Izrada sustava upravljanja robotske ruke za opsluživanje cnc glodalice

Sladonja, Ivan

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka, Faculty of Engineering / Sveučilište u Rijeci, Tehnički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:190:510838

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Sveučilišni diplomski studij strojarstva

Diplomski rad

IZRADA SUSTAVA UPRAVLJANJA ROBOTSKE RUKE ZA OPSLUŽIVANJE CNC GLODALICE

Ivan Sladonja

Rijeka, srpanj 2022. 0069067770

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Sveučilišni diplomski studij strojarstva

Diplomski rad

IZRADA SUSTAVA UPRAVLJANJA ROBOTSKE RUKE ZA OPSLUŽIVANJE CNC GLODALICE

Mentor: Doc. dr. sc. Kristina Marković

Komentor: Prof. dr. sc. Zoran Jurković

Ivan Sladonja

Rijeka, srpanj 2022. 0069067770

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA DIPLOMSKE ISPITE

Rijeka, 12. ožujka 2021.

Zavod: **Zavod za konstruiranje u strojarstvu**
Predmet: **Konstrukcijski elementi robota**
Grana: **2.11.01 opće strojarstvo (konstrukcije)**

ZADATAK ZA DIPLOMSKI RAD

Pristupnik: **Ivan Sladonja (0069067770)**
Studij: Diplomski sveučilišni studij strojarstva
Modul: Konstruiranje i mehatronika

Zadatak: **Izrada sustava upravljanja robotske ruke za opsluživanje CNC glodalice / Development of robotic arm control system for CNC milling machine feeding**

Opis zadatka:

Uloga robotike u industrijskim postrojenjima je sve značajnija. Zadatak ovog rada je izrada sustava upravljanja 3D tiskane open source robotske ruke za opsluživanje laboratorijske CNC glodalice. Za upravljanje robotske ruke će se koristiti operacijski sustav Robot Operating System - ROS. U radu je potrebno navesti korištenu literaturu i druge izvore informacija.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Sladonja

Zadatak uručen pristupniku: 15. ožujka 2021.

Mentor:

Doc. dr. sc. Kristina Marković

Prof. dr. sc. Zoran Jurković (komentor)

Predsjednik povjerenstva za diplomski ispit:

Prof. dr. sc. Kristian Lenić

IZJAVA

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studiranja na Tehničkom fakultetu uz korištenje navedene literature.

Ivan Sladonja

ZAHVALA

Zahvaljujem se mentorici doc. dr. sc. Kristini Marković i komentoru prof. dr. sc. Zoranu Jurkoviću te asistentici Maji Dundović na svim sugestijama i smjernica u realizaciji ovog diplomskog rada.

Posebnu zahvalu posvećujem svojoj obitelji, ocu Urošu, majci Ines i bratu Branku na pruženoj podršci i savjetima koji su mi uvelike koristili tokom studija.

Za kraj se zahvaljujem Adeli na pruženoj motivaciji i razumijevanju.

Sadržaj

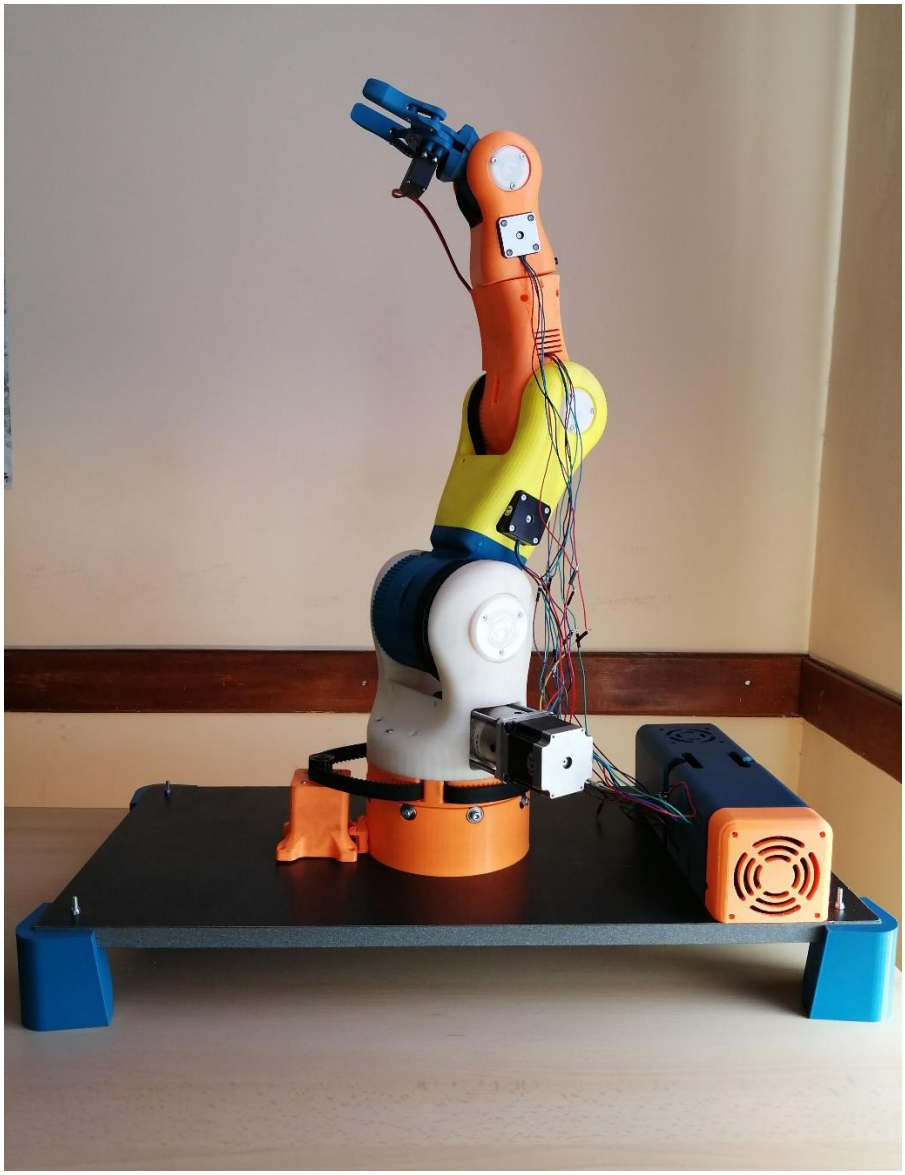
1. UVOD

Industrijske robotske ruke definiraju se kao mehanički mehanizmi koji se sastoje od najmanje tri osi rotacije te se koriste za automatizaciju proizvodnih procesa, čime se smanjuje ili u potpunosti eliminira potreba za ljudskom interakcijom. Robotske ruke automatski izvršavaju zadatke vezane za proizvodnju kroz programiranje ili kroz navođenje pomoću robotskog sustava. Uporaba industrijskih robotskih ruka postala je prilično uobičajena pojava u proizvodnim linijama.

Prvotna primjena robotskih ruka bila je u automobilskoj industriji, no napredak u tehnologiji doveo je do toga da sve više industrija implementira korištenje industrijske robote, uključujući farmaceutsku, prehrambenu i zrakoplovnu industriju. Industrijske robotske ruke mogu, između ostalog, automatizirati zavarivanje, bojanje, montažu, rukovanje materijalom, uklanjanje materijala, pakiranje i mnoge druge procese. Njihovo područje uporabe konstantno raste zbog sve veće preciznosti, točnosti i cjenovne pristupačnosti. Kroz automatizaciju s industrijskim robotskim rukama povećava se produktivnost, poboljšava kvaliteta proizvoda, skraćuje se vrijeme ciklusa i smanjuju troškovi. Proizvodni procesi su pojednostavljeni što rezultira ukupno učinkovitom proizvodnom linijom. Mnoge robotske ruke imaju višenamjenske mogućnosti, što omogućava jednoj robotskoj ruci da izvrši nekoliko faza u proizvodnom procesu.

Industrijski roboti mogu imati deset ili više osi rotacije, no najčešće je slučaj da robotske ruke imaju od tri do šest osi rotacije. Industrijske robotske ruke dizajnirane su za rad s rasponom pokreta koji oponaša, ili je sličan, ljudskoj ruci. Broj osi rotacije koje robot ima određuje njegove stupnjeve slobode (eng. *Degrees of freedom*). Šest osni roboti su najpopularniji za proizvodnju jer je njihov raspon pokreta najsličniji ljudskom.

U ovom je radu opisan postupak izrade sustava upravljanja za robotsku ruku. Robotska ruka koja je korištena u svrhu ovog diplomskog rada je *BCN3D MOVEO* (slika 1.1.) . Navedena robotska ruka ima pet osi rotacije, a konstruirali su je inženjeri tvrtke *BCN3D* u suradnji s Katalonskim odjelom za edukaciju. Njena je struktura kompletno prilagođena korištenju aditivnih tehnologija, tj. konstruirana je na način da se kompletna struktura robotske ruke proizvede uporabom 3D printera. Konstruktorima ove robotske ruke je također bio cilj educirati i potaknuti krajnjeg korisnika da stekne nova znanja i vještine u području 3D printanja kao i u području robotike. Iz tog su razloga sve potrebne CAD i STL datoteke dostupne na internetskoj stranici navedene tvrtke te je također moguće modificirati izgled i dizajn po vlastitoj želji, što je i navedeno u licenci.



Slika 1.1. Robotska ruka BCN3D

Za pokretanje zglobova koriste se koračni motori Nema 17 (eng. *Stepper motors*), a za otvaranje i zatvaranje prihvatnice (eng. *Gripper*) koristi se servo motor. Prijenos između motora i zglobova je ostvaren putem zupčastog remena dok je prijenos između prihvatnice i servo motora ostvaren spregom zupčanika.

Komunikacija između motora i računala je ostvarena koristeći *Arduino Mega 2560* mikroračunalo na kojeg je nadodana nadzorna ploča (eng. *Shield*) *RAMPS 1.4* koja može upravljati do pet koračnih motora s regulacijom koraka do 1/16 koraka. Komunikacija s motorima je ostvarena

koristeći *Driver TB6560* koji omogućuje regulaciju izlazne struje od 0.3 do 3.0 ampera kao i regulaciju koraka motora na puni korak, polukorak, 1/8 koraka te 1/16 koraka.

Softverski paket korišten za upravljanje navedenom robotskom rukom je Robotski operacijski sustav (eng. *Robot operating system*) ili skraćeno ROS. ROS sačinjava skup softverskih knjižnica, alata i konvencija koji uvelike pomažu u razvoju i izradi robotskih sustava. Kako bi korisnik u potpunosti mogao iskoristiti blagodati ROS-a potrebno je također posjedovati vještine programiranja u programskom jeziku „Python“ i/ili „C ++“ kao i osnovna znanja u Linux operacijskom sustavu. Iako su zadnje verzije ROS-a dostupne i na Windows operacijskim sustavima u ovom je radu opisan proces izrade sustava upravljanja koristeći ROS *Melodic Morenia* (slika 1.2.) instaliran na operacijskom sustavu Ubuntu 18.04 (Bionic).

Slika 1.2. Plakat robotskog operacijskog sustava[1]

2. ROBOTSKI OPERACIJSKI SUSTAV

Robot Operating System je fleksibilan okvir (eng. *Framework*) koji pruža različite alate i knjižnice za razvoj robotskog softvera. Nudi nekoliko moćnih značajki koje pomažu razvojnim programerima oko zadataka kao što su prosljeđivanje poruka, ponovna upotreba koda i implementacija najsuvremenijih algoritama za robotske aplikacije. Projekt ROS je 2007. godine pokrenuo Morgan Quigley, pod nazivom Switchyard, kao dio projekta robota Stanford STAIR. Glavni razvoj ROS-a dogodio se u Willow Garage. ROS zajednica raste vrlo brzo, te ima mnogo korisnika i programera u cijelom svijetu. Dosta vrhunskih robotskih tvrtki prelazi na korištenje ROS-a. Industrijski pokret ROS dobio je zamah u posljednjih nekoliko godina, zahvaljujući velikom broju istraživanja provedenih na tom području. ROS Industrial dodatno povećava ROS-ove mogućnosti prilagođavajući ga za industrijsku uporabu. [2]

2.1. Povijest ROS-a

Krajem 2006. godine na Sveučilištu Stanford doktorandi zaposleni u robotičkom laboratoriju Kennetha Salisburyja na Stanfordu Eric Berger i Keenan Wyrobek postavljaju temelje onoga što će naposljetku postati ROS. Radeći na robotima za obavljanje manipulativnih zadataka u ljudskom okruženju, dvojac je primijetio da mnoge njihove kolege sputava raznolika priroda robotike: izvrstan programer softvera možda nema potrebno znanje o hardveru, dok netko tko razvija najsuvremenije putanje planiranja možda neće znati kako napraviti potreban računalni prikaz. U pokušaju da riješe navedeni problem, krenuli su u razvoj osnovnog sustava koji bi bio temelj za ostale kolege iz akademskih krugova kako bi se na njemu objedinili. U svojim prvim koracima prema ovom objedinjujućem sustavu, proizveli su robotsku ruku PR1 kao hardverski prototip. Započeli su rad na softveru, koristeći najbolje segmente drugih ranih *open-source* robotskih softverskih okvira, posebice *switchyard*. Rano financiranje od 50.000 američkih dolara osigurali su Joanna Hoffman i Alain Rossmann, koji su podržali razvoj robota PR1. Dok su tražili sredstva za daljnji razvoj, Eric Berger i Keenan Wyrobek upoznali su Scotta Hassana, osnivača Willow Garage, koji je radio na autonomnom automobilu i solarnom autonomnom brodu. Hassan je podijelio Bergerovu i Wyrobekovu viziju "Linuxa za robotiku". Willow Garage pokrenut je u siječnju 2007. godine, a prva distribucija ROS-a predstavljena je 7. studenog 2007. [3]

2.1.1. Vremenska crta distribucija ROS-a

- **2009.** objavljena prva stabilna distribucija, ROS 0.4 *Mango Tango*, a potom je izrađen robot PR2
- **2010.** objavljena distribucija ROS 1.0 čije su mnoge značajke i danas u uporabi te je izrađen robot ROS C Turtle
- **2011.** objavljene su distribucije ROS *Diamondback* u ožujku te ROS *Electric Emys* u kolovozu
- **2012.** objavljena distribucija Ros *Fuerte* u travnju, distribucija Ros *Groovy Galapagos* u prosincu te Open Source Robotics Foundation preuzima projekt ROS
- **2014.** objavljena distribucija ROS *Indigo Igloo*. Ova je distribucija značajna jer je po prvi put pružena dugoročna podrška, što znači da su se ažuriranja i podrška pružale dulje vrijeme, konkretno pet godina.
- **2015.** objavljena ROS *Jade Turtle* distribucija
- **2016.** objavljena ROS *Kinetic Kame* distribucija. Ona je druga verzija ROS-a s dugoročnom podrškom
- **2017.** objavljena ROS *Lunar Loggerhead* distribucija
- **2018.** objavljena distribucija ROS *Melodic Morenia*
- **2020.** trenutno zadnja objavljena distribucija ROS *Neotic Ninjemys* čija podrška traje do svibnja 2025. [4]

2.2. Karakteristike ROS-a

ROS je kombinacija skupova upravljačkih programa, zbirka osnovnih algoritama, kao i skupa alata za vizualizaciju te ROS ekosustava. Glavna svrha ROS-ovog ekosustava je izmjena podataka koristeći međuspremnik u kojem procesi mogu komunicirati i razmjenjivati podatke jedni s drugima, čak kad rade s različitih strojeva. Softver u ROS-u organiziran je kao paket čija je prednost izvrsna modularnost pojedinih značajki. Može se reći da je ROS projekt vođen većinski od strane zajednice koja sačinjava programere diljem svijeta. Ovaj aktivni razvojni ekosustav ističe ROS od ostalih robotskih okvira. [2]

2.2.1. Prednosti ROS-a

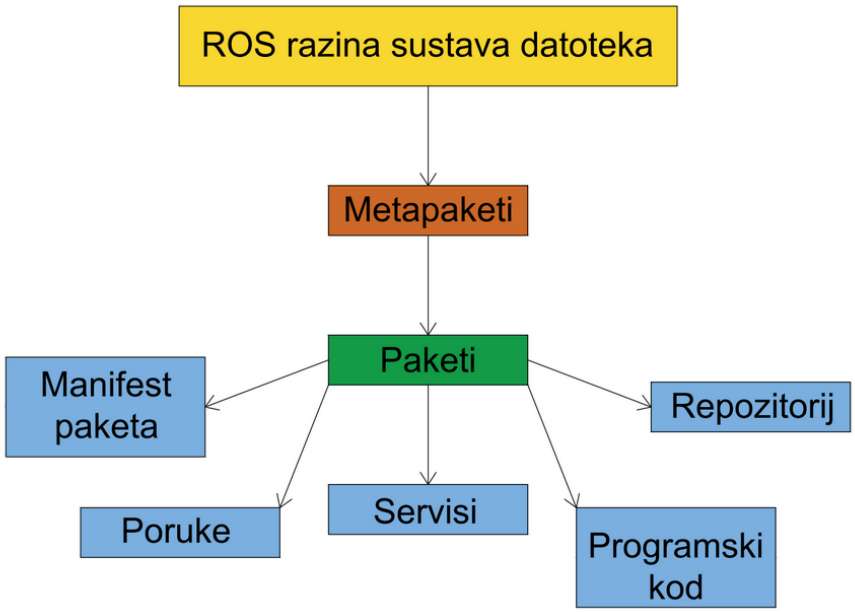
- **Vrhunske mogućnosti**: u ROS su integrirani mnogi alati. Primjerice alati poput *Simultaneous Localization and Mapping* i *Adaptive Monte Carlo Localization* koji se koristite za izvođenje autonomne navigacije u mobilnim robotima, i alat *MoveIt!* korišten za planiranje putanja kretanja robotskih manipulatora.
- **Veliki broj dostupnih alata**: ROS je prepun dodatnih alata otvorenog koda. Neki od njih su su *rqt_gui*, *RViz* i *Gazebo* koje se koristi za otklanjanje pogrešaka te vizualizaciju i izvođenje simulacije.
- **Podrška senzora i aktuatora**: ROS je prepun upravljačkih programa i paketa koji omogućuju korištenje velikog broja raznih senzora i aktuatora. Vrhunski senzori uključuju *Velodyne-LIDAR*, laserske skenere, *Kinect* te aktuatore kao što je *DYNAMIXEL servo*.
- **Operativnost među platformama**: međuspremnik za razmjenu podataka omogućuje komunikaciju između različitih čvorova (eng. *Node*). Ovi se čvorovi mogu programirati u bilo kojem programskom jeziku koji podržava ROS klijentske knjižnice. Moguće je kreirati čvorove u *C* ili *C++* programskom jeziku koji mogu komunicirati s čvorovima pisanim u drugim programskim jezicima kao što su *Python* ili *Java*. Ova vrsta fleksibilnosti nije dostupna u drugim robotskim okvirima.
- **Modularnost**: Jedan od problema koji se može pojaviti u većini samostalnih robotskih sustava je da ako bilo koji dio glavnog koda prestane s radom, cijeli se sustav može srušiti što dovodi do zastoja u radu robota. U ROS-u je situacija drugačija; kreiraju se posebni čvorovi za svaki pojedini proces tako da čak i ako se jedan čvor sruši, sustav i dalje može funkcionirati. Također, ROS pruža robusne metode za nastavak rada čak i kada su neki senzori ili motori izvan funkcije.
- **Aktivna zajednica**: ROS-ovu zajednicu čini veliki broj inženjera i hobista. Ovo je možda i najveća prednost ROS-a. Postoji web portal na kojem je organizirana podrška od strane iskusnijih korisnika na kojem je moguće pronaći rješenja za najbanalnije probleme kao i za kompleksne i veoma specifične probleme. Naravno, ne postoji garancija o pronalasku rješenja za svaki problem, no velika je šansa da će netko pružiti konkretan odgovor koji bi mogao biti rješenje za postavljeni upit o problemu. [5]

2.2.2. Nedostaci

- **Poteškoće u učenju**: ROS može biti teško savladati. Ima strmu krivulju učenja a programeri bi se trebali upoznati s mnogim novim konceptima kako bi u potpunosti iskoristili prednosti ROS-a.
- **Kompleksan i dugotrajan postupak učenja:** Glavni simulator u ROS-u je *Gazebo*. Iako *Gazebo* dobro funkcionira, započeti s njegovim korištenjem nije lak zadatak iz razloga što nema ugrađene značajke za programiranje. Za razliku od *Gazebo*-a, drugi simulatori kao što su *V-REP* i *Webots*, imaju ugrađene funkcionalnosti za prototip i programiranje robota.
- **Poteškoće prilikom modeliranja robota**: Modeliranje robota u ROS-u izvodi se pomoću URDF datoteke, što je zapravo opis robota koji se temelji na XML-u. U *V-REP*-u je moguće izravno kreirati 3D model robota koristeći grafičko sučelje. U programskom paketu *SolidWorks* implementirana je ekstenzija za konverziju 3D modela u URDF format, no taj dodatak nije dostupan u drugim 3D CAD alatima. [5]

2.3. Struktura ROS-a

ROS je više od razvojnog okvira. ROS možemo djelomično shvatiti kao operacijski sustav, budući da nudi ne samo alate i knjižnice nego čak i funkcije slične operacijskom sustavu, kao što su apstrakcija hardvera, upravljanje paketima i lanac alata za razvojne programere. Kao kod pravog operacijskog sustava, ROS datoteke su organizirane na tvrdom disku po određenoj hijerarhiji, kao prikazano na slici 2.1.

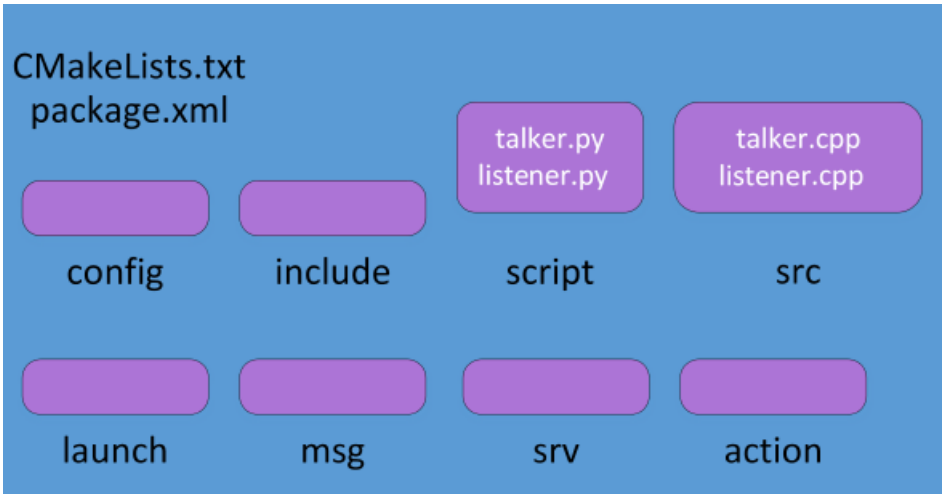


Slika 2.1. Hijerarhijska struktura ROS sustava datoteka

- **Metapaket** (eng. *Metapackage*) se odnosi na jedan ili više povezanih paketa koji se mogu međusobno grupirati. U principu, metapaketi su virtualni paketi koji ne sadrže izvorni kod ili tipične datoteke koje se obično nalaze u paketu.
- **Paketi** (eng. *Package*) su najosnovnija jedinica ROS softvera. Oni sadrže jedan ili više ROS programa (čvorova), knjižnice, konfiguracijske i ostale datoteke koje su organizirane zajedno kao jedna cjelina.
- **Manifest paketa** (eng. *Package Manifest*) je datoteka koja se nalazi unutar paketa a sadrži informacije o paketu, autoru, licenci, ovisnostima i ostale informacije. Ta se datoteka naziva „package.xml“.
- **Poruke** (eng. *Message*) su vrsta informacija koje jedan proces šalje drugom. Moguće je definirati vlastitu prilagođenu poruku unutar „msg“ mape unutar određenog paketa.
- **Servisi** (eng. *Services*) definiraju vrstu interakcije po principu zahtjeva/odgovora između određenih procesa.
- **Repozitoriji** (eng. *Repositories*) predstavlja skup paketa koji djele istu platformu održavanja, primjerice *GitHub*.

2.3.1. Ros paket

Tipična struktura ROS paketa prikazana je na slici 2.2. [2]

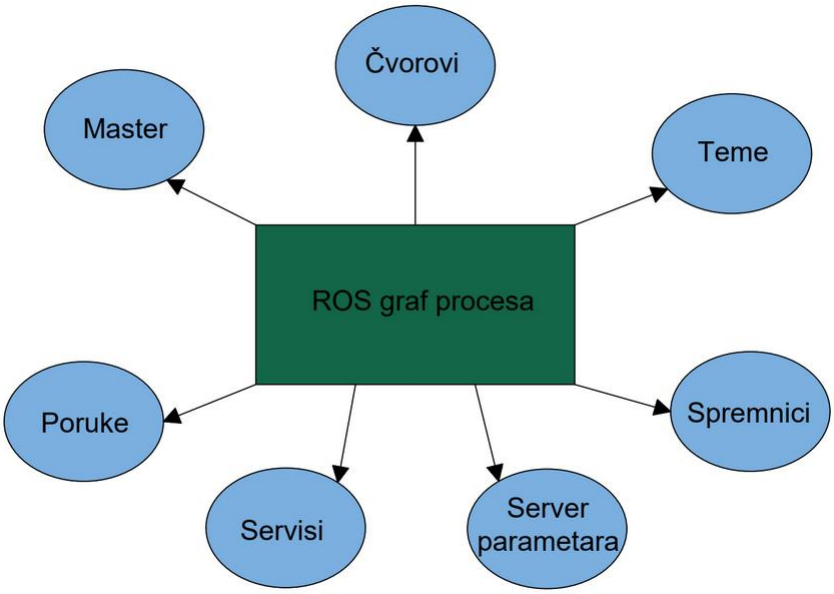


Slika 2.2. Struktura ROS paketa [2]

- *config*: Sve konfiguracijske datoteke koje se koriste u ovom ROS paketu pohranjuju se u ovoj mapi. Ovu mapu kreira korisnik i uobičajena je praksa da se mapa imenuje „config".
- *include/package_name*: Ova mapa se sastoji od zaglavlja i knjižnica koje se koristite unutar paketa.
- *script*: Ova mapa sadrži izvršne skripte. U ovom specifičnom slučaju u njoj se nalaze dvije Python skripte.
- *src*: Ova mapa pohranjuje izvorne kodove. U ovoj su mapi pohranjena dva *C++* izvorna koda.
- *launch*: Ova mapa sadrži datoteke za pokretanje koje se koriste za pokretanje jednog ili više čvora.
- *msg*: Ova mapa sadrži prilagođene poruke.
- *srv*: Ova mapa sadrži definicije usluga.
- *action*: Ova mapa sadrži datoteke akcija.
- *package.xml*: Ovo je datoteka manifesta paketa ovog paketa.
- *CMakeLists.txt*: Ove datoteke sadrže direktive za provođenje paketa.

2.4. Razina povezivanja procesa

Izvršavanje procesa u ROS-u se vrši pomoću mreže procesa. Glavni koncepti prilikom izvršavanja su čvorovi, master, poslužitelj parametara, poruke, teme, usluge, i spremnici (slika 2.3.).



Slika 2.3. Struktura grafa procesa

U nastavku slijede kratka objašnjenja pojedinih članova. [5]

- **Čvorovi**: Čvorovi su procesi koji obrađuju informacije. Svaki ROS čvor je napisan korištenjem ROS klijentskih knjižnica. Koristeći knjižnice klijenta, moguće je implementirati različite ROS funkcionalnosti, kao što su metode komunikacije između čvorova, što je posebno korisno kada različiti čvorovi robota moraju međusobno razmjenjivati informacije. Koristeći ROS komunikacijske metode čvorovi mogu međusobno komunicirati i razmjenjivati podatke. Jedan od ciljeva ROS čvorova je izgradnja većeg broja jednostavnih procesa, umjesto jednog velikog procesa sa svim funkcionalnostima. Budući da su ROS čvorovi strukturno jednostavni veoma je lako dijagnosticirati i otkloniti pogreške.
- **Master:** ROS Master omogućuje registraciju imena i traženje postojećih čvorova. Bez usluge ROS Master čvorovi se ne mogu međusobno povezati, pozvati ili razmjenjivati podatke.

- **Server parametara:** omogućuje pohranu podataka na središnjoj lokaciji. Svi čvorovi mogu pristupiti i mijenjati ove vrijednosti. Server parametara je dio ROS Master-a.
- **Poruke:** Čvorovi međusobno komuniciraju pomoću poruka. Poruke su zapravo struktura podataka koja sadrži upisano polje, koje može sadržavati skup podataka, a to se može poslati na drugi čvor.
- **Teme:** Svaka poruka u ROS-u se prenosi pomoću imenovanih sabirnica koje se nazivaju teme.
- **Usluge:** U nekim robotskim aplikacijama, komunikacija tipa objavi/pretplati nije prikladna. U slučajevima gdje je potrebna interakcijska struktura tipa zahtjev/odgovor koristi se ROS usluga. Njena karakterističnost je sinkronost. Funkcionira na način da klijent šalje upit i komunikacija staje sve dok se ne dobije traženi odgovor. ROS usluge treba koristiti samo za izračune i brze akcije.
- **Spremnici:** ROS pruža sustav pohrane podataka, kao što su podaci senzora, koji se pohranjuju u spremnik.

2.5. Najznačajniji alati ROS sustava

ROS sačinjava veliki broj alata korištenih u svrhu upravljanja robotskim sustavima. Najčešće se javlja potreba za istovremenom kombinacijom većeg broja alata kako bi se na ispravan način upravljalo robotskim sustavima.

2.5.1. RViz

ROS vizualizacija, ili skraćeno *RViz,* je moćan alat za 3D vizualizaciju u ROS-u. Omogućuje korisniku pregled modela robota, informacije senzora robota i ponovno reproducirati zabilježene informacija. Otklanjanje grešaka samo proučavajući podatke u brojčanom obliku je dovoljno velik izazov za 2D prostor dok je gotovo nemoguće u 3D prostoru. Vizualizacijom onoga što robot vidi, shvaćanjem načina na koji robot razmišlja i radi moguće je na jednostavan način otkloniti pogreške u robotskom sustavu. *RViz* prikazuje podatke 3D senzora, kamera, lasera, *Kinect*-a i drugih 3D

uređaja u obliku oblaka sačinjenog od točaka. 2D senzorski podaci s web kamera, RGB kamera ili 2D laserskog daljinomjera mogu se vidjeti u *RViz*-u kao slikovni podaci.

Ako stvarni robot komunicira s radnom stanicom putem *RViz*-a, *RViz* će prikazati trenutnu konfiguraciju stvarnog robota na simuliranom modelu. ROS tema koja sadrži konfiguracijske informacije robotske ruke kao i bilo koja druga ROS tema koja se objavljuje u svrhu pomicanja zglobova robotske ruke može biti prikazana u informacijskom sučelju *RViz*-a. *RViz* pruža mogućnost konfiguriranja Grafičkog korisničkog sučelje (eng. *Graphical user interface*) koji omogućuje korisniku prikaz samo informacija koje su relevantne za određeni zadatak. [6]

2.5.2. MoveIt!

MoveIt! je sofisticirani softverski paket napravljen isključivo za ROS a glavni su mu ciljevi računanje inverzne kinematike, planiranje kretanji tj. putanja robota, 3D percepcija okoline i provjera kolizija. *MoveIt!* je primarni izvor funkcionalnosti za manipulaciju robotima u ROS-u. *MoveIt!* dohvaća konfiguraciju robotske ruke (geometriju i informacije o zglobovima) putem URDF datoteke i ROS poruka, a za izvođenje koristi ROS vizualizacijski alat *RViz*. *MoveIt!* se koristi kod više od 100 robotskih ruku čije se URDF datoteke dostupne na službenoj Internet stranici *MoveIt!*-a. Moveit ima puno naprednih značajki te se koristi kod velikog broja industrijskih robota. [7]

2.5.3. Gazebo

Gazebo je okvir korišten za simulaciju robota. Kao višenamjenski alat za razvojne programere robota u ROS-u, *Gazebo* krajnjem korisniku pruža:

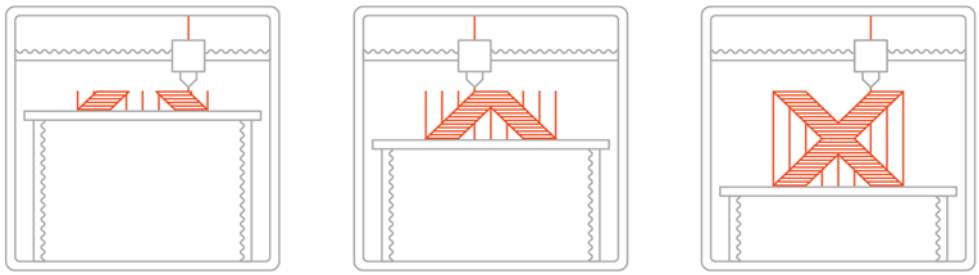
- dizajniranje robotskog modela
- brzo prototipiranje i testiranje algoritma
- testiranje korištenjem realnih senzora
- simulaciju različitih okruženja
- simulaciju podataka različitih senzora
- korištenje nekoliko jakih alata koji simuliraju fizička svojstva okoline

3. ROBOTSKA RUKA *BCN3D*

BCN3D MOVEO je robotska ruka s pet osi rotacije, konstruirana od strane inženjera tvrtke BCN3D u suradnji s Katalonskim odjelom za edukaciju. Elementi robotske ruke su koncipirani na način da se svi elementi mogu realizirati koristeći aditivne tehnologije, dok se koračnim motorima i servo motorom upravljaju zglobovi koristeći *Arduino mega 2560* mikroračunalo. Projekt je realiziran u svrhu edukacije krajnjeg korisnika na području aditivnih tehnologija kao i na području mehatronike i robotike. Svi nužni materijali (CAD datoteke, popis dijelova, priručnik za sklapanje te STL-datoteke) su besplatni i dostupni na službenim stranicama tvrtke BCN3D. [8]

3.1. 3D printeri korišteni za printanje dijelova robotske ruke

U svrhu izrade robotske ruke korištena su tri različita 3D printera. Većina dijelova printana je na Tehničkom fakultetu u Rijeci koristeći 3D printer Zortrax M200 dok su preostali dijelovi printani u tvrtki Izit, u Zagrebu, na 3D printeru Zortrax M300 dok je na Građevinskom fakultetu u Rijeci korišten 3D printer Stratasys Connex500. U tvrtki Izit su 3D printani dijelovi čiji su gabariti veći od radnog prostora 3D printera dostupnog na Tehničkom fakultetu, dok je printer Stratasys Connex500 korišten zbog velike preciznosti prilikom printanja detalja. Printer Zortrax M200 koristi *FDM* tehnologiju (engl. *Fused Deposition Modeling*) što znači da se materijal postepeno topi, sloj po sloj, i raspoređuje po unaprijed određenoj putanji dok se istovremeno radna podloga spušta za definiranu visinu sloja (slika 3.1).



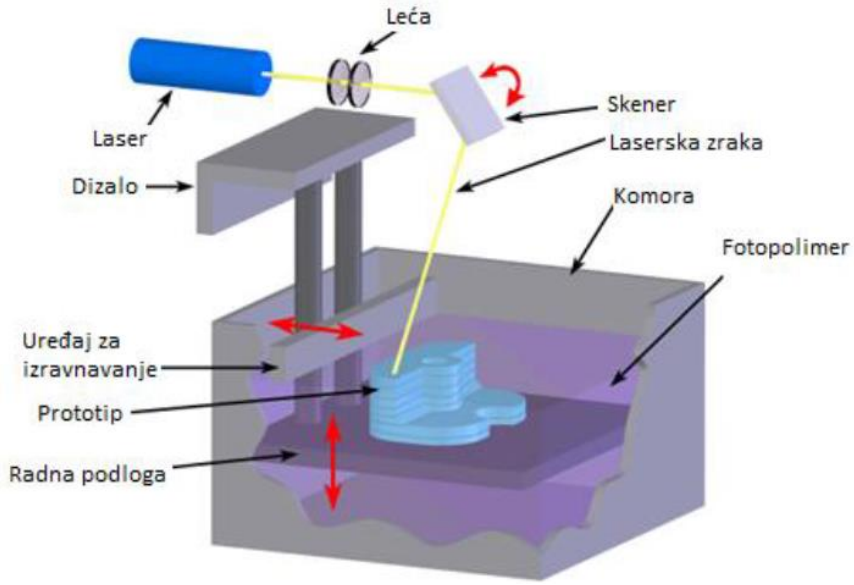
Slika 3.1. Princip FDM tehnologije [9]

Ova profesionalna metoda 3D printanja koristi termo-plastike industrijske kvalitete, a isprintani 3D dijelovi imaju stabilna mehanička, kemijska i termička svojstva. Dimenzije radnog prostora Zortrax M200 3D printera su 200 × 200 × 185 mm, dok preciznost po x i y osima iznosi 1,5 µm. Minimalna visina jednog sloja iznosi 25µm, a maksimalna 400µm. Promjer mlaznice je 0,4 mm. Materijal korišten za printanje je Z-Ultrat. Navedeni materijal je u suštini poboljšana izvedba ABS materijala dizajniran s ciljem veće čvrstoće i bolje površinske kvalitete od standardnog ABS-a. Karakteristike korištenog materijala dane su na slici 3.2.

Mehanička svojstva	Metrički sustav	Imperijalni sustav mjera	Testna metoda
Vlačna čvrstoća	32.60 MPa	4730 psi	ISO 527:1998
Naprezanje pri puknuću	30.70 MPa	4450 psi	ISO 527:1998
Produljenje pri maksimalnom vlačnom opterećenju	3.78%	3.78%	ISO 527:1998
Produljenje pri puknuću	4.87%	4.87%	ISO 527:1998
Opterećenje savijanja	54.00 MPa	7830 psi	ISO 178:2011
Modul fleksije	1.85 GPa	268 ksi	ISO 178:2011
Žilavost	5.26 kJ/m²	2.50 ft-lb/in²	ISO 180:2004
Termička svojstva	**Metrički sustav**	**Imperijalni sustav mjera**	**Testna metoda**
Temperatura radne podloge	106.40° C	224° F	ISO 11357-3:2014
Ostala svojstva	**Metrički sustav**	**Imperijalni sustav mjera**	**Testna metoda**
Protok taline	43.88 g/10 min Količina: 5 kg Temperatura: 260° C	0.0968 lb/10 min Količina: 11 lb Temperatura: 500° F	ISO 1133:2006
Specifična	1.179 g/cm³	9.84 lb/gal	ISO 1183-3:2003
Tvrdoća po Shore-u (D)	73.4	73.4	ISO 868:1998

Slika 3.2. Svojstva Z-Ultrat materijala [10]

3D printer *Stratasys* Connex500 koristi *PolyJet* tehnologiju (*engl. Photopolymer Jetting*). Princip PolyJet tehnologije sličan je kao kod konvencionalnih 2D injekt printera, no umjesto raspršivanja čestica tinte na papir, raspršuju se čestice kondenzirajućeg foto-polimera koje se pod UV svjetlom skrutnjuju (slika 3.3.).



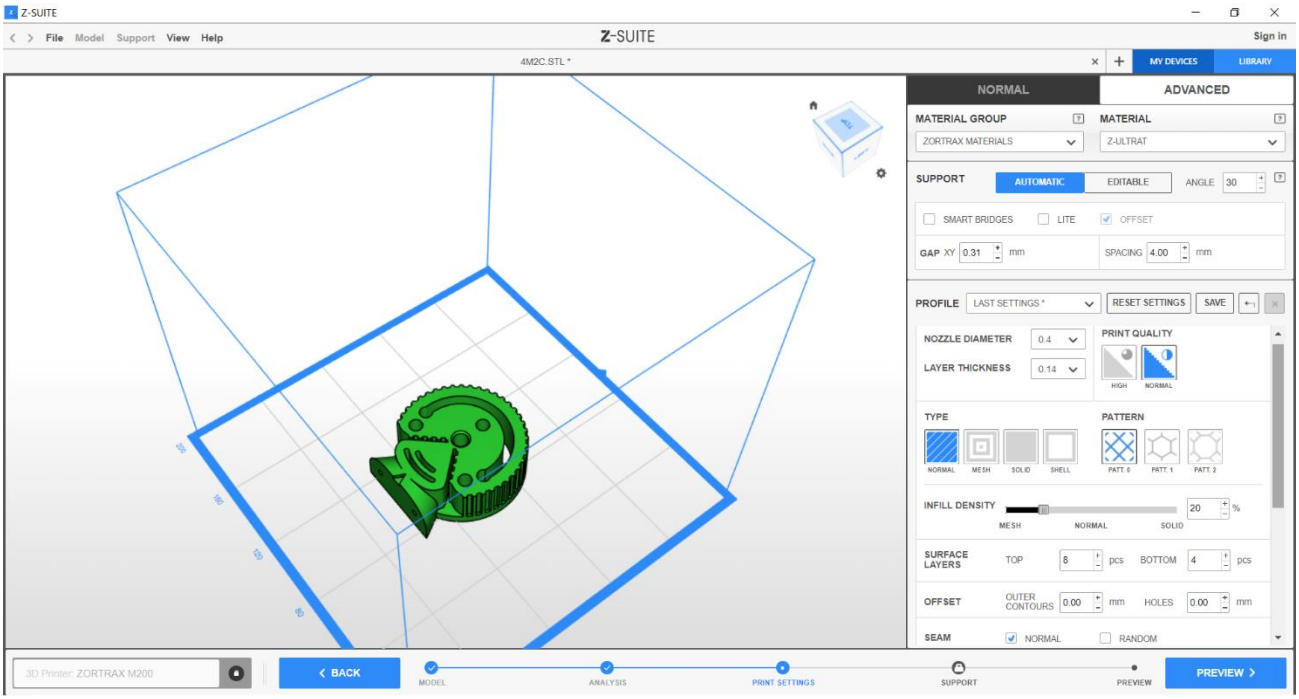
Slika 3.3. Princip PolyJet tehnologije [11]

Gabariti *Stratasys* Connex500 3D printera su 500 x 400 x 200 mm dok je minimalna visina sloja 16µm uz preciznost od ±100µm. Cijena ovog printera iznosi 2.5 milijuna kuna, dok cijena jednog kilograma materijala iznosi 1000 kuna. Jedna od prednosti korištenja *PolyJet* tehnologije je mogućnost proizvodnje dijelova zahtjevne geometrijske građe s vrlo tankom stijenkom. Nedostaci *PolyJet* tehnologije su lošija mehanička svojstva u odnosu na *FDM* tehnologiju, slaba otpornost na vlagu i promjene temperature. Konkretan razlog korištenja navedenog 3D printera je mala preciznost 3D printera korištenog na Tehničkom fakultetu prilikom 3D printanja loga Tehničkog fakulteta u Rijeci na određenim dijelovima. Razlog tome je nedovoljno mali promjer mlaznice u odnosu na zahtijevanu razinu detalja. Usporedba dobivenih površinskih kvaliteta *PolyJet* i *FDM* tehnologije prikazana je na slici 3.4.

Slika 3.4. Usporedba razine detalja PolyJet tehnologije (lijevo) i FDM tehnologije (desno)

3.2. Postupak printanje komponenti na ZORTRAX-U M200

U svrhu printanja dijelova na Zortrax-u M200 3D printeru korišten je službeni *slicer* Z-Suite. *Slicer* je program koji zadanu datoteku secira u slojeve kako bi se generirao potreban G-kod pomoću kojeg se upravlja 3D printerom. Nakon instalacije odabranog slicer-a i definiranja određenih parametra poput vrste 3D printera i veličina radne površine, potrebno je učitati željeni dio i pravilno ga pozicionirati. Zatim se odabiru željeni parametri printanja kao što su vrsta materijala, debljina i kvaliteta sloja, ispunjenost, putanja mlaznice i brzina printanja. Za većinu dijelova korišteni su već unaprijed definirani parametri za materijal Z-Ultrat. (slika 3.5.).



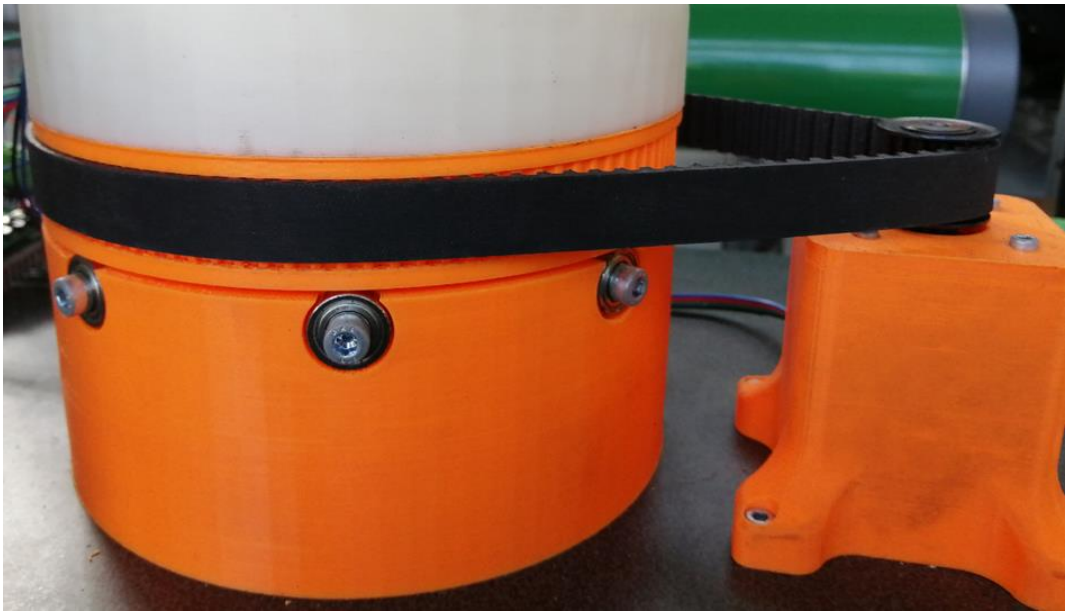
Slika 3.5. Korišteni parametri 3D printera

Nakon završenog postupka 3D printanja isprintani dio je potrebno ukloniti sa 3D printera i po potrebi pažljivo ukloniti višak materijala koji služi kao potpora prilikom printanja određenih dijelova. U suštini je postupak printanja sličan za većinu potrebnih dijelova.

Postupak sastavljanja dijelova podrobno je opisan u završnom radu. [12]

3.3. Vrste prijenosa

Kod konstrukcije robotske ruke potrebno je osigurati vrlo preciznu i finu rotaciju zglobova kako bi se osiguralo što preciznije pozicioniranje robotske ruke. Ova konkretna robotska ruka ima 5 osi rotacije. Zglobovi se rotiraju uz pomoć koračnih motora. Na vratila koračnih motora je ugrađena zupčasta remenica koja putem zupčastog remena prenosi okretni moment s koračnih motora na određeni zglob robotske ruke (slika 3.6.).



Slika 3.6. Prijenos okretnog momenta zupčastim remenom

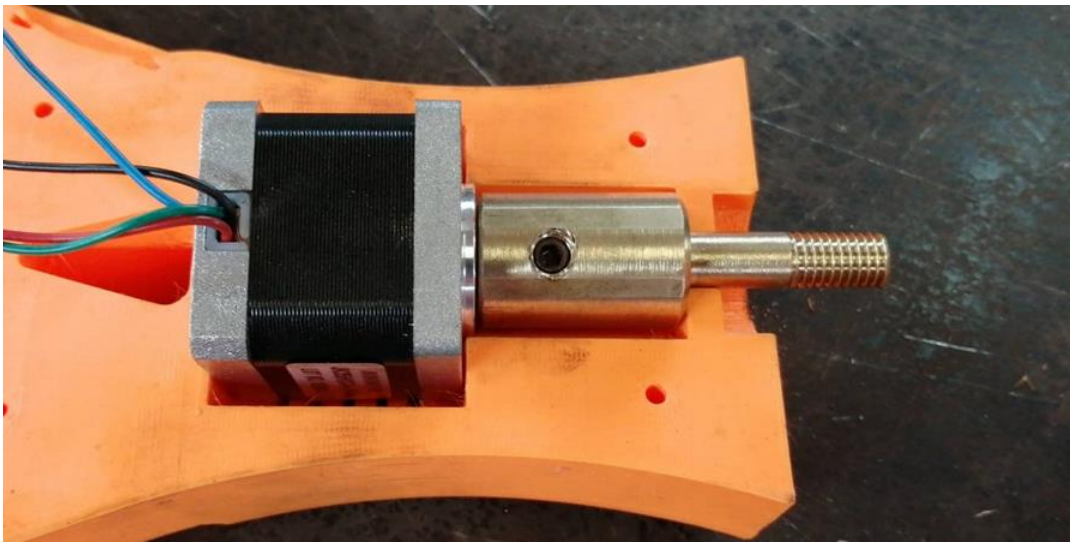
Kod ove robotske ruke korišteno je nekoliko različitih koračnih motora:

- 1 x Nema 17, s planetarnim prijenosom omjera 5:1
- 1 x Nema 23, s planetarnim prijenosom omjera 47:1
- 2 x Nema 17
- 1 x Nema 14

Vratilo koračnog motora se za jedan korak (eng. „Step“) okrene za 1.8. Koračnom motoru s planetarnim prijenosom prijenosnog omjera 47:1 se za jedan korak izlazno vratilo okrene za 0,0382° (slika 3.7.).

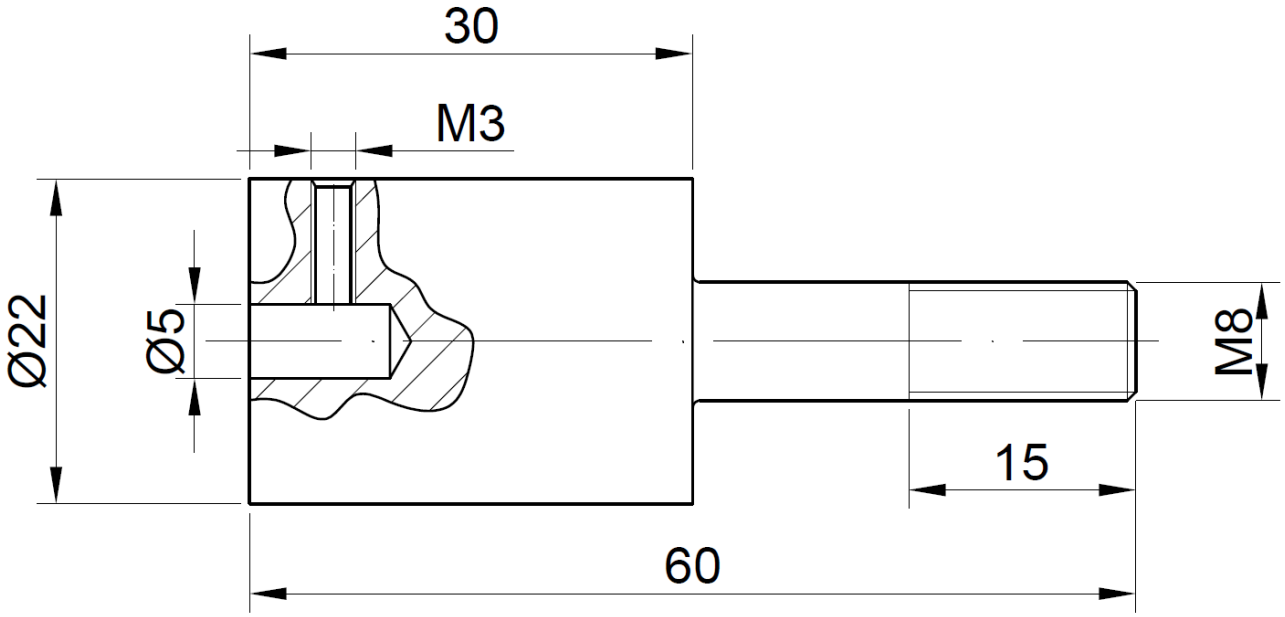
Slika 3.7. Koračni motor s planetarnim prijenosom

Također je korišten prijenos momenta pomoću krute spojke (slika 3.8.). Takvim se spojem osigurava kontinuirana veza između vratila motora i gonjenog tijela. Ova vrsta prijenosa okretnog momenta je bolja u odnosu na remeni prijenos jer kod remenog prijenosa može doći do klizanja između remena i remenice što kod prijenosa s krutom spojkom nije slučaj.



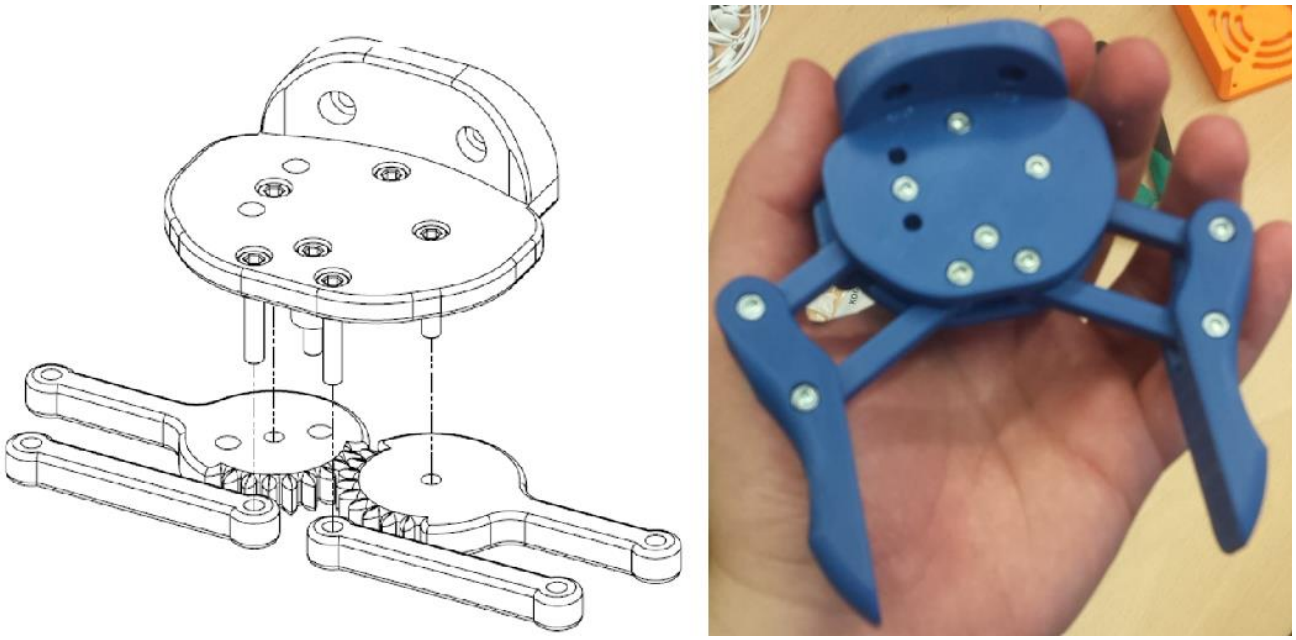
Slika 3.8. Prijenos okretnog momenta ostvaren krutom spojkom

Krutu spojku je izradio tokar prema shemi prikazanoj na slici 3.9.



Slika 3.9. Shema krute spojke

Korišten je i prijenos pomoću zupčanika u svrhu pokretanja prihvatnice. Dva polu-ozubljena zupčanika, s paralelnim osima, služe za stezanje i opuštanje prihvatnice (slika 3.10.). Njeno kretanje osigurava servo motor MG996R.



Slika 3.10. Prikaz sprega zupčanika prihvatnice (lijevo), sklop prihvatnice (desno)

3.4. Sustav upravljanja robotske ruke

Kako bi se omogućila komunikacija između ROS-a i robotske ruke potrebno je koristiti neku vrstu mikrokontrolera ili mikroračunala. U tu svrhu odabran je mikrokontroler *Arduino Mega 2560* iz razloga što se na njega može ukomponirati štit (eng. Shield) RAMPS 1.4. Iako postoje izvedbe koje koriste samo *Arduino* mikrokontroler savjetuje se uporaba štita. Glavna prednost korištenja štita je njegov neovisan sustav napajanja što znači da *Arduino* samo šalje signale štitu a štit, koristeći svoj sustav napajanja, pokreće motore te se tako uklanja rizik od oštećenja mikrokontrolera. Kako bi se poslani signal mogao pravilno pretvoriti u potrebnu količinu struje upotrebljavaju se *Driver TB6560* koji omogućuju veliki raspon podesivih parametra.

3.4.1. Arduino

Arduino je ime mikrokontrolera otvorene računalne i softverske platforme koji omogućava programerima i konstruktorima stvaranje sustava upravljanja između računala i fizičkog svijeta. *Arduino*-vo integrirano razvojno okruženje „IDE“ je aplikacija koju je moguće koristiti na više platformi (Windows, macOS, Linux,…) a koristi se u svrhu pisanja i prijenosa programa na kompatibilne *Arduino* ploče.

Arduino ploča sadrži mikrokontroler koji može procesuirati svaki ROS čvor pojedinačno. Ova sekvencijalna priroda olakšava korištenje i razumijevanje samog procesa i komunikaciju s vanjskim uređajima, kao što su motori, senzori i periferni uređaji. *Arduino* ima skup digitalnih i analognih ulazno/izlaznih *pin*-ova koji podržavaju širok izbor vanjskih senzora i aktuatora. Ovaj se mikrokontroler može koristi za projektiranje i izradu robota koji se pomoću povratnih informacija dobivenih od strane senzora može samostalno kretati u svom okruženju a može se uporabiti i u svrhu poboljšanja postojećeg robota s proširenim mogućnostima. [6]

3.4.2. Štit *RAMPS 1.4*

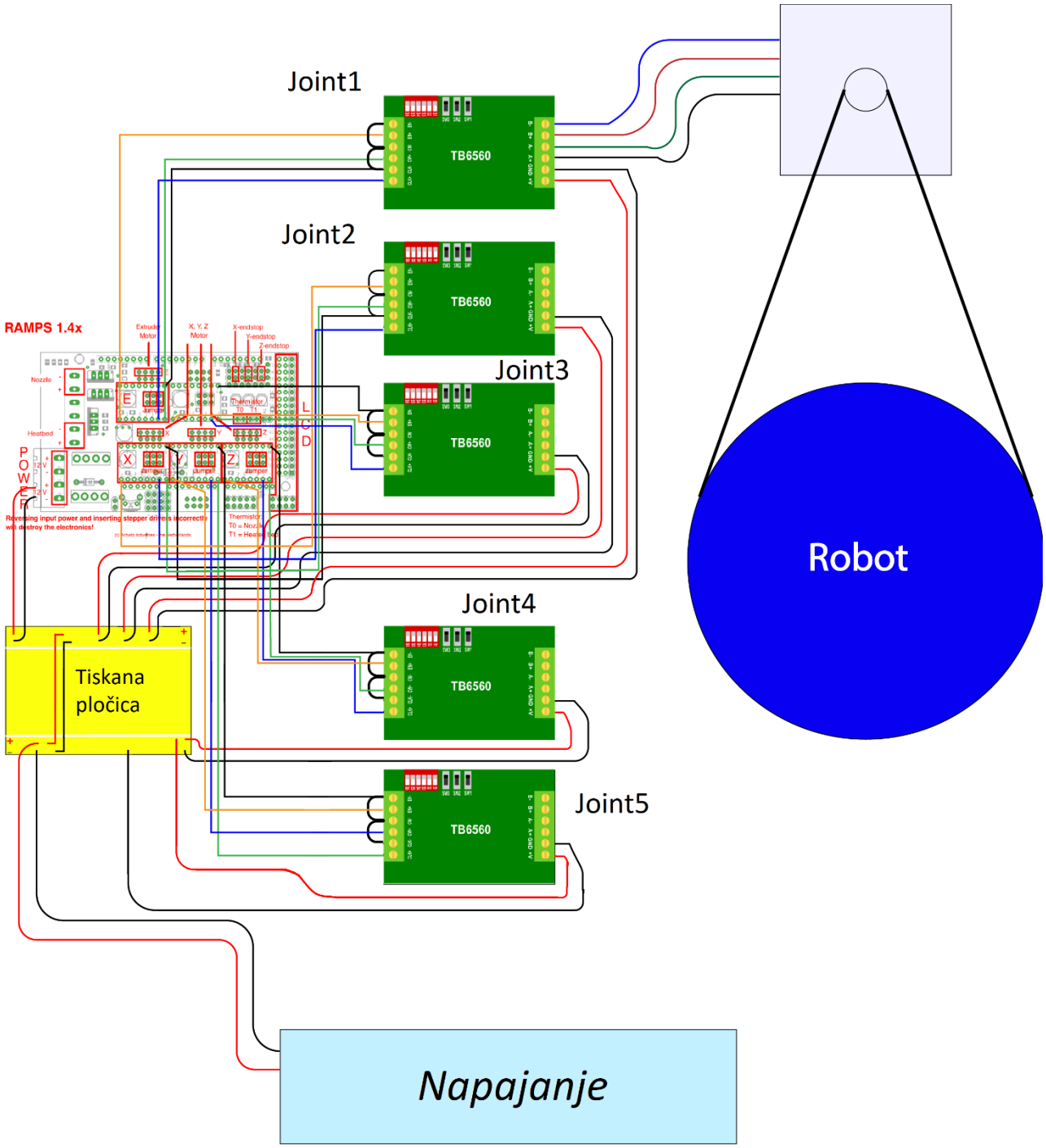
Repap Arduino mega pololu shield ili skraćeno RAMPS, je projekt započet u Engleskoj 2005. godine u svrhu razvoja jeftinog 3D pisača, a od ožujka 2014. godine RAMPS 1.4 je vjerojatno najraširenija elektronika za *RepRap* 3D pisače. Sastoji se od RAMPS 1.4 štita, Arduino Mega 2560 mikrokontrolera a može pokretati maksimalno pet koračnih motora s preciznošću od 1/16 koraka. Ovaj štit također podržava termistor printajuće glave, termistor *hotenda*, grijanu površinu, grijač glave ventilator printera, šest *limit switch*-eva, lcd ekran, bluetooth modul, opciju spajanja čitača SD kartice a neiskorištene *Arduino*-ve *pin*-ove je također moguće upotrijebiti u svrhu dodavanja ventilatora ili led rasvjete. Štit također ima vlastiti sustav napajanja za 12V ili 24V. [13]

3.4.3. Driver TB2650

Driver TB6560 je izvrstan mikropokretač koji koristi TOSHIBA TB6560 čip, zasnovan na čisto-sinusnoj tehnologiji regulacije struje. Zahvaljujući gore navedenoj tehnologiji i tehnologiji samo podešavanja parametra po predloženim specifikacijama proizvođača motorima, pokretani motori postižu rad s manjom bukom, nižim zagrijavanjem, uglađenijim kretanjem i imaju bolje performanse pri većoj brzini od većine pogona na tržištima. Pogodan je za pogon dvofaznih i četverofaznih hibridnih koračnih motora. *Driver* podržava namještanje različitih parametara kao što su raspon maksimalne izlazne struje koja se kreće od 0,3A do 3A, mogućnost regulacije koraka koračnih motora do 1/16 koraka, izlaznu struju od 50% i 100% u zaustavljenom stanju motora što je korisna opcija jer sprječava proklizavanje prilikom veće opterećenosti motora ili manjeg zagrijavanja kod manje opterećenosti motora. Također je još moguće namjestiti *Decay* opcije što je korisno kod malih pomaka koračnih motora jer ova opcija omogućuje veću finoću zakretanja osovine.

3.4.4. Shema spajanja aktuatorskih komponenta robotske ruke

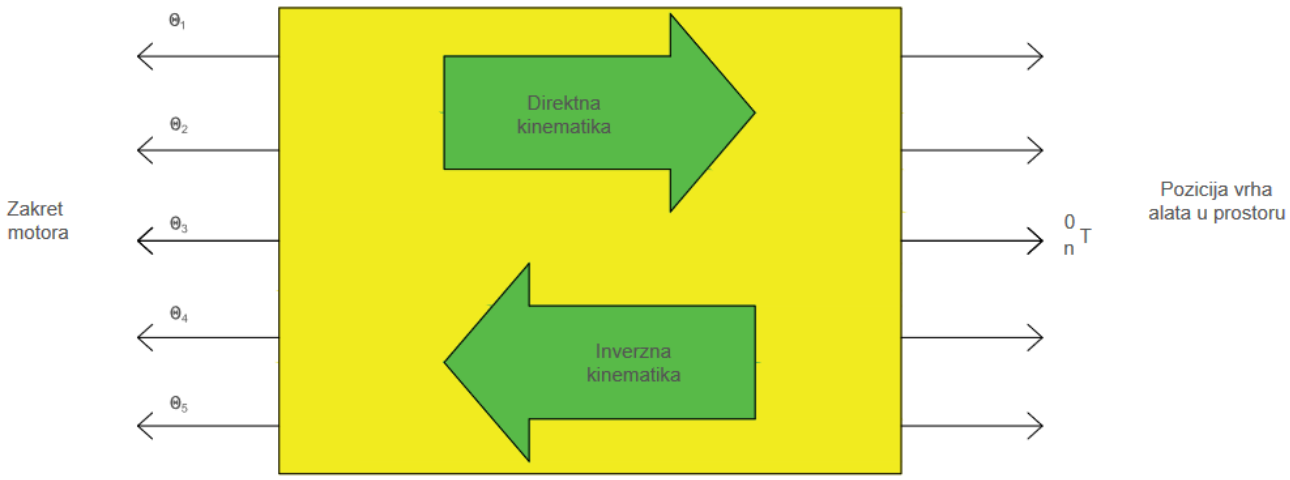
Nakon završenog postupka sastavljanja robotske ruke potrebno je pravilno povezati navedene elektroničke komponente. U tu su svrhu korištene spojne žice i tiskana ploča. Shema spajanja prikazana je na slici 3.11.



Slika 3.11. Shema spajanja električnih komponenata

4. KINEMATIKA ROBOTSKE RUKE

Kinematika predstavlja granu fizike koja se bavi proučavanjem gibanja tijela zanemarujući sile i momente koji utječu na njegovo gibanje. Kinematika robota proučava gibanje robota. Možemo ju podijeliti na inverznu i direktnu kinematiku. Inverzna kinematika je kompliciranija jer uključuje rješavanje kompleksnih jednadžbi i analitičkih problema, dok je direktna kinematika nešto jednostavnija samim time što ne zahtjeva izvođenje kompleksnih jednadžbi. Na slici 4.1. je prikazan odnos između direktne i inverzne kinematike.

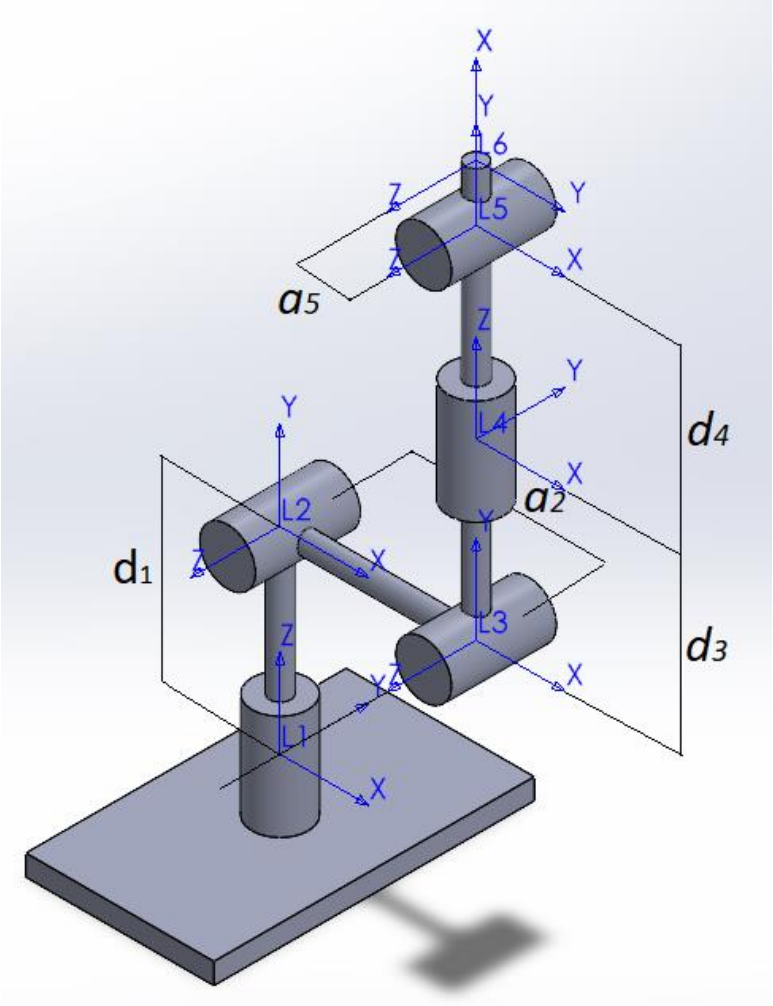


Slika 4.1. Veza između direktne i inverzne kinematike

4.1. Direktna kinematika (Denavit-Hartenberg-ova metoda)

Denavit-Hartenberg-ovom metodom se određuje položaj alata (prihvatnice) sustavnim pridruživanjem desno orijentiranih koordinatnih sustava svakom članku kinematičkog lanca. Denavit-Hartenberg-ova metoda koristi četiri parametra za opisivanje kinematike robotske ruke, a to su: kut zakreta zgloba oko Z osi (θ_k), odmak članka po Z osi, (d_k), kut rotacije oko X osi (α_k) i odmak članka po X osi (a_k). Pozicije na kojim se nalaze zglobovi robotske ruke određuju orijentaciju i poziciju vrha alata. Kako bismo riješili direktni kinematički problem nužno je napraviti simboličku shemu robotske ruke.

Sljedeći korak kod direktnog kinematičkog problema jest odrediti matricu direktnog kinematičkog problema robotske ruke. Za početak je potrebno zglobovima pridružiti brojeve od 1 do *n* pri čemu je zglob 1 vezan za bazu robota dok su krajnji zglobovi *n*-1 i *n* vezani uz orijentaciju prihvatnice. Bazi robota se pridružuje desno orijentirani koordinatni sustav L_0 na način da se os z_0 podudara s osi zgloba 1. Na ovaj je način potrebno pridružiti koordinatne sustave svim zglobovima (slika 4.2.).



Slika 4.2. Simbolička shema s koordinatnim sustavima

Vrijednosti a_k i d_k su dimenzije članaka robotske ruke. Parametar θ_k predstavlja kut rotacije oko osi z_{k-1} mjeren koji se mjeri od osi x_{k-1} prema osi x_k. Potrebno je još odrediti parametre α_k kao kut rotacije od osi z_{k-1} prema osi z_k oko osi x_k. Dobivene vrijednosti su prikazane u Tablici 1.

Tablica 1. DH parametri robotske ruke BCN3D MOVEO

Naziv zgloba	Kut zakreta oko Z osi, θ_k [°]	Odmak članka po Z osi, d_k [mm]	Odmak članka po X osi, a_k [mm]	Kut rotacije oko X osi, α_k [°]	Raspon kuta zakretanja oko z osi [°]
Joint1	θ_1	160	0	90	-120 do 120
Joint2	θ_2	0	210	0	-100 do 100
Joint3	θ_3	120	0	90	-100 do 100
Joint4	θ_4	90	0	90	-90 do 90
Joint5	θ_5	0	160	0	-90 do 90

Nakon određivanja potrebnih parametara moguće je, pomoću matrica homogenih transformacija, odrediti položaj vrha alata.

$$ {}^{k}_{k-1}T = \begin{bmatrix} cos\theta_k & -cos\alpha_k sin\theta_k & sin\alpha_k sin\theta_k & \alpha_k cos\theta_k \\ sin\theta_k & cos\alpha_k cos\theta_k & -sin\alpha_k cos\theta_k & \alpha_k sin\theta_k \\ 0 & sin\alpha_k & cos\alpha_k & d_k \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.1) $$

Matematički problem se može podijeliti u dva dijela kako bi se pojednostavio postupak računanja. Prvi problem se odnosi na pozicioniranje vrha prihvatnice dok se drugi problem odnosi na orijentaciju vrha prihvatnice u prostoru:

$$ {}^{5}_{0}T = {}^{3}_{0}T * {}^{5}_{3}T \quad (4.2) $$

Transformacijska matrica ${}^{3}_{0}T$ predstavlja prva tri zgloba pomoću kojih se određuje pozicija vrha alata u prostoru

$$ {}^{3}_{0}T = {}^{1}_{0}T * {}^{2}_{1}T * {}^{3}_{2}T \quad (4.3) $$

Uvrštavanjem podataka iz tablice 1 dobiva se oblik matrice ${}^{3}_{0}T$:

$$ {}^{3}_{0}T = \begin{bmatrix} c_1 & 0 & s_1 & 0 \\ s_1 & 0 & -c_1 & 0 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} c_2 & -s_2 & 0 & a_2c_2 \\ s_2 & c_2 & 0 & a_2s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} c_3 & 0 & s_3 & 0 \\ s_3 & 0 & -c_3 & 0 \\ 0 & 1 & 0 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.4) $$

Gdje je: $c_1 = \cos(\theta_1)$, $s_1 = \sin(\theta_1)$, $c_2 = \cos(\theta_2)$, $s_2 = \sin(\theta_2)$, $c_3 = \cos(\theta_3)$, $s_4 = \sin(\theta_4)$.

Transformacijska matrica ${}^{5}_{3}T$ predstavlja četvrti i peti zglob pomoću kojih se određuje rotacija vrha alata.

$$ {}^{5}_{3}T = {}^{4}_{3}T * {}^{5}_{4}T $$

$$ {}^{5}_{3}T = \begin{bmatrix} c_4 & 0 & s_4 & 0 \\ s_4 & 0 & -c_4 & 0 \\ 0 & 1 & 0 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} c_5 & -s_5 & 0 & a_5c_5 \\ s_5 & c_5 & 0 & a_5s_5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.5) $$

Gdje je: $c_4 = \cos(\theta_4)$, $s_4 = \sin(\theta_4)$, $c_5 = \cos(\theta_5)$, $s_5 = \sin(\theta_5)$.

Nakon množenja navedenih matrica dobije se transformacijska matrica robotske ruke

$$ {}^{5}_{0}T = {}^{1}_{0}T * {}^{2}_{1}T * {}^{3}_{2}T * {}^{4}_{3}T * {}^{5}_{4}T = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.6) $$

Prva tri stupca definiraju orijentaciju vrha alata dok zadnji stupac definira položaj vrha alata u prostoru. Izjednačavanjem četvrtog stupca transformacijske matrice (4.2) i četvrtog stupca transformacijske matrice (4.6) dobiju se X, Y i Z koordinate položaja vrha alata u radnom prostoru.

$$ p_x = a_5c_5\big((s_1s_4 + c_4(c_1c_2c_3 - s_3c_1s_2)\big) + d_4(c_1c_3s_2 + s_3c_1c_2) + a_5s_5(c_1c_3s_2 + s_3c_1c_2) + c_1c_2a_2 + s_1d_3 $$

$$ p_y = a_5c_5\big(-c_1s_4 + c_4(s_2c_2c_3 - s_3s_1s_2)\big) - c_1d_3 + d_4(s_1c_3s_2 + s_3s_1c_2) + a_5s_5(s_1c_3s_2 + s_3s_1c_2) + s_1c_2a_2 $$

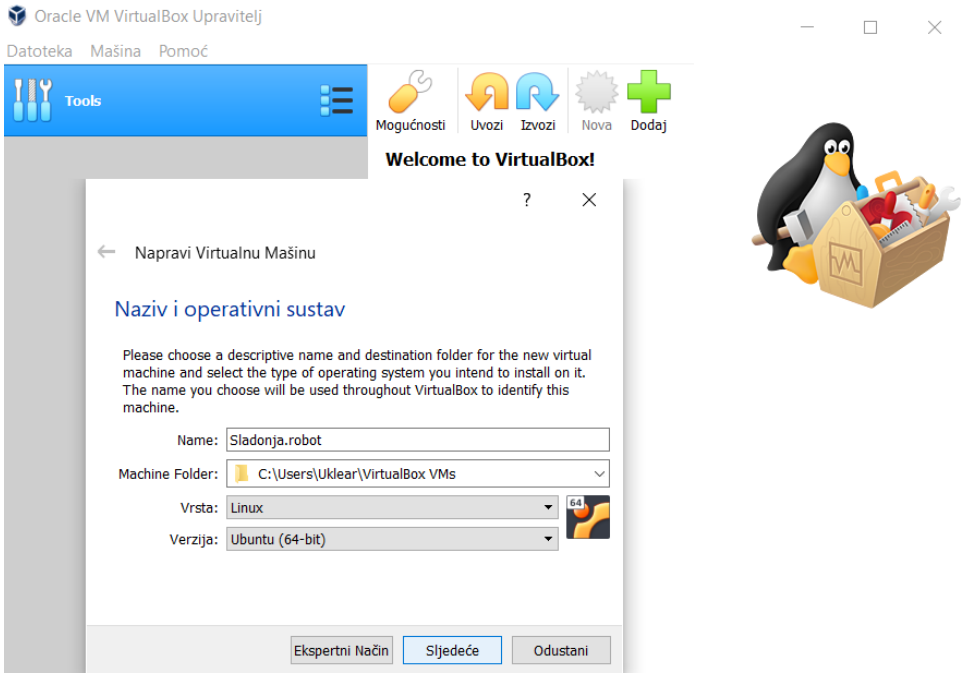
$$ p_z = d_1 + c_4c_5a_5(c_3s_2 + s_3c_2) + d_4a_5s_5(-c_2c_3 + s_3s_2) + a_2s_2 $$

5. PRAKTIČNI DIO

U narednom će poglavlju biti opisan postupak instalacije Linux operativnog sustava, instalacija ROS-a kao i ostalih potrebnih programa korištenih za pokretanje i upravljanje simulirane i stvarne robotske ruke. Iako se ROS može instalirati i na Windows operacijskom sustavu preporučljivo je korištenje Ubuntu operacijskog sustava temeljenog na Linux-u zbog veće podrške zajednice i bolje optimiziranosti ROS-a a u odnosu na Windows-e. Ubuntu operativni sustav je moguće instalirati direktno na računalo ili unutar Windows operativnog sustava koristeći jedan od programskih paketa kao što su Oracle *VM VirtualBox* ili *VMware*. Pomoću navedenih programa moguće je simulirati virtualni prostor na disku u kojemu se instalira odabrani operacijski sustav.

5.1. Instalacija Ubuntu 18.04 na virtualni disk

Prvi je korak, naravno, preuzimanje i instalacija samog programa na računalo. Navedeni je program moguće preuzeti putem službene stranice. Nakon uspješne instalacije može početi postupak instalacije Ubuntu 18.04 operativnog sustava. Klikom na „Dodaj“ otvara se prozor u kojega je potrebno upisati ime virtualnog prostora i odabrati vrstu operativnog sustava (slika 5.1.).

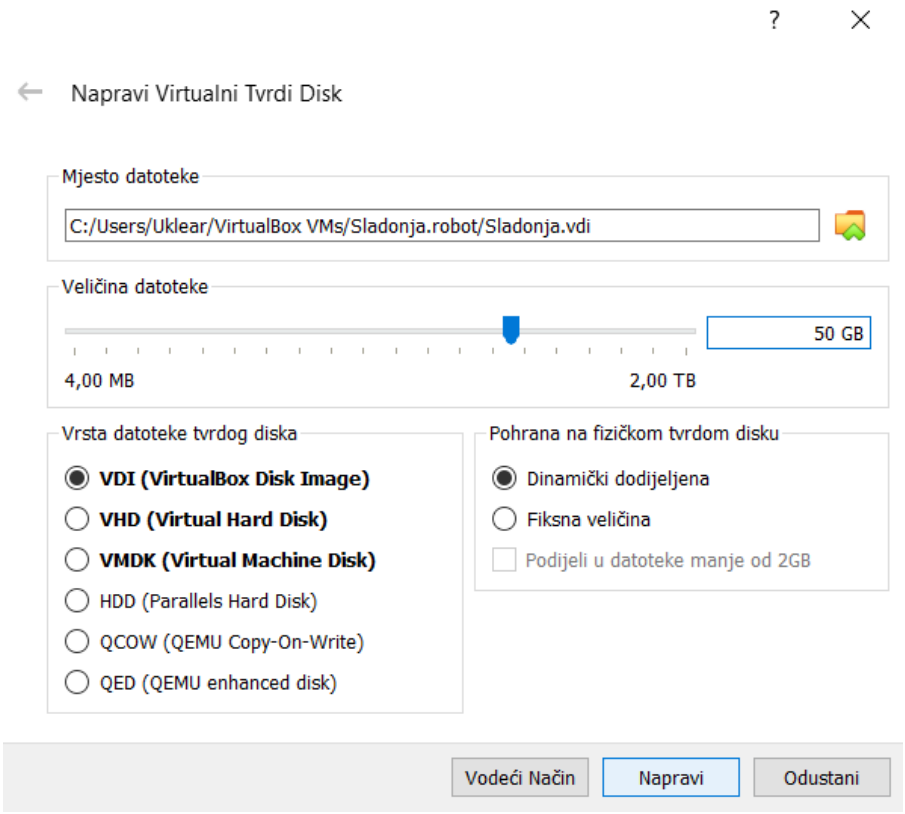


Slika 5.1. Kreiranje novog virtualnog prostora

U narednom se koraku dodjeljuje količina radne memorije virtualnom prostoru. Minimalna preporučena količina je 4GB radne memorije kako bi operacijski sustav i ostali programi normalno funkcionirali. Zatim je potrebno odabrati vrstu virtualnog prostora. Dane su tri mogućnosti:

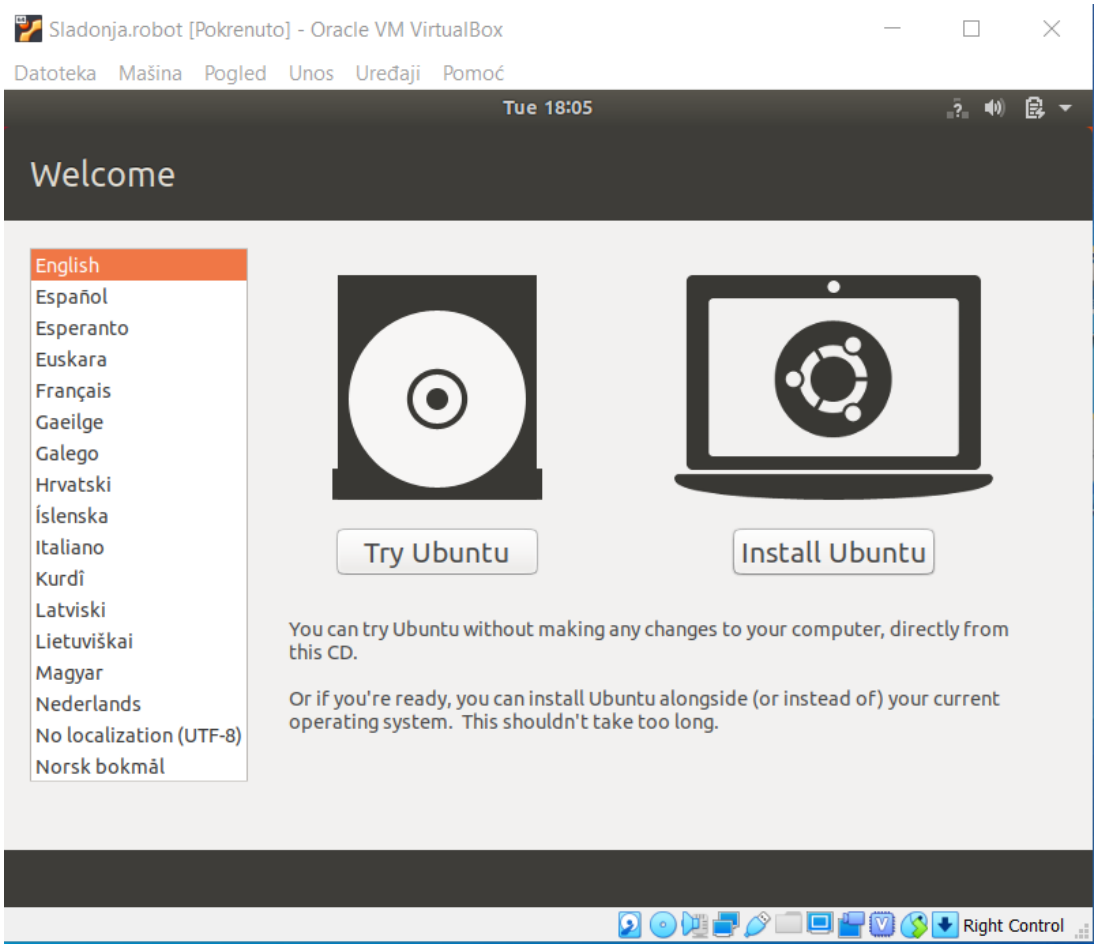
1) Nemoj dodavati virtualni tvrdi disk
2) Napravi virtualni tvrdi disk sada
3) Koristi postojeću datoteku virtualnog tvrdog diska

Prva opcija dinamički kontrolira veličinu virtualnog diska, tj. koristi onoliko prostora koliko je za navedenu instalaciju potrebno. Druga opcija omogućuje dodjeljivanje točno određene veličine virtualnom disku, dok treća opcija omogućuje odabir već napravljenog virtualnog prostora. U ovom je slučaju odabrana druga opcija. Nakon odabira vrste virtualnog diska potrebno je dodijeliti željenu količinu prostora za pohranu te odabrati vrstu pohrane na fizičkom tvrdom disku kao i vrstu datoteke tvrdog diska. Za vrstu pohrane odabrana je opcija *Dinamički dodijeljena*, a za vrstu datoteke tvrdog diska odabrana je opcija *VDI (VirtualBox Disk Image)* (slika 5.2.). Klikom na „Napravi“ završava postupak kreiranja virtualnog prostora.



Slika 5.2. Određivanje parametra virtualnog prostora

Sada je potrebno pokrenuti napravljeni virtualni prostor klikom na „Pokreni“ i odabrati željenu *.iso* datoteku operativnog sustava. Nakon pokretanja instalacijske datoteke operativnog sustava potrebno je izabrati između opcije instalacije operativnog sustava i opcije isprobavanja istog. Odabire se opcija instalacije iz razloga jer je opcija isprobavanja ograničena i primarno služi za testiranje operativnog sustava (slika 5.3.).

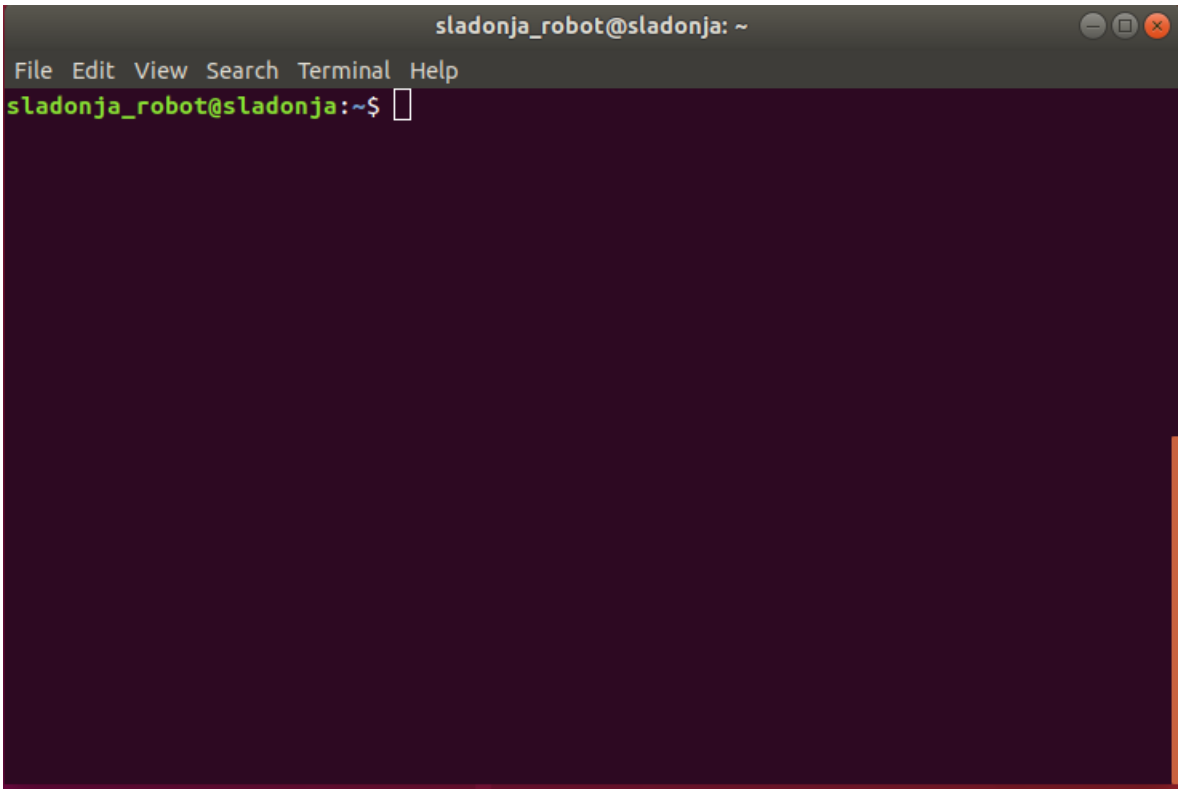


Slika 5.3. Instalacija operativnog sustava

U idućih se nekoliko koraka konfiguriraju opće postavke operacijskog sustava kao što su jezik, konfiguracija tipka tipkovnice, vremenska zona, ime korisničkog računa te lozinka korisničkog računa. Za kraj je potrebno postaviti opcije ažuriranja operativnog sustava kao i dozvole za preuzimanje potrebnih drajvera sa neslužbenih izvora. Nakon završetka konfiguracije navedenih postavka završava proces instalacije operacijskog sustava na virtualni prostor.

5.1.1. Osnovne naredbe u komandnom prozoru

Iako svaka iduća verzija ROS-a ima sve bolje razvijeno grafičko sučelje i dalje postoji potreba za korištenjem komandnog prozora (eng. *Terminal*) prilikom rada u ROS-u (slika 5.4.). Komandni prozor je glavni medij komunikacije između korisnika i Ros-a. U komandni prozor se upisuju naredbe koje je potrebno izvesti a također služi i kao sredstvo za dijagnozu i otklanjanje pogrešaka.



Slika 5.4. Komandni prozor

U nastavku slijede neke od glavnih neophodnih naredbi za snalaženje u komandnom prozoru:

- **$ *sudo*** – izvršava naredbu s administratorskim ovlastima
- **$ *ls*** (list) – ispisuje datoteke i mape u trenutnoj mapi
- **$ *cd ~/* -** mijenjanje direktorija
- **$ *pwd*** – daje povratnu informaciju o konkretnom direktoriju u kojem se nalazimo
- **$ *mkdir*** – stvara novu mapu u direktoriju
- **$ *rm*** – briše određenu mapu ili datoteku
- **$ *rmdir*** – briše mapu samo ako je prazna

- **$ *mv*** – mijenja lokaciju određene mape
- **$ *cp*** – kopira podatke s jedne lokacije na drugu
- **$ *lspci*** – daje povratnu informaciju o hardverskim komponentama računala
- **$ *lsusb*** – daje povratnu informaciju svih USB uređaja
- **$ *ps*** – prikazuje sve procese koji se trenutno izvode
- **$ *clear*** – briše sav sadržaj s komandne ploče
- **$ *kill*** – zaustavlja navedeni proces
- **$ *apt-get*** – instalira pakete koje zatražimo
- **$ *sudo reboot*** – ponovno pokretanje računala direktno iz upravljačkog prozora
- **$ *sudo poweroff*** – gasi računalo direktno iz upravljačkog prozora

U komandnom prozoru je moguće upravljati i kombinacijom tipki sa tipkovnice:

- Ctrl + Shift + C – kopira označeni teksta iz komandnog prozora
- Ctrl + Shift + V – zalijepi prethodno kopirani tekst u komandni prozor
- Ctrl + C – gasi trenutni proces koji se odvija putem komandnog prozora

Iduće se naredbe koriste za upravljanje ROS-a:

- **$ *roscore*** – pokreće zbirku čvorova i programa koji su preduvjeti ROS sustava te omogućuje komunikaciju između čvorova
- **$ *rosrun*** – pokreće izvršne programe i stvara čvorove
- **$ *rosnode*** – prikazuje informacije o čvorovima i navodi aktivne čvorove
- **$ *rosnode kill*** – gasi navedeni proces
- **$ *rostopic*** – prikazuje informacije o temama
- **$ *rostopic find*** – nalazi teme
- **$ *rosmsg*** – prikazuje informacije o vrstama poruka
- **$ *roslaunch*** – naredba koja pokreće alat za upravljanje ROS čvorovima
- **$ *rospack*** – prikazuje povratne informacije o određenom paketu
- **$ *roscd*** – omogućuje mijenjanje direktorija u ROS – u
- **$ *roswtf*** – prikazuje listu mogućih greška

- **$ *source catkin_ws/devel/setup.bash*** – zadaje putanju radnog prostora kako bi ROS mogao izvršavati radnje za definirani paket
- **$ *echo 'source catkin_ws/devel/setup.bash' >> ~/.bashrc*** – sprema navedenu putanju u .bashrc datoteku što omogućuje upravljanje zadanim paketom kroz ostale novootvorene komandne prozore

5.2. Instalacija i konfiguracija ROS sučelja

Nakon uspješne instalacije operativnog sustava i savladavanja osnovnih setova naredbi, može se započeti sa postupkom instalacije ROS-a. Sam postupak instalacije je vrlo jednostavan i temeljito je opisan na službenoj stranici ROS-a. [14]

Instalacija započinje otvaranjem komandnog prozora u kojeg se unose sljedeće naredbe:

$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list' – ovom se naredbom dozvoljava preuzimanje potrebnog sadržaja sa službene stranice na kojoj se nalaze ROS paketi

$ sudo apt install curl – instalacija alata *cURL* koji služi za preuzimanje sadržaja koristeći potrebne mrežne protokole

$ curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add - - dodavanje potrebne dozvole *cURL* alatu

$ sudo apt update – ažuriranje svih potrebnih programskih paketa

$ sudo apt install ros-melodic-desktop-full **–** ROS *Melodic*-a

Zadnja naredba se koristi za kompletnu instalaciju ROS-a na računalo. Putem ove naredbe se na računalo instaliraju ROS, *rqt, RViz,* kao i sve potrebne knjižnice, te 2D i 3D simulatori. Po potrebi je moguće instalirati samo određene značajke ROS sustava uz modifikaciju prethodne naredbe, no preporučljivo je napraviti kompletnu instalaciju. Nakon instalacije ROS-a potrebno je postaviti i spremiti zadanu putanju u *.bashrc* datoteku te instalirati još nekoliko programa koji služe kao posrednici ROS sustava koristeći naredbe:

$ echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc – sprema navedenu putanju u *.bashrc* datoteku

$ source /opt/ros/melodic/setup.bash – zadavanje putanje radnog prostora

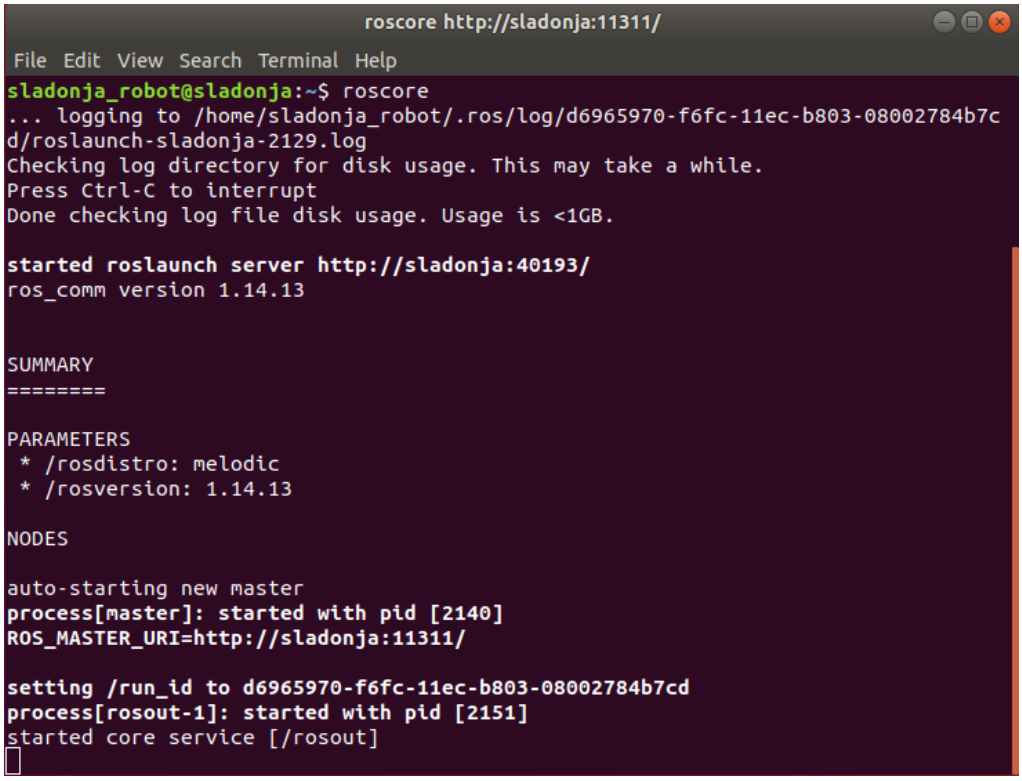
$ sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-wstool build-essential

$ sudo apt install python-rosdep – instalacija potrebnih *Python* paketa

$ sudo rosdep init – inicijalizacija *rosdep*-a

$ rosdep update **–** ažuriranje *rosdep*-a

Nakon uspješnog unošenja gore navedenih naredbi završava proces instalacije ROS-a. Ako se sada u komandni prozor unese naredba ***$ roscore*** trebao bi se dobiti rezultat prikazan na slici 5.5.



Slika 5.5. Izvršavanje naredbe "roscore"

5.3. URDF i SRDF datoteke

Za stvaranje funkcionalne simulacije u ROS-u potrebno je napraviti ujedinjeni format robotskog opisa (eng. *Unified Robotics Description Format*), skraćeno URDF. URDF je zapravo datoteka XML formata koja je posebno definirana za predstavljanje modela robota. Navedeni format se najčešće koristi u akademskim krugovima te u industriji za modeliranje kompleksnih robotskih sustava kao što su primjerice robotske ruke za proizvodne linije ili animatroničkih robota u zabavnim parkovima. URDF format je posebno popularan među korisnicima ROS-a iz razloga što nudi standardnu podršku za URDF modele. Kao i druge vrste XML datoteka, URDF datoteke sadrže različite XML elemente, kao što su primjerice <robot>, <link>, <joint> i druge, koji su hijerarhijski složeni u strukturu poznatu kao XML stablo. Na primjer, za elemente <link> i <joint> se kaže da su „djeca“ elementa <robot> i, recipročno, element <robot> „roditelj“ je elemenata <link> i <joint>. Podređeni elementi, kao što su <link> i <joint> pod elementom <robot>, mogu zauzvrat imati vlastite podređene elemente. Na primjer, element <link> ima podređene elemente <inertial> i <visual>. Element <visual> ima podređene elemente <geometry> i <material>. I element <material> ima podređeni element <color>. Takvi lanci podređenih elemenata neophodni su za definiranje svojstava i ponašanja roditeljskih elemenata (slika 5.6.).

```
<robot>
        <link>
                <inertial>
                        ...
                </inertial>
                <visual>
                        <geometry>
                                ...
                        </geometry>
                        <material>
                                <color />
                        </material>
                </visual>
        </link>
        ...
</robot>
```

Slika 5.6. Primjer elemenata u lanac XML datoteke

Osim podređenih elemenata, XML elementi u URDF modelu mogu imati atribute. Na primjer, elementi <robot>, <link> i <joint> imaju atribut <name> koji služi za identifikaciju elementa. Element <color> ima atribut rgb-a tj. numerički niz s vrijednostima crvene, zelene, plave boje. Atributi poput ovih pomažu u potpunom definiranju elemenata u modelu.

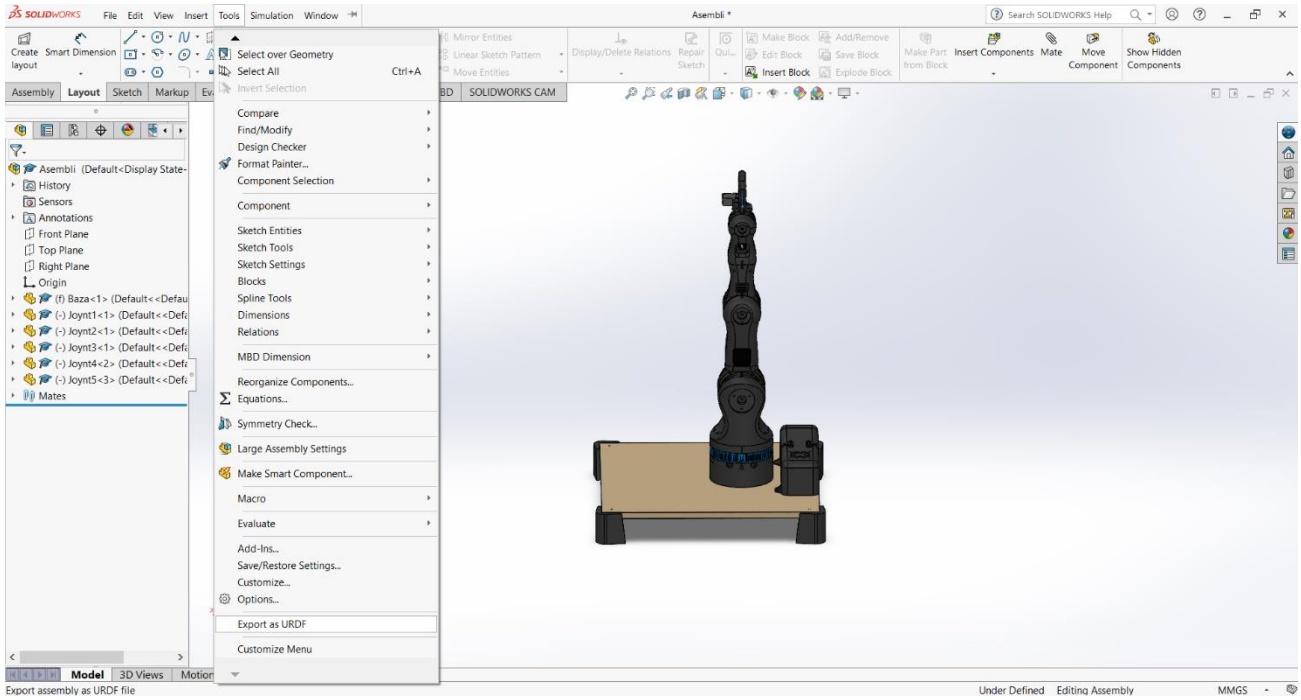
Semantički format opisa robota (eng. *Semantic Robot Description Format*) ili skraćeno SRDF nadopunjuje URDF datoteku. SRDF datoteka specificira zajedničke grupe zadane konfiguracije robota te sadrži dodatne informacije za provjeru kolizija zglobova robota kao i podatke o dodatnim transformacijama koje mogu biti potrebne za potpuno definiranje položaja robota. Preporučeni način za generiranje SRDF datoteke je korištenje *MoveIt Setup Assistant*-a.

5.3.1. Generiranje URDF datoteke

Za generiranje potrebne URDF Datoteke korišten je programski paket *SolidWorks* koji u sebi sadrži ekstenziju za stvaranje URDF datoteke iz postojećeg 3D modela. Iako u dostupnim datotekama već postoji funkcionalan sklop (eng. *Assembly*), koji bi se mogao upotrijebiti pri izradi URDF datoteke, preporučljivo je stvaranje vlastitog 3D sklopa koji sadrži što manji broj elemenata kako bi URDF datoteka imala što manji broj nepotrebnih elemenata. Time se olakšava sam postupak kreacije URDF datoteke, smanjuje se njena veličina te se pojednostavljuje kasnija konfiguracija prilikom kreiranja SRDF datoteke.

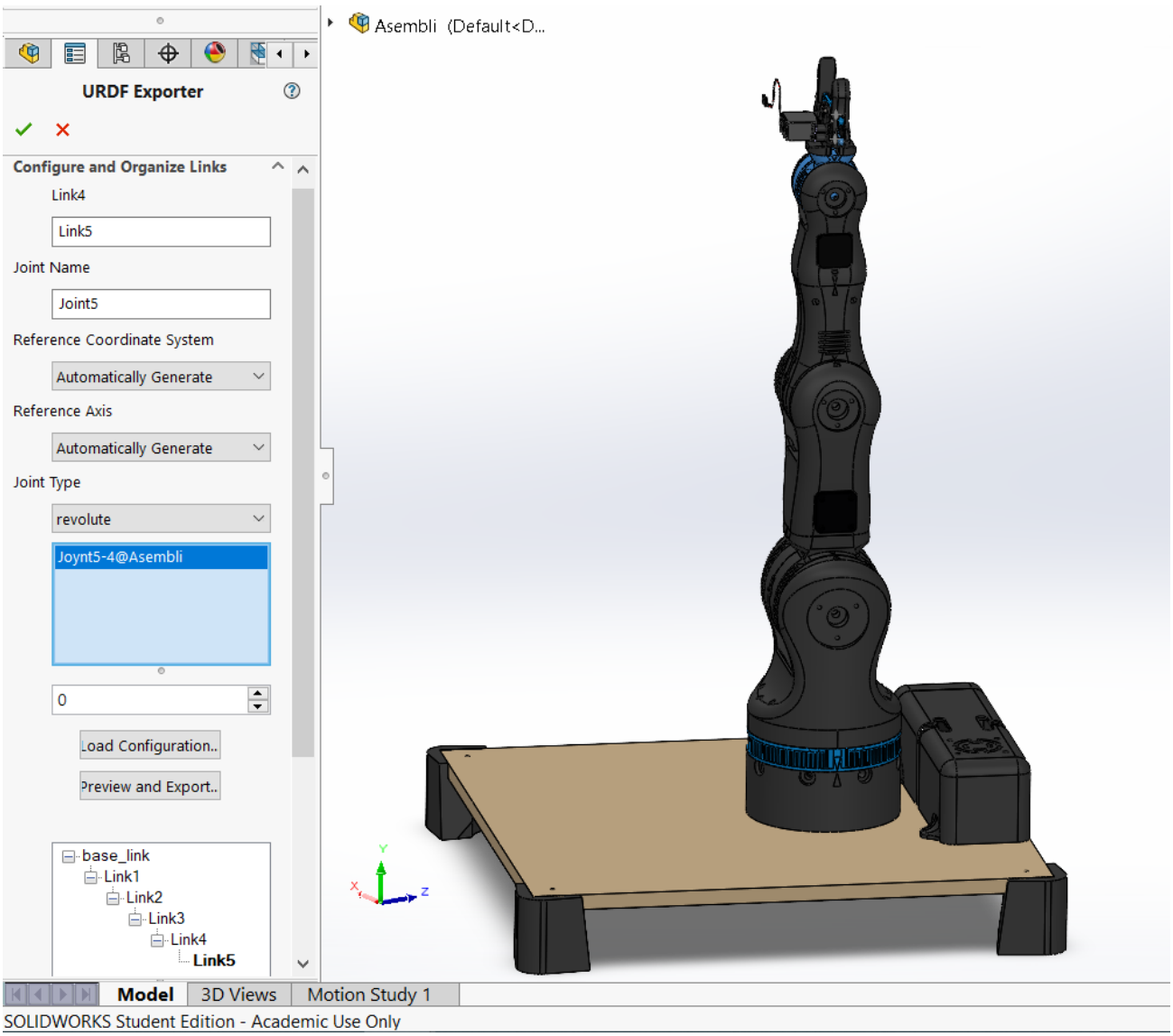
Nakon završetka postupka kreiranja funkcionalnog 3D sklopa može se pristupiti procesu kreiranja URDF datoteke. Valja naglasiti kako je ovaj proces iznimno delikatan i iziskuje veliku dozu pažnje jer korišteni alat nije dovoljno optimiziran i treba obratiti pozornost na poznate *bug*-ove koji se javljaju prilikom korištenja alata.

Postupak kreće odabirom alata za konverziju 3D sklopa u URDF datoteku, *Export as URDF*, koji se nalazi u padajućem izborniku *Tools* (slika 5.7.).



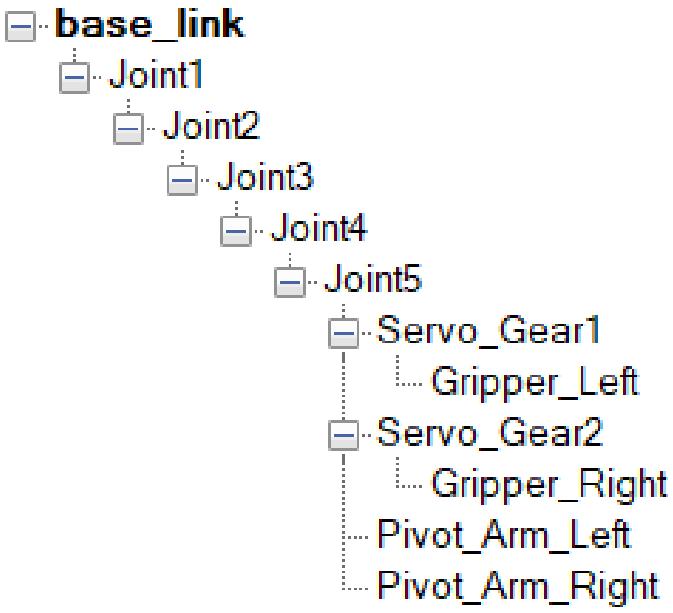
Slika 5.7. Odabir alata za izradu URDF datoteke

Zatim je potrebno odabrati dio robota koji predstavlja bazu tj. fiksni dio koji je nepokretan, što je u ovom slučaju element „Baza“. Zatim je potrebno definirati ime odabranog dijela, što je u ovom slučaju „base_link“. Potrebno je još definirati i broj elemenata koji se vežu na element „base_link“. U ovom slučaju imamo jedan element koji se veže za bazu robotske ruke, konkretno „Joint1“. Zatim je potrebno u stablu odabrati novokreirani element te odabrati navedeni dio „Joint1“. Također mu je potrebno dodijeliti ime te se postupak ponavlja onoliko puta koliko imamo zglobova. Zadnji zglob „Joint5“ se razlikuje po tome što je potrebno odabrati više elemenata tj. potrebno je odabrati i one elemente na koje se pričvršćuju elementi prihvatnice „Top Plate“ i „Bottom Plate“ (slika 5.8.).



Slika 5.8. Odabir elemenata "Joint5"

Postupak definiranja prihvatnice sa ponešto razlikuje od prethodnog postupka iz razloga što se jednim motorom upravlja većem brojem zglobova. Iz tog je razloga i struktura u stablu različita. Konačna struktura zglobova robotske ruke, uključujući prihvatnicu, dana je na slici 5.9.



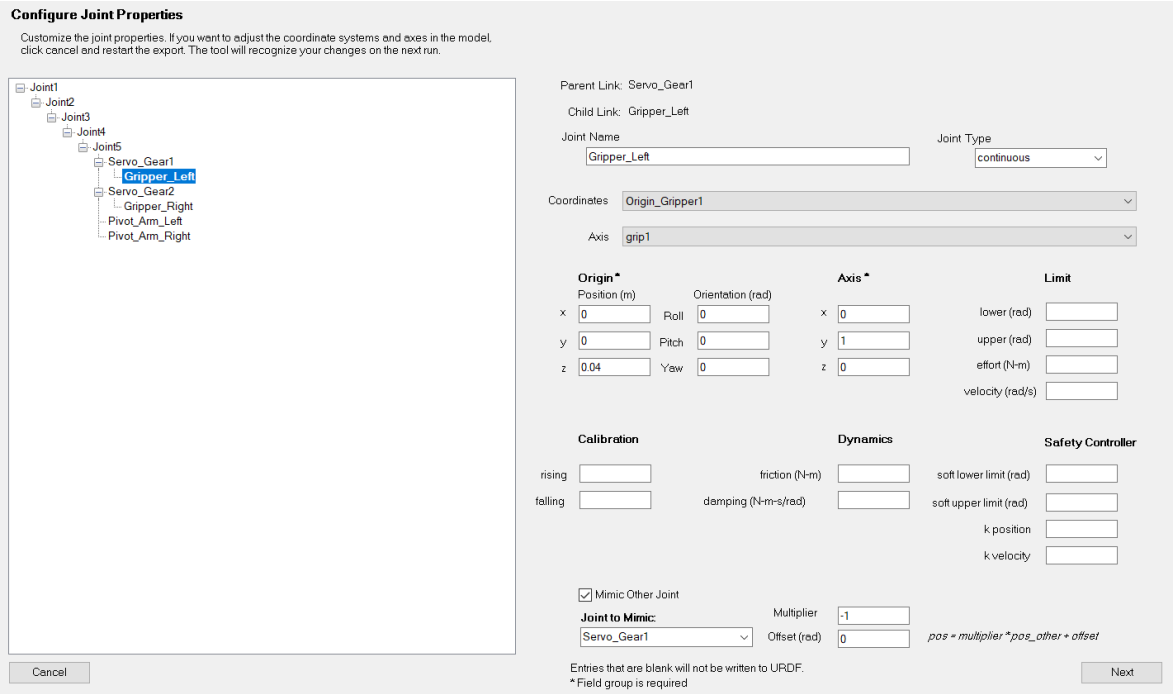
Slika 5.9. Struktura zglobova robotske ruke

Nakon završetka strukturiranja svih potrebnih elemenata u stablu, klikom na *Preview and Export,* generiraju se koordinatni sustavi sa potrebnim osima rotacije pojedinih zglobova. Zatim se otvara novi prozor *SolidWorks Assembly to URDF Exporter* kojeg je za sada potrebno zatvoriti. Kako su navedeni koordinatni sustavi i osi rotacije automatski generirani postoji šansa da su neadekvatno raspoređeni u odnosu na željene osi rotacije. Potrebno je ručno kreirati osi rotacije za svaki zglob te ih je zatim potrebno dodijeliti automatski generiranim osima. Također je potrebno konfigurirati koordinatne sustave svakog zgloba kako bi imali isti smjer i orijentaciju koordinatnih osi što uvelike olakšava neke od idućih koraka. Osi rotacije i ishodišta koordinatnih sustava moguće je definirati u prethodnoj fazi no prilikom dodjeljivanja navedenih parametra javlja se *bug* koji zanemaruje odabrane elemente. S toga je potrebno ručno postaviti željene osi rotacije i ishodišta koordinatnih sustava. Također je potrebno postaviti glavni koordinatni sustav na odabranu točku baze robota kako bi kasnije u simulaciji robotska ruka bila adekvatno pozicionirana u prostoru.

Nakon definiranja osi rotacije i koordinatnih sustava potrebno se vratiti u prozor *SolidWorks Assembly to URDF Exporter* ponovnim odabirom *Export as URDF* iz padajućeg izbornika *Tools*. Sada je potrebno definirati tip zgloba, na način da se u padajućem izborniku *Joint tipe* odabere opcija „Revolute“. To je potrebno napraviti za sve odabrane elemente. Iako se tip zgloba također mogao definirati prilikom definiranja elemenata zglobova, kao i kod odabira osi rotacije i ishodišta

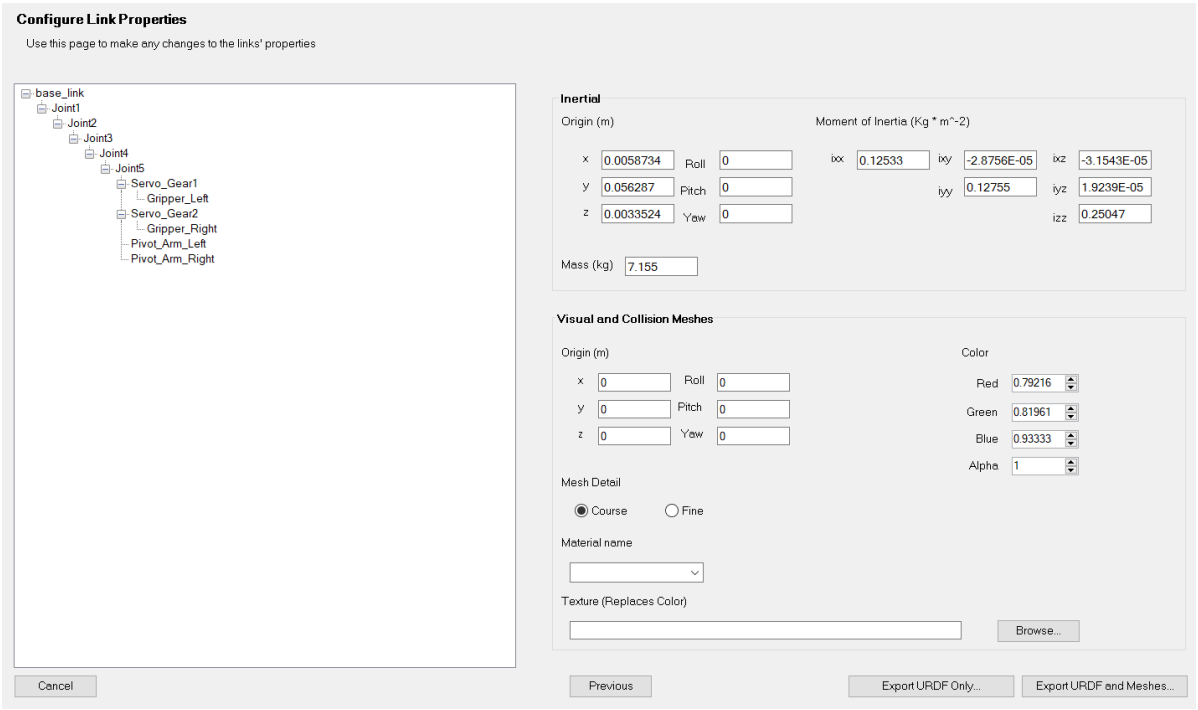
koordinatnog sustava, postoji *bug* koji većinu elemenata automatski definira kao fiksne. Također valja obratiti pozornost na polja „Axis“. Ako je prethodni postupak postavljanja osi rotacije i centra koordinatnih sustava ispravno odrađen tada se u poljima „x“ „y“ i „z“ nalaze vrijednosti „0" ili „1“ što znači da su zglobovi u odnosu na koordinatne osi s vrijednošću „0“ fiksni dok se rotacija zgloba odvija oko koordinatne osi koja ima vrijednost „1“. Potrebno je još provjeriti dali su dobro definirane osi rotacije i njihovi prefiksi pošto se zglobovi „Joint1“ i „Joint4“ rotiraju oko „z“ osi dok se zglobovi „ Joint2“, „Joint3“ i „Joint5“ rotiraju oko „x“ osi.

Kod konfiguracije zglobova prihvatnice također je potrebno obratiti pažnju na prethodno navedene parametre no potrebno je dodatno konfigurirati način njihovih kretnji. Element „Servo_Gear1“ pokreće servo motor te se navedeni element konfigurira kao i prethodni zglobovi. Nadalje, element „Gripper_Left“ mora oponašati kretnje elementa „Servo_Gear1“ i to recipročno, što se postiže tako da se postavi kvačica na opciju „Mimic Other Joint“ i odabire se navedeni element, „Servo_Gear1“, te je još potrebno u polje „Multiplier“ unijeti vrijednost „-1“ kako bi se definiralo recipročno oponašanje odabranog elementa. Slika Element „Servo_Gear2“ oponaša recipročnu vrijednost kretnje elementa „Servo_Gear1“, element „Gripper_Right“ oponaša recipročnu vrijednost kretnje elementa „Servo_Gear2“, element „Pivot_Arm_Left“ oponaša vrijednost kretnje elementa „Servo_Gear1“ dok element „Pivot_Arm_Righr“ oponaša vrijednost kretnje elementa „Servo_Gear2“.



Slika 5.10. Postavke zgloba "Gripper_Left"

Klikom na „Next“ pojavljuje se idući prozor *Configure Link Properties* gdje se mogu definirati vrijednosti boje pojedinog elementa, moment inercije, masa, vrsta materijala i dr. Ove vrijednosti nema potrebe mijenjati kako nemaju konkretnog utjecaja na daljnje korake (slika 5.11.).



Slika 5.11. Prozor „Configure Link Properties“

Klikom na *Export URDF and Meshes* kreira se potrebna URDF datoteka. Bitno je naglasiti da kod davanja imena treba obratiti pažnju na konvenciju koja nalaže da se moraju koristiti samo mala tiskana slova, samo određeni simboli te je zabranjeno korištenje razmaka.

5.4. Kreiranje ROS-paketa

Kako bi ROS „znao“ gdje potražiti i spremati potrebne datoteke potrebno je napraviti novu mapu. To je u suštini mapa koju ROS prepoznaje kao radni prostor (eng. *catkin workspace*) i u nju se spremaju i generiraju željeni ROS paketi. Ovu se mapu može kreirati klasičnim putem tj. desnim klikom miša i odabirom opcije *New folder* ili putem komandnog prozora. Ta se mapa najčešće kreira u direktoriju „Home“, no prvo je potrebno instalirati potrebne alate, putem komandnog prozora, koji koriste navedeni radni prostor:

$ sudo apt-get install ros-melodic-catkin python-catkin-tools – instalacija potrebnih *catkin* alata

Sada je moguće kreirati željeni radni prostor:

$ mkdir -p ~/bcn3d_ws/src – kreiranje radnog prostora pod nazivom „bcn3d_ws“ i podmape „src“

Ekstenzija -p omogućuje kreiranje podmape u glavnoj mapi. Sada je potrebno u mapu „src“ prebaciti prethodno generirane datoteke koje sadrže URDF datoteku Zatim je potrebno promijeniti lokaciju u komandnom prozoru u prethodno kreirani radni prostor što se postiže naredbom:

$ cd ~/bcn3d_ws – promjena lokacije komandnog prozora

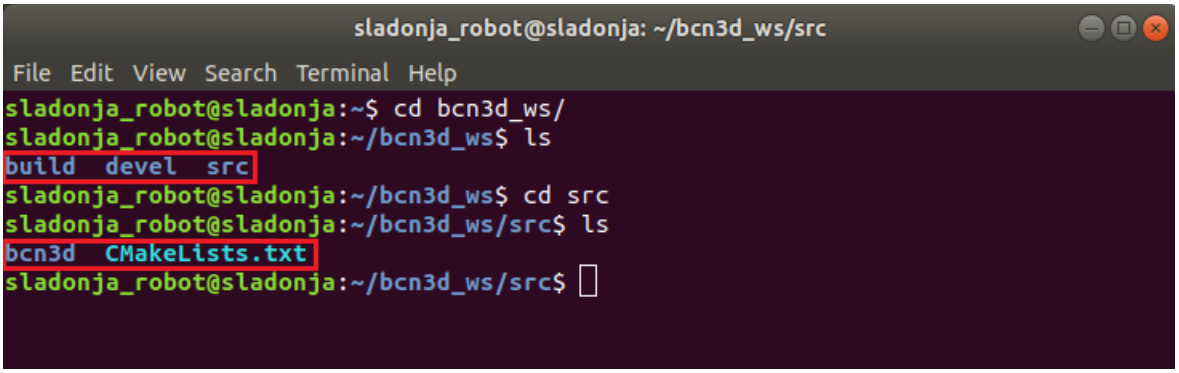
Kako bi se generirale preostale potrebne mape i datoteke potrebno je u komandni prozor unijeti naredbu:

$ catkin_make – naredba koja kreira potrebne datoteke i mape

$ source devel/setup.bash – zadavanje putanje novokreiranog radnog prostora

$ echo 'source devel/setup.bash' >> ~/.bashrc – sprema navedenu putanju u *.bashrc* datoteku

Nakon završetka izvršavanja prethodne naredbe moguće je primijetiti novokreiranu strukturu u mapi radnog prostora (slika 5.12.).



Slika 5.12. Novokreirana struktura u radnom prostoru

U mapi radnog prostora su kreirane dvije nove datoteke „build“ i „devel“. Mapa „build“ se smatra prostorom za razvoj tj. radni direktorij alata *catkin*. Mapa „devel“ se koristi za kreiranje datoteka koje služe za razvoj i testiranje. U mapi „src“ je kreirana datoteka „CMakeList.txt“ koja u sebi sadrži opis kreiranog projekta, pravila i ostale potrebne direktive potrebne za uspješno korištenje novokreiranog paketa.

5.4.1. Kreiranje simulacije robotske ruke

Za kreiranje i upravljanje simulacijom potrebno je instalirati *MoveIt!* paket s potrebnim vizualnim programima što se postiže naredbama:

$ sudo apt install ros-melodic-moveit – instalacija *MoveIt!* paketa

$ sudo apt-get install ros-melodic-moveit-visual-tools – instalacija potrebnih vizualnoh programa

Sada je moguće pokrenuti čarobnjaka za kreiranje simulacije robotske ruke korištenjem naredbe:

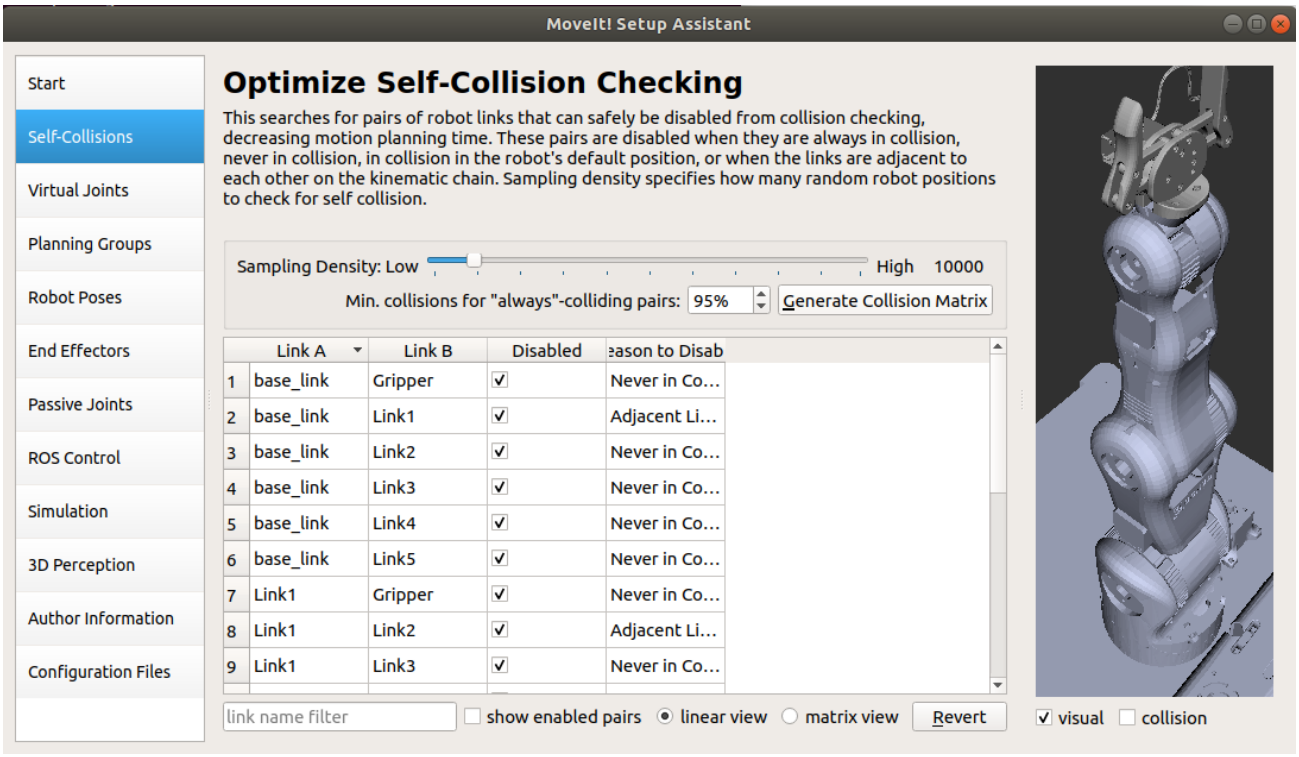
$ roslaunch moveit_setup_assistant setup_assistant.launch – pokretanje čarobnjaka za izradu simulacije (slika 5.13.)

Slika 5.13. Čarobnjak za izradu simulacije

Pomoću navedenog čarobnjaka kreira se novi paket koji sadrži sve potrebne datoteke u kojima su sadržane informacije pomoću kojih se definira robotska simulacija. Prvi je korak, nakon pokretanja simulacije, odabrati opciju za kreiranje novog paketa *Create New MoveIt Configuration Package*. Zatim je potrebno odabrati prethodno napravljenu URDF datoteku koja se nalazi unutar „src“ mape,

konkretno na lokaciji */home/sladonja_robot/bcn3d_ws/src/bcn3d/urdf/bcned.urdf*. Nakon odabira željene URDF datoteke, klikom na *Load Files*, čarobnjak javlja poruku „Success! Use the left navigation pane to continue“ što znači da je odabrana URDF datoteka uspješno učitana.

Prvi je korak, kreiranje matrice kolizije što se postiže klikom na *Generate Collision Matrix* (slika 5.14.). Na ovaj se način testiraju zglobovi koji mogu doći u međusobnu koliziju. Zatim se ti položaji zglobova automatski zabranjuju kako bi se izbjegla moguća oštećenja komponenata robota. Zabrana nepoželjnih kretnji također doprinosi smanjenju potrebnog vremena prilikom planiranja kretnji robota.

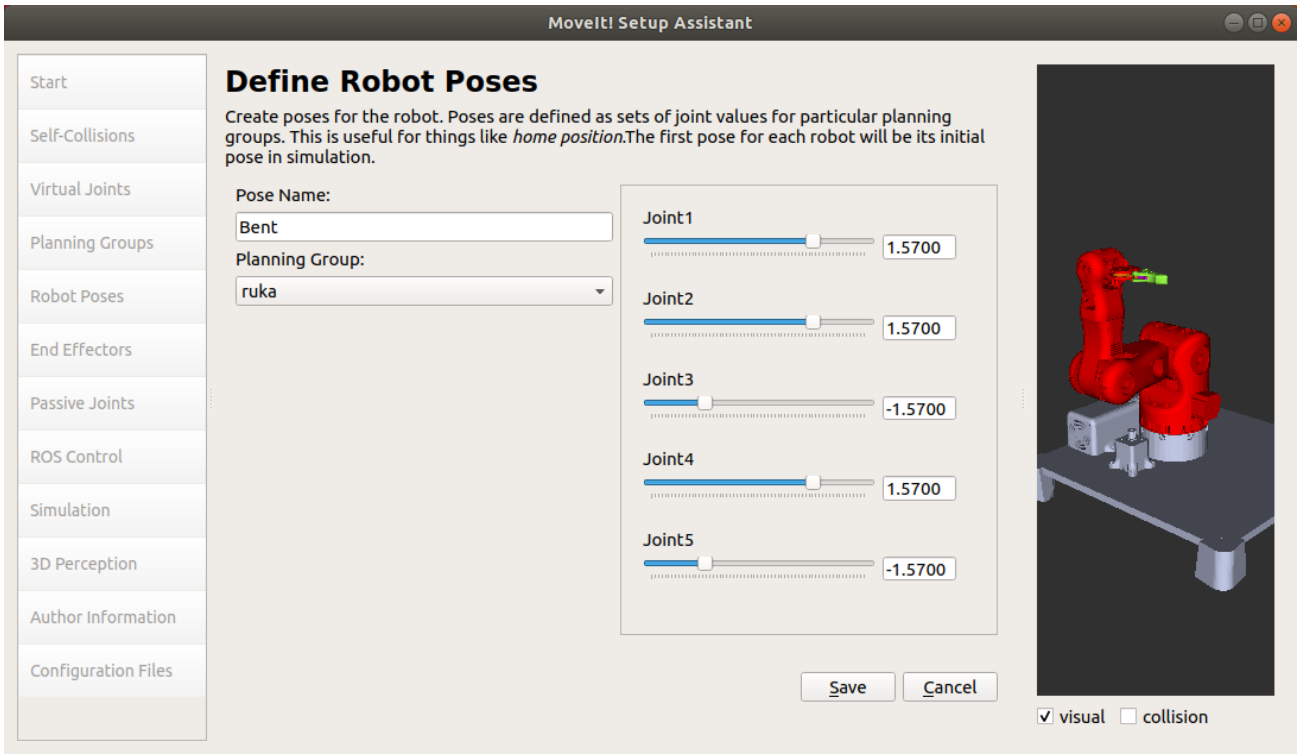


Slika 5.14. Generiranje matrice kolizija

Zatim se definiraju virtualni zglobovi (eng. *Virtual Joint*). To su oni zglobovi koji su zapravo nepokretni. U ovom je slučaju virtualni zglob „baza“ te se njega odabire u izborniku *Child link*. Potrebno je još definirati ime virtualnog zgloba „arm_virtual_joint“, njegov tip, „Fix“ te naziv prostora u kojemu se virtualni zglob nalazi „world“

Idući je korak definiranje glavnih grupa robota. U ovom se slučaju definira grupa „robot“ i grupa „gripper“. Kod definiranja grupe potrebno je svakoj grupi dodijeliti naziv te pripadajuće zglobove. Također je poželjno grupi „ruka“ dodijeliti jednog od ponuđenih kinematičkih rješavača. Oni služe kod proračunavanja kretnji zglobova robota kako bi tražene kretnje bile usklađene. U ovom je slučaju odabran *kdl* rješavač. Grupi „gripper“ nema potrebe dodjeljivati kinematički rješavača jer se kretnje zglobova odvijaju u jednoj ravnini.

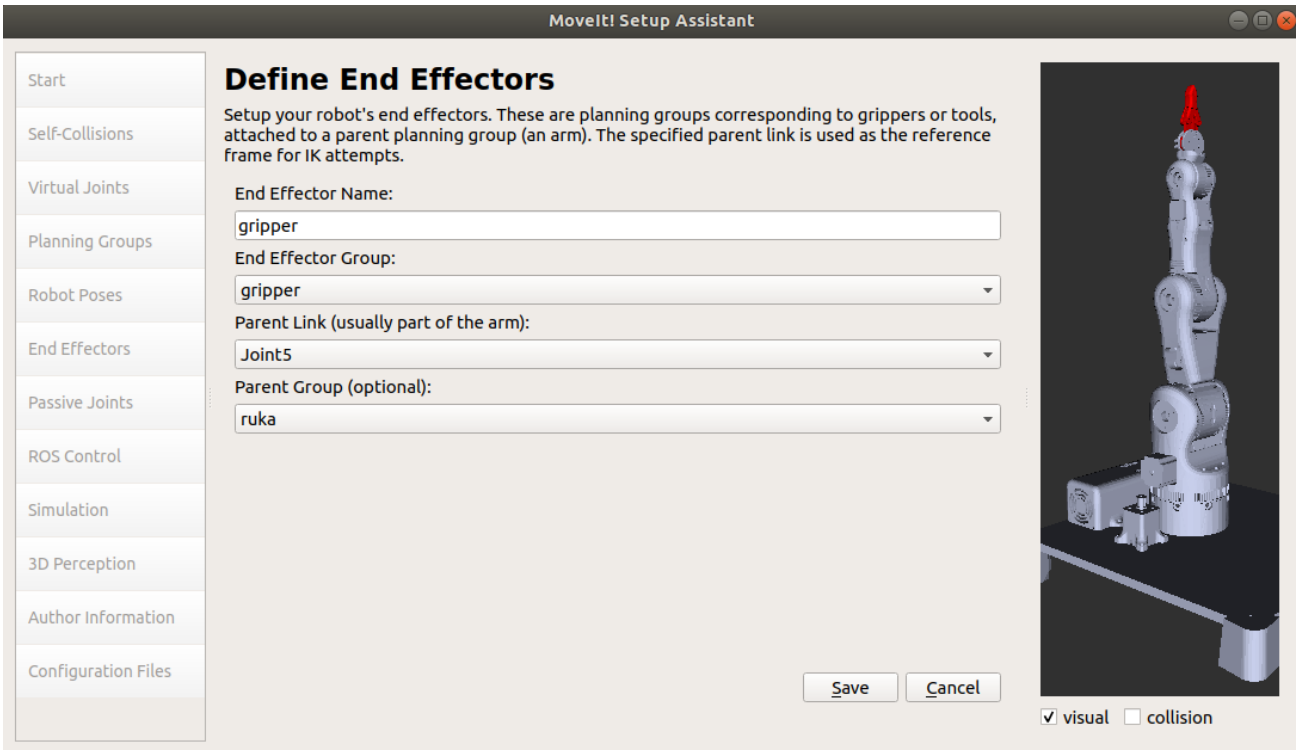
U kartici *Robot Pose* je moguće predefinirati željene poze robota pomoću kojih se kasnije može na jednostavan način postaviti robota u željeni položaj. Postupak kreće klikom na *Add Pose*, a zatim se odabire željena grupa. Mijenjanjem položaja klizača definira se željeni položaj svakog zgloba. Kreirana su dva položaja. Prvi položaj „Home“ definira početno stanje robotske ruke dok drugi položaj „Bent“ zakreće svaki zglob za devedeset stupnjeva (slika 4.15.). Također su definirane dvije pozicije za grupu „gripper“ koje postavljaju prihvatnicu u otvoreni i zatvoreni položaj.



Slika 5.15. Prikaz parametra položaja "Bent"

U kartici *End Effectors* se definira grupa prihvatnice. Postupak je sličan kao i kod postavljanja virtualnih zglobova. Dakle, potrebno je definirati naziv prihvatnice, grupu kojoj prihvatnica pripada i zglob na kojeg je pričvršćena. Također je moguće definirati grupu na koju je prihvatnica povezana

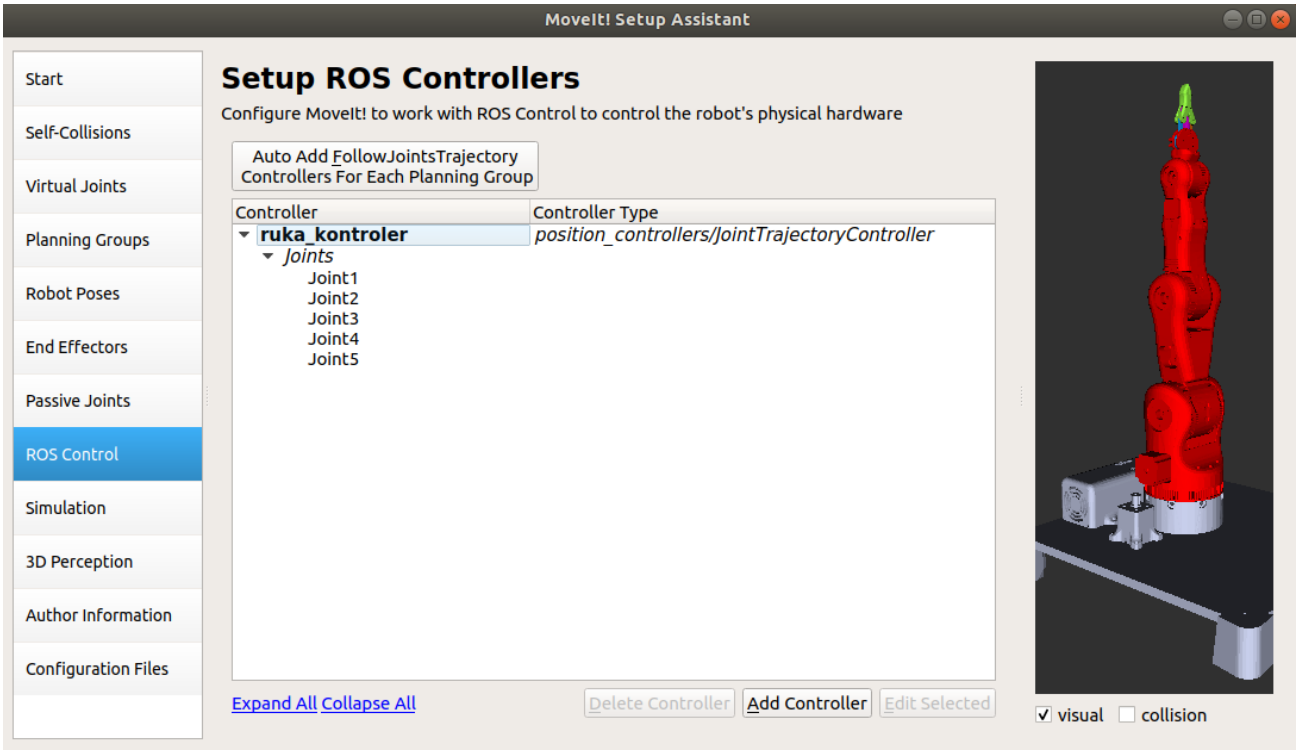
no u ovom slučaju to nije potrebno jer su definirane samo dvije grupe. Definirani parametri prikazani su na slici 5.16.



Slika 5.16. Definiranje grupe prihvatnice

Zatim se u kartici *Passive Joints* postavljaju pasivni zglobovi. To su svi zglobovi koji su indirektno pokretani primjerice elementi prihvatnice čiji je samo jedan element pokretan od strane motora. Konkretno su to zglobovi „Servo_gear2“, „Gripper_Left“, „Gripper_Right“, „Pivot_Arm_Left“ i „Pivot_Arm_Righr“.

U kartici „ROS Control“ se postavlja kontroler koji služi za pomicanje grupa robota. Moguće je automatski definirati kontrolere klikom na *Auto Add FollowJointsTrajectory Controllers For Each Planning Group*. Za ovu robotsku ruku dovoljno je definirati jedan kontroler koji je postavljen na vrhu zadnjeg zgloba „Joint5“. Ručno dodavanje se provodi tako što se klikne na *Add Controller* te se zatim dodjeljuje ime, tip kontrolera i grupa (ili posebno određeni zglob ili zglobovi) kojega se želi kontrolirati. Odabrani su parametri prikazani na slici 5.17.

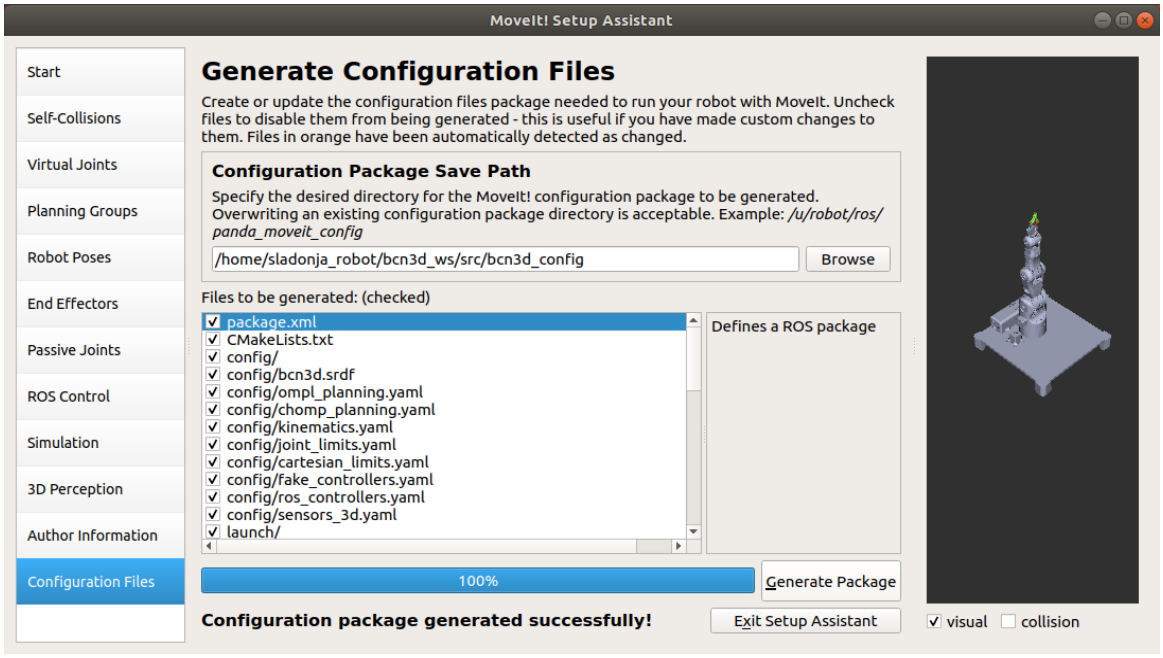


Slika 5.17. Definiranje kontrolera grupe "robot"

U kartici *Simulation* je moguće generirati modificiranu URDF datoteku koju se zatim zamjenjuje sa postojećom no taj korak se može preskočiti, dok se u kartici *3D Perception* mogu podesiti dodatni senzori primjerice kamere, senzori položaja i drugi, no u ovom slučaju se ne koristi nikakva vrsta senzora pa se ovu karticu može zanemariti.

U kartici *Author information* se unose podaci o autoru kreiranog paketa kao što su ime i prerime te adresa elektroničke pošte.

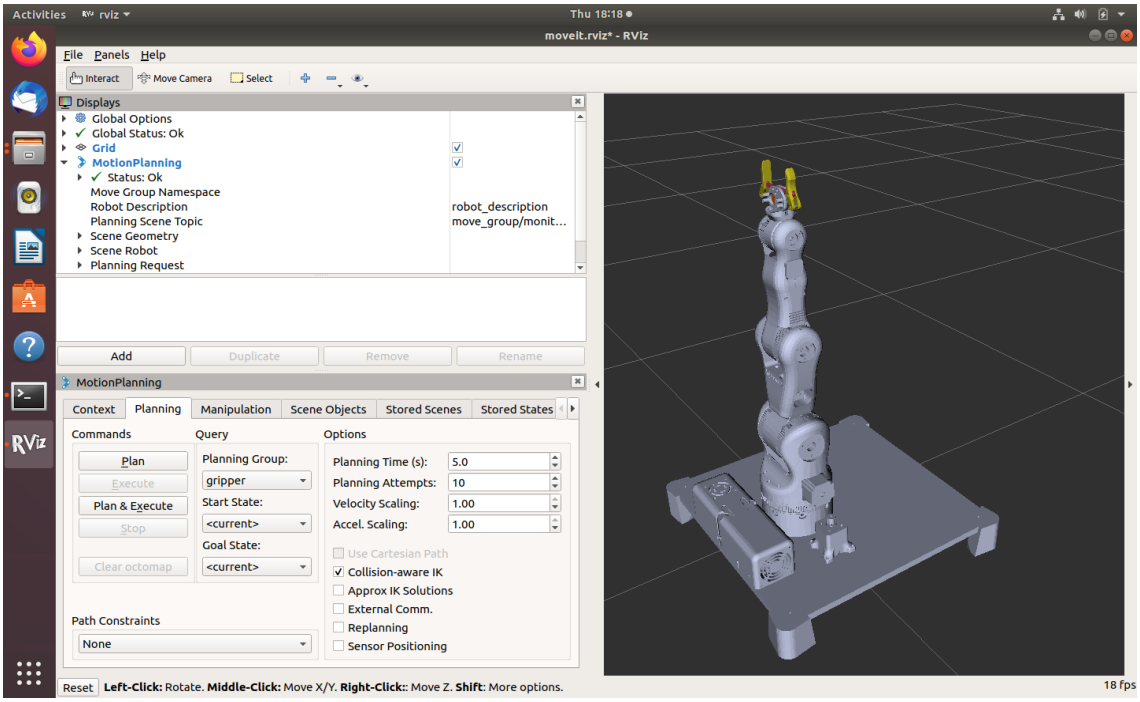
Za kraj, u kartici *Configuration Files* se odabiru mjesto i naziv za kreiranje ROS paketa. Najčešće se paketu dodjeljuje ime koje se sastoji od dva dijela koja su odvojena simbolom „_“. Prvi dio imena ostaje isti kao i ime generiranog paketa koji sadrži URDF datoteku dok je drugi dio naziva najčešće „config“, tako bi ime u ovom slučaju bilo „bcn3d_config“. Kreirani paket se također sprema u mapu „src“ gdje se već nalazi paket „bcn3d“ koji sadrži URDF datoteku. Preostaje još samo klikom na *Generate Package* generirati paket i klikom na *Exit Setup Assistent* zatvoriti čarobnjaka čime završava proces generiranja simulacije robotske ruke (slika 5.18.).



Slika 5.18. Odabir mjesta i naziva kreiranog paketa

Idućom se naredbom pokreće novokreirana simulacija kako bi se utvrdila uspješnost prethodnog postupka (slika 5.19):

$ roslaunch bcn3d_config demo.launch – pokretanje simulacije robotske ruke



Slika 5.19. Simuliranje robotske ruke

5.5. Implementacija skripte za pokretanje fizičkog robota

Kako bi bilo moguće upravljati fizičkim robotom potrebno je implementirati skriptu koja prenosi kretnje zglobova simuliranog robota na fizičkog robota. Potrebno je preuzeti skriptu i izmijeniti joj određene parametre kako bi bila funkcionalna za prethodno kreirane pakete. Također je potrebno napraviti novi ROS paket u kojega se sprema skriptu a je zatim potrebno instalirati i konfigurirati sučelje koje koristi mikrokontroler.

5.5.1. Kreiranje paketa za skriptu

Sada je potrebno kreirati novi ROS paket koji će sadržavati skriptu za pokretanje fizičkog robora. Kako bi kreirali paket potrebno je prebaciti lokaciju komandnog prozora u mapu „src“ gdje se već nalaze dva prethodno kreirana paketa naredbom:

$ cd ~/bcn3d_ws/src/ – premještanje u mapu „src“

Kreiranje novog paketa se sastoji od naredbe za kreaciju kojoj je potrebno pridodati određene ovisnosti (eng. Dependencies). ROS paketi ponekad zahtijevaju dodatne biblioteke i alate. Ove se knjižnice i alati obično nazivaju „Ovisnosti sustava“. Potrebne ovisnosti je također moguće dodavati ručno modificirajući datoteku „package.xml“ u „bcn3d_moveit“ paketu. Naredba za kreiranje paketa kojemu se pridružuju potrebne ovisnosti je:

$ catkin_create_pkg bcn3d_moveit std_msgs rospy roscpp message_generation message_runtime pluginlib eigen moveit_core moveit_ros_planning_interface moveit_ros_perception interactive_markers geometric_shapes moveit_visual_tools – instalacija paketa za skriptu uz popratne ovisnosti

Skriptu koja omogućuje upravljanje fizičkom robotskom rukom moguće je preuzeti iz dva izvora putem GitHub platforme [15] ili [16]. Preporuča se korištenje skripte iz prvog izvora iz razloga što je u robotu iz drugog izvora dodana šesta os rotacije što zahtijeva nešto više prepravljanja same skripte. Skripta je dana u Prilogu 1. Nakon preuzimanja skriptu je potrebno pohraniti u „src“ mapu „bcn3d_moveit“ paketa. Skriptu se tada može, ali i ne mora, preimenovati. Skripta je preimenovana u „bcn3d_convert“ kako bi joj ime bilo u skladu s projektom. U skripti je potrebno zamijeniti nazive svih varijabli koje se pozivaju na paket iz kojega je skripta preuzeta te ih promijeniti u

„bcn3d_moveit“ kako bi se te varijable pozivale na ovaj paket. Također je potrebno izmijeniti vektor „stepsPerRevolution“ koji se odnosi na broj koraka svakog koračnog motora. Potrebno je izračunati broj koraka koračnog motora potreban za rotaciju zgloba od 360°. Ta jednadžba sačinjava podatke o prijenosnom omjer zgloba, kut zakretanja jednog koraka koračnog motora i vrijednost mikrokoraka postavljenu na drajveru koristeći jednadžbu:

$$\frac{d_2}{d_1} \times \frac{360^\circ}{\alpha} \times m \quad (5.1)$$

gdje je:

d_2 – promjer gonjenog zupčanika

d_1 – promjer pogonskog zupčanika

α – kut zakretanja koračnog motora za jedan korak, $\alpha = 1{,}8^\circ$

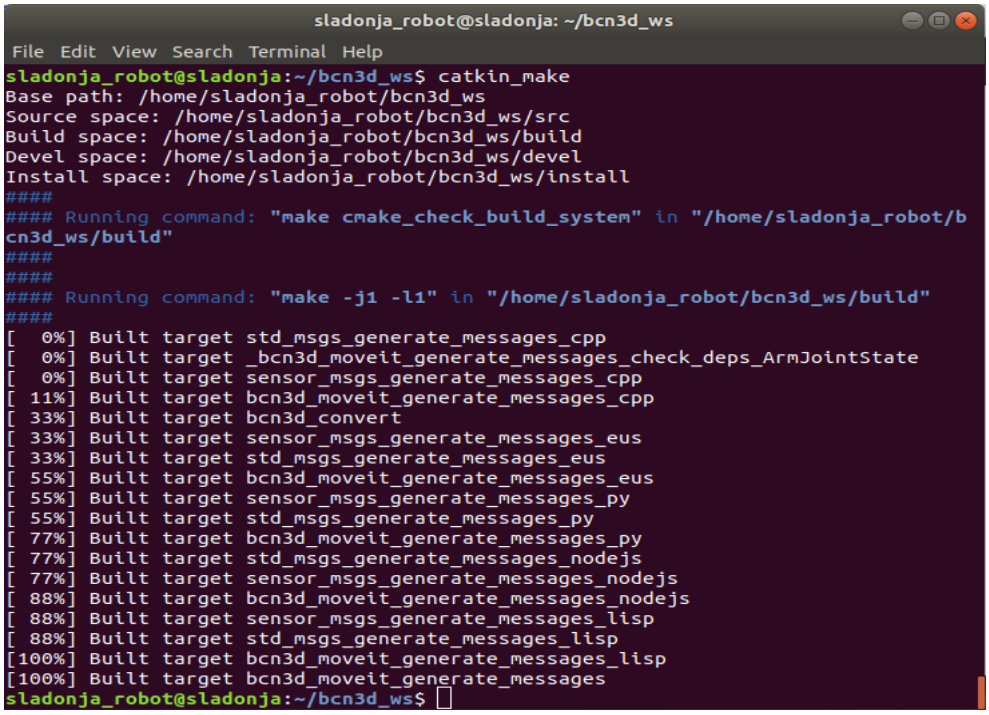
m – postavke mikrokoraka na drajveru

Vrijednosti dobivene putem ove jednadžbe je, za neke zglobove, potrebno djelomično ispraviti kako bi se postigla željena vrijednost rotacije. Bitno je naglasiti da je nakon svake izmjene u skripti potrebno izvršiti naredbu ***$ catkin_make*** kako bi se novo unesene vrijednosti implementirale. Potrebno je još modificirati datoteku „CMakeList.txt“ u paketu „bcn3d_moveit“. U toj su datoteci neke potrebne ovisnosti komentirane a potrebno je i ručno specificirati nazive potrebnih poruka te uključiti potrebne knjižnice. Sadržaj datoteke dan je u Prilogu 2. Za kraj je potrebno u komandnom prozoru prebaciti lokaciju u mapu „bcn3d_ws“ te koristeći naredbu ***$ catkin_make*** implementirati novokreiranu skriptu:

$ cd ~/bcn3d_ws/ – premještanje u mapu „bcnd3_ws“

$ catkin_make – implementacija skripte

U slučaju uspješnog postupka implementacije skripte, po završetku izvedbe naredbe ***$ catkin_make*** u komandnom se prozoru dobiva rezultat kao prikazan na slici 5.20.



Slika 5.20. Uspješna implementacija skripte

5.5.2. Postavljanje „Arduino Mega“ mikrokontrolera

Zadnji korak prije pokretanja fizičke robotske ruke je instalacija i konfiguracija sučelja mikrokontrolera. Instalaciju sučelja je moguće provesti na klasičan način, kao u operacijskom sustavu Windows ili putem komandnog prozora. U ovom je poglavlju opisan postupak koristeći komandni prozor. Prije same instalacije potrebno je preuzeti instalacijski paket sa službene stranice *Arduino*-a. Nakon preuzimanja je potrebno, koristeći komandni prozor, prebaciti se u instalacijsku mapu *Arduino*-a i pokrenuti instalacijsku datoteku (slika 5.21). Instalaciju se izvršava sljedećim naredbama:

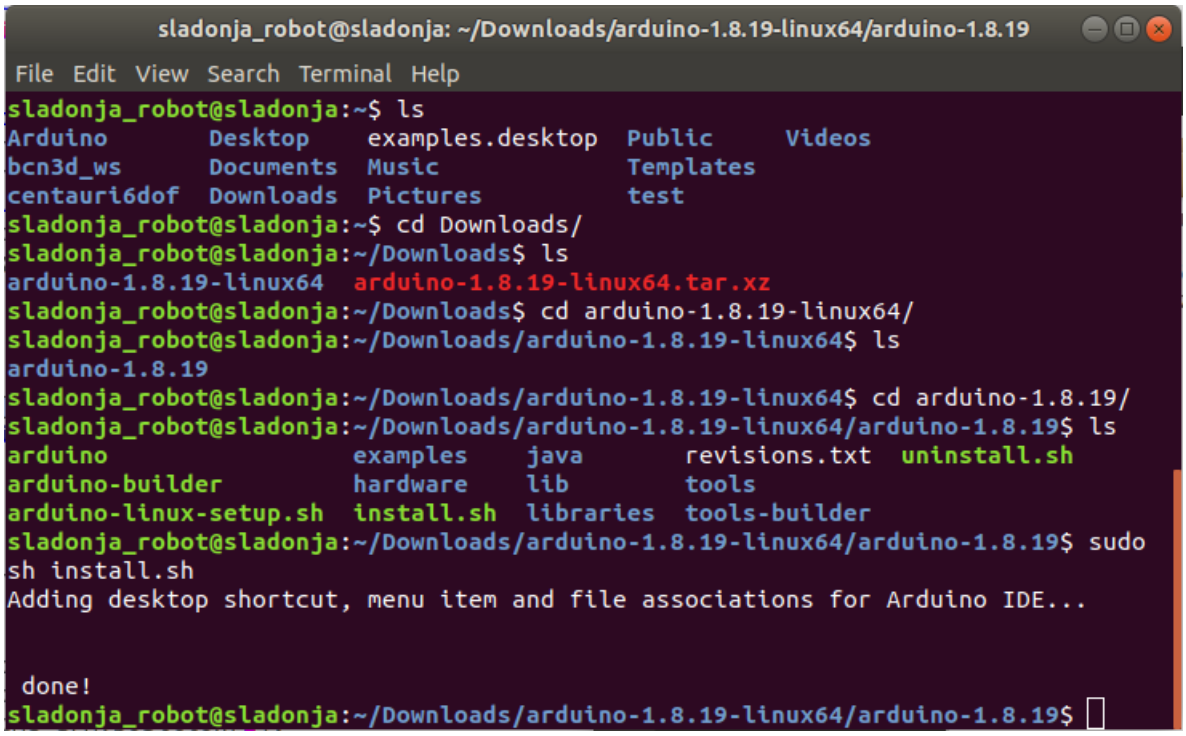
$ ls – ispis mapa i datoteka u trenutnoj mapi

$ cd Downloads – prebacivanje lokaciju u mapu „Downloads“

$ cd arduino-1.8.19-linux64/ – premještanje lokacije u mapu „arduino-1.8.19-linux64“

$ cd arduino-1.8.19/ –premještanje lokacije u mapu „arduino-1.8.19“

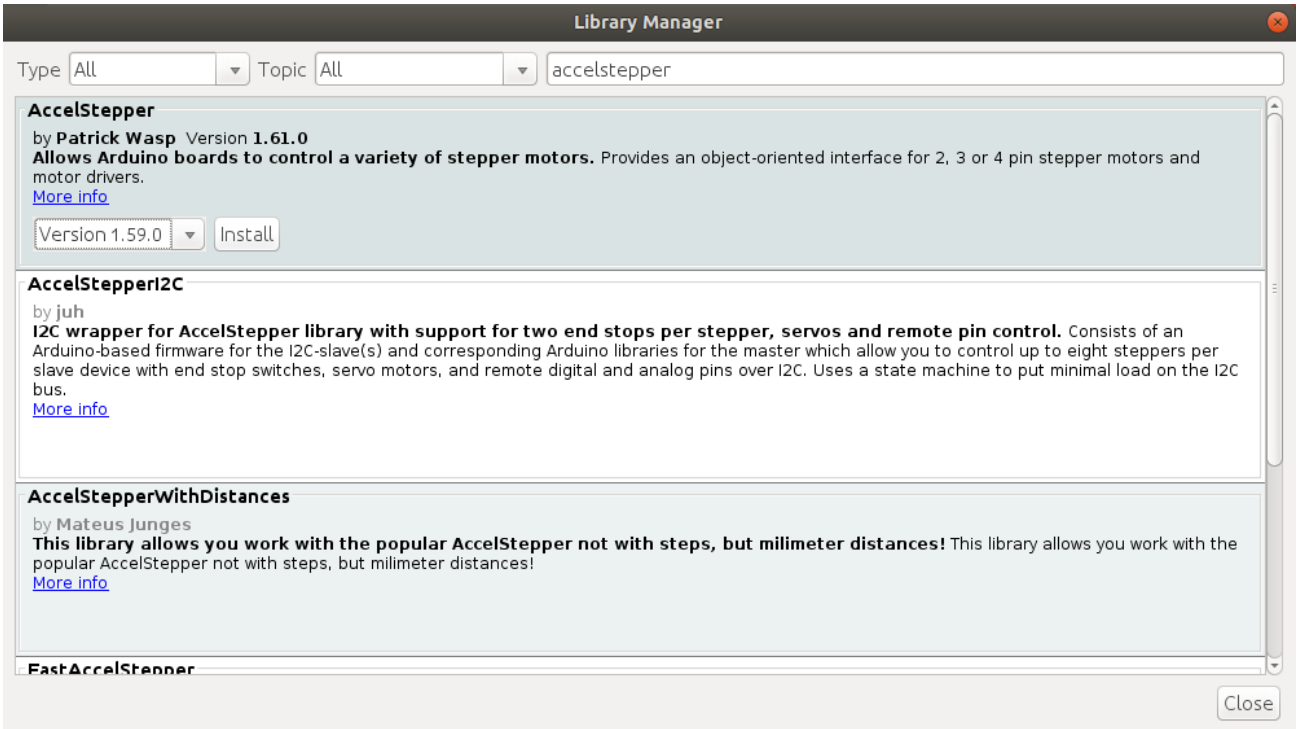
$ sudo sh install.sh – instalacija *Arduono* IDE sučelja



Slika 5.21. Instalacija "Arduino" IDE-a

Zatim je potrebno instalirati knjižnice “AccelStepper“ i „ros_lib“. One omogućuju upravljanje i komunikaciju koračnih motora s ROS-om. Kao i u prethodnom slučaju ove je biblioteke moguće instalirati na klasičan način ili putem komandnog prozora pa će u ovom slučaju biti prikazane obje metode.

Knjižnica „AccelStepper“ biti će instalirana klasičnom metodom. Postupak instalacije započinje tako što se otvara upravljačko sučelje *Arduino* IDE. Zatim je potrebno u kartici *Tools* odabrati *Manage Libraries…* i u tražilicu upisati „AccelStepper“ nakon čega će biti ponuđena opcija instalacije knjižnice. Moguće je odabrati neku od starijih verzija no preporučljivo je instalirati najnoviju dostupnu verziju (slika 5.22).



Slika 5.22. Instalacija knjižnice "AccelStepper"

Knjižnica „ros_lib“ je instalirana koristeći komandni prozor. Prije svega potrebno je instalirati alate *rosserial* i *rosserial-arduino* što se postiže izvršavanjem naredbi:

$ sudo apt-get install ros-indigo-rosserial-arduino – instalacija programa *rosserial-arduino*

$ sudo apt-get install ros-indigo-rosserial – instalacija programa *rosserial*

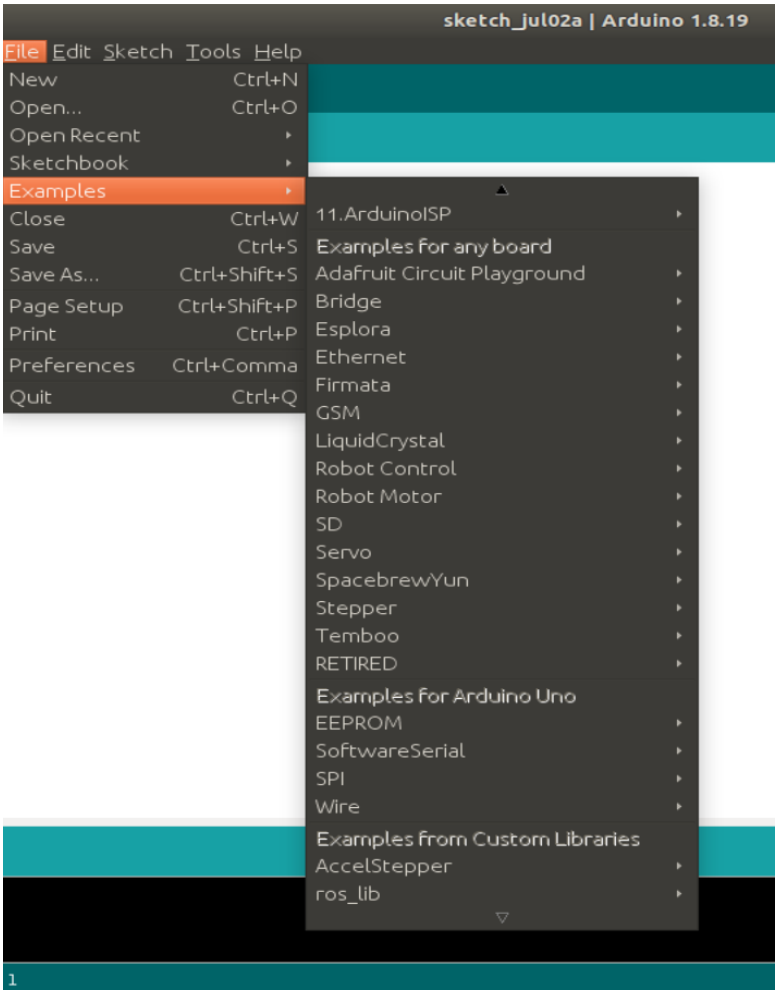
Nakon instalacije navedenih programa potrebno je promijeniti lokaciju komandnog prozora u mapu koja sadrži Arduino knjižnice:

$ cd ~/Arduino/libraries – promjena lokacije u mapu „libraries“

Sada je moguće instalirati „ros_lib“ knjižnicu koristeći naredbu:

$ rosrun rosserial_arduino make_libraries.py . – instalacija knjižnice „ros_lib“

Kako bi provjerili uspješnost instaliranih knjižnica potrebno je u kartici *File* odabrati *Examples* gdje bi trebalo biti dostupno nekoliko primjera navedenih knjižnica (slika 5.23.).



Slika 5.23. Prikaz knjižnica "AccelStepper" i "ros_lib"

Nakon instalacije potrebnih knjižnica potrebno je preuzeti i izmijeniti *Arduino*-v kod. Kod je također preuzet iz istog izvora kao i skripta [14] te se nalazi u Prilogu 3. U *Arduino*-vom kodu je također potrebno izmijeniti sve nazive varijabli koje se odnose na ROS paket „bcn3d_moveit“. Za kraj je potrebno priključiti *Arduino* u računalo i zatim je u kartici *Tools* potrebno odabrati korišteni *Arduino* mikrokontroler kao i korišteni ulaz. Korištenom ulazu je potrebno, putem komandnog prozora, dati dopuštenje korištenja serijske komunikacije:

$ ls -l /dev/ttyACM* – ispisuje korištene ulaze

$ sudo chmod a+rw /dev/ttyACM0 – dozvola serijske komunikacije ulazu „ACM0“

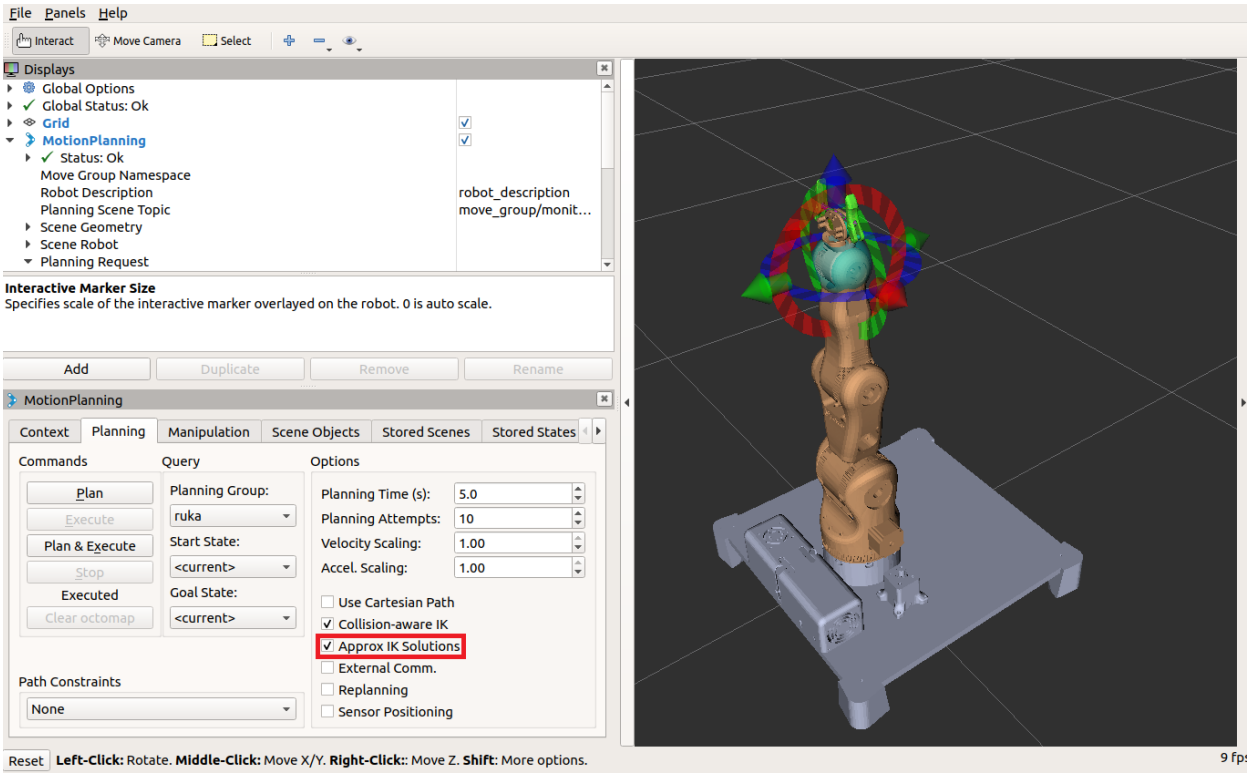
Nakon ovog koraka moguće je na *Arduino* učitati kod s čime završava proces instalacije i konfiguracije svih potrebnih parametara za upravljanjem fizičke robotske ruke.

5.6. Upravljanje robotskom rukom

Nakon uspješnog provođenja gore navedenih postupaka konačno je moguće prenijeti kretnje sa simulacije robotske ruke na fizičku robotsku ruku. Potrebno je pokrenuti četiri zasebna komandna prozora te u njih unijeti sljedeće naredbe:

1. Komandni prozor: ***$ roscore*** – pokreće zbirku čvorova i programa koji su preduvjeti ROS sustava te omogućuje komunikaciju između čvorova
2. Komandni prozor: ***$ roslaunch bcn3d_config demo.launch*** – pokretanje simulacije robotske ruke
3. Komandni prozor: ***$ rosrun rosserial_python serial_node.py /dev/ttyACM0*** – ROS čvor koji omogućuje serijsku komunikaciju
4. Komandni prozor: $ rosrun bcn3d_moveit bcn3d_convert – pokretanje skripte „bcn3d_convert“

Kako bi bilo moguće mijenjati položaj simulirane robotske ruke potrebno je označiti opciju *Approx IK Solutions* (slika 5.24.).



Slika 5.24. Upravljanje robotskom rukom

Ova opcija omogućuje korištenje inverzne kinematike koja služi za planiranje kretnji svih zglobova kako bi se postigla idealna putanja prihvatnice robotske ruke. Sada se svaki pokret sa robotske simulacije translatira na fizičkog robota.

Upravljanje fizičkog robota se može postići na nekoliko načina. Najjednostavniji način je odabir jedne od predefiniranih pozicija prilikom konfiguracije koristeći čarobnjaka. Ove se pozicije odabiru u padajućem izborniku *Start State* i *Goal State*. Tu je također moguće odabrati opcije *random valid* koja postavlja robotsku ruku u slučajnu poziciju vodeći računa o koliziji zglobova, opcija *random* postavlja robotsku ruku u slučajnu poziciju ne vodeći računa o koliziji zglobova pa se ta opcija uglavnom izbjegava, opcija *current* postavlja početno ili krajnje stanje robota u trenutnu poziciju dok opcija *previous* postavlja robota u prijašnji položaj. Moguće je također dodavanje novih položaja pozivanjem čarobnjaka ali se tada odabere opcija *Edit Existing MoveIt Configuration Package*

Iduća metoda je direktno manipuliranje simulacije pomoću interaktivnog znaka (eng. Interactive Marker). Prvo je potrebno odabrati željenu grupu kojom se želi upravljati. Željena se grupa odabire u padajućem izborniku *Planning Group*. Kako bi se upravljalo robotskom rukom potrebno je odabrati grupu „ruka“. Postavljanje robota u željenu poziciju se postiže tako što se mišem pomiče interaktivni marker. Zatim je potrebno kliknuti na *Plane* čime se planira optimalna putanja robota iz početnog položaja do željenog položaja. Nakon uspješnog planiranja pitanje moguće je na simulaciji vidjeti kretanje po isplaniranoj putanji. Klikom na *Execute* izvršava se kretnja po prethodno isplaniranoj putanji. Klikom na *Plane & Execute* robot automatski mijenja položaj u željenu poziciju odmah po uspješnom završetku planiranja putanje.

Robotskom rukom se može upravljati i korištenjem komandnog prozora koristeći naredbu:

```
$ rostopic pub joint_steps moveo_moveit/ArmJointState <Joint1 Joint2 Joint3 Joint4 Joint5 0>
```

- upravljanje robotskom rukom putem komandnog prozora

Ova metoda upravljanja koristi direktnu kinematiku. Moguće je zglobovima upravljati pojedinačno ili upravljati svim zglobovima od jednom. Koristeći ovu metodu potrebno je obratiti pažnju na unesene vrijednosti pomaka jer se u ovom slučaju ne predviđa kolizija već robot slijepo prati kretnje po zadanim parametrima.

6. ZAKLJUČAK

Stjecanje znanja iz područja robotike sve je veći imperativ inženjerima strojarstva i elektrotehnike. Iz tog razloga javlja se potreba za razvojem edukacijskih robota pomoću kojih će mladi inženjeri stjecati potrebna znanja i vještine za rad u tom području. Izvedba edukacijskog robota mora prvenstveno biti jeftinija u odnosu na industrijske robote, jednostavnijih oblika uz mogućnost nadogradnje. Poželjno je da su edukacijski roboti dostupni na *open-source* platformama kako bi se omogućio pristup svima željnim znanja a velika je prednost da se struktura robota može proizvesti korištenjem 3D printera.

Cilj ovog rada bio je opisati kompletan postupak instalacije okvira ROS korištenog za upravljanje robotskom rukom BCN3D MOVEO. ROS nudi veliki broj alata korištenih u svrhu upravljanja robotskih sustava a valja naglasiti da je ROS besplatan i svima dostupan.

U ovom je radu opisan kompletan postupak izrade edukacijske robotske ruke BCN3D MOVEO uključujući postupke 3D printanja potrebnih dijelova, postupak montaže i postupak upravljanja. Aktuatorski sustav se sastoji od pet *Nema* koračnih motora i jednog servo motora. Koračnim motorima se upravlja *Driver*-ima TB-6560. U svrhu upravljačke jedinice korišten je *Arduino MEGA 2560* mikrokontroler u kombinaciji s *RAMPS 1.4 Shield*-om. Robotskom rukom je moguće upravljati koristeći grafičko sučelje *MoveIt* u sklopu ROS-a ili korištenjem komandnog prozora. U radu je također izveden kinematički model robotske ruke korišten u svrhu rješavanja direktnog kinematičkog problema pomoću DH metode.

U svrhu izrade ove robotske ruke korištena su znanja stečena tokom studiranja iz područja programiranja, robotike, mehatronike, automatizacije i 3D modeliranja. Znanja iz navedenih područja su imperativ za upravljanje robotskim sustavima zbog njihove kompleksne strukture.

SAŽETAK I KLJUČNE RIJEČI

U ovom je radu opisana procedura prilagodbe ROS-a u svrhu upravljanja robotskom rukom BCN3D MOVEO Rad je podijeljen u šest poglavlja. U uvodnom poglavlju se govori o potrebi za razvojem industrijske robotike i o potencijalu ROS-a za primjenu u industrijskoj robotici. U drugom se dijelu opisuje povijest ROS-a, njegova struktura, prednosti i nedostaci. Također su opisani glavni alati ROS-a korišteni u svrhu ovog rada. U trećem se poglavlju opisuje korištena robotska ruka, 3D printanje njenih komponenata, način spajanja aktuatorskih komponenata te analiza naprezanja i deformacije. U četvrtom je poglavlju izračunata kinematika robotske ruke dok se u petom poglavlju detaljno opisuje postupak implementacije ROS okvira za upravljanje robotskom rukom.

Ključne riječi: ROS Melodic, BCN3D MOVEO, platforma otvorenog koda, vođenje robota, edukacijski robot

ABSTRACT AND KEY WORDS

This thesis describes the ROS adaptation procedure for the purpose of controlling the BCN3D MOVEO robotic arm. The thesis is divided into six sections. The introductory section describes the need for the development of industrial robots and the potential of ROS for application in industrial robotics. The second part describes the history of ROS, its structure, advantages and disadvantages. The main ROS tools used for the purpose of this thesis are also described. The third section describes the robotic arm, 3D printing of its components, the method of connecting the actuator components, and stress and deformation analysis. In the fourth section the kinematics of the robotic arm is calculated, while in the fifth chapter, the implementation procedure of the ROS framework for controlling the robotic arm is described in detail.

Key words: ROS Melodic, BCN3D MOVEO, open source, vođenje robota, educational robot

POPIS OZNAKA I KRATICA

ROS – robotski operacijski sustav (eng. Robot operating system)

CAD – oblikovanje pomoću računala (eng. Computer-aided Design)

URDF – ujedinjeni format robotskog opisa (eng. *Unified Robotics Description Format*)

SRDF – semantički format opisa robota (eng. *Semantic Robot Description Format*)

IDE – integrirano razvojno okruženje (eng. Integrated development environment)

RAMPS - repap Arduino mega pololu štit (eng. *Repap Arduino mega pololu shield*)

FDM – taložno očvršćivanje (eng. *Fused Deposition Modeling*)

PolyJet – tiskanje sloja fotoosjetljivog polimera (eng. *Photopolymer Jetting*)

ABS – akrilonitril/butadien/strien (eng. *Acrylonitrile butadiene styrene*)

θ_k – kut zakreta oko Z osi

d_k – odmak članka po Z osi

α_k – kut rotacije oko X osi,

${}^{5}_{0}T$ – matrica direktnog kinematičkog problema

POPIS SLIKA

POPIS TABLICA

LITERATURA

[1] „ROS Melodic Morenia“, s Interneta, http://wiki.ros.org/melodic 5. lipnja 2022.

[2] Ramkumar, G.; Lentin, J.: „ROS Robotics Projects“, Packt Publishing, Birmingham, 2019.

[3] Tellez, R.: „A History of ROS (Robot Operating System)“, s interneta, https://www.theconstructsim.com/history-ros/, 10. lipnja 2022.

[4] „Distributions“, s interneta, http://wiki.ros.org/Distributions, 7. lipnja 2022.

[5] Lentin, J.; Jonathan, C.: „Mastering ROS for Robotics Programming - Second Edition“ Packt Publishing, Birmingham, 2018.

[6] Fairchild, C.; Harman, T.: „ROS Robotics By Example - Second Edition“, Packt Publishing, Birmingham, 2017.

[7] „Moving robots into the future“, s interneta, https://moveit.ros.org/, 5. travnja 2022.

[8] „About BCN3D Technologies“, s Interneta, https://www.bcn3dtechnologies.com/en/about-us, 1. svibnja 2022.

[9] Marohnić, T.: „Uvod u ispravno konstruiranje i oblikovanje proizvoda za izradu tehnologijom 3D printa“, predavanje iz kolegija „Konstruiranje i oblikovanje”

[10] „Z-ULTRAT“, s Interneta, https://zortrax.com/filaments/z-ultrat, 2.svibnja 2022.

[11] Topčić, A.; Lovrić, S.; Fajić, A.; Cerjaković, E.: „Brza izrada prototipa i reverzibilno inženjerstvo u proizvodnji alata za livenje u pijesku“, M-PRINT Mostar, Tuzla, 2016.

[12] Sladonja, I.: „Izrada robotske ruke primjenom aditivne tehnologije“, završni rad, 2019.

[13] „RAMPS 1.4 Assembly Guide“, s Interneta https://www.reprap.org/mediawiki/images/0/06/RAMPS_dossier.pdf, 25. svibnja 2022.

[14] „Ubuntu install of ROS Melodic“, s Interneta, http://wiki.ros.org/melodic/Installation/Ubuntu, 15. lipnja 2022.

[15] „moveo ros“, s Interneta, https://github.com/jesseweisberg/moveo_ros, 30 srpnja 2018.

[16] „centauri6dof“, s Interneta, https://github.com/andresaraque/centauri6dof, 2. travnja 2021.

PRILOZI

Prilog 1. Skripta za upravljanje robotskom rukom

```
#include "ros/ros.h"
#include "sensor_msgs/JointState.h"
#include "bcn3d_moveit/ArmJointState.h"
#include "math.h"

bcn3d_moveit::ArmJointState arm_steps;
bcn3d_moveit::ArmJointState total;
int stepsPerRevolution[6] = {24800,140200,8500,400,0,0};
int joint_status = 0;
double cur_angle[6];
int joint_step[6];
double prev_angle[6] = {0,0,0,0,0,0};
double init_angle[6] = {0,0,0,0,0,0};
double total_steps[6] = {0,0,0,0,0,0};
int count = 0;



void cmd_cb(const sensor_msgs::JointState& cmd_arm)
{
  if (count==0){
    prev_angle[0] = cmd_arm.position[0];
    prev_angle[1] = cmd_arm.position[1];
```

```
    prev_angle[2] = cmd_arm.position[2];
    prev_angle[3] = cmd_arm.position[3];
    prev_angle[4] = cmd_arm.position[4];
    prev_angle[5] = cmd_arm.position[5];

    init_angle[0] = cmd_arm.position[0];
    init_angle[1] = cmd_arm.position[1];
    init_angle[2] = cmd_arm.position[2];
    init_angle[3] = cmd_arm.position[3];
    init_angle[4] = cmd_arm.position[4];
    init_angle[5] = cmd_arm.position[5];
    }

  ROS_INFO_STREAM("Recibido /move_group/fake_controller_joint_states");
  arm_steps.position1 = (int)((cmd_arm.position[0]-
prev_angle[0])*stepsPerRevolution[0]/(2*M_PI));
  arm_steps.position2 = (int)((cmd_arm.position[1]-
prev_angle[1])*stepsPerRevolution[1]/(2*M_PI));
  arm_steps.position3 = (int)((cmd_arm.position[2]-
prev_angle[2])*stepsPerRevolution[2]/(2*M_PI));
  arm_steps.position4 = (int)((cmd_arm.position[3]-
prev_angle[3])*stepsPerRevolution[3]/(2*M_PI));
  arm_steps.position5 = (int)((cmd_arm.position[4]-
prev_angle[4])*stepsPerRevolution[4]/(2*M_PI));
```

```
 arm_steps.position6 = (int)((cmd_arm.position[5]-prev_angle[5])*stepsPerRevolution[5]/(2*M_PI));

 ROS_INFO_NAMED("test", "arm_steps.position6 #2: %d", arm_steps.position6);

 if (count!=0){
  prev_angle[0] = cmd_arm.position[0];
  prev_angle[1] = cmd_arm.position[1];
  prev_angle[2] = cmd_arm.position[2];
  prev_angle[3] = cmd_arm.position[3];
  prev_angle[4] = cmd_arm.position[4];
  prev_angle[5] = cmd_arm.position[5];
 }

 total.position1 += arm_steps.position1;
 total.position2 += arm_steps.position2;
 total.position3 += arm_steps.position3;
 total.position4 += arm_steps.position4;
 total.position5 += arm_steps.position5;

 ROS_INFO_NAMED("test", "total_steps[4]: %f, total: %d", total_steps[5], total.position6);
 ROS_INFO_NAMED("test", "arm_steps.position5 #3: %d", arm_steps.position6);
 ROS_INFO_STREAM("Conversion hecha a /joint_steps");
 joint_status = 1;
```

```
  count=1;
}
int main(int argc, char **argv)
{
  ros::init(argc, argv, "bcn3d_moveit");
  ros::NodeHandle nh;
  ROS_INFO_STREAM("Funcion principal");
  ros::Subscriber sub = nh.subscribe("/move_group/fake_controller_joint_states",1000,cmd_cb);
  ros::Publisher pub = nh.advertise<bcn3d_moveit::ArmJointState>("joint_steps",50);
  ros::Rate loop_rate(50);
  while (ros::ok())
  {
    if(joint_status==1)
    {
      joint_status = 0;
      pub.publish(total);
      ROS_INFO_STREAM("Publicado en /joint_steps");
    }
    ros::spinOnce();
    loop_rate.sleep();
  }
  ros::spin();
  return 0;
}
```

Prilog 2. Sadržaj datoteke „CMakeList.txt“

```
cmake_minimum_required(VERSION 3.0.2)
project(bcn3d_moveit)
## Compile as C++11, supported in ROS Kinetic and newer
add_compile_options(-std=c++11)
find_package(Eigen3 REQUIRED)
# Eigen 3.2 (Wily) only provides EIGEN3_INCLUDE_DIR, not EIGEN3_INCLUDE_DIRS
if(NOT EIGEN3_INCLUDE_DIRS)
  set(EIGEN3_INCLUDE_DIRS ${EIGEN3_INCLUDE_DIR})
endif()
## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  geometric_shapes
  interactive_markers
  message_generation
  message_runtime
  moveit_core
  moveit_ros_perception
  moveit_ros_planning_interface
  moveit_visual_tools
  pluginlib
```

```
  roscpp
  rospy
  std_msgs
)
## System dependencies are found with CMake's conventions
find_package(Boost REQUIRED COMPONENTS system)


## Generate messages in the 'msg' folder
add_message_files(
  FILES
  ArmJointState.msg
)
## Generate added messages and services with any dependencies listed here
generate_messages(
  DEPENDENCIES
  sensor_msgs
  std_msgs
)
################################
## catkin specific configuration ##
################################
catkin_package(
  CATKIN_DEPENDS
    moveit_core
```

```
    moveit_ros_planning_interface
    interactive_markers
  DEPENDS
    EIGEN3
)
############
## Build ##
############
## Specify additional locations of header files
## Your package locations should be listed before other locations
include_directories(SYSTEM ${Boost_INCLUDE_DIR} ${EIGEN3_INCLUDE_DIRS})
include_directories(${catkin_INCLUDE_DIRS})
link_directories(${catkin_LIBRARY_DIRS})

## Declare a C++ library
add_executable(bcn3d_convert src/bcn3d_convert.cpp)

add_dependencies(bcn3d_convert bcn3d_moveit_generate_messages_cpp)

## Specify libraries to link a library or executable target against

target_link_libraries(bcn3d_convert
  ${catkin_LIBRARIES}
)
```

Prilog 3. *Arduino*-v kod

```
#if (ARDUINO >= 100)
 #include <Arduino.h>
#else
 #include <WProgram.h>
#endif
#include <ros.h>
#include <bcn3d_moveit/ArmJointState.h>
#include <Servo.h>
#include <std_msgs/Bool.h>
#include <std_msgs/String.h>
#include <math.h>
#include <std_msgs/Int16.h>
#include <std_msgs/UInt16.h>
#include <AccelStepper.h>
#include <MultiStepper.h>

// Joint 1
#define E1_STEP_PIN         36
#define E1_DIR_PIN          34
#define E1_ENABLE_PIN       30

// Joint 2
```

```
#define Z_STEP_PIN         46
#define Z_DIR_PIN          48
#define Z_ENABLE_PIN       62
#define Z_MIN_PIN          18
#define Z_MAX_PIN          19

// Joint 3
#define Y_STEP_PIN         60
#define Y_DIR_PIN          61
#define Y_ENABLE_PIN       56
#define Y_MIN_PIN          14
#define Y_MAX_PIN          15

// Joint 4
#define X_STEP_PIN         54
#define X_DIR_PIN          55
#define X_ENABLE_PIN       38

// Joint 5
#define E0_STEP_PIN        26
#define E0_DIR_PIN         28
#define E0_ENABLE_PIN      24
AccelStepper joint1(1,E1_STEP_PIN, E1_DIR_PIN);
AccelStepper joint2(1,Z_STEP_PIN, Z_DIR_PIN);
```

```
AccelStepper joint3(1,Y_STEP_PIN, Y_DIR_PIN);

AccelStepper joint4(1,X_STEP_PIN, X_DIR_PIN);

AccelStepper joint5(1, E0_STEP_PIN, E0_DIR_PIN);


Servo gripper;

MultiStepper steppers;


int joint_step[6];

int joint_status = 0;


ros::NodeHandle nh;

std_msgs::Int16 msg;


void arm_cb(const bcn3d_moveit::ArmJointState& arm_steps){

  joint_status = 1;

  joint_step[0] = arm_steps.position1;

  joint_step[1] = arm_steps.position2;

  joint_step[2] = arm_steps.position3;

  joint_step[3] = arm_steps.position4;

  joint_step[4] = arm_steps.position5;

  joint_step[5] = arm_steps.position6;

}


void gripper_cb( const std_msgs::UInt16& cmd_msg){
```

```
gripper.write(cmd_msg.data);
digitalWrite(13, HIGH-digitalRead(13));
}
ros::Subscriber<bcn3d_moveit::ArmJointState> arm_sub("joint_steps",arm_cb);
ros::Subscriber<std_msgs::UInt16> gripper_sub("gripper_angle", gripper_cb);
void setup() {
  pinMode(13,OUTPUT);
  joint_status = 1;

  nh.initNode();
  nh.subscribe(arm_sub);
  nh.subscribe(gripper_sub);

  joint1.setMaxSpeed(1500);
  joint2.setMaxSpeed(750);
  joint3.setMaxSpeed(2000);
  joint4.setMaxSpeed(500);
  joint5.setMaxSpeed(1000);

  steppers.addStepper(joint1);
  steppers.addStepper(joint2);
  steppers.addStepper(joint3);
  steppers.addStepper(joint4);
  steppers.addStepper(joint5);
```

```
  gripper.attach(11);
  digitalWrite(13, 1);
}

void loop() {
  if (joint_status == 1)
  {
    long positions[5];
    positions[0] = joint_step[0];
    positions[1] = -joint_step[1];
    positions[2] = joint_step[2];
    positions[3] = joint_step[3];
    positions[4] = -joint_step[4];

    steppers.moveTo(positions);
    nh.spinOnce();
    steppers.runSpeedToPosition();
    gripper.write(joint_step[5]);
  }
  digitalWrite(13, HIGH-digitalRead(13));
  joint_status = 0;
   nh.spinOnce();
  delay(0);
}
```