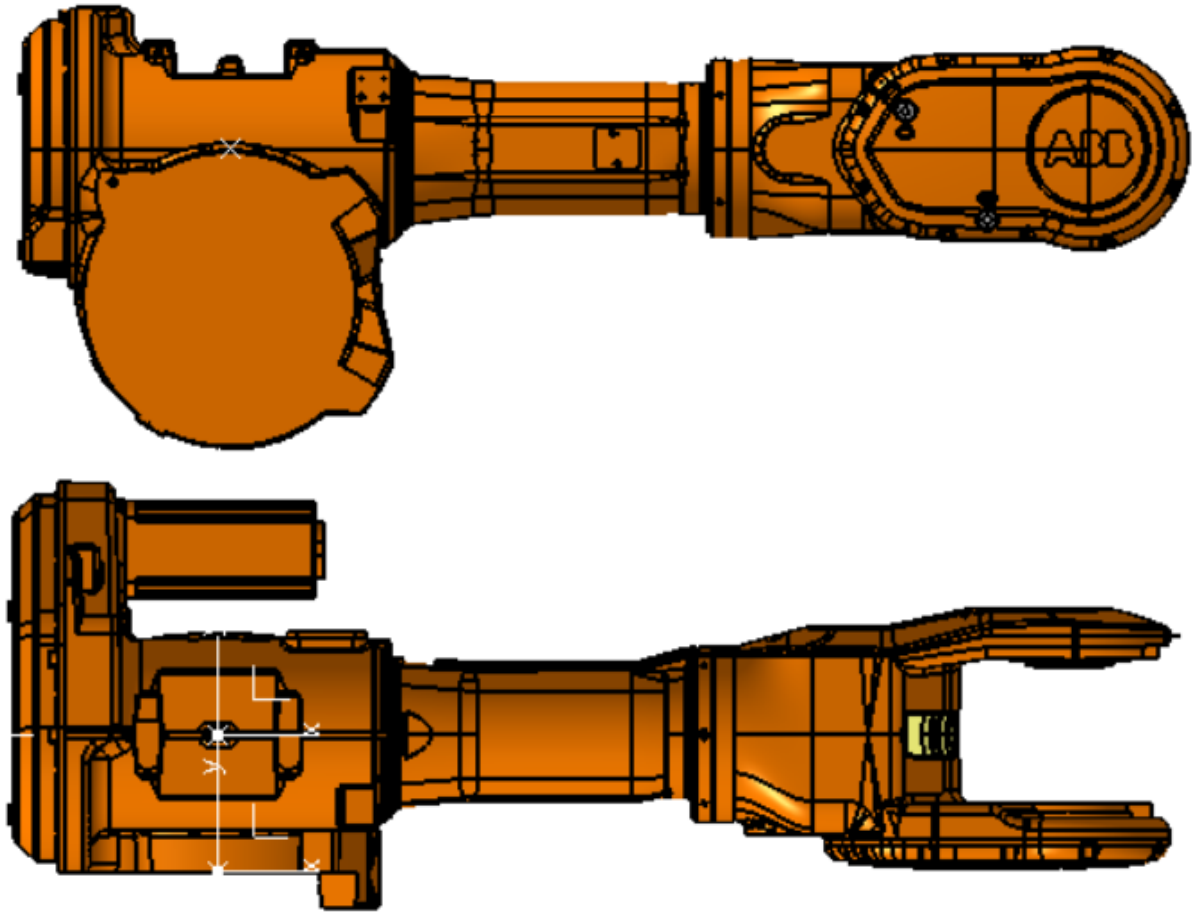
- *lower* – postavlja donju granicu kretanju zgloba (u rad za rotacijske te u m za translacijske zglobove). Za kontinuirano rotacijske zglobove ovaj parametar se izostavlja.
- *upper* - postavlja gornju granicu kretanju zgloba (u rad za rotacijske te u m za translacijske zglobove). Za kontinuirano rotacijske zglobove ovaj parametar se izostavlja.
- *effort* – postavlja gornju granicu naprezanja zgloba
- *velocity* – postavlja gornju granicu brzine zgloba

Postoje još neke oznake i elementi koji se mogu koristiti pri modeliranju robota, ali se prilikom razvoja sustava nisu koristili pa tako neće biti opisani.

4.2.1.2. Razvoj 3D modela za vizualizaciju robota

U prošlom poglavlju pokazan je način na koji se URDF datotekom može razviti vizualni prikaz robota u ROS sustavu. Korištenjem primitivnih modela kocke, valjka i sfere moguće je razviti kompleksne modele, ali to nije uobičajena praksa. Pri razvoju URDF datoteke moguće je učitati kompleksne modele razvijene nekim od dostupnih CAD alata. U ovom slučaju za razvoj 3D CAD modela robota korišten je programski paket *CATIA V5R21*.

CAD modele nije bilo potrebno razvijati ispočetka već su iskorišteni postojeći modeli koji se za robota ABB IRB 6640 mogu naći na službenoj stranici proizvođača ABB [15]. Modele je trebalo prilagoditi kako bi se mogli učitati prilikom izrade URDF datoteke vodeći računa o konvenciji u ROS sustavu o razvoju kinematskog lanca o kojoj će biti nešto više riječi u poglavlju o razvoju same konfiguracijske datoteke. Za sada je dovoljno reći samo kako je modele trebalo prilagoditi na način da se ishodište globalnog koordinatnog sustava poklopi sa koordinatnim sustavom zgloba. Najbolje će se ovo pokazati primjerom spajanja članaka 3 i 4 u zglob što je iz dva pogleda prikazano na slici 25.

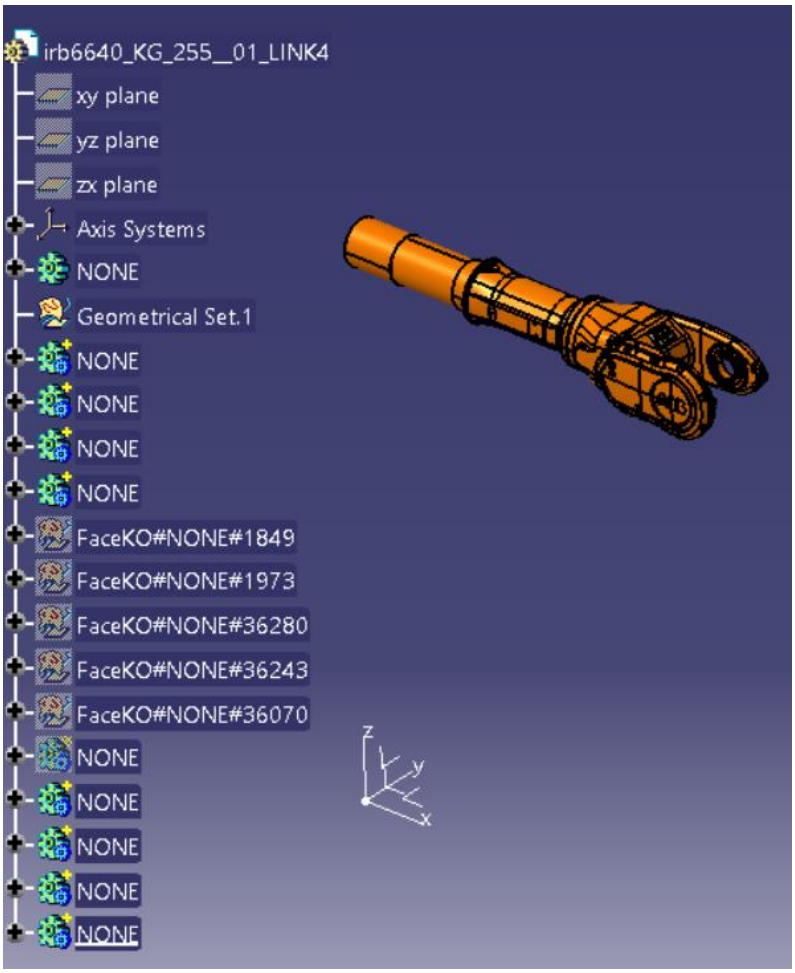


Slika 25 Zglob između članaka 3 i 4 robota ABB IRB 6640

CAD modeli su za preuzimanje dostupni u STEP formatu. STEP (*Standard for the Exchange of Product Data*) format, kako kaže puni naziv formata, uobičajeni je format za distribuciju CAD modela. Prednost ovog formata je što omogućuje otvaranje modela u bilo kojem CAD programskom paketu. Nadalje, spremanjem u STEP formatu model će zadržati visoku točnost

jer se sve značajke spremaju korištenjem matematičkih opisa krivulja i površina za razliku od STL formata koji će kod spremanja čvrsto tijelo (eng. *solid*) pretvoriti u poligonalnu mrežu. S druge strane modele je u programu *CATIA* prilagoditi što nije moguće u ovom formatu. Opis prilagodbe CAD modela robota dan je u ovom poglavlju na primjeru članka 4 robota prikazanog na slici 26.

Otvaranjem STEP modela članka 4 robota u programskom paketu *CATIA* može se po specifikacijskom stablu (slika 26) uočiti kako je članak 4 sačinjen od više *NONE* dijelova te raznih geometrijskih setova. Neki od tih dijelova zaista definiraju geometriju, a neki ne. Kako bi se ostvario ranije spomenuti cilj poklapanja globalnog ishodišta s ishodištem koordinatnog sustava zgloba između članaka 3 i 4, čija je pozicija definirana prema pravilima ROS sustava, potrebno je sve dijelove i geometrijske značajke koja utječu na geometriju članka spojiti u jedan dio.



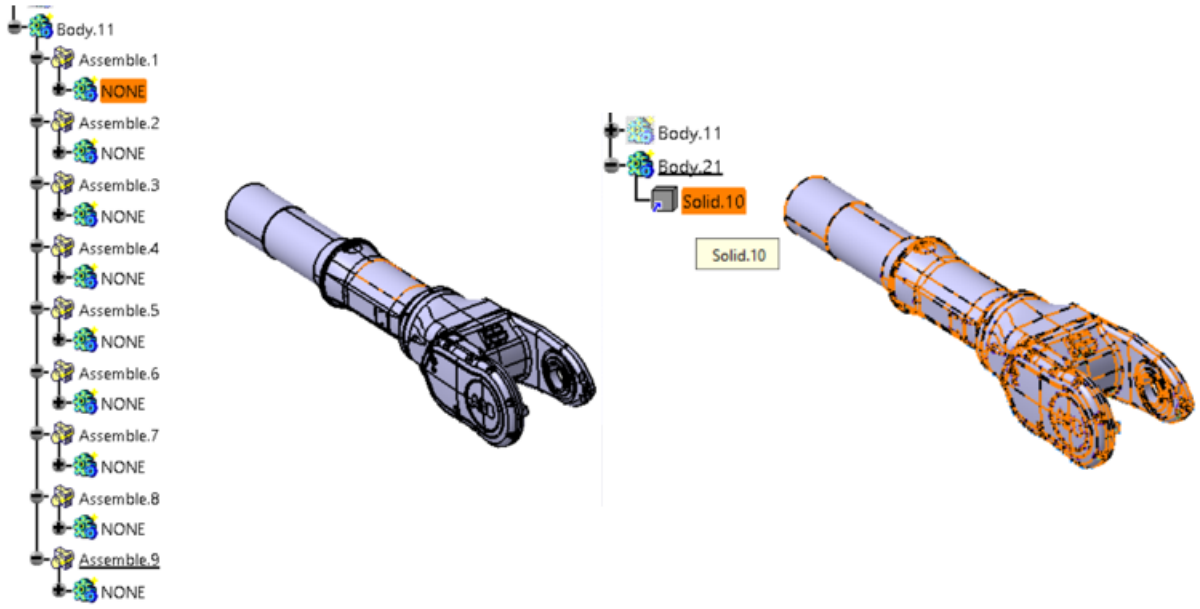
Slika 26 Specifikacijsko stablo članka 4

Prvi korak je dodavanje globalnog koordinatnog sustava. Na traci izbornika odabire se *Insert* i zatim naredba *Axis system*. Pojavljuje se izbornik u kojem se može definirati ishodište koordinatnog sustava, no kako je ovdje potrebno da ishodište bude baš u globalnoj nuli odabire se opcija *Ok* i koordinatni sustav se pojavljuje (Slika 26). Članak 4 ovako je pozicioniran u STEP datoteci jer su pojedini dijelovi robota vjerojatno spremljeni u tom formatu nakon definiranja sklopa svih dijelova robota. Sklapanje svih dijelova u sklop u programskom paketu *CATIA* zaista automatski postavlja dijelove na pravo mjesto u sklopu, ali to isto ne bi se dogodilo u ROS sustavu.

Zatim je potrebno na kraj specifikacijskog stabla dodati još jedan dio. Desnim klikom miša na vrh definicijskog stabla otvara se izbornik i odabire se naredba *Define In Work Object* koja osigurava da dio bude dodan na kraj, odnosno dno stabla. Zatim se ponovno odabire tipka *Insert* na alatnoj traci te naredba *Body*. Na dnu stabla pojavljuje se novi dio *Body.11*.

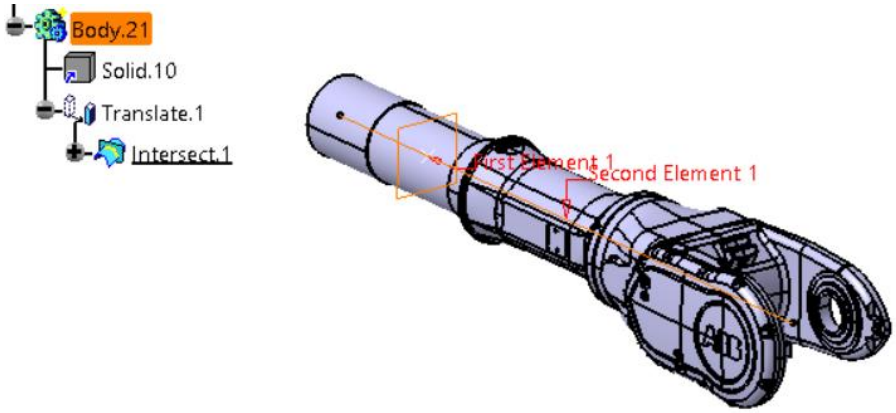
Sljedeći korak je dodavanje dijelova i značajki članka 4 ovdje nazvanih *NONE* jedno po jedno u sklop unutar dodane značajke *Body.11*. Prvo se utvrdi koje od svih značajki imaju utjecaj na geometriju članka i te značajke dodaju se u sklop dok se ostale jednostavno zanemaruju i odmah uklanjaju naredbom *Hide*. Desnim klikom miša na pojedinu značajku *NONE* može se odabrati naredba *Copy*. Zatim se desnim klikom miša na vrh stabla može odabrati naredba *Paste Special…* koja otvara istoimeni izbornik. U izborniku, odabire se opcija *As Result With Link*. Na taj način kopirana značajka pojavljuje se na dnu stabla što omogućuje dodavanje te novonastale značajke u *Body.11*. Nakon kopiranja pojedina značajka mijenja boju u sivu pa se i na taj način može pratiti koje je značajke još potrebno dodati u sklop.

Dodavanje kopiranih značajki u sklop unutar dijela *Body.11* izvršava se odabirom tipke *Insert* te zatim naredbe *Boolean Operations* te podnaredbe *Assemble*. Otvara se izbornik *Assemble* u kojem je potrebno odabrati dio koji se dodaje (*NONE*), dio kojemu se dodaje (*Body.11*) te pozicija u sklopu nakon koje se dodaje odabrani dio. Ovaj proces ponavlja se za svaki dio sve dok cijela geometrija članka 4 nije sastavljena u dijelu *Body.11*. U tom trenutku svi dijelovi sastavljeni u novom dijelu još uvijek nisu spojeni u jednu cjelinu. Da bi se to ostvarilo potrebno je kopirati novo tijelo *Body.11*, desnim klikom miša na vrh specifikacijskog stabla odabrati naredbu *Paste Special* te zatim opciju *As Result With Link* nakon čega naposlijetku nastaje jedno čvrsto tijelo (eng. *solid*) *Body.21*. Slika 27 prikazuje usporedno razliku u specifikacijskom stablu dijela prije i nakon posljednjeg koraka

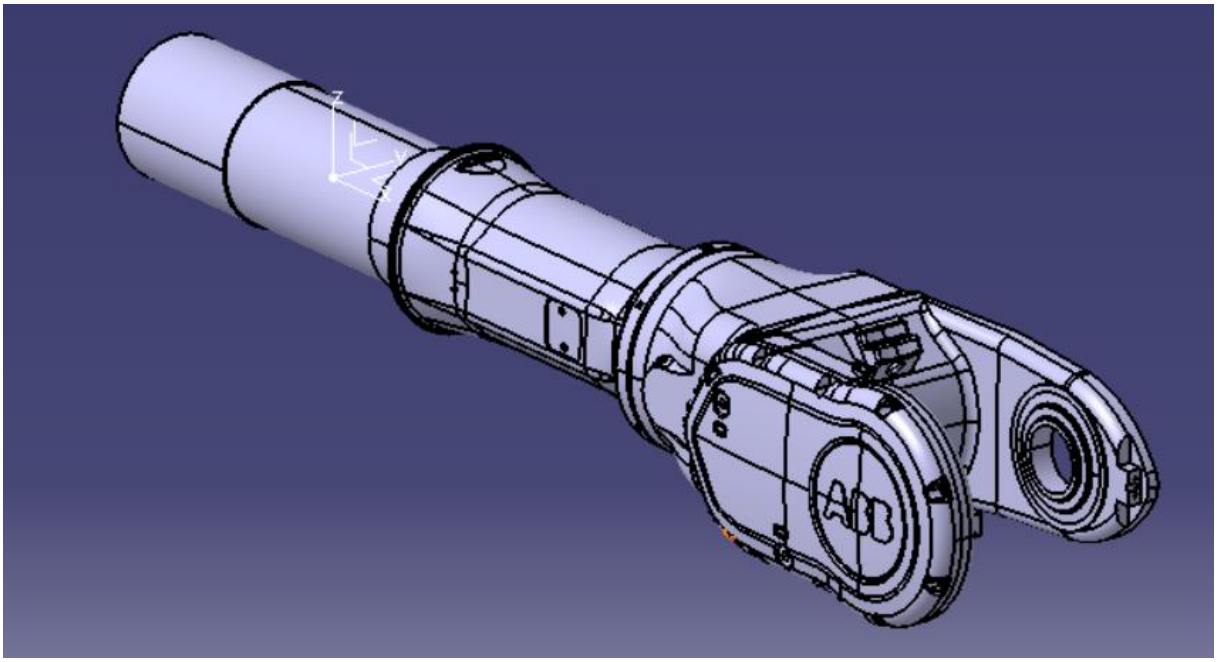


Slika 27 Usporedba dijelova *Body.11* i *Body.21*

Na kraju je potrebno translatirati novonastalo čvrsto tijelo članka 4 na način da se ishodište koordinatnog sustava zgloba između članaka 3 i 4 (točka prikazana slikom 25) poklopi sa ishodištem globalnog koordinatnog sustava. Odabire se naredba *Translation* te se otvara izbornik *Translation Definition*. Ovdje se odabire vektorska definicija translacije točka-točka (eng. *Point to point*). Prva točka definira se kao sjecište uzdužne osi članka 4 te ravnine čija je ta os normala (slika 28). Druga točka je globalno ishodište. Odabirom tipke *Ok* izvršava se translacija i prilagodba modela za ROS sustav je gotova. Konačna pozicija članka 4 s obzirom na globalni koordinatni sustav prikazana je na slici 29. Na kraju je još samo potrebno model spremiti u STL formatu koji je kompatibilan s ROS-om. Ovaj postupak uspješno je ponovljen za sve dijelove robota.



Slika 28 Definiranje točke za translaciju



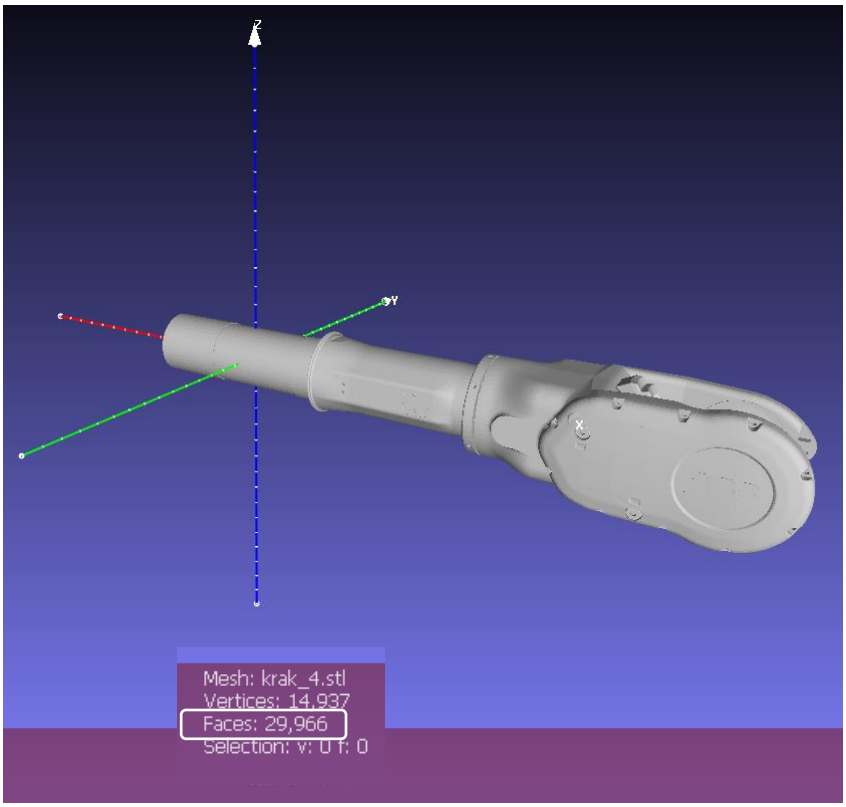
Slika 29 Konačna pozicija članka 4

4.2.1.3. Razvoj 3D modela za provjeru sudara

Već je spomenut razlog zašto se u ROS sustavu modeli za vizualizaciju mogu razlikovati od modela za provjeru sudara. Kolizijski modeli visoke kvalitete mogli bi opteretiti procesor toliko da se čak ugrozi izvršavanje procesa generiranja i izvršavanja valjanje trajektorije. Potrebna je dakle u tom smislu prilagodba 3D modela kako bi se optimiralo proces.

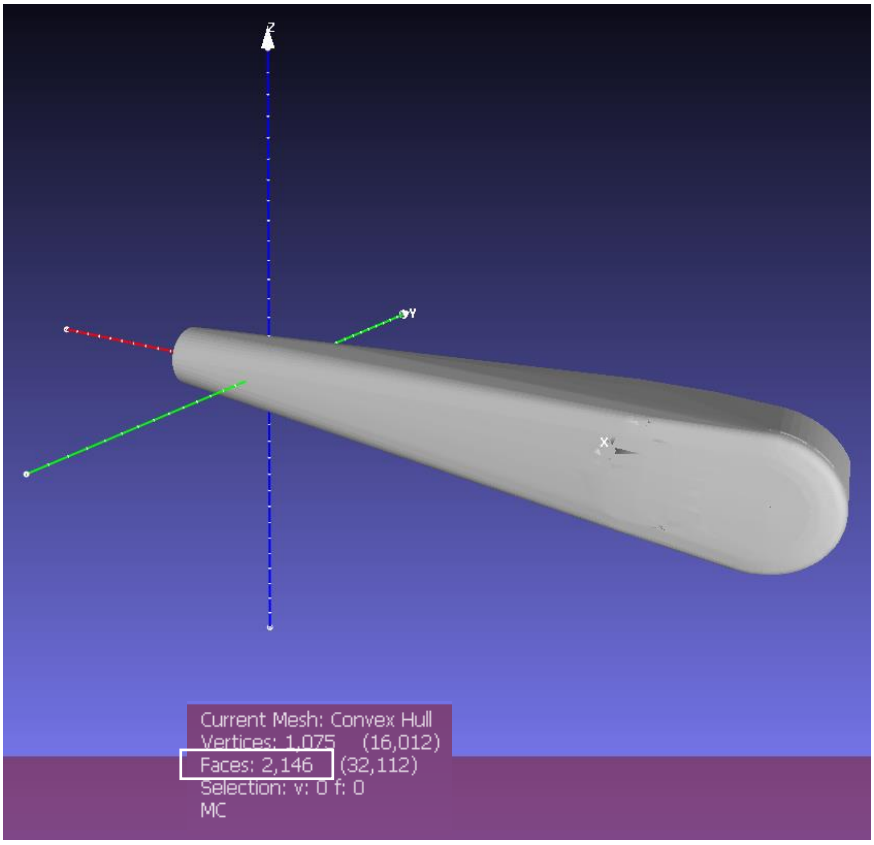
Modeli za provjeru sudara trebaju biti što jednostavniji, ali moraju obujmiti cijelu geometriju članka. Za razvoj modela za provjeru sudara odabran je programski paket *MeshLab 2020.12*. Ovaj softver nudi alate koji mogu prilagoditi poligonalnu mrežu STL formata izvršavanjem samo jedne naredbe.

Za primjer će se ponovno uzeti razvoj modela članka 4. Kada je model razvijen i spremljen u STL formatu može se otvoriti u programskom paketu *MeshLab 2020.12.* Slika 30 pokazuje kako model prije prilagodbe ima skoro 30 000 površina. To je puno više od preporučene brojke. Prema smjernicama na [6] poželjno je da model za provjeru sudara ima manje od 1000 površina.



Slika 30 Model članka 4 u programskom paketu *MeshLab* prije prilagodbe

Ponovno postavljanje poligonalne mreže STL datoteke (eng. *remeshing*) izvršava se odabirom tipke *Filters* na alatnoj traci te zatim odabirom na skup naredbi *Remeshing, Simplification and Reconstruction* i na kraju odabirom naredbe *Convex Hull.*



Slika 31 Model članka 4 u programskom paketu *MeshLab* nakon prilagodbe

Slika 31 prikazuje model članka 4 nakon prilagodbe. Vidi se kako se broj površina smanjio sa skoro 30 000 na nešto više od 2 000 površina. Ovo će znatno ubrzati proces provjere sudara i skratiti vrijeme odziva programa. Isti postupak ponovljen je za sve modele pojedinih dijelova robota.

4.2.1.4. Razvoj URDF konfiguracijske datoteke robota

3D modeli svih dijelova robota u STL formatu su nakon prilagodbe preko dijeljenog direktorija učitani s *Windows* sustava na sustav *Linux* i pohranjeni u direktorij *Meshes.* Prilagodba konfiguracijske datoteke izvršena je prema službenim uputama na [6] te po primjeru dostupnih konfiguracijskih datoteka ROS-I repozitorija odnosno ABB paketa.

Cijela konfiguracijska datoteka prilično je opširna pa će u ovom dijelu rada biti dan primjer razvoja konfiguracije na temelju četvrtog članka robota (*link_4*) te zglobova 1 i 2. Logika razvoja konfiguracije za ove elemente primjenjuje se na sve ostale pa neće biti dan opis kompletne datoteke. Kompletna URDF konfiguracijska datoteka dana je u Prilogu.

Datoteka započinje definiranjem verzije XML jezika koja se koristi te elementom <robot> koji je nadređen svim ostalim elementima. Unutar elementa <robot> definira se parametar *name* u koji se upisuje ime robota (abb_irb_6640_235_255).

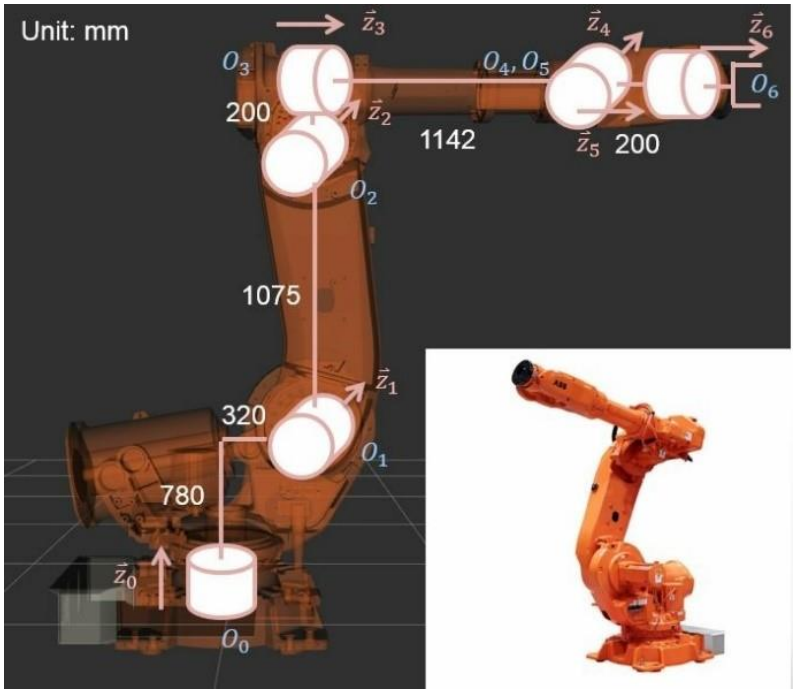
Zatim slijedi definiranje svih članaka robota elementom <link> za svaki pojedini članak. Ovdje se daje primjer samo za članak 4, ali svi ostali članci definirani su analogno tome.

```
<link name="link_4">
  <collision name="collision">
      <geometry>
      <mesh filename="package://abb_irb6640_support/meshes/irb6640_235_255/collision/krak_4.stl"
      scale="0.001 0.001 0.001"/>
      </geometry>
  </collision>
  <visual name="visual">
      <geometry>
      <mesh filename="package://abb_irb6640_support/meshes/irb6640_235_255/visual/krak_4.stl"
      scale="0.001 0.001 0.001"/>
      </geometry>
      <material name="yellow"/>
  </visual>
</link>
```

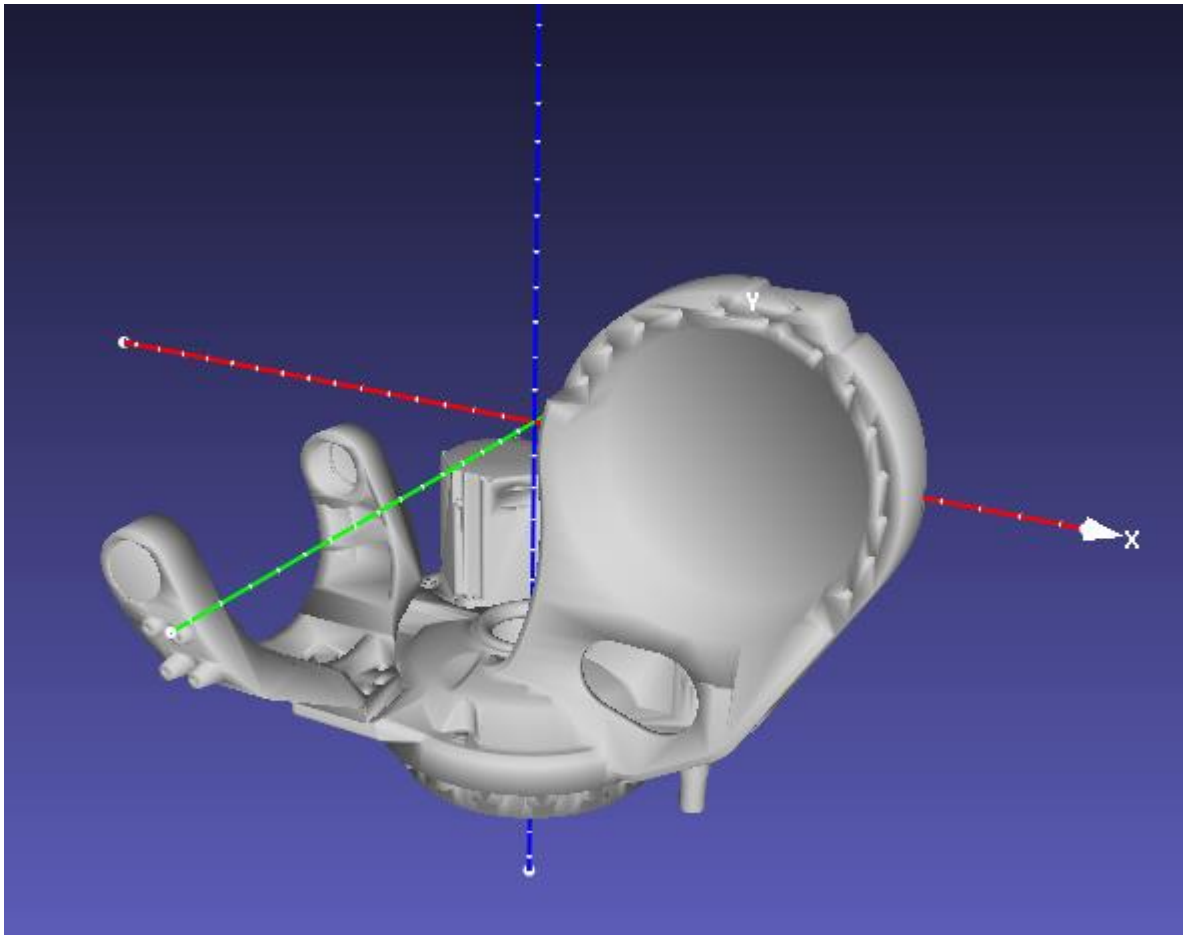
Upisivanjem parametra *name* unutar elementa <link> definira se ime članka kao *link_4.* Slijedi definiranje geometrije za provjeru sudara. Geometrija je definirana razvijenim modelom (Slika 31) te postavljanjem putanje do STL modela unutar paketa *abb_irb6640_support*. Isto tako definirana je i geometrija za vizualizaciju članka. Prvotno su STL modeli bili postavljeni bez parametra *scale* i tada se pri pokušaju vizualizacije nije uspjelo u *RViz* okolini prikazati model robota. No, model robota bio je prikazan samo što je bio toliko uvećan da ga se nije primijetilo. Nakon postavljanja parametra *scale* na vrijednost 0.001 po sve tri osi problem je riješen. Uzrok problema bio je u tomu što su u ROS-u prihvaćene sve službene mjerne jedinice SI sustava pa se tako ovdje svaka duljina izražava u m. U programskom paketu *CATIA* to nije tako, već je tamo kao glavna mjerna jedinica za duljinu postavljen mm.

Nakon što su definirani svi članci pristupilo se definiranju zglobova između pojedinih članaka robota. Svaki je zglob određen člankom robota koji se naziva roditeljem te člankom koji se naziva djetetom. Ishodište koordinatnog sustava zgloba postavlja se u odnosu na članak

koji je roditelj. Uz ovaj dogovor postoji još jedna konvencija koje se valja držati prilikom izrade konfiguracijskih datoteka robota, a ta je da se kinematski lanac robota mora razviti na način da transformacija iz jednog koordinatnog sustava zgloba u drugi nikad nije po dvije osi istovremeno. To ROS alatu *tf* olakšava izračun unaprijedne i inverzne kinematike i predstavlja još jedan način optimiranja programa. Ovo je prikazano slikom 32 na kojoj se vidi kinematski lanac robota, a bit će objašnjeno i na temelju konfiguracijske datoteke definiranjem zgloba 1 i zgloba 2.



Slika 32 Kinematski lanac robota [19]



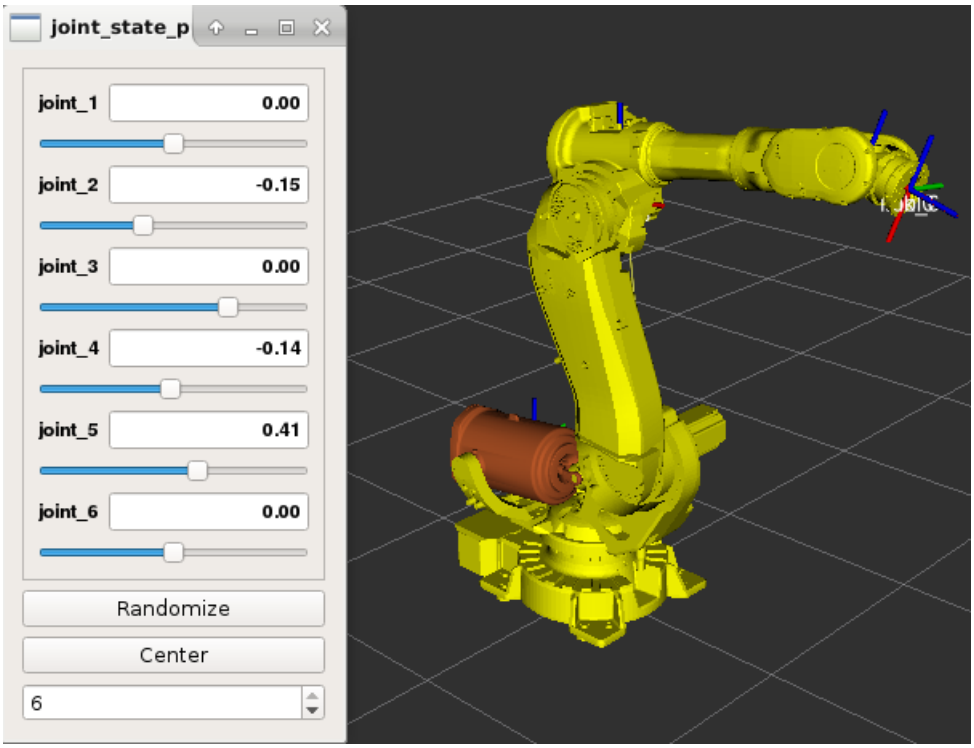
Slika 33 Pozicija koordinatnog sustava članka 1

```
<joint name="joint_1" type="revolute">
   <origin rpy="0 0 0" xyz="0 0 0.780"/>
   <axis xyz="0 0 1"/>
   <parent link="base_link"/>
   <child link="link_1"/>
   <limit effort="0" lower="-2.967" upper="2.967" velocity="1.7453"/>
</joint>
<joint name="joint_2" type="revolute">
   <origin rpy="0 0 0" xyz="0.320 0 0"/>
   <axis xyz="0 1 0"/>
   <parent link="link_1"/>
   <child link="link_2"/>
   <limit effort="0" lower="-1.134" upper="1.4855" velocity="1.5707"/>
</joint>
```

Kao što je vidljivo na slici 33 i u elementu <origin> zgloba 1, iako je zglob između baze i članka 1 robota ostvaren nešto niže, u konfiguracijskoj datoteci, koordinatni sustav zgloba pomaknut je 0.78 m u odnosu na koordinatni sustav baze. 3D model članka 1 tako je i razvijen da mu se koordinatni sustav nalazi baš u toj točki. Položaj koordinatnog sustava članka 1 prikazan je na slici 33. Na taj način omogućena je jednostavnija transformacija iz koordinatnog sustava članka 1 u koordinatni sustav zgloba 2, odnosno članka 2.

Osim parametara članaka roditelja te djeteta pri definiranju elementa <joint> još se definiraju os rotacije podređenim elementom <axis> i ograničenja kretanja pojedinog zgloba koja su zadana specifikacijama proizvođača koje su dostupne u tablici 1 te tablici 2. Analogno načinu na koji su definirana prva dva zgloba definiraju se svi ostali zglobovi robota.

Nakon razvoja konfiguracijske datoteke njena funkcionalnost ispitana je pokretanjem datoteke za pokretanje *test_irb6640_235_255.launch* koja učitava razvijenu konfiguracijsku datoteku i u *RViz-u* realizira model robota. Osim toga pokreću se još dva čvora - *joint_state_publisher* i *robot_state_publisher*. Čvor *joint_state_publisher* otvara prozor u kojem je moguće klizačima interaktivno pokretati zglobove robota što se vidi na slici 34. Čvor *robot_state_publisher* šalje u *RViz* stanja zglobova robota.



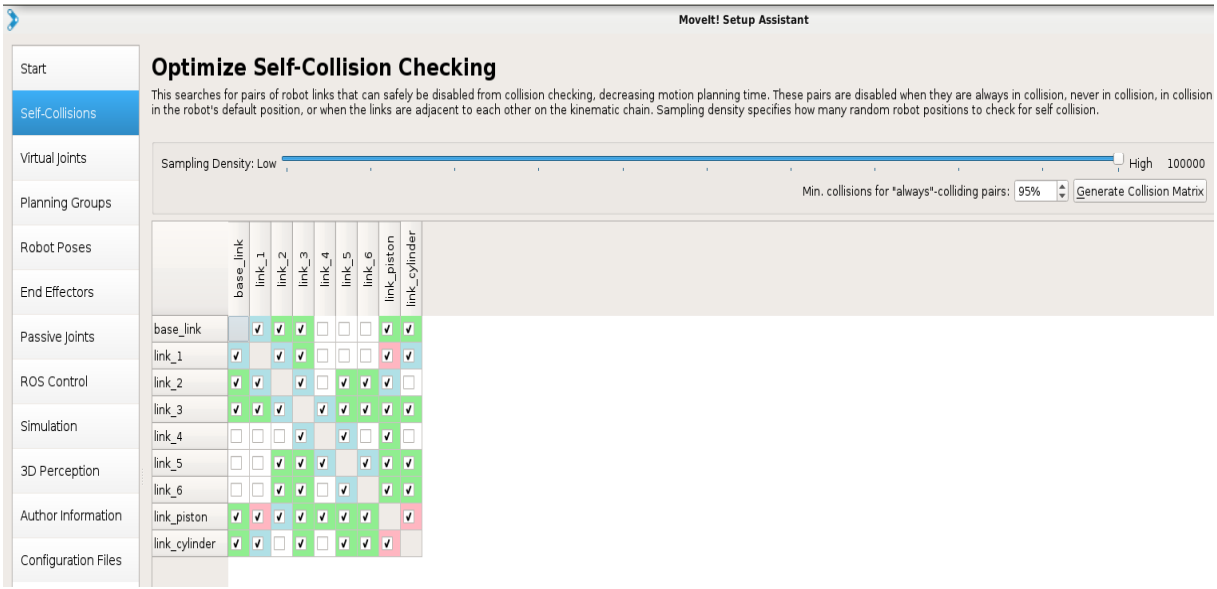
Slika 34 Uspješna vizualizacija modela robota u *RViz-u*

4.2.2. Razvoj MoveIt! konfiguracijskog paketa robota

Nakon što je potvrđena funkcionalnost URDF konfiguracijske datoteke robota pristupilo se razvoju *MoveIt!* konfiguracijskog paketa robota kako bi se omogućilo vođenje robota. Kako bi se izbjeglo ručno pisanje paketa i svih datoteka za pokretanje, *MoveIt!* nudi paket - *MoveIt! Setup Assistant* koji pomaže u razvoju SRDF (*Semantic Robot Description Format*) datoteke robota, različitih YAML konfiguracija kao i *roslaunch* datoteka potrebnih za generiranje trajektorije. Ovaj paket pokreće se iz terminala naredbom *rosrun moveit_setup_assistant* otvara se *MoveIt! Setup Assistant* prozor (Slika 35). Prvo što je potrebno jest izabrati hoće li se raditi razvoj novog *MoveIt!* konfiguracijskog paketa ili uređivanje već postojećeg. Odabire se opcija izrade novog paketa s obzirom da paket za robota ABB IRB 6640 nije pronađen u repozitoriju ROS-a, odnosno ROS-Industriala.

Slika 35 ***MoveIt! Setup Assistant***

Sljedeći korak je učitavanje prethodno razvijene URDF datoteke robota. Pritiskom na opciju *Browse* moguće je odabrati željenu datoteku unutar radnog prostora *catkin_ws.* Nakon uspješnog učitavanja URDF datoteke pojavljuje se vizualizacija modela robota kao potvrda. Nakon toga slijedi korak optimizacije provjere kolizije robota sa samim sobom (eng. *self-collision*). Ova opcija omogućuje optimizaciju procesa provjere kolizije, a tako i planiranja trajektorije. Program će simulirati robota u određenom broju nasumičnih pozicija u prostoru (uzimajući u obzir ograničenja kretanja zglobova iz URDF-a) te će na temelju toga predložiti parove članaka robota za koje se provjera samokolizije može isključiti. Gustoća uzorkovanja, odnosno broj nasumičnih pozicija može se podešavati (Slika 36). Pritiskom na tipku generirat će se matrica kolizije dijelova. Dva robotska članka mogu biti okarakterizirana kao *uvijek u kontaktu, nikad u kontaktu* ili *u kontaktu prema zadanim postavkama* ukoliko je tako zadano kinematskim lancem definiranim URDF datotekom.



Slika 36 Optimizacija provjere sudara u programskom paketu *MoveIt!*

Odabirom na sljedeću karticu, *Virtual Joints* otvara se prozor u kojem je potrebno definirati vezu robota i okoline. U *MoveIt!*-u to se radi na način da se definira virtualni nepomični zglob između koordinatnog sustava dijela robota koji je prvi u kinematskom lancu robota i okoline. U ovom slučaju dio robota koji je osnovni i na koji se vežu svi ostali jest baza robota i s tim dijelom se ostvaruje virtualni zglob. Osim nepomičnim zglobom vezu s okolinom moguće je ostvariti i planarnim zglobom koji ima tri stupnja slobode gibanja kojim se s okolinom povezuju mobilni roboti te plutajućim zglobom koji dopušta šest stupnjeva slobode gibanja i kojima su s okolinom povezani leteći roboti (dronovi).

Nakon ostvarivanja veze s okolinom potrebno je na sljedećoj kartici, *Define Planning Groups,* definirati skupine robotskih članaka za koje će se vršiti planiranje trajektorije. U ovom slučaju postoji samo jedna skupina i to je cijeli manipulator, ali ova opcija korisna je za robote koji imaju više ovakvih skupina, primjerice humanoidni roboti. Odabirom na opciju *Add Group* može se nova skupina dijelova robota za planiranje. Otvara se novi prozor gdje je potrebno navesti ime za skupinu koju se definira te zatim izabrati algoritam za numeričko rješavanje kinematskog problema (solver). Odabire se *KDL Kinematics Plugin* koji je zadani solver *MoveIt!* programa. Zadnji korak je definiranje skupine što je učinjeno definiranjem kinematskog lanca od početnog (baze) do završnog članka (tool0). Ovo je zadnji korak u razvoju paketa koji je obavezan, bez kojeg *Setup Assistant* ne može kreirati paket. Od

opcionalnih postavki program nudi postavljanje konfiguracijske datoteke za simulaciju robota u programu *Gazebo*. S obzirom da je odlučeno kako će simulacija robota biti izvršena u ABB-ovom programskom paketu *RobotStudio,* ovaj korak je preskočen.

Na kraju je još potrebno samo upisati informacije o autoru paketa. Pritiskom na tipku *Generate Package* program će izgraditi paket te velik broj različitih konfiguracijskih datoteka i datoteka za pokretanje. U ovom trenutku moguće je provjeriti funkcionalnost paketa planiranjem trajektorije za robota vizualiziranog u *RViz-u* što je i učinjeno pozivanjem naredbe *roslaunch abb_irb6640_moveit_config demo.launch*. No, da bi se ostvarilo slanje trajektorije robotu, stvarnom ili simuliranom potrebno je razviti još neke datoteke.

Prvo je potrebno kreirati *controllers.yaml* datoteku koja će definirati *action_sever* i tip poruke koja će se koristiti za upravljanje robota, a zatim prilagoditi *abb_irb6640_moveit_controller_manager.launch.xml* datoteku tako da učita parametre postavljene *controllers.yaml* datotekom u ROS server parametara. Na kraju potrebno je razviti datoteku za pokretanje (eng. *launch file*) *moveit_planning_execution.launch* koja će pokrenuti sve potrebne čvorove uz pozivanje ostalih datoteka potrebnih za vizualizaciju modela robota te komunikaciju *MoveIt!* programa s robotom. Sada je paket spreman za korištenje u kombinaciji s robotom.

4.3. Realizacija vođenja robota

ROS-Industrial svojim paketima omogućuje komunikaciju računala, na kojem je instaliran ROS sustav, i robota. Važno je istaknuti kako ROS niti u jednom trenutku ne zamjenjuje kontroler robota, već samo ostvaruje komunikaciju i generira putanju. Upravljanje svih motora robota i dalje se odvija unutar sustava robota i u njemu se ništa ne mijenja, već integrira sa ROS sustavom. Može se reći kako ROS omogućuje drugačiju vrstu interakcije korisnika s kontrolerom robota.

Realizacija vođenja robota ABB IRB 6640 odvila se u nekoliko faza. Prvo je ROS sustav prilagođen za vođenje ABB robota i ispitan interno korištenjem modela robota vizualiziranog u programima *MoveIt!* i *RViz.* Ovo je poslužilo kao potvrda ispravnosti razvijenih ROS paketa i datoteka za pokretanje ROS komunikacije. U drugoj fazi cilj je bio ostvariti vođenje robota u simulaciji kako bi se provjerila funkcionalnost prilagođenog ROS sustava u interakciji s kontrolerom robota. Tek kada je u simulaciji potvrđena pouzdanost ROS sustava vođenje robota ostvareno je na fizičkom robotu.

Nakon što je ROS sustav u potpunosti prilagođen za komunikaciju s robotom potrebno je kontroler robota podesiti za interakciju s ROS-om. U ovom poglavlju bit će opisan taj postupak u vođenju simuliranog robota. Virtualni kontroler u programu *RobotStudio* istovjetan je stvarnom kontroleru robota tako da ovaj postupak neće biti ponovo objašnjen i za slučaj vođenja stvarnog robota.

4.3.1. Vođenje robota izvedeno u simulaciji

Program *RobotStudio* instaliran je na računalu s operacijskim sustavom *Windows10,* dok je ROS pokretan preko virtualnog hardvera na operacijskom sustavu *Linux Debian Stretch.* Veza se ostvaruje putem mrežnog adaptera računala koji je postavljen prilikom inicijalizacije virtualnog hardvera.

Prvi korak u realizaciji vođenja simuliranog robota bilo je postavljanje robota, odnosno virtualnog kontrolera u programu. Odabirom na tipku *Virtual Controller*, na alatnoj traci programa *RobotStudio,* otvara se novi prozor u kojem je potrebno postaviti ime kontrolera te model robota. Potrebno je odabrati model IRB 6640 235kg 2.55m. Isto tako bitno je prije postavljanja označiti polje *Customize* kako bi se za kontroler mogle uključiti opcije:

- *623-1*: *Multitasking*
- *616-1*: *PC Interface*

potrebne za izvršavanje pozadinskih procesa te ostvarivanje veze s osobnim računalom. Odabirom na tipku *Ok* robot se učitava i stanica je spremna za rad.

4.3.1.1. Konfiguracija kontrolera IRC5

Prvi korak u ostvarenju komunikacije ABB kontrolera s ROS-om je instalacija ROS server koda na kontroler ABB robota. ROS-Industrial nudi paket *abb_driver* u kojem se nalazi jednostavan ROS driver za ABB industrijske robote pisan u programskom jeziku RAPID. RAPID je programski jezik razvijen za programiranje ABB robota. Ovaj driver je agnostičan prema modelu robota i radi na bilo kojem robotu s IRC5 kontrolerom.

Potrebno je u *HOME* direktoriju robotskog kontrolera stvoriti direktorij *ROS* te zatim u taj direktorij kopirati sve datoteke iz *rapid* direktorija *abb_drivera*. Ovo se na simuliranom kontroleru izvodi vrlo lako. Sa *GitHub* repozitorija [16] na računalo se preuzima *abb_driver* te

se datoteke iz direktorija *rapid* kopiraju u novi *ROS* direktorij. Ove datoteke mogu se podijeliti u dvije skupine:

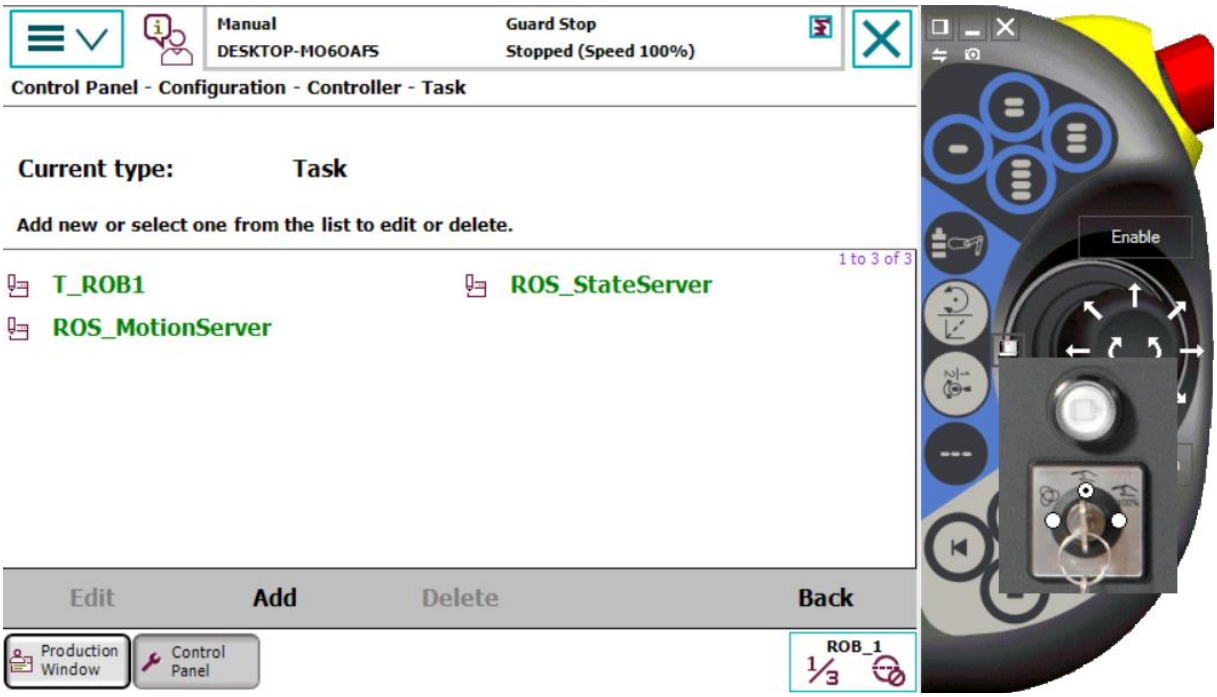
- Datoteke dijeljene među svim procesima
    - ROS_common.sys – globalne varijable i tipovi podataka koji dijele svi
    - ROS_socket.sys – kontrola komunikacija
    - ROS_messages.sys – definira specifične tipove poruka
- Moduli za specifične procese
    - ROS_stateServer.mod – objavljuje stanje zglobova i robota
    - ROS_motionServer.mod – prima naredbe za kretanje robota
    - ROS_motion.mod – robotu izdaje naredbe o kretanju

Kako bi se omogućilo povezivanje *RobotStudio* simulacije s ROS sustavom potrebno je prilagoditi *ROS_socket.sys* datoteku. Naime, funkcija *GetSysInfo()* koja se ovdje koristi trebala bi vratiti valjanu IP adresu, no to se ne događa pa je potrebno tu funkciju zamijeniti IP adresom. No, kako IP adresa nije statična trebalo bi često raditi provjeru IP adrese i ovisno o tome mijenjati ovu vrijednost. Zbog toga je, kako je već spomenuto, prilikom inicijalizacije virtualnog stroja *Linux* postavljen mrežni adapter prema *Windows* računalu. IP adresa mrežnog adaptera je statična pa tako neće biti potrebno pri svakom novom pokretanju *RobotStudia* mijenjati IP adresu u datoteci *ROS_socket.sys.*

Zatim se pristupa podešavanju postavki kontrolera putem virtualnog privjeska za učenje. Potrebno je na kontroleru stvoriti 3 nova procesa (eng. *task*) prikazana u tablici 3. Prije svega kontroler se mora postaviti u ručni način pomicanjem sklopke u srednji položaj (Slika 37). Na kontroleru se redom odabiru tipke *ABB - Control Panel – Configuration – Topics – Controller – Task* te se dolazi do prozora u kojem se jedan po jedan dodaju procesi upisivanjem podataka iz tablice 3. Nakon svakog novog postavljenog procesa kontroler nudi ponovno postavljanje (eng. *reset*), no može se pričekati dok se ne dovrši kompletna konfiguracija te onda napraviti ponovno postavljanje kontrolera. Slika 37 prikazuje prozor *Task* virtualnog kontrolera nakon postavljanja procesa.

Tablica 3 Procesi koji se postavljaju na kontroler robota [6]

Name	Type	Trust Level	Entry	Motion Task
ROS_StateServer	SEMISTATIC	NoSafety	main	NO
ROS_MotionServer	SEMISTATIC	SysStop	main	NO
T_ROB1	NORMAL		main	YES

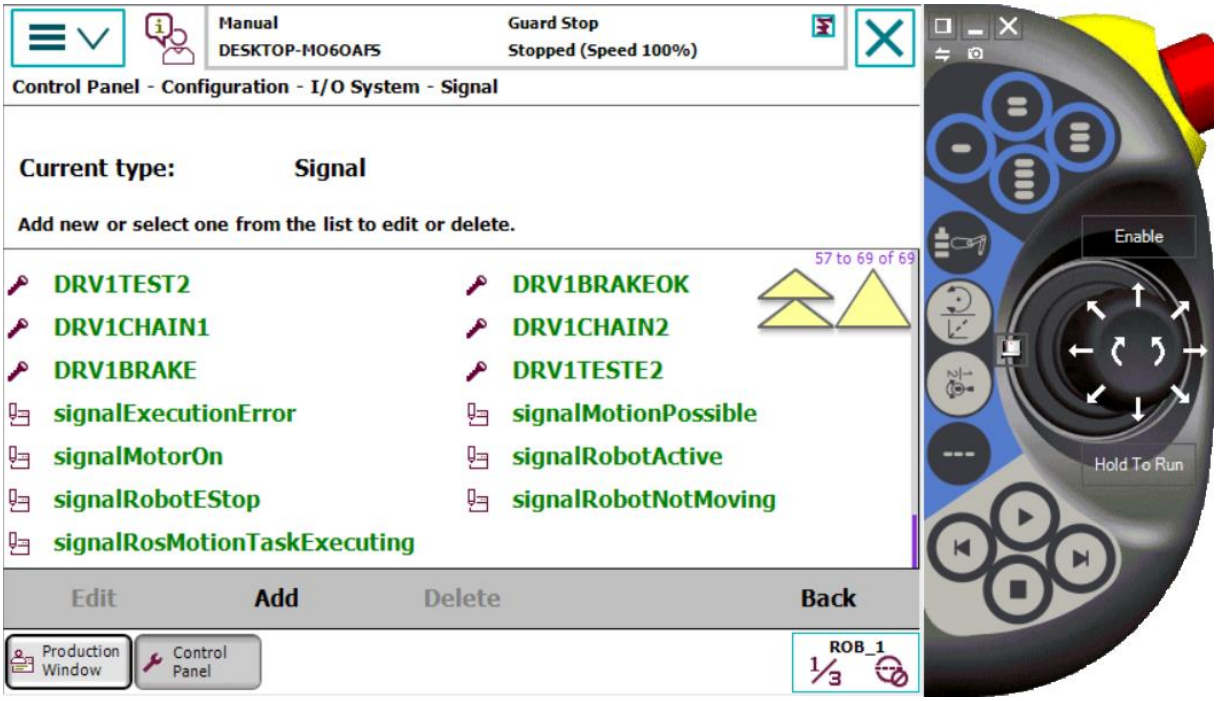


Slika 37 Procesi instalirani na virtualni kontroler

Sada je potrebno stvoriti različite signale u sustavu prema tablici 4. Na kontroleru se redom odabire *ABB – Control Panel – Configuration – Topics – I/O – Signal* gdje se jedan po jedan dodaju signali iz tablice 4. Slika 38 prikazuje prozor *Signal* virtualnog kontrolera nakon postavljanja svih potrebnih signala.

Tablica 4 Signali koji se postavljaju na kontroler robota [6]

Name	Type of Signal
signalExecutionError	Digital Output
signalMotionPossible	Digital Output
signalMotorOn	Digital Output
signalRobotActive	Digital Output
signalRobotEStop	Digital Output
signalRobotNotMoving	Digital Output
signalRosMotionTaskExecuting	Digital Output

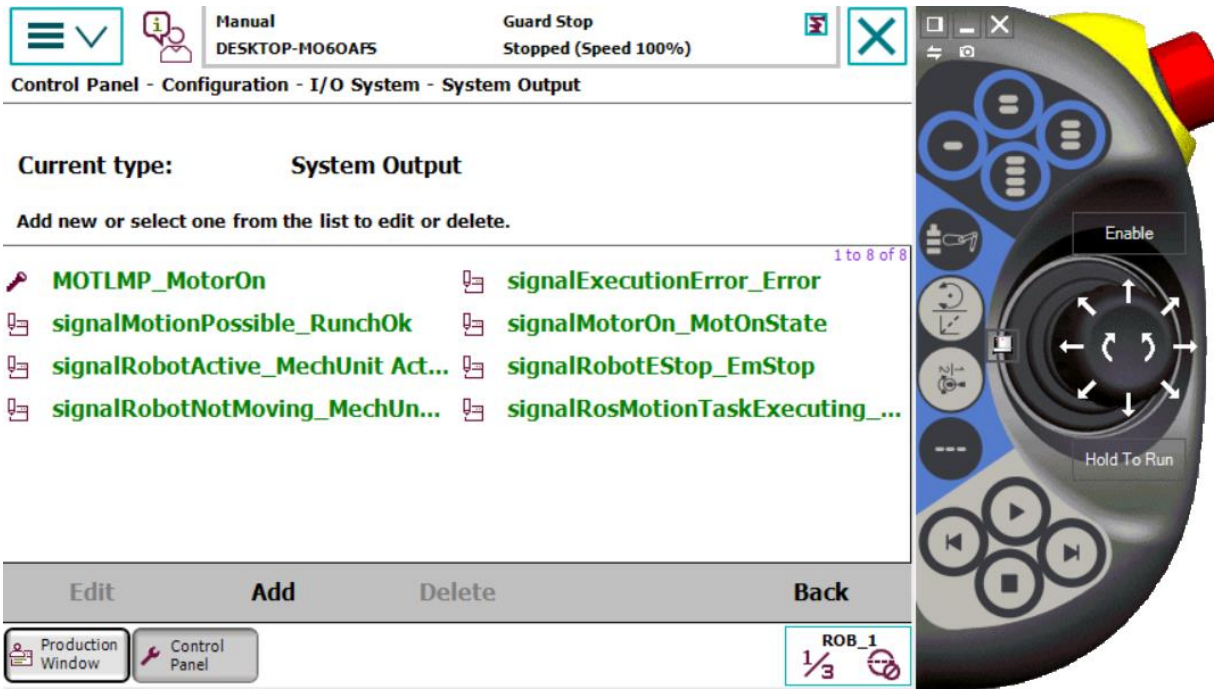


Slika 38 Signali uspješno postavljeni na virtualni kontroler

Nakon postavljanja signala potrebno je svaki od tih signala povezati sa specifičnim izlazima robotskog sustava kako je prikazano tablicom 5. Na virtualnom kontroleru odabire se redom *ABB – Control Panel – Configuration – Topics – I/O – System Output.* Slika 39 prikazuje prozor *System Output* nakon povezivanja signala sa specifičnim izlazima robotskog sustava poput primjerice obavijesti o aktivaciji sigurnosne kočnice (eng. *Emergency Stop*) ili aktivacije motora.

Tablica 5 Signali i pripadajući izlazi sustava [6]

Signal Name	Status	Arg 1	Arg 2	Arg 3	Arg 4
signalExecutionError	Execution Error	-	T_ROB1	-	-
signalMotionPossible	Runchain OK	-	-	-	-
signalMotorOn	Motors On State	-	-	-	-
signalRobotActive	Mechanical Unit Active	ROB_1	-	-	-
signalRobotEStop	Emergency Stop	-	-	-	-
signalRobotNotMoving	Mechanical Unit Not Moving	ROB_1	-	-	-
signalRosMotionTaskExecuting	Task Executing	-	T_ROB1	-	-



Slika 39 Signali povezani sa specifičnim izlazima robotskog sustava

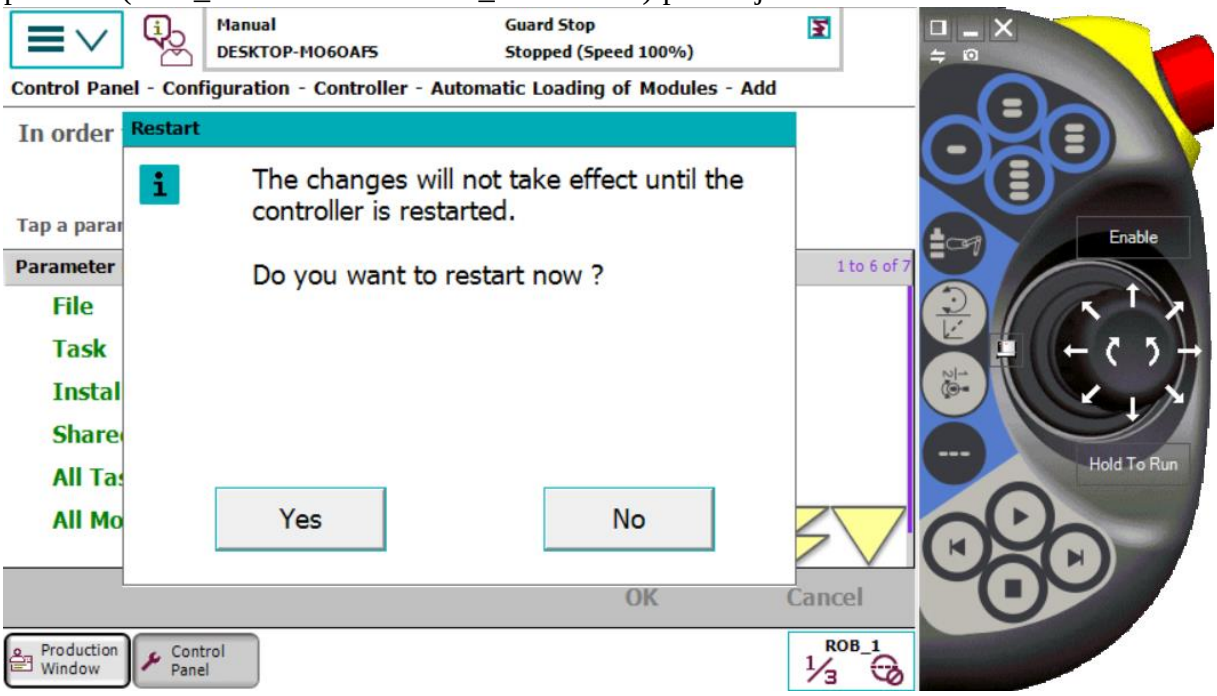
Na kraju je još potrebno postaviti automatsko učitavanje pojedinih modula u postavljene procese prema tablici 6. Na virtualnom kontroleru odabire se redom *ABB – Control Panel – Configuration – Topics – Controller – Automatic Loading of Modules*. Nakon postavljanja automatskog učitavanja zadnjeg modula može se kontroler ponovo postaviti (eng. *reset*) čime je konfiguracija kontrolera za komunikaciju s ROS sustavom završena (Slika 40).

Tablica 6 Učitavanje modula u procese [6]

File	Task	Installed	All Tasks	Hidden
HOME:/ROS/ROS_common.sys		NO	YES	NO
HOME:/ROS/ROS_socket.sys		NO	YES	NO
HOME:/ROS/ROS_messages.sys		NO	YES	NO
HOME:/ROS/ROS_stateServer.mod	ROS_StateServer	NO	NO	NO
HOME:/ROS/ROS_motionServer.mod	ROS_MotionServer	NO	NO	NO
HOME:/ROS/ROS_motion.mod	T_ROB1	NO	NO	NO

4.3.1.2. Pokretanje simuliranog robota

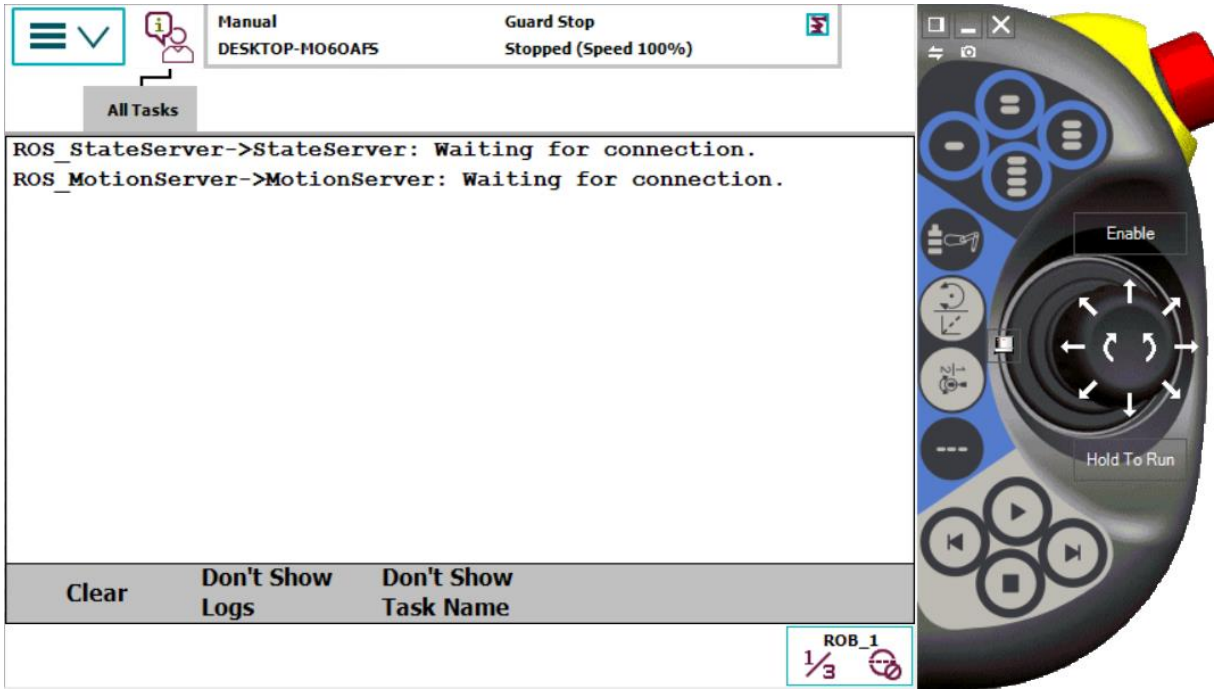
ABB ROS Server sastoji se od tri procesa. Prilikom konfiguracije kontrolera robota dva procesa (ROS_MotionServer i ROS_StateServer) postavljena su kao *SEMISTATIC* što znači



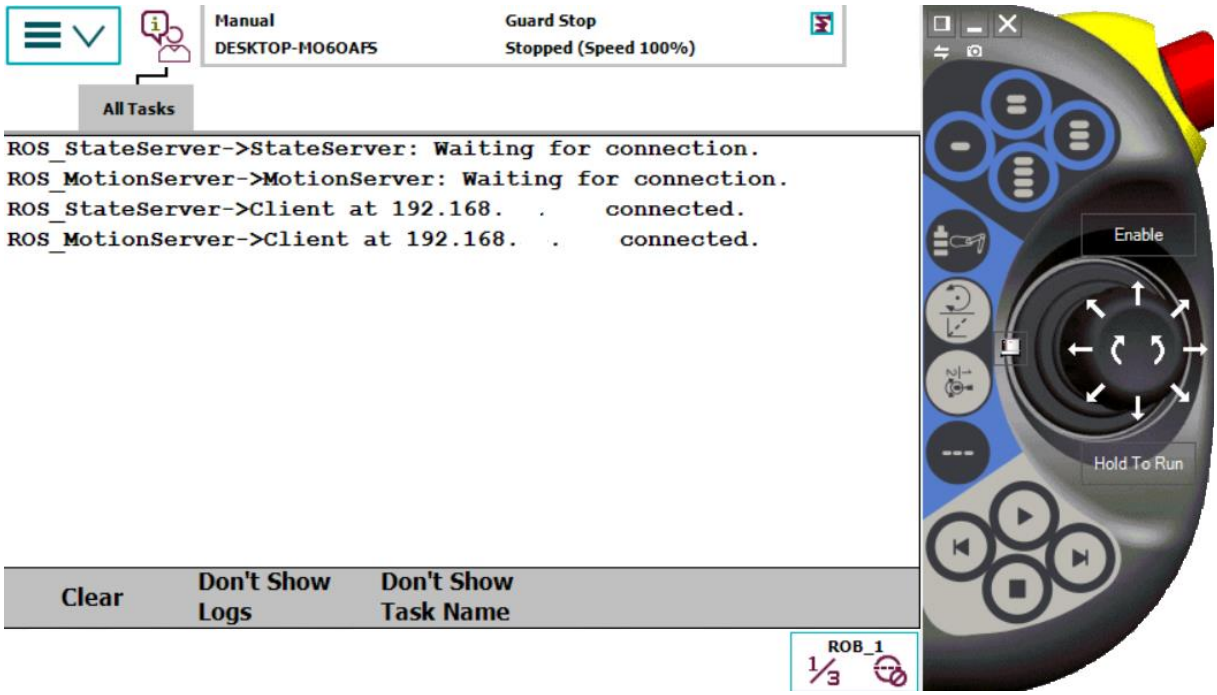
Slika 40 Ponovno postavljanje virtualnog kontrolera

da su pozadinski procesi, a treći proces (T_ROB1) postavljen je kao *NORMAL* i taj je zadužen za izvršavanje zadane putanje. Dva pozadinska procesa nije potrebno pokretati ručno, već se oni pokreću automatski pri podizanju kontrolera. Ovi procesi ne mogu se ručno započeti niti zaustaviti. Ponovo se pokreću svakim ponovnim pokretanjem kontrolera što je vidljivo na slici 41 koja prikazuje pokrenute procese. Kako s druge strane ROS sustav još nije pokrenut ispisuje se poruka na zaslonu virtualnog privjeska za učenje „*Waiting for connection*“. Kako bi ostvarili vezu između ROS sustava i simuliranog robota potrebno je još samo pokrenuti *MoveIt!* okruženje te potrebne čvorove za komunikaciju što se čini naredbom *roslaunch roslaunch abb_irb6640_moveit_config moveit_planning_execution.launch sim:=false robot_ip:=(IP*

adresa mrežnog adaptera). Izvršavanjem naredbe ostvarena je veza između ROS sustava i robota (što se vidi na slici 42) i može se početi zadavati robotu željene pozicije. *MoveIt!* će između trenutne i zadane pozicije isplanirati trajektoriju te ju stvorenim sustavom komunikacije poslati između ROS-a i kontrolera simuliranog robota



Slika 41 Pozadinski procesi ROS_StateServer i ROS_MotionServer prije povezivanja s ROS sustavom

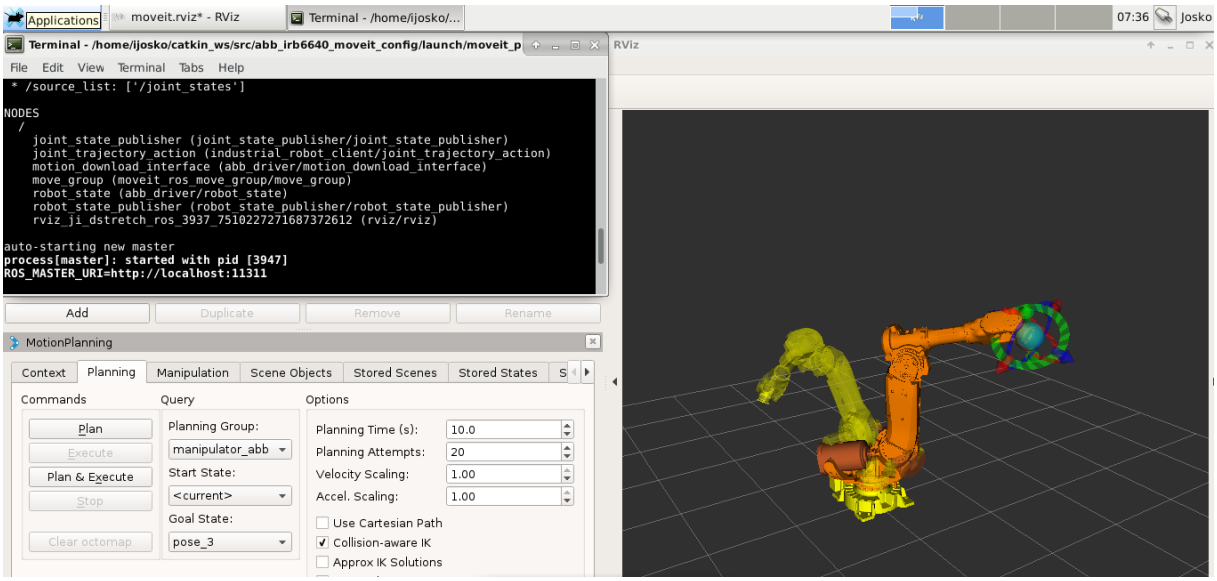


Slika 42 Pozadinski procesi ROS_StateServer i ROS_MotionServer nakon povezivanja s ROS sustavom

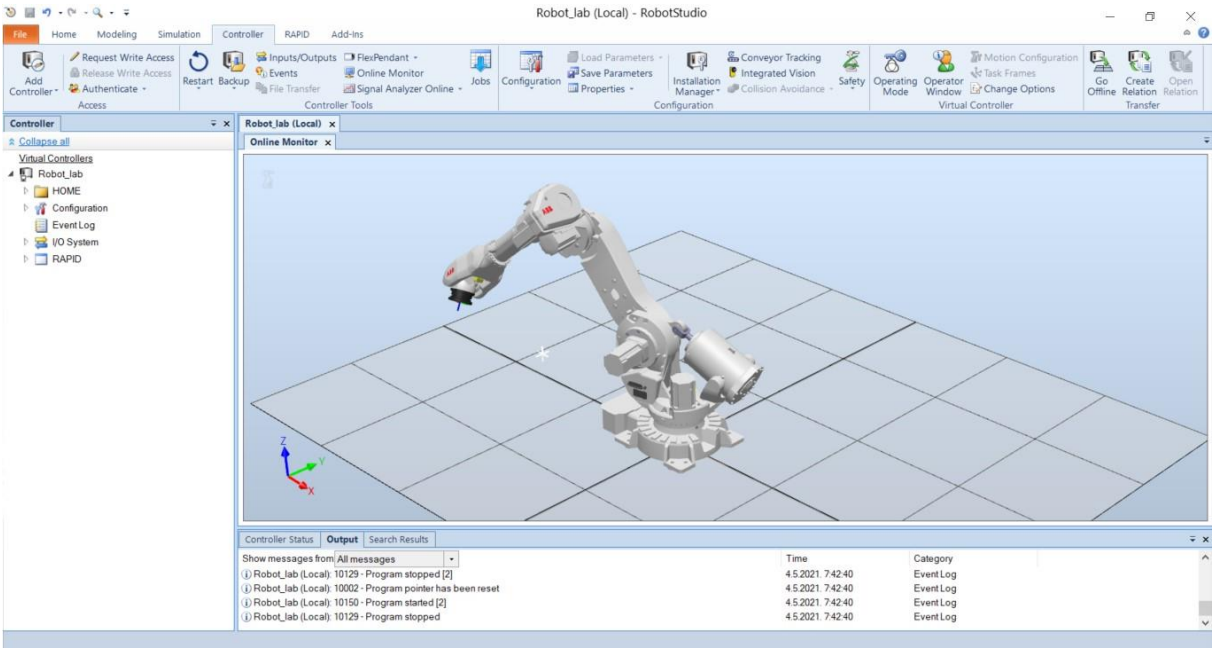
Pozicija simuliranog robota u *RViz*-u prikazana je žutom bojom, a željena pozicija narančastom bojom (slika 43). Gledajući usporedno slike 43 i 44 može se potvrditi da početna pozicija prikazana u *RVizu* zaista je pozicija, u ovom slučaju simuliranog, robota te da sustav zaista funkcionira. Na slici 43 mogu se vidjeti i pokrenuti čvorovi potrebni za realizaciju komunikacije ROS sustava s virtualnim kontrolerom. Željenu poziciju može se zadavati na više načina. Prvi način je postavljanje pozicije povlačenjem pokazivača vidljivog na slici 43 u obliku kugle. Drugi način je postavljanje željene pozicije u prozoru *Motion Planning*. Odabirom na *Goal State* izbornik prikazuju se različite opcije poput nasumične valjane pozicije (eng. *Random Valid*). Odabirom naredbe *Plan, MoveIt!* će generirati trajektoriju i poslati ju robotu no ona se još neće izvršiti. Tek odabirom naredbe *Execute* ili *Plan & Execute* isplanirana trajektorija bit će izvršena na robotu.

Izvršavanje trajektorija moguće je u dva načina rada: automatskom i ručnom. Način rada odabire se postavljanjem sklopke za odabir načina rada vidljivom na slici 37. U automatskom načinu rada potrebno je aktivirati motore pritiskom na sklopku za motore te pritisnuti *Play*. Nakon toga se u ROS sustavu može odabrati naredba *Execute* ili *Plan & Execute* te će se izvršiti

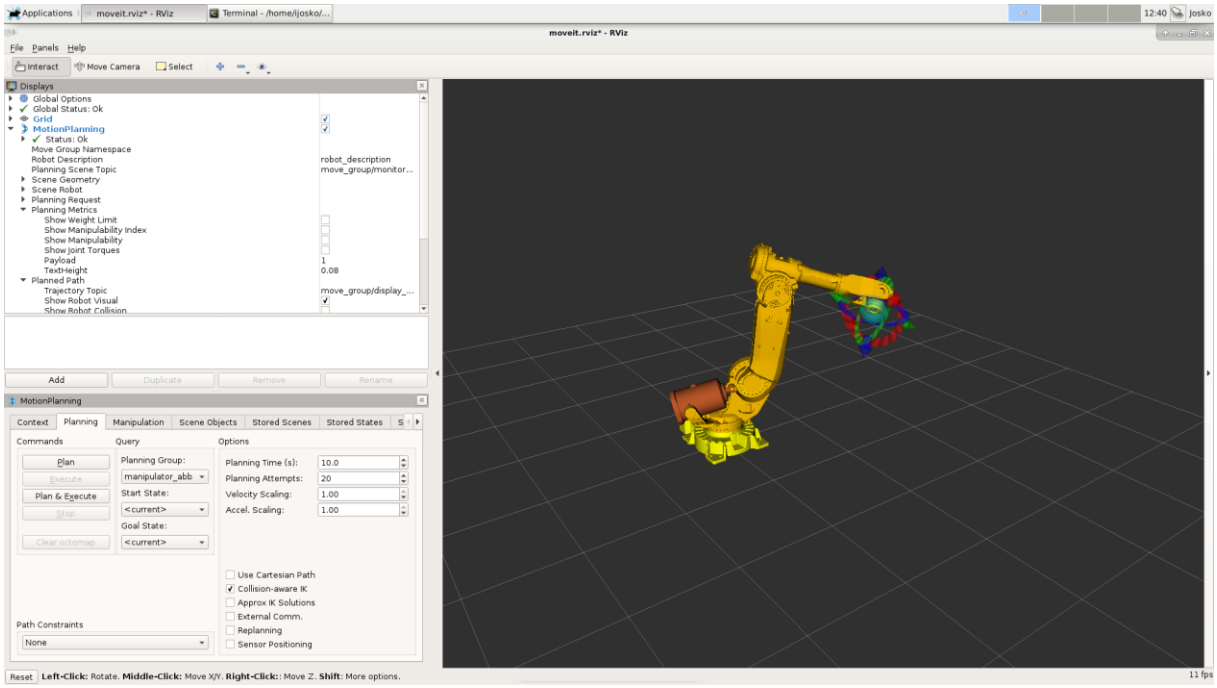
kretanje robota po planiranoj putanji. Na slikama 45 i 46 vidljivo je kako je robot u simulaciji zaista izvršio planiranu trajektoriju. U *RViz*-u se sada može vidjeti da su planirana i stvarna pozicija robota jednake, odnosno da je trajektorija izvršena.



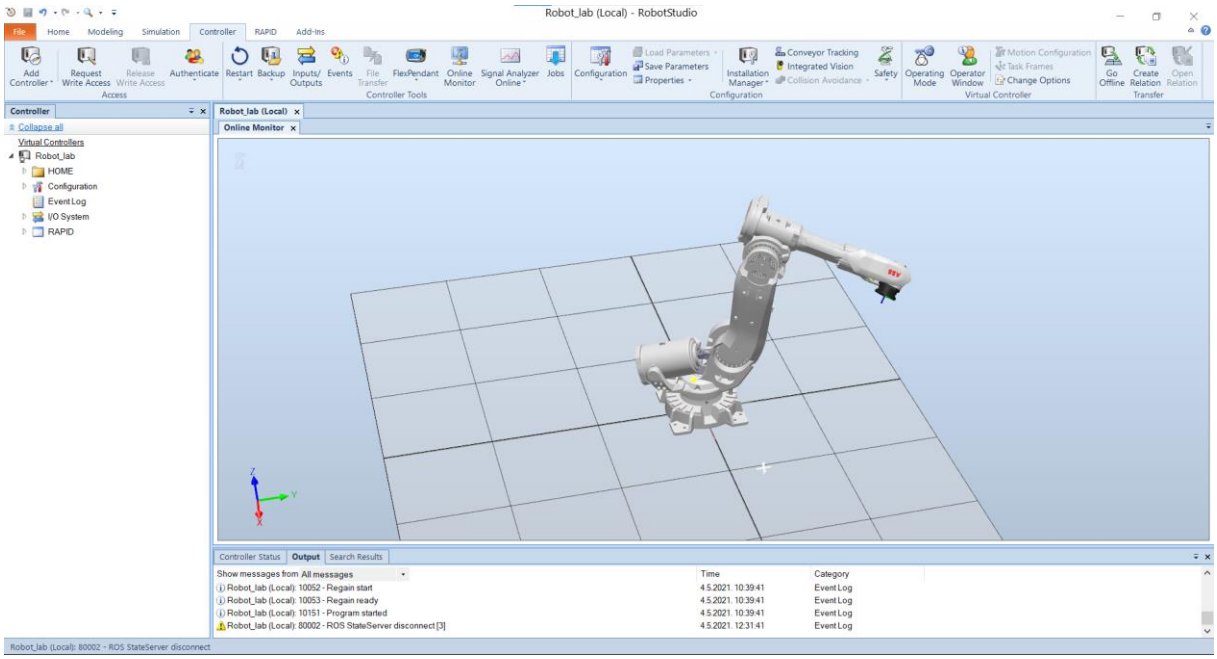
Slika 43 Programsko sučelje *MoveIt!/RViz* povezano s robotom simuliranim u programu *RobotStudio*



Slika 44 Trenutna pozicija robota prikazana u simulaciji u programu *RobotStudio*



Slika 45 Robot u programu *RViz* nakon izvršenja trajektorije



Slika 46 Robot u programu *RobotStudio* nakon izvršenja trajektorije

Osim navedena dva načina zadavanja trajektorije u ovom radu razvijen je i treći način zadavanja željene pozicije robotu, a to je pomoću Python skripte. Razvijene su dvije skripte, jedna za zadavanje pozicije u kartezijevom koordinatnom sustavu i druga za zadavanje pozicije položajem svakog pojedinog zgloba. U prilogu se nalaze čitave skripte, a ovdje će biti ukratko opisane.

```
moveit_commander.roscpp_initialize(sys.argv)
rospy.init_node('move_group_python_interface', anonymous=True)
```

Prvom naredbom inicijalizira se *moveit_commander* modul koji omogućuje komunikaciju *MoveIt!*-a i Pythona. Naredbom *rospy.init_node* pokreće se ROS čvor.

```
robot = moveit_commander.RobotCommander()
scene = moveit_commander.PlanningSceneInterface()
group = moveit_commander.MoveGroupCommander("manipulator_abb")
display_trajectory_publisher =
rospy.Publisher('/move_group/display_planned_path',
moveit_msgs.msg.DisplayTrajectory, queue_size=20)
```

Zatim se definiraju robot, robotsko okruženje, grupa kojom se upravlja (definirana u *MoveIt!*-u) te se toj grupi zadaje ciljana pozicija. Na kraju se pokreće čvor koji objavljuje trajektoriju u temu *'/move_group/display_planned_path'*.

Skripte za zadavanje položaja po poziciji zglobova i skripta za zadavanje položaja po kartezijskim koordinatama do ove linije su jednake. Sada slijede dva različita načina zadavanja položaja.

Zadavanje položaja po poziciji zglobova

```
group_variable_values = group.get_current_joint_values()
```

Ovo je funkcija koja očitava trenutnu stanje zglobova i upisuje to u varijablu *group_variable_values*.

```
group_variable_values[0] = -0.5
group_variable_values[1] = 0.3
group_variable_values[2] = 0.8
group_variable_values[3] = 2.6
group_variable_values[4] = -1.0
group_variable_values[5] = 0.0
group.set_joint_value_target(group_variable_values)
```

Zatim se postavljaju vrijednosti pojedinih zglobova u radijanima te se funkcijom *set_joint_values_target* postavlja ciljana pozicija.

```
plan1 = group.plan()

rospy.sleep(5)

group.go(wait=True)

moveit_commander.roscpp_shutdown()
```

Poziva se funkcija *group.plan()* koja izračunava trajektoriju te se nakon čekanja trajektorija izvršava. Naposljetku se gasi modul za komunikaciju.

Zadavanje položaja u kartezijevim koordinatama

```
pose_target = geometry_msgs.msg.Pose()
pose_target.orientation.w = 0
pose_target.orientation.x = 0
pose_target.orientation.y = 1
pose_target.orientation.z = 0
pose_target.position.x = 1.662
pose_target.position.y = -0.3
pose_target.position.z = 2.055
group.set_pose_target(pose_target)
```

Definira se pozicija u koju robot mora doći u prostornim koordinatama *x*, *y*, i *z* te po položaju u prostoru zadano u kvaternionima *w*, *x*, *y* i *z*. Kvaternionski zapis ovdje zapravo definira u kojem položaju mora biti koordinatni sustav posljednjeg članka robota u odnosu na globalni koordinatni sustav.

Izvršavanje trajektorije ponovo se izvršava identično kao u skripti za zadavanje položaja po poziciji zglobova.

4.3.2. Vođenje stvarnog robota

Kako se razvijeni ROS sustav za vođenje robota pokazao kao pouzdan u simuliranim uvjetima moglo se u konačnici pristupiti realizaciji vođenja fizičkog robota. U ovom smislu program *RobotStudio* u kojem je izvršena simulacija izuzetno je koristan jer omogućuje ispitivanje sustava bez rizika potencijalne materijalne štete ili ljudske ugroze.

Kako je već rečeno, proces konfiguracije stvarnog robotskog kontrolera istovjetan je onome virtualnog i stoga neće biti ponovo opisan. No, ipak se u nekoliko tehničkih detalja vođenje stvarnog i virtualnog robota razlikuje.

Prvi detalj je taj što su RAPID datoteke učitane na kontroler putem USB prijenosnika, za razliku od učitavanja na virtualni kontroler kada je to učinjeno dohvaćanjem s mreže. Nadalje, za vođenje stvarnog robota nije korišteno osobno računalo sa ROS sustavom instaliranom na virtualnom hardveru, već je korišteno industrijsko računalo. Industrijsko računalo puno je robusnije i pouzdanije, a s obzirom da se radi o poprilično skupoj opremi cilj je bio rizik od štete smanjiti koliko god je to moguće. Svi podaci potrebni za realizaciju vođenja uspješno su prebačeni s osobnog računala na industrijsko.

Prilikom simulacije vođenja su *RobotStudio* koji je pokretan na operacijskom sustavu *Windows10* i ROS pokretan operacijskim sustavom *Linux* povezani preko mrežnog adaptera. U slučaju vođenja stvarnog robota industrijsko računalo na kojem je ROS i upravljačko računalo robota povezani su *Ethernet* kabelom. Isto kao i kod simulacije vođenja, da bi se ostvarila komunikacija između ROS sustava i robota bilo je potrebno upisati valjanu IP adresu robota.

Vođenje stvarnog robota izvršeno je u ručnom načinu rada. Prilikom vođenja u ovom načinu rada potrebno je prvo postaviti pokazivač programa – PP u glavni program (eng. *main*) koji je u ovom slučaju T_ROB1. Zatim je potrebno pritisnuti prekidač za paljenje motora te ga držati pritisnutim tokom cijelog vremena izvršavanja programa, odnosno trajektorije. Ovaj prekidač je sigurnosna značajka kontrolera. U slučaju da dođe do ozljeđivanja operatora pretpostavka je kako će otpustiti ovaj prekidač što će isključiti motore robota. Zatim je potrebno pritisnuti tipku *Play* te nakon toga na ROS računalu pokrenuti izvršavanje trajektorije.

Razvijeni ROS sustav pokazao se jednako uspješan u vođenju stvarnog kao i u vođenju simuliranog robota

5. ZAKLJUČAK

U ovome radu uspješno je izvedena prilagodba ROS (eng. *Robotic Operating System*) sustava za vođenje industrijskog robota tipa ABB IRB6640.

U prvom dijelu rada opisan je razvoj primjene industrijskih robota te je pokazano kako, iako je razvoj industrijskih robota u prvih nekoliko desetljeća od prve implementacije tekao ubrzano, u posljednjih dvadesetak godina taj trend se znatno usporio, unatoč ulaganju značajnih sredstava. Kao jedan od razloga pojave ovakvog trenda uočena je zatvorenost velikih proizvođača robota i robotske opreme. Zatvorenost tržišta usporava razmjenu znanja i informacija što otežava inovaciju, a to u konačnosti dovodi do ovakvog trenda.

Robotski operacijski sustav – ROS, besplatan je i otvoren programski okvir za razvoj robotskog softvera. ROS nudi veliki broj isprobanih i dokazano pouzdanih alata i repozitorija kako bi olakšao razvoj složenih robotskih sustava. Kao takav predstavlja potencijalno rješenje ranije spomenutog problema. ROS – Industrial razvijen je kao proširenje ROS sustava za primjenu u industriji.

Pri razvoju sustava za vođenje industrijskog robota korišteni su alati koje nudi ROS, poput programskih paketa *RViz i MoveIt!,* kako bi se generirala trajektorija. Integracijom ROS sustava i robota ABB IRB6640 vođenje robota uspješno je realizirano. Implementacijom ROS sustava u vođenju dokazana je njegova funkcionalnost, a postoje još mnoge njegove značajke koje nisu implementirane.

Suvremeni trend razvoja obradnih sustava je primjena industrijskih robota na obratcima složenije geometrije gdje nisu tražene visoke točnosti i velike sile obrade. U radu je spomenuta tehnologija *Scan-N-Plan* koja je dostupna putem repozitorija ROS sustava, a čijom bi implementacijom bilo omogućeno generiranje trajektorije za robota na bazi skena površine obratka. Stoga se kao mogući sljedeći korak predlaže integracija razvijenog ROS sustava sa vizijskim sustavom.

LITERATURA

[1] F. C. S. J. Sanneman L, »The State of Industrial Robotics: Emerging Technologies, Challenges, and Key Research Directions,« MIT, Cambridge, MA USA, 2020.

[2] Gasparetto A, Scalera L. A Brief History of Industrial Robotics in the 20th Century. Advances in Historical Studies, 2019, 8, 25-35.

[3] Mayoral Vilches V. Envisioning the Future of Robotics. Robohub. 2017. [Mrežno]. Available: https://robohub.org/envisioning-the-future-of-robotics/, 13. travanj 2021.

[4] Singh B, Sellappan N, Kumaradhas P. Evolution of Industrial Robots and their Applications, International Journal of Emerging Technology and Advanced Engineering, May 2013, Volume 3, Issue 5.

[5] Lentin J, Cacace J. Mastering ROS for Robotics Programming. 2nd ed. Birmingham, Mumbai. Packt; 2018.

[6] »ros.org,« 2021.. [Mrežno]. Available: https://www.ros.org/about-ros/, 22. travanj 2021.

[7] Hasan B. »Towards data science,« 21st October 2019. [Mrežno]. Available: https://towardsdatascience.com/what-why-and-how-of-ros-b2f5ea8be0f3. 15. travanj 2021.

[8] »ROS Answers,« [Mrežno]. Available: https://answers.ros.org/questions/, 26. travanj 2021.

[9] Mayoral-Vilches V, Pinzger M, Rass S, Dieber B, Gil-Uriarte E. »Can ROS be used in industry? Red teaming ROS-Industrial,« 2020.. [Mrežno]. Available: https://arxiv.org/pdf/2009.08211.pdf, 27. travanj 2021.

[10] »The Construct,« [Mrežno]. Available: https://www.theconstructsim.com/history-ros/, 27. travanj 2021.

[11] F. E. Martinez A, Learning ROS for Robotics Programming, Birmingham: Pact Publishing, 2013.

[12] »Gazebo Simulator,« [Mrežno]. Available: http://gazebosim.org/ 28. travanj 2021.

[13] »MoveIt!,« [Mrežno]. Available: https://moveit.ros.org/, 29. travanj 2021.

[14] »ROS-Industrial,« [Mrežno]. Available: https://rosindustrial.org/, 29. travanj 2021..

[15] »ABB Products,« [Mrežno]. Available: https://new.abb.com/products/robotics/industrial-robots/irb-6640/irb-6640-cad, 1. svibanj 2021.

[16] »GitHub,« [Mrežno]. Available: https://github.com/ros-industrial/abb_driver, 10. travanj 2021.

[17] Chen S, Wen JT »Industrial Robot Trajectory Tracking Using Multi-Layer Neural Networks Trained by Iterative Learning Control,« *DeepAI,* 2019.

PRILOZI

I. CD-R disc

II. URDF konfiguracijska datoteka

III. Python skripte

URDF konfiguracijska datoteka

```
<?xml version="1.0"?>

<robot name="abb_irb6640_235_255">
  <!-- link list -->
  <link name="base_link">
    <collision name="collision">
      <geometry>
        <mesh
filename="package://abb_irb6640_support/meshes/irb6640_235_255/collision/ba
za.stl" scale="0.001 0.001 0.001"/>
      </geometry>
    </collision>
    <visual name="visual">
      <geometry>
        <mesh
filename="package://abb_irb6640_support/meshes/irb6640_235_255/visual/baza.
stl" scale="0.001 0.001 0.001"/>
      </geometry>
      <material name="yellow">
        <color rgba="1 1 0 1"/>
      </material>
    </visual>
  </link>
  <link name="link_1">
    <collision name="collision">
      <geometry>
        <mesh
filename="package://abb_irb6640_support/meshes/irb6640_235_255/collision/kr
ak_1.stl" scale="0.001 0.001 0.001"/>
      </geometry>
    </collision>
    <visual name="visual">
      <geometry>
        <mesh
filename="package://abb_irb6640_support/meshes/irb6640_235_255/visual/krak_
1.stl" scale="0.001 0.001 0.001"/>
      </geometry>
      <material name="yellow"/>
    </visual>
  </link>
  <link name="link_2">
    <collision name="collision">
      <geometry>
        <mesh
filename="package://abb_irb6640_support/meshes/irb6640_235_255/collision/kr
ak_2.stl" scale="0.001 0.001 0.001"/>
      </geometry>
    </collision>
    <visual name="visual">
      <geometry>
        <mesh
filename="package://abb_irb6640_support/meshes/irb6640_235_255/visual/krak_
2.stl" scale="0.001 0.001 0.001"/>
      </geometry>
      <material name="yellow"/>
    </visual>
```

```
  </link>
  <link name="link_3">
    <collision name="collision">
      <geometry>
        <mesh
filename="package://abb_irb6640_support/meshes/irb6640_235_255/collision/kr
ak_3.stl" scale="0.001 0.001 0.001"/>
      </geometry>
    </collision>
    <visual name="visual">
      <geometry>
        <mesh
filename="package://abb_irb6640_support/meshes/irb6640_235_255/visual/krak_
3.stl" scale="0.001 0.001 0.001"/>
      </geometry>
      <material name="yellow"/>
    </visual>
  </link>
  <link name="link_4">
    <collision name="collision">
      <geometry>
        <mesh
filename="package://abb_irb6640_support/meshes/irb6640_235_255/collision/kr
ak_4.stl" scale="0.001 0.001 0.001"/>
      </geometry>
    </collision>
    <visual name="visual">
      <geometry>
        <mesh
filename="package://abb_irb6640_support/meshes/irb6640_235_255/visual/krak_
4.stl" scale="0.001 0.001 0.001"/>
      </geometry>
      <material name="yellow"/>
    </visual>
  </link>
  <link name="link_5">
    <collision name="collision">
      <geometry>
        <mesh
filename="package://abb_irb6640_support/meshes/irb6640_235_255/collision/kr
ak_5.stl" scale="0.001 0.001 0.001"/>
      </geometry>
    </collision>
    <visual name="visual">
      <geometry>
        <mesh
filename="package://abb_irb6640_support/meshes/irb6640_235_255/visual/krak_
5.stl" scale="0.001 0.001 0.001"/>
      </geometry>
      <material name="yellow"/>
    </visual>
  </link>
  <link name="link_6">
    <collision name="collision">
      <geometry>
        <mesh
filename="package://abb_irb6640_support/meshes/irb6640_235_255/collision/kr
ak_6.stl" scale="0.001 0.001 0.001"/>
```

```
      </geometry>
    </collision>
    <visual name="visual">
      <geometry>
        <mesh
filename="package://abb_irb6640_support/meshes/irb6640_235_255/visual/krak_
6.stl" scale="0.001 0.001 0.001"/>
      </geometry>
      <material name="yellow"/>
    </visual>
  </link>
  <link name="tool0"/>
  <!--Cylinder and piston -->
  <link name="link_cylinder">
    <collision name="collision">
      <geometry>
        <mesh
filename="package://abb_irb6640_support/meshes/irb6640_235_255/collision/cy
linder_link.stl"/>
      </geometry>
    </collision>
    <visual name="visual">
      <geometry>
        <mesh
filename="package://abb_irb6640_support/meshes/irb6640_235_255/visual/cylin
der_link.dae"/>
      </geometry>
      <material name="yellow"/>
    </visual>
  </link>
  <link name="link_piston">
    <collision name="collision">
      <geometry>
        <mesh
filename="package://abb_irb6640_support/meshes/irb6640_235_255/collision/pi
ston_link.stl"/>
      </geometry>
    </collision>
    <visual name="visual">
      <geometry>
        <mesh
filename="package://abb_irb6640_support/meshes/irb6640_235_255/visual/pisto
n_link.dae"/>
      </geometry>
      <material name="yellow"/>
    </visual>
  </link>
  <!-- end of link list -->
  <!-- joint list -->
  <joint name="joint_1" type="revolute">
    <origin rpy="0 0 0" xyz="0 0 0.780"/>
    <axis xyz="0 0 1"/>
    <parent link="base_link"/>
    <child link="link_1"/>
    <limit effort="0" lower="-2.967" upper="2.967" velocity="1.7453"/>
  </joint>
  <joint name="joint_2" type="revolute">
    <origin rpy="0 0 0" xyz="0.320 0 0"/>
```

```
    <axis xyz="0 1 0"/>
    <parent link="link_1"/>
    <child link="link_2"/>
    <limit effort="0" lower="-1.134" upper="1.4855" velocity="1.5707"/>
  </joint>
  <joint name="joint_3" type="revolute">
    <origin rpy="0 0 0" xyz="0 0 1.075"/>
    <axis xyz="0 1 0"/>
    <parent link="link_2"/>
    <child link="link_3"/>
    <limit effort="0" lower="-3.142" upper="1.222" velocity="1.5707"/>
  </joint>
  <joint name="joint_4" type="revolute">
    <origin rpy="0 0 0" xyz="0 0 0.2"/>
    <axis xyz="1 0 0"/>
    <parent link="link_3"/>
    <child link="link_4"/>
    <limit effort="0" lower="-5.236" upper="5.236" velocity="2.9671"/>
  </joint>
  <joint name="joint_5" type="revolute">
    <origin rpy="0 0 0" xyz="1.142 0 0 "/>
    <axis xyz="0 1 0"/>
    <parent link="link_4"/>
    <child link="link_5"/>
    <limit effort="0" lower="-2.094" upper="2.094" velocity="2.4435"/>
  </joint>
  <joint name="joint_6" type="revolute">
    <origin rpy="0 0 0" xyz="0.2 0 0 "/>
    <axis xyz="1 0 0"/>
    <parent link="link_5"/>
    <child link="link_6"/>
    <limit effort="0" lower="-6.283" upper="6.283" velocity="3.3161"/>
  </joint>
  <joint name="joint_6-tool0" type="fixed">
    <parent link="link_6"/>
    <child link="tool0"/>
    <origin rpy="0 1.57079632679 0" xyz="0 0 0"/>
  </joint>
  <joint name="joint_cylinder" type="continuous">
    <origin rpy="0 0 0" xyz="-0.3647 0 -0.1455"/>
    <axis xyz="0 1 0"/>
    <parent link="link_1"/>
    <child link="link_cylinder"/>
    <mimic joint="joint_2" multiplier="-0.25"/>
  </joint>
  <joint name="joint_piston" type="continuous">
    <origin rpy="0 0 0" xyz="-0.22 0 -0.0672"/>
    <axis xyz="0 1 0"/>
    <parent link="link_2"/>
    <child link="link_piston"/>
    <mimic joint="joint_2" multiplier="-1.25"/>
  </joint>
  <!-- end of joint list -->
  <!-- ROS base_link to ABB World Coordinates transform -->
  <link name="base"/>
  <joint name="base_link-base" type="fixed">
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <parent link="base_link"/>
```

```
    <child link="base"/>
  </joint>
</robot>
```

Python skripta za zadavanje pozicije po poziciji zglobova

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

import sys
import copy
import rospy
import moveit_commander
import moveit_msgs.msg
import geometry_msgs.msg

moveit_commander.roscpp_initialize(sys.argv) # inicijaliziranje moveit
commander modula koji omogućuje komunikaciju MoveIt-a i Pythona
rospy.init_node('move_group_python_interface', anonymous=True) # stvaranje
ROS čvora

# definiraju se robot, robotsko okruženje, grupa kojom se upravlja
(definirana u MoveIt! konfiguraciji)i pokreće se čvor koji objavljuje
trajektoriju u temu '/move_group/display_planned_path'
robot = moveit_commander.RobotCommander()
scene = moveit_commander.PlanningSceneInterface()
group = moveit_commander.MoveGroupCommander("manipulator_abb")
display_trajectory_publisher =
rospy.Publisher('/move_group/display_planned_path',
moveit_msgs.msg.DisplayTrajectory, queue_size=20)

group_variable_values = group.get_current_joint_values() # funkcija koja
očitava trenutnu poziciju zglobova

# vrijednosti pozicije pojedinih zglobova u radijanima (zglob ovdje označen
s [0] je joint_1)
group_variable_values[0] = -0.5
group_variable_values[1] = 0.3
group_variable_values[2] = 0.8
group_variable_values[3] = 2.6
group_variable_values[4] = -1.0
group_variable_values[5] = 0.0
group.set_joint_value_target(group_variable_values) # postavlja se ciljana
pozicija zglobova

plan1 = group.plan() # funkcija koja izračunava trajektoriju

rospy.sleep(5)

group.go(wait=True) #izvršavanje trajektorije

moveit_commander.roscpp_shutdown() # gašenje modula
```

Python skripta za zadavanje pozicije u kartezijskom koordinatnom sustavu

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

import sys
import copy
import rospy
import moveit_commander
import moveit_msgs.msg
import geometry_msgs.msg

moveit_commander.roscpp_initialize(sys.argv)
rospy.init_node('move_group_python_interface', anonymous=True)

robot = moveit_commander.RobotCommander()
scene = moveit_commander.PlanningSceneInterface()
group = moveit_commander.MoveGroupCommander("manipulator_abb")
display_trajectory_publisher =
rospy.Publisher('/move_group/display_planned_path',
moveit_msgs.msg.DisplayTrajectory, queue_size=20)


pose_target = geometry_msgs.msg.Pose()
pose_target.orientation.w = 0
pose_target.orientation.x = 0
pose_target.orientation.y = 1
pose_target.orientation.z = 0
pose_target.position.x = 1.662
pose_target.position.y = -0.3
pose_target.position.z = 2.055
group.set_pose_target(pose_target)


plan1 = group.plan()

rospy.sleep(5)

group.go(wait=True)

moveit_commander.roscpp_shutdown()
```