Sustav upravljanja robotskim manipulatorom pomoću 3D kamere u ROS okviru

Meisel, Marko

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Diplomski sveučilišni studij računarstva

SUSTAV UPRAVLJANJA ROBOTSKIM MANIPULATOROM POMOĆU 3D KAMERE U ROS OKVIRU

Diplomski rad

Marko Meisel

Osijek, 2018.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek, 19.09.2018.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Marko Meisel
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 848 R, 28.09.2017.
OIB studenta:	40563808445
Mentor:	Prof.dr.sc. Robert Cupec
Sumentor:	
Sumentor iz tvrtke:	Ferdo Bošnjak
Predsjednik Povjerenstva:	Doc.dr.sc. Emmanuel-Karlo Nyarko
Član Povjerenstva:	Doc.dr.sc. Damir Filko
Naslov diplomskog rada:	Sustav upravljanja robotskim manipulatorom pomoću 3D kamere u ROS okviru
Znanstvena grana rada:	**Procesno računarstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	Izraditi sustav za upravljanje robotskim manipulatorom ABB IRB 2400L pomoću podataka dobivenih 3D kamerom. Sustav treba implementirati korištenjem Robot Operating System (ROS) okvira, kao aplikaciju koja se izvodi na osobnom računalu.
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	19.09.2018.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 04.10.2018.

Ime i prezime studenta:	Marko Meisel
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 848 R, 28.09.2017.
Ephorus podudaranje [%]:	1%

Ovom izjavom izjavljujem da je rad pod nazivom**: Sustav upravljanja robotskim manipulatorom pomoću 3D kamere u ROS okviru**

izrađen pod vodstvom mentora Prof.dr.sc. Robert Cupec

i sumentora

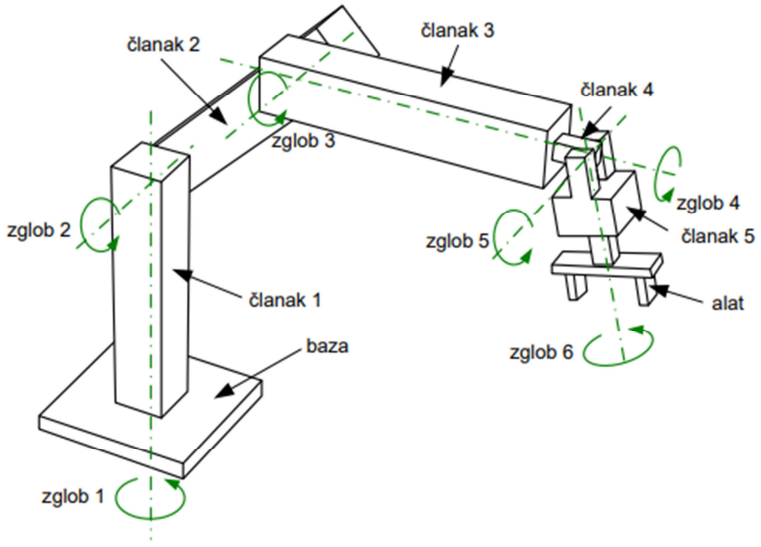
moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Sadržaj

1. Uvod

Robot je mehanički stroj koji omogućuje gibanje određenog alata ili predmeta u prostoru različitim putanjama pomoću odgovarajućih senzora. Robot se sastoji od više krutih tijela, koje zovemo člancima, koji su međusobno povezani pokretljivim zglobovima (slika 1.1.). Svrha robota može biti manipulacija fizičkim objektima, obrada površine, bušenje, zavarivanje i sl. Roboti se danas široko primjenjuju u industriji zbog svoje velike preciznosti i pozitivnog utjecaja na smanjenje troškova proizvodnje. Kako bi robot bio u mogućnosti odraditi zadani posao, potrebno je prvo postaviti odgovarajući alat na robota. Danas se u praktičnoj primjeni najčešće koriste roboti koji po funkciji odgovaraju ljudskoj ruci, te se oni nazivaju robotski manipulatori.



Slika 1.1. Robotski manipulator

Putanja po kojoj će se manipulator kretati može biti strogo definirana od korisnika ili je upravljački sustav može generirati ovisno o zadatku koji robot mora izvršiti. Ukoliko je strogo definirana od strane korisnika, manipulator neće biti u mogućnosti raditi u dinamičkoj okolini, gdje bi on morao mijenjati svoju putanju ovisno o trenutnoj poziciji objekata u svojoj radnoj okolini. Sustavi koji koriste ovaj tip programiranja manipulatora su vrlo česti, ali postoje zadaci koji zahtijevaju rad manipulatora u dinamičkoj okolini. Jedno od mogućih rješenja za ovakve zadatke je primjena računalnog vida za upravljanje manipulatorom gdje se za prikupljanje informacija iz okoline koriste se 2D i 3D kamere.

U svijetu danas postoji široka baza znanja iz područja robotike u vidu programskih modula i biblioteka čije korištenje uvelike olakšava razvoj novih robotskih aplikacija. Da bi se to znanje moglo učinkovito koristiti, vrlo je važno da tehnička rješenja budu realizirana na principu modularnosti odnosno kao sustavi pomoću kojih se jednostavno povezuju nove i postojeće komponente u funkcionalne cjeline. Za širenje i unaprjeđivanje navedene baze znanja, ključan je koncept otvorenog koda (engl. *Open source*), koji omogućuje brzu razmjenu znanja između istraživačkih i razvojnih grupa širom svijeta. Jedan od programskih okvira koji koristi modularnost i koncept otvorenog koda za svoj rad je ROS (engl. *Robot Operating System*) programski okvir. ROS omogućuje brzu i efikasnu realizaciju programskih rješenja koja zahtijevaju povezivanje osobnog računala s robotima (mogu biti mobilni roboti ili robotski manipulatori). Posebna je pažnja dana pri povezivanju osobnog računala s komercijalnim industrijskim manipulatorima, zbog kojih je razvijen projekt zvan industrijski ROS. ROS okvir je iznimno koristan za znanstvena istraživanja pošto je baza postojećih programskih rješenja razvijenih za osobno računalo neusporedivo veća od baze rješenja raspoloživih za određeni tip robota. Kroz ovaj diplomski rad razvijena je programska platforma zasnovana na ROS okviru, koja omogućuje upravljanje robotom pomoću programa, koji se izvodi na osobnom računalu te njegovo povezivanje sa sustavom kamere.

U poglavlju 2 dana je kratka povijest ROS okvira i ključne informacije za razumijevanje njegovog rada. U poglavlju 3 opisana je izrada i korištenje konfiguracijskog paketa koji omogućuje komunikaciju i rad s robotskim manipulatorom. U poglavlju 4 opisano je sučelje za komunikaciju između navedenog konfiguracijskog paketa i apstraktnog primjera programa kalibracije kamere. Razvijeni sustav je eksperimentalno ispitan pomoću industrijskog robota IRB 2400L. Rezultati pokusa gdje se provjerava točnost izrađenih modela prikazani su u poglavlju 6. Zaključak rada dan je u poglavlju 6.

2. Robot Operating System – ROS

ROS (engl. *Robot Operating System*) je programski okvir otvorenog koda namijenjen razvoju robotske programske podrške. Pruža servise za apstrakciju sklopovlja, upravljanje sklopovljem na niskoj programskoj razini, komunikaciju između procesa te upravljanje paketima [1]. ROS je namijenjen korištenju na Linux operativnom sustavu mada postoje eksperimentalne verzije ROS programskog okvira za operativne sustave Windows i macOS. Podržava programske jezike C++, Python i LISP. Cilj razvoja ROS okvira je omogućiti

standardizirani skup alata kako bi korisnici mogli brže i lakše razvijati svoja vlastita programska rješenja i kako bi vrlo lako mogli razmjenjivati kod.

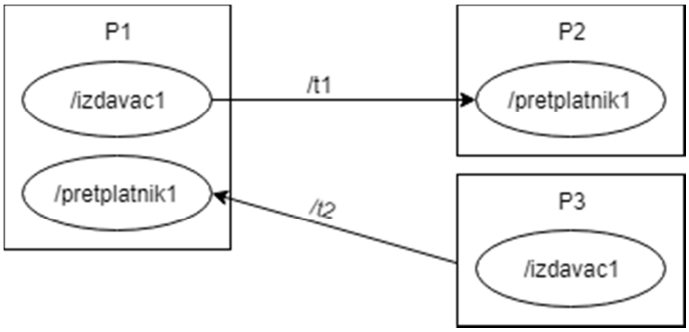
2.1 Kratka povijest

ROS je započeo svoj razvoj 2007. godine kada su se uzele već razvijene efikasne metode za rad s robotima iz raznih programskih okvira otvorenog koda, od kojih je najznačajniji "*Switchyard*" razvijen u laboratoriju za umjetnu inteligenciju sveučilišta Stanford [2]. Daljnji razvoj ROS okvira se izvodio u istraživačkom laboratoriju za robotiku Willow Garage gdje se i dan danas unaprjeđuje. Tri godine kasnije izdana je prva službena distribucija otvorenog koda nazvana "ROS Box Turtle" pod BSD (engl. *Berkeley Source Distibution*) licencom, koja označava besplatno korištenje distribucije u akademske i komercijalne svrhe. Iste se godine odmah počeo implementirati te su neka od značajnijih prvih implementacija ROS okvira autonomni auto na sveučilištu Austin u Teksasu i dron na sveučilištu Pennsylvania. Nove distribucije ROS okvira izlaze svake godine u mjesecu svibnju te imaju podršku dvije godine, dok svake druge godine izlaze distribucije s dugoročnom podrškom u trajanju od pet godina. Danas je ROS vrlo robustan i univerzalan okvir za razvoj robotske programske podrške zbog velikog broja alata, podrške za senzore i aktuatore i podrške za velik broj robota.

2.2 Osnovni koncept ROS okvira

Radom ROS okvira upravlja jezgra ROS okvira (engl. *Roscore*) koja omogućuje komunikaciju između procesa i upravljanje s vremenom [2]. Svi procesi pokretani koristeći ROS predstavljeni su kao čvorovi koji međusobno komuniciraju razmjenjivanjem poruka preko teme (engl. *Topic*) time čineći graf. Nit procesa koja šalje poruku u određenoj tematici naziva se izdavač (engl. *Publisher*) dok se nit koja prima poruku naziva pretplatnik (engl. *Subscriber*). Tema je naziv za tok poruka određenog tipa koji objedinjuje tip poruke koja se smije slati, vrijeme slanja poruke te same poruke [2]. Temu kreira izdavač koji prvo oglašava naziv i tip teme pa se nakon toga pretplatnici mogu pretplatiti na tu temu kako bi primali poruke. Jezgra ROS okvira svakoj poruci dodjeljuje vrijeme slanja poruke i uklanja zastarjele poruke u temama. ROS ima implementirane neke od najčešće korištenih poruka (npr. trenutna pozicija i orijentacija vrha alata robota, kutevi zglobova robota, trajektorija vrha alata, slika kamere itd.), ali, ukoliko je potrebno, korisnik može definirati i svoj tip. Nad jednom temom nema ograničenja koliko može biti izdavača i pretplatnika te isto tako jedan proces može

sadržavati više niti od kojih su neke izdavači a neke pretplatnici na različitim temama. Na slici 2.1 prikazan je graf s 3 procesa: P1 s jednim izdavačem i jednim pretplatnikom, P2 s jednim pretplatnikom i P3 s jednim izdavačem. Proces P1 šalje poruke procesu P2 preko teme t1 dok prima poruke od procesa P3 preko teme t2.



Slika 2.1 Primjer grafa čvorova ROS okvira

Za neke primjene, komunikacija čvorova putem teme je dovoljno dobra da bi se ostvarilo jednostavno razmjenjivanje poruka. Međutim, ponekad je potrebna dvosmjerna komunikacija između čvorova ili blokiranje izvođenja programa dok se ne izvrši zahtjev koji smo predali putem teme. Za ovakvu komunikaciju koristi se servis. Servis je jednostavan način dvosmjerne komunikacije između čvorova, gdje je za rad potrebno jedan čvor imenovati serverom, koji će posluživati zahtjeve klijentskih čvorova. ROS okvir kreira dvije teme: temu zahtjeva i temu odgovora. Čvor imenovan serverom postaje pretplatnik na temi zahtjeva i izdavač na temi odgovora. Klijent putem servisa postaje izdavač na temi zahtjeva i pretplatnik na temi odgovora. Kada klijent pošalje poruku putem teme zahtjeva, nakon što se zahtjev izvrši server će oglasiti status izvršenja na temi odgovora[3]. Korisnik sam može odabrati hoće li se izvođenje programa blokirati do odgovora servera. Tokom slanja zahtjeva za kretanje robota, vrlo često se blokira daljnje izvođenje programa do izvršenja zahtjeva, ali ponekad izvršavanje zahtjeva za kretanje traje dugo, pa je korisno povremeno od servera dobiti povratnu informaciju gdje se robot trenutno nalazi (može se raditi o vrhu alata robotskog manipulatora ili bazi mobilnog robota). Za ovakvu vrstu komunikacije koristi se akcija gdje se kreiraju tri teme: tema cilja (engl. *Goal*), tema povratne veze (engl. *Feedback*) i tema rezultata (engl. *Result*). Kod akcije je također potrebno jedan čvor imenovati serverom. Sada klijent može poslati zahtjev putem teme cilja, čekati odgovor za izvršenje zahtjeva na tematici rezultata i biti povremeno obavještavan o trenutnom statusu putem teme povratne veze.

2.3 Korisnički radni prostor

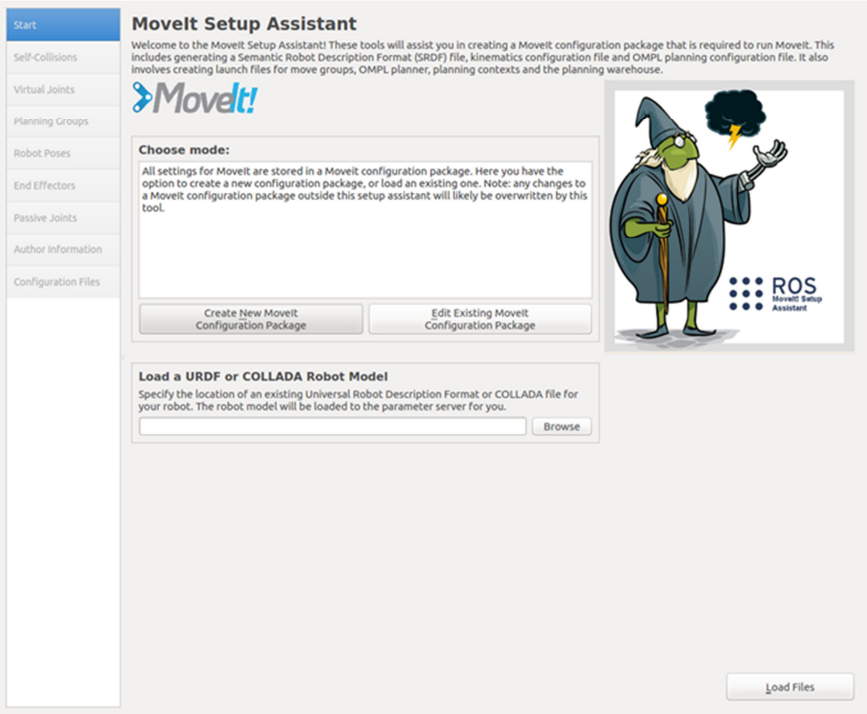
Korisnički radni prostor (engl. *Workspace*) je direktorij koji sadrži korisničke datoteke organizirane u pakete. Korisničke datoteke mogu biti C++ datoteke, Python skripte, zaglavlja programskog jezika i druge datoteke koje korisnik sam izrađuje. Paketi su logičke cjeline korisničkih datoteka koji pomažu u njihovoj organizaciji i omogućuju lakšu prenosivost između računala koja koriste ROS okvir. Kako bi kreirali radni prostor, potrebno je stvoriti direktorij radnog prostora. Konvencija za imenovanje direktorija je da ime počinje sa "catkin" (naziv prevoditelja ROS okvira) te završava sa "_ws" kako bi se naznačilo da se radi o radnom prostoru. Broj radnih prostora u ROS okviru nije ograničen. Također, svaki radni prostor mora sadržavati direktorij "src" gdje se nalaze paketi. Za potrebe ovog rada, kreirat će se dva radna prostora: "catkin_abb_ws" i "catkin_ws" u glavnom direktoriju trenutnog korisnika. Direktorij "catkin_abb_ws" koristit će se za konfiguracijski paket korištenog robotskog manipulatora dok će se direktorij "catkin_ws" koristiti za korisničke programe. Kreiranje radnih prostora koristeći terminal dan je u prilogu A ovog dokumenta.

2.4 Industrijski ROS i MoveIt! programski paket

Industrijski ROS (engl. *ROS-Industrial*) je paket ROS okvira koji pruža dodatne biblioteke, alate i upravljačke programe za rad s industrijskim sklopovljem [4]. Također, zajednica razvojnih inženjera, koja održava i unaprjeđuje ovaj paket, pruža već gotove konfiguracijske pakete za velik broj industrijskih manipulatora i korisnik ih može koristiti, izmijeniti ili stvoriti svoj paket za upravljanje robotskim manipulatorom. programski paket. Industrijski ROS podržava rad s velikim brojem sklopovlja tvrtki ABB, Adept, Fanuc, Kuka, Motoman, Robotiq i Universal Robots.

Za rad s konfiguracijskim paketima koristi se MoveIt!. MoveIt! je programski paket koji sadrži alate za modeliranje i upravljanje robotskim manipulatorima i mobilnim robotima što uključuje proračunavanje direktne i inverzne kinematike, planiranje trajektorije, vizualizaciju kretanja, simulaciju rada i navigaciju. Ovaj paket za rad koristi programsku datoteku tipa URDF (engl. *Universal Robot Description Format*) koju definira korisnik. URDF datoteka se temelji na XML jeziku i ona u potpunosti sadržava opis robotskog manipulatora s kojim se radi (npr. ovisnosti između zglobova manipulatora, međusobne udaljenosti pojedinih zglobova, materijal od kojega je napravljen pojedini članak itd.). Ukoliko je tokom rada s paketom potrebna i vizualizacija rada, tada je potrebno definirati modele pojedinih članaka.

Tokom planiranja kretanja robota, postoji mogućnost da robot dođe u takvu poziciju da se pojedini članak robota sudari s nekim drugim člankom odnosno da se robot sudari sa samim sobom. Kako bi se to izbjeglo, moguće je MoveIt! paketu definirati grubi model pojedinog članka pomoću kojeg može izbjegavati takve pozicije. Povezivanje URDF datoteke s modelima robotskog manipulatora omogućeno je kroz MoveIt! asistenta te korisnika vodi korak po korak kroz izradu konfiguracijskog paketa željenog manipulatora. Početni ekran MoveIt! asistenta prikazan je na slici 2.2. U jednom konfiguracijskom paketu nije ograničeno koliko može biti robotskih manipulatora ili mobilnih robota. Kako bi se moglo upravljati pojedinim robotom u konfiguracijskom paketu, svakom robotu se dodjeljuje grupa za planiranje trajektorije. Najčešće je u konfiguracijskom paketu samo jedan robotski manipulator kojem se po konvenciji dodjeljuje naziv grupe "manipulator".



Slika 2.2 Početni ekran MoveIt! asistenta za kreiranje konfiguracijskog paketa

3. Izrada i korištenje konfiguracijskog paketa za robot IRB 2400L

Konfiguracijski paket predstavlja skup datoteka koje omogućuju upravljanje određenim modelom robota. On sadrži sve specifičnosti za određeni model robota, čime je omogućena modularnost programskih rješenja izrađenih na ROS platformi, na način da se neko programsko rješenje može primijeniti na određeni tip robota jednostavnim uključivanjem konfiguracijskog paketa tog robota.

Robotski manipulator korišten u ovom radu je ABB IRB 2400L. Za njega ne postoji gotov konfiguracijski paket, ali postoji za njemu sličan model ABB IRB 2400[1] . U poglavlju 4.1 opisana je izrada konfiguracijskog paketa za robot IRB 2400L prilagodbom konfiguracijskog paketa za robot IRB 2400. Detalji vezani za ovu prilagodbu dani su u Prilogu B. Informacije dane u tom prilogu mogu biti od iznimne koristi inženjerima, koji budu u situaciji da moraju napraviti sličnu prilagodbu za neki drugi tip robota.

Konfiguracijski paket se može koristiti za upravljanje robotom pomoću računalnog programa napisanog za ROS platformu, koji se izvodi na osobnom računalu, preko kontrolera robota. Također se u razvoju nekog programskog rješenja može koristiti i simulator robota, kako je opisano u poglavlju 3.2.

Neki detalji o dodatnim programskim alatima korištenim za izradu konfiguracijskog paketa opisanog u ovom poglavlju i njegovu primjenu, dani su u Prilogu C.

3.1 Izrada konfiguracijskog paketa za robot IRB 2400L

Direktorije konfiguracijskog paketa robotskog manipulatora potrebno je spremiti u "src" datoteku radnog prostora "catkin_abb_ws". Postojeći konfiguracijski paket sastoji se od šest direktorija:

- abb – sadrži informacije o promjenama kroz inačice i autorima
- abb_driver – sadrži sistemske datoteke za kontroler robota, koje omogućuju komunikaciju s računalom
- abb_irb2400_moveit_config – sadrži datoteke koje opisuju mogućnosti manipulatora (maksimalni kutevi pojedinog zgloba, koji članci se trebaju provjeravati za koliziju tokom planiranja trajektorije itd.) te datoteke za pokretanje
- abb_irb2400_moveit_plugins – sadrži datoteke potrebne za rad direktne i inverzne kinematike
- abb_irb2400_moveit_support – sadrži vizualne i kolizijske modele pojedinog članka i URDF datoteku, objašnjenu u Prilogu B.2.
- abb_resources – sadrži datoteke koje opisuju boju korištenog materijala za izradu robotskog manipulatora

[1] https://github.com/ros-industrial/abb

Naime, oba modela imaju istu orijentaciju zglobova, no razmak između zglobova odnosno duljina članaka od dva do šest im je različita. Potrebno je izmijeniti URDF datoteku, vizualni i kolizijski model te IKFast kinematički dodatak kako bi MoveIt! paket ispravno proračunavao direktnu i inverznu kinematiku. Korišteni robot s označenim člancima prikazan je na slici 3.1.

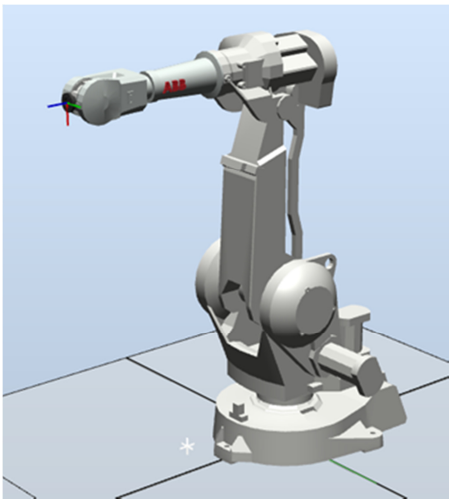
Slika 3.1 Korišten robotski manipulator sa označenim člancima

Izmjene koje je bilo potrebno napraviti da bi se postojeći konfiguracijski paket izrađen za model ABB IRB 2400 prilagodio modelu IRB 2400L uključuju

1. izmjenu vizualnog i kolizijskog modela, opisanu u Prilogu B.1;
2. izmjenu URDF datoteke, opisanu u Prilogu B.2 te
3. izradu datoteka direktne i inverzne kinematike, opisanu u Prilogu B.3.

3.2 Konfiguracija alata ABB RobotStudio i kontrolera robota

Konfiguracija alata RobotStudio za rad sa simulacijom i za rad sa stvarnim robotskim kontrolerom je skoro u potpunosti identična. Razlika je ta što kod rada sa stvarnim kontrolerom RobotStudio sam prepoznaje o kojem se robotskom manipulatoru radi, dok je kod simuliranja potrebno ručno odabrati model robotskog manipulatora. Za spajanje na stvarni kontroler koristi se funkcija "One Click Connect", koja se automatski spaja na kontroler. Za simuliranje manipulatora potrebno je kreirati novi prazni radni prostor i iz ABB biblioteke dodati korišteni robotski manipulator. Zatim je potrebno dodati korišteni robotski sustav koristeći model (engl. *New Robot system from layout*). Na ekranu se prikazuje robotski manipulator s kojim se radi, kao na primjeru prikazanom na slici 3.2.



Slika 3.2 Prikaz korištenog robotskog manipulatora u alatu RobotStudio

Ostatak procedure je isti za simulacijski model i stvarni kontroler. Potrebno je otići u postavke kontrolera i odabrati dodatne stavke koje će omogućiti komunikaciju s računalom (616-1 PC interface) i višezadaćni rad (623-1 Multitasking). Iako je sada omogućena komunikacija s računalom i višezadaćni rad, kontroler nema učitanu programsku podršku koju će pokretati za rad s ROS okvirom. Potrebna programska podrška se nalazi u konfiguracijskom paketu industrijskog ROS okvira u poddirektoriju "rapid" direktorija "abb_driver". Tu programsku podršku potrebno je kopirati u memoriju kontrolera robota i postaviti postavke za njihovo pokretanje. U alatu RobotStudio, s lijeve strane stisnemo desnim klikom na kreirani robotski sustav i odaberemo opciju "Open System Folder" kako bi prikazao direktorij kreiranog sustava. U ovom direktoriju kreiramo novi direktorij naziva "ROS" i u njega kopiramo programsku podršku iz direktorija "abb_driver". Kako bi spajanje računala na kontroler bilo uspješno, potrebno je postaviti lokalnu statičku IP adresu računala koje se spaja na kontroler. Za simulaciju ova adresa je proizvoljna dok za spajanje na stvarni kontroler robota ove podatke možemo dobiti sa samog kontrolera. Za simulaciju, korištena IP adresa je 192.168.1.42. Za stvarni rad s kontrolerom korištena IP adresa je 172.16.106.185. Sada je potrebno pomoću nekog uređivača teksta otvoriti datoteku "ROS_socket.sys" i u ovu datoteku zapisati željenu IP adresu. Željenu IP adresu potrebno je zapisati na 32. liniji umjesto naredbe "GetSysInfo" tako da ta linija sa željenom IP adresom za simulaciju izgleda ovako[7]:

IF (SocketGetStatus(server_socket) = SOCKET_CREATED) SocketBind server_socket, "192.168.1.42", port;

Nakon prebačene programske podrške u memoriju kontrolera, potrebno je postaviti zadaće koje kontroler treba obavljati koristeći tu programsku podršku. U alatu RobotStudio potrebno je putem uređivača konfiguracije (engl. *Configuration Editor*) otvoriti konfiguraciju

kontrolera. Zatim je potrebno odabrati u izborniku odabir Zadaće (engl. *Tasks*) i dodati tri zadaće prikazane u tablici 3.1.

Tablica 3.1 Tablica kreiranih zadaća

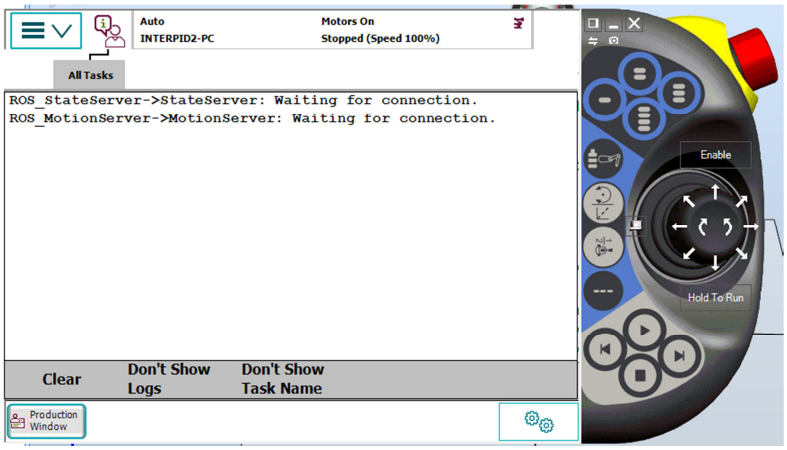
Naziv zadaće	ROS_StateServer	ROS_MotionSer ver	T_ROB1
Top zadaće	SEMISTATIC	SEMISTATIC	NORMAL
Razina povjerenja	NoSafety	SysStop	-
Unos	Main	Main	Main
Zadatak za kretanje?	Ne	Ne	Da

Zadaća "ROS_StateServer" šalje računalu trenutne kuteve zglobova robotskog manipulatora i njegovo trenutno stanje. Zadaća "ROS_MotionServer" prima željene kuteve zglobova i prosljeđuje ih zadaći "T_ROB1", koja izvršava kretanje robotskog manipulatora prema željenim kutevima. Sada je potrebno učitati datoteke koje smo prebacili u memoriju kontrolera u kreirane zadaće. Potrebno je u izborniku konfiguracije kontrolera odabrati automatsko učitavanje modula (engl. *Automatic Loading of Modules*) i dodati šest novih modula prikazanih u tablici 3.2.

Tablica 3.2 Tablica kreiranih modula

Putanja	Zadaća	Instaliran?	Sve zadaće?	Skriven
HOME:/ROS/ROS_common.sys	-	Ne	Da	Ne
HOME:/ROS/ROS_socket.sys	-	Ne	Da	Ne
HOME:/ROS/ROS_messages.sys	-	Ne	Da	Ne
HOME:/ROS/ROS_stateServer.mod	ROS_StateServer	Ne	Da	Ne
HOME:/ROS/ROS_motionServer.mod	ROS_MotionServer	Ne	Ne	Ne
HOME:/ROS/ROS_motion.mod	T_ROB1	Ne	Ne	Ne

Nakon toga, potrebno je ponovno pokrenuti kontroler robotskog manipulatora kako bi promjene bile učitane u radnu memoriju. Nakon ponovnog učitavanja kontrolera, na zaslonu kontrolera bi trebao biti prikazano da kontroler čeka na spajanje s računalom. Na slici 3.3 je prikazan zaslon virtualnog kontrolera robotskog manipulatora nakon ponovnog pokretanja.

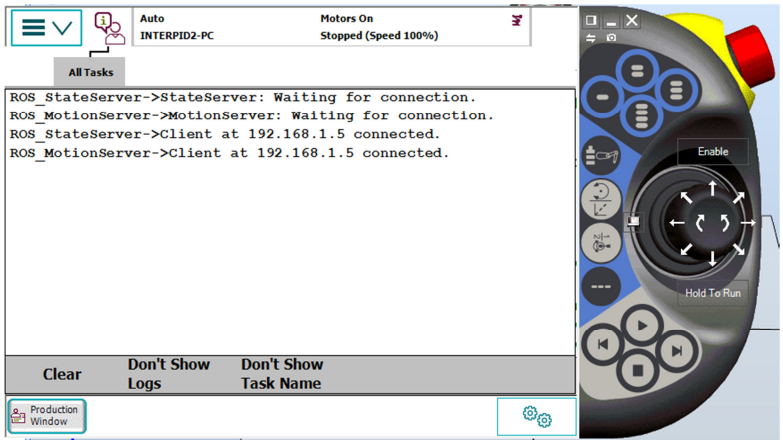


Slika 3.3 Zaslon kontrolera manipulatora dok čeka spajanje s računalom

Ukoliko je prikazana ovakva poruka na zaslonu, onda je moguće testirati rad ROS okvira s kontrolerom i njihovu komunikaciju tako da otvorimo terminal i pokrenemo slijedeću naredbu:

$ roslaunch abb_irb2400_moveit_config moveit_planning_execution.launch sim:=false robot_ip:=192.168.1.42

Nakon toga bi na zaslonu kontrolera robota trebalo biti prikazano uspješno spajanje te bi također trebala biti navedena i IP adresa računala koje se spojilo na kontroler. Slika uspješnog spajanja ROS okvira na virtualni stroj, gdje simuliramo rad, prikazana je na slici 3.4.



Slika 3.4 Zaslon konktrolera nakon uspješnog spajanja s računalom

Gore navedena naredba će se vrlo često koristiti, odnosno nju je potrebno izvršiti svaki puta kada se želimo spojiti na kontroler manipulatora, pa je korisno napraviti alias u Linux operativnom sustavu:

$ echo ""alias abb_conn_start= „roslaunch abb_irb2400_moveit_config moveit_planning_execution.launch sim:=false robot_ip:=192.168.1.42"" >> .bash_aliases

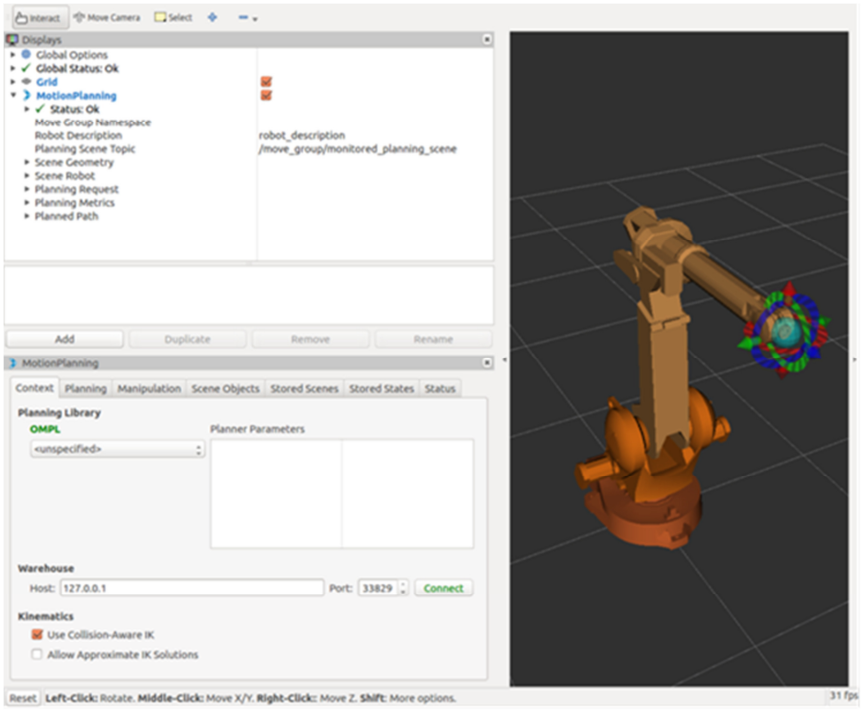
Umjesto pisanja cijele naredbe, sada spajanje na kontroler možemo izvršiti pozivanjem aliasa:

$ abb_conn_start

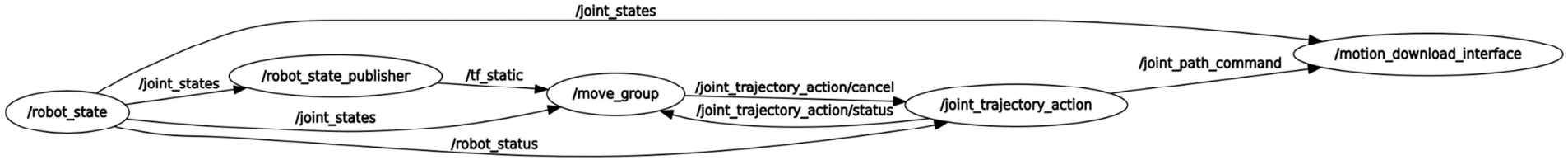
Navedena naredba pokreće potrebne čvorove kao i jezgru ROS okvira kako bi se komuniciralo s kontrolerom robotskog manipulatora. Pokrenuti čvorovi su:

- „/robot_state“ - dohvaća trenutne kuteve zglobova manipulatora s kontrolera robotskog manipulatora
- „/robot_state_publisher“ - proračunava direktnu kinematiku pomoću dohvaćenih trenutnih kuteva zglobova
- „/move_group“ - proračunava inverznu kinematiku i planiranje trajektorije koristeći trenutne i željene kuteve zglobova
- „/joint_trajectory_action“ - dobiva željene kuteve zglobova i prati kretanje robota prema željenim kutevima
- „/motion_download_interface“ - šalje željene kuteve zglobova kontroleru i prati uspješnost slanja.

Pojednostavljeni prikaz pokrenutih čvorova prikazan je na slici 3.6. Uz navedene čvorove, pokreće se i vizualizacijski alat ROS okvira zvan rViz prikazan na slici 3.5. U alatu rViz moguće je pratiti kretanje robotskog manipulatora. Moguće je i dodavati 3D objekte u radni prostor manipulatora radi njihove vizualizacije i izbjegavanja kolizije s tim objektima.



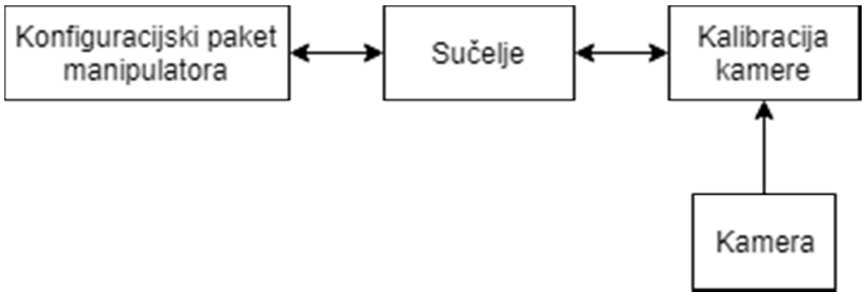
Slika 3.5 Vizualizacijski alat rViz s prikazanim modelom korištenog manipulatora



Slika 3.26 Pojednostavljeni grafički prikaz pokrenutih čvorova

4. Programsko rješenje za upravljanje manipulatorom korištenjem osobnog računala u ROS okviru

U ovom poglavlju opisana je izrada programskog rješenja za primjer kalibracije sustava robotska ruka-kamera iako opisano programsko rješenje može poslužiti za realizaciju različitih zadataka, koji uključuju upravljanje robotskim manipulatorom pomoću računalnog programa, koji se izvodi na osobnom računalu. Programsko rješenje možemo rastaviti na 4 cjeline prikazane na slici 4.1.



Slika 4.1 Cjeline programskog rješenja

Konfiguracijski paket manipulatora izrađen je u prethodnom poglavlju i čvorovi koji ga čine predstavljeni su slikom 4.6. Cjelina kamere sastoji se od jednog čvora. Podrška za rad ROS okvira s korištenom kamerom preuzeta je s repozitorija proizvođača[2]. U nastavku, pojasnit će se programsko rješenje cjeline sučelja. Rješenje je realizirano u programskom jeziku Python. Izrađen je modul za vrlo jednostavno upravljanje manipulatorom te servis koji pokreće taj modul i izvršava zahtjeve za kretanje. Modul sadrži klasu „abbRobot“ kojom su implementirane metode za upravljanje robotskim manipulatorom u prostoru vrha alata i u prostoru kutova zglobova, dohvaćanje trenutne pozicije i orijentacije vrha alata te funkcija za kontrolu i ispis trenutnih vrijednosti kuteva zglobova, pozicije i orijentacije vrha alata na zaslonu računala.

4.1 Sučelje

Na početku je potrebno uključiti potrebne module i biblioteke. Koristi se modul actionlib za kreiranje klijenta za komunikaciju s čvorom putem akcije za upravljanje u području kuteva zglobova. Biblioteka moveit_commander koristi se za upravljanje manipulatorom u prostoru vrha alata koristeći grupu za planiranje trajektorije. Potrebna je i biblioteka rospy za rad s

[2] https://github.com/orbbec/ros_astra_camera

ROS okvirom u Python programskom jeziku. Potrebni su nam tipovi poruka FollowJointTrajectoryAction, FollowJointTrajectoryActionFeedback i FollowJointTrajecotryGoal, koji se nalaze u modulu msg biblioteke control_msgs. Potrebni su i objekti Point, Pose i Quaternion, koji se nalaze u modulu msgs biblioteke geometry_msgs. Za pretvorbu Eulerovih kutova orijentacije u kvaternione, potrebno je uključiti i objekt quaternion_from_euler iz modula transformations biblioteke tf. Iz modula listener biblioteke tf uključit ćemo objekt TransformListener. Preko objekta JointState modula msg biblioteke sensor_msgs dohvaćat ćemo trenutne kuteve zglobova radi njihovog prikaza. Za zadavanje željenih kuteva zglobova, koristi se tip poruke JointTrajectoryPoint iz modula msg biblioteke trajectory_msgs.

```
import actionlib as act
import moveit_commander as mic
import rospy as rp
from control_msgs.msg import FollowJointTrajectoryAction as trajAction
from control_msgs.msg import FollowJointTrajectoryActionFeedback as feedback
from control_msgs.msg import FollowJointTrajectoryGoal as goal
from geometry_msgs.msg import Point, Pose, Quaternion
from sensor_msgs.msg import JointState
from tf.listener import TransformListener
from tf.transformations import quaternion_from_euler as qEuler
from trajectory_msgs.msg import JointTrajectoryPoint
import os
```

Upravljanje robotskim manipulatorom ostvareno je pomoću objekta klase abbRobot. Ovaj objekt omogućava upravljanje manipulatorom u području vrha alata odnosno u području kutova zglobova. Kod inicijalizacije objekta klase abbRobot, inicijalizira se objekt „manip“, koji upravlja manipulatorom u području vrha alata koristeći grupu za planiranje trajektorije, te objekt „client“ koji upravlja manipulatorom u području kutova zglobova, pri čemu za komunikaciju koristi akciju na čvoru „joint_trajectory_action“. Vrsta korištene poruke je FollowJointTrajectoryAction. Program zatim čeka dok objekt „client“ ne uspostavi uspješno komunikaciju sa željenim čvorom. Kreirat ćemo i objekt TransformListener koji stalno čita poruke na temi „/tf“

```
class abbRobot:
    def __init__(self):
        self.manip = mic.MoveGroupCommander("manipulator")
        self.client = act.SimpleActionClient('joint_trajectory_action', trajAction)
        rp.loginfo("Waiting for server joint_trajectory_action.")
        self.client.wait_for_server()
        rp.loginfo("Successfuly connected to server.")
        self.tfListener = TransformListener()
```

Metodu koja će se koristiti za upravljanje manipulatorom u području vrha alata nazvat ćemo „move2Point“. Ova metoda prima pet parametara. Može joj se predati jedna ili više željenih koordinata koje će manipulator obići. Prvi parametar je „self“, koji označava trenutnu instancu klase i on se automatski predaje metodi kada se ona pozove. Drugi parametar je „points“, koji sadrži listu željenih pozicija vrha alata. Treći parametar je „eAngles“, koji sadrži listu željenih orijentacija alata u obliku Eulerovih kutova. Ukoliko se ovaj parametar ne preda tokom pozivanja metode, koristi se predefinirana vrijednost za sve predane pozicije vrha alata. Sljedeći parametar je „ax“, koji definira konvenciju Eulerovih kutova i sastoji se od 4 znaka. Zadnji parametar definira na koji koordinatni sustav manipulatora se odnosi pozicija i orijentacija. Ukoliko se on ne preda tokom pozivanja funkcije, koristi se virtualni zglob „tool0“. Na početku metode radi provjere se ispisuje koliko je predano točaka. Zatim se provjerava je li predan jednak broj pozicija i orijentacija te je li predana točno jedna orijentacija. Ukoliko nije, tada će se ispisati poruka da nije moguće izvršiti kretanje manipulatora. Ukoliko jeste, tada metoda prolazi kroz svaku poziciju, ispiše je, isplanira se trajektorija te se pošalje manipulatoru i čeka se na dovršetak kretanja. Kada se kretanje dovrši, tada se ide na iduću točku.

```
def move2Point(self, points, eAngles=[[0, 0, 0]], ax='sxyz', end_effector='link_6'):
    print "Number of points recieved: ", len(points)
    if (len(points) == len(eAngles) or len(eAngles) == 1):
        for i in xrange(len(points)):
            if rp.is_shutdown():
                print "ROS has been shutdown. Exiting..."
                break
            print i + 1, ". point:", [round(pt, 5) for pt in points[i]]
            p = Point(points[i][0], points[i][1], points[i][2])
            index = i
            if len(eAngles) == 1:
                index = 0
            orient = Quaternion(
                *qEuler(eAngles[index][0], eAngles[index][1], eAngles[index][2], ax))
            self.manip.set_pose_target(
                Pose(p, orient), end_effector_link=end_effector)
            self.manip.go(True)
        rp.loginfo("Moving to multiple points finished.")
    else:
        print "Number of points recieved does not match number of euler angles
received\nneither number of euler angles is 1. Please check the input parameters."
```

Metodu koja se koristi za upravljanje manipulatorom u području kutova zglobova nazvat ćemo „jointAction“. Ova metoda prima dva parametra. Prvi parametar je, kao i kod metode „move2Point“„self“ parametar. Drugi parametar je lista koja sadrži šest kutova zglobova u radijanima za svaki željeni položaj. Na početku programa, ispisuje se koliko je željenih položaja predano. Zatim se kreira poruka koja se šalje serveru akcije. Nakon definirane poruke, prolazi se kroz svaku željenu poziciju manipulatora. Pozicija se ispisuje u terminalu u stupnjevima. Pozicija se zatim šalje serveru akcije i čeka se na rezultat. Tokom čekanja na rezultat, može se dobiti povratna informacija tako da se tokom slanja specificira naziv funkcije, koja će se pozvati kada server oglasi povratnu informaciju. Tu ćemo metodu nazvati „__feedback“.

```
def jointAction(self, jointAngles):
    print "Number of joint angles recieved: ", len(jointAngles)
    jointGoal = goal()
    jointGoal.trajectory.joint_names = [
        'joint_1', 'joint_2', 'joint_3', 'joint_4', 'joint_5', 'joint_6']
    jointGoal.trajectory.points.append(JointTrajectoryPoint())
    for joint in jointAngles:
        print "Going to joint values[degrees]: ", [round(i * 180 / pi, 2) for i in joint]
        jointGoal.trajectory.points[0].positions = joint
        self.client.send_goal(jointGoal, feedback_cb=self.__feedback)
        self.client.wait_for_result()
        print "[Result:] ", self.errorDict[self.client.get_result().error_code]
```

Rezultat kretanja prema željenim zglobovima server vraća kao brojku između -5 i 0 gdje 0 označava uspješno izvršeno kretanje. Kako u terminal ne bi ispisivali brojku nego njeno značenje, definirat ćemo rječnik „errorDict“ u klasi abbRobot, koji sadrži vraćenu brojku kao ključ te njen opis kao vrijednost.

```
errorDict = { 0: "Successful",
              1: "Invalid goal",
             -2: "Invalid joints",
             -3: "Old header timestamp",
             -4: "Path tolerance violated",
             -5: "Goal tolerance violated"}
```

Sada je potrebno definirati metodu povratne veze. Ova funkcija prima samo jedan parametar i to je parametar poruke. Također je potrebno definirati ovu metodu kao statičnu koristeći funkcijski dekorator „@staticmethod“ kako ne bi primala parametar „self“. U funkciji se ispisuje dobivena poruka koja prikazuje razliku između željene i trenutne pozicije robota.

```
@staticmethod
def __feedback(msg):
  print "[Feedback:] ", msg.error
```

Zatim ćemo kreirati funkciju getEEPoint koja će omogućiti korisniku da tokom pisanja svojeg programa može vrlo lako dohvatiti poziciju i orijentaciju vrha alata ukoliko su mu ti podatci potrebni. Funkcija će kao parametar primati tri parametra. Prvi parametar je parametar „self“. Drugi parametar je početni, a treći je krajnji zglob. Rezultat vraća poziciju i orijentaciju krajnjeg zgloba u odnosu na početni. Ako se drugi i treći parametar ne predaju, tada ćemo za početni zglob uzeti zglob „base_link“, koji je zapravo baza manipulatora, dok ćemo za krajnji uzeti „tool0“, koji je zapravo virtualni vrh alata. Kao prvu naredbu pozvat ćemo objekt tfListener, koji je kreiran u konstruktoru, te ćemo preko njega pozvati naredbu „waitForTransform“. Ova naredba prima četiri parametra. Prvi parametar je početni a drugi parametar je krajnji članak. Treći parametar je ROS trenutak u kojem želimo dohvatiti poziciju i orijentaciju. Četvrti parametar je najduže čekanje do nove poruke ukoliko nije već objavljena u tf temi. Za treći parametar predati funkciju Time ROS biblioteke, koja vraća trenutno ROS vrijeme, dok ćemo za četvrti parametar predati funkciju Duration iste biblioteke, koja vraća trenutno ROS vrijeme uvećano za parametar koji se predaje. Kada poruka dođe odnosno nakon čekanja na poruku, dohvaćamo poruku pomoću funkcije lookupTransform i spremamo u varijablu pts i orient. Funkcija prima tri parametra koja su jedna prvim triju parametara prethodne funkcije. U varijablu pts spremamo trenutnu poziciju, dok u orient spremamo trenutnu orijentaciju vrha alata te obje varijable vraćamo kao rezultat funkcije.

```
def getEEPoint(self, start_link='base_link',end_link='tool0'):
  self.tfListener.waitForTransform(start_link, end_link, rp.Time(), rp.Duration(0.5))
  ptData = self.tfListener.lookupTransform(start_link, end_link, rp.Time())
  pts = [pt for pt in ptData[0]]
  orient = [pt for pt in ptData[1]]
  return pts,orient
```

Dodatnu i zadnju funkcionalnost ovom modulu ćemo napraviti tako da korisnik vrlo jednostavno može napraviti program koji na ekranu ispisuje trenutne kuteve zglobova i trenutnu poziciju i orijentaciju alata radi kontrole. Funkcije ćemo kreirati van klase „abbRobot“. Korisnik, kako bi iskoristio ovu funkcionalnost modula, sve što treba je pozvati jointsInfo funkciju. Funkcija prima dva parametra, a to su parametar printoutRate, kojim određujemo koliko često želimo dohvaćanje i ispisivanje kuteva zglobova te pozicije i

orijentacije. Drugi parametar određuje hoće li se ROS čvor kreirati kao anoniman čvor. Ukoliko se taj parametar postavi da je istinit, tada će ROS okvir tokom kreiranja čvora na kraj naziva čvora dodati svoj jedinstveni identifikator. Unutar ove funkcije prvo ćemo kreirati ROS čvor naziva „abb_jointListener“ te postaviti parametar anonymous da bude jednak predanom drugom parametru. Zatim kreiramo objekt tfListener, koji na temi „/tf“ dohvaća i prati objave transformacijskih matrica koje dolaze s kontrolera robota. Nakon toga, kreiramo objekt pretplatnika koji će dohvaćati trenutne kuteve zglobova robota. Pretplatnik kao prvi parametar prima naziv teme na koju se pretplaćuje, drugi parametar je tip poruke koji se koristi u temi, treći parametar je funkcija koja će se pozvati kada dođe nova poruka u odabranu temu te četvrti parametar je dodatni parametar koji će pretplatnik predati pozvanoj funkciji. Zatim ćemo postaviti učestalost provjere poruka i pozivom funkcije „sleep“ narediti programu da pričeka vrijeme koje smo definirali kao učestalost provjere. I na kraju funkcijom spin dajemo informaciju ROS okviru da ovaj program izvodi beskonačno dugo dok ga korisnik ne ugasi.

```
def jointsInfo(printoutRate=0.5, anonym=False):
    rp.init_node("abb_jointListener", anonymous=anonym)
    tfListener = TransformListener()
    listener = rp.Subscriber( "/joint_states", JointState, __listenCb, tfListener)
    rate = rp.Rate(printoutRate)
    rate.sleep()
    rp.spin()
```

Na kraju, potrebno je još kreirati funkciju __listenCb koju će pozivati pretplatnik kojeg smo prethodno kreirali. Funkcija prima dva parametra te će se oba parametra predati automatikom. Prvi parametar je poruka koja sadrži vremenski novije kuteve zglobova, a drugi parametar je objekt tfListener. Kada dođe poruka, potrebno je obrisati sve što je trenutno ispisano u terminalu. Terminal ćemo očistiti pozivom funkcije os.system(„clear“). Zatim ćemo dohvatiti trenutnu poziciju i orijentaciju alata. Nakon dohvaćanja, ispisat ćemo kuteve zglobova i poziciju i orijentaciju vrha alata. Radi lakšeg čitanja trenutnih kuteva zglobova, sve kuteve ćemo pretvoriti u stupnjeve iz radijana.

```
def __listenCb(msg, tfListener):
    clear()
    tfListener.waitForTransform('base_link', 'tool0', rp.Time(), rp.Duration(0.5))
    ptData = tfListener.lookupTransform('base_link', 'tool0', rp.Time())
    print "Joint names         :", msg.name
    print "Joint angles [degree]:", [round(joint * 180 / pi, 2) for joint in msg.position]
    print "-----------"
    print "Point coordinates   :", [round(pt, 5) for pt in ptData[0]]
    print "Quaternions[x,y,z,w]:", [round(q, 5) for q in ptData[1]]
```

Potrebno je još kreirati direktorij unutar direktorija „src“ korisničkog paketa. Direktorij ćemo nazvati „abblib“. Također, unutar direktorija kreiramo novu praznu datoteku naziva „__init__.py“, kako bi Python prepoznao direktorij kao biblioteku. U direktorij „abblib“ spremimo napisani modul pod nazivom „abbCtrl.py“.

4.2 Korisnička poruka za servis kretanja manipulatora

Prije izrade servisa za kretanje manipulatora, potrebno je definirati poruku kojom će se komunicirati s tim servisom. Ukoliko korisnik želi, može koristiti već ugrađene poruke koje dolaze s ROS okvirom. U ovom podpoglavlju opisana je izrada korisničke poruke. Potrebno je kreirati novi direktorij unutar korisničkog paketa naziva „srv“ i unutar njega kreirati datoteku „move_req.srv“ koja će sadržavati definiciju poruke[8]. Nakon kreiranja datoteke, potrebno je dodati naziv datoteke u datoteku „CmakeLists.txt“, koja govori prevoditelju što sve mora uključiti u projekt tokom prevođenja. Potrebno je dodati slijedeće linije koda:

```
add_Service_files(
  FILES
  move_req.srv
)
```

Korisnička poruka za servis se definira datotekom „move_req.srv“ tako da se navedu željeni tipovi podataka koji definiraju zahtjev za kretnjom i odgovora međusobno odvojeni trostrukom crtom. Svaki željeni podatak mora biti u zasebnom retku. U ovoj poruci za zahtjev definiraju se pozicija vrha alata i njegova orijentacija, dok se kao odgovor definira cjelobrojna vrijednost koja označava uspješnost izvršenja zahtjeva za kretnjom.

```
geometry_msgs/Point point
geometry_msgs/Quaternion orientation
---
int8 retCode
```

Kada je korisnička poruka definirana, potrebno ju je prevesti na način da se pokrene prevoditelj „catkin_make“ u korijenskom direktoriju radnog prostora. Poruka je nakon ovoga spremna za korištenje.

4.3 Servis za kretanje

Kako bi ROS okvir uspio komunicirati s čvorom kalibracije kamere, potrebno je kreirati servis koji će komunicirati s tim čvorom. On se realizira datotekom „moveit_service.py“ smještenom u src direktoriju korisničkog paketa. U programsko rješenje servisa za kretanje uključuje se biblioteka rospy za rad s ROS okvirom, modul „abbctrl“, tip poruke kojom se šalje zahtjev za kretnjom i objekt „euler_from_quaternion“ iz modula transformations biblioteke tf.

```
import rospy as rp
from abblib.abbCtrl import abbRobot
from abb2400_sim.srv import move_req
from tf.transformations import euler_from_quaternion as Eulerq
```

Zatim se inicijalizira čvor u ROS okviru s nazivom „moveit_Service“. U čvoru se stvara servis naziva „move_req“ koji koristi tip poruke „move_req“. Svaki puta kada dođe novi zahtjev serveru, poziva se funkcija „serviceHandler“ opisana u nastavku. Izradit ćemo novi objekt klase abbRobot i ispisati na terminal da je čvor pokrenut.

```
rp.init_node("moveit_Service")
moveService=rp.Service("move_req",move_req, serviceHandler)
robot=abbRobot()
rp.loginfo("moveit_Service started")
rp.spin()
```

Funkcija „serviceHandler“, koja se izvršava svaki put kada dođe novi zahtjev za kretanje manipulatora, uzima tražene koordinate i orijentacije vrha alata i sprema ih u zasebne liste te ih prosljeđuje metodi „move2Point“ objekta abbRobot. Razmatrana funkcija je realizirana sljedećim programskim kodom.

```
def serviceHandler(req):
    pt=[req.point.x,req.point.y,req.point.z]
    euler=Eulerq([req.orientation.x,req.orientation.y,req.orientation.z,req.orientation.w])
    robot.move2Point([pt],[euler])
    return 0
```

4.4 Pokretanje programa

Prije pokretanja kalibracijskog programa potrebno je pokrenuti tri čvora: konfiguracijski paket manipulatora, čvor koji će primati sliku s kamere i čvor sučelja koji prima zahtjeve za kretanje manipulatora. Načelno je za svaki čvor potrebno otvoriti novu instancu terminala. Da bi se postupak pokretanja programa pojednostavnio, izrađena je datoteka za pokretanje (engl. *launch file*), koja omogućuje pokretanje svih gore navedenih čvorova jednom naredbom unutar jednog terminala. Kreiramo novi direktorij naziva „launch“. Ta datoteka pod nazivom „startAllNodes.launch“ se nalazi u direktoriju „launch“ unutar istog direktorija gdje se nalazi i direktorij „srv“ za servis. Navedena datoteka prima IP adresu kontrolera robotskog manipulatora. Ukoliko IP adresa nije predana tokom pokretanja datoteke za pokretanje, koristi se IP adresa 172.16.106.185. Datoteka za pokretanje pokreće čvor kamere, čvor servisa te datoteka za pokretanje „moveit_planning_execution.launch“

```
<launch>
  <arg name="robot_ip" default="172.16.106.185"/>

  <include file="$(find abb_irb2400_moveit_config)/launch/moveit_planning_execution.launch">
    <arg name="sim" value="false"/>
    <arg name="robot_ip" value="$(arg robot_ip)"/>
  </include>
  <node name="moveService" pkg="abb2400_sim" type="moveit_service.py" output="screen"/>
  <node name="astra" pkg="astra_camera" type="astra_camera_node" output="screen"/>
</launch>
```

Sada sve čvorove možemo pokrenuti jednom naredbom iz jednog terminala:

$roslaunch abb2400 startAllNodes.launch

4.5 Izrada programa za testiranje

Do sada smo izradili sve potrebne čvorove za rad s nekim sustavom koji će upravljati robotskim manipulatorom, gdje smo dali kao primjer sustav kalibracije kamere, no ukoliko takav sustav nemamo, nismo u mogućnosti kretati manipulator. U ovom podpoglavlju izradit ćemo jednostavan program koji će koristiti modul „abbCtrl“ izrađen u potpoglavlju 4.1. Za početak uključit ćemo potrebne module za rad ovog programa. Potrebni moduli su abbRobot iz izrađenog modula abbCtrl, rospy, kako bi mogli raditi s ROS okvirom te varijablu pi iz biblioteke math, kako bi mogli pretvoriti kuteve iz stupnjeva u radijane.

```
from abblib.abbCtrl import abbRobot
import rospy as rp
from math import pi
```

Zatim ćemo inicijalizirati program kao čvor ROS okvira pod nazivom 'abbMove_Main' tako da čvor nije anoniman. Izradit ćemo objekt klase abbRobot iz modula abbCtrl.

```
rp.init_node('abbMove_Main',anonymous=False)
robot=abbRobot()
```

Slijedeći dio koda izvršavat ćemo sve dok ROS okvir nije primio signal za gašenje. Zatražit ćemo od korisnika unos, gdje će broj 0 označavati vraćanje manipulatora u inicijalni položaj (svi kutevi manipulatora su vrijednosti 0), broj 1 će označavati upravljanje manipulatorom u području vrha alata a broj 2 će označavati upravljanje manipulatorom u području kuteva zglobova. Unos bilo kojeg drugog broja će uzrokovati gašenje programa.

```
while not rp.is_shutdown():
  choice=input("Choose: "+
  "\n0:    Init joints"+
  "\n1:    Manual end effector points"+
  "\n2:    Manual joints"+
  "\nothers:  exit"+
  "\n>> ")
```

Za slučaj da korisnik unese vrijednost 0, nećemo tražiti dodatni unos od korisnika, već ćemo samo pozvati funkciju jointAction s parametrom vrijednosti kuteva zglobova kako bi vratili manipulator u inicijalni položaj.

```
if(choice==0):
  robot.jointAction([[0,0,0,0,0,0]])
```

Ukoliko korisnik na početku unese vrijednost 1, tada ćemo dodatno od korisnika zatražiti unos koordinate vrha alata, dok orijentacije nećemo tražiti od korisnika radi jednostavnijeg zadavanja kretanja. Zatim ćemo te pozicije predati funkciji „move2Point“ objekta robot. Pošto nismo predali orijentaciju vrha alata, za sve Eulerove kuteve koristit će se vrijednost 0 i to u odnosu na koordinatni sustav šestog zgloba.

```
elif(choice==1):
  x=input("Enter x coordinate: ")
  y=input("Enter y coordinate: ")
  z=input("Enter z coordinate: ")
  robot.move2Point([[x,y,z]])
```

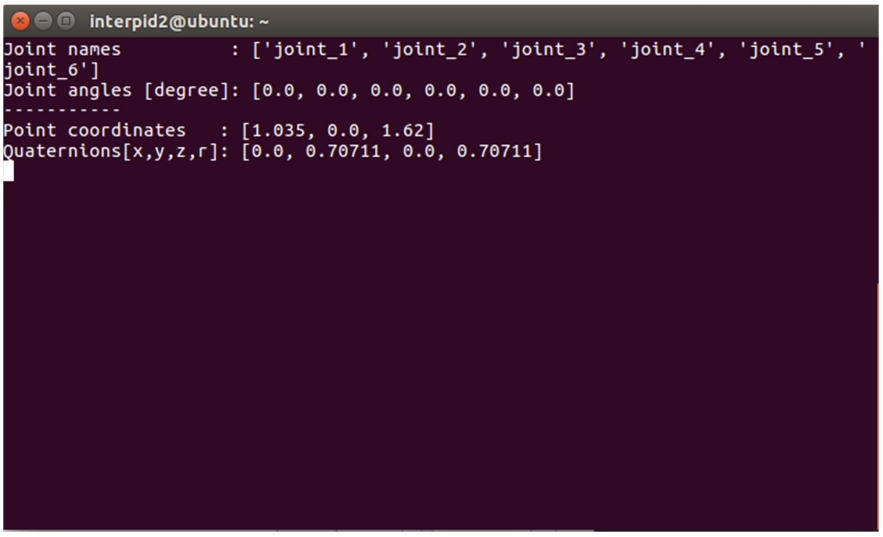
Slijedeći mogući izbor je za vrijednost 2 kada će program upitati korisnika za željene kuteve zglobova na koje želi dovesti manipulator. Inicijalizirat ćemo listu kuteva te ćemo upitati korisnika za svaki zglob željenu vrijednost u stupnjevima. Nakon unosa, svaki zglob ćemo pretvoriti iz stupnjeva u radijane. Kada su svih šest kuteva uneseni, predajemo kuteve funkciji jointAction kako bi pokrenuli kretanje robota. Za sve ostale izbore ćemo prekinuti izvođenje programa. Na posljetku, spremit ćemo program pod nazivom abbCmd.py u mapu abb2400.

```
elif(choice==2):
    jointList=[[0,0,0,0,0,0]]
    print "Enter the joint values in degress"
    for i in xrange(6):
        temp=input("Enter the %i. joint value: "%(i+1))*pi/180
        jointList[0][i]=temp
    robot.jointAction(jointList)
else:
    break
```

Na posljetku, izradit ćemo vrlo jednostavan program koji će pozivati funkciju jointsInfo iz modula abbCtrl i time ćemo imati stalan ispis trenutnih kuteva zglobova i pozicije i orijentacije vrha alata. Postavit ćemo učestalost ispisa na 0.5 Hz i program pokrenuti kao anoniman čvor. Program ćemo spremiti pod nazivom abbPoseNode.py u mapi abb2400. Izgled ispisa prikazan je na slici 4.2.

```
from abblib import abbCtrl

abbCtrl.jointsInfo(0.5,True)
```



Slika 4.2 Prikaz ispisa abbPoseNode.py programa

Program pokrećemo naredbom u terminalu:

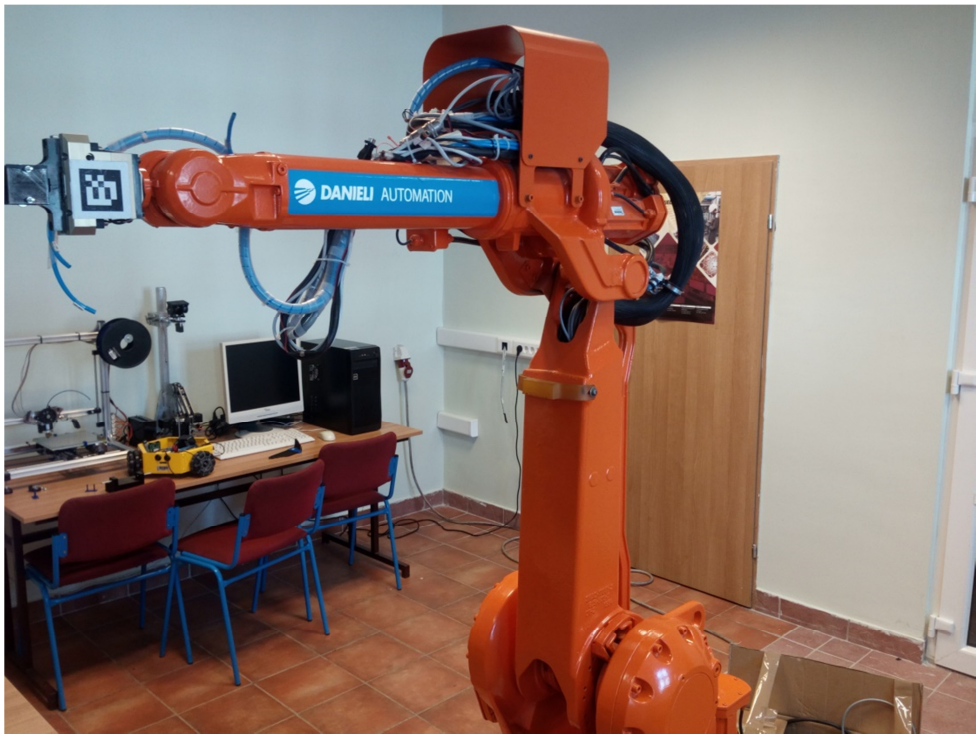
$rosrun abb2400 abbPoseNode.py

5. Eksperimentalno ispitivanje razvijenog programskog rješenja

Funkcionalnost razvijenog programskog rješenja za upravljanje robotskim manipulatorom testirana je pokusom sa stvarnim industrijskim robotskim manipulatorom IRB 2400L. Pokus je opisan u ovom poglavlju. Cilj pokusa je bio utvrditi točnost direktne kinematike ROS okvira u odnosu na direktnu kinematiku koja se izvodi na kontroleru.

5.1 Robotski manipulator

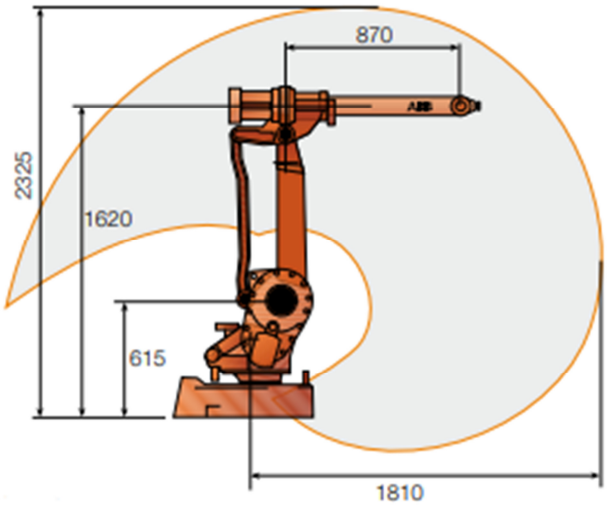
Manipulator na kojem je ispitana kalibracija je manipulator tvrtke ABB. ABB je svjetski poznati proizvođač robotskih manipulatora koji je proizveo i pustio u pogon više od 300.000 robotskih manipulatora. Model korištenog manipulatora je IRB 2400L (slika 5.1.).



Slika 5.1 Robotski manipulator IRB 2400L

Navedeni manipulator je na korištenje ustupila tvrtka DANIELI SYSTEC d.o.o., koja surađuje sa Fakultetom elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.

Model IRB 2400L ima dohvat 1.8 metara i nosivost od 7 kg, te su njegove dimenzije prikazane na slici 6.2.



Slika 5.2 Dimenzije manipulatora IRB 2400L [9]

Navedeni manipulator ima široku primjenu u industriji, a primarne funkcije su mu lučno zavarivanje, prijenos i obrada materijala. Manipulator je veliku nosivost mijenjao za veliko radno područje, uske članke i zglobove, što ga čini optimalnim za ranije navedene primjene.

Robusna konstrukcija koju tvrtka ABB naziva „Foundry Plus“ i minimalistički pristup prilikom dizajna dijelova manipulatora doprinosi visokoj pouzdanosti i dugačkim intervalima između održavanja.

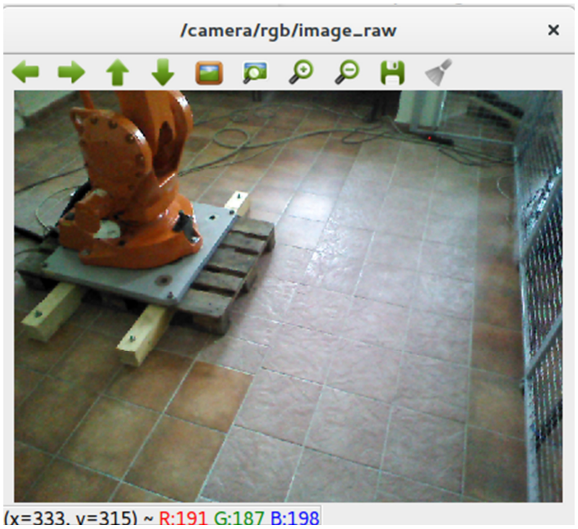
5.2 Testiranje izrađenih rješenja

Na početku pokrenemo sve potrebne programe za upravljanje s manipulatorom upisivanjem naredbe u terminal:

$roslaunch abb2400 startAllNodes.launch

Sada ćemo testirati komunikaciju ROS okvira s kamerom, kako bi utvrdili da ROS okvir ispravno s njom komunicira. Testiranje ćemo provesti s alatom ROS okvira zvanim image_view, koji se spaja na željenu temu kamere koju mu odredimo i prikazuje sliku koju šalje kamera. Kako bi prikazali RGB sliku, potrebno je, u terminalu, napisati slijedeću naredbu dok slika 5.3 prikazuje sliku s kamere.

$rosrun image_view imageview image:=“/camera/rgb/image_raw“



Slika 5.3 Prikaz slike kamere preko alata image_view

Sada kada znamo da kamera prikazuje sliku korištenjem ROS okvira, testirat ćemo kretanje manipulatora te ćemo testirati točnost direktne kinematike izrađene u potpoglavlju 3.1. Pokrenut ćemo program abbPoseNode.py, koji će nam na terminal ispisivati vrijednosti kuteva zglobova te poziciju i orijentaciju vrha alata dobivenu direktnom kinematikom za koju smo izradili kinematičke dodatke. Točnost direktne kinematike ćemo testirati na način da ćemo zadati 8 točaka. Za svaku zadanu vrijednost, očitat ćemo poziciju vrha alata s programa i poziciju vrha alata s kontrolera te za svaku os izračunati prosječno odstupanje. Očitane vrijednosti s kontrolera i terminala te prosječno odstupanje prikazano je u tablici 5.1.

Tablica 5.1 Tablica odstupanja

RB	Vrijednosti očitane s terminala			Vrijednosti očitane s kontrolera			Δx [mm]	Δy [mm]	Δz [mm]
	x [mm]	y [mm]	z [mm]	x [mm]	y [mm]	z [mm]			
1	1035	0	1000	1034.7	-0.1	1000.3	0,30	0,10	-0,30
2	1000	-500	1000	999.7	-499.9	1000.2	0,30	-0,10	-0,20
3	1000	500	1000	999.8	499.8	1000.3	0,20	0,20	-0,30
4	1000	-165.96	1.360	999.4	-165.9	1360.4	0,60	-0,06	-0,24
5	1000	-999.77	1.151	999.7	-999.5	1151.3	0,30	-0,27	-0,13
6	1000	-1000	1500	999.7	-999.6	1500.1	0,30	-0,40	-0,10
7	1319	902.7	1.023	1318.5	902.7	1023.4	0,40	0,27	-0,24
8	1250	0	1750	1249.6	0	1749.9	0,40	0,00	0,10
Prosječno odstupanje:							0.35	-0.03	-0.18

Za sve pozicije vrha alata korištene su iste vrijednosti Eulerovih kuteva i to [0,0,0]. Vidljivo je iz tablice 5.1. da je prosječno odstupanje ispod milimetra te time možemo zaključiti da su modeli (vizualni i kolizijski), URDF datoteka i kinematički dodatci izrađeni poprilično točno te da su odstupanja vrlo mala te čak (zavisno o primjeni) i zanemariva.

6. Zaključak

U ovom diplomskom radu opisano je programsko rješenje za kretanje robotskog manipulatora zasnovano na ROS platformi. Uspostavljena je razvoja platforma koja omogućuje brz i učinkovit razvoj novih naprednih robotskih aplikacija te, s obzirom da je zasnovana na ROS okviru, omogućuje jednostavnu razmjenu razvijenih rješenja. Izrađena platforma omogućuje jednostavno dodavanje novih funkcionalnosti manipulatoru ABB IRB 2400L, dok se, uz manje modifikacije, može primijeniti i na druge tipove robota. Programsko rješenje je eksperimentalno ispitano pomoću robotskog sustava koji se sastoji od industrijskog robotskog manipulatora, RGB-D kamere i osobnog računala. Provedeni pokus pokazao je odstupanje direktne kinematike u odnosu na vrijednosti vrha alata na kontroleru. Utvrđeno je da je prosječno odstupanje ispod milimetra što je za mnoge primjene zanemarivo.

Literatura

[1] https://en.wikipedia.org/wiki/Robot_Operating_System , posjećeno 21.2.2018

[2] M. Guigley, B. Gerkey, W. D. Smart, Programming Robots with ROS, O'Reilly, 2015

[3] L. Joseph, Mastering ROS for Robotics Programming, Packt Publishing Ltd., 2015

[4] http://wiki.ros.org/Industrial , posjećeno 26.2.2018

[5] http://wiki.ros.org/Industrial/Tutorials/Create%20a%20URDF%20for%20an%20Industrial%20Robot , posjećeno 17.3.2018.

[6] http://openrave.org/docs/latest_stable/ , posjećeno 17.3.2018.

[7] http://wiki.ros.org/abb/Tutorials/RobotStudio , posjećeno 24.3.2018.

[8] http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv , posjećeno 30.3.2018.

[9] ABB IRB 2400L data sheet

Sažetak

U radu je obrađena izrada sustava upravljanja robotskim manipulatorom korištenjem ROS okvira, gdje je kao primjer korištenja ovakvog sustava dan program za kalibraciju 3D kamere. Opisano programsko rješenje realizirano je korištenjem programskog jezika Python. U radu je opisana izrada vizualnih i kolizijskih modela korištenjem alata Solidworks i Meshlab te izrada URDF datoteke. Korištenjem URDF datoteke i programskog alata openRAVE opisana je izrada kinematičkih dodataka za direktnu i inverznu kinematiku korištenog robotskog manipulatora. Cjelokupni realizirani sustav testiran je na industrijskom robotskom manipulatoru ABB IRB 2400L. Komunikacija ROS okvira i kamere testirana je korištenjem RGB-D kamere Orbbec Astra. Točnost izrađenog sustava ispitana je eksperimentalno korištenjem zadanih pozicija vrha alata i pozicija očitanih na kontroleru korištenog industrijskog robota.

Ključne riječi: robotski manipulator, Robot Operating System (ROS), model za vizualizaciju, kolizijski model, URDF datoteka

Abstract

In this thesis development of robotic manipulator control using ROS framework is described. As an application example of developed system, 3D camera calibration program is presented. The presented solution was implemented using Python programming language. This thesis also describes creation of visual and collision model using Solidworks and Meshlab tools and creation of a URDF file. Kinematics add-ons were created using the URDF file and openRAVE environment. The system was implemented and tested on an industrial robotic manipulator ABB IRB 2400L. Communication between ROS and the camera was tested using RGB-D camera Orbbec Astra. The accuracy of the developed system was experimentally tested by comparing a sequence of target tool positions with the positions from controller of the used robotic manipulator.

Keywords: robotic manipulator, Robot Operating System, ROS, visualization model, collision model, URDF file

Životopis

Marko Meisel rođen je 20. Srpnja 1994. godine u Osijeku te je odrastao u Valpovu gdje je i završio svoje osnovnoškolsko obrazovanje. Završivši osnovnu školu, 2009. godine upisuje Elektrotehničku i prometnu školu Osijek koju završava 2013. godine stječući zanimanje Tehničar za računalstvo. Iste godine upisuje preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek. Nakon završenog preddiplomskog studija, 2016. godine upisuje diplomski studij računarstva, izborni blok procesno računarstvo na istom fakultetu.

Prilog A - Izrada potrebnih radnih prostora

Korisnički paket možemo kreirati kroz grafičko korisničko sučelje ili koristeći naredbe u terminalu operativnog sustava:

$ mkdir -p catkin_ws/src

Ova naredba kreira direktorij "catkin_ws" i poddirektorij "src" te je sada potrebno u direktoriju radnog prostora pokrenuti prevoditelj kako bi iskonfigurirao radni prostor.

$ cd catkin_ws

$ catkin_make

Naredba “catkin_make” konfigurira radni prostor i kreira dva dodatna direktorija u direktoriju radnog prostora: "devel" (direktorij koji prevoditelj koristi za izradu izvršnih datoteka) i "build" (direktorij u kojeg prevoditelj stavlja gotove izvršne datoteke). Iako je radni prostor kreiran, ROS okvir nema zapisanu putanju do radnog prostora. Putanja do radnog prostora se može dodati pokretanjem "setup.bash" datoteke u direktoriju "devel" koju je prevoditelj kreirao. Problem kod ovakvog pokretanja navedene datoteke je da je putanja dodana samo za terminal u kojem je naredba pokrenuta te se ona briše ukoliko se taj terminal zatvori. Rješenje ovog problema je da tokom pokretanja, terminal automatski pokrene ovu datoteku. Naime postoje dvije datoteke ".bash_rc" i ".bash_aliases" koje svaka instanca terminala izvršava kada se pokrene. Naredba za izvršavanje datoteke "setup.bash" kada se pokrene instanca terminala se može zapisati u bilo koju od te dvije datoteke. Preporuka je da se naredba zapiše u datoteku ".bash_aliases", jer brisanje ove datoteke neće uzrokovati nestabilnost sustava. Zapisivanje naredbe u ".bash_aliases" datoteku možemo izvršiti koristeći terminal:

$ echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bash_aliases

Kako bi se promjene u datoteci izvršile, potrebno je zatvoriti trenutnu instancu terminala i otvoriti novu. Za izradu drugog radnog prostora, postupak odnosno redoslijed naredbi u terminalu je isti:

$ mkdir -p catkin_abb_ws/src

$ cd catkin_abb/src

$catkin_make

$ echo "source ~/cathin_abb_ws/devel/setup.bash" >> ~/.bash_aliases

Također, trenutnu instancu terminala potrebno je zatvoriti i otvoriti novu kako bi promjene postale aktivne. Zatim je potrebno stvoriti korisnički paket unutar radnog prostora. Prilikom kreiranja mogu se navesti paketi o kojima je naš paket ovisan odnosno bez kojih naše programsko rješenje neće raditi. Korisnički paket kreiramo naredbom

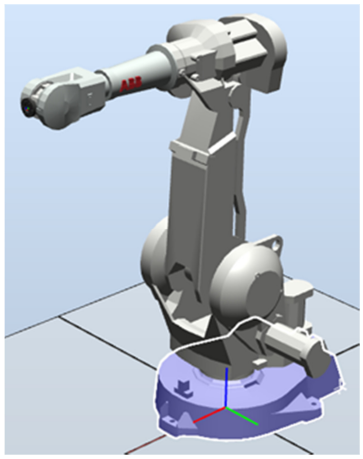
$ catkin_create_pkg abb2400 control_msgs message_generation message_runtime rospy roscpp geometry_msgs std_msgs

Prvi parametar naredbe catkin_create_pkg je naziv našeg korisničkog paketa. Ostali parametri su svi paketi o kojima je izvođenje našeg paketa ovisno.

Prilog B - Prilagodba konfiguracijskog paketa za robot ABB IRB 2400L

B.1 Izmjena vizualnog i kolizijskog modela

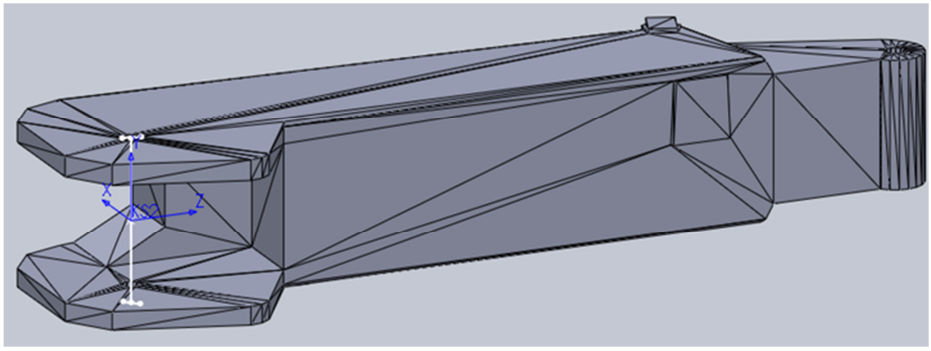
Korišteni 3D model robotskog manipulatora ABB IRB 2400L može se preuzeti sa stranice tvrtke ABB[3] u raznim formatima. Preuzet je cjelokupni 3D model manipulatora u STL formatu. Svaki zglob je potrebno spremiti u zasebnu datoteku i translatirati glavni koordinatni sustav programskog alata Solidworks u odgovarajuću točku. Orijentacija koordinatnog sustava svakog zgloba mora biti ista kao i orijentacija baznog koordinatnog sustava. Orijentacija baznog koordinatnog sustava prikazana je na slici B.1, gdje je crvenom bojom označena x-os, zelenom bojom y-os te plavom bojom z-os.



Slika B.1 Korišten manipulator sa označenim osima baznog koordinatnog sustava

Točka drugog zgloba, u koju trebamo translatirati glavni koordinatni sustav, nalazi se na polovištu linije koja spaja središta provrta donjeg dijela članka. U alatu Solidworks možemo skicirati te linije i pronaći tu točku te u nju postaviti referentni koordinatni sustav. Kod tog skiciranja, uzeti su vanjski promjeri provrta s obje strane. Skicirane linije su prikazane na slici B.2 bijelom bojom, dok je referentni koordinatni sustav prikazan tamno plavom.

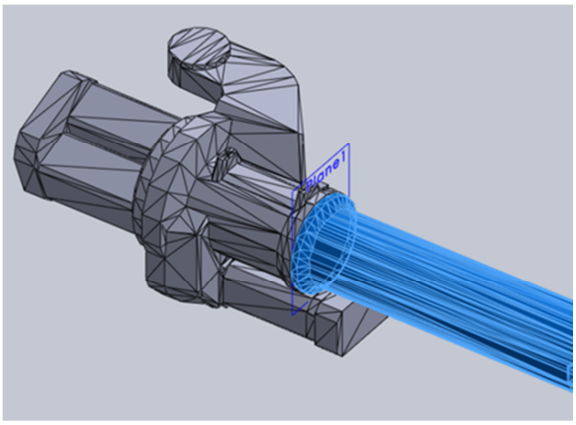
[3] http://new.abb.com/products/robotics/industrial-robots/irb-2400/irb-2400-cad



Slika B.2 Prikaz drugog članka manipulatora s pripadajućim koordinatnim sustavom

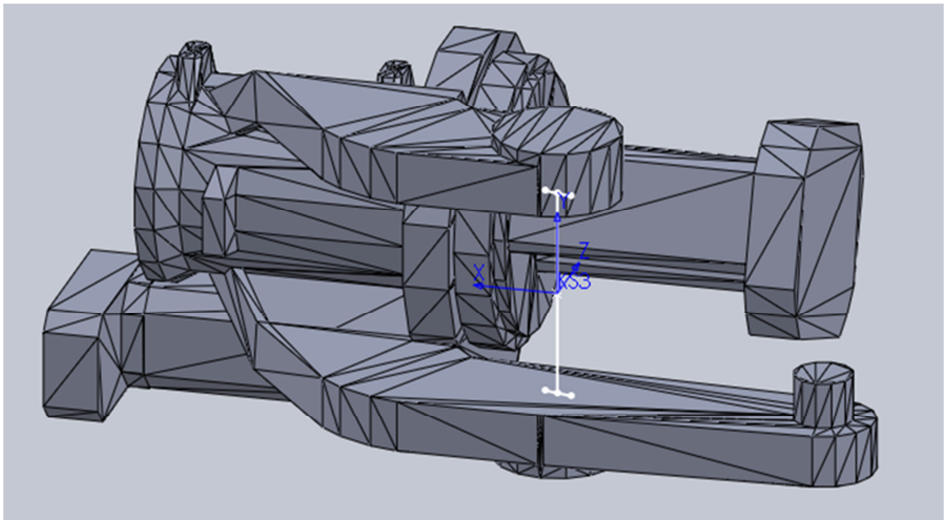
Referentni koordinatni sustav se postavlja u postavkama spremanja u datoteku. Pošto alat Solildworks koristi milimetar kao mjernu jedinicu, dok ROS koristi metar, potrebno je tu jedinicu promijeniti. U postavkama spremanja STL datoteke potrebno je promijeniti mjernu jedinicu u metre. Pri dnu prozora potrebno je izabrati novi glavni koordinatni sustav. Ako se cijeli model ne nalazi u pozitivnim dijelovima sve tri osi, Solidworks će tokom spremanja translatirati koordinatni sustav. Potrebno je označiti opciju "*Do not translate STL output data to positive space*" kako Solidworks ne bi translatirao glavni koordinatni sustav odnosno da koordinatni sustav ostane tamo gdje smo ga postavili.

Prije određivanja koordinatnog sustava trećeg zgloba, podijelit ćemo model četvrtog članka na mjestu gdje dodiruje treći članak. Također ćemo stopiti dio četvrtog članka, koji prolazi kroz treći, s modelom trećeg članka. Ovo odjeljivanje članka nije potrebno napraviti za rad ROS okvira s modelom, ali će pojednostaviti određivanje koordinatnog sustava četvrtog članka i vizualizaciju u ROS okviru. Model trećeg članka, koji je stopljen s odvojenim dijelom četvrtog članka, prikazan je na slici B.3 osjenčan sivom bojom. Također, na istoj slici prikazan je konačni model četvrtog članka osjenčan plavom bojom i ravnina odvajanja četvrtog članka.



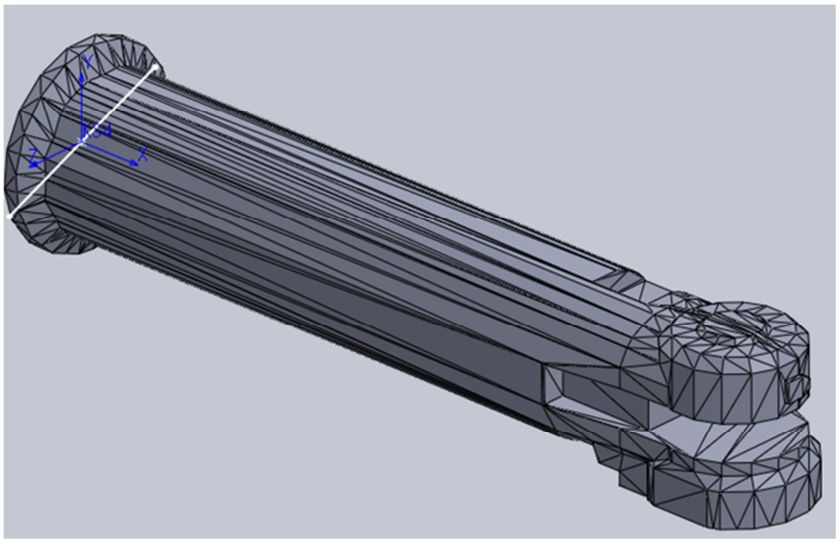
Slika B.3 Prikaz odjeljivanja trećeg i četvrtog članka s ravninom

Točku trećeg zgloba u koju trebamo translatirati glavni koordinatni sustav najlakše ćemo naći koristeći drugi članak. Pronaći ćemo polovište linije koja spaja središta provrta gornjeg dijela drugog članka i u tu točku postaviti referentni koordinatni sustav. Zatim ćemo spremiti model trećeg zgloba na isti način kao što je opisano za drugi zglob, pazeći pri tom da se za novi glavni koordinatni sustav odabere referentni koordinatni sustav KS3. Model trećeg članka s pripadajućim referentnim koordinatnim sustavom prikazan je na slici B.4.



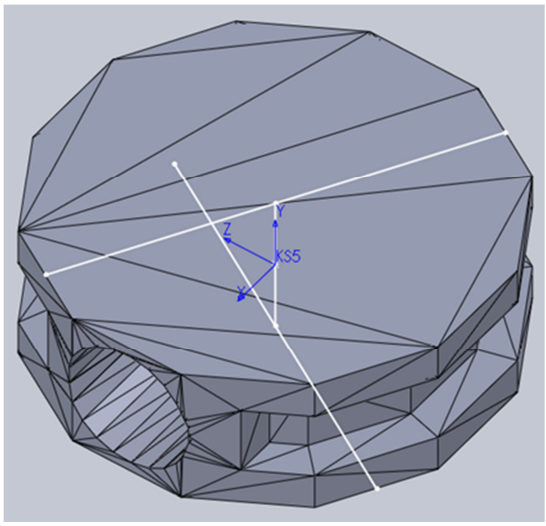
Slika B.4 Prikaz trećeg članka manipulatora s pripadajućim koordinatnim sustavom

Točku četvrtog zgloba, u koju trebamo translatirati glavni koordinatni sustav, možemo pronaći koristeći početak četvrtog članka, tako da pronađemo središte promjera početnog dijela. Nakon toga, postavimo referentni koordinatni sustav u središte. Zatim spremimo model četvrtog članka kao i dosadašnje članke pazeći da se za novi glavni koordinatni sustav odabere referentni koordinatni sustav KS4. Model četvrtog članka s pripadajućim referentnim koordinatnim sustavom prikazan je na slici B.5.



Slika B.5 Prikaz četvrtog članka manipulatora s pripadajućim koordinatnim sustavom

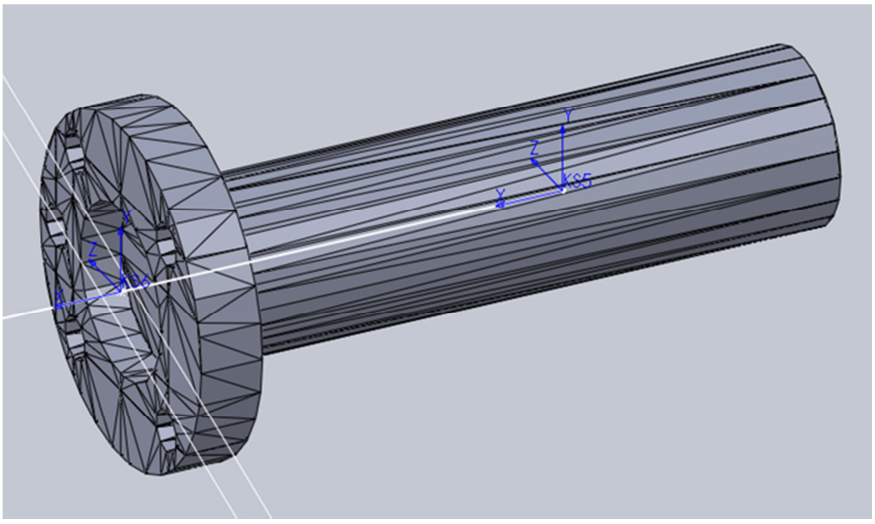
Točka petog zgloba u koju trebamo translatirati glavni koordinatni sustav nalazi se u polovištu linije koja spaja središte gornje i donje plohe. Nakon što odredimo tu točku, u nju postavimo referentni koordinatni sustav. Zatim spremimo model petog članka kao i dosadašnje članke pazeći da se za novi glavni koordinatni sustav odabere referentni koordinatni sustav KS5. Model petog članka zajedno s pripadajućim referentnim koordinatnim sustavom prikazan je na slici B.6.



Slika B.6 Prikaz petog članka manipulatora s pripadajućim koordinatnim sustavom

Točka šestog zgloba nalazi se u samom vrhu šestog članka. Točku u koju trebamo translatirati glavni koordinatni sustav najlakše ćemo dobiti tako da postavimo ravninu koja prolazi kroz plohu vrha šestog članka. Zatim povučemo liniju iz ishodišta referentnog koordinatnog sustava KS5 tako da je paralelna sa x-osi KS5 i prolazi kroz nacrtanu ravninu. Točka gdje se sijeku ravnina i pravac je točka u kojoj nacrtamo ishodište referentnog sustava KS6. Zatim spremimo model šestog članka kao i dosadašnje članke, pazeći da se za novi glavni

koordinatni sustav odabere referentni koordinatni sustav KS6. Model šestog članka s pripadajućim koordinatnim sustavom KS6 i koordinatnim sustavom prijašnjeg članka prikazani su na slici B.7.

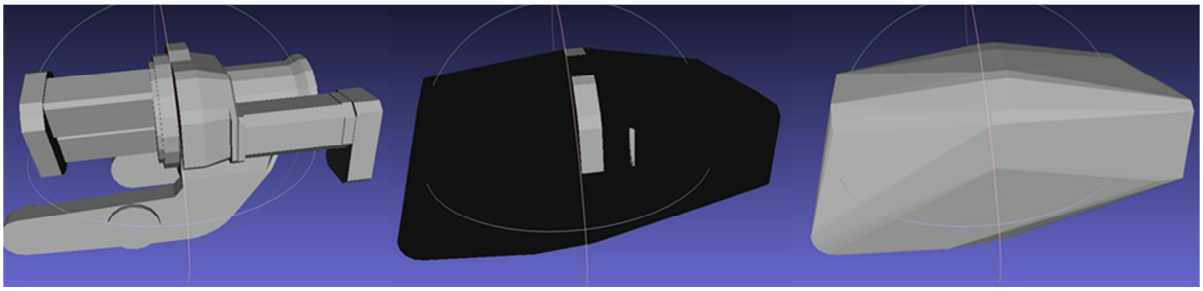


Slika B.7 Prikaz šestog članka manipulatora s pripadajućim koordinatnim sustavom

ROS okvir zahtjeva da modeli članaka robotskog manipulatora budu u collada datoteci s dodatkom .dae. Solidworks nema podršku za spremanje modela u taj oblik datoteke, tako da je potrebno sve STL datoteke pretvoriti u collada datoteke. Pretvaranje iz jednog u drugi oblik možemo napraviti pomoću alata MeshLab. Otvorimo STL datoteku u alatu MeshLab i spremimo svaki model članka zasebno u dae datoteku. Svaki članak nazovemo imenom "link_x.dae", gdje je x redni broj članka kojeg se sprema.

Svaki model kojeg smo spremili, sastoji se od skupa geometrijskih oblika koji su postavljeni na određen način u prostoru, kako bi zajedno tvorili taj model. Ovaj način postavljanja geometrijskih oblika bez razmaka i preklapanja između oblika u prostoru naziva se teselacija. Teselacija modela članaka koje smo spremili u alatu Solidworks je vrlo visoka. Kada bi se pomoću ovakvog modela radio proračun sudara robotskog manipulatora sa samim sobom (engl. *Autocollision*), proračun bi trajao predugo, zbog toga što je potrebno provjeriti svaki geometrijski oblik. Iz tog razloga potrebno je smanjiti teselaciju modela. Smanjenjem teselacije izgubit ćemo detalje modela ali ćemo uvelike ubrzati proračun sudara te se ovakav model naziva kolizijski model. Za izradu kolizijskog modela koristit ćemo dobivene vizualne modele pojedinog članka i alat MeshLab. Otvorimo model članka u alatu MeshLab i primijenimo filtar konveksna ljuska (engl. *Convex Hull*). Zavisno o modelu, postoji mogućnost da normale poligona neće biti ispravno orijentirane. Ti poligoni bit će prikazani crnom bojom. U tom slučaju potrebno je okrenuti orijentaciju normala modela (engl. *Invert*

Faces Orientation). Ukoliko orijentacija normala nije jednolika (prikazani poligoni crnom bojom), potrebno je ponovno orijentirati koherentno sve poligone (engl. *Re-Orient all faces coherently*)[7]. Na slici B.8, lijevo je prikazan početni model trećeg članka, u sredini isti model s primijenjenim filtrom konveksna ljuska i desno model nakon ponovne orijentacije normala poligona.



Slika B.8 Izrada kolizijskog modela trećeg članka

Ove transformacije vizualnog u kolizijski model primjene se za sve modele članaka i spreme se pod istim imenom ali ovaj puta u STL formatu. Postojeće vizualne i kolizijske modele je potrebno zamijeniti novima. U konfiguracijskom paketu nalazi se direktorij "meshes". U tom direktoriju se nalaze direktoriji "irb2400" i "irb2400_common". Oba direktorija sadrže dva istoimena direktorija a to su direktoriji "visual" i "collision". U oba "visual" direktorija zamijenimo postojeće vizualne modele dok u oba "collision" direktorija zamijenimo kolizijske modele s novima.

B.2 Izmjena URDF datoteke

Nakon izrađenih vizualnih i kolizijskih modela, potrebno je izmijeniti URDF datoteku. URDF datoteka sadrži opis odnosa svakog članka robotskog manipulatora. Sastoji se od dva dijela: prvi dio za pojedini članak sadrži putanju do datoteka koja sadrže vizualni i kolizijski model članka. Pošto su postojeći modeli zamijenjeni novima, ove putanje nećemo mijenjati. Drugi dio datoteke sadrži udaljenosti između pojedinog koordinatnog sustava zgloba i prethodnog, tip zgloba (translacijski ili rotacijski), os rotacije ili translacije zgloba (zavisno o tipu), nazivi prethodnog i slijedećeg zgloba za svaki zglob te maksimalni otkloni zgloba u oba smjera (za translacijske zglobove u metrima i za rotacijske u radijanima). U datoteci je potrebno izmijeniti udaljenosti između koordinatnih sustava. Udaljenosti vrlo lako možemo pronaći pomoću alata Solidworks, gdje imamo postavljene koordinatne sustave za svaki zglob iz prethodnog podpoglavlja. Mjerenje u alatu Solidworks nam pokazuje udaljenosti po

pojedinim osima kao i euklidsku udaljenost između dvije točke. Primjer drugog dijela URDF datoteke za treći zglob s već izmijenjenim vrijednostima prikazan je u slijedećem kodu:

```
<joint name="joint_3" type="revolute">

   <origin rpy="0 0 0" xyz="0 0 0.855"/>
   <parent link="link_2"/>
   <child link="link_3"/>
   <axis xyz="0 1 0"/>
   <limit effort="0" lower="-1.0472" upper="1.1345" velocity="2.618"/>
</joint>
```

Udaljenost između prvog i drugog zgloba također možemo izmjeriti mada nemamo nacrtan koordinatni sustav prvog zgloba. Razlog tome je što se ishodište baznog koordinatnog sustava i koordinatnog sustava prvog zgloba nalaze u istoj točki, pa razmatranu udaljenost možemo izmjeriti kao udaljenost između baznog koordinatnog sustava i koordinatnog sustava drugog zgloba. U tablici B.1 prikazane su izmjerene udaljenosti u metrima za svaki zglob po pojedinim osima.

Tablica B.0.1 Tablica udaljenosti pojedinog zgloba u odnosu na prethodni

Naziv zgloba	Udaljenost			
	U odnosu na zglob	x[m]	y[m]	z[m]
joint_1	base_link	0	0	0
joint_2	joint_1	0.1	0	0.615
joint_3	joint_2	0	0	0.855
joint_4	joint_3	0.306	0	0.15
joint_5	joint_4	0.564	0	0
joint_6	joint_5	0.065	0	0

B.3 Izrada datoteka direktne i inverzne kinematike

Za sada imamo izmijenjene vizualne i kolizijske modele te opisnu URDF datoteku. Koristeći URDF datoteku potrebno je sada izmijeniti datoteke koje se koriste za rad robotskog manipulatora. Datoteke koje trebamo izmijeniti nalaze se u "src" direktoriju unutar direktorija "irb2400_kinematics". Direktorij "irb2400_kinematics" se nalazi unutar "abb_irb2400_moveit_plugins" direktorija u konfiguracijskom paketu. Ovdje se nalaze dvije

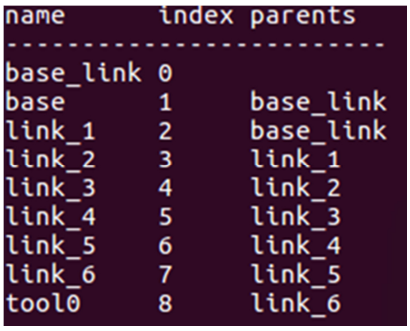
datoteke: datoteka za rješavanje direktne i inverzne kinematike te pomoćna datoteka za rješavanje direktne i inverzne kinematike. Obje datoteke potrebno je zamijeniti novima koje ćemo izraditi koristeći programski alat openRAVE. Prije svega, potrebno je izmijenjenu URDF datoteku pretvoriti u collada datoteku. URDF datoteku ćemo pretvoriti u collada datoteku koristeći alat ROS okvira collada_urdf i terminal [5]. Koristeći terminal promijenimo trenutni radni direktorij u onaj gdje se nalazi URDF datoteka te pokrenemo slijedeću naredbu:

$ rosrun collada_urdf urdf_to_collada irb2400.urdf irb2400.dae

Navedena naredba uzet će datoteku "irb2400.urdf", pretvoriti će ju u collada datoteku i spremiti pod nazivom "irb2400.dae" u istom direktoriju gdje nam se nalazi originalna URDF datoteka. Podatke o člancima možemo dobiti naredbom:

$ openrave-robot.py irb2400.dae --info links

Ova naredba nam daje podatke o rednim brojevima pojedinog članka i njegovom prethodniku ako ga ima. Primjer ispisa prikazan je na slici B.9.



Slika B.9 Ispis svih članaka collada datoteke

Zatim je potrebno izraditi datoteku za rješavanje direktne i inverzne kinematike. Ovu datoteku izradit ćemo koristeći ikfast metodu te ju možemo pokrenuti pomoću naredbe:

$ python `openrave-config --python-dir`/openravepy/_openravepy_/ikfast.py --robot=irb2400.dae --iktype=transform6d --baselink=1 --eelink=8 –savefile=abb_irb2400_manipulator_ikfast_solver.cpp

Ova naredba pokreće python skriptu ikfast.py koja kreira datoteku u programskom jeziku C++. Ova skripta prima nekoliko parametara:

- robot – naziv collada datoteke koju smo kreirali s alatom collada_urdf
- iktype – ovim parametrom se određuje tip rješavanja inverzne kinematike
- base_link – redni broj baznog članka
- eelink – redni broj članka hvataljke

• savefile – naziv datoteke u koju će se spremiti izrađen kod za rješavanje direktne i inverzne kinematike

Prije izrade pomoćne datoteke, provjerit ćemo ispravnost rada datoteke za rješavanje direktne i inverzne kinematike. Ispravnost rada ove datoteke provjerit ćemo koristeći datoteku ikfastdemo.cpp[4] . Ovu datoteku otvorimo pomoću nekog uređivača teksta (npr. gedit, vim, nano itd.) te izmijenimo naziv datoteke na 63. liniji tako da pod navodnicima zapisana datoteka koju želimo testirati odnosno ova linija treba izgledati na slijedeći način:

#include "abb_irb2400_manipulator_ikfast_solver.cpp"

Spremimo izmjene te je sada potrebno ovu datoteku pretvoriti u izvršnu datoteku pomoću g++ prevoditelja kojeg pozovemo naredbom:

$ g++ ikfastdemo.cpp -lstdc++ -llapack -o compute -lrt

Prevoditelj stvara novu izvršnu datoteku naziva "compute" koja izvršava računanje direktne i inverzne kinematike. Prije pokretanja datoteke, potrebno je znati neka od rješenja za direktnu i inverznu kinematiku kako bi se moglo usporediti s onim što izvršna datoteka izračuna. Poznato nam je od prije rješenje direktne kinematike kada su svih šest kuteva zglobova jednaki 0. U tom se slučaju vrh alata robotskog manipulatora nalazi u točki x=1.035 , y=0 i z=1.62. Koordinate su zapisane u metrima. Sada kada znamo barem jedno rješenje direktne kinematike, možemo ju provjeriti pomoću naredbe:

$ ./compute fk 0 0 0 0 0 0

Parametar fk označava da se traži od izvršne datoteke izračun direktne kinematike (engl. *Forward Kinematics*) te ostalih šest parametara označava kuteve zglobova robota. Rezultat proračuna prikazan je na slici B.10.

```
Found fk solution for end frame:

 Translation:  x: 1.035000  y: 0.000000  z: 1.620000

    Rotation     0.000000   0.000000   1.000000
      Matrix:    0.000000   1.000000   0.000000
                -1.000000   0.000000   -0.000000
```

Slika B.10 Ispis direktne kinematike

Nakon glavne datoteke sada je potrebno izraditi pomoćnu datoteku za rješavanje direktne i inverzne kinematike. Naredba koju ćemo pokrenuti služi za izradu pomoćne datoteke ukoliko se kreira novi konfiguracijski paket iz razloga što naredba osim što izrađuje pomoćnu datoteku također i zapisuje postavke prevoditelja u direktoriju

[4] https://raw.githubusercontent.com/davetcoleman/clam_rosbuild/master/clam_ik/src/ikfastdemo.cpp

"abb_irb2400_moveit_plugins". Problem je što izmjenjujemo postojeći direktorij gdje su postavke već zapisane pa će naredba zapisati postojeće postavke. Zapisivanje tih postavki će uzrokovati grešku tokom prevođenja. Kako bi spriječili da se dogodi greška tokom prevođenja, potrebno je u direktoriju "src" direktorija "abb_irb2400_moveit_plugins" napraviti sigurnosnu kopiju datoteka "CmakeLists.txt" i "package.xml". Kada je sigurnosna kopija izrađena, možemo se vratiti u direktorij "urdf". Sada možemo pokrenuti naredbu za izradu pomoćne datoteke za rješavanje direktne i inverzne kinematike:

$ rosrun moveit_ikfast create_ikfast_moveit_plugin.py abb_irb2400 manipulator abb_irb2400_moveit_plugins abb_irb2400_manipulator_ikfast_solver.cpp

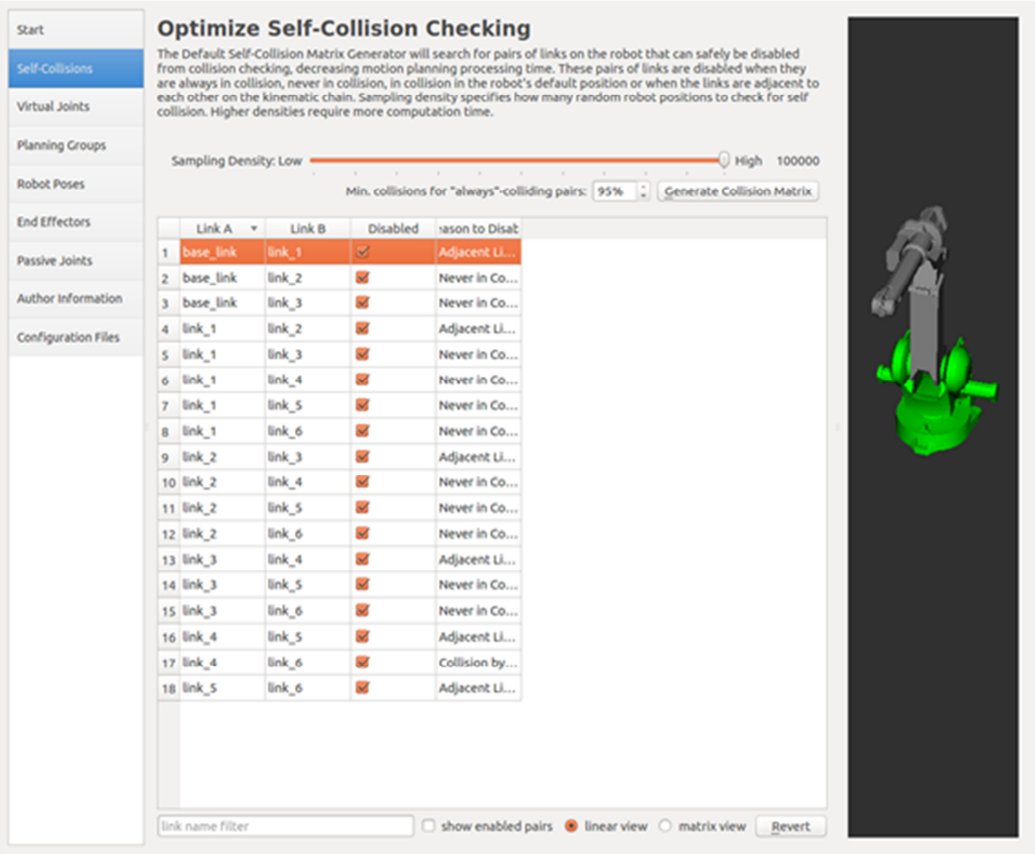
Ova naredba pokreće skriptu koja kreira pomoćnu datoteku za rješavanje direktne i inverzne kinematike, u tom direktoriju kreira direktorij "src" i u njega kopira glavnu i pomoćnu datoteku, kreira skriptu "update_ikfast_plugin.sh" za automatsku izmjenu navedenih datoteka te dodaje postavke prevoditelja. Naredba prima parametre:

- abb_irb2400 – naziv robotskog manipulatora
- manipulator – naziv grupe za planiranje trajektorije
- abb_irb2400_moveit_plugins – naziv direktorija za dodatke
- abb_irb2400_manipulator_ikfast_solver.cpp – naziv glavne datoteke za direktnu i inverznu kinematiku

Zatim je potrebno otići u direktorij "src" direktorija "abb_irb2400_moveit_plugins". U direktoriju je potrebno obrisati datoteke "package.xml" i "CMakeList.txt" te vratiti istoimene starije datoteke koje smo stavili u sigurnosnu kopiju. Unutar ovog direktorija je potrebno premjestiti direktorij "src" u direktorij "ir2400_kinematics". Zadnji korak pri izmjeni postojećeg konfiguracijskog paketa je pokrenuti MoveIt! Asistenta kako bi se ponovno generirala matrica kolizije i povezala glavna te pomoćna datoteka za rješavanje direktne i inverzne kinematike:

$roslaunch abb_irb2400_moveit_config setup_assistant.launch

U prozoru MoveIt! Asistenta pod karticom „self-collisions“ potrebno je ponovno generirati maticu kolizije (Sl. B.11).



Slika B.10.1 Prikaz MoveIt! asistenta prilikom proračuna matrice kolizije

Zatim pod zadnjom karticom „configuration files“ generiramo paket. Sada je postupak izmjene konfiguracijskog paketa robotskog manipulatora dovršen te je potrebno pokrenuti prevoditelj u korijenskom direktoriju radnog prostora.

$ cd catkin_abb_ws

$ catkin_make

Prilog C - Dodatni programski alati

Za rad s 3D modelom robota korišten je 3D CAD alat Solidworks[5] tvrtke Dassault Systèmes pomoću kojeg će se iz cjelokupnog modela robotskog manipulatora kreirati svaki potrebni članak robota zasebno. Pošto je ovaj alat izrađen za operativni sustav Windows, potrebno je na Linux računalo instalirati virtualizacijski alat koji će pokretati taj operativni sustav kao virtualni stroj. Korišten je virtualizacijski alat VirtualBox[6] tvrtke Oracle . Za izradu kolizijskog modela manipulatora, korišten je programski alat MeshLab[7] . Za izradu datoteka potrebnih za izračunavanje inverzne i direktne kinematike korišten je programski alat openRAVE[8][6]. Iako ROS podržava simulaciju rada robotskog manipulatora, za simulaciju korišten je alat RobotStudio[9] tvrtke ABB. Ovim putem simuliramo ne samo rad robotskog manipulatora, već i socket komunikaciju između ROS okvira i kontrolera manipulatora. RobotStudio je pokretan na Windows virtualnom stroju.

[5] https://launch.solidworks.com
[6] https://www.virtualbox.org/wiki/VirtualBox
[7] www.meshlab.net
[8] http://openrave.programmingvision.com/wiki/index.php/Main_Page
[9] http://new.abb.com/products/robotics/robotstudio