Integracija robotskih senzorskih sustava za interakciju sa složenim površinama

Androšić, Albert

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

ALBERT ANDROŠIĆ

Zagreb, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET STROJARSTVA I BRODOGRADNJE

INTEGRACIJA ROBOTSKIH SENZORSKIH SUSTAVA ZA INTERAKCIJU SA SLOŽENIM POVRŠINAMA

Mentor:

Doc. dr. sc. Filip Šuligoj, mag. ing. mech.

Student:

Albert Androšić

Zagreb, 2024.

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

Zahvaljujem mentoru doc. dr. sc. Filipu Šuligoju i asistentu mag. ing. Branimiru Ćaranu na pruženoj prilici, ukazanom vremenu i mnogim savjetima prilikom izrade ovog rada pod njihovim mentorstvom.

Također se zahvaljujem cijeloj ekipi CRTA-e na pruženoj pomoći i savjetima tijekom izrade rada.

Ovom prilikom zahvalio bih se svojim roditeljima, Sanji i Zdenku, i cijeloj obitelji na strpljenju, bodrenju i podršci bez kojih ne bih uspješno završio studij. Hvala svim prijateljima i kolegama za podršku u danima odmora od učenja i brojne uspomene koje su učinile ove studentske dane, mjesece i godine još boljima. I za kraj, želim zahvaliti svojoj djevojci Katarini na velikoj podršci, ljubavi i razumijevanju tijekom cijelog studija.

Albert Androšić

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske ispite studija strojarstva za smjerove:
Proizvodno inženjerstvo, inženjerstvo materijala, industrijsko inženjerstvo i menadžment, mehatronika i robotika, autonomni sustavi i računalna inteligencija

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 - 04 / 24 - 06 / 1**	
Ur.broj: **15 - 24 -**	

DIPLOMSKI ZADATAK

Student: **Albert Androšić** JMBAG: 0035214819

Naslov rada na hrvatskom jeziku: **Integracija robotskih senzorskih sustava za interakciju sa složenim površinama**

Naslov rada na engleskom jeziku: **Integration of robotic sensory systems for interaction with complex surfaces**

Opis zadatka:

Funkcionalnost kontrole sile postaje sve značajnija u modernoj robotici, gdje se od robota očekuje da uz sposobnost vizualne percepcije okruženja, ostvari i fizičku interakciju s istim. Ovo uključuje sposobnost robota da interpretira i reagira na različite fizičke parametre, kao što su sila i moment, u realnom vremenu. Takva vrsta interakcije omogućuje robotima da se adekvatno prilagode složenim i dinamičnim okruženjima, što je ključno za buduće primjene u industriji, zdravstvu i drugim sektorima. Za ovu svrhu u radu se predlaže koristiti ROS2 (eng., Robot Operating System 2), koji omogućuje integraciju i kontrolu različite opreme i njihove funkcionalnosti. Kao hardverska komponenta koristi se Franka Panda robot u kombinaciji s Realsense D435 dubinskom kamerom.

Zadaci istraživanja uključuju:

1. Dizajniranje prilagodljivog robotskog alata (dubinska kamera u ruci i sferni krajnji alat) koji se može montirati na postojeću prirubnicu, i zakrivljeni radni komad.

2. Upotreba ROS2 za integraciju funkcionalnosti i kontrolu robota te akviziciju oblaka točaka.

3. Implementacija kontrole sile kako bi se održavao konstantan kontakt sa zakrivljenom površinom.

4. Postavljanje scenarija u kojem robot pomiče alat kako bi pokrio put, planiran na temelju oblaka točaka dobivenog dubinskom kamerom, preko zakrivljene površine održavajući konstantnu silu.

5. Analiza rezultata, posebno točnosti pri održavanju konstantne sile i planirane putanje.

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

Zadatak zadan:	Datum predaje rada:	Predviđeni datumi obrane:
16. studenoga 2023.	18. siječnja 2024.	22. – 26. siječnja 2024.

Zadatak zadao:

Doc.dr.sc. Filip Šuligoj

Predsjednik Povjerenstva:

Prof. dr. sc. Ivica Garašić

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS DIJAGRAMA

POPIS PRILOGA

SAŽETAK

Cilj ovog diplomskog rada je integracija robota Franke Emike u ROS2 operativni sustav i njegovo osposobljavanje za praćenje zakrivljene 3D površine pomoću ugrađenih senzora sile i momenta. U radu je naveden postupak instalacije ROS2 operativnog sustava kao i pripadajućih podsustava za uspješno spajanje robota s računalom. Detaljno su opisani postupci integracije robota Franka Emika u ROS2 okruženje, prilagodba postojećih ROS2 paketa poput MoveIt2 te njihova prilagodba potrebama ovog istraživanja. Za snimanje radne okoline korištena je Intel RealSense D435i kamera, a njezine fizičke karakteristike, zajedno s instalacijom potrebnih biblioteka i alata za upotrebu u ROS2 sustavu te postupak „eye-in-hand“ kalibracije za što precizniju vizualizaciju prostora detaljno su opisani. Navedena „eye-in-hand“ kamera upotrijebljena je za generiranje 3D slike pomoću oblaka točaka pomoću kojeg korisnik u ROS2 okruženju odabire pozicijske markere kao ciljne točke robotske ruke. Što se konstrukcijskog aspekta ovog rada tiče, u radu su dokumentirani modeli nosača dubinske kamere za „eye-in-hand“ konfiguraciju i ticala sa sfernim vrhom, koji se može montirati na prihvatnicu i uhvatiti pomoću pomičnih prstiju. Modeli su konstruirani pomoću 3D pisača i montirani na prirubnicu, odnosno prihvatnicu. Radni komad korišten za testiranje ovog sustava nije dizajniran za potrebe ovog rada, već je to testni uzorak laboratorija CRTA-e koji se koristi u nekoliko projekata.

Temelj ovog rada leži u izradi kontrolera koji omogućuje precizno praćenje zakrivljene površine te održavanje konstantnog kontakta između radnog alata i površine regulacijom sile i održavanjem njezine vrijednosti unutar zadanih granica. Sedam momentnih senzora raspoređenih u svim zglobovima robota koriste se za estimaciju sile u radnom alatu, koja se potom regulira putem diskretnog pomaka robota duž zadane putanje. Putanja od točke do točke definirana je pomoću MoveIt2 paketa s ciljem optimizacije procesa. Provedena je eksperimentalna validacija točnosti pozicioniranja radnog alata korištenjem kalibracijske ploče, koja se koristi i za kalibraciju kamere te validacija praćenja 3D zakrivljene površine pomoću diskretnog kontrolera sile i kombinacije triju ili četiriju točaka.

Ključne riječi: Franka Emika, ROS2, oblak točaka, MoveIt2, konstantno održavanje sile, praćenje oblika površine

SUMMARY

The goal of this master's thesis is the integration of Franka Emika robot in the ROS2 operating system and its adaptation for tracking a curved 3D surface tracking using built-in force and torque sensors. The thesis outlines the process of installing ROS2 operating system and its subsystems for successfully connecting the robot to the computer. The procedures for integrating Franka Emika robot into the ROS2 environment, adapting existing ROS2 packages such as MoveIt2, and customizing them to suit the needs of this research are also described in detail.

For capturing the working environment, an Intel RealSense D435i camera was utilized. Its physical characteristics, along with the installation of necessary libraries and tools for use in the ROS2 system, as well as the "*eye-in-hand"* calibration process for precise spatial visualization are thoroughly explained. The mentioned *"eye-in-hand"* camera was used to generate a 3D image through point cloud data, allowing the user in the ROS2 environment to select positional markers as target points for the robotic arm. Regarding the construction aspect of this paper, the thesis documents the models of the depth camera mount for the "eye-in-hand" configuration and the tactile tool with a spherical tip, which can be mounted on a robot gripper and gripped by movable fingers. These models were constructed using a 3D printer and mounted on the flange and the gripper. The workpiece used for testing this system was not designed specifically for this thesis but is a sample from the CRTA laboratory used in several projects.

The foundation of this research lies in the development of a controller enabling precise tracking of the curved surface and maintaining constant contact between the tool and the surface by regulating force and keeping its value within specified limits. Seven torque sensors distributed across all robot joints are employed to estimate the force at the tool, which is then regulated by discrete robot movements along the defined path. The point-to-point path is defined using MoveIt2 package to optimize the process. An experimental validation of the accuracy of tool positioning was conducted using a calibration board, which is also used for camera calibration. Also, the tracking of the 3D curved surface using the discrete force controller was validated using four and three point combinations.

Key words: Franka Emika robot, Pointcloud, ROS2, MoveIt2, constant force maintenance, following the shape of the surface

1. Uvod

Proteklih godina robotika je sve značajnije integrirana u svakodnevni život, dok se u industriji već dugi niz godina koristi na visokoj razini. Primjena robotike proširila se na različite sektore pa se tako može naći u prehrambenoj, metalurškoj, energetskoj, automobilskoj industriji itd. Na primjer, u automobilskoj industriji, grupa robota na proizvodnoj traci međusobno surađuje kako bi izvršila montažu automobila, bilo da se radi o kompletnom vozilu ili samo jednoj od njegovih komponenti. Ovaj napredni sustav omogućuje učinkovitu i preciznu proizvodnju, što dodatno ilustrira utjecaj robotike na različite aspekte naše svakodnevice.

Slika 1. Prikaz robota na proizvodnoj traci koji zajedno slažu automobil

Roboti i robotske ruke, kao kompleksni sustavi, predstavljaju inovativno tehničko dostignuće koje kombinira mehatroniku, računalno programiranje i senzorske tehnologije. Ovi autonomni proizvodi tehnologije često su oblikovani prema ljudskom ili životinjskom uzoru s sposobnostima izvršavanja raznovrsnih zadataka ovisno o njihovom programiranju i konstrukciji. Tijelo robota sastoji se od mehaničkih dijelova, senzora, aktuatora i energetskih izvora, koristeći fleksibilne i napredne materijale radi prilagodljivosti. Različiti oblici mobilnosti, poput kotača, nogu ili krila, prilagođavaju se specifičnim primjenama. Središnji sustav, odnosno „mozak“, robota predstavlja njegov računalni sustav koji interpretira informacije dobivene od senzora te upravlja aktuatorima. Opremljen procesorom i memorijom, to računalo izvršava složene algoritme koji omogućuju inteligentno ponašanje robota. Napredne umjetne inteligencije, poput neuronskih mreža, često se koriste za poboljšanje sposobnosti učenja i prilagodbe. Roboti se programiraju kako bi izvršavali

specifične zadatke te mogu biti programirani prema unaprijed definiranim skriptama ili putem učenja iz iskustava (strojno učenje). Programer određuje kako će robot reagirati na različite situacije i zadatke, omogućujući mu prilagodljivost i autonomiju u radu. U cjelini, roboti predstavljaju dinamičnu domenu tehnologije koja kontinuirano evoluira, proširujući granice mogućnosti u mnogim sektorima našeg društva. Njihova sveprisutnost i sve sofisticiranije funkcionalnosti čine ih ključnim akterima u suvremenom tehničkom napretku.

Još jedan od sektora gdje se robotika sve više primjenjuje je medicinska robotika gdje su preciznost i ponovljivost od iznimne važnosti. Nije više dovoljno samo imati robotsku ruku za obavljanje zadataka pa se uvode razni dodatni senzori kako bi robot pružao optimalne rezultate. Neki od tih senzora su senzori dodira i sile, koji su sada već integrirani u većinu robota, 2D i 3D kamere visoke rezolucije, termalne kamere, zvučni senzori, senzori udaljenosti i blizine kao što su ultrazvučni, infracrveni i LIDAR senzori. Također, koriste se biološki senzori te senzori za mjerenje vlage i tlaka, ovisno o specifičnoj primjeni. Integracija ovih senzora omogućuje robotima da prikupe raznovrsne informacije o okolini, poboljšavajući tako njihovu sposobnost izvršavanja zadataka.

Jedan od najčešćih senzora koji se danas koriste u gotovo svim sektorima su 2D i 3D kamere. 2D kamere snimaju dvodimenzionalne slike, pružajući informacije o širini i visini objekta u vidnom polju. Ove kamere često se koriste za prepoznavanje oblika, praćenje pokreta i analizu boja. S druge strane, 3D kamere omogućuju snimanje trodimenzionalnih informacija, pružajući podatke o dubini scene. Ove kamere su izuzetno korisne u prostornom prepoznavanju, praćenju udaljenosti objekata te stvaranju trodimenzionalnih mapa okoline. Oba tipa kamera imaju široku primjenu, uključujući industrijsku automatizaciju, medicinsku dijagnostiku, sigurnosne sustave itd.

Senzor sile predstavlja ključnu komponentu u mnogim aplikacijama, a u kontekstu ovog rada koristit će se kako bi robot precizno pratio radnu površinu održavajući stalni kontakt s njom. Ova tehnološka inovacija omogućuje robotu osjet i reakciju na promjene u sili koje djeluju na određenoj točki ili komponenti. Osnovni princip rada senzora sile temelji se na promjeni nekog fizikalnog parametra pod utjecajem sile, a zatim pretvaranju te promjene u električni signal koji se može kvantificirati. Postoje različiti tipovi senzora sile, uključujući optičke, piezoelektrične, elektromagnetske i mnoge druge.

U drugom poglavlju ovog diplomskog rada predstavljena je robotska ruka Franka Emika, koja će biti ključna za sva istraživanja provedena u sklopu ovog rada. Detaljno su opisane osnovne postavke robotske ruke, istaknuta su njezina ograničenja te je objašnjen način spajanja s ROS2 operativnim sustavom. Proučena su ključna svojstva ove robotske ruke koji pruža temeljni uvid u tehničke aspekte korištenog hardvera.

U trećem poglavlju opisana je Intel RealSense D435i kamera korištena za snalaženje robota u prostoru, odnosno za robotsku navigaciju u radnom okruženju. Ova sekcija pruža dublji uvid

u karakteristike kamere, kalibracijsku metodu kojom je kamera smještena u okolinu robota te način postupak kojim je obavljena intrinzična i ekstrinzična kalibracija pomoću ViSP vodiča.

Četvrto poglavlje prikazuje i opisuje proces izrade i modeliranja 3D modela koji su ključni za izvođenje svih radnji, a izrađeni su pomoću Prusa 3D printera. Prvi model je „ticalo“, radni alat robota koji služi za uspostavljanje kontakta između robota i radne površine. Drugi model sastoji se od dvije komponente koje omogućuju montiranje RealSense kamere na robotsku prirubnicu na način da omogući robotu nesmetan rad i slobodno kretanje. Sklop prirubnice, prihvatnice i ticala i nosača s kamerom su također prikazani i opisani u ovom poglavlju.

U petom poglavlju, detaljnije je objašnjen ROS 2 operativni sustav s fokusom na način rada i principima komunikacije između pojedinih čvorova i alata. Poseban naglasak stavljen je na MoveIt2 paket, ključni alat u ROS2 sustavu koji olakšava upravljanje kretanjem robota putem ROS2 okruženja. Ovaj paket integrira kontrolere kako bi omogućio preciznu kontrolu robota. Opisan je i postupak stvaranja ROS2 radnog područja te njegova primjena prilikom preuzimanja i instaliranja novih paketa ili izrade vlastitih. Detaljno je objašnjen modul kamere koji je bitan za stvaranje oblaka točaka (eng. "Pointcloud"), a također je opisano kako se taj modul poziva putem terminala te kako se rezultati prikazuju unutar ROS 2 radnog okruženja. Još jedna od stavki koja je analizirana u ovom poglavlju je čvor *„PointStamped“* koji generira markere za označavanje željenih ciljnih točaka robota.

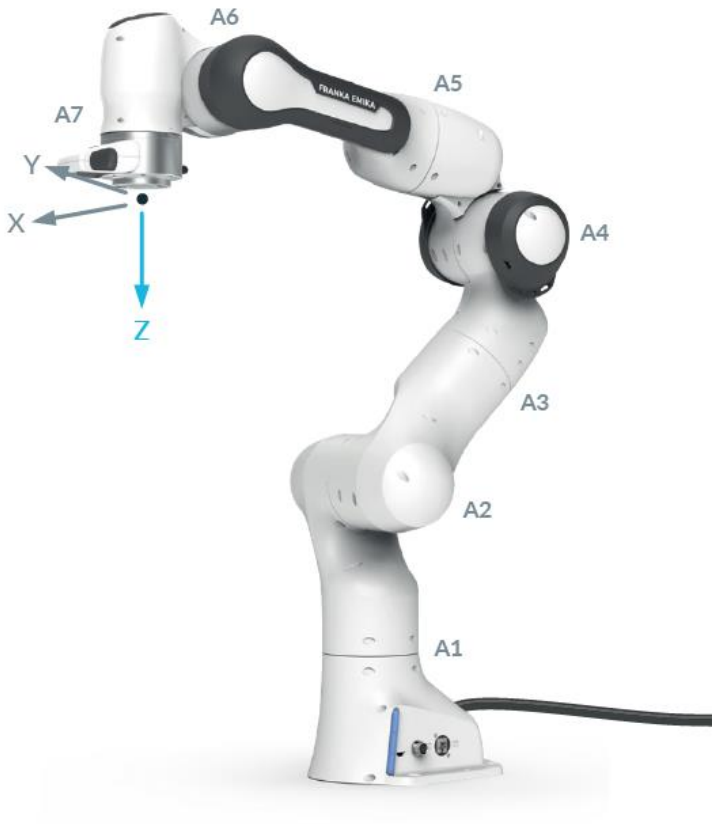
U šestom poglavlju prikazana je kompleksna radna stanica unutar *„rviz2“* okruženja, kako u simuliranom okruženju tako i u stvarnom prostoru. Detaljno je opisan postupak pozivanja robota i korištenje MoveIt2 paketa za upravljanje njegovim pokretima putem čvorova u „*rviz2“* i samostalnih Python skripti. Dodatno, u tom poglavlju objašnjen je proces očitavanja trenutnih sila koje djeluju na radni alat putem momentnih senzora smještenih na zglobovima robota. Prikazan je način kako se ove sile koriste za regulaciju i održavanje radnog alata u stalnom kontaktu s 3D površinom u cilju postizanja optimalne kontrole pokreta robota.

U sedmom poglavlju provedena je analiza rezultata s obzirom na točnost pozicioniranja radnog alata te točnost održavanja kontakta s površinom, održavanja konstante sile te točnosti praćenja krivulje u odnosu na početnu i krajnju točku.

Zaključak sažima dobivene rezultate tijekom nekoliko iteracija istraživanja i pruža osvrt na moguća proširenja rada. Ovo poglavlje predstavlja završni dio rada koji sumira ključne spoznaje i identificira smjerove za daljnje istraživanje u području robotske manipulacije i navigacije.

2. FRANKA Emika Panda robot

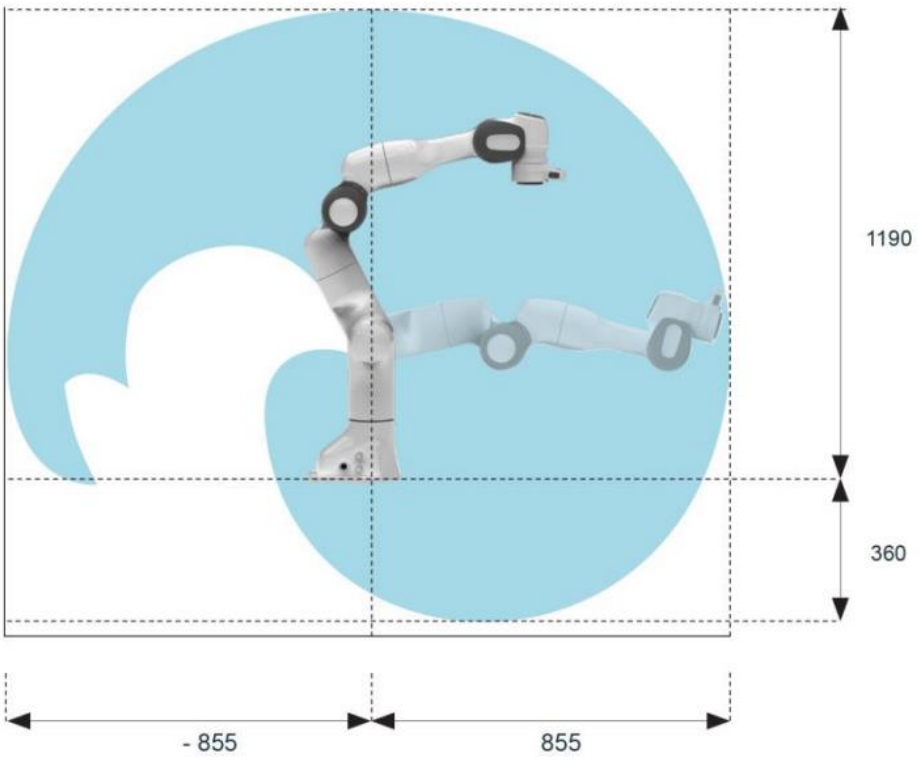
2.1 Mehaničke karakteristike



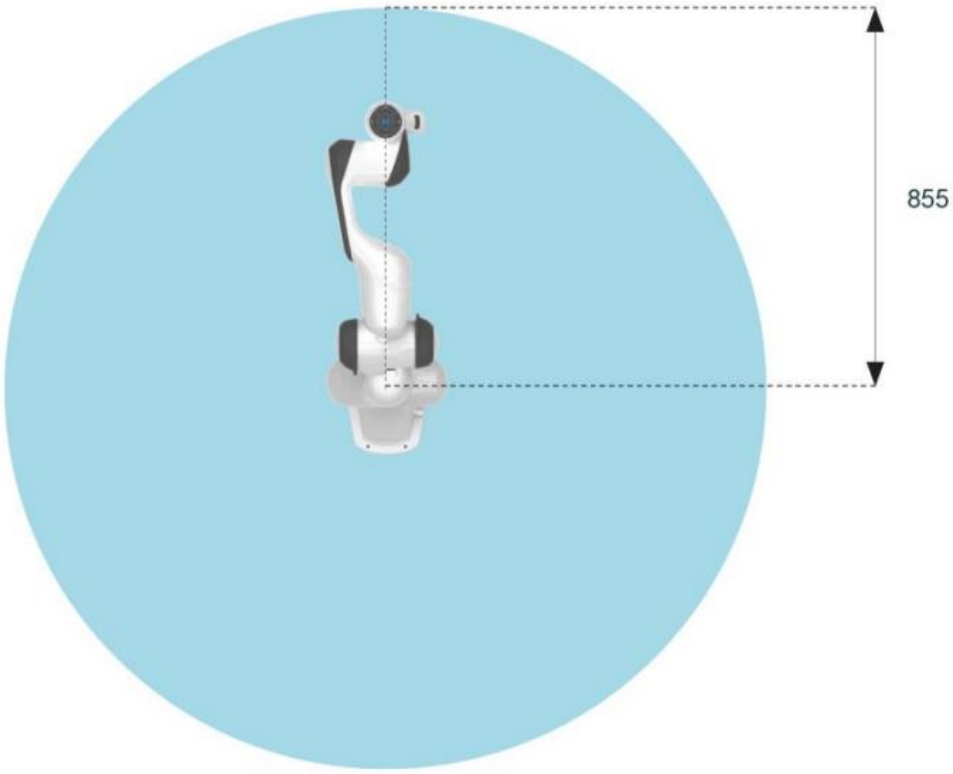
Slika 2. Franka Emika robotska ruka[16]

Franka Emika Panda je robotski manipulator sa sedam stupnjeva slobode koji se ističe po svojoj visokoj preciznosti i fleksibilnosti u različitim aplikacijama. Ovaj robot ima sposobnost izvođenja kompleksnih pokreta i zadataka zahvaljujući svojim sedam slobodnih stupnjeva koji mu omogućuju široki raspon kretanja. Panda je opremljena senzorima visoke preciznosti i naprednom kontrolom, što je čini pogodnom za rad u okruženju s ljudima i izvršavanje preciznih zadataka. Osim visoke preciznosti i fleksibilnosti, Franka Emika Panda ističe se i po svojoj jednostavnosti programiranja putem sučelja koje omogućuje intuitivno upravljanje. Korisnici mogu lako definirati zadatke i putanje korištenjem grafičkog sučelja, što čini ovog robota pristupačnim i za neiskusne korisnike u području robotike.

Dimenzije radnog okruženja robota, bez ruke "Franka Hand", u najudaljenijim položajima od ishodišta koordinatnog sustava postolja robota su 1190 mm u pozitivnom smjeru z osi i 360 mm u negativnom smjeru z osi. Dodatno, u horizontalnoj ravnini, odnosno u smjeru x i y osi, radni prostor obuhvaća kružnicu s polumjerom od 855 mm. Kako bi se lakše dočaralo, prikaz radnog okruženja može se vidjeti na priloženim slikama 3 i 4.



Slika 3. Bočni pogled na radni prostor robotske ruke[16]



Slika 4. Gornji pogled na radni prostor robotske ruke[16]

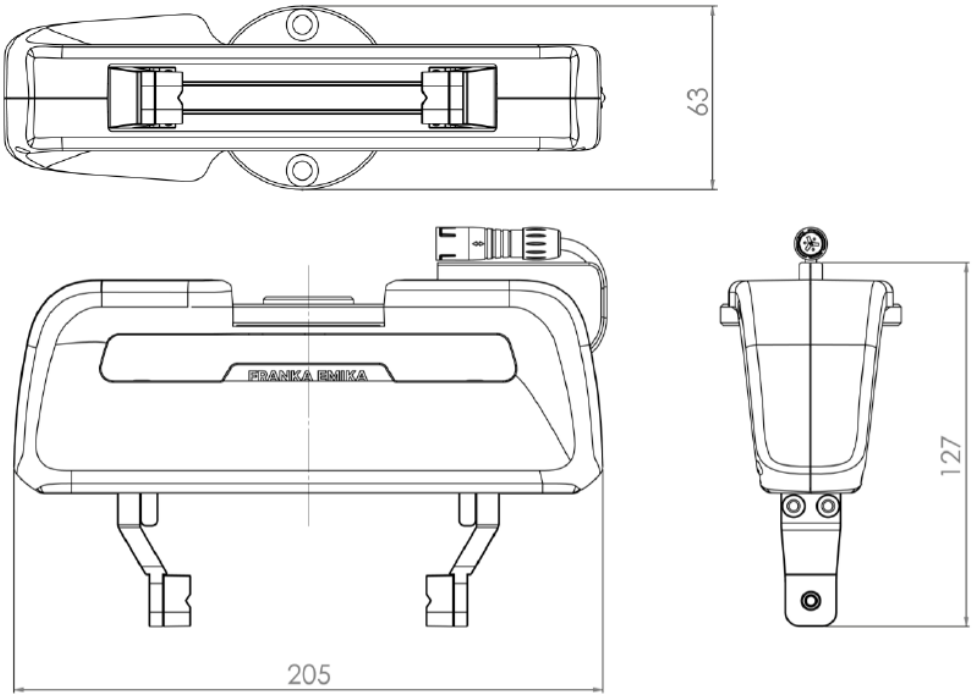
Neke od mehaničkih karakteristika ove robotske ruke prikazane su u sljedećoj tablici:

Tablica 1. Prikaz tehničkih karakteristika robota Franka Emika

Stupnjevi slobode	7 DOF
Težina	17.8 kg
Pokretna masa	12.8 kg
Teret	3 kg
Maksimalni doseg	855 mm
Ograničenja pozicija zglobova [°]	A1: -166/166, A2: -101/101, A3: -166/166, A4: -176/-4, A5: -166/166, A6: -1/215, A7: -166/166
Ograničenja brzina zglobova [°/s]	A1: 150, A2: 150, A3: 150, A4: 150, A5: 180, A6: 180, A7: 180
Ograničenje Kartezijske brzine	Brzina krajnjeg alata do 2 m/s
Ponovljivost	+/- 0.1 mm (ISO 9283)
Osjetljivost	Momentni senzori u svih 7 zglobova
Rezolucija sile	<0.05 N
Ponovljivost sile	0.15 N
Rezolucija momenta	0.02 Nm
Ponovljivost momenta	0.05 Nm

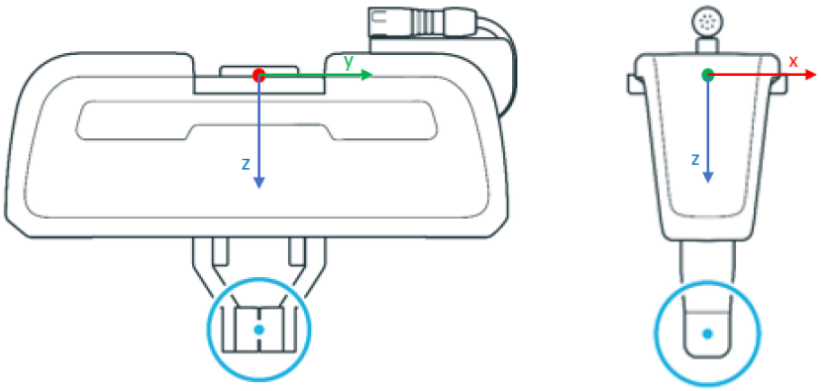
Kako se može primijetiti u Tablici 1, Franka Emika je opremljena momentnim senzorima u svih sedam zglobova s vrlo visokom točnošću što će biti jako korisno u kontekstu ovog rada.

Još jedna od komponenti Franka Emika robota je prihvatnica poznata kao „Franka Hand“ koja se montira na prirubnicu pomoći dva vijka dimenzija M6x12 (DIN7984) i jednog cilindričnog zatika dimenzija Ø6x10 h8 (ISO2338B). Gabaritne dimenzije, vidljive na slici 5, iznose 63x205x127, a težina je 730 g. Ova prihvatnica ima silu hvatanja od 30 do 70 N koju može kontinuirano održavati, dok joj je maksimalna sila hvatanja 140 N. Krajnji koordinatni sustav prihvatnice se nalazi na sredini mjesta hvatanja njenih prstiju, što je prikazano na slici 6, zajedno s njenom matricom transformacije.



Slika 5. Gabaritne dimenzije prihvatnice "Franka Hand"[17]

$$F^{T^{TCP}} = \begin{bmatrix} R_{xx} & R_{xy} & R_{xz} & p_x \\ R_{yx} & R_{yy} & R_{yz} & p_y \\ R_{zx} & R_{zy} & R_{zz} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.707 & 0.707 & 0 & 0 \\ -0.707 & 0.707 & 0 & 0 \\ 0 & 0 & 1 & 0.1034 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

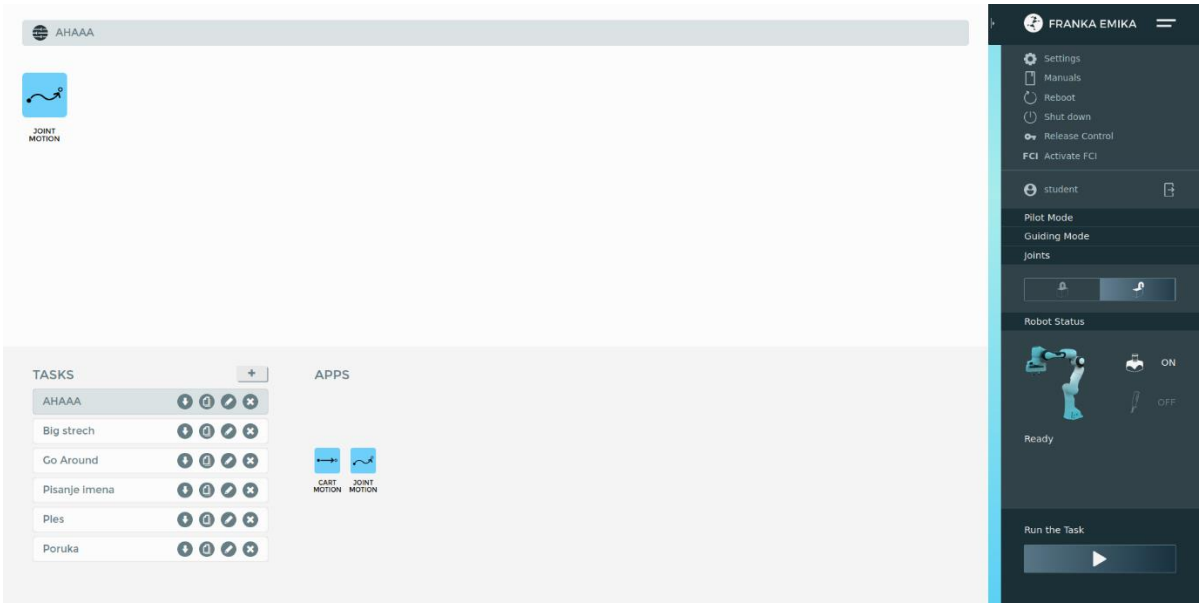


Slika 6. Točka hvatanja prihvatnice "Franka Hand"[17]

2.2 Franka Desk i korištenje FCI-a za spajanje u ROS2

Povezivanje Franke i računala moguće je preko LAN veze na način da se PC radna stanica spaja s kontrolnom kutijom Franke, a ne s bazom robota. Nužno je povezati računalo i robota Franku preko kontrolne kutije zbog mogućnosti bržeg prijenosa podataka (1000 Mb/s), a time i preciznije radnje s njim. Kako bi ostvarili još bolju vezu računala i robota preporuča se korištenje „real-time“ kernela[2] koji omogućuje komunikaciju s robotom u realnom vremenu. U ovom radu korištena je verzija „6.2.1-rt3“ kernela za Ubuntu 22.04 koji se prilikom paljenja računala izabire u osnovnom ulazno/izlaznom sustavu (BIOS). Real-time kernel se koristi za bržu i sigurniju komunikaciju s robotom i njime se osigurava dobivanje vjernijih podataka o poziciji robota, silama i momentima koji se razvijaju u motorima te sličnim parametrima.

Uspješnim povezivanjem Franke, preko jednog od internetskih poslužitelja možemo se spojiti s robotskim sučeljem „Franka Desk“, gdje je omogućeno njegovo programiranje. Programiranje Franke u Desk-u ostvaruje se preko opcija u polju „APPS“, vidljivom na slici 7. Povlačenjem ikona u radni prostor robotu se mogu zadavati radnje i pripadajuće postavke. Na desnoj strani sučelja prikazana je slika robota s indikacijom njegova stanja - bijela boja označava zaustavljeni robot, dok plava boja označava da je spreman za rad. Iznad slike robota, nalazi se izbornik za otključavanje i zaključavanje zglobova te izbornici za „Pilot Mode“ i „Guiding Mode“. "Pilot Mode" omogućuje upravljanje prstima "Franka Hand" prihvatnice putem gumba na samom robotu ili putem računala. S druge strane, "Guiding Mode" omogućuje premještanje robota korištenjem "Dead Switcha", tj. pritiskom na dva gumba na posljednjem zglobu robota. Ovaj način rada omogućuje odabir translacijskog, rotacijskog ili kombiniranog gibanja u bilo kojoj osi.

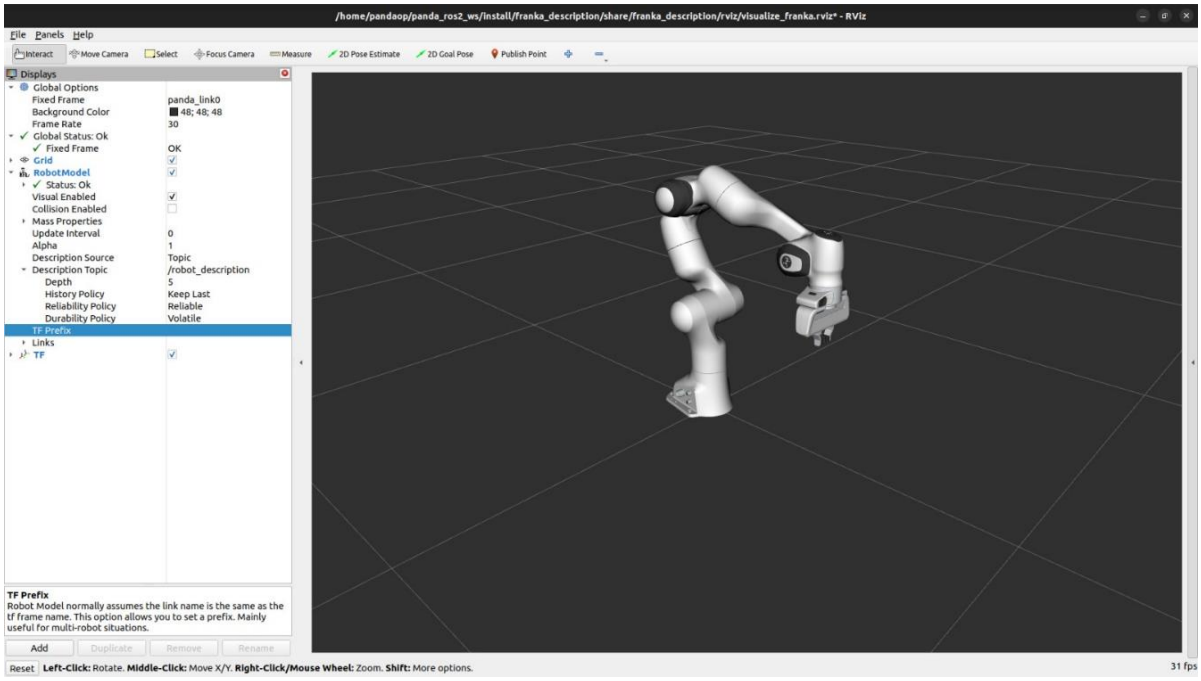


Slika 7. Izgled Franka sučelja za upravljanje robotom

Iako Franka Emika ima svoje vlastito korisničko sučelje, za potrebe ovog istraživanja koristit će se ROS2 operativni sustav. Odabir ROS2 sustava sve više dobiva na popularnosti i primjenjuje se u različitim aplikacijama, stoga je cilj dokazati njegovu primjenjivost i u ovakvim istraživanjima. Integracija Franke u ROS2 olakšat će primjenu ovog robota u daljnjim projektima i istraživanjima. Da bi robot bio sposoban raditi preko vanjskih sustava, potrebno je aktivirati FCI (Franka Control Interface). FCI omogućuje komunikaciju i kontrolu robota putem vanjskih programa ili sustava, čime se pruža fleksibilnost i proširuje spektar mogućnosti upotrebe robota u različitim okruženjima.

Aktiviranje FCI onemogućuje rad u sučelju Franke „Desk"-u te daje drugim sustavima, poput ROS2, mogućnost da preuzmu kontrolu nad Franka robotom i integriraju ga u šire sustave automatizacije, istraživanja ili industrijske primjene. Ova integracija pruža korisnicima mogućnost učinkovitog upravljanja robotom kroz standardizirane komunikacijske protokole, čime se pojednostavljuje razvoj i implementacija naprednih sustava koji koriste Franka Emika robote u svojim aplikacijama.

Kako bi se preko ROS2 sustava moglo upravljati robotom Franka Emika potrebno je preuzeti biblioteku *„libfranka"*[1] i pakete u ROS2 koji služe za upravljanje robotima[7]. Nakon preuzimanja, pakete je potrebno instalirati pomoću naredbe *"colcon build"*. Uspješnom instalacijom mogu se pokrenuti „launch" datoteke i Python skripte kao ROS2 čvorove, koji omogućuju upravljanje robotom izvršavanjem jednostavnih zadataka. Pokretanjem čvora *„franka.launch.py robot_ip:=192.168.40.45"* unutar *„franka_bringup"* direktorija, robot Franka s IP adresom 192.168.40.45 će biti prikazan u *„rviz2"*. Više informacija o ROS2 sustavu, postavljanju radnog okruženja i instalaciji paketa bit će raspravljeno u 5. poglavlju.



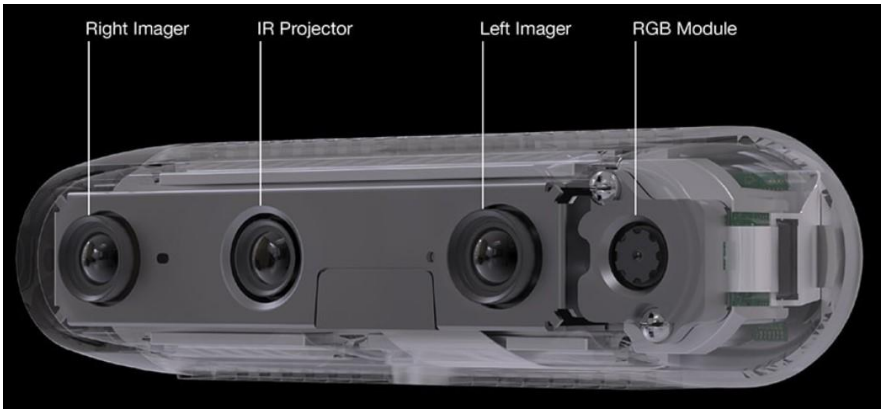
Slika 8. Prikaz Franka Emika robota u *„rviz2"*

3. Intel RealSense D435i kamera

3.1 Tehničke karakteristike kamere

Kamere Intel RealSense serije D400 predstavljaju stereovizijski dubinski sustav s uključenim stereo dubinskim modulom i vizijskim procesorom. Ove kamere omogućuju povezivanje s glavnim procesorom ili računalom putem USB 2.0/USB 3.1 Gen1 ili MIPI konekcije. Zbog svojih kompaktnih dimenzija i jednostavnosti integracije, ove kamere pružaju inženjerima fleksibilnost pri implementaciji u različite proizvode.

Slika 9. Intel RealSense D43i kamera[19]



Slika 10. Leće kamere[19]

Na slikama 9 i 10 prikazana je jedna od kamera iz serije D400, konkretno D435i kameru koja će biti upotrijebljena u okviru ovog istraživanja. Ova kamera predstavlja napredni sustav dubinskog viđenja koji nudi mogućnost preciznog mjerenja dubine u stvarnom vremenu. Osim stereo dubinskog modula, D435i također uključuje dodatni set senzora, uključujući

akcelerometar i žiroskop, što omogućuje dodatne informacije o orijentaciji i pokretima kamere. Ovaj model kamere posebno je koristan u robotici i računalnom vidu jer pruža ne samo informacije o dubini scene već i o promjenama u položaju kamere tijekom vremena. Ova kombinacija dubinskog viđenja i senzora pokreta čini D435i snažnim alatom za različite primjene, uključujući robotiku, stvarnost proširenu (AR), virtualnu stvarnost (VR), praćenje pokreta i druge scenarije gdje precizno mjerenje dubine i orijentacije igraju ključnu ulogu. U okviru ovog istraživanja, bit će postavljena na robota oko prirubnice pomoću nosača koji je definiran u četvrtom poglavlju. Dimenzije ove kamere su 90x25x25 mm, a težina iznosi 72 grama. Dodatne tehničke karakteristike, a koje su značajne, navedene su u Tablici 2.

Tablica 2. Tehničke karakteristike kamere D435i

Referentna osnova	55 mm
Dubinski FOV HD [°]	H: 87±3 / V: 58±1 / D: 95±3
Dubinski FOV VGA [°]	H: 75±3 / V: 62±1 / D: 89±3
IR projektor FOV [°]	H: 90 / V: 63 / D: 99
FOV senzora boje	H: 69±3 / V: 42±1 / D: 77±1
IMU	6 DoF

3.2 Kalibracija kamere

Nakon montiranja kamere na robota, nužno je provesti kalibraciju kako bi kamera pružala precizne informacije o poziciji objekata u okolini robota. Za ovu kalibraciju korišten je vodič "Visual Servoing Platform". ViSP vodič se može instalirati praćenjem uputa s linka[3]. ViSP vodičem omogućeno je definiranje unutarnjih postavki (intrinzičnih) kamere te matrice transformacije iz alata robota (TCP - panda_hand_tcp) do koordinatnog sustava kamere (camera_link). Drugim riječima, kroz korištenje kalibracijske "šahovske" ploče, postavljena je transformacija između baze robota (panda_link0) i kamere. Kalibracija kamere se sastoji od kalibracije intrinzičnih i ekstrinzičnih karakteristika.

Intrinzična kalibracija kamere procjenjuje parametre koji omogućuju izračun odnosa između mjernih jedinica kamere (pozicije piksela na slici) piksela i mjernih jedinica u prostoru (normalizirane pozicije u metrima u području slike). Za potrebe ove kalibracije parametri *u* i *v* prikazivat će poziciju piksela u na digitaliziranoj slici, koji će biti povezani s koordinatama *x* i *y* u prostoru.

ViSP kalibracija za izračun intrinzičnih parametara koristi ove jednadžbe za pretvorbu metara u piksele:

$$u = u_0 + x \cdot p_x \cdot (1 + k_{ud} * r^2) \tag{1}$$

$$v = v_0 + y \cdot p_y \cdot (1 + k_{ud} * r^2) \tag{2}$$

$$gdje\ je\text{:}\ r^2 = x^2 + y^2 \tag{3}$$

i ove jednadžbe za pretvorbu piksela u metre:

$$x = (u - u_0) \cdot \frac{1 + k_{du} \cdot r^2}{p_x} \tag{4}$$

$$y = (v - v_0) \cdot (1 + k_{du} \cdot \frac{r^2)}{p_y} \tag{5}$$

$$gdje\ je\text{:}\ r^2 = \left(\frac{u - u_{0)}}{p_x}\right)^2 + \left(\frac{v - v_0}{p_y}\right)^2 \tag{6}$$

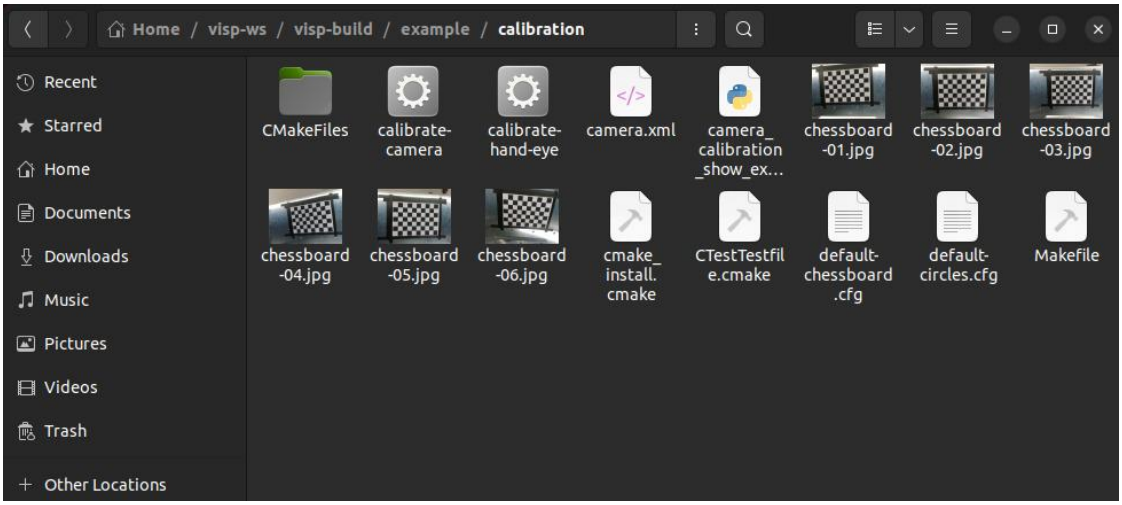
Ostali parametri predstavljaju:

u_0, v_0 – koordinate glavne točke u pikselima

p_x, p_y – odnos između fokalne dužine i veličine piksela

k_{ud}, k_{du} – parametri za korekciju distorzije, k_{ud} je distorzijski parametar za transformaciju nedistorziranih u distorzirane slike, dok je k_{du} distorzijski parametar za transformaciju koordinata iz distorziranih u nedistorzirane slike.

Prije početka intrinzične kalibracije treba ući u *„visp_ws“* radni direktorij gdje se nalaze svi potrebni programi za izvođenje kalibracije. Kod kalibracija kamere iznimno je važno da veličine crno-bijelih kvadratića odgovaraju veličini koja je zapisana u *„default-chessboard.cfg“* datoteci. Veličina kvadratića ploče koja je korištena ovdje iznosi 0.025 mm. Kada je sve spremno, naredbom „cd“ može se pristupiti direktoriju koji sadrži sve potrebne naredbe, a to je *„/visp/visp-build/example/calibration“* i započeti kalibraciju.



Slika 11. Radni direktorij intrinzične kalibracije

Prvi korak kalibracije je snimanje slika kalibracijske ploče iz najmanje 5 pozicija robota pomoću sljedeće naredbe u terminalu:

```
./tutorial-grabber-realsense --seqname chessboard-%02d.jpg --record 1
```

Slika 12. Naredba za prikupljanje slika za intrinzičnu kalibraciju[6]

Nakon dobivanja slika, u terminalu se pokreće sljedeća naredba,

```
./calibrate-camera default-chessboard.cfg
```

Slika 13. Naredba za izračun parametara[6]

koja će prikupljene slike proanalizirati i na temelju njih izračunati sve potrebne parametre i zapisati ih u dokument *„camera.xml“* koji onda možemo iskoristiti za ekstrinzičnu kalibraciju.

Vrijednosti intrinzične kalibracije ove Intel RealSense D435i kamere su bile:

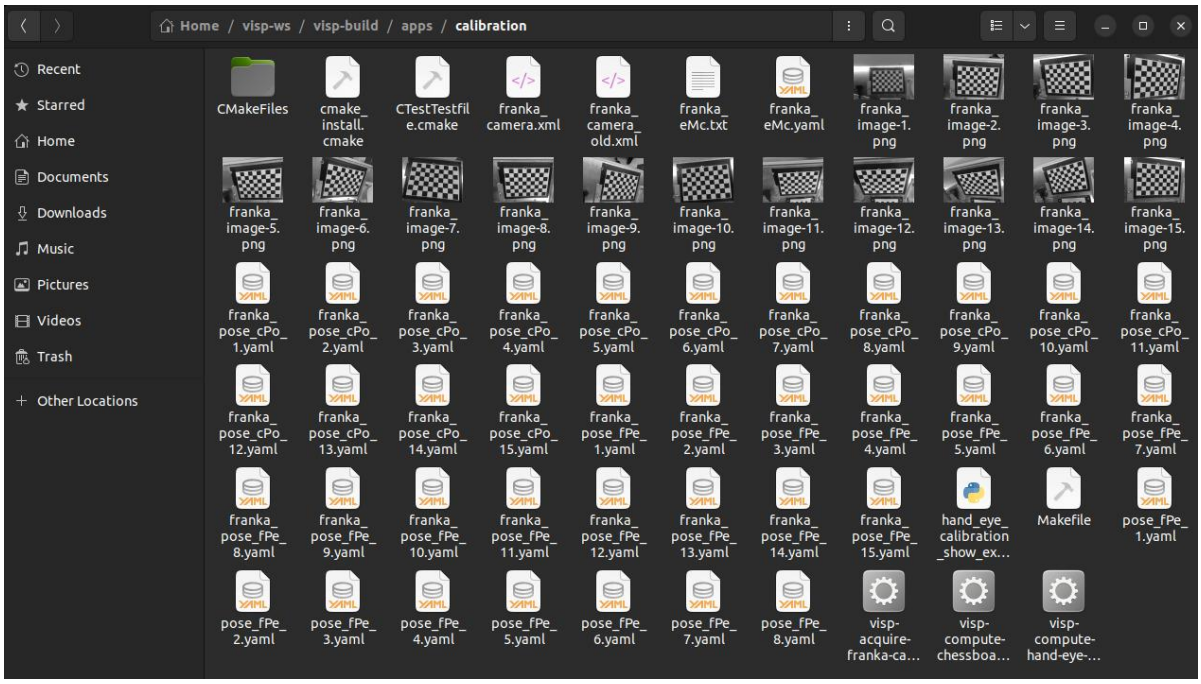
```
<!--Name of the camera-->
<name>Camera</name>
<!--Size of the image on which camera calibration was performed-->
<image_width>640</image_width>
<image_height>480</image_height>
<!--Intrinsic camera parameters computed for each projection model-->
<model>
  <!--Projection model type-->
  <type>perspectiveProjWithoutDistortion</type>
  <!--Pixel ratio-->
  <px>574.87231031670876</px>
  <py>576.13823277245751</py>
  <!--Principal point-->
  <u0>316.02662950777773</u0>
  <v0>242.22329361838487</v0>
</model>
<model>
  <!--Projection model type-->
  <type>perspectiveProjWithDistortion</type>
  <!--Pixel ratio-->
  <px>592.37601484123707</px>
  <py>593.36644511991585</py>
  <!--Principal point-->
  <u0>319.97801529161262</u0>
  <v0>245.10382251617989</v0>
  <!--Undistorted to distorted distortion parameter-->
  <kud>0.061754532983368957</kud>
  <!--Distorted to undistorted distortion parameter-->
  <kdu>-0.060638167330601261</kdu>
</model>
<!--Additional information-->
<additional_information>
  <date>2024/01/14 23:15:30</date>
  <nb_calibration_images>6</nb_calibration_images>
  <calibration_pattern_type>Chessboard</calibration_pattern_type>
  <board_size>9x6</board_size>
  <square_size>0.025</square_size>
  <global_reprojection_error>
    <without_distortion>0.239582</without_distortion>
    <with_distortion>0.154833</with_distortion>
```

Slika 14 Vrijednosti parametara iz „camera.xml“ datoteke

Kako je navedeno u vodiču, uzete su vrijednosti s distorzijom za ekstrinzičnu kalibraciju kako bi se postigla bolja preciznost kalibracije i poboljšalo pozicioniranje radnog alata na radnoj površini.

Ekstrinzična kalibracija kamere definira položaj koordinatnog sustava kamere u odnosu na bazu robota. U ovom slučaju, kamera je postavljena na prirubnicu robota, pa će se izvesti "eye-in-hand" kalibracija kamere[5] iz koordinatnog sustava prihvatnice do kamere. Izračunom te matrice transformacije, moći ćemo model kamere implementirati u cijeli robotski sustav.

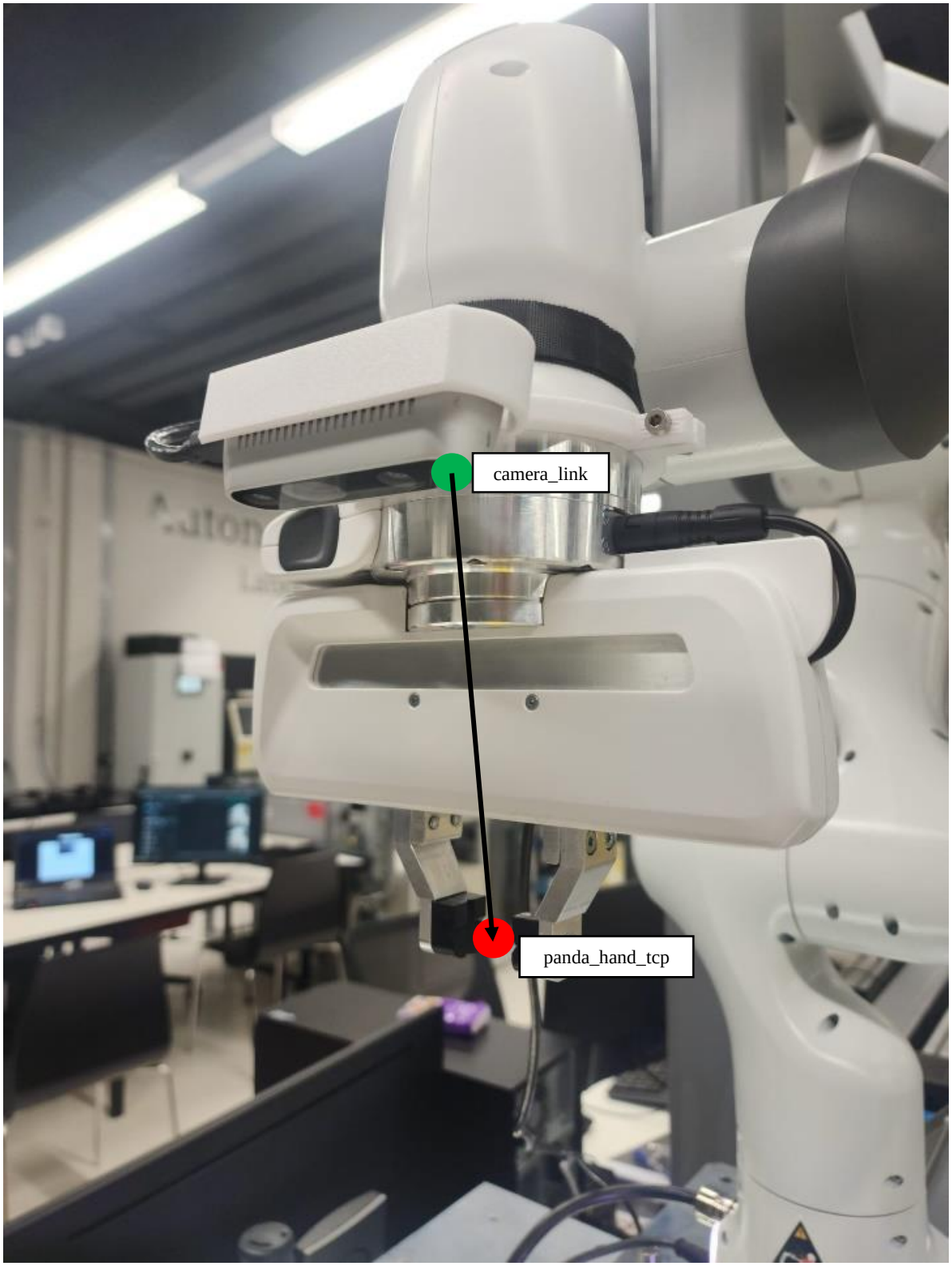
Kao i kod intrinzične kalibracije, prije početka same kalibracije potrebno je ući u radni direktorij koji sadrži sve potrebne naredbe i parametre. Za ekstrinzičnu kalibraciju taj direktorij je „/visp/visp-build/apps/calibration“.



Slika 15. Radni direktorij ekstrinzične kalibracije

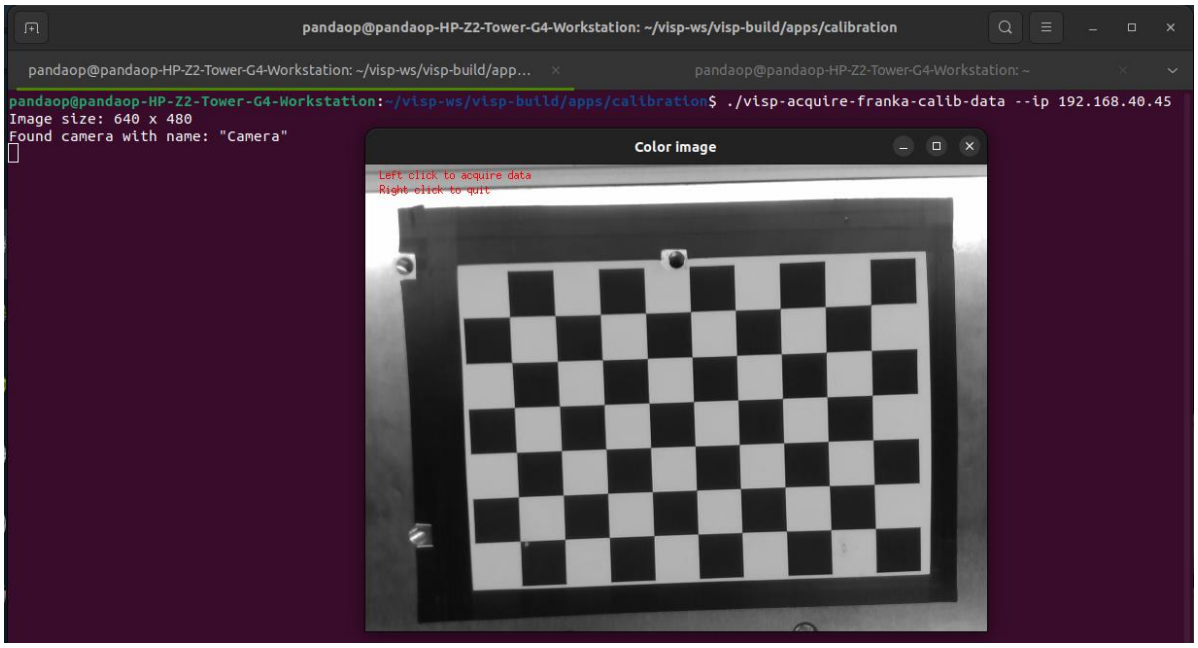
Ovom kalibracijom će se kompletirati cijeli kinematski lanac robota od baze do kamere i prihvatnice. Postupak uključuje pomicanje kamere, odnosno robota, u najmanje 10 različitih položaja, gdje je cijela šahovska ploča u potpunosti vidljiva. Ova pažljiva procedura omogućuje detaljnu kalibraciju i precizno definiranje položaja kamere u odnosu na bazu robota. Kroz ovaj korak, stvara se temelj za precizno izvođenje zadataka koji uključuju vizualno prepoznavanje i manipulaciju objektima u okolini robota.

Sklop robota i kamere pomoću kojeg će se izvoditi kalibracija prikazan je na sljedećoj slici:



Slika 16. Položaj kamere u odnosu na robota („*eye-in-hand*“)

Pokretanjem naredbe sa slike 17 može započeti prikupljanje slika, odnosno snimanje položaja robota.



Slika 17. Naredba za prikupljanje slika u raznim konfiguracijama robota

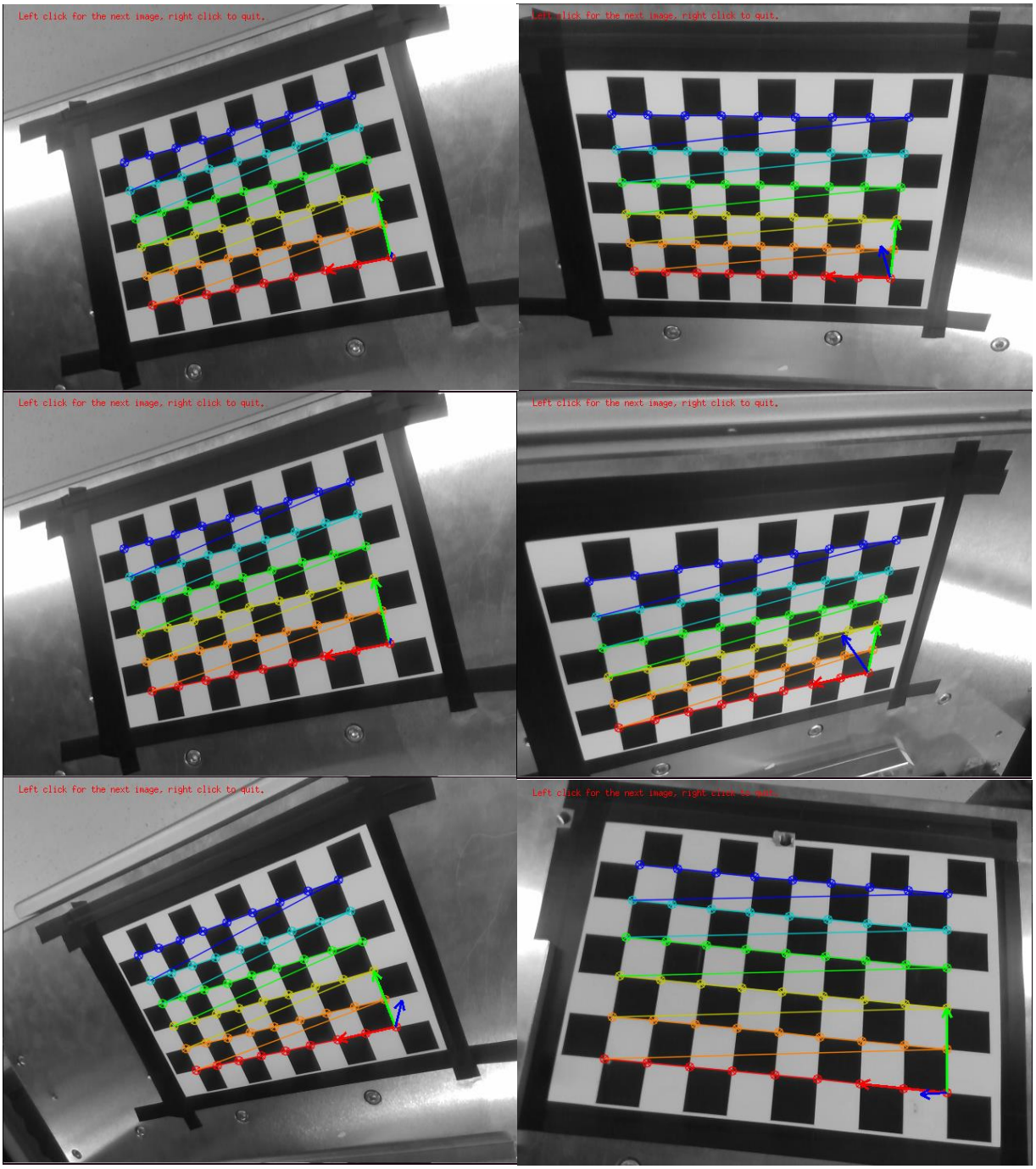
Pokretanjem ove naredbe otvara se prozor kojim je moguće postaviti robota u poziciju iz koje vidi cijelu ploču i spremiti tu poziciju u direktorij za daljnje korake. Nakon prikupljenih pozicija, potrebno je promijeniti veličinu kvadratića na 0.025 mm, kao što je navedeno i u intrinzičnoj kalibraciji, te pokrenuti naredbu sa slike 18.

```
$ ./visp-compute-chessboard-poses --square-size 0.0262 --input franka_image-%d.png --intrinsic franka_camera.xml --output franka_pose_cPo_%d.yaml
```

Slika 18. Naredba za generiranje koordinatnih sustava na sve slike[5]

Ova naredba će na prikupljene slike iz različitih robotskih pozicija generirati koordinatne sustave kojima se može provjeriti točnost kalibracije prije izračuna same matrice.

Izgled tako generiranih koordinatnih sustava je prikazan na slici 19.



Slika 19. Ekstrinzična kalibracija kamere preko slika kalibracijske ploče iz više robotskih konfiguracija

Na ovih 6 slika, prikazani koordinatni sustavi predstavljaju predloženi koordinatni sustav koji će kalibracijski algoritam koristiti za izračun matrice transformacije. Linije jasno prate izmjenu crno-bijelih kvadratića iz svakog kuta, što potvrđuje da možemo nastaviti s posljednjim korakom kalibracije.

Pokretanjem posljednje naredbe, sa slike 20,

```
$ ./visp-compute-hand-eye-calibration --data-path . --fPe franka_pose_fPe_%d.yaml --cPo franka_pose_cPo_%d.yaml --output franka_eMc.yaml
```

Slika 20. Naredba za generiranje matrice transformacije[5]

dobiveni su sljedeći rezultati:

```
** Hand-eye calibration succeed

** Hand-eye (eMc) transformation estimated:
-0.02029880595  -0.9937189484  0.1100482168  0.07525244249
0.9997809961  -0.01961471718  0.007295386718  -0.03438864915
-0.005090999368  0.1101722035  0.9938994755  -0.1154851146
0  0  0  1
** Corresponding pose vector [tx ty tz tux tuy tuz] in [m] and [rad]: 0.07525244249  -0.03438864915
 -0.1154851146  0.08200451838  0.09177904476  1.589046084

** Translation [m]: 0.07525244249 -0.03438864915 -0.1154851146
** Rotation (theta-u representation) [rad]: 0.08200451838  0.09177904476  1.589046084
** Rotation (theta-u representation) [deg]: 4.698512804 5.258551913 91.04563408
** Rotation (quaternion representation) [rad]: 0.03679821932  0.04118438209  0.7130590787  0.6989252
378
** Rotation (r-x-y-z representation) [rad]: -0.007340033758  0.1102715613  1.591220596
** Rotation (r-x-y-z representation) [deg]: -0.4205529558  6.318095065  91.17022443
```

Slika 21. Rezultati ekstrinzične transformacije

Matrica transformacije glasi:

$$T = \begin{bmatrix} -0.02029880595 & -0.9937189484 & 0.1100482168 & 0.07525244249 \\ 0.9997809961 & -0.01961471718 & 0.007295386718 & -0.03438864915 \\ -0.005090999368 & 0.1101722035 & 0.993899475 & -0.1154851146 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Od svih vrijednosti na slici 21, koja su izračunata za definiranje položaja kamere u odnosu na prihvatnicu, odnosno bazu robota potrebni su pomaci po *x*, *y* i *z* osi te rotacije *roll*, *pitch* i *yaw* označeni plavim kvadratom. Te vrijednosti bit će upotrijebljene u URDF dokumentu *„camera_D435i.xacro“* i poslužit će za prikazivanje 3D modela kamere i svih njenih koordinatnih sustava u *„rviz2“*.

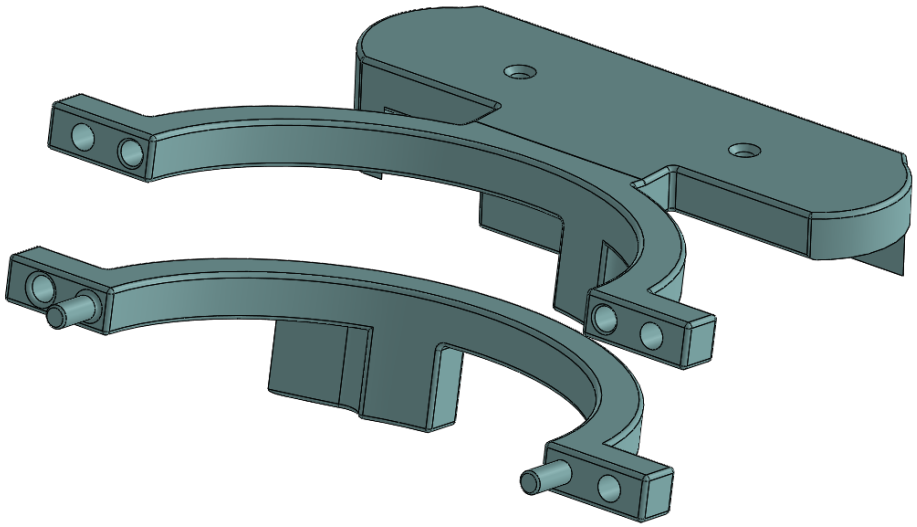
4. Projektiranje nosača kamere i radnog alata

3D modeli nosača kamere i ticala su oblikovani u programu SOLIDWORKS te su izrađeni korištenjem 3D pisača Prusa i3 MK3S+, koji je prikazan na slici 22. Za preciznost dimenzija i funkcionalnost, modeli su razvijeni uz referencu 3D modela robota Franke, odnosno modela prirubnice i prihvatnice "Franka Hand". Oba modela su napravljena od PETG plastičnog materijala relativno visoke izdržljivosti i fleksibilnosti koja je ovdje tražena.

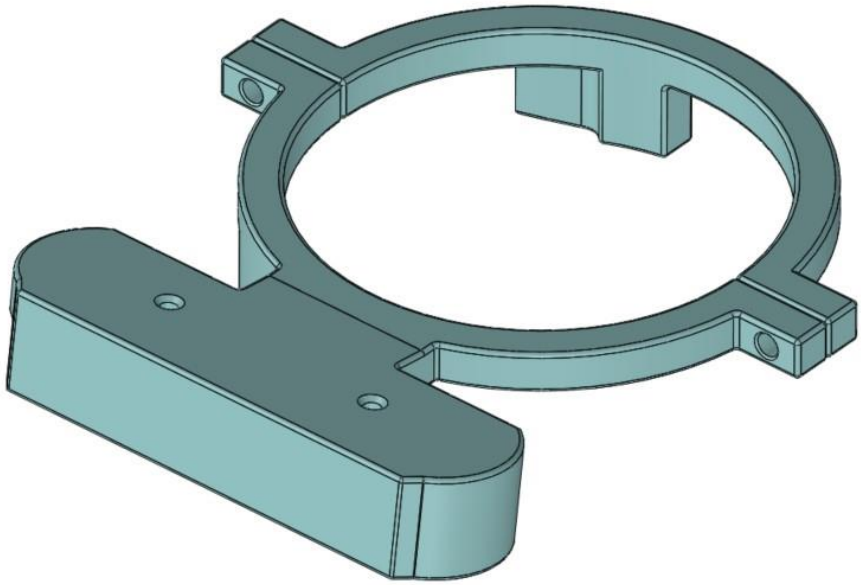
Slika 22. 3D pisač i3 MK3S+[9]

4.1 Nosač kamere Intel RealSense D435i

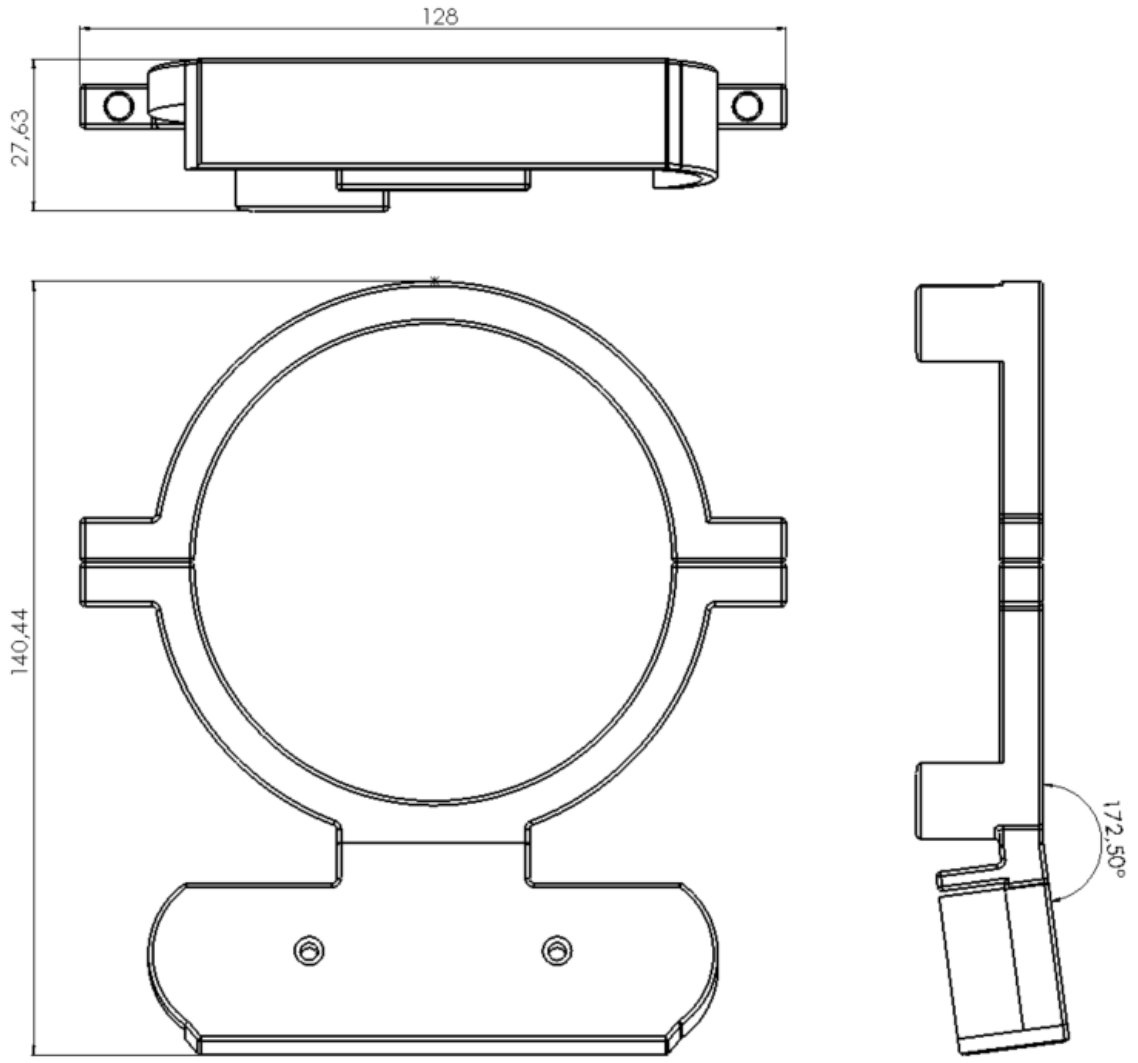
Nosač kamere je napravljen od dva dijela koji se povezuju pomoću dva cilindra za vođenje i dva vijka M4. Stabilnost i čvrsti spoj s prirubnicom osigurani su oblikom, pomoću dva graničnika i trenjem, odnosno steznom silom vijaka. Ovaj dizajn ima za cilj osigurati čvrsto povezivanje dijelova nosača, sprječavajući neželjeno pomicanje i osiguravajući stabilnost kamere tijekom rada robota.



Slika 23. Rastavljeni prikaz nosača kamere



Slika 24. 3D prikaz nosača kamere

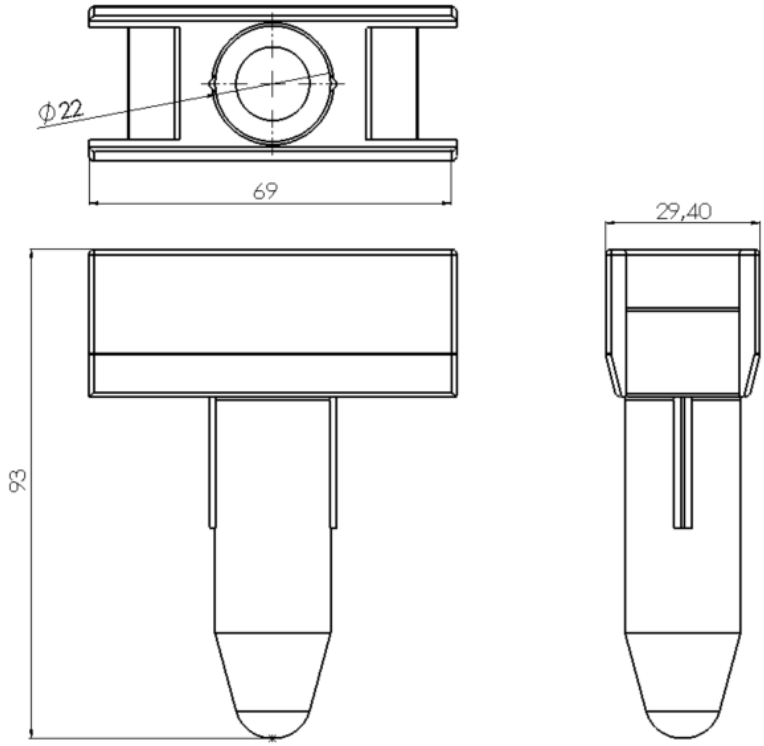


Slika 25. Gabaritne dimenzije nosača kamere

4.2 Dodirno ticalo

Dodirno ticalo je napravljeno malih dimenzija kako bi se spriječilo njegovo uvijanje i osigurala preciznost u dodiru s okolinom. Ticalo se postavlja na prste prihvatnice „Franka Hand“ i naredbom *„grasp“* hvata silom od 70 N.

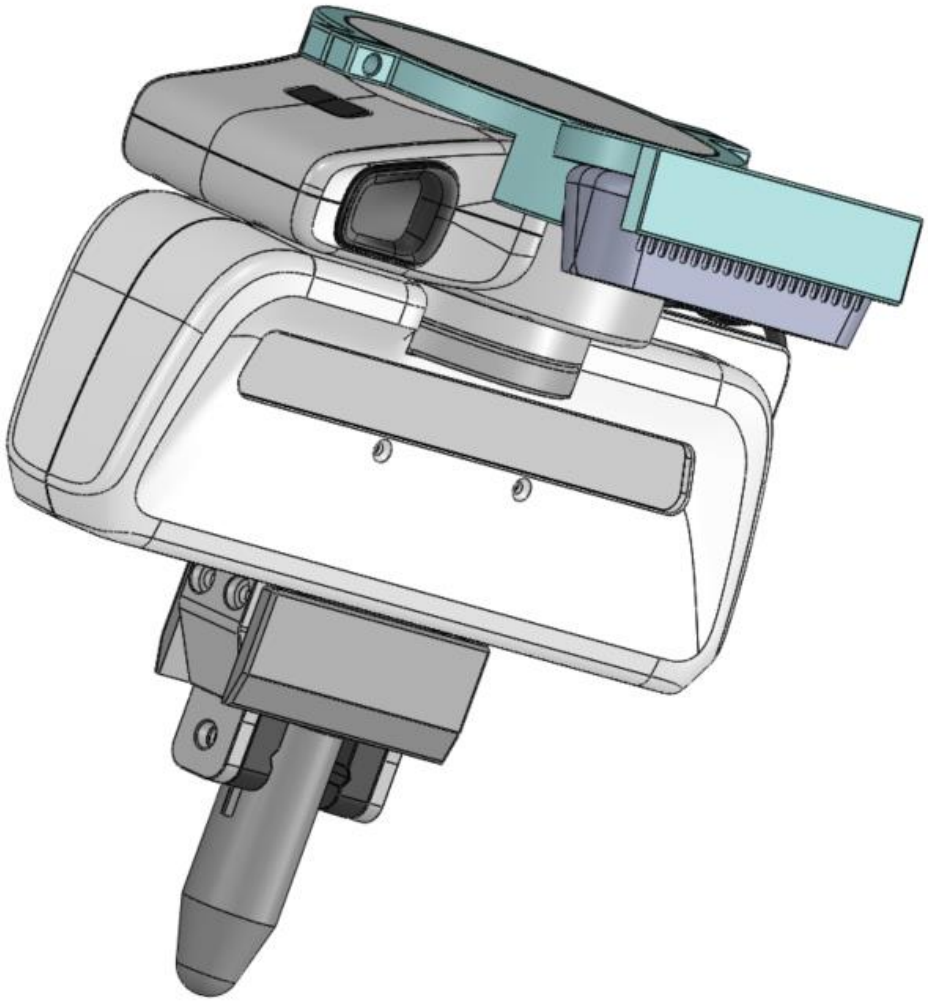
Slika 26. 3D prikaz dodirnog ticala



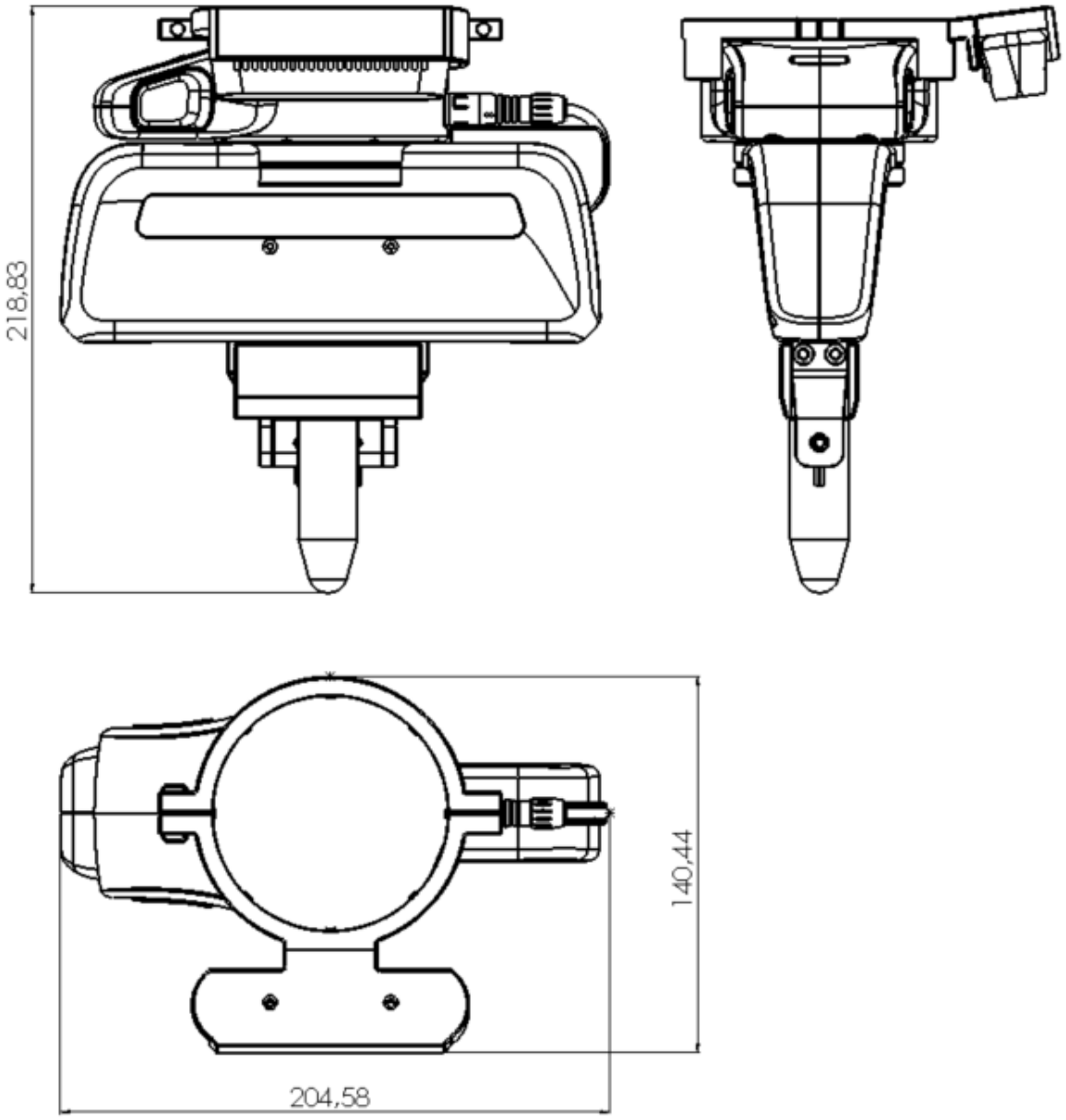
Slika 27. Gabaritne dimenzije dodirnog ticala

4.3 Sklop prirubnice, prihvatnice „Franka Hand“, cala i nosača s kamerom

Konačni izgled prihvatnice i prirubnice s uključenim svim modelima prikazan je na slici 28. Montaža nosača kamere je osmišljena je tako da ne ograničava slobodu gibanja robota u bilo kojem smjeru te da omogućuje jednostavnu montažu i demontažu. Kamera D435i pričvršćuje se na nosač pomoću dva M3 vijka koji osiguravaju stabilnost i nepomičnost tijekom gibanja robota. Kut nosača od 172.5° prilagođen je kako bi kamera mogla vidjeti vrh robotskog alata i maksimizirati prostor ispred sebe. Gabaritne dimenzije ovog sklopa iznose 204.58x140.44x218.83.



Slika 28. 3D prikaz sklopa prirubnice, prihvatnice "Franka Hand" s 3D modelima



Slika 29. Gabaritne dimenzije sklopa robota s 3D modelima

5. ROS2 za planiranje gibanja

5.1 ROS2 i MoveIt2

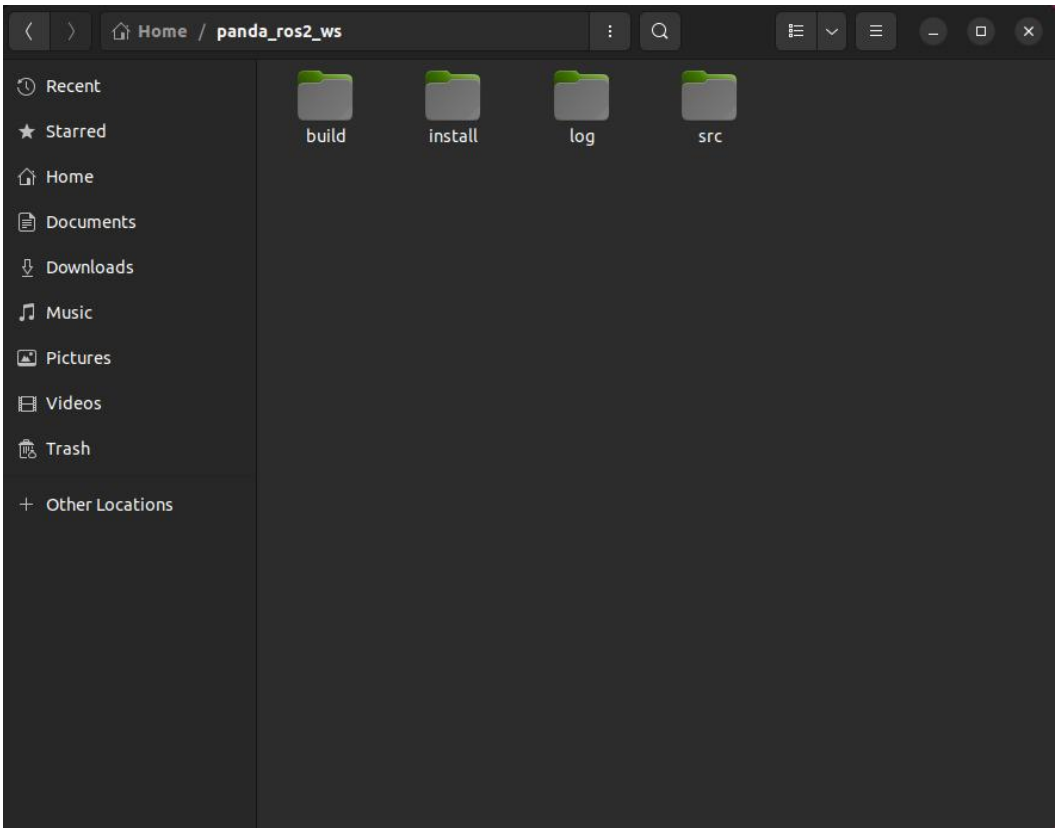
Za optimalan način rada ROS2 računalo bi trebalo raditi na Linux Ubuntu 22.04 (ova verzija je najpogodnija za integraciju sa ostalim programima koji se koriste u ovom radu) operativnom sustavu.

The Robot Operating System ROS2 je „*open-source"* radno okružje koje služi za razvijanje i ponovno korištenje robotskih aplikacija. Glavna je prednost što je dostupan svima pa tako omogućuje suradnju stručnjaka, razvojnih inženjera i raznih hobista koji svi imaju zajednički cilj, a to je približavanje robotike svima i poboljšanje robotike kao grane. Također postoje razna rješenja poznatih problema koja su dostupna svima na web stranicama[10]. U ovom se radu koristi ROS2 *humble* iz razloga što je za primjenu MoveIt2 paketa i kod primjene na stacionarnim robotima ova verzija najstabilnija i najviše razvijena. Iz istog razloga je odabrana verzija 22.04 Ubuntu-a, koja jedina podržava ovu verziju ROS2. Instalacija ROS2 na računalo je detaljnije objašnjena na službenoj stranici[10].

MoveIt2 je platforma za robotsku manipulaciju u ROS2 okruženju. MoveIt2 objedinjuje najnovija postignuća u planiranju gibanja, manipulaciji, 3D percepciji, kinematici, kontroli i navigaciji. Ovaj će se paket koristiti za upravljanje fizičkim i simuliranim robotom. Prednost MoveIt2 platforme proizlazi iz toga što upravlja robotom nizom kontrolera na koje možemo direktno utjecati preko računala. Također se istovremeno može upravljati s više robota ili kao što je to slučaj u ovom radu, sa simulacijom i robotskom rukom. U ovom radu, MoveIt2 će biti korišten za planiranje kartezijskih putanja (engl. *„Cartesain Path"*) iz jedne točke u drugu.

5.2 Način rada

ROS2 komunicira unutar sebe preko tema (engl. *„Topic"*) koje se ponašaju kao „teme razgovora". Ako bi se na neku temu željelo davati informacije tada bi trebao postojati „ROS2 pošiljatelj (engl. *„Publisher"*). Pošiljatelj je vrsta čvora (engl. „*node"*) (tip izvršne jedinice koja je spojena na ROS2 mrežu) koja neprekidno šalje podatke na temu. Kako bi se razgovor upotpunio potreban nam je slušatelj, u slučaju ROS2 pretplatnik (engl. „*Subscriber"*). Pretplatnik je također vrsta čvora koja prima informacije koje šalje pošiljatelj na istu temu. To je osnovni princip komunikacije unutar ROS2. Kako bi koristili ugrađene funkcije ili već postojeće s interneta potrebno je stvoriti radno područje ili okruženje (engl. *„workspace"*) [13]. Primjer takvog radnog okruženja „panda_ros2_ws" koji je korišten za izradu ovog rada.



Slika 30. ***"panda_ros2_ws"*** **direktorij**

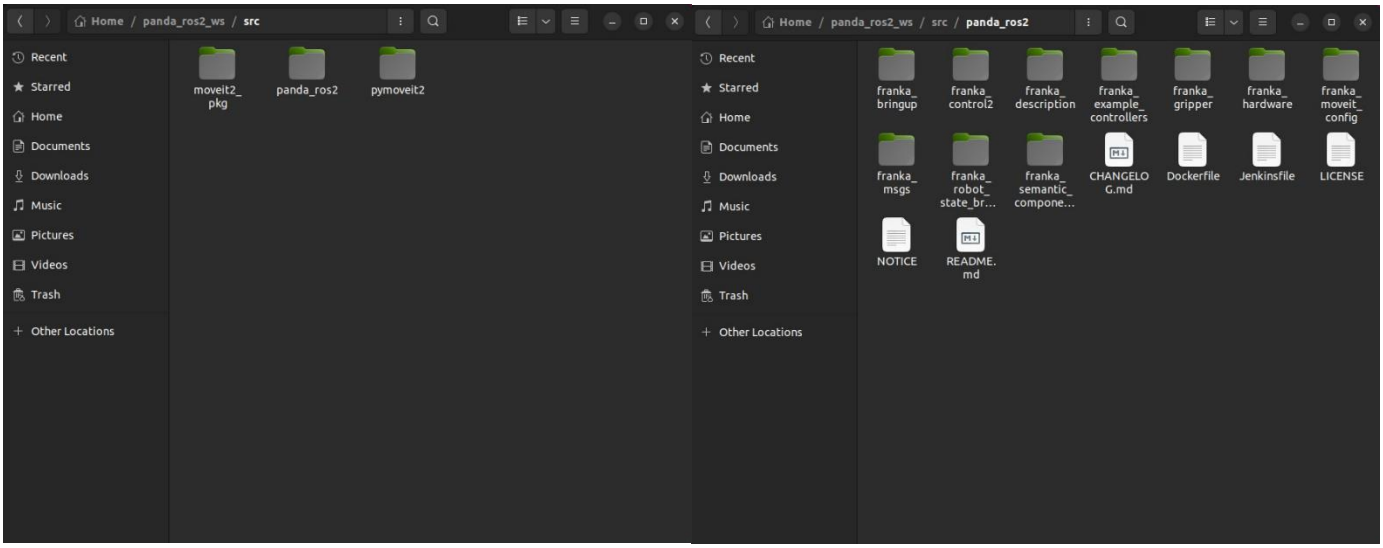
Unutar radnog područja se grade paketi. Paketi su način na koji je ROS2 organiziran i mogu sadržavati: čvorove, konfiguracijske datoteke, ROS2 neovisne biblioteke, programe drugih proizvođača i slično. Cilj paketa je da pruže korisne funkcionalnosti na način koji je lako obraditi tako da bi se taj program mogao jednostavno ponovno iskoristiti.

5.3. Postavljanje „panda_ros2_ws“ radnog okruženja

Radno područje „panda_ros2_ws“ stvara se naredbom: *„mkdir -p /panda_ros2_ws/src“*, koja stvara *src* direktorij unutar radnog okruženja. Unutar *src* direktorija nalaze se svi ROS2 paketi potrebni za upravljanje robotom. Osim tih paketa, za ovaj rad, potreban je i *„py_moveit“*[14] paket te jedan paket u kojem će se pohranjivati sve Python skripte potrebne za ovo istraživanje, nazvan *„moveit2_pkg“*. Direktorij za upravljanje Franka Emika robotom zove se *„panda_ros2“*[7], koji sadrži pakete za osnovno upravljanje. Ti paketi prikazani su na slici 31.

Od svih tih paketa, za ovaj rad, najbitnija su dva koja služe za pokretanje robota u *„rviz2“* i ostvarivanje njegovog gibanja. To su paketi *„franka_description“* i *„franka_moveit_config“*. Paket *„franka_description“* sadrži sve potrebne datoteke koje opisuju Franka Emika robotsku ruku, prihvatnicu „Franka Hand“, vrste zglobova, dimenzije i ograničenja. U *„franka_moveit_config“* paketu se nalaze datoteke za pokretanje kontrolera, ali istovremeno poziva prethodni paket kako bi prikazao robota u „rviz2“.

Nakon prikupljanja svih paketa direktorij se može izgraditi (engl. „build“) naredbom *„colcon build“* kojom se stvaraju još tri direktorija, *„build“* (sve datoteke koje generira *„colcon“* idu u ovaj direktorij), *„install“* (sve konfiguracijske datoteke stvorene nakon izgradnje direktorija smještene su ovdje) i *„log“* (direktorij u kojem se prikazuju pogreške prilikom izgradnje direktorija, ako ih ima). Uspješnom izgradnjom direktorija robot je spreman za korištenje. Dodatni paketi za upravljanje robotom će se detaljno objasniti u sljedećem poglavlju.



Slika 31. Prikaz *"src"* direktorija (lijevo) i *"panda_ros2"* paketa (desno)

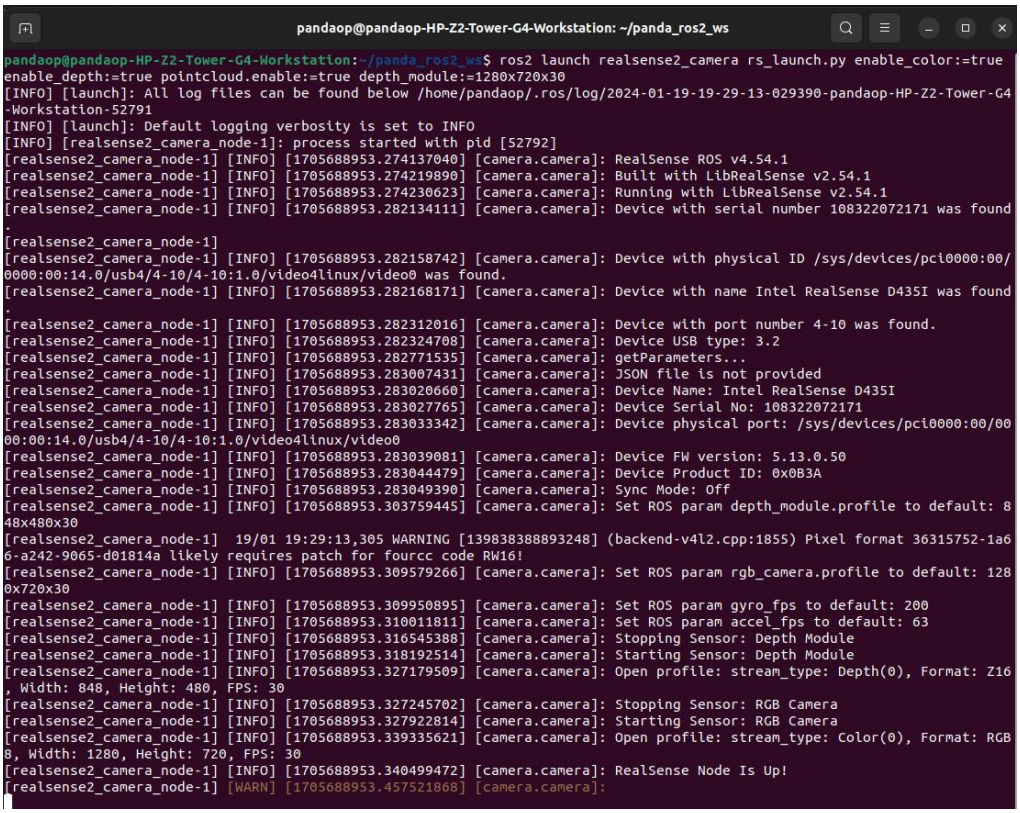
5.4 Oblak točaka (engl. *„Pointcloud“*)

Modul kamere, koji će se koristiti u okviru ovog istraživanja, je dubinski modul koji generira oblak točaka (engl. *„Pointcloud“*). Ovaj modul koristi kombinaciju stereo vizualne percepcije i dubinskih senzora kako bi stvorio trodimenzionalni prikaz okoline. Trodimenzionalna slika formirana je od mnogo malih točaka koje zajedno čine trodimenzionalni prikaz u sustavu *"rviz2"* unutar ROS2.

„rviz2“ je ROS2 grafičko sučelje koje se koristi za prikaz raznih informacija. Unutar *„rviz2“* se moguće pretplatiti na aktivne teme. Primarno će se koristiti za prikaz simuliranog kretanja robota i prikaza trodimenzionalne slike kamere, odnosno oblaka točaka.

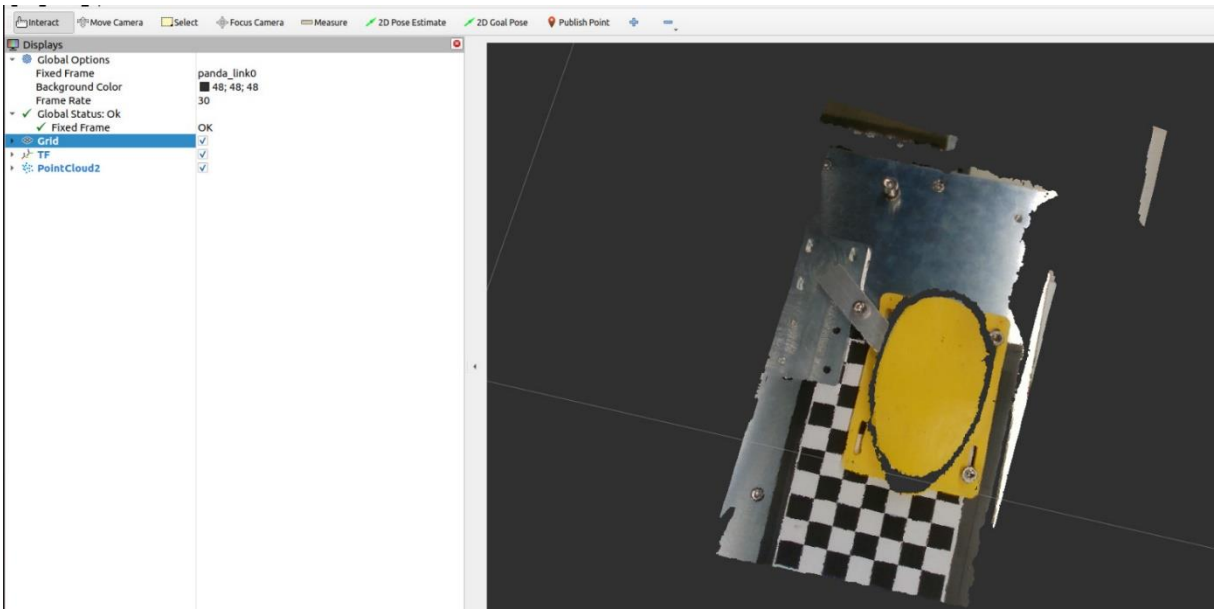
Upravljanje kamerom u ROS2 omogućeno je pomoću paketa *„realsense-ros“*[8]. Za korištenje čvorova za pokretanje kamere potrebno je preuzeti i instalirati biblioteku "librealsense2". Uspješnom instalacijom biblioteke može se instalirati paket *„realsense-ros“* kao Debian paket s ROS2 servera ili preuzimanjem i instaliranjem izvornog koda *„realsense-ros“*.

Za prikaz pointclouda u *"rviz2"*, potrebno je u terminalu pokrenuti čvor s nodom *„ros2 run realsense2_camera realsense2_camera_node“* ili *„launch“* datotekom *„ros2 launch realsense2_camera rs_launch.py“* i potrebnim argumentima.



Slika 32. Pokretanje *"rs_launch.py"* skripte s potrebnim argumentima

Pokretanjem jednog od navedenih čvorova, automatski će se uspostaviti veza s temama kamere koje objavljuju podatke. Jedna od njih je *"/camera/depth/color/points"* koja pruža oblak točaka u obliku trodimenzionalne slike vidljivog polja kamere. Tu ćemo sliku upotrijebiti za odabir ciljnih točaka robota.

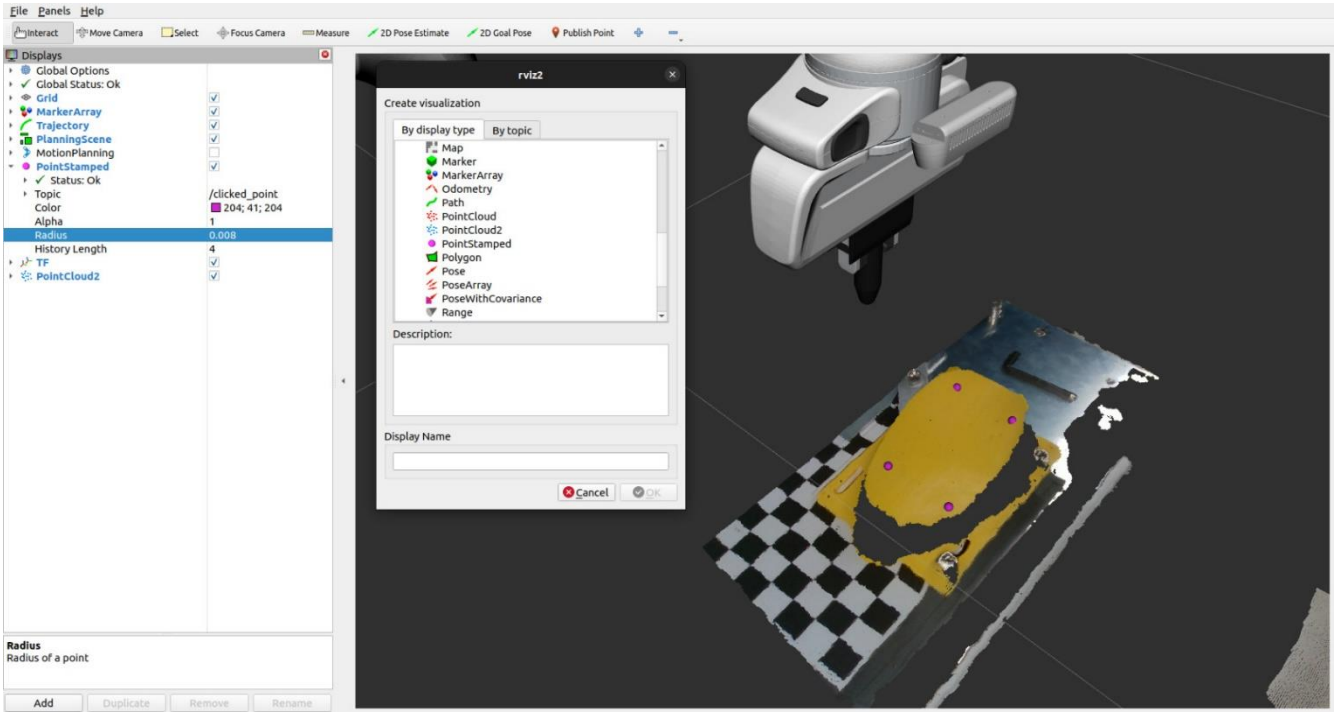


Slika 33. Prikaz *„pointclouda“* u rviz2

Prikaz slike nije savršen i postoji odstupanje od nekoliko milimetara koje se može primijetiti kao treperenje točaka. Problem za to nije riješen u *„rviz2“*, ali se s *„realsense-viewer“*-om može primijetiti poboljšanje. Integracija alata *„realsense-viewer“* u cjelokupni sustav ili poboljšavanje slike u *„rviz2“* nije razmatrana u ovom radu, no može se uzeti u obzir kao mogućnost za buduće proširenje istraživanja.

5.5 Metoda odabira točke pomoću „*Pointclouda*“

Postoji mnogo metoda za dobivanje koordinata točke s prostorne slike, no ROS2 nudi praktični alat u *"rviz2"* sučelju pod nazivom *"Publish Point"*. Klikom na 3D sliku u radnom prostoru *„rviz2“* prikazat će se ljubičasti marker, koji označava mjesto željene točke i pripadne x, y i z koordinate. Ovaj princip funkcionira samo u slučaju dobre kalibracije kamere. Princip rada ove funkcije u ROS2 je taj da „Publish Point“ prilikom klika na određenu točku u prostoru šalje poruku (engl. *„publish“*) oblika *„geometry_msgs/msg/PointStamped“* na temu *„/publish_point“*. Pretplaćivanjem *(*engl. *„subscribe“)* na tu temu možemo pristupiti tim podacima i iskoristiti ih za zadavanje koordinata ciljne točke.



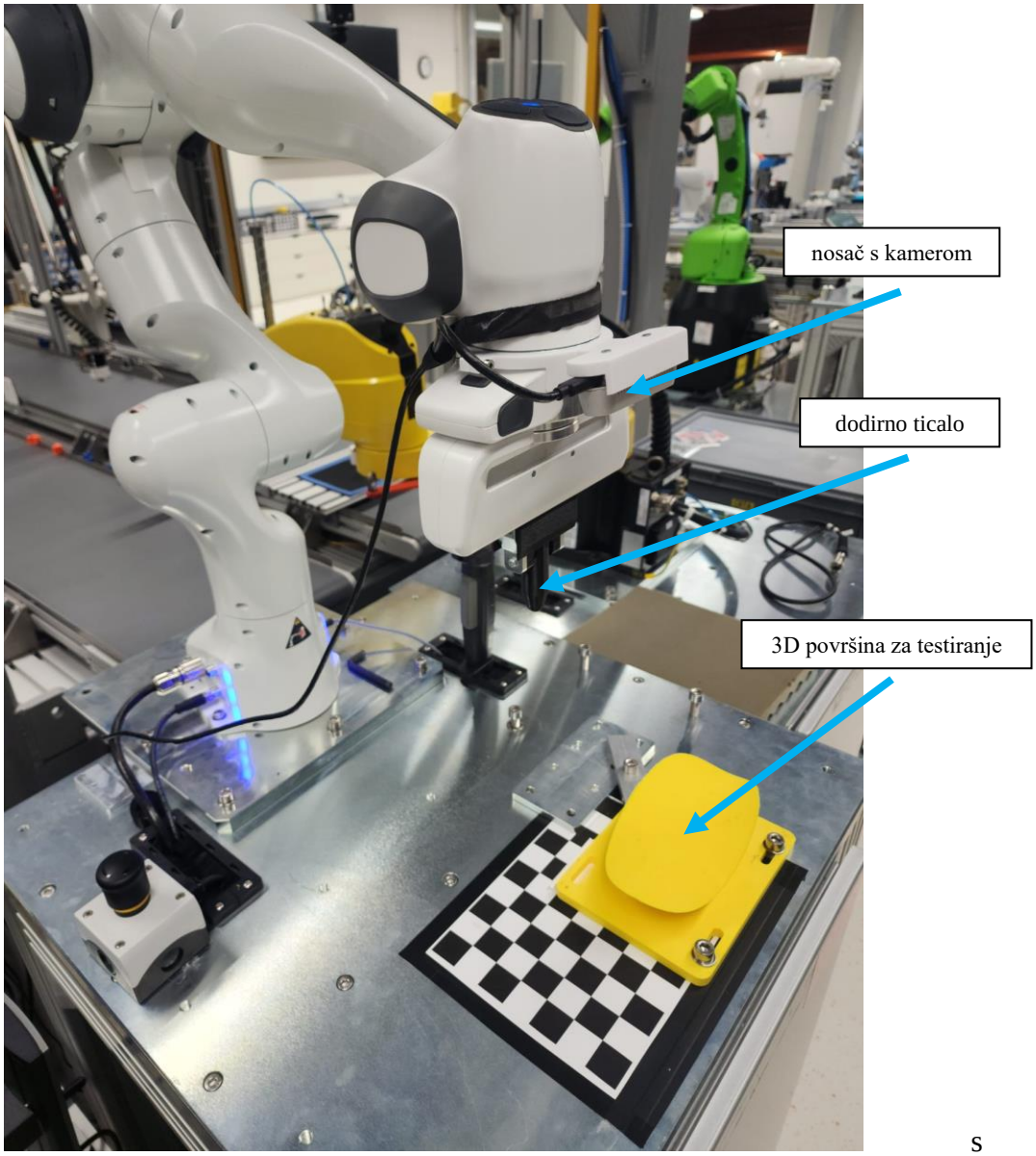
Slika 34. Prikaz čvora *"PointStamped"* s pozicijskim markerima

Vizualizacijski čvor *„PointStamped“* koji omogućuje izbor pozicijskog markera može se naći u izborniku prikazanom na slici 34, kojeg otvara tipka *„Add“*. Pomoću tog čvora i aktivacijom *„Publish Point“* alata mogu se odabirati markeri u radnom prostoru *„rviz2“*. S desne strane sučelja *„rviz2“*, pod čvorom *„PointStamped“* može se odabrati tema na koju se želimo pretplatiti, a to je *„/clicked_point“* te ima mogućnost odabira boje markera, njegove veličine te broja točaka koje će pohraniti prije nego se resetira.

6. Postavljane eksperimentalne radne stanice

Nakon uspješne instalacije svih ROS2 paketa, konstruiranja 3D modela, postavljanja kamere na robota i kalibracije kamere, slijedi postavljanje radne stanice i radnog okruženja u ROS2. To će biti postignuto modifikacijom postojećih *„launch.py“* datoteka, pisanjem novih *„launch.py“* datoteka i pisanjem novih skripti u Pythonu. Također, potrebno je unaprijediti ROS2 okruženje dodavanjem konstruiranih 3D modela, što uključuje modeliranje *„.urdf.xacro“* i *„.xacro“* datoteka robota te izradu novih URDF-a datoteka – jednu za kameru i jednu za ticalo.

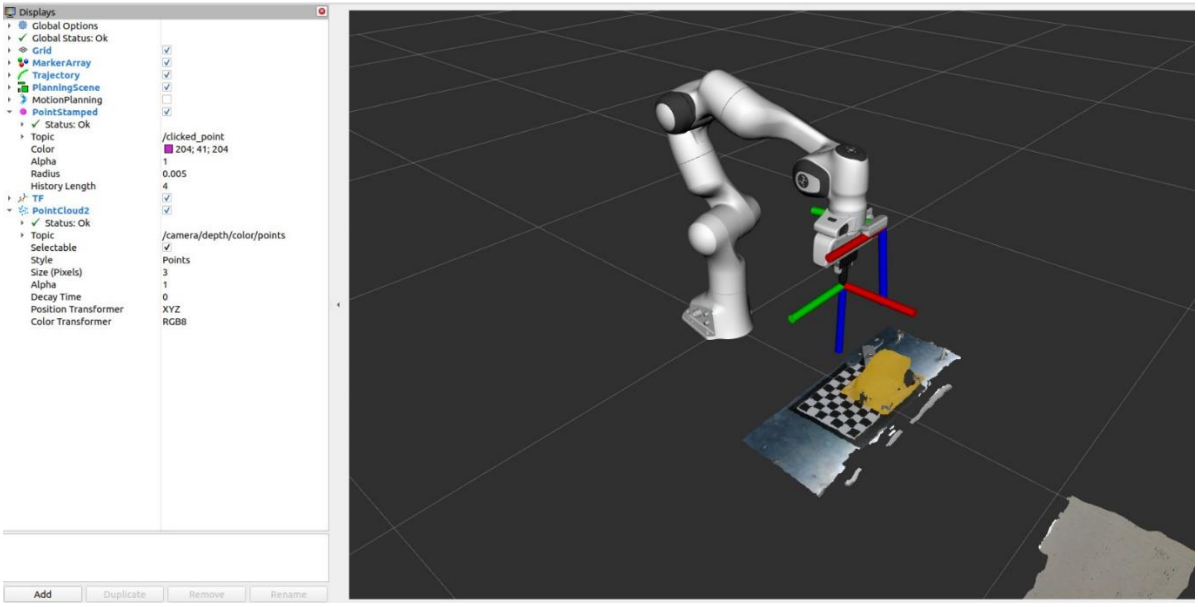
Konačni izgled radne stanice u prostoru prikazan je na slici 35.



S

Slika 35. Izgled radne stanice u prostoru

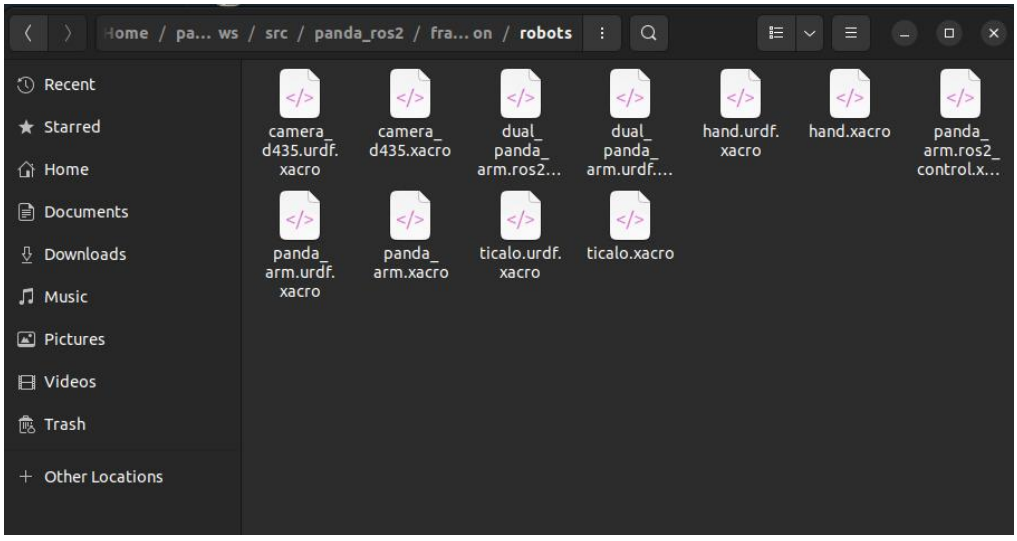
Konačni izgled ROS2 okruženja je prikazan na slici 36.



Slika 36. Izgled radne stanice u *"rviz2"*

6.1 Integracija svih elemenata preko URDF datoteka

Kako bi se integriralo modele kamere i radnog alata s njihovim koordinatnim sustavima, u *„rviz2“* potrebno je unijeti njihove URDF datoteke u paket *„franka_description“*, pod direktorij *robots*.



Slika 37. Direktorij *"robots"* sa svim URDF datotekama

Kamera uz glavni koordinatni sustav „camera_link“ sadrži još osam koordinatnih sustava koje ćemo modelirati unutar URDF datoteke. Ti koordinatni sustavi su: *„camera_depth_frame“*, *„camera_depth_optical_frame“*, *„camera_color_frame“*, *„camera_color_optical_frame“*, *„camera_infra1_frame“*, *„camera_infra1_optical_frame“*, *„camera_infra2_frame“* i *„camera_infra2_optical_frame“*.

Dodirno ticalo je jednostavnije te su za njegovu definiciju dovoljna su dva koordinatna sustava, jedan koji označava ishodište samog modela *„panda_ticalo“* i drugi koji označava vrh, *„panda_ticalo_tcp“*.

URDF datoteke, koje definiraju modele kamere i alata, nalaze se na kraju rada u Prilogu 1 i 2. Paralelno s izradom samih URDF datoteka za dodatne elemente sustava, ključno je integrirati ih u sam sustav robota kako bi se mogli generirati bez potrebe za dodatnim naredbama. Integracija uključuje usklađivanje novih URDF datoteka s postojećim dijelovima robotskog sustava, omogućujući njihovo glatko funkcioniranje unutar istog okruženja. Ovaj ključni korak integracije zahtijeva prilagodbe kako bi se novi elementi pravilno povezali s ostalim dijelovima robota. Posebna pažnja posvećuje se kinematičkim i dinamičkim karakteristikama kako bi se osiguralo ispravno djelovanje novih dodataka. Cilj je da novi elementi budu prepoznati u ROS2 okruženju, doprinoseći cjelovitosti sustava.

Detalji o načinu integracije prikazani su na slikama 38 i 39, pružajući vizualni uvid u korake integracije novih elemenata u robotski sustav. Ova vizualna podrška omogućuje jasnije razumijevanje postupka integracije i povezivanja novih komponenti s postojećim dijelovima robota unutar ROS2 okruženja.

```
<xacro:arg name="ticalo" default="false"/> <!-- Should ticalo be mounted?"
-->
<xacro:arg name="camera" default="false"/> <!-- Should camera be mounted?"
-->
<xacro:arg name="robot_ip" default=""/> <!-- IP address or hostname of the robot" -->
<xacro:arg name="use_fake_hardware" default="false"/>
<xacro:arg name="fake_sensor_commands" default="false"/>

<xacro:include filename="$(find franka_description)/robots/panda_arm.xacro"/>
<xacro:panda_arm arm_id="$(arg arm_id)" safety_distance="0.03"/>

<xacro:if value="$(arg hand)">
  <xacro:include filename="$(find franka_description)/robots/hand.xacro"/>
  <xacro:hand ns="$(arg arm_id)" rpy="0 0 ${-pi/4}" connected_to="$(arg arm_id)_link8" safety_distance="0.03"/>
</xacro:if>
  <xacro:if value="$(arg ticalo)">
  <xacro:include filename="$(find franka_description)/robots/ticalo.xacro"/>
  <xacro:ticalo ns="$(arg arm_id)" rpy="0 0 0" connected_to="$(arg arm_id)_hand" safety_distance="0.0002"/>
</xacro:if>
<xacro:if value="$(arg camera)">
  <xacro:include filename="$(find franka_description)/robots/camera_d435.xacro"/>
  <xacro:camera ns="$(arg arm_id)" rpy="0 0 0" connected_to="$(arg arm_id)_hand_tcp" safety_distance="0.01"/>
</xacro:if>
```

Slika 38. Modifikacija datoteke *"panda_arm.urdf.xacro"*

```
<group_state name="open" group="hand">
  <joint name="${arm_id}_finger_joint1" value="0.035"/>
  <joint name="${arm_id}_finger_joint2" value="0.035"/>
</group_state>
<group_state name="close" group="hand">
  <joint name="${arm_id}_finger_joint1" value="0"/>
  <joint name="${arm_id}_finger_joint2" value="0"/>
</group_state>
<!-- <end_effector name="hand" parent_link="${arm_id}_link8" group="hand" parent_group="${arm_id}_arm"/> -->
<end_effector name="ticalo_tcp" parent_link="${arm_id}_ticalo_tcp" group="hand" parent_group="${arm_id}_arm"/>
<end_effector name="hand_tcp" parent_link="${arm_id}_hand_tcp" group="hand" parent_group="${arm_id}_manipulator"/>
```

Slika 39. Modifikacija datoteke *"panda_arm_hand.xacro"*

6.2 Korištenje ROS2 paketa za izvođenje jednostavnih radnji u *„rviz2“* i u prostoru

Kako bi se MoveIt2 paket mogao koristiti u ovom radu s navedenim robotom, svim modifikacijama i dodatnim elementima potrebno je nadopuniti *„launch“* datoteku *„moveit.launch.py“* koja pokreće *„rviz2“* i robotske kontrolere. Na sljedećim slikama prikazane su nadopune programskog koda koje služe za inicijalizaciju kamere i ticala.

```
def generate_launch_description():
    robot_ip_parameter_name = 'robot_ip'
    load_ticalo_parameter_name = 'load_ticalo'
    load_camera_parameter_name = 'load_camera'
    use_fake_hardware_parameter_name = 'use_fake_hardware'
    load_gripper_parameter_name = 'load_gripper'
    fake_sensor_commands_parameter_name = 'fake_sensor_commands'

    robot_ip = LaunchConfiguration(robot_ip_parameter_name)
    use_fake_hardware = LaunchConfiguration(use_fake_hardware_parameter_name)
    load_ticalo = LaunchConfiguration(load_ticalo_parameter_name)
    load_camera = LaunchConfiguration(load_camera_parameter_name)
    load_gripper = LaunchConfiguration(load_gripper_parameter_name)
    fake_sensor_commands = LaunchConfiguration(fake_sensor_commands_parameter_name)
```

Slika 40. Modifikacija *"launch"* datoteke, 1. dio

```
robot_description_config = Command(
    [FindExecutable(name='xacro'), ' ', franka_xacro_file, ' hand:=', load_gripper,
     ' robot_ip:=', robot_ip, ' ticalo:=', load_ticalo, ' camera:=', load_camera,
     ' use_fake_hardware:=', use_fake_hardware, ' fake_sensor_commands:=', fake_sensor_commands])
```

Slika 41. Modifikacija *"launch"* datoteke, 2. dio

```
ticalo_arg = DeclareLaunchArgument(
    load_ticalo_parameter_name,
    default_value='true',
    description='Mount Ticalo on the Franka Gripper as end-effector')
camera_arg = DeclareLaunchArgument(
    load_camera_parameter_name,
    default_value='true',
    description='Mount Camera d435i on the robot.')
```

Slika 42. Modifikacija *"launch"* datoteke, 3. dio

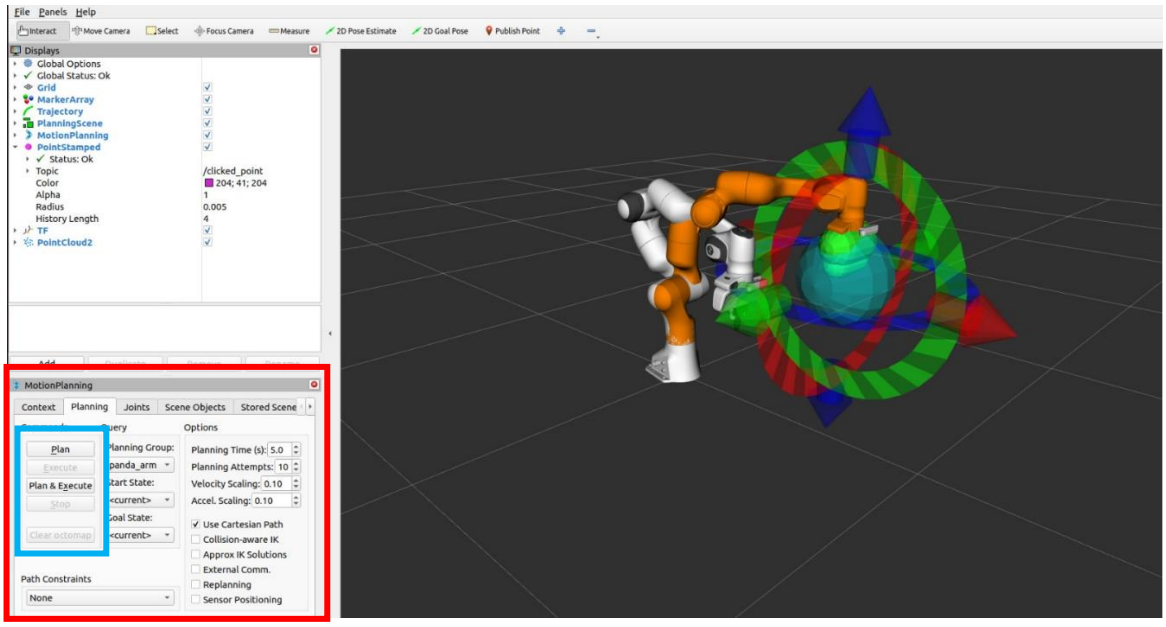
```
return LaunchDescription(
    [robot_arg,
     use_fake_hardware_arg,
     fake_sensor_commands_arg,
     ticalo_arg,
     camera_arg,
     load_gripper_arg,
     db_arg,
     rviz_node,
     robot_state_publisher,
     run_move_group_node,
     ros2_control_node,
     mongodb_server_node,
     joint_state_publisher,
     gripper_launch_file
     ]
    + load_controllers
)
```

Slika 43. Modifikacija *"launch"* datoteke, 4. dio

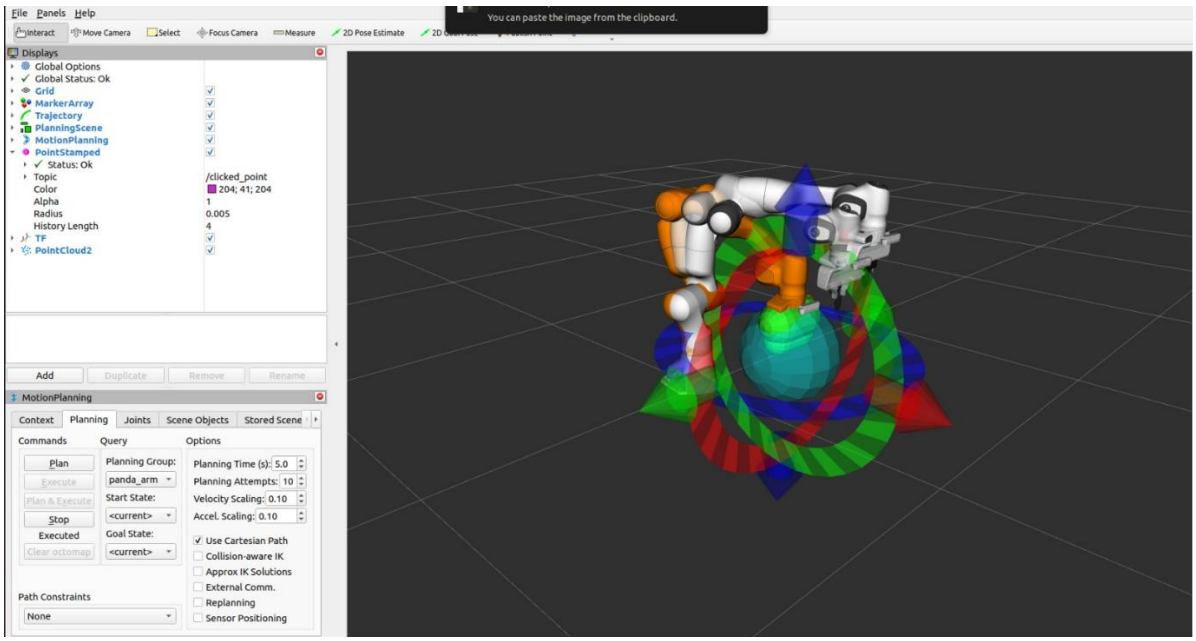
Modifikacijom *„launch“* datoteke *„moveit.launch.py“* , potrebno je ponovno izgraditi (engl. *„build“*) paket „franka_moveit_conifg“ koristeći naredbu *„colcon build“*.

Nakon izgradnje, pokretanjem nove *„launch“* datoteke otvara se prozor *„rviz2“* s prikazom robota, kamere, novog radnog alata i interaktivnog markera za pomicanje robota. Marker omogućuje translacijsko, rotacijsko ili kombinirano gibanje robota. Gibanje se definira pomoću čvora „MotionPlanning“ i može biti definirano preko zglobova (*„joint motion“*) ili linearno (*„linear motion“*) pomoću kartezijskih koordinata.

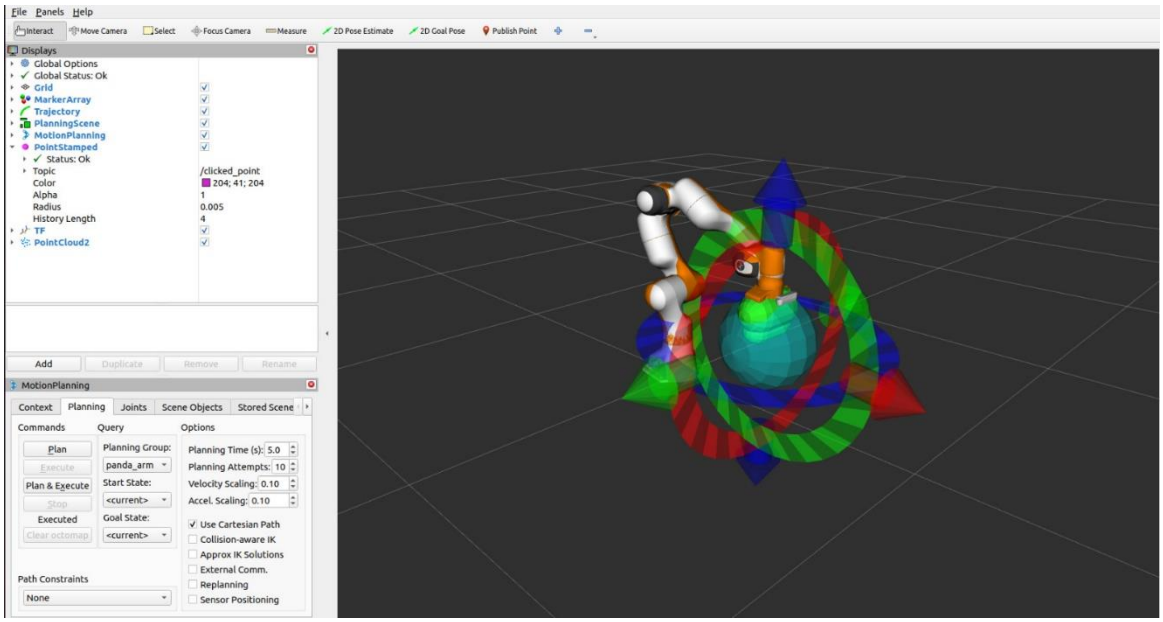
Na slikama 45, 46 i 47 prikazani su koraci za postavljanje i ostvarivanje nove ciljne točke za robota putem linearnog (kartezijskog) gibanja.



Slika 44. Odabir željene pozicije pomoću markera



Slika 45. Gibanje robota u željenu poziciju pomoću *"MotionPlanning"* čvora

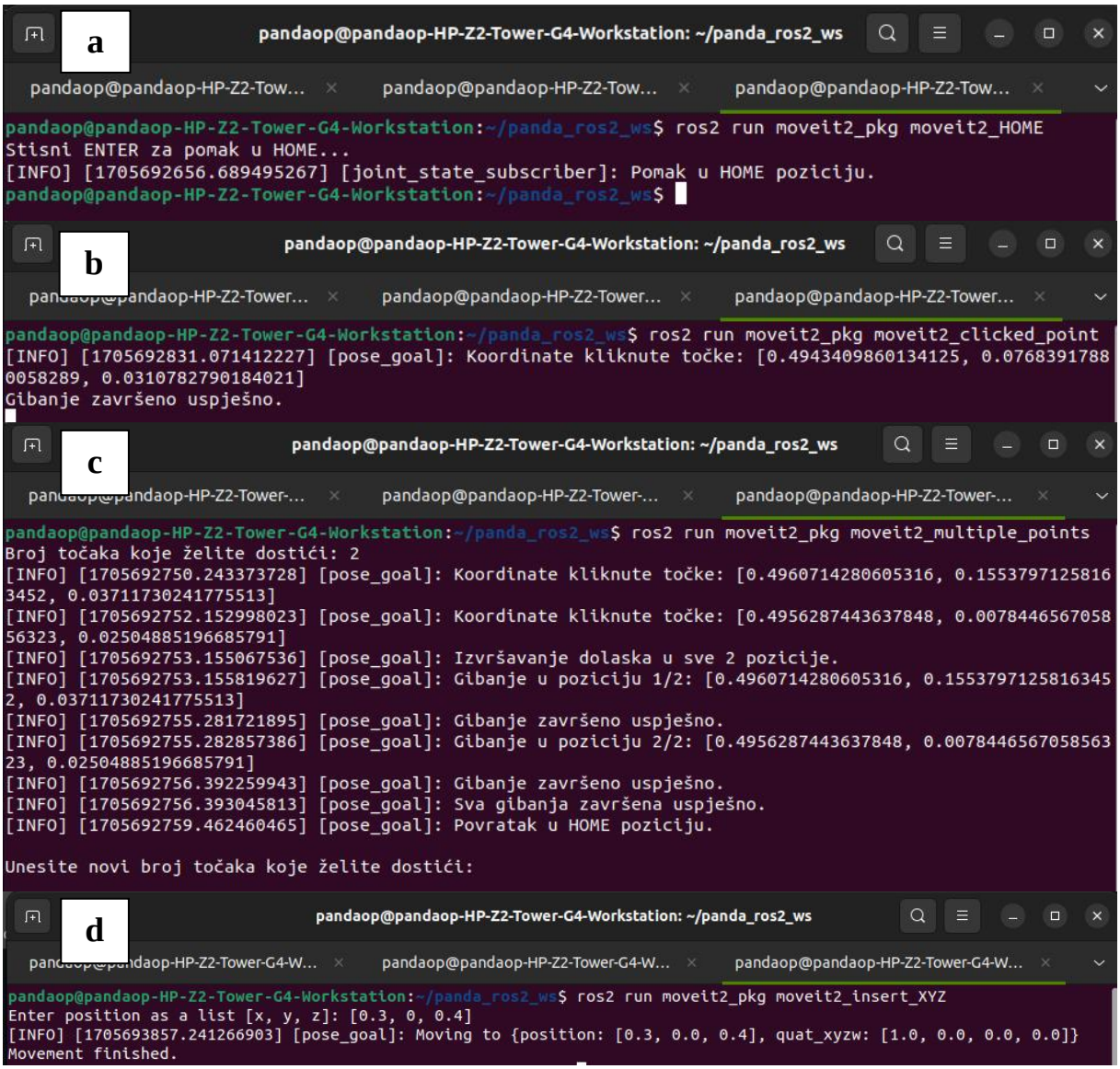


Slika 46. Dolazak robota u željenu poziciju

6.3 Korištenje *„pymoveit2“* paketa i P*ython* skripti za gibanje robota

Osim jednostavnog korištenja MoveIt2 paketa za upravljanje kretanjem robota, postoje razni paketi na GitHub repozitoriju koji su korišteni za kontrolu sličnih ili istih robota. Jedan od takvih paketa je *„pymoveit2“* paket[14]. *„pymoveit2“* paket sadrži funkcije koje definiraju ograničenja i obilježja robotskih pokreta, ali ne i same pokrete, već se parametri gibanja, kao što su točke, orijentacija i način gibanja moraju definirati u posebnim *Python* skriptama. Pozivanjem klase MoveIt2 iz „pymoveit2.py“ omogućuje se korištenje funkcija i parametara definiranih unutar nje.

U nastavku su prikazana četiri terminala gdje su pozvane *Python* skripte različitih zadaća.



Slika 47. Terminali kod pokretanja skripti *"moveit2_HOME"* (a), *"moveit2_clicked_point"* (b), *"moveit2_multipe_points"* (c) i *"moveit2_insert_XYZ"* (d)

Kako bi skripte mogle pokretati robota potrebno je upaliti čvor za aktivaciju MoveIt2 *„moveit.launch.py“* i kamere *„rs_launch.py“*, u slučaju da se točke biraju pomoću oblaka točaka. Prilikom rada s površinom potrebno je podesiti parametre kolizije i kontakta. Pomoću skripte „SetCollisionBehaviour.py“[20] postiže se upravo to i omogućuje podizanje praga sile koja može djelovati na vrh alata.

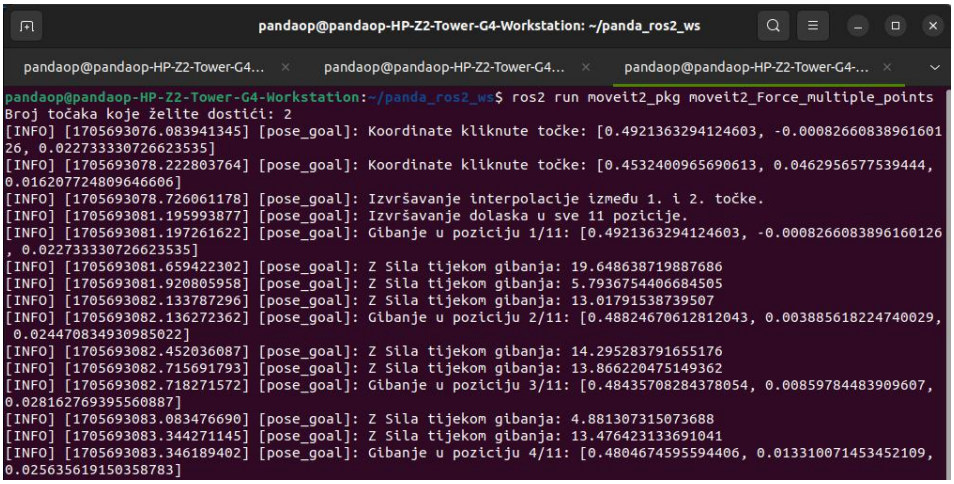
6.4 „Cartesian Force“ kontroler

Kako u ROS2 nije integriran kontinuirani kontroler sile, bit će potrebno od postojećeg *„panda_arm_controller“*-a napisati logiku kojom će konstantno provjeravati silu i na taj način diskretno pratiti zadanu putanju.

Robot u sebi ima sedam momentnih senzora smještenih u svakom od sedam zglobova koji na temelju momenata u svakom pojedinačnom zglobu estimiraju silu na vrhu radnog alata. U ovom slučaju na vrhu dodirnog ticala. Tema na koju se senzori pretplaćuju (engl. *„subscribe“*) i gdje šalju podatke o silama i momentima u zglobovima i na radnom alatu se zove *„/franka_robot_state_broadcaster/robot_state“*[16]. Podaci o silama i momentima na alatu izraženi su relativno u odnosu na bazu robota i nalaze se pod atributom „o_f_ext_hat_k“. Ovaj atribut omogućuje regulaciju sile na radnom alatu putem estimiranih vanjskih sila koje djeluju na alat.

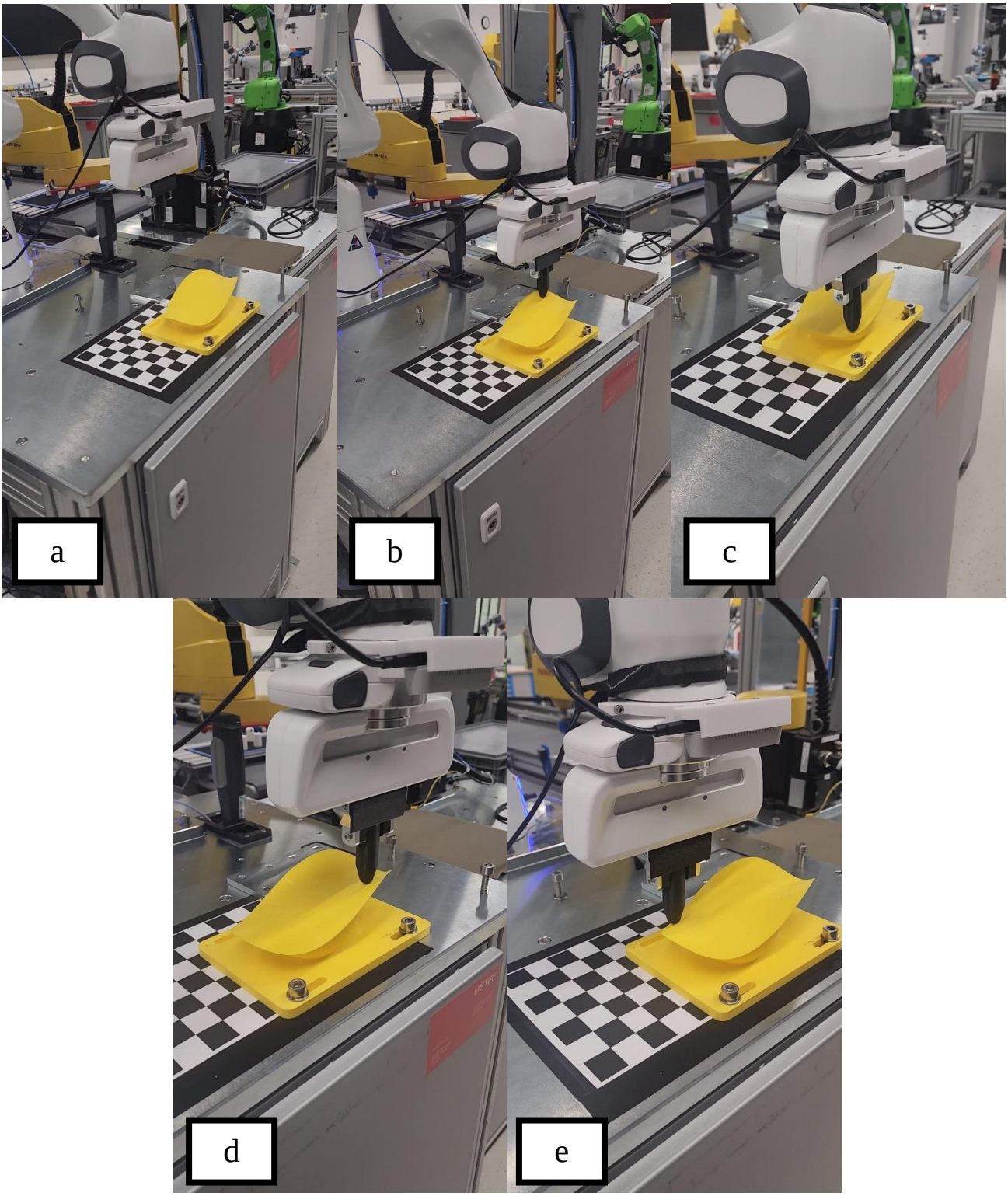
Ovaj kontroler sile se oslanja na paket *„pymoveit2“* te se implementira u *Python* skripti *„moveit2_Force_multiple_points“*: Skripta koristi funkcije paketa, poput *„move_to_pose()“* i *„wait_until_executed()“*, kako bi upravljala pokretima robota. Pored teme *„/franka_robot_state_broadcaster/robot_state“*, skripta koristi i teme *„/clicked_point“* te *„/cartesian_flag“*. Tema *„/clicked_point“* služi za odabir ciljnih točaka pomoću *„PointStamped“* čvora, kao što je i prethodno opisano, dok tema *„/cartesian_flag“* služi za odabir između zglobnog i linearnog (cartesian) gibanja. Pokretanjem skripte u terminalu, bira se vrijednost varijable *„number_of_points“* koja određuje broj željenih ciljnih točaka. Nakon odabira broja točaka, skripta izvršava interpolaciju između susjednih točaka s korakom od 5 milimetara. Gibanje robota se odvija diskretno, svakih 5 milimetara duž x i y koordinate, svaki put kada vrijednost sile uđe unutar zadanog raspona. Tijekom gibanja skripta ažurira z koordinatu svake sljedeće točke kako bi sila brže dosegla zadanu granicu.

Python skripta koja definira ovo kretanje robota priložena je na kraju rada u Prilogu 3, dok je izgled terminala nakon jedne izvršene iteracije prikazan na slici 48.



Slika 48. Jedna iteracija *"moveit2_Force_multiple_points.py"* skripte

Prikaz jedne iteracije praćenja oblika 3D površine prikazan je na slici 49.



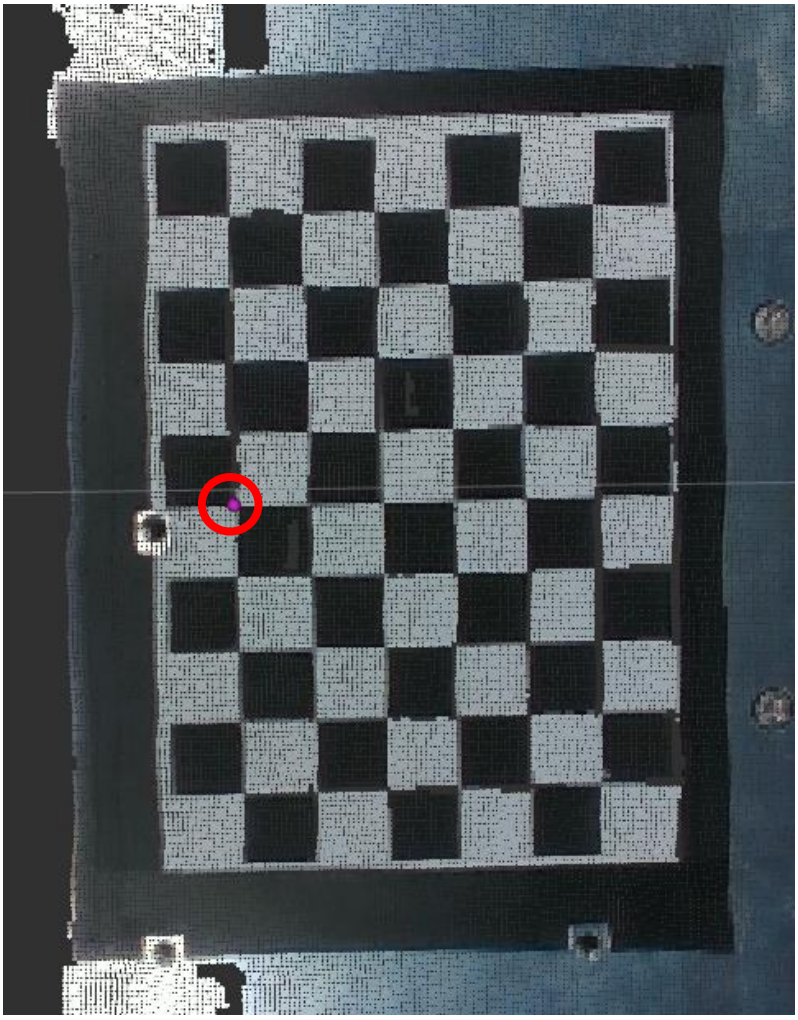
Slika 49. Praćenje 3D površine u zadavanjem četiri točke: a) početna pozicija, b) prva točka, c) druga točka, d) treća točka, e) četvrta točka

7. Analiza rezultata

7.1 Validacija dolaska robota u zadanu poziciju

Validacija dolaska robota u zadanu poziciju testirati će se pomoću *„moveit2_clicked_point.py“* skripte. Robot će biti doveden u tri točke gdje će se na temelju podataka o trenutnoj poziciji vrha radnog alata i zadane točke odrediti točnost pozicioniranja.

7.1.1 Prva validacijska točka



Slika 50. Položaj prve točke na kalibracijskoj ploči u *"rviz2"*

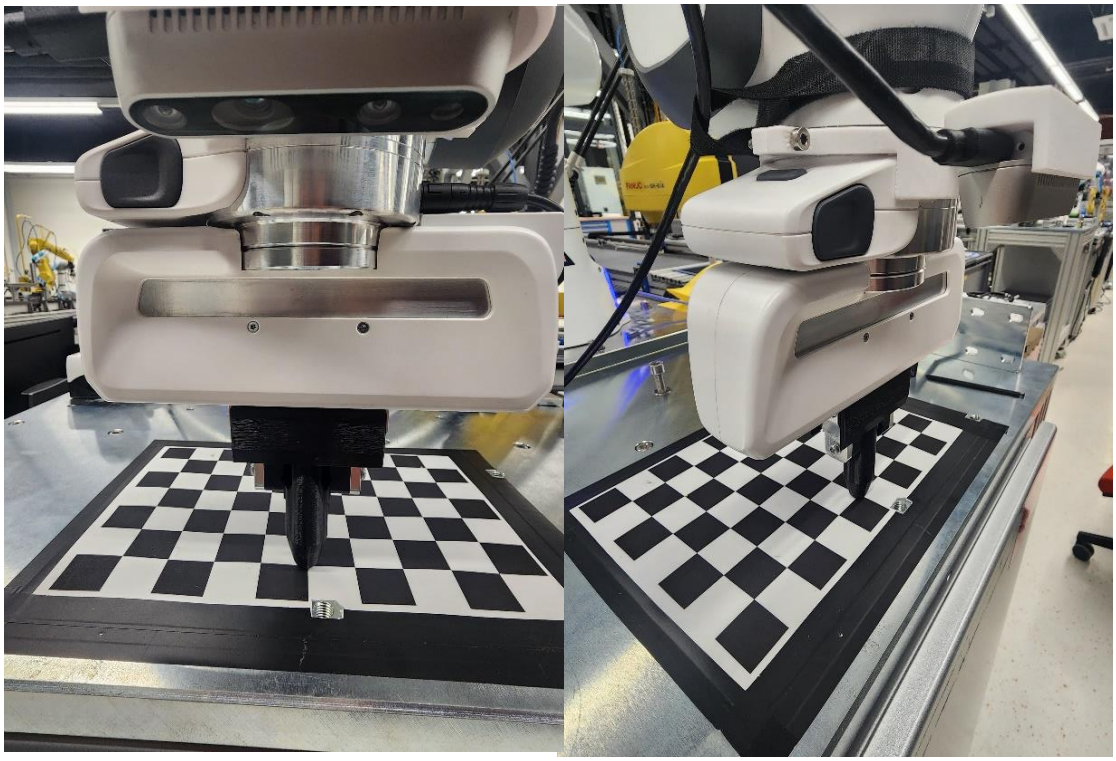
Na slici 50 prikazana je prva validacijska točka s koordinatama x = 0.5213582 m, y = 0.0050558 m, z = -0.0225594 m. Pokretanjem skripte robot se pozicionirao u točku s koordinatama x = 0.523432 m, y = 0.0055470 m, z = -0.0176137 m.

Usporedbom ovih vrijednosti mogu se izračunati greške pozicioniranja pojedine koordinate:

Δx = 2.07 mm

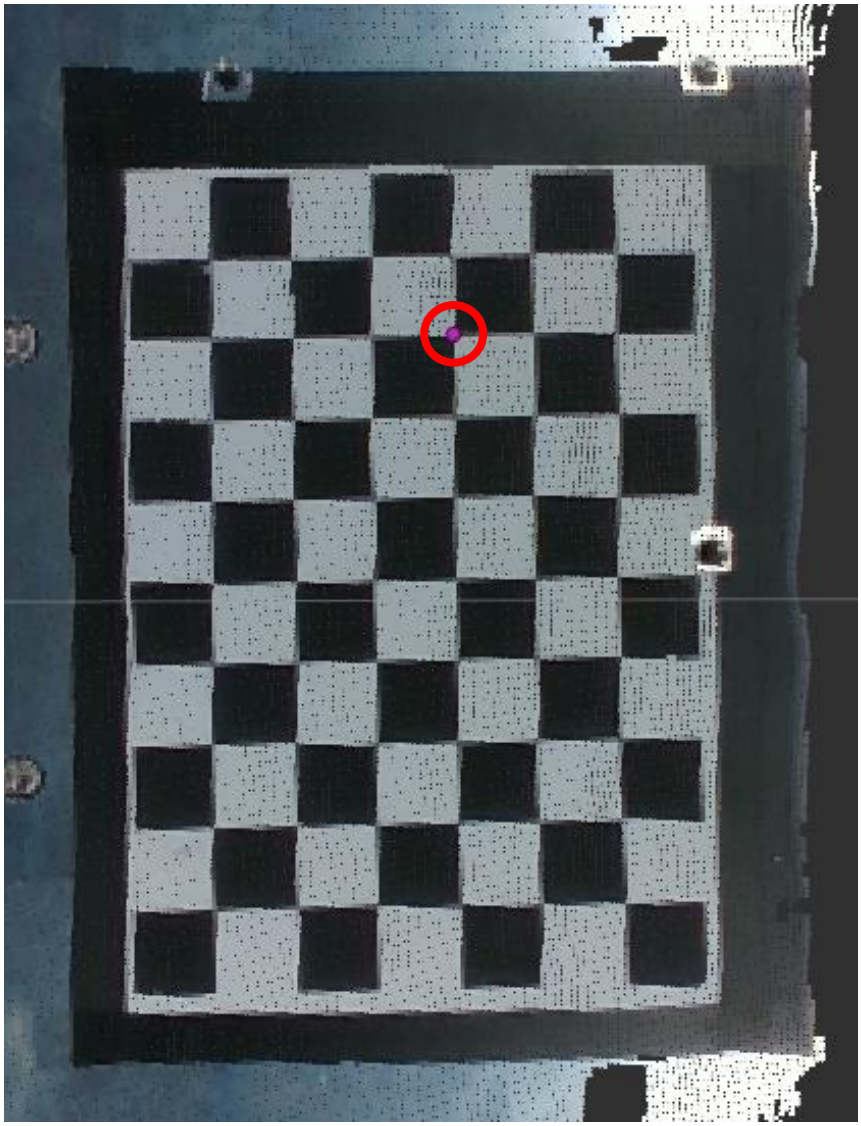
Δy = 0.49 mm

Δz = 4.95 mm



Slika 51. Dolazak robota u prvu validacijsku točku

7.1.2 Druga validacijska točka



Slika 52. Položaj druge točke na kalibracijskoj ploči u *"rviz2"*

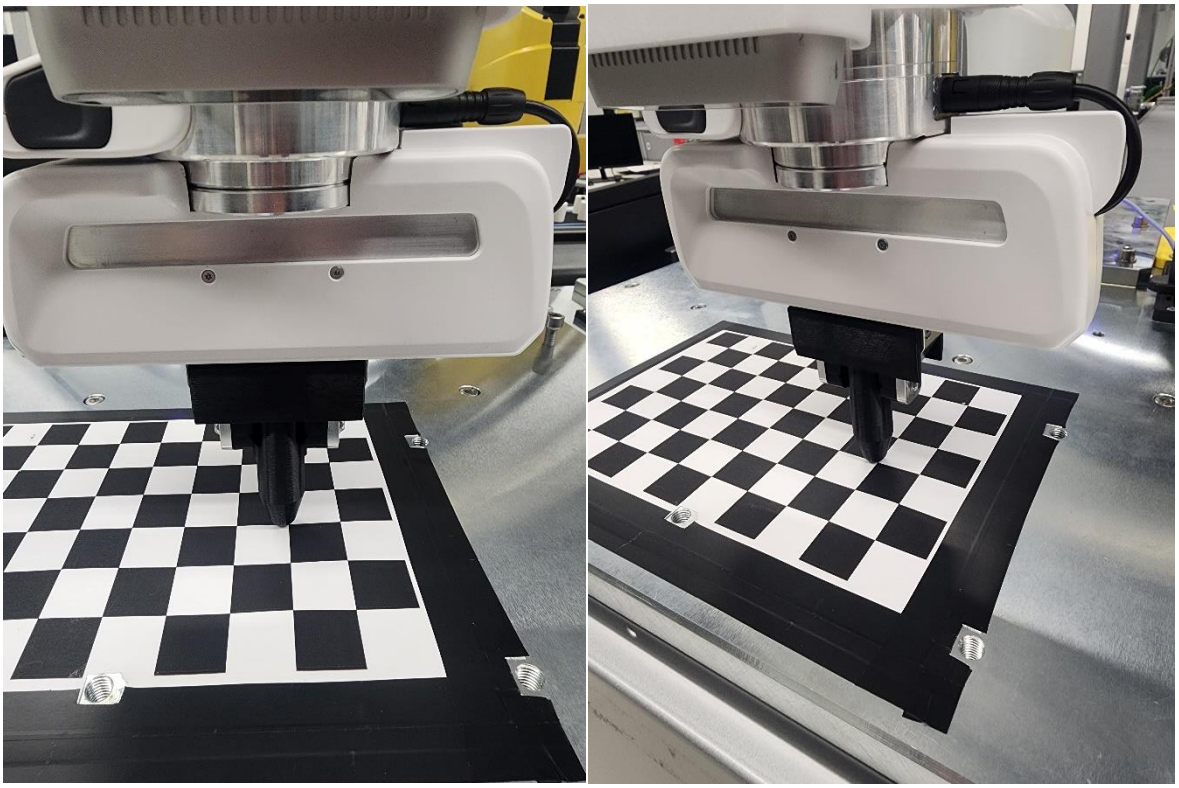
Na slici 52 prikazana je prva validacijska točka s koordinatama x = 0.4702012 m, y = 0.0802097 m, z = -0.0194274 m. Pokretanjem skripte robot se pozicionirao u točku s koordinatama x = 0.4704 m, y = 0.0799142 m, z = -0.017356 m.

Usporedbom ovih vrijednosti mogu se izračunati greške pozicioniranja pojedine koordinate:

Δx = 0.2 mm

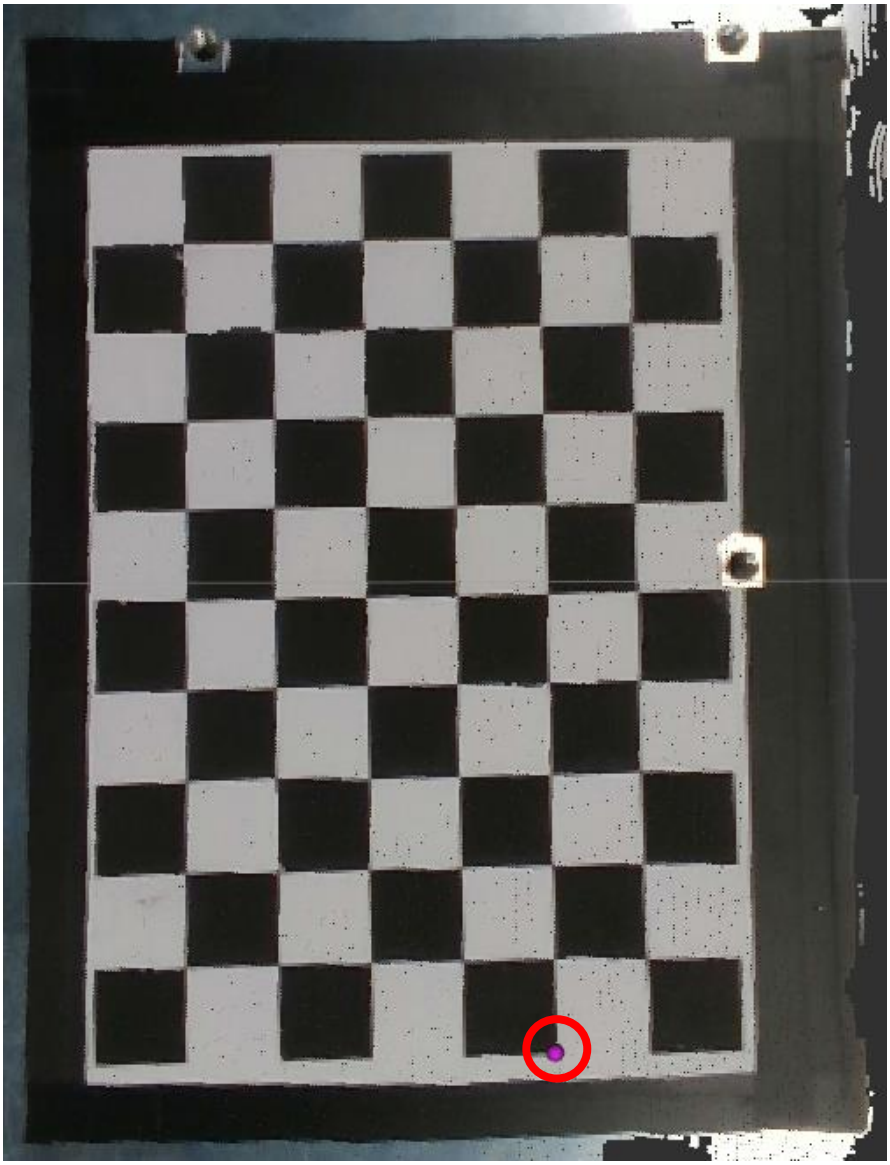
Δy = 0.3 mm

Δz = 2.07mm



Slika 53. Dolazak robota u drugu validacijsku točka

7.1.3 Treća validacijska točka



Slika 54. Položaj treće točke na kalibracijskoj ploči u *"rviz2"*

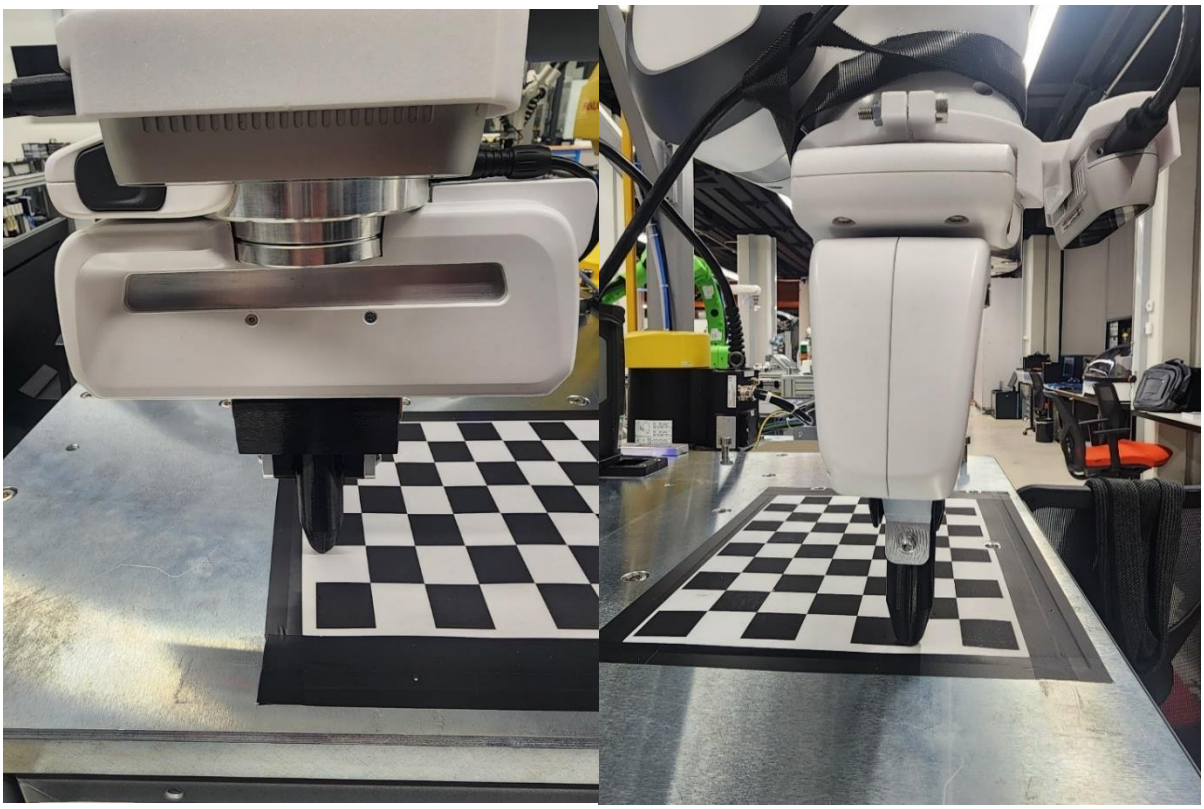
Na slici 54 prikazana je druga validacijska točka s koordinatama x= 0.4967807 m, y = -0.12631285 m, z = -0.0236727 m. Pokretanjem skripte robot se pozicionirao u točku s koordinatama x = 0.500766 m, y = -0.126173 m, z = -0.0175038 m.

Usporedbom ovih vrijednosti mogu se izračunati greške pozicioniranja pojedine koordinate:

Δx = 3.98 mm

Δy = 0.14 mm

Δz = 6.17 mm



Slika 55. Dolazak robota u treću validacijsku točku

7.1.4 Usporedba rezultata

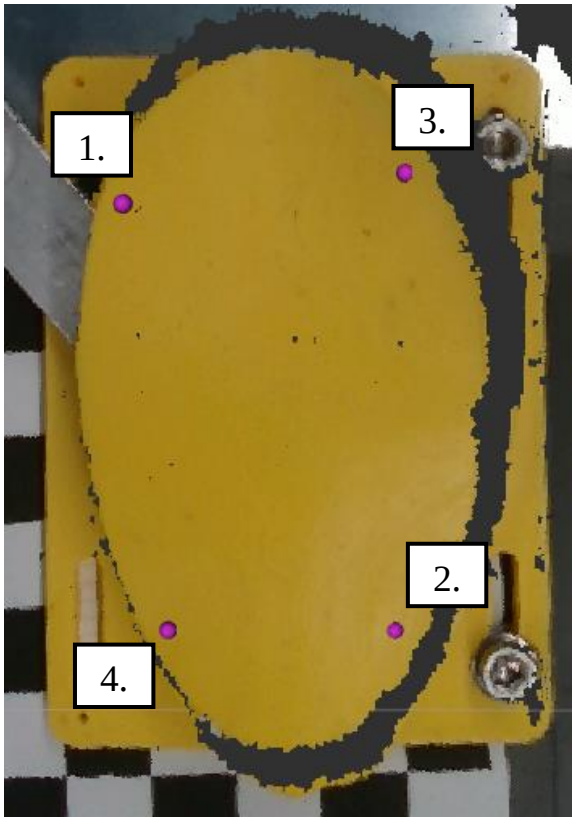
.

Usporedbom svih rezultata može se vidjeti odstupanje od +/- 4 mm na x koordinati, +/- 0.5 mm na y koordinati i +/- 6 mm na z koordinati što se smatra prihvatljivim za potrebe ovog istraživanja. Veća odstupanja x, y i z koordinate su posljedica nesavršenosti oblaka točaka. Zbog treperenja slike u *„rviz2“* može se dogoditi da ciljna točka bude nekoliko milimetara lijevo ili desno, viša ili niža od točke u kojoj se robot zapravo pozicionirao. Analizom slika stvarnog robota u kontaktu s kalibracijskom pločom uočava se minimalno odstupanje od zadane točke, što je vidljivo samo uz pomoć preciznog mjernog uređaja ili temeljne analize slike. Također, važno je napomenuti da veliki promjer vrha radnog alata otežava jasno promatranje stvarnog (centralnog) vrha alata.

7.2 Validacija praćenja zakrivljene 3D površine

Validacija praćenja 3D zakrivljene površine bit će testirana *„moveit2_Force_multiple_points.py“* skriptom preko koje ćemo robotu zadati tri kombinacije točaka i vidjeti u kolikoj mjeri i s kojim oscilacijama vertikalne Z sile i koordinate robot uspješno prati površinu. Zadane granice sile su između 8 i 14 N. Radni komad koji je vizualiziran pomoću kamere i korišten za validaciju nije dizajniran za potrebe ovog rada, već je to testni uzorak laboratorija CRTA-e koji se koristi u više projekata.

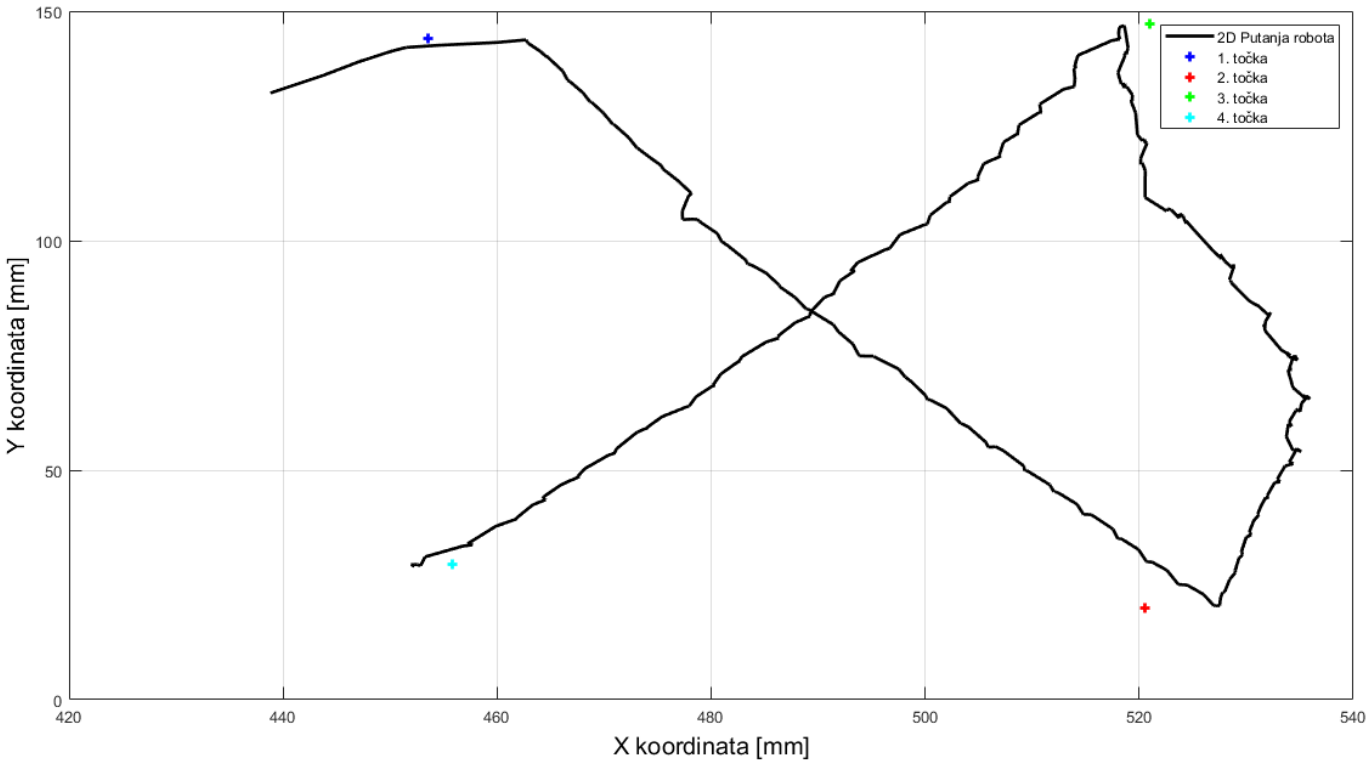
7.2.1 Validacija prve kombinacije točaka



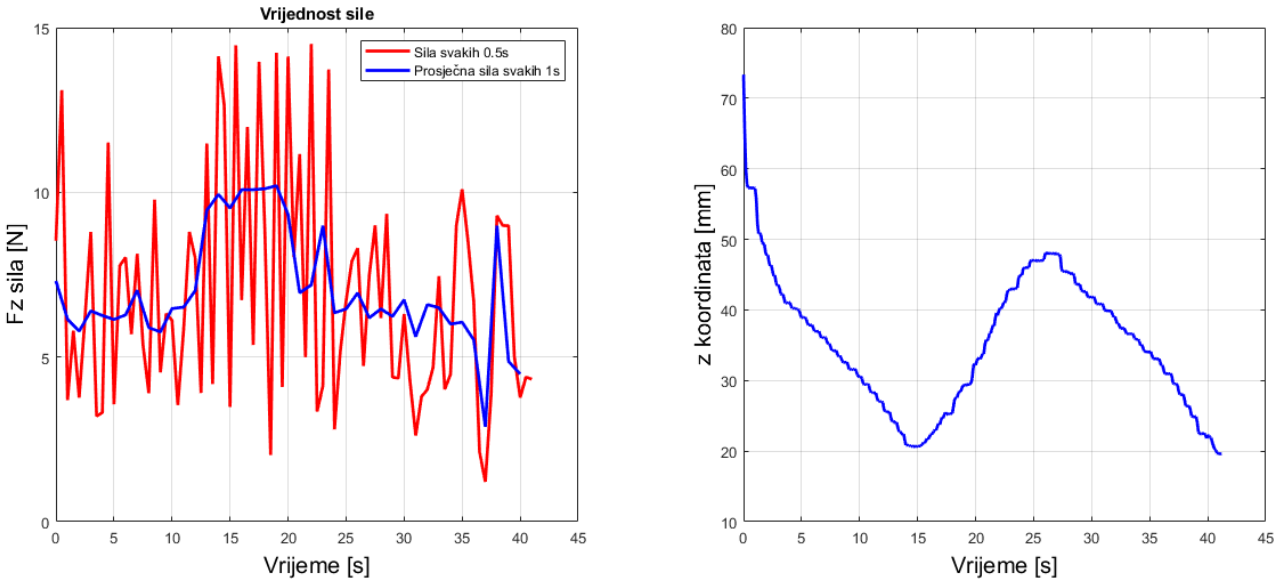
Slika 56. Raspored prve kombinacije točaka

Dijagram 1 pokazuje putanju robota kroz četiri zadane točke u x-y ravnini. Preciznost dolaska u željene točke je vidljiva zahvaljujući markerima koji ih označuju (+). Najveća pogreška je vidljiva kod 2. točke gdje odstupanje y koordinate iznosi 8.31 mm. Razlog tome leži u nagibu površine i promjeru ticala od 22 mm, što sprječava vrh alata da dosegne željenu točku.. Što se x koordinate tiče, odstupanje iznosi 2 mm što je vrlo dobar i jako poželjan rezultat. Kod

ostalih točaka, odstupanja su manja, a praćenje je ostvareno vrlo precizno. Važno je uzeti u obzir specifičnosti svake točke i okolnosti koje utječu na preciznost kretanja robota, poput nagiba površine i promjera ticala.



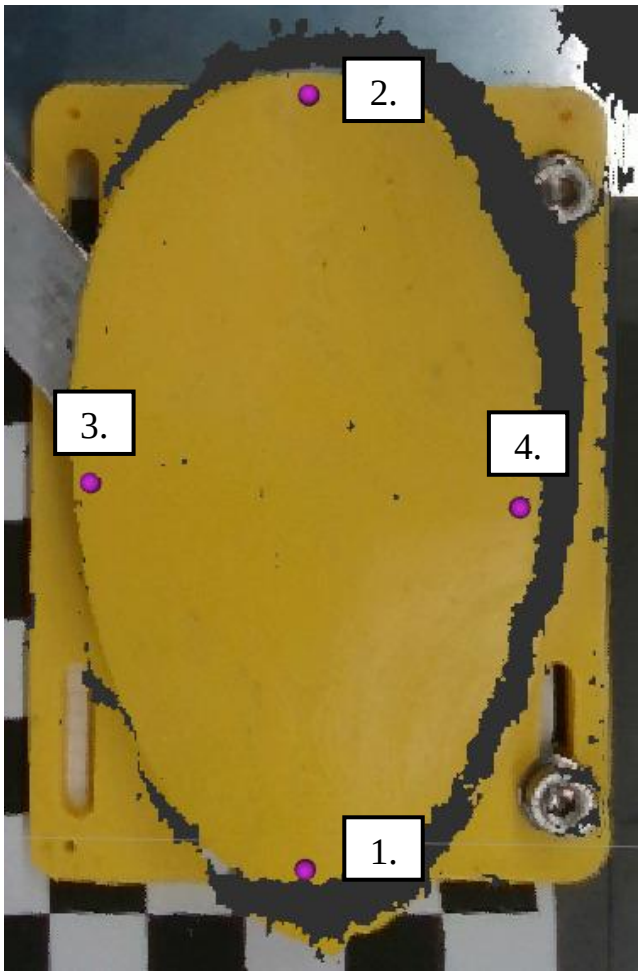
Dijagram 1. 2D putanja prve kombinacije točaka



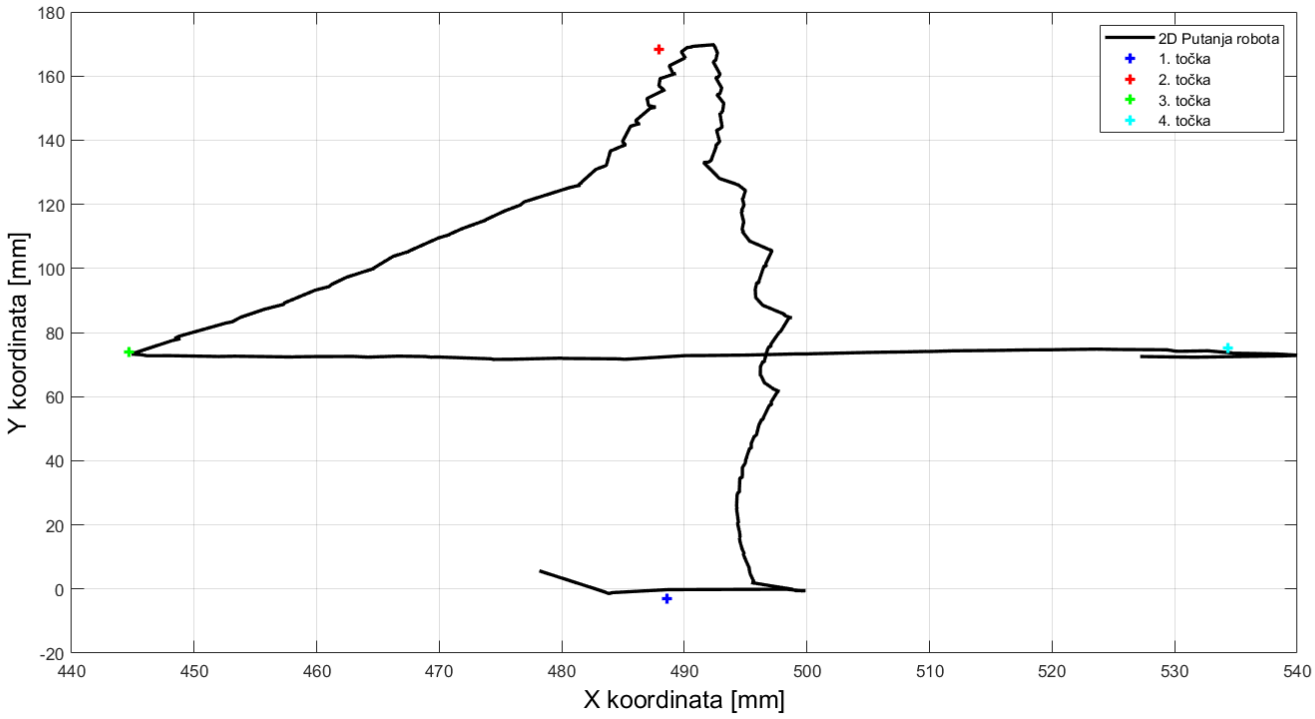
Dijagram 2. Vrijednost sile i z koordinate duž putanje 1

Na dijagramu 2 prikazana su dva grafa koji prikazuju vrijednost sile u smjeru z osi i vrijednost koordinate z. Na lijevom grafu, crvenom bojom je označena vrijednost sile očitana svakih 0.5 sekundi, dok je plavom prikazana srednja vrijednost sile uzeta u rasponu od 10 vrijednosti. Najveće odstupanje od zadanog raspona iznosi 6.5 N ispod donje granice i 0.5 N iznad gornje granice, što predstavlja vrlo dobar rezultat za diskretno gibanje. Srednja vrijednost vertikalne sile, prikazana plavom bojom, kreće se većinu vremena između 7 i 9 N, odnosno oko donje granice zadanih granica i nikada ne prelazi gornju granicu što je vrlo zadovoljavajući podatak. Tijekom cijelog puta, sila nikada nije pala ispod nule, što zadovoljava zahtjev za održavanjem konstantnog kontakta alata s površinom. Koordinata z, prikazana na desnom grafu, prati oblik površine s malim diskretnim pomacima kao posljedicom regulacije vertikalne sile.

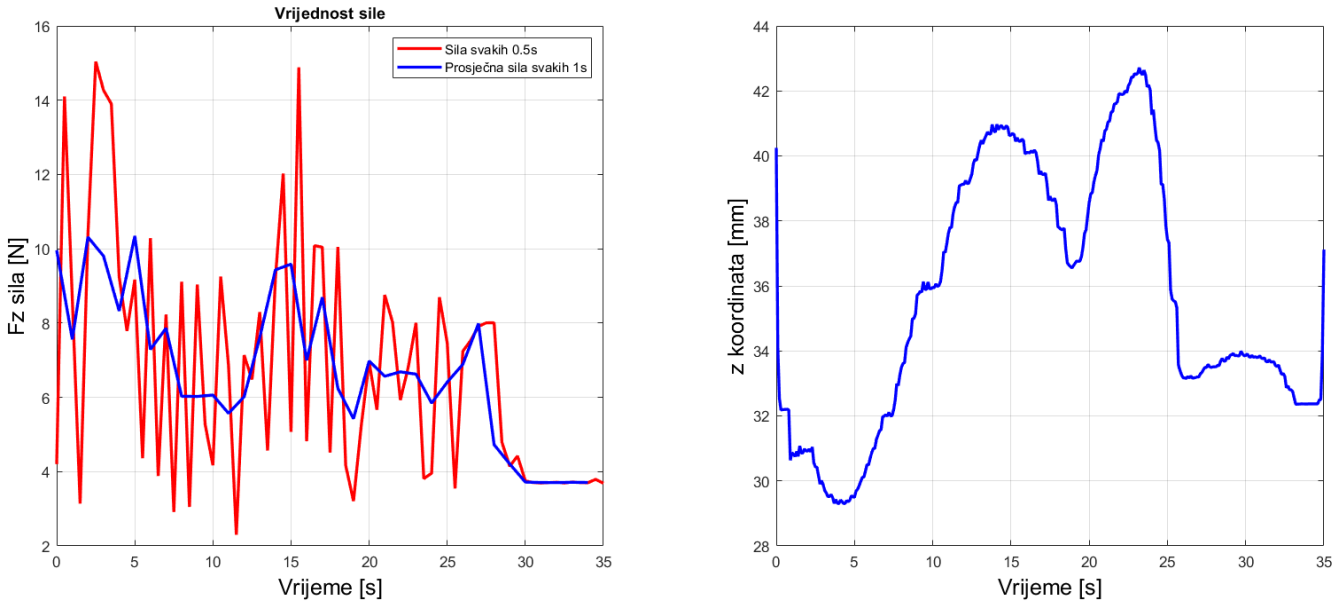
7.2.3 Validacija druge kombinacije točaka



Slika 57. Raspored druge kombinacije točaka



Dijagram 3. 2D putanja druge kombinacije točaka



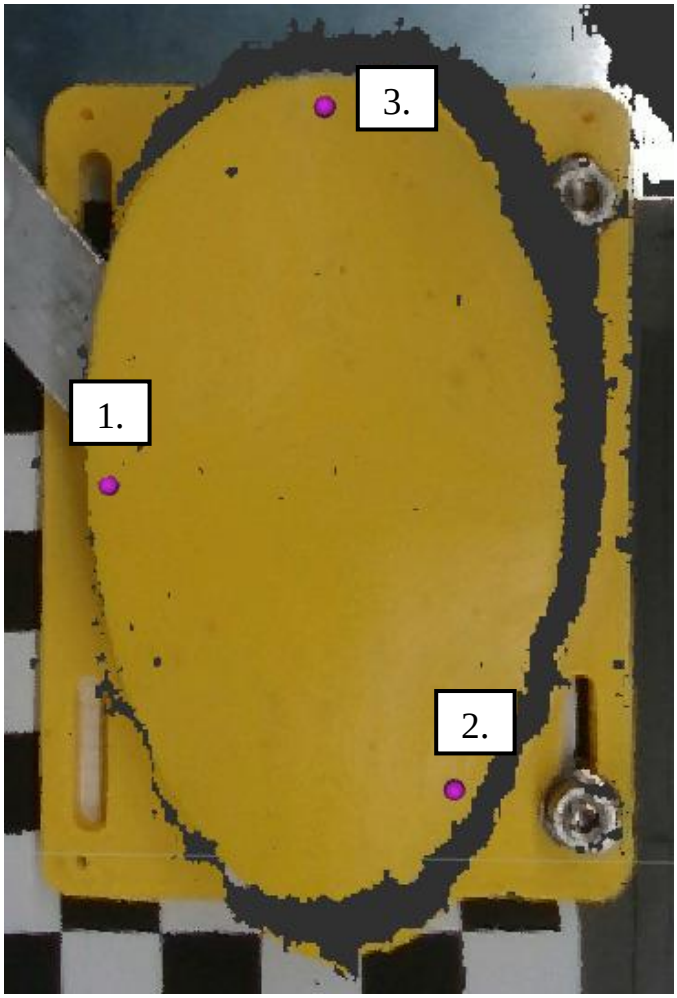
Dijagram 4. Vrijednost sile i z koordinate duž putanje 2

Dijagram 3 pokazuje iste parametre kao i dijagram 1, a to je putanja robota u x-y ravnini. Najveća pogreška je vidljiva kod 2. točke, kao i u prvom slučaju, gdje odstupanje y koordinate iznosi 5.25 mm, a odstupanje x koordinate 0.84 mm. Razlog tome je nagib kosine i primijenjena sila koji uzrokuju mali kutni pomak vrha alata u odnosu na njegovo ishodište. Pomak robota u smjeru x osi nakon dostizanja 4. točke rezultat je pomaka robota u početnu

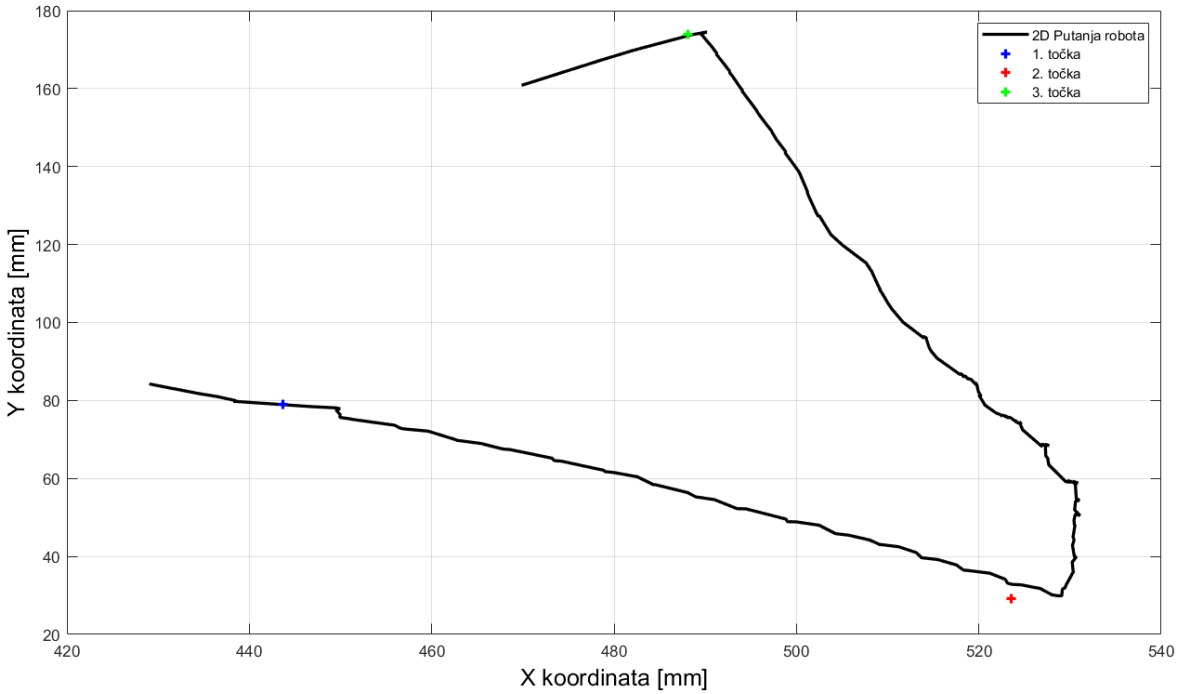
poziciju. Taj pokret stvara trzaj koji na djelić sekunde pomakne vrh radnog alata preko ciljne točke. Usprkos tome, praćenje je ostvareno vrlo precizno s najvećim odstupanjem od 5 mm što je za ovu primjenu vrlo dobro.

Na dijagramu 4, na lijevoj strani koja prikazuje vrijednost sile kroz vrijeme vidi se najveća amplituda od 5.7 N gledano od donjeg praga sile. Gornja najveća amplituda iznosi 15 N, što znači da je najveći skok od zadane granice 1 N. U ovom primjeru sila se vrlo dobro ponaša i robot je uspijeva držati ispod gornje granice kroz gotovo čitav put. Koordinata z, prikazana na desnoj strani, prati oblik površine s malim diskretnim pomacima kao posljedicom regulacije vertikalne sile, na sličan način kao u prvom primjeru.

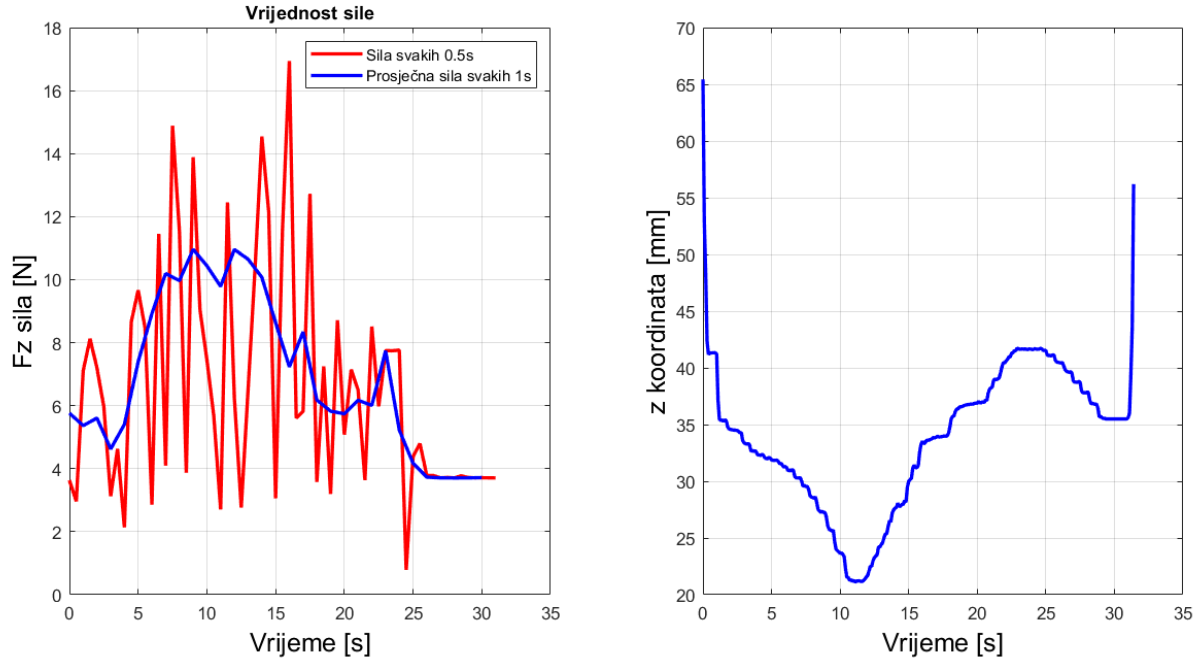
7.2.4 Validacija treće kombinacije točaka



Slika 58. Raspored treće kombinacije točaka



Dijagram 5. 2D putanja treće kombinacije točaka



Dijagram 6. Vrijednost sile i z koordinate duž putanje 3

S dijagrama 5 vidimo da je putanja u ovom primjeru validacije najtočnije ispraćena jer se i 1. i 3. točka nalaze točno na pravcu putanje, dok 2. točka odstupa s putanje 3.7 mm po y osi i 0.12 mm po x osi. Osim toga nema nikakvih drugih odstupanja što je najbolji rezultat dosad. Na dijagramu 6, sila skače od 16.9 N do 2.7 N, dok je zadnji impuls sile s vrijednosti 0.7 N

posljedica pomaka robota u početnu poziciju. Srednja vrijednost sile se nalazi unutar traženih granica, osim na početku i na kraju, odnosno pri pomaku u prvu točku i povratku u početnu poziciju. Desna strana, koja prikazuje vrijednost z koordinate ponaša se isto u svim primjerima, pa tako i ovdje, odnosno diskretnim pomacima prati zakrivljenu površinu i time regulira silu na vrhu radnog alata. Naglo spuštanje i podizanje vrijednosti z koordinate, u svim primjerima, posljedica je početka gibanja iz početne pozicije i vraćanja robota u tu istu poziciju.

8. Zaključak

U diplomskom radu uspješno je provedena integracija robota Franka Emika u ROS2 humble operacijski sustav. Svi potrebni paketi, za manipulaciju robotom, uspješno su integrirani u ROS2 radno okruženje *„panda_ros2_ws“* te su u model robota dodani model kamere i ticala pomoću URDF datoteka. Nadalje, zahvaljujući pažljivo i precizno provedenoj *„eye-in-hand“* kalibraciji kamere točnost pozicioniranja pomoću oblaka točaka je svedena na +/- 4 milimetara. Prilagodbom integriranih ROS2 paketa Franke potrebama ovog rada i razvojem vlastitih programa pomoću *Python* programskog jezika uspješno je modeliran kontroler sile koji diskretnim pomacima z osi regulira vrijednost sile u granici između 8 i 14 N. Zbog diskretnog gibanja robota korakom od 5 milimetara, pojavljuje se dosta oscilacija, koje bi se dodatnim podešavanjem parametara mogle smanjiti. Maksimalna greška dobivena u testnoj validaciji iznosi +/- 5.25 mm, što je za kontaktnu robotiku i rad s nepravilnom geometrijom vrlo dobro. Što se sile tiče, maksimalno odstupanje od zadane granice iznosi 1.21 N, što je 6.79 N manje od niže granice zbog posljedice strmog nagiba površine. U svim primjerima ostvaren je konstantni kontakt s površinom duž cijele putanje, jer sila ni u kojem trenutku ne ide ispod nule, što je bio i cilj ovog diplomskog rada.

Daljnjim istraživanjem metode generiranja oblaka točaka u *„rviz2“* mogla bi se dodatno stabilizirati slika pomoću koje izabiremo ciljne točke robota, a time i pozicioniranje samog robota i u projektima bez kontaktnog rada. Nadalje, diskretno gibanje, prikazano u ovom radu, ima svojih mana, kao što su neravnomjerni pomaci, velike amplitude sile i stepeničasto ponašanje z koordinate te bi za neke kompliciranije industrijske potrebe bilo praktičnije koristiti ovakav kontroler s konstantnim, ravnomjernim, linearnim gibanjem robota. U ovom radu to nije bilo moguće zbog kompleksnosti izrade samog kontrolera iz nule, koji bi savršeno pratio površinu.

Sve u svemu, bilo mi je iznimno drago raditi na robotskoj ruci Franka Emika i vidjeti iz prve ruke mogućnosti ovakvog sustava u vizijskoj i kontaktnoj primjeni. ROS2 sustav svakog se dana sve više razvija i nudi sve više mogućnosti u robotici, automatizaciji i ostalim područjima i samo je pitanje vremena kada će kontroler sile s kontinuiranim gibanjem biti integriran u ROS2 radno okruženje.

Literatura:

[1] Instalacija libfranke i ROS2 paketa, https://frankaemika.github.io/docs/franka_ros2.html, Pristup: 12.1.2024.

[2] Instalacija real-time kernela za Ubuntu 22.04, https://frankaemika.github.io/docs/installation_linux.html#preempt, Pristup:12.1.2024.

[3] Instalacija ViSP vodiča, https://visp-doc.inria.fr/doxygen/visp-3.2.0/tutorial-install-ubuntu.html#install_ubuntu_visp_config, Pristup: 12.1.2024.

[4] *„pymoveit2“* paket, https://github.com/AndrejOrsula/pymoveit2, Pristup: 12.1.2024.

[5] Ekstrinzična kalibracija kamere, https://visp-doc.inria.fr/doxygen/visp-daily/tutorial-calibration-extrinsic.html, Pristup: 12.1.2024.

[6] Intrinzična kalibracija kamere, https://visp-doc.inria.fr/doxygen/visp-3.2.0/tutorial-calibration-intrinsic.html, Pristup: 12.1.2024.

[7] Paketi za pokretanje Franka robota, https://github.com/tenfoldpaper/panda_ros2, Pristup: 12.1.2024.

[8] Paketi za upravljanje kamerom preko ROS2 sustava, https://github.com/IntelRealSense/realsense-ros, Pristup: 12.1.2024.

[9] https://www.prusa3d.com/category/original-prusa-i3-mk3s/, Pristup: 12.1.2024.

[10] Instalacija ROS2 Humble verzije, https://docs.ros.org/en/humble/Installation.html, Pristup: 12.1.2024.

[11] GitHub repozitorij, https://github.com/, Pristup: 12.1.2024.

[12] MoveIt 2, https://moveit.picknik.ai/main/index.html, Pristup: 12.1.2024.

[13] Izrada novog radnog područja, https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Creating-A-Workspace/Creating-A-Workspace.html, Pristup: 12.1.2024.

[14] Izrada ROS2 paketa, https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Creating-Your-First-ROS2-Package.html, Pristup: 12.1.2024.

[15] „Robot State“ struktura podataka o silama i momentima, https://frankaemika.github.io/libfranka/structfranka_1_1RobotState.html, 12.1.2024.

[16] Franka Emika, Data Sheet, Robot Arm & Control Manual

[17] Franka Emika, Franka Hand Product Manual

[18] Intel RealSense D400 Series Product Family Manual

[19] https://www.intelrealsense.com/depth-camera-d435/ , Pristup: 12.1.2024.

[20]https://projects.saifsidhik.page/franka_ros_interface/_modules/franka_tools/collision_behaviour_interface.html, Pristup: 12.1.2024.

PRILOG 1. URDF datoteka kamere Intel RealSense D435i

```
<?xml version="1.0" encoding="utf-8"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro" name="camera">
<!--
License: Apache 2.0. See LICENSE file in root directory.
Copyright(c) 2017 Intel Corporation. All Rights Reserved
This is the URDF model for the Intel RealSense 430 camera, in it's
aluminum peripherial evaluation case.
-->
  <xacro:macro name="camera" params= "connected_to:='' ns:='' rpy:='0 0 0'
xyz:='0 0 0' safety_distance:=0">
    <xacro:property name="d435_cam_depth_to_left_ir_offset" value="0.015"/>
    <xacro:property name="d435_cam_depth_to_right_ir_offset" value="0.050"/>
    <xacro:property name="d435_cam_depth_to_projector_offset" value="0.029"/>
    <xacro:property name="d435_cam_depth_to_centerline" value="0.0325"/>

    <xacro:property name="d435_cam_width" value="0.090"/>
    <xacro:property name="d435_cam_height" value="0.025"/>
    <xacro:property name="d435_cam_depth" value="0.02505"/>

    <xacro:property name="d435_cam_depth_px" value="0.0148"/>
    <xacro:property name="d435_cam_depth_py" value="0.0175"/>
    <xacro:property name="d435_cam_depth_pz" value="${d435_cam_height/2}"/>

    <xacro:unless value="${connected_to == ''}">
      <joint name="camera_link_joint" type="fixed">
        <parent link="${connected_to}"/>
        <child link="camera_link"/>
        <origin xyz="0.07525244249 -0.03438864915 -0.1154851146"
rpy="0.1102715613 -0.007340033758 1.589046084"/>
      </joint>
    </xacro:unless>

    <link name="camera_link">
       <visual>
       <origin xyz="${d435_cam_depth_to_centerline} 0 0" rpy="0 0 ${pi}"/>
        <geometry>
          <mesh
filename="package://franka_description/meshes/visual/d435.dae"/>
        </geometry>
      </visual>
    </link>

    <joint name="camera_depth_joint" type="fixed">
      <origin xyz="${d435_cam_depth_px} 0 0" rpy="0 0 0"/>
      <parent link="camera_link"/>
```

```
      <child link="camera_depth_frame"/>
    </joint>
    <link name="camera_depth_frame"/>
    <joint name="camera_depth_optical_joint" type="fixed">
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <parent link="camera_depth_frame"/>
      <child link="camera_depth_optical_frame"/>
    </joint>
    <link name="camera_depth_optical_frame"/>

    <joint name="camera_infra1_joint" type="fixed">
      <origin xyz="${d435_cam_depth_to_left_ir_offset} 0 0" rpy="0 0 0" />
      <parent link="camera_link" />
      <child link="camera_infra1_frame" />
    </joint>
    <link name="camera_infra1_frame"/>
    <joint name="camera_infra1_optical_joint" type="fixed">
      <origin xyz="0 0 0" rpy="0 0 0" />
      <parent link="camera_infra1_frame" />
      <child link="camera_infra1_optical_frame" />
    </joint>
    <link name="camera_infra1_optical_frame"/>
    <joint name="camera_infra2_joint" type="fixed">
      <origin xyz="${d435_cam_depth_to_right_ir_offset} 0 0" rpy="0 0 0" />
      <parent link="camera_link" />
      <child link="camera_infra2_frame" />
    </joint>
    <link name="camera_infra2_frame"/>
    <joint name="camera_infra2_optical_joint" type="fixed">
      <origin xyz="0 0 0" rpy="0 0 0" />
      <parent link="camera_infra2_frame" />
      <child link="camera_infra2_optical_frame" />
    </joint>
    <link name="camera_infra2_optical_frame"/>
    <joint name="camera_color_joint" type="fixed">
      <origin xyz="0 0 0" rpy="0 0 0" />
      <parent link="camera_link" />
      <child link="camera_color_frame" />
    </joint>
    <link name="camera_color_frame"/>
    <joint name="camera_color_optical_joint" type="fixed">
      <origin xyz="0 0 0" rpy="0 0 0" />
      <parent link="camera_color_frame" />
      <child link="camera_color_optical_frame" />
    </joint>
    <link name="camera_color_optical_frame"/>
  </xacro:macro>
</robot>
```

PRILOG 2. URDF datoteka radnog alata

```
<?xml version="1.0" encoding="utf-8"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro" name="ticalo">
  <!-- safety_distance: Minimum safety distance in [m] by which the collision
volumes are expanded and which is enforced during robot motions -->
  <xacro:macro name="ticalo" params="connected_to:='' ns:='' rpy:='0 0 0'
xyz:='0 0 0' safety_distance:=0">

    <xacro:unless value="${connected_to == ''}">
      <joint name="${ns}_ticalo_joint" type="fixed">
        <parent link="${connected_to}"/>
        <child link="${ns}_ticalo"/>
        <origin xyz="0 0 0.0664" rpy="${rpy}"/>
      </joint>
    </xacro:unless>

    <link name="${ns}_ticalo">
       <visual>
        <origin xyz="-0.0147 -0.035 0.093" rpy="${-pi/2} 0 0"/>
        <geometry>
          <mesh
filename="package://franka_description/meshes/visual/Ticalo_v2_14_1_2024.STL"
scale="0.001 0.001 0.001"/>
        </geometry>
        <material name="black_metal">
          <color rgba="0.1 0.1 0.1 1"/>
        </material>
      </visual>
    </link>
    <!--
     Define the hand_tcp frame xyz="-0.012 -0.01 0.128"
     -->
    <link name="${ns}_ticalo_tcp"/>
    <joint name="${ns}_ticalo_tcp_joint" type="fixed">
      <origin xyz="-0.0031 -0.0073 0.093" rpy="0 0 0"/>
      <parent link="${ns}_ticalo"/>
      <child link="${ns}_ticalo_tcp"/>
    </joint>
  </xacro:macro>
</robot>
```

PRILOG 3. Python skripta za postavljanje kolizijskih ograničenja

```
import rclpy
from franka_msgs.srv import SetForceTorqueCollisionBehavior,
SetFullCollisionBehavior, SetCartesianStiffness

DEFAULT_VALUES = {
    'torque_acc_lower': [50.0, 50.0, 50.0, 50.0, 50.0, 50.0, 50.0],
    'torque_acc_upper': [100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0],
    'torque_lower': [50.0, 50.0, 50.0, 50.0, 50.0, 50.0, 50.0],
    'torque_upper': [100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0],
    'force_acc_lower': [20.0, 20.0, 20.0, 20.0, 20.0, 20.0],
    'force_acc_upper': [50.0, 50.0, 50.0, 50.0, 50.0, 50.0],
    'force_lower': [20.0, 20.0, 20.0, 20.0, 20.0, 20.0],
    'force_upper': [50.0, 50.0, 50.0, 50.0, 50.0, 50.0],
    'cartesian_stiffness':[2000.0, 2000.0, 2000.0, 200.0, 200.0, 200.0]
}


class CollisionBehaviourInterface:
    def __init__(self):
        self.node = rclpy.create_node('collision_behaviour_interface')
        self.ft_collision_behaviour_handle =
self.node.create_client(SetForceTorqueCollisionBehavior,
'/param_service_server/set_force_torque_collision_behavior')
        self.full_collision_behaviour_handle =
self.node.create_client(SetFullCollisionBehavior,
'/param_service_server/set_full_collision_behavior')
        self.cartesian_stiffness_handle =
self.node.create_client(SetCartesianStiffness,
'/param_service_server/set_cartesian_stiffness')


        while not
self.ft_collision_behaviour_handle.wait_for_service(timeout_sec=1.0) or not
self.full_collision_behaviour_handle.wait_for_service(timeout_sec=1.0) or not
self.cartesian_stiffness_handle.wait_for_service(timeout_sec=1.0):
            self.node.get_logger().info('Waiting for collision behaviour
services...')

        self.node.get_logger().info('Collision behaviour services found.')

    def set_ft_contact_collision_behaviour(self, torque_lower=None,
torque_upper=None, force_lower=None, force_upper=None,
cartesian_stiffness=None):
        if torque_lower is None:
            torque_lower = DEFAULT_VALUES['torque_lower']
```

```
        if torque_upper is None:
            torque_upper = DEFAULT_VALUES['torque_upper']
        if force_lower is None:
            force_lower = DEFAULT_VALUES['force_lower']
        if force_upper is None:
            force_upper = DEFAULT_VALUES['force_upper']
        if cartesian_stiffness is None:
            cartesian_stiffness = DEFAULT_VALUES['cartesian_stiffness']

        request = SetForceTorqueCollisionBehavior.Request()
        request.lower_torque_thresholds_nominal = torque_lower
        request.upper_torque_thresholds_nominal = torque_upper
        request.lower_force_thresholds_nominal = force_lower
        request.upper_force_thresholds_nominal = force_upper

        future = self.ft_collision_behaviour_handle.call_async(request)
        rclpy.spin_until_future_complete(self.node, future)
        if future.result() is not None:
            self.node.get_logger().info(f"CollisionBehaviourInterface:
Collision behaviour change request: {'Success' if future.result().success else
'Failed!'}.\nDetails: {future.result().error}")
            return future.result().success
        else:
            self.node.get_logger().warn(f"CollisionBehaviourInterface:
Collision behaviour change service call failed.")
            return False

    def set_force_threshold_for_contact(self, cartesian_force_values):
        return
self.set_ft_contact_collision_behaviour(force_lower=cartesian_force_values)

    def set_force_threshold_for_collision(self, cartesian_force_values):
        return
self.set_ft_contact_collision_behaviour(force_upper=cartesian_force_values)

    def set_collision_threshold(self, joint_torques=None,
cartesian_forces=None):
        return
self.set_ft_contact_collision_behaviour(torque_upper=joint_torques,
force_upper=cartesian_forces)

    def set_contact_threshold(self, joint_torques=None,
cartesian_forces=None):
        return
self.set_ft_contact_collision_behaviour(torque_lower=joint_torques,
force_lower=cartesian_forces)

    def set_cartesian_stiffness(self, stiffness_values):
```

```
        return
self.set_ft_contact_collision_behaviour(cartesian_stiffness=stiffness_values)

def main(args=None):
    rclpy.init(args=args)
    collision_interface = CollisionBehaviourInterface()

    cartesian_stiffness_new = [2500.0, 2500.0, 2500.0, 100.0, 100.0, 100.0]
    joint_torques = [100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0]
    cartesian_forces = [60.0, 60.0, 60.0, 60.0, 60.0, 60.0]

    collision_interface.set_collision_threshold(joint_torques=joint_torques,
cartesian_forces=cartesian_forces)


collision_interface.set_cartesian_stiffness(stiffness_values=cartesian_stiffne
ss_new)

    rclpy.spin(collision_interface.node)
    collision_interface.node.destroy_node()
    rclpy.shutdown()

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

PRILOG 4. Python skripta za diskretno gibanje robota uz regulaciju sile

```
#!/usr/bin/env python3

from threading import Thread, Event
import time
import numpy as np
import rclpy
from rclpy.callback_groups import ReentrantCallbackGroup
from rclpy.node import Node
from geometry_msgs.msg import PointStamped
from std_msgs.msg import Bool
from franka_msgs.msg import FrankaState
from pymoveit2 import MoveIt2
from pymoveit2.robots import panda


class PoseGoalNode(Node):
    def __init__(self):
        super().__init__('pose_goal')
        self.clicked_point_event = Event()
        self.points_to_move = []  # List to store points
        self.number_of_points = None  # Number of points to reach
        self.iteration_number = 1

        # Create a single instance of MoveIt2 in the constructor
        self.moveit2 = MoveIt2(
            node=self,
            joint_names=panda.joint_names(),
            base_link_name=panda.base_link_name(),
            end_effector_name=panda.end_effector_name(),
            group_name=panda.MOVE_GROUP_ARM,
            callback_group=ReentrantCallbackGroup(),
        )

        # Subscribe to the topics
        self.subscription = self.create_subscription(PointStamped,
'/clicked_point', self.clicked_point_callback, 10)
        self.cartesian_subscription = self.create_subscription(Bool,
'/cartesian_flag', self.cartesian_flag_callback, 10)
        self.franka_state_subscription = self.create_subscription(FrankaState,
'/franka_robot_state_broadcaster/robot_state', self.check_force, 10)
        # Default values
        self.fixed_quaternion = [1.0, 0.0, 0.0, 0.0]
        self.cartesian_flag = True  # Set to True by default
```

```
    def clicked_point_callback(self, msg):
        position = [msg.point.x, msg.point.y, msg.point.z - 0.0015]
        quat_xyzw = self.fixed_quaternion

        # Append the point to the list only if it's not already in the list
        if len(self.points_to_move) == 0 or position != self.points_to_move[-
1][0]:
            self.points_to_move.append((position, quat_xyzw,
self.cartesian_flag))

            # Print the clicked point
            self.get_logger().info(f"Koordinate kliknute točke: {position}")

        if len(self.points_to_move) == self.number_of_points:
            # Perform interpolation
            for i in range(len(self.points_to_move) - 1):
                start_point = self.points_to_move[i]
                end_point = self.points_to_move[i+1]
                self.interpolate_points(start_point, end_point)

            # Print all points in the terminal
            del self.points_to_move[:self.number_of_points]
            unique_points = []
            for point in self.points_to_move:
                if point not in unique_points:
                    unique_points.append(point)
            self.points_to_move = unique_points
            # self.get_logger().info("Svi punktovi za pomicanje:")
            # for point_idx, (position, _, cartesian_flag) in
enumerate(self.points_to_move):
            #     self.get_logger().info(f"Punkt {point_idx + 1}: Pozicija:
{position}")

            # Set the event to signal that interpolation is done
            self.clicked_point_event.set()

    def check_force(self, msg):
        # Retrieve the latest force values from the robot state
        x_force = msg.o_f_ext_hat_k[0]
        y_force = msg.o_f_ext_hat_k[1]
        z_force = - msg.o_f_ext_hat_k[2]
        self.z_force = z_force

    def cartesian_flag_callback(self, msg):
        self.cartesian_flag = msg.data


    def move_to_pose(self, position, quat_xyzw, cartesian):
```

```
        self.moveit2.move_to_pose(position=position, quat_xyzw=quat_xyzw,
cartesian=cartesian)
        self.moveit2.wait_until_executed()

    def interpolate_points(self, start_point, end_point):
        time.sleep(0.5)
        self.get_logger().info(f"Izvršavanje interpolacije između
{self.iteration_number}. i {self.iteration_number+1}. točke.")
        self.iteration_number = self.iteration_number + 1
        interpolation_step = 0.005  # 5 mm
        start_position, _, _ = start_point
        end_position, _, _ = end_point

        # Calculate the number of interpolated points needed for each axis
        num_interpolated_points_x = int(np.ceil(np.abs(end_position[0] -
start_position[0]) / interpolation_step))
        num_interpolated_points_y = int(np.ceil(np.abs(end_position[1] -
start_position[1]) / interpolation_step))
        num_interpolated_points_z = int(np.ceil(np.abs(end_position[2] -
start_position[2]) / interpolation_step))

        # Choose the maximum number of points among axes
        num_interpolated_points = max(num_interpolated_points_x,
num_interpolated_points_y, num_interpolated_points_z)

        interpolated_all_points = []

        for i in range(num_interpolated_points + 1):
            alpha = i / num_interpolated_points
            interpolated_position = [
                start + alpha * (end - start) for start, end in
zip(start_position, end_position)
            ]
            interpolated_quat_xyzw = start_point[1]

            interpolated_all_points.append((interpolated_position,
interpolated_quat_xyzw, start_point[2]))

        # Add all interpolated points to the points_to_move list
        self.points_to_move.extend(interpolated_all_points)

        # for j, (position, _, _) in enumerate(interpolated_all_points):
        #     self.get_logger().info(f"Interpolirana pozicija {j +
1}/{len(interpolated_all_points)}: {position}")
        # self.get_logger().info("Interpolacija završena uspješno.")
        time.sleep(0.5)
```

```
    def execute_all_moves(self):

        initial_point = self.points_to_move[0]
        initial_position, quat_xyzw, cartesian = initial_point
        elevated_position = [initial_position[0], initial_position[1],
initial_position[2] + 0.015]

        # Move to the elevated position and wait for 0.5 seconds
        self.move_to_pose(elevated_position, quat_xyzw, cartesian)
        time.sleep(0.5)

        self.get_logger().info(f"Izvršavanje dolaska u sve
{len(self.points_to_move)} pozicije.")
        for i, point in enumerate(self.points_to_move):
            position, quat_xyzw, cartesian = point
            self.get_logger().info(f"Gibanje u poziciju {i +
1}/{len(self.points_to_move)}: {position}")

            # Move to the current point
            self.move_to_pose(position, quat_xyzw, cartesian)

            while not self.clicked_point_event.is_set():

                # Continuously check force while in motion
                z_force = self.z_force
                self.get_logger().info(f"Z Sila tijekom gibanja: {z_force}")

                # Adjust z-coordinate based on force
                if 8 <= z_force <= 14:
                    # Force is within the desired range
                    break
                elif z_force < 0:
                        position[2] -= 0.004
                        # self.get_logger().info(f"Namještanje pozicije:
{position}")
                        self.move_to_pose(position, quat_xyzw, cartesian)
                elif 0 < z_force < 8:
                    # Force is too low, go down on z-coordinate
                    if z_force >= 4:
                        position[2] -= (8 - z_force) * 0.00043
                        # self.get_logger().info(f"Namještanje pozicije:
{position}")
                        self.move_to_pose(position, quat_xyzw, cartesian)
                        for next_point in self.points_to_move[i + 1:]:
                            next_point[0][2] = position[2] - 0.0004
                    else:
                        position[2] -= 0.0002
```

```
                        # self.get_logger().info(f"Namještanje pozicije:
{position}")
                        self.move_to_pose(position, quat_xyzw, cartesian)
                else:
                    # Force is too high, go up on z-coordinate
                    if z_force <= 20:
                        position[2] += (z_force - 14) * 0.00044
                        # self.get_logger().info(f"Namještanje pozicije:
{position}")
                        self.move_to_pose(position, quat_xyzw, cartesian)
                        for next_point in self.points_to_move[i + 1:]:
                            next_point[0][2] = position[2] + 0.003
                    else:
                        position[2] += 0.0002
                        # self.get_logger().info(f"Namještanje pozicije:
{position}")
                        self.move_to_pose(position, quat_xyzw, cartesian)

            # Reset the event for the next point
            self.clicked_point_event.clear()

        self.get_logger().info("Sva gibanja završena uspješno.")
        time.sleep(1.5)
        self.moveit2.move_to_pose(position=[0.4, 0.1, 0.15],
quat_xyzw=self.fixed_quaternion, cartesian=self.cartesian_flag)
        self.moveit2.wait_until_executed()
        self.get_logger().info("Povratak u početnu poziciju.")
        print("\n")
        new_number_of_points = int(input(f"Unesite novi broj točaka koje
želite dostići: "))
        print("\n")
        self.number_of_points = new_number_of_points

def main():
    rclpy.init()

    node = PoseGoalNode()

    executor = rclpy.executors.MultiThreadedExecutor(2)
    executor.add_node(node)
    executor_thread = Thread(target=executor.spin, daemon=True, args=())
    executor_thread.start()

    try:

        node.number_of_points = int(input("Broj točaka koje želite dostići:
"))
```

```
        while rclpy.ok():
            # Wait for the user to click on a point
            node.clicked_point_event.wait()
            node.clicked_point_event.clear()

            # Add logic to determine when to execute all movements
            if len(node.points_to_move) >= node.number_of_points:

                node.execute_all_moves()
                node.points_to_move = []
                node.iteration_number = 1

    except KeyboardInterrupt:
        pass
    finally:
        rclpy.shutdown()

if __name__ == "__main__":
    main()
```